RTE-IV Assembler

Reference Manual

HEWLETT
PACKARD

HEWLETT-PACKARD COMPANY
Data Systems Division
11000 Wolfe Road
Cupertino, California 95014

MANUAL PART NO. 92067-90003
Printed in U.S.A. October 1980

PRINTING HISTORY

The Printing History below identifies the Edition of this Manual and any Updates that are included. Periodically, Update packages are distributed which contain replacement pages to be merged into the manual, including an updated copy of this Printing History page. Also, the update may contain write-in instructions.

Each reprinting of this manual will incorporate all past Updates, however, no new information will be added. Thus, the reprinted copy will be identical in content to prior printings of the same edition with its user-inserted update information. New editions of this manual will contain new information, as well as all Updates.

To determine what manual edition and update is compatible with your current software revision code, refer to the appropriate Software Numbering Catalog, Software Product Catalog, or Diagnostic Configurator Manual.

Third Edition ..... Jan 1980
Change 1 ..... Apr 1980
Reprinted (Change 1 incorporated) ..... Apr 1980
Change 2 ..... Oct 1980
Reprinted (Change 2 incorporated) ..... Oct 1980

HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

PREFACE

This manual describes the Assembler which is designed to operate under the control of HP 1000 RTE based Operating Systems.

This manual assumes that the reader is an experienced assembly language programmer who is familiar with operating systems and computer instruction sets.

HOW TO USE THIS MANUAL

The Assembler is a common program that executes under various HP Operating Systems and under various machine instruction sets. This manual describes the Assembler in its totality. Therefore, the user should keep in mind what operating system he is using and what machine the resulting object code is to execute on.

All users should read Section I and II.

Section I attempts to guide the user to the proper machine instruction set(s). It also discusses the assembly process in general, program relocation, assembly options, and assembler input and output.

Section II describes the source statement format that the Assembler accepts as input.

Section III describes all of the available machine instructions. It should be noted that while the Assembler will correctly assemble these instructions, the resulting object code will correctly execute only on the intended hardware. The above situation occurs because various machine's instruction sets are subsets of the total instruction set that the Assembler accepts. The user is encouraged to use the appropriate hardware manual in conjunction with this manual to ensure that the assembly code written will execute on the hardware that it is intended for. See Section I for details.

Section IV describes all of the available assembler pseudo instructions.

Throughout the manual, gray shaded areas indicate sections that are intended for specific CPU hardware. The intention will be noted in the shaded area.

In addition, nine appendices are supplied, as follows:

Appendix A	describes the Hewlett-Packard character set.
Appendix B	summarizes all of the available machine and pseudo instructions (including instruction formats).
Appendix C	presents a one-sentence definition of all available machine and pseudo instructions, arranged alphabetically by mnemonic.
Appendix D	presents a tabular summary of the binary format of all available machine instructions.
Appendix E	describes how to run an assembly.
Appendix F	describes the valid instruction set for the M, E, F and L-SERIES computers.
Appendix G	lists and describes all of the assembler error messages.
Appendix H	presents output data formats.
Appendix I	discusses the RTE Cross Reference Table Generator.

CONTENTS

CONTENTS (continued)

ILLUSTRATIONS

TABLES

INTRODUCING THE ASSEMBLER

SECTION I

The Assembler permits the programmer to use all supported machine instructions for HP 1000 Computers and it is assumed that object programs produced by the Assembler will be executed on an HP 1000 Computer.

The Assembler translates symbolic source language instructions into an object program for execution on the computer. The source language provides mnemonic machine operation codes, assembler-directing pseudo instructions, and symbolic addressing. The assembled program may be absolute or relocatable.

The source program may be assembled as a complete entity or it may be subdivided into several relocatable subprograms (or a main program and several subroutines), each of which may be assembled separately. When relocatable object programs and subprograms are desired to be executed, they are relocated and linked to one another by the relocating loader.

Absolute object programs may be loaded by the Bootstrap Loader. There are no intermediate steps needed to prepare the code before it is executed.

The Assembler can read the source input from a disc file or an input device. The Assembler outputs the resultant relocatable or absolute object program to a disc file or an output device.

If the object programs produced by the Assembler are relocated and executed under control of an operating system other than the RTE Operating System, the following restrictions apply:

ENT pseudo instructions with absolute or common symbols as operands must *not* be used.

I/O instructions using externally-defined select codes must *not* be used.

I/O select codes must *not* be defined via the ENT pseudo instruction.

Memory reference instructions must *not* refer to *external* symbols with *offset* values.

1-1. HP 1000 L-SERIES SYSTEMS

The L-SERIES instruction set is a subset of the HP 1000 instruction set. If the programmer is using L-SERIES hardware and its associated operating system, refer him to Appendix F and the HP 1000 L-SERIES REFERENCE MANUAL for a guide to the valid instruction set. The particular features of using the Assembler on an L-SERIES system are documented throughout this manual. The Assembler will correctly assemble any HP 1000 machine instruction. Therefore it is important that the user of an L-SERIES computer refer to the above Appendix and manual so that the proper instruction set is used for assembly programs destined to be executed on L-SERIES hardware.

1-2. ASSEMBLY PROCESSING

The Assembler is a two pass system. A *pass* is defined as a processing cycle of the source program input.

In the first pass, the Assembler creates a symbol table from the names used in the source statements and (if requested) prints a symbol table listing on the standard list output device. It also checks for certain possible error conditions and prints error messages on the console device if necessary.

During pass two, the Assembler again examines each statement in the source program along with the symbol table and produces the binary object program. It outputs the object program to an output device or a disc file. If requested, the Assembler also outputs the source program listing to a list output device or a list file. Additional error messages may also be printed on the system console device.

If the source input is being read from a non-disc device, it is written on the disc at the start of pass 1; for pass 2, the source is then read from the disc. However, if there is not sufficient space available on the disc to do this, the Assembler will be suspended until enough disc space is available.

1-3. SYMBOLIC ADDRESSING

Symbols may be used for referring to machine instructions, data, constants, and certain other pseudo operations. A symbol represents the address for a computer word in memory. A symbol is defined when it is used as a label for a location in the program, a name of a common storage segment, the label of a data storage area or constant, the label of an absolute or relocatable value, or a location external to the program.

Through use of simple arithmetic operators, symbols may be combined with other symbols or numbers to form an expression which may identify a location other than that specifically named by a symbol. Symbols appearing in operand expressions, but not specifically defined, and symbols that are defined more than once are considered to be an error by the Assembler.

1-4. MEMORY ADDRESSING

1-5. PAGING

The computer memory is logically divided into pages of 1024 words each. A page is defined as the largest block of memory which can be addressed directly by the memory address bits of a memory reference instruction (single-length). These memory reference instructions have 10 bits to specify a memory address, and thus the page size is 1024 locations (2000 octal). Octal addresses for each page, up to the maximum memory size, are shown in table 1-1.

Provision is made to address directly one of two pages: page zero (the base page, consisting of locations 00000_8 through 01777_8), and the current page (the page in which the instruction itself is located). Memory reference instructions include a bit (bit 10) reserved to specify one or the other of these two pages. To address locations in any other page, indirect addressing is used. Page references are specified by bit 10 as follows:

Logic 0 = page zero (Z)
Logic 1 = current page (C)

1-6. INDIRECT ADDRESSING

All memory reference instructions reserve a bit to specify direct or indirect addressing. For single-length memory reference instructions, bit 15 of the instruction word is used; for extended arithmetic memory reference instructions, bit 15 of the address word is used. Indirect addressing uses the address part of the instruction to access another word in memory, which is taken as a new memory reference for the same instruction. This new address word is a full 16 bits long, 15 bits of address plus another direct-indirect bit. The 15-bit length of the address permits access to any location in memory. If bit 15 again specifies indirect addressing, still another address is obtained. This multiple-step indirect addressing may be done to any number of levels. The first address obtained in the indirect phase which does not specify another indirect level becomes the effective address for the instruction. Direct or indirect addressing is specified by bit 15 as follows:

Logic 0 = direct
Logic 1 = indirect

1-7. PROGRAM RELOCATION

Relocatable programs are relocated at absolute addresses by the relocating loader.

Relocatable code assumes a starting location of 00000, and this location is termed the relative, or *relocatable* origin. The absolute origin (termed the relocation base) of a relocatable program is determined by the loader. The value of the absolute origin is added to the zero-relative value of each operand address to obtain the absolute operand address. The absolute origin, and thus the values of every operand address, may vary each time the program is loaded.

A relocatable program may be composed of several independently assembled or compiled subprograms. Each of the subprograms will have a relative origin of 00000. Each subprogram is then assigned a unique absolute origin upon being loaded.

The operand values produced by the Assembler may be *program* relocatable, *base page* relocatable, or *common* relocatable. Each of these segments of the program has a separate relocation base or origin. Operands that are references to locations in the main portion of the program are incremented by the program relocation base; those referring to the base page, by the base page relocation base; and those referring to common storage, by the common relocation base.

If the loader or system generator encounters an operand that is a reference to a location in a page other than the current page or base page, a link is established. A link is a word in the base page or current page which is allocated to contain the full 15-bit address of the referenced location. The address of the link is then substituted as an indirect address in the instruction in the current page. If other similar references are made to the same location, they are linked through the same link.

1-8. PROGRAM LOCATION COUNTER

The Assembler maintains a counter, called the program location counter, that assigns consecutive memory addresses to source statements.

The initial value of the program location counter is established according to the use of either the NAM or ORG pseudo operation at the start of the program. The NAM operation causes the program location counter to be set to zero for a relocatable program; the ORG operation specifies the absolute starting location for an absolute program.

Through use of the ORB pseudo operation a relocatable program may specify that certain operations or data areas be allocated to the base page. If so, a separate counter, called the base page location counter, is used in assigning these locations.

1-9. SOURCE PROGRAM

Figure 1-1 shows an assembler coding form and the code for a simple program which counts the number of 1's and 0's in the A-register. The first statement is the control statement, which in this example contains the assembly options R (for a relocatable source program), L (a program listing is to be output to the list file), and T (a listing of the symbol table is to be output to the list file). See paragraph 1-9 and Table 1-2 for a further discussion of control statement parameters.

Table 1-1. Logical Memory Address/Pages

MEMORY SIZE	PAGE	OCTAL ADDRESSES
4K	0	00000 to 01777
	1	02000 to 03777
	2	04000 to 05777
	3	06000 to 07777
8K	4	10000 to 11777
	5	12000 to 13777
	6	14000 to 15777
	7	16000 to 17777
12K	8	20000 to 21777
	9	22000 to 23777
	10	24000 to 25777
	11	26000 to 27777
16K	12	30000 to 31777
	13	32000 to 33777
	14	34000 to 35777
	15	36000 to 37777
24K	16	40000 to 41777
	17	42000 to 43777
	18	44000 to 45777
	19	46000 to 47777
	20	50000 to 51777
	21	52000 to 53777
	22	54000 to 55777
	23	56000 to 57777
32K	24	60000 to 61777
	25	62000 to 63777
	26	64000 to 65777
	27	66000 to 67777
	28	70000 to 71777
	29	72000 to 73777
	30	74000 to 75777
	31	76000 to 77777

Following the control statement, the first statement of the program (other than remarks or a HED statement) must be a NAM statement for a relocatable program or an ORG statement to indicate the origin of an absolute program. The last statement must be an END statement and may contain a transfer address for the start of a relocatable program. Each statement is terminated by an end-of-statement or end-of-record mark if not on cards.

1-10. ASSEMBLY OPTIONS

The control statement must be the first statement in the source program and it specifies the desired assembly options:

ASMB,p_1,p_2,. . . ,p_n

"ASMB," is in positions 1-5 of the statement. Following the comma are one or more parameters, in any order. The parameters are shown in Table 1-2. The parameters in the control statement may be overridden when the Assembler is invoked. See Appendix E for options available at Assembler run time.

Since they contradict one another, F and X must never appear in the control statement for the same source program. Similarly, A and R must never appear together. If neither A nor R is specified, R is assumed. If T is omitted, the symbol table listing will *not* be output to the list file. If L and Q are both specified, the one specified last will be used. If B is specified, it is ignored.

"ASMB" alone or with either A or R as the only option specified, will direct the Assembler to process the source information without producing any output. Error messages will be output to the list device or list file, however. Thus, the user may use this method to examine the source for errors prior to producing the final object code.

1-11. BINARY OUTPUT

The binary output is defined by the ASMB control statement. The binary output includes the object code for the instructions translated from the source program. It does not include system subroutines referenced within the source program (arithmetic subroutine calls, .IOC., .DIO., .ENTR, etc.). If a binary output file name or logical unit number is not specified in the run command for the Assembler, no binary output is produced.

1-12. SYMBOL TABLE

Figure 1-2 shows a sample symbol table listing produced when a source program was assembled. Columns 1 through 5 contain the name of the label. Column 7 specifies the type of relocation for the operand field, and columns 9 through 14 contain the value of the label. (In the example shown in figure 1-2, the locations are relative because the source program is relocatable.)

The characters that designate an external symbol or type of relocation for the Operand field or the symbol are as follows:

Character	Relocation Base
Blank	Absolute
R	Program relocatable
C	Common relocatable
X	External symbol
B	Base page relocatable
S	Substitution code
E	Extended Memory Area

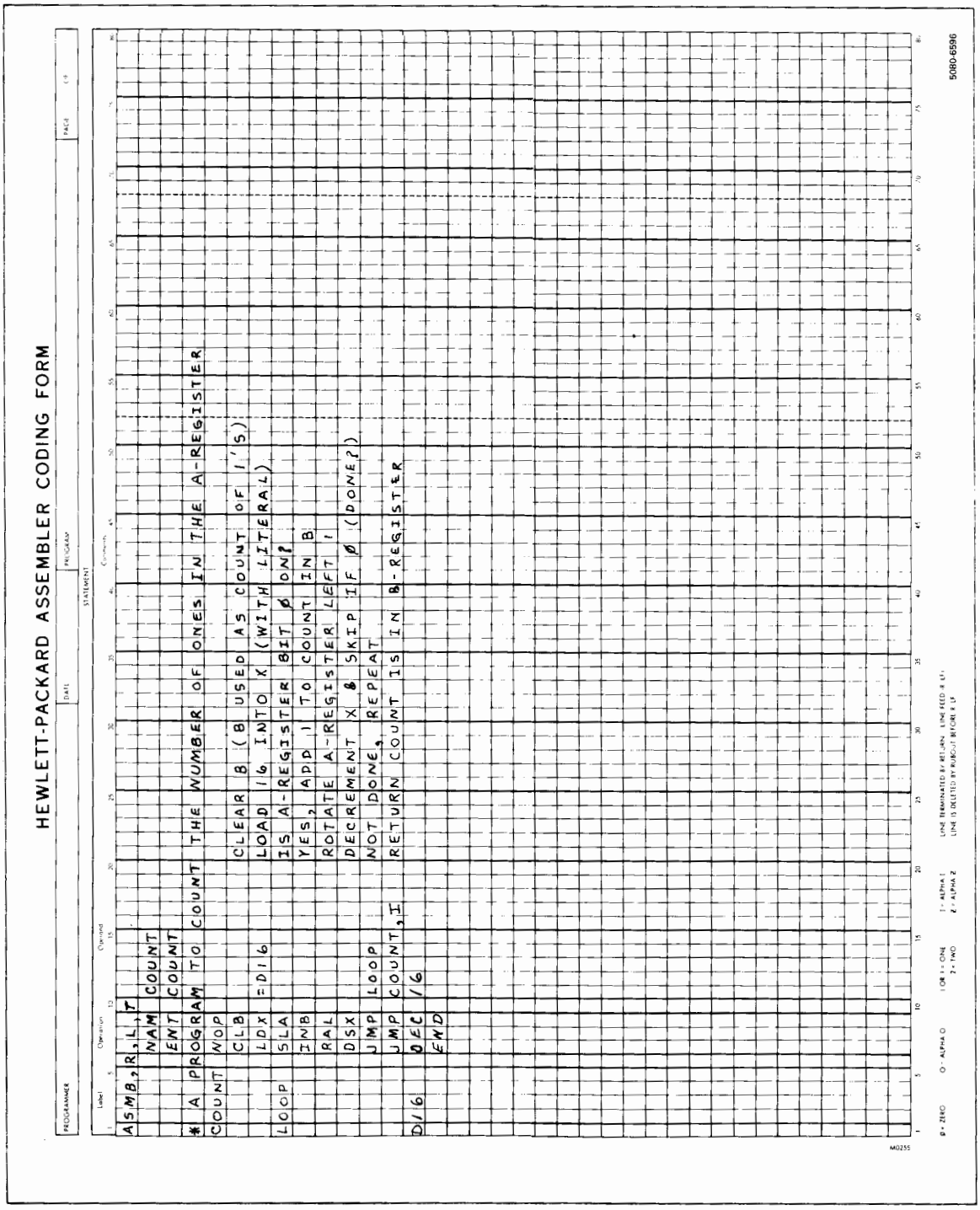


Figure 1-1. Source Program

Table 1-2. Control Statement Parameters

PARAMETER	MEANING
A	Absolute assembly. The addresses generated by the Assembler are to be interpreted as absolute locations in memory. The program is a complete entity; external symbols, common storage references and entry points are not permitted. Note that an absolute program *cannot* be executed on RTE.
R	Relocatable assembly. The object program may be loaded anywhere in memory. All operands which refer to memory locations are automatically adjusted as the program is loaded. Operands referring to memory locations greater than 1777_8 must be relocatable expressions. Programs may contain external symbols and entry points, and may refer to common storage.
L	List output. A program listing is to be output to the list file or list device. Columns 8-13 of the listing will contain the object code of the instruction. This includes both the opcode and the address of the operand if it is a memory reference instruction.
Q	List output. A program listing is to be output to the list file or list device. Columns 8-13 of the listing will contain only the operand address for single word memory reference instructions. The entire object code will be listed otherwise.
T	Symbol table print. A listing of the symbol table is to be printed on the standard list output device.
N,Z	Selective assembly. Sections of the program are to be included or excluded at assembly time depending upon the option specified. See the descriptions of the IFN and IFZ pseudo instructions in Section IV of this manual.
C	Cross reference table print. All references to statement labels, external symbols, and user-defined opcodes are to be listed on the standard list output device after the end of the assembly.
F	Floating point instructions. The floating point machine instructions are to be used instead of the software simulation routines for the following floating point operations: FDV, FMP, FAD, and FSB. Not applicable on L-Series hardware.
X	No EAU hardware. Signifies that the object program will be executed on a machine which does *not* have the Extended Arithmetic Unit (EAU) hardware. This parameter prevents the use of the following EAU instructions: ASR, ASL, RRR, RRL, LSR, LSL, and SWP. In addition, it causes all occurrences of the MPY, DIV, DLD, and DST instructions to be substituted with a call to the appropriate subroutine in the relocatable library.
P	Used as an override option when the assembler is invoked (see Appendix E). It has no effect when specified in the control statement of an assembly language program but is included here for completeness.
B	Ignored if specified.

```
 PAGE 0001 #01        1:30 PM  TUE.,  6  DEC., 1977

0001   R 000001       ASMB,R,L,T
COUNT  R 000005
BIT0   R 000010
BIT1   R 000013
BIT2   R 000016
MORE   R 000022
BIT3   R 000023
LESS1  R 000024
LESS2  R 000026
EVEN   R 000027
**   NO ERRORS PASS=1   **RTE ASMB 92067-16011**
```

Figure 1-2. Symbol Table Listing

1-13. LIST OUTPUT

Columns	Content
1-4	Source statement sequence number generated by the Assembler
5-6	Blank
7-11	Location (octal)
12	Blank
13-18	Object code word in octal
19	Relocation or external symbol indicator
20	Blank
21-80	First 60 characters of source statement

Lines consisting entirely of comments (i.e., * in column 1) are printed as follows:

Columns	Content
1-4	Source statement sequence number
5-80	Up to 76 characters of comments

At the end of each pass, the following is printed on the list device:

Pass 1 =
** NO ERRORS PASS#1 **RTE ASMB xxxxx-yyyyy**

or

nnnn ERRORS PASS#1 **RTE ASMB xxxxx-yyyyy

Pass 2 =
** NO ERRORS *TOTAL **RTE ASMB xxxxx-yyyyy**

or

nnnn ERRORS *TOTAL **RTE ASMB xxxxx-yyyyyy

The value *nnnn* indicates the number of errors. Pass 2 error count includes the total error count of pass 1 and pass 2. xxxxx-yyyyy is the Assembler's part number.

