64000

HP64000 Logic Development System

C/64000 Compiler Reference Manual

HEWLETT PACKARD

CERTIFICATION

Hewlett-Packard Company certifies that this product met its published specifications at the time of shipment from the factory. Hewlett-Packard further certifies that its calibration measurements are traceable to the United States National Bureau of Standards, to the extent allowed by the Bureau's calibration facility, and to the calibration facilities of other International Standards Organization members.

WARRANTY

This Hewlett-Packard system product is warranted against defects in materials and workmanship for a period of 90 days from date of installation. During the warranty period, HP will, at its option, either repair or replace products which prove to be defective.

Warranty service of this product will be performed at Buyer's facility at no charge within HP service travel areas. Outside HP service travel areas, warranty service will be performed at Buyer's facility only upon HP's prior agreement and Buyer shall pay HP's round trip travel expenses. In all other cases, products must be returned to a service facility designated by HP.

For products returned to HP for warranty service, Buyer shall prepay shipping charges to HP and HP shall pay shipping charges to return the product to Buyer. However, Buyer shall pay all shipping charges, duties, and taxes for products returned to HP from another country.

HP warrants that its software and firmware designated by HP for use with an instrument will execute its programming instructions when properly installed on that instrument. HP does not warrant that the operation of the instrument, or software, or firmware will be uninterrupted or error free.

LIMITATION OF WARRANTY

The foregoing warranty shall not apply to defects resulting from improper or inadequate maintenance by Buyer, Buyer-supplied software or interfacing, unauthorized modification or misuse, operation outside of the environment specifications for the product, or improper site preparation or maintenance.

NO OTHER WARRANTY IS EXPRESSED OR IMPLIED. HP SPECIFICALLY DISCLAIMS THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

EXCLUSIVE REMEDIES

THE REMEDIES PROVIDED HEREIN ARE BUYER'S SOLE AND EXCLUSIVE REMEDIES. HP SHALL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, WHETHER BASED ON CONTRACT, TORT, OR ANY OTHER LEGAL THEORY.

ASSISTANCE

Product maintenance agreements and other customer assistance agreements are available for Hewlett-Packard products.

For any assistance, contact your nearest Hewlett-Packard Sales and Service Office.

CW&A 9/79

FOLD HERE

HEWLETT PACKARD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD
FIRST CLASS PERMIT NO. 1303 COLORADO SPRINGS, COLORADO

POSTAGE WILL BE PAID BY ADDRESSEE

HEWLETT-PACKARD
Logic Product Support Dept.
Attn: Technical Publications Manager
Centennial Annex - D2
P.O. Box 617
Colorado Springs, Colorado 80901-0617

FOLD HERE

Your cooperation in completing and returning this form will be greatly appreciated. Thank you.

READER COMMENT SHEET

Operating Manual
C/64000 Compiler Reference Manual
64800-90907, February 1983

Your comments are important to us. Please answer this questionaire and return it to us. Circle the number that best describes your answer in questions 1 through 7. Thank you.

1. The information in this book is complete:

Doesn't cover enough (what more do you need?)	1 2 3 4 5	Covers everything

2. The information in this book is accurate:

Too many errors	1 2 3 4 5	Exactly right

3. The information in this book is easy to find:

I can't find things I need	1 2 3 4 5	I can find info quickly

4. The Index and Table of Contents are useful:

Helpful	1 2 3 4 5	Missing or inadequate

5. What about the "how-to" procedures and examples:

No help	1 2 3 4 5	Very helpful
Too many now	1 2 3 4 5	I'd like more

6. What about the writing style:

Confusing	1 2 3 4 5	Clear

7. What about organization of the book:

Poor order	1 2 3 4 5	Good order

8. What about the size of the book:

too big/small	1 2 3 4 5	Right size

Comments: _____

_____

_____

_____

Particular pages with errors?

_____

Name (optional): _____
Job title: _____
Company: _____
Address: _____

Note: If mailed outside U.S.A., place card in envelope. Use address shown on other side of this card.

C Compiler
Reference Manual

Printing History

Each new edition of this manual incorporates all material updated since the previous edition. Manual change sheets are issued between editions, allowing you to correct or insert information in the current edition.

The part number on the back cover changes only when each new edition is published. Minor corrections or additions may be made as the manual is reprinted between editions.

First Printing . . . . . June 1982 (Part Number 64800-90907)
Reprinted . . . . . February 1983

Table of Contents

Chapter 1: General Information

Chapter 2: Compiler Description

Table of Contents (Cont'd)

Chapter 1

General Information

Introduction

This manual provides a description of the C/64000 compiler and its operation on the HP Model 64000 Logic Development System. A description of the compiler options and their use is also included. Microprocessor dependent features of the compiler are documented in processor-dependent supplements.

NOTE

Refer to Chapter 5 in the System Software Manual for SOFTWARE UPDATING PROCEDURES when updating the system from a tape cartridge or flexible disc.

C/64000 Compiler

The C/64000 compiler is an application program that translates C/64000 source programs into relocatable object files, and optionally generates a listing file.

C/64000 is an implementation of the C programming language "standard", defined by Kernighan and Ritchie in "The C Programming Language" published by Prentice-Hall, 1978. The language has been enhanced to improve its utility as a tool for microprocessor system programming.

This manual assumes the user has knowledge of the C language as defined by Kernighan and Richie and other reference books on standard C. The emphasis in this manual is to document the specific C/64000 implementation restrictions and extensions to Kernighan and Richie.

The C/64000 compiler uses a four-pass compilation process, plus an optional preprocessor, to translate source programs directly into relocatable code for the target microprocessor. Relocatable files for a particular microprocessor may be linked together to produce an absolute program file. Then, by using the emulator, the absolute file can be loaded into emulation memory and executed in the proper microprocessor environment.

Implementation Restrictions

Restrictions

The following items are unspecified by the standard and may impose implementation restrictions.

- The #line preprocessor instruction is not implemented.
- The standard library functions (printf, getchar, etc...) are not included.
- Register variables are treated as auto and are not specially optimized.
- #include <FILE> is not available since there are no "standard places"; however, #include "FILE" is available.
- Strings may not extend to multiple lines and, therefore, are limited to 238 characters.

Dependencies

The following items are unspecified by the standard and may cause implementation dependencies.

- Pointers and integers (type int) are not necessarily the same size. Care must be taken when mixing them.
- Fields within records are assigned left to right.
- All shifts are logical, not arithmetic.
- The order of parameter passing may be from left to right or right to left depending on the specific code generator used.
- The preprocessor instructions, '#include FILE' and '#include "FILE"', are treated identically. FILE may optionally contain a userid and disc #. If they are not present, the defaults are used.

C/64000 Extensions

C/64000 contains enhancements that provide more versatility for microprocessor programming.

- Program code and constants may be compiled to a separate relocatable area from data and variables allowing the design of ROM and RAM memory systems.

- Variables may be assigned to absolute memory locations permitting easy access to memory I/O addresses.

- The first fifteen characters of an identifier are significant, although a truncation to eight characters may be forced for compatibility with other C systems.

- Arthmetic may be done with short variables without conversion to type int, or with float variables without converting to type double.

- The keywords may be defined to be upper case instead of the standard lower case.

- Standard 64000 constants ending in the letters B, D, H, O, and Q may be used to indicate binary, decimal, hexadecimal, and octal constants in addition to the standard C forms by use of the EXTENSIONS option.

- Structures may be assigned, compared for equality and inequality, passed as parameters, or returned from functions.

- The enumeration type defined in the November 15, 1978 supplement to "The C Programming Language" reference manual may be used. This type is describe in appendix B.

- A shift by a negative value is equivalent to a shift in the opposite direction by a positive value. Thus, a>>b and a<<-b are equivalent.

- Large constants (i.e. strings, real numbers, etc...) are optimized. Those constants which match in full or in part will be in the same locations in memory. (Exception: variables which are initialized to constants do not go in the constant section. Variables which are initialized to point to constants do point into the constant section.)

. With the $USER_DEFINED$ option, the user may selectively redefine the meaning of the arithmetic operators +, -, *, /, %, ==, !=, <, >, <=, and >=. For example, (*) may be redefined to do matrix multiplication when its operands are two-dimensional matrices.

. The functions **ABS, SQRT, SIN, COS, ARCTAN, LN** and **EXP** are available in real number libraries.

C/64000 Environment

The C/64000 compiler will run on any HP 64000 system that includes expanded host memory capability. The compiled code may be run using the proper emulation subsystem for the target microprocessor or on any independent system which uses the same target microprocessor. The following paragraphs list the C/64000 character set and the representation of intrinsic data types.

