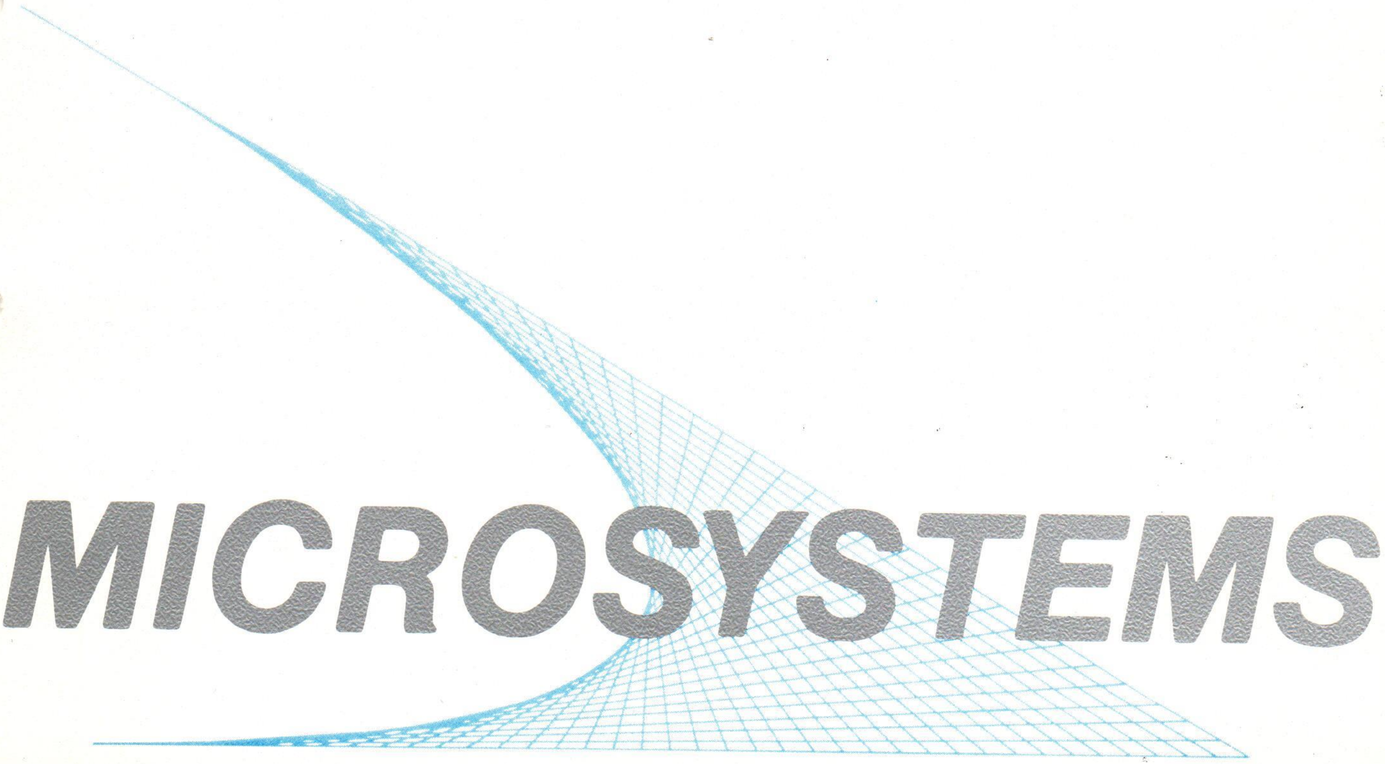
MOTOROLA

M6809XPASC/D1

M6809
Cross Pascal Compiler On EXORmacs
User's Manual



QUALITY • PEOPLE • PERFORMANCE

M6809XPASC/D1

MARCH 1983

M6809

CROSS PASCAL COMPILER ON EXORmacs

USER'S MANUAL

The information in this document has been carefully checked and is believed to be entirely reliable. However, no responsibility is assumed for inaccuracies. Furthermore, Motorola reserves the right to make changes to any products herein to improve reliability, function, or design. Motorola does not assume any liability arising out of the application or use of any product or circuit described herein; neither does it convey any license under its patent rights nor the rights of others.

EXORciser, EXORdisk, EXORmacs, EXORterm, MDOS, VERSAdos, and VMC 68/2 are trademarks of Motorola Inc.

LARK is a trademark of Control Data Corporation.

First Edition

TABLE OF CONTENTS

TABLE OF CONTENTS (cont'd)

TABLE OF CONTENTS (cont'd)

LIST OF ILLUSTRATIONS

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 SCOPE

The purpose of this manual is to assist users in developing M6809 Pascal programs, using the M6809 Cross Pascal Compiler on EXORmacs or VMC 68/2 Microcomputer System.

The manual presents general information on the operation of the compiler. It also provides general information on the linking of Pascal programs, assembly language programs, and appropriate runtime libraries into S-record-format modules, which may then be downloaded to the M6809 system and executed.

1.2 OPERATING ENVIRONMENT

- VERSAdos Operating System

1.3 MINIMUM HARDWARE REQUIREMENTS

1.3.1 EXORmacs Development System

- EXORmacs Chassis
- EXORterm 155 Display Console
- EXORdisk III Disk Drive Unit
- Model 703 printer
- 384K bytes of RAM

1.3.2 VMC 68/2 Microcomputer System

- VMC 68/2 System (which includes an 8-inch LARK disk drive unit, MLD-16, and 384K bytes of RAM)
- EXORterm 155 Display Console, or user-supplied dumb ASCII RS-232C terminal
- Model 703 printer or equivalent

1.4 RELATED PUBLICATIONS

You should be familiar with the following manuals, as pertinent to your system:

EXORdisk II/III Operating System (MDOS) User's Guide (M68MDOS3)

MC6809 Programming Manual (M6809PM)

Macro Assemblers Reference Manual (M68MASR)

VERSAdos System Facilities Reference Manual (M68KVSF)

M68000 CRT Text Editor User's Manual (M68KEDIT)

8-Bit Cross Linkage Editor on EXORmacs User's Guide (M68XLINK)

Pascal Programming Structures for Motorola Microprocessors (TB304)

1.5 GENERAL STRUCTURE OF PASCAL COMPILER

As shown in Figure 1-1, the Pascal compiler consists of two phases. Phase 1 processes a source program and produces a source listing and error messages, as well as an intermediate code file. This intermediate code is then input to Phase 2, which creates a pseudo-assembly listing as well as a relocatable 6809 object file. The object file is then combined with needed routines from the runtime library by the cross linkage editor, and a transportable S-record module results.

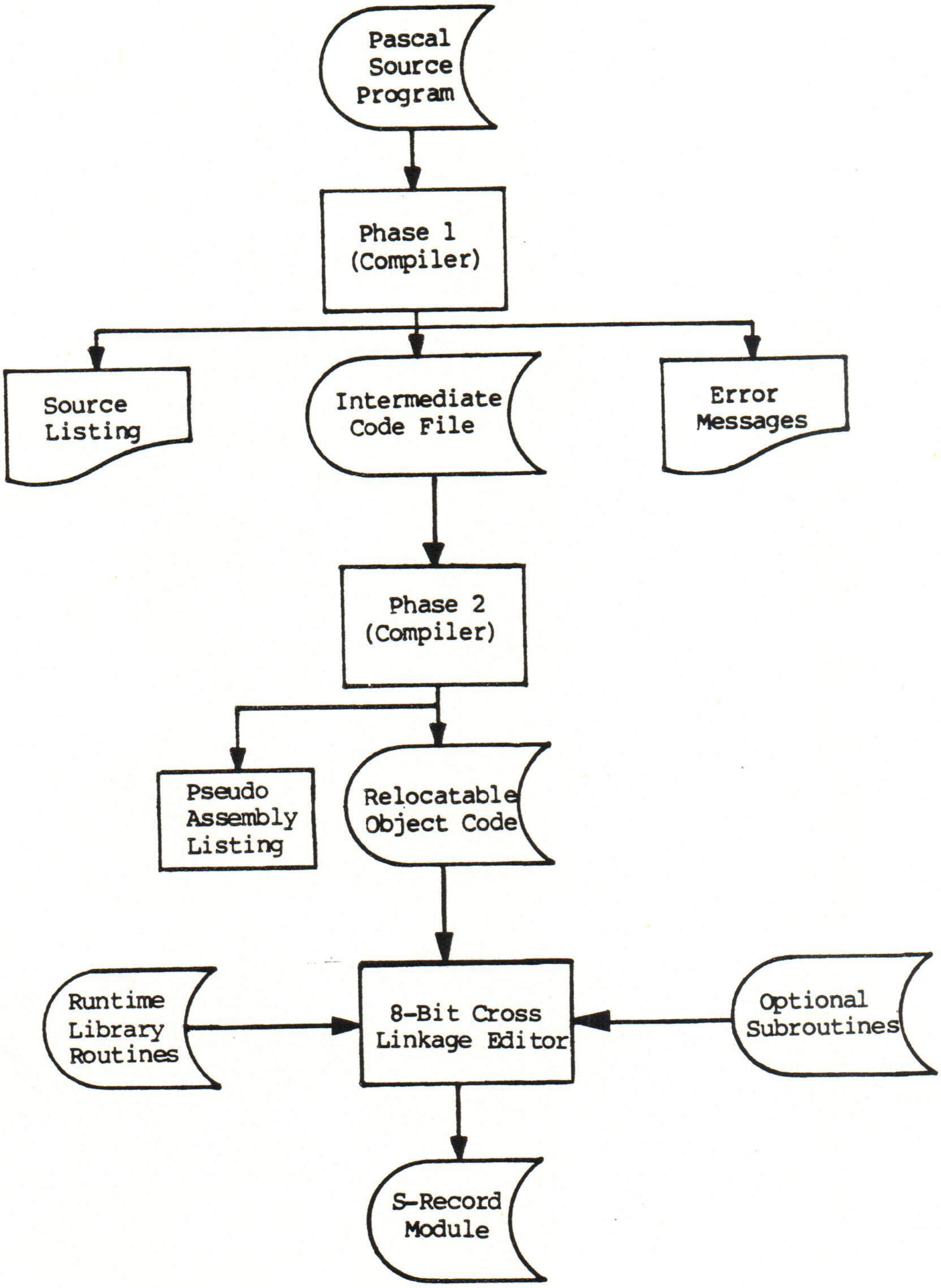


FIGURE 1-1. Pascal Program Processing

CHAPTER 2

PREPARING A PROGRAM

2.1 GENERAL

The Pascal source program is usually created by the CRT Text Editor, using the language constructs as defined in the publication entitled "Pascal Programming Structures for Motorola Microprocessors". The source program file is stored on disk.

2.2 THE OPTION COMMENT

The Pascal source program may include options (Table 2-1) that affect the Phase 1 source and object output, options that control runtime checks, and miscellaneous options. The options are named in an option comment, which is enclosed within braces (i.e., { }) or within the symbol pairs (* and *). A dollar sign immediately follows the left brace or left symbol pair to identify the comment as an option comment. The format of the option comment is:

```
{$Xs,...,Xs}

     or

(*$Xs,...,Xs*)
```

where X is a capital letter corresponding to one of the options shown in Table 2-1. The s is either a plus (+), a minus (-), or an equal (=) sign. A plus assigns a TRUE value enabling the option; a minus assigns a FALSE value disabling the option; and an equal assigns a non-Boolean value to the option. Table 2-1 shows the default value assigned to each option at the beginning of the program.

One or more options, separated by commas with no intervening spaces, may be specified in the comment. The option comment may appear at any point in a program at which a comment is normally allowed.

2.3 BACKUS-NAUR FORM

The syntax description in this manual uses a syntax known as Backus-Naur Form (BNF). A brief description of pertinent symbols is included below. These symbols and their meanings are:

< >	The angular brackets enclose a symbol, known as a syntactic variable, that is replaced by one of a class of symbols it represents.
|	This symbol indicates that a choice is to be made. One of several symbols separated by this symbol should be selected.
[ ]	Square brackets enclose a symbol that is optional. The enclosed symbol may occur zero or one time.
[ ]...	Square brackets followed by periods enclose a symbol that is optional/repetitive. It may appear zero or more times.

Operator inputs are to be terminated by a carriage return.

TABLE 2-1. Source Program Options

	DEFAULT	
OPTION	VALUE	MEANING
A=x	A=2	where x has the value 1, 2, or 4. It specifies the number of bytes to be used for integer arithmetic. In an integer operation, both operands are converted to the largest of the following: (1) the size of the lefthand operand, (2) the size of the righthand operand, and (3) the size specified by the A= option. For example: {$A=2} il:=il+jl-kl; The 1-byte integer variables il and jl are both extended to 2 bytes prior to calculating their sum. The 1-byte integer variable kl is extended to 2 bytes before subtracting it from the intermediate sum. The final answer is truncated to 1 byte and then assigned to il. Integer constants will always be treated as if they had at least the size specified by the A= option. For {$A=2}, the constant 1 is treated as if it were 2 bytes in size.
C-	C+	Generate an intermediate code file during Phase 1. If C- is specified, an intermediate code file is not generated. (Eliminating this file reduces the time necessary to generate the listing and any errors.)
D	D-	This combines the K and R options to (1) generate code to perform runtime checks which verify that array indices and subrange type variables are in range, and (2) include statement numbers in the object code. The numbers relate to executable units and are found on the source listing. If an error condition occurs at runtime, the current statement number is reported.
E		Page eject. Whenever this option is encountered in the source program, the Phase 1 listing will advance to the top of the next page. (This option has no default value, and no plus or minus sign.)
F=<fn>		where <fn> is the name of a VERSAdos file containing Pascal source code. The option allows the compiler to process a source file as if another file were included inline in the source. Immediately after the line which contains the F= option, Phase 1 will obtain its input from the specified file. When the end of the "included file" is encountered, Phase 1 will return to the original input file. The remainder of the line which follows the comment containing the F= option must not contain any more text. Any source text found there is ignored. An option comment may contain only one F= option, and the included file may not contain any F= options at all. Include files may not be nested.

TABLE 2-1. Source Program Options (cont'd)

OPTION	DEFAULT VALUE	MEANING
		A typical use of include files is to contain all the global declarations for a set of subprogram modules in one. For example: {$F=GLOBALS.SA,L- Include globals and turn listing off}
K	K-	Include statement numbers in the object code. The statement numbers relate to executable units and are found on the source listing. If an error condition occurs at runtime, the current statement number is reported. NOTE: Each statement number requires at least three additional bytes of object code each time the executable unit counter is updated.
L	L+	Generate a source listing on the Phase 1 listing file or device. If L- is specified, a listing is generated only of lines containing errors and the associated error messages.
O	O-	Enter source statements as comments in the Phase 2 input.
P	P-	Include statement numbers in the object code, but only at function/procedure entry and exit points. If an error condition occurs at runtime, the number displayed will indicate in which function/procedure the error was detected.
R	R-	Generate code to perform runtime checks which verify that array subscripts and subrange type variables are in range. NOTE: Each range check requires seven additional bytes of object code.
W	W-	Generate a warning during Phase 1 processing if non-standard Pascal features are used. Standard Pascal comprises only the language features proposed by Jensen and Wirth.

2.3.1 File Name Format

In the command line descriptions, some syntactic variables are replaced by disk file names. A VERSAdos disk file name consists of six fields:

<volume name>:<user number>.<catalog>.<file name>.<extension>(<protect key>)

If any of these fields is omitted, the system will fill them in with default values, as follows:

a. If <volume name> is omitted, the volume specified at system logon, or in the last session control USE command, or specified in the first command parameter (overrides defaults) will be used.

b. If <user number> is not supplied, the user number supplied at logon, or in the last USE command, or specified in the first command parameter (overrides defaults) is the default.

c. If a default catalog has been supplied at logon, or with the USE command, or specified in the first command parameter (overrides defaults), and <catalog> is not specified, then the default catalog will be used. Any <catalog> specified will override the default catalog. If a default catalog has been specified and a null catalog is required, entering a '&' (ampersand) as the <catalog> will produce a null catalog. If a default catalog has not been specified and <catalog> is omitted, then a null catalog will be used.

d. If <user number> and/or <catalog> is being specified, <file name> may be omitted and will default to the file name specified by the first command parameter.

e. If the <extension> field (or "suffix") is not supplied, a default will be supplied. See command line descriptions in Chapters 3 and 4 for default extensions.

f. If <protect key> -- 2- to 4-character (AA-PP) access protection code -- is not specified, it defaults to PPPP (any user may read or write to the files). If only 2 characters are specified, they are assumed to be the read code, and the write code defaults to public write (PP).

The following file names are equivalent if the last USE command specified VOL1 and the user logged on as user 3:

```
VOL1:3..TESTPROG.SA
3..TESTPROG.SA
TESTPROG.SA
TESTPROG        (if default extension is .SA)
```

2.4 CROSS PASCAL ENHANCEMENTS

Compared to the current resident M6809 Pascal on the EXORciser development system, the cross Pascal compiler offers a number of enhancements.

2.4.1 One- and Four-Byte Integers

One- and four-byte integers are now supported. Any integer variable declared in the subrange -128..127 will be allocated one byte of storage. Variables of an enumerated type with 128 or fewer enumerated constants will be allocated one byte of storage. Any integer variable declared outside of the subrange -32768..32767 will be allocated four bytes of storage. An integer operation may utilize mixed sizes of integer variables (integer operands are automatically converted to the size of the larger). Array indexing operations will be converted to 2-byte integers as well as CASE statement indices.

The predefined type INTEGER will default to two-byte integers (due to the explosion of code size which results if 4-byte integers are used). The predefined constant MAXINT, however, is defined to be the largest 4-byte integer, 2147483647. For example:

```
VAR
    il:   -128..127;          {allocated 1 byte of storage}
    i,i2: integer;            {allocated 2 bytes of storage}
    i4:-maxint..maxint;       {allocated 4 bytes of storage}

    color: (red, orange, yellow, green, blue, indigo, violet);
                               {allocated 1 byte of storage}

BEGIN
    i := il + i2 + i4;        {il is extended to 2 bytes and then added to i2.
                               The sum is then extended to 4 bytes and added to
                               i4. The result is then truncated to 2 bytes and
                               stored in i}
```

2.4.2 A Option

If doing 1-byte arithmetic, it is very easy to overflow. For example, the expression il + jl - kl may yield a legitimate 1-byte value, but the intermediate result il + jl may overflow into two bytes and cause an erroneous final answer. Therefore, the A= option was implemented so that intermediate overflow may be prevented. The A= option is a comment option (in that it can only be specified in an option comment in the source) which specifies the number of bytes to be used for integer arithmetic. The possible values are A=1, A=2, or A=4 for 1-, 2-, or 4-byte arithmetic, respectively. Essentially, this means that when doing an integer operation, both operands are converted to the largest of the following: (1) the size of the left-hand operation; (2) the size of the right-hand operation; and (3) the size specified by the A= option. For example:

```
    {$A=2}
    il := il + jl - kl;
```

The 1-byte integer variables il and jl are both extended to two bytes prior to calculating their sum. The 1-byte integer variable kl is extended to two bytes before subtracting it from the intermediate sum. The final answer is truncated to one byte and then assigned to il.

Integer constants will always be treated as if they had at least the size specified by the A= option. For {$A=2}, the constant 1 is treated as if it were two bytes in size.

The default arithmetic size is two bytes, {$A=2}.

2.4.3 F Option

The F= option allows the compiler to process a source file as if another file were included inline in the source. The F= option is a comment option and has the following format:

```
{$F=<filename>}
```

Immediately after the line which contains the F= option, Phase 1 will obtain its input from the specified file. When the end of the "included file" is encountered, Phase 1 will return to the original input file.

The remainder of the line which follows the comment containing the F= option must not contain any more text. Any source text found there is ignored. An option comment may contain only one F= option; the included file may not contain any F= options at all. Include files may not be nested.

A typical use of include files is to contain all the global declarations for a set of subprogram modules in one. For example:

```
{$F=GLOBALS.SA,L-  Include globals and turn listing off}
```

2.4.4 String Functions

The string functions as described in Pascal Programming Structures for Motorola Microprocessors are now implemented. Recall that on the M6809, strings are implemented as a sequence of up to 254 ASCII characters preceded by a current length byte. The string functions differ slightly from the string functions as implemented for the M68000 Resident Pascal Compiler with regard to error handling and boundary conditions. The following remarks describe the M6809 implementation.

CONCAT(s1,s2,...,sn) Return a string which is the concatenation of the strings s1, s2, ..., sn. If the result is longer than 254 characters, the string is truncated with no error indication given.

COPY(s, j, len) Return a substring of s, starting at position j, and of length len. If j is outside the range 1..current_length, then the null string is returned. If beginning at position j, there are not len characters left in the string, the remainder of the string is returned.

DELETE(s, j, len) Return a string which is the string s with the characters starting at position j, for a length len, deleted. If j is outside the range 1..current_length, then the string s is returned. If beginning at position j, there are not len characters left in the string, the remainder of the string is deleted.

INSERT(s1, s2, j) Return s2 with s1 inserted into it starting at position j. If j is 0, return s2 unmodified. If j is greater than the current size of s2, concatenate s1 onto the end of s2. If the result is longer than 254 characters, truncate the string.

