MACRO ASSEMBLER LANGUAGE REFERENCE MANUAL

OPERATING SYSTEM SOFTWARE

MAKES MICROS RUN LIKE MINIS
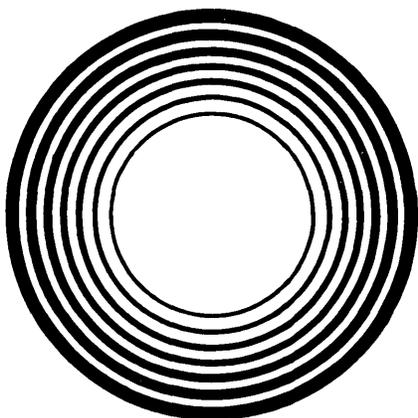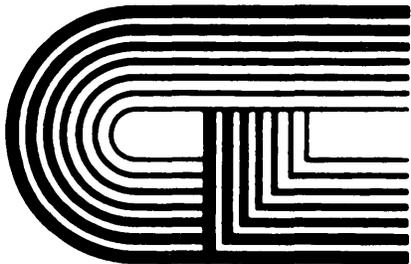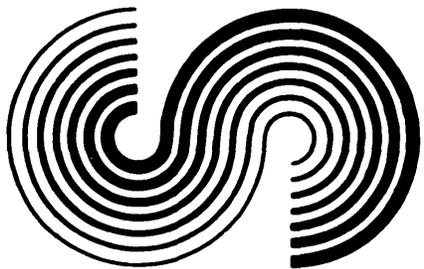
MACRO ASSEMBLER LANGUAGE REFERENCE MANUAL

Second Edition

Documentation by: C. P. Williams
Software by: Timothy S. Williams

OPERATING SYSTEM SOFTWARE

MAKES MICROS RUN LIKE MINIS

PROPRIETARY NOTICE

PREFACE

This manual describes the OASIS MACRO Assembler Language. It assumes the reader is familiar with computer software fundamentals and has had some exposure to assembly language programing on micro-computers.   The section "Additional and Reference Material" below lists documents that may prove helpful in reviewing those areas.

The user who is unfamiliar with OASIS should first read or review those chapters of interest in the OASIS System Reference Manual to become familiar with system conventions.

Included in this manual is a chapter on "Interfacing to OASIS" which provides information about writing device drivers, assembly language subroutines that are called by a BASIC program, console class code drivers, etc.

This manual, named MACRO, like all OASIS documentation manuals, has the manual name and revision number in the lower, inside corner of each page of the body of the manual. In most chapters of the manual the last primary subject being discussed on a page will be identified in the lower outside corner of the page.

Additional and Referenced Material

The following manuals and publications were used in the creation of this manual and contain additional information not included in this document.

ZILOG Z80 Assembler Manual

ZILOG Z80-CPU Technical Manual

ZILOG Z80-CPU Programming Reference Card

OASIS System Reference Manual

OASIS Text Editor Reference Manual

OASIS EXEC Language Reference Manual

OASIS DEBUG Reference Manual

OASIS Diagnostic & Conversion Utilities Reference Manual

TABLE OF CONTENTS

Section   **Page**

TABLE OF CONTENTS

Section   **Page**

CHAPTER 1

INTRODUCTION

The OASIS MACRO Assembler (usually refered to as the Assembler) is a symbolic assembly program for the Z80 CPU. It is a two-pass assembler (requiring the source program to be read twice to complete the assembly process) designed to run under the OASIS Operating System. It is, therefore, device independent, allowing complete user flexibility in the selection of standard input and output device options.

The Assembler performs many functions, making machine language programming easier, faster, and more efficient. Basically, the Assembler processes the programmer's source program statements by translating mnemonic operation codes to the binary codes needed for the appropriate machine instruction, relating symbols to numeric values, assigning memory addresses for program instructions and data, and preparing an output listing of the program which includes any errors encountered during the assembly.

The MACRO Assembler may be used to generate either absolute or relocatable object code from a source program file. The type of object file produced is controlled by the occurence of certain directives in the source file. Both types of object programs must be processed by the LINK command before they can be executed as programs.

The value assigned to an instruction mnemonic is the binary bit configuration recognized by the processor for that instruction. Predefined symbols are kept separately by MACRO and recognized as reserved symbols only when they are encountered in the proper context. In context other than that where their usage is predefined, the symbol will assume whatever value the user may wish to assign. For example:

```
              LD      A,0
              JP      CALL
      LX:     ADD     2
      CALL:   CALL    INPUT
              JP      XXXX
```

The Assembler has no problem differentiating the two CALL symbols since the one in the op-code field is predefined and the one in the label and operand fields are user-defined.

Along with relating symbols to numbers, another major function of the Assembler is to enable the programmer to reference a symbol that is defined later in the program. This is called FORWARD REFERENCING, and is resolved by the second pass of the assembly process (some directives restrict the use of forward referencing). References may be made to symbols defined in other programs (EXTERNAL REFERENCING). The values of these symbols is resolved by the linking editor (LINK).

An optional function of the MACRO Assembler is that of producing a tabulated listing of all user-defined symbols, their value and all references to them. This CROSS-REFERENCE table generation consists of recording all definitions of, and references to, user-defined symbols, sorting the references, and merging them with their values.

Another function of the MACRO Assembler is the maintenence of up to 16 PABs (Program Address Blocks) which may be used to locate data and code at assembly time. By using PABs the programmer gains the ability to write programs whose

actual execution addresses are determined at load time (relocatable programs).

A final function of the MACRO Assembler is to maintain assembler macros, hence the name MACRO Assembler. A macro is a single user-defined instruction that is replaced at assembly time with one or more assembler instructions and/or directives.

1.1 Creating A Source File

An assembly language source file is created by using the system editor. Refer to the OASIS System Reference Manual for complete details on using the EDIT program.

Assembly language source files usually have a file type of ASSEMBLE. When the Editor is invoked and given a file description including a file type of ASSEMBLE, MACRO or COPY, the Editor sets some of its global commands to the values associated with an assembly source file. These values include setting LINEMODE OFF because line numbers are not normally used in the source file; setting TABSET 10 16 29 which allows for the standard format of source statements; setting CASEMODE AM.

CHAPTER 2

OASIS MACRO COMMAND

The MACRO command allows the user to translate Z80 source code and MACRO directives into machine object code. The format of the MACRO command is:

MACRO <file-desc> [(<option> ...[)]]

Where:

file-desc Indicates the file description of the source file to be assembled. When the file type is omitted from this description the default file type of ASSEMBLE is used.

MACRO COMMAND Options

Options for the MACRO command include the following:

NOOBJ   Indicates that no object file is to be produced.

OBJ[=fd] Indicates that an object file is to be produced. This is a default option. An fd following OBJ indicates that the object file is to be written to the disk whose directory has that label. when fd is omitted the system disk is used.

TYPE   Indicates that the listing is to be displayed on the console terminal. Specifying this option pre-sets LIST to on.

PRINTER[n] Indicates that the listing is to be displayed on the primary printer or PRINTERn. Specifying this option pre-sets LIST to on.

DISK[=fd] Indicates that the listing is to be written to a disk file with "LISTING" as the filetype. An fd following LIST indicates that the listing file is to be written to the disk whose directory has that label. When fd is omitted the system disk is used. Specifying this option pre-sets LIST to on. The listing file created will be in packed format, using ANSI forms control characters.

LIST   Indicates that LIST is to be pre-set to on. Any LIST directives in the source program may change this setting.

NOLIST   Indicates that LIST is to be preset to off. Any LIST directives in the source program may change this setting.

SYM   Indicates that the symbol table is to be included in any listing. This option may only be specified if an output device has been specified (TYPE, PRINTER, or DISK).

NOSYM   Indicates that no symbol table listing is to be produced. This is a default option.

XREF   Indicates that a cross-reference table is to be produced. This option is only effective when a listing device has been specified (TYPE, PRINTER, or DISK).

NOXREF   Indicates that no cross-reference table is to be produced. This is a default option.

MACRO   - 3 -

COPY   Indicates that the source and object code produced from COPY files included in the assembly are to be included in any listing. This option is only effective when a listing device has been specified.

NOCOPY   Indicates that the source and object code produced from COPY files included in the assembly are not to be included in the listing. This is a default option.

DATA   Indicates that data defined by DC and DW directives is to be fully included in any object code listing. This option is only effective when a listing device has been specified. In addition to the DC and DW directives this option specifies that code generated by a REPT directive is also to be included in any listing.

NODATA   Indicates that only the first four bytes of the data defined by DC and DW directives is to be included in any object code listing. Also, code generated by a REPT directive is not listed when this option is in effect. This is a default option.

MACRO   Indicates that macro expansions are to be included in any listing. This option is only effective when a listing device has been specified.

NOMACRO   Indicates that only the macro calls, not the macro expansions, are to be included in any listing produced.

IF   Indicates that source code not assembled into object code due to conditional assembly is to be included in any listing produced. This option is only effective when a listing device has been specified.

NOIF   Indicates that source code not assembled is to be omitted from any listing produced. This is a default option.

EXTRN   Indicates that all undefined symbols are to be treated as external symbol references (16 bit) and are not to be reported as errors.

NOEXTRN   Indicates that all undefined symbols are to be treated and reported as errors. This is a default option.

During the assembly process the segment names and line numbers are displayed on the console during both passes, unless the option TYPE was specified.

CHAPTER 3

MACRO INSTRUCTION SYNTAX

An assembly language program consists of a sequence of statements in the assembler language. Each statement is written on one line, and terminated by a carriage return. The MACRO Assembler is a free format assembler in the sense that the various statement elements are not placed at specific numbered columns on the line. The Editor does have default tab settings for the elements but these are used only for purposes of consistency of the listing and are not required by the Assembler.

There are four elements in an assembler statement, separated from each other by specific characters. These elements are identified by their order of appearance in the statement and by the separating (delimiting) character which follows or proceeds the elements. Statements are written in the general form:

line# label: op-code operand1,operand2 ;comment

Not all of the elements are required for any specific instruction.

3.1 Line Numbers

The line number field is completely optional. The Assembler will create line numbers for the source statements if there are no line numbers on the statements. When a line number is included on the source statement it must: be the first field, use only digits, and be followed by a space character.

Line numbers may be used for some of the source statements and not others. The Assembler, when an unnumbered line is encountered, adds one to the line number used for the previous statement. This facilitates identifying the lines associated with a multi-segmented source file. The first line of each segment would be numbered by the programmer and the following lines would be unnumbered.

3.2 Labels

The label field is an optional field that may be specified with any or all of the op-codes and directives. When used the label field must be the first field in the source line (following the line number, if used).

Labels are used to reference a specific location during assembly. A label may be used on a line that is not referenced or even on a line by itself.

A label is a sequence of one or more characters terminated by a colon (:). A label must start with a letter character (local labels are an exception to this rule) and may include only letters, digits, and the dollar sign ($). No embedded spaces are allowed.

Labels longer than eight characters are tokenized internally to eight characters by taking the first four characters and the last four characters of the actual label. It is possible that this may cause a duplicate label error.

The dollar sign character ($) used as a label by itself in the operand field, is valid and indicates the current location counter.

Labels are of three types: global, local and external. A global label must be uniquely defined within a source program and may be referenced from anywhere in the program.

A local label may be duplicately defined within a source program but must be uniquely defined between two global labels, and has a value only between those two global labels. Local labels are identified by preceding the label with a period (.). All references to a local label must include the preceding period. The character following the period in a local label must be alphabetic. Macro local labels only have a value in the macro defining them. Macro local labels are identified by preceding the label with an at-sign (@). All references to a macro local label must include the preceding at sign and these references may only be from within the macro defining them.

Local labels are maintained internally in the assembly process by appending the most recent global label to the local label (macro labels are maintained by appending the macro name and macro index to the label).

External labels are labels whose value is defined outside of the source program. The values of these external labels are resolved by the Link program.

Examples:

```
        LABEL               DONE
        LST05               OBJECT
        NAME                .OKAY
        OKAY                M0000010
        NOOBJECT            .OUTPUT
        .INPUT              @LABEL
```

3.3 Op-codes

The op-code field of a source statement may only include the directives and Z80 op-codes described in this manual. An op-code is separated from a label by the colon. An op-code is separated from any operands by a space or tab character.

Op-codes must be spelled exactly as specified in this manual.

3.4 Operands

Operands modify the op-codes and provide the information needed by the assembler to perform the designated operation. Certain symbolic names are reserved as key words in the assembly language operand fields. These are:

1. The contents of 8 bit registers are specified by the character corresponding to the register names. The register names are A, B, C, D, E, H, L, I, R.

2. The contents of 16 bit double registers and register pairs consisting of two 8 bit registers are specified by the two characters corresponding to the register name or register pair. The names of double registers are IX, IY and SP. The names of register pairs are AF, BC, DE, and HL.

3. The contents of the auxiliary register pairs consisting of two 8 bit registers are specified by the two characters corresponding to the register pair names followed by an apostrophe. The auxiliary register pair names are AF´, BC´, DE´, and HL´. Only the pair AF´ is actually allowed as an operand, and then only in the EX AF,AF instruction.

4. The state of the five flags is testable as follows:

```
FLAG            ON              OFF
=========================================
Carry           C               NC
Zero            Z               NZ
Sign            M (minus)       P (positive)
Parity          PE (even)       PO (odd)
Overflow        V               NV
```

3.4.1 Expressions

Expressions in the operand may be simple or complex. A simple expression is an expression that includes only one term. A complex expression includes more than one term with logical, arithmetic, or relational operators joining them.

Expressions are allowed as operands whenever the symbols n, nn, or d are used in the syntax of the instruction.

Expressions are analyzed in a left to right manner with no implied hierarchy except that specified by parentheses or brackets.

Expressions wholly contained within parentheses are evaluated as an indirect address reference. Expressions that contain sub-expressions in parentheses or brackets are evaluated as indicating a hierarchy of evaluation. Parentheses and brackets are equivalent but not interchangable, that is, they must appear in pairs.

A term in an expression may be any one of the following:

    label   Indicates the current value of the specified label.

    numeric-constant May be any unsigned numeric value less than 65536
            expressed in decimal (default or terminate with D), hexadecimal
            (terminate with the character H), octal (terminate with the
            character Q or O) (Q assumes number is 16 bit octal; O assumes
            number is two 8 bit octal numbers), binary (terminate with the
            character B). All numeric constants must have a digit for the
            first character. If necessary a hexadecimal value may have a
            leading 0 such as 0FFFFH.

    string-constant One or two ASCII characters enclosed within a pair of
            single or double quotes. (Storage definition directives DC and
            DW allow longer strings.)

The arithmetic operators allowed by the assembler include:

    + Addition or unary plus
    - Unary minus or binary subtraction (two's complement)
    / Division
    * Multiplication

The logical operators allowed by the assembler include:

.AND. Logical and, bit by bit
.OR. Logical inclusive or, bit by bit
.XOR. Logical exclusive or, bit by bit
.NOT. Unary one's complement
.MOD. Modulo (remainder function)
.SHL. Logical shift left (vacated bits replaced by 0)
.SHR. Logical shift right (vacated bits replaced by 0)

The relational operators allowed by the assembler include:

.EQ. Test equality, arithmetic or string - both must be same
.GT. Test for greater than
.LT. Test for less than
.UGT. Test for unsigned greater than
.ULT. Test for unsigned less than
.NUL. Empty string or value.

The following examples represent typical expressions:

```
BASE+2100H
'A'-1
(LSTDSK)
LNKLIT+8            8 is a decimal number
256                256 is a decimal number
.LABEL1            refers to the local label
$                  indicates the current location counter
'"'
1232560            evaluated as: 0101001110101110B or 53AEH
123256Q            evaluated as: 1010011010101110B or A6AEH
10110101B+324-12H/2Q   evaluated as: 243 decimal or F3H
123+(45D-LABEL)
(123+(45D-LABEL))
(123+[45D-LABEL])
```

Further restrictions in the use of expressions are discussed in the chapter on PABs.

3.5 Comments

Comments may be included on any source line. To indicate a comment use the semi-colon character (;). All characters after the semi-colon will be ignored by the assembler except for listing purposes. A comment may start in any column.

PROGRAM ADDRESS BLOCKS (PABS)

The concept of Program Address Blocks (PABs) may be used extensively when programming with the MACRO Assembler. A PAB is a name assigned to an address (either relative or absolute) that is referenced in a source program to define the relationship between groups of code (instructions and/or data).

The use of PABs allows the programmer to write several modules of code (each module probably performing a small, but complete, function), each module defining the instructions and data that it requires and accessing data defined in other modules, that, when linked together, form a contiguous program and data blocks.

Absolute programs must use an absolute PAB. When no PAB is defined in a program the Assembler assumes that a relocatable PAB is implied.

Normally a relocatable program would only have one or two PABs defined. One PAB would probably contain nonvolatile instructions and another PAB containing nonvolatile and volatile data to be used by the program. However, a complex program such as an operating system or compilor might use several PABs. In this complex program the PABs would be differentiated by major functions such as device drivers, logical I/O, arithmetic package, etc.

There are essentially three types of PABs.

The Absolute PAB

The absolute PAB is the Program Address Block that most assembly language programmers are familiar with. An absolute PAB is one whose base address is assigned by the programmer using the ABS directive. Symbols, instructions, and data defined using an absolute PAB can be completely resolved by the Assembler into executable machine code.

Instructions assembled in an absolute PAB can only be executed when the instructions are loaded at the address they were assembled at (unless the programmer uses position independant programming methods).

Programs that use an absolute PAB may only have the one PAB defined.

Different segments of code using the same absolute PAB name would, when linked together, form a contiguous block of code, each segment being appended to the previous.

The Relocatable PAB

The relocatable PAB is one whose base address is assigned by the program loader at load time. Symbols, instructions, and data defined using a relocatable PAB are only completely resolved when the program is loaded into memory for execution. The relocatable PAB allows the user to write, assemble, and link programs that can be executed at any address they may be loaded at.

A relocatable PAB is defined by the programmer using the REL directive. More than one relocatable PAB may be defined and used in a program. (The MACRO Assembler allows sixteen PABs per assembly, the LINK program allows 128 PABs per load module.

Different segments of code using the same relocatable PAB name would, when linked together, form a contiguous block of code, each segment being appended to the

previous.

The Common Relocatable PAB

The common relocatable PAB is very similar to the relocatable PAB. Its base address is assigned by the program loader at load time and the symbols, instructions, and data defined using a common relocatable PAB are only completely resolved when the program is loaded into memory for execution.

A common relocatable PAB is defined by the programmer using the COM directive. More than one common relocatable PAB may be defined and used in a program.

The difference between a relocatable PAB and a common relocatable PAB is that when different segments of code, using the same common relocatable PAB name, are linked together the code from one segment overlays the previous segment's code.

This type of PAB is very useful for buffer definitions where several modules use the same memory area for volatile working storage. Each segment would define the layout of the buffer with the specific symbols and locations that it requires. This sounds like the same result as using the EQU directive and in fact might produce the same results. However, when the common relocatable PABs are used no one segment would have to allocate the maximum buffer size that would be used: the Linkage editor would create the PAB as large as required by the segment that defined the largest area. Mainly, when a common relocatable PAB can be used and is called for in the design of the program, it results in a more easily coded and maintained program.

The uses of PABs is probably best explained with a few examples. Rather than invent some meaningless examples at this time it would be best to look at the programs in the appendix "Program Examples".

4.1 PAB Restrictions

Programming with PABs provides more versatility and makes the programming task more dynamic but it does carry some restrictions. These restrictions are mainly related to the use of symbols that are defined in a relocatable or common relocatable PAB.

There are a few, but important, rules to keep in mind when formulating expressions containing symbols. They are:

1. All relocatable symbols have full word (16 bit) values. This means that a relocatable symbol or expression can only be used when a 16 bit value can be used.

2. The sum or difference of a relocatable symbol and an absolute symbol is a relocatable value.

3. The difference between two relocatable symbols defined in the same PAB is an absolute value.

4. The sum, difference, product, or quotient of two absolute symbols is an absolute value.

5. The difference between two relocatable symbols defined in different PABs is an error.

6. All other operations between two relocatable symbols defined in the same PAB or in different PABs is an error.

7. All other operations between a relocatable symbol and an absolute symbol are errors.

Another restriction in relation to relocatable PABs is that the execution address is not known at assembly time. This seems obvious and of little importance except when the program listing is taken into account: the addresses listed at the left side of an assembly listing are not necessarily the execution addresses!

CHAPTER 5

MACRO DIRECTIVES & PSEUDO-OPS

The OASIS MACRO Assembler provides many directives and pseudo-ops that make programming in the Z80 assembler language easier by providing a means of assigning values to labels, allocating and initializing storage, conditional assembly, linking together several source files, incorporating source files into other files, and access to powerful system subroutines incorporated in the operating system.

ABS Directive

The ABS directive defines an absolute PAB. The ABS directive, unlike the ORG directive discussed later, does not change the location counter of the instructions following - the USING directive is responsible for that. The general format of the directive is:

ABS [exp] [; comment]

Although, as indicated, the label field is required for the ABS directive the expression field is not. The label field is used by the USING directive to specify which PAB to use. The expression field, when specified, indicates the address that the PAB is to start on.

A PAB definition, such as the ABS directive, implies a USING directive following. It is not necessary for you to follow an ABS directive with a USING directive.

ALIGN Directive

The ALIGN directive can not be used when relocatable PABs have been defined.

The ALIGN directive allows the programmer to set the location counter to a user defined boundary. The general format of the directive is:

[<label>] ALIGN <exp> [;comment]

All of the terms of <exp> must have been previously defined - no forward references. <Exp> is evaluated and then the location counter is set to the value of the current location counter plus current location counter modulo <exp>. This, in effect, advances the location counter to the next <exp> boundary. For instance if the location counter is 315 and an ALIGN 256 is encountered then the location counter is set to 512.

BP Pseudo-op

The BP pseudo-op allows a break-point to be assembled into a program. The general format of the BP is:

[<label>] BP [;comment]

When assembled the BP directive occupies one byte of storage (a RST instruction). During program execution this code will cause a jump into the DEBUG program. If the DEBUG program has not been loaded then the instuction has no effect. Refer to the <u>OASIS Dynamic Debugging Reference Manual</u>.

COM Directive

The COM directive defines a common PAB. The general format of the directive is:

<label> COM [<exp>] [;comment]

As indicated, the label field is required for the COM directive, like all of the
PAB definition directives. The <exp> field, when specified, is not used for the
location of object code but only for listing purposes. A common PAB is a
relocatable PAB but differs from a relocatable PAB defined by the REL directive in
that the Linkage editor overlays common PABs of the same name instead of appending
them. When several object files are being Linked that use common PABs each of the
common PABs define the same address area, starting with relative location zero.
(DS directives in a common PAB only cause previously undefined storage in the PAB
to be set to zero.)

A PAB definition, such as the COM directive, implies a USING directive following.
It is not necessary for you to specify a USING directive immediately following a
COM directive.

COPY Directive

The COPY directive allows the programmer to specify that a sequence of code is to
be found in another source file. The general format of the directive is:

[<label>] COPY <file-desc> [;comment]

The COPY directive is not a macro! No parameter replacement is allowed. When the
COPY directive is encountered by the assembler the specified file is copied into
memory and assembled at the current location counter as if the code were included
in the source program. The copied code will be listed according to the
specifications of the LIST directive.

This directive allows the programmer to easily reference frequently used sequences
of code without entering the code in each program that references it.

When the <file-desc> only specifies a file name the default file type of COPY is
used.

DB Directive

The DB (Define Byte) directive is a synonym of the DC (Define Constant) directive
discussed next.

DC Directive

The DC directive is the most general form of the storage definition directives.
The general format of the directive is:

[<label>] DC <exp list> [;comment]

Similar to the DW directive the DC directive allows the terms in the expression
list to be forward references. Each expression is evaluated independently of the
others. The individual expressions may be string literals (enclosed in quotes), 16
bit words (enclosed in parentheses), integer values, floating point values (decimal
point specified), floating point scientific format values (decimal point and

exponent specified).

The various forms of an expression are evaluated and assembled according to the following rules:

strings   strings Each ASCII character in the quoted string is evaluated and the 7 bit code is generated, one per byte. If the quotes are double quotes (") the last byte will have bit position 7 set (1). If the quotes are single quotes (') the last byte is not altered.

words   The expression within the parentheses is evaluated identical to the DW directive.

integers The expression is evaluated with the least significant 8 bits assembled at the current location counter. Overflow error results if the high order 8 bits are not zero or FFH.

floating point floating point The expression is evaluated and the six byte value is assembled at the current location counter. Floating point values are formatted using a normalized, binary, hidden bit format, consistent with the way that BASIC maintaines its numeric variables.

The hidden bit format is used to gain one more bit of precision without using an extra bit of storage. The floating point value is formatted in two sections: one byte of signed characteristic and five bytes of signed mantissa. The value of the characteristic and mantissa is in binary, the eight bits of characteristic has a decimal range of about +38 to -38.

The value is first normalized, that is, the mantissa is first converted to a fraction less than +1, greater than -1, adjusting the characteristic accordingly. Then the mantissa is repeatedly multiplied by two (and the characteristic decremented) until the most significant "on" bit is in the most significant position. Then, since it is known that this bit is always "on" the position is occupied by the sign bit of the mantissa. This hidden bit technique implies that the hidden bit will have to be re-instated before the number can actually be used in arithmetic operations.

The following example of the conversion of a floating point number from external format to internal format illustrates the result of the hidden bit, normalized format:

```
     Value = 123.
     Binary = 0111 1011. (7BH)
     Conversion:
     Char  Mantissa
     -----  ------------------------------------------------
      00    01111011 00000000 00000000 00000000 00000000
            Normalize decimal point:
      08    01111011 00000000 00000000 00000000 00000000
            Save sign bit, set to absolute value:
      08    01111011 00000000 00000000 00000000 00000000
            Adjust MSB:
      07    11110110 00000000 00000000 00000000 00000000
            Change MSB to sign bit:
      07    01110110 00000000 00000000 00000000 00000000
```

DC Examples

```
Addr Obj-Code  Line *** Source Statement ***

0000 00          1       DC    0
0001 3412        2       DC    (1234H)
0003 4142534F    3       DC    'ABSOLUTE'
0007 4C555445
000B 4142534F    4       DC    "ABSOLUTE"
000F 4C5554C5
0013 00010241    5       DC    0,1,2,'A'
0016            6       DS    0FH
0025 FF00        7       DW    255
0027 01999999    8       DC    1.2
002B 9999
002D 0B1A5224    9       DC    1234.567
0031 DD2D
0033 1516B43F   10       DC    1234567.890123
0037 1EF5
0039 FD7CD6E9   11       DC    .123456789
003D B9C6
```

DS Directive

The DS directive allows the programmer to advance the location counter a specified amount, thus reserving a storage area. The general format of the directive is:

[<label>] DS <exp> [;comment]

All of the terms used in <exp> must have been previously defined — no forward references. <Exp> is evaluated and the location counter is advanced that many bytes.

DW Directive

The DW directive allows the programmer to define words of storage to be specific values. The general format of the directive is:

[<label>] DW <exp list> [;comment]

The expressions in the list are separated by commas. Each expression is evaluated independently of the other expressions in the list. The terms of the expressions may include forward references. Each expression is evaluated and assembled at the current location counter. The word is assembled with the least significant 8 bits (LSB) first followed by the most significant 8 bits (MSB). The location counter is advanced by two for each expression evaluated.

EJECT Directive

The EJECT directive indicates that a page eject is to be generated in the listing. The general format of the directive is:

EJECT [;comment]

The EJECT directive is only effective when a listing is being generated. The directive, when encountered, causes an immediate page eject to be generated in the listing. The EJECT directive itself is not listed in the listing, although it does advance the line number.

ELSE Directive

The ELSE directive allows the programmer to specify an alternate set of instructions to be assembled when the <exp> of an IF directive is evaluated to be false. The general format of the directive is:

[<label>] ELSE [;comment]

The ELSE directive is an extension of a prior IF (or ELSEIF) directive and therefore the ELSE directive may only be used between an IF and ENDIF ENDIF directive or between an ELSEIF and ENDIF directive.

When the <exp> of an IF or ELSEIF directive is evaluated to be false the assembler searches forward for an ELSE (or ELSEIF) directive. The instructions following the ELSE directive are then assembled. When the <exp> is evaluated to be true the instructions following the ELSE directive are not assembled.

ELSEIF Directive

The ELSEIF directive provides "case" statement conditional assembly capability. The general format of the directive is:

[<label>] ELSEIF <exp> [;comment]

When used, the ELSEIF directive must be between an IF, ELSEIF or ELSE directive and an ELSE, ELSEIF, or ENDIF directive. All terms in the <exp> must have been previously defined.

Only one ELSE statement is allowed per IF statement but there may be several ELSEIF statements following an IF statement.

During the analysis of an IF - ELSEIF...ELSEIF - ENDIF statement group assembly of source statements is suppressed until a true condition is detected for one of the IF, ELSEIF, or ELSE statements. When this occurs the statements are assembled until an ELSE, ELSEIF, or ENDIF statement is encountered--then the statements are skipped until the matching ENDIF is encountered.

Examples

```
LABEL1:  EQU    1
LABEL2:  EQU    0
LABEL3:  EQU    LABEL2*LABEL1
LABEL4:  EQU    LABEL2.AND.LABEL3
         IF     LABEL1
                            This code will be assembled
         ELSEIF LABEL2
                            This code will not be assembled
         ELSE
                            This code will be assembled
         ENDIF
         IF     LABEL3
                            This code will not be assembled
         ENDIF
         IF     LABEL4
                            This code will not be assembled
         ELSE
                            This code will be assembled
         ENDIF
                            This code will be assembled
```

END Directive

The END directive specifies the physical end of the source code. In addition this directive may specify the entry point address. The general format of the directive is:

[<label>] END [<exp>] [;comment]

It is not necessary to terminate the source program with the END directive, however, it is recommended and when used, it will be the last line of code analyzed.

When the <exp> is specified it indicates the address to be used for the entry point. That is the address at which execution will begin when the program is executed.

ENDIF Directive

The ENDIF directive is required to terminate the instructions that are to be conditionally assembled. The general format of the directive is:

[<label>] ENDIF [;comment]

Every IF directive must have a matching ENDIF directive.

ENDM Directive

The ENDM directive indicates the physical end of a macro prototype definition. The general format of the directive is:

<p align="center">ENDM [;comment]</p>

The usage of this directive is explained in the chapter on Macros.

ENTER Directive

The ENTER directive is identical to the VALUE directive except that the <exp> is entered from the keyboard during pass one of assembly. The general format of the directive is:

<p align="center"><label> ENTER [<quoted string prompt>] [;comment]</p>

When the ENTER directive is encountered during pass one of the assembly the <quoted string prompt> is displayed on the console. If the <quoted string prompt> is omitted the <label> name is displayed for prompting purposes. At this time the operator enters the expression to be assigned to <label>.