CHARACTER SET

Alphabetic characters	- All upper and lower case characters (A through Z and a through z).
Numeric characters	- Digits 0 through 9 for decimal numbers, including A through F (and a thru f) hexadecimal numbers.
Special characters	- Blank, dollar sign, apostrophe, left and right parentheses, comma, plus, minus, equals, less than, greater than, decimal point, slash, colon, semi-colon, left and right brackets, left and right braces, caret, asterisk, underscore (_), ampersand, exclamation point, quotation marks, pound sign, percent sign, tilde, backslash, and question mark.

DATA TYPES

Intrinsic Data Types

short	- An 8-bit signed integer in the range -128 to +127.
unsigned short	- An 8-bit unsigned integer in the range 0 to 255.
int	- A 16-bit signed integer in the range -32768 to +32767.
unsigned [int]	- A 16-bit unsigned integer in the range 0 to 65535.
long	- For processors which support 32-bit arithmetic, a 32-bit signed integer in the range -2,147,483,648 to +2,147,483,647. For processors which do not support 32-bit arithmetic, equivalent to int.
unsigned long	- For processors which support 32-bit arithmetic, a 32-bit unsigned integer in the range 0 to 4,294,967,295. For processors which do not support 32-bit arithmetic, equivalent to unsigned int.
char	- An 8-bit value defined by the ASCII character set. Equivalent to an unsigned short.
float	- A 32-bit real number in the single IEEE format.
double	- A 64-bit real number in the double IEEE format.

DATA TYPES (Cont'd)

Derived Data Types

The derived data types have a representation that is dependent on the specified microprocessor. They may contain holes (unused bytes) due to memory alignment requirements. For statically allocated variables, these holes will be filled with zeroes, if INIT_ZEROES is **ON**. A list of the derived data types is as follows:

pointers
arrays
structures (struct)
unions (union)
enumerated (enum)
user (typedef)

Chapter 2

Compiler Description

General

C/64000 uses a four-pass compilation process plus an optional preprocessor. The preprocessor, which may be disabled by the NOPREPROCESS directive, handles macros, include files, and conditional compilation as defined in "The C Programming Language" by Kernighan and Ritchie.

The four passes are referred to as pass 1, pass 1a, pass 2, and pass 3. Pass 1, which is machine independent, reads the C source, checks for lexical, syntax, and semantic errors, and produces an intermediate language file on disc. Pass 1a, adds additional information (information that was not available when the file was originally produced) to the intermediate language file. This pass is also machine independent. Pass 2, reads the intermediate language file and generates code for the chosen microprocessor by producing a tokenized assembler file on disc. Pass 3, reads the tokenized assembler file and generates a relocatable object file if there were no errors in the first three passes. It also generates a list file if requested.

The optional list file may contain source lines only or source lines mixed with the generated assembly language code if requested.

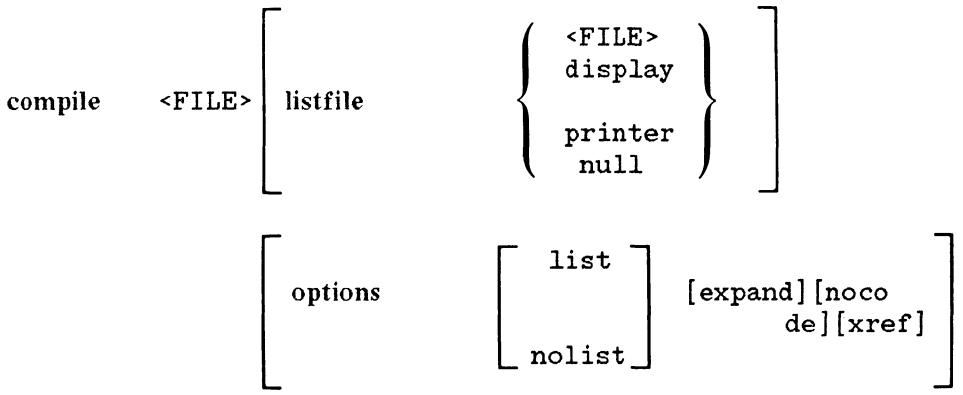
Only the preprocessor and pass 1 actually pass through the source file. Pass 3 will also use the source file if a listing file is requested.

Command Syntax

The following pages provide the formal syntax definition for the compile command.

compile

SYNTAX



Default Values

listfile — The default is to the predefined listfile.
If there is no predefined list file, a null list file is the default.

options — If no entry is made for any of the options, the default values will be:

list - LIST directives in the source
nolist — file will be obeyed.

expand - LIST_CODE and FULL_LIST directives in the source source file will be obeyed.

nocode - EMIT_CODE directives in the source file will be obeyed.

xref - XREF directives in the source file will be obeyed.

FUNCTION

The compile command tells the compiler to translate a C source program (file) into relocatable object code for a microprocessor.

compile (Cont'd)

Command Parameters:

<FILE> A variable representing the source file name, userid, and disc number. The syntax for <FILE> is:

<FILE> => <FILE NAME> [:<USERID>][:<DISC#>]

where:

<FILE NAME> - Up to nine alphanumeric characters, beginning with an upper case alphabetic character.

<USERID> - Up to six alphanumeric characters, beginning with an upper case alphabetic character.

<DISC#> - Represents the logical unit number of the system disc on which the source file is located. Allowable entries are decimal numbers representing the desired disc number.

The file type must be a source file; no other file type can be specified with the compile command. The first line is the "C" indicator and the second line must be the name of the target processor, enclosed in quotation marks (e.g. - "8086").

listfile A key word which specifies a listing file for compiler output. When listfile is specified, one of the following must be specified also:

<FILE>
display
printer
null

compile (Cont'd)

options A key word which allows specification of options for the compile process. When "options" is specified, one or more of the following may be specified:

```
list
nolist
expand
nocode
xref
```

where:

list or nolist - allows specification of the source program list with error messages or no source listing except for error messages. All LIST directives in the source file are ignored.

expand - specifies a list of all source lines with an expansion of the assembly language. Also shows include files and expanded macros if used. All LIST_CODE and FULL_LIST directives in the source file are ignored.

nocode - specifies the suppression of object code generation. Only the source code will be listed in pass 2.

xref - specifies a symbol cross-reference listing for the source file. All XREF directives in the source file are ignored.

Compiler Directives

C/64000 allows compiler directives (options) to be treated as if they were comments. C interprets the following construct as a compiler directive:

$<compiler_directive>$

where <compiler_directive> is any of the directives listed in the following paragraphs. Compiler directives may be inserted between any two tokens (identifiers, numbers, strings, and special symbols). They are used to inform the compiler about changing needs within a program. A compiler directive is a separator (as is a space or comment) in the C program. Compiler directives must begin with a dollar sign ($). The end of a directive is also indicated with a dollar sign. The end of the input line also indicates the end of the compiler directive. Therefore, a directive must exist entirely on one line. Directives which are not syntactically correct are ignored and the directive line is skipped until the dollar sign or the end of the line is detected. The meaning of the compiler directives and their default values are given below. The status of a directive (ON or OFF) is indicated after the directive in the source file. The values: ON, TRUE, and plus sign (+) are equivalent, as are the values: OFF, FALSE, and minus sign (-). If a directive is stated without indicating ON or OFF, the compiler will recognize the directive is ON. Other microprocessor-dependent compiler directives are described in the microprocessor-dependent supplement.

Normally, the preprocessor will ignore compiler directives and treat them like text. To apply a directive to the preprocessor it must be preceded by a pound (#) sign in column 1, as follows:

#$<compiler_directive>$

Directives, preceded by the pound sign in column 1, will affect both the preprocessor and the subsequent passes of the compiler. Although any directive may appear in the above format, only certain directives affect the preprocessor. The specific directives that affect the preprocessor are so indicated in the following descriptions of the directives.

List of Directives

All directives, prior to use in a source program, will assume their initial value when the compiler is called.

AMNESIA [ON] [OFF]

Initialized Value: **OFF**

Description:

ON causes the compiler to forget the contents of registers after the registers are used in an operation. This directive may be used to ensure that variables representing memory mapped I/O ports are reloaded everytime they are needed.

ASM_FILE

Initialized Value: **OFF**

Description:

The source file is created into a file whose name consists of the letters "ASM" followed by the microprocessor designator (e.g. ASM8086, ASMZ8001). This assembler source will be accepted by the assembler as a source file for the selected microprocessor. The assembler source file will also contain intermixed C language source lines as assembler comments.

ASMB_SYM [ON] [OFF]

Initialized Value: **ON**

Description:

ON causes the compiler to generate an asmb_sym file for use during emulation. **OFF** suppresses the generation of the file.

DEBUG [ON] [OFF]

Initialized Value: **OFF**

Description:

ON causes all arithmetic operations to be checked for overflow, underflow, or divide by zero operation. (See specific C/64000 microprocessor-dependent supplement for run-time error descriptions.)