LENGTH(s) Return the current length of the string s.

POS(s1,s2) Return the position of the first occurrence of the string s2 in the string s1. Return 0 if s2 does not occur within s1.

2.4.5 Real Numbers

While Phase 1 can process real numbers and floating point operations, and Phase 2 can generate runtime library calls to perform the floating point calculations, there are NO floating point routines currently implemented in the runtime library. The user who wishes to perform floating point calculations will have to supply a set of routines compatible with the calling sequences utilized by Phase 2 of the compiler. Chapter 8 on floating point routines describes the required interface.

CHAPTER 3

COMPILING A PROGRAM

3.1 GENERAL

A Pascal source program prepared by the user must be processed by the compiler to produce a relocatable object file, from which a transportable S-record module can be created.

The M6809 Pascal Compiler, referred to as the "compiler", consists of two programs. The first of these, Phase 1 of the compiler, is invoked using the XPAS09 command. When the first phase completes, the user activates Phase 2 via the XPAS092 command. The output produced by Phase 2 must be processed by the Cross Linkage Editor, described in Chapter 4 of this manual. The resulting S-record module is ready to be downloaded and run.

3.2 COMPILER PHASE 1

Phase 1 processes a Pascal source program, checking the syntax of each statement it encounters. If any errors are detected, they are brought to the attention of the user. These errors should be eliminated by editing the source program to correct illegal statements. Phase 1 should again be invoked to compile the modified program. When no errors are reported, Phase 1 processing is complete.

3.2.1 Phase 1 Output

Phase 1 of the compiler produces two types of output. First, it generates an intermediate file which is used to produce the relocatable object file from Phase 2. This file is of no value if errors were detected during Phase 1 processing.

Second, it produces an optional listing of the source program containing error codes along with other useful information. When an error is detected, a line is added to the program listing containing the phrase "**ERROR --" followed by the line number of a previous error, or "0" if this is the first error. Also on this line appears an error code positioned beneath the symbol that was being processed when the error was discovered.

Each line of the source listing file contains the following fields:

LINE Source program line number. Up to five digits may appear in this field.

LOC LOC stands for location. If enclosed in parentheses, this field contains the offset in the data section of the first variable declared in this statement; otherwise, this field contains an executable unit number, roughly corresponding to a statement number. If an error condition occurs while the program is running and a debug option (D, K, or P) was selected, the executable unit number of the statement being processed will be reported to indicate the point of failure.

LVL — LVL stands for level. Level numbers indicate the static structure of a program. The main program is at level 0. A level 1 procedure is contained in the main program and in no other procedure. A level n procedure is contained by procedures at level 0 through n-1. Level numbers are useful when determining the scope of variables or procedures.

B — B is an abbreviation for block beginner. A block beginner is one of the following symbols: BEGIN, REPEAT, or CASE. When one of these keywords is encountered, the B level is incremented. If multiple keywords that increase the B level occur on one line, the level corresponding to the first beginner is printed.

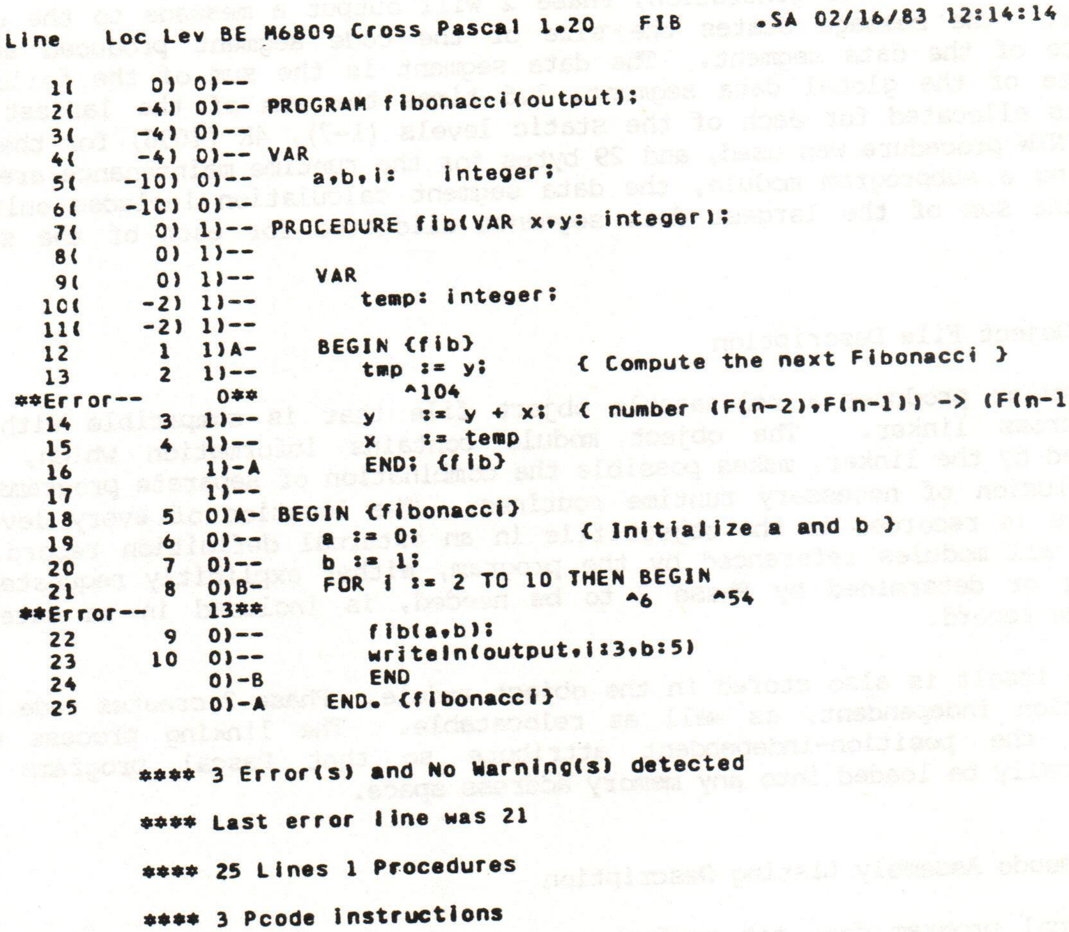
E — E stands for block terminator. A block terminator is either of the symbols: END or UNTIL. An END will match either an earlier BEGIN or a previous CASE symbol. An UNTIL is always associated with an earlier REPEAT. The E level is decremented when a block terminator is processed. If multiple block terminators are encountered in a line, the level of the last block terminator is printed.

Block levels are described by increasing letters of the alphabet. If a block beginner does not appear in a line, its B field contains a dash (-); if no block terminator is found on a line, its E field is also a dash. The B and E fields enable the user to quickly determine the block structure of a program. A common error is to fail to provide a matching block terminator for each block beginner. Often an examination of these fields will pinpoint the location of the error.

The remaining field contains a copy of the source statement, truncated to the current line length. No automatic formatting of source statements is performed.

At the end of the listing, a summary of the compilation is provided. A count of syntax errors, warnings, lines of source, procedures, and P-codes (intermediate code instructions) is given. If any errors or warnings occurred, the line number of the last error is listed.

An example of a source program listing containing three errors is shown in Figure 3-1. This figure shows how lines containing errors are chained together and also illustrates the other fields described above.



```
Line   Loc Lev BE M6809 Cross Pascal 1.20    FIB    .SA 02/16/83 12:14:14

   1(      0) 0)--
   2(     -4) 0)-- PROGRAM fibonacci(output);
   3(     -4) 0)--
   4(     -4) 0)-- VAR
   5(    -10) 0)--    a,b,i:    integer;
   6(    -10) 0)--
   7(      0) 1)-- PROCEDURE fib(VAR x,y: integer);
   8(      0) 1)--
   9(      0) 1)--     VAR
  10(     -2) 1)--        temp: integer;
  11(     -2) 1)--
  12       1  1)A-     BEGIN {fib}
  13       2  1)--        tmp := y;        { Compute the next Fibonacci }
                              ^104
 **Error--      0**
  14       3  1)--        y    := y + x;  { number (F(n-2),F(n-1)) -> (F(n-1),F(n)) }
  15       4  1)--        x    := temp
  16          1)-A        END; {fib}
  17          1)--
  18       5  0)A- BEGIN {fibonacci}
  19       6  0)--     a := 0;             { Initialize a and b }
  20       7  0)--     b := 1;
  21       8  0)B-     FOR i := 2 TO 10 THEN BEGIN
                                         ^6    ^54
 **Error--     13**
  22       9  0)--        fib(a,b);
  23      10  0)--        writeln(output,i:3,b:5)
  24          0)-B        END
  25          0)-A     END. {fibonacci}

        **** 3 Error(s) and No Warning(s) detected

        **** Last error line was 21

        **** 25 Lines 1 Procedures

        **** 3 Pcode instructions
```

FIGURE 3-1. Pascal Listing with Errors

3.3 COMPILER PHASE 2

Phase 2 of the compiler processes the intermediate code produced by Phase 1 and generates an object module that can be linked to create an S-record module. This phase collects intermediate code until it encounters a store operation, a branch statement, or the destination of a branch statement. It then generates, in the form of a relocatable object module, the machine code equivalent of the corresponding group of intermediate instructions. One object module is generated for the entire input file.

As code is generated, optimization is performed. Intermediate code is scanned and, when possible, two or more consecutive instructions are replaced by an equivalent single intermediate code instruction. Code is created by individual code generators, one for each possible machine instruction. These communicate with each other to ensure that they collectively form instructions that are optimized in quantity, memory requirements, register usage, and number of memory references.

3.3.1 Phase 2 Output

Phase 2 of the compiler produces an object file and, optionally, a pseudo assembly listing. The listing is not needed normally, and is suppressed unless the user specifically requests it.

On completion of code generation, Phase 2 will output a message to the user's console. The message states the size of the code segment produced and an estimate of the data segment. The data segment is the sum of the following: the size of the global data segment, 1.5 times the sum of the largest data segments allocated for each of the static levels (1-7), 4K (4096) for the heap if the NEW procedure was used, and 29 bytes for the runtime maintenance area. If compiling a subprogram module, the data segment calculation includes only 1.5 times the sum of the largest data segments allocated for each of the static levels.

3.3.2 Object File Description

The compiler produces a relocatable object file that is compatible with the 8-bit cross linker. The object module contains information which, when extracted by the linker, makes possible the combination of separate programs and the inclusion of necessary runtime routines. The location of every level 1 procedure is recorded in the object file in an external definition record. A list of all modules referenced by the program, either explicitly requested by the user or determined by Phase 2 to be needed, is included in an external reference record.

The code itself is also stored in the object module. Phase 2 creates code that is position independent, as well as relocatable. The linking process will preserve the position-independent attribute so that Pascal programs may theoretically be loaded into any memory address space.

3.3.3 Pseudo Assembly Listing Description

If a Pascal program does not perform as expected, debugging may be necessary. The most convenient way to perform this activity is by including facilities in the program to inform the user of its progress, reporting the values of critical variables at appropriate times. Occasionally it might be desirable to conduct debugging of individual machine instructions rather than source statements. The pseudo assembly listing greatly facilitates this activity.

This listing contains the following information:

a. Pascal source statements are present if the O option was selected when Phase 1 processing was requested. To the right of the source statement appears a statement number that matches the statement number appearing at the beginning of each line of the Phase 1 listing. This makes it easy to find a specific source statement in the pseudo assembly listing.

b. Between source statements appears a representation of the code that was stored in the object file. This appears in a similar format to that which would be produced by an assembler. Machine code for instructions does not match exactly what was actually put into the object file, because fixups of instructions containing forward references and instructions requiring linkage for completion cannot be shown in final form.

c. An assembly language instruction equivalent to the machine code representation appears on the right side of the pseudo assembly listing. This code may serve as a basis for users desiring to modify code generated by Phase 2, but will not, in general, assemble correctly.

3.4 RUNNING PHASE 1

Phase 1 of the compiler is loaded and run in response to the following VERSAdos command line:

```
XPAS09  <source>[,[<intermediate>],[<list>]][;<options>]
```

where the syntactic variables are defined as follows:

source — the source file containing the Pascal program. If not specified, the filename extension will default to .SA. Multiple source files may be specified provided they are separated by slashes (/).

intermediate — the destination file which will contain the intermediate representation of the program. If not specified, the filename will default to the first <source> filename but with extension .PC.

list — the listing file or device. If not specified, the filename will default to the first <source> filename but with extension .PL.

options — various flags for controlling the generation of the intermediate representation or the listing. Each option is a single letter possibly preceded by a minus sign or followed by an equal sign and integer value. In the latter case, a comma is used to separate the integer value from subsequent options.

OPTION	DEFAULT	MEANING
C	C	Produce intermediate code. -C would suppress the intermediate file.
D	-D	Debug mode. Generate code to check for values and indices out of range. Also generate code to maintain the executable unit counters. Default (-D) is non-debug mode.
K	-K	Generate code to maintain the executable unit counters. Default (-K) is no counter maintenance.
L	L	Produce a source listing. -L would suppress the listing file (except for lines containing compile time errors.
O	-O	Include the source statements in the intermediate file for Phase 2 listing purposes. Default (-O) excludes source.
P	-P	Generate code to maintain the executable unit counters but only at procedure/ function entry and exit points. Default (-P) is no counter maintenance.

R	-R	Generate code to check for values and indices out of range (runtime checking). Default (-R) is no checking.
W	-W	Warn if non-standard Pascal features are used. Default (-W) is no warnings.
Z=n	Z=36	Set stack/heap (symbol table) size used by compiler to nK. Value of n must be at least 36 (the default value, 36K bytes). The default value will be adequate for compiling most programs. However, some larger programs may cause Phase 1 to abort with error 1008, 1010, or 1011. In such cases, Phase 1 should be executed with a larger Z= option.

In the following example, all of these command lines are equivalent:

```
XPAS09    TESTPROG;W
XPAS09    TESTPROG,TESTPROG,TESTPROG;W
XPAS09    TESTPROG.SA,TESTPROG.PC,TESTPROG.PL;W
XPAS09    TESTPROG,.PC,.PL;W
```

All of the above commands direct Phase 1 to process a source program contained in TESTPROG.SA and produce intermediate code in TESTPROG.PC and a listing in TESTPROG.PL. The option causes a warning at every occurrence of a non-standard Pascal feature.

A common form of the command is:

```
XPAS09    TESTPROG,,#;-L
```

This command compiles TESTPROG.SA, creates intermediate code in TESTPROG.PC, and displays only lines containing compile time errors and associated error messages on the console screen.

3.5 RUNNING PHASE 2

Phase 2 of the compiler is invoked with the following VERSAdos command:

```
XPAS092    <intermediate>[,[<object>][,<list>]][;<options>]
```

where the syntactic variables are defined as follows:

intermediate	the file containing the intermediate representation of the Pascal program. If not specified, the filename extension will default to .PC.
object	the destination file which will contain the 6809 relocatable object code. If not specified, the filename will default to the <intermediate> filename but with extension .RX.
list	the listing file or device. If not specified, the filename will default to the <intermediate> filename but with extension .LS. However, the default value for Phase 2 is not to produce a listing.
options	various flags for controlling the generation of the listing. Each option is a single letter possibly preceded by a minus sign or followed by an equal sign, value, and comma.

OPTION	DEFAULT	MEANING
A	-A	Include assembly code in the listing. Default (-A) suppresses the assembly code.
H	-H	Include the hexadecimal machine code corresponding to the assembly code in the listing. Default (-H) suppresses the machine code.
S	-S	Include the source as assembly comment lines in the listing (when the O option was enabled in Phase 1). Default (-S) suppresses the source.
L	-L	Produce a pseudo-assembly listing (equivalent to specifing AHS options) of the generated code. Default (-L) suppresses the listing.
N	-N	If the listing is enabled, then produce a narrow, 80-column listing. Default (-N) enables a wide, 132-column listing.
Z=n	Z=32	Set stack/heap size used by Phase 2 to nK. Value of n must be at least 32 (the default value, 32K bytes). The default value will be adequate for code generation for most programs. However, certain programs may cause Phase 2 to abort with error 1008, 1010, or 1011. In such cases, Phase 2 should be executed with a larger Z= option.

Since the listing file output by Phase 2 is normally not needed, it is suppressed by default. For example:

```
        XPAS092    TESTPROG
```

This command processes the intermediate code in TESTPROG.PC:0, creates an object module in the file TESTPROG.RX, and produces no listing.

Another example shows how a listing is produced:

```
        XPAS092    TESTPROG;LN
```

The above example processes TESTPROG.PC, creates a relocatable object module in TESTPROG.RX, and generates an 80-column listing in TESTPROG.LS.

CHAPTER 4

RUNNING THE CROSS LINKAGE EDITOR

4.1 GENERAL

Relocatable object modules generated by Phase 2 of the compiler are processed by the 8-bit Cross Linkage Editor (referred to as the "linker") to produce a transportable S-record module. A Pascal program requires the linker because:

a. Every Pascal program refers to runtime routines which reside in the Runtime library,

b. If a program is to be combined with one or more subprograms that were compiled separately, the linkage between modules must be constructed, and

c. If a Pascal program calls a procedure or function written in assembly language, the load module must include object modules produced by the M6809 Cross Assembler, XASM09.

In all these cases, the linker is required to assign memory space to each required object module, enable intermodule communication, and create an absolute load module in S-record format.

4.2 RUNTIME ROUTINES

The Pascal Runtime library (PAS09LIB) provides certain standard functions that may be optionally used to perform general services. A group of functions and procedures is also provided, which interfaces the Pascal program with the MDOS operating system to provide for input or output (other environments are considered in Chapter 7). A routine is provided to establish the environment required by a Pascal program. Some frequently requested code sequences that perform such activities as manipulating strings or vectors are implemented as runtime routines to reduce program code size.

Whenever a reference is made to one of the runtime routines, an external reference record is produced by the compiler as part of the object module. The linker will include only referenced runtime routines in the S-record module.

4.3 SEPARATE COMPILATION

Pascal supports separate compilations so that the user may group one or more procedures or functions, utilizing only local variables, into a subprogram. The linker can combine as many subprograms as desired. The locations of all level 1 procedures are made known to the linker by external symbol definition records within the object module. The linker can thus resolve references between the program and subprogram or between two subprograms.

4.4 ASSEMBLY LANGUAGE PROCEDURES

Pascal permits the user to refer to procedures or functions written in assembly language. If such routines are required, they should be written as shown in Chapter 5. The linker will enable any Pascal program or subprogram to utilize assembly language routines.

4.5 STACK AND HEAP USAGE

While a Pascal program is running, two types of memory allocation are used: a stack and a heap.

Variables that are global or local and appear in VAR declarations are allocated space on the stack. Global variables -- i.e., those declared in the main program -- occupy space for the duration of the program run. Local variables are allocated stack space when the procedure or function in which they are declared is entered, and relinquish their space on the stack when their containing routine is exited.

Variables appearing in a NEW statement are not stored on the stack, but occupy space on the heap. An appropriate amount of space is allocated on the heap whenever a NEW statement is processed during program execution. This space is not relinquished until a DISPOSE statement is executed.

The stack is built in the highest address of the allocated data segment and grows toward lower addresses; the heap grows from lowest addresses toward higher addresses. The stack and heap may share the data segment space in any ratio, but their total space requirements must not exceed the total memory available or the program will generate a stack/heap overflow error code and abort.

4.6 PASCAL PROGRAM MEMORY ORGANIZATION

The M6809 has a maximum of 64K bytes of address space, part of which is possibly occupied by an operating system, monitors, ROM's, etc. The amount of memory available to the user is divided into three sections: the program section, the data section, and the stack/heap section.

The program section, called PSCT, contains the Pascal object modules, the Pascal library routines, and possibly user-supplied assembly language routines. This program section is more fully described in the Macro Assemblers Reference Manual.

The data section, called DSCT, contains only the data areas, if any, that are required by any user-supplied assembly language routines. Neither the Pascal object modules nor the Pascal library routines require any data area, because all of their variables are allocated on the stack. The Pascal initialization routine, however, does allocate one unused byte in a named-common DSCT section, merely to mark the highest address in the data section. The basic concept of PSCT and DSCT is to separate the code and data sections for a ROM/RAM environment (see the Macro Assembler Manual for more details).

The stack/heap section is allocated from the area of memory above the data section (i.e., at higher addresses), which is unassigned by the linker. It contains the Pascal global and local variable space, the runtime maintenance area, and the dynamic-variable allocations area (NEW variables).

The data/stack/heap section (RAM) may be located either above or below the program section, although a smaller load module results from locating the data/stack/heap section above the program section. The two possible load maps are shown in Figure 4-1.