If there are errors, the message PG *xxx* is printed on the list device immediately preceding the **nnnn ERRORS* message, where *xxx* is the page number where the final error was detected. The same message appears in the listing following each error and it points to the page number where the previous error was detected. The backwards pointer following the first error in the program is PG 000.

A heading is printed by the Assembler at the top of every page of the list output. The heading consists of the page and tape number of the listing followed by the time of day. The HED pseudo-instruction may be used to print out a user-defined header in addition to the standard header.

SOURCE STATEMENT FORMAT

SECTION II

A source language statement consists of a label, an operation code, an operand (or operands) and comments. The label is used when needed as a reference by other statements. The operation code may be a mnemonic machine operation or an assembly directing pseudo code. An operand may be an expression consisting of an alphanumeric symbol, a number, a special character, or any of these combined by arithmetic operators. An operand may also be a literal. Indicators may be appended to an operand to specify certain functions such as indirect addressing. The comments portion of the statement is optional.

2-1. STATEMENT OF CHARACTERISTICS

The fields of the source statement appear in the following order:

1. Label
2. Opcode
3. Operands
4. Comments

2-2. FIELD DELIMITERS

One or more spaces separate the fields of a statement. A single space as the first character of a statement signifies that there is no label for this statement.

2-3. CHARACTER SET

The characters that may appear in a statement are as follows:

A through Z
0 through 9
. (period)
* (asterisk)
+ (plus)
− (minus)
, (comma)
= (equals)
() (parentheses)
(space)

Any other ASCII characters may appear in the Comments field. (See Appendix A.)

The letters A through Z, the numbers 0 through 9, and the period may be used in an alphanumeric symbol. In the first position in the Label field, an asterisk indicates a comment; in the Operand field, it represents the value of the program location counter for the current instruction. The plus and minus are used as operators in arithmetic address expressions. The comma separates several operation codes, or an expression and an indicator in the Operand field. An equals sign indicates a literal value. The parentheses are used only in the COM pseudo instruction.

Spaces separate fields of a statement and operands in a multi-operand field. They may also be used to enhance the appearance of the listing. Within a field they may be used freely when following +, −, ,, or (.

2-4. STATEMENT LENGTH

A statement may contain up to 80 characters including blanks, but excluding the end-of-statement mark.

2-5. LABEL FIELD

The Label field identifies the statement and may be used as a reference point by other statements in the program.

The field starts in position one of the statement. It is terminated by a space. A space in position one signifies that the statement is unlabeled.

2-6. LABEL SYMBOL

A label may have one to five characters consisting of A through Z, 0 through 9, and the period.

Note: The Assembler allows the use of certain other characters in the Label field. However, they are reserved for use in Hewlett-Packard programs.

The first character must be alphabetic or a period. A label of more than five characters could be entered on the source statement, but the Assembler flags this condition as an error and truncates the label from the right to five characters. Some examples are shown in figure 2-1.

Each label must be unique within the program; two or more statements may not have the same symbolic name. Names which appear in the Operand field of an EXT or COM pseudo instruction may not also be used as statement labels in the same subprogram. However, names

appearing in a COM pseudo instruction may be defined as entry points in an ENT pseudo instruction. Some examples are shown in figure 2-2.

2-7. ASTERISK

An asterisk in position one indicates that the entire statement is a comment. Positions 2 through 80 are available; however, positions 1 through 76 only are printed as part of the assembly listing. An asterisk within a label is illegal in any position.

2-8. OPCODE FIELD

The operation code defines an operation to be performed by the computer or the Assembler. The Opcode field follows the Label field and is separated from it by at least one space. If there is no label, the operation code may begin anywhere after position one. The Opcode field is terminated by a space immediately following an operation code. Operation codes are organized in the following categories:

Machine operation codes:

- Memory Reference
- Register Reference
- Input/Output, Overflow, and Halt
- Extended Arithmetic Unit
- **(M, E and F-Series)**
- **Floating Point**
- **Memory Mapping**
- Decimal Arithmetic

Pseudo operation codes:

- Assembler control
- Object program linkage
- Address and symbol definition
- Constant definition
- Storage allocation
- RTE-L Pseudo Instructions
- Arithmetic subroutine calls
- Assembly Listing Control
- Define User Opcodes
- Code-replacement definition

Operation codes are discussed in detail in Sections III and IV.

2-9. OPERAND FIELD

The meaning and format of the Operand field depend on the type of operation code used in the source statement. The field follows the Opcode field and is separated from it by at least one space. If more than one operand is required, they are separated from one another by at least one space.

An Operand may contain an expression consisting of one of the following:

- Single symbolic term
- Single numeric term
- Asterisk
- Combination of symbolic terms, numeric terms, and the asterisk joined by the arithmetic operators + and −

An expression may be followed by a comma, an indirect addressing indicator (see paragraph 2-20), and a Clear Flag indicator (see paragraph 2-21). Programs may also contain a literal value in the Operand field. (See paragraph 2-19.)

2-10. SYMBOLIC TERMS

A symbolic term may be one to five characters consisting of A through Z, 0 through 9, and the period. The first character must be alphabetic or a period. Some examples are shown in figure 2-3.

A symbol used in the Operand field must be a symbol that is defined elsewhere in the program in one of the following ways.

- As a label in the Label field of a machine operation or a user-defined instruction
- As a label in the Label field of a BSS, EMA, ASC, DEC, DEX, OCT, DEF, BYT, ABS, EQU, DBL, DBR or REP pseudo operation
- As a name in the Operand field of a COM or EXT pseudo operation
- As a label in the Label field of an arithmetic subroutine pseudo operation

The value of a symbol is absolute or relocatable depending on the assembly option selected by the user. The Assembler assigns a value to a symbol as it appears in one of the above fields of a statement. If a program is to be loaded in absolute form, the values assigned by the Assembler remain fixed. If the program is to be relocated, the actual value of a symbol is established on loading. A symbol may be assigned an absolute value through use of the EQU pseudo instruction.

A symbolic term may be preceded by a plus or minus sign. If preceded by a plus or no sign, the symbol refers to its associated value. If preceded by a minus sign, the symbol refers to the two's complement of its associated value. A single negative symbolic operand may be used only with the ABS pseudo operation.

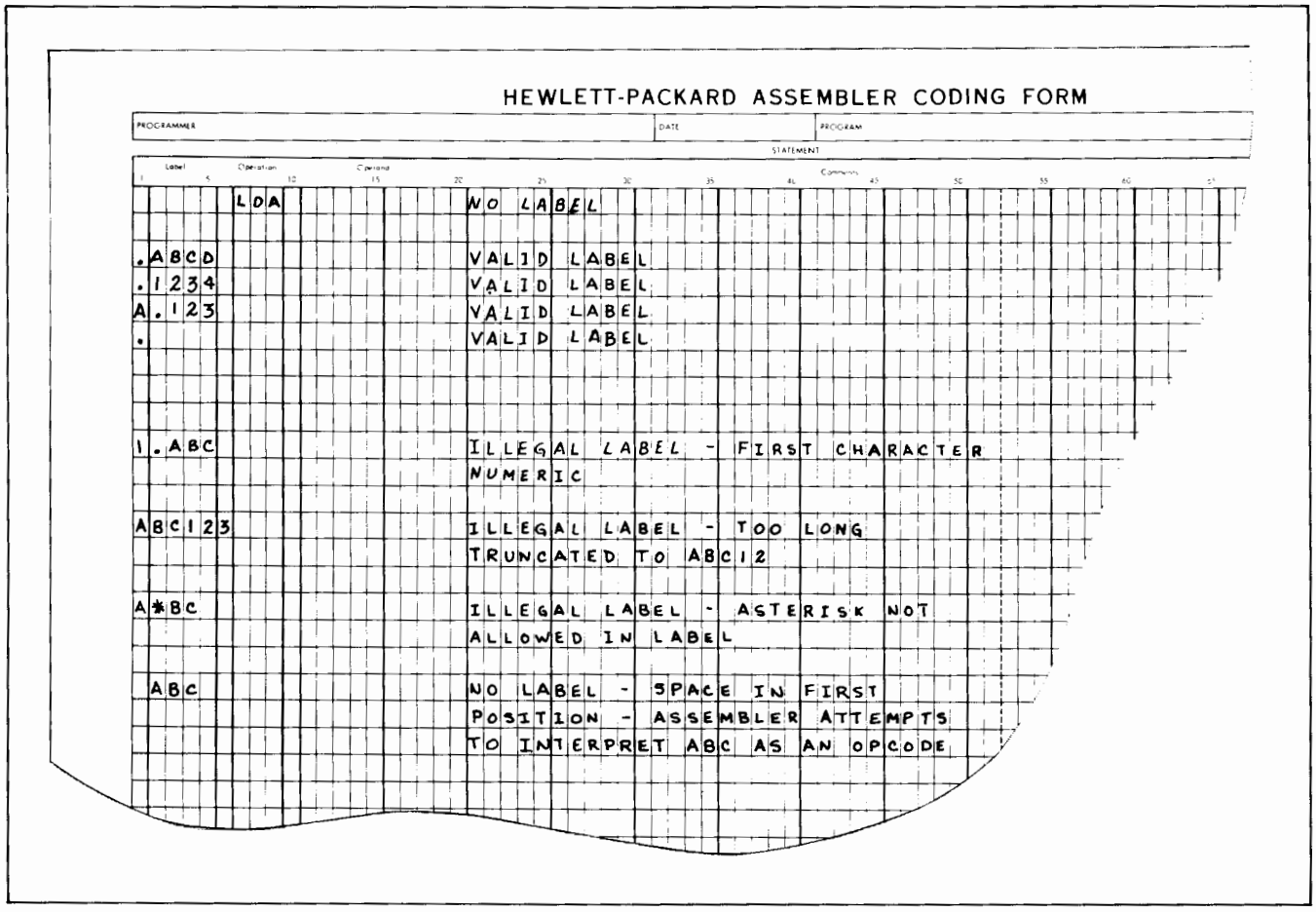


Figure 2-1. Label Examples

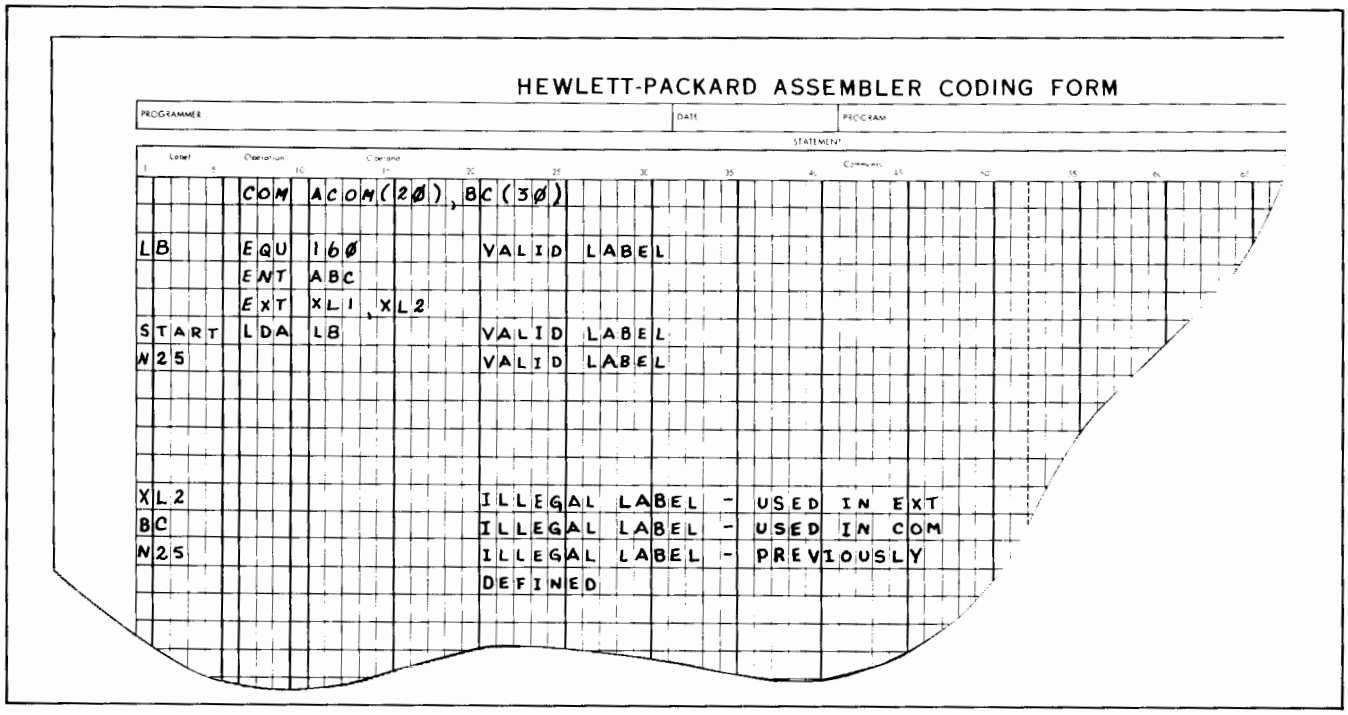


Figure 2-2. Label Usage Examples

2-11. NUMERIC TERMS

A numeric term may be decimal or octal. A decimal number is represented by one to five digits within the range 0 to 32767. An octal number is represented by one to six octal digits followed by the letter B (0 to 177777B).

If a numeric term is preceded by a plus or no sign, the binary equivalent of the number is used in the object code. If preceded by a minus sign, the two's complement of the binary equivalent is used. A negative numeric operand may be used only with the DEX, DEC, OCT, BYT and ABS pseudo operations.

For a memory reference instruction in an absolute program, the maximum value of a numeric operand depends on the type of machine or pseudo instruction. In a relocatable program, the value of a numeric operand may not exceed 1777_8. Numeric operands are absolute. Their value is not altered by the assembler or the loader.

2-12. ASTERISK

An asterisk in the Operand field refers to the value in the program location counter at the time the source program statement is encountered. The asterisk is considered a relocatable term in a relocatable program.

2-13. EXPRESSION OPERATORS

The asterisk, symbols, and numbers may be joined by the arithmetic operators + and − to form arithmetic address expressions. The Assembler evaluates an expression and produces an absolute or relocatable value in the object code. Some examples are shown in figure 2-4.

2-14. EVALUATION OF EXPRESSIONS

An expression consisting of more than one operand is reduced to a single value. In expressions containing more than one operator, evaluation of the expression proceeds from left to right. The algebraic expression A−(B−C+5) must be represented in the Operand field as A−B+C−5. Parentheses are not permitted in operand expressions.

The range of values that may result from an operand expression depends on the type of operation. The Assembler evaluates expressions as follows:†

Pseudo Operations:	
2 or 3-word Memory Reference:	modulo $2^{15}-1$
1-word Memory Reference:	modulo $2^{10}-1$
Input/Output:	$2^6 - 1$ (maximum value)

†The evaluation of expressions by the Assembler is compatible with the addressing capability of the hardware instructions (e.g., up to 32K words through Indirect Addressing). The user must take care not to create addresses which exceed the memory size of the particular configuration.

2-15. EXPRESSION TERMS

The terms of an expression are the numbers and the symbols appearing in it. Decimal and octal integers, and symbols defined as being absolute in an EQU pseudo operation are absolute terms. The asterisk and all symbols that are defined in the program are relocatable or absolute depending on the type of assembly. (RTE Assembler allows externals with offset and indirect external references.)

Within a relocatable program, terms may be program relocatable or common relocatable or base page relocatable. A symbol that names an area of common storage is a common relocatable term. A symbol that is defined in any statement other than COM or EQU is a relocatable term. Within one expression all relocatable terms must be program relocatable, common relocatable or base page relocatable; the types may not be mixed.

2-16. ABSOLUTE AND RELOCATABLE EXPRESSIONS

An expression is absolute if its value is unaffected by program relocation. An expression is relocatable if its value changes according to the location into which the program is loaded. In an absolute program, all expressions are absolute. In a relocatable program, an expression may be program relocatable, common relocatable, base page relocatable, or absolute (if less than 2000_8) depending on the definition of the terms composing it.

2-17. ABSOLUTE EXPRESSIONS

An absolute expression may be any arithmetic combination of absolute terms. It may contain relocatable terms alone, or in combination with absolute terms. If relocatable terms appear, there must be an even number of them; they must be of the same type; and they must be paired by sign (a negative term for each positive term). The paired terms do not have to be contiguous in the expression. The pairing of terms by type cancels the effect of relocation; the value represented by a pair remains constant.

An absolute expression reduces to a single absolute value. The value of an absolute multi-term expression may be negative only for ABS pseudo operations. A single numeric term also may be negative in an OCT, DEX, BYT, or DEC pseudo instruction. In a relocatable program the value of an absolute expression must be less than 2000_8 for instructions that reference memory locations (Memory Reference, DEF, Arithmetic subroutine calls, etc.).

If P_1 and P_2 are program relocatable terms; C_1 and C_2, common relocatable; and A, an absolute term; then the following are absolute terms:

$A - C_1 + C_2$	$A - P_1 + P_2$	$C_1 - C_2 + A$
$A + A$	$P_1 - P_2$	$-C_1 + C_2 + A$
$* - P_1$	$-P_1 + P_2$	$-A - P_1 + P_2$

The asterisk is program relocatable.

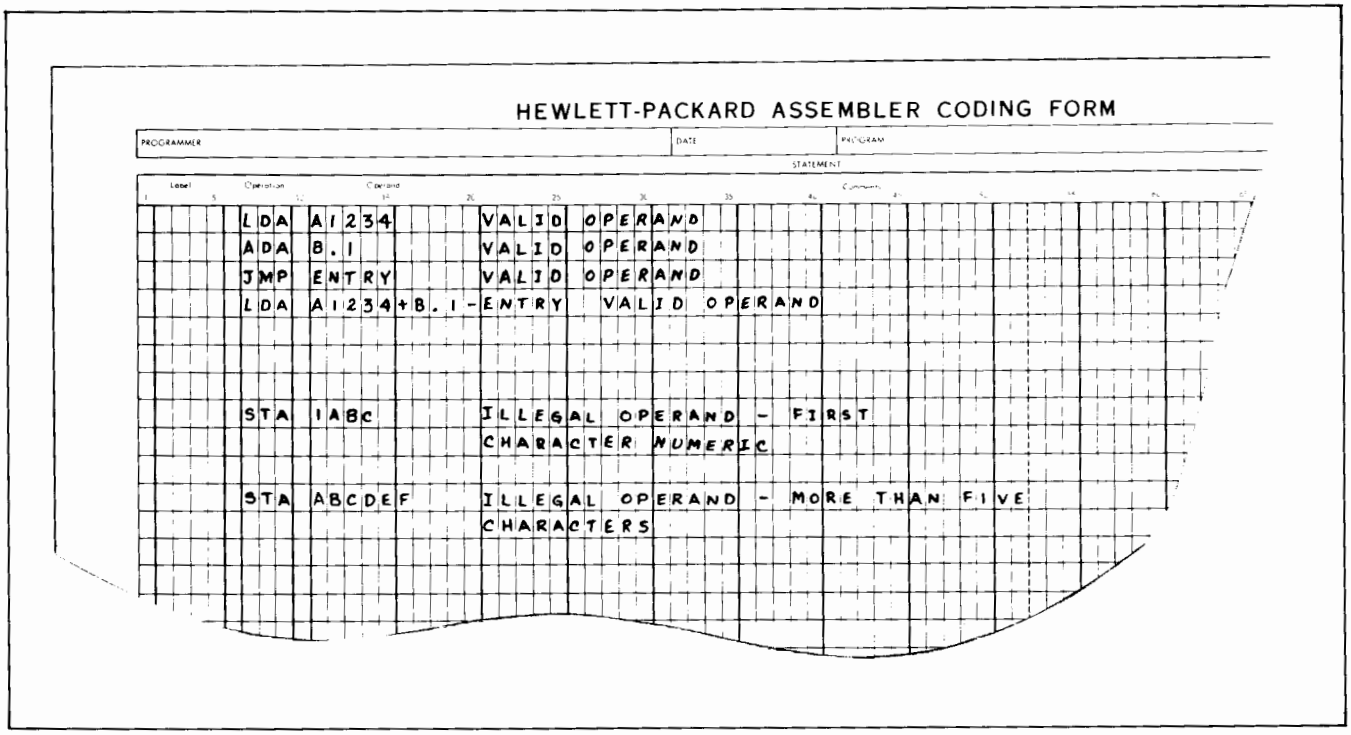


Figure 2-3. Symbolic Operand Examples

```
LDA SYM+6       ADD 6 TO THE VALUE OF SYM
ADA SYM-3       SUBTRACT 3 FROM THE VALUE OF SYM
  .
  .
  .
JMP *+5         ADD 5 TO THE CONTENTS OF THE
  .             PROGRAM LOCATION COUNTER.
  .
  .
STB -A+C-4      ADD - VALUE OF A, THE VALUE OF C
  .             AND SUBTRACT 4.
  .
  .
STA XTA-*       SUBTRACT VALUE OF PROGRAM
                LOCATION COUNTER FROM VALUE OF
                XTA.
```

Figure 2-4. Expression Operator Examples

2-18. RELOCATABLE EXPRESSIONS

A relocatable expression is one whose value is changed by the loader. All relocatable expressions must have a positive value.