The ENTER directive must have a label. When the ENTER directive is encountered by the assembler during pass one the operator is allowed to enter the value (this value may be in the form of an expression using literal and previously defined labels. The label being defined with the ENTER directive may have been previously defined and used.

Examples

```
DEBUG:    ENTER   'Is this a debugging assembly? (Y/N)'
LABEL1:   ENTER   'Please type the value of LABEL1'
          .
          .
          IF      DEBUG.EQ.'Y'
          .
          .
          ENDIF
```

ENTRY Directive

The ENTRY directive allows you to specify that a label, defined in the current assembly, is an external reference (EXTRN) of another assembly. The general format of the directive is:

<p align="center">ENTRY <label>[,<label>]... [;comment]</p>

The list of labels may be forward references to labels defined later in the assembly but the labels must be defined at some time during the current assembly. This directive is the logical inverse of the EXTRN directive.

The ENTRY directive would be used in a module of source code that defines a label(s) whose value will be needed in another module(s) that is not to be assembled with this one but will be LINKed with the current module.

For more explanation of the use of this directive and the EXTRN directive see the

OASIS LINK Editor Reference Manual.

EQU Directive

The EQU directive allows the programmer to assign a value to a label. The general format of the directive is:

<center><label> EQU <exp> [;comment]</center>

The EQU directive must have a label. All terms in <exp> must have been previously defined - no forward references are allowed.

When the EQU directive is encountered by the assembler <exp> is evaluated and assigned to <label>.

A label that has been equated with the EQU directive may not have been defined by any other directive or instruction in the program.

ERR Directive

The ERR directive is used to display an error message during the assembly process. Normally this would be used in conjunction with the conditional directives when an invalid condition has been detected. The general format of the directive is:

<center>ERR 'message' [;comment]</center>

When the ERR directive is encounted the message is displayed on the console along with the line number and the error message is included in any listing file being generated. This directive does not cause the assembly process to be cancelled but it will cause the return code to be set to a non-zero value. This return code can be displayed when the RDYMSG has been set ON and it can be tested by an EXEC program.

EX Pseudo-ops

The EX pseudo-op provides a convenient method of expressing some frequently used register exchanges with the Z80 registers.

MACRO Pseudo-op		Equivalent Z80 Instruction	
EX	AF,AF'	EXA	
EX	AF,AF	EXA	
EX	HL,DE	EX	DE,HL
EX	BCDEHL,BCDEHL'	EXX	

As can be seen, the pseudo-ops are more graphic in their meaning and would be very useful for the programmer who is unfamiliar with the Z80 exchange instructions.

EXITM Directive

The EXITM directive is used in a macro prototype, usually in conjunction with the conditional directives, to skip to the ENDM directive. The general format of the directive is:

EXITM [;comment]

The EXITM directive is discussed in the chapter on Macros.

EXTRN Directive

The EXTRN directive allows you to specify that a label is defined externally to the current assembly. The general format of the directive is:

EXTRN <label>[,<label>]... [;comment]

The list of labels specified in the operand field cannot include any labels defined during the current assembly, either before or after this directive.

For more information regarding the use of this directive and the ENTRY directive see the OASIS LINK Editor Reference Manual.

IF Directive

The IF directive allows the programmer to include code that is assembled only when an expression is true. The general format of the directive is:

[<label>] IF <exp> [;comment]

All terms referenced in <exp> must have been defined previously in the program. No forward references are allowed.

The <exp> is evaluated and, if true, the instructions following are assembled. When the value of the <exp> is false the instructions following, up to the next ELSE, ELSEIF, or ENDIF, are not assembled.

LD Pseudo-ops

The LD pseudo-op provides a convenient method of performing some frequently used double register loads that are not available in the Z80 instruction set. The general format of the pseudo-op is:

[<label>] LD <rr>,<rr> [;comment]

[<label>] LD <rr´>,<ii+d> [;comment]

[<label>] LD <ii+d>,<rr´> [;comment]

Where:

rr   Is any of the double register pairs: BC, DE, HL, IX, or IY.

rr´   Is any of the double register pairs: BC, DE, or HL.

ii   Is either of the index register pairs: IX or IY.

d   Is a signed displacement value.

The LD pseudo-op is the same op-code as the Z80 LD instruction except in its permissible syntax. The LD pseudo-op generates the corresponding instructions to perform the desired load. For example- the pseudo-instruction: LD HL,DE will generate the Z80 instructions: LD H,D and LD L,E.

LINK Directive

The LINK directive provides a means of segmenting the source program into more workable units. The general format of the directive is:

<p align="center">LINK <file-desc> [;comment]</p>

When the LINK directive is encountered by the assembler the specified file is used for the next line of source code. Obviously the LINK directive should be the last line of code in the current file as any code following the LINK directive will be ignored.

When the <file-desc> only specifies a file name the file type used in the OASIS MACRO command is used - that command had a default file type of ASSEMBLE.

LIST Directive

The LIST directive specifies how (and if) the assembler is to list the source program. The general format of the directive is:

<p align="center">LIST [<option list>] [;comment]</p>

The LIST directive is only effective when one of the listing output options was specified in the OASIS MACRO command. The LIST directive may be used more than once in a source program to change the listing options. Similar to the USING and ORG directives, when the option list is specified the current list options are pushed onto an 8 level, internal LIST stack. When the option list is omitted the previous list options are popped from this LIST stack.

The options that may be specified include:

ON   Indicates that a listing is to be created.

OFF   Indicates that no listing file is to be created.

COPY   Indicates that code found in a "COPY" file is to be included in the listing.

NOCOPY   Indicates that code found in a "COPY" file is not to be included in the listing. This option does not affect the object program generated.

IF   Indicates that source code not assembled due to conditional assembly is included in the listing.

NOIF   Indicates that source code not assembled due to conditional assembly is not included in the listing.

DATA   Indicates that all data generated by the storage definition directives is to be included in the listing.

NODATA   Indicates that only the first four bytes of data generated by each storage definition directive is to be included in the listing.

MACRO   Indicates that macro expansions are to be included in the listing.

NOMACRO   Indicates that macro expansions are not to be included in the listing.

The options specified in the CSI MACRO command initially set the various list options, however (assuming a listing output device was specified) the LIST directive may override these options.

MACLIB Directive

The MACLIB directive allows the programmer to specify that a file of macro definitions is to be located and remembered. The general format of the directive is:

[<label>] MACLIB <file name>[.<file type>][:<file disk>][;comment]

When the MACLIB directive is encountered by the assembler the specified file (default file type of MACLIB) is located. The macro definitions contained in the file are noted and the macros may be used by the program just as if the macro were defined by the program.

No listing of the MACLIB file will be produced. The MACLIB file may only contain macro definitions.

MACRO Directive

The MACRO Directive specifies that the code following (up to and including the ENDM directive) is a macro prototype definition. The general format of the directive is:

MACRO [;comment]

The MACRO directive, along with the other macro related directives, is discussed in the chapter on Macros.

ORG Directive

The ORG directive allows the programmer to change the value of the location counter. This location counter is used to determine the address at which to assemble the next instruction. The general format of the directive is:

[<label>] ORG [<exp>] [;comment]

The ORG directive always specifies that the location counter is to be changed. When the ORG directive is encountered in an ABS PAB the expression specifying the new location counter is absolute. When the ORG directive is encountered in a REL or COM PAB the expression specifying the new location counter must be a relocatable expression.

All of the terms in <exp> must have been previously defined - no forward references are allowed. When the ORG directive is encountered <exp> is evaluated and assigned to the location counter and <label>, when specified.

When <exp> is specified with the ORG directive the current location counter is placed on an internal 8 level ORG stack. When <exp> is omitted the previous element on the internal ORG stack is popped off.

This feature allows the programmer to place the code defining the working storage near the code referencing this storage even though in fact the address of the working storage may be any place in memory.

For example:

```
Addr Obj-Code  Line *** Source Statement ***

                  1 MAIN:    ABS
4000              2          ORG    4000H
4000  320090      3          LD     (LABEL1),A
4003  E3          4          EX     (SP),HL
4004  7E          5          LD     A,(HL)
4005  23          6          INC    HL
4006  E3          7          EX     (SP),HL
4007  FE45        8          CP     VALUE2
4009  3805        9          JR     C,LABEL2
400B  3A0090     10          LD     A,(LABEL1)
400E  37         11          SCF
400F  C9         12          RET
9000             13          ORG    9000
9000  0000       14 LABEL1:  DC     (0)
9002  00         15 LABEL3:  DC     0
4010             16          ORG
4010             17 LABEL2:
```

REL Directive

The REL directive is used to define the relocatable PAB. The general format of the directive is:

[<label>] REL [<exp>] [;comment]

Unlike the ABS directive, the label field is not required when there is only one REL PAB in a program. When the label field is omitted the PAB will be assigned the name of the program. When the label field is specified it is used by the USING directive to specify which PAB to use for assembling code. The <exp> field, when used, specifies an address relative to the load address of the program that the PAB is to start on for listing purposes only. Obviously, since this defines a relocatable PAB, the actual addresses used during execution time may be different.

A PAB definition, such as the REL directive, implies a USING directive following. It is not necessary for you to specify a USING directive immediately following a REL directive.

REPT Directive

The REPT directive allows you to duplicate a line of source code several times without coding several times. The general format of the directive is:

REPT [<exp>] [;comment]

When the REPT directive is encountered by the Assembler the next sequential line of code will be duplicated the number of times specified by <exp>. <exp> must be in the range of 1 - 65535. No forward referencing is allowed.

The line that follows the REPT directive cannot have a label in the label field as that label would be duplicated along with the rest of the code. This, of course, would cause a duplicate label error.

The listing of the duplicated lines of code is controlled by the DATA/NODATA option of the LIST directive.

SC Pseudo-op

The SC allows the assembly language programmer to utilize various portions of the operating system. The general format of a System Call is:

[<label>] SC <exp> [;comment]

The <exp> specifies which system routine control is to be transferred to. Although <exp> may have a value between 0 and 255 the actual number of system routines implemented is less. Reference to a system call number not implemented will cause system call number 0 to be executed. SC 0 will cause control to return to the OASIS operating system.

When assembled the SC occupies two bytes of storage.

The system routines implemented and the requirements for usage are discussed in the chapter on System Calls.

SUBT Directive

The SUBT directive allows the programmer to specify a sub-heading to be printed on each page. The general format of the directive is:

SUBT <quoted string> [;comment]

The <quoted string> replaces the second heading line message at the top of each subsequent page of the listing.

TITLE Directive

The TITLE directive allows the programmer to specify the heading to be printed at the top of each page in the listing. The general format of the directive is:

TITLE <quoted string> [;comment]

The TITLE directive is only effective when a listing is being generated. When the TITLE directive is encountered by the assembler the heading for the next page of the listing is changed to be the <quoted string> (exlusive of the delimiting quotes) and a page eject is generated in the listing. The TITLE directive itself is not listed in the listing, however the line number is incremented.

USING Directive

The USING directive is used in conjuntion with the ABS, COM or REL directives to specify the PAB that instructions following belong to. The general format of the directive is:

USING [<label>] [;comment]

The USING directive can not have a label. The label specified in the operand portion of the directive must be of a previously defined PAB (no forward references). When label is used in the operand position the current "USING PAB" is pushed onto an 8 level USING stack. When <label> is omitted the last "USING PAB" is popped from this USING stack.

When a PAB is defined by the ABS, COM, or REL directive a USING directive is implied. There is no need for you to follow a PAB definition with a USING directive unless you wish to specify some code "using" a different PAB than the one just defined. This implied USING performs a push onto the USING stack just as if you had specified the USING directive yourself. In fact, when you specify the USING directive following a PAB definition there will be two pushes onto the USING stack.

VALUE Directive

The VALUE directive is similar to the EQU directive with the added ability of redefining a previously defined label in the program. The general format of the directive is:

<label> VALUE <exp> [;comment]

The VALUE directive must have a label. All terms in <exp> must have been previously defined - no forward references are allowed.

CHAPTER 6

MACROS

Macros are predefined sections of source code which may be used to facilitate the coding of commonly used procedures. Macro source code is modified by the MACRO Assembler to include labels and expressions passed as arguments by the main body of source statements. Macro definitions are called "Macro Prototypes" and are saved for later access by the MACRO Assembler.

The OASIS MACRO Assembler allows macro prototypes to be defined either within a source file (must be defined before referenced), in an external macro source file (file type of MACRO, one per file), in an external macro library file (file type MACLIB, one or more per file), or in a COPY file that was copied before the macro was referenced.

6.1 Preparing Macro Prototypes

Macro prototypes must be defined in the following format:

```
          MACRO    [;comment]
[&<label>]  name    [&<symbol>[(<default>)]][,&<symbol>[(<default>)]]...
   one or more assembly language statements and macro directives
          ENDM
```

Each prototype must start with the MACRO directive and end with the ENDM directive.

The second statement of each prototype is called a "Macro Prototype Header" and defines the name of the macro and any labels and symbols that may be replaced during assembly. The name may be any 1 to 8 character symbol that is not already predefined by the MACRO Assembler (Z80 op-codes and MACRO directives). All arguments shown in brackets are optional and may be omitted if not needed.

Notice that the label and symbols are preceded by the ampersand character. This is also true of the assembly statements within a macro. The ampersand character always precedes a substitution label or symbol.

Labels and symbols shown in the prototype header define items in the statements that follow that may be replaced at assembly time. Following each symbol in the header a default expression may be defined. The default will be used if a macro reference in the source program fails to supply a replacement expression for the preceding symbol. Spaces or commas may be used to separate the times in the list.

More than one macro may be defined in a program.

6.2 Macro Calls

Code from a macro prototype is included in assemblies by the means of "macro calls". The general form of macro calls is:

[<label>] name [<exp>[,<exp>]...] [;comment]

The name used in the instruction field will be assumed to be a macro name if it is not a recognizable MACRO Assembler instruction mnemonic or directive. The label and expression arguments in brackets are optional. Arguments defined in the expression field are positional and must be defined in the same order as related symbols in the macro's prototype header (except keywords).

Notice that a macro call does not use the ampersand character.

There is a purposeful similarity between the format of a macro call and macro prototype header. They are closely related and determine the final code that will be included in the assembly.

```
Header:  [&<label>] name [&<sym>[(<def>)] [,&<sym>[(<def>)]...]]
Call:    [<label>] name [<expression>  [,<expression>]...]
```

The label for the call will replace the occurrences of the header label in prototype code during expansion. The first expression in the call will replace the first header symbol in the prototype code, the second expression will replace the second symbol, and so forth.

Arguments may be omitted in each list of macro call expressions by coding only the trailing comma to indicate the missing expression. Trailing commas after the last expression included in a list are not required.

The rules for substitution are:

Macro Call	Prototype Header	Action
Label	No label	Label is defined normally before expanded macro code is processed.
Label	Label	Call label substituted in expanded macro code.
No label	No label	No change.
No label	Label	Prototype label is omitted.
Symbol	No symbol	Call symbol ignored.
Symbol	Symbol	Call symbol substituted for occurrences in macro code.
No symbol	Symbol-no default	Header symbol disappears in expanded code.
No symbol	Symbol-default	Default substituted for occurrences in macro code.

Expansion example:

Macro prototype:

```
            MACRO
&LABEL:     CLEAR   &FIELD,&SIZE(80)
                    ; Clear &FIELD to zeros for length &SIZE
&LABEL:     LD      B,&SIZE             ; Get field length
            LD      HL,&FIELD           ; Point to &FIELD
LOOP:       LD      (HL),0              ; Set byte to zero
            INC     HL                  ; Point next
            DJNZ    LOOP                ; Repeat till done
            ENDM
```

Macro Call:

```
LOOP:       CLEAR   BUFFER
```

Expansion:

```
                        ; Clear BUFFER to zeros for length 80
        LOOP:   LD      B,80            ; Get field length
                LD      HL,BUFFER       ; Point to BUFFER
        LOOP:   LD      (HL),0          ; Set byte to zero
                INC     HL              ; Point next
                DJNZ    LOOP            ; Repeat till done
```

In the above example the symbols &LABEL and &FIELD in the prototype have been replaced by "LOOP" and "BUFFER" provided by the macro call. The symbol "SIZE" did not have a replacement expression in the macro call so the default "80" was substituted.

6.3 Macro Keywords

The MACRO Assembler provides an alternate format for prototype headers and macro calls to allow easier implementation of macros with long symbol lists. This alternate format uses the keyword feature.

As described above the symbols in a prototype header and a macro call are positional, meaning that a one to one match is made between the first symbol defined in the header and the first position of the call, the second symbol defined in the header and the second position of the call, etc.

When the keyword feature is used the symbols are no longer positionally defined and called. This is important when a long list of symbols and defaults are defined in a header but only a few are used in the call.

A symbol is defined as a keyword in a macro call by using the symbol with an equal sign (=) followed by the value.

Example:

Macro Prototype:

```
                MACRO
        &LABEL: TEST    &A(1),&B(2),&C,&D(256),&E(0),&F(5),&G(1)
        &LABEL: DC      &A,&B
                DC      &C
                DS      &D
                DC      &E,&F,&G
                ENDM
```

Macro Call:

```
        VALUE:  TEST    ,5,5,G=128
```

Expansion:

```
        VALUE:  DC      1,5
                DC      5
                DS      256
                DC      0,5,128
```

6.4 Labels

Labels within a macro are of three types: global, local, and macro local. The global label within a macro functions the same as it does outside of a macro: it can be referenced from anywhere in a program. A global label defined within a macro is different from a global label defined outside of a macro in that the definition of the global label does not affect local labels.

The local label defined within a macro functions the same as it does outside of a macro: it can only be referenced from locations between two global labels (global labels defined outside of the macro.

The macro local label is a label that has a value only when reference from within the macro defining it. A macro local label is a label whose first character is a @.

6.5 Concatenation

The concatenation character, vertical bar (|) is used in inner macro calls and macro prototype expressions to separate a macro symbol from a literal that is to be concatenated to the replaced value of the symbol. Macro symbols may be concatenated by merely concatenating the symbol references in the prototype.

Example:

Macro prototype:

```
          MACRO
          MSG        &AAA,&BBB
MSG&AAA:  DC         &BBB|LOC,&BBB|SIZE
          DC         'ERROR IN PHASE DCT&AAA'
          DC         (&BBB&AAA)
          ENDM
```

Macro Call and Expansion

```
          MSG        024,PHS4

MSG024:   DC         PHS4LOC,PHS4SIZE
          DC         'ERROR IN PHASE DCT024'
          DC         (PHS4024)
```

In order to include the vertical bar character as part of a macro or macro call you must duplicate it: ||

6.6 Macro Substrings

Substrings of macro variables can be used by specifying the starting and ending character positions of the variable, within parenthesis, immediately following the variable name. For example: &NAME(3,5) indicates the substring of the value of the variable &NAME from position three through position five (three characters). Any time a variable name is used followed by a left parenthesis character the assembler will try to substing the variable. When the left parenthesis character is used and substringing is not desired you must use the concatenation character described above.

6.7 Macro Nesting

The OASIS MACRO Assembler allows the nesting of macro calls within macro calls up to eight levels deep. Macro Local labels cannot be passed as arguments to inner macros. Local labels may be passed as arguments to inner macros but this usage may be restricted by the definition of global labels (same as non-macro code). The passage of global labels and other arguments is unrestricted.

6.8 Macro Reserved Variables

Within a macro prototype or macro call there are four reserved variables available to the user. These variables allow you to access the current date and time, the program name, and the current macro index value. If these variables are to be used as labels then they should be concatenated with other characters to generate unique labels. The variables are as follows:

&DATE   current date in mm/dd/yy format
&TIME   current time in hh:mm:ss format
&PROG   current source program name
&INDEX   current macro call index number

6.9 Macro Comments

Comments may be included in a macro prototype in the same manner as comments in the main program. Macro symbols may be included as part of a comment and these will be expanded.

A comment may be included in a macro prototype that is not to be expanded or even listed in any listing file created. This type of comment (macro comment) is indicated by pairing the comment delimiter (;;).

6.10 Macro Example

```
          MACRO                 ; Create FCB
&LABEL:   FCB   &CHANNNEL,&MODE,&BUFFER
          IF    .NUL.&LABEL     ; Asm only if &LABEL is empty
          ERR   'Label field required for FCB'
          EXITM
          ENDIF
          IF    &CHANNEL.LT.0.OR.&CHANNEL.GT.16
          ERR   'ACB channel number out of range'
          EXITM
          ENDIF
&LABEL:   DC    &CHANNEL
          IF    .NUL.&MODE
          ERR   'Access mode required'
          EXITM
          ELSE                  ;; Test the access mode specified
          IF    '&MODE(1,3)'.EQ.'INP'
          DC    90H
          ELSEIF '&MODE(1,1)'.EQ.'O'
          DC    88H
          ELSEIF '&MODE(1,1)'.EQ.'D'
          DC    40H
          ELSEIF '&MODE(1,3)'.EQ.'IND'
          DC    20H
          ELSE
          ERR   'Access mode undefined'
          EXITM
          ENDIF
          ENDIF
          IF    '&BUFFER'.EQ.''
          DC    (@BUFF)
@BUFF:    DS    255
          ELSE
          DC    (&BUFFER)
          ENDIF
          ENDM
```

CHAPTER 7

SYSTEM CALLS

This chapter describes all of the system calls implemented in this version of the MACRO Assembler. They are described because they do exist and are available for use, not because they should be used by the programmer. In fact, some of these system calls should not be used: 10, 11, 27, 28, 50, and 51. These system calls are related to physical disk I/O and, if used indiscreetly, may destroy the resident operating system or the contents of a disk or disks. Any consequential damages caused by the use of these specific system calls are the responsibility of the user.

7.1 Documentation Conventions

This chapter describes the syntax and operation of the system calls available to the programmer using the OASIS MACRO Assembler. Each system call is presented in the same format:

1. System call heading, centered on the page.

2. Function of the system call.

3. Input parameters. This area defines all of the parameters that are required to be defined before the system call is invoked.

4. Output parameters. This area defines any parameters that are returned to the calling program.

5. Description. A general descriptive text of the function of the system call.

6. Other system calls used. This area specifies if any other system calls are used to perform the function and what they are.

7. Other registers altered. Any registers that may be changed by the system call, excluding those specified as output parameters, are listed in this area.

8. Example. A specific example of the calling sequence and result of the system call is given. An example is not given if the system call is obvious or trivial.

System control blocks are referenced frequently through this chapter. Refer to the appendix on System Control Blocks for information regarding the content and format of each of the control blocks.

==
SC 0 QUIT
==

Function: Reload the Command String Interpreter - restart.

Input parameters:

> Reg A - Return Code

Output parameters: none

Description:

> The Command String Interpreter is reloaded, the system stack is reset and
> control is passed to the CSI. This system call is generally used when an
> assembly program is finished its execution and control is to return to the
> operating system.

> The value in the A register is the return code. This return code is
> displayed if RDYMSG is set ON and is accessible by the EXEC language.

Other system calls used: MOUNT (9), RD1 (10), CRLF (18), LOOKUP (20), FETPRG (30)

Other registers altered: all (control returns to operating system)

Example Calling Sequence:

```
        LD      A,16        ; Return code
        SC      0           ; Re-load CSI & exit
        END
```

==
SC 1 KEYIN
==

Function: Accept a line of input from the console keyboard.

Input parameters:

> Reg B - Max line length to accept
> Reg DE - Address of buffer to store line

Output parameters:

> Reg A - Actual line length accepted

Description:

> Up to B characters are accepted from the console input device (CONIN).
> All characters will be echoed to the console output device, dependent upon
> the controls set in the console control byte. Entry is terminated by
> entry of B characters or a carriage return. (The console control byte may
> specify that any control character terminates input.) When the input is
> terminated a carriage return, line feed is echoed to the console output
> device.

If the input is not terminated by a carriage return (B characters entered) then a carriage return is appended to the end of the character string in the buffer. For this reason the buffer length should be B+1.

Other system calls used: CONIN (4), CONOUT (5)

Other registers altered: C, D, E, H, L

Example Calling Sequence:

```
            LD      B,64        ; Length
            LD      DE,AREA     ; Input buffer
            SC      1           ; Get line from console
            .
            .
            .
AREA:       DS      65          ; Buffer
```

==
SC 2 DISPLAY
==

Function: Display characters on console output device.

Input parameters:

Reg DE - Address of first character to output

Output parameters:

Reg DE - Address of last char output plus one

Description:

Characters from the buffer addressed by register pair DE are displayed on the console output device. A null character (00) terminates output to the console and returns from the system call.

A carriage return will be displayed as a carriage return, line feed and the system call will be exited. A line feed will be displayed as a carriage return, line feed, output continues. An HT character (09H) will be displayed as the proper number of spaces according to the Tab Set Block (TSB).

Other system calls used: CONOUT (5), CRLF (18)

Other registers altered: A

Example Calling Sequence:

```
            LD      DE,MSG      ; Point to message string
            SC      2           ; Display on console
            .
            .
            .
MSG:        DC      'Any old thing',0DH
```

===
SC 3 CONST
===

Function: Get status of console input device.

Input parameters: none

Output parameters:

           Flag Z - set if no character ready; reset otherwise

Description:

           The console input device is queried: the zero flag (Z) is reset if at
           least one character is available for input, the zero flag is set if no
           characters are available.

Other system calls used: none

Other registers altered: A

Example Calling Sequence:

           SC   3   ; Test console ready
           JR   Z,NOTRDY   ; Jump if no char ready

===
SC 4 CONIN
===

Function: Accept one character from the console input device.

Input parameters: none

Output parameters:

           Reg A - contains character input

Description:

           One character is accepted from the console input device. Return from this
           system call is performed only after a character is accepted. The
           character will be echoed to the console output device with editing
           performed according to the switches set for upper/lower case, rubout,
           graphic display, etc. This system call never echos control characters
           (values <32 or > 128).

           Note: When there is information available from the EXEC stack this system
           call will retrieve a character from that stack and echo it to the console
           if the stack display switch is in effect.

Other system calls used: CONOUT (5)

Other registers altered: none

Example Calling Sequence:

```
        SC      4                       ; Read & echo char from console
```

===
SC 5 CONOUT
===

Function: Display one character on console output device.

Input parameters:

        Reg C   - character to be displayed

Output parameters: none

Description:

        The character contained in register C is displayed on the console output
        device (CONOUT) with editing performed according to the console control
        byte: graphics, printer echo, etc.

        Note: When there is information available from the EXEC stack this system
        call will retrieve a character from that stack and echo it to the console
        if the stack display switch is in effect.

Other system calls used: PRTOUT (8)

Other registers altered: A

Example Calling Sequence:

```
        LD      C,'?'                   ; Load a question mark
        SC      5                       ; Display on console
```

===
SC 6 SYSIN
===

Function: Accept one character from console.

Input parameters: none

Output parameters:

        Reg A   - contains character input

Description:

        One character is accepted from the console input device. Return from this
        cystem call is performed only after a character is accepted.   The
        character will always be echoed to the console output device (status of
        Console Echo-key ignored) with editing performed according to the switches
        set for upper/lower case, rubout, graphic display, etc. The character
        will never be echoed to the printer device (status of Printer Echo-key
        ignored). This system call never echos control characters (values < 32 or

> 128).

The status of the EXEC stack and the stack display switch is ignored by this system call (character is <u>always</u> accepted from CONIN and displayed on CONOUT).

Other system calls used: SYSOUT (7)

Other registers altered: none

Example Calling Sequence:

```
        SC      6              ; Get char from CONIN
```

===
SC 7 SYSOUT
===

Function: Display one character on console output device.

Input parameters:

Reg C -- character to be output

Output parameters: none

Description:

The character contained in register C is diaplayed on the console output device (CONOUT) with editing performed according to the console control byte: graphics, etc. The status of the Console Echo-key and the Printer Echo-key is ignored.

The status of the EXEC stack and the stack display switch is ignored (character is <u>always</u> displayed on the CONOUT).

Other system calls used: none

Other registers altered: A

Example Calling Sequence:

```
        LD      C,12H          ; Load DC2 char
        SC      7              ; Output to console
```

===
SC 8 PRTOUT
===

Function: Output one character to Printer 1.

Input parameters:

Reg C -- character to be output

Output parameters: none

Description:

    If Printer 1 is not attached then this system call is exited. If the printer is attached then the character in the C register is output to that device along with any editing or options specified in the attachment of that device.

Other system calls used: none

Other registers altered: A

Example Calling Sequence:

```
        LD      C,0CH       ; Form feed
        SC      8           ; Output to PRINTER1
```

===
SC 9 MOUNT
===

Function: Allow change of diskette on a specified drive.

Input parameters:

    Reg B — logical drive code (0 – 7) = (S – G)

Output parameters: none

Description:

    Internal switches are set to indicate that the next read or write to this disk must first read the diskette ID. If the drive code in the B register specifies a drive that is not attached or is invalid then nothing is done by this system call.

Other system calls used: none

Other registers altered: A

Example Calling Sequence:

```
        LD      B,1         ; Drive code for A
        SC      9           ; Perform mount on A
```

===
SC 10 RD1
===

Function: Read one sector from a diskette.

Input parameters:

                Reg B - logical drive code (0 - 7) = (S - G)
                Reg DE - sector address, relative to 0
                Reg HL - buffer address

Output parameters: none

Description:

        Specified drive is selected, if legal, and the indicated sector is read
        into the location specified by the HL register pair. If the drive or
        sector is illegal or an error is detected during the read no error status
        is returned--disk errors are reported to the operator for handling (see SC
        74).

        This system call, when used in a multi-user environment, checks the Sector
        Lock Table (SLT) and waits if the requested sector is locked by another
        partition.

        <u>Caution</u>: Use of this system call is not advised.

Other system calls used: RD (50)

Other registers altered: A, C

Example Calling Sequence:

                LD   B,0   ; Drive S
                LD   DE,1   ; Sector 1
                LD   HL,BUFFER   ; Memory address
                SC   10   ; Read a sector
                •
                •
                •
    BUFFER: DS   256

===
SC 11 WR1
===

Function: Write one sector to a disk.

Input parameters:

                Reg B - logical drive code (0 - 7) = (S - G)
                Reg DE - sector number, relative to 0
                Reg HL - buffer address

Output parameters: none

Description:

> The specified drive, if legal, is selected and the data at the location indicated by register pair HL is written to the specified sector. If the drive or sector number is illegal or an error is detected during the write operation no error status is returned—disk errors are reporated to the operator for handling (see SC 74).

> This system call, when used in a multi-user environment, checks the Sector Lock Table (SLT) and waits if the requested sector is locked by another partition.

> Caution: Use of this system call is not advised.

Other system calls used: WR (51)

Other registers altered: A, C

Example Calling Sequence:

```
            LD      B,1         ; Drive A
            LD      DE,(SECT)   ; Sector address
            LD      HL,DMA      ; Memory address
            SC      11          ; Write a sector
            •
            •
            •
SECT:       DC      (112)       ; Must be 16 bit word
DMA:        DS      256
```

==
SC 12 IPL
==

Function: Perform initial program load.