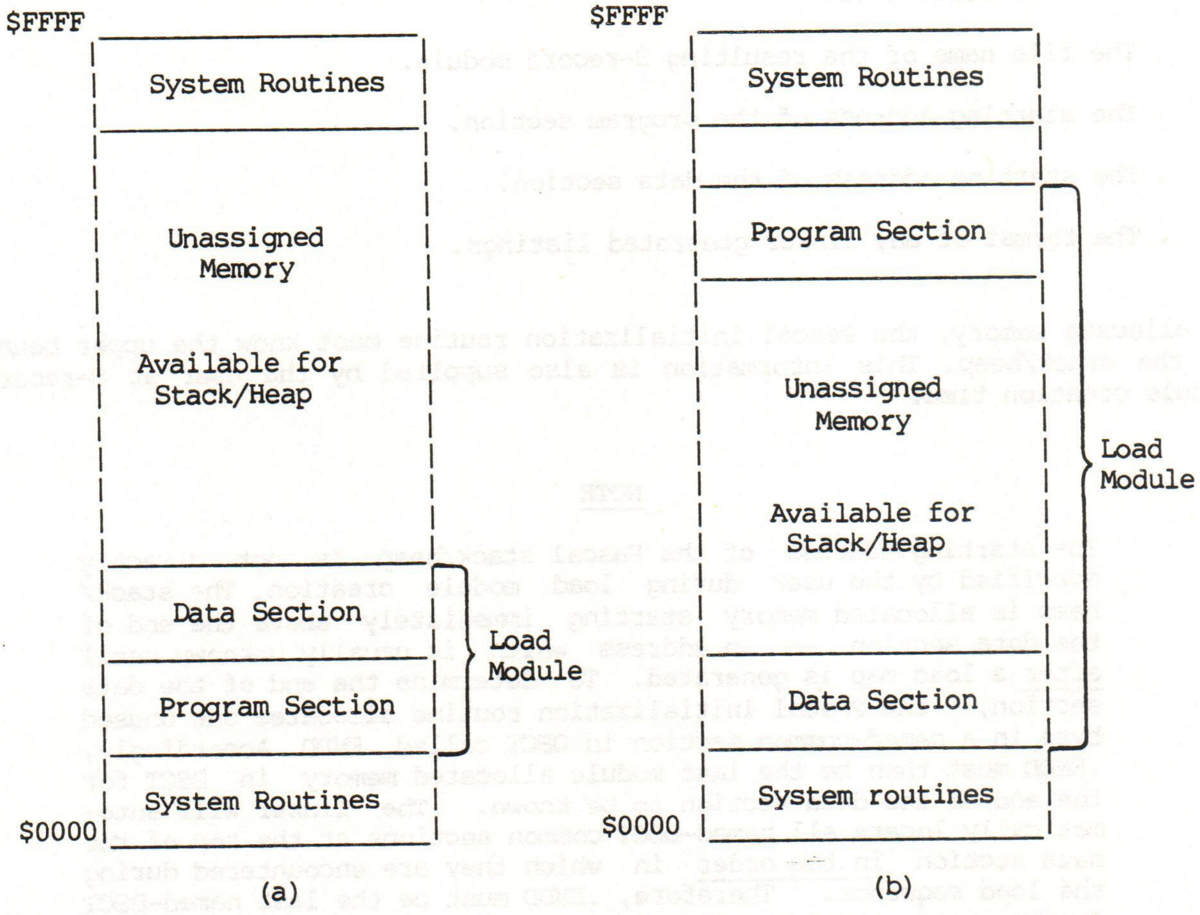


FIGURE 4-1. Pascal Load Module Format

4.7 INVOKING THE CROSS LINKAGE EDITOR

The Cross Linkage Editor is an interactive program which creates an S-record module based on information supplied by the user. The following information is required:

- The file names of the relocatable object modules which are to be included in the S-record module.
- The file names of any library routines which are to be searched to satisfy external references.
- The file name of the resulting S-record module.
- The starting address of the program section.
- The starting address of the data section.
- The format of any linker generated listings.

To allocate memory, the Pascal initialization routine must know the upper bound of the stack/heap. This information is also supplied by the user at S-record module creation time.

NOTE

> The starting address of the Pascal stack/heap is not directly specified by the user during load module creation. The stack/heap is allocated memory starting immediately above the end of the data section -- an address which is usually unknown until after a load map is generated. To determine the end of the data section, the Pascal initialization routine allocates one unused byte in a named-common section in DSCT called .ENDD. Accordingly, .ENDD must then be the last module allocated memory in DSCT for the end of the data section to be known. The linker will automatically locate all named-DSCT common sections at the top of the data section in the order in which they are encountered during the load sequence. Therefore, .ENDD must be the last named-DSCT common encountered by the linker, which implies that all user routines which access a named-DSCT common must be loaded before the Pascal runtime library. If this requirement is not followed, then the Pascal stack/heap will overwrite portions of the user data section.

The linker is invoked by the following VERSAdos command:

```
=XLINK  <object>[,[<absolute>][,<list>]];A[<options>]
```

where the syntactic variables are defined as follows:

object — the file containing the M6809 relocatable object code. If not specified, the extension of .RX is assumed. Multiple object file names may be specified provided they are separated by slashes (/).

absolute — the destination file which will contain the M6809 absolute S-records. If not specified, the filename will default to the <object> filename but with extension .MX.

list — the listing file or device. If not specified, the listing will be directed to the user's console. If a filename is specified, the extension will default to .LL.

options — various flags for controlling the generation of the listing. Each option is a single letter possibly preceded by a minus sign or followed by an equal sign, value, and comma. Some useful options are:

OPTION	DEFAULT	MEANING
A	-A	Accept user commands from the command input device (the user's console). NOTE: This option must be specified.
H	-H	List information found in the header record of each object module. Default (-H) suppresses this listing.
I	-I	List the command line and all user commands. Default (-I) suppresses this listing.
L=<fn>	-L	Search the specified library files at the end of pass 1. Default (-L) suppresses this search. To search the Pascal runtime library, the following option is specified: L=PAS09LIB.RX.
M	-M	List a map of the resulting absolute module. Default (-M) suppresses this listing.
X	-X	List the external definition directory. Default (-X) suppresses this listing.
Z=n	Z=35	Set stack/heap size used by linker to nK. Value of n must be at least 35 (the default value, 35K bytes). The default value will be adequate for linking for most programs. However, certain programs may cause the linker to abort with error 1008, 1010, or 1011. In such cases, the linker should be executed with a larger Z= option.

The A option causes the cross linker to accept additional commands from the user. This is necessary in order to specify the stack/heap bounds of the user's program, as well as the order of allocation and starting address for DSCT (the data section) and PSCT (the program (code) section). The following user commands are required to successfully link an M6809 Pascal program (consult the 8-bit Cross Linkage Editor on EXORmacs Development System User's Guide (M68XLINK) for more information):

LOCATE PSCT,DSCT <address>

This command causes the program to be allocated memory, below the data, starting at location of <address>. An <address> of $2000 allows the program to run with MDOS resident. If MDOS is not required (for I/O for example), then the allocation address can be set as the user desires. Recall that the runtime initialization routine is going to allocate a 1-byte common section in DSCT which is going to be the lower bound of the stack/heap.

DEF .DHIGH <address>

This command defines the externally-referenced symbol .DHIGH to have the specified value. This value is going to be the upper bound of the stack/heap.

DEF .SIZE <value>

This command defines the value for the externally-referenced symbol .SIZE. If the value is non-zero, then the value specified for .DHIGH is ignored and the initialization routine will size memory, allocating all RAM above the common section which marks the end of the data section to the stack/heap.

IN <filename>

This command causes other files to be included in the absolute S-record module.

LIB <filename>

This command causes the indicated library file to be searched to satisfy external references.

END

This command terminates the user's input.

EXAMPLES:

```
     =XLINK  TESTPROG/ASM1/ASM2;AML=PAS09LIB
      !LOCATE  PSCT,DSCT $2000
      !DEF  .DHIGH $DFFF
      !DEF  .SIZE 0
      !END
```

These commands link the object module TESTPROG.RX with PAS09LIB.RX, ASM1.RX, and ASM2.RX; creates the absolute S-record module TESTPROG.MX; and generates a load map on the user's terminal. The program section is allocated memory starting at $2000. The data section is allocated memory immediately above the program section. The stack/heap is allocated memory from the end of the data section to $DFFF.

```
     =XLINK TESTPROG;AL=PAS09LIB
      !LOCATE PSCT $2800
      !LOCATE DSCT $4000
      !DEF .DHIGH $4FFF
      !DEF .SIZE 0
      !END
```

This command links the object module TESTPROG.RX with PAS09LIB.RX and creates the absolute S-record module TESTPROG.MX. No load map is generated. The program section is allocated memory starting at $2800. There is no data section, since only Pascal modules are included. The lower bound of the stack/heap is $4000. The upper bound of the stack/heap is $4FFF.

4.8 DOWNLOADING TO M6809

The final step is the transfer of the M6809 absolute S-records contained in a VERSAdos file to an M6809 for execution. If an EXORciser-based M6809 is available, then a simple way to accomplish the transfer is by means of a floppy disk. The S-record files are first copied to a floppy diskette, using the VERSAdos COPY command. Then, on the EXORciser, the VERSAdos diskette file is converted into an MDOS-loadable file by the following commands:

```
     =VMCOPY <VERSAdos S-record filename>[,<MDOS S-record filename>]
     =EXBIN <MDOS S-record filename>[,<MDOS loadable filename>]
```

where the VERSAdos diskette must be in drive 1 and the MDOS diskette in drive 0. The VMCOPY command converts a VERSAdos file into an MDOS file. The <VERSAdos S-record filename> must include user number, filename, and extension. The MDOS filename will default to the same name but with extension .SA, if not specified. The EXBIN command converts the S-record file into an MDOS-loadable file; the extensions, if not specified, default to .LX for the input and .LO for the output.

NOTE

VMCOPY is not included in the standard MDOS set of utilities, but is available from Motorola Microsystems Field Service.

An alternate method for downloading is to utilize the TRANSFER command under VERSAdos. This method is described in the utilities chapter of the VERSAdos System Facilities Reference Manual, M68KVSF.

4.9 THE PASCAL RUNTIME ENVIRONMENT

Program execution begins in the initialization routine at the address specified by the external label .INIT. The initialization routine performs three functions:

1. allocation of the stack/heap area,
2. initialization of the runtime maintenance area, and
3. initialization of the registers.

The externally-known variable .INITS in the initialization routine defines the address of a six-byte area in which the size-memory flag, the start-of-stack address, and the end-of-stack address are stored. In particular:

.INITS+1:	Size-memory flag (.SIZE)
.INITS+2:	Lower stack/heap bound (address of .ENDD)
.INITS+4:	Upper stack/heap bound (.DHIGH)

These locations are initialized by the linker and are utilized by the initialization routine in the allocation of the stack/heap. It is possible to patch these locations and thus modify the stack/heap location after the S-record module has been created.

Since the Pascal stack/heap is allocated between .ENDD and .DHIGH (if .SIZE is not set), the program section must reside either above .DHIGH or below .ENDD. If the size-memory flag is set, the initialization routine will size memory starting from .ENDD and allocate all of available RAM to the stack/heap. In this case, the program section must reside either below .ENDD or in ROM.

The runtime maintenance area (RMA) resides at the low address end of the stack/heap area. To facilitate access, the RMA is aligned on a page boundary, and the direct page register is initialized to the most significant byte of the RMA address. The direct mode of addressing is utilized in all RMA accesses. A map of the RMA is shown in Figure 4-3. A description of the RMA is provided in Table 4-1.

The heap begins immediately above the RMA, and grows upward toward the stack. The area of memory from the end of the data section (.ENDD) to the next page boundary (the RMA) is not used by any Pascal program.

The stack starts at the high address end of the stack/heap area and grows downward toward the heap.

Certain of the M6809 registers are assigned initial values. These registers are:

S The Pascal stack. Initial value is .DHIGH+1 or first non-RAM byte, depending upon the value of .SIZE.

Y Global data segment pointer. The value is the initial S-value minus six (S-6).

DP Most significant byte of the runtime maintenance area address.

Upon completion, the initialization routine branches to the externally-defined symbol .ENTRY, which is the main program entry point. The initial entry code initializes the display level-zero data segment pointer and the current data segment pointer, and allocates the required area on the stack for the global data segment.

Offset from DP register		
hex	dec	
1B	27	RMA PHYSICAL ADDRESS
19	25	FREELIST HEADER NODE
17	23	
15	21	HEAP POINTER
14	20	RESERVED
12	18	STATEMENT COUNTER
10	16	CURRENT DISPLAY POINTER
E	14	DISPLAY LEVEL 7 POINTER
C	12	DISPLAY LEVEL 6 POINTER
A	10	DISPLAY LEVEL 5 POINTER
8	8	DISPLAY LEVEL 4 POINTER
6	6	DISPLAY LEVEL 3 POINTER
4	4	DISPLAY LEVEL 2 POINTER
2	2	DISPLAY LEVEL 1 POINTER
0	0	DISPLAY LEVEL 0 POINTER

FIGURE 4-3. Pascal Runtime Maintenance Area

TABLE 4-1. Pascal Runtime Maintenance Area Description

CONTENTS	DP OFFSET	DESCRIPTION
Display Level 0 Pointer	0	Pointer to main program's global data area.
Display Level 1 Pointer	2	Pointer to current level one procedure's data area.
Display Level 2 Pointer	4	Pointer to current level two procedure's data area.
Display Level 3 Pointer	6	Pointer to current level three procedure's data area.
Display Level 4 Pointer	8	Pointer to current level four procedure's data area.
Display Level 5 Pointer	10	Pointer to current level five procedure's data area.
Display Level 6 Pointer	12	Pointer to current level six procedure's data area.
Display Level 7 Pointer	14	Pointer to current level seven procedure's data area.
Current Display Pointer	16	Pointer to the currently executing procedure's data area.
Statement Counter	18	Statement number of the currently executing Pascal statement if the D, K, or P option was enabled.
Reserved	20	
Heap Pointer	21	Pointer to the current top of the heap area.
Freelist Header Node	23	Pointer to the start of the freelist, followed by a double-byte of zeros.
RMA Physical Address	27	Address of the start of the RMA.

CHAPTER 5

ASSEMBLY ROUTINE LINKAGE

5.1 GENERAL

An assembly language routine may be called externally by a Pascal program using normal Pascal argument passing. Such a routine may:

a. Perform a function not available in Pascal -- e.g., data manipulation or I/O not provided in the System Library, or some mathematics not supported by Pascal.

b. Optimize some code to be used repetitively in a real-time environment. The Pascal compiler does optimize, but a user-written assembly language routine may be shorter and faster.

5.2 PROGRAM PREPARATION

There are two requirements which must be satisfied in order to include an assembly language subroutine in a Pascal program. The first is to declare the external assembly language routine in the Pascal program. This is done by declaring a level 1 procedure or function -- i.e., one contained only by the main program, using the forward directive. A good place for these declarations is prior to the first non-external procedure heading.

For example:

```
FUNCTION SUMTHREE (I,J,K:INTEGER):INTEGER; FORWARD;
```

The external assembly language subroutine may then be called just as any Pascal procedure or function.

The second requirement concerns the file which contains the assembly language routine. This file must have an entry point, which has been declared external with an XDEF, with the same name as the procedure or function in the Pascal program. The entry point would normally be declared in PSCT or the program section of the assembly language routine.

5.3 CALLING A ROUTINE

Calling an assembly language routine is identical in format -- and its runtime requirements are identical in system usage -- to a regular function or procedure call in Pascal. Parameters, for example, are placed on the top of the stack, beneath the return address, in the order they are declared -- the first parameter is stacked first and the last parameter is nearest the top of the stack. If the assembly language routine is declared a function, the space for the return value is below the first parameter on the stack.

For example, given the declaration and call in the following Pascal program fragment:

```
        FUNCTION SUMTHREE (I,J,K:INTEGER):INTEGER; FORWARD;


        BEGIN
          A:= SUMTHREE(3,5,7);
```

the stack would look as follows upon entry to the assembly language subroutine named SUMTHREE:

```
                      ________________________
  TOP OF STACK -------->|                      |
                      |    RETURN ADDRESS     |  low address
                      |       2 bytes         |
                      |                       |
                      |_______________________|
      POSITIVE        |                       |
      OFFSETS         |   FORMAL PARAMETER    |
       FROM           |     K; 2 bytes;       |
       STACK          |      value = 7        |
      POINTER         |                       |
         |            |_______________________|
         |            |                       |
         |            |   FORMAL PARAMETER    |
         |            |     J; 2 bytes;       |
         |            |      value = 5        |
         |            |_______________________|
         |            |                       |
         V            |   FORMAL PARAMETER    |
                      |     I; 2 bytes;       |
                      |      value = 3        |
                      |                       |
                      |_______________________|
                      |                       |
                      |    FUNCTION VALUE     |
                      |  SUMTHREE; 2 bytes;   |
                      |  value is undefined   |
                      |_______________________|  high address
```

The size of parameters depends on the type.

A VAR parameter passes a two-byte address of the actual parameter, which can be used to reference the actual parameter via indirection. A value parameter passes the value of the expression which corresponds to the formal parameter.

Boolean parameters occupy one byte on the stack. This byte has the value of one for true and the value of zero for false.

Character parameters use one byte on the stack. This byte has the value of the ASCII code for the character passed in it.

Integer parameters occupy one, two, or four bytes on the stack. They are stored as two's complement numbers.

Set parameters require eight bytes on the stack, with the byte nearest the top of the stack containing bits 63-56 and the byte farthest from the top of the stack containing bits 7-0.

Arrays and records occupy a number of bytes equal to their length.

Strings should always be passed to assembly language routines as VAR parameters, due to the complexity of determining their actual size on the stack.

Pointers require two bytes on the stack and they contain the address of the variable they reference.

The assembly language subroutine is responsible for preserving the value of registers DP and Y during its execution. It is also responsible for removing all parameters passed to it by the Pascal program and for storing a value in the return value location if the subroutine was declared as a function.

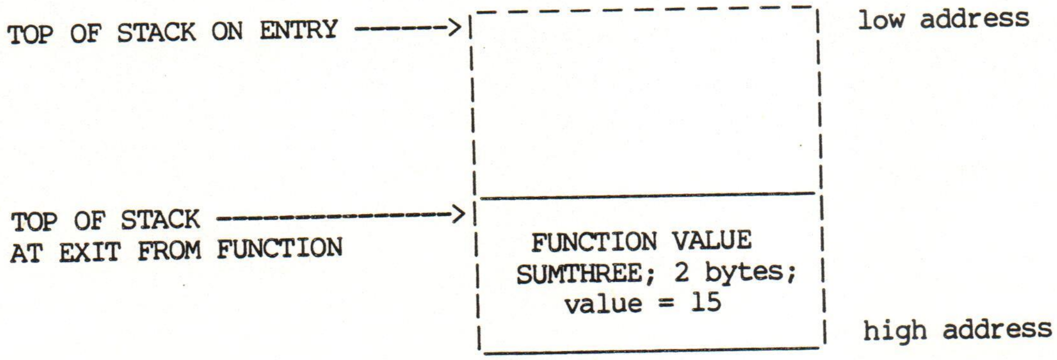
The value of the Y register may be of use to the assembly language routine, since it points to the base of the global variable area. To reference a variable in this area, a negative displacement from the register must be used.

The assembly language subroutine is free to use the space between the top of the stack and the top of the heap for local data storage. The address of the top of the heap is kept in the RMA at offset 21 (see Figure 4-3).

If the stack pointer ever contains an address that is less than the address of the top of the heap, a stack/heap overflow condition has occurred. If a stack/heap overflow has occurred, then both the stack and the heap may contain invalid data.

Control may be returned to the Pascal program by means of either a return from subroutine instruction or a jump indirect through the X- register which contains the return address. No matter which method is used, it is up to the assembly language subroutine to adjust the stack so as to remove the passed parameters. If the assembly language routine returned a function value, then the stack pointer should point to that location on the stack where the space was reserved for the return value prior to the call. If the assembly language routine did not return a function value, the stack pointer should point just below where the first parameter was pushed on the stack.

Following is a picture of the stack for the SUMTHREE routine, seen earlier, just before the return to the Pascal program:



5.4 ROUTINE LINKAGE

An assembly language routine is linked with a Pascal program by means of the 8-bit Cross Linker.

5.5 SAMPLE PROGRAM

The following example demonstrates the linkage between a Pascal program and an assembly-language routine.

The assembly language routine (see paragraph 5.5.1) utilizes the MDOS system call DSPLZ to output the contents of a text file buffer to the system console without a carriage return. The routine refers to various fields in the file pointer and the file descriptor, both of which are described in Chapter 7. The MDOS I/O Control Block (IOCB) is described in the MDOS User's Guide, as is the MDOS system call DSPLZ. Note that it was necessary to reset the buffer pointers (one in the file pointer and one in the file descriptor), since these are accessed by other Pascal routines.

The Pascal program (paragraph 5.5.2) uses the assembly language routine to prompt the user for input. The routine PROMPT performs essentially the same function as the Pascal routine WRITELN, but without the closing carriage return. Note that PROMPT was declared FORWARD in the program and that the compiler recognizes that it was external.