A relocatable expression may contain an odd number of relocatable terms, alone, or in combination with absolute terms. All relocatable terms must be of the same type. Terms must be paired by sign with the odd term being positive.

A relocatable expression reduces to a single positive relocatable term, adjusted by the values represented by the absolute terms and paired relocatable terms associated with it.

If P_1, P_2, and P_3 are program relocatable terms; C_1, C_2 and C_3, common relocatable; and A, an absolute term; then the following are relocatable terms:

$P_1 - A$	$C_1 - A$	$P_1 - P_2 + *$
$P_1 - P_2 + P_3$	$C_1 - C_2 + C_3$	$C_1 + A$
$* + A$	$* - P_1 + P_2$	$* - A$
$P_2 + A$	$A + C_1$	$- A - P_1 + P_2 + P_3$
	$C_1 - C_2 + C_3 - A$	$A + *$
		$- C_1 + C_2 + C_3$

2-19. LITERALS

Literal values may be specified as operands in relocatable programs. (Literals are not allowed in absolute programs.) The Assembler converts the literal to its binary value, assigns an address to it, and substitutes this address as the operand. Locations assigned to literals are those immediately following the last location used by the program.

A literal is specified by using an equal sign and a one-character identifier defining the type of literal. The actual literal value is specified immediately following this identifier; no spaces may intervene.

The identifiers are:

=D a decimal integer, in the range −32767 to 32767, including zero.†

=F a floating point number; any positive or negative real number in the range 10^{-38} to 10^{38}, including zero.†

=B an octal integer, one to six digits, $b_1b_2b_3b_4b_5b_6$, where b_1 may be 0 or 1, and $b_2 - b_6$ may be 0 to 7.†

=A two ASCII characters.†

=L an expression which, when evaluated, will result in an absolute value. All symbols appearing in the expression must be previously defined.

†See CONSTANT DEFINITION, Section IV.

If the same literal is used in more than one instruction or if different literals have the same value (e.g., =B100 and =D64), only one value is generated, and all instructions using these literals refer to the same location.

Literals may be specified only in the following memory reference, register reference, EAU, and pseudo instructions:

(M, E and F-Series)

ADA	CPA	MBT	may use =D, =B, =A, =L
ADB	CPB	JRS	
ADX	DIV	MPY	
ADY	IOR	MVW	
AND	LDA	SBS	
CBS	LDB	TBS	
CBT	LDX	XOR	
CMW	LDY		
DLD	FDV	FSB	may use =F
FMP	FAD		

Examples are as follows:

LDA	=D7980	A-Register is loaded with the binary equivalent of 7980_{10}.
IOR	=B777	Inclusive OR is performed with contents of A-Register and 777_8.
LDA	=ANO	A-Register is loaded with binary representation of ASCII characters NO.
LDB	=LZETZ-ZOOM+68	B-Register is loaded with the absolute value resulting from the expression.

(M, E, and F-Series)

FMP	=F39.75	Contents of A- and B-Registers multiplied by floating point constant 39.75.

(L-Series)

```
        JSB   .FMP
        DEF   LIT
        DEC   0
              .
              .
              .
LIT     DEC   39.75
```

Jump to software simulation routine (see Section 3-28). A- and B-Registers multiplied by floating point constant 39.75.

2-20. INDIRECT ADDRESSING

The HP computers provide an indirect addressing capability for memory reference instructions. The operand portion of an indirect instruction contains the address of another location. The secondary location may be the operand or it may be indirect also and give yet another location, and so forth. The chaining ceases when a location

is encountered that does not contain an indirect address. Indirect addressing provides a simplified method of address modifications as well as allowing access to any location in core. See Section I, paragraph 1-5 for a further discussion of indirect addressing.

The Assembler allows specification of indirect addressing by appending a comma and the letter I to any memory reference operand. The actual address of the instruction may be given in a DEF pseudo operation; this pseudo operation may also be used to indicate further levels of indirect addressing. An example is shown in figure 2-5.

A relocatable assembly language program, however, may be designed without concern for the pages in which it will be stored; indirect addressing is not required in the source language. When the program is being loaded, the loader provides indirect addressing whenever it detects an operand which does not fall in the current page or the base page. The loader substitutes a reference to a program link location (established by the loader in either the base page or the current page) and then stores an indirect address in the particular program link location. If the program link location is in the base page, all references to the same operand from other pages will be via the same link location.

2-21. CLEAR FLAG INDICATOR

The majority of the input/output instructions can alter the status of the input/output interrupt flag after execution or after the particular test is performed. In source language, this function is selected by appending a comma and a letter C to the Operand field. Some examples are shown in figure 2-6.

2-22. COMMENTS FIELD

The Comments field allows the user to transcribe notes on the program that will be listed with source language coding on the output produced by the Assembler. The field follows the Operand field and is separated from it by at least one space. The end-of-record mark, the end-of-statement mark, (CR) (LF), or the 80th character of a statement terminates the field. The statement length should not exceed 60 characters, the width of the source language portion of the listing. A whole line (up to 76 characters), however, can be specified as a comment by inserting an asterisk in the first position. On the list output, statements consisting entirely of comments begin in position 5 rather than 21 as with other source statements. Any characters beyond the above limits will not appear on the listing.

If there is no operand present, the Comments field should be omitted in the NAM and END pseudo operations and in the input/output statements, SOC, SOS, and HLT. If a comment is used, the Assembler attempts to interpret it as an operand. This limitation applies also to multi-operand instructions.

```
AB      LDA SAM,I       EACH TIME THE ISZ IS EXECUTED,
AC      ADA SAM,I       THE EFFECTIVE OPERAND OF AB AND
AD      ISZ SAM         AC CHANGE ACCORDINGLY.
          .
          .
          .
SAM     DEF ROGER
```

Figure 2-5. Indirect Addressing Example

```
STC  13B,C   SET CONTROL AND CLEAR THE FLAG OF SELECT CODE 13 (OCTAL)
OTB  16B,C   CLEAR FLAG OF SELECT CODE 16 (OCTAL) ALONG WITH OUTPUT TO DEVICE
```

Figure 2-6. Clear Flag Examples

MACHINE INSTRUCTIONS

SECTION III

The Assembler language machine instruction codes take the form of three-letter mnemonics. Each source statement corresponds to a machine operation in the object program produced by the Assembler.

Notation used in representing source language instruction is as follows:

label	Optional statement label
m	Memory location — an expression
I	Indirect addressing indicator
sc	Select code — an expression
C	Clear interrupt flag indicator
comments	Optional comments
[]	Brackets defining a field or portion of a field that is optional
{ }	Brackets indicating that one of the set may be selected.
lit	literal

Instructions shaded in gray are implemented on the M- and E-Series computers that contain the optional DMS instruction set. These instructions are not implemented on the L-SERIES computers.

NOTE

For the HP 1000 L-Series Computers: Instructions suffixed with * are implemented in software under RTE-L/XL. If the user intends to code these instructions for execution on L-SERIES hardware he should consult paragraph 3-28 and Appendix F.

3-1. MEMORY REFERENCE

The memory reference instructions perform arithmetic, logical, jump, word manipulation, byte manipulation, and bit manipulation operations on the contents of memory locations and the registers. An instruction may directly address the 2048_{10} words of the current and base pages. If required, indirect addressing may be used to refer to all $32{,}768_{10}$ words of memory. Expressions in the Operand field are evaluated modulo 2^{10}.

External memory references may be made with + or − offsets, with indirects or both.

If the program is to be assembled in relocatable form, the Operand field may contain relocatable or absolute expressions; however, absolute expressions must be less than 2000_8 in value. If the program is to be assembled in absolute form, the Operand field may contain any expression which is consistent with the location of the program. Literals may not be used in absolute programs. Absolute programs must be complete entities; they may not refer to external subroutines or to common storage.

3-2. JUMP AND INCREMENT-SKIP

Jump and Increment-Skip instructions may alter the normal sequence of program execution.

label	JMP	m [,I]	comments

Jump to m. Jump indirect inhibits interrupt until the transfer of control is complete, or three levels of indirecting have occurred.

label	JSB	m [,I]	comments

Jump to subroutine. The address for label+1 is placed into the location represented by m and control transfers to m+1. On completion of the subroutine, control may be returned to the normal sequence by performing a JMP m,I.

label	ISZ	m [,I]	comments

Increment, then skip if zero. ISZ adds 1 to the contents of m. If m then equals zero, the next single-word instruction in memory is skipped.

3-3. ADD, LOAD AND STORE

Add, Load, and Store instructions transmit and alter the contents of memory and of the A- and B-Registers. A literal, indicated by "lit", may be either =D, =B, =A, or =L type. See Section II, paragraph 2-19 for a further discussion of literals.

label	ADA	{ m [,I] / lit }	comments

Add the contents of m to A.

label	ADB	{ m [,I] / lit }	comments

Add the contents of m to B.

label	LDA	{ m [,I] / lit }	comments

Load A with the contents of m.

label	LDB	{ m [,I] / lit }	comments

Load B with the contents of m.

label	STA	m [,I]	comments

Store contents of A in m.

label	STB	m [,I]	comments

Store contents of B in m.

In each instruction, the contents of the sending location is unchanged after execution.

3-4. LOGICAL OPERATIONS

The logical instructions allow bit manipulation and the comparison of two computer words.

label	AND	{ m [,I] / lit }	comments

The logical product ("AND") of the contents of m and the contents of A are placed in A.

label	XOR	{ m [,I] / lit }	comments

The modulo-two sum (exclusive "or") of the bits in m and the bits in A is placed in A.

label	IOR	{ m [,I] / lit }	comments

The logical sum (inclusive "or") of the bits in m and the bits in A is placed in A.

label	CPA	{ m [,I] / lit }	comments

Compare the contents of m with the contents of A. If they differ, skip the next single word instruction; otherwise, continue.

label	CPB	{ m [,I] / lit }	comments

Compare the contents of m with the contents of B. If they differ, skip the next single-word instruction; otherwise, continue.

3-5. WORD PROCESSING

The word processing instructions allow the user to move a series of data words from one array in memory to another or to compare (word-by-word) the contents of two arrays in memory.

label	MVW*	{ literal / m [,I] }	comments

Move words. The A-register contains the starting (lowest) word address of the source array. The B-register contains the starting (lowest) word address of the destination array. These addresses *must not be indirect.* The number of words to be moved is specified by *literal* or by the value contained in *m* [,I]. The specified number of words are moved from the source array into the destination array. As each word is moved, the A- and B-registers are incremented by one. The source array is not altered.

label	CMW*	{ literal / m [,I] }	comments

Compare words. The A-register contains the starting (lowest) word address of array #1. The B-register contains the starting (lowest) word address of array #2. These addresses *must not be indirect.* The number of word comparisons to be performed is specified by *literal* or by the value contained in *m* [,I]. The two arrays are compared word-by-word beginning at the specified addresses. The operation is finished when an inequality is detected or when the specified number of word comparisons have been performed. When the operation is finished, the A-register contains the word address of the last word in array #1 which was compared, except when the two arrays are equal. In this case, the A-register contains the starting address of array #1, incremented by the count parameter. The B-register contains the starting address of array #2 incremented by the "count" parameter (*literal* or the value in *m* [,I]). If the two arrays are equal, execution proceeds at the next sequential source language instruction (P+3). If array #1 is "less than" #2, execution proceeds at instruction P+4. If array #1 is "greater than" array #2, execution proceeds at instruction P+5. The two arrays are not altered.

3-6. BYTE PROCESSING

The byte processing instructions allow the user to copy a data byte from memory into the A- or B-register, copy a data byte from the A- or B-register into memory, copy a series of data bytes from one array in memory to another, compare (byte-by-byte) the contents of two arrays in memory, or scan an array in memory for particular data bytes.

A *byte address* is defined as two times the word address of the memory location containing the particular data byte. If the byte location is the low order half of the memory location (bits 0-7), bit 0 of the byte address is set; if the byte location is the high order half of the memory location (bits 8-15), bit 0 of the byte address is clear. Byte addresses *may not be indirect.*

label	LBT*	comments

Load byte. The B-register contains the byte address of the byte to be loaded. The specified byte is copied from memory into bits 0-7 of the A-register (bits 8-15 of the A-register are set to zeros). The B-register is then incremented by one. The memory location is not altered.

label	SBT*	comments

Store byte. The B-register contains the byte address into which the byte is to be stored. Bits 0-7 of the A-register are copied into the specified memory byte location (bits 8-15 of the A-register are ignored). The B-register is then incremented by one. The A-register is not altered.

label	MBT*	{ literal / m [,I] }	comments

Move bytes. The A-register contains the starting (lowest) byte address of the source array. The B-register contains the starting (lowest) byte address of the destination array. The number of bytes to be moved is specified by *literal* or by the value contained in *m* [,I]. The specified number of bytes are moved from the source array into the destination array. As each byte is moved, the A- and B-registers are incremented by one. The source array is not altered.

label	CBT*	{ literal / m [,I] }	comments

Compare bytes. The A-register contains the starting (lowest) byte address of array #1. The B-register contains the starting (lowest) byte address of array #2. The number of byte comparisons to be performed is specified by *literal* or by the value contained in *m* [,I]. The two arrays are compared byte-by-byte beginning at the specified addresses. The operation is finished when an inequality is detected or when the specified number of byte comparisons have been performed. If the two arrays are equal, execution proceeds at the next sequential source language instruction (P+3); the A-register contains the address of the next byte beyond the field length and the B-register contains the starting byte address of array #2 incremented by the "count" parameter (*literal* or the value in *m* [,I]. If array #1 is "less than" array #2, execution proceeds at instruction P+4. If array #1 is "greater than" array #2, execution proceeds at instruction P+5. In both of the not-equal cases the A-register contains the byte address of the byte in array #1 where the comparison stopped and the B-register contains the starting byte address of array #2 incremented by the "count" parameter. The two arrays are not altered.

label	SFB*	comments

Scan for byte. The A-register contains a test byte in bits 0-7 and a termination byte in bits 8-15. The B-register contains the starting (lowest) byte address of the array to be scanned. The array is compared byte-by-byte against both the test and termination bytes starting at the specified address. The operation is finished when a positive comparison is detected or when the end of memory is reached. If the test byte is detected, execution proceeds at the next sequential source language instruction (P+1) and the B-register contains the address of the test byte in the array If the termination byte is detected, execution proceeds at instruction P+2 and the B-register contains the address plus one of the termination byte in the array.

The scanning operation will not continue indefinitely even if neither the termination byte nor test byte exists in memory. These bytes are in the A-register with byte addresses 000 and 001, respectively. Thus, if no match is made by the time the B-register points to the last byte in memory, the B-register will roll over to zero and the next test will match the termination byte in the A-register with itself.

3-7. BIT PROCESSING

The bit processing instructions allow the user to selectively test, set, or clear bits in a memory location according to the contents of a mask. In the descriptions below, *addr1* and *addr2* may be operand expressions.

label	TBS*	{ literal / addr1[,I] }	addr2[,I]	comments

Test bits. *literal* is a test mask, *addr1*[,I] is the address of a memory location containing a test mask, and *addr2*[,I] is the address of a memory location containing the bits to be tested. The bits in *addr2*[,I] which correspond to the "1" bits in the mask are tested. All other bits in *addr2*[,I] are ignored. If all the tested bits in *addr2*[,I] are set, execution proceeds at the next sequential source language instruction (P+3). If any of the tested bits in *addr2*[,I] are clear, execution proceeds at instruction P+4.

label	SBS*	{ literal / addr1[,I] }	addr2[,I]	comments

Set bits. *literal* is a mask, *addr1*[,I] is the address of a memory location containing a mask, and *addr2*[,I] is the address of a memory location containing the bits to be set. The bits in *addr2*[,I] which correspond to the "1" bits in the mask are set. All other bits in *addr2*[,I] are not affected. Functionally, the SBS instruction is a "logical OR" operation.

label	CBS*	{ literal / addr1[,I] }	addr2[,I]	comments

Clear bits. *literal* is a mask, *addr1*[,I] is the address of a memory location containing a mask, and *addr2*[,I] is the address of a memory location containing the bits to be cleared. The bits in *addr2*[,I] which correspond to the "1" bits in the mask are cleared. All other bits in *addr2*[,I] are not affected.

3-8. REGISTER REFERENCE

The register reference instructions include a shift-rotate group, an alter-skip group, an index register group, and NOP (no operation). For the shift-rotate and alter-skip groups, the instruction mnemonics within each group may be combined into a single source statement to cause multiple operations to be executed during one memory cycle. In such cases, successive mnemonics within a single source statement are separated from one another by a comma.

3-9. SHIFT-ROTATE GROUP

This group contains 19 basic instructions that can be combined to produce more than 500 different single cycle operations.

CLE	Clear E to zero
ALS	Shift A left one bit, zero to least significant bit. Sign unaltered
BLS	Shift B left one bit, zero to least significant bit. Sign unaltered
ARS	Shift A right one bit, extend sign; sign unaltered
BRS	Shift B right one bit, extend sign; sign unaltered
RAL	Rotate A left one bit
RBL	Rotate B left one bit
RAR	Rotate A right one bit
RBR	Rotate B right one bit
ALR	Shift A left one bit, clear sign, zero to least significant bit
BLR	Shift B left one bit, clear sign, zero to least significant bit
ERA	Rotate E and A right one bit
ERB	Rotate E and B right one bit
ELA	Rotate E and A left one bit
ELB	Rotate E and B left one bit
ALF	Rotate A left four bits
BLF	Rotate B left four bits
SLA	Skip next single-word instruction if least significant bit in A is zero
SLB	Skip next single-word instruction if least significant bit in B is zero

These instructions may be combined as follows:

label { ALS / ARS / RAL / RAR / ALR / ALF / ERA / ELA } [,CLE] [,SLA] [, { ALS / ARS / RAL / RAR / ALR / ALF / ERA / ELA }] comments

label { BLS / BRS / RBL / RBR / BLR / BLF / ERB / ELB } [,CLE] [,SLB] [, { BLS / BRS / RBL / RBR / BLR / BLF / ERB / ELB }] comments

CLE, SLA, or SLB appearing alone or in any valid combination with each other are assumed to be a shift-rotate machine instruction.

