

Not enough arguments supplied

The number of actual parameters in a procedure call is less than the number of formal parameters for that procedure.

Out of symbol table storage

The pass one symbol table is full. Break your program into smaller modules (or get more external memory).

Program too large (interfile1).

The pass one intermediate file is larger than .WORK.
Increase the size of the .WORK file.

Program too large (variable space exceeds maximum).

The program uses more than 60K of variable space. Move large arrays to external memory.

Recursive attribute doesn't match forward ref

This procedure is defined to be recursive, but it's forward reference declaration does not include the attribute RECURSIVE, or visa versa.

RETURN statement not allowed

This RETURN statement does not occur within a procedure or a WHEN statement.

Statement outside of module body

This statement appears after the END of a module. Statements cannot appear outside a module.

String constant too long

This string constant is longer than 128 characters.

Subscript not allowed

A subscript appears in the declaration of an external variable, a label, a literal, a data array, a forward reference procedure, or a formal parameter of a procedure.

System error with disk configuration (.WORK is wrong length).

The length of .WORK is not a multiple of eight sectors.
Recreate it with a sector length that is a multiple of eight.

System error with disk configuration (.WORK missing).

The file .WORK is not on the system device. Create a .WORK file on the system device.

Too many arguments supplied

The number of actual parameters in a procedure call is greater than the number of formal parameters for that procedure.

Too many END statements

There is an extra END statement. This is often caused by changing the compound statement of the THEN clause of an IF statement to a simple statement and forgetting to remove its corresponding END.

Too many digits in number

More than six octal digits appear in an octal constant or more than four hexadecimal digits appear in a hexadecimal constant or more than eight decimal digits appear to the left or to the right of the decimal point in a floating point constant.

Too many externals declared

More than 4096 external declarations appear in a single source file. Remove unnecessary external declarations.

Too many nested 'BEGIN' statements

BEGIN statements or procedures have been nested beyond the compiler's capability to deal with them. Change any BEGINs that do not define new localization levels to DOs and unnest procedures.

Too many nested insert files

Insert files have been nested beyond the compiler's capability to deal with them. Redefine the insert file structure so it does not nest as deeply or construct a module from the lower level functions.

Too many numeric constants

The compiler has run out of floating point constant stack space. Use multiple statements and temporary variables to evaluate the expression.

Too many procedures/labels

There are too many procedures (or labels) defined in the source file. Break your program into smaller modules.

Undeclared procedure argument

A formal parameter of this procedure has not been declared.

Undefined symbol '<identifier>'

An attempt has been made to use an identifier before it is declared.

Warning: public variable declared inside a proc

A variable declared within a procedure has been declared to be PUBLIC. This is okay, but the public variable will be undefined if the procedure in which it is declared is never called by the program.

Warning: overwriting contents of data array

One or more elements of a DATA array are being overwritten; DATA arrays are meant to hold constants and should never be altered.

'WHEN' not allowed in procedure

This WHEN statement appears within a procedure. WHEN statements must appear at the outermost layer (i.e., not within a procedure).

Pass 2 Error Messages

Argument types do not match

The type of an actual parameter in this procedure call does not match the type of the corresponding formal parameter.

Expression too complicated at line <line #> (<info>)

The compiler has run out of stack space (<info> is 'push') or expression blocks (<info> is 'get') or automatic temporaries (<info> is 'too many temps in recursive proc'). Use multiple statements and temporary variables to evaluate the expression.

Floating point not allowed with fixed point

An attempt has been made to assign a floating point value to a fixed point variable (this includes decimal constants larger than 65535). Use the INT function if this was intended.

Program too large (pass 2 intermediate file).

The pass two intermediate file is larger than .WORK.
Increase the size of the .WORK file.

Program too large (too many variables declared).

This program uses more than 60K of variable space (including temporaries allocated by pass two). Move large arrays to external memory.

Too many numeric constants

The compiler has run out of floating point constant stack space. Use multiple statements and temporary variables to evaluate the expression.

Too many procedures/labels

There are too many procedures (or labels) defined in the source file. Break your program into smaller modules.

Pass 3 Error Messages

Configuration mismatch: Library compiled with Model X processor [in <library name>] (defined in <library name>).

This library was compiled for a later model processor than the one the main program is being compiled for. Recompile the source module for the same processor model that the main program is being compiled for.

Duplicate definition: <identifier> at line <line #> [in <library name>] (defined in <library name>).

This identifier defined (i.e., declared to be PUBLIC) in the specified library is redefined at this line in MAIN or in the specified library. Change one of the declarations to EXTERNAL.

Duplicate WHEN statement <when ID> [in <library name>] (defined in <library name>).

A WHEN statement (specified by the number <when ID>) in the specified library (or MAIN) also appears in another library. Remove one of the WHENs or combine the two.

Library "<library name>" incompatible with current compiler. Please recompile it.

This library was created by an outdated version of the compiler. Recompile its source module.

Memory conflict - starting RAM address is too low. For this program, the RAM area must be at or above location <#> decimal.

An attempt has been made (with the RAM statement) to set the start of the variable area before the end of the object code. Set the RAM to start at location <#> or above.

Not enough external memory to run program.

The swap file for this program is larger than the amount of external memory in this computer. For this program to run on this computer, change some swapping procedures to non-swapping.

Not enough memory for linker symbol table.

There is not enough internal memory for the linker symbol table. This program cannot be compiled without external memory or more internal memory. Verify the configuration by running the CONFIGUR program.

Not enough memory for Pass3 intermediate file.

This computer does not have enough internal memory to run the XPL compiler. Verify the configuration by running the CONFIGUR program.

Object file too big for compilation.

The internal memory image for this program (or the size of the library being compiled) is larger than 64K; change some non-swapping procedures to swapping (or break this module into smaller pieces).

Program too large for compilation (libraries exceed work file length).

The pass three linker intermediate file is larger than .WORK. Increase the size of the .WORK file.