5.5.1 Assembly Language Routine Listing

```
Motorola M6809 X-Assembler  1.10                     PROMPT  .SA 02/16/83 12:30

 P                    *
 P                    *          PROCEDURE PROMPT (VAR FIL: TEXT);
 P                    *
 P                    *          THIS ASSEMBLY LANGUAGE ROUTINE INTERFACES TO A PASCAL
 P                    *          PROGRAM AND FORCES WHATEVER IS IN THE BUFFER FOR THE
 P                    *          FILE POINTED TO BY FIL TO BE WRITTEN TO THE CONSOLE
 P                    *          WITHOUT A CARRIAGE RETURN.
 P                    *
 P                    *
 P                               NAM                  PROMPT
 P                               XDEF      PROMPT
 P                    *
 P                    *  SYSTEM CALL MACRO
 P                    *
 P                    SCALL      MACR
 P 0000                          SWI
 P 0000                          FCB                  \0
 P 0000                          ENDM
 P                    *
 P                    *  FILE DESCRIPTOR OFFSETS
 P                    *
 A      0000          NXTPTR     EQU       00         NEXT COMPONENT
 A      000C          IOCB       EQU       12         ACTIVE IOCB
 A      0010          IOCDBS     EQU       IOCB+4     DATA BUFFER START ADDRESS
 P                    *
 P                    *  STATUS OF STACK
 P                    *
 P                    *          ENTRY:    0: RETURN ADDRESS
 P                    *                    2: ADDRESS OF FILE POINTER
 P                    *
 P                    *          EXIT:     NONE
 P                    *
 P 0000 3420          PROMPT     PSHS      Y          SAVE GLOBAL POINTER
 P 0002 10AE64                   LDY       4,S        ADDRESS OF FILE POINTER
 P 0005 AEA4                     LDX       0,Y        ADDRESS OF NEXT CHAR IN BUFFER
 P 0007 10AE22                   LDY       2,Y        ADDRESS OF FILE DESCRIPTOR
 P 000A C604                     LDB       #4         LOAD EOT
 P 000C E784                     STB       0,X        APPEND EOT TO BUFFER
 P 000E AEA810                   LDX       IOCDBS,Y   ADDRESS OF BUFFER
 P 0011                          SCALL     12         OUTPUT BUFFER (DSPLZ)
 P 0013 ECA810                   LDD       IOCDBS,Y   RESET BUFFER
 P 0016 EDF804                   STD       [4,S]      POINTERS
 P 0019 C30001                   ADDD      #1
 P 001C EDA4                     STD       NXTPTR,Y
 P 001E 3560                     PULS      Y,U        GET GLOBAL PTR AND RET ADDR
 P 0020 3262                     LEAS      2,S        DISCARD PARAMETER
 P 0022 6EC4                     JMP       0,U        RETURN
 P                               END

***** TOTAL ERRORS      0--   0
***** TOTAL WARNINGS    0--   0
```

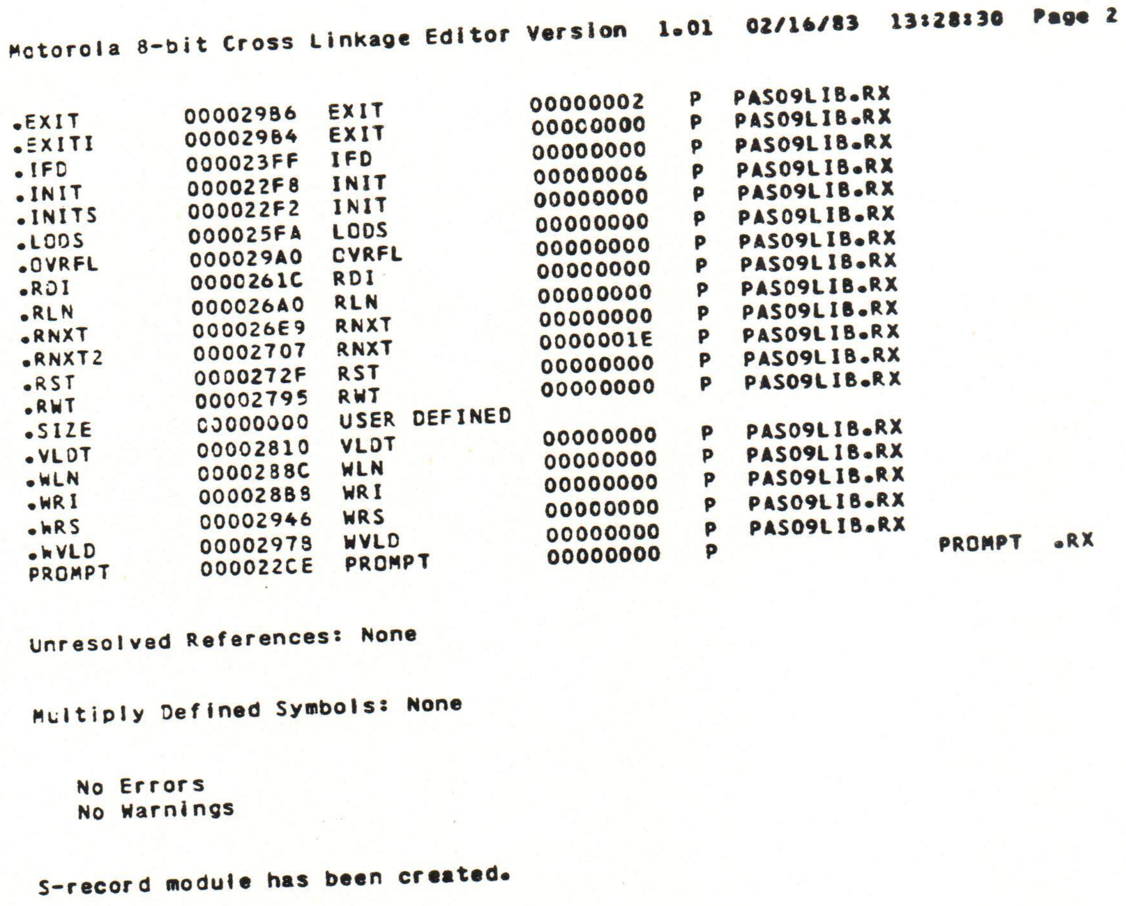
5.5.2 Pascal Program Listing

```
Line    Loc Lev BE M6809 Cross Pascal 1.20    SORT     .SA 02/16/83 13:21:38

   1(       0) 0)-- {-------------------------------------------------------------------}
   2(       0) 0)-- {                                                                   }
   3(       0) 0)-- {                               S O R T                             }
   4(       0) 0)-- {                                                                   }
   5(       0) 0)-- {     This program demonstrates the linking of a Pascal             }
   6(       0) 0)-- {     program with an assembly language subroutine.                 }
   7(       0) 0)-- {                                                                   }
   8(       0) 0)-- {     The driver program simply asks for an array of numbers,       }
   9(       0) 0)-- {     one by one, sorts the numbers in increasing numerical         }
  10(       0) 0)-- {     order, and prints the results.                                }
  11(       0) 0)-- {                                                                   }
  12(       0) 0)-- {     An assembly language routine is used to prompt output         }
  13(       0) 0)-- {     to the console without having to do a writeln.  This          }
  14(       0) 0)-- {     allows for prompting for input and having the input           }
  15(       0) 0)-- {     entered on the same line.                                     }
  16(       0) 0)-- {                                                                   }
  17(       0) 0)-- {-------------------------------------------------------------------}
  18(       0) 0)--
  19(      -8) 0)-- PROGRAM sort(input,output);
  20(      -8) 0)--
  21(      -8) 0)--    CONST
  22(      -8) 0)--       max+array+size = 5000;             {maximum array size}
  23(      -8) 0)--
  24(      -8) 0)--    TYPE
  25(      -8) 0)--       index+range = 1..max+array+size; {range of indices into array}
  26(      -8) 0)--
  27(      -8) 0)--    VAR
  28(-10008) 0)--       number+array:  ARRAY [index+range] OF integer;
  29(-10010) 0)--       array+size:    0..max+array+size;   {actual size of array}
  30(-10014) 0)--       i,j:           index+range;         {indices into array}
  31(-10016) 0)--       temp:          integer;             {used for swapping elements}
  32(-10017) 0)--       exchange:      boolean;             {any exchanges?}
  33(-10017) 0)--
  34(-10017) 0)--    {Declare the needed assembly language routine as external}
  35(-10017) 0)--
  36(       0) 1)--    PROCEDURE prompt (VAR fil: text);   FORWARD;
  37(       0) 1)--
**** PROMPT   Assumed external
  38        1  0)A-    BEGIN {sort}
  39           0)--
  40           0)B-       REPEAT                                {loop for each array}
  41           0)--
  42        2  0)--          writeln(output);                   {ask for size of array}
  43        3  0)--          writeln(output);
  44        4  0)--          write(output,'Input size of array (0 to quit): ');
  45        5  0)--          prompt(output);
  46           0)--
  47        6  0)--          readln(output,array+size);        {get array size}
  48           0)--
  49        7  0)C-          IF array+size > 0 THEN BEGIN
  50           0)--
  51        8  0)D-             FOR i := 1 TO array+size DO BEGIN  {read numbers, one by one}
  52        9  0)--                write(output,'Input number ', i:3, ': ');
  53       10  0)--                prompt(output);
  54       11  0)--                readln(input,number+array[i]);
  55           0)-D                END;
```

```
Line    Loc Lev BE M6809 Cross Pascal 1.20   SORT    .SA 02/16/83 13:21:38

 56          0)--
 57          0)--            {Using a simple bubble sort -- sort the numbers.}
 58          0)--
 59      12  0)--            j := array+size - 1;
 60          0)D-            REPEAT
 61      13  0)--              exchange := false;
 62      14  0)--              FOR i := 1 TO j DO
 63      15  0)E-                IF number+array[i] > number+array[i+1] THEN BEGIN
 64      16  0)--                  temp                := number+array[i];
 65      17  0)--                  number+array[i]     := number+array[i+1];
 66      18  0)--                  number+array[i+1]   := temp;
 67      19  0)--                  exchange            := true;
 68          0)-E                  END; {THEN and FOR}
 69      20  0)--              j := j - 1;
 70      21  0)-D              UNTIL (NOT exchange) OR (j < 1);
 71          0)--
 72      22  0)--            writeln(output);          {now output the results}
 73      23  0)--            writeln(output);
 74      24  0)--            writeln(output,'Numbers in sorted order are:');
 75      25  0)--            FOR i := 1 TO array+size DO
 76      26  0)--              writeln(output,number+array[i]:5);
 77          0)--
 78          0)-C            END; {THEN}
 79      27  0)-B          UNTIL array+size <= 0;
 80          0)--
 81      28  0)--        writeln(output);
 82      29  0)--        writeln(output);
 83      30  0)--        writeln(output,'Done - Thank You');
 84          0)-A        END.  {sort}

     **** No Error(s) and No Warning(s) detected

     **** 84 Lines 1 Procedures

     **** 261 Pcode instructions
```

5.5.3 Load Map Listing

```
Motorola 8-bit Cross Linkage Editor Version  1.01  02/16/83  13:28:30  Page 1


Command Line:

LINK SORT,SORT,SCRT;AIMXL=PAS09LIB.RX


Options in Effect:  A,-B,-H,I,L,M,Q,-U,X


User Commands:

LCCATE PSCT,DSCT $2000
DEF .DHIGH $DFFF
DEF .SIZE 0
IN PROMPT.RX
END


Load Map:

Module        S  T  Start     End       Externally Defined Symbols

SCRT          P     00002000  000022CD  .ENTRY    00002000
PROMPT        P     000022CE  000022F1  PROMPT    000022CE
INIT          P     000022F2  0000235D  .INIT     000022F8  .INITS    000022F2
CLO           P     0000235E  000023D6  .CLO      0000235E
ENT           P     000023D7  000023FE  .ENT      000023D7
IFD           P     000023FF  000025F9  .IFD      000023FF
LCDS          P     000025FA  0000261B  .LODS     000025FA
RCI           P     0000261C  0000269F  .RDI      0000261C
RLN           P     000026A0  000026E8  .RLN      000026A0
RNXT          P     000026E9  0000272E  .RNXT     000026E9  .RNXT2     00002707
RST           P     0000272F  00002794  .RST      0000272F
RWT           P     00002795  0000280F  .RWT      00002795
VLDT          P     00002810  00002888  .VLDT     00002810
WLN           P     0000288C  000028B7  .WLN      0000288C
WRI           P     000028B8  00002945  .WRI      000028B8
WRS           P     00002946  0000297A  .WRS      00002946
WVLD          P     0000297B  0000299F  .WVLD     0000297B
OVRFL         P     000029A0  000029B3  .OVRFL    000029A0
EXIT          P     000029B4  00002A11  .EXIT     000029B6  .EXITI    000029B4
CVHEX         P     00002A12  00002A28  .CVHEX    00002A12
.ENDD         D  C  00002A29  00002A29


Table of Externally Defined Symbols:

Name          Address   Module          Displ     Sect Library       Input

.CLO          0000235E  CLO             00000000   P   PAS09LIB.RX
.CVHEX        00002A12  CVHEX           00000000   P   PAS09LIB.RX
.DHIGH        0000DFFF  USER DEFINED
.ENT          000023D7  ENT             00000000   P   PAS09LIB.RX
.ENTRY        00002000  SORT            00000000   P                 SORT    .RX
```



```
Motorola 8-bit Cross Linkage Editor Version  1.01  02/16/83  13:28:30  Page 2

.EXIT          000029B6  EXIT          00000002   P  PAS09LIB.RX
.EXITI         000029B4  EXIT          00000000   P  PAS09LIB.RX
.IFD           000023FF  IFD           00000000   P  PAS09LIB.RX
.INIT          000022F8  INIT          00000006   P  PAS09LIB.RX
.INITS         000022F2  INIT          00000000   P  PAS09LIB.RX
.LODS          000025FA  LODS          00000000   P  PAS09LIB.RX
.OVRFL         000029A0  OVRFL         00000000   P  PAS09LIB.RX
.RDI           0000261C  RDI           00000000   P  PAS09LIB.RX
.RLN           000026A0  RLN           00000000   P  PAS09LIB.RX
.RNXT          000026E9  RNXT          00000000   P  PAS09LIB.RX
.RNXT2         00002707  RNXT          0000001E   P  PAS09LIB.RX
.RST           0000272F  RST           00000000   P  PAS09LIB.RX
.RWT           00002795  RWT           00000000   P  PAS09LIB.RX
.SIZE          00000000  USER DEFINED
.VLDT          00002810  VLDT          00000000   P  PAS09LIB.RX
.WLN           0000288C  WLN           00000000   P  PAS09LIB.RX
.WRI           000028B8  WRI           00000000   P  PAS09LIB.RX
.WRS           00002946  WRS           00000000   P  PAS09LIB.RX
.WVLD          00002978  WVLD          00000000   P  PAS09LIB.RX
PROMPT         000022CE  PROMPT        00000000   P                   PROMPT  .RX

Unresolved References: None

Multiply Defined Symbols: None

    No Errors
    No Warnings

S-record module has been created.
```

CHAPTER 6

PROGRAM EXECUTION

6.1 MDOS PROGRAM EXECUTION

Once an executable load module has been created with the extension .CM, it may be executed under MDOS by typing the file name in response to the MDOS prompt. (The non-MDOS environment is considered in Chapter 7.) Any file variables in the program header are defaulted to local (i.e., temporary) disk files, unless an external file assignment is made (see paragraph 6.2). The exception to the file defaults is for standard files 'input' and 'output', which will default to the system console (#CN).

6.2 EXTERNAL FILE ASSIGNMENT

External runtime file assignments may be specified in two ways:

a. By a special form of the reset and rewrite procedures.

b. By forming a correspondence between the file variables in the program header and the command line.

6.2.1 Resource Name Strings

File assignment using the reset and rewrite procedures is described in the handbook, Pascal Programming Structures for Motorola Microprocessors. The syntax for this form of the reset and rewrite procedure calls is:

```
  RESET   (<file-variable>,<resource-name-string>);
REWRITE   (<file-variable>,<resource-name-string>);
```

The resource name string may be any string-valued expression, including but not limited to string constants and variables. The resource name string for MDOS file name conventions is defined as follows:

```
resource-name-string  ::=  <file or device>[;<option list>]|;<option list>
file or device        ::=  <file name>|<device name>
file name             ::=  <name>[.<suffix>][:<logical unit>]|
                               :<logical unit>
device name           ::=  #<device mnemonic>
device mnemonic       ::=  LP|CN|CP|CR
option list           ::=  [<option>[,]|I<integer>,]...
option                ::=  0|1|2|3|5|7|C|D|F|N|R|S|W
```

The options primarily affect the format of new files as they are created for output. When opening an existing file for input, the file options are essentially overwritten by the already existing attributes of the file. The following file-attribute options are defined:

C   Contiguous diskette space allocation.
D   Delete protection.
N   Non-compression of spaces in ASCII records.
S   System attribute.
W   Write protection.

The format of the file records determines whether record I/O or logical sector I/O is performed during the access. In general, logical sector I/O is performed whenever possible, and that is when (1) the component size is an even multiple of the sector size (128 bytes) or (2) the component size is greater than 254 bytes (the maximum size for record I/O). The following file formats are defined:

0 User-defined records. Logical sector I/O.

1 Binary records. Defaults to format 3 or 7, depending upon the device. Record I/O.

2 Memory-imaged records. Logical sector I/O.

3 Binary records (8-bit data bytes). Record I/O.

5 ASCII text records. Record I/O.

7 ASCII-converted binary records (7-bit data bytes). Record I/O.

Other options defined:

F Forces file-mode I/O for non-diskette devices. (Default for non-diskette devices is non-file mode.)

R Forces record I/O, overriding the default logical sector I/O when the component size is 128 bytes. It will cause a runtime error to specify the R option if the component size is greater than 254 bytes.

I<integer> The size of the initial sector allocation for a new file.

The following default values are utilized for all file variables:

Device type	=	DK
Logical unit	=	0
File name	=	PFxxxx.SA
File format	=	0 (if non-text and sector I/O)
	=	3 (if non-text and record I/O)
	=	5 (if text)
File Attributes Set	=	None
File/Non-file Mode	=	File mode
Record/Sector I/O	=	Record I/O (if component size less than 255 bytes and not 128 bytes)
	=	Sector I/O (if component size greater than 254 bytes or 128 bytes)
Initial Sector Allocation	=	128 sectors

Examples of resource name strings:

```
FILE1.SA
SAM.RO:1;DI24
CRT.CM:1;SCI48,2
#LP;5
#CR;F7
:1
```

The last example is a special case in that normally when a resource name string is provided, a local file variable is marked as being external. However, when only a logical unit is specified as the resource name string, the local attribute is maintained and the file will be deleted at the appropriate time. This allows the user to direct temporary files to logical units other than :0 (the default).

6.2.2 Command Line File Assignments

The second way file assignments may be specified is by forming a correspondence between the file variables in the program header and the parameters in the MDOS command line which invoked the program's execution. For example, given the following program header:

```
PROGRAM test (input,output,infile,libfile);
```

Assuming that program test is now contained in file TEST.CM:0, then a possible MDOS command line to execute the program would be:

```
TEST  I=FILEA,O=#LP,FILEB.RK,FILEC
```

NOTES

1. Standard file 'input' is specified by 'I=' preceding the file/device name with which it is to be associated.

2. Standard file 'output' is specified by 'O=' preceding the file/device name with which it is to be associated.

3. The default file/device for both standard 'input' and 'output' is #CN (the console).

4. All other external files are associated with file variables by first ignoring file variables 'input' and 'output' in the program header and I= and O= file designations in the command line, and then establishing a one-to-one correspondence between the remaining files in the two lists. In the example above, 'infile' will be associated with file FILEB.RK:0 and 'libfile' with file FILEC.SA:0.

5. The default suffix for all file names is .SA, and the default logical unit is :0.

6. The standard files 'input' and 'output' may be specified in any order on the MDOS command line. The foregoing example could have been written:

   ```
   TEST  FILEB.RK,O=#LP,FILEC,I=FILEA
   ```

   and the same file assignments would have resulted.

7. All file variables listed in the program header for which no corresponding external file is listed in the MDOS command line are treated as temporary files. A runtime-generated file name will be supplied for each temporary file. All temporary files are deleted upon program termination.

6.3 PROGRAM TERMINATION

Upon program termination, the compiler will cause a two-byte integer value of zero to be loaded into the D register, and then a branch to the Pascal exit routine (.EXIT). If an error occurs, a non-zero value (the error code) will be loaded into the D register before the exit routine is called. If an MDOS I/O error occurs, the X register will also contain the address of the IOCB (I/O Control Block) associated with the error.

If no error occurred, the exit routine will simply reenter resident MDOS via the .MDENT system call. If an MDOS I/O error occurred, the appropriate system error message is displayed by the .MDERR system call. If any error occurred, the following message will be displayed on the console by the .DSPLY system call:

```
ERR-xx AT yyyy
```

where: xx = Runtime error number.

yyyy = Value of the program counter or, if statement counting is enabled, the value of the statement counter.

Both values are in hexadecimal. Appendix D contains a list of the runtime error numbers.

CHAPTER 7

PASCAL-MDOS I/O INTERFACE

7.1 DATA STRUCTURES

A Motorola Pascal program will reference a number of runtime routines in order to perform its I/O. In order to do the I/O in the most general way, advantage was taken of the MDOS device-independent I/O functions. This allows a file variable to refer to either a device or a file on diskette at the expense of having MDOS present in memory whenever the program is executed. The purpose of this chapter is to document the interface for the non-MDOS user.