The shift-rotate instructions must be given in the order shown. At least one and up to four are included in one statement. Instructions referring to the A-register may not be combined in the same statement with those referring to the B-register.

3-10. ALTER-SKIP GROUP

The alter-skip group contains 19 basic instructions that can be combined to produce more then 700 different single cycle operations.

CLA	Clear the A-Register
CLB	Clear the B-Register
CMA	Complement the A-Register
CMB	Complement the B-Register
CCA	Clear, then complement the A-Register (set to ones)
CCB	Clear, then complement the B-Register (set to ones)
CLE	Clear the E-Register
CME	Complement the E-Register
CCE	Clear, then complement the E-Register
SEZ	Skip next single-word instruction if E is zero

SSA	Skip if sign of A is positive (0)
SSB	Skip if sign of B is positive (0)
INA	Increment A by one
INB	Increment B by one
SZA	Skip if contents of A equals zero
SZB	Skip if contents of B equals zero
SLA	Skip if least significant bit of A is zero
SLB	Skip if least significant bit of B is zero
RSS	Reverse the sense of the skip instructions. If no skip instructions precede in the statement, skip the next instruction

These instructions may be combined as follows:

label	[{CLA / CMA / CCA}] [,SEZ] [, {CLE / CME / CCE}] [,SSA] [,SLA] [,INA] [,SZA] [,RSS]	comments
label	[{CLB / CMB / CCB}] [,SEZ] [, {CLE / CME / CCE}] [,SSB] [,SLB] [,INB] [,SZB] [,RSS]	comments

The alter-skip instructions must be given in order shown. At least one and up to eight are included in one statement. Instructions referring to the A-register may not be combined in the same statement with those referring to the B-register. When two or more skip functions are combined in a single operation, a skip occurs if any one of the conditions exists. If a word with RSS also includes both SSA and SLA (or SSB and SLB), a skip occurs only when sign and least significant bit are both set (1).

3-11. INDEX REGISTER GROUP

This group contains 32 instructions which perform various operations involving the use of index registers X and Y. Instructions in this group may directly address all $32{,}768_{10}$ words of memory. Indirect addressing may also be used if desired.

label	CAX*	comments

Copy A to X. The contents of the A-register are copied into the X-register. The A-register is not altered.

label	CBX*	comments

Copy B to X. The contents of the B-register are copied into the X-register. The B-register is not altered.

label	CAY*	comments

Copy A to Y. The contents of the A-register are copied into the Y-register. The A-register is not altered.

label	CBY*	comments

Copy B to Y. The contents of the B-register are copied into the Y-register. The B-register is not altered.

label	CXA*	comments

Copy X to A. The contents of the X-register are copied into the A-register. The X-register is not altered.

label	CXB*	comments

Copy X to B. The contents of the X-register are copied into the B-register. The X-register is not altered.

label	CYA*	comments

Copy Y to A. The contents of the Y-register are copied into the A-register. The Y-register is not altered.

label	CYB*	comments

Copy Y to B. The contents of the Y-register are copied into the B-register. The Y-register is not altered.

label	XAX*	comments

Exchange A and X. The contents of the A-register are copied into the X-register and the contents of the X-register are copied into the A-register.

label	XBX*	comments

Exchange B and X. The contents of the B-register are copied into the X-register and the contents of the X-register are copied into the B-register.

label	XAY*	comments

Exchange A and Y. The contents of the A-register are copied into the Y-register and the contents of the Y-register are copied into the A-register.

label	XBY*	comments

Exchange B and Y. The contents of the B-register are copied into the Y-register and the contents of the Y-register are copied into the B-register.

label	ISX*	comments

Increment X and skip if zero. The contents of the X-register are incremented by one and then tested. If the new value in X is zero, the next sequential single-word instruction (P+1) is skipped and execution proceeds at instruction P+2; if the new value in X is non-zero, execution proceeds at instruction P+1.

label	ISY*	comments

Increment Y and skip if zero. The contents of the Y-register are incremented by one and then tested. If the new value in Y is zero, the next sequential single-word instruction (P+1) is skipped and execution proceeds at instruction P+2; if the new value in Y is non-zero, execution proceeds at instruction P+1.

label	DSX*	comments

Decrement X and skip if zero. The contents of the X-register are decremented by one and then tested. If the new value in X is zero, the next sequential instruction (P+1) is skipped and execution proceeds at instruction P+2; if the new value in X is non-zero, execution proceeds at instruction P+1.

label	DSY*	comments

Decrement Y and skip if zero. The contents of the Y-register are decremented by one and then tested. If the new value in Y is zero, the next sequential single-word instruction (P+1) is skipped and execution proceeds at instruction P+2; if the new value in Y is non-zero, execution proceeds at instruction P+1.

label	LDX*	{ m [,I] / literal }	comments

Load X from memory. The contents of the specified memory location are copied into the X-register. Indirect addressing may be used. The memory location is not altered.

label	LDY*	{ m [,I] / literal }	comments

Load Y from memory. The contents of the specified memory location are copied into the Y-register. Indirect addressing may be used. The memory location is not altered.

label	STX*	m [,I]	comments

Store X into memory. The contents of the X-register are copied into the specified memory location. Indirect addressing may be used. The X-register is not altered.

label	STY*	m [,I]	comments

Store Y into memory. The contents of the Y-register are copied into the specified memory location. Indirect addressing may be used. The Y-register is not altered.

label	LAX*	m [,I]	comments

Load A from memory indexed by X. The contents of the specified memory location are copied into the A-register. Indirect addressing may be used. The address of the memory location is computed by adding the contents of the X-register to *m* or to *m*,I. Note that indirect addressing (if specified) is performed first and then the address is indexed. The X-register and the memory location are not altered.

label	LBX*	m [,I]	comments

Load B from memory indexed by X. The contents of the specified memory location are copied into the B-register. Indirect addressing may be used. The address of the memory location is computed by adding the contents of the X-register to *m* or to *m*,I. Note that indirect addressing (if specified) is performed first and then the address is indexed. The X-register and the memory location are not altered.

label	LAY*	m [,I]	comments

Load A from memory indexed by Y. The contents of the specified memory location are copied into the A-register. Indirect addressing may be used. The address of the memory location is computed by adding the contents of the Y-register to *m* or to *m*,I. Note that indirect addressing (if specified) is performed first and then the address is indexed. The Y-register and the memory location are not altered.

label	LBY*	m [,I]	comments

Load B from memory indexed by Y. The contents of the specified memory location are copied into the B-register. Indirect addressing may be used. The address of the memory location is computed by adding the contents of the Y-register to *m* or to *m*,I. Note that indirect addressing (if specified) is performed first and then the address is indexed. The Y-register and the memory location are not altered.

label	SAX*	m [,I]	comments

Store A into memory indexed by X. The contents of the A-register are copied into the specified memory location. Indirect addressing may be used. The address of the memory location is computed by adding the contents of the X-register to *m* or to *m*,I. Note that indirect addressing (if specified) is performed first and then the address is indexed. The A-register and the X-register are not altered.

label	SBX*	m [,I]	comments

Store B into memory indexed by X. The contents of the B-register are copied into the specified memory location. Indirect addressing may be used. The address of the memory location is computed by adding the contents of the X-register to *m* or to *m*,I. Note that indirect addressing (if specified) is performed first and then the address is indexed. The B-register and the X-register are not altered.

label	SAY*	m [,I]	comments

Store A into memory indexed by Y. The contents of the A-register are copied into the specified memory location. Indirect addressing may be used. The address of the memory location is computed by adding the contents of the Y-register to *m* or to *m*,I. Note that indirect addressing (if specified) is performed first and then the address is indexed. The A-register and the Y-register are not altered.

label	SBY*	m [,I]	comments

Store B into memory indexed by Y. The contents of the B-register are copied into the specified memory location. Indirect addressing may be used. The address of the memory location is computed by adding the contents of the Y-register to *m* or to *m*,I. Note that indirect addressing (if specified) is performed first and then the address is indexed. The B-register and the Y-register are not altered.

label	ADX*	m [,I]	comments

Add memory to X. The contents of the specified memory location are algebraically added to the contents of the X-register. Indirect addressing may be used. The memory location is not altered.

label	ADY*	m [,I]	comments

Add memory to Y. The contents of the specified memory location are algebraically added to the contents of the Y-register. Indirect addressing may be used. The memory location is not altered.

label	JLY*	m [,I]	comments

Jump and load Y. Control transfers unconditionally to the specified memory location and the address P+2 is loaded into the Y-register. Indirect addressing may be used. This instruction is used for calling subroutines. The subroutines use the Y-register to access parameters and to return control (by way of the JPY instruction) to the calling program.

label	JPY*	m	comments

Jump indexed by Y. Control transfers unconditionally to the specified memory location. Indirect addressing may *not* be used. The address of the memory location is computed by adding the contents of the Y-register to *m*. This instruction is used for returning control from subroutines to the calling program (assuming that they were entered by way of JLY instructions).

3-12. NO-OPERATION INSTRUCTION

When a no-operation is encountered in a program, no action takes place; the computer goes on to the next instruction. A full memory cycle is used in executing a no-operation instruction.

label	NOP	comments

A subroutine to be entered by a JSB instruction should have a NOP as the first statement. The return address can be stored in the location occupied by the NOP during execution of the program. A NOP statement causes the Assembler to generate a word of zero.

3-13. INPUT/OUTPUT, OVERFLOW, AND HALT

The input/output instructions allow the user to transfer data to and from an external device, to enable or disable external interrupts, and to check the status of I/O devices and operations. Input/output instructions are also used to control CPU functions such as memory protect, power fail recovery and overflow conditions.

Input/output instructions require the designation of a select code, sc, which indicates one of 64_{10} input/output channels or functions.

Note: When Memory Protect is enabled, execution of most I/O instructions *is prohibited.*

Expressions used to represent select codes must have a value of less than 2^6. The value specifies the device or operation referenced. Unlike memory reference instruc-

tions, I/O instructions cannot use indirect links. The select code (sc) may be a label which was previously defined as an external symbol by an EXT pseudo-instruction. In such a case, the entry point referred to by the EXT pseudo-instruction must be an absolute value less than 64_{10} (any other value will change the instruction.

Since input/output instructions are generally hardware/architecture dependent, the instructions presented here are meant to be used as a summary and coding guide. The user is referred to the appropriate CPU hardware manual for a detailed description of the I/O architecture and its operation.

3-14. INPUT/OUTPUT

Assembly language programs normally perform I/O through calls to EXEC. Consult the appropriate RTE Programming and Operating manual for more information.

If the memory protect hardware option is present and enabled, it protects the operating system from alteration. Most instructions of this section cause memory protect violations to occur. They are included here for users who desire to write their own drivers.

To perform I/O, the programming, installation and service manuals of the CPU and I/O card being programmed should be consulted for the meaning of these instructions.

RTE-L users should refer to the RTE-L Driver Designer's manual for the operating system and I/O conventions.

label	STC	sc [,C]	comments

Set I/O control bit specified by sc.

label	CLC	sc [,C]	comments

Clear I/O control bit specified by sc.

label	LIA	sc [,C]	comments

Load into A the contents of the I/O buffer indicated by sc.

label	LIB	sc [,C]	comments

Load into B the contents of the I/O buffer indicated by sc.

label	MIA	sc [,C]	comments

Merge (inclusive "or") the contents of the I/O buffer indicated by sc into A.

label	MIB	sc [,C]	comments

Merge (inclusive "or") the contents of the I/O buffer indicated by sc into B.

label	OTA	sc [,C]	comments

Output the contents of A to the I/O buffer indicated by sc.

label	OTB	sc [,C]	comments

Output the contents of B to the I/O buffer indicated by sc.

label	STF	sc	comments

Set the flag bit indicated by sc.

label	CLF	sc	comments

Clear the flag bit to zero indicated by sc.

label	SFC	sc	comments

Skip the next single-word instruction if the flag indicated by sc is clear.

label	SFS	sc	comments

Skip the next single-word instruction if the flag bit indicated by sc is set. If sc = 1, the overflow is tested.

3-15. OVERFLOW

The overflow bit may be accessed by the following instructions:

label	CLO	comments

Clear the overflow bit.

label	STO	comments

Set overflow bit.

label	SOC	[C]	comments

Skip the next single-word instruction if the overflow bit is clear. The C option clears the bit after the test is performed.

label	SOS	[C]	comments

Skip the next single-word instruction if the overflow bit is set. The C option clears the bit after the test is performed.

The C option is identified by the sequence "space C space" following either "SOC" or "SOS". Any letter other than a "C" in this position will be treated as a comment.

3-16. HALT

label	HLT	{ [sc [,C]] / [C] }	comments

Halt the computer. The machine instruction word is displayed in the T-register. If the C option is used, the flag bit associated with channel sc is cleared.

If neither the select code nor the C option is used, the comments portion must be omitted.

3-17. EXTENDED ARITHMETIC UNIT (EAU)

label	MPY	{ m [,I] / lit }	comments

The MPY instruction multiplies the contents of the A-Register by the contents of m. The product is stored in registers B and A. B contains the sign of the product and the 15 most significant bits; A contains the least significant bits.

label	DIV	{ m [,I] / lit }	comments

The DIV instruction divides the contents of registers B and A by the contents of m. The quotient is stored in A and the remainder in B. Initially B contains the sign and the 15 most significant bits of the dividend; A contains the least significant bits.

label	DLD	{ m [,I] / lit }	comments

The DLD instruction loads the contents of locations m and m + 1 into registers A and B, respectively.

label	DST	m [,I]	comments

The DST instruction stores the contents of registers A and B in locations m and m + 1, respectively.

MPY, DIV, DLD, DST results in two machine words: a word for the instruction code and one for the operand.

The following seven instructions provide the capability to shift or rotate the B- and A-Registers n number of bit positions to the right or left, where $1 \leq n \leq 16$.

label	ASR	n	comments

The ASR instruction arithmetically shifts the B- and A-Registers right n bits. The sign bit (bit 15 of B) is extended.

label	ASL	n	comments

The ASL instruction arithmetically shifts the B- and A-Register left n bits. Zeroes are placed in the least significant bits. The sign bit (bit 15 of B) is unaltered. The overflow bit is set if bit 14 differs from bit 15 before each shift; otherwise, exit with overflow bit cleared.

label	RRR	n	comments

The RRR instruction rotates the B- and A-Registers right n bits.

label	RRL	n	comments

The RRL instruction rotates the B- and A-Registers left n bits.

label	LSR	n	comments

The LSR instruction logically shifts the B- and A-Registers right n bits. Zeroes are placed in the most significant bits.

label	LSL	n	comments

The LSL instruction logically shifts the B- and A-Registers left n bits. Place zeroes into the least significant bits.

	SWP		

Exchange the contents of the A- and B-Registers. The contents of the A-Register are shifted into the B-Register and the contents of the B-Register are shifted into the A-Register.

3-18. FLOATING POINT

The instructions in this group are used for performing arithmetic operations on floating point operands. The user specifies whether or not floating point machine instructions are available via a parameter in the control statement (see table 1-2). If the floating point machine instructions are *not* available, the instructions in this group result in calls to arithmetic subroutines except for FIX and FLT (see paragraph 4-7). The Operand field may contain any relocatable expression or absolute expression resulting in a value of less than 2000_8.

label	FMP*	{ m [,I] / =Fn }	comments

Multiply the two-word floating point quantity in registers A and B by the two-word floating point quantity in locations m and m+1 or the quantity defined by the literal. Store the two-word floating point product in registers A and B.

label	FDV*	{ m [,I] / =Fn }	comments

Divide the two-word floating point quantity in registers A and B by the two-word floating point quantity in locations m and m+1 or the quantity defined by the literal. Store the two-word floating point quotient in A and B.

label	FAD*	{ m [,I] / =FN }	comments

Add the two-word floating point quantity in registers A and B to the two-word floating point quantity in locations m and m+1 or the quantity defined by the literal. Store the two-word floating point sum in A and B.

label	FSB*	{ m [,I] / =Fn }	comments

Subtract the two-word floating point quantity in m and m+1 or the quantity defined by the literal from the two-word floating point quantity in registers A and B and store the difference in A and B.

label	FIX*	comments

Convert the floating-point number contained in the A- and B-registers to a fixed-point number. The result is returned in the A-register. After the operation is completed, the contents of the B-register are meaningless.

label	FLT*	comments

Convert the fixed-point number contained in the A-register to a floating-point number. The result is returned in the A- and B-registers.

3-19. DYNAMIC MAPPING SYSTEM (M, E AND F-SERIES ONLY)

The basic addressing space of the HP 1000 Computer Series is 32,768 words, which is referred to as *logical* memory. The amount of memory actually installed in the computer system is referred to as *physical* memory. An HP 1000 Computer with the optional Dynamic Mapping System (DMS) has an addressing capability for one million words of memory. The DMS allows physical memory to be mapped into logical memory through the use of four dynamically alterable memory maps.

3-20. MEMORY ADDRESSING

The basic memory addressing scheme provides for addressing 32 pages of logical memory, each of which consists of 1,024 words. This memory is addressed through a 15-bit memory address bus shown in figure 3-1. The upper 5 bits of this bus provide the page address and the lower 10 bits provide the relative word address within the page.

The Memory Expansion Module (MEM), which is part of the DMS option, converts the 5-bit page address into a 10-bit page address and thereby allows 1,024 (2^{10}) *pages* to be addressed. This conversion is accomplished by allowing the original 5-bit address to identify one of the 32 registers within a "memory map." Each of these map registers contains the new user-specified 10-bit page address. This new page address is combined with the original 10-bit relative address to form a 20-bit memory address bus as shown in figure 3-2.

3-21. STATUS AND VIOLATION REGISTERS

The MEM also includes a status register and a violation register. As shown in table 3-1, the MEM status register contents enable the programmer to determine whether the MEM was enabled or disabled at the time of the last interrupt and the address of the base page fence. The MEM

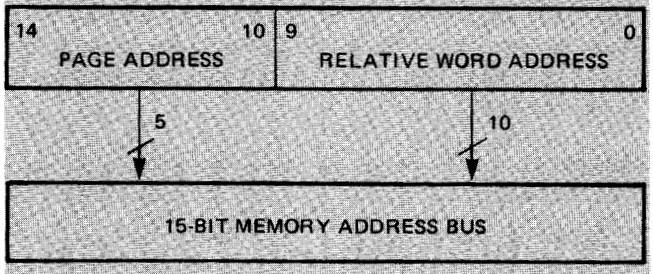


Figure 3-1. Basic Memory Addressing Scheme

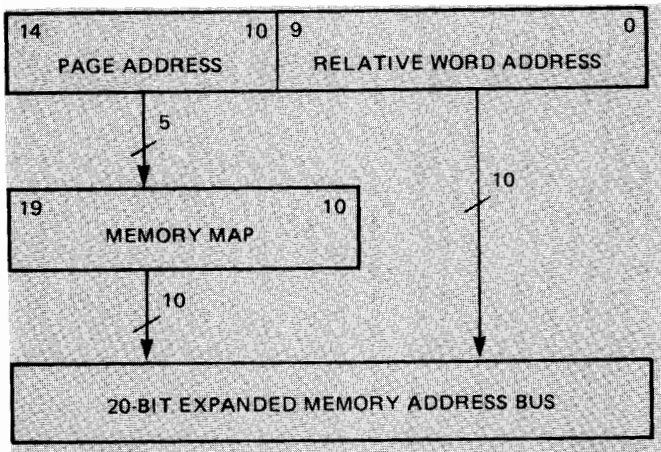


Figure 3-2. Expanded Memory Addressing Scheme

Table 3-1. MEM Status Register Format

BIT	SIGNIFICANCE
15	0 = MEM disabled at last interrupt 1 = MEM enabled at last interrupt
14	0 = System map selected at last interrupt 1 = User map selected at last interrupt
13	0 = MEM disabled currently 1 = MEM enabled currently
12	0 = System map selected currently 1 = User map selected currently
11	0 = Protected mode disabled currently 1 = Protected mode enabled currently
10	Portion mapped*
9	Base page fence bit 9
8	Base page fence bit 8
7	Base page fence bit 7
6	Base page fence bit 6
5	Base page fence bit 5
4	Base page fence bit 4
3	Base page fence bit 3
2	Base page fence bit 2
1	Base page fence bit 1
0	Base page fence bit 0

*Bit 10	Mapped Address (M)
0	Fence $\leq$ M $<$ 2000_8
1	1 $<$ M $<$ Fence

violation register contents enable the programmer to determine whether a fault occurred in the hardware or the software so that the proper corrective steps may be taken. Refer to table 3-2.