Program too large for compilation (too many external references).

The external relocation table is full. Redesign your modules to have fewer and tighter interfaces (or get more internal memory).

Program too large for compilation (too many keys).

There is not enough memory to compile this program.

Program too large for compilation (too much swapping scon).

There is not enough internal memory for the swapping string constant relocation table. Change some swapping procedures with many string constants to non-swapping or move some string constants in swapping procedures to DATA arrays.

Program too large (object file/IF collision).

The .WORK file is not large enough to hold both the object file and the intermediate file. Increase the size of .WORK.

Program too large (pass3 object file).

The .WORK file is not large enough to hold both the object file and the intermediate file. Increase the size of .WORK.

Specified library '<library name>' is not a relocatable binary.

The specified library is not the compiled version of a module (file type must be RELOC).

Specified library '<library name>' is zero-length.

The specified library is an empty file.

Specified library '<library name>' is too large.

The specified library is larger than 64K (and thus was not created by the compiler).

System file '.RTB-7' missing.

This file should be in .SYSTEM or top-level system catalog.

System file '.RTC-7' missing.

This file should be in .SYSTEM or top-level system catalog.

Too many public symbols defined.

The pass three symbol table is full. Redesign your modules to have fewer and tighter interfaces (or get more external memory).

Too many public symbols defined (symbol table overflow).

The pass three symbol table is full. Redesign your modules have fewer and tighter interfaces (or get more external memory).

Too many libraries referenced.

The pass three library table is full. Redesign your program to have fewer modules.

Too many libraries referenced (library table overflow).

The pass three library table is full. Redesign your program to have fewer modules.

Types don't match: <identifier> at line <line #>
[in <library name>] (defined in <library name>).

The type of this public identifier (or the parameter type list of this procedure) is different in the specified library (or MAIN) than it was in the defining library. This is usually caused by changing the parameters to a public procedure, but not recompiling all modules that include an external declaration for that procedure (whether they actually use it or not).

Unresolved reference: <identifier> at line <line #>
[in <library name>].

This identifier has been declared EXTERNAL in the specified library, but no corresponding PUBLIC declaration appears in the program.

.WORK not large enough to create swap file.

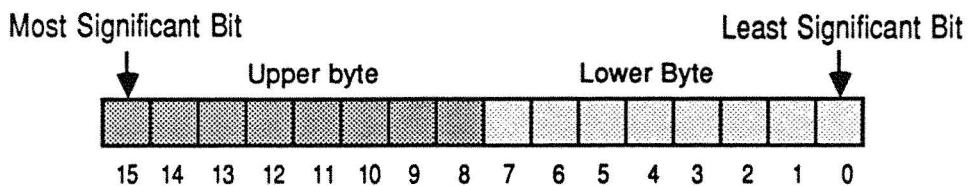
The combination of the pass three linker intermediate file
and the swap file is larger than .WORK. Increase the size of
the .WORK file.

Appendix E - Internal Representations

Fixed Point Numbers

Each fixed point data element is a 16-bit, single word quantity. Fixed point numbers can be interpreted as signed integers in the range of -32,768 to +32,767 or as unsigned integers in the range of 0 to 65,535. Both signed and unsigned integers are processed and stored identically inside the computer; the difference lies in the user interpretation of the bit patterns that are stored in memory.

The bits that make up a fixed point number are labeled 0 through 15 as shown in the following diagram.

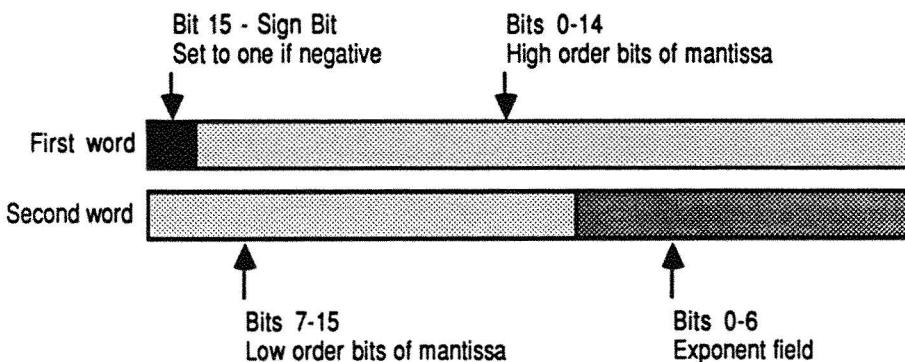


Negative integers are stored in memory in two's complement form. Bit 15 is the sign bit and will be set to a one in the case of a negative quantity. To obtain the negative value of a number in two's complement form, take the one's complement of the number (invert each bit) and then add one. Two's complement numbers are defined in such a way that when a positive number is added to its negative counterpart the result is a 16-bit word of all zeros. Note the following examples:

00000000000000011	=	3
00000000000000010	=	2
00000000000000001	=	1
00000000000000000	=	0
11111111111111111	=	-1
11111111111111110	=	-2
11111111111111101	=	-3

Floating Point Numbers

Floating point numbers are signed reals in the range of plus and minus 5.5 E-20 to 9.0 E18. They are stored in two consecutive locations of memory in a compacted form as shown in the following diagram:



Floating point variables are stored in terms of a sign bit, a 24-bit mantissa, and a 7-bit exponent field. The sign bit is a zero in the case of a positive number, or a one in the case of a negative number. To prevent multiple representations of the same number, the mantissa is always normalized so that its most significant bit is a one, except in the case of the number zero where the sign, mantissa, and exponent field are all zeros. The binary point is always located to the left of the most significant bit of the mantissa.

The exponent field represents a power of two exponent that is used to scale the mantissa up to 64 places left or right. The exponent field is stored in excess-64 notation so that an exponent field of all zeros represents -64 and all ones represents +63.