The means by which a Pascal program communicates with its environment is through its file variables. A file variable is allocated space on the Pascal stack and consists of two parts: a file pointer and a file descriptor.

It should be noted that the format of the file descriptor, as shown in paragraph 7.1.2, is dictated by the fact that it is designed to support file I/O in an MDOS environment. For the non-MDOS user, the file descriptor can be formatted in whatever manner the user deems best for his/her application. For example, it would be highly unlikely that a user doing non-MDOS I/O would require an I/O Control Block (IOCB) in the file descriptor.

7.1.1 File Pointer (FP)

A File Pointer (FP) is allocated by the Pascal Compiler in the local variable space of the current block for each file variable declared within that block. The FP consists of two addresses:

a. the address of the current file component (this is the Pascal pointer associated with the file variable), and

b. the address of the File Descriptor (FD), which contains necessary data to maintain an MDOS-compatible file.

The FD address is set up by a call to the FD initialize routine, and is unchanged during the remainder of program execution. The FP component pointer is initialized by the reset (or rewrite) routine, and then modified as necessary by the various I/O routines.

FP Picture

```
          Offset    ___________________________________
          ------
            0:    |_ current component address _______|

            2:    |_ file descriptor address _________|
```

7.1.2 File Descriptor (FD)

The Pascal Compiler generates a call to the FD initialization routine, upon procedure/function/program entry, for each file declared in that program block, in order to allocate and initialize its file descriptor. Certain generally necessary data is passed to the initialize routine via the stack, and an FD is formatted, as follows:

FD Picture for MDOS09

```
          Offset  ____________________________
              0: |   next component address   |
                 |____________________________|
              2: |       component size       | (in bytes)
                 |____________________________|
              4: |            file            | (32-bit integer)
                 |          position          |
                 |____________________________|
              8: |        file status         |
                 |____________________________|
             10: |     record end address     |
                 |____________________________|
             12: |                            |
                 |        active IOCB         |
                 |                            |
                 |____________________________|
             49: |                            |
                 |        backup IOCB         |
                 |                            |
                 |____________________________|
             86: |                            |
                 |       sector buffer        |
                 |                            |
                 |____________________________|
            342: |                            |
                 |       record buffer        |
                 |                            |
                 |____________________________|
```

TABLE 7-1. File Descriptor Status Bits

BIT	MEANING
0	Standard File Output
1	Standard File Input
2	Text File
3	Local File
4	Indexed File
5	Reserved
6	Reserved
7	Reserved
8	File Open
9	End-of-File
10	End-of-Line
11	Sector I/O
12	Device Specified
13	Drive Specified
14	Suffix Specified
15	Name Specified

Least Significant Bit = Bit 0

7.2 INPUT SCHEME

According to the Pascal standard, when a reset is done on a file variable, the first component of the file is immediately accessible by the file pointer (FP). The problem which arises from this requirement concerns associating a file variable with an interactive device such as a console. When a reset is done, the first component must be available, but will not be until the user enters the first component. Rather than forcing the program to wait until that first component is entered, Motorola Pascal has adopted the scheme which is known as 'lazy I/O'.

Essentially, the FP need not point to a valid file component until that component is accessed. Therefore, when a reset is done, the FP is left 0, indicating that the actual read has not been done yet. When the file component is then accessed, it is done so via the peek function, which causes the actual physical read to be done, if necessary. Following the peek, the FP is valid and points to the current file component.

Consider a reset followed by a get on the same file. After the reset, the first component should be accessible; after the get, the second file component should be accessible; but after the reset, the FP is 0 and no read has been done. The get procedure must cause the read of the first component (since the current FP was 0) and then set the FP to 0 to indicate that the second read has yet to be performed. During a subsequent file component access, the peek function will cause the second read at that time.

7.3 I/O ROUTINES

7.3.1 Initialize File Descriptor (IFD)

Entry Point: .IFD
Runtime Errors: 28

Stack Parameters:
0: Return address
2: Component size in bytes
4: Initial file status
6: Position in program header
8: Address of file pointer

Return Parameters:
0: Initialized file descriptor

Allocate a file descriptor (FD) on the stack, initialize the FD and backup IOCB with the data given and default values, and put its address in the proper field of the file pointer (FP). Initialize the current component address with the address of the record buffer. No action is taken with regard to the MDOS I/O system.

If the file variable was mentioned in the program header, its relative position in that header is found in the indicated parameter. The position is derived by counting from 1, beginning at the left, all file identifiers except 'input' and 'output'. The value of this parameter is irrelevant for local files and for the files 'input' and 'output'. If the local bit is not set in the initial file status, or if standard file 'input' or 'output' is indicated, then the MDOS command line is scanned for the indicated file position or for the 'I=' or 'O=' prefix. If a file name is found at the indicated position, then that name is used to initialize the backup IOCB in the FD. The appropriate status bits are set depending upon what portion of the file name was specified.

7.3.2 Assign File to I/O Resource (AFI)

Entry Point: .AFI
Runtime Errors: 28,29,2A

Stack Parameters: 0: Return address
2: Resource name string [n]
3+n: Address of FP

Return Parameters: 0: Address of FP

Cause the name field of the backup IOCB of the FD of the given file to be set to the MDOS file/device name contained in the resource name string. Adjust the file descriptor flags as indicated by any possible options in the resource name string. Abolish any previous assignment that may have been in effect.

7.3.3 Reset (RST)

Entry Point: .RST
Runtime Errors: 20,26

Stack Parameters: 0: Return address
2: Address of FP

If the file is opened, close it. Copy the backup IOCB to the active IOCB. Reserve the device and open it for input. Set the current component address in the FP to 0. Initialize the file position to 1. Set the opened bit in the file status word.

7.3.4 Rewrite (RWT)

Entry Point: .RWT
Runtime Errors: 21,26

Stack Parameters: 0: Return address
2: Address of FP

If file is opened, close it and delete it if it is a disk file. Copy the backup IOCB to the active IOCB. Reserve the device, create the file, and open it for output. Set the end-of-file and opened bits in the file status word. Initialize the file position to 1. Initialize the current component address in the FP to the address of the record buffer if doing record I/O, or to the address of the sector buffer if doing sector I/O.

7.3.5 Close (CLO)

Entry Point: .CLO
Runtime Errors: 25,26

Stack Parameters: 0: Return address
2: Address of FP

If the file is opened for output, complete the last I/O operation. Close the file, release the device, and delete the file if it was a local disk file. Clear the opened bit in the file status word.

7.3.6 Get (GET)

Entry Point: .GET
Runtime Errors: 22,24

Stack Parameters: 0: Return address
2: Address of FP

If the current component address in the FP is 0, fetch the next component. Increment the file position. Clear the current component address in the FP.

7.3.7 Peek (PEE)

Entry Point: .PEE
Runtime Errors: 24,27

Stack Parameters: 0: Return address
2: Address of fP

Return Parameters: 0: Address of component

If the current component address in the FP is 0, fetch the next component. Push the current component address onto the stack.

7.3.8 Put (PUT)

Entry Point: .PUT
Runtime Errors: 23,25

Stack Parameters: 0: Return address
2: Address of FP

Increment the current component address in the FP by the component size. If greater than the end of the record buffer, output the record and reset the current component address. Increment the file position.

7.3.9 Read Character (RDC)

Entry Point: .RDC
Runtime Errors: 22,24

Stack Parameters: 0: Return address
2: Address of variable
4: Address of FP

Return Parameters: 0: Address of FP

If the current component address in the FP is 0, fetch the next component (a character). Store the current character into the indicated variable. Increment the file position. Clear the current component address in the FP.

7.3.10 Read Boolean (RDB)

Entry Point: .RDB
Runtime Errors: 22,24,33

Stack Parameters:	0: Return address 2: Address of variable 4: Address of FP
Return Parameters:	0: Address of FP

If the current component address in the FP is 0, fetch the next component. Skip over blank characters. Read the character string 'TRUE' or 'FALSE', converting all characters to uppercase as they are read. Character string scan is terminated by the first non-alphanumeric or non-matching character. If true or false was read, store the Boolean value, one or zero, respectively, into the indicated variable.

7.3.11 Read Integer (RDI)

Entry Point: .RDI
Runtime Errors: 22,24,31

Stack Parameters:	0: Return address 2: Address of variable 4: Address of FP
Return Parameters:	0: Address of FP

If the current component address in the FP is 0, fetch the next component. Skip over blank characters. Read the numeric character string. Terminate the character scan on the first non-numeric character. Store the integer value into the indicated variable.

7.3.12 Read String (RDS)

Entry Point: .RDS
Runtime Errors: 22,24

Stack Parameters:	0: Return address 2: Size of variable 4: Address of variable 6: Address of FP
Return Parameters:	0: Address of FP

If the current component address in the FP is 0, fetch the next component. Read until the end of the current line. Store up to the maximum number of characters into the indicated variable. (Note: size of string parameter includes the length byte.)

7.3.13 Read Packed Array of Characters (RDV)

Entry Point:	.RDV
Runtime Errors:	22,24

Stack Parameters:	0:	Return address
	2:	Size of variable
	4:	Address of variable
	6:	Address of FP
Return Parameters:	0:	Address of FP

If the current component address in the FP is 0, fetch the next component. Read until the end of the current line. Store the number of characters as indicated by the stack parameter, padding with blanks if necessary, into the indicated variable.

7.3.14 Write Character (WRC)

Entry Point:	.WRC
Runtime Errors:	None

Stack Parameters:	0:	Return address
	2:	Format length
	4:	Character value
	5:	Address of FP
Return Parameters:	0:	Address of FP

Move the required number of blanks to the record buffer. Move the indicated character to the record buffer. Increment the current component address in the FP and the file position after each character.

7.3.15 Write Boolean (WRB)

Entry Point:	.WRB
Possible Errors:	None

Stack Parameters:	0:	Return address
	2:	Format length
	4:	Boolean value
	5:	Address of FP
Return Parameters:	0:	Address of FP

Move the required number of blanks to the record buffer. Move the character string 'TRUE' for a true value or the character string 'FALSE' for a false value. Increment the current component address in the FP and the file position after each character.

7.3.16 Write Integer (WRI)

Entry Point:	.WRI
Possible Errors:	None

Stack Paramters:	0:	Return address
	2:	Format length
	4:	Integer value
	6:	Address of FP
Return Parameters:	0:	Address of FP

Move the required number of blanks to the record buffer. Convert the integer value to a character string and move the characters to the record buffer. Increment the current component address in the FP and the file position after each character.

7.3.17 Write String (WRS)

Entry Point:	.WRS
Runtime Errors:	None

Stack Parameters:	0:	Return address
	2:	Format length
	4:	String [n] value
	5+n:	Address of FP
Return Parameters:	0:	Address of FP

Move the required number of blanks to the record buffer. Move the string characters to the record buffer. Increment the current component address in the FP and the file position after each character.

7.3.18 Write Packed Array of Character (WRV)

Entry Point:	.WRV
Runtime Errors:	None

Stack Parameters:	0:	Return address
	2:	Array length (n)
	4:	Format Length
	6:	Array value
	6+n:	Address of FP
Return Parameters:	0:	Address of FP

Move the required number of blanks to the record buffer. Move the array characters to the record buffer. Increment the current component address in the FP and the file position after each character.

7.3.19 Read Past End-of-Line (RLN)

Entry Point: .RLN
Runtime Errors: 22,24

Stack Parameters:
0: Return address
2: Address of FP

If the current component address in the FP is 0, fetch the next character. Read characters, incrementing the current component address in the FP and the file position after each character, until the end of the line is reached. Clear the current component address in the FP.

7.3.20 Write End-of-Line (WLN)

Entry Point: .WLN
Runtime Errors: 25

Stack Parameters:
0: Return address
2: Address of FP

Move a carriage return character to the record buffer. Output the record and reset the current component address in the FP.

7.3.21 End-of-Line Status (EOL)

Entry Point: .EOL
Runtime Errors: 24

Stack Parameters:
0: Return address
2: Address of fP

Return Parameters:
0: Boolean value

If the current component address in the FP is 0, fetch the next component. Return the value of the end-of-line status bit of the indicated text file.

7.3.22 End-of-File Status (EOF)

Entry Point: .EOF
Runtime Errors: 24

Stack Parameters:
0: Return address
2: Address of FP

Return Parameters:
0: Boolean value

If the current component address in the FP is 0, fetch the next component. Return the value of the end-of-file status bit of the indicated file.

7.3.23 Page (PAG)

Entry Point: .PAG
Runtime Errors: None

Stack Parameters: 0: Return address
2: Address of fP

If device type is #CN, move 'ESC X' to the record buffer; otherwise, move a form-feed to the record buffer. Increment the current component address in the FP.

7.4 EXAMPLES

Given the following Pascal program, the subsequent I/O routine calls are generated in the following order:

```
Program example (input, output);
Var i : integer
Begin
    read (i);
    write ('Hello Number ', i:1);
    writeln

End.
```

```
Push     addr(input)
Push     0
Push     6
Push     1
Call     IFD
Push     addr(input)
Call     RST
Push     addr(output)
Push     0
Push     5
Push     1
Call     IFD
Push     addr(output)
Call     RWT
Push     addr(input)
Push     addr(i)
Call     RDI
Pop      addr(input)
Push     addr(output)
Push     'Hello Number '
Push     0
Call     WRS
Push     i
Push     1
Call     WRI
Pop      addr(output)
Push     addr(output)
Call     WLN
Push     addr(input)
Call     CLO
Push     addr(output)
Call     CLO
```

7.5 EXIT ROUTINE

The only other routine which utilizes MDOS system calls is the program termination routine EXIT, which has the entry point .EXIT. The routine expects an error-code in the B- or A-register. If both are zero, no error has occurred. If the B-register is non-zero, that value is the error code. If the B-register is zero, an MDOS I/O error has occurred, in which case the A-register contains the error code and the X-register points to the IOCB of the offending file. The MDOS system calls which are used are MDERR, DSPLY, and MDENT.

7.6 I/O UTILITY ROUTINES

Table 7-2 describes all of the MDOS system calls that the I/O routines make, as well as the assembly-language utilities and routines they reference. The three utility routines are described in the following paragraphs. Note that the four routines which reference .RNXT also all reference .RNXT2.

7.6.1 Validate (VLDT)

Entry Point: .VLDT
Runtime Errors: 24

Stack Parameters: 0: Return Address

Exit Parameters: None

Register Values: Entry: Y: Address of File Pointer (FP)

Exit: Y: Address of File Descriptor (FD)
U: Address of File Pointer (FP)
A: MSB of Status

If the current component address in the FP is not 0, then return. Otherwise, compare next component address in the FD with the record end address. If the next component address is greater than or equal, then read the next record, check for EOF, and reset the next component address and the record end address (end of valid data plus one). Store next component address as the current component address in the FP. Increment next component address by the component size and store. If a text file, check if current character is a carriage return; if it is, replace it by a space, and set EOL.

TABLE 7-2. External References

	MDOS SYSTEM CALLS										ASSEM. ROUT.				
	.PFNAM	.RESRV	.OPEN	.GETRC	.GETLS	.PUTRC	.PUTLS	.CLOSE	.RELES	.DIRSM	.VLDT	.RNXT	.WVLD	.CLO	.EXIT
IFD	*														*
AFI	*													*	*
RST		*	*					*	*						*
RWT		*	*					*	*	*					*
CLO						*	*	*	*	*					*
GET											*				*
PEE											*				*
PUT						*	*								*
RDC											*				*
RDB											*	*			*
RDI											*	*			*
RDS											*	*			*
RDV											*	*			*
WRC													*		
WRB													*		
WRI													*		
WRS													*		
WRV													*		
RLN											*				*
WLN						*									*
EOL											*				
EOF											*				
VLDT				*	*										
RNXT											*				
WVLD															

7.6.2 Read Next (RNXT)

Entry Points: .RNXT, .RNXT2
Runtime Errors: 22

Stack Parameters: 0: Return Address

Exit Parameters: None

Register Values:
Entry: Y: Address of File Pointer (FP)
U: Address of File Descriptor (FD)

Exit: Y: Address of File Pointer (FP)
U: Address of File Descriptor (FD)
B: Current Character
A: MSB of Status

File must be a text file. If the next character address is greater than or equal to the record end address, then clear the current component address in the FP, call VLDT, and check for EOF. Otherwise, store the next component address in the current component address in the FP, and then increment the next component address by one. (Second entry point is here.) Increment the position counter. Get the character. If it is a carriage return, replace it by a space and set EOL status.

7.6.3 Write Validate (WVLD)

Entry Point: .WVLD
Runtime Errors: None

Stack Parameters: 0: Return address

Exit Parameters: None

Register Values:
Entry: Y: Address of File Pointer (FP)

Exit: Y: Address of File Pointer (FP)

File must be a text file. If the next component address is greater than the record end address, then return. Otherwise, store the next component address in the current component address in the FP, and then increment the next component address by one. Increment the position counter.

7.7 EXAMPLES OF NON-MDOS ROUTINES

Following is an example of the runtime routines needed to write characters to an ACIA in an EXORciser environment without MDOS. The routines which need to be modified are IFD, RWT, WRC, and CLO. Each of these assembly language routines contains a header which describes the function of that routine. Note that the IFD routine, for the sake of generality, still creates a file descriptor which the other routines access.