3-22. MAP SEGMENTATION

All registers within the memory map are dynamically alterable. The MEM includes four separate memory maps: the User Map, System Map, and two Dual-Channel Port Controller (DCPC) Maps. See figure 3-3. These maps are addressed as a contiguous register block.

3-23. POWER FAIL CHARACTERISTICS

A power failure automatically enables the System Map, and a minimum of 500 microseconds is assured the programmer for executing a power fail routine. Since all maps are disabled and none are considered valid upon the restoration of power, the power fail routine should include instructions to save as many maps as desired.

Table 3-2. MEM Violation Register Format

BIT	SIGNIFICANCE
15	Read violation*
14	Write violation*
13	Base page violation*
12	Privileged instruction violation*
11	Reserved
10	Reserved
9	Reserved
8	Reserved
7	0 = ME bus disabled at violation 1 = ME bus enabled at violation
6	0 = MEM disabled at violation 1 = MEM enabled at violation
5	0 = System map enabled at violation 1 = User map enabled at violation
4	Map address bit 4
3	Map address bit 3
2	Map address bit 2
1	Map address bit 1
0	Map address bit 0

*Significant when associated bit is set.

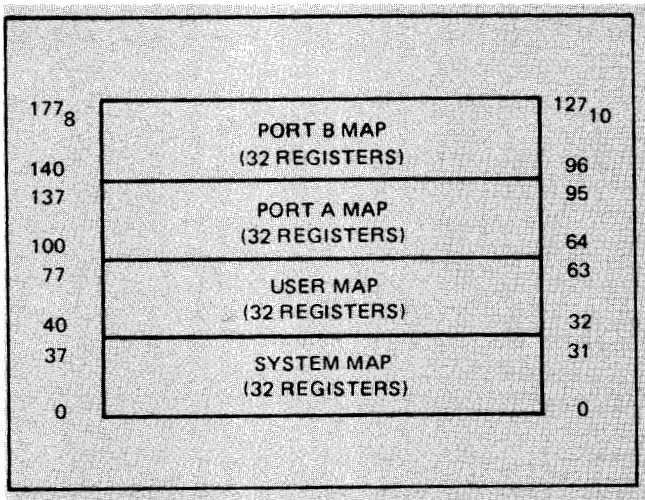
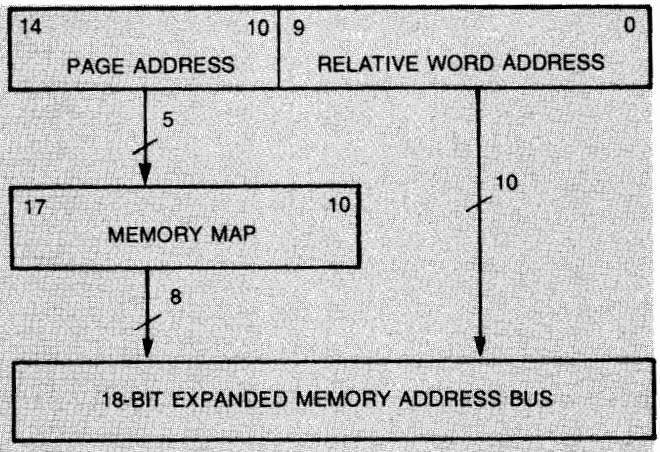


Figure 3-3. Map Segmentation



(Figure 3-3a. RTE-XL Expanded Memory Addressing Scheme

3-24. PROTECTED MODE

The protected mode of operation is a program state created by the Dynamic Mapping System. The protected mode is entered by executing an STC 05 instruction and is exited by the CPU acknowledging an interrupt. The protected mode reserves a block of memory and prevents access to this block by other users.

3-25. MEM VIOLATION

An interrupt request which attempts to access the protected block of memory (while in the protected mode) will cause a MEM violation.

3-25A. DYNAMIC MAPPING SYSTEM (XL ONLY)

The basic addressing space of the HP 1000 Computer Series is 32,768 words, which is referred to as *logical* memory. The amount of memory actually installed in the computer system is referred to as *physical* memory. An HP 1000 L-Series Computer with the XL memory system has an addressing capability of 262,144 words. The XL allows physical memory to be mapped into logical memory through the use of a dynamically alterable memory map and a set of DMA relocation registers.

3-25B. MEMORY ADDRESSING

The XL memory system converts the 5-bit page address into an 8-bit page address and thereby allows 256 (2^8) *pages* to be addressed. This conversion is accomplished by allowing the original 5-bit address to identify one of the 32 registers within the "memory map." Each of these map registers contains the user-specified 8-bit page address. This new page address is combined with the original 10-bit relative address to form an 18-bit memory address as shown in figure 3-3a.

3-25C. DMA RELOCATION REGISTERS

At any time during the execution of a user program or the operating system, an I/O interface card may transfer a word of data between a device and main memory through Direct Memory Addess (DMA). The interface specifies a logical address within a petition and also specifies the number of a relocation register that contains the starting physical page of the partition. The upper five bits of the logical address are added to the 8-bit contents of the relocation register to compute the physical page number for the DMA transfer. DMA does not use the mapping registers, since the mapping registers are typically allocated to a program executing in a different partition than the one which is receiving the I/O transfer.

3-25D. PROGRAMMING CHARACTERISTICS

When power is initially applied to an XL memory system, logical mapping of instruction fetches from memory is disabled (the physical address equals the logical address). The contents of the map registers and DMA relocation registers are set by storing into reserved memory locations in physical page zero. Maps are enabled via special I/O group instructions to allow execution or data transfer within any set of 32 pages of memory. Whenever an interrupt occurs (I/O, power-fail, parity, etc.) mapping is disabled and the contents of the mapping registers may be changed.

3-25E. COMPARISON OF M, E, F SERIES TO XL MAPPING

In the following description of instructions, the operational characteristics are described in terms of M, E, and F

Series hardware. To understand the descriptions in terms of XL hardware, simply apply the following definitions:

1. *mapping disabled*: The logical address is equivalent to the physical address for *instruction* fetches and indirect *address* resolution.

2. system map enabled: Same as *mapping disabled*.

3. *mapping enabled*: The logical address is translated to a physical address based on the contents of the map registers for *instruction* fetches and indirect *address* resolution.

4. user map enabled: Same as *mapping enabled*.

5. *alternate map*: If mapping is enabled for *instruction* fetches, the logical address is equivalent to the physical address for *data* references. If mapping is disabled for *instruction* fetches, the logical address is translated to physical address based on the contents of the map registers for *data* references.

The XL memory system does *not* include DCPC port maps, system map, violation register, status register, or base page fence. Therefore, instructions referencing those hardware registers are not available on the XL.

The XL has no read/write protect bits in the map registers. Reads and writes are normally allowed to all pages specified in the map registers, with the exception of cross map store instructions. If memory protect is enabled (the normal case while mapping is enabled), cross map stores do *not* modify memory in the alternate partition. No violation interrupt is generated and execution continues as if the store had succeeded.

3-26. DYNAMIC MAPPING SYSTEM INSTRUCTIONS

If the computer on which the object program is to be run includes a Dynamic Mapping System, the following group of instructions may be used.

label	DJP	m [,I]	comments

Disable MEM and jump. This instruction disables the translation and protection features of the MEM hardware. Prior to disabling, the P-register is set to the effective memory address. As a result of executing this instruction, normal I/O interrupts are held off until the first opportunity following the fetch of the next instruction, unless three or more levels of indirect addressing are used.

This instruction will normally generate a MEM violation when executed in the protected mode. In this case, the status of the MEM is not affected and the jump will not occur; however, if the System map is enabled, the instruction is allowed. If none of the maps are enabled, the instruction defaults to JMP* + 1,I.

label	DJS	m [,I]	comments

Disable MEM and jump to subroutine. This instruction disables the translation and protection features of the MEM hardware. Prior to disabling, the P-register is set one count past the effective memory address. The return address is written into the location specified by *m* [,I] . As a result of executing this instruction, normal I/O interrupts are held off until the first opportunity following the fetch of the next instruction, unless three or more levels of indirect addressing are used.

This instruction will normally generate a MEM violation when executed in the protected mode. In this case, the status of the MEM is not affected and the jump will not occur; however, if the System map is enabled, the instruction is allowed.

label	JRS	{addr1 [,I] / literal}	addr2 [,I]	comments

Jump and restore status. *addr1* contains the address of the status word memory location, *literal* specifies the status word, and *addr2* contains the jump address.

This instruction causes the status of MEM to be restored as indicated by bits 15 and 14 of the status word. Only bits 15 and 14 of the status word are used; the remaining bits (13-0) of the status word are ignored. Bits 15 and 14 restore the MEM status as follows:

Bit 15 = 0 MEM is disabled

Bit 15 = 1 MEM is enabled

Bit 14 = 0 System map is selected

Bit 14 = 1 User map is selected.

As a result of executing this instruction, normal I/O interrupts are held off until the first opportunity following the fetch of the next instruction, unless three or more levels of indirect addressing are used.

This instruction will normally generate a MEM violation when executed in the protected mode. In this case, the status of the MEM is not affected and the jump will not occur; however, if the System map is enabled, the instruction is allowed.

label	LFA	comments

Load fence from A. This instruction loads the contents of the A-register into the base page fence register. (The base page fence register contains the "fence" address, which

specifies the address where reserved (mapped) memory begins. Attempts to access memory at any address below this fence will not be allowed.) Bits 9 through 0 of the A-register specify the address in page zero where shared (unmapped) memory is separated from reserved (mapped) memory. Bit 10 is used as follows to specify which portion is mapped:

Bit 10	Mapped Address (M)
0	Fence $\leq$ M $<$ 2000_8
1	1 $<$ M $<$ Fence

This instruction will always generate a MEM violation when executed in the protected mode. In this case, the fence is not altered. However, if the System map is enabled, the instruction is allowed in protected mode.

label	LFB	comments

Load fence from B. This instruction loads the contents of the B-register into the base page fence register. Bits 9 through 0 of the B-register specify the address in page zero where shared (unmapped) memory is separated from reserved (mapped) memory. Bit 10 is used as follows to specify which portion is mapped:

Bit 10	Mapped Address (M)
0	Fence $\leq$ M $<$ 2000_8
1	1 $<$ M $<$ Fence

This instruction will always generate a MEM violation when executed in the protected mode. In this case, the fence is not altered. However, if the System map is enabled, the instruction is allowed in protected mode.

label	MBF*	comments

Move bytes from alternate map. This instruction moves a string of bytes using the alternate program map for source reads and the currently enabled map for destination writes. (The alternate map is the map which is not enabled. For example, if the system map is enabled, the User map is the alternate map and vice versa.) The A-register contains the source byte address and the B-register contains the destination byte address. The X-register contains the octal number of bytes to be moved. Both the source and destination byte address must begin on even word boundaries.

This instruction is interruptible on an even number of byte transfers, thus maintaining the even word boundaries in the A- and B-registers.

The interrupt routine is expected to save and restore the current contents of the A-, B-, and X-registers to allow continuation of the instruction at the next entry. When the byte string move is completed, the X-register will always be zero and the A- and B-registers will contain their original value incremented by the number of bytes moved.

This instruction can cause a MEM violation only if read or write protection rules are violated. (For example, if an attempt is made to write to the reserved (mapped) section of memory.)

label	MBI*	comments

Move bytes into alternate map. This instruction moves a string of bytes using the currently enabled map for source reads and the alternate program map for destination writes. The A-register contains the source byte address and the B-register contains the destination byte address. The X-register contains the octal number of bytes to be moved. Both the source and destination byte addresses must begin on even word boundaries.

This instruction is interruptible on an even number of byte transfers, thus maintaining the even word boundaries in the A- and B-registers. The interrupt routine is expected to save and restore the current contents of the A-, B-, and X-registers to allow continuation of the instruction at the next entry. When the byte string move is completed, the X-register will always be zero and the A- and B-registers will contain their original value incremented by the number of bytes moved.

This instruction will always cause a MEM violation when executed in the protected mode and no bytes will be transferred.

label	MBW*	comments

Move bytes within alternate map. This instruction moves a string of bytes with both the source and destination addresses established through the alternate program map. The A-register contains the source byte address and the B-register contains the destination byte address. The X-register contains the octal number of bytes to be moved. Both the source and destination byte addresses must begin on even word boundaries.

This instruction is interruptible on an even number of byte transfers, thus maintaining the even word boundaries in the A- and B-registers.

The interrupt routine is expected to save and restore the current contents of the A-, B- and X-registers to allow continuation of the instruction at the next entry. When the byte string move is completed, the X-register will always be zero and the A- and B-registers will contain their original value incremented by the number of bytes moved.

This instruction will always cause a MEM violation when executed in the protected mode and no bytes will be transferred.

label	MWF*	comments

Move words from alternate map. This instruction moves a string of words using the alternate program map for source

reads and the currently enabled map for destination writes. The A-register contains the source address and the B-register contains the destination address. The X-register contains the octal number of words to be moved.

This instruction is interruptible. The interrupt routine is expected to save and restore the current contents of the A-, B-, and X-registers to allow continuation of the instruction at the next entry. When the word string move is completed, the X-register will always be zero and the A- and B-registers will contain their original value incremented by the number of words moved.

This instruction can cause a MEM violation only if read and write protection rules are violated.

label	MWI*	comments

Move words into alternate map. This instruction moves a string of words using the currently enabled map for source reads and the alternate program map for destination writes. The A-register contains the source address and the B-register contains the destination address. The X-register contains the octal number of words to be moved.

This instruction is interruptible. The interrupt routine is expected to save and restore the current contents of the A-, B-, and X-registers to allow continuation of the instruction at the next entry. When the word string move is completed, the X-register will always be zero and the A- and B-registers will contain their original value incremented by the number of words moved.

This instruction will always cause a MEM violation when executed in the protected mode and no words will be transferred.

label	MWW*	comments

Move words within alternate map. This instruction moves a string of words with both the source and destination addresses established through the alternate program map. The A-register contains the source address and the B-register contains the destination address. The X-register contains the octal number of words to be moved.

This instruction is interruptible. The interrupt routine is expected to save and restore the current contents of the A-, B-, and X-registers to allow continuation of the instruction at the next entry. When the word string move is completed, the X-register will always be zero and the A- and B-registers will contain their original value incremented by the number of words moved.

This instruction will always cause a MEM violation when executed in the protected mode and no words will be transferred.

label	PAA	comments

Load/store Port A map per A. This instruction transfers the 32 Port A map registers to or from memory. If bit 15 of the A-register is clear, the Port A map is *loaded* from memory starting from the address specified in bits 14-0 of the A-register. If bit 15 of the A-register is set, the Port A map is *stored* into memory starting at the address specified in bits 14-0 of the A-register. When the load/store operation is complete, the A-register will be incremented by 32 to allow multiple map instructions.

An attempt to load any map register when in the protected mode will cause a MEM violation. An attempt to store the Port A map is allowed within the constraints of write protected memory.

label	PAB	comments

Load/store Port A map per B. This instruction transfers the 32 Port A registers to or from memory. If bit 15 of the B-register is clear, the Port A map is *loaded* from memory starting from the address specified in bits 14-0 of the B-register. If bit 15 of the B-register is set, the Port A map is *stored* into memory starting at the address specified in bits 14-0 of the B-register. When the load/store operation is complete, the B-register will be incremented by 32 to allow multiple map instructions.

An attempt to load any map register when in the protected mode will cause a MEM violation. An attempt to store the Port A map is allowed within the constraints of write protected memory.

label	PBA	comments

Load/store Port B map per A. This instruction transfers the 32 Port B registers to or from memory. If bit 15 of the A-register is clear, the Port B map is *loaded* from memory starting from the address specified in bits 14-0 of the A-register. If bit 15 of the A-register is set, the Port B map is *stored* into memory starting at the address specified in bits 14-0 of the A-register. When the load/store operation is complete, the A-register will be incremented by 32 to allow multiple map instructions.

An attempt to load any map register when in the protected mode will cause a MEM violation. An attempt to store the Port B map is allowed within the constraints of write protected memory.

label	PBB	comments

Load/store Port B map per B. This instruction transfers the 32 Port B map registers to or from memory. If bit 15 of the B-register is clear, the Port B map is *loaded* from memory starting from the address specified in bits 14-0 of the B-register. If bit 15 of the B-register is set, the Port B map is *stored* into memory starting at the address specified in bits 14-0 of the B-register. When the load/store operation is complete, the B-register will be incremented by 32 to allow multiple map instructions.

An attempt to load any map register when in the protected mode will cause a MEM violation. An attempt to store the Port B map is allowed within the constraints of the write protected memory.

label	RSA	comments

Read status register into A. This instruction reads the contents of the MEM status register into the A-register. This instruction can be executed at any time. The format of the MEM status register is given in table 3-1.

label	RSB	comments

Read status register into B. This instruction reads the contents of the MEM status register into the B-register and can be executed at any time. The format of the MEM status register is shown in table 3-1.

label	RVA	comments

Read violation register into A. This instruction reads the contents of the MEM violation register into the A-register and can be executed at any time. The format of the MEM violation register is shown in table 3-2.

label	RVB	comments

Read violation register into B. This instruction reads the contents of the MEM violation register into the B-register and can be executed at any time. The format of the MEM violation register is shown in table 3-2.

label	SJP	m [,I]	comments

Enable System map and jump. This instruction causes the MEM hardware to use the set of 32 map registers, referred to as the System map, for translating all programmed memory references. Prior to enabling the System map, the P-register is set to the effective memory address. As a result of executing this instruction, normal I/O interrupts are held off until the first opportunity following the fetch of the next instruction, unless three or more levels of indirect addressing are used.

This instruction will normally generate a MEM violation when executed in the protected mode. In this case, the status of the MEM is not affected and the jump will not occur; however, if the System map is enabled, the instruction is allowed and effectively executes a JMP *+1, I.

label	SJS	m [,I]	comments

Enable System map and jump to subroutine. This instruction causes the MEM hardware to use the set of 32 map registers, referred to as the System map, for translating all programmed memory references. Prior to enabling the System map, the P-register is set one count past the effective memory address. The return address is written into the location specified by *m* [,I] . As a result of executing this instruction, normal I/O interrupts are held off until the first opportunity following the fetch of the next instruction, unless three or more levels of indirect addressing are used.

This instruction will normally generate a MEM violation when executed in the protected mode. In this case, the status of the MEM is not affected and the jump will not occur; however, if the System map is enabled, the instruction is allowed and effectively executes a JSB *+1,I.

label	SSM	m [,I]	comments

Store status register in memory. This instruction stores the 16-bit contents of the MEM status register into the addressed memory location. The status register contents are not altered. This instruction is used in conjunction with the JRS instruction to allow easy processing of interrupts, which always select the System map (if the MEM is enabled). The format of the MEM status register is listed in table 3-1.

This instruction can cause a MEM violation only if write protection rules are violated.

label	SYA	comments

Load/store System map per A. This instruction transfers the 32 System map registers to or from memory. If bit 15 of the A-register is clear, the System map is *loaded* from memory starting from the address specified in bits 14-0 of the A-register. If bit 15 of the A-register is set, the System map is *stored* into memory starting at the address specified in bits 14-0 of the A-register. When the load/store operation is complete, the A-register will be incremented by 32 to allow multiple map instructions.