Input parameters:

>   Reg B – Physical drive number
>   Reg DE – Logical sector number of NUCLEUS

Output parameters: none

Description:

> This system call performs the transfer of control from the boot-loader process to the operating system. In effect, it loads all of the device drivers that are sysgened, initializes any interrupt structure, asks the operator for the date and time, performs system call 75 (NEWSYS), then sets a switch indicating that no account is logged onto and exits with system call 0 (QUIT)

> The contents of the registers A, H, and L are passed to the internal routine INITINT.

> After this system call has been used for the initial program load it

disables itself by inserting a branch to system call 0.

Other system calls used: 0 (QUIT), 1 (KEYIN), 2 (DISPLAY), 53 (TIMER), 61 (DEVINIT), 75 (NEWSYS)

Other register altered: all

===
SC 13 WRFDIR
===

Function: Write file directory entry.

Input parameters:

>   Reg B - Logical drive code (0 - 7) = (S - G)
>   Reg DE - Address of DEB

Output parameters:

>   Flag C - Set if error; reset otherwise
>   Flag Z - Reset if error; set otherwise

Description:

>   The directory entry addressed by the DE register pair is written to the
>   directory of the drive addressed by the B register. The directory entry
>   block (DEB) must be completely filled in (all 32 bytes). If the directory
>   is full or if the directory entry is a duplicate of an entry already on
>   file the carry flag is set and the zero flag is reset; otherwise the carry
>   flag is reset and the zero flag is set.
>
>   The user is advised to not use this system call to create directory
>   entries. When files are created using the other appropriate system calls
>   the directory entry is automatically created.

Other system calls used: WR (11), LOOKUP (20)

Other registers altered: A

===
SC 14 HEXI
===

Function: Convert hexadecimal number to 16 bit binary.

Input parameters:

>   Reg DE - Address of hex string

Output parameters:

>   Reg DE - Address of byte following string
>   Reg HL - Binary result
>   Flag C - Set if overflow; reset otherwise

Description:

    The string of characters addressed by the DE register pair is converted to a binary value, conversion stopping on the first non-hexadecimal digit. The resultant value is placed in the HL register pair, the DE register pair is adjusted to point to the character following the last hexadecimal digit or trailing 'H'. The system call is exited.

Other system calls used: none

Other registers altered: A

Example Calling Sequence:

```
        LD      DE,AREAH     ; Point ASCII hex
        SC      14           ; Convert to binary
        .
        .
        .
AREAH:  DC      'ABCDH'      ; Hex value
```

===
SC 15 DECI
===

Function: Convert decimal number to 16 bit binary.

Input parameters:

    Reg DE - Address of decimal string

Output parameters:

    Reg DE - Address of byte following
    Reg HL - Result
    Flag C - Set if overflow; reset otherwise

Description:

    The decimal string of characters addressed by the DE register pair is converted to an unsigned binary integer value and placed in the HL register pair. Conversion stops when a non-numeric character is encountered. The DE register pair is adjusted to point to the first character following the digits or trailing 'D' character. The system call is exited.

Other system calls used: none

Other registers altered: A

Example Calling Sequence:

```
            LD      DE,AREAD      ; Point ASCII Decimal
            SC      15            ; Convert to binary
            .
            .
            .
AREAD:      DC      '12345'       ; Decimal value
```

===
SC 16 HEXO
===

Function: Convert 8 bit value to hexadecimal characters.

Input parameters:

        Reg B - Byte to be converted
        Reg DE - Address of storage area

Output parameters:

        Reg DE - Address of next location following

Description:

    The 8 bit value in the B register is converted to the two hexadecimal
    character equivalent. These two characters are placed in the storage area
    addressed by the DE register pair. The DE register pair is adjusted to
    point to the location following the second character. The system call is
    exited.

Other system calls used: none

Other registers altered: A

Example Calling Sequence:

```
            LD      B,(HL)        ; Get byte to convert
            LD      DE,AREAH      ; Conversion area
            SC      16            ; Convert binary to hex
            .
            .
            .
AREAH:      DS      2             ; Conversion area
```

===
SC 17 DECO
===

Function: Convert 16 bit unsigned value to decimal string.

Input parameters:

        Reg DE - Address of storage area
        Reg HL - Value to be converted

Output parameters:

> Reg DE - Address of location following

Description:

> The 16 bit value in the HL register pair is converted to the ASCII character decimal equivalent (leading zeros are suppressed). The resultant string is placed in the storage area addressed by the register pair DE and the register pair DE is adjusted to point to the following location. The system call is exited.

Other system calls used: none

Other registers altered: A, H, L

Example Calling Sequence:

```
            LD      DE,AREA       ; Work area
            LD      HL,(NUMBER)   ; Get number
            SC      17            ; Convert to decimal
            LD      A,0DH         ; Get a CR
            LD      (DE),A        ; Mark end
            .
            .
            .
AREA:       DS      6
NUMBER:     DC      256
```

===
SC 18 CRLF
===

Function: Display carriage return, line feed on console.

Input parameters: none

Output parameters: none

Description:

> A carriage return and a line feed character are displayed on the console output device.

Other system calls used: CONOUT (5)

Other registers altered: A

Example Calling Sequence:

```
            SC      18            ; Display CR/LF
```

===
SC 19 MSEC
===

Function: Wait specified number of milliseconds

Input parameters:

        Reg A - Number of milliseconds

Output parameters: none

Description:

> The number of milliseconds indicated by the contents of the A register are
> "waited". An instruction sequence is performed that requires exactly one
> millisecond to execute. The content of the A register is then
> decremented. If the A register is not zero then the loop is executed
> again. If the A register is zero then control is returned to the
> instruction following the system call.

> Note: If the A register contains a zero upon entry then 256 msec will
> elapse before control is returned. Any interrupts that occur while this
> routine is executing will cause minor inaccuracies in the actual elapsed
> time.

Other system calls used: TIMER (53)

Other registers altered: A

Example Calling Sequence:

```
            LD        A,10        ; Get count
            SC        19          ; Wait for 10 msec
                                  ; Wait for 1 second
            LD        A,232       ; Initial value
            SC        19          ; Wait 232 msec
            SC        19          ; Wait 256 msec
            SC        19          ; Wait 256 msec
            SC        19          ; Wait 256 msec
```

===
SC 20 LOOKUP
===

Function: Locate directory entry of specified file.

Input parameters:

        Reg DE - Address of DCB
        Reg HL - Address of 256 byte work area

Output parameters:

```
If found-   Flag Z  - Set
            Reg B   - Logical drive number (0 - 7) = (S - G)
            Reg DE  - Sector address of directory block.
            Reg HL  - Address within work area of entry

If not found- Flag Z - Reset
              Flag C - Reset if directory not full
                       Set if directory full
```

Description:

The specified file description is searched for in the directory of the drive indicated. If the directory entry for the file is found then the relevant information is placed in the indicated registers and the system call is exited.

If the directory entry for the file is not found then the relevant information is placed in the indicated registers and the system call is exited. In this situation the calling program should create a directory entry for the file at the location within the work area and write the work area to disk at the indicated disk address.

This method of creating file entries is not intended to be used for general purpose file creation - the system utilities provide this ability with proven safety. Be very carefull if you do use this system call!

Other system calls used: RD1 (10), DIV (38), TSTDEV (58)

Other registers altered: C

Example Calling sequence:

```
            LD      DE,FNFTFD       ; Point DCB
            LD      HL,WORK         ; Point work area
            SC      20              ; Directory lookup
            JP      NZ,NOFND        ; Branch if not found
            .
            .
            .
WORK:       DS      256
FDFTFD:     DC      1,'TEST    ','FILE    ' ; TEST.FILE:A
```

===
SC 21 GETUCB
===

Function: Get address of UCB (Unit Control Block).

Input parameters:

Reg B - Logical device number

Output parameters:

                Reg HL — Address of UCB for physical device
                Reg C — Physical device number
                Flag C — set if no attachment

Description:

        The logical device indicated is tested for an attachment to a physical
        device. If no attachment then the carry flag is set and the system call
        is exited. If the device is attached then the address of the Unit Control
        Block is placed in the HL register and the physical device number that the
        logical device is attached to is placed in the C register.

Other system calls used: TSTDEV (58)

Other registers altered: A

Example Calling sequence:

```
            LD      B,9         ; Log device number
            SC      21          ; Point UCB of CONOUT
            LD      DE,10       ; Displacement to delay
            ADD     HL,DE       ; Point delay value
            LD      (HL),0      ; Reset to zero
```

==
SC 22 LOAD
==

Function: Load a program.

Input parameters:

                Reg HL — Load address
                Reg DE — Address of DCB

Output parameters: none

Description:

        The program specified by the directory control block pointed to by the DE
        register pair is loaded into memory at the load address referenced by the
        HL register pair.

Other system calls used: RD (50)

Other registers altered: A, B, C

Example Calling sequence:

```
        LD      DE,SUBRNAME   ; Point to name
        LD      HL,SUBR       ; Memory address
        SC      22            ; Load it
        CALL    SUBR          ; Execute the program
        .
        .
        .
SUBRNAME:DC     0,'USER     ','PROGRAM ' ; USER.PROGRAM:S
SUBR:   EQU     $             ; Load here
```

===
SC 23 PRINT
===

Function: Output a line to printer 1

Input parameters:

> Reg DE - Address of line to print

Output parameters: none

Description:

> The characters in the buffer addressed by the register pair DE are
> transmitted to printer 1 until a carriage return or null is encountered.
> Carriage returns and line feed characters are printed as a carriage
> return, line feed sequence. Other editing is performed according to the
> options associated with the attached printer.

Other system calls used: PRTOUT (8)

Other registers altered: A, C

Example Calling sequence:

```
        LD      DE,LINE       ; Point to message
        SC      23            ; Output to PRINTER1
        .
        .
        .
LINE:   DC      'Now is the time',10,'for all etc.',13
```

===
SC 24 ASSIGN
===

Function: Store ACB (Assign Control Block)

Input parameters:

> Reg B - ACB number (0 - 16)
> Reg DE - Address of formatted ACB

Output paramters:

                Reg A — Set to 255 if error
                Flag C — Set if error; reset otherwise

Description:

        The ACB number is verified to be in the range 0-16, if not the value 255
        is placed in the A register and the system call is exited. The formatted
        ACB referenced by the DE register pair is placed in the specified ACB.
        The A register is set to zero and the system call is exited.

Other system calls used: none

Other registers altered: C, H, L

Example Calling sequence:

            LD   B,6   ; Channel 6
            LD   DE,ACB   ; Point to my copy of ACB
            SC   24   ; Store assign control block
            .
            .
            .
    ACB: DC   1,'FILENAME','FILETYPE',1

===
SC 25 ADRV
===

Function: Convert logical drive code to logical drive number.

Input paramters:

        Reg B — Logical drive code (S - G, *)

Output parameters:

        Reg A — Logical drive number (0 - 7, 255)
        Flag C — Set if error.

Description:

        The drive code (alphabetic) is converted into a number in the range of 0
        thru 7. If the drive code is an asterisk (*) the number is 255.

Other system calls used: TSTDEV (58)

Other registers altered: none

Example Calling sequence:

            LD   B,'A'   ; Load drive code
            SC   25   ; Convert to number

==
SC 26 BDRV
==

Function: Convert logical drive number to logical drive code.

Input parameters:

        Reg B - Logical drive number (0 - 7, 255)

Output parameters:

        Reg A - Logical drive code (S - G, *)

Description:

        The logical drive number is converted to the external logical drive code
        (alphabetic).

Other system calls used: none

Other registers altered: none

Example Calling sequence:

```
        LD      A,(FD)      ; Get logical drive number
        LD      B,A         ; Move to B
        SC      26          ; Convert to drive code
        .
        .
        .
FD:     DC      1
```

==
SC 27 ALLOC
==

Function: Allocate disk space.

Input paramters:
        Reg B - Logical drive number (0 - 7) = (S - G)
        Reg DE - Number of 1K disk blocks to allocate

Output parameters:

        Reg HL - Sector address of first block.
        Flag Z - Status:
                set - okay
                reset - error

Description:

        The specified disk allocation map is searched for a contiguous block of
        unallocated disk space equal to the number of disk blocks desired. If
        insufficient space is available the Z flag is reset. If space is
        available the Z flag is set, the allocation map is updated and written to

the disk, and the first sector address of the allocated disk space is loaded into the HL register pair.

Caution: Use of this system call is not advised.

Other system calls used: RD1 (10), WR1 (11)

Other registers altered: none

Example Calling sequence:

```
            LD      B,0         ; Drive S
            LD      DE,1        ; One block
            SC      27          ; Allocate
            JP      NZ,FULL     ; Branch if full
            LD      (SECT),HL   ; Else save sector address
```

===
SC 28 DEALL
===

Function: Deallocate disk space

Input parameters:

```
            Reg B  - Logical disk drive number (0 - 7) = (S - G)
            Reg DE - Number of 1K blocks to deallocate
            Reg HL - Starting sector number
```

Output parameters:

```
            Flag Z - Status:
                     set - okay
                     reset - error
```

Description:

The specified disk allocation map is searched for the indicated allocated space. If the indicated space is not already allocated the Z flag is reset and the system call is exited. Otherwise the allocation map is updated and written to disk; the Z flag is set and the system call is exited.

Caution: Use of this system call is not advised.

Other system calls used: RD1 (10), WR1 (11)

Other registers altered: H, L

Example Calling sequence:

```
            LD      B,0         ; Drive S
            LD      DE,1        ; 1K bytes
            LD      HL,(SECT)   ; Sector address
            SC      28          ; Return to avail status
```

===
SC 29 ERASE
===

Function: Erase logical file from a disk.

Input parameters:

       Reg DE - Address of DCB

Output parameters:

       Reg A - Return Code:
             00 Successful
             FF File protected
       Flag Z - Status:
             set - okay
             reset - error

Description:

       The directory for the specified disk drive is searched for a match with
       the file description. When a match is found the file disk space is
       deallocated, the directory entry is placed in a delete status and the
       directory block is updated on disk.

Other system calls used: RD1 (10), WR1 (11), LOOKUP (20), DEALL (28)

Other registers altered: none

Example Calling sequence:

```
          LD        DE,FN        ; Point to DCB
          SC.       29           ; Erase file if it exists
          .
          .
          .
FN:       DC        1,'TEST    ','FILE      '
```

===
SC 30 FETCH
===

Function: Load program into memory, execute and return to CSI.

Input parameters:

       Reg B - Logical drive code
       Reg DE - Directory entry pointer

Output parameters: none

Description:

       The eventual return address is replaced with the address of the boot
       loader; system call 22 is executed with control returned to the boot

loader upon completion of the program execution.

Other system calls used: LOAD (22)

Other registers altered: B, C

Example Calling Sequence:

```
            LD      DE,DCB          ; Point to DCB
            LD      HL,WORK         ; Point work space
            SC      20              ; Get directory entry
            EX      DE,HL           ; DE points directory
            LD      A,(DCB)         ; Point to drive
            LD      B,A             ;
            SC      30              ; Load & execute
            .
            .
DCB:        DC      1,'MYPROG  ','COMMAND '
WORK:       DS      256
```

==
SC 31 RENAME
==

Function: Rename a logical disk file.

Input parameters:

```
            Reg DE — Address of DCB
            Reg HL — Address of new DCB
```

Output parameters:

```
            Reg A   — Variable:
                      00 if okay
                      04 if old not found
                      08 if new exists
                      0A Protected file
```

Description:

The new drive code is set equal to the old drive code. The directory for the specified disk is searched for the old file description. If the directory entry cannot be found then the A register is set to 04 and the system call is exited. If the file is found then the directory is searched for the new file description. If the directory entry is found then the A register is set to 08 and the system call is exited.

If the old file description does exist and the new file description doesn't exist then the old file entry is placed in delete status, the directory block is updated, the new file entry is created (duplicating the attributes of the old file), and the directory block is updated. The system call is exited.

Other system calls used: WR1 (11), LOOKUP (20)

Other registers altered: H, L

Example Calling Sequence:

```
                LD      DE,OLD      ; Point to old name
                LD      HL,NEW      ; Point to new name
                SC      31          ; Rename it
                JR      Z,OKAY      ; Error?
     ERROR:     .
                .
                .
     OLD:       DC      1,'OLD      ','FILE
     NEW:       DC      1,'NEW      ','DESCRIPT'
```

===
SC 32 OPEN
===

Function: Open a logical file.

Input parameters:

>   Reg DE - Address of FCB

Output parameters:

>   Reg A   - Return code:
>   00 Successful
>   01 Already open
>   04 Invalid file definition
>   08 Invalid file number
>   0A File protected
>   10 Disk full
>   20 Directory full
>   40 File not found
>   Reg B   - Device assigned to file
>   Flag Z - Status:
>   set - okay (Reg A = 0)
>   reset - error (Reg A <> 0)

Description:

>   The file specified by the FCB is opened in the mode indicated with the
>   appropriate return code set if the open is unable to be accomplished.
>   Register B is set to the logical drive code that a new sequential file was
>   opened to if the drive was not specified explicitly in the ACB.
>
>   This system call checks the File Lock Table (FLT) and waits if the file is
>   locked by another partition. When the file is not locked by another
>   partition or is released by that partition this system call will lock the
>   file if specified by the FCB.

Other system calls used: RD1 (10), WR1 (11), WRFDIR (13), LOOKUP (20), ALLOC (27),
DEALL (28), ERASE (29), DPACK (46), TSTDEV (58)

Other registers altered: C, H, L

Example Calling Sequence:

```
            LD      B,16            ; Assign I/O ch 16
            LD      DE,ACB16
            SC      24
            LD      DE,FCB1         ; Point to FCB
            SC      32              ; Open the file
            JR      NZ,ERROR        ; BRIF error
            .
            .
            .
FCB1:       DC      16              ; ACB = 16
            DC      10001100B       ; Seq, append
            DC      (BUFF1)         ; I/O buffer
BUFF1:      DS      256
ACB16:      DC      1,'REPORT  ','LISTING '
            DC      1
```

===
SC 33 CLOSE
===

Function: Close a logical file.

Input parameters:

        Reg DE – Address of FCB

Output parameters:

        Reg A – Return code
                 00 Successful
                 08 Invalid file number
                 10 Disk full
        Flag Z – Status:
                 set – okay (Reg A = 0)
                 reset – error (Reg A <> 0)

Description:

        The specified file is logically and physically closed with the appropriate
        return code set. Closing a file involves the updating of the disk file
        with the data in the I/O buffer; updating the directory entry for the
        file; flagging the ACB as closed.

        This system call unlocks the file and all related sectors from the FLT and
        SLT.

Other system calls used: WR1 (11), LOOKUP (20), DPACK (46)

Other registers altered: B, C, H, L

Example Calling Sequence:

```
                              ; Using current assign
              LD    DE,FCB1   ; Open FCB1
              SC    33
              JR    NZ,ERROR  ; BRIF error
              .
              .
              .
FCB1:         DC    16
              DC    10001000B ; Seq, output
              DC    (BUFF1)   ; I/O buffer
BUFF1:        DS    256
```

===
SC 34 RDSEQ
===

Function: Get a logical record from a sequential file.

Input parameters:

                Reg DE - Address of FCB
                Reg HL - Address of record area

Output parameters:

                Reg AF - Return code
                        00 Successful
                        01 End of File
                        08 Invalid file number
                        FF File not open
                Flag Z - Status:
                        set - okay (Reg A = 0)
                        reset - error (Reg A <> 0)

Description:

        The ACB is validated for: open, sequential, and input. The A register is
        set to 255 if ACB invalid. The ACB is tested for an EOF condition and the
        appropriate return code is set if true and the system call is exited. If
        everything is okay the next record is passed to the record buffer
        addressed by the HL register pair with file input performed as required.
        ASCII sequential file records are always terminated with a carriage return
        character (0DH).

        This system call, like all logical record input/output system calls,
        maintains the Sector Lock Table (SLT) according to the FCB.

Other system calls used: INPUT (1), RD1 (10), DEVIN (63)

Other registers altered: B, C

Example Calling Sequence:

```
                LD      DE,FCB1      ; Get record from file
                LD      HL,BUFF      ; Put in BUFF buffer
                SC      34           ; Do it
                JR      NZ,CHKERR    ; Analyze error routine
                .
                .
                .
    FCB1:       DC      10           ; I/O ch 10
                DC      10010000B    ; Seq input
                DC      (BUFF1)      ; I/O buffer
    BUFF1:      DS      256
    BUFF:       DS      128          ; Max rec length = 128
```

==
SC 35 WRSEQ
==

Function: Write a logical record to a sequential file.

Input parameters:

                Reg DE — Address of FCB
                Reg HL — Address of record

Output parameters:

                Reg AF — Return code
                        00 Successful
                        08 Invalid file number
                        10 Disk full
                        FF File not open, etc.
                Flag Z — Status:
                        set — okay (Reg A = 0)
                        reset — error (Reg A <> 0)

Description:

        The ACB is validated: open, sequential, and output or append. The
        appropriate return code is set when invalid and the sysem call is exited.
        The record is transferred to the file buffer and physical output is
        performed as required. When the file is a disk file and the file requires
        more allocation to perform the physical output then the file is expanded.

        This system call, like all logical record input/output system calls,
        maintains the Sector Lock Table (SLT) according to the FCB.

        Note: Be sure that the record addressed by the HL register pair contains a
        carriage return character (0DH) as the terminating character.

Other system calls used: DISPLAY (2), WR1 (11), ALLOC (27), DEVOUT (64)

Other registers altered: B, C

Example Calling Sequence:

```
            LD      DE,FCB2      ; Write seq record
            LD      HL,BUFF      ; From BUFF buffer
            SC      35
            JR      Z,OKAY       ; Skip if okay
            CP      10H          ; Check for disk full
            JR      Z,DFULL      ; BRIF full
                                 ;  else ignore error
OKAY:       .
            .
            .
FCB2:       DC      2            ; I/O channel 2
            DC      10001000B    ; Seq output
            DC      (BUFF2)      ; I/O buffer
BUFF2:      DS      256
BUFF:       DS      128
```

==
SC 36 GETDATE
==

Function: Get formatted date.

Input parameters:

       Reg DE - Address for storage

Output parameters: none

Description:

    The packed system date is unpacked and placed in the storage location addressed by the DE register pair. The format of the resulting date string is determined by the currently set date format (see the "SET COMMAND", DATEFORM option in the OASIS System Reference Manual). The DE register pair is adjusted to point to the byte following the last character of the date string.

Other system calls used: HEXO (16), DATEOUT (106)

Other registers altered: A

Example Calling Sequence:

```
            LD      DE,WORK      ; Point to work area
            SC      36           ; Get system date
            LD      A,13         ; Get CR
            LD      (DE),A       ; Mark end
            LD      DE,MSG       ; Point to beginning of message
            SC      2            ; Display on console
            .
            .
            .
MSG:        DC      'The current date is '
WORK:       DS      9
```

==
SC 37 GETTIME
==

Function: Get formatted time.

Input parameters:

        Reg DE - Address of storage

Output parameters: none

Description:

        The current packed system time is unpacked and placed in the storage
        location addressed by the DE register pair. The colon character is used
        to separate the hours, minutes, and seconds. The DE register pair is
        adjusted and the system call is exited.

Other system calls used: HEXO (16)

Other registers altered: A

Example Calling Sequence:

```
                LD      DE,WORK      ; Point to work area
                SC      37           ; Get system time
                LD      A,13         ; Get CR
                LD      (DE),A       ; Mark end
                LD      DE,MSG       ; Point to message
                SC      2            ; Display on console
                .
                .
                .
     MSG:       DC      'The current time is '
     WORK:      DS      9
```

==
SC 38 DIV
==

Function: 16 Bit, binary, unsigned divide.

Input parameters:

        Reg DE - Divisor
        Reg HL - Dividend

Output parameters:

        Reg DE - Remainder
        Reg HL - Quotient
        Flag C - Set if divide by zero; reset otherwise

Description:

> The divisor is tested. If zero the HL register pair is set to zero, the
> carry flag is set and the system call is exited. The value in the HL
> register pair is divided by the value in the DE register pair. The result
> is placed in the HL register pair and any remainder is placed in the DE
> register pair.

Other system calls used: none

Other registers altered: A

Example Calling Sequence:

```
            LD      DE,(VALUE1)     ; Divide value1
            LD      HL,(VALUE2)     ; into value2
            SC      38
            JR      C,DIVZERO       ; Divide by zero err?
            .
            .
VALUE1:     DS      2
VALUE2:     DS      2
```

===
SC 39 MUL
===

Function: 16 bit, unsigned, integer multiply.

Input parameters:

> Reg DE — Multiplier
> Reg HL — Multiplicand

Output parameters:

> Reg HL — Product
> Flag C — Set if overflow; reset otherwise

Description:

> The value in the HL register pair is multiplied by the value in the DE
> register pair. The result is placed in the HL register pair. If overflow
> occurs (more than 16 bits of product) the carry flag is set. The sytem
> call is exited.

Other system calls used: none

Other registers altered: A

Example Calling Sequence:

```
            LD      DE,(VALUE1)    ; Multiply value1
            LD      HL,(VALUE2)    ; by value2
            SC      39
            JR      C,OVERFLO      ; BRIF error
            .
            .
            .
VALUE1:     DC      (3)
VALUE2:     DC      (12345)
```

==
SC 40 RDDIR
==

Function: Read logical record from a direct disk file.

Input parameters:

        Reg BC - Record number
        Reg DE - Address of FCB
        Reg HL - Address of record storage area

Output parameters:

        Reg A   - Return code
                   00 Successful
                   08 Invalid ACB number
                   80 Invalid record number
                   FF File not open, etc.
        Flag Z - Status:
                   set - okay (Reg A = 0)
                   reset - error (Reg A <> 0)

Description:

   The required I/O overlay is loaded, if necessary. The ACB is tested for
   an open, direct file and the appropriate return code is set if invalid.
   The record number and the file's filesize are compared. If the record is
   outside of the filesize the appropriate return code is set. The record is
   transferred from the file buffer with physical input performed as
   required.

   This system call, like all logical record input/output system calls,
   maintains the Sector Lock Table (SLT) according to the FCB.

Other system calls used: RD1 (10), DIV (38), MUL (39)

Other registers altered: none

Example Calling Sequence:

```
                LD      HL,(RECNUM)     ; Get record number
                LD      B,H             ; Copy to BC reg
                LD      C,L             ;
                LD      HL,BUFF         ; Point to record buffer
                LD      DE,FCB1         ; Point to FCB, ch 1
                SC      40              ; Get the record
                JR      NZ,RDERR        ; Jump on error
                .
                .
RECNUM:         DS      2               ; Current record number
FCB1:           DC      1,01011000B     ; Direct I/O with record lock
                DW      IOBUFF1         ; I/O buffer addr
BUFF:           DS      32              ; Record buffer
BUFF1:          DS      256             ; I/O Buffer
```

===
SC 41 WRDIR
===

Function: Write a logical record to a direct disk file.

Input parameters:

                Reg BC - Record number
                Reg DE - Address of FCB
                Reg HL - Address of record to be written

Output parameters:

                Reg A   - Return code
                          00 Successful
                          08 Invalid ACB number
                          0A Protected file
                          80 Invalid record number
                          FF File not open
                Flag Z - Status:
                          set - okay (Reg A = 0)
                          reset - error (Reg A <> 0)

Description:

        The required I/O overlay is loaded, if necessary. The ACB is tested for
        an open, direct file and the appropriate return code is set if invalid.
        The file's filesize is compared to the record number specified and the
        appropriate return code is set if the record number is invalid. The
        record is transferred to the file buffer with physical output performed as
        required.

        This system call, like all logical record input/output system calls,
        maintains the Sector Lock Table (SLT) according to the FCB.

Other system calls used: RD1 (10), WR1 (11), MUL (39), DIV (38)

Other registers altered: none

Example Calling Sequence:

```
            LD      HL,(RECNUM)    ; Get record number
            LD      B,H            ; Copy to BC reg
            LD      C,L            ;
            LD      HL,BUFF        ; Point to record storage
            LD      DE,FCB1        ; Point to FCB, ch 1
            SC      41             ; Write it
            JR      NZ,WRERR       ; Jump on error
            •
            •
RECNUM:     DS      2              ; Record to be accessed
FCB1:       DC      1,01011000B    ; Ch 1, direct I/O with record lock
            DW      BUFF1          ; I/O buffer address
BUFF1:      DS      256            ; I/O buffer
BUFF:       DS      32             ; Record buffer
```

==
SC 42 NUMBER
==

Function: Convert numeric string (hex or dec) to 16 bit value.

Input parameters:

        Reg DE — Address of character string

Output parameters:

        Reg DE — Address of character following
        Reg HL — Result
        Flag C — Set if overflow; reset otherwise

Description:

        The string of characters is examined and the number base is determined.
        The appropriate conversion routine is used to produce the equivalent 16
        bit value in the HL register pair.

Other system calls used: DECI (15), HEXI (14)

Other registers altered: A

Example Calling Sequence:

```
            LD      DE,INPUT       ; Point to number string
            SC      42             ; Convert it
            JR      C,CONERR       ; Jump on overflow
            LD      (NUMB),HL      ; Save value
            •
            •
INPUT:      DC      '12345D'       ; Number to convert
NUMB:       DS      2              ; Value
```

==
SC 43 RDIX
==

Function: Read a logical record from an indexed disk file.

Input parameters:

                    Reg BC – Address of key
                    Reg DE – Address of FCB
                    Reg HL – Address of record storage area

Output parameters:

                    Reg A – Return code
                             00 Successful
                             01 Record not found
                             08 Invalid ACB number
                             FF File not open
                    Flag Z – Status:
                             set – okay (Reg A = 0)
                             reset – error (Reg A <> 0)

Description:

        The required I/O overlay is loaded, if necessary. The ACB is tested for
        an open, indexed file and the appropriate return code is set. The record
        key is searched for in the file. If the record key is found the record is
        transferred to the record address specified in the HL register pair and
        the return code is set. If the record key is not found the return code is
        set and the relative record number of the next record that would logically
        collate after the specified key is saved in the ACB.

        This system call, like all logical record input/output system calls,
        maintains the Sector Lock Table (SLT) according to the FCB.

Other system calls used: RD1 (10), DIV (38), MUL (39), (system subroutines)

Other registers altered: none

Example Calling Sequence:

```
                LD      HL,KEY          ; Point to key string
                LD      B,H             ; Copy to BC reg
                LD      C,L             ;
                LD      HL,BUFF         ; Point to input buffer
                LD      DE,FCB1         ; FCB for ch 1
                SC      43              ; Read the record
                JR      NZ,NOFIND       ; Jump if record not found
                .
                .
FCB1:           DC      1,00111000B     ; Ch 1, Indexed I/O with record lock
                DW      BUFF1           ; I/O buffer address
BUFF1:          DS      256             ; I/O buffer
KEY:            DS      10              ; Key of 10 characters
BUFF:           DS      122             ; Rec of 122 characters
```

===
SC 44 RDNIX
===

Function: Read the next logically sequential record of indexed file.