Also included is a very short Pascal program which utilizes the ACIA routines. Note that the phase 2 listing refers to routines .IFD, .RWT, .WRC, and .CLO. To be completely MDOS-independent, a new EXIT routine which does not reference any MDOS system calls would also have to be provided.

```
Mctorola M6809 X-Assembler  1.10  FIX :   5.        .IFD     .SA 02/17/83 13:32:34

P                  ***********************************************************************
P                  *         This routine will initialize a file descriptor for          *
P                  *         non-MDOS I/O to the terminal ACIA.  The file type           *
P                  *         must be text.  The file descriptor is allocated on          *
P                  *         the stack, overlaying the passed parameters.                *
P                  ***********************************************************************
P                  *
P                  IFD       IDNT      1,0                 INIT FILE DESCRIPTOR
P                            XDEF      .IFD                ENTRY POINT
P                  *
P                  *         EQUATES
P                  *
A         FCF4     ACIA      EQU       $FCF4               TERMINAL ACIA ADDRESS
A         0015     CTRLV     EQU       $15                 ACIA CONTROL REGISTER VALUE
P                  *
P                  *         STRUCTURE OF THE FILE DESCRIPTOR
P                  *
P                  *         6-BYTE CONTROL BLOCK WITH THE FOLLOWING FORMAT:
P                  *
P                  *         OFFSET:   0: ACIA ADDRESS
P                  *                   2: ACIA CONTROL REG VALUE
P                  *                   3: CHAR BUFFER
P                  *                   4: STATUS
P                  *
P                  *         * * * * * * * * * * * * * * * * * *
P                  *
P                  *         STATUS OF STACK:
P                  *
P                  *         ENTRY:    0: RETURN ADDRESS
P                  *                   2: COMPONENT SIZE (ALWAYS ONE)
P                  *                   4: INITIAL STATUS
P                  *                   6: PARAMETER POSITION (N/A)
P                  *                   8: ADDR CF FILE POINTER
P                  *
P                  *         EXIT:     0: INITIALIZED FILE DESCRIPTOR
P                  *
P                  *         * * * * * * * * * * * * * * * * * * *
P                  *
P 0000 AE68        .IFD      LDX       8,S                 ADDRESS OF FILE POINTER
P 0002 3364                  LEAU      4,S                 ADDRESS OF DESCRIPTOR (ON STACK)
P 0004 EF02                  STU       2,X                 INIT FD ADDR IN FP
P 0006 3343                  LEAU      3,U                 CHAR BUFFER ADDRESS
P 0008 EF84                  STU       0,X                 INIT CHAR PTR IN FP
P 000A EC64                  LDD       4,S                 GET STATUS
P 000C ED68                  STD       8,S                 STORE IN FD
P 000E CCFCF4                LDD       #ACIA               GET ACIA ADDRESS
P 0011 ED64                  STD       4,S                 STORE IN FD
P 0013 C615                  LDB       #CTRLV              GET CONTROL REG VALUE
P 0015 E766                  STB       6,S                 STORE IN FD
P 0017 3540                  PULS      U                   GET RET ADDR
P 0019 3262                  LEAS      2,S                 DISCARD EXTRA BYTES
P 001B 6EC4                  JMP       0,U                 RETURN
P                            END

***** TOTAL ERRORS          0--   0
***** TOTAL WARNINGS        0--   0
```

```
Mctorola M6809 X-Assembler  1.10  FIX :   5.        .RWT      .SA 02/17/83 13:56:40

  1 P                 ******************************************************
  2 P                 *        This routine will initialize the ACIA pointed  *
  3 P                 *        to by the passed file pointer.                 *
  4 P                 ******************************************************
  5 P                 RWT      IDNT      1.0              REWRITE FILE
  6 P                          XDEF      .RWT             ENTRY POINT
  7 P                 *
  8 P                 *        FILE DESCRIPTOR OFFSETS
  9 P                 *                                   ACIA ADDRESS
 10 A         0000    ACIA     EQU       0                CONTROL REGISTER
 11 A         0002    CNTLV    EQU       2                CHAR BUFFER
 12 A         0003    CHBUF    EQU       3                STATUS
 13 A         0004    STATUS   EQU       4
 14 P                 *
 15 P                 *        STATUS OF STACK
 16 P                 *
 17 P                 *        ENTRY:     0: RETURN ADDRESS
 18 P                 *                   2: ADDRESS OF FILE POINTER
 19 P                 *
 20 P 0000 EE62       .RWT     LDU       2,S              ADDRESS OF FILE POINTER
 21 P 0002 EE42                LDU       2,U              ADDRESS OF FILE DESCRIPTOR
 22 P 0004 AEC4                LDX       ACIA,U           GET ACIA ADDRESS
 23 P 0006 C603                LDB       #3               RESET ACIA
 24 P 0008 E784                STB       0,X
 25 P 000A E642                LDB       CNTLV,U          INIT CONTROL REG
 26 P 000C E784                STB       0,X
 27 P 000E 3043                LEAX      CHBUF,U          ADDRESS OF CHAR BUFFER
 28 P 0010 AFF802              STX       [2,S]            RESET BUFFER POINTER
 29 P 0013 3540                PULS      U                GET RETURN ADDR
 30 P 0015 3262                LEAS      2,S              DISCARD PARAMETER
 31 P 0017 6EC4                JMP       0,U              RETURN
 32 P                          END

***** TOTAL ERRORS      0--     0
***** TOTAL WARNINGS    0--     0
```

```
Motorola M6809 X-Assembler  1.10  FIX :   5.        .WRC      .SA 02/17/83 14:11:14

  1 P                *****************************************************************
  2 P                *         This routine will write a character to the ACIA      *
  3 P                *         preceded by the appropriate number of spaces.        *
  4 P                *****************************************************************
  5 P                WRC       IDNT      1.0                 WRITE CHARACTER
  6 P                          XDEF      .WRC                ENTRY POINT
  7 P                *
  8 P                *         FILE DESCRIPTOR OFFSETS
  9 P                *
 10 A      0000      ACIA      EQU       0                   ACIA ADDRESS
 11 A      0002      CNTLV     EQU       2                   CONTROL REGISTER
 12 A      0003      CHBUF     EQU       3                   CHAR BUFFER
 13 A      0004      STATUS    EQU       4                   STATUS
 14 P                *
 15 P                *         STATUS OF STACK
 16 P                *
 17 P                *         ENTRY:    0: RETURN ADDRESS
 18 P                *                   2: FIELD WIDTH
 19 P                *                   4: CHARACTER VALUE
 20 P                *                   5: ADDRESS OF FILE POINTER
 21 P                *
 22 P                *         EXIT:     0: ADDRESS OF FILE POINTER
 23 P                *
 24 P 0000 AE65      .WRC      LDX       5,S                 ADDRESS OF FILE POINTER
 25 P 0002 AE02                LDX       2,X                 ADDRESS OF FILE DESCRIPTOR
 26 P 0004 AE84                LDX       ACIA,X              GET ACIA ADDRESS
 27 P 0006 EC62                LDD       2,S                 GET FIELD WIDTH
 28 P 0008 2F0D                BLE       WRC04               ZERO OR NEGATIVE - NO SPACE.
 29 P 000A 5A        WRC02     DECB                          DECREMENT FIELD WIDTH
 30 P 000B 270A                BEQ       WRC04               NO SPACE NEEDED
 31 P 000D 3404                PSHS      B                   SAVE NEW FIELD WIDTH
 32 P 000F C620                LDB       #'                  GET A SPACE
 33 P 0011 8D0E                BSR       WRITE               OUTPUT SPACE
 34 P 0013 3504                PULS      B                   RECOVER FIELD WIDTH
 35 P 0015 20F3                BRA       WRC02               LOOP
 36 P 0017 E664      WRC04     LDB       4,S                 GET CHAR VALUE
 37 P 0019 8D06                BSR       WRITE               OUTPUT CHARACTER
 38 P 001B 3540                PULS      U                   GET RETURN ADDR
 39 P 001D 3263                LEAS      3,S                 DISCARD MOST PARAMETERS
 40 P 001F 6EC4                JMP       0,U                 RETURN
 41 P 0021 A684      WRITE     LDA       0,X                 GET ACIA STATUS
 42 P 0023 8402                ANDA      #2                  TDR EMPTY?
 43 P 0025 27FA                BEQ       WRITE               NO
 44 P 0027 E701                STB       1,X                 OUTPUT CHAR
 45 P 0029 39                  RTS                           RETURN
 46 P                          END

***** TOTAL ERRORS         0--   0
***** TOTAL WARNINGS       0--   0
```

```
Mctorola M6809 X-Assembler  1.10  FIX :   5.          .CLO       .SA 02/17/83 14:16:20

  1 P                     ****************************************************************
  2 P                     *          This routine will close the file which means         *
  3 P                     *          for an ACIA do nothing.                              *
  4 P                     ****************************************************************
  5 P                     CLO        IDNT      1.0                      CLOSE FILE
  6 P                                XDEF      .CLO                     ENTRY POINT
  7 P                     *
  8 P                     *          FILE DESCRIPTOR OFFSETS
  9 P                     *
 10 A          0000       ACIA       EQU       0                        ACIA ADDRESS
 11 A          0002       CNTLV      EQU       2                        CONTROL REGISTER
 12 A          0003       CHBUF      EQU       3                        CHAR BUFFER
 13 A          0004       STATUS     EQU       4                        STATUS
 14 P                     *
 15 P                     *          STATUS OF STACK
 16 P                     *
 17 P                     *          ENTRY:    0: RETURN ADDRESS
 18 P                     *                    2: ADDRESS OF FILE POINTER
 19 P                     *
 20 P 0000 3510           .CLO       PULS      X                        GET RETURN ADDRESS
 21 P 0002 3262                      LEAS      2,S                      DISCARD FP ADDRESS
 22 P 0004 6E84                      JMP       0,X                      RETURN
 23 P                                END

***** TOTAL ERRORS        0--    0
***** TOTAL WARNINGS      0--    0
```

```
Line    Loc Lev BE M6809 Cross Pascal 1.20     TESTPROG.SA 02/17/83 14:22:16

   1(     -4)  0)-- PROGRAM testprog (output);
   2(     -4)  0)--
   3(     -4)  0)-- VAR
   4(    -20)  0)--      message: ARRAY [1..16] OF char;
   5(    -22)  0)--      i:       integer;
   6(    -22)  0)--
   7         1  0)A- BEGIN
   8         2  0)--     message := 'ACIA Test Output';
   9         3  0)--     write(output,message[1]:5);
  10         4  0)--     FOR i := 2 TO 16 DO write(output,message[i]:1);
  11         6  0)--     write(output,chr(13):1); {CR}
  12         7  0)--     write(output,chr(10):1); {LF}
  13            0)-A     END.

       **** No Error(s) and No Warning(s) detected

       **** 13 Lines 0 Procedures

       **** 86 Pcode instructions
```

```
                          *M6809 Code Generator  1.20
                          *M6809 Cross Pascal 1.20   TESTPROG.SA 02/17/83 14:22:16
                          *        PROGRAM testprog (output);
                          *
                          *        VAR
                          *           message: ARRAY [1..16] OF char;
                          *           i:         integer;
                          *
                          *        BEGIN
                          *           message := 'ACIA Test Output';
0000    17  0000               LBSR    .ENT
0003        00                 FCB     0
0004        0000               FDB     L1
0006    30  3C                 LEAX    -4,Y
0008    34  10                 PSHS    X
000A    5F                     CLRB
000B    4F                     CLRA
000C    34  06                 PSHS    D
000E    C6  05                 LDB     #5
0010    34  06                 PSHS    D
0012    C6  01                 LDB     #1
0014    34  06                 PSHS    D
0016    17  0000               LBSR    .IFD
0019    30  3C                 LEAX    -4,Y
001B    34  10                 PSHS    X
001D    17  0000               LBSR    .RWT
0020    30  8C 02              LEAX    *+5,PCR
0023    20  10                 BRA     *+18
0025        41                 FCC     16,ACIA Test Output
0035    CC  0010               LDD     #16
0038    17  0000               LBSR    .LODV
003B    30  A8 EC              LEAX    -20,Y
003E    CC  0010               LDD     #16
0041    17  0000               LBSR    .STRV
                          *            write(output,message[1]:5);
0044    30  3C                 LEAX    -4,Y
0046    34  10                 PSHS    X
0048    E6  A8 EC              LDB     -20,Y
004B    34  04                 PSHS    B
004D    CC  0005               LDD     #5
0050    34  06                 PSHS    D
0052    17  0000               LBSR    .WRC
0055    32  62                 LEAS    2,S
                          *            FOR i := 2 TO 16 DO write(output,message[i]:1);
0057    CC  0002               LDD     #2
005A    ED  A8 EA              STD     -22,Y
005D    C6  10                 LDB     #16
005F    ED  A8 E8              STD     -24,Y
0062 10A3   A8 EA              CMPD    -22,Y
0066 102D   0000               LBLT    L3
006A                     L2    EQU     *
006A    30  3C                 LEAX    -4,Y
006C    34  10                 PSHS    X
006E    30  A8 EB              LEAX    -21,Y
0071    EC  A8 EA              LDD     -22,Y
0074    E6  8B                 LDB     D,X
0076    34  04                 PSHS    B
0078    CC  0001               LDD     #1
007B    34  06                 PSHS    D
007D    17  0000               LBSR    .WRC
0080    32  62                 LEAS    2,S
0082    EC  A8 EA              LDD     -22,Y
0085 10A3   A8 E8              CMPD    -24,Y
```

```
0089 1027  0000            LBEQ   L3
008D   C3  0001            ADDD   #1
0090   ED  AB EA           STD    -22,Y
0093   20  D5              BRA    *-41
0095                  L3   EQU    *
                      *           write(output,chr(13):1); {CR}
0095   30  3C              LEAX   -4,Y
0097   34  10              PSHS   X
0099   C6  0D              LDB    #13
009B   34  04              PSHS   B
009D   CC  0001            LDD    #1
00A0   34  06              PSHS   D
00A2   17  0000            LBSR   .WRC
00A5   32  62              LEAS   2,S
                      *           write(output,chr(10):1); {LF}
00A7   30  3C              LEAX   -4,Y
00A9   34  10              PSHS   X
00AB   C6  0A              LDB    #10
00AD   34  04              PSHS   B
00AF   CC  0001            LDD    #1
00B2   34  06              PSHS   D
00B4   17  0000            LBSR   .WRC
00B7   32  62              LEAS   2,S
                      *           END.
00B9   30  3C              LEAX   -4,Y
00BB   34  10              PSHS   X
00BD   17  0000            LBSR   .CLO
00C0                  L1   EQU    24
00C0   CC  0000            LDD    #0
00C3   17  0000            LBSR   .EXIT
00C6                       END
```

```
Motorola 8-bit Cross Linkage Editor Version  1.01  02/17/83  14:29:30  Page 1


Command Line:

LINK TESTPROG,TESTPROG,TESTPROG;AIMXL=PAS09LIB.RX


Options in Effect:  A,-B,-H,I,L,M,Q,-U,X


User Commands:

LOCATE PSCT $1000
LOCATE DSCT $4000
DEF .DHIGH $47FF
DEF .SIZE 0
IN IFD.RX
IN RWT.RX
IN WRC.RX
IN CLO.RX
END
```

```
Load Map:

Module          S  T  Start     End        Externally Defined Symbols

TESTPR          P     00001000  000010C5   .ENTRY     00001000
IFD             P     000010C6  000010E2   .IFD       000010C6
RWT             P     000010E3  000010FB   .RWT       000010E3
WRC             P     000010FC  00001125   .WRC       000010FC
CLO             P     00001126  0000112B   .CLO       00001126
INIT            P     0000112C  00001197   .INIT      00001132   .INITS     0000112C
ENT             P     00001198  000011BF   .ENT       00001198
LODV            P     000011C0  000011E2   .LODV      000011C0
STRV            P     000011E3  00001201   .STRV      000011E3
OVRFL           P     00001202  00001215   .OVRFL     00001202
EXIT            P     00001216  00001273   .EXIT      00001218   .EXITI     00001216
CVHEX           P     00001274  0000128A   .CVHEX     00001274
.ENDD           D  C  00004000  00004000

Table of Externally Defined Symbols:

Name          Address     Module          Displ       Sect Library         Input

.CLO          00001126    CLO             00000000    P                    CLO     .RX
.CVHEX        00001274    CVHEX           00000000    P    PAS09LIB.RX
.DHIGH        000047FF    USER DEFINED
.ENT          00001198    ENT             00000000    P    PAS09LIB.RX
.ENTRY        00001000    TESTPR          00000000    P                    TESTPROG.RX
.EXIT         00001218    EXIT            00000002    P    PAS09LIB.RX
.EXITI        00001216    EXIT            00000000    P    PAS09LIB.RX
.IFD          000010C6    IFD             00000000    P                    IFD     .RX
.INIT         00001132    INIT            00000006    P    PAS09LIB.RX




Motorola 8-bit Cross Linkage Editor Version   1.01   02/17/83   14:29:30   Page 2

.INITS        0000112C    INIT            00000000    P    PAS09LIB.RX
.LODV         000011C0    LODV            00000000    P    PAS09LIB.RX
.OVRFL        00001202    OVRFL           00000000    P    PAS09LIB.RX
.RWT          000010E3    RWT             00000000    P                    RWT     .RX
.SIZE         00000000    USER DEFINED
.STRV         000011E3    STRV            00000000    P    PAS09LIB.RX
.WRC          000010FC    WRC             00000000    P                    WRC     .RX

Unresolved References: None

Multiply Defined Symbols: None

   No Errors
   No Warnings

S-record module has been created.
```

7.8 PASCAL AND INTERRUPTS

There are certain restrictions regarding what can be done with Pascal in an interrupt environment.

a. Nearly all of the system runtime routines utilize the Y register. If an interrupt occurs during a system routine, the Y register is not likely to contain the address of the global data area. If a Pascal module is to be called to service the interrupt, it will be necessary to load the Y register with the global data address (found in the RMA as the display level zero pointer) before calling the routine.

b. The I/O routines, as well as NEW and DISPOSE, are not re-entrant. If an interrupt occurs while either NEW or DISPOSE is modifying the heap pointer and the freelist links, and then a subsequent call to NEW or DISPOSE is made in the course of the interrupt processing, the result is unpredictable but quite likely disastrous.

c. MDOS system calls will clear the DP register. If an interrupt occurs during an MDOS system utility, the DP register will have to be reset to the proper value before a Pascal routine can be called to process the interrupt. The correct value of the DP register must be known by the user from the load map, or must have been saved prior to the interrupt.

d. The statement counter may temporarily have an erroneous value upon returning from an interrupt processed by a Pascal routine.

CHAPTER 8

EXISTING FLOATING POINT SUPPORT

8.1 GENERAL

The compiler will accept real integers and real function calls and generate the appropriate intermediate code and runtime routine calls. However, when linked, errors will occur stating that the floating point runtime routines are not available. Paragraph 8.3 contains a list of the runtime routines which eventually will be provided for M6809 Cross Pascal.

8.2 STANDARD TYPES

The M6809 cross Pascal compiler on EXORmacs supports three precisions of floating point values:

REAL	Single precision 32-bit IEEE format
DREAL	Double precision 64-bit IEEE format
XREAL	Extended precision 80-bit IEEE format

(See the EXORmacs Resident Pascal User's Manual, M68KPASC, for exact details of the floating point representation.)

Not supported are the following:

INFINITY
NAN
ROUNDING MODES
ROUNDING PRECISION
INFINITY CLOSURE MODES
EXCEPTION MODES
FLOATING POINT CONTROL BLOCK

8.3 CALLING SEQUENCE

The code generator (Phase 2) translates all floating point operations into runtime library calls. Operands are pushed onto the hardware stack and the result is (usually) returned on the hardware stack. The code generator assumes that the values of the Y-register and the DP-register are preserved by the library routine, and the CC-register is usually left indeterminate. (The Y-register contains the address of the global data area, and the DP-register contains the address of the runtime maintenance area.) Some routines are also expected to save the values in the X-register and the U-register. The D-register is always considered volatile.

The following entry points are referenced by the code generator. The suffix 'R' indicates REAL; 'W' indicates DREAL; and 'X' indicates XREAL.

Each of the following routines is expected to preserve the DP, U, X, and Y registers.

.ABR	.ABW	.ABX	Absolute value
.ADR	.ADW	.ADX	Addition
.CMPR	.CMPW	.CMPX	Comparison
.DVR	.DVW	.DVX	Division
.MPR	.MPW	.MPX	Multiplication
.NGR	.NGW	.NGX	Negation
.REMR	.REMW	.REMX	Remainder
.SBR	.SBW	.SBX	Subtraction
.SQRR	.SQRW	.SQRX	Square
.SQTR	.SQTW	.SQTX	Square root
.CVTHR	.CVTHW	.CVTHX	Convert 1-byte integer to real
.CVTIR	.CVTIW	.CVTIX	Convert 2-byte integer to real
.CVTJR	.CVTJW	.CVTJX	Convert 4-byte integer to real
.CVTRW	.CVTWR		Convert between single/double
.CVTRX	.CVTXR		Convert between single/extended
.CVTWX	.CVTXW		Convert between double/extended
.CVBHR	.CVBHW	.CVBHX	Convert below 1-byte integer to real
.CVBIR	.CVBIW	.CVBIX	Convert below 2-byte integer to real
.CVBJR	.CVBJW	.CVBJX	Convert below 4-byte integer to real
.CVBRW			Convert below single to double
.CVBRX			Convert below single to extended
.CVBWX			Convert below doublt to extended

Each of the following routines is expected to preserve the DP and Y registers.

.ATNR	.ATNW	.ATNX	Inverse tangent
.COSR	.COSW	.COSX	Cosine
.EXPR	.EXPW	.EXPX	Exponential
.LOGR	.LOGW	.LOGX	Natural logarithm
.PWRR	.PWRW	.PWRX	Power
.RNDR	.RNDW	.RNDX	Round to 4-byte integer
.SINR	.SINW	.SINX	Sine
.TANR	.TANW	.TANX	Tangent
.TRCR	.TRCW	.TRCX	Truncate to 4-byte integer
.RDR	.RDW	.RDX	Read
.WRR	.WRW	.WRX	Write

CHAPTER 9

M6809-PASCAL LIMITATIONS

9.1 EXPRESSION COMPLEXITY

During Phase 1 of a Pascal compilation, expressions are translated to a reverse Polish form. The form uses a push-down stack for the operands, based on the precedence of the operators. If the precedence of the current operator is less than that of the next operator, pushing continues. The operators then operate on the top one or two operands on the stack, leaving the result on the top of the stack.

Phase 2 simulates the expression stack, using the processor's hardware stack and registers. It loads operands onto the stack -- actually into the processor's registers -- and then performs the appropriate operation. When the processor wants to load an operand but the appropriate register is in use, it pushes the current contents of the register(s) onto the hardware stack in order to free a register.

To remember what is on the expression stack, Phase 2 maintains a 32-element array. Each element of the array describes one data item on the hardware stack. This limits the complexity of expressions that Phase 2 can handle to 32 levels of parentheses. When the array overflows, Phase 2 emits an error message of EXPR STACK OVERFLOW.

A scalar (integers, Booleans, characters, enumerated types, etc.) is put in the D register. Pointers are put in the X register. Sets, strings, records, and arrays are always pushed directly onto the hardware stack. Each requires only one element of the expression stack array.

9.2 DATA STRUCTURES

The amount of memory that can be allocated for the global variables of an M6809 Pascal program is limited to 32000 bytes. Likewise, the amount of memory that can be allocated for the local variables of each procedure or function is limited to 32000 bytes. The maximum size of any single data structure (array or record) is similarly limited to 32000 bytes.

String constants are limited to a maximum of 64 characters. Strings are limited to 254 characters. Sets are fixed at eight bytes, which is 64 items. The packed attribute has no effect on data allocation.

9.3 PROGRAM CODE

The amount of memory that can be allocated for the code for each procedure, function, or main program is limited to 32000 bytes.

The standard procedures pack and unpack are not implemented.

Procedure or function identifiers may not be passed as parameters. Level one procedure and function identifiers must differ over the first six characters. The maximum number of procedures and functions which can be declared within one compilation module is limited to 400.

During the processing of a Pascal program, Phase 1 of the compiler generates labels in the intermediate file for each Pascal statement. For example, an if...then...else statement will require two generated labels. The maximum number of compiler-generated and user-defined labels that Phase 2 is capable of handling is limited to 400 for each procedure or function. If the label table overflows, an error message is displayed and Phase 2 will abort. The user can correct this problem by subdividing the offending procedure or function into two or more subprocedures.

During program execution, overflow checking is not performed during expression evaluation even if runtime checking is enabled.