EMIT_CODE [ON][OFF]

Initialized Value: **ON**

Description:

ON specifies that executable code is to be emitted to the relocatable code file.

END_ORG

Description:

Used to change variable address assignment from absolute to relocatable mode.

ENTRY [ON][OFF]

Initialized Value: **ON**

Description:

ON causes an external reference to the identifier "entry" to be generated when the function "main" is declared. The run time libraries will contain the routine "entry" which contains a transfer address and initialization code. After initialization, the "entry" routine calls the "main" function. This routine is described in greater detail in the microprocessor-dependent supplement manuals. **OFF** will disable the generation of an external reference to the "entry" routine.

If the $UPPER_KEYS$ directive is **ON**, the procedure "MAIN" will declare "ENTRY" external.

EXTENSIONS [ON] [OFF]

Initialized Value: **OFF**

NOTE

When properly formatted, this directive also affects the preprocessor.

Description:

ON allows the programmer to use certain extensions to the C language. **OFF** disallows the use of these language extensions. Currently the only extensions available are those that allow the use of numbers ending in B (binary), D (decimal), H (hexadecimal), and O or Q (octal) in addition to standard C constants. The ending characters must be upper-case letters. Hexadecimal numbers declared in this form must use upper case A through F and, if a hexadecimal number starts with a letter, it must be preceded by a zero (0). A number ending in B or D is treated as binary or decimal unless the number is preceded by 0X. In these cases, the number is treated as a hexadecimal number. Numbers in this form may be followed by the letter "L", indicating long.

FIXED_PARAMETERS [ON] [OFF]

Initialized Value: **OFF**

Description:

ON indicates that all functions declared subsequently will always pass the same number and types of parameters. **OFF** indicates that the number and types may vary between calls.

The value of this directive must be the same at all declarations of the function (i.e., the definition, external declarations, and implicit declarations) since the method of parameter passing may vary. Functions declared with this option **ON** may generate more efficient code. They are guaranteed to be compatible with functions and procedures declared in Pascal/64000. Routines declared with FIXED_PARAMETERS ON may not be passed as parameters or have their address taken, i.e., they must be followed by a "(".

FULL_LIST [ON] [OFF]

Initialized Value: **OFF**

Description:

On causes "include" files to be listed and macros to be expanded in the listfile. Lines with errors will be shown whether this directive is **ON** or **OFF**.

INIT_ZEROES [ON] [OFF]

Initialized Value: **ON**

Description:

ON causes all static and external variables not explicitly initialized to be initialized to zero. **OFF** causes them to remain uninitialized. Turning this option **OFF** will speed up the compiler and produce shorter relocatable and absolute files. It may also be useful when working with a ROM-based system where data cannot be initialized.

LINE_NUMBERS [ON] [OFF]

Initialized Value: **ON**

Description:

ON causes the compiler to generate symbols for the C/64000 source line numbers. These symbols are found in the asmb_sym file after the compilation. They may be used during emulation to trace the execution of a C/64000 program by source line number. The symbols are constructed by placing a pound sign (#) in front of the line number. Line number symbols are created only for lines that cause executable code to be generated (i.e., line number symbols will not be created for lines in the external declaration sections of the program).

LIST [ON] [OFF]

Initialized Value: **ON**

Description:

ON causes the source file to be copied to the listfile. **OFF** suppresses the listing except for lines that contain errors.

LIST_CODE [ON] [OFF]

Initialized Value: **OFF**

Description:

ON specifies that the program listfile will contain the symbolic form (assembly language) of the code produced, intermixed with the source lines.

LIST_OBJ [ON] [OFF]

Initialized Value: **OFF**

Description:

ON causes the listing to contain the relocatable object code generated by the third pass of the compiler. The listing will also contain the relocatable object code generated by the third pass when the compile time option "expand" of the compiler directive LIST_CODE is specified.

LONG_NAMES [ON] [OFF]

Initialized Value: **ON**

NOTE

When properly formatted, this directive also affects the preprocessor.

Description:

ON causes identifiers to be truncated at fifteen characters. **OFF** causes identifiers to be truncated at eight characters as defined in standard C. If the option is **OFF** care must be taken (when emulating) to use only the first eight characters since fifteen characters are significant in emulation.

OPTIMIZE [ON] [OFF]

Initialized Value: **OFF**

Description:

ON may cause certain run-time checks to be ignored, such as pre-checking the range values of a switch statment. This mode is typically susceptible to bad out-of-range data at run time. This directive should only be used for well-structured programs that have been thoroughly debugged. Refer to the specific microprocessor-dependent supplement for additional information.

ORG number

Description:

All variables declared auto, register, or without a type until END_ORG is encountered will be allocated sequential absolute addresses starting from "number". "number" may be represented with a hexadecimal constant. Parameters, functions, and variables declared external are not affected by this option. ORGed variables may not be initialized.

PAGE

Initialized Value: **null**

Description:

This option causes a form feed to be output to the listfile.

RECURSIVE [ON] [OFF]

Initialized Value: **ON**

Description:

ON causes all compiled procedures to allow recursive or reentrant calling sequences until a subsequent RECURSIVE OFF is encounterd. **OFF** causes procedures to be compiled in a static mode which does not allow for recursive or reentrant calling sequences.

SEPARATE [ON] [OFF]

Initialized Value: **OFF**

Description:

ON enables the separation of program and constants and data such that program code and constants are put in the PROG relocatable area and data in the DATA relocatable area. **OFF** puts program code, constants, and data into the PROG relocatable area. This directive should be set **ON** before the first line of code if external data is to be affected. Refer to the specific microprocessor-dependent supplement for additional information.

SHORT_ARITH [ON] [OFF]

Initialized Value: **OFF**

Description:

Among **short** and **char** variables, **ON** causes arithmetic to be accomplished without conversion to type **int** and, among **float** variables, without conversions to type **double**. **OFF** will cause conversions unless the result is guaranteed the same without the conversion. This directive has no affect on parameter passing which always uses type **int** or **double**.

STANDARD [ON] [OFF]

Initialized Value: **OFF**

NOTE

When properly formatted, this directive also affects the preprocessor.

Description:

ON causes a warning to be issued for any feature of C/64000 which is not a feature of "standard" C as defined by Kernighan & Ritchie.

TITLE "string"

Initialized Value: **null**

Description:

The first 50 characters of the string are moved into the header line printed at the top of each subsequent page of the listfile.

UPPER_KEYS [ON] [OFF]

Initialized Value: **OFF**

NOTE

When properly formatted, this directive also affects the preprocessor.

Description:

ON causes the compiler to recognize upper case keywords instead of lower case.

USER_DEFINED

Initialized Value: **null**

Description:

C/64000 allows the user to redefine the semantics of certain operators in the language. User defined operators are created by using the option $USER_DEFINED$ just prior to a typedef statement.

For user defined operators, the compiler will not generate in-line code to perform the operations, but the compiler will generate calls to user provided run-time routines. The run-time routine name will be a composite of the user's type name and the operation being performed, TYPENAME_OPERATION. The first eleven characters of the user's type name are concatenated with an underscore and three characters identifying the operation. Only one type may be declared in a USER_DEFINED typedef statement.

The following is a list of the operators that can be user defined and the run-time routine names that the compiler will create when the operations are used on a user type.

Operation	Symbol	Run-time Routine
1) Add	+	<typename>_ADD
2) Negate	-	<typename>_NEG
3) Subtract	-	<typename>_SUB
4) Multiply	*	<typename>_MUL
5) Divide	/	<typename>_DIV
6) Modulus	%	<typename>_MOD
7) Equal Comparison	==	<typename>_EQU
8) Not Equal Comparison	!=	<typename>_NEQ
9) Less Than or Equal to Comparison	<=	<typename>_LEQ
10) Greater Than or Equal to Comparison	>=	<typename>_GEQ
11) Less Than Comparison	<	<typename>_LES
12) Greater Than Comparison	>	<typename>_GTR

Refer to the specified microprocessor-dependent supplement for additional information on this directive.

WARN [ON][OFF]

Intialized Value: **ON**

NOTE

When properly formatted, this directive also affects the preprocessor.

Description:

Specifies that warning messages be written to the listing file. When this directive is **OFF**, only error messages will be displayed and listed.

WIDTH number

Initialized Value: **240**

NOTE

When properly formatted, this directive also affects the preprocessor.

Description:

The number specifies the number of significant characters (width) in the source file to be compiled. Additional characters are ignored and if WARN is **ON**, a warning message will be generated.