Input parameters:

>   Reg BC — Address of key storage area
>   Reg DE — Address of FCB
>   Reg HL — Address of record storage area

Output parameters:

>   Reg A   — Return code
>   00 Successful
>   01 End of file
>   08 Invalid ACB number
>   FF File not open
>   Flag Z — Status:
>   set — okay (Reg A = 0)
>   reset — error (Reg A <> 0)

Description:

> The required I/O overlay is loaded, if necessary. The ACB is tested for
> an open, indexed file and the appropriate return code is set. Using the
> relative record number in the ACB indicating the disk address of the next
> logically sequential record in the file, the record and key are read into
> the file buffer and transferred to the key and record storage areas
> specified by the BC and HL register pairs. The following logically
> sequential record is located and the relative record number is saved in
> the ACB. The return code is cleared and the system call is exited.
>
> This system call, like all logical record input/output system calls,
> maintains the Sector Lock Table (SLT) according to the FCB.

Other system calls used: RD1 (10), RDIX (43)

Other registers altered: none

Example Calling Sequence:

```
        LD      HL,KEY      ; Point to key string
        LD      B,H         ; Copy to BC reg
        LD      C,L         ;
        LD      HL,BUFF     ; Point to input buffer
        LD      DE,FCB1     ; FCB for ch 1
        SC      44          ; Read the next record
        JR      NZ,NOFIND   ; Jump if record not found
        •
        •
FCB1:   DC      1,00110000B ; Ch 1, Indexed Input
        DW      BUFF1       ; I/O buffer address
BUFF1:  DS      256         ; I/O buffer
KEY:    DS      10          ; Key of 10 characters
BUFF:   DS      122         ; Rec of 122 characters
```

===
SC 45 WRIX
===

Function: Write a logical record to an indexed disk file.

Input parameters:

          Reg BC - Address of key
          Reg DE - Address of FCB
          Reg HL - Address of record

Output parameters:

          Reg A - Return code
                   00 Successful
                   0A Protected file
                   10 File full - record not written
                   FF File not open
          Flag Z - Status:
                   set - okay (Reg A = 0)
                   reset - error (Reg A <> 0)

Description:

          The required I/O overlay is loaded, if necessary. The ACB is tested for
          an open, indexed file and the appropriate return code is set. The file is
          searched for a current record with the same key. If a record does exist
          the record is overwritten with the new record the return code is cleared
          and the system call is exited. If a record does not exist a location for
          the new record is found and the record is written to the file. The return
          code is cleared and the system call is exited. If no space is available
          for the new record the return code is set to 10H and the system call is
          exited. No attempt is made to write the record to the file in this
          situation.

          This system call, like all logical record input/output system calls,
          maintains the Sector Lock Table (SLT) according to the FCB.

Other system calls used: RD1 (10), WR1 (11)

Other registers altered: none

Example Calling Sequence:

```
                LD      HL,KEY        ; Point to key string
                LD      B,H           ; Copy to BC reg
                LD      C,L           ;
                LD      HL,BUFF       ; Point to input buffer
                LD      DE,FCB1       ; FCB for ch 1
                SC      45            ; Write the record
                JR      NZ,ERR        ; Jump if error
                .
                .
FCB1:           DC      1,00101000B   ; Ch 1, Indexed output
                DW      BUFF1         ; I/O buffer address
BUFF1:          DS      256           ; I/O buffer
KEY:            DS      10            ; Key of 10 characters
BUFF:           DS      122           ; Rec of 122 characters
```

===
SC 46 DATEPACK
===

Function: Pack system date and time into 24 bit value.

Input parameters:

        Reg DE - Address of storage area

Output parameters:

        Reg DE - Address of location following

Description:

        The system date and system time are converted, formatted, and packed into
        a 24 bit (3 byte) format. The result is placed in the location addressed
        by the DE register pair and the DE register pair is adjusted. The system
        call is exited.

        This system call is normally only used for converting the date and time
        for use in a file's directory entry, although it can be used for other
        purposes. There is no corresponding unpack system call.

Other system calls used: none

Other registers altered: A, B, C

Example Calling Sequence:

```
            LD      DE,DIR+25    ; Point to storage
            SC      46           ; Get date and time
            .
            .
DIR:        DS      32           ; Directory entry buffer
```

===
SC 47 LABEL
===

Function: Find disk with specific label.

Input parameters:

        Reg DE - Address of 8 character label

Output parameters:

        Reg A - Logical drive number (0 - 7) = (S - G)
        Flag C - Set if not mounted; reset otherwise

Description:

        The disks mounted in the attached disk drives are interrogated for a match
        with the specified disk label. The drive code of the first match found is
        placed in the A register. If no match is found the carry flag is set.

Other system calls used: RD1 (10), ADRV (25), TSTDEV (58)

Other registers altered: B, C

Example Calling Sequence:

```
            LD      DE,LABEL     ; Point to desired label
            SC      47           ; Find disk with label
            JR      C,ERR        ; Check if found
            LD      (DRIVE),A    ; Save drive number
            .
            .
            .
LABEL:      DC      'WORK
DRIVE:      DS      1
```

===
SC 48 GETSCR
===

Function: Get base address of your System Communication Region.

Input parameters: none

Output parameters:

        Reg IY - SCR address

Description:

The first address of your SCR is placed in the IY index register and the system call is exited.

Other system calls used: none

Other registers altered: none

Example Calling Sequence:

```
SC      48          ; Get SCR base
LD      (BASE),IY   ; Save base address
```

==
SC 49 WAIT
==

Function: Wait for operator to release current console page.

Input parameters: none

Output parameters: none

Description:

The Console Screen Wait-key status is tested and, if disabled, the system call is exited. When the Console Screen Wait-key is enabled the page pause prompt character (^) is displayed at the lower left hand corner of the console output device (CONOUT) (unless console terminal class is 0) and processing is suspended until the operator types a key to indicate that the page may be released. At this time a CR is displayed on the console and control is returned to the calling program.

Other system calls used: none

Other registers altered: A

Example Calling Sequence:

```
.                       ; Code to output 'page'
.                       ; of information
.
SC      49              ; Wait at bottom if
                        ; enabled
```

==
SC 50 RD
==

Function: Read multiple sectors of a disk.

Input parameters:

```
Reg B  - Logical drive number (0 - 7) = (S - G)
Reg C  - Number of sectors to read
Reg DE - First sector address
Reg HL - Storage address
```

Output parameters: none

Description:

The specified drive is selected, if legal, and the sector specified by the contents of the DE register pair are read into the location indicated by the HL register pair. The sector count is decremented, the DE register pair is incremented, the HL register pair is adjusted, and, if the count is greater than zero the next sector is read.

This system call, when used in a multi-user environment, checks the Sector Lock Table (SLT) and waits if the requested sector is locked by another partition.

Caution: Use of this system call is not advised.

Other system calls used: SYSIN (6), SYSOUT (7), DECO (17), ADRV (25), DIV (38), SYSDISP (52)

Other registers altered: A, C

Example Calling Sequence:

```
          LD      B,0           ; Drive S
          LD      DE,256        ; Starting at sector 256
          LD      C,16          ; For 16 sectors
          LD      HL,BUFFER     ; Read into buffer
          SC      50            ; Read the sectors
          .
          .
BUFFER:
          REPT    16            ; Buffer for 16 sectors
          DS      256           ;
```

==
SC 51 WR
==

Function: Write multiple sectors to disk.

Input parameters:

```
Reg B  - Logical drive number (0 - 7) = (S - G)
Reg C  - Sector count
Reg DE - First sector address
Reg HL - Address of data to be written
```

Output parameters: none

Description:

> The specified drive is selected, if legal. The data stored at the
> location referenced by the HL register pair is written to the sector
> specified by the DE register pair. The DE register pair is incremented,
> the HL register pair is adjusted, and the sector count in decremented. If
> the sector count is not zero then the next sector is written.

> This system call, when used in a multi-user environment, checks the Sector
> Lock Table (SLT) and waits if the requested sector is locked by another
> partition.

> Caution: Use of this system call is not advised.

Other system calls used: SYSIN (6), SYSOUT (7), DECO (17), ADRV (25), DIV (38),
SYSDISP (52)

Other registers altered: A, C

Example Calling Sequence:

```
              LD       B,0          ; Drive S
              LD       DE,256       ; Starting at sector 256
              LD       C,16         ; For 16 sectors
              LD       HL,BUFFER    ; Write from buffer
              SC       51           ; Write the sectors
              .
              .
   BUFFER:
              REPT     16           ; Buffer for 16 sectors
              DS       256          ;
```

===
SC 52 SYSDISP
===

Function: Display character on console output device.

Input parameters:

> Reg DE — Address of first character to output

Output parameters:

> Reg DE — Address of last character output plus one

Description:

> Characters from the buffer addressed by register pair DE are displayed on
> the console output device. A null character (00) terminates output to the
> console and returns from the system call.

> A carriage return will be displayed as a carriage return, line feed and
> the system call will be exited. A line feed will be displayed as a
> carriage return, line feed, output continues. An HT character (09H) will
> be displayed as the proper number of spaces according to the Tab Set Block

(TSB).

This system call, unlike SC 2 (DISPLAY) will always display the characters on the console and never echo them to the printer (the status of Console Echo-key and Printer Echo-key is ignored).

Other system calls used: SYSOUT (7)

Other registers altered: A

Example Calling Sequence:

```
            LD        DE,MSG        ; Point to message string
            SC        52            ; Display on console
            .
            .
            .
MSG:        DC        'This is a message',13
```

===
SC 53 TIMER
===

Function: Set up for a clocked interrupt (event)

Input parameters:

```
            Reg DE - Number of "ticks"
            Reg HL - Address of TEB
```

Output parameters: none

Description:

This system call initiates a Timer Event. The contents of the DE register pair are stored in the TEB (Timer Event Block) specified by the contents of the HL register pair. The required links are made to other TEBs and control is returned to the instruction following the system call.

When the number of "ticks" specified by the DE register pair have elapsed the interrupt service routine is executed. The service routine must physically follow and be contiguous to the TEB. Upon entry interrupts are enabled.

It is the responsibility of the interrupt service routine to save any and all registers used and to execute a RET when service is complete (not a RETI).

The TEB should in no way be modified by the user until the interrupt service routine has been entered. Any changes to this TEB or any other TEB still in process will cause the operating system to act erratically, at best.

The length of time for a "tick" is dependent upon the system. Refer to the Supplemental documentation supplied with the OASIS System Reference Manual for the specific length of time for a "tick" on your machine.

Other system calls used: none

Other registers altered: none

Example Calling Sequence:

```
                LD      DE,60           ;Set up for timed interrupt
                LD      HL,LABEL1
                SC      53              ;Start the clock
                .
                .
                .
    LABEL1:     DS      6               ; TEB for above-must be 6 bytes
                .                       ;Code for interrupt service
                .                       ;must follow the TEB
                .
                RET                     ;Resume normal processing
```

===
SC 54 EXCMD
===

Function: Execute a command.

Input parameters:

>   Reg DE - Address of CSI command text

Output parameters: none

Description:

>   The Command String Interpreter is loaded and the command, with options, specified by the DE register is executed. The command is translated to upper case before interpretation. Upon completion of the command the system call is exited back to the CSI level.

>   When the first character of the string of characters addressed by the DE register pair is a '>' the string will be displayed on the user's console before it is executed.

Other system calls used: ?????

Other registers altered: all (No Return)

Example Calling Sequence:

```
                LD      DE,COMMAND      ; Point to command string
                SC      54              ; Transfer control
    COMMAND:    DC      'ERASE *.BACKUP:A (NOQUERY NOTYPE)',13
                END
```

===
SC 55 GETMEM
===

Function: Get stored memory size.

Input parameters: none

Output parameters:

        Reg HL — Address of 'end of memory'

Description:

        The currently stored value of the address of the end of memory is placed
        in the HL register pair and the system call is exited. This value may not
        be the actual address of the physical end of memory determined when the
        system was first IPL'd. The value is the currently saved address. This
        address can be changed by system call 56.

Other system calls used: none

Other registers altered: none

Example Calling Sequence:

```
        SC      55          ; Get current EOM
        LD      (EOM),HL    ; Save current EOM
        LD      DE,-1000
        ADD     HL,DE       ; Compute new EOM
        SC      56          ; Protect it
```

===
SC 56 PUTMEM
===

Function: Store memory size.

Input parameters:

        Reg HL — Address of end of memory

Output parameters: none

Description:

        The value in the HL register pair updates the currently stored value of
        the address of the end of memory. This system call is the logical inverse
        of system call 55.

Other system calls used: none

Other registers altered: none

Example Calling Sequence: see SC 55 (GETMEM)

===
SC 57 PUTQET
===

Function: Set quit error trap (System Cancel-key).

Input parameters:

> Reg HL - Address of break routine

Output parameters: none

Description:

> The value in the HL register is loaded into the quit error trap vector and the system call is exited. This routine addressed by HL will be given control whenever the System Cancel-key is typed. An address of zero (0000) in the HL register pair indicates that the user QET is to be disabled.

> An example of the use of this system call is the BASIC interpreter. The BASIC interpreter sets the quit error trap to execute a routine that closes all open files before exiting.

Other system calls used: none

Other registers altered: none

Example Calling Sequence:

```
          LD        HL,QETSERVC    ; Point to service routine
          SC        57             ; Set trap
          .
          .
QETSERVC:                          ; Routine to handle
                                   ; System Cancel-key entry
```

===
SC 58 TSTDEV
===

Function: Test device attachment.

Input parameters:

> Reg B - Logical device number

Output parameters:

> Reg A - Physical device number
> Flag Z - Set if not attached; reset otherwise
> Flag C - Set if not attached; reset otherwise

Description:

>The specified device is tested for attachment. If the device is attached the physical device number that it is attached to is placed in the A register, the Z flag is reset and the system call is exited. If the device is not attached to anything then the A register is set to FF, the Z flag is set and the system call is exited.

Other system calls used: none

Other registers altered: none

Example Calling Sequence:

```
        LD      B,16            ; Point to COMM1 device
        SC      58              ; Test if attached
        JR      Z,NOCOMM        ; Jump if not
        .
        .
```

===
SC 59 GETPL
===

Function: Get console/printer page and line parameters.

Input parameters:

>Reg B - Logical device number

Output parameters:

>Reg B - Line length
>Reg C - Page length
>Reg A - Class code

Description:

>The device number specified in the B register is validated to determine if it is the console or one of the printer devices. If the device number is invalid the system call is exited. If the device number is valid then the ATTACHed line and page size parameters are loaded into the B and C register, respectively and the class code is loaded into the A register. If the specified device is not attached then zero values are returned in the registers.

Other system calls used: GETUCB (21), TSTDEV (58)

Other registers altered: none

Example Calling Sequence:

```
        LD      B,9             ; Point to CONOUT device
        SC      59              ; Get parameters
        LD      (CLASS),A       ; Save class code
        LD      A,B             ;
        LD      (LINE),A        ; Save line length
        LD      A,C             ;
        LD      (PAGE),A        ; Save page length
        .
        .
```

==
SC 60 DELIX
==

Function: Delete record from indexed file.

Input parameters:

```
        Reg BC — Address of key
        Reg DE — Address of FCB
        Reg HL — Address of record storage area
```

Output parameters:

```
        Reg A   — Return code
                  00 Successful
                  08 Invalid ACB number
                  FF File not open
        Flag Z  — Status:
                  set — okay (Reg A = 0)
                  reset — error (Reg A <> 0)
```

Description:

The required I/O overlay is loaded, if necessary. The ACB is tested for an open, indexed file and the appropriate return code is set. The record key is searched for in the file. If the record key is found the record is transferred to the record address specified in the HL register pair, the record key buffer is modified to indicate that the record is deleted (first character changed to OFFH value) and the record is written back. The record linkages are updated to reflect the deleted record.

If the record key is not found the relative record number of the next record that would logically collate after the specified key is saved in the ACB.

This system call, like all logical record input/output system calls, maintains the Sector Lock Table (SLT) according to the FCB. If any of the sectors needed for the search and deletion of the record are locked by another partition this system call will wait for the sector to be released.

Other system calls used: RD1 (10), WR1 (11), RDIX (43)

Other registers altered: none

Example Calling Sequence:

```
                LD      HL,KEY          ; Point to record key
                LD      B,H             ; Copy to BC reg
                LD      C,L             ;
                LD      HL,REC          ; Point to record buffer
                LD      DE,FCB1         ; Point to channel 1 file
                SC      60              ; Delete the record
                JR      NZ,DELERR       ; Jump on error
                .
                .
FCB1:           DC      1,00111000B     ; Indexed, ch 1 with record lock
                DC      BUFFER          ; I/O buffer address
BUFFER:         DS      256             ;
KEY:            DS      32              ; Key is 32 character long
REC:            DS      32              ; Record is 32 character long
```

==
SC 61 DEVINIT
==

Function: Initialize a device driver.

Input parameters:

        Reg B — Logical device number

Output parameters: none

Description:

        The physical device driver attached to the logical device specified in the
        B register is entered at its initialization entry point. The actual
        process of initialization is device dependent.

        The address of the UCB associated with this device is loaded into the IY
        register and passed to the device driver along with the B register.

        Note: This system call is used by the ATTACH command when a device is
        first attached and should not be used by user programs.

Other system calls used: GETUCB (21), LOAD (22), TSTDEV (58)

Other registers altered: all

Example Calling Sequence:

```
                LD      B,17            ; Point to COMM2 device
                SC      61              ; Init driver
                .
                .
```

===
SC 62 DEVST
===

Function: Get status of device driver.

Input parameters:

        Reg B - Logical device number

Output parameters:

        Flag Z - Set if input character not ready; reset otherwise
        Flag C - Set if ready for output; reset otherwise

Description:

        The attachment of the specified device is tested. If the device is not
        attached the system call is exited. If the device is attached the status
        of the physical device attached to the logical device specified in the B
        register is returned in the Z flag.

        The address of the UCB associated with this device is loaded into the IY
        register and passed to the device driver along with the B register.

        If the device driver is user written (see chapter on Interfacing to OASIS)
        the status of the device is dependent upon the device driver subroutine
        accessed by entry point 1.

Other system calls used: GETUCB (21), TSTDEV (58)

Other registers altered: A

Example Calling Sequence:

```
        LD      B,17        ; Point to COMM2 device
        SC      62          ; Get driver status
        •
        •
```

===
SC 63 DEVIN
===

Function: Get input of device driver.

Input parameters:

        Reg B - Logical device number

Output parameters:

        Reg A - Character input

Description:

>The attachment of the specified device is tested. If no device is attached the system call is exited. If a device is attached the physical device driver attached to the logical device specified in the B register is entered at the input entry point. OASIS physical device drivers will not return to the caller until a character is ready. Use system call 62 to test if a character is ready.

>The address of the UCB associated with this device is loaded into the IY register and passed to the device driver along with the B register.

Other system calls used: GETUCB (21), TSTDEV (58)

Other registers altered: none

Example Calling Sequence:

```
        LD      B,17        ; Point to COMM2 device
        SC      63          ; Get device input
        .
        .
```

==
SC 64 DEVOUT
==

Function: Put output to device driver.

Input parameters:

>Reg B - Logical device number
>Reg C - Character to be output

Output parameters: none

Description:

>The attachment of the specified device is tested. If not attached the system call is exited. If attached the physical device driver attached to the logical device specified in the B register is given the character in the C register. OASIS device drivers will save the character and return immediatly. The communication of the character to the device is dependent upon the specific device driver.

>The address of the UCB associated with this device is loaded into the IY register and passed to the device driver along with the B register.

>This system call handles the logic related to printer and console line feed and form feed delays, page parity and listing device closure.

Other system calls used: GETUCB (21), TSTDEV (58)

Other registers altered: A

Example Calling Sequence:

```
        LD      B,17        ; Point to COMM2 device
        LD      C,A         ; Get character to output
        SC      64          ; Output char to device
        .
        .
```

===
SC 66 GETLAB
===

Function: Get disk label of a drive.

Input parameters:

```
        Reg B  - Logical drive number (0 - 7) = (S - G)
        Reg DE - Address of storage area (8 bytes)
```

Output parameters: none

Description:

The drive code is tested for validity: if greater than 7 then the system
call is exited. The specified drive's UCB is tested to determine if the
disk label must be read from the disk - if so then the label is read. The
disk label is transferred to the storage area addressed by the DE register
pair and the system call is exited.

Other system calls used: RD1 (10)

Other registers altered: A

Example Calling Sequence:

```
        LD      B,0         ; Point to system disk
        LD      DE,LABEL    ;
        SC      66          ; Get disk label
        .
        .
LABEL:  DC      '        ',0
```

===
SC 67 PUTDEV
===

Function: Store device driver address.

Input parameters:

```
        Reg B  - Physical device number
        Reg HL - Address of device driver
```

Output parameters: none

Description:

> The device number is verified to be in the range 8-32, if not the system call is exited. The address specified is loaded into the device table, overlaying any current device address in that location of the table. An address of zero (0000) in the HL register pair indicates that the specified device has been unloaded.
>
> This system call is normally only used by the ATTACH command. It will be a lot easier for the user to allow that command to set the driver address as all of the other related house-keeping is performed by the command at that time. This system call might be used by the user for a program that uses a device in a manner different from all other programs and has its own driver for the device embedded in its code.

Other system calls used: GETUCB (21), TSTDEV (58)

Other registers altered: A, D, E, H, L

Example Calling Sequence:

```
        LD      B,17        ; Point to COMM2
        LD      HL,ENTRY    ; Point to device driver
        SC      67          ; Set driver address
        .
        .
```

==
SC 68 DEVUNINIT
==

Function: De-initialize a device driver.

Input parameters:

> Reg B - Logical device number

Output parameters: none

Description:

> The attachment of the specified device is tested. If not attached the system call is exited. If attached the physical device driver attached to the logical device specifed in the B register is entered at the de-initialize entry point.
>
> The address of the UCB associated with this device is loaded into the IY register and passed to the device driver along with the B register.
>
> Note: This system call is used by the ATTACH command when a device is detached and should not be used by user programs.

Other system calls used: GETUCB (21), TSTDEV (58)

Other registers altered: can be all

Example Calling Sequence:

```
        LD      B,17        ; Point to COMM2 device
        SC      68          ; Un-init driver
        .
        .
```

===
SC 69 TSTESCC
===

Function: Test if Program Cancel-key entered.

Input parameters: none

Output parameters:

```
        Flag Z - Status
                Set = Not entered
                Reset = Entered
```

Description:

The system control flag is tested to determine if the Program Cancel-key has been entered. The Program Cancel-key is defined in the System Reference Manual. If the Program Cancel-key has been entered then the Z flag is set and the A register contains a non-zero value. The control flag is cleared by this test process. If the key has not been entered then the Z flag is reset and the A register is set to zero.

The status of the control flag is also cleared by System Call 30 and by the CSI.

The Program Cancel-key is only used by OASIS language products such as the BASIC interpreter, Text Editor, and the Debugger. It would be consistent to use it in user programs that are iterative and/or interactive in function.

Other system calls used: none

Other registers altered: none

Example Calling Sequence:

```
        SC      69          ; Test program cancel
        JR      NZ,NOCAN    ; Jump if not
        .
        .
```

===
SC 70 EXCMDR
===

Function: Execute a program and return.

Input parameters:

> Reg HL — Return address
> Reg DE — Address of command string buffer

Output parameters: none

Description:

> This is the system call used by the system programs BASIC and EDIT when a CSI sub-command is executed. The DE register contains the address of a work area which is the CSI command string along with any options desired, terminated by a CR (13).

> When this system call is executed high memory is set to the address in the HL register, the CSI is loaded and it interprets the command in the work area specified by the DE register pair. This command may be any valid command (including an EXEC) that can fit in the memory available with the exceptions of: DEBUG and ATTACH when the device being ATTACHed is not currently attached to a logical device. These exceptions are due to the fact that those commands would normally cause a program to be loaded into high memory and "protected" at that location.

> After the command has completed its execution control returns to the current high memory location.

> Execution of this system call will disable any and all timer tasks whose TEB location is not included in the "protected" memory area, and a disk error trap set up by SETDET (SC 74).

Other system calls used: EXCMD (54)

Other registers altered: all (unknown)

Example Calling Sequence: not recommended for use by end user.

===
SC 71 BUFFI
===

Function: Get character from buffer.

Input parameters:

> Reg HL — Address of buffer prefix
>> Prefix: Byte 0 = buffer length
>> 1 = current size
>> 2 = current location
>> Prefix followed by buffer storage.

Output parameters:

> Reg A — Next character from buffer
> Flag C — Set if buffer empty

Description:

This system call gets the next character ready for output from a FIFO buffer, probably loaded by system call 72 (BUFFO). The two system calls should be used in conjunction with each other to assist you in maintaining a FIFO stack of up to 256 byte length.

This routine and the BUFFO routine are designed to be operated by interrupt service routines although they could be used for normal processing.

Other system calls used: none

Other registers altered: none

Example Calling Sequence:

```
            LD      HL,BUFFER    ; Point to buffer
            SC      71           ; Get a byte
            .
            .
BUFFER:     DC      128          ; Buffer length
            DC      0            ; Currently used
            DC      0            ; Current byte
            DS      128          ;
```

===
SC 72 BUFFO
===

Function: Add character to buffer.

Input parameters:

```
            Reg A   - character to be added to buffer
            Reg HL  - address of buffer prefix
                     Prefix:  Byte 0 = buffer length
                                   1 = current size
                                   2 = current location
                     Prefix followed by buffer storage.
```

Output parameters: none

Description:

This system call adds one character to a FIFO buffer maintaining the buffer pointers, etc. This routine should be used in conjunction with the BUFFI system call and is designed to be the buffer management for an interrupt service routine, although it could be used for normal programming.

When there is no room in the buffer for the character to be added the routine "hangs" until space becomes available. If the characters are not being removed by an interrupt routine the routine will continue in a two-instruction loop.

Other system calls used: none

Other registers altered: none

Example Calling Sequence:

```
            LD      HL,BUFFER    ; Point to buffer
            SC      72           ; Put a byte
            .
            .
BUFFER:     DC      128          ; Buffer length
            DC      0            ; Currently used
            DC      0            ; Current byte
            DS      128          ;
```

===
SC 73 PUTCON
===

Function: Get/set console control byte.

Input parameters:

> Reg B - Enable mask
> Reg C - Disable mask

Output parameters:

> Reg A - Result

Description:

> The console control byte is a bit-mapped byte controlling the console
> display and keyboard. The byte in the B register is logically ORed with
> the control byte and the byte in the C register is logically 1's
> complemented and ANDed with the control byte. The resulting control byte
> status is returned in the A register. If the B and C registers contain
> zero then the control byte is not changed and the system call merely
> returns the status of the control byte.

> The bit-mapping of the control byte is as follows:

> Bit Function

> 7   Echo, on/off. When this bit set then all non-control characters
>   typed on CONIN are displayed on CONOUT, after conversion due to
>   the status of the other bits in this control byte.

> 6   Fold to upper. When this bit is set then all lowercase
>   characters typed on CONIN are converted to uppercase.

> 5   Fold to lower. When this bit is set and bit 6 is off then all
>   characters typed on CONIN are converted to their inverse
>   casemode (only letter characters are affected).

Bits 5 and 6 function as a unit:

```
6     5     Function
---   ---   ----------------------------------
OFF   OFF   No translation
ON    X     Translate to upper
OFF   ON    Translate to inverse
```

4   CTRL stop. When this bit is set then entry of any control character (value less than 32) will terminate the input.

3   CTRL delete. When this bit is set then all control characters typed on CONIN are ignored (except BS (8), TAB (9), CR (13), and CAN (24)).

2   CTRL graphic. When this bit is set and bit 7 is set then all control characters typed on CONIN are displayed on CONOUT in their graphic equivalent (an up arror (^) followed by the character equal to the control character + 64).

1   Not used.

0   Stack. Indicates EXEC stacked data available. This bit is not changeable by the system call.

Other system calls used: none

Other registers altered: none

Example Calling Sequence:

```
        ; The following instructions will set the console
        ; control byte to perform the following:
        ;       set echo on
        ;       no case translation
        ;       accept and display CTRL char in graphics
        ;
LD      B,10000100B    ; Enable mask
LD      C,01111010B    ; Disable mask
SC      73             ; Set console control
.
.
```

===
SC 74 PUTDET
===

Function: Trap disk errors before message displayed.

Input parameters:

Reg HL — address of user error routine

Output parameters: none (see description)

Description:

      This system call does not have any output parameters upon return to the calling program; however, when a disk error does occur certain registers do have defined values:

                  Reg B - disk drive number
                  Reg DE - relative sector number
                  Reg HL - memory location of disk buffer
                  Reg A - disk error code
                          1 = Disk not ready
                          2 = Disk write protected
                          3 = Disk not initialized
                          4 = Data CRC error
                          5 = Invalid parameters
                          6 = Disk label changed
                          7 = Sector not found
                          8 = Track not found
                          9 = Address (sector/track header) CRC error

      When the disk error occurs control is transferred to the address specified in the HL register pair. After your routine has done its processing and is ready to return control to OASIS the A register should be set to one of the following values:

                        00 Ignore error
              01 - FE Retry operation (no change)
                    FF Quit - return to CSI

      To disable your disk error routine then use this system call with the HL register containing 0. (Your routine will automatically be disabled when the CSI is loaded.)

      An example of the use of this system call is the VERIFY command. That command performs disk readability diagnostics and therefore needs to gain control when a disk error occurs.

Other system calls used: none

Other registers altered: none (see description)

Example Calling Sequence:

```
        LD      HL,DISKERR   ; Point to error routine
        SC      74           ; Inform OS
        .
        .
```

===
SC 75 NEWSYS
===

Function: Change system disk.

Input parameters:

        Reg B - new physical drive number (0 - 7)

Output parameters: none

Description:

        This system call performs the same operation as the ATTACH command when the system disk is to be changed. Register B is loaded with the new physical drive number of the system disk. When the system call is executed the current system disk is accessed to read in any necessary overlays, a message is displayed to the operator asking for the new system disk to be mounted in the specified drive. After the operator loads the disk and responds to the message the new system disk is accessed, the necessary SYSTEM files are located and control returns to the CSI.

        The new system disk must contain a SYSTEM.NUCLEUS of the same version as the current system disk. The results will be unpredictable if the version is different.

Other system calls used: BOOT (0)

Other registers altered: none

Example Calling Sequence:

```
        LD      B,0         ; Point to drive 0
        SC      75          ; Change system disk
        .
        .
```

===
SC 76 DELAY
===

Function: Delay processing for specified period of time.

Input parameters:

        Reg A - formated delay time.
            Bit 7,6 - Unit of measure
                    00 = 1/1000 (millisecond)
                    01 = 1/100 second
                    10 = 1/10 second
                    11 = 1 second
            Bit 5-0 - count (1 - 63)

Output parameters: none

Description:

        This is a general purpose processing delay routine. It was developed for the timing delay required by serial I/O devices but can be used for any purpose. When the system call is executed the formated delay factor in the A register is decoded into milliseconds and a TEB is initiated for the

specified time.   Then the system call waits for the   TEB to be exhausted
before returning control to the calling program.