CHAPTER 10

SAMPLE PROGRAM COMPILATION AND EXECUTION

10.1 COMPILER PHASE 1 LISTING

```
Line     Loc Lev BE M6809 Cross Pascal 1.20   QUEENS  .SA 02/17/83 15:15:22

   1(      -8) 0)-- PROGRAM  queens (input,output,listing);
   2(      -8) 0)--
   3(      -8) 0)--   {Using backtracking, this program prints all possible placements
   4(      -8) 0)--    of n queens on an  n x n  chessboard so that they are nonattacking}
   5(      -8) 0)--
   6(      -8) 0)--   CONST  maxsize = 15;   {maximum size of chessboard}
   7(      -8) 0)--
   8(      -8) 0)--   VAR
   9(      -8) 0)--      n,       {board size}
  10(      -8) 0)--      row,     {current row}
  11(     -14) 0)--      i        {loop index}                   : integer;
  12(     -44) 0)--      col      {column of particular row} : ARRAY [1 .. maxsize] OF integer;
  13(     -48) 0)--      listing  {file to print results}    : text;
  14(     -48) 0)--
  15(       0) 1)--   FUNCTION  place (k: integer): boolean;
  16(       0) 1)--
  17(       0) 1)--      {This function returns TRUE if a queen can be placed in the k'th
  18(       0) 1)--       row and col[k]'th column.  Otherwise, it returns FALSE.  col is
  19(       0) 1)--       a global array whose first k-1 values have been set.}
  20(       0) 1)--
  21(       0) 1)--      VAR
  22(      -1) 1)--         failed   {failed to place a queen} : boolean;
  23(      -3) 1)--         i        {loop index}              : integer;
  24(      -3) 1)--
  25         1  1)A-      BEGIN {place}
  26         2  1)--         failed := false;
  27         3  1)--         i := 1;
  28         4  1)--         WHILE (i < k) AND (NOT failed) DO
  29            1)B-            BEGIN  {check for two in same column or two in same diagonal}
  30         5  1)--               failed :=   (col[i] = col[k])
  31            1)--                        OR (abs(col[i] - col[k]) = abs(i-k));
  32         6  1)--               IF NOT failed
  33         7  1)--                  THEN  i := i + 1  {go on to next check}
  34            1)-B            END; {WHILE}
  35         8  1)--         place := NOT failed  {set up return value}
  36            1)-A      END; {place}
  37            1)--
  38         9  0)A-   BEGIN {queens}
  39        10  0)--      rewrite (listing);
  40        11  0)--      writeln ('SPECIFY SIZE OF BOARD:');
  41        12  0)--      readln  (n);
  42        13  0)--      IF  (n <= 0) OR (n > maxsize)
  43        14  0)--         THEN  writeln ('INVALID BOARD SIZE')
  44            0)--         ELSE
  45            0)B-            BEGIN
  46        15  0)--               writeln (listing);  writeln (listing);
  47        17  0)--               writeln (listing);  writeln (listing);
  48        19  0)--               writeln (listing,'PLACEMENTS OF QUEENS FOR A  ',n:1,
  49            0)--                        ' X ',n:1,'  BOARD:');
  50        20  0)--               writeln (listing);
  51        21  0)--               col[1] := 0;   {col[x] is current column for row x}
  52        22  0)--               row    := 1;
  53        23  0)--               WHILE row > 0 DO  {for all rows do}
  54            0)C-                  BEGIN
  55        24  0)--                     col[row] := col[row] + 1; {move to next column}
  56        25  0)--                     WHILE (col[row] <= n) AND (NOT place(row)) DO
```

```
Line   Loc Lev BE M6809 Cross Pascal 1.20    QUEENS   .SA 02/17/83 15:15:22

  57     26  0)--                            col[row] := col[row] + 1;  {try next column}
  58     27  0)--                         IF col[row] <= n  {a position is found}
  59         0)--                            THEN
  60     28  0)--                               IF row = n  {is a solution completed?}
  61         0)--                                  THEN     {yes - print it}
  62         0)D-                                     BEGIN
  63     29  0)--                                        FOR i := 1 TO n DO
  64     30  0)--                                           write (listing,col[i]:4);
  65     31  0)--                                        writeln (listing)
  66         0)-D                                     END {THEN}
  67         0)--                                  ELSE     {no - go to next row}
  68         0)D--                                    BEGIN
  69     32  0)--                                        row := row + 1;
  70     33  0)--                                        col[row] := 0
  71         0)-D                                     END {ELSE}
  72     34  0)--                            ELSE  row := row - 1  {backtrack}
  73         0)-C                      END; {WHILE}
  74     35  0)--                   writeln (listing);
  75     36  0)--                   writeln ('SEARCH COMPLETE')
  76         0)-B                END {ELSE}
  77         0)-A         END. {queens}

        **** No Error(s) and No Warning(s) detected

        **** 77 Lines 1 Procedures

        **** 320 Pcode instructions
```

10.2 COMPILER PHASE 2 LISTING

```
                                 *M6809 Code Generator  1.20
                                 *M6809 Cross Pascal 1.20   QUEENS  .SA 02/17/83 15:15:22
                                 *        PROGRAM  queens (input,output,listing);
                                 *
                                 *           {Using backtracking, this program prints all possible placements
                                 *            of n queens on an  n x n  chessboard so that they are nonattacking}
                                 *
                                 *           CONST  maxsize = 15;     {maximum size of chessboard}
                                 *
                                 *           VAR
                                 *              n,         {board size}
                                 *              row,       {current row}
                                 *              i          {loop index}                : integer;
                                 *              col        {column of particular row}  : ARRAY [1 .. maxsize] OF integer;
                                 *              listing    {file to print results}     : text;
                                 *
                                 *           FUNCTION  place (k: integer): boolean;
                                 *
                                 *              {This function returns TRUE if a queen can be placed in the k'th
                                 *               row and col[k]'th column.  Otherwise, it returns FALSE.  col is
                                 *               a global array whose first k-1 values have been set.}
                                 *
                                 *              VAR
                                 *                 failed   {failed to place a queen} : boolean;
                                 *                 i        {loop index}              : integer;
                                 *
                                 *              BEGIN {place}
                                 *                 failed := false;
0000    17  0000                      LBSR    .ENT
C003        01                        FCB     1
C004        0000                      FDB     L1
0006    5F                            CLRB
0007    DE  02                        LDU     2
C009    E7  5F                        STB     -1,U
                                 *                 i := 1;
0008    CC  0001                      LDD     #1
000E    ED  5D                        STD     -3,U
                                 *                 WHILE (i < k) AND (NOT failed) DO
C010                             L2   EQU     *
0010    DE  02                        LDU     2
0012    EC  5D                        LDD     -3,U
C014 10A3   46                        CMPD    6,U
0017    2D  03                        BLT     *+5
0019    5F                            CLRB
001A    20  02                        BRA     *+4
001C    C6  01                        LDB     #1
001E    34  04                        PSHS    B
0020    E6  5F                        LDB     -1,U
C022    C8  01                        EORB    #1
C024    E4  E0                        ANDB    0,S+
0026 1027   0000                      LBEQ    L3
                                 *                    BEGIN  {check for two in same column or two in same diagonal}
                                 *                       failed :=    (col[i] = col[k])
                                 *                                 OR (abs(col[i] - col[k]) = abs(i-k));
002A    30  A8 D2                     LEAX    -46,Y
002D    EC  5D                        LDD     -3,U
002F    58                            ASLB
0030    49                            ROLA
C031    EC  8B                        LDD     D,X
0033    34  06                        PSHS    D
0035    EC  46                        LDD     6,U
C037    58                            ASLB
0038    .49                           ROLA
```

```
0039    EC   8B                 LDD    D,X
003B  10A3   E1                 CMPD   0,S++
003E    27   03                 BEQ    *+5
0040    5F                      CLRB
0041    20   02                 BRA    *+4
0043    C6   01                 LDB    #1
0045    34   04                 PSHS   B
0047    EC   5D                 LDD    -3,U
0049    58                      ASLB
004A    49                      ROLA
004B    EC   8B                 LDD    D,X
004D    34   06                 PSHS   D
004F    EC   46                 LDD    6,U
0051    58                      ASLB
0052    49                      ROLA
0053    30   8B                 LEAX   D,X
0055    35   06                 PULS   D
0057    A3   84                 SUBD   0,X
0059    2A   05                 BPL    *+7
005B    43                      COMA
005C    53                      COMB
005D    C3   0001               ADDD   #1
0060    34   06                 PSHS   D
0062    EC   5D                 LDD    -3,U
0064    A3   46                 SUBD   6,U
0066    2A   05                 BPL    *+7
0068    43                      COMA
0069    53                      COMB
006A    C3   0001               ADDD   #1
006D  10A3   E1                 CMPD   0,S++
0070    27   03                 BEQ    *+5
0072    5F                      CLRB
0073    20   02                 BRA    *+4
0075    C6   01                 LDB    #1
0077    EA   E0                 ORB    0,S+
0079    E7   5F                 STB    -1,U
                      *                     IF NOT failed
                      *                        THEN  i := i + 1  {go on to next check}
007B    C8   01                 EORB   #1
007D  1027   0000               LBEQ   L4
                      *                  END; {WHILE}
0081    EC   5D                 LDD    -3,U
0083    C3   0001               ADDD   #1
0086    ED   5D                 STD    -3,U
0088                  L4        EQU    *
0088    20   86                 BRA    *-120
008A                  L3        EQU    *
                      *             place := NOT failed  {set up return value}
                      *          END; {place}
008A    DE   02                 LDU    2
008C    E6   5F                 LDB    -1,U
008E    C8   01                 EORB   #1
0090    E7   48                 STB    8,U
0092                  L1        EQU    3
0092    17   0000               LBSR   .RET
0095         01                 FCB    1
0096         0002               FDB    2
                      *
                      *         BEGIN {queens}
                      *            rewrite (listing);
0098    17   0000               LBSR   .ENT
009B         00                 FCB    0
009C         0000               FDB    L5
009E    30   A8 D0              LEAX   -48,Y
00A1    34   10                 PSHS   X
00A3    CC   0001               LDD    #1
```

```
C0A6   34  06           PSHS   D
00A8   C6  04           LDB    #4
00AA   34  06           PSHS   D
00AC   C6  01           LDB    #1
00AE   34  06           PSHS   D
00B0   17  0000         LBSR   .IFD
00B3   30  38           LEAX   -8,Y
00B5   34  10           PSHS   X
00B7   5F               CLRB
00B8   4F               CLRA
00B9   34  06           PSHS   D
00BB   C6  05           LDB    #5
00BD   34  06           PSHS   D
00BF   C6  01           LDB    #1
00C1   34  06           PSHS   D
00C3   17  0000         LBSR   .IFD
00C6   30  38           LEAX   -8,Y
00C8   34  10           PSHS   X
00CA   17  0000         LBSR   .RWT
00CD   30  3C           LEAX   -4,Y
00CF   34  10           PSHS   X
00D1   5F               CLRB
C0D2   4F               CLRA
00D3   34  06           PSHS   D
00D5   C6  06           LDB    #6
00D7   34  06           PSHS   D
00D9   C6  01           LDB    #1
00DB   34  06           PSHS   D
C0DD   17  0000         LBSR   .IFD
00E0   30  3C           LEAX   -4,Y
00E2   34  10           PSHS   X
C0E4   17  0000         LBSR   .RST
00E7   30  A8 D0        LEAX   -48,Y
00EA   34  10           PSHS   X
00EC   17  0000         LBSR   .RWT
                  *                writeln ('SPECIFY SIZE OF BOARD:');
00EF   30  38           LEAX   -8,Y
00F1   34  10           PSHS   X
00F3   30  8C 02        LEAX   *+5,PCR
00F6   20  17           BRA    *+25
00F8       16           FCB    22
00F9       53           FCC    22,SPECIFY SIZE OF BOARD:
C10F   17  0000         LBSR   .LODS
C112   5F               CLRB
0113   4F               CLRA
C114   34  06           PSHS   D
0116   17  0000         LBSR   .WRS
0119   17  0000         LBSR   .WLN
                  *                 readln  (n);
011C   30  3C           LEAX   -4,Y
011E   34  10           PSHS   X
0120   30  32           LEAX   -14,Y
0122   34  10           PSHS   X
0124   17  0000         LBSR   .RDI
0127   17  0000         LBSR   .RLN
                  *                  IF  (n <= 0) OR (n > maxsize)
                  *                       THEN  writeln ('INVALID BOARD SIZE')
012A   EC  32           LDD    -14,Y
012C   2F  03           BLE    *+5
012E   5F               CLRB
012F   20  02           BRA    *+4
0131   C6  01           LDB    #1
0133   34  04           PSHS   B
0135   EC  32           LDD    -14,Y
0137 1083  000F         CMPD   #15
013B   2E  03           BGT    *+5
```

```
013D   5F                 CLRB
013E   20  02             BRA    *+4
0140   C6  01             LDB    #1
0142   EA  E0             ORB    0,S+
0144 1027  0000           LBEQ   L6
0148   30  38             LEAX   -8,Y
014A   34  10             PSHS   X
014C   30  8C 02          LEAX   *+5,PCR
014F   20  13             BRA    *+21
0151       12             FCB    18
0152       49             FCC    18,INVALID BOARD SIZE
0164   17  0000           LBSR   .LODS
0167   5F                 CLRB
0168   4F                 CLRA
0169   34  06             PSHS   D
016B   17  0000           LBSR   .WRS
016E   17  0000           LBSR   .WLN
                    *                   ELSE
0171   16  0000           LBRA   L7
0174                L6    EQU    *
                    *                      BEGIN
                    *                        writeln (listing);  writeln (listing);
0174   30  A8 D0          LEAX   -48,Y
0177   34  10             PSHS   X
0179   17  0000           LBSR   .WLN
017C   30  A8 D0          LEAX   -48,Y
017F   34  10             PSHS   X
0181   17  0000           LBSR   .WLN
                    *                        writeln (listing);  writeln (listing);
0184   30  A8 D0          LEAX   -48,Y
0187   34  10             PSHS   X
0189   17  0000           LBSR   .WLN
018C   30  A8 D0          LEAX   -48,Y
018F   34  10             PSHS   X
0191   17  0000           LBSR   .WLN
                    *                        writeln (listing,'PLACEMENTS OF QUEENS FOR A  ',n:1
0194   30  A8 D0          LEAX   -48,Y
0197   34  10             PSHS   X
0199   30  8C 02          LEAX   *+5,PCR
019C   20  1D             BRA    *+31
019E       1C             FCB    28
019F       50             FCC    28,PLACEMENTS OF QUEENS FOR A
01BB   17  0000           LBSR   .LODS
01BE   5F                 CLRB
01BF   4F                 CLRA
01C0   34  06             PSHS   D
01C2   17  0000           LBSR   .WRS
01C5   EC  32             LDD    -14,Y
01C7   34  06             PSHS   D
01C9   CC  0001           LDD    #1
01CC   34  06             PSHS   D
01CE   17  0000           LBSR   .WRI
                    *                                  ' X ',n:1,'  BOARD:');
01D1   30  8C 02          LEAX   *+5,PCR
01D4   20  04             BRA    *+6
01D6       03             FCB    3
01D7       20             FCC    3, X
01DA   17  0000           LBSR   .LODS
01DD   5F                 CLRB
01DE   4F                 CLRA
01DF   34  06             PSHS   D
01E1   17  0000           LBSR   .WRS
01E4   EC  32             LDD    -14,Y
01E6   34  06             PSHS   D
01E8   CC  0001           LDD    #1
01EB   34  06             PSHS   D
```

```
01ED    17  0000                LBSR   .WRI
01F0    30  8C 02               LEAX   *+5,PCR
01F3    20  09                  BRA    *+11
01F5        08                  FCB    8
01F6        20                  FCC    8,  BOARD:
01FE    17  0000                LBSR   .LODS
0201    5F                      CLRB
0202    4F                      CLRA
0203    34  06                  PSHS   D
0205    17  0000                LBSR   .WRS
0208    17  0000                LBSR   .WLN          writeln (listing);
                      *
020B    30  A8 D0               LEAX   -48,Y
020E    34  10                  PSHS   X
0210    17  0000                LBSR   .WLN
                      *                              col[1] := 0;  {col[x] is current column for row x}
0213    5F                      CLRB
0214    4F                      CLRA
0215    ED  A8 D4               STD    -44,Y         row    := 1;
                      *
0218    C6  01                  LDB    #1
021A    ED  34                  STD    -12,Y         WHILE row > 0 DO  {for all rows do}
                      *
021C                  L8        EQU    *
021C    EC  34                  LDD    -12,Y
021E  102F  0000                LBLE   L9              BEGIN
                      *                                  col[row] := col[row] + 1; {move to next column}
                      *
0222    30  A8 D2               LEAX   -46,Y
0225    58                      ASLB
0226    49                      ROLA
0227    30  8B                  LEAX   D,X
0229    34  10                  PSHS   X
0228    30  A8 D2               LEAX   -46,Y
022E    EC  34                  LDD    -12,Y
0230    58                      ASLB
0231    49                      ROLA
0232    EC  8B                  LDD    D,X
0234    C3  0001                ADDD   #1
0237    ED  F1                  STD    [0,S++]           WHILE (col[row] <= n) AND (NOT place(row)) DO
                      *
0239                  L10       EQU    *
0239    30  A8 D2               LEAX   -46,Y
023C    EC  34                  LDD    -12,Y
023E    58                      ASLB
023F    49                      ROLA
0240    EC  8B                  LDD    D,X
0242  10A3  32                  CMPD   -14,Y
0245    2F  03                  BLE    *+5
0247    5F                      CLRB
0248    20  02                  BRA    *+4
024A    C6  01                  LDB    #1
024C    34  04                  PSHS   B
024E    32  7F                  LEAS   -1,S
0250    EC  34                  LDD    -12,Y
0252    34  06                  PSHS   D
0254    17  FDA9                LBSR   *-596
0257    35  04                  PULS   B
0259    C8  01                  EORB   #1
0258    E4  E0                  ANDB   0,S+
0250  1027  0000                LBEQ   L11                  col[row] := col[row] + 1;  {try next column}
                      *
0261    30  A8 D2               LEAX   -46,Y
0264    EC  34                  LDD    -12,Y
0266    58                      ASLB
0267    49                      ROLA
```

```
0268   30   88                LEAX   D,X
026A   34   10                PSHS   X
C26C   30   A8 D2             LEAX   -46,Y
C26F   EC   34                LDD    -12,Y
0271   58                     ASLB
0272   49                     ROLA
0273   EC   88                LDD    D,X
C275   C3   0001              ADDD   #1
0278   ED   F1                STD    [0,S++]
027A   20   8D                BRA    *-65
027C                   L11    EQU    *              IF col[row] <= n  {a position is found}
                       *
                       *                              THEN

027C   30   A8 D2             LEAX   -46,Y
C27F   EC   34                LDD    -12,Y
0281   58                     ASLB
0282   49                     ROLA
0283   EC   88                LDD    D,X
0285 10A3   32                CMPD   -14,Y
0288 102E   0000              LBGT   L12
                       *                                  IF row = n    {Is a solution completed?}
                       *                                    THEN        {yes - print it}

023C   EC   34                LDD    -12,Y
C28E 10A3   32                CMPD   -14,Y
0291 1026   0000              LBNE   L13
                       *                                      BEGIN
                       *                                        FOR i := 1 TO n DO

0295   CC   0001              LDD    #1
C298   ED   36                STD    -10,Y
029A   EC   32                LDD    -14,Y
029C   ED   A8 CE             STD    -50,Y
C29F 10A3   36                CMPD   -10,Y
02A2 102D   0000              LBLT   L15
02A6                   L14    EQU    *
                       *                                          write (listing,col[i]:4);

02A6   30   A8 D0             LEAX   -48,Y
02A9   34   10                PSHS   X
02AB   30   A8 D2             LEAX   -46,Y
02AE   EC   36                LDD    -10,Y
02B0   58                     ASLB
0231   49                     ROLA
0232   EC   88                LDD    D,X
0234   34   06                PSHS   D
0236   CC   0004              LDD    #4
02B9   34   06                PSHS   D
02BB   17   0000              LBSR   .WRI
C23E   32   62                LEAS   2,S
02C0   EC   36                LDD    -10,Y
02C2 10A3   A8 CE             CMPD   -50,Y
02C6 1027   0000              LBEQ   L15
02CA   C3   0001              ADDD   #1
02CD   ED   36                STD    -10,Y
02CF   20   D5                BRA    *-41
02D1                   L15    EQU    *
                       *                                          writeln (listing)

02D1   30   A8 D0             LEAX   -48,Y
02D4   34   10                PSHS   X
02D6   17   0000              LBSR   .WLN
                       *                                      END {THEN}
                       *                                  ELSE    {no - go to next row}

02D9   16   0000              LBRA   L16
02DC                   L13    EQU    *
                       *                                      BEGIN
                       *                                        row := row + 1;

02DC   EC   34                LDD    -12,Y
02DE   C3   0001              ADDD   #1
```

```
02E1  ED 34          STD    -12,Y                       col[row] := 0
                *                                          END {ELSE}
                *
02E3  30 A8 D2       LEAX   -46,Y
02E6  58             ASLB
02E7  49             ROLA
02E8  30 8B          LEAX   D,X
02EA  5F             CLRB
02EB  4F             CLRA
02EC  ED 84          STD    0,X                  ELSE  row := row - 1  {backtrack}
                *
02EE            L16  EQU    *
02EE  16 0000        LBRA   L17
02F1            L12  EQU    *                     END; {WHILE}
                *
02F1  EC 34          LDD    -12,Y
02F3  83 0001        SUBD   #1
02F6  ED 34          STD    -12,Y
02F8            L17  EQU    *
02F8  16 FF21        LBRA   *-220
02FB            L9   EQU    *                   writeln (listing);
                *
02FB  30 A8 D0       LEAX   -48,Y
02FE  34 10          PSHS   X
0300  17 0000        LBSR   .WLN                writeln ('SEARCH COMPLETE')
                *
0303  30 38          LEAX   -8,Y
0305  34 10          PSHS   X
0307  30 8C 02       LEAX   *+5,PCR
030A  20 10          BRA    *+18
030C     0F          FCB    15
030D     53          FCC    15,SEARCH COMPLETE
031C  17 0000        LBSR   .LODS
031F  5F             CLRB
0320  4F             CLRA
0321  34 06          PSHS   D
0323  17 0000        LBSR   .WRS
0326  17 0000        LBSR   .WLN
                 *                     END {ELSE}
                 *            END. {queens}
0329             L7   EQU   *
0329  30 A8 D0        LEAX  -48,Y
032C  34 10           PSHS  X
032E  17 0000         LBSR  .CLO
0331  30 38           LEAX  -8,Y
0333  34 10           PSHS  X
0335  17 0000         LBSR  .CLO
0338  30 3C           LEAX  -4,Y
033A  34 10           PSHS  X
033C  17 0000         LBSR  .CLO
033F              L5  EQU   50
033F  CC 0000         LDD   #0
0342  17 0000         LBSR  .EXIT
0345                  END
```