For example, the floating point number -25.0 is represented internally as follows:

-25.0000 = 1110010000000000
000000001000101

and is interpreted (in binary) like this:

Sign bit = 1
Mantissa = .110010000000000000000000
Exponent = 1000101

Computing the decimal equivalent of this number is done in the following way:

Mantissa = (1*0.5) + (1*0.25) + (1*0.03125) = 0.78125
Exponent = 69 - 64 = 5
Mantissa with exponent = 0.78125 * 32 = 25.00
Result with sign bit = -25.00

The following table presents the internal bit format for some floating point numbers.

<u>Number</u>	<u>Internal Format</u>
0.000000	0000000000000000 0000000000000000
1.000000	0100000000000000 0000000010000001
0.500000	0100000000000000 0000000010000000
25.00000	0110010000000000 000000001000101
0.100000	0110011001100110 0110011000111101

Character Strings

Character strings can be stored in fixed point arrays, using a special format that makes manipulating strings convenient for the programmer. An array that contains ASCII characters has the string length stored in the first word (element zero), and the string characters stored in the rest of the array, starting with element one. The zeroeth element of the array contains the number of used 8-bit bytes in the array, with each byte of the array representing one ASCII character. Each 16-bit array element therefore contains two bytes (two characters), the lower half being an even byte, and the upper half being an odd byte:

Array (0)	Character length of string	
Array (1)	Byte 1	Byte 0
Array (2)	Byte 3	Byte 2
Array (3)	Byte 5	Byte 4
etc.		

Character strings in this standard format can be easily read from and written to the terminal using the LINPUT and PRINT statements. There are also two built-in functions (BYTE and PBYTE) that are used to process textual string information.

Appendix F - Memory Layout

This appendix describes the layout of internal and external memory while an XPL program is executing, and how that program can access and use different areas of memory. There are three basic categories of information in this appendix: the disk image of a program, the internal memory structure, and the external memory structure.

Every ABLE computer has some amount of internal memory, from 16K up to 64K words. The internal memory size of your computer must be specified for the Monitor with the CONFIGUR program. External memory can be purchased as an option and does not need to be specified in the system configuration. The computer simply uses external memory if it finds it in the system. Note that a program which requires external memory (i.e., one that has swapping procedures) will not run on a system that does not have external memory.

If you are going to use the information in this section for programming purposes, you must have a copy of the -XPL programming library on your system. The -XPL file SYSLITS (:-XPL:SYSLITS) contains the system literals that will allow you to access sections of memory during program execution. This programming library can be obtained from New England Digital Customer Service.

It is important to use the system literals in SYSLITS for all programs that access memory. The actual locations used to store things in memory often change with new software releases and SYSLITS is updated to accommodate these changes. Always use the version of -XPL that is compatible with the software version you are running to make sure you are accessing the correct memory locations.

SYSLITS provides the basic information needed to use the system literals. For additional information, refer to the manual "ABLE Series Operating System Reference Manual".

The Configuration Table

The configuration table is a special area of memory that contains information about the system. There is a copy of the configuration table resident in internal memory at all times. There is also a configuration table stored on disk with every program. The table contains information such as how much memory is in the system and what storage devices are attached to the system. When a program is run, the system's configuration is copied into the program's configuration table.

The format of the configuration table is outlined in the -XPL file SYSLITS. The format of the table is the same on disk as it is in memory. There is a pointer to the start of the configuration table which can be obtained by using the literal C#CONTAB. Using this pointer, all the information stored in the table is available by using other literals. For example, the literal C#STKLEN can be used to find out how much internal memory is used for the program's runtime stack.

The Swap File

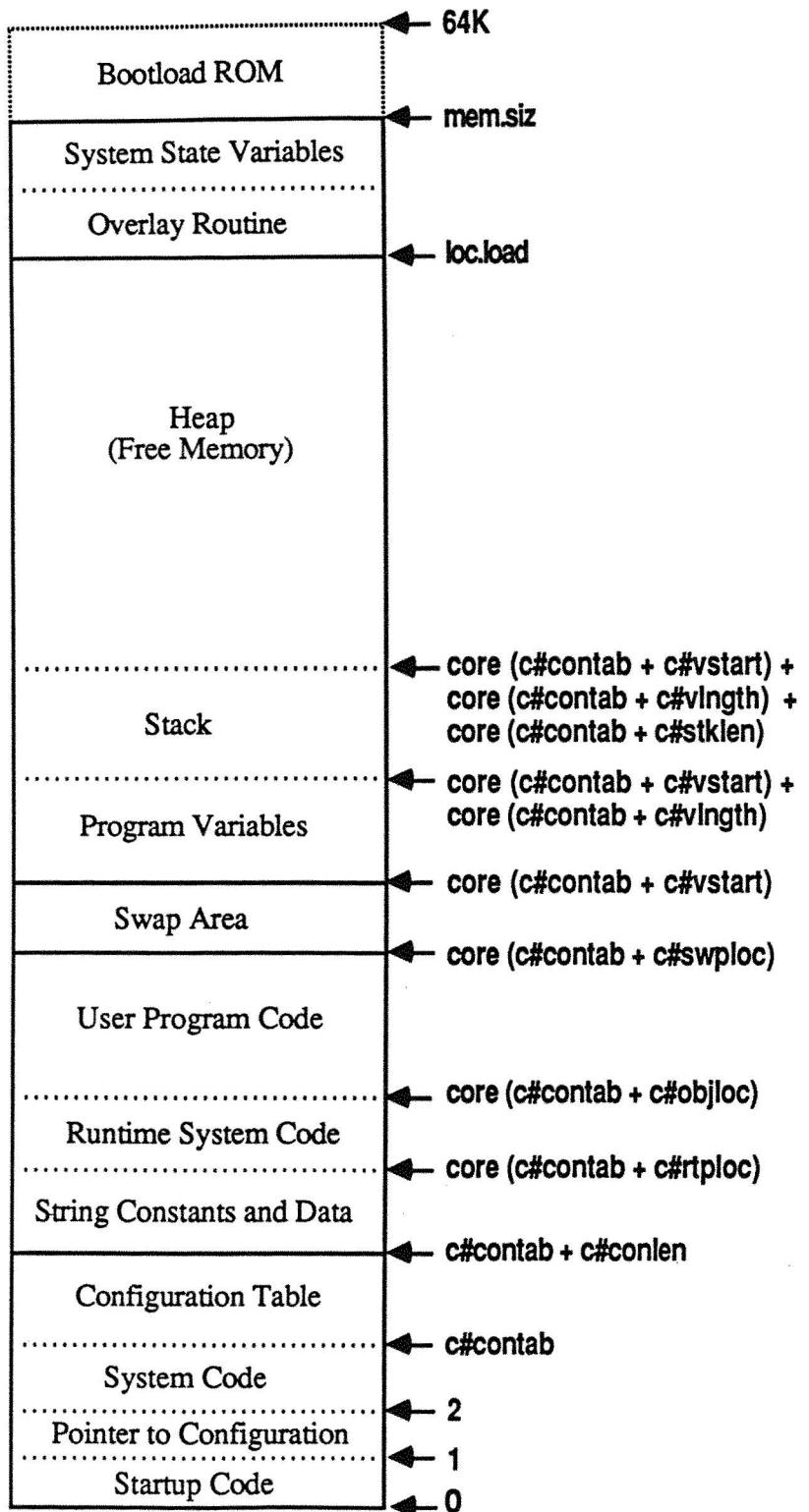
All procedures that are declared to be swapping are stored in external memory until they are needed by the program. When a program is compiled, the last section of the compiled code contains the swap file for the program. The swap file contains the code for all the swapping procedures, as well as a lookup table with the location of each swapping procedure (the swap lookup table).