Note: If not in the protected mode, the MEM provides no protection against altering the contents of maps while they are currently enabled.

An attempt to load any map in the protected mode will cause a MEM violation. An attempt to store the System map is allowed within the constraints of write protected memory.

label	SYB	comments

Load/store System map per B. This instruction transfers the 32 System map registers to or from memory. If bit 15 of the B-register is clear, the system map is *loaded* from memory starting from the address specified in bits 14-0 of the B-register. If bit 15 of the B-register is set, the System map is *stored* into memory starting at the address specified in bits 14-0 of the B-register. When the load/store operation is complete, the B-register will be incremented by 32 to allow multiple map instructions.

Note: If not in the protected mode, the MEM provides no protection against altering the contents of maps while they are currently enabled.

An attempt to load any map in the protected mode will cause a MEM violation. An attempt to store the System map is allowed within the constraints of write protected memory.

label	UJP	m [,I]	comments

Enable User map and jump. This instruction causes the MEM hardware to use the set of 32 map registers, referred to as the User map, for translating all programmed memory references. Prior to enabling the User map, the P-register is set to the effective memory address. As a result of executing this instruction, normal I/O interrupts are held off until the first opportunity following the fetch of the next instruction, unless three or more levels of indirect addressing are used. If the User map is already enabled, the instruction defaults to JMP *+1,I.

This instruction will normally generate a MEM violation when executed in the protected mode. In this case, the status of the MEM is not affected and the jump will not occur; however, if the System map is enabled, the instruction is allowed.

label	UJS	m [,I]	comments

Enable User map and jump to subroutine. This instruction causes the MEM hardware to use the set of 32 map registers, referred to as the User map, for translating all programmed memory references. Prior to enabling the User map, the P-register is set one count past the effective memory address. The return address is written into the location specified by *m* [,I]. As a result of executing this instruction, normal I/O interrupts are held off until the first opportunity following the fetch of the next instruction, unless three or more levels of indirect addressing are used. If the User map is already enabled, the instruction defaults to JMP *+1,I.

This instruction will normally generate a MEM violation when executed in the protected mode. In this case, the status of the MEM is not affected and the jump will not occur; however, if the System map is enabled, the instruction is allowed.

label	USA	comments

Load/store User map per A. This instruction transfers the 32 User map registers to or from memory. If bit 15 of the A-register is clear, the User map is *loaded* from memory starting from the address specified in bits 14-0 of the A-register. If bit 15 of the A-register is set, the User map is *stored* into memory starting at the address specified in bits 14-0 of the A-register. When load/store operation is complete, the A-register will be incremented by 32 to allow multiple instructions.

Note: If not in the protected mode, the MEM provides no protection against altering the contents of maps while they are currently enabled.

An attempt to load any map in the protected mode will cause a MEM violation. An attempt to store the User map is allowed within the constraints of write protected memory.

label	USB	comments

Load/store User map per B. This instruction transfers the 32 User map registers to or from memory. If bit 15 of the B-register is clear, the User map is *loaded* from memory starting from the address specified in bits 14-0 of the B-register. If bit 15 of the B-register is set, the User map is *stored* into memory starting at the address specified in bits 14-0 of the B-register. When the load/store operation is complete, the B-register will be incremented by 32 to allow multiple map instructions.

Note: If not in the protected mode, the MEM provides no protection against altering the contents of maps while they are currently enabled.

An attempt to load any map in the protected mode will cause a MEM violation. An attempt to store the User map is allowed within the constraints of write protected memory.

label	XCA*	m [,I]	comments

Cross compare A. This instruction compares the contents of the A-register with the contents of the addressed memory location. If the two 16-bit words are not identical, the next instruction is skipped; i.e., the P-register advances two counts instead of one count. If the two words are identical, the next instruction is executed. Neither the A-register nor memory cell contents are altered.

This instruction uses the alternate program map for the read operation. If neither the System map nor the User map is enabled (i.e., MEM is disabled), then a compare directly with physical memory occurs. This instruction will cause a MEM violation only if read protection rules are violated.

label	XCB*	m [,I]	comments

Cross compare B. This instruction compares the contents of the B-register with the contents of the addressed memory location. If the two 16-bit words are not identical, the next instruction is skipped; i.e., the P-register advances two counts instead of one count. If the two words are identical, the next instruction is executed. Neither the B-register contents nor memory cell contents are altered.

This instruction uses the alternate map for the read operation. If neither the System map nor the User map is enabled (i.e., MEM is disabled), then a direct compare with physical memory occurs.

This instruction will cause a MEM violation only if read protection rules are violated.

label	XLA*	m [,I]	comments

Cross load A. This instruction loads the contents of the specified memory address into the A-register. The contents of the memory cell are not altered.

This instruction uses the alternate program map to fetch the operand. If the MEM is currently disabled, then a load directly from physical memory occurs.

This instruction will cause a MEM violation only if read protection rules are violated.

label	XLB*	m [,I]	comments

Cross load B. This instruction loads the contents of the specified memory address into the B-register. The contents of the memory cell are not altered.

This instruction uses the alternate program map to fetch the operand. If the MEM is currently disabled, then a load directly from physical memory occurs.

This instruction will cause a MEM violation only if read protection rules are violated.

label	XMA	comments

Transfer maps internally per A. This instruction transfers a copy of the entire contents (32 map registers) of the System map or the User map to the Port A map or the Port B map as determined by the control word in the A-register, as follows:

Bit No.	Significance
15	0 = System map 1 = User map
0	0 = Port A map 1 = Port B map

(Bits 14-1 are ignored)

This instruction will always cause a MEM violation when executed in the protected mode.

label	XMB	comments

Transfer maps internally per B. This instruction transfers a copy of the entire contents (32 map registers) of the System map or the User map to the Port A map or the Port B map as determined by the control word in the B-register, as follows:

Bit No.	Significance
15	0 = System map 1 = User map
0	0 = Port A map 1 = Port B map

(Bits 14-1 are ignored)

This instruction will always generate a MEM violation when executed in the protected mode.

label	XMM	comments

Transfer map or memory. This instruction transfers a number of words either from sequential memory locations to sequential map registers or from maps to memory. Bits 0-9 of memory correspond to 0-9 of the map and bits 14 and 15 of memory relate to bits 10 and 11 of the map. The A-register points to the first register to be accessed and the B-register points to the starting address of the table in memory.

Maps are addressed as contiguous space and a wraparound count from 127 to 0 can and will occur. It is the programmer's responsibility to avoid this error. The X-register indicates the number of map registers to be transferred.

A positive number in X will cause the maps to be loaded with the corresponding data from memory. A negative (two's complement) number in X will cause the maps to be stored into the corresponding memory locations.

The instruction is interruptible after each group of 16 registers has been transferred. A, B and X are then reset to allow re-entry at a later time. The X-register will always be zero at the completion of the instruction; A and B will be advanced by the number of registers moved. An attempt to load any map register in Protected Mode will generate a MEM violation. An attempt to store map registers is allowed within the constraints of Write Protected memory.

label	XMS	comments

Transfer maps sequentially. This instruction transfers a number of words to sequential map registers. The A-register points to the first register to be accessed and

the B-register is the base quantity (page number). The X-register indicates the number of map registers to be affected. A positive quantity will cause the word found in the page number to be used as a base quantity to be loaded into the first register. The next register will be loaded with the base quantity plus one, and so forth up to the number of registers. Bits 0-9, 14 and 15 are used as described in XMM. An attempt to load any map register in Protected Mode will generate a MEM violation. An attempt to store map registers is allowed within the constraints of Write Protected memory.

label	XSA*	m [,I]	comments

Cross store A. This instruction stores the contents of the A-register into the addressed memory location. The previous contents of the memory cell are lost; the A-register contents are not altered.

This instruction uses the alternate program map for the write operation. If the MEM is currently disabled, then a store directly into physical memory occurs.

This instruction will always cause a MEM violation when executed in the protected mode.

label	XSB*	m [,I]	comments

Cross store B. This instruction stores the contents of the B-register into the addressed memory location. The previous contents of the memory cell are lost; the B-register contents are not altered.

This instruction uses the alternate program map for the write operation. If the MEM is currently disabled, then a store directly into physical memory occurs.

This instruction will always cause a MEM violation when executed in the protected mode.

3-27. HP 1000 FENCE REGISTERS

There are two separate fences available on the HP 1000 M, E and F-Series Computers: the memory protect fence and the base page fence.

The HP 1000 L-SERIES Computer has a memory protect fence.

The memory protect fence allows you to select a block of memory which will be protected against alteration by any programmed instruction. The memory protect fence register (which specifies the upper bound of the protected area) is loaded from the A- or B-register by an OTA or OTB instruction. See the appropriate hardware manual for specific details on the memory protect fence.

The base page fence is only available in HP 1000 M, E and F-Series computers which have the Dynamic Mapping System. This fence specifies which part of the base page is mapped. This determines where shared memory is separated from reserved memory on the base page. The base page fence register is loaded from the A- or B-register by an LFA or LFB instruction.

Instructions which modify the fence registers cannot be executed while the computer is in the protected mode.

3-28. HP 1000 M, E, F-SERIES INSTRUCTION REPLACEMENTS

The RTE-L/XL system library contains software substitutions for all the HP 1000 M, E, and F-Series CPU instructions that are not included in the HP 1000 L-SERIES instruction set except for the optional DMS instruction set which is not simulated on the HP 1000 L-SERIES hardware.

These instruction replacements should enable most user programs written in assembly language to be transported from a HP 1000 M, E, and F-Series computer to a HP 1000 L-SERIES by simply editing those instruction mnemonic codes into the JSB. <mmnemonic> format for the system library routines in RTE-L/XL.

If the user should happen to code an instruction that is not valid for the L-SERIES hardware, the Assembler will not report this fact as an error. The Assembler assembles the full HP 1000 instruction set. The L-SERIES instruction set is a subset of the HP 1000 instruction set. (See Appendix F for a summary of the valid instruction sets for the M, E, F and L-SERIES computers.) Since neither the Assembler nor the Loader will report unimplemented instructions the L-SERIES processor will trap and report as an error any instruction that is not valid for its instruction set. The following error will be reported on the system console:

```
Name ABORTED UI    Address
```

where:

`Name`	is the name of the program and
`Address`	is the address of the offending instruction.

3-29. REPLACEMENT FORMATS

The name of the software subroutine is formed by preceding the instruction mnemonic with a period (decimal point). The calling sequence is transformed as shown in Figure 3-4. All instructions that are recoded to use the software implementation must be declared as external to the program.

1-word instructions:

```
LABEL XYZ COMMENTS is edited to -->

LABEL JSB .XYZ COMMENTS
```

2-word instructions:

```
LABEL XYZ <operand> COMMENTS is edited to -->

LABEL JSB .XYZ COMMENTS
      DEF <operand>
```

3-word instructions (CBT, CMW, MBT, MVW):

```
LABEL MBT <operand> COMMENTS is edited to -->

LABEL JSB .MBT COMMENTS
      DEF <operand>
      DEC 0
```

3-word instructions (CBS, SBS, TBS):

```
LABEL CBS <operand 1> <operand 2> COMMENTS

      is edited to -->

LABEL JSB .CBS COMMENTS
      DEF <operand 1>
      DEF <operand 2>
```

Figure 3-4. HP 1000 M, E, F-Series Instruction Replacement Formats

3-30. HP 1000 M, E, F-SERIES SOFTWARE REPLACEMENTS

The following list represents the HP 1000 M, E, F-Series instructions that have software substitutions in the RTE-L/XL system library.

ONE WORD		TWO WORD		THREE WORD (2 operand)	THREE WORD (1 operand)
.CAX	.LBT	.ADX	.LDY	.CBS	.CBT
.CAY		.ADY		.SBS	.CMW
.CBX	.MBF		.SAX	.TBS	.MBT
.CBY	.MBI	.FAD	.SAY		.MVW
.CXA	.MBW	.FDV	.SBX		
.CXB	.MWF	.FMP	.SBY		
.CYA	.MWI	.FSB	.STX		
.CYB	.MWW		.STY		
		.JLY			
.DSX	.SBT	.JPY	.XCA		
.DSY	.SFB		.XCB		
		.LAX	.XLA		
.FIX	.XAX	.LAY	.XLB		
.FLT	.XAY	.LBX	.XSA		
	.XBX	.LBY	.XSB		
.ISX	.XBY	.LDX			
.ISY					

PSEUDO INSTRUCTIONS

SECTION IV

The pseudo instructions control the Assembler and its listed output, establish program relocatability, and define program linkage as well as specify various types of constants, blocks of memory, and labels used in the program.

4-1. ASSEMBLER CONTROL

The Assembler control pseudo instructions establish and alter the contents of the base page and program location counters, and terminate assembly processing. Labels may be used but they are ignored by the Assembler.

The NAM statement, which must be the first statement in an Assembler source program, includes optional parameters defining the program type, priority, and time values. This information is used to fill in the NAM record of the program module (see Appendix H for the format of the NAM record).

NAM	name [,type,pri,res,mult,hr,min,sec,msec id]

name
is the name of the program.

type (RTE-IV)
is the program type. (Defaults to 3 if not specified):

0 — system program or driver.

1 — memory resident.

2 — real-time disc resident.

3 — background disc resident.

4 — background disc resident with Table Area II access.

5 — program segment (RT or BG).

6 — library, reentrant or privileged subroutines (note that if called by a memory resident program, these routines are relocated into the Memory Resident Library. After memory resident loading they become Type 7).

7 — library, utility subroutines (appended to calling program).

8 — if program is a main, it is deleted from the system,

or,

if program is a subroutine, it is used to satisfy any external references during generation; however, it is not loaded in the relocatable library area of the disc.

13 — (Table Area II) system entry points that contain pointers and system values that are defined at generation. Table Area II is a combination of these relocated Type 13 modules and system tables that are built by the generator.

14 — same as Type 6, but automatically included in the Memory Resident Library. They become Type 7 after memory resident loading.

15 — (Table Area I) system entry points that must be included in the system and user maps. Table Area I is a combination of these relocated Type 15 modules and I/O tables that are built by the generator.

30 — Subsystem Global Area (SSGA).

Note: In some cases the primary type code (i.e., 1, 2, 3, 4) may be expanded by adding 8, 16, or 24 to the number. These expanded types allow such features as: access to real-time COMMON by background programs, and access to SSGA.

CAUTION

The primary type code of a main program and its segments must not be changed because the relationship between the program and its segments would be lost.

type (RTE-L/XL)
is the program type. The only significant program types in RTE-L are:

5 — program segment (RT or BG). The RTE-L generator will not load segmented programs. They must be loaded on-line.

6 — library, reentrant or privileged subroutines. The user may direct the generator to relocate the program into the Memory Resident Library or System Common. If so, programs that reference this code will not have the subroutine appended to it, they will use the memory resident code. If not loaded in the Memory Resident Library or System Common, a copy of the code will be appended to each referencing program.

7 — library, utility subroutines (appended to calling program.) If the routine is included in the System Relocation phase of a generation, all entry points are not retained in the snap file.

All other program types are not significant in RTE-L/XL. Determining whether the program is Real-Time or Background is made at load time by the appropriate LOADR or GENERATOR command.

pri
is the priority (1 to 32767, set to 99 if not given).

res
is the resolution code

mult
is the execution multiple

hr
is hours

min
is minutes

sec
is seconds

msec
is tens of milliseconds

id
comments field (separated from operand by a space)

COMMENTS

The parameters of the NAM statement, beginning with *type* and ending with *msec,* are separated by commas. A blank space within the parameter field will terminate that field and cause the Assembler to recognize the next entry as the comment field (*id*). The first parameter must be separated from the program *name* by a comma. The parameters are optional, but to specify any particular parameter, those preceding it must also be specified, as shown below:

```
NAM    EX1,2,99,1,999,10,20,30,30
NAM    EX2,1,10    THIS IS ID OF PROGRAM.
```

Starting immediately after the first blank, the identifier field is placed in the relocatable NAM record following the parameters (a blank space separates the parameter and comment fields). In the following example a part number is shown in the comments field of the second line:

```
NAM    PRGRM     THIS IS ON RELOC. RECORD
NAM    MYNAM,1,9,4    25117-80345B
```

The identifier (comments) field (*id*)can be a maximum of 73 characters due to the restriction of the source statement size. The identifier will be truncated after column 80.

	ORB	comments

ORB defines the portion of a relocatable program that must be assigned to the base page by the Assembler. The Label field (if given) is ignored, and the statement requires no operand. All statements that follow the ORB statement are assigned contiguous locations in the base page. Assignment to the base page terminates when the Assembler detects an ORG, ORR, or END statement.

When more than one ORB is used in a program, each ORB causes the Assembler to resume assigning base page locations at the address following the last assigned base page location. An example is shown in figure 4-1.

An ORB statement in an absolute program has no significance and is flagged as an error.

	ORG	m	comment

The ORG statement defines the origin of an absolute program, or the origin of subsequent sections of absolute or relocatable programs.

An absolute program must begin with an ORG statement. The operand m, must be a decimal or octal integer specifying the initial setting of the program location counter.

```
          NAM PROG     ASSIGN ZERO AS RELATIVE STARTING
            .          LOCATION FOR PROGRAM PROG.
            .
            .
          ORB          ASSIGN ALL FOLLOWING STATEMENTS
                       TO BASE PAGE.
      IAREA BSS 100
            .
            .
          ORR          CONTINUE MAIN PROGRAM.
            .
            .
          ORB          RESUME ASSIGNMENT AT NEXT
            .          AVAILABLE LOCATION IN BASE PAGE.
            .
            .
          ORR          CONTINUE MAIN PROGRAM.
```

Figure 4-1. ORB Example

ORG statements may be used elsewhere in the program to define starting addresses for portions of the object code. For absolute programs the Operand field, m, may be any expression. For relocatable programs, m must not be common relocatable or absolute. An expression is evaluated modulo 2^{15}. Symbols must be previously defined. All instructions following an ORG are assembled at consecutive addresses starting with the value of the operand.

	ORR	comment

ORR resets the program location counter to the value existing when an ORG or ORB instruction was encountered. An example is shown in figure 4-2.

More than one ORG statement may occur before an ORR is used. If so, when the ORR is encountered, the program location counter is reset to the value it contained when the first ORG of the string occurred. An example is shown in figure 4-3.

If a second ORR appears before an intervening ORG or ORB the second ORR is ignored.

```
        NAM RSET        SET PLC TO VALUE OF ZERO, ASSIGN
FIRST   ADA             RSET AS NAME OF PROGRAM.
          .
          .
          .
        ADA CTRL        ASSUME PLC AT FIRST+2280.
        ORG FIRST+2926  SAVE PLC VALUE OF FIRST+2280
          .             AND SET PLC TO FIRST+2926.
          .
          .
        JMP EVEN+1      ASSUME PLC AT FIRST+3004
        ORR             RESET PLC TO FIRST+2280.
```

Figure 4-2. ORR Example (with Single ORG)

```
        NAM RSET        SET PLC TO ZERO
FIRST   ADA
          .
          .
          .
        LDA WYZ         ASSUME PLC AT FIRST+2250
        ORG FIRST+2500  SET PLC TO FIRST+2500
          .
          .
          .
        LDB ERA         ASSUME PLC AT FIRST+2750
        ORG FIRST+2900  SET PLC TO FIRST+2900
          .
          .
          .
        CLE             ASSUME PLC AT FIRST+2920
        ORR             RESET PLC TO FIRST+2250
```

Figure 4-3. ORR Example (with Multiple ORGs)

The IFN and IFZ pseudo instructions cause the inclusion of instructions in a program provided that either an "N" or "Z", respectively, is specified as a parameter for the ASMB control statement.† The IFN or IFZ instruction precedes the set of statements that are to be included. The pseudo instruction XIF serves as a terminator. If XIF is omitted, END acts as a terminator to both the set of statements and the assembly.

```
        IFN     comments
         .
         .
         .
        XIF
```

All source language statements appearing between the IFN and the XIF pseudo instructions are included in the program if the character "N" is specified on the ASMB control statement.

