64000

HP64000 Logic Development System

File Format Reference Manual

HEWLETT PACKARD

HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

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
File Format Reference Manual
64980-90933, July 1984

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.

File Format
Reference Manual

LOGIC SYSTEMS DIVISION
COLORADO SPRINGS, COLORADO, U. S. A.

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. Vertical bars in a page margin indicates the location of reprint corrections.

First Printing ..... May, 1984

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont'd)

TABLE OF CONTENTS (Cont'd)

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS (Cont'd)

LIST OF ILLUSTRATIONS (Cont'd)

LIST OF TABLES

LIST OF TABLES (Cont'd)

Chapter 1

64000 SYSTEM FILES AND THEIR ORGANIZATION

INTRODUCTION

This manual provides descriptions of the Hewlett-Packard 64000 System file structures. However, it is not intended to provide methods for manipulating these files. For that information, please refer to the applicable HP documentation on Simulated I/O, terminal mode and HP protocol, copy command, HOST Pascal, and HPIB transfers.

The first two chapters of this manual provide an overview of the 64000 System file types, their generation, and their usage. Succeeding chapters more fully describe the contents of each file type by showing the overall file structure and by presenting the specific details of each record type in the file.

FILE TYPES

A collection of information is stored on a 64000 system disc as a file. Each 64000 file has four associated attributes which give it a unique identification: the name, the userid, the disc logical unit (LU) number, and the file type. For more information on the name, userid, and LU number, refer to the System Software Reference Manual.

There are 15 designated file type names currently in use by the 64000 system. These are the names which appear in the column labeled "TYPE" in a directory listing. Each file type name has a file type number associated with it. The file type number is stored in one 16-bit word. The 64000 System file type names and numbers are shown in Table 1-1.

Table 1-1. 64000 System File Type Numbers and Associated Names

File Type #	File Type Name	Abbreviation
1	system	
2	source	
3	relocatable	reloc
4	absolute	
5	listing	
6	emulation command	emul_com
7	linker command	link_com
8	trace	
9	prom	
10	data	
11	assembler database	asmb_db
12	assembler symbol	asmb_sym
13	linker symbol	link_sym
0800H–08FFH	temporary	temp
8000H–8FFFH	device	
all others	simulated I/O	sim_I/O

Chapter 2

64000 SYSTEM FILE GENERATION AND USAGE

SYSTEM SOFTWARE GENERATION

The relationships between the files generated and used by the 64000 System in the process of creating software for execution on target microprocessors are shown in Figure 2-1.

EDITOR

The 64000 System Editor allows the user to create new files of type :source or type :listing or to edit existing source or listing files. In the process of generating microprocessor software, files of type :source must be used. The next step is to assemble or compile the source file.

ASSEMBLERS

Assemblers in the 64000 System accept source files as input and typically create two files, each having the same name, userid, and disc LU number as the source file which was assembled. The first, a file of type :reloc, contains the relocatable code, along with global and external symbol information. The second, of type :asmb_sym, is an assembler symbol file which contains all of the symbols defined in the source file along with their relocatable addresses.

Additionally, a file of type :listing, showing the code generated by each assembly language instruction (in hexadecimal) and symbol usage, will be generated by the 64000 System Assembler when the user requests it. For more detailed information on file generation and usage, refer to the 64000 System Assembler/Linker Reference Manual.

PASCAL AND "C" COMPILERS

The 64000 System Pascal and "C" Compilers accept source files as input and typically create two files, each having the same name, userid, and disc LU number as the source file which was compiled. The first, a file of type :reloc, contains the relocatable code, along with global and external symbol information. The second, of type :asmb_sym, is an assembler symbol file which contains all of the local compiler generated symbols, along with their relocatable addresses.

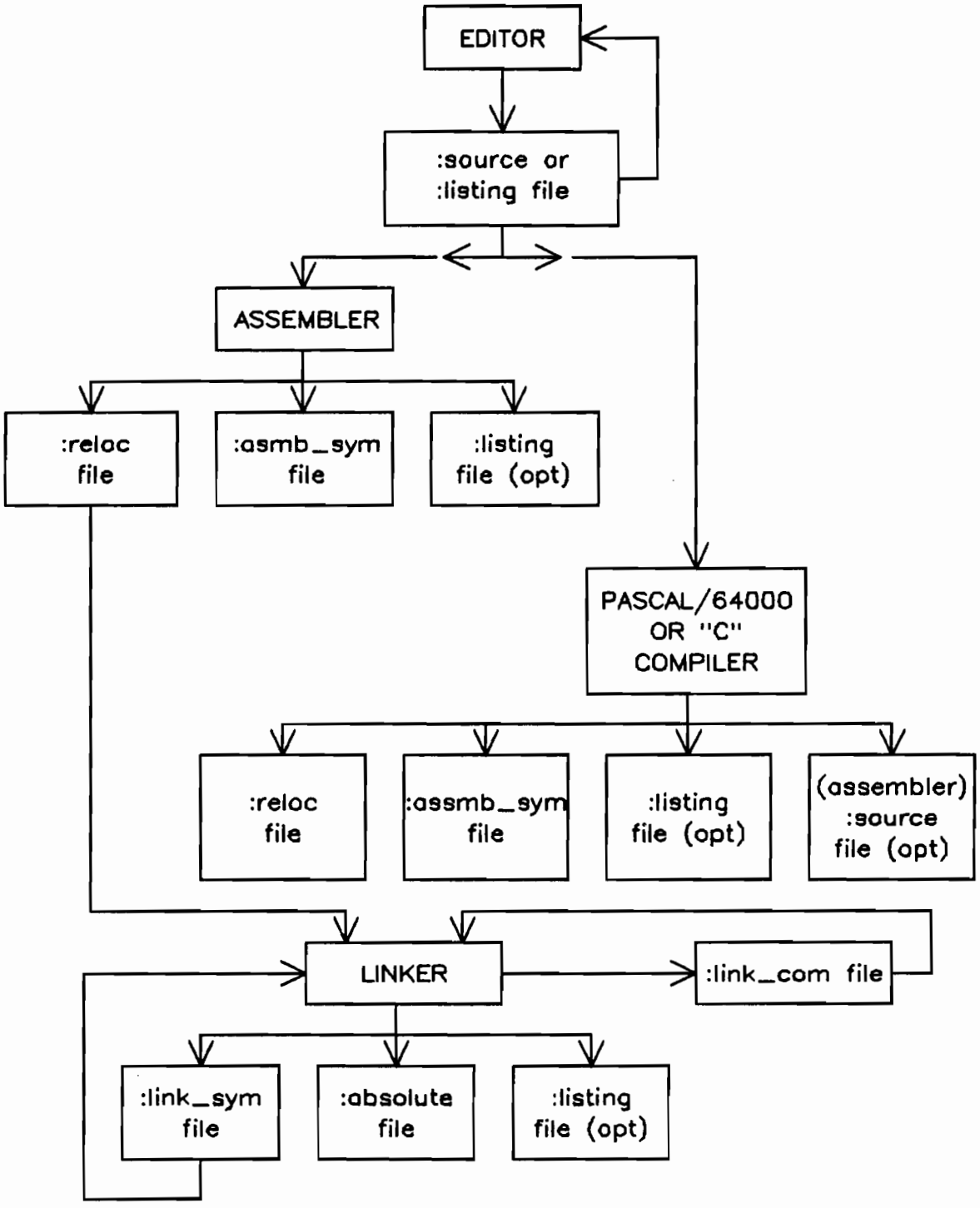

Figure 2-1. File Generation and Usage During Microprocessor Software Generation Activities

The user may specify the generation of files of type :listing or :source by the 64000 System Pascal and "C" Compilers. The listing file gives assembly language code generated and symbol usage, while the source file consists of assembly language code which can be assembled to generate separate relocatable and assembler symbol files.

For more information on file generation and usage, refer to the Pascal/64000 Compiler Reference Manual or the C/64000 Compiler Reference Manual.

PASCAL COMPILER GENERATED SYMBOLS. Assembler symbol files which are generated by the Pascal/64000 Compilers contain any user-defined labels within procedures. The compiler generates these labels in the form LABEL__NN, where NN is the number of the label; they are always local. In addition, the asmb__sym file contains several types of symbols which are defined by the compiler itself. Line number labels of the form #dddd, where dddd is a fixed-length, four-digit number which is right-justified and blank-filled, are specified for each line of source code. Their addresses correspond to those of the first executable line of code which is generated as a result of the source statement. These line numbers are then referenced by measurement system module software upon user request.

Also defined is a Procedure Entry Label which has the name of the procedure itself. It will be a global symbol if the procedure is global and its address corresponds to that of the first executable line of assembly language code generated for the procedure. (The main program is always global.) Other symbols generated by the Pascal/64000 Compilers are of the form Xfunc, where X is a character and func is the user-defined procedure name, truncated (if necessary) so that the total label has a maximum length of 15 characters.

The first of these symbols, called an End Label, consists of the letter "E" concatenated with the procedure name; it is generated for each procedure. Its address is the same as that of the last byte of assembly language code generated for the procedure. This includes any data associated with the procedure which is in the PROG area. The End Label will be global if the procedure is global.

The return instruction from a procedure is labeled with the letter "R" concatenated with the procedure name and called a Return Label. Its address corresponds to that of the assembly language return instruction. The Return Label will be global if the procedure is global.

If a procedure has an associated data area in memory, the beginning of that area will be labeled with the letter "D" concatenated with the procedure name. This is called a Data Label, and it is always a local symbol.

Other labels may be used by specific processor Pascal/64000 compilers. For more information, refer to the appropriate Pascal/64000 Compiler Supplement Manual.

"C" COMPILER GENERATED SYMBOLS. Assembler symbol files which are generated by the 64000 System "C" Compilers contain any user-defined labels within functions. These labels are always local. In addition, the asmb_sym file contains several types of symbols which are defined by the compiler itself. Line number labels of the form #dddd, where dddd is a fixed-length, four-digit number which is right-justified and blank-filled, are specified for each line of source code. Their addresses correspond to those of the first executable line of code which is generated as a result of the source statement. These line numbers are then referenced by measurement system module software upon user request.

The first of these symbols, called a Function Entry Label, has the name of the function itself. It will be a global symbol if the function is global and its address corresponds to that of the first executable line of assembly language code generated for the function. Other symbols generated by the 64000 System "C" Compilers are of the form Xfunc, where X is a character and func is the user-defined function name, truncated, if necessary, so that the total label has a maximum length of 15 characters.

The first of these symbols, called an End Label, consists of the letter "E" concatenated with the function name; it is generated for each function. Its address is the same as that of the last byte of assembly language code generated for the function. This includes any data associated with the function which is in the PROG area. The End Label will be global if the function is global.

The return instruction from a function is labeled with the letter "R" concatenated with the function name and called a Return Label. Its address corresponds to that of the assembly language return instruction. The Return Label will be global if the function is global.

If a function has an associated data area in memory, the beginning of that area will be labeled with the letter "D" concatenated with the function name. This is called a Data Label, and it is always a local symbol.

Other labels may be used by specific processor "C" compilers. For more information, refer to the appropriate "C" Compiler Supplement Manual.

LINKERS

The final step in the microprocessor software generation process on the 64000 System is the creation of the absolute file. This is accomplished by a 64000 System Linker which accepts relocatable and/or linker symbol files as inputs and typically produces three files which have the same name, userid, and LU number as specified in the answer to the question "Absolute file?" which is asked during linker configuration.

The first, a file of type :absolute, contains the absolute data to be used to program a PROM or is loaded into the microprocessor memory space by the emulator along with information indicating where the code is to be loaded. If only :link_sym files have been linked, the absolute file consists solely of a header record with no data records specified.

A linker symbol file, type :link_sym, is the second file type generated by the linker. It contains a list of the sources from which the relocatable files that were linked were derived and the starting addresses of each of the segments of code (PROG, DATA, COMN, and ABSOLUTE) that were produced for that file. Additionally, it contains a list of all global symbols from those sources and their addresses. (Global symbols are those symbols which are defined in the source file as being global.) If any :link_sym files have been linked, the global symbols from each of those files, along with their addresses, are included in the new linker symbol file which is created. This new linker symbol file DOES NOT contain the list of the original source files from which those global symbols were gathered.

A file of type :link_com is the third file type generated by the 64000 System Linker. This file stores the answers given to the questions that were asked during linker configuration. configured. Future links may be accomplished simply by calling up the link command file and having the linker read the configuration information.

A file of type :listing, showing the load map for the absolute file and global symbol usage, will be generated by the 64000 System Linker when the user requests it.

For more detailed information, refer to the 64000 System Assembler/Linker Reference Manual, the Pascal/64000 Compiler Reference Manual, or the C/64000 Compiler Reference Manual.

SYMBOL DEFINITIONS AND LOOK-UP

An important part of the 64000 System microprocessor software generation process is the creation of symbols which may be used to reference addresses from the 64000 Measurement System modules. Two types of symbols are identified in the 64000 System: local and global; they may be defined by the user in the source program or supplied by the 64000 System Compilers. (Refer to "PASCAL AND "C" COMPILERS" in this chapter.)

LOCAL SYMBOLS. Local symbols are those symbols which are defined in, and only referenced by, a single source program. They are not declared to be global by the user. Local symbol names and their relocatable addresses are stored in the :asmb_sym file. It is important to note that the address stored with the local symbol name in the asmb_sym file is not necessarily the same address that will be associated with the symbol in the absolute file.

GLOBAL SYMBOLS. Global symbols are those symbols which are defined in one source program and may be referenced by it and any other source program. They are declared by the user to be global. Global symbol names and their associated addresses in the absolute file are stored in the :link_sym file.

SYMBOL LOOK-UP. When the user specifies a symbol name instead of an actual address in a Measurement System module such as the Software Performance Analyzer or the Emulation System, the address value of the symbol must be looked up or calculated. This is done using the appropriate :link_sym and :asmb_sym files. Each Measurement Module has a particular absolute file name associated with it at any time. In the 64000 System Emulators, for example, that is the name of the last file to be loaded via a "load_memory" or "continue" command. Similarly, the link_sym file to be searched for symbol information is the one with the same name as the absolute file currently associated with the Measurement System module. For more information, refer to the appropriate Measurement System module reference manual.

When a symbol is specified in a Measurement System module as SYMBOL_NAME, the appropriate :link_sym file will be searched to determine whether it is a global symbol. If it is found, the address will be obtained and the symbol resolution is complete. If it is not found, then a message indicating that it is not a global symbol will be displayed, and the user has the opportunity to define the source file in which the local symbol was specified. This definition takes the form SYMBOL_NAME:SOURCE_NAME.

When a symbol is specified in a Measurement System module as SYMBOL_NAME:SOURCE_NAME, the appropriate :link_sym file is searched to find the SOURCE_NAME. If the source name is not found, a resulting message is displayed for the user. If the source name is found, information on relocating symbols in the areas that were used by the source (PROG, DATA, COMN, etc.) is obtained and the :asmb_sym file of the same name is then searched to find SYMBOL_NAME. If the symbol name is not found, a resulting message is displayed for the user.

When the symbol name is found, its relocatable address is obtained and combined with the pertinent information from the :link_sym file to complete the symbol address resolution. It is important to note that the 64000 System must have access to all of the appropriate files in order to do the symbol address resolution if symbolic reference is desired.

EMULATORS

Files generated and used by the 64000 System Emulators are outlined in Figure 2-2. These emulators can configure from or generate configuration information files of type :emul_com. Emulation command (emul_com) files are given the name provided to the emulation configuration question "Command file name?"

If no command file name is given, a file of type :emul_com named EcnfgXY:HP (where X is the System Bus Address of the station in use and Y is the card slot number containing the Emulation Control Card of the Emulator in use) is automatically generated or rewritten by the Emulation System. This file may then be used to configure the emulator automatically in the future, either by the system if the emulator is exited and re-entered with a "continue", or by the user if so specified.

The 64000 System Emulators can reload into the emulator or store away information gathered when a trace command was executed in files of type :trace.

Upon leaving an emulation session, the 64000 Emulation System creates or rewrites a file of type :trace named EcnfgXY:HP (where X is the System Bus Address of the station in use and Y is the card slot number containing the Emulation Control Card of the Emulator in use). This file may then be used to access that trace information automatically in the future, either by the system if the emulator is exited and re-entered with a "continue," or by the user if so specified. Similarly, microprocessor memory may be loaded from or stored to files of type :absolute.

Symbol address information from the files of type :link_sym and :asmb_sym associated with the last absolute file which has been loaded by the Emulator can be accessed by the 64000 System Emulators. For more information on how this works, refer to "SYMBOL DEFINITIONS AND LOOK-UP" in this chapter.

Finally, files of type :listing will be generated by the 64000 System Emulators when the user issues a "list" or "listfile" command. For more information, refer to the Emulator Reference Manual for the specific microprocessor emulator being used.

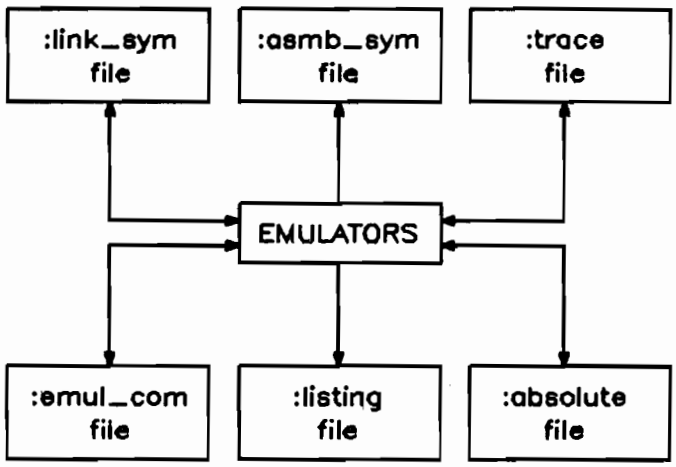

Figure 2-2. File Generation and Usage During Emulation

SOFTWARE PERFORMANCE ANALYZER

The files generated and used by the 64000 System Software Performance Analyzer are outlined in Figure 2-3.

The Software Performance Analyzer (Model 64310A) works in conjunction with a 64000 System Emulator. It can configure from or generate configuration information files of type :trace.

When a user specifies the name of the absolute file he is working with, symbol address information from the files of type :link_sym and :asmb_sym associated with this absolute file is available to the Software Performance Analyzer. For more information on how this works, refer to "SYMBOL DEFINITIONS AND LOOK-UP" in this chapter.

In order for the 64000 System Software Performance Analyzer to automatically reference a Pascal or "C" procedure, the Return Labels and End Labels must be available to it (refer to PASCAL AND "C" COMPILERS" on Page 2-1). Additionally, to specify a module as a range of compiler source line numbers, the assembler symbol file for the source must contain line number labels.

The file spa_table:HP of type :data provides the algorithms for use in the statistical analysis done by the Software Performance Analyzer.

Finally, files of type :listing will be generated by the 64000 System Software Performance Analyzer when the user issues a "copy" command. For more information, refer to the Software Performance Analyzer Reference Manual.

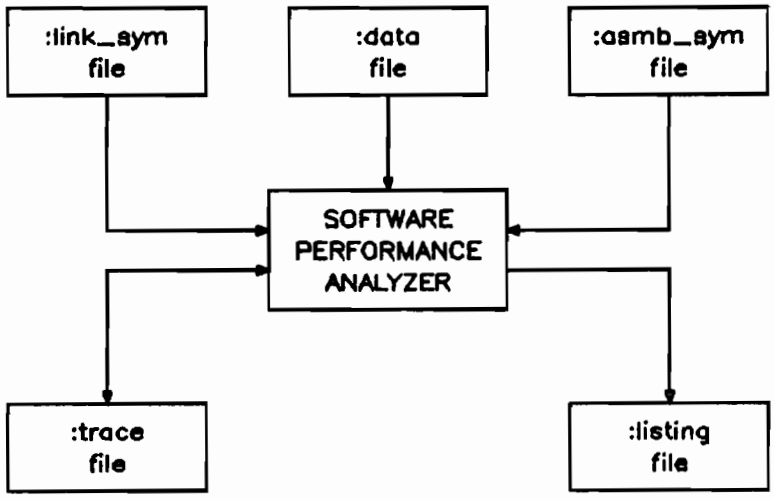

Figure 2-3. File Generation and Usage for Software Performance Analyzer

TIMING ANALYZER

The files generated and used by the 64000 System Timing Analyzer are outlined in Figure 2-4. The Timing Analyzer (Model 64600A) can configure from or generate configuration information files of type :trace. Additionally, files of type :listing will be generated by the 64000 System Timing Analyzer when the user issues a "copy" command. For more information, refer to the Timing Analyzer Reference Manual.

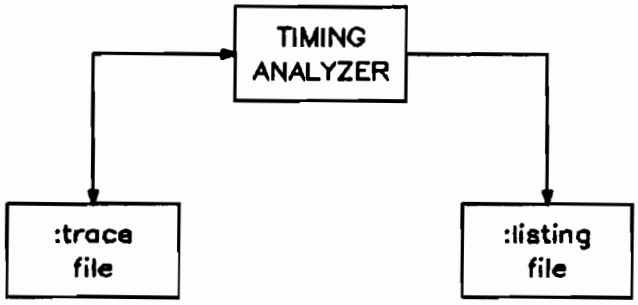

Figure 2-4. File Generation and Usage for Timing Analyzer

STATE/SOFTWARE ANALYZER

The files generated and used by the 64000 System State/Software Analyzer are outlined in Figure 2-5. The State/Software Analyzer (Model 64620A) can configure from or generate configuration information files of type :trace. This trace file exists in one of two forms: the first contains only analyzer configuration information and the second contains trace data as well as the configuration information.

When using the symbol information from the :asmb__sym and link__sym files, the state analyzer creates an :asmb__db file. This file allows the analyzer to quickly access the symbol table information. Additionally, files of type :listing will be generated by the 64000 System State/Software Analyzer when the user issues a "copy" command. For more information, refer to the State/Software Analyzer Reference Manual.

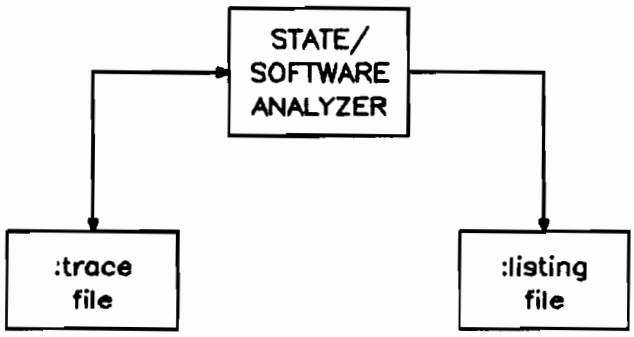

Figure 2-5. File Generation and Usage for State/Software Analyzer

Chapter 3
SYSTEM FILE FORMAT

SYSTEM FILE (FILE TYPE 1)

System files contain software to run the 64000 system. System file names begin with a lower-case letter, making normal access to these files by a system user impossible. All system files are stored under userid :HP. System files are not stored in a record format but are images of memory. The only access of system files permitted is the ability to display, copy or remove modules of system files from a floppy disc equipped station via the system generator (sys_gen) program.

Chapter 4

SOURCE AND LISTING FILE FORMATS

SOURCE FILE (TYPE 2) and LISTING FILE (TYPE 5)

A source file is a user-generated file consisting of a series of ASCII records. Each ASCII source record is of variable length and may contain up to 128 sixteen-bit words. Each 16-bit word is made up of two 8-bit ASCII bytes.

NOTE

The 64000 system editor program will only read or write 240 characters (120 words). In editing a file with lines longer than 240 characters, those characters in words 120 through 127 will be truncated.

A listing file is generated whenever the user issues a "listfile <FILE>" or "list <something> to <FILE>" command. Its format is exactly the same as that of a source file. For a pictorial representation of the source and listing file formats, see Figures 4-1 and 4-2.