Although you have access to the TEB syscall and MSEC this is a much easier
and straight-forward method of long delays (up to a minute).

Processing of your program is suspended for the specified length of time
but all interrupt service routines are still enabled.

Other system calls used: MSEC (53)

Other registers altered: none

Example Calling Sequence:

```
            LD      A,(DELAY)      ; Set up for delay
            SC      76             ; Delay processing
            •
            •
   DELAY:   DC      11000101B      ; 5 second interval
```

===
SC 77 GETACB
===

Function: Point to Assign Control Block entry.

Input parameters:

         Reg B   - ACB number (0 - 16)

Output parameters:

         Reg HL - Address of ACB

Description:

        The address of any assigned Assign Control Block for the number specified
        by the contents of the B register is returned in the HL register pair.
        This ACB is not the ACB address used in system call 24 but the internal
        copy of that ACB.

Other system calls used: none

Other registers altered: A

Example Calling Sequence: not recommended for use by end user.

===
SC 78 CONESC
===

Function: Analyze escape sequence and execute if system defined.

Input parameters:

    Reg A - Second character of escape sequence

Output parameters:

    Reg A - Status:
             00 System handled
                unchanged = undefined

Description:

This system call is used by the SYSTEM.CLASSnn files to cause the system to act on a system defined escape sequence. When an escape character is detected the next character received is loaded into the A register and this system call is executed. If the system recognizes the character as a command then the appropriate action is taken, the A register is cleared and the Z flag is set. If the character is not recognized then the A register is left unchanged and the Z flag is reset.

This system call could be used by a program to force a system defined function such as toggling the printer echo feature, etc. Merely load the A register with the character corresponding to the second character of the escape sequence that would be used to invoke the function from the keyboard. For a listing of these functions and character see the OASIS System Reference Manual, "System Control Keys".

Other system calls used: none

Other registers altered: none (may not return if A reg contains 'Q')

Example Calling Sequence:

    LD   A,'P'   ; Toggle the PRT echo
    SC   78   ;
    •
    •

==
SC 79 SNU
==

Function: Select next user.

Input parameters: none

Output parameters: none

Description:

The next active user partition is selected and control of the system transfers to it.

Although this system call is used by all other system calls that are waiting for action (input/output operations) you should use it in any code that is performing a wait without a system call. (The next user will be

selected automatically when your time slice elapses but the performance of the system will be enhanced if you can give up control instead of just looping.)

When your user partition is activated again your program will continue execution at the instruction after this system call.

Note: On single user system this system call returns immediately.

Other system calls used: none

Other registers altered: none

Example Calling Sequence:

```
          SC        79              ; Select next user
```

===
SC 80 GETBASE
===

Function: Get monitor (NUCLEUS) location.

Input parameters: none

Output parameters:

        Reg IY - Monitor address

Description:

        The first address of the SYSTEM.NUCLEUS is placed in the IY index register
        and the system call is exited.

Other system clls used: none

Other registers altered: none

Example Calling Sequence:

```
          SC        80          ; Get NUCLEUS base
          LD        (BASE),IY   ; Save
          •
          •
```

===
SC 81 GETMFG
===

Function: Get manufacturer number of system.

Input parameters: none

Output parameters:

        Reg A   - Manufacturer number

Description:

>   Each computer manufacturer that supports the OASIS operating system is
>   assigned a unique value. This value can be accessed with this system call
>   and used to determine if the manufacturer is the same as required by the
>   program requesting it (some programs may use hardware dependant code). By
>   using this system call a program can determine what type of computer it is
>   running on.

Other system calls used: none

Other registers altered: A

Example Calling Sequence:

```
SC        81            ; Get MFG number
LD        (MFG),A       ; Save
```

===
SC 82 GETPIN
===

Function: Get your user partition number.

Input parameters: none

Output parameters:

>   Reg A   - Your user partition identification number (PIN)

Description:

>   Your user partition identification number is return in the A register.

>   Note: On single user systems this system call will always return a 1.

Other system calls used: none

Other registers altered: none

Example Calling Sequence:

```
SC        82                 ; Get PIN
```

===
SC 83 UNLOCK
===

Function: Release a file record for another partition's use.

Input parameters:

>   Reg BC - Address of key, indexed files or
>   Record number, direct files
>   Reg DE - Address of FCB

Output parameters: none

Description:

>The sectors of the record currently locked in the file referenced by the FCB are unlocked, allowing other users to access it. If the record is not currently locked or the system is a single-user system then nothing is performed except the return from the system call.

Other system calls used: none

Other registers altered: none

Example Calling Sequence:

```
            LD      BC,(RECNUM)    ; Get record number
            LD      DE,FCB1)       ; Point to FCB, ch 1
            SC      83             ; Unlock the record
            .
            .
            .
RECNUM:     DS      2              ; Current record number
FCB1:       DC      1,01011000B    ; Direct I/O
            DW      IOBUFF1        ; I/O buffer address
```

===
SC 84 ONEONLY
===

Function: To indicate that your partition has exclusive use of a function/resource.

Input parameters:

>Reg HL — Address of semaphore

Output parameters: none

Description:

>The byte addressed by the HL registers is tested to determine if another user has exclusive control of it. If no other user has control then the byte is flagged to indicate that you have control and the system call is exited. If another user does have control then the next user is selected; upon return to your partition the byte is tested again, etc.

>The byte addressed by the HL registers should be in a system area, such as a loaded re-entrant program.

>Note: On a single user system this system call returns immediately.

Other system calls used: SNU (79)

Other registers altered: none

Example Calling Sequence:

```
        LD      HL,USERFLAG   ; Point to your user communication flag
        SC      84            ; Get exclusive use of flag
        .
        .
        .
        LD      HL,USERFLAG   ; Point to your user communication flag
        SC      85            ; Release exclusive use of flag
```

==
SC 85 NOTONLY
==

Function: To release exclusive use of a function/resource.

Input parameters:

        Reg HL - Address of semaphore

Output parameter: none

Description:

        The byte addressed by the HL registers is tested to determine if another
        user has exclusive control of it. If your partition has exclusive control
        of the byte then that control is released. If your partition does not
        have exclusive control of the byte then the system call is exited with no
        action taken.

        The byte addressed by the HL registers should be in a system area, such as
        a loaded re-entrant program.

        Note: On a single user system this system call returns immediately.

Other system calls used: none

Other registers altered: none

Example Calling Sequence:

```
        LD      HL,USERFLAG   ; Point to your user communication flag
        SC      84            ; Get exclusive use of flag
        .
        .
        .
        LD      HL,USERFLAG   ; Point to your user communication flag
        SC      85            ; Release exclusive use of flag
```

==
SC 86 SETPIN
==

Function: To activate another, specific partition to execute some code.

Input parameters:

              Reg A - Partition number to activate
              Reg HL - Address to start execution at

Output parameters: none

Description:

       This system call is used by the system and the multi-user commands START,
       STOP, FORCE, and MSG to cause another partition to become active and
       execute some code.

       It is advised that the end user does not use this system call.

Other system calls used: none

Other registers altered: none

===
SC 87 GETLUB
===

Function: Get base address of LUB table.

Input parameters: none

Output parameters:

              Reg IY - Base address of LUB table

Description:

       This system call is used in some system commands. The user should not use
       it.

Other system calls used: none

Other registers altered: A

===
SC 88 MSG
===

Function: Send a message to another user's console.

Input parameters:

              Reg A - Partition number to send message to
              Reg DE - Address of message to send

Output parameters: none

Description:

> This system call is not intended for general usage.
>
> The message addressed by the DE register pair is displayed on the user's console owned by the partition specified in the A register. If the partition is invalid, or inactive the system call will return immediately. If the destination's message switch is set off the message will still be sent.
>
> Note: On a single user system this system call returns immediately.

Other system calls used: SETPIN (86)

Other registers altered: none

===
SC 89 EXCLUSIVE
===

Function: To gain exclusive control of key system tables, etc.

Input parameters: none

Output parameters: none

Description:

> Certain critical system tables are locked so that other partitions cannot access them, thus allowing your program to alter them in some way without damage to other user's processes.
>
> The system tables locked by this system call include: PCB table, schedule table, mailbox file, etc.
>
> Note: On a single user system this system call returns immediately.

Other system calls used: ONEONLY (84)

Other registers altered: none

Example Calling Sequence:

```
        SC      89              ; Get exclusive use of system tables
        .
        .
        .
        SC      90              ; Release exclusive use of system tables
```

===
SC 90 UNEXCLUSIVE
===

Function: Release exclusive control of system tables.

Input parameters: none

Output parameters: none

Description:

> The critical system tables locked by SC 89 are released for other user's use.

> The system tables released by this system call include: PCB table, schedule table, mailbox file, etc.

> Note: On single user systems this system call returns immediately.

Other system calls used: NOTONLY (85)

Other registers altered: none

Example Calling Sequence:

```
        SC        89              ; Get exclusive use of system tables
        •
        •
        •
        SC        90              ; Release exclusive use of system tables
```

===
SC 91 GETWORK
===

Function: Get base address of your SCR work area.

Input parameters: none

Output parameters:

> Reg HL - Address of 256 byte SCR work area

Description:

> The first address of the start of the 256 byte work buffer used by your partition's System Communication Region is return in the HL register pair.

Other system calls used: GETSCR (48)

Other registers altered: none

===
SC 92 GETPRIV
===

Function: Get the current privilege level of user.

Input parameters: none

Output parameters:

> Reg A - Privilege level of user

Description:

The current privilege level of the user is returned in the A register.

Other system calls used: none

Other registers altered: none

Example Calling Sequence:

```
        SC      92              ; Get privilege level
        CP      3               ; Compare with 3
        JR      NC,OKAY         ; BRIF less
        SC      0               ; Else exit
```

===
SC 93 COMPARE
===

Function: Perform string comparison.

Input parameters:

```
        Reg BC - Length
        Reg DE - Address of string 1
        Reg HL - Address of string 2
```

Output parameters:

When string 1 = string 2:

```
        Reg BC - 00
        Reg DE - Address of byte following string 1
        Reg HL - Address of byte following string 2
        Flag Z - Set
        Flag C - Reset
```

When string 1 <> string 2:

```
        Reg BC - Count of bytes remaining
        Reg DE - Address of string 1 byte not equal
        Reg HL - Address of string 2 byte not equal
        Flag Z - Reset
        Flag C - Set if string 2 > string 1; reset otherwise
```

Description:

The string of characters addressed by the DE register is compared with the string of characters addressed by the HL register for the number of characters specified by the BC register. If the two sequences of characters exactly equal each other then the Z flag is set and the C flag is reset. If the two strings do not equal each other then the Z flag is reset and the C flag is set if the second string is greater in value than the first string.

Other system calls used: none

Other registers altered: A

Example Calling Sequence:

```
                LD      A,(STRING1)    ; Get length
                LD      C,A            ; Copy to C reg
                LD      B,0            ;
                LD      DE,(STRING1+1) ; Point to string
                LD      HL,(STRING2+1) ; Point to string
                SC      93             ; Compare strings
                JR      Z,.MATCH       ; BRIF equal
                .
                .
                .
     STRING1: DC      5,'ABCDE'
     STRING2: DC      5,'ABCde'
```

==
SC 94 RDBIN
==

Function: Get binary data stream from sequential file.

Input parameters:

```
          Reg B  - Byte count to get
          Reg HL - Storage area
          Reg DE - Address of FCB
```

Output parameters:

```
          Reg A  - Return code
                   00 Successful
                   01 End of file
                   08 Invalid file number
                   FF File not open
          Flag Z - Status:
                   set - okay (Reg A = 0)
                   reset - error (Reg A <> 0)
```

Description:

   The required I/O overlay is loaded, if necessary. The ACB is validated:
   open, sequential, and input. The A register is set to 255 if ACB invalid.
   The ACB is tested for an EOF condition and the appropriate return code is
   set if true and the system call is exited. If everything is okay the
   number of bytes indicated is read in from the file and transferred to the
   buffer designated by the HL register pair.

   This system call, like all logical record input/output system calls,
   maintains the Sector Lock Table (SLT) according to the FCB.

Other system calls used: KEYIN (1), RD1 (10), DEVIN (63)

Other registers altered: B, C

Example Calling Sequence:

```
                LD      B,25            ; Get next 25 bytes from file
                LD      DE,FCB1         ; Point to file on ch 1
                LD      HL,BUFF         ; Put in BUFF buffer
                SC      94              ;
                JR      NZ,CHKERR       ; Jump if read error
                .
                .
FCB1:           DC      1,10010000B     ; Sequential input, ch 1
                DW      BUFF1           ; I/O buffer
BUFF1:          DS      256             ;
BUFF:           DS      25              ; Input buffer
```

==
SC 95 WRBIN
==

Function: Put binary data stream to sequential file.

Input parameters:

          Reg B   - Byte count to write
          Reg DE - Address of FCB
          Reg HL - Address of data to write

Output parameters:

          Reg A   - Return code
                    00 Successful
                    08 Invalid file number
                    10 Disk full
                    FF File not open
          Flag Z - Status:
                    set - okay (Reg A = 0)
                    reset - error (Reg A <> 0)

Description:

    The required I/O overlay is loaded, if necessary. The ACB is validated:
    open, sequential, and output or append. The appropriate return code is
    set when invalid and the system call is exited. The number of bytes
    specified in the B register are writen to the file buffer and physical
    output is performed as required. When the file is a disk file and the
    file requires more allocation to perform the physical output then the file
    is expanded.

    This system call, like all logical record input/output system calls,
    maintains the Sector Lock Table (SLT) according to the FCB.

Other system calls used: DISPLAY (2), WR1 (11), ALLOC (27), DEVOUT (64)

Other registers altered: B, C

Example Calling Sequence:

```
            LD        B,25              ; Write 25 bytes to file
            LD        DE,FCB1           ; On ch 1
            LD        HL,BUFF           ; From buffer BUFF
            SC        95                ;
            JR        NZ,CHKERR         ; Jump on error
            .
            .
FCM1:       DC        1,10001000B       ; Ch 1, seq
            DW        BUFF1             ;
BUFF1:      DS        256               ; I/O buffer
BUFF:       DS        25                ; Data buffer
```

==
SC 96 ERRDIS
==

Function: Display error message on console

Input parameters:

            Reg DE — Tokenized parameter list
            Reg HL — Error number

Output parameters: none

Description:

        This system call is used by all system program to display error and
        standard information messages kept in the SYSTEM.ERRMSG file (see System
        Reference Manual).

        The DE register pair need only be loaded with the address of the tokenized
        parameter list if the message contains parameter replacement codes. The
        tokenized parameter list is a list of parameters in ASCII, each parmeter
        eight (8) bytes in length with no delimiting characters. Use trailing
        spaces if the parameter is not eight characters.

Other system calls used: DISPLAY (2)

Other registers altered: none

Example Calling Sequence:

```
            LD        DE,PARAM          ; Point to parameters
            LD        HL,47             ; Display message # 47
            SC        96                ;
            .
            .
PARAM:      DC        '123     HELLO    '
```

==
SC 97 ERRQUIT
==

Function: Display error message and re-boot.

Input parameters:

                Reg DE - Address of parameter list
                Reg HL - Message number
                Reg A   - Return code

Output parameters: none

Description:

        This system call is identical to system call 96 (ERRDIS) except that
        control does not return to the calling program. After the message is
        displayed control will return to the CSI with the return code set to the
        value in the A register.

Other system calls used: ERRDIS (96), DISPLAY (2), QUIT (0)

Other registers altered: all (no return)

Example Calling Sequence:

                LD   HL,23   ; Message # 23
                LD   A,4   ; Return code = 4
                SC   97   ;
                •
                •

==
SC 98 OVERLAY
==

Function: Program overlay load (for system use only).

Input parameters:

                Reg A   - Directory type (1 = relocatable, 2 = absolute)
                Reg B   - Drive code
                Reg DE - Starting disk address of program
                Reg HL - Address of overlay list:
                        0-1 Memory address to load into
                        1-3 Length to load, in bytes
                        4   Number of sectors to load
                        5-6 Sector number, relative to program start

Output parameters: none

Description:

        The overlay segment of your program indicated by the input parameters is
        loaded into memory at the address indicated. This system call always

performs the overlay, even if it is the same overlay as is already in memory. Therefore, it is the responsibility of your program to test whether the overlay is needed.

The drive code and starting sector number of your program used in the input registers B and DE respectively are available when your program is first invoked by the CSI. For more information refer to the chapter "Interfacing to OASIS" in this manual.

When the overlay is relocatable the sector count of the overlay must include the relocation table.

Other system calls used: RD (50)

Other registers altered: none

Example Calling Sequence:

```
              LD        A,(OVERLAY)    ; Get current overlay number
              CP        1              ; Test if already loaded
              JR        Z,OVERLAY+1    ; BRIF is
              LD        A,1            ; Segment is relocatable
              LD        B,(PRGDRIVE)   ; Drive code of program
              LD        DE,(PRDSECT)   ; Starting sector of program
              LD        HL,SEG1TABLE   ; Overlay table 1
              SC        98             ; Get overlay
              JR        OVERLAY+1      ; Continue in overlay
              .
              .
              .
SEG1TAB:      DC        (OVERLAY)      ; Address of overlay region
              DC        (OVEREND-OVERLAY+1) ; Overlay region length
              DC        4              ; Sector count, including rel table
              DC        23             ; Relative sector # of segment
```

==
SC 99 CALLOC
==

Function: Conditional allocation.

Input parameters:

```
              Reg B   - Logical drive code (0 - 7) = (S - G)
              Reg DE  - Maximum desired blocks of allocation
              Reg HL  - Minimum desired blocks of allocation
```

Output parameters:

```
              Reg DE  - Actual number of blocks allocated
              Reg HL  - Sector number of first block allocated
              Flag Z  - Set if able to allocate minimum; reset otherwise
```

Description:

> The disk is tested for its largest contiguous area available. If this
> area is smaller in size than that requested for the minimum allocation the
> Z flag will be reset and the system call exited. If this area is at least
> the size of the minimum allocation requested space will be allocated, up
> to the maximum space requested. The return registers are set to reflect
> the amount and location of the space actually allocated.

Other system calls used: RD1 (10), WR1 (11), ALLOC (27)

Other registers altered: A

Example Calling Sequence:

```
        LD      B,1           ; Point to A drive
        LD      DE,20         ; Maximum of 20 blocks
        LD      HL,4          ; Minimum of 4 blocks
        SC      99            ; Allocate space
        JR      NZ,NOSPACE    ; Insufficient space
        LD      (SIZE),DE     ; Save actual alloc size
        LD      (SECT),HL     ; Save first sect number
        .
        .
        .
```

===
SC 100 DISPATCH
===

Function: Perform table lookup.

Input parameters:

> Reg DE – Address of string to lookup
> Reg HL – Address of start of table
> > Table: Minimum spelling
> > Match string
> > Related address

Output parameters:

> Reg HL – Related address if match found
> Flag Z – Set if match found
> > reset otherwise

Description:

> The table designated by the HL register pair is searched for a match with
> the string addressed by the DE register pair. If a match is found the Z
> flag is set and the HL register pair is loaded with the third field in the
> matching table entry. If no match is found the Z flag is reset and the HL
> register pair is undefined.
>
> The string addressed by the DE register pair and the strings in the table
> are of variable length. The string to look up is terminated by a

non-alphanumeric non dollar sign character. The last character of the strings in the table is marked with the parity bit (bit 7 on). This is automatically performed by the assembler when the double quote mark is used (see DC directive).

The end of the table is marked with a binary zero entry.

Other system calls used: none

Other registers altered: A, B, C

Example Calling Sequence:

```
          LD      DE,STRING      ; Point to string
          LD      HL,TABLE       ; Point to table
          SC      100            ; Lookup
          JR      NZ,NOTFOUND    ; BRIF not found
          JP      (HL)           ; Else branch to related address
          .
          .
          .
STRING:   DC      'THIS IS A STRING',0
TABLE:    DC      1              ; Minimum spelling
          DC      "FILELIST"     ; Match string
          DC      (FILELIST)     ; Related routine
          DC      4,"FILT8080",(FILT8080)
          DC      2,"FORCE",(FORCE)
          DC      10,"THIS IS A STRING",(EXIT)
          DC      0              ; End of table
```

==
SC 101 GETUSER
==

Function: Get the current user account number.

Input parameters: none

Output parameters:

Reg A - User account number id

Description:

The user id number currently logged onto this partition is returned in the A register. The user id number is the number used by the system to distinguish different owning accounts. The system accounts have an id number of zero; user accounts have an id number in the range 1 - 254.

Other system calls used: none

Other Registers altered: none

Example Calling Sequence:

```
          SC      101            ; Get user id
```

```
        LD        (CUR$USER),A   ; Save it
                  .
                  .
                  .
```

===
SC 102 CHARIN
===

Function: Perform console input character translate and escape sequence actions.

Input parameters:

        Reg A - Character input
        Reg IY - Address of UCB

Output parameters:

        Reg A - Character to be used
        Flag C - Set if character to be ignored by driver
                 reset otherwise

Description:

   This system call provides a simple and consistent method for a device
   driver to make sure that the OASIS system console escape sequences are
   handled properly. It is advised that all user written device drivers that
   accept input from a device and that might be attached as a console device
   use this system call for each character that is input. (The driver should
   check to see if it is the console first to improve performance.)

   This system call tests to see if the device is the conole input device.
   If not then the system call is exited with the carry flag reset. When the
   device is the console the system call checks to see if there is a
   SYSTEM.CLASSnn file loaded--if so then the character is passed to that
   routine. If not then the character is checked to see if it is part of an
   escape sequence. When the character is part of an escape sequence from
   the console input device the appropriate action is taken and the carry
   flag is set before the system call is exited.

Other system calls used: CONESC (78)

Other registers altered: A

===
SC 103 PUTVECT
===

Function: To insert an interrupt vector address into the system table.

Input parameters:

        Reg A - Relative vector number
        Reg DE - Vector transfer address

Output parameters: none

Description:

This system call is used to inform the operating system where an interrupt service routine is located at. It is mandatory that this system call be used for this purpose in a multiuser, multi-memory bank system and it should be used in all other types of systems for convenience and consistency.

The relative vector number in the A register is a number in the range of 2 - 6 (mode 0), 0 - 7 (mode 1) or 0 - 127 (mode 2), corresponding to the desired priority of the interrupt (mode 1) or the vector number that the interrupting device will give to the system when it interrupts (mode 0 or 2). The interrupt service routines for the three modes of interrupts are similar except the mode 1 service routine must first poll its device to determine if it was the device causing the interrupt; if not then the routine performs a return without enabling interrupts (the system will call the next routine in the vector table). The relative vector number for mode 1 determines the "priority" or sequence that the service routine will be called when an interrupt occurs.

The vector transfer address in the DE register pair is the address of the interrupt service routine for the vector number in the A register. When the system has multiple memory banks available to it the operating system will keep track of which bank that particular address is in.

Other system calls used: none

Other registers altered: none

Example Calling Sequence:

```
        LD      A,2             ; Vector number 2
        LD      DE,INT          ; Input interrupt
        SC      103             ; Put vector
```

==
SC 104 GETBYTE
==

Function: Transfer byte(s) from another partition space.

Input parameters:

        Reg A   - Partition identification number of partition to get from
        Reg BC - Count of bytes to get
        Reg DE - Address of buffer to transfer bytes to (your partition)
        Reg HL - Address of buffer to transfer bytes from (his partition)

Output parameters:

        Reg BC - 0
        Reg DE - Address following bytes transferred (your partition)
        Reg HL - Address following bytes transferred (his partition)
                  Interrupts are disabled

Description:

This system call functions similar to an LDIR instruction in a single user system.

In a multi-user system this system call allows you to transfer a character or string of characters from another partition to your partition, even though that other partition may be in different bank of memory.

Note: Upon return from this system call interrupts have been disabled. It is your responsibility to re-enable them if they should be on.

Other system calls used: none

Other registers altered: none

Example Calling Sequence:

```
LD      A,2         ; From partition two
LD      HL,5000     ; From his location 5000
LD      DE,4F00     ; To my location 4000
LD      BC,20       ; For 32 bytes
SC      104         ; Transfer
EI                  ; Enable interrupts
```

===
SC 105 PUTBYTE
===

Function: Transfer byte(s) to another partition space.

Input parameters:

Reg A - Partition identification number of partition to put to
Reg BC - Count of bytes to put
Reg DE - Address of buffer to transfer bytes to (his partition)
Reg HL - Address of buffer to transfer bytes from (your partition)

Output parameters:

Reg BC - 0
Reg DE - Address following bytes transferred (his partition)
Reg HL - Address following bytes transferred (your partition)
         Interrupts are disabled

Description:

This system call functions similar to an LDIR instruction in a single user system.

In a multi-user system this system call allows you to transfer a character or string of characters to another partition from your partition, even though that other partition may be in different bank of memory.

Note: Upon return from this system call interrupts have been disabled. It is your responsibility to re-enable them if they should be on.

Other system calls used: none

Other registers altered: none

Example Calling Sequence:

```
        LD      A,2             ; To partition two
        LD      HL,5000         ; From my location 5000
        LD      DE,4F00         ; To his location 4000
        LD      BC,20           ; For 32 bytes
        SC      105             ; Transfer
        EI                      ; Enable interrupts
```

===
SC 106 DATEOUT
===

Function: Translate a packed BCD date to string format.

Input parameters:

```
        Reg C  - Month number (BCD)
        Reg H  - Day number (BCD)
        Reg L  - Year number (BCD)
        Reg DE - Storage area
```

Output parameters:

```
        Reg DE - End of formatted date plus one
```

Description:

The date specified by the C, H, and L registers is converted and formatted according to the currently set DATEFORM.

Note: This system call does not validate the date input.

Other system calls used: none

Other registers altered: none

Example Calling Sequence:

```
        LD      C,03            ; Month 3 - March
        LD      H,22            ; Day 22
        LD      L,93            ; Year 1993
        LD      DE,BUFFR        ; Storage area
        SC      106             ; Convert to string form
```

===
SC 107 WAITINT
===

Function: Deactivate current partition until interrupt occurs.

Input parameters: none

Output parameters: none

Description:

> This system call is similar to system call 79 (SNU) in that the next user partition is activated. Unlike SC 79 this system call informs the operating system that the current partition is not to be activated again until an interrupt occurs that needs this partition to be serviced.
>
> This system call allows greater throughput for a multi-user system in that any partition using it that is waiting for an event to happen (i.e., waiting for the operator to type another key) will not waste a lot of CPU time merely determining that it is still waiting.
>
> When this system call is used (as it is in all OASIS supplied device drivers) control will return to the instruction following the call when any interrupt occurs from a device attached to this partition. However, the interrupting device may not be the event that was required by your partition. Therefore upon return to your program you should re-check the status of the device that you were waiting for.

Other system call used: none

Other registers altered: none

Example Calling Sequence:

```
IN:     CALL    STATUS      ; Check status of device
        JR      NZ,IN1      ; Skip if ready
        SC      107         ; Else deactivate
        JR      IN          ; Re-check status
IN1:
```

==
SC 108 FINDPGM
==

Function: Return address of a loaded, re-entrant program.

Input parameters:

> Reg DE – Address of program name desired (8 characters)

Output parameters:

> Flag Z – Set if found
>   Reset otherwise
> Reg HL – Address of program if found

Description:

> This system call searches the Re-entrant Program Table (RPB) for a match with the program name specified by the DE register pair. If the program is found in the table the starting address is loaded into the HL register pair, the Z flag is set and the system call exited.

When the program name specified by the DE register pair is not found in
the RPB the Z flag is reset and the system call is exited.

Other system calls used: none

Other registers altered: none

Example Calling Sequence:

```
            LD       DE,NAME         ; Point to program name
            SC       108             ; Find program
            JR       NZ,LOADIT       ; BRIF not found
            JP       (HL)            ; Else branch to loaded program

    NAME:   DC       'BASIC      '
```

8.1 Addressing modes

Most of the Z80 instructions operate on data stored in internal CPU registers, external memory or in the I/O ports. Addressing refers to how the address of this data is generated in each instruction. This section gives a brief summary of the types of addressing used in the Z80.

Immediate – In this mode of addressing the byte following the op-code in memory contains the actual operand. Examples of this type of instruction would be to load the accumulator with a constant, where the constant is the byte immediatley following the op-code.

<p align="center">LD A,25</p>

Immediate Extended – This mode is merely an extension of immediate addressing in that the two bytes following the OP code are the operand. Examples of this type of instruction would be to load the HL register pair with 16 bits of data.

<p align="center">LD HL,LABEL</p>

Modified Page Zero Addressing – The Z80 has a special single byte call instruction to any of 8 locations in page zero of memory. This instruction (referred to as a restart) sets the Program Counter (PC) to an effective address in page zero. The value of this instruction is that it allows a single byte to specify a complete 16 bit address where commonly called subroutines are located, thus saving memory space.

<p align="center">RST 38</p>

Relative Addressing – Relative addressing uses one byte of data following the op-code to specify a displacement from the existing program to which a program jump can occur. This displacement is a signed two's complement number that is added to the address of the op-code of the following instruction.

<p align="center">JR LABEL</p>

The value of relative addressing is that it allows jumps to nearby locations while only requiring two bytes of memory space. For most progams, relative jumps are by far the most prevalent type of jump due to the proximity of related program segments. The signed displacement can range between +127 and −128. Another advantage is that it allows for relocatable code.

Extended Addressing – Extended addressing provides for two bytes (16 bits) of address to be included in the instruction. This data can be an address to which a program can jump or it can be an address where an operand is located. Extended addressing is required for jumps with a displacement greater than 127.

<p align="center">LD HL,(LABEL)</p>

When extended addressing is used to specify the source or destination address of an operand, the notation (nn) is used to indicate the content of memory at nn, where nn is the 16 bit address specified in the instruction. This means

that the two bytes of address nn are used as a pointer to a memory location. The use of the parentheses always means that the value enclosed within them is used as a pointer to a memory location. For example, (1200) refers to the contents of memory at location 1200.

Indexed Addressing – In this type of addressing the byte of data following the op-code contains a displacement which is added to one of the two index registers (the op-code specifies which index register is used) to form a pointer to memory. The contents of the index register are not altered by this operation. An example of an indexed instruction would be to load the contents of the memory location (Index Register + displacement) into the accumulator. The displacement is a signed two's complement number. Indexed addressing greatly simplifies programs using tables of data since the index register can point to the start of any table. Two index registers are provided since very often operations require two or more tables. Indexed addressing also allows for relocatable code.

<div align="center">

LD HL,(IX+3)

</div>

To indicate indexed addressing the notation: (IX+d) or (IY+d) is used. Here d is the displacement specified after the OP code. The parentheses indicate that this value is used as a pointer to external memory.

Register Addressing – Many of the Z80 op-codes contain bits of information that specify which CPU register is to be used for an operation. An example of register addressing would be to load the data from register B into register C.