All source language statements appearing between the IFZ and XIF pseudo instructions are included in the program if the character "Z" is specified on the ASMB control statement.

```
        IFZ     comments
         .
         .
         .
        XIF
```

When the particular letter is not included on the control statement, the related set of statements appears on the Assembler output listing but is not assembled.

Any number of IFN-XIF and IFZ-XIF sets may appear in a program, however, they may not overlap. An IFZ or IFN intervening between an IFZ or IFN and the XIF terminator results in a diagnostic being issued during compilation; the second pseudo instruction is ignored.

Both IFN-XIF and IFZ-XIF pseudo instructions may be used in the program; however, only one type will be selected in a single assembly. Therefore, if both characters "N" and "Z" appear in the control statement, the character which is listed last will determine the set of coding that is to be assembled. Some examples are shown in figures 4-4 and 4-5.

In figure 4-4, the program TRAVL will perform computations involving either or neither CAR or PLANE considerations depending on the presence or absence of Z or N parameters in the Control Statement.

In figure 4-5, the program WAGES computes a weekly wage value. Overtime consideration will be included in the program if "Z" is included in the parameters of the Control Statement.

```
        NAM TRAVL
          .
          .
          .
        IFZ
        LDA CAR
        CMA,SZA
        JMP NO.GO
        LDA MILES
        DIV SPEED
        STA GAS
        XIF
          .
          .
          .
        IFN
        LDA PLANE
        CMA,SZA
        JMP NO.GO
        LDA TIME
        CPA COST
        XIF
NO.GO   HLT 77
          .
          .
          .
        END
```

Figure 4-4. IFN/XIF and IFZ/XIF Example

```
        NAM WAGE
          .
          .
          .
        JSB HOUR
        MPY TIME1
        IFZ
        JSB OVTIM
        MPY TIME2
          .
          .
          .
TIME1   DEC 40
TIME2   BSS 1
        END
```

Figure 4-5. IFZ/XIF Example

†See "Assembly Options" in Section I of this manual.

The REP pseudo instruction causes the repetition of the statement immediately following it a specified number of times.

label	REP	n	comments

The statement following the REP in the source program is repeated n times. The n may be any absolute expression. Comment lines (indicated by an asterisk in character position 1) are not repeated by REP. If a comment follows a REP instruction, the comment is ignored and the instruction following the comment is repeated.

A label specified in the REP pseudo instruction is assigned to the first repetition of the statement. A label should not be part of the instruction to be repeated; it would result in a doubly defined symbol error.

Example:

```
        CLA
TRIPL   REP   3
        ADA   DATA
```

The above source code would generate the following:

```
        CLA             Clear the A-Register;
                        the content of DATA is
TRIPL   ADA   DATA      tripled and stored in the
        ADA   DATA      A-Register.
        ADA   DATA
```

Example:

```
FILL    REP   100B
        NOP
```

The example above loads 100_8 memory locations with the NOP instruction. The first location is labeled FILL.

Example:

```
        REP   2
        MPY   DATA
```

The above source code would generate the following:

```
        MPY   DATA
        MPY   DATA
```

	END	[m]	comments

This statement terminates the program; it marks the physical end of the source language statements. The Operand field, m, may contain a name appearing as a statement label in the current program or it may be blank. If this is a main program m must be specified. If a name is entered, it identifies the entry point to the module.

If the Operand field is blank, the Comments field must be blank also, otherwise, the Assembler attempts to interpret the first five characters of the comments as the transfer address symbol.

The label field of the END statement is ignored.

4-2. OBJECT PROGRAM LINKAGE

Linking pseudo instructions provides a means for communication between a main program and its subroutines or among several subprograms that are to be run as a single program. These instructions may be used only in a relocatable program.

The Label field of this class is ignored in all cases. The Operand field is usually divided into many subfields, separated by commas. In the case of the COM pseudo instruction, the first space not preceded by a comma or a left parenthesis terminates the entire field.

	COM	$name_1[(size_1)]$ $[,name_2[(size_2)]$, . . . , $name_n[(size_n)]]$	comments

COM reserves a block of storage locations that may be used in common by several subprograms. Each name identifies a segment of the block for the subprogram in which the COM statement appears. The sizes are the number of words allotted to the related segments. The size is specified as an octal or decimal integer. If the size is omitted, it is assumed to be one.

Any number of COM statements may appear in a subprogram. Storage locations are assigned contiguously; the length of the block is equal to the sum of the lengths of all segments named in all COM statements in the subprogram.

To refer to the common block, other subprograms must also include a COM statement. The segment names and sizes may be the same or they may differ. Regardless of the names and sizes specified in the separate subprograms, there is only one common block for the combined set. It has the same relative origin; the content of the n^{th} word or common storage is the same for all subprograms. An example is shown in figure 4-6.

The LDA instructions in the two subprograms each refer to the same location in common storage, location 7.

```
PROG1  COM  ADDR1(5),ADDR2(10),ADDR3(10)
             .
             .
             .
       LDA  ADDR2+1     PICK UP SECOND WORD OF SEGMENT
             .          ADDR2
             .
       END
             .
             .
PROG2  COM  AAA(2),AAB(2),AAC,AAD(20)
             .
             .
       LDA  AAD+1       PICK UP SECOND WORD OF SEGMENT
                        AAD
```

Organization of common block:

PROG1 name	PROG2 name	Common Block
ADDR1	AAA	(location 1)
		(location 2)
	AAB	(location 3)
		(location 4)
	AAC	(location 5)
ADDR2	AAD	(location 6)
		(location 7)
		(location 8)
		(location 9)
		(location 10)
		(location 11)
		(location 12)
		(location 13)
		(location 14)
		(location 15)
ADDR3		(location 16)
		(location 17)
		(location 18)
		(location 19)
		(location 20)
		(location 21)
		(location 22)
		(location 23)
		(location 24)
		(location 25)

Figure 4-6. COM Examples

The segment names that appear in the COM statements can be used in the Operand fields of DEF, ABS, EQU, ENT or any memory reference statement; they may not be used as labels elsewhere in the program.

The loader establishes the origin of the common block; the origin cannot be set by the ORG pseudo instruction. All references to the common area are relocatable.

Two or more subprograms may declare common blocks that differ in size. The subprogram that defines the largest block must be the first submitted for loading.

4-3. PROGRAM AND SYSTEM COMMON

All common specified in an assembly language program with the COM statement is stored in either the program common area or system common area.

The use of system or program common is specified at load time using the appropriate loader command.

Under RTE operating systems other than RTE-L the generator will load those programs using common, so that the common is system common.

Under the RTE-L operating system, by default the generator will load those programs using common, so that the common is program common. If the user wants to use system common, he should use the LOD statement with the appropriate loader command. See the section on RTE-L PSEUDO INSTRUCTIONS and the RTE-L RELOCATING LOADER REFERENCE MANUAL for specific details on the use of the LOD instruction and loader commands. The user can optionally specify to the RTE-L generator that the common be system common instead of program common.

Unlike program common, system common is shared by all other programs on the system. This implies a great deal of coordination. Any program that uses system common should know how all other programs use system common. Only in this fashion can the programmer ensure that his data in system common will be secure. If the programmer does not know the interaction process of other programs who use system common, he should use program common.

ENT	$name_1$ [,$name_2$,...,$name_n$]	comments

ENT defines entry points to the program or subprogram. Each name is a symbol that is assigned as a label for some machine operation in the program. Entry points allow another subprogram to refer to this subprogram. All entry points must be defined in the program.

Symbols appearing in an ENT statement may not also appear in an EXT statement in the same subprogram. Labels defined as absolute by EQU statements or defined by COM statements may be declared as entry points.

EXT	$name_1$ [,$name_2$,...,$name_n$]	comments

This instruction designates labels in other subprograms that are referenced in this subprogram. The symbols must be defined as entry points by the other subprograms.

The symbols defined in the EXT statement may appear in memory reference statements, certain I/O statements or EQU or DEF pseudo instructions. An external symbol may be used with a + or − offset or specified as indirect. References to external locations are processed by the loader as indirect addresses linked through the base page or in some cases through a current page link.

Symbols appearing in EXT statements may not also appear in ENT or COM statements in the same subprogram. The label field is ignored. Examples of the use of EXT and ENT are shown in figures 4-7 through 4-10.

The RPL pseudo instruction is used to define a code replacement record for the RTE system generator or RTE relocating loader.

label	RPL	m	comments

The instructions to be replaced must be of the form = JSB SUB where SUB is an external reference. The JSB SUB will be replaced by the octal value of the RPL definition whenever it is encountered by the generator or loader. Examples are shown in figure 4-11.

```
PROGA NOP
      LDA SAMD       SAMD AND SAND ARE REFERENCED IN
       .             PROGA, BUT ARE ACTUALLY
       .             LOCATIONS IN PROGB.
       .
      JMP SAND
      EXT SAMD,SAND
      ENT PROGA
      END
       .
       .
PROGB NOP
       .
       .
SAMD  OCT 767
SAND  STA SAMD
       .
       .

      ENT SAMD,SAND
       .
       .
      JSB PROGA
       .
       .
      EXT PROGA
       .
       .
      END
```

Figure 4-7. ENT/EXT Examples

```
EXT BUF,PTR
 .
 .
 .
LDA BUF+1    EXTERNAL WITH + OR - OFFSET.
STA PTR,I    EXTERNAL INDIRECT.
```

Figure 4-8. EXT with Offset

```
            ENT CHAN,CMLBL
             .
             .
             .
      CHAN  EQU 12B
            COM CMLBL  (20)
```

Figure 4-9. ENT in COMmon and ENT Defining
An External I/O Reference

```
ASMB,R,B,L
       NAM MAIN
*
*      DECLARE CHAN1,CHAN2 AS ENTRY POINTS
*
       ENT CHAN1,CHAN2
       EXT OUTPT,INPUT

START  JSB INPUT  INPUT A CHARACTER
       JSB OUTPT  OUTPUT TO DEVICE 2
       LIA 1B     READ SWITCH REGISTER
       SSA        IS BIT 15 ON?
       HLT 55B    YES, HALT
       JMP START  DO ANOTHER ONE
*
*      DEFINE THE I/O CHANNELS FOR THE DRIVERS INPUT,OUTPT BY
*      SETTING THE LABELS CHAN1,CHAN2 EQUIVALENT TO THE ABSOLUTE
*      LOCATIONS 10,11.

CHAN1  EQU 10B
CHAN2  EQU 11B
       END START
```

```
ASMB,R,B,L
       NAM IOPRG
*      SUBROUTINE ENTRY POINTS
       ENT INPUT,OUTPT
*      DECLARE I/O CHANNELS TO BE EXTERNAL
       EXT CHAN1,CHAN2
*
*      INPUT      SUBROUTINE
*
INPUT  NOP
       STC CHAN1,C     SET CONTROL ON CHANNEL 1
       SFS CHAN1
       JMP *-1    WAIT ON FLAG
       LIA CHAN1  LOAD WORD
       JMP INPUT,I     RETURN
*
*      OUTPUT SUBROUTINE
*
OUTPT  NOP
       OTA CHAN2 OUTPUT WORD
       STC CHAN2,C     STROBE TO DEVICE
       SFS CHAN2
       JMP *-1
       JMP OUTPT,I     RETURN

       END
```

Figure 4-10. EXT, ENT for I/O Channel

```
.FAD RPL   105000B
IFIX RPL   105100B
```

```
        EXT    .FAD
                :
                :
        JSB    .FAD
                :
                :
        JSB    IFIX
                :
                :
```

Figure 4-11. Label RPL Octal Value

The relocation of the program would result in the following:

```
                :
                :
105000
                :
                :
105100
                :
                :
```

Note that the instruction value is 105000 instead of 114XXX.

Note that the instruction value is 105100 instead of 114XXX.

4-4. ADDRESS AND SYMBOL DEFINITION

The pseudo operations in this group assign a value, a word address, or a byte address to a symbol which is used as an operand elsewhere in the program.

label	DEF	m [,I]	comments

The address definition statement generates one word of memory as a 15-bit address which may be used as the object of an indirect address found elsewhere in the source program. The symbol appearing in the label is that which is referenced; it appears in the Operand field of a Memory Reference instruction.

The operand field of the DEF statement may be any positive expression in an absolute program; in a relocatable program it may be a relocatable expression or an absolute expression with a value of less than 2000_8. Symbols that do appear in the Operand field may appear as operands of EXT or COM statements, in the same subprogram and as entry points in other subprograms.

The expression in the Operand field may itself be indirect and make reference to another DEF statement elsewhere in the source program. Some examples are shown in figure 4-12.

The DEF statement provides the necessary flexibility to perform address arithmetic in programs which are to be assembled in relocatable form. *Relocatable programs should not modify the operand of a memory reference instruction.* Figure 4-13 illustrates what *not* to do. If TBL and LDTBL are in different pages, the Loader processes TBL as an indirect address linked through the base page. The ISZ erroneously increments the Loader-provided link to the base page rather than the value of TBL. Assuming that the loader assigns the absolute locations shown in figure 4-14, the ISZ will index the contents of location 2000_8 which is a LDA 700,I, and change it to LDA 701,I. Now we will use whatever happens to be in 701 rather than the link we intended to use which is in 700. We change the *link* instead of its *contents*.

```
LDTBL LDA TBL
        .
        .
        .
      ISZ LDTBL
        .
        .
        .
TBL   BSS 100
```

Figure 4-13. Example of Incorrect Address Modification

```
        NAM PROGN     ZERO-RELATIVE START OF PROGRAM.
        EXT SINE,SQRT
        COM SCMA(20),SCMB(50)
         .
         .
        JSB SINE      EXECUTE SINE ROUTINE
         .
         .
        LDA XCMA,I    PICK UP COMMON WORD INDIRECTLY.
         .
         .
XCMA    DEF SCMA      SCMA IS A 15-BIT ADDRESS.
         .
         .
        JSB XSQ,I     GET SQUARE ROOT USING TWO-LEVEL
XSQ     DEF XSQR,I    INDIRECT ADDRESSING.
         .
         .
XSQR    DEF SQRT      SQRT IS A 15-BIT ADDRESS.
        END PROGN
```

Figure 4-12. DEF Examples

INSTRUCTION			PAGE	ABSOLUTE LOCATION OF CODE	OPCODE	OPERAND (PAGE) & LOCATION
(Loader-assigned indirect link on base page)			(0)	(700)	DEF	4000
LDTBL	LDA	TBL	(1)	(2000)	LDA	(0) 700 (I)
	ISZ	LDTBL	(1)	(3000)	ISZ	(1) 2000
TBL	BSS	100	(2)	(4000)	BSS	

Figure 4-14. Loader-Assigned Locations for Figure 4-13

The example shown in figure 4-15 assures correct address modification during program execution. Assume that the sequence shown in figure 4-15 is assigned (by the loader) the absolute locations shown in figure 4-16. The LDA 2000,I picks up the *contents* of the location pointed to by ITBL (location 4000_8). The ISZ 2000 indexes the pointer DEF 4000 to point to 4001. The next LDA will reference location 4001, DEF TBL+1. This is what we intend.

label	ABS	m	comments

ABS defines a 16-bit absolute value to be stored at the location represented by the label. The Operand field, m, may be any absolute expression; a single symbol must be defined as absolute elsewhere in the program. Examples are shown in figure 4-17.

```
ITBL   DEF TBL
LDTBL  LDA ITBL,I
         .
         .
         .
       ISZ ITBL
         .
         .
         .
TBL    BSS 100
```

Figure 4-15. Example of Correct Address Modification

INSTRUCTION			PAGE	ABSOLUTE LOCATION OF CODE	OPCODE	OPERAND (PAGE) & LOCATION
ITBL	DEF	TBL	(1)	(2000)	DEF	4000
	LDA	ITBL,I	(1)	(2001)	LDA	(1) 2000,I
	ISZ	ITBL	(1)	(3000)	ISZ	(1) 2000
	TBL	BSS 100	(2)	4000	BSS	

Figure 4-16. Loader-Assigned Locations for Figure 4-15

```
AB    EQU  35        ASSIGNS THE VALUE OF 35
                     TO THE SYMBOL AB
M35   ABS  -AB       M35  CONTAINS -35.
P35   ABS  AB        P35  CONTAINS 35.
P70   ABS  AB+AB     P70  CONTAINS 70.
P30   ABS  AB-5      P30  CONTAINS 30.
P36   ABS  36        P36  CONTAINS 36.
```

Figure 4-17. ABS Examples

label	EQU	m	comments

The EQU pseudo operation assigns to a symbol a value other than the one normally assigned by the program location counter. The symbol in the Label field is assigned the value represented by the Operand field. The Operand field may contain any expression. The value of the operand may be common, base page or program relocatable as well as absolute, but it should not be negative. Symbols appearing in the operand must be previously defined in the source program.

The EQU instruction may be used to symbolically equate two locations in memory, or it may be used to give a value to a symbol. The EQU statement does not result in a machine instruction. Some examples are shown in figures 4-18 and 4-19.

label	DBL	m	comments

label	DBR	m	comments

Define Left Byte and Define Right Byte. The DBL and DBR pseudo instructions each generate one word of memory which contains a 16-bit byte address. For DBL, the byte address being defined is the left half (bits 8-15) of word location *m;* for DBR, it is the right half (bits 0-7). Indirect addressing may *not* be used. A *byte address* is defined as two times the word address of the memory location containing the particular byte. If the byte location is the left half of the memory location (bits 8-15), bit 0 of the byte address is clear; if the byte location is the right half of the memory lcoation (bits 0-7), bit 0 of the byte address is set. In an absolute program, *m* may be any positive expression. In a relocatable program, *m* may be any absolute expression with a value less than 200_8 or any relocatable expression. The generated word may be referenced (via *label*) in the Operand field of LDA and LDB instructions elsewhere in the source program for the purpose of loading byte addresses into the A- and B-registers.

CAUTION

Care must be taken when using the *label* of a DBL or DBR pseudo instruction as an indirect address elsewhere in the source program. The programmer must keep track of whether he is using word addresses or byte addresses.

```
        NAM  FAM
         .
         .
         .
J3      BSS  2          SET ASIDE TWO WORDS FOR STORAGE
         .
         .
         .
        LDA  J3
        ADA  ONE
        STA  J3+1
JFOUR   EQU  J3+1       THE SYMBOLS JFOUR AND J3+1 BOTH IDENTIFY
         .              THE SAME LOCATION. THE "AND" OPERATION
         .              IS PERFORMED ON THIS LOCATION.
         .
MWH     AND  JFOUR
         .
```

Figure 4-18. EQU Example

```
        NAM STOTB
         .
         .
         .
        COM TABLA(10) DEFINES A 10 WORD TABLE, TABLA.
         .
         .
         .
TABLB   EQU TABLA+5    NAMES WORDS 6 THROUGH 10 OF
         .             TABLA AS TABLB.
         .
         .
         .
        LDA TABLB+1    LOADS CONTENTS OF 7TH WORD
         .             COMMON INTO A. THE STATEMENT LDA
         .             TABLA+6 WOULD PERFORM THE SAME
         .             OPERATION
         .
        NAM REG
         .
         .
         .
A       EQU 0          DEFINES SYMBOL A AS 0 (LOCATION
B       EQU 1          OF A-REGISTER), AND SYMBOL B AS
         .             1 (LOCATION OF B-REGISTER).
         .
        LDA B          LOADS CONTENTS OF B-REGISTER
                       INTO A-REGISTER.
```

Figure 4-19. EQU Examples

Examples:

```
BYT1    DBL   WORD1
BYT2    DBR   WORD1
          .
          .
          .
WORD1   NOP
```

If WORD1 has the relocatable address 2002_8, then BYT1 will contain the relocatable value 4004_8 and BYT2 will contain the relocatable value 4005_8.