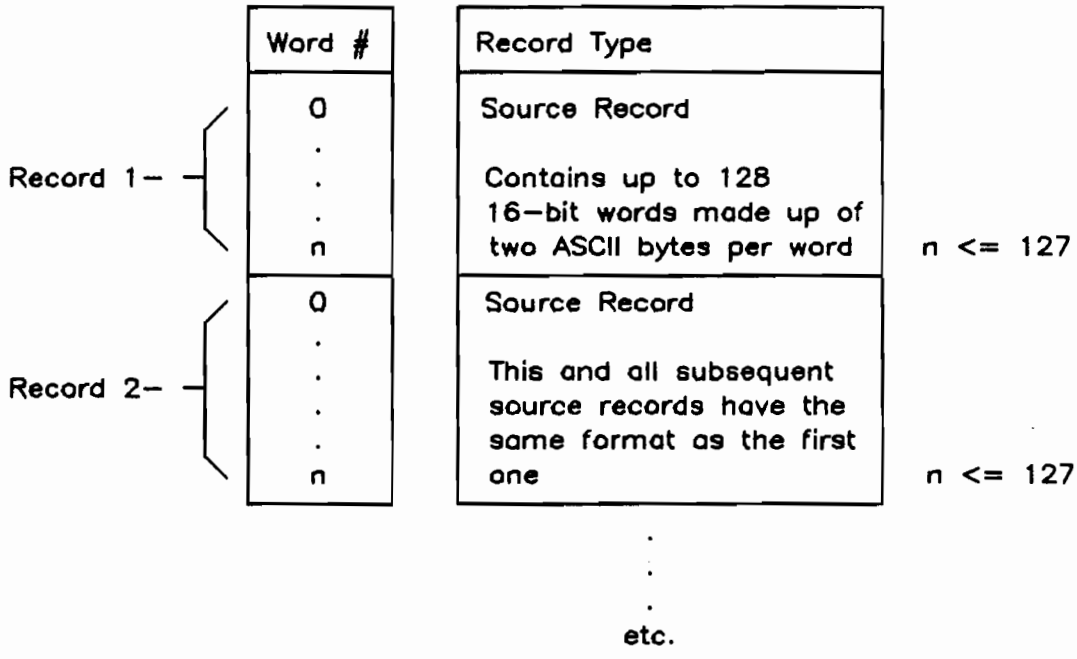

Figure 4-1. Source and Listing Files - Overall Structure

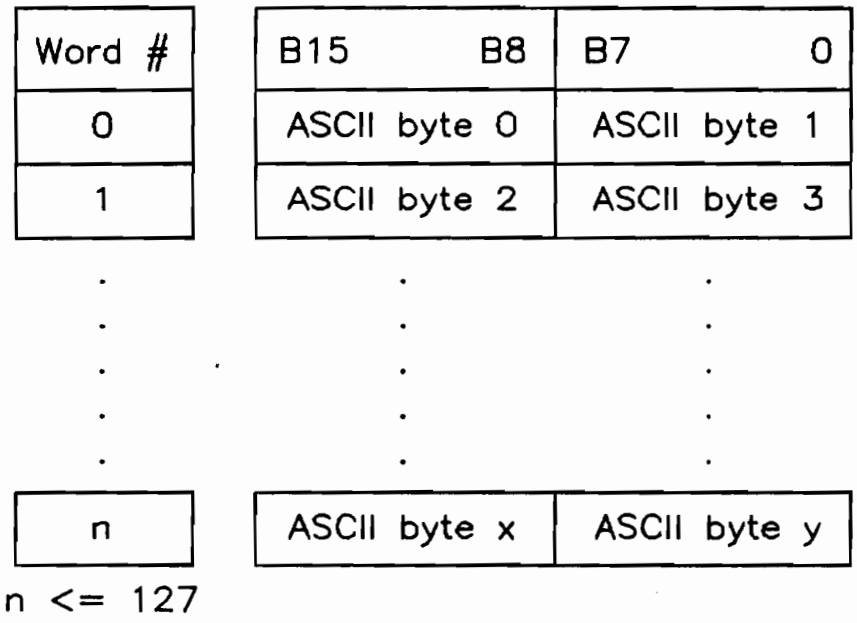

Figure 4-2 Source and Listing Files - Source Record Format

WORD n - If the last byte of word n is not used, it should be padded with an ASCII blank (20H) such that all records are word-aligned.

SPECIAL SOURCE OR LISTING FILES

Ndestfile:HP

When ending an edit session the Editor creates an intermediate file named Ndestifile:HP, where N is the System Bus address of the station in use. This file is of type :souce or :listing, depending upon the type of file being edited.

CAUTION

The occurrence of a power failure or of SHIFT-RESET being pressed before the edit session is completely ended will result in that intermediate file remaining on the disc. Depending upon when the power failure or SHIFT RESET occurs, it is possible that the original file and the current file which is ending will BOTH be lost. There is no way to recover either file.

To eliminate an Ndestfile:HP, edit a file which already exists from the station at System Bus Address N and end the edit session.

Chapter 5
RELOCATABLE FILE FORMAT

RELOCATABLE FILE (FILE TYPE 3)

Relocatable files are produced by the HP 64000 system assemblers and compilers, and are the input used by the HP 64000 system linker. As such, they must provide to the linker all the information it needs to produce absolute code. For each relocatable file the first record must be a Name record. This is followed by a variable number of Global Symbol Records, External Symbol Records, and Double Records in any order. The last record must be an End record.

Relocatable files are also the input and output of the library command. The command library A to B appends the relocatable file A to the relocatable file B. This creates a file which consists of one or more relocatable file overall structures. All future references to a relocatable file in this document are referring to a file created by assembling or compiling a source file.

In the discussion of the relocatable file format it is important to keep in mind the relationship between relocatable file names and source file names. The name of the source file that is assembled or compiled is permanently saved in the Name Record of the Relocatable File. Since files may be renamed, it is possible to have a relocatable file with a name that is different from the source file name which was saved in the Relocatable Name Record. In the case of a library of relocatables there is only one relocatable file name, but multiple source file names are associated with the library.

For this document, the term Source Name is used to refer to the source file which was assembled or compiled to create an individual relocatable file or one of the individual relocatable files in a library.

For a pictorial representation of the relocatable file format, see Figures 5-1 through 5-10.

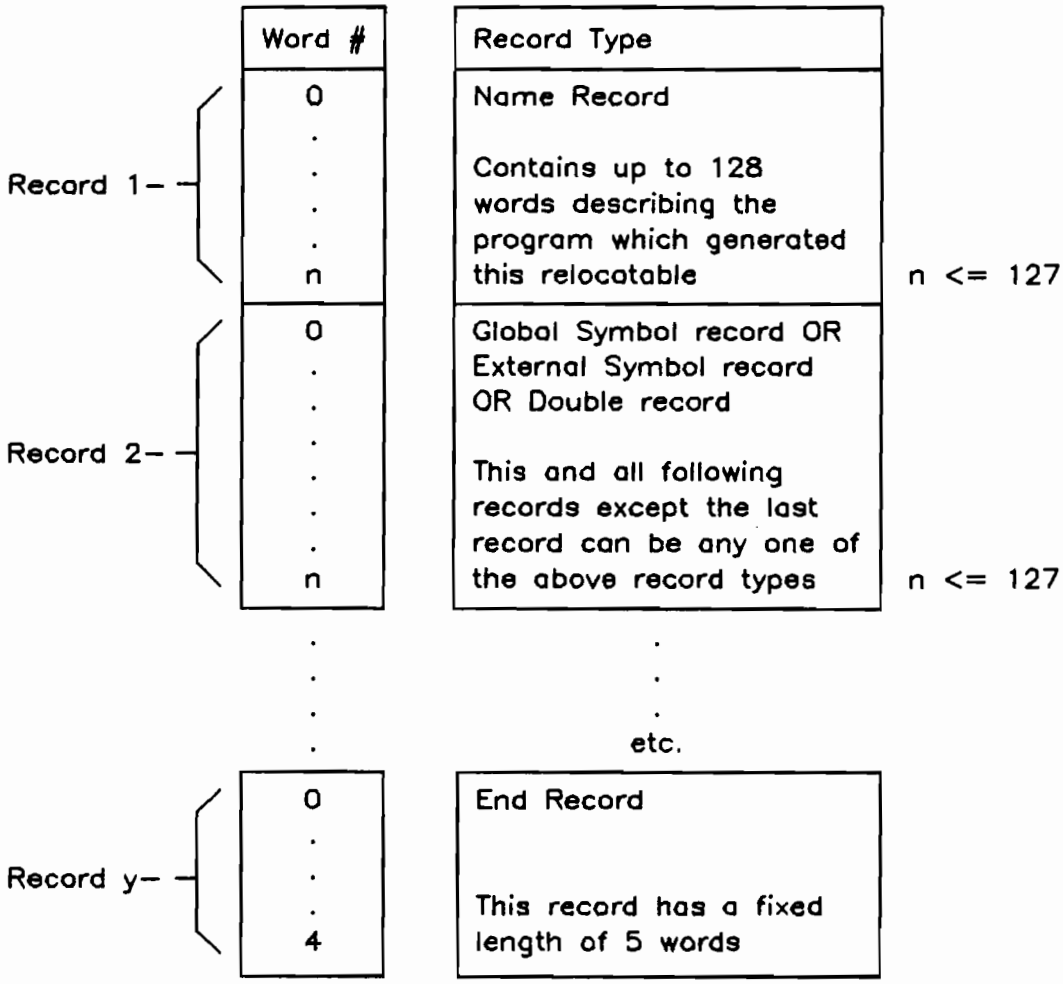

Figure 5-1. Relocatable Files - Overall Structure

A library of relocatable files consists of one or more relocatable file blocks concatenated together. (Name/record for next file immediately follows End record of previous file.)

NAME RECORD FORMAT

WORD 0 - A Record ID of 1 is specified to indicate that this is a Name Record. The Record ID is used internally to the relocatable file and should not be confused with the 64000 System file type number, which is 3.

WORD 1 through WORD 8 make up the Source File Name Description Block. This block provides the source file name which generated the relocatable file. For a complete description of this block see Appendix B.

WORD 9-10 - The PROG segment length is the number of bytes or words (processor dependent) of code which are produced by the assembler or compiler as the PROG relocatable code.

WORD 11-12 - The DATA segment length is the number of bytes or words (processor dependent) of code which are produced by the assembler or compiler as the DATA relocatable code.

WORD 13-14 - The COMN segment length is the number of bytes or words (processor dependent) of code which are produced by the assembler or compiler as the COMN relocatable code.

WORD 15 - A word containing the number of external symbols defined in this file. The maximum number of external symbols in a relocatable file is 512.

WORD 16-23 - This group of words is used to define the name of the linker table used to generate the corresponding absolute file. For a complete description of this block see Appendix C.

WORD 24 - contains the Date on which the relocatable was created. Encoded as a binary number describing the number of days since January 1, 1976. January 1, 1976 = 0.

WORD 25 - contains the time at which the relocatable was created. Encoded as a binary number describing the number of minutes since midnight. Midnight = 0.

WORD 26 through WORD 36 - contains 22 characters of comment (unused characters must be set to blank (20H). This field is given a value using the pseudo opcode NAME in the assembler and otherwise is unused (all bytes set to 20H). This field is output in the comments field of the linker listing file.

WORD 37 through n-1 contain up to 22 (all are optional) Absolute Code Segment Blocks. These blocks define the memory space used by the ORG'd code segments. These words are not used unless there are absolute code segments to be described.

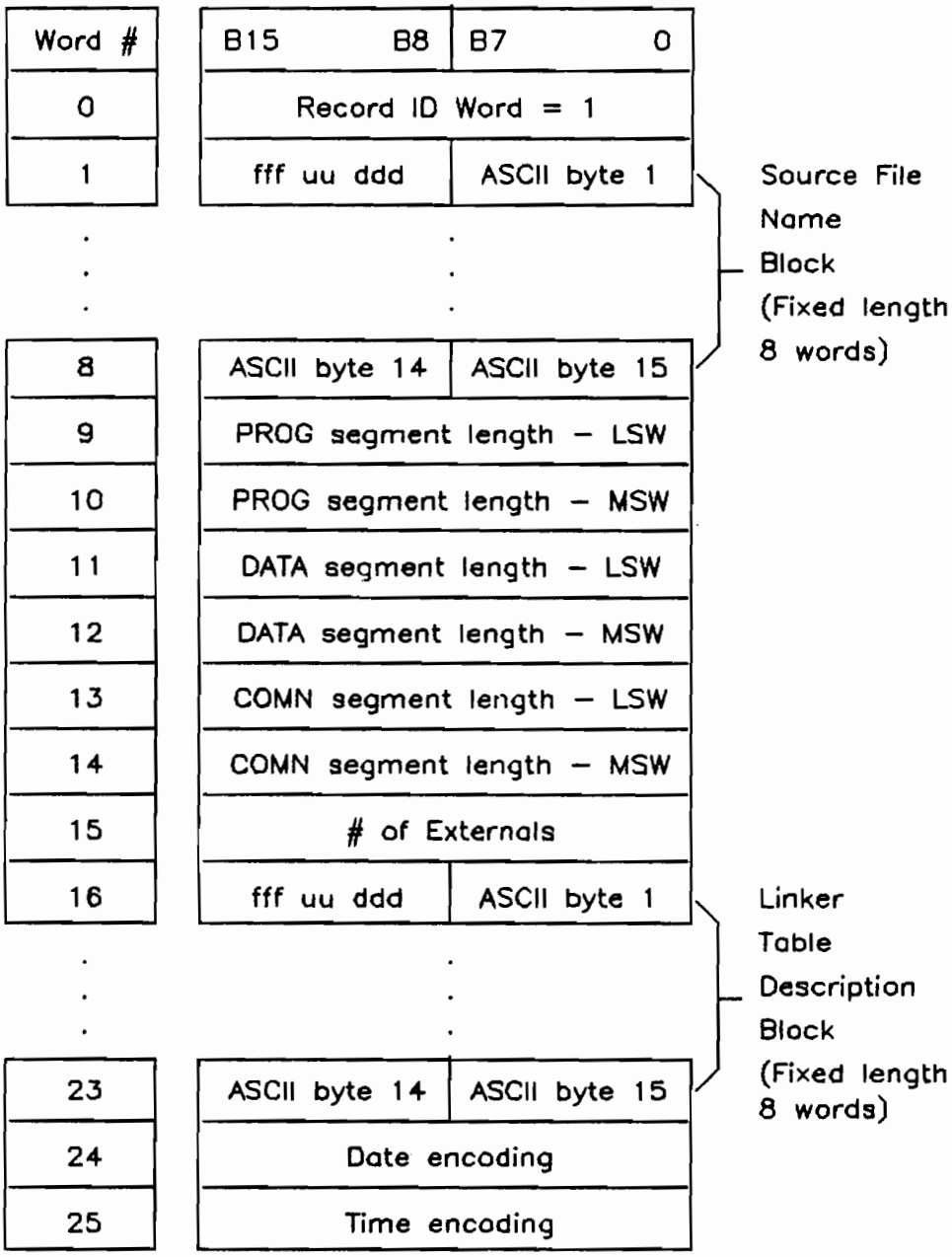

Figure 5-2. Relocatable File - Name Record Format
(Continued on next page)

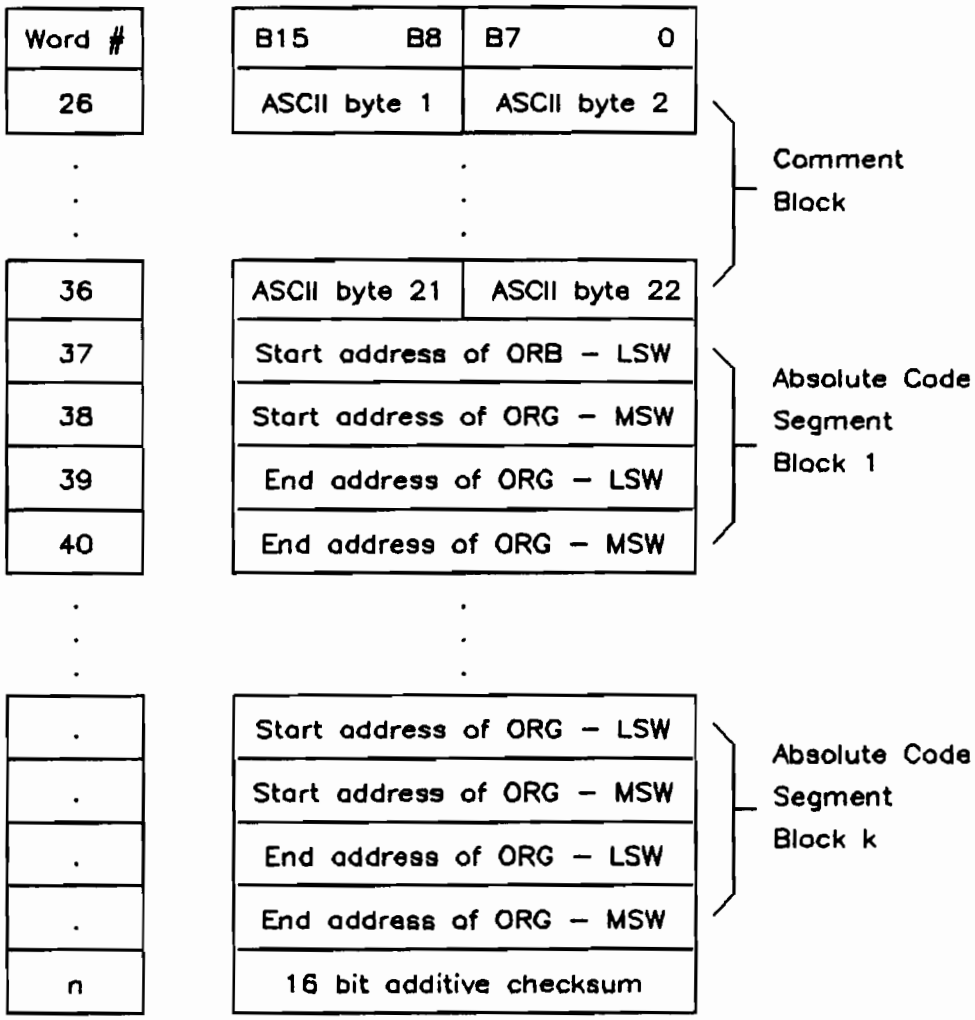

Figure 5-2. Relocatable File - Name Record Format (Cont'd)

GLOBAL RECORD FORMAT

WORD 0 - A Record ID of 2 is specified to indicate that this is a Global Symbol record. The Record ID is used internally to the relocatable file and should not be confused with the 64000 System file type number, which is 3.

WORD 1 through n-1 make up the k Global Symbol Definition Blocks. These blocks describe the global symbols generated as part of the relocatable file. Each global symbol definition block has a variable length of from 2 to 10 words.

The structure of each Global Symbol Definition Block is as follows:

WORD 0 through WORD sss (of a Global Symbol Definition Block) make up the Global Symbol Name Description Block. This block provides the name of the global symbol being defined. For a complete description of this block see Appendix D.

WORDS sss+1 and sss+2 (of a Global Symbol Definition Block) contain the value of the symbol. WORD sss+2 exists only in those processors which generate 2 words for each address (see Appendix A).

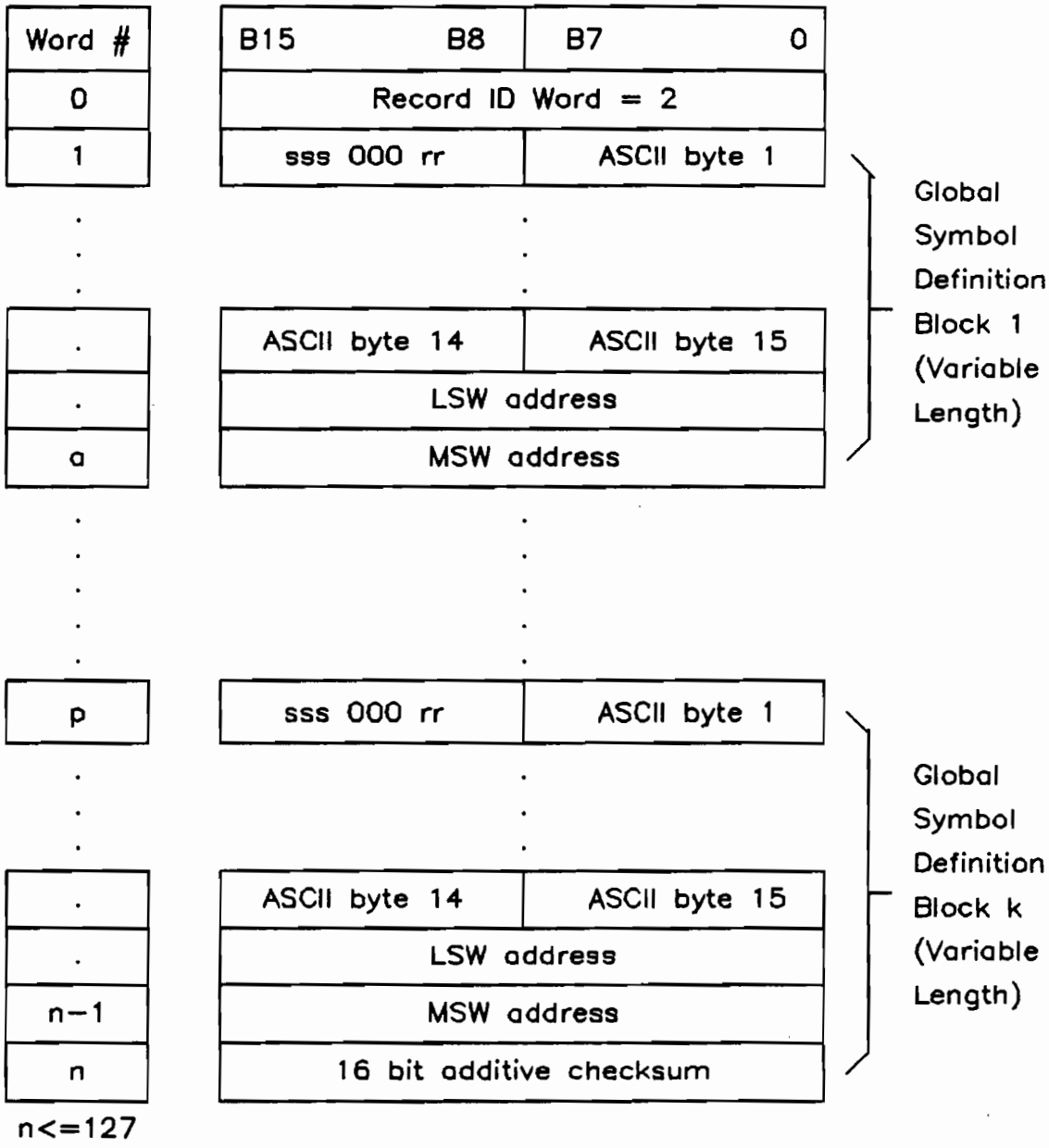

Figure 5-3. Global Symbol Record - Overall Structure

EXTERNAL RECORD FORMAT

WORD 0 - A Record ID of 4 is specified to indicate that this is an External Symbol record. The Record ID is used internally to the relocatable file and should not be confused with the 64000 System file type number, which is 3.

WORD 1 through n-1 make up the k External Symbol Definition Blocks. These blocks describe the external symbols required by the relocatable code. Each external symbol definition block has a variable length of from 2 to 9 words.

The structure of each External Symbol Definition Block is:

WORD 0 through WORD sss (of an External Symbol Definition Block) make up the External Symbol Name Description Block. This block provides the name of the global symbol being defined. For a complete description of this block see Appendix D.

WORD sss+1 - contains the external ID number of the symbol. The up to 512 symbols are numbered from 0 to 511. The external ID number is used when this external symbol is referenced in a double record in the same relocatable file.