Additional Information

16-bit Integers

For microprocessors that do not allow 32-bit integers, the number 32768 (0X8000) can only be interpreted as a negative value since its sign bit is set. The expression -32768 is a legal value, but it is scanned as being the negation of the positive value 32768. As a result, the compiler first detects it as the "out of range" positive value and gives the user an appropriate warning message:

"506: Warning: +32768 is treated as -32768 by the compiler"

NOTE

The warning is not printed if the microprocessor allows 32-bit integers.

As long as the user really wants the value -32768, he may ignore this warning message. The user will be able to suppress this message entirely by using 0X8000 to express the value -32768.

Use of ORG Option

The use of the compiler directive ORG to assign variables to absolute memory locations does not allocate any absolute memory space. The reference to these variables are explicit absolute addresses in the relocatable file. The linker will not detect or report a memory overlap if the user's absolute addresses interfere with other defined memory areas.

Real Number Function

The functions **ABS, SQRT, SIN, COS, ARCTAN, LN,** and **EXP** are available in the runtime libraries. They have one double parameter and return a double. They must be declared external to access them.

Example:

```
double  x,y;
extern double SIN ();
        .
        .
        .
x=SIN(y);
```

For information on linking these functions refer to the appropriate microprocessor dependent supplement.

Chapter 3

How to Compile a Program

General

The usual process of software generation with the compiler is as follows:

a. Create source program files with editor.

b. Compile source program files.

c. Link relocatable files.

d. Emulate absolute files.

e. Debug.

The following sections of this manual will provide insight into the structure of the source file, compiling the source file, and linking relocatable files. Refer to the appropriate microprocessor-dependent supplement for information on emulating and debugging.

The Source File

The C/64000 compiler takes as input a program source file created with the editor. The basic form of a source file is:

```
"C"
"Z8001"

/*C PROGRAM*/
     .
     .
     .
     .
     .
```

The first line of the source file must be the special compiler directive indicating that the C compiler is to be use.

The second line of the source file must be the special compiler directive which indicates the processor for which the file will be compiled. In the example form given above, the Z8001 microprocessor is specified.

An alternative form of the directive is:

```
"C" NOPREPROCESS
"8086"
```

This form indicates that the preprocessor is not loaded. This will result in a small savings in compilation time.

Compiling

When your program is complete, it is ready for compiling. To compile a program, press the compile soft key. The key word, compile, will appear on the command line and the soft key configuration will change to:

<FILE> _____ _____ _____ _____ _____ _____ _____

Next, enter the name of the source file you want to compile. When the file has been entered, the soft key configuration will change to:

_____ **listfile options** _____ _____ _____ _____ _____

If you want a listing file for the compile program, press the listfile soft key. The key word, listfile, will appear on the command line and the soft key configuration will change to:

<FILE> display printer null _____ _____ _____ _____

At this point, choose the listing file you want as indicated by the soft keys. If you do not choose a listfile, the compiler will default to the predefined listfile that was choosen when the userid was set. (Refer to the System Software Reference manual for setting the userid.)

When you have chosen the listfile, you can choose compile options.

If you do not want to specify any options, press the RETURN key to compile your source file.

If you want to specify options, press the options soft key. The key word, options, will appear on the command line and the soft key configuration will change to:

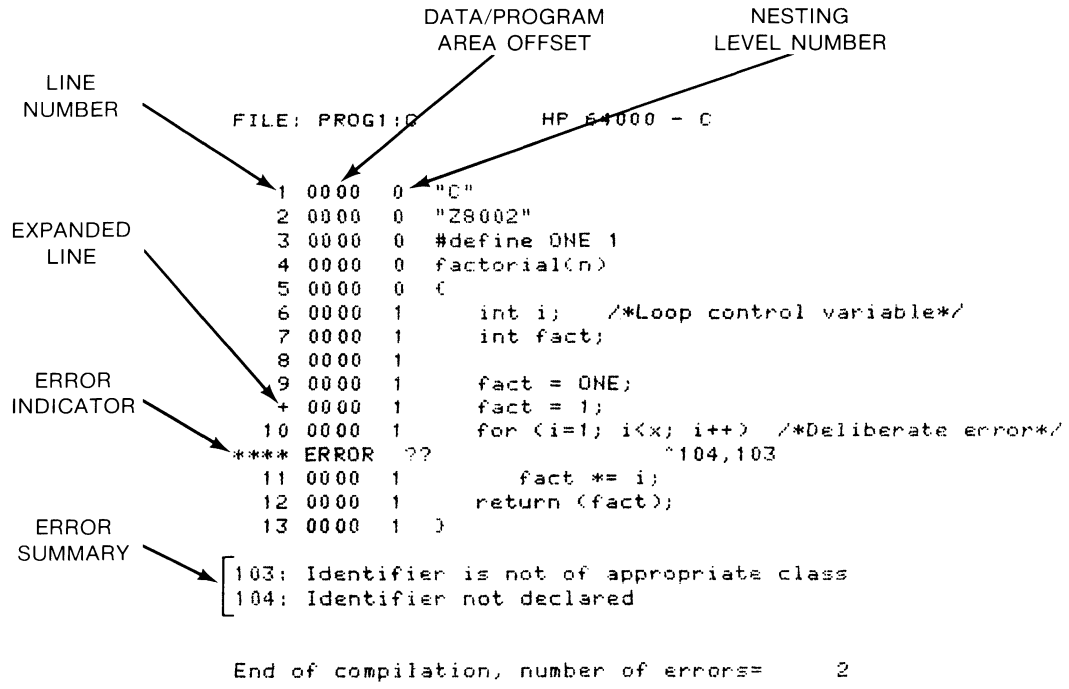
list nolist expand nocode xref _____ _____ _____

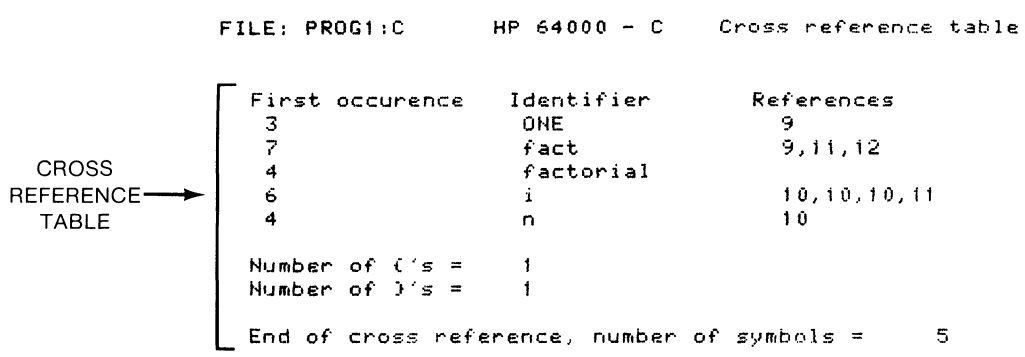
Press the soft key of the option or options you want to specify; then press the **RETURN** key to compile your source file.