When a program is run, the swap file is copied into external memory. A section of internal memory called the swap area is reserved to hold any procedure that is swapped into internal memory. The swap area is as large as the largest swapping procedure. Only one swapping procedure is resident in internal memory at any one time.

Memory Image: Internal

On the following page is a detailed diagram of internal memory while a program is executing. Located out to the right of each section are the system literals you would use to find each area during program execution. The pointer to the configuration table is found by using the literal C#CONTAB.

The word size of useable internal memory in your system is set up automatically in the variable MEM.SIZ when you insert :-XPL:SYSLITS into a program. For example, if you have the full 60K in your system, MEM.SIZ will be 61440 words (60*1024).



The upper part of useable memory is reserved for the system state variables and the overlay routine. The system state variables contain information about the current file, the current system and user catalogs, and other information that needs to be preserved when overlaying from one program to another. The literals for accessing these system variables are in SYSLITS.

The internal memory not explicitly used by the program (the heap) can be used by the user program. Access to this memory is faster than access to external memory, so the heap is often used as a transfer buffer for copying files from one place to another or as a data storage area when time consuming operations are being performed. The following program segment figures out how much free memory is available and uses that area as a buffer.

```
insert ':-xpl:syslits'; /* get the system literals */

dcl free_start fixed; /* start of free memory */
dcl free_end   fixed; /* end   of free memory */
...
/* The start of the heap is found by starting at the program
variable area, adding the length of the variable area to
that, then adding the length of the stack. */
free_start = core (c#contab + c#vstart) +
            core (c#contab + c#vlength) +
            core (c#contab + c#stklen);

free_end   = loc.load; /* end of free memory */

total_free = free_end - free_start; /* word length of heap */
sectors = shr (total_free, 8); /* sector length of heap */

/* Recompute the word length of heap so that it is a
multiple of a sector boundary (for using it with READDATA
and WRITEDATA) */

total_free = shl (sectors, 8);

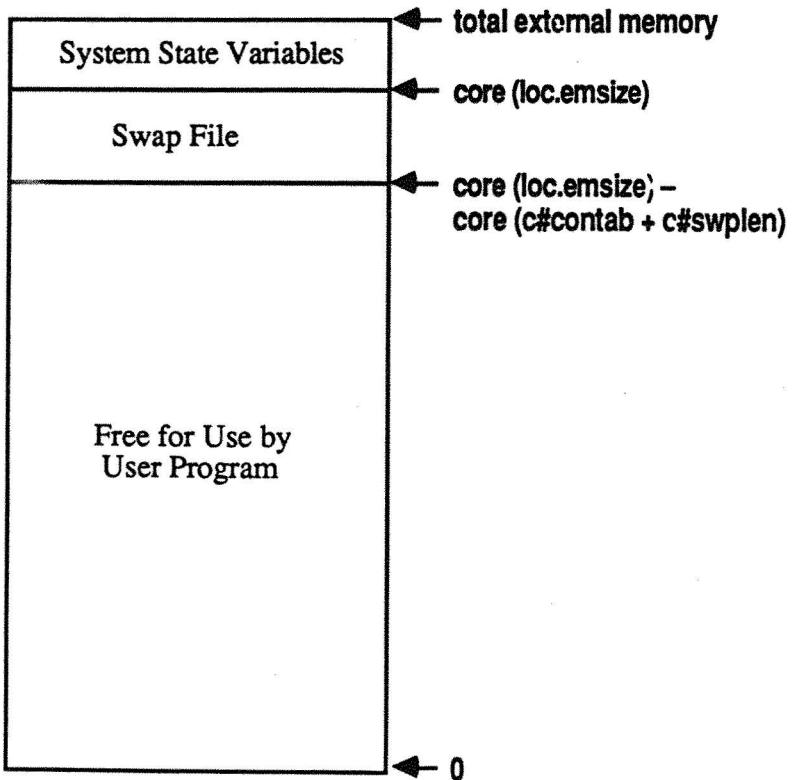
/* The following call to READDATA uses the LOCATION function
to read data from disk into free memory. */

call readdata (ms_sec, ls_sec, loc (free_start), total_free);
```