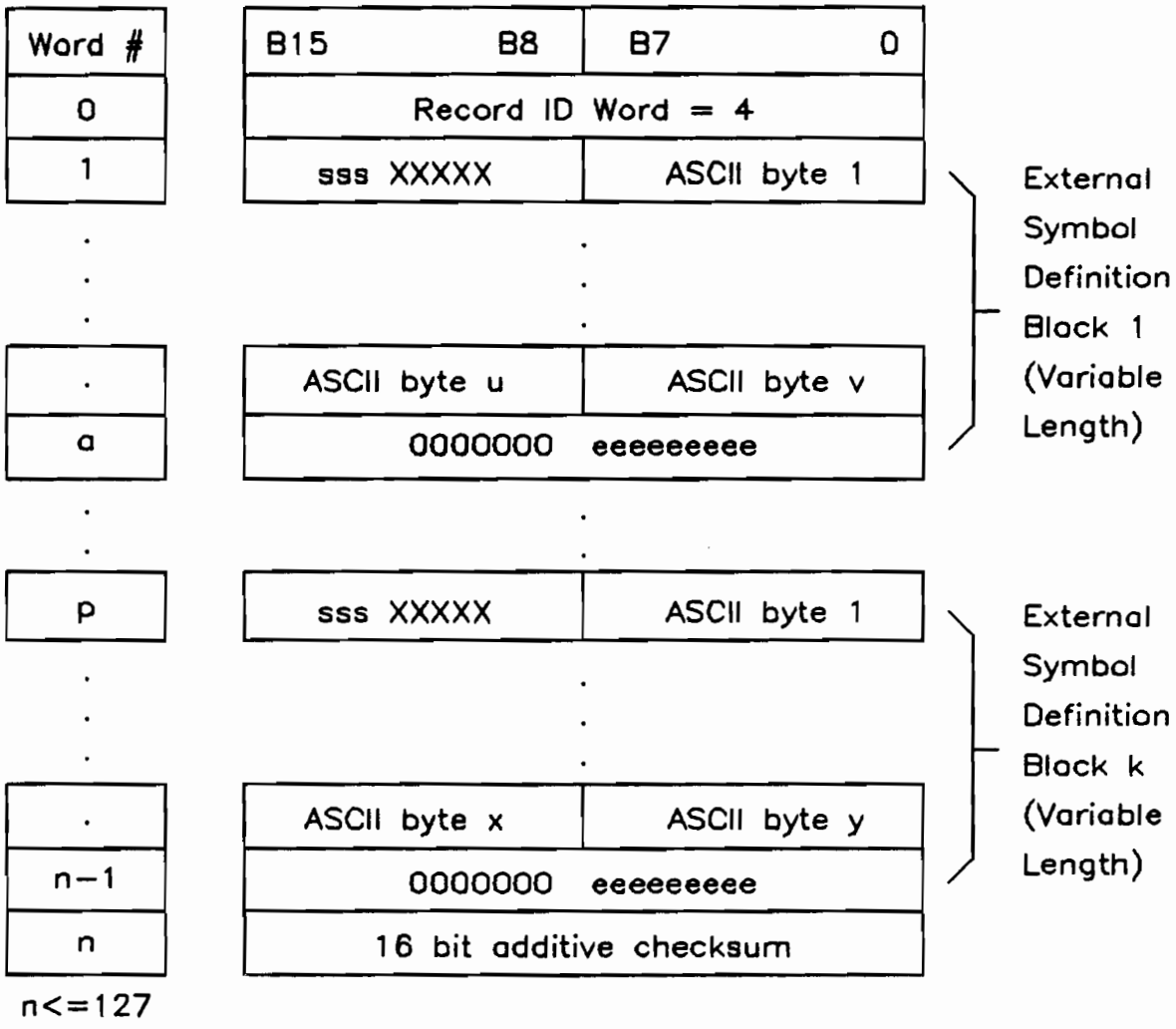

Figure 5-4. External Symbol Record - Overall Structure

DOUBLE RECORD FORMAT

WORD 0 - A Record ID of 3 is specified to indicate that this is a Double record. This record is called a double record because it holds two kinds of information, 1) relocation and 2) external references. The Record ID is used internally to the relocatable file and should not be confused with the 64000 System file type number, which is also 3.

WORD 1 (and WORD 2 only for processors using 2 words for each address; see Appendix A) contains relative address of first byte or word (processor dependent) defined in this record.

Word 2 (3) - rr indicates with respect to which relocation counter the relative address described in WORD 1 is being relocated.

00 = absolute (no relocation)
01 = PROG
10 = DATA
11 = COMN

WORD 3(4) through n-1 contain the k T Parameter Blocks. These blocks are variable length from 2 to 41 words.

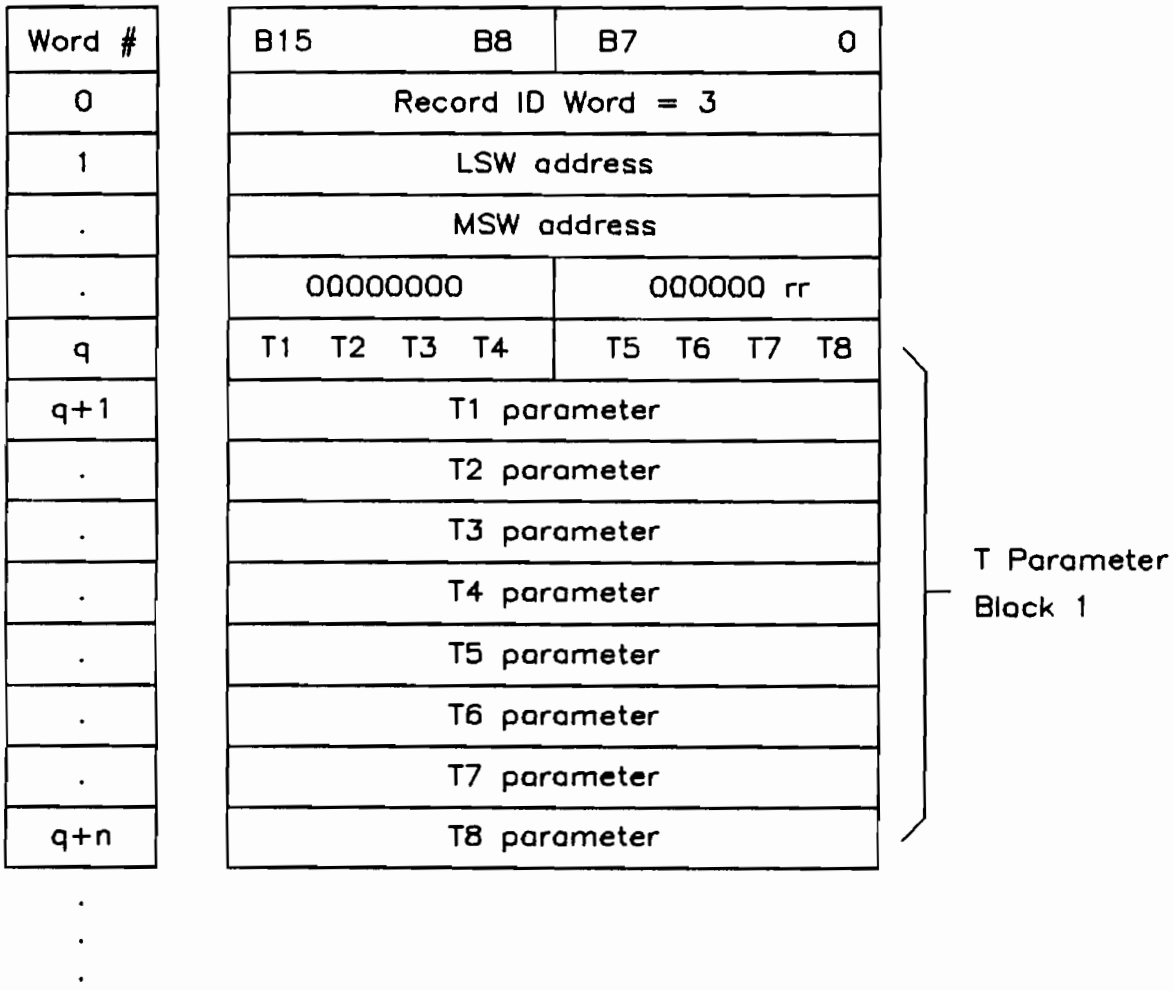

Figure 5-5. Double Record - Overall Structure
(Continued on next page)

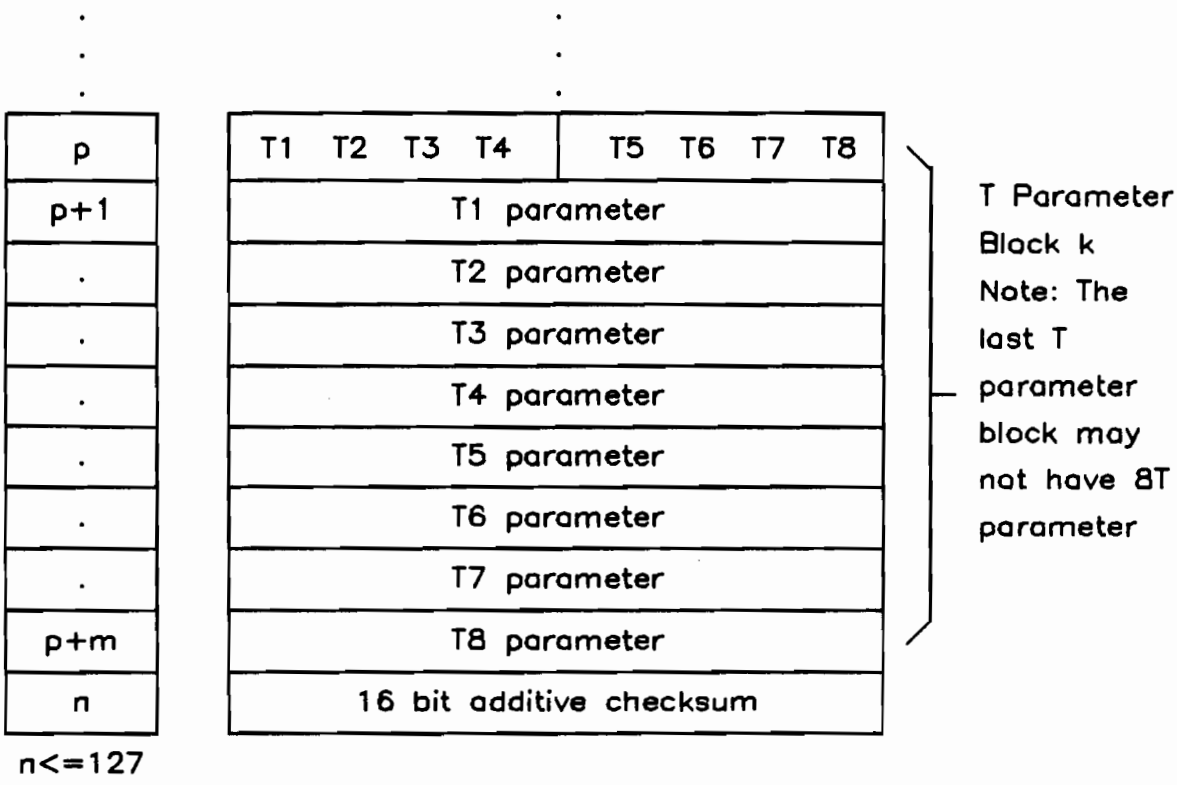

Figure 5-5. Double Record - Overall Structure (Cont'd)

The structure of each T Parameter Block is:

WORD 0 - contains 8 two bit fields T1 through T8 describing the action to be performed by the linker on the T parameters which follow. All T Parameter Blocks in a record must be complete (use all of T1 through T8) except the last data description block.

WORD 1 through the end of this T Parameter Block contain the T Parameters as described below.

T-PARAMETERS

If Ti = 00, the required T-parameter is a single word of which the low order byte contains a byte of code which will be written as is to the absolute file by the linker. The high order byte is not used.

Word #
1

B15 B8	B7 0
00000000	HEX byte

Figure 5-6. Ti = 00 Format

If Ti = 01, the required T-parameter is a single word in which both bytes of data will be written as is by the linker to the absolute file. The byte in B15 through B8 is the first byte loaded, B7 through B0 is the second byte loaded.

Word #	B15 B8	B7 0
1	HEX byte	HEX byte

Figure 5-7. Ti = 01 Format

If Ti = 10, the required T-parameter is from 2 to 5 words of data organized as follows:

Word #	B15 B8	B7 0
1	0000000 r	r fffffff
2	LSW address	
.	MSW address	
.	instruction skeleton	
.	instruction skeleton	

Figure 5-8. Ti = 10 Format

WORD 1 - The contents of Word 1 is broken down as follows:

rr - bits 9,8 - contain a code describing which relocation counter to use in relocating this symbol.

00 = absolute (no relocation)
01 = PROG
10 = DATA
11 = COMN

fffffff -- a 7 bit field used to indicate to the linker the format number for processor dependent instructions. See Appendix A for description of processor dependent format numbers.

WORD 2 (and WORD 3 for processors using 2 words for each address) contains the address to be relocated (see Appendix A).

WORD 3 (or WORDs 4 & 5 as indicated in Appendix A) contains an instruction skeleton. A given format number for a specific processor requires having or not having a 16 bit or 32 bit (depending on processor) instruction skeleton. See Appendix A for details.

If Ti = 11, the required T-parameter is from 2 to 5 words of data organized as shown in Figure 5-9.

Word #	B15 B8	B7 0
1	eeeeeeee	e fffffff
2	signed displacement	
.	signed displacement	
.	instruction skeleton	
.	instruction skeleton	

Figure 5-9. Ti = 11 Format

WORD 1 - The contents of Word 1 is broken down as follows: eeeeeeeee -- a nine bit field which holds the external identification number of a symbol in an external symbol record. The external symbol record defining this symbol must be physically located before any data definition record referencing it.

fffffff -- a 7 bit field used to indicate to the linker the format for processor dependent instructions. See Appendix A for description of processor dependent format numbers.

WORD 2 (and WORD 3 for processors using 2 words for each address) contains the address to be relocated (refer to Appendix A).

WORD 3 (or WORDs 4 & 5 for processors indicated in Appendix A) contains an instruction skeleton. A given format number for a specific processor requires having or not having a 16 bit or 32 bit (depending on processor) instruction skeleton. See Appendix A for details.

END RECORD FORMAT

WORD 0 - A Record ID of 5 is specified to indicate that this is an End record. The Record ID is used internally to the relocatable file and should not be confused with the 64000 System file type number, which is 3.

WORD 1 - bits 2,1, and 0 (rrr) define with respect to which relocation counter the transfer address of the program is defined.

000 = absolute (no relocation)
001 = PROG
010 = DATA
011 = COMN
100 = no transfer address for this module

WORD 2 and WORD 3 contain the transfer address for the program relative to the relocation counter specified in word 1. Note the MSW is not optional but is set to 0 for processors requiring only 1 word of address (see Appendix A).

WORD 4 - 16 bit additive checksum of this record.

Word #	B15 B8	B7 0
0	Record ID Word = 5	
1	00000000	00000 rrr
2	LSW address	
3	MSW address	
4	16 bit additive checksum	

Figure 5-10. End Record - Overall Structure

Chapter 6

ABSOLUTE FILE FORMAT

ABSOLUTE FILE (FILE TYPE 4)

An absolute file is a binary object file generated by the linker, or by the emulation system when a "store memory to <FILE>" command is executed. It consists of a variable number of records, the first of which, called the Processor Information Record, provides information about the microprocessor for which the file is intended. All subsequent records, called Data Records, are of variable length up to 128 sixteen-bit words and contain header information about the record along with data words.

For a pictorial representation of the absolute file format, see Figures 6-1, 6-2, and 6-3.

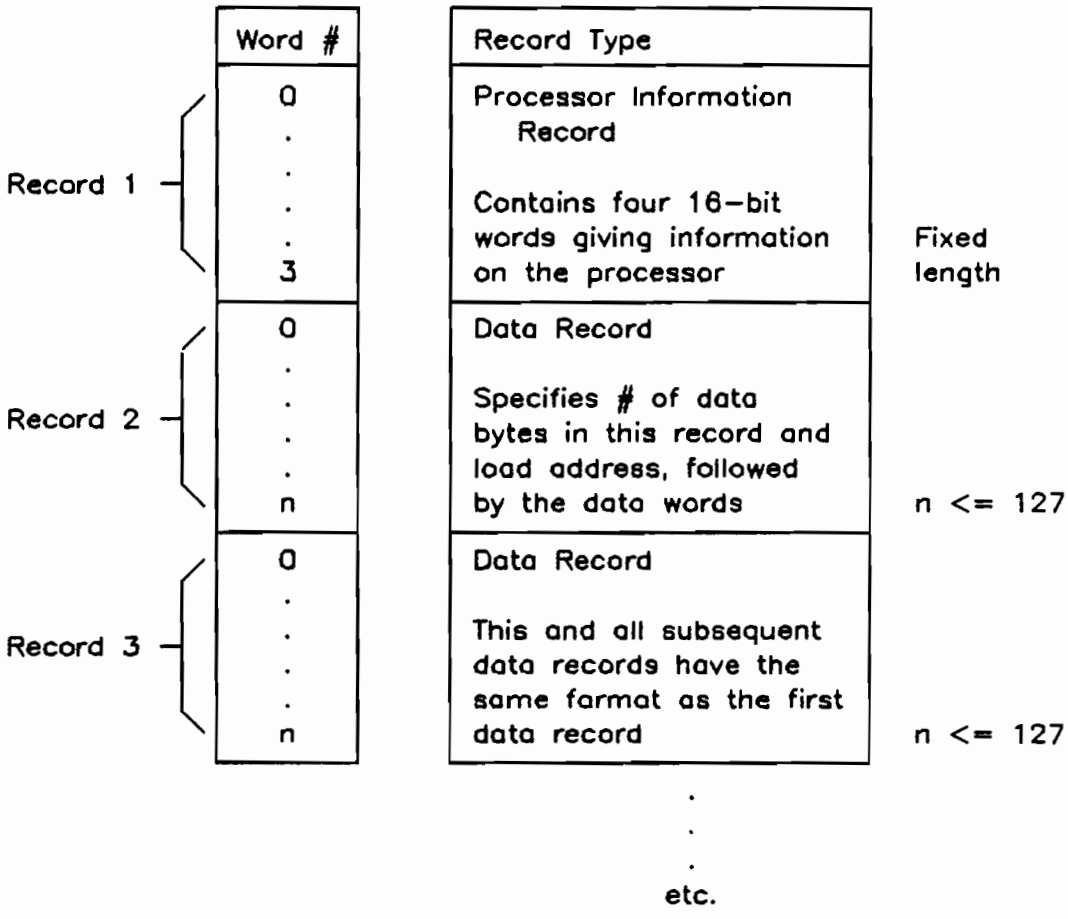

Figure 6-1. 64000 System Absolute Files - Overall Structure

PROCESSOR INFORMATION RECORD

WORD 0 - Data Bus Width is the width of processor data bus (i.e. 8,16, etc. See Appendix A).

WORD 1 - Data Width Base is the minimum addressable entity (group of bits) used by the microprocessor. Usually this will be 8, but not always (see Appendix A).

WORD 2-3 - Transfer Address is the value to be loaded into the microprocessor Program Counter by the emulator. It is generated only by the linker and is set to zero when an absolute file is created by storing memory from the emulator. The Most Significant Word of the Transfer Address should be set to zero if it is not needed by the processor (see Appendix A).

Word #	B15 B8	B7 0
0	Data Bus Width	
1	Data Width Base	
2	Transfer Address LS Word	
3	Transfer Address MS Word	

Fixed Length = 4 words

Figure 6-2. Absolute File -Processor Information Record Format

DATA RECORD

WORD 0 - # of Data Bytes in this record expressed in binary.

WORD 1-2 - Load Address is the binary address in the microprocessor memory space into which the first data byte (from WORD 3) should be loaded. Subsequent data from this record is loaded into the following microprocessor memory space. The Most Significant Word of the Load Address should be set to zero if it is not needed by the processor (see Appendix A).

WORD 3 through n - Data Words are the binary representation of the absolute data to be loaded into microprocessor memory space.

WORD n - If the last byte of word n is not used, it should be set to 0.

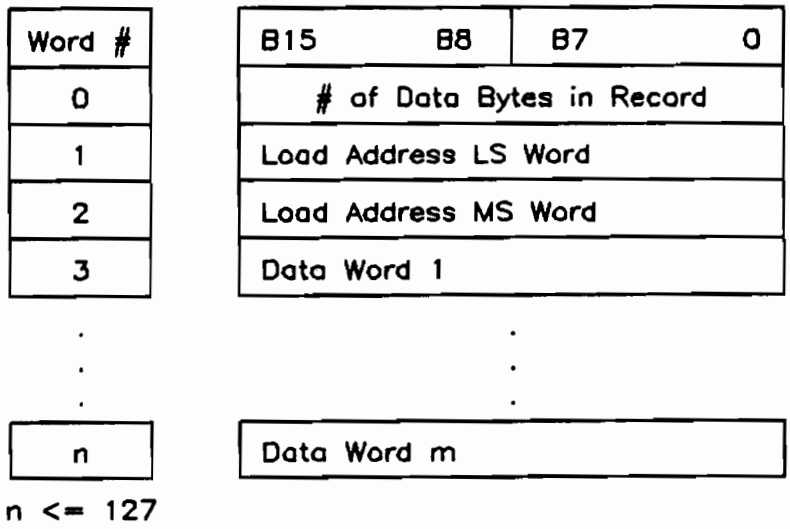

Figure 6-3. Absolute File - Data Record Format

Chapter 7

EMULATION COMMAND FILE FORMAT

EMULATION COMMAND FILE (FILE TYPE 6)

Emulation command files contain information used to configure an emulator. These files are very complicated, and the user will find it easier to change, create, or decode an :emul_com file by using measurement system. For this reason, the details of the structure of an emulation command file will not be described.

SPECIAL EMUL_COM FILES

EcnfgXY:HP

If a name is not given for the emul_com file when you are configuring the emulator, an emul_com file will be created for you with the name EcnfgXY:HP where X is the System Bus address of your station, and Y is the card cage number of the slot which has the emulator control card in it.

Chapter 8

LINKER COMMAND FILE FORMAT

LINKER COMMAND FILE (FILE TYPE 7)

Linker command files contain information used to tell the linker how to create an absolute file with the same name as the :link_com file (which files to link together, where to relocate them, etc.). The format used to store this information is a memory dump which has a very complex format. The user will find it easier to modify, create, or decode a link command file by using the linker. For this reason, the details of the structure of a linker command file will not be described.

Chapter 9
TRACE FILE FORMAT

TRACE FILE (FILE TYPE 8)

Trace files are created and used by the HP64000 system software performance analyzer, state analyzer, timing analyzer, and emulation with analysis systems. They are used to store configuration information and/or measurement data. Each module has defined its trace files differently. The basic structure of a trace file is an Identification Record followed by a variable number of additional records (see Figures 9-1 and 9-2).