Output Listings

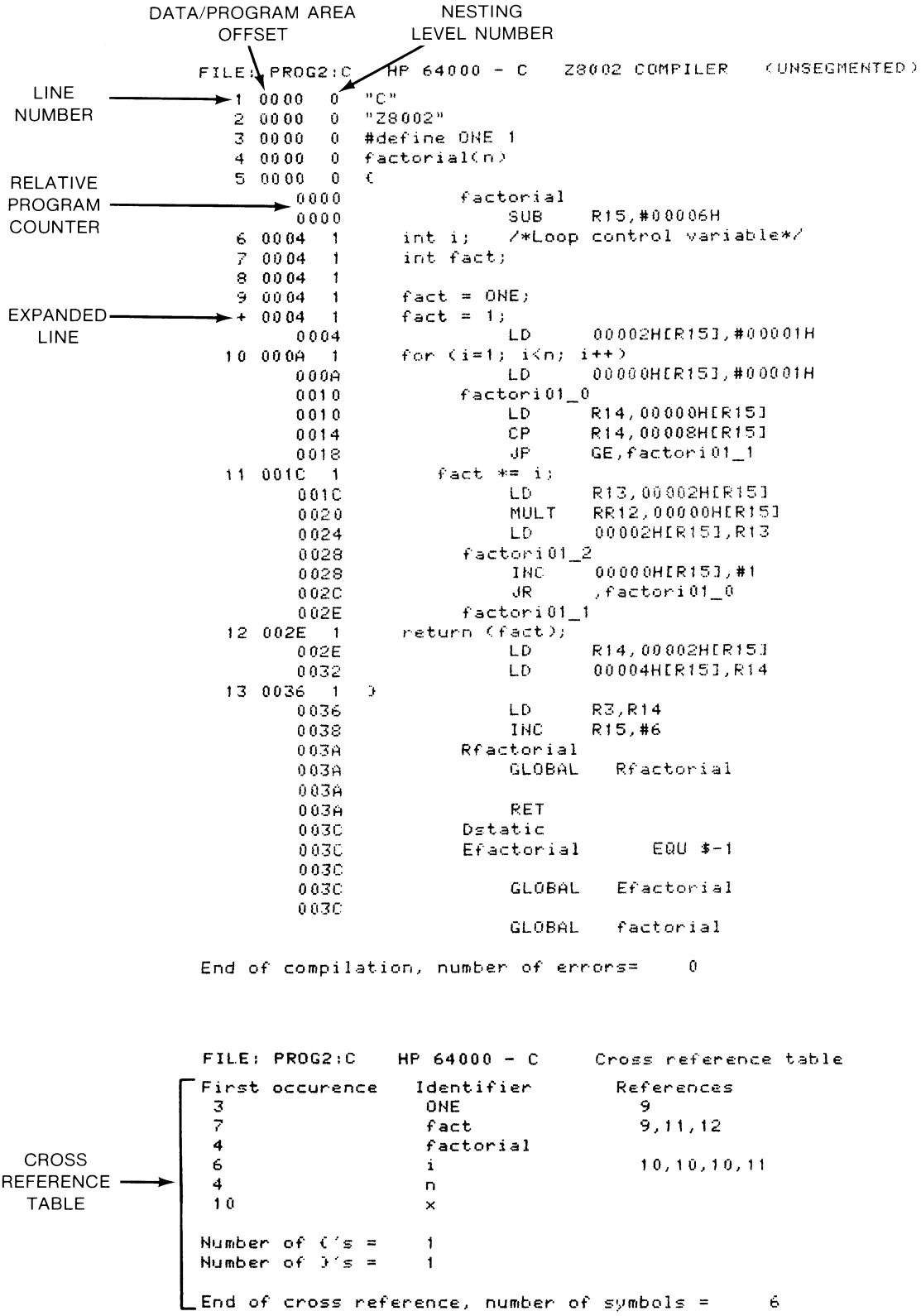
The compiler will output relocatable code and make listings according to the options specified or their default value. The following examples show typical output listings that the compiler will produce.

The following listing is an example of a Z8002 expanded output listing to the printer with a cross reference (xref) table.





The listing which follows is an example of an Z8002 compiler listing to the printer with the expand option.




```
FILE: PROG2:C    HP 64000 - C    Z8002 COMPILER    (UNSEGMENTED)

 1  0000   0  "C"
 2  0000   0  "Z8002"
 3  0000   0  #define ONE 1
 4  0000   0  factorial(n)
 5  0000   0  {
              0000          factorial
              0000              SUB     R15,#00006H
 6  0004   1      int i;   /*Loop control variable*/
 7  0004   1      int fact;
 8  0004   1
 9  0004   1      fact = ONE;
+ 0004   1      fact = 1;
              0004              LD      00002H[R15],#00001H
10  000A   1      for (i=1; i<n; i++)
              000A              LD      00000H[R15],#00001H
              0010          factori01_0
              0010              LD      R14,00000H[R15]
              0014              CP      R14,00008H[R15]
              0018              JP      GE,factori01_1
11  001C   1         fact *= i;
              001C              LD      R13,00002H[R15]
              0020              MULT    RR12,00000H[R15]
              0024              LD      00002H[R15],R13
              0028          factori01_2
              0028              INC     00000H[R15],#1
              002C              JR      ,factori01_0
              002E          factori01_1
12  002E   1      return (fact);
              002E              LD      R14,00002H[R15]
              0032              LD      00004H[R15],R14
13  0036   1  }
              0036              LD      R3,R14
              0038              INC     R15,#6
              003A          Rfactorial
              003A              GLOBAL  Rfactorial
              003A
              003A              RET
              003C          Dstatic
              003C          Efactorial      EQU $-1
              003C
              003C              GLOBAL  Efactorial
              003C
                                GLOBAL  factorial

End of compilation, number of errors=    0
```

```
FILE: PROG2:C    HP 64000 - C     Cross reference table

First occurence   Identifier       References
 3                 ONE                9
 7                 fact               9,11,12
 4                 factorial
 6                 i                  10,10,10,11
 4                 n
 10                x

Number of ('s =    1
Number of )'s =    1

End of cross reference, number of symbols =     6
```

Chapter 4

Linker Instructions

Introduction

A system application program, referred to as the linker (link), combines relocatable object modules into one file, producing an absolute image that is stored by the Model 64000 for execution in an emulation system or for programming PROMS. Interaction between the user and the linker remains basically the same regardless of which microprocessor assembler or compiler is being supported.

To prepare object code modules for the Model 64000 load program, the linker performs two functions:

a. Relocation: allocates memory space for each relocatable module of the program and relocates operand addresses to correspond to the relocated code.

b. Linking: symbolically links relocatable modules.

You may optionally select an output listing of the program load map and a cross-reference (xref) table. The linker also generates a listing that contains all errors that were noted. These error messages will contain a description of the error along with the file name and relocation/address information when applicable.

In addition to the above output listings, the linker constructs a global symbol file (link_sym type) and stores this file under the same file name assigned the absolute image/command file. This global file may be used for symbolic referencing during emulation. The link_sym file also contains the relocation address for all programs. This information is used to relocate asmb_sym type during emulation.

Linker Requirements

The following information is required by the linker:

a. File names of all object files to be loaded.

b. File names of libraries to be searched to resolve any unsatisfied externals.

c. Relocation information (load addresses for all relocatable areas).

d. Listing and debugging options as follows:

   1) List (Load Map): file/program name, relocatable load addresses, and absolute load addresses.

   2) Xref: symbols, value, relocation, and defining and referencing modules.

e. File name for command/absolute image file.

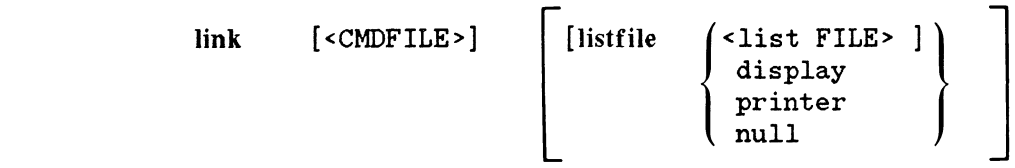
Since the linking operation will usually be required each time there is a software change and the information in items a through e remain constant for any given application, the linking control information is automatically saved in a command file with the same name as the absolute image file. The command file is distinguished from the absolute image file by file type.

Linker Syntax

The command line in which Model 64000 commands are entered is accessed by way of the development station keyboard. Each system application function (edit, compile, assemble, link, emulate, etc.) can be called using keyboard soft keys. A syntax description of the link command follows.

link

SYNTAX



Parameters:

<CMDFILE> - name of an established command/absolute image file.

listfile - soft key used to specify a destination for output listing other than the system default list file.

<list FILE> - name of the file where the linker output listing will be stored. If the assigned file name does not exist, a new list file is created.

display - designates the system CRT as the output listing destination.

printer - designates the system line printer as the output listing destination.

null - specifies that no listing is to be generated. Error messages, however, will be routed to the display area of the system CRT.

options - soft key used to specify linker options. The following options are available:

edit or noedit - specifies if an existing command file is to be edited.

list or nolist - specifies if a load map listing is to be generated.

xref or noxref - specifies if a xref listing is to be generated.

link (Cont'd)

Default Values

<CMDFILE>:	If a file name is not given, the linker will begin building a new command file.
listfile:	Linker output listing defaults to the device specified by the userid listfile default statement. If the listfile default statement does not specify an output device, the linker defaults to the null listing function.
options:	If the options softkey is not used, the linker will default to the list options specified in the command file and to noedit. If the options softkey is used, the linker will default to list, noxref, and noedit.

FUNCTION

The linker combines and relocates all object files into one absolute image file that can be loaded into the HP Model 64000.

DESCRIPTION

The linker may be called by one of two methods: simple calling or interactive calling.

The simple calling method is used when interaction with an established command file is not required. That is, the current information in the command file is valid and no changes are required.

The interactive calling method is used when building a new command file or when the information in a current command file needs revision.

How to Use the Linker

Simple Calling Method

a. Ensure that the following soft key prompts are displayed on the system CRT:

edit compile assemble link emulate prom_prog run --ETC---

b. Press the link soft key. The soft key configuration will be:

<CMDFILE> listfile options _____ _____ _____ _____ _____

c. The next prompt is CMDFILE. Type in the name of the established command file to be linked. The soft key configuration will change to:

_____ listfile options _____ _____ _____ _____ _____

d. If it is necessary to change the output listing destination, press the listfile soft key. The soft key configuration will change to:

<FILE> display printer null _____ _____ _____ _____

e. Route the linker output listing to the desired location by selecting the FILE option, or by pressing the display soft key, the printer soft key, or the null soft key.