10.3 LINKER LISTING

The load map describes the memory allocations and the library routines in the resulting executable load module. The Pascal module will contain the addresses and names of each of the level one procedures and functions in the program. The main program entry point is designated .ENTRY.

```
Mctorola 3-bit Cross Linkage Editor Version  1.01  02/17/83  15:17:45  Page 1


Command Line:

LINK QUEENS,QUEENS,QUEENS;AIMXL=PAS09LIB.RX


Cotions in Effect:  A,-B,-H,I,L,M,Q,-U,X


User Commands:

LOCATE PSCT,DSCT $2000
DEF .DHIGH $DFFF
DEF .SIZE 1
END


Load Map:

Module          S  T  Start     End       Externally Defined Symbols

QUEENS          P     00002000  00002344  PLACE     00002000  .ENTRY     00002098
INIT            P     00002345  000023B0  .INIT     00002348  .INITS     00002345
CLO             P     000023B1  00002429  .CLO      000023B1
ENT             P     0000242A  00002451  .ENT      0000242A
IFD             P     00002452  0000264C  .IFD      00002452
LODS            P     0000264D  0000266E  .LODS     0000264D
RDI             P     0000266F  000026F2  .RDI      0000266F
RET             P     000026F3  0000270E  .RET      000026F3
RLN             P     0000270F  00002757  .RLN      0000270F
RNXT            P     00002758  0000279D  .RNXT     00002758  .RNXT2     00002776
RST             P     0000279E  00002803  .RST      0000279E
RWT             P     00002804  0000287E  .RWT      00002804
VLDT            P     0000287F  000028FA  .VLDT     0000287F
WLN             P     000028FB  00002926  .WLN      000028FB
WRI             P     00002927  000029B4  .WRI      00002927
WRS             P     000029B5  000029E9  .WRS      000029B5
WVLD            P     000029EA  00002A0E  .WVLD     000029EA
OVRFL           P     00002A0F  00002A22  .OVRFL    00002A0F
EXIT            P     00002A23  00002A80  .EXIT     00002A25  .EXITI     00002A23
CVHEX           P     00002A81  00002A97  .CVHEX    00002A81
.ENDD           D  C  00002A98  00002A98


Table of Externally Defined Symbols:

Name          Address   Module         Displ     Sect Library        Input

.CLO          000023B1  CLO            00000000   P   PAS09LIB.RX
.CVHEX        00002A81  CVHEX          00000000   P   PAS09LIB.RX
.DHIGH        0000DFFF  USER DEFINED
.ENT          0000242A  ENT            00000000   P   PAS09LIB.RX
.ENTRY        00002098  QUEENS         00000098   P                  QUEENS   .RX
.EXIT         00002A25  EXIT           00000002   P   PAS09LIB.RX
```

```
Motorola 8-bit Cross Linkage Editor Version  1.01  02/17/83  15:17:45  Page 2

.EXITI      00002A23  EXIT          00000000   P   PAS09LIB.RX
.IFD        00002452  IFD           00000000   P   PAS09LIB.RX
.INIT       0000234B  INIT          00000006   P   PAS09LIB.RX
.INITS      00002345  INIT          00000000   P   PAS09LIB.RX
.LODS       0000264D  LODS          00000000   P   PAS09LIB.RX
.OVRFL      00002A0F  OVRFL         00000000   P   PAS09LIB.RX
.RDI        0000266F  RDI           00000000   P   PAS09LIB.RX
.RET        000026F3  RET           00000000   P   PAS09LIB.RX
.RLN        0000270F  RLN           00000000   P   PAS09LIB.RX
.RNXT       00002758  RNXT          00000000   P   PAS09LIB.RX
.RNXT2      00002776  RNXT          0000001E   P   PAS09LIB.RX
.RST        0000279E  RST           00000000   P   PAS09LIB.RX
.RWT        00002804  RWT           00000000   P   PAS09LIB.RX
.SIZE       00000001  USER DEFINED
.VLDT       0000287F  VLDT          00000000   P   PAS09LIB.RX
.WLN        000023F8  WLN           00000000   P   PAS09LIB.RX
.WRI        00002927  WRI           00000000   P   PAS09LIB.RX
.WRS        000029B5  WRS           00000000   P   PAS09LIB.RX
.WVLD       000029EA  WVLD          00000000   P   PAS09LIB.RX
PLACE       00002000  QUEENS        00000000   P                    QUEENS   .RX

Unresolved References: None

Multiply Defined Symbols: None

    No Errors
    No Warnings

S-record module has been created.
```

10.4 EXECUTION

The execution of the program is shown below. The file variable 'listing' is associated with the external file RESULT.SA:1 by specifying it on the command line. The RESULT file is then listed to show the solutions for a standard chess board.

```
=EXBIN QUEENS
=QUEENS RESULT:1
SPECIFY SIZE OF BOARD:
8
SEARCH COMPLETE
=


 PAGE 001   RESULT   .SA:1


 PLACEMENTS OF QUEENS FOR A  8 X 8  BOARD:

     1    5    8    6    3    7    2    4
     1    6    8    3    7    4    2    5
     1    7    4    6    8    2    5    3
     1    7    5    8    2    4    6    3
     2    4    6    8    3    1    7    5
     2    5    7    1    3    8    6    4
     2    5    7    4    1    8    6    3
     2    6    1    7    4    8    3    5
     2    6    8    3    1    4    7    5
     2    7    3    6    8    5    1    4
     2    7    5    8    1    4    6    3
     2    8    6    1    3    5    7    4
     3    1    7    5    8    2    4    6
     3    5    2    8    1    7    4    6
     3    5    2    8    6    4    7    1
     3    5    7    1    4    2    8    6
     3    5    8    4    1    7    2    6
     3    6    2    5    8    1    7    4
     3    6    2    7    1    4    8    5
     3    6    2    7    5    1    8    4
     3    6    4    1    8    5    7    2
     3    6    4    2    8    5    7    1
     3    6    8    1    4    7    5    2
     3    6    8    1    5    7    2    4
     3    6    8    2    4    1    7    5
     3    7    2    8    5    1    4    6
     3    7    2    8    6    4    1    5
     3    8    4    7    1    6    2    5
     4    1    5    8    2    7    3    6
     4    1    5    8    6    3    7    2
     4    2    5    8    6    1    3    7
     4    2    7    3    6    8    1    5
     4    2    7    3    6    8    5    1
     4    2    7    5    1    8    6    3
     4    2    8    5    7    1    3    6
     4    2    8    6    1    3    5    7
     4    6    1    5    2    8    3    7
     4    6    8    2    7    1    3    5
     4    6    8    3    1    7    5    2
     4    7    1    8    5    2    6    3
     4    7    3    8    2    5    1    6
     4    7    5    2    6    1    3    8
     4    7    5    3    1    6    8    2
     4    8    1    3    6    2    7    5
     4    8    1    5    7    2    6    3
     4    8    5    3    1    7    2    6
     5    1    4    6    8    2    7    3
     5    1    8    4    2    7    3    6
     5    1    8    6    3    7    2    4
     5    2    4    6    8    3    1    7
     5    2    4    7    3    8    6    1
     5    2    6    1    7    4    8    3
```

```
 PAGE 002   RESULT   .SA:1

     5    2    8    1    4    7    3    6
     5    3    1    6    8    2    4    7
     5    3    1    7    2    8    6    4
     5    3    8    4    7    1    6    2
     5    7    1    3    8    6    4    2
     5    7    1    4    2    8    6    3
     5    7    2    4    8    1    3    6
     5    7    2    6    3    1    4    8
     5    7    2    6    3    1    8    4
     5    7    4    1    3    8    6    2
     5    8    4    1    3    6    2    7
     5    8    4    1    7    2    6    3
     6    1    5    2    8    3    7    4
     6    2    7    1    3    5    8    4
     6    2    7    1    4    8    5    3
     6    3    1    7    5    8    2    4
     6    3    1    8    4    2    7    5
     6    3    1    8    5    2    4    7
     6    3    5    7    1    4    2    8
     6    3    5    8    1    4    2    7
     6    3    7    2    4    8    1    5
     6    3    7    2    8    5    1    4
     6    3    7    4    1    8    2    5
     6    4    1    5    8    2    7    3
     6    4    2    8    5    7    1    3
     6    4    7    1    3    5    2    8
     6    4    7    1    8    2    5    3
     6    8    2    4    1    7    5    3
     7    1    3    8    6    4    2    5
     7    2    4    1    8    5    3    6
     7    2    6    3    1    4    8    5
     7    3    1    6    8    5    2    4
     7    3    8    2    5    1    6    4
     7    4    2    5    8    1    3    6
     7    4    2    8    6    1    3    5
     7    5    3    1    6    8    2    4
     8    2    4    1    7    5    3    6
     8    2    5    3    1    7    4    6
     8    3    1    6    2    5    7    4
     8    4    1    3    6    2    7    5
```

APPENDIX A

INTERNAL REPRESENTATION OF DATA

Integer:

```
15              8 7              0
----------------------------------
|s |             |               |
----------------------------------
```

Size: 2 bytes (default size)

Format: Signed two's-complement

Range: -32,768 to 32,767

for an integer subrange type within the range -128 to 127, inclusive:

```
7              0
----------------
|s |           |
----------------
```

Size: 1 byte

Format: Signed two's-complement

Range: -128 to 127

for an integer subrange type that extends outside the range -32,768 to 32,767, inclusive, but is within the range -2,147,483,648 to 2,147,483,647, inclusive:

```
31            24 23           16 15            8 7             0
-----------------------------------------------------------------
|s |             |               |               |              |
-----------------------------------------------------------------
```

Size: 4 bytes

Format: Signed two's-complement

Range: -2,147,483,648 to 2,147,483,647

NOTE: MAXINT = 2147483647.

Character:

```
 7   6          0
-------------------
|0 |            |
-------------------
```

Size: 1 byte

Format: 7-bit ASCII

Range: 0 to 127

Boolean:

```
 7              0
-------------------
|               |
-------------------
```

Size: 1 byte

Values: 0 = False

1 = True

Set:

```
 63                 56 55               48 47               40 39                32
-----------------------------------------------------------------------------------
|                    |                   |                   |                   |
-----------------------------------------------------------------------------------
 31                 24 23               16 15                8 7                  0
-----------------------------------------------------------------------------------
|                    |                   |                   |                   |
-----------------------------------------------------------------------------------
```

Size: 8 bytes

Range: Up to 64 elements

Enumerated Scalar:

```
 7              0
-------------------
|0|             |
-------------------
```

Size: 1 byte

Representation: 0 to 127

```
15               8 7              0
-----------------------------------
|0 |             |                |
-----------------------------------
```

Size: 2 bytes

Representation: 0 to 32,767

```
31               24 23             16 15              8 7               0
-------------------------------------------------------------------------
|0|                |                 |                |                 |
-------------------------------------------------------------------------
```

Size: 4 bytes

Representation: 0 to 2,147,483,647

String:

```
-------------------------------------          ----------------------
|Cur. Length   |       |       |    .     .     .   |        |        |
-------------------------------------          ----------------------
```

Size: 1 to 255 bytes

Representation: Current-length byte and
0 to 254 ASCII characters

Pointer:

```
15               8  7              0
------------------------------------
|                                  |
------------------------------------
```

Size: 2 bytes

Range: 0 to 65,535

File Pointer:

```
Offset  ---------------------------
        |                         |
  0:    | current component pointer |
        ---------------------------
        |                         |
  2:    | file descriptor pointer |
        ---------------------------
```

File Descriptor:

```
    Offset
hex        decimal
                     _______________________________      ________
 0          0       |  next component pointer       |     Pascal
 2          2       |  component size               |     Parameter
 4          4       |                               |     Block
                    |  file position                |       |
                    |_______________________________|       |
 8          8       |  file status                  |       |
 A         10       |  Record end address           |       v
 C         12       |  error        |  transfer type|     ________
 E         14       |  data buffer pointer          |     I/O
10         16       |  data buffer start            |     Control
12         18       |  data buffer end              |     Block
14         20       |  generic device word          |       |
16         22       |log. unit number|              |       |
                    |                               |       |
                    |  file name         ___________|       |
1E         30       |                |  extension   |       |
20         32       |  extension     |   RIB        |       |
22         34       |  RIB           | file descriptor|     |
24         36       |  reserved       ______________|       |
26         38       |                | directory entry|     |
28         40       |  reserved      |initial file size|    |
2A         42       |initial file size| sector buf start|   |
2C         44       | sector buf start| sector buf end |    |
2E         46       | sector buf end |internal pointer|     v
30         48       |internal pointer |             |     ________
                    |                               |     Backup
                    /                               /     I/O
                    /  backup IOCB                  /     Control
                    |                               |     Block
54         84       |_______________________________|       v
56         86       |                               |     ________
                    |                               |
                    /  sector buffer                /
                    /                               /
                    |                               |
154       340       |_______________________________|
156       342       |                               |
                    |                               |
                    /  record buffer                /
                    /                               /
                    |                               |
                    |_______________________________|
```

APPENDIX B ·

ASCII CHARACTER SET

CHARACTER	COMMENTS	HEX VALUE
NUL	Null or tape feed	00
SOH	Start of Heading	01
STX	Start of Text	02
ETX	End of Text	03
EOT	End of Transmission	04
ENQ	Enquire (who are you, WRU)	05
ACK	Acknowledge	06
BEL	Bell	07
BS	Backspace	08
HT	Horizontal Tab	09
LF	Line Feed	0A
VT	Vertical Tab	0B
FF	Form Feed	0C
CR	Carriage Return	0D
SO	Shift Out (to red ribbon)	0E
SI	Shift In (to black ribbon)	0F
DLE	Data Link Escape	10
DC1	Device Control 1	11
DC2	Device Control 2	12
DC3	Device Control 3	13
DC4	Device Control 4	14
NAK	Negative Acknowledge	15
SYN	Synchronous Idle	16
ETB	End of Transmission Block	17
CAN	Cancel	18
EM	End of Medium	19
SUB	Substitute	1A
ESC	Escape, prefix	1B
FS	File Separator	1C
GS	Group Separator	1D
RS	Record Separator	1E
US	Unit Separator	1F

CHARACTER	COMMENTS	HEX VALUE
SP	Space or Blank	20
!	Exclamation point	21
"	Quotation mark (diaeresis)	22
#	Number sign	23
$	Dollar sign	24
%	Percent sign	25
&	Ampersand	26
'	Apostrophe, acute accent, closing single quote	27
(	Opening parenthesis	28
)	Closing parenthesis	29
*	Asterisk	2A
+	Plus sign	2B
,	Comma (cedilla)	2C
-	Hyphen (minus)	2D
.	Period (decimal point)	2E
/	Slant	2F
0	Digit 0	30
1	Digit 1	31
2	Digit 2	32
3	Digit 3	33
4	Digit 4	34
5	Digit 5	35
6	Digit 6	36
7	Digit 7	37
8	Digit 8	38
9	Digit 9	39
:	Colon	3A
;	Semicolon	3B
<	Less than	3C
=	Equals	3D
>	Greater than	3E
?	Question mark	3F

CHARACTER	COMMENTS	HEX VALUE
@	Commercial at	40
A	Uppercase letter A	41
B	Uppercase letter B	42
C	Uppercase letter C	43
D	Uppercase letter D	44
E	Uppercase letter E	45
F	Uppercase letter F	46
G	Uppercase letter G	47
H	Uppercase letter H	48
I	Uppercase letter I	49
J	Uppercase letter J	4A
K	Uppercase letter K	4B
L	Uppercase letter L	4C
M	Uppercase letter M	4D
N	Uppercase letter N	4E
O	Uppercase letter O	4F
P	Uppercase letter P	50
Q	Uppercase letter Q	51
R	Uppercase letter R	52
S	Uppercase letter S	53
T	Uppercase letter T	54
U	Uppercase letter U	55
V	Uppercase letter V	56
W	Uppercase letter W	57
X	Uppercase letter X	58
Y	Uppercase letter Y	59
Z	Uppercase letter Z	5A
[	Opening bracket	5B
\	Reverse slant	5C
]	Closing bracket	5D
^	Circumflex	5E
_	Underline	5F

CHARACTER	COMMENTS	HEX VALUE
'	Grave accent, Opening single quote	60
a	Lowercase letter a	61
b	Lowercase letter b	62
c	Lowercase letter c	63
d	Lowercase letter d	64
e	Lowercase letter e	65
f	Lowercase letter f	66
g	Lowercase letter g	67
h	Lowercase letter h	68
i	Lowercase letter i	69
j	Lowercase letter j	6A
k	Lowercase letter k	6B
l	Lowercase letter l	6C
m	Lowercase letter m	6D
n	Lowercase letter n	6E
o	Lowercase letter o	6F
p	Lowercase letter p	70
q	Lowercase letter q	71
r	Lowercase letter r	72
s	Lowercase letter s	73
t	Lowercase letter t	74
u	Lowercase letter u	75
v	Lowercase letter v	76
w	Lowercase letter w	77
x	Lowercase letter x	78
y	Lowercase letter y	79
z	Lowercase letter z	7A
{	Opening brace	7B
|	Vertical line	7C
}	Closing brace	7D
~	Tilde	7E
DEL	Delete	7F

APPENDIX C

PASCAL LANGUAGE PROCESSOR ERRORS

1:	error in simple type
2:	identifier expected
3:	'program' expected
4:	')' expected
5:	':' expected
6:	illegal symbol
7:	error in parameter list
8:	'of' expected
9:	'(' expected
10:	error in type
11:	'[' expected
12:	']' expected
13:	'end' expected
14:	';' expected
15:	integer expected
16:	'=' expected
17:	'begin' expected
18:	error in declaration part
19:	error in field-list
20:	',' expected
21:	'*' expected
50:	error in constant
51:	':=' expected
52:	'then' expected
53:	'until' expected
54:	'do' expected
55:	'to'/'downto' expected
56:	'if' expected
57:	'file' expected
58:	error in factor
59:	error in variable

101: identifier declared twice

102: low boundary exceeds high boundary

103: identifier is not of appropriate class

104: identifier not declared

105: sign not allowed

106: number expected

107: incompatible subrange types

108: file not allowed here

109: type must not be real

110: tagfield type must be scalar or subrange

111: incompatible with tagfield type

112: index type must not be real

113: index type must be scalar or subrange

114: base type must not be real

115: base type must be scalar or subrange

116: error in type of standard procedure parameter

117: unsatisfied forward reference

118: forward reference type identifier in variable declaration

119: forward declared; repetition of parameter list not allowed

120: function result type must be scalar, subrange or pointer

121: file value parameter not allowed

122: forward declared function; repetition of result type not allowed

123: missing result type in function declaration

124: fixed-point output format allowed for real only

125: error in type of standard function parameter

126: number of parameters does not agree with declaration

127: illegal parameter substitution

128: result type of parameter function does not agree with declaration

129: type conflict of operands

130: expression is not of set type

131: tests on equality allowed only

132: strict inclusion not allowed

133: file comparison not allowed

134: illegal type of operand(s)

135: type of operand must be Boolean

136: set element type must be scalar or subrange

137: set element types not compatible

138: type of variable is not array

MOTOROLA GMBH GESCHÄFTSBEREICH HALBLEITER

POSTFACH 1229 · MÜNCHNER STRASSE 18 · D-8043 UNTERFÖHRING