Memory Image: External

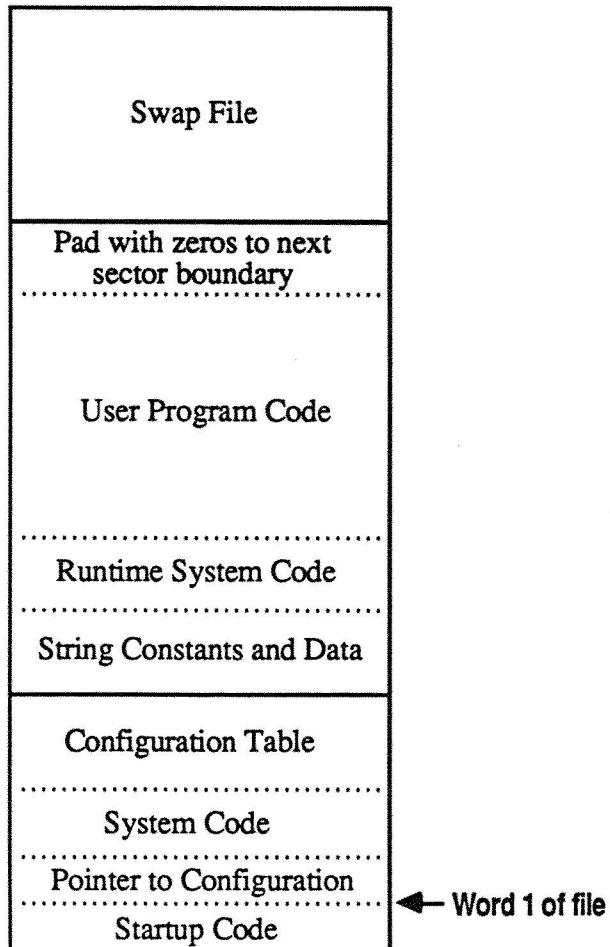
The diagram below shows the layout of external memory while a program is executing. As with internal memory, a small area at the end is reserved for system state variables. Unlike internal memory, where locations and lengths are specified in words, external memory is divided into sectors. The literal LOC.EMSIZEx is used to find the number of useable sectors of external memory.

All of external memory except for the state variable area and the swap file is free for user applications, starting at sector zero. There are several built-in routines that are provided for reading from and writing to external memory. See the appendix "Built-in Functions" for descriptions of how to use these routines.



Disk Image

All compiled programs on disk are stored in a particular format. This format is similar to the structure of internal memory while a program is running. The following diagram illustrates this format:



The configuration table for a program can be examined by looking at the first sector of the program. Currently the configuration pointer for a program is located in the word at address one (check SYSLITS to make certain this is the case for your version of software). By using this pointer and the rest of the configuration literals in SYSLITS, information about the memory requirements of the program can be obtained. The literal C#OBJLEN can be used to get the length of the code up to the end of the user program code.

The following program segment reads the first sector of a program on disk, then uses SYSLITS to find out the length of the swap file for that program.

```
insert ':-xpl:syslits';

dcl buf (255)      fixed; /* buffer for first sector of file */
dcl contab        pointer; /* pointer to configuration table */
dcl swap_sectors fixed; /* number of sectors for swap file */
...
/* read the first sector of the program saved on disk */
call readdata (f#ms_sector, f#ls_sector, buf, 256);

contab = buf (1); /* get pointer to the config table */

/* The sector length of the swap file is located in a word
   offset defined by C#SWPLEN. So by adding C#SWPLEN to the
   start of the configuration table, we find the length of
   the swap table. */

swap_sectors = buf (contab + c#swplen);
```


Appendix G - D4567 Operation

The Hardware Multiply/Divide Unit (D4567) can be used with the READ function and the WRITE statement to perform multiplications and divisions of unsigned integers at a higher rate than is possible with the multiply (*) and divide (/) operators. This is possible because there is no sign correction that is required when using unsigned integers, while the sign correction routine is automatically invoked with * and /. Of course, this feature is only available when the optional Hardware Multiply/Divide Unit is connected to the computer.

The computer accesses the Multiply/Divide Unit by using READ function and the WRITE statement specifying devices 4, 5, 6, and 7. For programming purposes, the Hardware Multiply/Divide Unit can be viewed as two sixteen bit registers that can be loaded, read, and operated on by the computer. These two registers are called the A register and the B register and each can be read or written by the computer specifying device addresses 4 and 5, respectively. The A register (device 4) is cleared whenever the B register (device 5) is loaded in order to speed up both the multiplication and division operations. The advantage of this feature will become evident shortly.

Device addresses 6 and 7 are used to issue commands to the unit. Writing a fixed point number to device 6 directs the multiplier section to multiply the unsigned 16-bit number in the B register (device 5) by the unsigned 16-bit number that is being written to device 6. An unsigned 32-bit product is first formed and then added to the contents of the A register (device 4). The upper 16 bits of the result can then be accessed by reading the A register (device 4), while the lower 16 bits of the result are available in the B register (device 5).

If the programmer is only using the lower 16 bits of the result (device 5), then the computed number is correct for both signed and unsigned integers as long as the two operands were within range. No sign correction is required in this case.

A division is performed by first loading a 32-bit divisor into the A register (MS word) and B register (LS word), then writing a fixed point number to device 7. An integer unsigned division is performed in 1.8 microseconds producing a quotient in the B register (device 5) and a remainder in the A register (device 4). The following program segments demonstrate the use of the Multiply/Divide Unit.

```
declare (i, j) fixed;

write (5) = i; /* load B register and clear A */
write (4) = 2; /* load A register with addend */
write (6) = j; /* multiply I times J and add 2 */
i = read (4); /* upper 16 bits in A register */
j = read (5); /* lower 16 bits in B register */

write (5) = j; /* for divide, always load lower bits first */
write (4) = i; /* load upper 16 bits of divisor */
write (7) = 100; /* write dividend to device 7 */
i = read (5); /* quotient in the B register */
j = read (4); /* remainder in the A register */
```