NOTE

Pressing the null soft key results in no output listing. Error messages will be displayed on the system CRT.

f. If the FILE option is desired in step e, type in the file name under which the listing is to be stored. You can then review your output listing on the system CRT using the edit function and your assigned file name.

g. The soft key configuration will change to:

```
_____ _____ options _____ _____ _____ _____ _____
```

h. Refer to the "options" default description in the LINK SYNTAX definition block.

i. If the options soft key is not used, the linker defaults to the list options specified in the command file and to noedit. To override the command file list options (for this link only), press the options soft key. The soft key configuration will change to:

```
edit  nolist  xref  _____ _____ _____ _____ _____
```

If only the options soft key is used, the linker defaults to list, noxref, and noedit. Any of these defaults may be changed by pressing the appropriate soft key.

j. After accomplishing step i, press the **RETURN** key. The linker will link the relocatable modules and produce the desired output listing.

Interactive Calling Method

The interactive calling method allows the user to create a new linker command file or edit an existing linker command file.

a. Ensure that the following soft key prompts are displayed on the system CRT:

```
edit compile assemble link emulate prom_prog run --ETC--
```

b. Press the link soft key. The soft key configuration will change to:

```
<CMDFILE> listfile options _____ _____ _____ _____ _____
```

c. The user may start creating a new linker command file by not specifying any command file. An existing command file may be modified by specifying the command file name and the edit option.

NOTE

In the following paragraphs, the procedures are written for establishing a new command file. If an existing command file is being edited, just type in the changes required after each query. If no changes are required for a particular query, proceed to the next query. In all instances, to proceed to the next query, press the **RETURN** key.

d. The command query displayed in the command line of the system CRT is:

Object files? file1,file2,...,filen

The query asks for the names of the files to be linked and relocated. Type in the names of the files and then proceed to the next query.

NOTE

The soft key configuration "prompts" will change with each query from the linker. The soft key "prompts" indicate the type of information that is required.

Object files that are listed after the "Object files?" query may contain relocatable object modules, no-load files, and previously linked linker-symbol files (for global symbol references).

No-load files are differentiated from normal relocatable files by enclosing the no-load files in parentheses. Linker symbol files are specified by including the file type ':link_sym' in the file name.

Example:

FILE1,(FILE2,FILE3),FILE4:link_sym

NOTE

Refer to the paragraphs in this chapter that discuss no-load and link_sym files for additional information.

e. The next command query displayed in the command line on the system CRT is:

Library files? lib1,lib2,...,libn

Interrogation for library files is the same as for object files. After all object files have been linked, the linker determines if any external symbols remain undefined. The linker then searches the library files for object modules that define these symbols. The linker relocates and links only those relocatable modules that satisfy external references. Since a library file may contain more than one object module, all of its relocatable modules may not be linked. Refer to the paragraph in this chapter that discusses libraries and their construction.

NOTE

No-load files or linker symbol files, used for global referenceing, must not be listed after this query. The no-load and link_sym files can only be referenced during the "Object files?" query.

After typing in the list of reference library files (or if library files are not referenced in the program), proceed to the next query.

f. The next command query displayed in the command line on the system CRT is:

Load addresses:PROG,DATA,COMN=addr,addr,addr

This query allows selection of separate, relocatable memory areas for the different modules of the program. For example, if you type in the following addresses:

Load addresses:PROG,DATA,COMN=1000H,2000H,3000H

the linker will relocate the PROG file module to memory location starting at address 1000H, the DATA module will be relocated to

memory location starting at address 2000H, and the COMN module will be relocated to memory location starting at address 3000H.

NOTE

Load addresses may be entered using any number base (binary, octal, decimal, or hexadecimal); however, the addresses listed in the load map are give in hexadecimal only.

The default addresses are zeros. After entering the load addresses or if the default addresses are acceptable, proceed to the next query.

g. The next command query displayed in the command line on the system CRT is:

More files? no

The linker asks if more files are to be linked. If the response is yes, the linker begins interrogation again, allowing additional object and library files to be specified with new load addresses. When specifying new relocatable areas, the user may continue with the previously relocatable area by typing "CONT" in the appropriate field (or using the **CONT** soft key). The relocatable area is treated as if no new address was assigned.

Example:

```
Load addresses:PROG,DATA,COMN=0BCCH,CONT,3FFCH
```

The default condition to the "more files?" query is no. Proceed to the next query.

h. The next command query displayed in the command line on the system CRT concerns output listing options. It has the following syntax:

List,xref=on off

The linker asks you to specify what output listings are required. Using the on or off soft key, select, in the sequence indicated in the syntax statement (list, xref), the desired output listings. After inserting the requirements, proceed to the next query.

NOTE

The output listings indicated after the list,xref= query are the command file values that will be used during this and future link operations. They can be overridden by using the options soft key during the linker call.

i. The next command query displayed in the command line on the system CRT is:

Absolute file name=name

This final query from the linker allows you to assign a name to the new command/absolute image file that you are about to link. The absolute image file that is created by the linker is always associated with a link command file of the same name. A global symbol file is also established under the name of the command/absolute image file name. The global symbol file contains all global symbols and their relocation values.

After entering the absolute file name, press the **RETURN** key.

The linker will link, relocate the files, and save the linking information in the command file.

Linker Output

The linker listings may be output to the system display, line printer, or any file. The following information may be included in the linker output listing:

a. List (Load Map)

b. Cross-reference table

c. Error messages

NOTE

Certain error messages contain more than 80 characters and will not be completely displayed on the system CRT. However, complete error messages will be printed when using the line printer or a list file for listings.

List (Load Map)

A load map is a listing of the memory areas allocated to each relocatable file. The listing begins with the first file linked and proceeds to list all other linked files with their allocated memory locations. An example of a load map listing that will be printed on the system printer is as follows:

```
FILE/PROG NAME    PROGRAM   DATA   COMMON  ABSOLUTE    DATE                 TIME  COMMENTS
-------------------------------------------------------------------------------------------
KYBD:SAVE         0000                                 Thu. 5 Jun 1980.     11:37
EXCT:SAVE                                  0B00-0B34   Thu. 5 Jun 1980.     10:38
DSPL:SAVE                   A100                       Thu. 5 Jun 1980.     11:38
next address      0021      A121

REG1:SAVE         B000                                 Thu. 5 Jun 1980.     11:52
REG2:SAVE         B103                                 Thu. 5 Jun 1980.     11:53
REG3:SAVE         B206                                 Thu. 5 Jun 1980.     11:58
next address      B30C

Libraries
PARAMETER:SAVE    0021                                 Thu. 5 Jun 1980.     11:43
MULTEQUAT:SAVE    0221                                 Thu. 5 Jun 1980.     11:45
next address      0421      A121

XFER address=0B00  Defined by EXCT
No. of passes through libraries= 1
absolute & link_com file name=SETAG1:SAVE
Total# of bytes loaded= 0782
```

A brief description of each column in the listing is as follows:

a. FILE/PROG NAME - this column will contain the name of the files that are linked. In the event library files are referenced, not only will the master library file be listed, but its subsections will be indented to indicate that they are part of the main library file. No-load files will be displayed in parentheses (...).

b. PROGRAM - this column will indicate the first address (hexadecimal) of a memory block that contains the PROG relocatable code in the file listed in the FILE/PROG NAME column.

c. DATA - this column will indicate the first address (hexadecimal) of a memory block that contains the DATA relocatable code in the file listed in the FILE/PROG NAME column.

d. COMMON - this column will indicate the first address (hexadecimal) of a memory block that contains the COMN relocatable code in the file listed in the FILE/PROG NAME column.

e. ABSOLUTE - this column will indicate the hexadecimal addresses of a memory block that contains the absolute code assigned by the file listed in the FILE/PROG NAME column.

NOTE

The "next address" statement in the load map listing indicates the next available hexadecimal address in the PROG, DATA, or COMN memory areas. It may also be used to determine the number or bytes (words for 16-bit processors) that are contained in each area (next address - starting address = total bytes).

f. DATE - this column will indicate the date that the file listed in the FILE/PROG NAME column was assembled (assuming the system date/time clock was current).

g. TIME - this column will indicate the time that the file listed in the FILE/PROG NAME column was assembled (assuming the system date/time clock was current).

h. COMMENTS - this column will contain user comments entered during assembly by the assembler pseudo NAME instruction.