<div align="center">

LD A,B

</div>

Implied Addressing – Implied addressing refers to operations where the op-code automatically implies one or more CPU reigisters as containing the operands. An example is the set of arithmetic operations where the accumulator is always implied to be the destination of the results.

<div align="center">

ADD C

</div>

Register Indirect Addressing – This type of addressing specifies a 16 bit CPU register pair (such as HL) to be used as a pointer to any location in memory. This type of instruction is very powerful and it is used in a wide range of applications. The symbol (HL) specifies that the contents of the HL register are to be used as a pointer to a memory location.

<div align="center">

LD A,(HL)

</div>

Bit Addressing – The Z80 contains a large number of bit set, reset and test instructions. These instructions allow any memory location or CPU register to be specified for a bit operation through one of three previous addressing modes (register, register indirect and indexed) while three bits in the op-code specify which of the eight bits is to be manipulated.

<div align="center">

SET 3,D

</div>

Many instructions include more than one operand (such as arithmetic instructions or loads). In these cases, two types of addressing modes may be employed.

<div align="center">

BIT 7,(IX)

</div>

8.2 Registers

The Z80 CPU contains 208 bits of Read/Write static memory that are accesible to the programmer. This memory is configured into eighteen 8 bit registers and four 16 bit registers.

General Purpose Registers

There are two matched sets of general purpose registers, each set containing six 8 bit registers that may be used individually as 8 bit registers (B, C, D, E, H, L) or as 16 bit register pairs by the programmer. One set is called BC, DE, and HL while the complementary set is called BC', DE', and HL'. At any one time the programmer can select only one set of registers to work with, althrough a single exchange command exchanges the contents of the entire set. In systems where fast interrupt response is reguired, one set of general purpose registers and an accumlator/flag register may be reserved for handling this very fast routine. Only a simple exchange command need be executed to go between the routines.

Accumulator and Flag Registers

The CPU includes two independent 8 bit accumulators (A and A') and associated 8 bit flag registers (F and F'). The accumulator holds the results of 8 bit arithmetic or logical operations while the flag register indicates specific conditions for 8 or 16 bit operations, such as indicating whether or not the result of an operation is equal to zero. The programmer selects the accumulator and flag pair that he wishes to use with a single exchange instruction so that he may easily work with the contents of either pair.

Special Purpose Registers

1. **Program Counter (PC).** The Program Counter holds the 16 bit address of the current instruction being fetched from memory. The PC is automatically incremented after its contents have been transferred to the address lines. When a program jump occurs the new value is placed in the PC, overriding the incrementer.

2. **Stack Pointer (SP).** The stack pointer holds the 16 bit address of the current top of a stack located anywhere in external system RAM memory. The external stack memory is organized as a last-in, first-out (LIFO) file. Data can be pushed onto the stack from specific CPU registers or popped off of the stack into specific CPU registers through the execution of PUSH and POP instructions. The data popped from the stack is always the last data pushed onto it. The stack allows simple implementation of multiple level interrupts, unlimited subroutine nesting and simplification of many types of data manipulation.

3. **Two Index Registers (IX and IY).** The two independent index registers hold a 16 bit base address that is used in indexed addressing modes. In this mode, an index register is used as a base to point to a region in memory from which data is to be stored or retrieved. An additional byte is included in indexed instructions to specify a displacement from this base. This displacement is specified as a two's complement signed integer. This mode of addressing greatly simplifies many types of programs, especially where tables of data are used.

4. **Interrupt Page Address Register (I).** The Z80 CPU can be operated in a mode where an indirect call to any memory location can be achieved in response to an

interrupt. The I Register is used for this purpose to store the high order 8 bits of the indirect address while the interrupting device provides the lower 8 bits of the address. This feature allows interrupt routines to be dynamically located anywhere in memory with absolute minimal access time to the routine.

Caution: The Interrupt Page Address Register is used extensively by the OASIS Operating System. Any change to this register will cause unpredictable and probably disastrous results.

5. **Memory Refresh Register (R).** The Z80 CPU contains a memory refresh counter to enable dynamic memories to be used with the same ease as static memories. This 7 bit register is automatically incremented after each instruction fetch. The data in the refresh counter is sent out on the lower portion of the address bus along with a refresh control signal while the CPU is decoding and executing the fetched instruction. This mode of refresh is totally transparent to the programmer and does not slow down the CPU operation. The programmer can load the R register for testing purposes, but this register is normally not used by the programmer.

8.3 Flags

The flag register (F and F') supplies information to the user regarding the status of the CPU at any given time. The bit positions for each flag is shown below:

```
  7   6   5   4   3   2   1   0
 ─── ─── ─── ─── ─── ─── ─── ───
  S   Z   X   H   X  P/V  N   C
```

Where:

$$
\begin{aligned}
S &= \text{Sign flag} \\
Z &= \text{Zero flag} \\
H &= \text{Half-Carry flag} \\
P/V &= \text{Parity /Overflow flag} \\
N &= \text{Add/Subract flag} \\
C &= \text{Carry flag} \\
X &= \text{Not used}
\end{aligned}
$$

Each of the two CPU flag registers contains 6 bits of status information which are set or reset by CPU operations. Four of these bits are testable (C,P/V,Z and S) for use with conditional jump, call or return instructions. Two flags are not testable (H,N) and are used for BCD arithmetic.

Carry Flag (C)

The carry flag is sometimes referred to by the symbol CY.

The carry bit is set or reset depending on the operation being performed. For ADD instructions that generate a carry and SUBTRACT instructions that generate no borrow, the carry flag will be set. The carry flag is reset by an ADD that does not generate a carry and a SUBTRACT that generates a borrow. Also the DAA instruction will set the carry flag if the conditions for making the decimal adjustment are met.

For instructions RLA, RRA, RLS and RRS, the carry bit is used as a link between the LSB and MSB for any register or memory location. During instrucitons RLCA, RLC s and SLA s, the carry contains the last value shifted out of bit 7 of any register

or memory location. During instructions RRCA, RRC s, SRA s and SRL s the carry contains the last value shifted out of bit 0 of any register or memory location.

For the logical instructions AND s, OR s and XOR s, the carry will be reset.

The carry flag can also be set (SCF) and complemented (CCF).

Add/Subtract Flag (N)

This flag is used by the decimal adjust accumulator instruction (DAA) to distinguish between ADD and SUBTRACT instructions. For all add instructions, N will be set to 0. For all subtract instructions N will be set to 1.

Parity/Overflow Flag (P/V)

This flag is set to a particular state depending on the operation being performed.

For arithmetic operations, this flag indicates an overflow condition when the result in the Accumulator is greater than the maximum possible number (+127) or is less than the minimum possible number (-128). This overflow condition can be determined by examining the sign bits of the operands.

This flag is also used with logical operations and rotate instructions to indicate the parity of the result. The number of '1' bits in a byte are counted. If the total is odd, then P is set to 0. If the total is even then P is set to 1.

When inputting a byte from an I/O device, the flag will be adjusted to indicate the parity of the data.

Zero Flag (Z)

The zero flag is set or reset if the result generated by the execution of certain instructions is a zero.

For 8 bit arithmetic and logical operations, the Z flag will be set to a 1 if the resulting byte in the Accumulator is zero. If the byte is not zero, the Z flag is reset to 0.

For compare and search instructions, the Z flag will be set to a 1 if a comparison is found between the value in the accumulator and the memory location pointed to by the contents of the register pair HL.

When testing a bit in a register or memory location, the Z flag will contain the complemented state of the indicated bit.

Sign Flag (S)

The sign flag stores the state of the most significant bit of the accumulator. When the CPU performs arithmetic operations on signed numbers, binary two's complement notation is used to represent and process numeric information. Therefore bit 7 of the accumulator indicates the sign of the result.

When inputting a byte from an I/O device to a register, the S flag will indicate either positive (S=0) or negative (S=1) data.

CHAPTER 9

INTERFACING TO OASIS

9.1 General Information

All programs to be accessed by the Command String Interpreter should be written as a "large" subroutine using a RET instruction when finished or, preferably, SC 0 (QUIT).

When a program is executed by the CSI the HL register pair will contain the address of the first character of the tokenized command string (the program name is excluded); the IX index register will contain the address of the list of delimiters used in the command string; the B register will contain the drive code that the program came from; the DE register pair will contain the starting sector number of the program on disk.

The tokenized command string is a list of the words used in the command, each word translated to upper case and filled out (or truncated) to eight characters (trailing spaces are added when necessary). The open parentheses at the beginning of an option list is considered to be a word by itself and the list is terminated by a token of a carriage return (ODH).

The list of delimiters used is merely a list of the characters that were used to separate the words in the command string. This list matches in a one-to-one relation to the tokenized command string starting with the delimiter between the program name and the first word following. When multiple characters (spaces) are used to separate two words only the first character is placed in the list of delimiters. An open parentheses is assumed to be followed by a space character even when no delimiter is actually used. The list is always terminated by a CR charcter.

For example:

>PROG NAME.TYPE:LABEL (OPT1 OPT2,OPT3

When control is passed to the program named PROG the HL and IX registers will be addressing the following character strings:

```
(HL): 4E414D45 20202020 54595045 20202020 'NAME    TYPE    '
      4C414245 4C202020 28202020 20202020 'LABEL   (       '
      4F505431 20202020 4F505432 20202020 'OPT1    OPT2    '
      4F505433 20202020 0D202020 20202020 'OPT3    .       '

(IX): 202E3A20 20202C0D                   ' .:   ,.'
```

The quotes used in the tokenized list are only for documenting the trailing spaces and are not actually in the list.

Note: The list of tokens is always terminated by a CR token.

The information provided by these two registers allows the program to access all of the data and options specified in the command.

The information provided in the B and DE registers allows the program to get any program overlay segments, if used.

9.2 Peripheral Device Drivers

The OASIS operating system contains many of the device drivers that are normally
needed. For special peripherals or applications it might be desirable for you to
write your own device driver.

A user written device driver should be written using the same format and protocalls
as the OASIS device drivers, even when you don't plan to interface OASIS to your
driver—you may want to in the future.

OASIS device drivers are written as relocatable subroutines. Each device driver
has five entry point vectors, one for each major function of the driver. The
sequence of these entry point vectors is as follows:

```
JP   ST      ; ST is entry point of device status subroutine
JP   IN      ; IN is entry point of input byte routine
JP   OUT     ; OUT is entry point of output byte routine
JP   INIT    ; INIT is entry point of device initialization
JP   UNINIT  ; UNINIT is entry point of device de-intialization
```

It is not necessary to actually use the jump instructions at these entry points but
each entry point vector must be three bytes in length.

Each of the five routines in a device driver is a subroutine that is called by
certain system calls.   These subroutine functions, requirements, and system calls
are described below.

ST   Accessed by system call 62. Input to this routine is the physical
      device number in the B register, and the UCB address in the IY
      register. The responsibility of this routine is to return the status
      of the device in the Z and C flags. This routine should not actually
      read the byte of data. If it is necessary to read the byte to
      determine the status then the byte should be saved in an input buffer
      area.

                  Z   Set = no input available
                  Z   Reset = input available
                  C   Set = output ready
                  C   Reset = output not ready

      If the device is an output only device then this routine should always
      set the Z flag, indicating that there is no data to be read in from the
      device.

IN   Accessed by system call 63. Input to this routine is the physical
      device number in the B register, and the UCB address in the IY
      register. The responsibility of this routine is to return one byte of
      input from the device in the A register. If no byte is available from
      the device this routine should wait (use SC 107 for interrupt driven
      device or SC 79 for non-interrupt driven devices). It should be the
      responsibility of the calling program to test if a byte was available
      or not.   When register A is set to zero it means that a data byte of
      zero was input, not that there was no byte available.

      If the device driver is for an output only device then this entry point
      should return immediately.

This routine (non-interrupt system) or the interrupt input routine should use system call 102 (CHARIN) for every character input to trap any escape sequence entered and to perform character translations.

OUT Accessed by system call 64. Input to this routine is the physical device number in the B register, the UCB address in the IY register, and the character to be output in register C. This routine accepts a byte of output from register C and outputs the byte to the device. An interrupt driven device might just store the byte in its buffer and return control to the caller, allowing an interrupt service routine to actually output the byte. However, this routine should handle all error conditions relating to output to the device.

INIT Accessed by system call 61. Input to this routine is the physical device number in the B register, and the UCB address in the IY register. The responsibility of this routine is to initialize the device driver and the device. The OASIS ATTACH command calls this entry point once when the device is attached to a logical name.

If the device is an interrupt driven device this routine would establish the interrupt vector using SC 103, initialize the I/O buffer, etc.

UNINIT Accessed by system call 68. Input to this routine is the physical device number in the B register, and the UCB address in the IY register. The responsibility of this routine is to un-intiialize the device. The OASIS ATTACH command calls this entry point once when the device is detached from a logical name.

If the device is an interrupt driven device this routine would probably make sure that the I/O buffer was empty, disable the interrupt for this routine using SC 103, etc.

When an interrupt service routine is entered the interrupts are disabled. The routine must enable the interrupts before an RETI instruction is executed. The interrupts may be enabled any time after entry to the routine but make sure that the routine is prepared for another interrupt to itself when the interrupts are enabled.

All routines, interrupt or otherwise, should restore the status of any registers used and not specified as part of the input or output parameters.

Multi-user note: an interrupt driven device driver must take into account the fact that the owning partition may not be the active partition when the calling interrupt occurs. It may be necessary to activate the owning partition in order to service the interrupt. The system calls 84 (ONEONLY), 85 (NOTONLY), and 86 (SETPIN) may assist you in this task.

Interfacing user written device drivers to OASIS

To interface a user written device driver to the OASIS operating system you must follow these steps:

1. Decide upon a device number. OASIS references physical device

drivers by their number. The numbers used by OASIS for the device drivers supplied may be found by listing the file SYSTEM.DEVNAMES. If your device driver is to replace the one provided with the operating system then you should use the same number as that (you may want to save the OASIS driver by renaming it).

2. After you have decided upon a number for your device driver then you must give it a name that OASIS will recognize as a device driver. All device drivers have a file name of SYSTEM and a file type of DEVnn where nn is the device number. The OASIS LINK command has an option (SYSTEM) that will cause the load image program generated to have a name of SYSTEM and a file type equal to the file name of the object file being processed.

   The device number that you use to give a name to your device driver also determines the UCB number that it uses. Keep in mind that external device numbers (device names, attach numbers, etc.) are base 1 and the internal device numbers are base 0.

3. If you are not replacing an existing device driver you will probably have to add an entry to the SYSTEM.DEVNAMES file so that the driver can be loaded by the ATTACH command by specifying a name rather than a number. The format of this file is discussed in the <u>OASIS System Reference Manual</u> in the appendix "System Files".

4. Attach your device to a logical device name using the ATTACH command. Your device driver is now available for other programs to use by referencing the logical name or number attached to the device. If the system is re-booted the driver will not be reloaded automatically unless a SYSGEN was performed while your device was ATTACHed. To reload your driver all that is necessary is that it be re-ATTACHed.

For an example listing of a peripheral device driver refer to the appendix on "Program Examples".

9.3 Disk Device Drivers

The OASIS operating system contains at least one disk device driver to handle the
disk(s) that the operating system resides on. Disk drivers to handle other types
of disk drives and controllers can be written by the end user or distibutor and can
be loaded with the ATTACH command to make multiple disk drivers on-line at one
time.

A user written disk device driver should be written using the same format and
protocalls as the OASIS disk device driver.

OASIS disk drivers are written as relocatable subroutines. Each disk driver has
four entry point vectors, one for each major function of the driver. The sequence
of these entry point vectors is as follows:

```
JP   SELECT    ; SELECT is entry point of disk select subroutine
JP   RESTORE   ; RESTORE is entry point of disk restore subroutine
JP   READ      ; READ is entry point of disk read subroutine
JP   WRITE     ; WRITE is entry point of disk write subroutine
```

It is not necessary to actually use the jump instructions at these entry points but
each entry point vector must be three bytes in length.

Each of the four routines in a disk device driver is a subroutine that is called by
certain system calls.   These subroutine functions, requirements, and system calls
are described below.

SELECT Accessed by system calls 50 and 51. Index register IY contains the
address of the UCB of the disk to be selected; register A contains the
physical drive number (0 – 7) of the drive to be selected. This
physical drive number may have to be adjusted to properly address the
drive(s) associated with this device driver.   This routine doesn't
perform any function with the disk drive or controller—it merely
specifies which drive subsequent operations are to be performed on.

RESTORE Accessed by system calls 50 and 51. Index register IY contains the
address of the UCB of the disk to be restored. This routine's function
is to "recalibrate" the drive—position the heads on track 0 with the
assumption that it is unknown where the heads are currently located at.

It is probable that this routine would perform no direct function other
than setting a switch indicating that the next read or write operation
to this drive is to first perform the restore operation.

READ Accessed by system call 50. Index register IY contains the address of
the UCB of the disk to be read from; register A contains the number of
consecutive sectors to be read; register B contains the head number;
register C contains the sector number; register pair DE contains the
cylinder number; register pair HL contains the address in memory that
the information is to be read into. All values are base zero.

This routine should perform the physical I/O required to read the
specified sectors into the memory area indicated. Sectors are always
considered 256 bytes long, independant of the actual sector size of the
disk. It is the responsibility of this routine to adjust the number
and location of the sectors desired to correspond with the physical

sector size of the disk, if different.

This routine should not perform any error recovery procedures. If an error occurs the operation should be stopped, the pertinent registers adjusted to reflect the location of the error, the A register should be set to reflect the type of error, and the Z flag should be reset to indicate that an error occured. Any retry or recovery operations will be handled by system software outside of this device driver.

When the disk read is succesfull the pertinent registers should be adjusted to point to the sector following that which was just read, the A register should be set to zero and the Z flag should be set.

This routine, as called by the OASIS system call, never asks to read consecutive sectors that cross a cylinder or head boundary.

WRITE Accessed by system call 51. Index register IY contains the address of the UCB of the disk to be written to; register A contains the number of consecutive sectors to be written; register B contains the head number; register C contains the sector number; register pair DE contains the cylinder number; register pair HL contains the address in memory that the information is to be written from. All values are base zero.

This routine should perform the physical I/O required to write the specified sectors from the memory area indicated. Sectors are always considered 256 bytes long, independant of the actual sector size of the disk. It is the responsibility of this routine to adjust the number and location of the sectors desired to correspond with the physical sector size of the disk, if different.

This routine should not perform any error recovery procedures. If an error occurs the operation should be stopped, the pertinent registers adjusted to reflect the location of the error, the A register should be set to reflect the type of error, and the Z flag should be reset to indicate that an error occured. Any retry or recovery operations will be handled by system software outside of this device driver.

When the disk write is succesfull the pertinent registers should be adjusted to point to the sector following that which was just written, the A register should be set to zero and the Z flag should be set.

This routine, as called by the OASIS system call, never asks to write consecutive sectors that cross a cylinder or head boundary.

Note that there is no initialization entry point. It is the responsibility of the select routine to check if the device needs initialization (maybe a DC of zero is coded--when routine is first loaded that location will still be zero--that the select routine sets to a one after the device is initialized).

Disk error codes

The following standard error codes should be returned by a disk device driver when an error occurs:

         1 - Disk not ready
         2 - Disk write protected

```
3 - Disk not initialized -- possibly a time out or wrong density dete
4 - Data CRC error
5 - Invalid parameters -- can't happen
6 - Disk label changed -- or disk changed or door opened
7 - Sector not found
8 - Track not found
9 - Address CRC error
```

Interfacing user written disk device drivers to OASIS

To interface a user written disk device driver to the OASIS operating system you must follow these steps:

1. Decide upon a device number--OASIS references the disk drivers by their number. The numbers used by OASIS for disk devices are in the range of 1 through 8. however, do not use a number associated with the disk driver included in the OASIS NUCLEUS (generally 1 thru 4).

2. After you have decided upon a number for your driver then you must give it a name that OASIS will recognize as a device driver. All device drivers have a file name of SYSTEM and a file type of DEVnn where nn is the device number. The OASIS LINK command has an option (SYSTEM) that will cause the load image program generated to have a name of SYSTEM and a file type equal to the file name of the object file being linked.

3. Add the device name of your driver to the SYSTEM.DEVNAMES file. A disk driver may have multiple entries in this file to reflect the multiple disks that it controls. A record in this file for a disk device driver has the following format:

   <logical name> <device number> D <other numbers shared>

   The <logical name> is a two to eight character name that you will use when you ATTACH a drive code to a disk. It is best if the name also identifies the disk drive in some meaningful manner. For example, a disk driver for a XYZ hard disk drive should probably be named XYZ1 or XYZ2, etc.

   The <device number> is the number that you decided upon in step 1.

   <Other numbers shared> is a list of device numbers that are controlled by this one disk driver. This is best explained by an example. Say that you have written a disk driver for a disk controller that interfaces to four drives, numbered 5, 6, 7, and 8. You would probably use the number 5 for the disk driver number and the name XYZ5 for its logical name. The entries in the SYSTEM.DEVNAMES file would then look like this:

   ```
   XYZ5 5 D 5 6 7 8
   XYZ6 6 D 5 6 7 8
   XYZ7 7 D 5 6 7 8
   XYZ8 8 D 5 6 7 8
   ```

   The above example indicates that device numbers 5, 6, 7, and 8 are all controlled by device driver number 5. There will only be one copy of the driver loaded into memory for all four drives that may be attached

to it.

4. Attach your device to a logical name using the ATTACH command, similar to the way you attach logical names to the OASIS supplied disk driver--A, B, etc.

Note that there is no un-initialize entry point in a disk device driver. This means that the device driver will not be unloaded from memory when all disks are detached from it. Once this auxillary disk driver is loaded into memory the only way to recover the memory used by the driver is to re-boot the system (assuming that it was not sysgened).

For an example listing of a disk device driver refer to the appendix on "Program Examples".

9.4 Terminal Class Code Drivers

The OASIS operating system provides a uniform interface to the console terminal cursor controls. Because most terminal manufacturers use a slightly different and unique coding sequence to control the actions of the terminal it is cumbersome for an application program to be coded such that it is capable of communicating with different types of terminals. In OASIS an application program is coded using an internally defined standard (another, unique standard) for cursor control. The characters used in the standard are described in the <u>OASIS System Reference Manual</u>, appendix "Terminal Class Codes".

The translation between the OASIS internal standard and the control codes used by the actual terminal is performed in a small subroutine that interfaces between the operating system and the device driver. Several different terminal class code subroutines are supplied with the operating system.

If your terminal uses a set of cursor control codes that is not handled by one of the class code subroutines supplied you will have to write your own or not use cursor control. However it is very easy to write your own subroutine to handle your particular terminal due to the macro definitions supplied in the file CLASS.MACLIB.

To write your own terminal class code subroutine create an assembly program with the name CLASSnn where the nn is the class code number you wish to use. Use the MACLIB pseudo op-code to get the macro definitions in CLASS.MACLIB file into your program. The three macros you will be using are described below:

INIT   Performs the subroutine initialization required of a class code subroutine. This macro reference must be the first code in your program. This macro routine has all the code in it to handle any character translations undefined with the following macro (DEFINE) and handles all of the standard, OASIS input escape sequences (see <u>OASIS System Reference Manual</u>, chapter "System Control Keys").

        This macro also allows you to specify up to eight characters that are to be translated and the values that they are translated to. To use this feature merely specify the translation list in the operand field (see example six in the appendix of program examples in this manual).

DEFINE Defines the relationship between the internal codes and the codes used by the terminal. The first argument to this macro is the name of the internal code such as CLEAR, HOME, EOL, etc. Subsequent arguments to this macro are the characters to be sent to the terminal to perform the desired function. All of the ASCII control characters are defined with the appropriate value so that you may use names such as ESC, DC1, etc.

        The DEFINE macro reference is used as many times as is necessary to define the functions of the terminal. No special sequence is required and any undefined functions may be omitted.

        Any function that your terminal is not capable of performing (i.e., BON) should be defined with no output list (see example six again).

        Any function that your terminal is not capable of performing but can be simulated by the operating system (EOL and EOS only) should not be defined in your program. When this is done the operating system will simulate the

function by outputting spaces and repostioning the cursor to the orginal location.

Any function that should be followed with the ATTACHed form feed delay should have its definition end with the argument 8CH which will be interpreted by the macro as indicating the form feed delay is to be added after the output of the function.

DCA   Indicates the start of the cursor address coding routine. The DCA macro name may be followed by a numeric operand specifying the pre-defined class code number that uses the exact same cursor control algorithm.

The DCA macro call is followed by the routine that will output to the terminal the proper codes to perform the addressing of the cursor. Upon entry to the routine the following registers will be defined:

    A Control character to translate (not used by your cursor address
       routine)
    B Device number of the console terminal (always a 9)
    C Same as register A
    H Column number to position to, base 0
    L Line number to position to, base 0

When your routine outputs the codes to the terminal you must use system call 64. (Using system call 2, 5, 7, or 52 might cause an infinite loop.)

After you have output the proper codes to the terminal clear the carry flag and perform a return. If the carry flag is set when you return it will indicate to OASIS that the function could not be performed and that OASIS is to try to simulate it with software. This may be done by performing a HOME followed by line feeds and non-destructive cursor advances.

Example six in the appendix of program examples lists a terminal class code subroutine for the SOROC IQ 120 terminal.

9.5 System Start-up Program

The OASIS operating system provides the capability of loading and executing a program (machine language) automatically whenever the operating system is "booted". The program that is loaded must be named SYSTEM.STARTUP, must reside on the system disk, be owned by the system (public) account, and be relocatable.

The SYSTEM.STARTUP program may do "anything" that you may require a this type of program to do (i.e., automatically interface to calendar circuitry). This program is loaded after memory is sized by the operating system but before any device drivers are loaded. The program is loaded into current high memory and is called by the operating system. The program should do whatever is necessary at this time and exit by executing a RET instruction. Before exiting the carry flag should be set if the program is to remain in memory or reset if the program is to be overlaid by the system.

9.6 USR Programs

A USR program is an assembler language subroutine accessed by a BASIC language program through a special function call. Only one parameter is passed to the subroutine and only one parameter may be returned to the BASIC program. The input and output parameter types must be the same: 16 bit numeric or a character string.

The USR routine must be a relocatable program

A USR subroutine may have an unlimited number of entry points but each entry point may only perform processing on one type of parameter. This is due to the fact that there is no way of detecting what the parameter type is. A USR routine may perform processing independent of the input and/or output parameter.

The BASIC program accesses the various entry points of a USR routine by specifying the address of the entry point relative to the load address of the subroutine. It is best to make the entry points simple, such as: 0, 3, 6, etc. To do this jump vectors should be used, similar to the device drivers discussed above. This not only makes the entry point addressing simple but also allows for modifications to the program without requiring changes to the entry point addressing in the BASIC program.

A USR routine may use, without restoring, any and all of the registers. BASIC makes no assumptions regarding the integrity of the registers (with the exception of the HL register pair and the SP!). The USR routine, in turn, should make no assumptions about the integrity of the registers (except the HL register pair and the SP!) as BASIC may use any and all of the registers between calls to your USR.

A numeric parameter is passed to a USR routine via the HL register pair. If a parameter is to be returned to the BASIC program it must be placed in the HL register pair. This implies a limit of 16 bit numbers.

A string parameter is passed to a USR routine via the BASIC string accumulator. The string accumulator start address is in the HL register. The string accumulator is a 256 byte buffer used by BASIC for all string manipulations. The first byte of this buffer is a count of the number of characters following. The string parameter returned to the BASIC program may be in the string accumulator or in an internal buffer (up to 256 bytes). In either case the HL register pair must address the first byte of the buffer used when the return is made to BASIC and this first byte must be the count of the characters in the buffer. If the string accumulator is used care must be taken to insure that the 256 byte limit is not exceeded because volatile information precedes and follows this buffer.

When LINKing your USR routine be sure to use the USR option as it will cause the file type of your load module to be BASICUSR, a requirement of the BASIC interpreter.

For example listings of USR routines refer to the appendix on Program Examples.

9.7 BASIC Fields

It is not advised that you write programs that access the variables in a BASIC program directly. This is primarily due to the fact that the variable storage area of BASIC is dynamic, even its base address. You should use the USR feature of BASIC to pass the field to your assembly program. However, it may be necessary for you to know the format of variables maintained by BASIC, internally to BASIC and/or externally in a file.

Format of BASIC Variables

BASIC variables are formatted the same whether maintained internally or on a disk file. However, file fields have an extra byte of information preceding the content of the field. This extra byte is a code indicating that the field is a string, integer, or floating point field.

String Fields

String fields are simplistic in format: the code is a binary 6 followed by the length of the string (range of 0 to 255) followed by the individual characters of the string.

Integer Fields

An integer field has a code of a binary 4 followed by the 16 bit signed binary number, most significant byte first.

Floating Point Fields

A floating point field has a code of a binary 3 followed by a one byte characteristic in excess 128 format (characteristic in two's complement plus 128), followed by a nibble (four bits) specifying the sign of the mantissa, followed by 52 bits of the normalized mantissa in BCD.

A code field of 0 indicates the end of record.