Warning: There is a limitation in the D4567 that requires the programmer to read the result out of register B (device 5) between commands, whether the result is needed or not. The following sequence is illegal and will produce erroneous results:

```
/* evaluate i*7/12 in full 32-bit resolution */

write (5) = i;
write (6) = 7;
write (7) = 12;
i = read (5); /* pick up result */
```

The correct sequence is:

```
write (5) = i;
write (6) = 7;
i = read (5); /* this must be here !! */
write (7) = 12;
i = read (5); /* pick up result */
```

INDEX

\$D compile-time switch, 135
\$M compile-time switch, 135
% (fractional multiply), 25, 26
* (multiplication), 25, 26
+ (addition), 25
+ (unary plus), 25
- (subtraction), 25
- (unary minus), 25
-SYMTAB- file, 135
-XPL programming library, 101, 136, 163
.WORK file, 148
/ (division), 25, 26
32-bit precision, 26, 27, 171
< (less than), 29
<= (less than or equal), 29
<> (not equal), 29
= (equal), 29
> (greater than), 29
>= (greater than or equal), 29
^= (not equal), 29
~= (not equal), 29

ABS function, 106
Absolute value (see ABS)
Actual parameters, 56
Addition, 25
ADDR function, 50, 106
 with LOCATION, 51
Aliases for operators, 141
AND, 31
 bit operator, 32
 logical operator, 31
 parentheses with, 36
 symbols for, 32
 truth table of, 34
Anti-logarithm, 112
Appendices, 99
Arccosine, 107
Arcsine, 107
Arctangent (see ATN)
Arguments (see Parameters)
Arithmetic functions, 104
 ABS, 106
 ATN, 107
 COS, 111
 EXP, 112
 INT, 117
 LOG, 119
 SIN, 129
 SQR, 129
 TAN, 132
Arithmetic operators, 25
 summary of, 141
Array index (see Subscripts)
Arrays, 45

BEGIN statement, 37
Binary arithmetic operators, 25
Bit manipulation, 32, 104
 AND, 32
 NOT, 32
 operators, 32
 OR, 32
 ROT, 33, 125
 SHL, 33, 127
 SHR, 33, 128
 summary of, 142
 XOR, 32
Bit operators, 32
 summary of, 142
Black box, 78
Block manipulation, 104
 BLOCKMOVE, 108
 BLOCKSET, 108
 EXPORT, 112
 EXTSET, 114
 IMPORT, 116
Block structure, 64

BLOCKMOVE function, 108
 BLOCKSET function, 108
 Boolean data type, 14

- FALSE, 14
- logical operators, 31
- relational operators, 29
- TRUE, 14
- use of, 63

 BREAK (see WHEN BREAK)
 Built-in functions, 101

- alphabetic list, 106
- categorical list, 104

 BYTE function, 48, 109

 CALL statement, 55, 56
 CASE statement (see DO CASE)
 CHARACTER, 20
 Character output, 20
 Character strings (see Strings)
 CHR, 20
 Combining expressions, 31
 Comments, 9, 53, 54, 139
 Compilation control, 135

- compile-time options, 135
- CONFIGURATION, 136
- EOF symbol, 137
- PDL, 137
- RAM, 137

 Compile-time options, 135
 Compiler, 147

- XPL compatibility, 163
- .WORK file, 148
- efficiency of, 147
- Pass 1 errors, 148
- Pass 2 errors, 154
- Pass 3 errors, 155
- structure of, 147
- types of errors, 147

 Compound statement, 37, 143
 Conditional execution (see IF)
 CONFIGUR program, 163
 Configuration of Monitor, 136
 CONFIGURATION statement, 136
 Configuration table, 164, 169
 Constants, 14

- data list, 49
- fixed point, 14
- floating point, 15
- hexadecimal, 14
- octal, 14
- precomputation of, 25, 28
- string, 15
- summary of, 139

 using literals for, 18, 89
 Control-Q, 20
 Control-S, 20
 Conversions, 35

- numeric, 35
- of parameter types, 60

 CORE array, 50, 110
 COS function, 111
 Cosine (see COS)
 Current catalog, 76
 Current device, 126
 Custom interrupts, 96

 D3 Real Time Clock, 95
 D16 Scientific Timer, 95
 D136 Real Time Clock, 95
 D140 Communications Processor, 96
 D4567 (see Hardware
 Multiply/Divide)
 Dangling ELSE clause, 39
 DATA declaration, 49
 Data types, 13

- boolean, 14
- conversion of, 35
- fixed point, 13
- floating point, 13
- pointer, 14
- summary of, 139

 DCL, 17
 Decimal point, 15
 DECLARE statement, 17
 Declarations, 17

- AUTOMATIC, 69
- arrays, 45
- arrays as parameters, 57
- data list, 49
- DECLARE statement, 17
- EXTERNAL, 81
- formal parameters, 56
- global, 53
- labels, 67
- literals, 18
- PUBLIC, 80
- procedures, 72
- STATIC, 69
- summary of, 143
- variables, 17

 Device directories, 101
 Devices, 94

- device numbers, 89
- device specifiers, 74
- storage devices, 101

 Directories, 101

DISABLE statement, 93
 Disk image of a program, 168
 Disk storage (see Storage device I/O)
 DISKERROR (see WHEN DISKERROR)
 Division, 25, 26
 DO CASE statement, 42
 DO loop, 40
 DO statement, 37
 DO WHILE loop, 39
 Dump option, 135
 Dynamic variables (see AUTOMATIC)

 ELSE clauses, 38
 ENABLE statement, 93
 End of file (see EOF)
 END statement, 37, 55, 79
 ENTER statement, 76
 asterisk with, 76
 with INSERT, 77
 with LIBRARY, 84
 EOF symbol, 137
 Example module, 86
 Example program, 10
 Exceptions, 93, 97
 summary of, 146
 Exclusive or (see XOR)
 EXIT function, 111
 EXP function, 112
 EXPORT function, 112
 Expression evaluation, 31
 Expressions, summary of, 141
 External memory, 163
 built-in functions for, 104
 description of, 163
 memory image, 167
 state variables, 167
 swap file, 164
 swapping procedures, 71
 EXTERNAL storage class, 81
 summary of, 140
 EXTREAD function, 113
 EXTSET function, 114
 EXTWRITE function, 115