Cross-reference Table

The cross-reference table lists all global symbols, the relocatable object modules that define them, and the relocatable modules that reference them. An example of a cross-reference listing that will be listed on the system printer is as follows:

SYMBOL	R	VALUE	DEF BY	REFERENCES
DSPL6	P	0034	PGM68D	PGM68E
KYBD6	P	0001	PGM68K	PGM68E

A brief description of each column in the cross-reference listing is as follows:

a. SYMBOL - all global symbols will be listed in this column.

b. R (Relocation) - in this column a letter will identify the type of program module. The letters that are available and their definitions are:

```
A = Absolute
C = Common (COMN)
D = Data (DATA)
P = Program (PROG)
U = Undefined
```

c. VALUE - relocated address of the symbol.

d. DEF BY - this column will contain the file name that defines the global symbol.

e. REFERENCES - this column will list the file names that reference the global symbol.

"No-Load" Files

Files that are enclosed in parentheses in the "Object files?" query indicates to the linker that no code is to be generated for the file. Relocation and linking occurs in the same manner as if the file was a load file; however, the absolute image file generated by the linker does not contain the object code for the no-load file. No-load files may be useful in linking to existing ROM code or in the design of software systems requiring memory overlays.

Linker Symbol File

The linker creates a global symbol file for every link operation. The global file name is the same as the assigned command/absolute image file name assigned to the link. The user may find that linking to a common piece of code (global) is simplified by referring to that code by its linker-symbol file. This is accomplished by referencing the correct linker-symbol file name during the "Object files?" query by the linker. The linker-symbol file name referenced at the time of the query must be specified by the type ':link_sym'.

```
Object files? PGM68k,Pgm68D:link_sym
```

Library Files

Libraries are a collection of relocatable modules that are stored on the system disc and may be referenced by the linker.

If a library file name is given as a response to the "Object files?" query, all the relocatable modules in the library file will be relocated and linked. If a library file name is given as a response to the "library files?" query, only those relocatable modules that define the unsatisfied externals will be relocated and linked. The remaining relocatable modules in the library file are ignored.

It is possible to combine relocatables into a library by using the system library command. Refer to the System Software Reference Manual for a detailed description of the library command.

Error Messages

When an error is detected during the link process, the linker will determine if the error is fatal or nonfatal. If the error is classified as fatal, the linker will abort the linking process. If the error is nonfatal, the linker will continue the linking process, but will generate error messages that will be listed in the output listing. A description of each error message is given in the following paragraphs.

Fatal Error Messages

Upon encountering a fatal error the linker will display one of the following messages on the system CRT STATUS line. The linker will abort the link process and return control of the system to the monitor.

a. **Out of Memory in Pass 1.**

   The linker will issue this message to indicate that there is insufficient memory to accommodate the current operation. To correct this situation, reduce the number of files, global symbols, and/or external symbols used during the current link.

NOTE

As a general rule, the available memory space can handle programs containing approximately 3000 symbols. However, if cross-reference symbol tables are required, the symbol handling capability is reduced to approximately 1500 symbols.

b. **Out of Memory in Pass 2.**

The linker will issue this message to indicate that there is insufficient memory to accomodate the current operation. To correct this situration, reduce the number of files, global symbols, and/or external symbols used during the current link.

c. **Out of Memory in Xref.**

The linker will issue this message to indicate that there is insufficient memory to accommodate the building of a cross-reference table. This error does not affect the absolute file since it is created and stored prior to the linker attempting to build the cross-refernce file. To correct this situation, reduce the number of files, global symbols, and/or external symbols used during the current link.

d. **Target Processors Disagree.**

The linker will issue this message if the relocatable modules to be linked are designed for different processors. Ensure that all relocatable modules assigned for linking are written for the same type microprocessor.

e. **Checksum Error.**

The linker will issue this message if it is unable to read a relocatable file due to a checksum error or other irregularities in the file. To correct this situration, reassemble the relocatable file, then, re-link.

f. **Linker System Error.**

The linker will issue this message if it detects a hardware or software failure in the Model 64000. To correct this situation re-link the relocatable modules or run the hardware performance verification program.

g. **File Manager Errors.**

The linker will issue certain messages if the system file manager is unable to perform the specified file operation as requested by the linker. Refer to the System Software Reference Manual (Appendix A) for a list of File Manager Errors.

Nonfatal Error Messages

Upon encountering nonfatal errors, the linker will continue the link operation and print the error messages (except initialization errors) in the output listing. An error message that is listed will contain a description of the error and the name of the file where the error occurred. If the null list file is in effect, the linker will direct the error messages to the data area of the system CRT.

a. **Illegal entry: re-enter.**

   During initialization the linker will indicate in the STATUS line on the system CRT that the user has made an illegal response to an interrogation. To correct this situation, re-enter the proper response.

b. **Duplicate symbol.**

   During pass 1 of the link process, the linker detects that the same symbol has been declared global by more than one relocatable module. The first definition holds true. The relocatable module that first defines the symbol may be found in the cross-reference table. To correct this error, remove the extra global declarations.

c. **Load address out of range.**

   The linker has tried to relocate code beyond the addressing range of the specified microprocessor. To correct this situation, reassign the relocatable addresses.

d. **Multiple transfer address.**

   During pass 1, the linker finds that the transfer address has been defined by more than one relocatable module. The first definition holds true. The relocatable module that first defined the transfer address will be given at the conclusion of the linking. To correct this situation, remove the extra transfer address. Reassemble the amended relocatable moldule; then, re-link. If a xfer address is defined by both a nonload program and a load program, no error will be given. The load program xfer address takes precedence.

e. **Undefined symbol.**

During pass 2, the linker finds that a symbol has been declared external but not defined by a global definition. To correct this situation, define the symbol.

f. **Out of memory in xref.**

Unlike the fatal error (Out of Memory in Xref), this error occurs when memory space is available for a complete symbol table but only a portion of the cross-reference table. The linker will complete the xref operation, listing only that portion of the cross-reference table for which memory space was available. To correct this situation, reduce the number of files, global symbols, and/or external symbols used during the current link.

g. **Memory overlap.**

Relocatable program areas have been overlapped in memory. The error message will list the program names and the overlapping areas.

h. **Address out of range.**

The operand address is not within a valid addressing range for the specific microprocessor involved.

Appendix A

Compile Time Errors

The following errors are detected by the first pass of the compiler. Errors are also detected by the second pass of the compiler. These errors are microprocessor dependent and are listed in the microprocessor-dependent supplements.

When errors appear in groups, usually only the first message is meaningful. This is because some of the following error messages appear as a result of the first error. In particular, any time the WARNING message (number 0) is indicated, the compiler will attempt to resume compilation at the next logical token. In some instances, correctly. In these situations, the user should use the editor function to correct the first error.

List of Error Messages

```
 0:  WARNING: attempted syntax error recovery here
 1:  Error in simple type
 2:  Identifier expected
 4:  ' ) ' expected
 5:  ' : ' expected
 6:  Illegal symbol
 7:  Error in parameter list
 9:  ' ( ' expected
10:  Error in type
11:  ' [ ' expected
12:  ' ] ' expected
14:  ' ; ' expected
15:  Integer expected
16:  ' = ' expected
18:  Error in declaration part
19:  Error in field list
```

List of Errors (Cont'd)

20: ' . ' expected
21: ' * ' expected
25: Statement begin symbol expected

30: Type name expected
33: " ' " expected
34: ' " ' expected
35: ' { ' expected
36: ' } ' expected
37: ' , ' expected
38: Type name or storage class expected

40: WHILE expected
41: Illegal storage class
42: Undeclared parameter
43: Duplicate definition of parameter
44: Multiple storage classes
45: Multiple types

50: Error in constant
58: Error in factor
59: Error in variable

60: Lvalue expected
61: Pointers must be same size
62: ELSE without IF
63: Break or continue statement not in FOR, WHILE, DO, or SWITCH
64: CASE or default without SWITCH
65: Duplicate CASE or fault
66: SWITCH must be type int or enum

70: Only type int may be field
71: Field larger than wordsize
72: Named field may not be size zero

List of Errors (Cont'd)

101: Identifier declared twice
103: Identifier is not of appropriate class
104: Identifier not declared

129: Type conflict of operands

131: Tests on equality allowed only
134: Illegal type of operand(s)

145: Type conflict

156: Multidefined case label

165: Multidefined label
166: Multideclared label
167: Undeclared label
168: Undefined label

182: Array of functions not allowed
183: Function can not return array
184: Function can not return function
185: Array dimension may not be negative
186: Structure member may not be function
187: Cannot take address of fixed parameter function;
i.e. ' (' expected

190: Structure may not contain instance of itself
191: Member defined twice
192: Type of variable may not be intialized
193: Variable may not be initialized
194: Missing ' { ' on array or structure initialization
195: Static or external variable may only be initialized to constant
196: Variable declared external or type def may not be initialized
197: Only static and external arrays and structures may be initialized
198: Too many initializers

List of Errors (Cont'd)

200: Cannot initialize non pointer sized integer to address
203: Integer constant exceeds range
207: Overflow in real operation
208: Error in real operation

210: Only first dimension of array may be unspecified

220: Only one type may be declared in $USER_DEFINED$ type def declaration

250: Too many nested scopes of identifiers
254: Too many long constants in this procedure
255: Too many errors on this source line
256: Too many external references
257: Too many externals
259: Expression too complicated

270: # not in column 1 or PREPROCESS not specified. Remainder of line ignored.