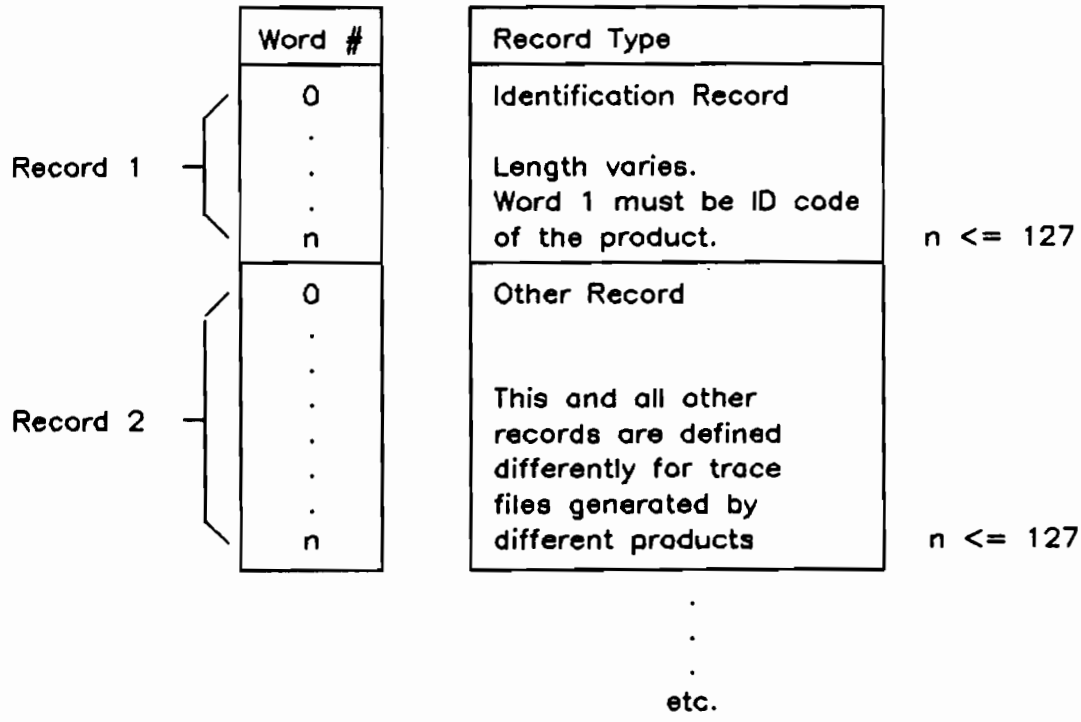

Figure 9-1. Trace Files - Overall Structure

WORD 0 - The 4 HEX digit ID code describing the board which generated the trace file.

0101 -- HP64310 Software Performance Analyzer
1001 -- HP64601 Timing Analyzer
1100 -- HP64620 State Analyzer
0102 -- HP64300/HP64302 internal analysis running with emulation
0000 -- Measurement System

WORD 2 through n - provides other information needed by the product. See "Product Descriptions" for more complete information.

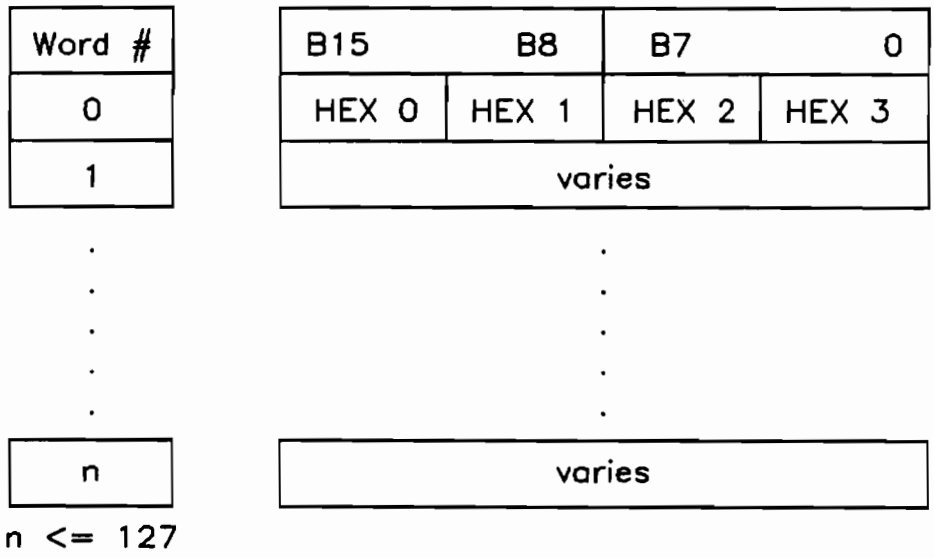

Figure 9-2. Identification Record - Overall Structure

PRODUCT DESCRIPTIONS

SOFTWARE PERFORMANCE ANALYZER

WORD 0 - A board ID of 0101H is specified to indicate that this trace file was created by a Software Performance Analyzer (see Figures 9-3 and 9-4).

WORD 1 - file lock provides the user with a way to protect his files from being over-written. A zero in this location means the file may be read or written. A non-zero value means the file can only be read.

Software Performance Analyzer stores only configuration information in the trace file. There is no data stored. In most cases, it is simplest for the user to modify his configuration by using measurement systems and modifying a previous configuration file. However, if a user is doing remote development, it may be useful to reconfigure the EVENT_ARRAY remotely. For such a purpose, only that part of the configuration file is described here. IMPORTANT: To incorrectly change this trace file should be attempted ONLY by a highly experienced SPA user. It is very important that the user does not modify any part of the trace file other than the event array. Note also that NO guarantees are made concerning operation of SPA if invalid values are written into the event array.

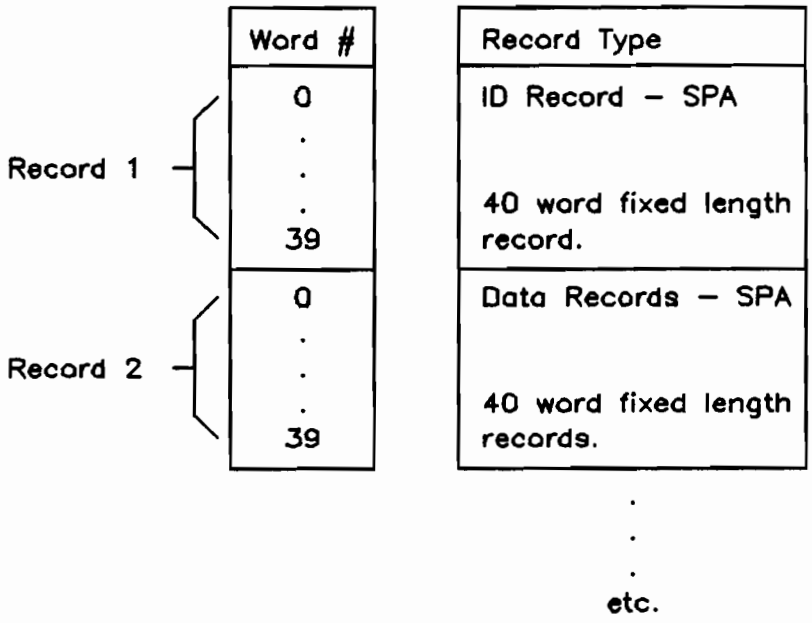

Figure 9-3. Trace File Format - Software Performance Analyzer

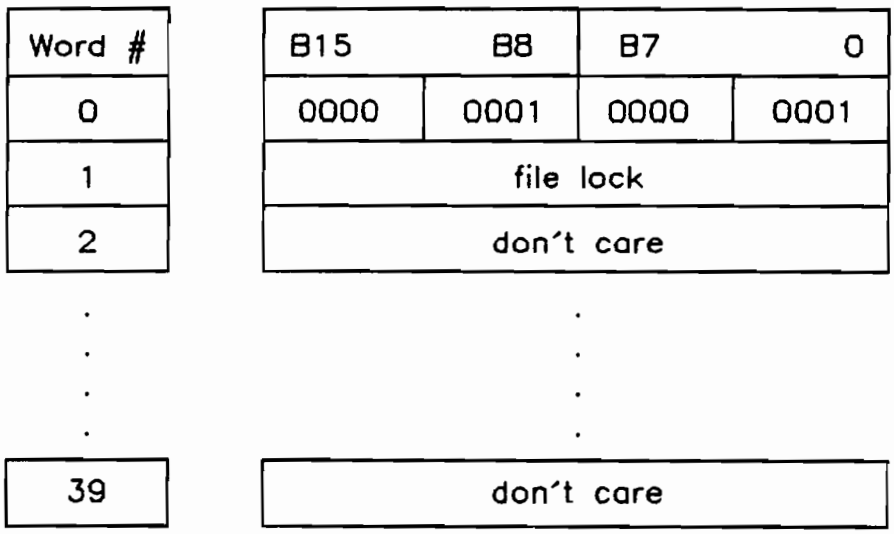

Figure 9-4. Identification Record - Software Performance Analyzer

The EVENT_ARRAY is stored in records 1 through 29, and the first 28 words of record 30. There are 99 EVENT_ARRAY entries, each having a length 12 words. Since all records have a fixed length of 40 words, the array entries will cross record boundaries. The EVENT_ARRAY has a format as shown in Figure 9-5. See Figure 9-6 for a detailed description of the Data record format.

WORD 0 through WORD 7 - Provides the 15 character title for the event specified as well as some information indicating what the event was. Word 0 through word 6 contain the first 14 characters packed two characters per word. The fifteenth character is stored in the upper byte of word 7. All unused characters must be set to blank (20H). The lower byte of word 7 is decoded as follows:

Bits 7,6,5,4 - this four-bit field defines the number of bytes in the title field that contain title information.

Bit 3 - This bit defines whether the event specified is a time event. A one indicates a time event.

Bit 2 - This bit defines whether the event specified is an address event. A one indicated an address event.

Bit 1 - This bit indicates whether the title field contains a title for this event. A one indicates that the title exists.

Bit 0 - This bit indicates whether this is a valid entry in the event array. A one indicates it is a valid event.

Word #	B15 B8	B7 0
0	ASCII 0	ASCII 1
1	ASCII 2	ASCII 3
2	ASCII 4	ASCII 5
3	ASCII 6	ASCII 7
4	ASCII 8	ASCII 9
5	ASCII 10	ASCII 11
6	ASCII 12	ASCII 13
7	ASCII 14	IIII t a n v
8	LSW lower bound	
9	MSW lower bound	
10	LSW upper bound	
11	MSW upper bound	

Figure 9-5. Event Array Entry

Word #	B15 B8	B7 0
0	data 0	
1	data 1	
2	data 2	
.	.	
.	.	
.	.	
.	.	
39	data 39	

Figure 9-6. Data Record Format

WORD 8 through 11 - lower bound/upper bound: For an address event, these fields are defined to contain a 32-bit absolute address. At present only the lower 24 bits are used. For a time event, the boundary field is defined as shown in Figure 9-7.

Word #
0
1

B15 B8	B7 0
16-bit decimal number	
exp	digits

Figure 9-7. Time Event Boundary Definition

To encode the field:

1) Time = (x.yza X 10 exp w) microseconds. This number is in scientific notation (w is a multiple of 3).

The binary equivalent of decimal xyza (no decimal point) is stored in the first word.

The number of digits to the right of the decimal point is stored in the lower byte of the second word.

The exponent (w) is stored in the upper byte of the second word.

For example: If time is 65.74 X 10 exp 3 microseconds, the time event would be stored as shown in Figure 9-8.

Word #
0
1

B15 B8	B7 0
0001 1001	1010 1110
00000011	00000010

Figure 9-8. Time Event Boundary Example

TIMING ANALYZER

WORD 0 - A board ID of 1001H is specified to indicate that this trace file was created by a Timing Analyzer (see Figures 9-9 and 9-10).

WORD 1 - ID code of data structure provides a version number. It is set up as an HP date code and converted to HEX. For example if the ID code was 08AA, that would be 2218 decimal or week 18 of 1982.

WORD 3 and WORD 4 - provide a 32-bit checksum of the rest of the records of the file. The checksum is computed by adding the 16-bit words together. To get the LSW of the checksum, and every time there is an overflow incrementing the MSW of the checksum.

WORD 5 - file lock provides the user with a way to protect his files from being over-written. A zero in this location means the file may be read or written. A non-zero value means the file can only be read.

The Timing Analyzer currently stores only configuration information in the trace file. There is no data stored. If the user wishes to modify his configuration he should do so by entering measurement systems and modifying a previous configuration file. IMPORTANT: Be very careful if you modify any part of a trace file and then try to use it again to configure the timing analyzer;to do otherwise may cause unexpected results.

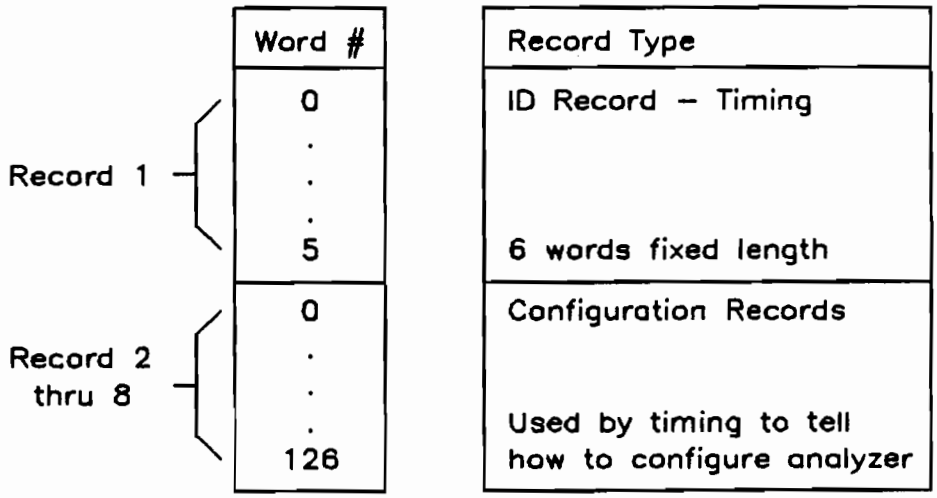

Figure 9-9. Trace File Format - Timing Analyzer

Word #	B15 B8	B7 0
0	0001 0000	0000 0001
1	ID code of data structure	
2	0000 0000	0000 0000
3	LSW checksum of entire file	
4	MSW checksum of entire file	
5	file lock	

Figure 9-10. Identification Record - Timing Analyzer

STATE/SOFTWARE ANALYZER

WORD 0 - A board ID of 1100H is specified to indicate that this trace file was created by the Model 64620 State Analyzer (see Figures 9-11 and 9-12).

WORD 1 -The identification code provides a version number for the data structure.

WORD 2 - Modification number provides a version number. It is a binary encoded decimal number representing an HP date code. For example, if the modification number was 2218 , that would represent week 18 of 1982.

WORD 3 through 10 - Provides the State Analyzer software with the name of the disassembler table it is to use. This should be in the form of an HP64000 system file name (nnnnnnnnn:uuuuuu). nnnnnnnnn is the file name, up to nine characters. uuuuuu is the userid of the file, up to 6 characters. The file name should be packed with the extra bytes set to blank (20H). The colon (:) must be included.

WORD 11 and WORD 12 - provide a 32-bit checksum of the rest of the records of the file. The checksum is computed by adding the 16-bit words together. to get the LSW of the checksum, and every time there is an overflow incrementing the MSW of the checksum.

WORD 13 - file lock provides the user with a way to protect his files from being overwritten. A zero in this location means the file may be read or written. A non-zero value means the file can only be read.

The State Analyzer stores configuration information in the trace file. However, the information is stored as an image of memory, and as such it would not be feasible to try to extract or modify these parts of the trace files.

NOTE: Modifying any part of the configuration portion of the trace file trace file and then using it again to configure the state analyzer may cause unexpected results.

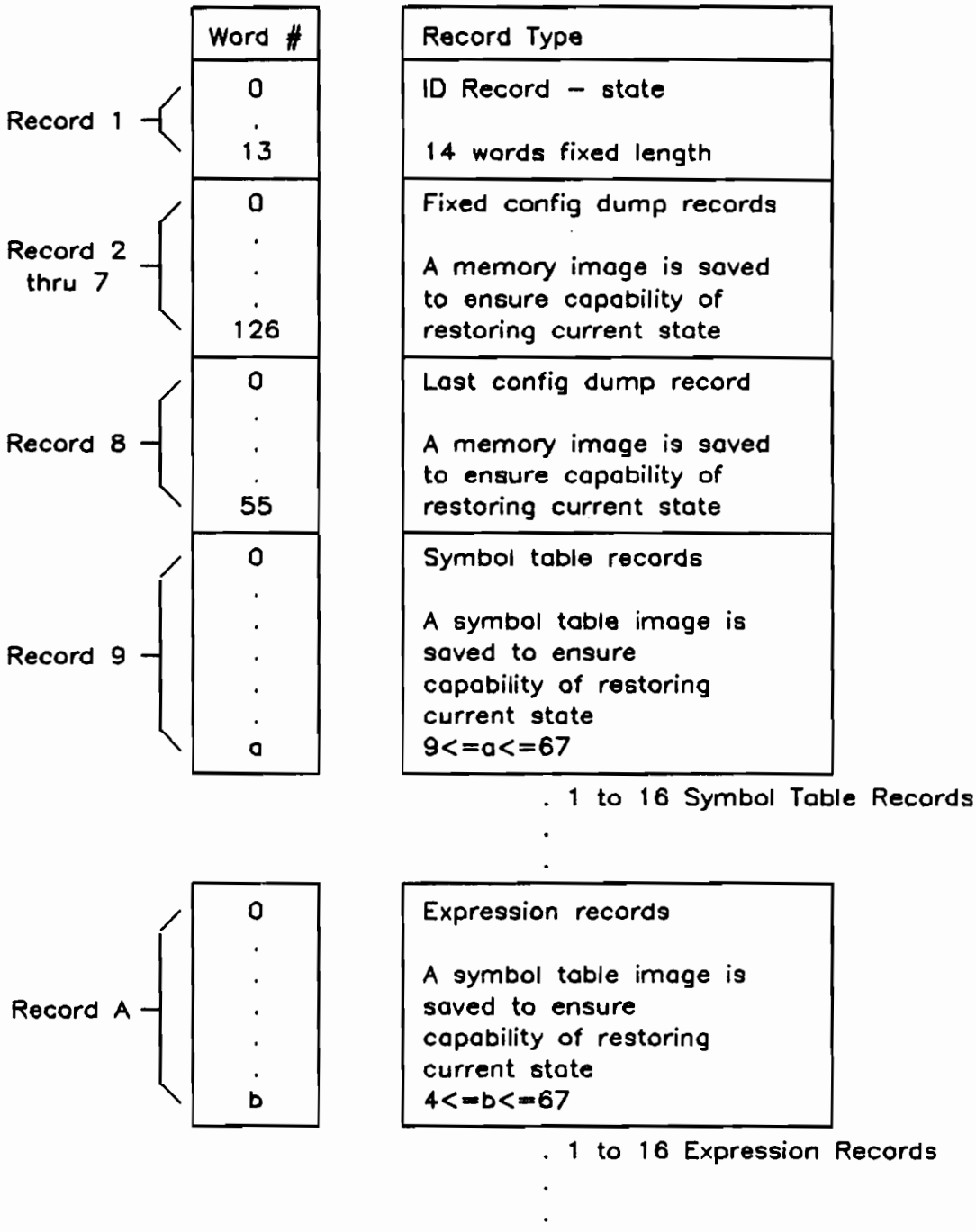

Figure 9-11. Trace File Format - State Analyzer
(Continued on next page)

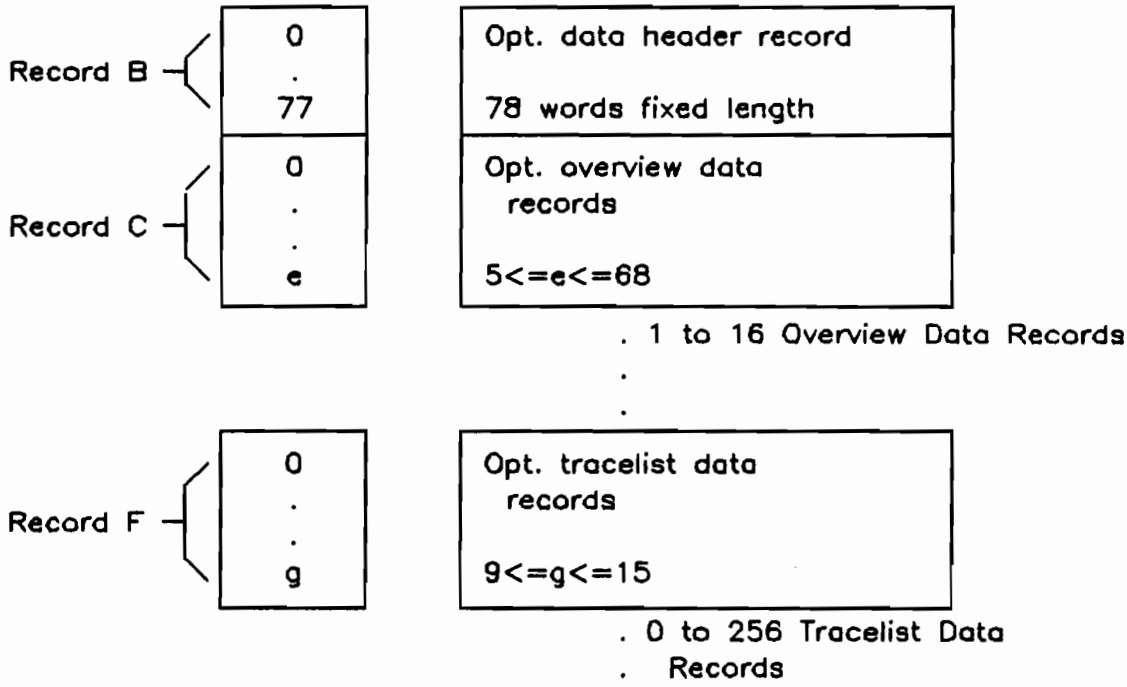

Figure 9-11. Trace File Format - State Analyzer (Cont'd)

Word #	B15	B8	B7	0
0	0001	0001	0000	0000
1	0111	0010	0000	0100
2	modification number			
3 to 10	Disassembler table file name packed in standard HP 64000 file name format			
11	LSW checksum of entire file			
12	MSW checksum of entire file			
13	file lock			

Figure 9-12. Identification Record - State Analyzer

The configuration dump records contain information used to configure the analyzer. These records are memory images and are very complicated. Users will find it easier to create or change the information using the measurement system. For this reason, only the structures of the configuration records (Configuration Dump, Symbol Table, and Expression Records) are shown (see Figures 9-13 thru 9-16). NOTE: Modifying any of therecords and then attempting to use them again to configure the state analyzer may cause unexpected results.

Word #	B15	B8	B7	0
0	0000	0000	0000	0001
1	relative position pointer			
2 to 124	123 words of memory image			
125	LSW checksum of record			
126	MSW checksum of record			

Figure 9-13. Fixed Configuration Dump Record Format

Word #	B15 B8	B7 0
0	0000 0000	0000 0001
1	relative position pointer	
2 to 53	52 words of memory image	
54	LSW checksum of record	
55	MSW checksum of record	

Figure 9-14. Last Configuration Dump Record Format

Word #	B15 B8	B7 0
0	0000 0000	0000 0002
1	relative position pointer	
2 to a−2	a−3 words of memory image	
a−1	LSW checksum of record	
a	MSW checksum of record	

Figure 9-15. Symbol Table Record Format

Word #	B15 B8	B7 0
0	0000 0000	0000 0003
1	relative position pointer	
2 to b−2	b−3 words of memory image	
b−1	LSW checksum of record	
b	MSW checksum of record	

Figure 9-16. Expression Record Format

OPTIONAL DATA HEADER RECORD FORMAT - STATE ANALYZER TRACE FILE

(See Figure 9-17.)

Word - 0 An ID code of 0100 is given to indicate that this record is the data header record.

Word - 1 Signifies whether the count is time count or state count.

Word - 2 Specifies the width of the analyzer used in the measurement

Word - 3 Indicate the mode used in the measurement.
bit 0 - sequence on
bit 1 - window 1 on
bit 2 - window 2 on

Word 4 - If the trigger condition was met the HISTORY_FLAG will be 0, otherwise it will be equal to 1 signifying that the information represents data history.

Word 5 - the number of words added in the trace list data records. This value can be from 2 thru 8.

Word 6 and Word 7 - the valid range of lines contained in the data records.

Word 8 and Word 9 - specifies the first and last line of valid trace data.

Word 10 - specifies the number of valid data cells in the overview records.

Word 11 - this value is the identification code of the preprocessor used to capture the data.

Words 12 to 75 - This information represents special information required by the inverse assembler and should not be required by a user.

Words 76 and Word 77 - provide a 32-bit checksum of the rest of the record. The checksum is computed by adding the 16-bit words together to get the LSW of the checksum and also every time there is an overflow incrementing the MSW of the checksum.