 FALSE, 14
 FDIV (fractional divide), 25, 26
 FIND DEVICE function, 116
 Fixed point data type, 13
 constants, 14

 conversions, 35
 internal format, 159
 overflow, 25, 26, 29
 range of, 13
 Floating point data type, 13
 constants, 15
 conversions, 35
 internal format, 160
 overflow, 25, 27
 range of, 13
 storing data, 103
 Floppy disk (see Storage device I/O)
 Flow of control, 37
 summary of, 144
 Formal parameters, 56
 Forward reference procedures, 72
 Fractional divide, 25, 26
 Fractional multiply, 25, 26
 Free memory (see Heap)
 Functions, 62
 built-in, 101
 used in expressions, 63

 Global declarations, 53
 GOTO statement, 44, 61, 68

 Hand Operated Processor (see HOP)
 Hardware manipulation, 89
 assembly language, 91
 Hardware Multiply/Divide, 171
 interface devices, 89
 interrupt processing, 93
 READ, 89, 123
 WRITE, 90, 132
 Hardware Multiply/Divide, 171
 limitation of, 172
 speed of, 172
 with arithmetic operators, 27
 Heap (free memory), 166
 Hexadecimal constants, 14
 HOP, 90, 130

 Identifiers, 16
 summary of, 140
 IEQ, 30
 IF statement, 38
 IGE, 30
 IGT, 30
 ILE, 30
 ILT, 30

IMPORT function, 116
 INE, 30
 Infinite loop, 18, 40
 Initialization of variables,
 69, 80
 INPUT statement, 21
 INSERT statement, 75
 with ENTER, 77
 INT function, 35, 61, 117
 Interface devices, 89
 Internal formats, 159
 fixed point, 159
 floating point, 160
 strings, 161
 Internal memory, 50
 ADDR, 50
 conserving, 71
 CORE array, 50
 description of, 163
 getting a pointer to, 50
 LOCATION, 51
 memory image, 164
 overlay routine, 166
 reading from, 50
 state variables, 166
 swap area, 164
 swapping procedures, 71
 writing to, 50
 Internal representations, 159
 Interrupts, 93
 default state of, 93
 DISABLE, 93
 ENABLE, 93
 identifiers for, 94
 in WHEN, 94
 INVOKE, 97
 summary of, 146
 swapping procedures, 94
 WHEN statement, 93
 with PRINT, 20
 Introduction, 7
 INVOKE statement, 97
 Iterative DO loop, 40

 Keywords, 16

 Labels, 43, 44
 declaration of, 67
 Libraries, 73, 82
 existence at compile-time, 84
 LIBRARY statement, 82
 link map for, 135
 linking of, 82
 order of, 82
 referenced from modules, 84
 summary of, 145
 LIBRARY statement, 82
 with ENTER, 84
 Line numbered files, 75
 LINPUT statement, 22
 Link map, 135
 Linking, 79, 82
 LIT, 18
 LITERALLY, 18
 Literals, 18
 LOC, 51
 Local declarations, 54
 LOCATION function, 51, 118
 with ADDR, 51
 LOG function, 119
 Logarithm (see LOG)
 Logical operators, 31
 summary of, 142
 Loop variables, 40, 46
 Loops, 37
 DO WHILE, 39
 infinite, 40
 iterative DO, 40
 loop variables, 40, 46
 negative increments, 41

 Macros (see Literals)
 Memory, 50, 163
 built-in functions for, 104
 conserving, 71
 external memory, 101, 163
 internal memory, 101, 163
 layout of, 163
 pointers into, 50
 polyphonic, 101
 swapping procedures, 71
 MOD function (remainder), 26
 Modular programming (see
 Procedures)
 MODULE statement, 79
 Modules, 73, 78
 compilation of, 79
 END statement in, 79
 ENTER statement, 76
 example of, 86
 EXTERNAL storage class, 81
 INSERT statement, 75
 linking of, 79
 PUBLIC storage class, 80
 scope, 80, 81

summary of, 145
 with WHEN, 79
Modulus function (see MOD)
Multiplication, 25, 26

Natural anti-logarithm, 112
Natural logarithm, 119
Negation, 25
Negative loop increments, 41
Nesting, 55
 INSERT statements, 75
 comments, 53
 libraries, 82
 modules, 79
 procedure calls, 63
 program blocks, 68
 with AUTOMATIC, 69
NOT, 31
 bit operator, 32
 logical operator, 31
NULL, 14
Null statement, 42
Numeric conversions, 35, 60
Numeric input, 21

OCTAL, 21
 Octal constants, 14
 Octal output, 21
 One's complement, 32, 159
 Operating system, 163
Operators, 25
 aliases for, 141
 arithmetic, 25
 bit, 32
 logical, 31
 relational, 29
 summary of, 141
 unsigned relational, 30
Optical disk (see Storage device I/O)
OR, 31
 bit operator, 32
 logical operator, 31
 parentheses with, 36
 symbols for, 32
 truth table of, 34
Order of evaluation, 36
Overlay routine, 166