Examples:

Field	Type	Contents
06055061676520	S	Page
041234	I	+4660
04FEA7	I	−345
038202345678000000	F	+.2345678E+2
037E81234567890123	F	−.1234567890123E−2

SYSTEM CALL SUMMARY

```
 0 QUIT        Reload the Command String Interpreter - restart
 1 KEYIN       Accept a line of input from the console keyboard
 2 DISPLAY     Display characters on console output device
 3 CONST       Get status of console input device
 4 CONIN       Accept one character from the console input device
 5 CONOUT      Display one character on console output device
 6 SYSIN       Accept one character from console, ignoring ESC,O and ESC,P
 7 SYSOUT      Display one character on console, ignoring ESC,O and ESC,P
 8 PRTOUT      Output one character to PRINTER1 device
 9 MOUNT       Allow change of diskette on a specified drive
10 RD1         Read one sector from disk
11 WR1         Write one sector to disk
12 IPL         Perform initial program load
13 WRFDIR      Create new file directory entry
14 HEXI        Convert hexadecimal number to 16 bit binary
15 DECI        Convert decimal number to 16 bit binary
16 HEXO        Convert 8 bit value to hexadecimal characters
17 DECO        Convert 16 bit unsigned value to decimal string
18 CRLF        Display carriage return, line feed on console
19 MSEC        Wait specified number of milliseconds
20 LOOKUP      Locate directory entry of file
21 GETUCB      Get address of UCB
22 LOAD        Load a program
23 PRINT       Output a line to PRINTER1 device
24 ASSIGN      Store ACB
25 ADRV        Convert logical drive code to drive number
26 BDRV        Convert drive number to logical drive code
27 ALLOC       Allocate space for file on disk
28 DEALL       Deallocate space for file on disk
29 ERASE       Erase logical file from a disk
30 FETCH       Load program in memory, execute, restart
31 RENAME      Rename a logical disk file
32 OPEN        Open a logical file
33 CLOSE       Close a logical file
34 RDSEQ       Read a logical record from a sequential file
35 WRSEQ       Write a logical record to a sequential file
36 GETDATE     Get formatted date
37 GETTIME     Get formatted time
38 DIV         16 bit, binary, unsigned divide
39 MUL         16 bit, binary, unsigned, integer multiply
40 RDDIR       Read logical record from a direct disk file
41 WRDIR       Write a logical record to a direct disk file
42 NUMBER      Convert numeric string to 16 bit value
43 RDIX        Read a logical record from an indexed disk file
44 RDNIX       Read the next logical record from an indexed disk file
45 WRIX        Write a logical record to an indexed disk file
46 DATEPACK    Pack system date and time into 24 bits
47 LABEL       Find disk with specified label
48 GETSCR      Get base address of user System Communication Region
49 WAIT        Wait for operator to release current console page
50 RD          Read multiple sectors of a disk
51 WR          Write multiple sectors to a disk
52 SYSDISP     Display characters on console, ignoring ESC,O and ESC,P
53 TIMER       Set up for a clocked interrupt
```

54	EXCMD	Execute a command
55	GETMEM	Get current high memory
56	PUTMEM	Set new high memory
57	PUTQET	Change routine for service of System Cancel-key
58	TSTDEV	Test device attachment
59	GETPL	Get console/printer page and line parameters
60	DELIX	Delete a record from an indexed file
61	DEVINIT	Initialize a device driver
62	DEVST	Get status of device driver
63	DEVIN	Get input from device driver
64	DEVOUT	Put output to device driver
66	GETLAB	Get label of specified disk drive
67	PUTDEV	Store device driver address
68	DEVUNINIT	Uninitialize a device driver
69	TSTESCC	Test if Program Cancel-key entered
70	EXCMDR	Execute a program and return
71	BUFFI	Get character from buffer
72	BUFFO	Put character to buffer
73	PUTCON	Get/set console control byte
74	PUTDET	Set address of disk error trap
75	NEWSYS	Change system disk
76	DELAY	Delay processing for specified period of time
77	GETACB	Point to Assign Control Block
78	CONESC	Perform System Control-key function
79	SNU	Select next user
80	GETBASE	Get base address of NUCLEUS
81	GETMFG	Get system manufacturer number
82	GETPIN	Get your user partition id number
83	UNLOCK	Unlock record of file
84	ONEONLY	Set flag for exclusive use of resource
85	NOTONLY	Release exclusive use of resource
86	SETPIN	Activate specific partition
87	GETLUB	Get Logical Unit Block table base address
88	MSG	Send message to another user's console
89	EXCLUSIVE	Get exclusive control of key resources
90	UNEXCLUSIVE	Release exclusive control of key resources
91	GETWORK	Get user System Communication work area address
92	GETPRIVLEV	Get current privilege level
93	COMPARE	Perform string comparison
94	RDBIN	Get binary data stream from file
95	WRBIN	Put binary data stream to file
96	ERRDIS	Display error message
97	ERRQUI	Display error message and quit
98	OVERLAY	Load overlay of program
99	CONDALL	Conditional allocation
100	DISPATCH	Perform table lookup
101	GETUSER	Get current user account number
102	CHARIN	Console input character analysis
103	PUTVECT	Point vector to interrupt service routine
104	GETBYTE	Get bytes from another partition
105	PUTBYTE	Put bytes to another partition
106	DATEOUT	Convert BCD date to standard format
107	WAITINT	Deactivate partition until interrupt occurs
108	FINDPGM	Get address of re-entrant program

MACRO

SC	Inputs	Outputs
0 QUIT		no return
1 KEYIN	DE=addr,B=len	A=len,DE=next
2 DISPLAY	DE=addr	DE=next
3 CONST		
4 CONIN		A=char
5 CONOUT	C=char	
6 SYSIN		A=char
7 SYSOUT	C=char	
8 PRTOUT	C=char	
9 MOUNT	B=drive	
10 RD1	B=drive,DE=sect,HL=addr	
11 WR1	B=drive,DE=sect,HL=addr	
12 IPL	B=drive,DE=sect	
13 WRFDIR	B=drive,DE=DEB	
14 HEXI	DE=addr	HL=num,DE=next
15 DECI	DE=addr	HL=num,DE=next
16 HEXO	B=number,DE=addr	DE=next
17 DECO	DE=addr,HL=number	DE=next
18 CRLF		
19 MSEC	A=msec count	
20 LOOKUP	DE=DCB,HL=buffer addr	Z,DE=sec,HL=addr
21 GETUCB	B=device	HL=UCB
22 LOAD	DE=DCB,HL=addr	
23 PRINT	DE=addr	DE=next
24 ASSIGN	B=ACB #,DE=ACB	
25 ADRV	B=ASCII drive #	A=bin drive
26 BDRV	B=bin drive #	A=ASCII drive
27 ALLOC	B=drive,DE=block count	HL=sect
28 DEALL	B=drive,DE=block count,HL=sect	
29 ERASE	DE=DCB	
30 FETCH	B=drive,DE=DEB	no return
31 RENAME	DE=old DCB,HL=new DCB	
32 OPEN	DE=FCB	
33 CLOSE	DE=FCB	
34 RDSEQ	DE=FCB,HL=addr	
35 WRSEQ	DE=FCB,HL=addr	
36 GETDATE	DE=addr	DE=next
37 GETTIME	DE=addr	DE=next
38 DIV	DE=divisor,HL=dividend	HL=quotient
39 MUL	DE=multiplier,HL=muliplicand	HL=product
40 RDDIR	BC=key,DE=FCB,HL=addr	
41 WRDIR	BC=key,DE=FCB,HL=addr	
42 NUMBER	DE=addr	DE=next,HL=number
43 RDIX	BC=key addr,DE=FCB,HL=addr	
44 RDNIX	BC=key addr,DE=FCB,HL=addr	
45 WRIX	BC=key addr,DE=FCB,HL=addr	
46 DATEPACK	DE=addr	DE=next
47 LABEL	DE=label addr	A=drive
48 GETSCR		IY=SCR
49 WAIT		
50 RD	B=drive,C=count,DE=sect,HL=addr	
51 WR	B=drive,C=count,DE=sect,HL=addr	
52 SYSDISP	DE=addr	DE=next
53 TIMER	DE=count,HL=TEB	

MACRO   – 141 –

SC		Inputs	Outputs
54	EXCMD	DE=addr	no return
55	GETMEM		HL=addr
56	PUTMEM	HL=addr	
57	PUTQET	HL=addr	
58	TSTDEV	B=device #	NZ,A=code
59	GETPL	B=dev	B=line,C=page
60	DELIX	BC=key,DE=FCB,HL=rec	
61	DEVINIT	B=dev	
62	DEVST	B=dev	NZ=in rdy,CF=out rdy
63	DEVIN	B=dev	A=char
64	DEVOUT	B=dev,C=char	
66	GETLAB	B=dev,DE=addr	
67	PUTDEV	B=phy dev,DE=addr	
68	DEVUNINIT	B=dev	
69	TSTESCC		Z=no ESC,C
70	EXCMDR	DE=cmd,HL=ret addr	All regs modified
71	BUFFI	HL=buffer	A=char
72	BUFFO	A=char,HL=buffer addr	
73	PUTCON	B=sets,C=clears	A=new mask
74	PUTDET	HL=addr	
75	NEWSYS	B=new phy S	
76	DELAY	A=time code	A=0
77	GETACB	B=ACB #	HL=ACB addr
78	CONESC	A=2nd esc char	
79	SNU		
80	GETBASE		IY=base
81	GETMFG		A=MFG
82	GETPIN		A=PIN
83	UNLOCK	DE=FCB	
84	ONEONLY	HL=resource	
85	NOTONLY	HL=resource	
86	SETPIN	A=PIN,HL=addr	
87	GETLUB		IY=LUB
88	MSG	A=PIN,DE=addr	DE=next
89	EXCLUSIVE		
90	UNEXCLUSIVE		
91	GETWORK		HL=addr
92	GETPRIVLEV		A=priv
93	COMPARE	BC=len,DE=1st,HL=2nd	
94	RDBIN	B=len,DE=FCB,HL=addr	
95	WRBIN	B=len,DE=FCB,HL=addr	
96	ERRDIS	DE=parms,HL=msg #	
97	ERRQUI	A=RC,DE=parm,HL=msg #	no return
98	OVERLAY	B=drive,DE=base,HL=addr	
99	CONDALL	B=drive,DE=min,HL=max	DE=actual,HL=addr
100	DISPATCH	DE=string,HL=table	Z,HL=arg from table
101	GETUSER		A=user
102	CHARIN	A=char	NC,A=new char
103	PUTVECT	A=num,DE=addr	
104	GETBYTE	BC=len,DE=my,HL=his	DI, like LDIR
105	PUTBYTE	BC=len,DE=his,HL=my	DI, like LDIR
106	DATEOUT	C=mm,H=dd,L=yy,DE=addr	DE=next
107	WAITINT		
108	FINDPGM	DE=name	Z,HL=addr

Operator Cancelled

*** Duplicate Label —or— Phase Error ***

Indicates that the address of the instruction has a different value between pass one and pass two. Usually indicates that the label is defined more than once.

nn errors in program

Indicates the total number of detected errors in the program.

*** Invalid Expression ***

*** Label Error ***

Indicates that an invalid character was used in a label. Labels must use only the alphabetic characters and the dollar sign character. Local labels must start with a period character. Macro local labels must start with the at (@) character.

*** Label Required ***

The label field is blank on a directive that requires a label. These directives include: ABS, COM, ENTER, EQU, REL, and VALUE.

*** Macro Definition Error ***

Indicates a construction or syntax error in a macro definition. Usually results from a missing ENDM directive or an attempt to define a macro within a macro definition.

*** Nested too Deep ***

Indicates that an attempt was made to push more than eight IF, ORG, USING, LIST, or macro calls onto their respective nesting stacks or an attempt was made to pop one of the above from their stack when no argument was on their stack.

*** Overflow ***

Indicates that more bits are required to contain value than are permitted in expression type. For example a relative jump of more than +127 or −128.

*** Relocation Error ***

Indicates that an expression containing relocatable symbols is in error. Usually the error is one of the following: a difference between two relocatable symbols of different PABs; the sum of two relocatable symbols; the product of two relocatable symbols; the quotient of two relocatable symbols; the product or quotient of a relocatable symbol and an absolute symbol; a valid relocatable expression used in an operand that may only have eight or seven bits of precision.

*** Segment not Found ***

Indicates that the file description of a COPY or LINK directive can not be found in any of the attached directories.

*** Statement Syntax Error ***

Indicates that the operand is invalid for the op-code or that there is a missing delimiter in the operand.

Symbol Table Overflow

The size of the symbol table is determined by the amount of available memory during the assembly process. There are several things that can be done to remove this error: add more memory; unload the system Debugger if loaded; unload unused device drivers; remove unreferenced symbols from the program; reduce the use of local labels; use shorter symbol names; segment the program to allow for smaller assemblies (make the LINK program join them together).

*** Undefined Operation ***

Indicates an invalid op-code or directive was used or a reference is made to an undefined macro. Specifically the Assembler searches its op-code table, its directive table, internally defined macros, external macro files. When the op-code field does not match any of these it is determined to be an undefined operation.

*** Undefined Symbol ***

Indicates a reference was made to a symbol not defined.

APPENDIX C

CONTROL BLOCK DEFINITIONS

The following short diagram illustrates the bidirectional communication linkages that are followed when a program (user or system) requests input or output to a logical file. Non file input or output is similar except that the program links directly to the Logical Unit Block.

```
                        Program
                         /|\
                          |
                         \|/
              File Control Block (FCB)
                         /|\
                          |
                         \|/
             Assign Control Block (ACB)
                         /|\
                          |
                         \|/
             Logical Unit Block (LUB)
                         /|\
                          |
                         \|/
              Unit Control Block (UCB)
                         /|\
                          |
                         \|/
              Pysical Device Driver
                         /|\
                          |
                         \|/
                        I/O Port
                         /|\
                          |
                         \|/
                  Peripheral Device
```
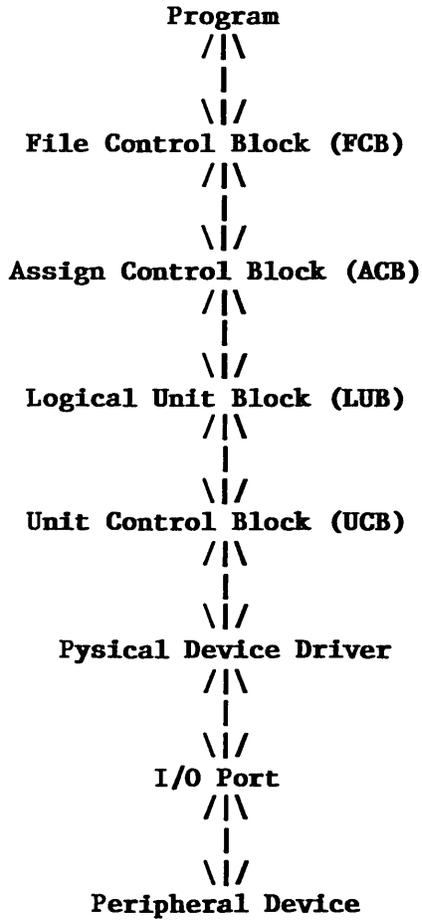
Unit Control Block (UCB)

Devices (8 - 31)

```
===================================================================
Byte   Description
-------------------------------------------------------------------
00     Driver address
02     Line length
03     Page length
04     Class code
05     Code Baud           Code Baud
       ---- ----           ---- ----
       1    75             8    2400
       2    110            9    4800
       3    134.5          10   7200
       4    150            11   9600
       5    300            12   19200
       6    600            13   38400
       7    1200
05     Bit 7 on indicates CONIN device
       Bit 6 on indicates ESC lead in received (used by class cod
06     CR/LF delay
07     FF/EOS/EOL delay
08     Bit   On                  Off
       ---   ----------------    --------------------
       7     Parity enable       No parity
       6     Odd parity          Even parity
       5     8 bit data          7 bit data
       4     Syncronous          Asyncronous
       3     Page parity         No page parity
       2     Auto LF             No auto LF
       1     No FF               FF ability
       0     SDLC
09     Overflow count
0A     Current line
0B     Reserved
0C     Reserved
0D     Speed delay
0E     Device driver length
10     Translate routine length
12     Input buffer address
14     Output buffer address
16     Translate routine address
18     Video base address/ i-o address base
1A     Video cursor address
1C     Bit   Output-busy         Input-busy
       ----  ------------        ----------
       0     -CTS                -DTR
       1     XOFF/XON            XOFF/XON
       2     ETX/ACK             ETX/ACK
       3     -DTR
1D     Work area (2 bytes)
1F     Owner pin
```

Unit Control Block (UCB)

Disk Devices (0 - 7)

```
====================================================================
Byte  Description
--------------------------------------------------------------------
00    Driver address
02    Volume id label
0A    Number of heads
0B    Number of cylinders
0D    Number of sectors
0E    Directory size
10    Clusters available (blocks)
12    Reserved
13    WP/IBM/Additional -

      Bit  Meaning
      ---- ----------------
      7    Write protected
      6    Track 0 single density
      5-0  Number additional map sectors

14    Current cylinder
16    Head load delay
17    Step time delay
18    Settle time delay
19    Work area (6 bytes)
1F    Owner (0FFH=public)
```

Timer Event Block (TEB)

```
====================================================================
Byte Description
--------------------------------------------------------------------
00-01 Number of ticks remaining
02    Reserved
03    Partition id number of owner
04-05 Forward link to TEB (0 = node)
```

File Control Block (FCB)

```
========================================================================
Byte  Description
------------------------------------------------------------------------
00    ACB number (0 - 20)
01    File format and I/O mode

      Bit   On
      ---   -----------------
      7     Sequential
      6     Direct
      5     Indexed
      4     Input
      3     Output
      2     Append - (sequential format only)
      1     Reserved
      0     File lock

      Both bits 3 and 4 on means update with record lock.

02-03 Address of I/O buffer
        (same length as sector size)
```

Assign Control Block (ACB)

```
========================================================================
Byte  Description
------------------------------------------------------------------------
00    Drive code (0-7, 255=all)
01-08 File name (trailing blanks, if necessary)
09-10 File type (trailing blanks, if necessary)
11    Logical device number, base 0

        0 = not assigned
        1 = any disk
        8 = console
      255 = dummy

12-1F System defined
```

Directory Control Block (DCB)

```
========================================================================
Byte  Description
------------------------------------------------------------------------
00    Drive code (0-7, 255=all)
01-08 File name (trailing blanks, if necessary)
09-10 File type (trailing blanks, if necessary)
```

Directory Entry Block (DEB)

```
=====================================================================
Byte   Contents
---------------------------------------------------------------------
00     File format:
       FF   = Deleted
       00   = Empty - never used
       x1   = Relocatable
       x2   = Absolute
       x4   = Sequential
       x8   = Direct
       10   = Indexed
01-08  File name.
09-10  File type.
11-12  Record count.
13-14  Block count.
15-16  Address of 1st sector.
17-18  Variable by file format:
       I    = Key length - 1 and record length - 1
       S    = Record length of longest record
       D    = Allocated record length
       A,R  = Record length (sector length)
19-1B  Date and time of update.
1C     Owner Id.
1D     Shared from owner Id.
1E-1F  Variable by file format:
       I    = Allocated file size
       S    = Disk address of last sector in file
       D,R  = Zero
       A    = Origin address
```

Partition Control Block (PCB)

```
=====================================================================
Byte   Description
---------------------------------------------------------------------
00-01  SCR address
02     Bank and activity flags:

       Bit   On
       ---   ----------------
        7    Not active
        6    Waiting for disk
        5    Waiting for interrupt
        4    Waiting for resource
       3-0   Bank number
```

Bank Control Block (BCB)

```
========================================================================
Byte   Contents
------------------------------------------------------------------------
00-01  Nucleus origin (global bank)
02-03  Nucleus end + 1
04-05  Bank 0 low address
06-07  Bank 0 high address + 1
08-09  Bank 1 low address
0A-0B  Bank 1 high address + 1
         .
         .
         .
40-41  Bank 15 low address
42-43  Bank 15 high address + 1
```

Re-entrant Program Block (RPB)

```
========================================================================
Byte   Contents
------------------------------------------------------------------------
00-07  Program 1 name, eight character, padded if necessary
08-09  Program 1 start address
0A-11  Program 2 name, eight character, padded if necessary
12-13  Program 2 start address
         .
         .
         .
46-4D  Program 8 name, eight character, padded if necessary
4E-4F  Program 8 start address
```

Sector Lock Table (SLT)

```
========================================================================
Byte   Contents
------------------------------------------------------------------------
```

The following six byte entry is repeated as often as necessary.

```
00     Drive and ACB number (0FF indicates end of table)

       Bit  Meaning
       ---  --------------------
       7-3  ACB number
       2-0  Drive number

01-02  Sector start address
03-04  Sector end address
05     Owner Partition id number (PIN)
```

File Lock Table (FLT)

```
==================================================================
Byte  Contents
------------------------------------------------------------------
```

      The following four byte entry is repeated as often as nece

00   Drive number (OFF indicates end of table)
01-02 Sector number (from DEB)
03   Owner Partition id number (PIN))

PROGRAMMING EXAMPLES

This appendix contains a listing of several working programs. The first example is the listing of the program VERIFY, which is an early version of the VERIFY program included with the operating system.

The second example is a USR subroutine to be used by a BASIC program. This routine is not provided as part of the operating system but you might wish to add it as it is a useful routine to have available. The basic function of the routine is to translate a string of characters to uppercase.

Example three is a sophisticated serial device driver (SIO). This driver is probably more lengthy than the serial driver on your system (although it may actually be the driver on your system) because it is designed to interface to a complex, programmable, serial I/O integrated circuit. Included in the driver is all the code necessary to analyze and support the various options that may be specified with the ATTACH command and the various primary devices that a serial device may be used as (CONSOLE, PRINTER, other).

The forth example is a simple, parallel printer device driver. This driver performs the minimum tasks necessary to drive a parallel printer output port.

The fifth example is a disk driver for a hard disk drive. The particular drive and controller that this driver was programmed for is relatively intelligent (performed a lot of the detail work itself), and included direct memory access (DMA) capability.

Example six is a terminal class code control character translator (SYSTEM.CLASS4:S file). The example given is for a SOROC IQ 120 terminal.

Please note the abundant use of comments in the examples. It is a good practice to use a lot of comments, especially in assembly language coding--no speed or memory usage degradation occurs and you, or another programmer, will appreciate them at a later date.

Also note that most of the labels are coded on separate lines. This also facilitates program maintenance.

Example 1 — VERIFY Command

Addr Obj-Code Line *** Source Statement ***

```
                      1 CODE:     REL
0000                  2 VERIFY:
0000 E5               3            PUSH   HL          ; Save token loc
0001 113F01           4            LD     DE,HELPL     ; Point literal
0004 0609             5            LD     B,9          ; Len
0006                  6 TESTH:
0006 1A               7            LD     A,(DE)       ; Get mask byte
0007 BE               8            CP     (HL)         ; Compare
0008 200C             9            JR     NZ,NOHELP    ; BRIF not HELP
000A 13              10            INC    DE           ; Bump
000B 23              11            INC    HL
000C 10F8            12            DJNZ   TESTH        ; Loop
000E 114801          13            LD     DE,HELPM     ; Message
0011 CF02            14            SC     2            ; Display
0013 AF              15            XOR    A            ; RC = 0
0014 CF00            16            SC     0            ; Quit
0016                 17 NOHELP:
0016 E1              18            POP    HL           ; Get loc back
0017 7E              19            LD     A,(HL)       ; Get drive
0018 47              20            LD     B,A          ; Move
0019 CF19            21            SC     25
001B 320501          22            LD     (DRIVE),A    ; Save drive
001E 3009            23            JR     NC,OKFD      ; BRIF ok
0020 11E600          24            LD     DE,MSG1      ; Else display err msg
0023 CF02            25            SC     2            ;  And quit - RC = 16
0025 3E10            26            LD     A,16
0027 CF00            27            SC     0
0029                 28 OKFD:
0029 21B800          29            LD     HL,QUIT      ; Set System cancel-key
002C CF39            30            SC     57           ;  clean up
002E 21C100          31            LD     HL,ERR       ; Set disk error routine
0031 CF4A            32            SC     74
0033 3A0501          33            LD     A,(DRIVE)    ; Get drive code
0036 47              34            LD     B,A          ; Move
0037 CF09            35            SC     9            ; Mount drive
0039 CF15            36            SC     21           ; Get UCB
003B E5              37            PUSH   HL           ; Save it
003C DDE1            38            POP    IX           ; Into IX
003E DD7E08          39            LD     A,(IX+8)     ; Get msb
0041 07              40            RLCA                ; Exchange bits 7-4
0042 07              41            RLCA                ;  with bits 3-0
0043 07              42            RLCA
0044 07              43            RLCA
0045 E60F            44            AND    OFH          ; Number surfaces
0047 32FD00          45            LD     (SURF),A     ; Save
004A DD6E09          46            LD     L,(IX+9)     ; Get tracks/surface
004D DD7E08          47            LD     A,(IX+8)     ;  12 bit value
0050 E60F            48            AND    OFH          ; Mask
0052 67              49            LD     H,A
0053 220101          50            LD     (TRACKS),HL  ; Save number of track
0056 DD7E0A          51            LD     A,(IX+10)    ; Number sectors/track
0059 6F              52            LD     L,A          ; Move to HL
```

```
005A 2600        53          LD    H,0
005C 3AFD00      54          LD    A,(SURF)    ; Get number of surfac
005F 5F          55          LD    E,A         ; Move to DE
0060 1600        56          LD    D,0
0062 CF27        57          SC    39          ; Multiply
0064 22FF00      58          LD    (CYL),HL    ; Store total sect/cyl
0067 ED5B0101    59          LD    DE,(TRACKS) ; Get tracks/cyl
006B CF27        60          SC    39          ; Compute sect/drive
006D 22F900      61          LD    (TOTLEN),HL ; Total length
0070 210000      62          LD    HL,0
0073 220301      63          LD    (TRACK),HL  ; Clear track/cyl
0076 22FB00      64          LD    (SECT),HL   ;  and sect/track
0079 CF12        65          SC    18          ; CR/LF on console
007B            66 LOOP:
007B 0E0D        67          LD    C,13        ; Display CR only
007D CF05        68          SC    5           ;  on console
007F 2A0301      69          LD    HL,(TRACK)  ; Get current track #
0082 113501      70          LD    DE,WORK     ; Convert to ASCII str
0085 CF11        71          SC    17
0087 AF          72          XOR   A
0088 12          73          LD    (DE),A      ; Mark end of string
0089 112E01      74          LD    DE,MSG3     ; Display current track
008C CF02        75          SC    2
008E 2A0301      76          LD    HL,(TRACK)  ; Point to current track
0091 23          77          INC   HL          ; Add one
0092 220301      78          LD    (TRACK),HL  ; Save as next track #
0095 3A0501      79          LD    A,(DRIVE)   ; Drive code
0098 47          80          LD    B,A
0099 3AFF00      81          LD    A,(CYL)     ; Cyl length
009C 4F          82          LD    C,A
009D ED5BFB00    83          LD    DE,(SECT)   ; Sector number
00A1 21C601      84          LD    HL,BUFF
00A4 CF32        85          SC    50          ; Read
00A6 2AFF00      86          LD    HL,(CYL)    ; Get cyl len
00A9 19          87          ADD   HL,DE       ; Compute next sect addr
00AA 22FB00      88          LD    (SECT),HL   ; Store
00AD EB          89          EX    DE,HL       ; Put to DE
00AE 2AF900      90          LD    HL,(TOTLEN) ; Get total size
00B1 B7          91          OR    A           ; Clear CY
00B2 ED52        92          SBC   HL,DE       ; Test if done
00B4 20C5        93          JR    NZ,LOOP     ; Loop if not
00B6 CF12        94          SC    18          ; Else CR/LF
00B8            95 QUIT:
00B8 3A0501      96          LD    A,(DRIVE)   ; Get drive code
00BB 47          97          LD    B,A
00BC CF09        98          SC    9           ; Mount it
00BE AF          99          XOR   A           ; RC = 0
00BF CF00       100          SC    0           ; Quit
00C1           101 ERR:
00C1 F5         102          PUSH  AF          ; Save all registers
00C2 C5         103          PUSH  BC
00C3 D5         104          PUSH  DE
00C4 E5         105          PUSH  HL
00C5 F630       106          OR    '0'
00C7 321801     107          LD    (ERRCD),A   ; Save error code in msg
00CA EB         108          EX    DE,HL       ; Convert track, sect
```

```
00CB  112401      109          LD      DE,ERRSECT  ;  for display
00CE  44          110          LD      B,H
00CF  CF10        111          SC      16
00D1  45          112          LD      B,L
00D2  CF10        113          SC      16
00D4  3E48        114          LD      A,'H'
00D6  12          115          LD      (DE),A
00D7  13          116          INC     DE
00D8  3E0D        117          LD      A,13        ; Mark end of message
00DA  12          118          LD      (DE),A
00DB  110601      119          LD      DE,MSG2     ; Display error msg
00DE  CF02        120          SC      2
00E0  E1          121          POP     HL          ; Restore all register
00E1  D1          122          POP     DE
00E2  C1          123          POP     BC
00E3  F1          124          POP     AF
00E4  AF          125          XOR     A           ; Ignore
00E5  C9          126          RET                 ; Continue
00E6  44726976    127 MSG1:    DC      'Drive Code Missing',13
00EA  6520436F
00EE  6465204D
00F2  69737369
00F6  6E670D
00F9              128 TOTLEN:  DS      2
00FB              129 SECT:    DS      2
00FD              130 SURF:    DS      2
00FF              131 CYL:     DS      2
0101              132 TRACKS:  DS      2
0103              133 TRACK:   DS      2
0105              134 DRIVE:   DS      1
0106  4469736B    135 MSG2:    DC      'Disk Error Code = '
010A  20457272
010E  6F722043
0112  6F646520
0116  3D20
0118              136 ERRCD:   DS      1
0119  2C205365    137          DC      ', Sector = '
011D  63746F72
0121  203D20
0124              138 ERRSECT: DS      10
012E  54726163    139 MSG3:    DC      'Track: '
0132  6B3A20
0135              140 WORK:    DS      10
000D              141 CR:      EQU     13
000A              142 LF:      EQU     10
013F  48454C50    143 HELPL:   DC      'HELP    ',CR
0143  20202020
0147  0D
0148  46756E63    144 HELPM:   DC      'Function: Full disk read to check'
014C  74696F6E
0150  3A204675
0154  6C6C2064
0158  69736B20
015C  72656164
0160  20746F20
0164  63686563
```

```
0168 6B
0169 20202020     145          DC      '          disk errors.',LF
016D 20202020
0171 20206469
0175 736B2065
0179 72726F72
017D 732E0A
0180 0A           146          DC      LF
0181 53796E74     147          DC      'Syntax:  VERIFY drive',LF
0185 61783A20
0189 20205645
018D 52494659
0191 20647269
0195 76650A
0198 0A           148          DC      LF
0199 57686572     149          DC      'Where:',LF
019D 653A0A
01A0 20206472     150          DC      '  drive   is the drive to be verified'
01A4 69766520
01A8 20206973
01AC 20746865
01B0 20647269
01B4 76652074
01B8 6F206265
01BC 20766572
01C0 69666965
01C4 64
01C5 00           151          DC      0
01C6              152 BUFF:
01C6              153          END
```

No assembly errors.