Word #	B15 B8	B7 0
0	0000 0001	0000 0000
1	state or time count flag	
2	number of bits of analysis	
3	sequencer mode	
4	HISTORY_FLAG	
5	MICRO_SIZE 2<=g<=8	
6	MIN_LINE	
7	MAX_LINE	
8	FIRSTVALID	
9	LASTVALID	
10	NUM_OF_OVERVIEW 0<=e<=4096	
11	PP_ID	
12 to 75	soft tag array for IAL	
76	LSW checksum of record	
77	MSW checksum of record	

Figure 9-17. Optional Data Header Record Format

OPTIONAL OVERVIEW DATA RECORD FORMAT - STATE ANALYZER FILE (See Figure 9-18.)

Word 0 - the ID code for the overview data record

Word 1 - starting line number of the overview data it can range from 0 to 4096.

Word 2 to Word e-2 - Overview data of from 1 to 64 words each containing 4 nibbles of overview data.

Words e-1 and e - provide a 32-bit checksum of the rest of the record. The checksum is computed by adding the 16-bit words together to get the LSW of the checksum, and everytime there is an overflow incrementing the MSW of the checksum.

Word #	B15 B8	B7 0
0	0000 0000	0000 0001
1	starting line number	
2 to e-2	captured overview data	
e-1	LSW checksum of record	
e	MSW checksum of record	

Figure 9-18. Optional Overview Data Record Format

OPTIONAL TRACELIST DATA RECORD FORMAT - STATE ANALYZER TRACE

Word 0 - ID code for tracelist data

Word 1 - the 16-bit two's complement number of the first trace list line number.

Word 2 - specifies the trace status

bit 15 - Overview Trigger flag, signifies that the trigger on overview event condition was true for this state.

bit 14 - Micro count reset flag **

bit 13 - Store block flag, indicates that there is a discontinuity in the trace data due to a store disable occurring.

bit 12 - always zero

bit 11 - Sequence occurrence flag **

bit 10 - Sequence resource 2 **

bit 9 - Sequence resource 1 **

bit 8 - Sequence resource 0 **

bit 7 - Window 2 enable flag **

bit 6 - Window 1 enable flag **

bit 5 - Sequence enable flag **

bit 4 - always zero

bit 3 to 0 - Logical sequencer state **

** These flags are used by the state analyzer to correctly format and calculate the various information in the trace. Since the data in the trace list data record have been formatted correctly, these flags are not needed by the user.

Word 3 thru Word 4 - the number of states or clock ticks (40 nsec) relative to the FIRST_VALID line of the trace data.

Word 6 thru Word g-2 - captured trace data, the least significant bits in word 6.

Words g-1 and g - provide a 32-bit checksum of the rest of the record. The checksum is computed by adding the 16-bit words together to get the LSW of the checksum, and every time there is an overflow incrementing the MSW of the checksum.

Word #	B15 B8	B7 0
0	0000 0001	0000 0002
1	trace list line number	
2	trace status	
3	LSW of count for the state	
4	MID of count for the state	
5	MSW of count for the state	
6	captured state data	
7 to g−2	captured state data	
g−1	LSW checksum of record	
g	MSW checksum of record	

Figure 9-19. Optional Tracelist Data Record Format

EMULATION WITH INTERNAL ANALYSIS

The variables in the trace specification record are all related to the commands as the user sets up emulation. This record is very complicated and its details are not needed by the user. Therefore, the details of its structure will not be presented here.

TRACE FILES (See Figures 9-20 thru 9-23.)

WORD 0 - run/step indicates the status of the emulator. 0 indicates the emulator was stepping, therefore data (if present) is not valid. Not 0 indicates the emulator was running.

WORD 1 - data stored indicates whether data was stored in this trace file. 0 indicates data is not stored in the file. Not 0 indicates that data follows.

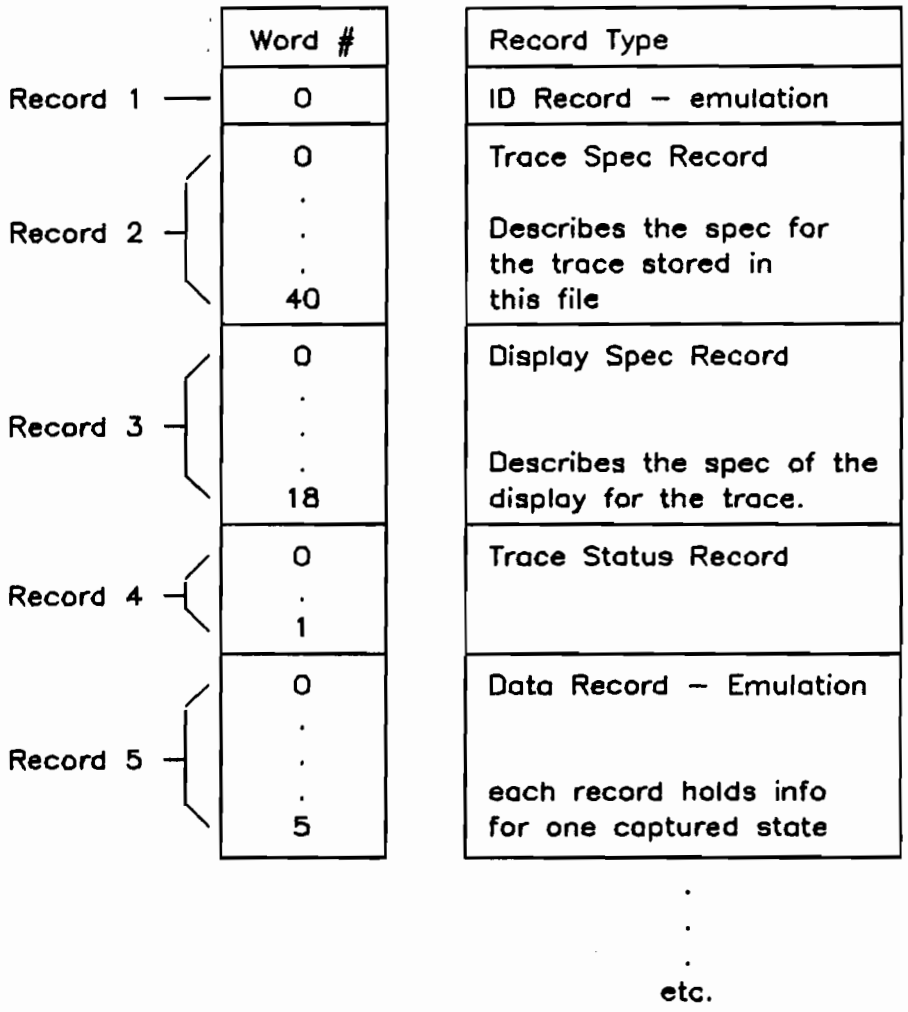

Figure 9-20. Trace File Format - Internal Analysis

Word #
0

B15	B8	B7	0
0000	0001	0000	0010

Figure 9-21. Identification Record - Internal Analysis

Word #	B15 B8	B7 0
0	Trace point	
1	Start of trace	
2 – 3	Not needed by user	
4	Count Overflow	
5 – 14	Not needed by user	
15	Prestore	
16	Totalstore	
17 – 18	Not needed by user	

Figure 9-22. Display Spec Record Format

Word #	B15 B8	B7 0
0	run/step	
1	data stored	

Figure 9-23. Trace Status Record Format

One data record is stored for each state captured (see Figure 9-24). Information stored is the address, data, and status during the state, and a 24-count value. Count starts at 0FFFFFFH at the beginning of the measurement and counts down to 0.

Word #	B15 B8	B7 0
0	LSW address	
1	MSW address	
2	data	
3	status	
4	LSW of countdown	
5	MSW of countdown	

Figure 9-24. Data Record Format - Emulation

SPECIAL TRACE FILES

When you exit emulation, a trace file will be created called EcnfgNM:HP where N is the HPIB address of the station, and M is the slot number of the emulator being used. This file is used by emulation when options continue is specified.

MEASUREMENT SYSTEM

When you exit measurement systems, a trace file will be created called meas_sysN:HP where N is the HPIB address of the station. This file is used by measurement system when options resume is specified. The contents of this file are not needed by the user.

Chapter 10
PROM FILE FORMAT

PROM FILE (FILE TYPE 9)

Prom files are no longer created by the 64000 system. The format of existing prom files is the same as the absolute file format. Refer to Chapter 6 for information about the absolute file format.

Chapter 11
DATA FILE FORMAT

DATA FILE (FILE TYPE 10)

Data files (file type = 10) are for the convenience of the user. The only :data file used by the system is spa_table:HP used by the Software Performance Analyzer. Because of its file name, this file is not accessible by the user. SPA will never use a users data file so the exact format of this data file is not needed by the user. Data files are created and accessed using Sim I/O or HOST Pascal. The system commands {copy, rename, directory, purge, and recover} will work on files of type :data.

For a pictorial representation of the data file format, see Figures 11-1 and 11-2.

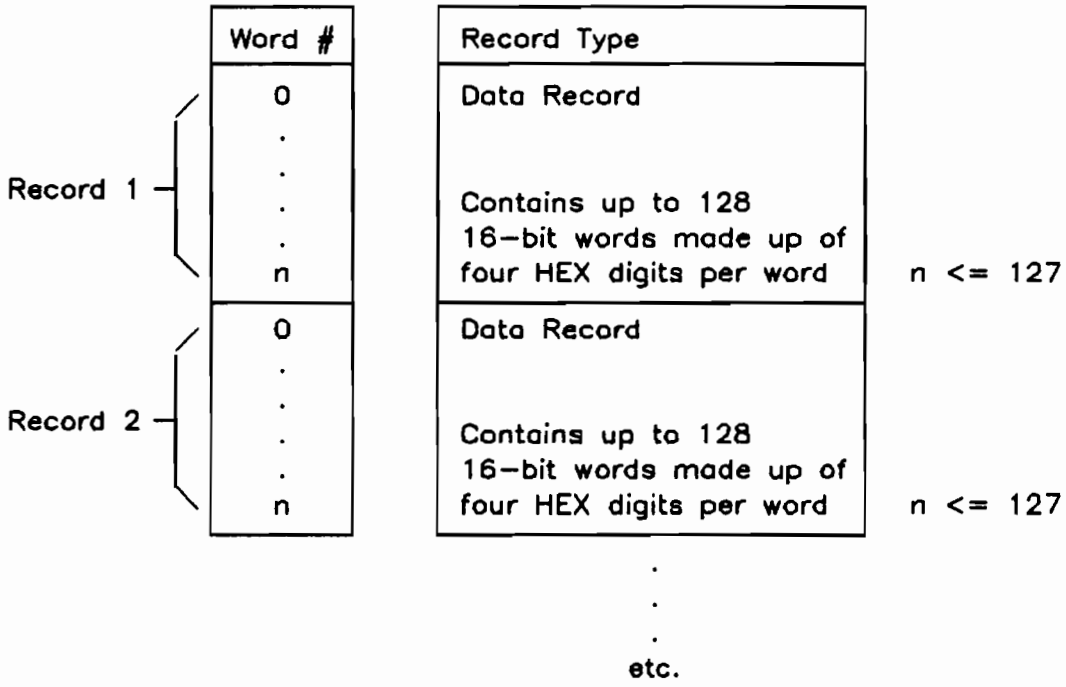

Figure 11-1. Data Files - Overall Structure

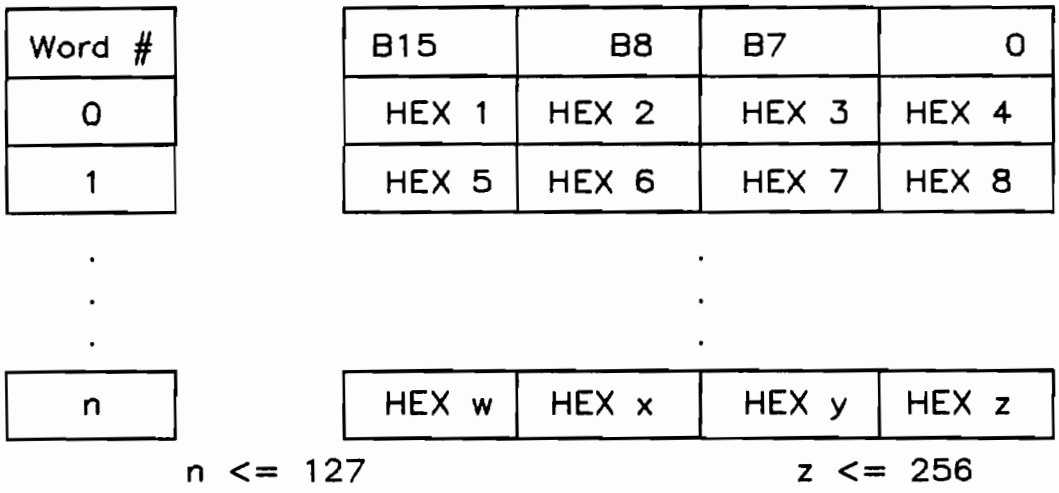

Figure 11-2. Data Record Format

Chapter 12

STATE ANALYZER DATABASE FILE FORMAT

STATE ANALYZER DATABASE FILE (FILE TYPE 11)

The state analyzer database file is created by the state analyzer. It combines the relevant information in the link_sym and asmb_sym files which apply to the absolute file being debugged. The structure of this file allows quick access to the symbols during operation of the analyzer (see Figure 12-1).

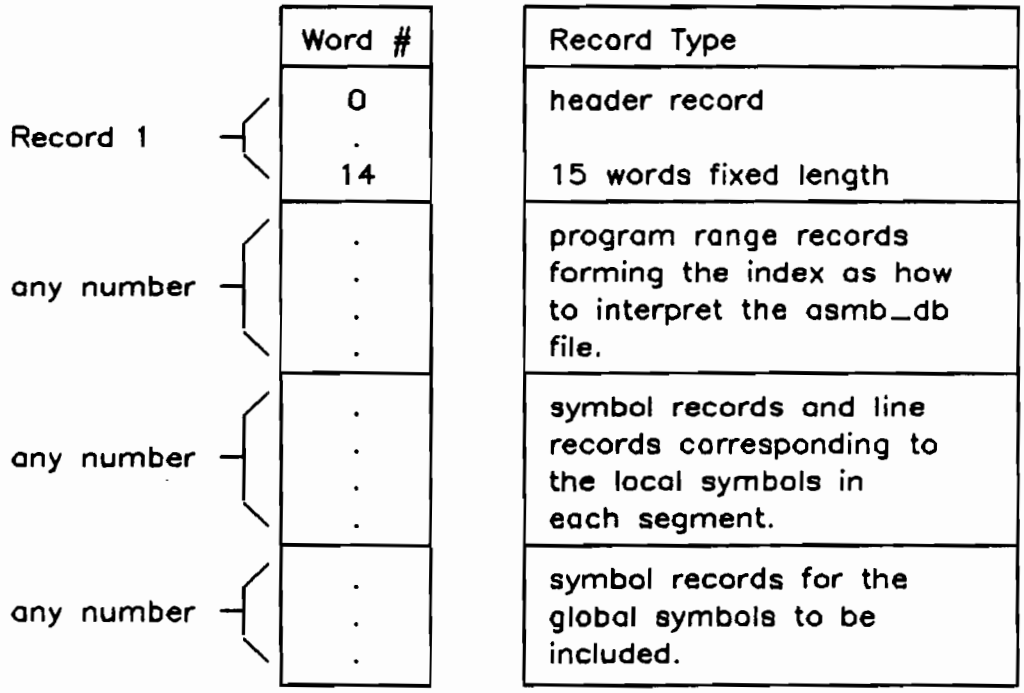

Figure 12-1. Database File Format - State Analyzer

HEADER RECORD FORMAT - STATE ANALYZER DATABASE FILE (See Figure 12-2)

There is a program range record for each segment of the program: PROG, DATA COMM and all absolute segments.

Word 0 - ID code for the database header record

Word 1 thru Word 4 - define the range of physical address within which all of the segments fall.

Word 5 thru Word 8 - range of record numbers for local and global symbols

Word 9 -address information

bit 15 to bit 8 (highshift) - specifies the number of bits to the right to shift a 16-bit segment number to create the physical address.

bit 7 to bit 0 (address size) - indicates whether the addresses to be used are 1 or 2 words long

Word 10 - total number of local symbols in the database

Word 11 - total number of line number symbols in the database

Word 12 - total number of global sysmbols in the database

Word 13 - the identification code of the database file structure

Word 14 - modification of this specific database file structure

Word #	B15 B8	B7 0
0	0000 0000	0000 0000
1	LSW low bound	
2	MSW low bound	
3	LSW high bound	
4	MSW high bound	
5	first local sym rec no	
6	last local sym rec no	
7	first global sym rec no	
8	last global sym rec no	
9	highshift	address size
10	number of local symbols	
11	number of line numbers	
12	number of global symbols	
13	0000 0000	0000 0001
14	0002 0003	0004 0004

Figure 12-2. Header Record Format

32-BIT PROGRAM RANGE RECORD - STATE ANALYZER DATABASE FILE (See Figure 12-3.)

There is a program range record for each segment in the program: PROG, DATA, COMM and each absolute segment.

Word 0 - 32-bit program range record ID code

Word 1 - number of the first symbol record for this segment

Word 2 - number of the last symbol record for this segment

Word 3 - record number of the first line number symbol record for this segment

Word 4 - record number of the last line number symbol record for this segment.

Word 5 thru Word 8 - physical address range of this segment. The LSW and MSW are always in physical address form and are sorted in increasing address order in the file. In the case of the 8086/88, the data in the link__sym file has been converted.

Word #	B15 B8	B7 0		
0	0000	0000	0000	0002
1	symbol start record			
2	symbol end record			
3	line start record			
4	line end record			
5	LSW line bound			
6	MSW low bound			
7	LSW high bound			
8	MSW high bound			
9	fff u u 0 rr		c1	
9 – 16	var length file name			

Figure 12-3. 32-Bit Program Range Record Format

16-BIT SYMBOL RECORD - STATE ANALYZER DATABASE FILE (See Figure 12-4.)

The symbol records contain the symbols and their values, a 16-bit symbol record will be used to conserve space if the address is less than 65536.

Word 0 - ID code for the 16-bit symbol record

Word 1 - symbol length and type

bit 15 to bit 13 - number of 16-bit words minus 1 in the symbol name

bit 12 to bit 10 - always zero

bit 9 to bit 8 - type of segment:
0 - absolute
1 - program relocatable
2 - data relocatable
3 - common relocatable

bit 7 to bit 0 - the beginning of the symbol name filling out the record in standard 64000 format (refer to Appendix B).

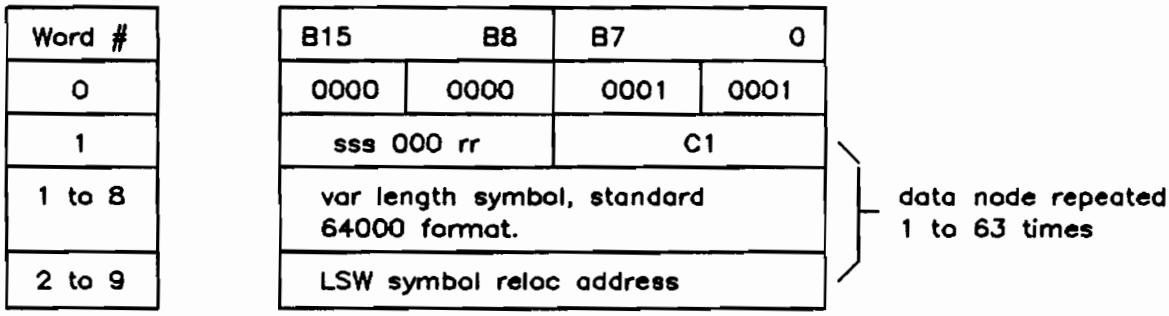

Figure 12-4. 16-Bit Symbol Record Format

32-BIT SYMBOL RECORD - STATE ANALYZER DATABASE FILE (See Figure 12-5.)

Word 0 - ID code for the 32-bit symbol record

Word 1 - symbol length and type

bit 15 to bit 13 - number of 16-bit words minus 1 in the symbol name

bit 12 to bit 10 - always zero

bit 9 to bit 8 - type of segment:
0 - absolute
1 - program relocatable
2 - data relocatable
3 - common relocatable

bit 7 to bit 0 - the beginning of the symbol name filling out the record in standard 64000 format (refer to Appendix B).

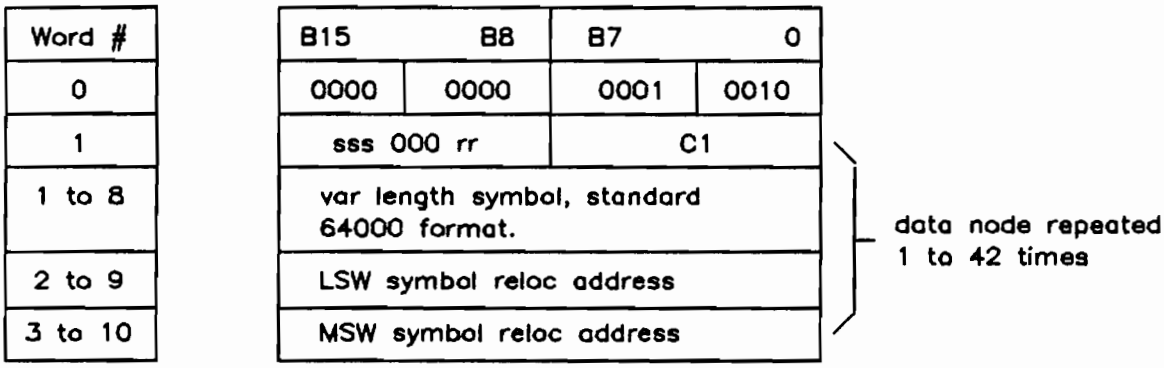

Figure 12-5. 32-Bit Symbol Record Format

16-BIT LINE NUMBER SYMBOL RECORD - STATE ANALYZER DATABASE FILE
(See Figure 12-6.)

Word 0 - ID code for the 16-bit line number symbol

Word 1 to Word 2 - the range of line numbers assigned to the value in Word 3

Word 3 - value of the above line number symbols

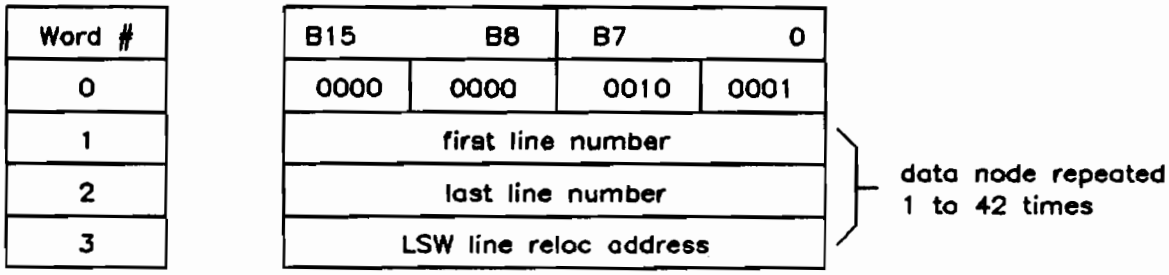

Figure 12-6. 16-Bit Line Number Symbol Record Format

32-BIT LINE NUMBER SYMBOL RECORD - STATE ANALYZER DATABASE FILE
(See Figure 12-7.)

Word 0 - ID code for the 32-bit line number symbol

Word 1 to Word 2 - the range of line numbers assigned to the value in Word 3 & Word 4.

Word 3 and Word 4 - the value of the above line number symbols.