Parameters, 55, 56
 actual, 56

 formal, 56
 pass by reference, 59
 pass by value, 58
Parentheses, 36
Parity with LINPUT, 23
Pass 1 compiler errors, 148
Pass 2 compiler errors, 154
Pass 3 compiler errors, 155
Pass by reference, 59
Pass by value, 58
Pathnames, 74
PBYTE function, 48, 120
PDL statement, 69, 137
PL/I, 7
Pointer data type, 14, 50
Pointer functions, 105
 ADDR, 106
 CORE, 110
 LOCATION, 118
Polyphonic memory, 101
 built-in functions for, 104
POLYREAD function, 121
POLYWRITE function, 122
Precedence, 36
 summary of, 141
PRINT statement, 19
 CHARACTER, 20
 CHR, 20
 OCTAL, 21
 STRING, 21
Printers, 19
 directing program output to, 135
SEND statement, 19
PROC, 55
Procedures, 55
 actual parameters, 56
 CALL statement, 55
 definition of, 55
 description of, 53
 END statement in, 55
 EXTERNAL, 81
 formal parameters, 56
 forward reference, 72
 functions, 62
 GOTO statement, 61
 invoking, 55
 LOCATION, 51
 names of, 55
 parameters, 55
 pass by reference, 59
 pass by value, 58
 PUBLIC, 81
 recursive, 70
 RETURN statement, 61

RETURNS attribute, 62
 summary of, 145
 swapping, 71
 Program disk image, 168
 Program structure, 53, 54,
 68, 73
 Program style, 54, 68
 Program termination, 105
 EXIT, 111
 STOP, 130
 PUBLIC storage class, 80
 link map, 135
 restriction of, 80
 summary of, 140
 Push down stack, 61, 68,
 69, 137

 RAM statement, 137
 Random access memory (see RAM)
 RCVDCHARACTER function, 95, 96,
 123
 READ function, 89, 123, 171
 Read only memory (see ROM)
 READDATA function, 124
 Real variables (see Floating)
 RECURSIVE attribute, 70, 72
 Recursive procedures, 70
 forward reference of, 72
 with AUTOMATIC, 70
 Reference appendices, 99
 Registers, 91
 Relational operators, 29
 summary of, 141
 RELOC, 82
 Relocatable binaries (see
 Libraries)
 Remainder function (see MOD)
 Remote computer control, 19
 Reserved words (see Keywords)
 Restricted scope (see Scope)
 RETURN statement, 61
 RETURNS attribute, 62
 ROM, 137
 ROT function, 32, 33, 125
 Rotate (see ROT)

 Sample program, 10
 Scope, 64, 69
 definition of, 64
 in modules, 80, 81
 of labels, 67
 of variables, 69

 restricting, 37
 SCSI devices, 101
 SEND statement, 19
 CHARACTER, 20
 CHR, 20
 OCTAL, 21
 STRING, 21
 Sequence number table, 135
 SET CURDEV function, 126
 Shift left (see SHL)
 Shift right (see SHR)
 SHL function, 32, 33, 127
 SHR function, 32, 33, 128
 SIN function, 129
 Sine (see SIN)
 Speed, 27
 of arithmetic operations, 27,
 29
 of Hardware Multiply/Divide,
 27, 172
 of READ and WRITE, 89
 SQR function, 129
 Square root (see SQR)
 Statement labels, 43
 Statements, summary of, 143
 STATIC storage class, 69
 summary of, 140
 Statistics program, 10
 STOP statement, 130
 Storage classes, 69, 80
 AUTOMATIC, 69
 defaults, 69
 EXTERNAL, 81
 PUBLIC, 80
 STATIC, 69
 summary of, 140
 Storage device I/O, 101, 104
 device numbers, 102
 EXTREAD, 113
 EXTWRITE, 115
 floating point data, 103
 POLYREAD, 121
 POLYWRITE, 122
 READDATA, 124
 SCSI devices, 101
 WRITEDATA, 133
 STRING, 21
 String constants, 15
 String functions, 104
 BYTE, 48, 109
 PBYTE, 48, 120
 Strings, 47
 BYTE, 48, 109
 input of, 22

internal format, 161
 length of, 47
 output of, 21
 PBYTE, 48, 120
 reading from, 48
 writing to, 48
 Subscripts, 45
 out of bounds, 46, 57
 Subtraction, 25
 Swap area, 164
 SWAP attribute, 71, 72
 Swap file, 164
 SWAPINIT function, 131
 Swapping procedures, 71
 forward reference of, 72
 saving memory space, 71
 speed of, 71
 swap area, 164
 swap file, 164
 SWAPINIT function, 131
 with WHEN, 94
 Symbol table, 135
 SYSLITS file, 136, 163
 System literals, 163
 System state variables, 166, 167
 Systems programming, 105
 FIND DEVICE, 116
 RCVDCHARACTER, 123
 SET CURDEV, 126
 SWAPINIT, 131

 Table of contents, 3
 TAN function, 132
 Tangent (see TAN)
 Tape drive (see Storage device I/O)
 Terminal control sequences, 20
 Terminal input, 21
 INPUT, 21
 LINPUT, 22
 numeric, 21
 strings, 22
 summary of, 146
 Terminal interrupts, 95
 Terminal output, 19
 PRINT, 19
 SEND, 19
 characters, 20
 numeric, 19
 octal format, 21
 strings, 21
 summary of, 146

 Treeenames, 74, 77
 Trigonometric functions (see Arithmetic)
 TRUE, 14
 Two's complement, 159

 Unary arithmetic operators, 25
 Unresolved reference, 81
 Unsigned integers, 13, 30
 Unsigned relational operators, 30

 Variables, 13
 declarations, 17
 identifiers, 16
 initialization of, 69, 80
 storage classes of, 69, 80, 81
 Vectors (see Arrays)

 WHEN BDB14INT, 96
 WHEN BDB15INT, 96
 WHEN BREAK, 97
 WHEN D03INT, 95
 WHEN D16INT, 95
 WHEN D136INT, 95
 WHEN D140INT, 96
 WHEN DISKERROR, 97
 WHEN TTIINT, 95
 WHEN TTOINT, 95
 WHEN statement, 93
 example of, 96
 in modules, 79
 INVOKE, 97
 return from, 93
 swapping procedures, 94
 with GOTO, 94
 Winchester disk (see Storage device I/O)
 WRITE statememt, 90, 132, 171
 WRITEDATA function, 133

 XOR, 32
 bit operator, 32
 truth table of, 34