280: Preprocessor syntax error
281: Unimplemented preprocessor instruction
282: Number of parameters does not agree with macro declaration
283: Identifier not #defined
284: Macro may not have more than 20 parameters
285: #if without #endif
286: #if instruction may not contain multiline macro
287: #else of #endif without #if
288: Preprocessor stack overflow. Simplify constant expression
289: Error in constant expression

304: Element expression out of range

398: Implementation restriction

List of Errors (Cont'd)

400: State stack overflow; break up program; parsing stopped
401: Fatal parser error; previous error; parsing stopped
402: End of source before end of compilation
403: Symbol table overflow; delete symbols; parsing stopped
404: Semantic stack overflow; break up program; parsing stopped
405: End of source before end of comment
406: Out of expression tree storage; simplify expression
407: Pop of empty stack; caused by previous error; parsing stopped
408: Illegal stack entry; missing semicolon?; parsing stopped

410: Too many indirect; simplify expression
411: Constant expression expected
412: More than 20 syntax errors; parsing stopped
414: More than 255 subroutines; break program into modules
416: More than 255 large constants

450: Feature not implemented
455: Language extensions used in extensions off mode
456: Too many user defined operation types

500: Warning: illegal compiler option; option ignored
503: Warning: source line exceeds allowed length
504: Warning: non-standard feature used
506: Warning: +32768 is treated as -32768 by the compiler

511: Warning: variable assumed to be function returning integer
512: Warning: expanded line larger than 240 characters; multiple lines created
513: Warning: duplicate macro name; New definition holds
514: Warning: function treated as variable parameters, not fixed
515: Warning: integer not pointer size

Appendix B

ENUM <ENUMERATION> TYPES

The enum type is similar to Pascal scalar types. The declaration of an enum type is similar to that of a struct or union type:

```
enum OPT_TAG_FIELD
{ENUMERATOR,ENUMERATOR,...ENUMERATOR};
```

where OPT_TAG_FIELD is an optional tag field similar to the tag field or a structure.

ENUMERATOR is in one of the following forms:

```
IDENTIFIER

IDENTIFIER = CONSTANT
```

Normally, the identifiers represent consecutive integers starting at zero. If the second form is used, the appropriate identifier represents the constant and all subsequent identifiers are consecutive.

The form:

```
enum TAG_FIELD
```

is also available. This declares a variable to be of a previously declared enum type.

Examples:

```
enum COLOR {black,brown,red,orange,yellow,green,blue,
            violet,grey,white} band1,band2,band3;

 This declares three variables band1, band2, and band3 to
 be of type COLOR.
```

and

```
enum QUALITY {poor=1,acceptable,good,excellent};

 This declares QUALITY to have values poor = 1,
 acceptable = 2, good = 3, and excellent = 4.
```

and

```
enum QUALITY x,y;
```

This declares x and y of type QUALITY.

and

```
enum {zero,three=3,seven=7,eight,m1=-1,another_zero}
v1,v2;
```

This declares v1 and v2 to be an enumerated type where zero=0, three=3, seven=7, eight=8, m1=-1, another_zero=0.

Although **enum** types are represented internally by integers, they are not integers and are not compatible with them.

No checking is done to see if a number is valid for a given **enum** type. For example:

```
v1=three; v1++;
```

will not cause an error even though there is no value whose representation is 4.

Appendix C

Compiler Generated Symbols for C Programs

Compiler Generated Labels

This section discusses compiler generated labels. Whenever the symbol "func" appears it refers to the name of the enclosing function truncated, if necessary, so that the total label will fit into 15 characters. The external declaration section is considered to be a function by the name of static (STATIC if UPPER_KEYS is ON) for purposes of label generation.

Function Entry

The function entry has the label func, i.e. the function name itself. This label will be declared global if the function is global.

End Label

The end of a function is indicated by the label **Efunc**. This label marks the end of the PROG section associated with the function. This includes any data associated with the function which is in the PROG section (due to the value of the SEPARATE option. The end label will be declared global if the function is global. This label may be used in a trace as in trace only address range func thru Efunc.

Return Label

The return instruction from a function is always labeled **Rfunc**. This label will be declared global if the function is global.

Data Label

If a function has an associated data area in memory, this data area will be marked **Dfunc**. The data label is never global. It may be used in tracing local data as in trace address Dfunc+N where N may be calculated from the relocation information in the listing.

User Labels

A label defined by the user within a function, is given the same name by the compiler. These labels are always local.

Jump Labels

These are labels generated by compiler jumps from statements such as if, for, while, switch, do, etc... The labels are of the form funcLNN_XXXX where func is truncated to 7 characters, NN is a unique number based on the function and XXXX is a unique number for the labels. Jump labels are always local and will normally be unique within a program.

Certain processors may make use of other types of labels. See the specific processor supplement manual for details.

Duplicate Symbols

Although these labels aid in program tracing, they generate a potential for duplicate symbols. If these symbols are local, this will not cause a problem unless the ASM_FILE is assembled, or an attempt is made to trace on one of these variables. If the symbols are global, an error will occur at link time. The following can cause duplicate symbols.

If the first 14 characters of two function names match, the D, E, and R labels will be duplicate. If the function func exists, as well as a user symbol such as Efunc (any function or global variable) a duplicate symbol will occur. Using the same label number in two functions will cause duplicate local symbols. Using a reserved assembler symbol (such as a register name) may cause duplicate symbol errors in the ASM_FILE.

In the following example, note the following:

- The variables a, b, and c can be accessed as Dstatic, Dstatic+4, and Dstatic+8.
- The procedure has_a_long_name was truncated to form the other labels. However, emulation also truncates identifiers to fifteen characters (i. e. Dhas_a_long_name and Dhas_a_long_nam are equivalent).
- The variable x can be accessed as Dhas_a_long_nam.
- The use of two labels named my_label, although legal in C, will cause duplicate symbols if the file is assembled, and cannot be traced.

Example:

```
   1 0000  0  "C"
   2 0000  0  "Z8002"
   3 0000  0  $RECURSIVE OFF$
   4 0000  0     static int a;
   5 0002  0     static int b;
   6 0004  0     static int c;
   7 0006  0
   8 0006  0  has_a_long_name()
   9 0006  0  {
        0000          has_a_long_name
  10 0000  1     int x;
  11 0000  1
  12 0000  1     if (a==b) x = 5;
        0000               LD     R14,Dstatic
        0004               CP     R14,Dstatic+00002H
        0008               JP     NE,has_a_101_0
        000C               LD     Dhas_a_long_nam,#00005H
  13 0012  1     else x = 3;
        0012               JP     ,has_a_101_1
        0016          has_a_101_0
        0016               LD     Dhas_a_long_nam,#00003H
        001C          has_a_101_1
  14 001C  1     my_label:;
        001C          my_label
  15 001C  1  }
        001C          Rhas_a_long_nam
        001C              GLOBAL   Rhas_a_long_nam
        001C
        001C              RET
        001E          Dhas_a_long_nam
        001E              RMB    00002H
  16 0006  0
  17 0006  0  static func_2()
  18 0006  0  {
        0020          Ehas_a_long_nam EQU $-1
        0020
        0020              GLOBAL   Ehas_a_long_nam
        0020          func_2
  19 0020  1     my_label:;
        0020          my_label
  20 0020  1  }
        0020          Rfunc_2
        0020              RET
        0022          Dfunc_2
        0022              RMB    00000H
        0022          Dstatic
        0022              WVAL   00000H
        0024              WVAL   00000H
        0026              WVAL   00000H
        0028          Efunc_2         EQU $-1
        0028
                          GLOBAL   has_a_long_name
```

Index

The following index lists important terms and concepts of this manual along with the location(s) in which they can be found. The numbers to the right of the listings indicate the following manual areas:

- Chapters - references to chapters appear as "Chapter X", where "X" represents the chapter number.
- Appendices - references to appendices appear as "Appendix Y", where "Y" represents the letter designator of the appendix.
- Figures - references to figures are represented by the capital letter "F" followed by the section figure number.
- Other entries in the Index - references to other entries in the index are preceded by the word "See" followed by the reference entry.

a

c

d

e

f

g

h

i

j

l

n

o

p

r

s

t

u

w

x

64800-90907, FEBRUARY 1983
Replaces: 64800-90907, June 1982

PRINTED IN U.S.A.