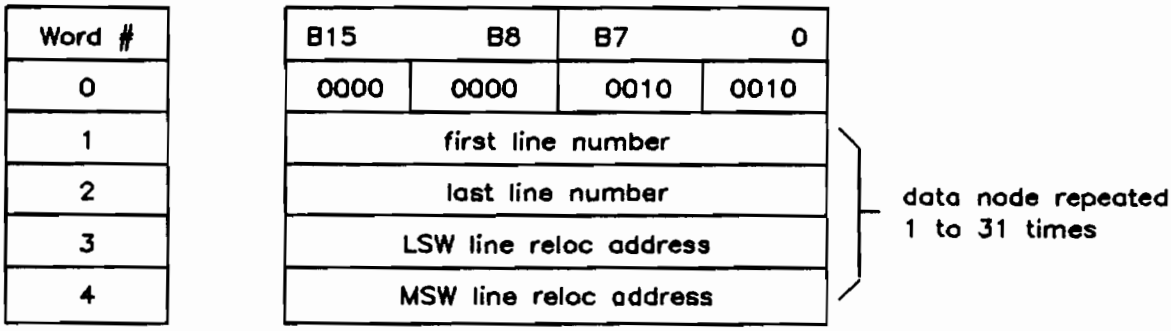

Figure 12-7. 32-Bit Line Number Symbol Record Format

Chapter 13

ASSEMBLER SYMBOL FILE FORMAT

ASSEMBLER SYMBOL FILE (FILE TYPE 12)

An assembler symbol (:asmb__sym) file is generated whenever a source file is assembled or compiled, except with options no__code. It consists of one or more records of up to 128 sixteen-bit words long which contain descriptions for local symbols defined in the source file.

For a pictorial representation of the assembler symbol file format, see Figures 13-1 and 13-2.

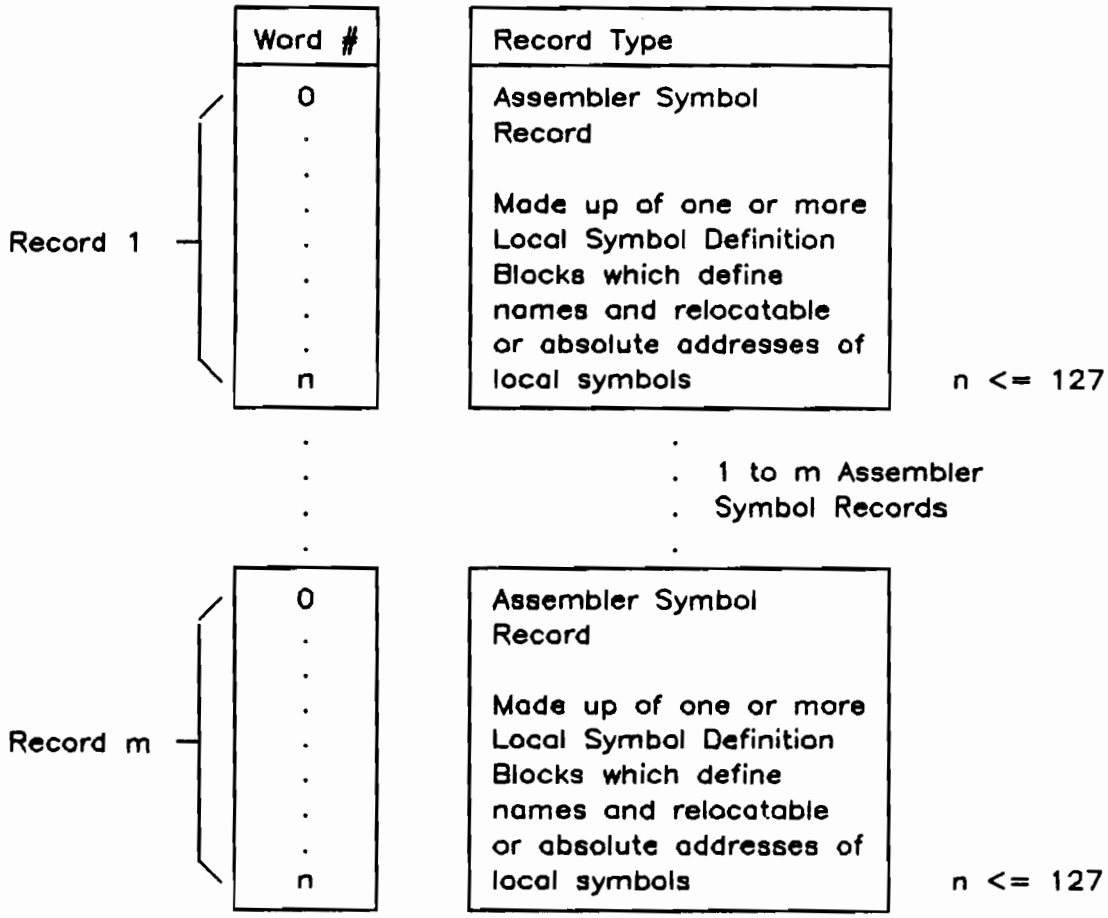

Figure 13-1. Assembler Symbol File - Overall Structure

ASSEMBLER SYMBOL RECORD

WORD 0 - A Record ID of 6 is specified to indicate that this is an Assembler Symbol Record. The Record ID is used internally to the linker symbol file and should not be confused with the 64000 System file type number, which is 12.

WORD 1 through n-1 make up the k Local Symbol Definition Blocks. These blocks define the local symbols from the source file. Each local symbol block has a variable length of from 2 to 10 words.

The structure of each Local Symbol Definition Block is:

WORD 0 through WORD sss (of a Local Symbol Definition Block) make up the Local Symbol Name Description Block. This block provides the name of the local symbol being defined. For a complete description of this block see Appendix D.

WORDS sss+1 and sss+2 (of a Local Symbol Definition Block) contain the value of the symbol. WORD sss+2 exists only in those processors which generate 2 words for each address (see Appendix A).

See the User-Definable Emulator Manual for more information about symbol usage in the UDE.

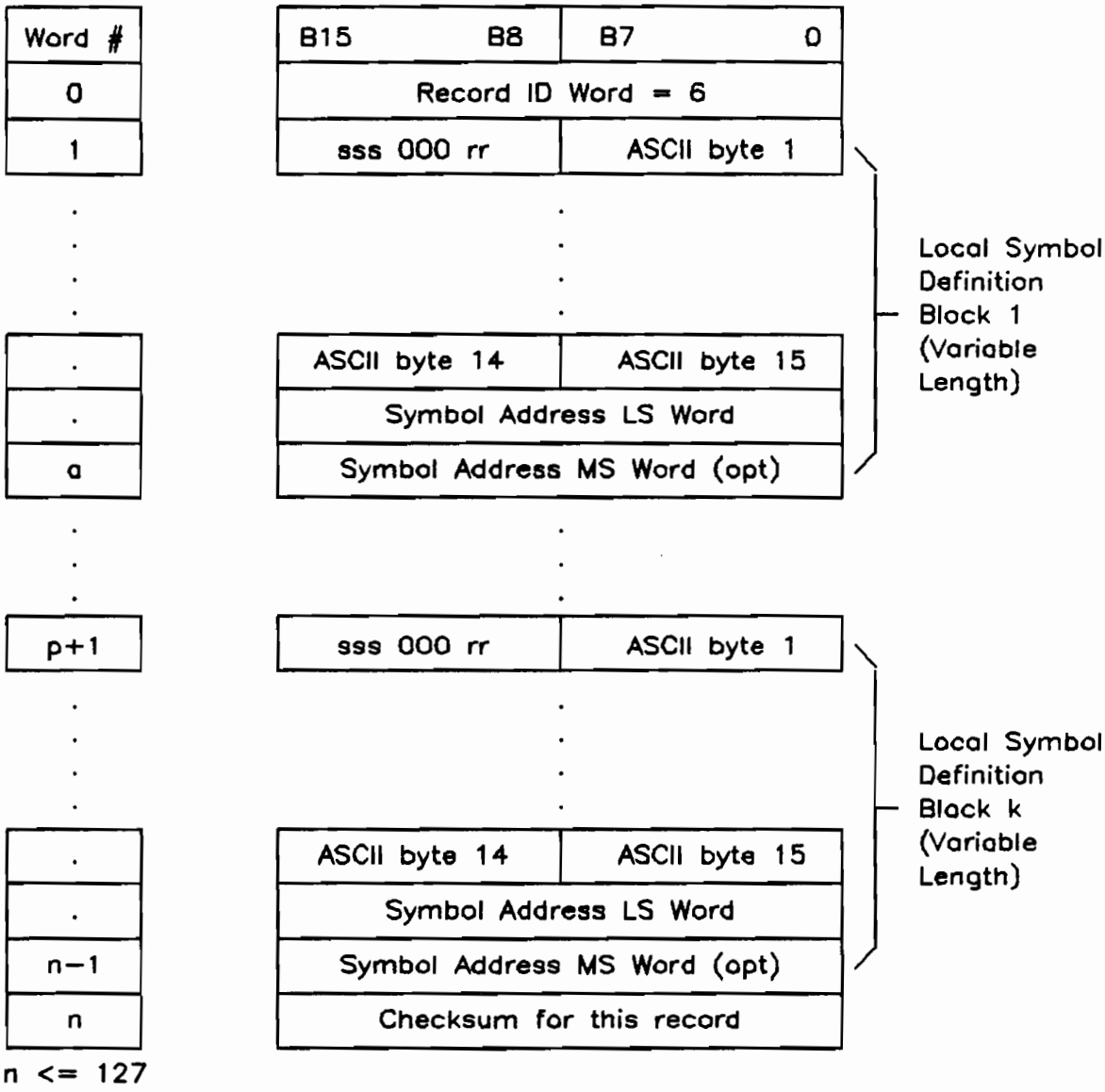

Figure 13-2. Assembler Symbol File -Assembler Symbol Record Format

Chapter 14

LINKER SYMBOL FILE FORMAT

LINKER SYMBOL FILE (FILE TYPE 13)

A linker symbol (:link_sym) file is generated by the linker and it contains four types of records.

There is one Processor Configuration Record, which is used to configure the linker when only link_sym files are being linked. It is followed by one or more Global Symbol Records which provide global symbol names and their relocated addresses. Next, the linker symbol file contains one or more Relocatable Name Records, which provide the names of the relocatable files which created each segment in the absolute file. These are followed by one or more Memory Space Records which give lists of the memory bounds of each of the sections of code which were linked.

In the discussion of the linker symbol file format it is important to keep in mind the relationship between relocatable file names and relocatable file names. The name of the relocatable file that is assembled or compiled is also permanently saved in the Name Record of the Relocatable File (see description of Relocatable File Record Formats). Since files may be renamed, it is possible to have a relocatable file with a name that is different from the relocatable file name which was saved in the Relocatable Name Record. It is also possible to combine multiple relocatable files into a single relocatable file (a library.) In this case there is only one relocatable file name, but multiple relocatable file names are associated with the library.

Relocatable Names are specified in the Relocatable Name Record; File Names in the Memory Space Record.

For a pictorial representation of the linker symbol file format, see Figures 14-1 through 14-5.

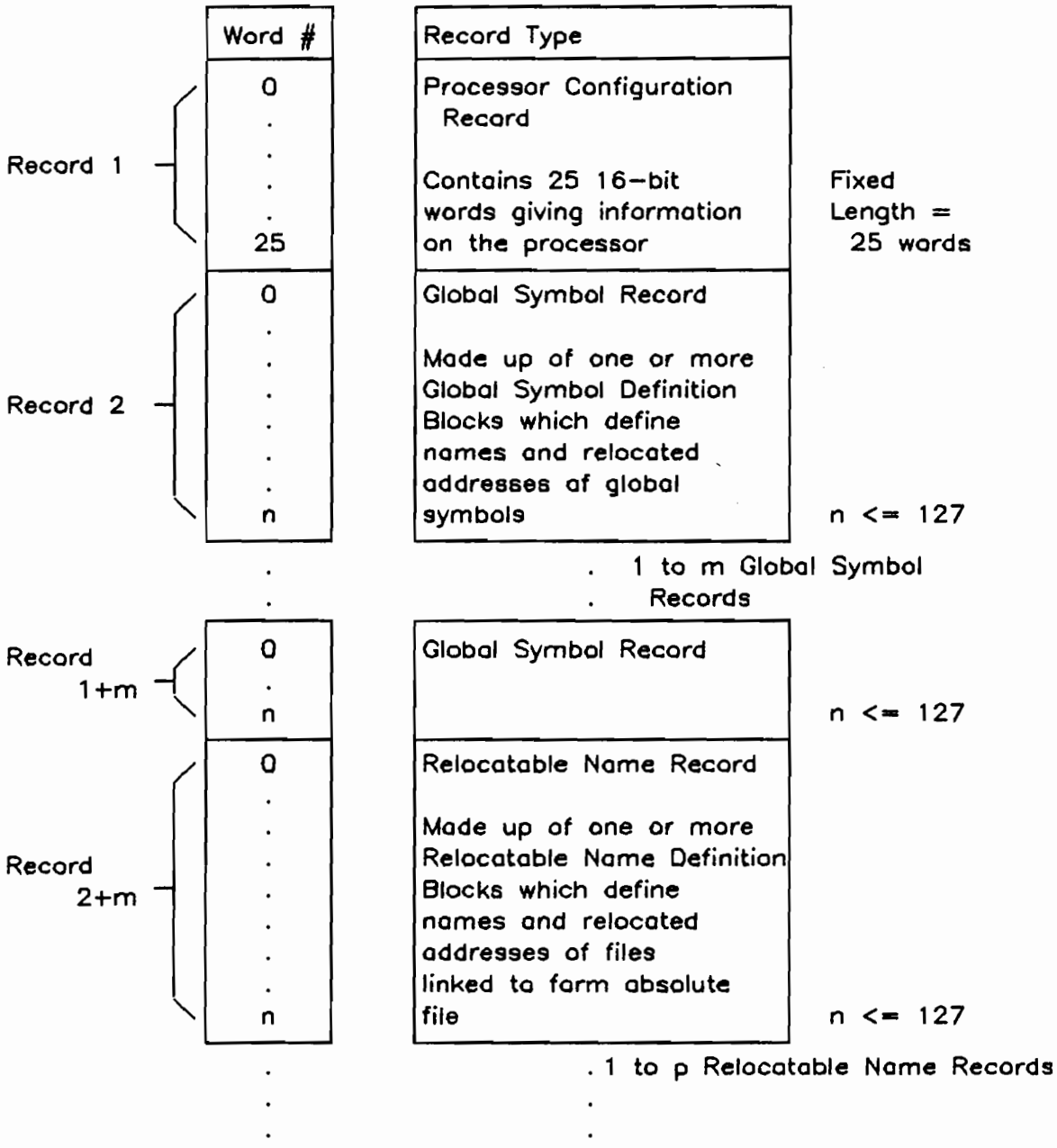

Figure 14-1. Linker Symbol File - Overall Structure
(Continued on next page)

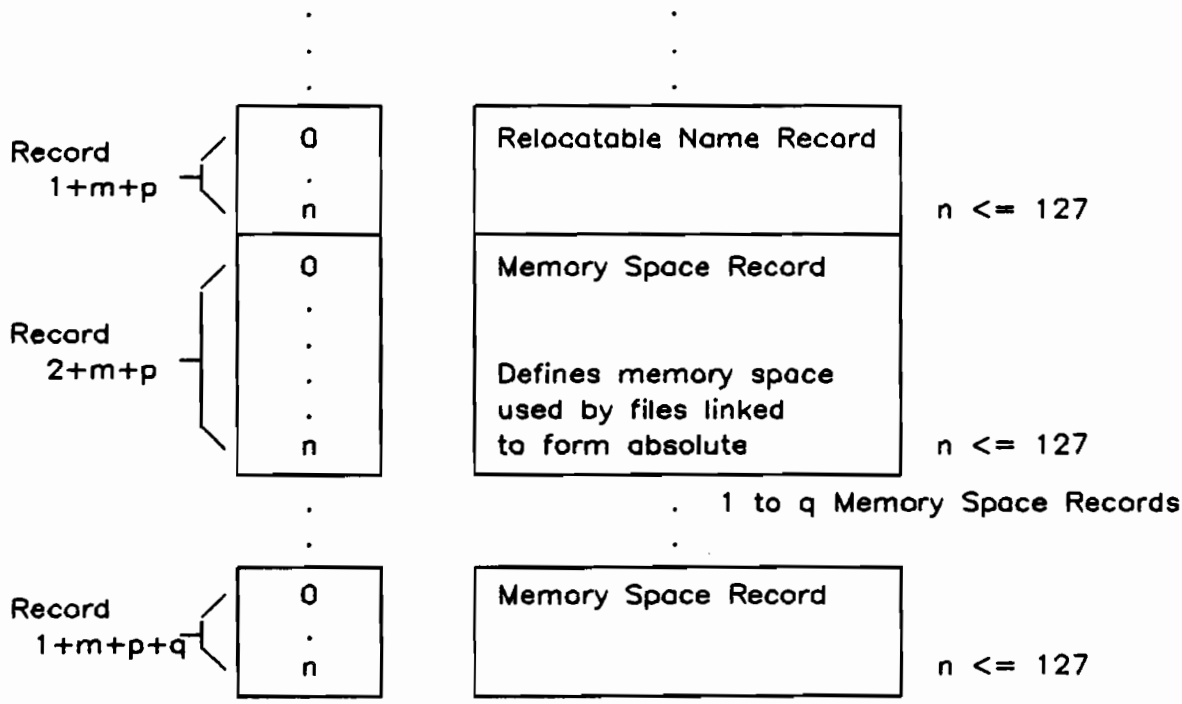

Figure 14-1. Linker Symbol File - Overall Structure (Cont'd)

PROCESSOR CONFIGURATION RECORD

WORD 0 - A Record ID of 1 is specified to indicate that this is a Processor Configuration Record. The Record ID is used internally to the linker symbol file and should not be confused with the 64000 System file type number, which is 13.

WORD 1-15 - These Pad Words containing all zeros are added so that Words 16-23 in this record match the corresponding words in the Name Records in 64000 System Relocatable files.

WORD 16-23 - This group of words is used to define the name of the linker table used to generate the corresponding absolute file. For a complete description of this block see Appendix C.

WORD 24 - Hishift = a number of bits to be used as follows:

For conversion of 32-bit logical addresses consisting of a 16-bit segment and a 16-bit offset to physical addresses, the segment is loaded into the most significant 16 bits of a 32 bit register. It is then shifted RIGHT by the number of bits specified in Hishift and the resulting 32-bit number is added to the 16-bit offset value to obtain the physical address. (For the Intel 8086 and 8088 processors the value of Hishift is 12.)

Address Size is the number of words necessary to define microprocessor addresses. See Appendix A for a description of the number of address words needed for each processor. This field should always contain 1 or 2.

WORD 25 - The checksum word contains the arithmetic sum of the binary values of words 0 through 24.

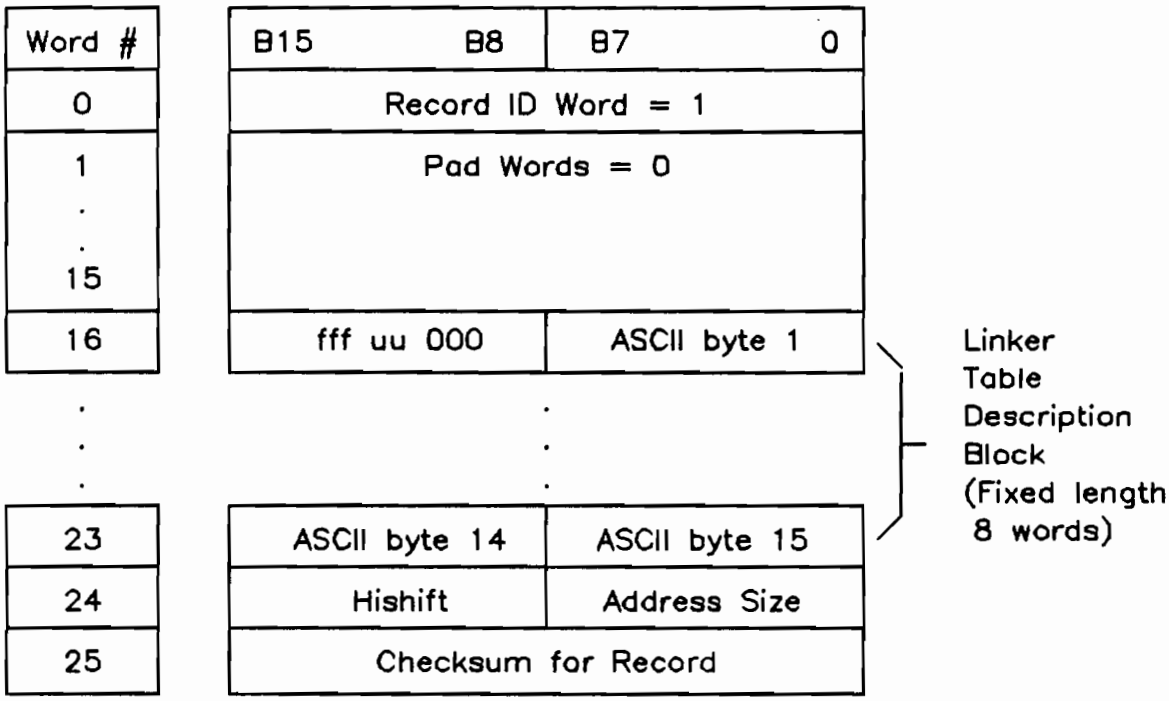

Figure 14-2. Linker Symbol File -Processor Configuration Record Format

GLOBAL SYMBOL RECORD

WORD 0 - A Record ID of 2 is specified to indicate that this is a Global Symbol Record. The Record ID is used internally to the linker symbol file and should not be confused with the 64000 System file type number, which is 13.

WORD 1 through n-1 make up the k Global Symbol Definition Blocks. These blocks describe the global symbols generated as part of the relocatable file. Each global symbol definition block has a variable length of from 2 to 10 words.

The structure of each Global Symbol Definition Block is:

WORD 0 through WORD sss (of a Global Symbol Definition Block) make up the Global Symbol Name Description Block. This block provides the name of the global symbol being defined. For a complete description of this block see Appendix D.

WORDS sss+1 and sss+2 (of a Global Symbol Definition Block) contain the value of the symbol. WORD sss+2 exists only in those processors which generate 2 words for each address (see Appendix A).

See the User-Definable Emulator Manual for more information about symbol usage in the UDE.

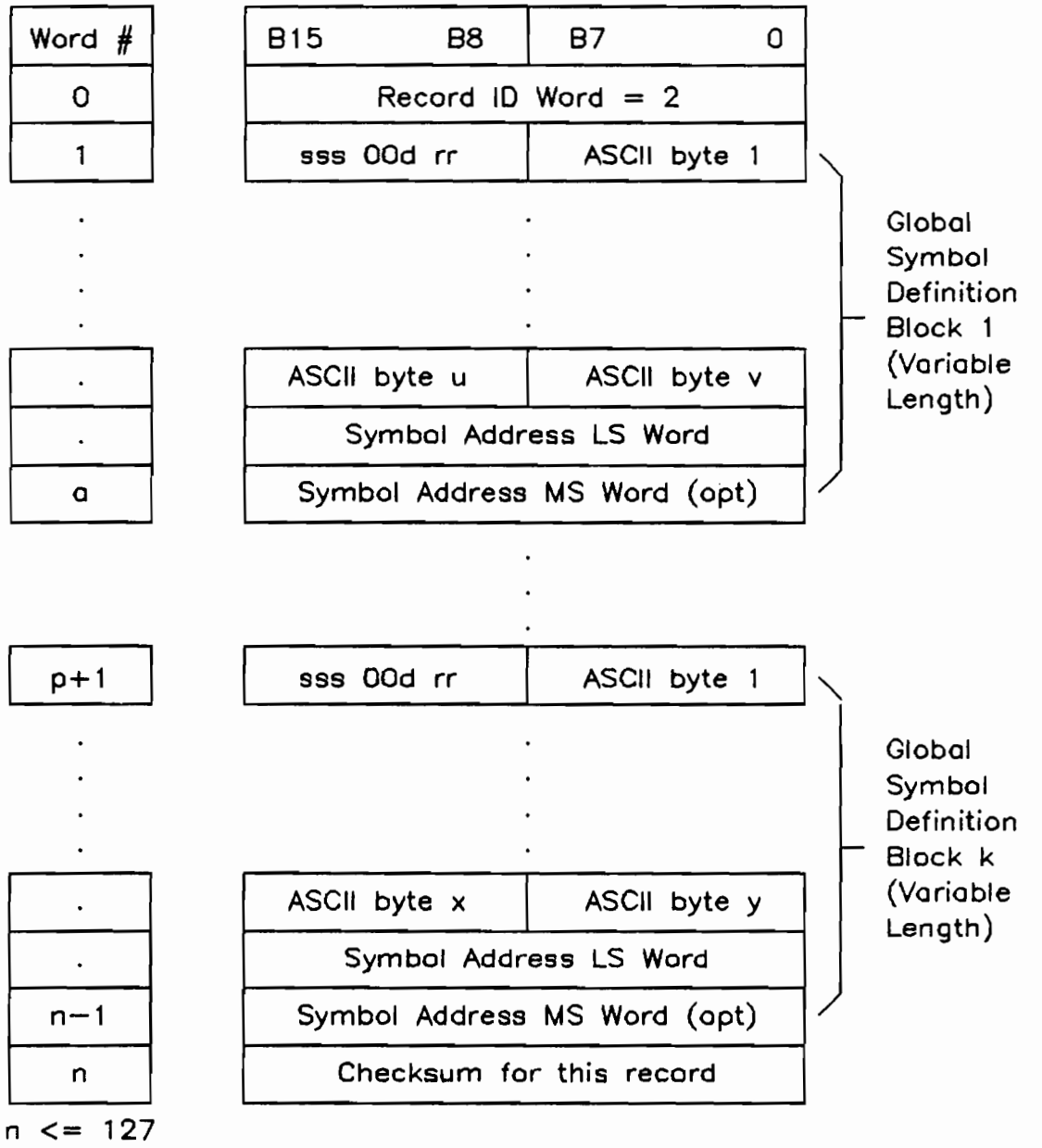

Figure 14-3. Linker Symbol File - Global Symbol Record Format

RELOCATABLE NAME RECORD

WORD 0 - A Record ID of 3 is specified to indicate that this is a Program Name Record. The Record ID is used internally to the linker symbol file and should not be confused with the 64000 System file type number, which is 13.