Cross Reference List

Symbol--- Value Type Line *** References ***

Symbol	Value	Type	Line	*** References ***			
BUFF	01C6	R 00	152	84			
CR	000D	A 00	141	143			
CYL	00FF	R 00	131	58	81	86	
DRIVE	0105	R 00	134	22	33	79	96
ERR	00C1	R 00	101	31			
ERRCD	0118	R 00	136	107			
ERRSECT	0124	R 00	138	109			
HELPL	013F	R 00	143	4			
HELPM	0148	R 00	144	13			
LF	000A	A 00	142	145	146	147	148 149
LOOP	007B	R 00	66	93			
MSG1	00E6	R 00	127	24			
MSG2	0106	R 00	135	119			
MSG3	012E	R 00	139	74			
NOHELP	0016	R 00	17	9			
OKFD	0029	R 00	28	23			
QUIT	00B8	R 00	95	29			
SECT	00FB	R 00	129	64	83	88	
SURF	00FD	R 00	130	45	54		

```
TESTH    0006  R 00      6     12
TOTLEN   00F9  R 00    128     61     90
TRACK    0103  R 00    133     63     69     76     78
TRACKS   0101  R 00    132     50     59
VERIFY   0000  R 00      2
WORK     0135  R 00    140     70
```

Example 2 - BASIC USR Subroutine

```
Addr Obj-Code   Line *** Source Statement ***

                  2 UPPER:    REL
0000 C30300       3           JP      ENTRY0      ; Convert to upper case only
0003              4 ENTRY0:
0003 E5           5           PUSH    HL          ; Save current HL
0004 7E           6           LD      A,(HL)      ; Get string length
0005 47           7           LD      B,A         ; Copy to B reg
0006              8 .LOOP:
0006 23           9           INC     HL          ; Point next character
0007 7E          10           LD      A,(HL)      ; Get character
0008 FE61        11           CP      'a'         ; Test lowercase a
000A 3807        12           JR      C,.NOTLOW   ; Ignore if not lowercase
000C FE7B        13           CP      '{'         ; Test lowercase z
000E 3003        14           JR      NC,.NOTLOW  ; Ignore if not lowercase
0010 D620        15           SUB     32          ; Translate to uppercase
0012 77          16           LD      (HL),A      ; Restore to string
0013             17 .NOTLOW:
0013 10F1        18           DJNZ    .LOOP       ; Repeat
0015 E1          19           POP     HL          ; Restore HL register
0016             20 STRIP:
0016 E5          21           PUSH    HL          ; Restore
                 22                               ; The following code will str
                 23                               ; trailing blanks from the st
0017 7E          24           LD      A,(HL)      ; Get string length
0018 47          25           LD      B,A         ; Copy to B reg
0019 85          26           ADD     L           ; Compute end address
001A 3001        27           JR      NC,.NOC
001C 24          28           INC     H
001D            29 .NOC:
001D 6F          30           LD      L,A
001E            31 .LOOP:
001E 7E          32           LD      A,(HL)      ; Get ending character
001F FE20        33           CP      ' '         ; Test if space
0021 2003        34           JR      NZ,.RET
0023 2B          35           DEC     HL          ; Point prior
0024 10F8        36           DJNZ    .LOOP
0026            37 .RET:
0026 78          38           LD      A,B
0027 E1          39           POP     HL
0028 77          40           LD      (HL),A      ; Store adjusted count
0029 C9          41           RET                 ; Return to caller
0000            42           END
```

No errors in program

Example 3 - Serial Device Driver

Addr Obj-Code Line *** Source Statement ***

```
                     1
                     2 DEV17:    REL                    ; relocatable
                     3
0000 C30F00          4           JP    ST              ; get status
0003 C35D00          5           JP    IN              ; get byte
0006 C37F00          6           JP    OUT             ; put byte
0009 C39C00          7           JP    INIT            ; initialize
000C C35401          8           JP    UNIN            ; un-initialize
                     9
000F                10 ST:
                    11 ;
                    12 ; get SIO status
                    13 ;
000F 3A5C01         14           LD    A,(BUFI)        ; get count
0012 B7             15           OR    A               ; test if any
0013 F5             16           PUSH  AF              ; save
0014 DB13           17           IN    A,(DA+2)        ; get port status
0016 CB57           18           BIT   2,A             ; test txrdy
0018 2841           19           JR    Z,.NOTRDY       ; brif not ready
001A FD7E1C         20           LD    A,(IY+28)       ; get enab type
001D CB47           21           BIT   0,A             ; CTS/DTR
001F 202D           22           JR    NZ,.ENAB1       ;
0021 CB4F           23           BIT   1,A             ;
0023 2021           24           JR    NZ,.ENAB2       ; brif DC1/DC3
0025 CB57           25           BIT   2,A             ; test
0027 282F           26           JR    Z,.RDY          ; brif not ETX/ACK
0029               27 .ENAB3:
0029 F1            28           POP   AF              ; get in flags
002A F5            29           PUSH  AF              ; re-save
002B 2810          30           JR    Z,.TEST3        ; brif no char rdy
002D F1            31           POP   AF              ; else, throw away
002E CD6600        32           CALL  INCH            ; get char
0031 E67F          33           AND   7FH             ; mask
0033 FE06          34           CP    ACK             ; test ACK
0035 20D8          35           JR    NZ,ST           ; brif not
0037 FD361D00      36           LD    (IY+29),0       ; store
003B 18D2          37           JR    ST              ; go around
003D              38 .TEST3:
003D FD7E1D        39           LD    A,(IY+29)       ; get busy
0040 FE80          40           CP    128             ; wait for ACK?
0042 2014          41           JR    NZ,.RDY         ; brif ready
0044 1815          42           JR    .NOTRDY         ; else, busy
0046             43 .ENAB2:
0046 FD7E1D        44           LD    A,(IY+29)       ; get busy flag
0049 B7            45           OR    A               ; test
004A 200F          46           JR    NZ,.NOTRDY      ; brif busy
004C 180A          47           JR    .RDY            ;
004E             48 .ENAB1:
004E 3E10          49           LD    A,10H           ;
0050 D313          50           OUT   (DA+2),A        ; reset ext/status int
0052 DB13          51           IN    A,(DA+2)        ; get reg 0
0054 CB5F          52           BIT   3,A             ; test DTR
```

```
0056 2803        53          JR      Z,.NOTRDY    ;
0058             54  .RDY:
0058 F1          55          POP     AF           ; get input status
0059 37          56          SCF                  ; turn on cy
005A C9          57          RET                  ; return
005B             58  .NOTRDY:
005B F1          59          POP     AF           ; get input status
005C C9          60          RET                  ; return
                 61
005D             62  IN:
                 63  ;
                 64  ; get byte from SIO
                 65  ;
005D CD0F00      66          CALL    ST           ; get status
0060 2004        67          JR      NZ,INCH      ; brif some char
0062 CF6B        68          SC      107          ; deactivate until interrupt
0064 18F7        69          JR      IN           ; loop
0066             70  INCH:
0066 C5          71          PUSH    BC           ; save regs
0067 D5          72          PUSH    DE           ;
0068 E5          73          PUSH    HL           ;
0069 215C01      74          LD      HL,BUFI      ; point buffer
006C F3          75          DI                   ;
006D 35          76          DEC     (HL)         ; decr length
006E 4E          77          LD      C,(HL)       ; get length
006F 0600        78          LD      B,0          ; zero msb
0071 23          79          INC     HL           ; point first char
0072 7E          80          LD      A,(HL)       ; load it
0073 2805        81          JR      Z,.MT        ; brif buffer now empty
0075 545D        82          LD      DE,HL        ; copy register
0077 23          83          INC     HL           ;
0078 EDB0        84          LDIR                 ; compress the buffer
007A             85  .MT:
007A FB          86          EI                   ; turn on ints
007B E1          87          POP     HL           ; restore regs
007C D1          88          POP     DE           ;
007D C1          89          POP     BC           ;
007E C9          90          RET                  ; return
                 91
007F             92  OUT:
                 93  ;
                 94  ; put byte to device
                 95  ;
007F CD0F00      96          CALL    ST           ; get status
0082 3804        97          JR      C,OUT1       ; brif output ready
0084 CF4F        98          SC      79           ; snu (non interrupt output)
0086 18F7        99          JR      OUT          ; loop
0088             100 OUT1:
0088 FD341D      101         INC     (IY+29)      ; bump count
008B FD7E1D      102         LD      A,(IY+29)    ; load
008E FE80        103         CP      128          ; max?
0090 2006        104         JR      NZ,OUT2      ; no
0092 3E03        105         LD      A,ETX        ; else, send ETX
0094 D311        106         OUT     (DA),A       ; write
0096 18E7        107         JR      OUT          ; wait for ACK
0098             108 OUT2:
```

```
0098 79           109        LD    A,C          ; get char
0099 D311         110        OUT   (DA),A       ; write
009B C9           111        RET                ; return
009C              112 INIT:
009C FD229D01     113        LD    (UCB),IY     ; save ucb address
00A0 3E18         114        LD    A,18H        ;
00A2 D313         115        OUT   (DA+2),A     ; reset device
00A4 FD7E05       116        LD    A,(IY+5)     ; get baud rate
00A7 E6F0         117        AND   0F0H         ; mask
00A9 47           118        LD    B,A          ; save enab
00AA FD7E05       119        LD    A,(IY+5)     ; load again
00AD E60F         120        AND   0FH          ; mask
00AF 2006         121        JR    NZ,.SOMEB    ; brif some
00B1 3E0B         122        LD    A,11         ; default to 9600
00B3 B0           123        OR    B            ; merge
00B4 FD7705       124        LD    (IY+5),A     ;
00B7              125 .SOMEB:
00B7 E60F         126        AND   0FH          ; mask
00B9 FE0E         127        CP    14           ; too big?
00BB 3806         128        JR    C,.OKB       ; brif ok
00BD 3E0B         129        LD    A,11         ; else, 9600
00BF B0           130        OR    B            ; merge
00C0 FD7705       131        LD    (IY+5),A     ;
00C3              132 .OKB:
00C3 E60F         133        AND   0FH          ; mask
00C5 3D           134        DEC   A            ; less one
00C6 5F           135        LD    E,A          ; save
00C7 87           136        ADD   A            ; times two
00C8 83           137        ADD   E            ; times three
00C9 5F           138        LD    E,A          ;
00CA 1600         139        LD    D,0          ; zero high
00CC 219F01       140        LD    HL,BAUD      ; point table
00CF 19           141        ADD   HL,DE        ; offset
00D0 0E25         142        LD    C,CTC        ;
00D2 0602         143        LD    B,2          ; two bytes
00D4 EDB3         144        OTIR               ; program it
00D6 E5           145        PUSH  HL           ; save pointer
00D7 F3           146        DI                 ; turn off ints
00D8 3E08         147        LD    A,8          ; vector/2
00DA 11CA01       148        LD    DE,RETI      ; dummy addr
00DD CF67         149        SC    103          ; put vect
00DF 3C           150        INC   A            ;
00E0 CF67         151        SC    103          ; put vect
00E2 11CD01       152        LD    DE,INTI      ; input interrupt
00E5 3C           153        INC   A            ;
00E6 CF67         154        SC    103          ; put vect
00E8 3C           155        INC   A            ;
00E9 CF67         156        SC    103          ;
00EB 3E02         157        LD    A,2          ; reg 2
00ED D313         158        OUT   (DA+2),A     ;
00EF 3E10         159        LD    A,010H       ; int vector
00F1 D313         160        OUT   (DA+2),A     ;
00F3 E1           161        POP   HL           ; get pointer
00F4 3E04         162        LD    A,4          ; wr 4
00F6 D313         163        OUT   (DA+2),A     ;
00F8 FDCB087E     164        BIT   7,(IY+8)     ; parity enable?
```

```
00FC 280C        165            JR     Z,.NOPAR        ; brif none
00FE FDCB0876    166            BIT    6,(IY+8)        ; test even/odd
0102 3E0D        167            LD     A,00001101B     ; even
0104 2006        168            JR     NZ,.OUT         ;
0106 3E0F        169            LD     A,00001111B     ; odd
0108 1802        170            JR     .OUT            ;
010A             171  .NOPAR:
010A 3E0C        172            LD     A,00001100B     ; noparity
010C             173  .OUT:
010C B6          174            OR     (HL)            ; merge clocks
010D D313        175            OUT    (DA+2),A        ;
010F 3E03        176            LD     A,3             ; wr 3 (rcv logic)
0111 D313        177            OUT    (DA+2),A        ;
0113 FDCB087E    178            BIT    7,(IY+8)        ; parity?
0117 3EC1        179            LD     A,11000001B     ; default
0119 2802        180            JR     Z,.NP           ; brif ok
011B 3E41        181            LD     A,01000001B     ; else, 7 bits
011D             182  .NP:
011D FDCB1C66    183            BIT    4,(IY+28)       ; auto enable?
0121 2802        184            JR     Z,.NOEN         ; no
0123 CBEF        185            SET    5,A             ; else, turn on
0125             186  .NOEN:
0125 D313        187            OUT    (DA+2),A        ;
0127 3E01        188            LD     A,1             ; wr 1 (control)
0129 D313        189            OUT    (DA+2),A        ;
012B 3E1C        190            LD     A,00011100B     ; int mask
012D D313        191            OUT    (DA+2),A        ;
012F 3E05        192            LD     A,5             ; wr 5 (trns)
0131 D313        193            OUT    (DA+2),A        ;
0133 FDCB087E    194            BIT    7,(IY+8)        ; test parity
0137 3EEA        195            LD     A,11101010B     ; default
0139 2802        196            JR     Z,.NTP          ; brif ok
013B 3EAA        197            LD     A,10101010B     ; else parity = 7 bits
013D             198  .NTP:
013D D313        199            OUT    (DA+2),A        ;
013F FB          200            EI                     ; allow ints now
0140 AF          201            XOR    A               ; leave pointing to 0
0141 D313        202            OUT    (DA+2),A        ;
0143 FD771D      203            LD     (IY+29),A       ;
0146 FD771E      204            LD     (IY+30),A       ;
0149 FDCB1C6E    205            BIT    5,(IY+28)       ; test enable 2
014D C8          206            RET    Z               ;
014E 3EFF        207            LD     A,0FFH          ;
0150 FD771E      208            LD     (IY+30),A       ; set sw
0153 C9          209            RET                    ; return
                 210
0154             211  UNIN:
0154 AF          212            XOR    A               ;
0155 D313        213            OUT    (DA+2),A        ;
0157 3E18        214            LD     A,00011000B     ; reset channel
0159 D313        215            OUT    (DA+2),A        ;
015B C9          216            RET                    ; return
                 217
015C 00          218  BUFI:     DC     0               ; buffer length
015D             219            DS     64              ; the buffer itself
                 220
```

```
0011              221 DA:      EQU    11H          ; port address
0025              222 CTC:     EQU    25H          ;
019D              223 UCB:     DS     2            ;
0011              224 DC1:     EQU    11H          ;
0013              225 DC3:     EQU    13H          ;
0003              226 ETX:     EQU    03H          ;
0006              227 ACK:     EQU    06H          ;
                  228
019F              229 BAUD:
019F 076680       230          DC     7,102,80H    ; 75 = 32x16x102.4 timer
01A2 074680       231          DC     7,70,80H     ; 110 = 32x16x699.8181 timer
01A5 073980       232          DC     7,57,80H     ; 134.5 = 32x16x57.1003 timer
01A8 4780C0       233          DC     47H,128,0C0H ; 150 = 64x128
01AB 4740C0       234          DC     47H,64,0C0H  ; 300 = 64x64
01AE 4720C0       235          DC     47H,32,0C0H  ; 600 = 64x32
01B1 4710C0       236          DC     47H,16,0C0H  ; 1200 = 64x16
01B4 4708C0       237          DC     47H,8,0C0H   ; 2400 = 64x8
01B7 4704C0       238          DC     47H,4,0C0H   ; 4800 = 64x4
01BA 470580       239          DC     47H,5,80H    ; 7200 = 32x5.3333
01BD 4702C0       240          DC     47H,2,0C0H   ; 9600 = 64x2
01C0 4701C0       241          DC     47H,1,0C0H   ; 19200 = 64x1
01C3 470240       242          DC     47H,2,40H    ; 38400 = 16x2
                  243
                  244
01C6              245 SIORET:
01C6 C1           246          POP    BC           ; restore regs
01C7 FDE1         247          POP    IY           ;
01C9 F1           248          POP    AF           ; restore a,flag
01CA              249 RETI:
01CA FB           250          EI                  ; turn on ints
01CB ED4D         251          RETI                ; return
                  252
01CD              253 INTI:
                  254 ;
                  255 ; service receiver interrupt
                  256 ;
01CD FB           257          EI                  ; turn on ints
01CE F5           258          PUSH   AF           ; save reg A,F
01CF FDE5         259          PUSH   IY           ;
01D1 FD2A9D01     260          LD     IY,(UCB)     ; point to ucb
01D5 C5           261          PUSH   BC           ; save B,C
01D6 3E01         262          LD     A,1          ; read reg 1
01D8 D313         263          OUT    (DA+2),A     ;
01DA DB13         264          IN     A,(DA+2)     ; get second status
01DC 47           265          LD     B,A          ; save it
01DD DB11         266          IN     A,(DA)       ; get char
01DF FDCB086E     267          BIT    5,(IY+8)     ; 8 bit char
01E3 2002         268          JR     NZ,.EIGHT    ; yes
01E5 CBBF         269          RES    7,A          ; turn off parity
01E7              270 .EIGHT:
01E7 4F           271          LD     C,A          ; save char
                  272 ;
                  273 ; test parity
                  274 ;
01E8 CB60         275          BIT    4,B          ; test for parity even
01EA 2806         276          JR     Z,.NOPE      ; brif not
```

```
01EC 0E3F       277           LD      C,'?'       ; replace char
01EE 3E30       278           LD      A,30H       ;
01F0 D313       279           OUT     (DA+2),A    ; reset parity error
01F2            280 .NOPE:
01F2 AF         281           XOR     A           ; reset to zero
01F3 D313       282           OUT     (DA+2),A    ;
01F5 CF66       283           SC      102         ; translate input char
01F7 38CD       284           JR      C,SIORET    ; ignore it?
01F9 4F         285           LD      C,A         ; save char
01FA 3A5C01     286           LD      A,(BUFI)    ; get prev count
01FD FE40       287           CP      64          ; test full
01FF 28C5       288           JR      Z,SIORET    ; full?, ignore
0201            289 ROC:
0201 FD7E1E     290           LD      A,(IY+30)   ; see if enab2
0204 B7         291           OR      A           ;
0205 2814       292           JR      Z,.NOENAB   ; not
0207 79         293           LD      A,C         ;
0208 E67F       294           AND     7FH         ;
020A FE11       295           CP      DC1         ;
020C 280A       296           JR      Z,.CTLQ     ;
020E FE13       297           CP      DC3         ;
0210 2009       298           JR      NZ,.NOENAB  ;
0212            299 .CTLS:
0212 FD771D     300           LD      (IY+29),A   ; set the busy sw
0215 C3C601     301           JP      SIORET      ;
0218            302 .CTLQ:
0218 AF         303           XOR     $'UK$'  021B               305 .NOENAB:
021B 79         306           LD      A,C         ; get this char
021C            307 R2:
021C D5         308           PUSH    DE          ; save DE and HL regs
021D E5         309           PUSH    HL          ;
021E 215C01     310           LD      HL,BUFI     ; point buffer
0221 F3         311           DI                  ; turn off ints
0222 34         312           INC     (HL)        ; incr count
0223 5E         313           LD      E,(HL)      ; load it
0224 1600       314           LD      D,0         ; zero high
0226 19         315           ADD     HL,DE       ; point next
0227 FB         316           EI
0228 77         317           LD      (HL),A      ; store the character
0229 E1         318           POP     HL          ; restore regs
022A D1         319           POP     DE          ;
022B C3C601     320           JP      SIORET      ; return
                321
022E            322           END
```

No assembly errors.

Example 4 – Parallel Printer Device Driver

```
Addr Obj-Code  Line *** Source Statement ***

                  1            REL
0000              2 BEGDEV:
0000 C30F00       3            JP    STATUS     ; Return status
0003 C31900       4            JP    INPUT      ; Get input from device
0006 C31A00       5            JP    OUTPUT     ; Put output to device
0009 C31900       6            JP    INIT       ; Initialize driver
000C C32300       7            JP    DEINIT     ; Deinitialize driver
                  8 ;
                  9 ; Status routine - output only device
                 10 ;
000F             11 STATUS:
000F DB01        12            IN    A,(STATO)  ; Get device status byte
0011 E601        13            AND   STAMSK     ; Test for busy
0013 2002        14            JR    NZ,.BUSY   ; BRIF not ready
0015 37          15            SCF              ; Turn on carry flag
0016 C9          16            RET              ; Return with Z and C set
0017             17 .BUSY:
0017 AF          18            XOR   A          ; Set Z flag - reset C flag
0018 C9          19            RET              ;
                 20 ;
                 21 ; Input routine - output only device
                 22 ;
0019             23 INPUT:
0019 C9          24            RET
                 25 ;
                 26 ; Output routine
                 27 ;
001A             28 OUTPUT:
001A CD0F00      29            CALL  STATUS     ; Get device status
001D 30FB        30            JR    NC,OUTPUT  ; Loop till ready
001F 79          31            LD    A,C        ; Copy character to A reg
0020 D300        32            OUT   (DATAO),A  ; Output the character
0022 C9          33            RET              ; Return to caller
                 34 ;
                 35 ; Initialization routine
                 36 ;
0019             37 INIT:    EQU   INPUT        ; No initialization needed
                 38 ;
                 39 ; Deinitialization routine
                 40 ;
0023             41 DEINIT:  EQU   INPUT        ; No deinitialization
0001             42 STATO:   EQU   1            ; Printer status port
0000             43 DATAO:   EQU   0            ; Printer data port
0001             44 STAMSK:  EQU   01           ; Mask to get status bit
                 45            END
No assembly errors.

Cross Reference List

Symbol--- Value Type  Line *** References ***
  BEGDEV   0000  C 00      2
 .BUSY     0017  C 00     17     14
```

```
DATAO      0000   A 00      43    32
DEINIT     0019   C 00      41     7
INIT       0019   R 00      37     6
INPUT      0019   C 00      23     4    37
OUTPUT     001A   C 00      28     5    30
STAMSK     0001   A 00      44    13
STATO      0001   A 00      42    12
STATUS     000F   C 00      11     3    29
```

Example 5 - Disk Device Driver

```
Addr Obj-Code  Line *** Source Statement ***

                  2 N$DISKIO: REL
                  3
                  4          ENTRY DISK
                  5
0000              6 DISK:
                  7 ;
                  8 ; transfer vector
                  9 ;
0000 C30C00      10          JP    SEL
0003 C31700      11          JP    RES
0006 C31900      12          JP    READ
0009 C32000      13          JP    WRITE
                 14 ;
                 15 ; select drive
                 16 ;
000C             17 SEL:
000C E603        18          AND   3          ; mask
000E 327F00      19          LD    (DESC+4),A  ; store
0011 3E07        20          LD    A,7         ; force controller to select
0013 328A00      21          LD    (DESC+15),A
0016 C9          22          RET              ; return
                 23 ;
                 24 ; rezero
                 25 ;
0017             26 RES:
0017 AF          27          XOR   A          ; not implemented
0018 C9          28          RET
                 29 ;
                 30 ; read
                 31 ;
0019             32 READ:
0019 328300      33          LD    (DESC+8),A  ; store
001C 3E00        34          LD    A,0         ; get cmd
001E 1805        35          JR    COM         ; go common
                 36 ;
                 37 ; write
                 38 ;
0020             39 WRITE:
0020 328300      40          LD    (DESC+8),A  ; store
0023 3E01        41          LD    A,1         ; cmd
                 42 ;
                 43 ; common
                 44 ;
0025             45 COM:
0025 DDE5        46          PUSH  IX          ; save ix
0027 DD217B00    47          LD    IX,DESC
002B DD7701      48          LD    (IX+1),A    ; store
                 49 ;
                 50 ; store head, cyl and sector
                 51 ;
002E DD7105      52          LD    (IX+5),C    ; sector
0031 DD7202      53          LD    (IX+2),D    ; msb cyl
```

```
0034 DD7303     54              LD      (IX+3),E      ; lsb cyl
                55 ;
                56 ; store mem address
                57 ;
0037 DD7406     58              LD      (IX+6),H      ; msb mem
003A DD7507     59              LD      (IX+7),L      ; lsb
                60 ;
                61 ; perform operation
                62 ;
003D C5         63              PUSH    BC            ; save regs
003E D5         64              PUSH    DE
003F E5         65              PUSH    HL
0040 CD7700     66              CALL    DESC-4        ; jump to vector
0043 E1         67              POP     HL            ; restore regs
0044 D1         68              POP     DE
0045 C1         69              POP     BC
                70 ;
                71 ; restore regs
                72 ;
0046 DD4E05     73              LD      C,(IX+5)      ; sector
0049 DD6606     74              LD      H,(IX+6)      ; mem
                75 ;
                76 ; test for error
                77 ;
004C DD7E00     78              LD      A,(IX)        ; get status
004F DDE1       79              POP     IX            ; restore ix reg
0051 B7         80              OR      A             ; test
0052 C8         81              RET     Z             ; return no error
                82 ;
                83 ; decode the error
                84 ;
0053 CB6F       85              BIT     5,A           ; test illegal
0055 2018       86              JR      NZ,ERR5       ; brif is
0057 CB5F       87              BIT     3,A           ; test format error
0059 200C       88              JR      NZ,ERR3
005B CB57       89              BIT     2,A           ; test checksum
005D 200C       90              JR      NZ,ERR4
005F CB67       91              BIT     4,A           ; test seek
0061 2010       92              JR      NZ,ERR7
                93 ;
                94 ; else, disk fault - overrun
                95 ;
0063 3E01       96              LD      A,1
0065 180E       97              JR      ERR
0067           98 ERR3:
0067 3E03       99              LD      A,3
0069 180A      100              JR      ERR
006B          101 ERR4:
006B 3E04      102              LD      A,4
006D 1806      103              JR      ERR
006F          104 ERR5:
006F 3E05      105              LD      A,5
0071 1802      106              JR      ERR
0073          107 ERR7:
0073 3E07      108              LD      A,7
0075          109 ERR:
```

```
0075 B7          110          OR    A              ; set nz
0076 C9          111          RET                  ; return
                 112 ;
                 113 ; descriptor follows
                 114 ;
0077 CD40F4      115          CALL  0F440H         ; prom address
007A C9          116          RET                  ; return
007B 00          117 DESC:    DC    0              ; status
007C 00          118          DC    0              ; command (0=read,1=write)
007D 00          119          DC    0              ; msb track
007E 00          120          DC    0              ; lsb track
007F 00          121          DC    0              ; head
0080 00          122          DC    0              ; sector
0081 00          123          DC    0              ; msb mem addr
0082 00          124          DC    0              ; lsb mem addr
0083 00          125          DC    0              ; sector count
0084 00          126          DC    0              ; unit
0085 01          127          DC    1              ; option
0086 00          128          DC    0              ; max head
0087 03          129          DC    1000.SHR.8     ; max track msb
0088 E8          130          DC    1000.AND.0FFH  ; max track lsb
0089 40          131          DC    64             ; max sector
008A 07          132          DC    7              ; curr unit
008B 00          133          DC    0              ; curr track msb
008C 00          134          DC    0              ; curr track lsb
008D 00          135          DC    0              ; error count
008E 00          136          DC    0              ; err track
008F 00          137          DC    0              ; err track
0090 00          138          DC    0              ; err head
0091 00          139          DC    0              ; err sector
0092 FF          140          DC    0FFH           ; bad track table
                 141
0093             142          END
```

No assembly errors.

Example 6 - Class Code Conversion

```
TITLE       'Class Code 4 (SOROQ IQ) Terminal Conversion'
            ;
            ; Entry parameters:
            ;
            ; A - control character to translate
            ; B - console device number
            ; C - control character to translate
            ; H - cursor address column number
            ; L - cursor address line number
            ;
MACLIB      CLASS ; Get MACRO definitions

            ; Translate value 11 to 26 (UP ARROW)
            ; Translate value 12 to 6 (RIGHT ARROW)
            ; Translate value 30 to 1 (HOME)

INIT        OBH,1AH,OCH,06H,1EH,01H
DCA         4       ; Use class 4 cursor controls
DEFINE      HOME,RS
DEFINE      CLEAR,ESC,'*',8CH
DEFINE      EOS,ESC,'Y',8CH
DEFINE      EOL,ESC,'T',8CH
DEFINE      LEFT,BS
DEFINE      RIGHT,FF
DEFINE      UP,VT
DEFINE      EU,ESC,+,8CH
DEFINE      PON,ESC,29H
DEFINE      POFF,ESC,28H
DEFINE      FON,ESC,26H
DEFINE      FOFF,ESC,27H
DEFINE      BON    ; Function not available
DEFINE      BOFF   ; Function not available
DEFINE      RVON   ; Function not available
DEFINE      RVOFF  ; Function not available
DEFINE      ULON   ; Function not available
DEFINE      ULOFF  ; Function not available

END
```

```
>ASM CLASS4 ( / SYSTEM

>MACRO CLASS4

>LINK CLASS4 (SYSTEM

>ATTACH CONSOLE SIO1 (B19200 C4 FF6
```

APPENDIX E

CHARACTER SET

```
==================================================================
|   \ MSD|  0   |  1   |  2   |  3   |  4   |  5   |  6   |  7   |
|    \   |      |      |      |      |      |      |      |      |
|LSD  \  | 000  | 001  | 010  | 011  | 100  | 101  | 110  | 111  |
|=================================================================|
| 0 0000 | NUL  | DLE  | SP   |  0   |  @   |  P   |  `   |  p   |
| 1 0001 | SOH  | DC1  |  !   |  1   |  A   |  Q   |  a   |  q   |
| 2 0010 | STX  | DC2  |  "   |  2   |  B   |  R   |  b   |  r   |
| 3 0011 | ETX  | DC3  |  =   |  3   |  C   |  S   |  c   |  s   |
|-----------------------------------------------------------------|
| 4 0100 | EOT  | DC4  |  $   |  4   |  D   |  T   |  d   |  t   |
| 5 0101 | ENG  | NAK  |  %   |  5   |  E   |  U   |  e   |  u   |
| 6 0110 | ACK  | SYN  |  &   |  6   |  F   |  V   |  f   |  v   |
| 7 0111 | BEL  | ETB  |  '   |  7   |  G   |  W   |  g   |  w   |
|-----------------------------------------------------------------|
| 8 1000 | BS   | CAN  |  (   |  8   |  H   |  X   |  h   |  x   |
| 9 1001 | HT   | EM   |  )   |  9   |  I   |  Y   |  i   |  y   |
| A 1010 | LF   | SUB  |  *   |  :   |  J   |  Z   |  j   |  z   |
| B 1011 | VT   | ESC  |  +   |  ;   |  K   |  [   |  k   |  {   |
|-----------------------------------------------------------------|
| C 1100 | FF   | FS   |  ,   |  <   |  L   |  \   |  l   |  |   |
| D 1101 | CR   | GS   |  -   |  =   |  M   |  ]   |  m   |  }   |
| E 1110 | SO   | RS   |  .   |  >   |  N   |  ^   |  n   |  ~   |
| F 1111 | SI   | US   |  /   |  ?   |  O   |  _   |  o   | DEL  |
==================================================================
```

A more complete character set chart is available in the OASIS System Reference Manual.

MACRO

Reader's Comments

Name _____ Date ___/___/___

Organization _____

Street _____

City _____ State _____ Zip _____

Name of manual: _____

Did you find errors in this manual? If so, specify with page number.

_____
_____
_____
_____
_____
_____
_____
_____

Did you find this manual understandable, usable, and well-organized?
Please make suggestions for improvement.

_____
_____
_____
_____
_____
_____
_____

Is there sufficient documentation on associated system programs required for use of
the software described in this manual? If not, what material is missing and where
should it be placed?

_____
_____
_____
_____
_____
_____
_____

Indicate the type of user/reader that you most nearly represent:

___ Assembly language programmer
___ Higher-level language programmer (BASIC, FORTRAN, etc.)
___ Occasional programmer (experienced)
___ User with little programming experience
___ Student programmer
___ Non-programmer interested in computer concepts and capabilities
___ Data entry operator

Mail to: OASIS Documentation
                  Phase One Systems, Inc.
                  7700 Edgewater Drive #830
                  Oakland, CA 94621