WORD 1 through n-1 make up the k Relocatable Name Definition Blocks. These blocks describe the relocatable files which generated the absolute files. Each relocatable name definition block has a fixed length of 14 words.

The structure of each Relocatable Name Definition Block is:

WORD 0 through WORD 7 make up the Relocatable Name Description Block. This block provides either the relocatable file name from which this absolute segment was generated or the file name of the library containing the relocatable code from which this absolute segment was created.

WORD 8-9 - The Program Load Address is the address in the microprocessor memory space into which the first byte of the code designated by the linker to reside in program space will be loaded. The remaining bytes of code will follow in subsequent memory locations. If the processor only requires one word of address (see Appendix A), the MSW should be filled with 0's.

WORD 10-11 - The Data Load Address is the address in the microprocessor memory space into which the first byte of the code designated by the linker to reside in data space will be loaded. The remaining bytes of code will follow in subsequent memory locations. If the processor only requires one word of address (see Appendix A), the MSW should be filled with 0's.

WORD 12-13 - The Common Load Address is the address in the microprocessor memory space into which the first byte of the code designated by the linker to reside in common space will be loaded. The remaining bytes of code will follow in subsequent memory locations. If the processor only requires one word of address (see Appendix A), the MSW should be filled with 0's.

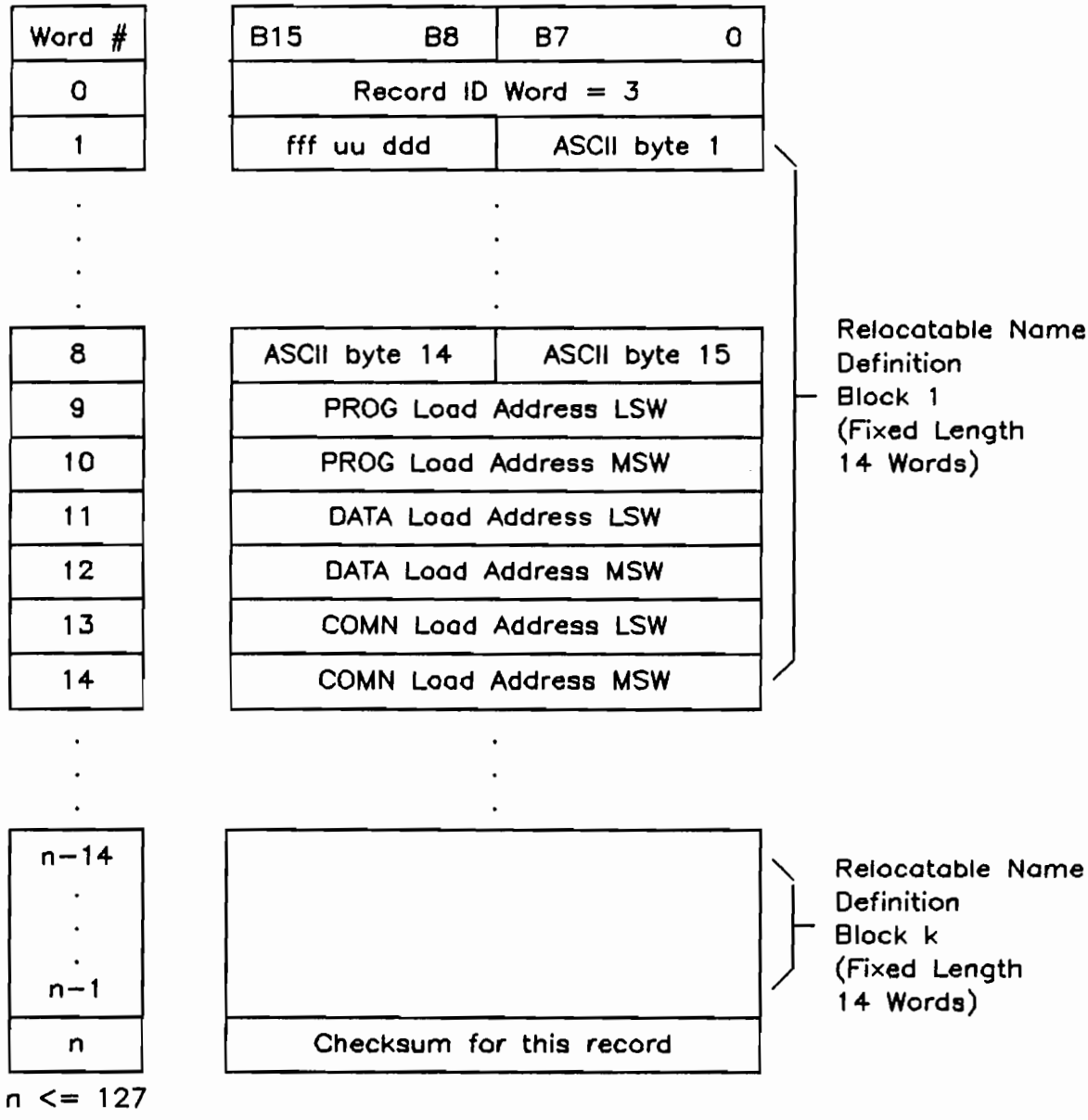

Figure 14-4. Linker Symbol File - Relocatable Name Record Format

MEMORY SPACE RECORD

There are one or more Memory Space Definition Blocks for each Relocatable Name Definition Block in the Linker Symbol File. If only Program space is occupied by the relocatable, then there will be one Memory Space Definition Block corresponding to the Relocatable Name Definition Block. If Program space and Data space are occupied, then there will be two Memory Space Definition Blocks for the Relocatable Name Definition Block, etc.

For absolute code, the Memory Space Definition Blocks contain the ORG'd addresses supplied in the Name Record of the 64000 System Relocatable File.

WORD 0 -A Record ID of 4 is specified to indicate that this is a Memory Space Record. The Record ID is used internally to the linker symbol file and should not be confused with the 64000 System file type number, which is 13.

WORD 1 through n-1 make up the k Memory Space Definition Blocks.

The structure of each Memory Space Definition Block is:

WORD 0-1 -The Low Bound Address is the Load Address as defined in the Relocatable Name Definition Block,or in the case of absolute code, the address as defined in the Absolute Code Segment Block of the Name Record in 64000 System Relocatable Files. It is keyed to either Program, Data, Common, or Absolute by the contents of the rr bits in WORD 4 of this block. If the processor only requires one word of address (see Appendix A), the MSW should be filled with 0's.

Memory Space Definition Blocks appear in the Memory Space Record in sorted order on the Low Bound Address, with the record with the smallest Low Bound Address appearing first, the next largest next, etc.

WORD 2-3 -The Hi Bound Address is the last address filled by the code which started at the Low Bound Address defined in WORD 1-2. If the processor only requires one word of address, the MSW should be filled with 0's.

WORD 4 - The contents of Word 4 of each Memory Space Definition Blocks are broken down as follows:

Relocatable Index # - bits 15-2 - is obtained by numbering each of the Relocatable Name Definition Blocks in the Relocatable Name Record starting with the number 0. The Memory Space Definition Block is then tied to the Relocatable Name Definition Block by supplying its numerical order number in this field.

rr - bits 1,0 - a two bit field used to indicate the program counter with respect to which the code has been relocated. The meaning of the value is as follows:

00 - Absolute
01 - Program
10 - Data
11 - Common

WORD 5 through WORD 12 make up the Relocatable file Description Block. This block provides the name of the relocatable file from which this segment of the code was generated. For a complete description of this block see Appendix E.

WORD 13 - bits 14,13,12 (ddd) - contains the disc number where the file described in this block resides.

WORD 13 - All remaining bits are reserved for future use by the 64000 system and should contain 0's

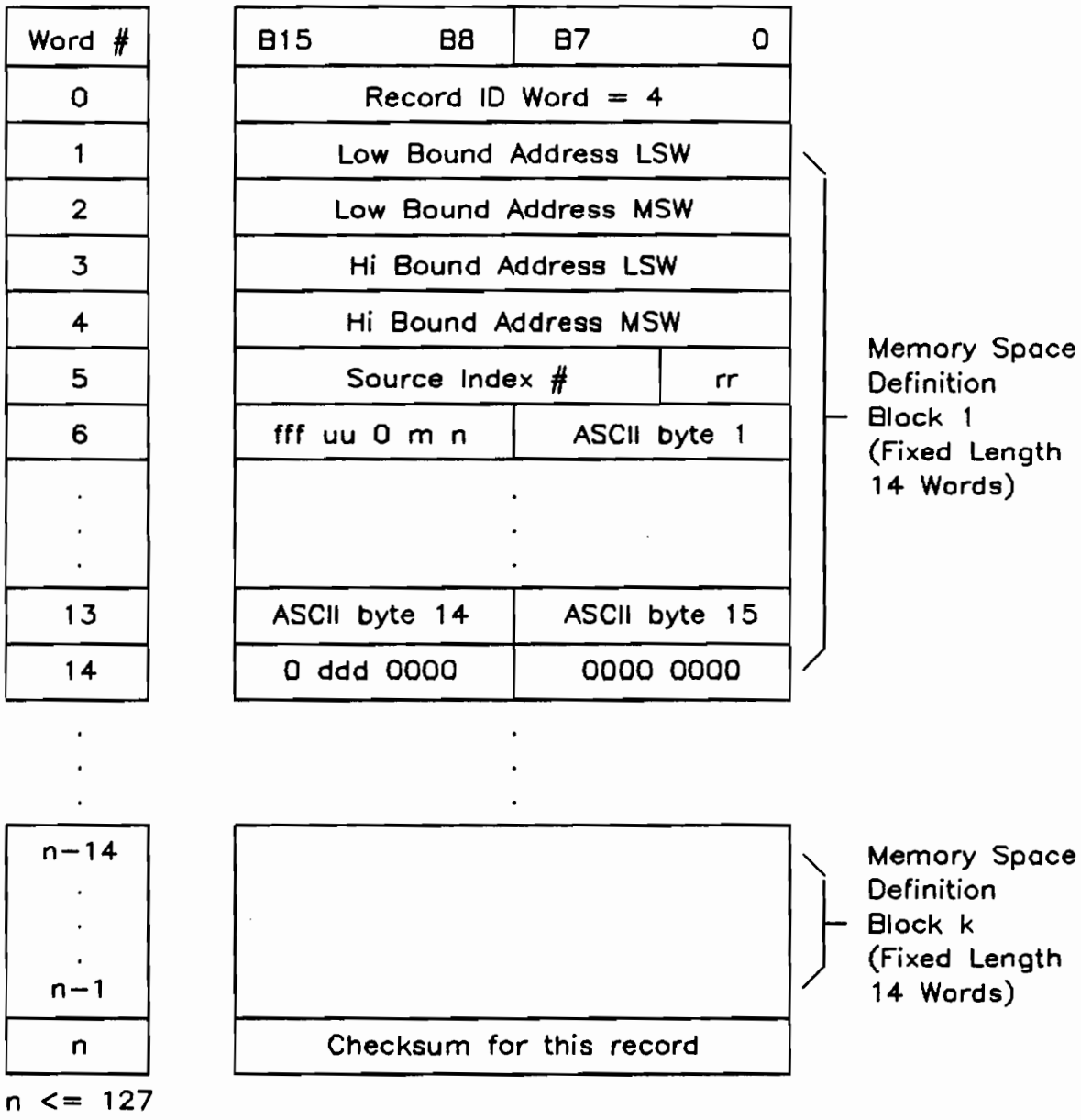

Figure 14-5. Linker Symbol File - Memory Space Record Format

Chapter 15
TEMPORARY FILE FORMAT

TEMPORARY FILE (FILE TYPE 800H - 8FFH)

REGULAR TEMP FILES

Temporary files are created in several ways, but are rarely seen by the user. A temporary file will exist on a disc if someone is using one of the modules which uses a temporary file. In the normal course of events, that temp file will disappear when the process is finished. However, if the 64000 station is powered off, or a shift RESET is performed, the temporary files may remain on the disc. A temporary file is always associated with a System Bus address, so that in order to get rid of a temp file one must work at the station with that System Bus address.

SPECIAL TEMP FILES

scratch1N:HP, scratch2N:HP, Ndestfile:HP, Ntempfile:HP

One process that uses temporary files is the editor. Editor temporary files may have one of the following names, scratch1N:HP, scratch2N:HP, Ndestfile:HP, or Ntempfile:HP. In all these cases, N is the System Bus address of the station which created the file. If an editor temporary file remains on your disc, get into the editor, perform the action listed below and then end or RESET RESET out of the file. (NOTE: do NOT shift RESET) If the temporary file is a Ntempfile:HP you will need to do a copy or extract in the editor. If the temporary file is a scratchxN:HP, try merging a large file or paging back and forth in a large file.

sym__N:default userid

Emulation will use temporary files for sorting symbols. The file name associated with an emulation temp file is sym__N:HP where N is the System Bus address of the station. To get rid of this file, get into emulation and display local__symbols or display global__symbols. The file should be gone when you get out of emulation.

Chapter 16
DEVICE FILE FORMAT

DEVICE FILE (FILE TYPE 8000H-8FFFH)

Device files are dummy files used by the file manager system so that the printer, display, and HPIB input and output can be treated in the same manner as disc files. Information regarding the devices is kept in the file directory under a name selected for that device. File names for the devices are:

printer:HP
display:HP
hpibi:HP (HPIB input)
hpibo:HP (HPIB output)

The device files printer:HP and display:HP can be accessed using Simulated I/O and the copy command. The files hpibi:HP and hpibo:HP can only be accessed using the copy command in a standalone system. For more information on accessing these files, see the System Software Reference Manual for information about the copy command, and any emulation manual for information about Simulated I/O.

Chapter 17
SIMULATED I/O FILE FORMAT

SIMULATED I/O FILE (all non-specified types)

Simulated I/O files are for the convenience of the user. They are not used by the HP64000 in a system capacity. They are created and accessed using Simulated I/O from a 64000 System Emulator or HOST Pascal. The system commands {copy, rename, directory, purge, and recover} will not work on files of type :sim_io. If you want to access files made by Simulated I/O or HOST Pascal using system commands, use file type = 10 (data files).

At some point, file type numbers which are currently unassigned may be assigned to new file types. It is suggested that user programs use files of type 10 (data), and not use other file types which are currently unassigned.

For a pictorial representation of the Simulated I/O file format, see Figures 17-1 and 17-2.

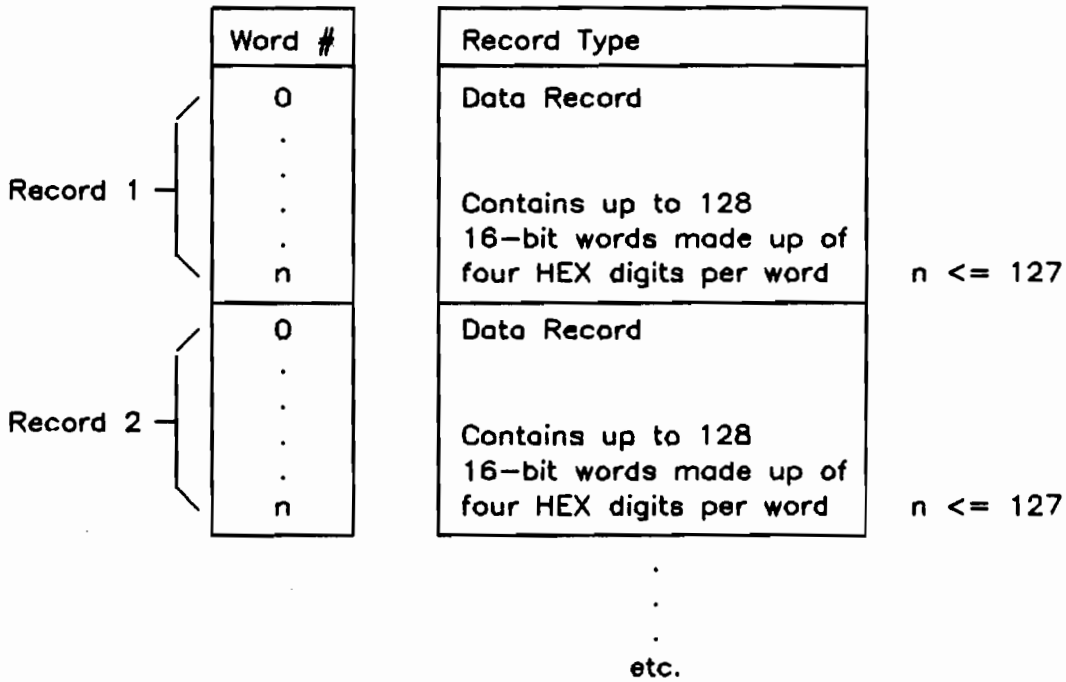

Figure 17-1. Simulated I/O Files - Overall Structure

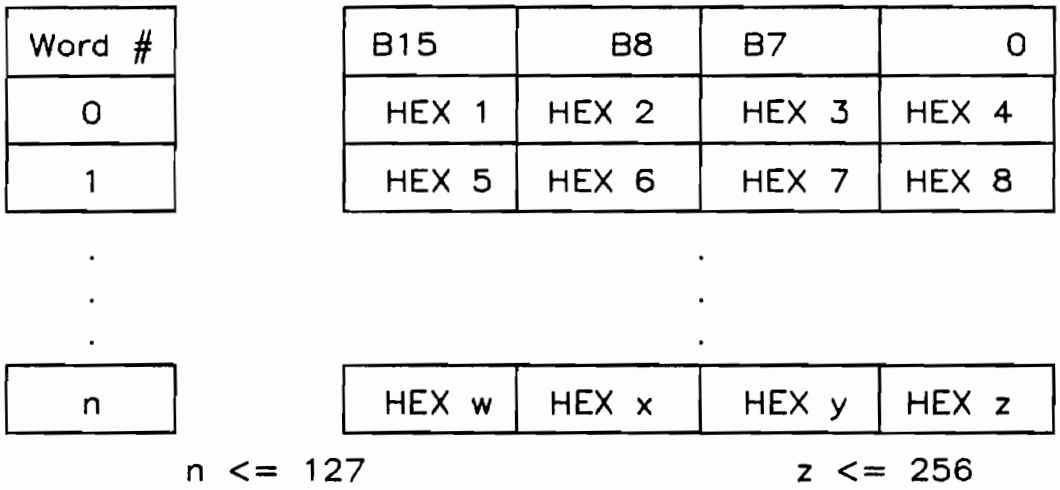

Figure 17-2. Data Record Format

Appendix A

SUPPORTED PROCESSORS AND FORMAT NUMBER DESCRIPTIONS

Supported microprocessors are listed below in Table A-1.

Table A-1. Supported Microprocessors

micro-processor	words used to define address	words used to define skeleton	processor data bus width	data width base
8080/85 Z80	1 word	1 word	8 bits	8 bits
6800/01/ 03/05/09	1 word	1 word	8 bits	8 bits
650x	1 word	1 word	8 bits	8 bits
68000	2 words	2 words	16 bits	8 bits
8021/22 /41/48	1 word	1 word	8 bits	8 bits
9900/40/ 85/89/99	1 word	1 word	16 bits	8 bits
9980	1 word	1 word	8 bits	8 bits
99xxx	1 word	1 word	16 bits	16 bits
1802	1 word	1 word	8 bits	8 bits
F8	1 word	1 word	8 bits	8 bits
Z8	1 word	1 word	8 bits	8 bits
8086/88/89/ 186/188/286	2 words	2 words	16 bits	8 bits
Z8001/2	2 words	2 words	16 bits	8 bits
8051	1 word	1 word	8 bits	8 bits
1750A	1 word	1 word	16 bits	16 bits
TMS320	2 words	2 words	16 bits	16 bits

FORMAT NUMBERS AND SKELETON REQUIREMENTS

8080/85 and Z80

Uses linker table 18085_Z80:HP

The first step in all cases is to read the 16-bit address/displacement field passed by the assembler and add to it the value of the relocation counter or external symbol specified. This quantity will be referred to as the address in the definitions listed in Table A-2. All output is to the absolute file.

Table A-2. Format Number Descriptions for 8080/85, Z80

Format #	words of skeleton	Definition
0	0	Write address – high byte, low byte
1	0	Write address – low byte, high byte
2	0	Write low byte of address
3	0	Write high byte of address
4	0	Write low byte of address Check address in range 0 to 255
5	0	Write low byte of address Check address in range −128 to 127
6	0	Write low byte of address Check address in range −126 to 129

6800/01/02/03/05/09

Uses linker table l68XX:HP

The first step in all cases is to read the 16-bit address/displacement field passed by the assembler and add to it the value of the relocation counter or external symbol specified. This quantity will be referred to as the address in the definitions shown in Table A-3. All output is to the absolute file.

Table A-3. Format Number Descriptions for 6800/01/02/03/05/09

Format #	words of skeleton	Definition
0	0	Write address – high byte, low byte
1	0	offset = current PC – address –1 Write low byte of offset Check offset in range –128 to 127
2	0	Write low byte of address Check address in range –128 to 255
3	0	offset = current PC – address –2 Write offset – high byte, low byte
4	1	6808 only. Skeleton has value of DP register. offset = address – skeleton Write low byte of offset. Check for offset in range 0 to 255

650X

Uses linker table l650X:HP

The first step in all cases is to read the 16-bit address/displacement field passed by the assembler and add to it the value of the relocation counter or external symbol specified. This quantity will be referred to as the address in the definitions shown in Table A-4. All output is to the absolute file.

Table A-4. Format Number Descriptions for 650X

Format #	words of skeleton	Definition
0	0	Write address – low byte, high byte
1	0	offset = current PC – address –1 Write low byte of offset Check offset in range –128 to 127
2	0	Write low byte of address Check address in range 0 to 255
3	0	Write low byte of address
4	0	Write low byte of address Check address in range –128 to 255

68000

Uses linker table l68000:HP

The first step in all cases is to read the 32-bit address/displacement field passed by the assembler and add to it the value of the relocation counter or external symbol specified. This quantity will be referred to as the address in the definitions listed in Table A-5. All output is to the absolute file.

Table A-5. Format Number Descriptions for 68000

Format #	words of skeleton	Definition
0	0	Write low byte of address Check address in range −128 to 255
1	0	Write address − high byte, low byte Check address in range −32768 to 65535
2	0	Write address − high word: high byte, low byte, low word: high byte, low byte
3	0	offset = address − current PC Write offset − high byte, low byte Check offset in range −32768 to 32767
4	0	Offset = address − current PC −1 If offset is odd, add 1 to put on even boundary. Write low byte of offset Check offset in range −128 to 127
5	0	Offset = address − current PC −1 Write low byte of offset Check offset in range −128 to 127
6	0	Write address − high byte, low byte Check address in range −32768 to 32767
7	0	Write low byte of address Check address in range −128 to 127
8	0	offset = address − current PC Write offset − high word: high byte, low byte, low word: high byte, low byte
9	0	Offset = address − current PC −1 If offset is odd, add 1 to put on even boundary. Write low byte of offset Check offset in range −126 to 129
10	2	Skeleton bits 7,6,5,4 contain object code. Check address in range 0 − 16 Output byte consisting of bits 7,6,5,4 from the skeleton and bits 3,2,1,0 from the address.

8021/22/41/48

Uses linker table 18048:HP

The first step in all cases is to read the 16-bit address/displacement field passed by the assembler and add to it the value of the relocation counter or external symbol specified. This quantity will be referred to as the address in the definitions shown in Table A-6. All output is to the absolute file.

Table A-6. Format Number Descriptions for 8021/22/41/48

Format #	words of skeleton	Definition
0	0	Write low byte of address Check address in range of current page
1	0	Write low byte of address
2	1	8048 only. Check address in range 0 to 4096 Write byte consisting of bits 10,9,8 of address, and bits 4,3,2,1,0 of skeleton Then write low byte of address
3	0	Write address – high byte, low byte
4	0	Write high byte of address
5	1	8021/41 only Check address in range 0 to 1024 Write byte consisting of bits 10,9,8 of address, and bits 4,3,2,1,0 of skeleton Then write low byte of address

9900/40/85/89/99

Uses linker table l99XX:HP

The first step in all cases is to read the 16-bit address/displacement field passed by the assembler and add to it the value of the relocation counter or external symbol specified. This quantity will be referred to as the address in the definitions shown in Tables A-7 and A-8. All output is to the absolute file.

Table A-7. Format Number Descriptions for 9900/40/85/89/99

Format #	words of skeleton	Definition
0	0	Write low byte of address Check address in range −128 to 127
1	0	Write address − high byte, low byte
2	0	offset = address − current PC −1 Write low byte of offset Check offset in range −128 to 127
3	0	Write low byte of address

Table A-8. Format Number Descriptions for 9980

Format #	words of skeleton	Definition
0	0	Write low byte of address Check address in range −128 to 127
1	0	Write address − high byte, low byte
2	0	offset = address − current PC −1 Write low byte of offset Check offset in range −128 to 127
3	0	Write low byte of address

99XXX

Uses linker table l99XXX:HP

The first step in all cases is to read the 16-bit address/displacement field passed by the assembler and add to it the value of the relocation counter or external symbol specified. This quantity will be referred to as the address in the definitions shown in Table A-9. All output is to the absolute file.

Table A-9 Format Number Descriptions for 99XXX

Format #	words of skeleton	Definition
0	0	Write low byte of address Check address in range −128 to 127
1	0	Write address − high byte, low byte
2	0	offset = address − current PC −1 Write low byte of offset Check offset in range −128 to 127
3	0	Write low byte of address

1802

Uses linker table l1802:HP

The first step in all cases is to read the 16-bit address/displacement field passed by the assembler and add to it the value of the relocation counter or external symbol specified. This quantity will be referred to as the address in the definitions shown in Table A-10. All output is to the absolute file.

Table A-10. Format Number Descriptions for 1802

Format #	words of skeleton	Definition
0	0	Write address – high byte, low byte
1	0	Write low byte of address
2	0	Write high byte of address
3	0	Offset = address – current page Write low byte of offset Check offset in range of current page

F8

Uses linker table lF8:HP

The first step in all cases is to read the 16-bit address/displacement field passed by the assembler and add to it the value of the relocation counter or external symbol specified. This quantity will be referred to as the address in the definitions shown in Table A-11. All output is to the absolute file.

Table A-11. Format Number Descriptions for F8

Format #	words of skeleton	Definition
0	0	offset = address − current PC Write low byte of offset Check offset in range −128 to 127
1	0	Write address − high byte, low byte
2	0	Write low byte of address Check address in range 0 to 255
3	1	Write byte consisting of skeleton bits 7,6,5,4 and address bits 3,2,1,0 Check address in range 0 to 15
4	1	Write byte consisting of skeleton bits 7,6,5,4,3 and address bits 2,1,0 Check address in range 0 to 7

Z8

Uses linker table |Z8:HP

The first step in all cases is to read the 16-bit address/displacement field passed by the assembler and add to it the value of the relocation counter or external symbol specified. This quantity will be referred to as the address in the definitions shown in Table A-12. All output is to the absolute file.

Table A-12. Format Number Descriptions for Z8

Format #	words of skeleton	Definition
0	0	Write low byte of address Check address in register range: o to 128, 240 to 255
1	0	Write low byte of address Check address is even and in register range: 0 to 128, 240 to 255
2	0	Write low byte of address Check address in range 0 to 255
3	0	Offset = address − current PC −1 Write low byte of offset Check offset in range −128 to 127
4	0	Write address − high byte, low byte
5	0	Write low byte of address
6	0	Write high byte of address

8086/88/89/186/188/286

Uses linker table 18086:HP for 8086, 8089, or 80186
Uses linker table 18088:HP for 8088, 8089, or 80188
Uses linker table 180286:HP for 80286

The first step in all cases is to read the 32 bit address/displacement field passed by the assembler and add to it the value of the relocation counter or external symbol specified. This quantity will be referred to as the address in the definitions shown in Table A-13. All output is to the absolute file.

Table A-13. Format Number Descriptions for 8086/88/89/186/188/286

Format #	words of skeleton	Description
0	0	Write low byte of address (low word) Check address in range −128 to 127
1	0	Write low byte of address (low word) Check address in range −256 to 255
2	0	Write low byte of address (low word)
3	0	Write high byte of address (low word)
4	0	Write address − low word: high byte, low byte
5	0	Write address − low word: low byte, high byte
6	0	Write address − high word: low byte, high byte
7	0	Write address − low word: low byte, high byte high word: low byte, high byte
8	0	offset = address − current PC −1 Write low byte of offset (low word) Check offset in range −128 to 127
9	0	offset = address − current PC −2 Write offset low word − high byte, low byte Check offset in range −32768 to 32767
10	0	Writes NOP; increments current module number
11	0	Writes current module number
12	0	Write low byte of address (low word) Check address in range 0 to 255
13	0	Write address − low word: low byte, high byte, high word: low byte

Z8001/2

Uses linker table lZ8000:HP

The first step in all cases is to read the 32 bit address/displacement field passed by the assembler and add to it the value of the relocation counter or external symbol specified. This quantity will be referred to as the address in the definitions shown in Table A-14. All output is to the absolute file.

Table A-14. Format Number Descriptions for Z8001/2

Format #	words of skeleton	Definition
0	0	Write byte consisting of 0 followed by low 7 bits of high word of address. Then write Low byte of low word of address. Check for high byte of low word of address = 0
1	0	Write byte consisting of 1 followed by low 7 bits of high word of address. Then write a byte of 0's.
2	0	Write byte consisting of 0 followed by low 7 bits of low word of address
3	0	Write low byte of low word of address. Check upper byte of low word = 0
4	2	Write high byte of low word of skeleton Write byte consisting of bit 7 of the low word of skeleton and bits 6,5,4,3, 2,1,0 of the low byte of the low word of the address. Check for upper 9 bits of low word of address = 0. Check for high word of address = legal segment
5	2	Write high byte of low word of skeleton Write low byte of low word of address. Check for upper byte of address = 0 Check for high word of address to be legal segment

Table A-14. Format Number Descriptions for Z8001/2 (Cont'd)

6	2	Write byte consisting of bits 15,14,13, 12 of the low word of skeleton followed by bits 11,10,9,8 of the low word of address. Then write low byte of low word of address. Check for upper 4 bits of low word of address = 0. Check for high word of address = legal segment
7	0	offset = address − current PC −2 Write low word of offset − high byte low byte. Check for high word of address to be legal segment.
8	0	Write low byte of low word of address
9	0	Write low byte of low word of address twice. Check high byte of low word = 0
10	2	offset = address − 1 Write high byte of low word of skeleton Write byte consisting of bits 7,6,5,4 of low word of skeleton followed by bits 3,2,1,0 of low word of offset. Check bits 15 to 4 of offset = 0
11	2	Write high byte of low word of skeleton Write byte consisting of bits 7,6,5,4 of low word of skeleton followed by bits 3,2,1,0 of low word of address. Check bits 15 to 4 of address = 0

Table A-14. Format Number Descriptions for Z8001/2 (Cont'd)

12	2	Write high byte of low word of skeleton Write byte consisting of bits 7,6,5,4,3 of low word of skeleton followed by bits 2,1,0 of low word of address. Check bits 15 to 3 of address = 0
13	2	Write high byte of low word of skeleton Write low byte of low word of address. Check for upper byte of address = 0
14	2	Write high byte of low word of skeleton If address = 2, offset = 2 If address = 1, offset = 0 Otherwise ERROR. Write byte consisting of bits 7,6,5,4, 3,2,of low word of skeleton, followed by bits 1,0 of low word of offset.
15	0	If address >32 then offset = 32 Otherwise offset = address Write low word of offset – high byte low byte
16	0	If address >16 then offset = 16 Otherwise offset = address Write low word of offset – high byte low byte

Table A-14. Format Number Descriptions for Z8001/2 (Cont'd)

17	0	If address > 8 then offset = 8 Otherwise offset = address Write low word of offset – high byte low byte
18	0	If address >32 then offset = −32 Otherwise offset = twos complement of address. Write low word of offset – high byte low byte
19	0	If address >16 then offset = −16 Otherwise offset = twos complement of address. Write low word of offset – high byte low byte
20	0	If address >8 then offset = −8 Otherwise offset = twos complement of address. Write low word of offset – high byte low byte
21	0	Write low word of address – high byte, low byte. Check upper word = 0.

Table A-14. Format Number Descriptions for Z8001/2 (Cont'd)

22	0	Write low word of address – high byte low byte
23	0	Write byte consisting of 1 followed by low 7 bits of low word of address. Then write a byte of 0's.
24	0	Write low word of address – high byte, low byte
25	0	Write address – high word: high byte, low byte, low word: high byte, low byte
26	0	Write byte consisting of 1 followed by low 7 bits of high word of address. Then write a byte of 0's. Then write the low word of address – high byte, low byte.
27	2	Write high byte of low word of skeleton Write low byte of low word of address Check address in range –128 to 255
28	0	Write low word of address – high byte, low byte. Check address in range –32768 to 65535
29	0	Write low byte of low word of address twice. Check address in range –128 to 255.

8051

Uses linker table l8051:HP

The first step in all cases is to read the 16-bit address/displacement field passed by the assembler and add to it the value of the relocation counter or external symbol specified. This quantity will be referred to as the address in the definitions shown in Table A-15. All output is to the absolute file.

Table A-15. Format Number Descriptions for 8051

Format #	words of skeleton	Definition
0	0	offset = address − current PC −1 Write low byte of offset Check offset in range −127 to 128
1	0	Write low byte of address Check address in range −256 to 255
2	1	Check address in range 0 to 2048 Write byte consisting of bits 10,9,8, of address, and bits 4,3,2,1,0 of skeleton Then write low byte of address
3	0	Write address − high byte, low byte
4	0	Write high byte of address
5	0	Write low byte of address
6	0	Write low byte of address Check address in range 0 to 255

1750A

Uses linker table l1750A:HP

The first step in all cases is to read the 32 bit address/displacement field passed by the assembler and add to it the value of the relocation counter or external symbol specified. This quantity will be referred to as the address in the definitions shown in Table A-16. All output is to the absolute file.

Table A-16. Format Number Descriptions for 1750A

Format #	words of skeleton	Description
0	0	Write low word of address high byte, low byte
1	0	offset = address − current PC write low byte of offset check offset in range −128 to 127
2	0	Write address high word − high byte, low byte; low word − high byte, low byte

TMS320

Uses linker table lTMS320:HP

The first step in all cases is to read the 32 bit address/displacement field passed by the assembler and add to it the value of the relocation counter or external symbol specified. This quantity will be referred to as the address in the definitions shown in Table A-17. All output is to the absolute file.

Table A-17. Format Number Descriptions for TMS320

Format #	words of skeleton	Description
0	0	output low byte of address, check for <256
1	0	output low byte of address, check for <128
2	2	OR skeleton with bits 3,2,1,0 of address. Output low byte, check address <16
3	2	OR skeleton with bits 2,1,0 of address. Output low byte, check address <8
4	0	Write low word of address high byte, low byte, check address <4096
5	2	OR bits 15,14,13 of skeleton with bits 12–0 of address. Write high byte, low byte
6	2	Write low byte of skeleton. Check for skeleton =0
7	2	OR skeleton with address – output low byte. Check address = 0,1 or 4.
8	2	OR low bit of address with skeleton Write low byte, check address <=1
9	0	Write low word of address high byte, low byte. Check address <=65535 >=–32767
10	0	Output 32 bit address high word: high byte, low byte, low word: high byte, low byte

Appendix B

FILE NAME DESCRIPTION - FIXED LENGTH

WORD 0 - The contents of Word 0 (see Figure B-1) is broken down as follows:

fff - bits 15,14,13 - contain the number of 16-bit words necessary to define the file name. The file name is packed two characters per word, with the first character stored in the last byte of word 1. This number is calculated as follows: Take the number of characters in the file name and subtract one. Divide this number by two and round up. For example, the file name START1 consists of 6 ASCII characters. This would require 3 words in addition to word 1 in which to fit all the characters of the name. (6-1) /2 = 2.5 which rounds up to 3. If the last character of the file name is stored in the upper byte of a word, the lower byte must contain a blank (20H).

uu - bits 12,11 - contain the number of 16-bit words necessary to define the userid associated with the file name. The userid is packed, two characters per word, starting in the word after the one containing the last byte of the file name. For example the userid HP is two characters which will fit into one 16-bit word; the number appearing in location uu will be 1. If the last character of the userid is in the upper byte of a word, the lower byte must contain a blank (20H). If the field uu contains zero, the current userid is used. To express the blank userid, the field uu contains one, and the both bytes of that word contain blanks (20H).

Word #	B15 B8	B7 0
0	fff uu ddd	ASCII byte 0
1	ASCII byte 1	ASCII byte 2
2	ASCII byte 3	ASCII byte 4
3	ASCII byte 5	ASCII byte 6
4	ASCII byte 7	ASCII byte 8
5	ASCII byte 9	ASCII byte 10
6	ASCII byte 11	ASCII byte 12
7	ASCII byte 13	ASCII byte 14

File Name Description Block

Figure B-1. File Name Description Block

ddd - bits 10,9,8 - contains the disc number where the source file corresponding to this relocatable file resides.

ASCII byte 0 - bits 7-0 - contain the first character of the file name. (All file names are made up of at least one character.)

WORD 1-7 - The remaining characters in the file name and the characters making up the files' userid appear in words 1-7 of the Name record. The userid characters start in the word immediately following the word containing the last character of the file name. The words following the one in which the last character of the userid is stored are not used. The linker does not care what values these words hold.

The example described above file = START1:HP:0 would be stored as shown in Figure B-2.

0	011 01 000	S (53H)
1	T (54H)	A (41H)
2	R (52H)	T (54H)
3	1 (31H)	(20H)
4	H (48H)	P (50H)
5	xxxxxxxx	xxxxxxxx
6	xxxxxxxx	xxxxxxxx
7	xxxxxxxx	xxxxxxxx

Figure B-2. File Name Description Example

Appendix C

LINKER TABLE DESCRIPTION - FIXED LENGTH

WORD 0 - The contents of Word 0 (see Figure C-1) is broken down as follows:

fff - bits 15,14,13 - contain the number of 16-bit words necessary to define the file name of the linker. The file name is packed two characters per word with the first character stored in the last byte of word 1. This number is calculated as follows: Take the number of characters in the file name and subtract one. Divide this number by two and round up. For example, the file name I68000 consists of 6 ASCII characters. This would require 3 words in addition to word 1 in which to fit all the characters of the name. (6-1) /2 = 2.5 which rounds up to 3. If the last character of the file name is stored in the upper byte of a word, the lower byte must contain a blank (20H).

uu - bits 12,11 - contain the number of 16-bit words necessary to define the userid associated with the file name. The userid is packed, two characters per word, starting in the word after the one containing the last byte of the file name. For example the userid HP is two characters which will fit into one 16-bit word; the number appearing in location uu will be 1. If the last character of the userid is in the upper byte of a word, the lower byte must contain a blank (20H). If the field uu contains zero, the current userid is used. To express the blank userid, the field uu contains one, and the both bytes of that word contain blanks (20H).

Word #	B15 B8	B7 0
0	fff uu ddd	ASCII byte 0
1	ASCII byte 1	ASCII byte 2
2	ASCII byte 3	ASCII byte 4
3	ASCII byte 5	ASCII byte 6
4	ASCII byte 7	ASCII byte 8
5	ASCII byte 9	ASCII byte 10
6	ASCII byte 11	ASCII byte 12
7	ASCII byte 13	ASCII byte 14

Linker Table Description Block (Fixed length 8 words)

Figure C-1. Linker Table Description Block

ddd - bits 10,9,8 - contains the disc number where the source file corresponding to this relocatable file resides.

ASCII byte 1 - bits 7-0 - contain the first character of the file name. (All file names are made up of at least one character.)

WORD 1 through 7 - The remaining characters in the file name and the characters making up the files' userid appear in words 1-7 of the Name record. The userid characters start in the word immediately follow the word containing the last character of the file name. The words following the one in which the last character of the userid is stored are not used. The linker does not care what values the words hold.

The standard linker table names are:

11000	18048	18088	1Z8000
11802	18051	19980	1LINK (used to link
1650X	18080	199XX	the User-Definable
168000	18085_Z80	1F8	Linker)
168XX	18086	1Z8	

Other linker table names may have been defined using the User-Definable Assembler/Linker, and those names will also appear in this record.

The example described above file = 168000:HP:1 would be stored as:

0	011 01 001	1 (6CH)
1	6 (36H)	8 (38H)
2	0 (30H)	0 (30H)
3	0 (30H)	0 (30H)
4	H (48H)	P (50H)
5	xxxxxxxx	xxxxxxxx
6	xxxxxxxx	xxxxxxxx
7	xxxxxxxx	xxxxxxxx

Figure C-2. Linker Table Description Example

Appendix D

SYMBOL NAME DESCRIPTION - VARIABLE LENGTH

WORD 0 - The contents of Word 0 (see Figure D-1) is broken down as follows:

sss - bits 15,14,13 - contain the number of 16-bit words necessary to define the symbol name. The symbol name is packed two characters per word, with the first character stored in the last byte of word 1. This number is calculated as follows: Take the number of characters in the symbol name and subtract one. Divide this number by two and round up. For example, the symbol name ASSEMBLER1 consists of 10 ASCII characters. This would require 5 words in addition to word 1 to fit all the characters of the name . (10-1) /2 = 4.5, which rounds up to 5. If the last character of the symbol name is stored in the upper byte of a word, the lower byte must contain a blank (20H).

Word #	B15 B8	B7 0
0	sss 00d rr	ASCII byte 0
1	ASCII byte 1	ASCII byte 2
2	ASCII byte 3	ASCII byte 4
3	ASCII byte 5	ASCII byte 6
4	ASCII byte 7	ASCII byte 8
5	ASCII byte 9	ASCII byte 10
6	ASCII byte 11	ASCII byte 12
7	ASCII byte 13	ASCII byte 14

Symbol Name Description Block (Variable Length) Max = 8 Words

Figure D-1. Symbol Name Description Block

d - This bit is 0 in all file types except link__sym. In a link__sym file, this bit indicates whether the symbol was defined. A 1 in this bit indicated that the reference to this symbol was not resolved.

rr -bits 9,8 -contain a code describing which relocation counter to use in relocating this symbol.

00 = absolute (no relocation)
01 = PROG
10 = DATA
11 = COMN

ASCII byte 0 - bits 7-0 - contains the first character of the symbol name. (All symbol names are made up of at least one character.)

Up to WORD sss+1 - Additional characters in symbol name. Note that for a one character symbol name, sss = 0, so there are no additional words used.

The example described above symbol = ASSEMBLER relocatable wrt DATA counter would be stored as shown in Figure D-2.

0	101 000 10	A (41H)
1	S (53H)	S (53H)
2	E (45H)	M (4DH)
3	B (42H)	L (4CH)
4	E (45H)	R (52H)
5	1 (31H)	(20H)

Figure D-2. Symbol Name Description Example

Appendix E

MEMORY SPACE RECORD - SOURCE NAME DESCRIPTION - FIXED LENGTH

WORD 1 - The contents of Word 1 (see Figure D-1) is broken down as follows:

fff - bits 15,14,13 - contain the number of 16-bit words necessary to define the file name. The file name is packed two characters per word, with the first character stored in the last byte of word 1. This number is calculated as follows: Take the number of characters in the file name and subtract one. Divide this number by two and round up. For example, the file name START1 consists of 6 ASCII characters. This would require 3 words in addition to word 1 in which to fit all the characters of the name. (6-1) /2 = 2.5 which rounds up to 3. If the last character of the file name is stored in the upper byte of a word, the lower byte must contain a blank (20H).

uu - bits 12,11 - contain the number of 16-bit words necessary to define the userid associated with the file name. The userid is packed, two characters per word, starting in the word after the one containing the last byte of the file name. For example the userid HP is two characters which will fit into one 16-bit word; the number appearing in location uu will be 1. If the last character of the userid is in the upper byte of a word, the lower byte must contain a blank (20H). If the field uu contains zero, the current userid is used. To express the blank userid, the field uu contains one, and the both bytes of that word contain blanks (20H).

Word #	B15 B8	B7 0
0	fff uu 0 m n	ASCII byte 0
1	ASCII byte 1	ASCII byte 2
2	ASCII byte 3	ASCII byte 4
3	ASCII byte 5	ASCII byte 6
4	ASCII byte 7	ASCII byte 8
5	ASCII byte 9	ASCII byte 10
6	ASCII byte 11	ASCII byte 12
7	ASCII byte 13	ASCII byte 14

Source Name Description Block

Figure E-1. Memory Space Record - Source Name Description Block

0 - bit 10 - is reserved for future use by the 64000 system and should contain 0's.

m - bit 9 - contains a 1 if and only if multiple relocatables are collected under one File Name (as for a library) or if the Program Name Definition Block for the relocatable in the link_sym file is not the same name as the Source file from which the relocatable was obtained. Otherwise, this bit contains a 0.

n - bit 8 - contains a 1 if and only if this file was no-loaded. Otherwise, this bit contains a 0.

ASCII byte 0 - bits 7-0 - contain the first character of the file name. (All file names are made up of at least one character.)

WORD 1-7 - The remaining characters in the file name and the characters making up the files' userid appear in words 1-7 of the Name record. The userid characters start in the word immediately following the word containing the last character of the file name. The words following the one in which the last character of the userid is stored are not used. The linker does not care what values these words hold.

The example described above file = START 1:HP would be stored as:

0	011 01 000	S (53H)
1	T (54H)	A (41H)
2	R (52H)	T (54H)
3	1 (31H)	(20H)
4	H (48H)	P (50H)
5	xxxxxxxx	xxxxxxxx
6	xxxxxxxx	xxxxxxxx
7	xxxxxxxx	xxxxxxxx

Figure E-2. Memory Space Record - Source Name Description Example

64980-90933, JULY 1984
Replaces: 64980-90933, May 1984

HEWLETT
PACKARD

PRINTED IN U.S.A.