4-5. CONSTANT DEFINITION

The pseudo instructions in this class enter a string of one or more constant values into consecutive words of the object program. The statements may be named by labels so that other program statements can refer to the fields generated by them.

label	ASC	n, <2n characters>	comments

ASC generates a string of 2n alphanumeric characters in ASCII code into n consecutive words.† One character is right justified in each eight bits; the most significant bit is zero. n may be any expression resulting in an unsigned decimal value in the range 1 through 28. Symbols used in an expression must be previously defined. Anything in the Operand field following 2n characters is treated as comments. If less than 2n characters are detected before the end-of-statement mark, the remaining characters are assumed to be spaces, and are stored as such. The label represents the address of the first two characters. An example is shown in figure 4-20.

label	DEC	$d_1[.d_2,...,d_n]$	comments

DEC records a string of decimal constants into consecutive words. The constants may be either integer or real (floating point), and positive or negative. If no sign is specified, positive is assumed. The decimal number is converted to its

†To enter the code for the ASCII symbols which perform some action (e.g., CR and LF), the OCT pseudo instruction must be used.

```
TTYP   ASC  3,ABCDE
```

causes the following:

NOTE
Columns 7 and 15 below will contain zeros.

ALPHABETIC

	15 14	8 7 6	0
TTYP	A		B
	C		D
	E		(space)

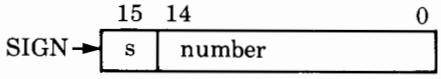
EQUIVALENT IN OCTAL NOTATION

	15 14	8 7 6	0
TTYP	1 0 1		1 0 2
	1 0 3		1 0 4
	1 0 5		0 4 0

Figure 4-20. ASC Example

binary equivalent by the Assembler. The label, if given, serves as the address of the first word occupied by the constant.

A decimal integer must be in the range of -2^{15} to $2^{15}-1$. It is converted into one binary word and appears as follows:

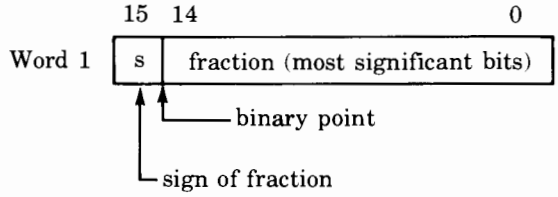


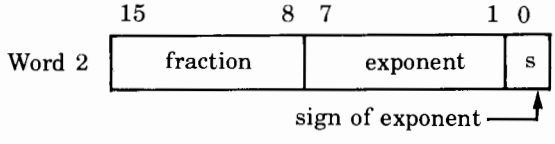
Some examples are shown in figure 4-21.

A floating point number has two components, a fraction and an exponent. The exponent specifies the power of 10 by which the fraction is multiplied. The fraction is a signed or unsigned number which may be written with or without a decimal point. The exponent is indicated by the letter E and follows a signed or unsigned decimal integer. The floating point number may have any of the following formats:

±n.n ±n. ±n.nE±e ±.nE±e ±n.E±e ±nE±e

The number is converted to binary, normalized (leading bits differ), and stored in two computer words. If either the fraction or the exponent is negative, that part is stored in two's complement form.





```
INT    DEC 50,+328,-300,+32768,-32768
```

causes the following (octal representation)

	15	14				0
INT	0	0	0	0	6	2
	0	0	0	5	1	0
	1	7	7	3	2	4
	1	0	0	0	0	0
	1	0	0	0	0	0

Note: The values $\pm 2^{15}$ ($\pm$32768) are both converted to 100000_8.

Figure 4-21. DEC Examples (Integer)

```
DEC .45E1
DEC 45.00E-1
DEC 4500E-3
DEC 4.5
```

are all equivalent to

$.45 \times 10^1$

and are stored in normalized form as:

15	14														0
0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0

15							8	7						1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0

Figure 4-22. DEC Examples (Floating Point)

```
DEC -.695,400E-4
```

are stored as:

1	0	1	0	0	1	1	1	0	0	0	0	1	0	1	0
0	0	1	1	1	0	1	1	0	0	0	0	0	0	0	0

0	1	0	1	0	0	0	1	1	1	1	0	1	0	1	1
1	0	0	0	0	1	0	1	1	1	1	1	1	0	0	1

Figure 4-23. DEC Examples (Floating Point)

The floating point number is made up of a 7-bit exponent with sign and a 23-bit fraction with sign. The number must be in the approximate range of 10^{-38} to 10^{+38}. Examples are shown in figures 4-22 and 4-23.

label	DEX	$d_1[,d_2,...,d_n]$	comments

DEX records a string of extended precision decimal constants into consecutive words within a program. Each such extended precision constant occupies three words as shown in figure 4-24.

An extended precision floating point number is made up of a 39-bit fraction and sign and a 7-bit exponent and sign.

This is the only form used for DEX. All values, whether they be floating point, integer, fraction, or integer and fraction, will be stored in three words as just described. This storage format is basically an extension of that used for DEC, as previously described. Some examples are shown in figure 4-25.

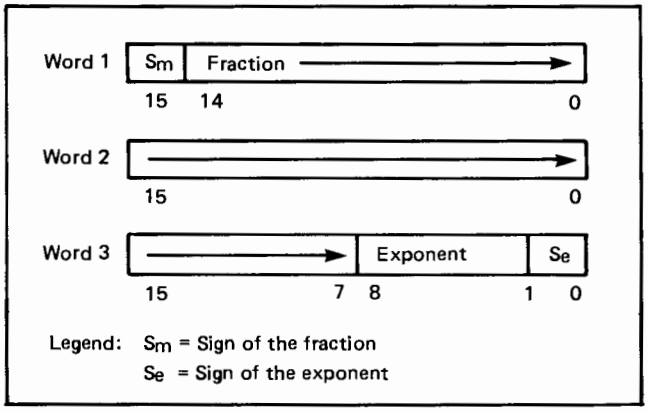


Figure 4-24. DEX Memory Format

label	DEY	$d_1, [,d_2,...,d_n]$	comments

DEY records a string of extended precision decimal constants into consecutive words within a program. Each such extended precision constant occupies four words as shown in Figure 4-24A.

A four-word extended precision floating point number is made up of a 55-bit fraction and sign and a 7-bit exponent and sign.

This storage format is basically an extension of that used for DEX, as previously defined.

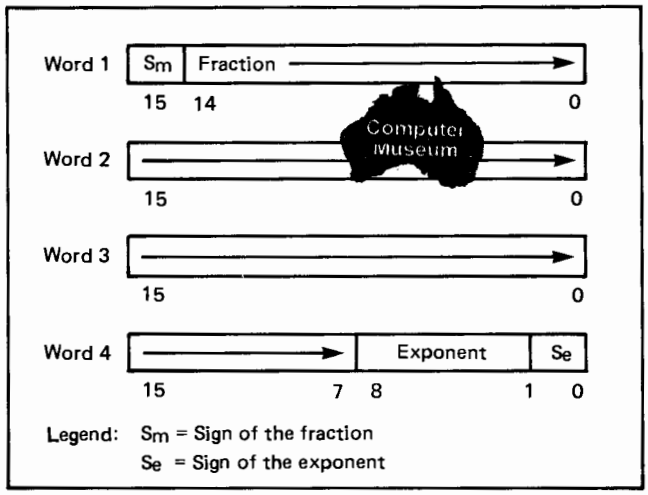


Figure 4-24A. DEY Memory Format

label	OCT	$o_1[,o_2,...,o_n]$	comments

OCT stores one or more octal constants in consecutive words of the object program. Each constant consists of one to six octal digits (0 to 177777). If no sign is given, the sign is assumed to be positive. If the sign is negative, the two's complement of the binary equivalent is stored. The constants are separated by commas; the last constant is terminated by a space. If less than six digits are indicated for a constant, the number is right justified in the word. A label, if used, acts as the address of the first constant in the string. The letter B must not be used after the constant in the Operand field; it is significant only when defining an octal term in an instruction other than OCT. Some examples are shown in figure 4-26.

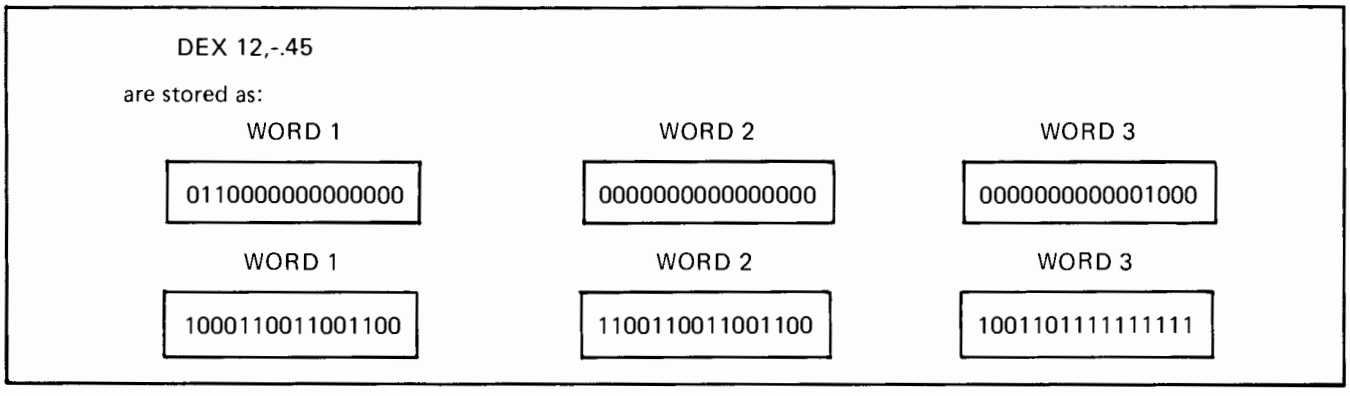


Figure 4-25. DEX Examples

```
        OCT +0
        OCT -2
NUM     OCT 177,20405,-36
        OCT 51,77777,-1,10101
        OCT 107642,177077
        OCT 1976                        ILLEGAL: CONTAINS DIGIT 9
        OCT -177777
        OCT 177B                        ILLEGAL : CONTAINS CHARACTER B
```

The above statements are stored as follows:

	15	14				0
	0	0	0	0	0	0
	1	7	7	7	7	6
NUM	0	0	0	1	7	7
	0	2	0	4	0	5
	1	7	7	7	4	2
	0	0	0	0	5	1
	0	7	7	7	7	7
	1	7	7	7	7	7
	0	1	0	1	0	1
	1	0	7	6	4	2
	1	7	7	0	7	7
	X	X	X	X	X	X
	0	0	0	0	0	1
	X	X	X	X	X	X

The result of attempting to define an illegal constant is unpredictable

Figure 4-26. OCT Examples

label	BYT	b_1,b_2, ... b_n	comments

Define Octal Byte constants. The BYT pseudo instruction generates octal constants in consecutive byte locations of memory. Each constant in the Operand field (b_1,b_2, ... b_n) consists of one to three octal digits, must be within the range 0 through 377, and may be preceded by a plus (+) or minus (−) sign. If a constant is not signed, it is assumed to be positive. If a constant is negative, the two's complement of the binary equivalent (truncated to eight bits) is stored. If the Operand field contains an odd number of constants, bits 0-7 of the final word generated will be clear (zeros). Since the constants are assumed to be octal, the letter "B" must not be used. Some examples are shown in figure 4-27.

4-6. STORAGE ALLOCATION

The storage allocation statements reserve a block of memory for data or for a work area.

label	BSS	m	comments

The BSS pseudo operation advances the program or base page location counter according to the value of the operand. The Operand field may contain any expression that results in a positive integer. Symbols, if used, must be previously defined in the program. The label, if given, is the name assigned to the storage area and represents the address of the first word. The initial content of the area set aside by the statement is unaltered by the loader.

label	EMA	m_1,m_2	comments

The EMA pseudo-instruction defines an extended memory area (EMA) where m_1 is the EMA size in pages and m_2 is the mapping segment (MSEG) size in pages. m_1 and m_2 must be expressions that evaluate to non-relocatable integers. m_1 must be in the range 0 to 1023 inclusive and m_2 must be in the range 0 to 31 inclusive. If m_1 evaluates to 0 the maximum possible size for EMA will be assigned at dispatch time. If m_2 evaluates to 0 the maximum possible size for MSEG will be assigned at load time.

The EMA pseudo-instruction may only be used in a relocatable program. Only one EMA pseudo-instruction per program is allowed.

An EMA pseudo-instruction must have a label which is the name assigned to the storage area. This label represents the logical address of the first word in the MSEG and is determined at load time. EMA labels may appear in memory reference statements, and in EQU or DEF pseudo-instructions.

References to EMA labels are processed as indirect addresses through a base page link at load time. EMA labels may be referenced in other subprograms or segments by declaring them as externals in the other subprograms or segments. They should not be declared as entry points in the program in which they appear.

The following restrictions apply to the use of EMA labels:

1. EMA labels may not be used with an offset.
2. EMA labels may not be used with indirect.
3. EMA labels may not appear in an ENT or COM statement in the same subprogram.

```
ALF    BYT 50,377,-10,2,-312
```

causes the following (octal representation):

	15	14				0
ALF	0	2	4	3	7	7
	1	7	4	0	0	2
	0	3	3	0	0	0

Figure 4-27. BYT Examples

The following example illustrates the use of a forty page EMA that has a five page MSEG. In the example, the main program calls MMAP to map the third MSEG into its logical address space. Then it stores the value at the start of the third MSEG into the 1028th word of the third MSEG. Then it calls a subroutine to process that element. The subroutine loads the value into the B-register to process it, and then returns to the calling program. Refer to Figure 4-28 for a pictorial explanation of the elements that are being addressed.

```
        NAM EMAPR,3
        EXT EMASB
        EXT MMAP
EMALB   EMA 40,5        40 pages of EMA, 5 pages per MSEG
ADEMA   DEF EMALB
D1027   DEC 1027

* CALL MMAP TO MAP THIRD MSEG INTO PROGRAMS LOGICAL ADDRESS SPACE

EMAPR   JSB MMAP
        DEF RTN
        DEF IPGS        Offset in pages of MSEG being mapped
        DEF NPGS        Number of pages in MSEG

* STORE FIRST WORD OF THIRD MSEG INTO 1028TH WORD OF THIRD MSEG

RTN     LDA EMALD       First word of MSEG referenced directly
        LDB ADEMA       Use B-Reg to reference 1028th word of MSEG
        RBL,CLE,SLB,ERB Resolve one indirect
        LDB 1,I
        ADB D1027
        STA 1,I         Store A-Reg into 1028th word of current MSEG

* JSB PASSING OFFSET ADDRESS TO SUBROUTINE

        JSB EMASB
        DEF D1027       Pass offset address as parameter
          .
          .
          .
IPGS    DEC 10
NPGS    DEC 5
```

```
        NAM EMASB,7     External subroutines and segments
        ENT EMASB       declare EMA as an external
        EXT EMALB
ADEMA   DEF EMALB

* SUBROUTINE ENTRY POINT IS HERE, A-REG IS USED TO COMPUTE ADDRESS

EMASB   NOP
        LDA EMASB,I     Get address of offset
        LDA 0,I         Get offset value
        LDB ADEMA
        RBL,CLE,SLB,ERB
        LDB 1,I
        ADA 1           Add in address of current MSEG start
        LDB 0,I         Load 1028th word of current MSEG into B-Reg
          .
          .
          .
        ISZ EMASB
        JMP EMASB,I     Return to caller
        END
```

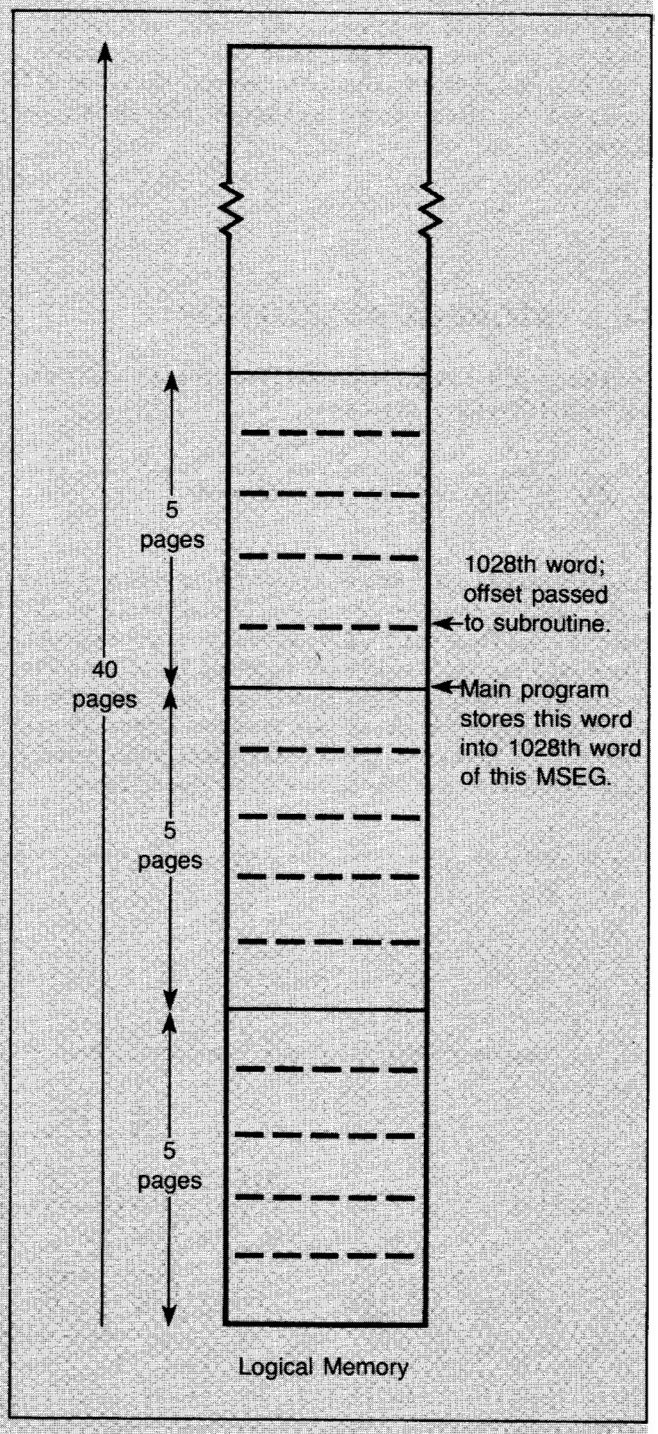


Figure 4-28. EMA Logical Memory for Example Program

4-7. RTE-L PSEUDO INSTRUCTIONS

The Assembler recognizes two new pseudo instructions for use in the RTE-L operating system. Under RTE-IV operating systems the information records that are produced by these two pseudo-instructions are ignored by the RTE-IV loader and generator.

Two useful pseudo instructions are available to aid in the loading and generation process of RTE-L programs and systems. The LOD pseudo instruction allows the programmer to specify LOADR instructions at the program code level rather than at load time.

The GEN pseudo instruction can help simplify RTE-L system generations. With the GEN record the user can set up interface and device driver defaults into the program (i.e. driver) code which are then put into files and can be accessed by the generator. This process simplifies the preparation for generation by relieving the need of getting and specifying all of the parameters needed for the Interface Table (IFT) and Device Tables (DVT).

The use of LOD and GEN pseudo instructions does not directly affect the size of a relocatable program or driver. They occupy no code space.

4-8. LOD STATEMENT

label	LOD	n,<ASCII Loader Command>	comments

LOD passes to the RTE-L loader the specified LOADR command. The first operand, n specifies the number of words occupied by the second operand. One character is right justified in each eight bits; the most significant bit is zero. n may be any expression resulting in an unsigned decimal value in the range 1 through 28. Symbols used in an expression must be previously defined. Anything in the operand field following 2n characters is treated as comments. If an odd number of characters is present or less than 2n characters are detected before the end of the operand field, the remaining characters are assumed to be spaces, and are stored as such. Examples are shown in Figure 4-29.

The LOD command enables the user to programmatically give commands to LOADR. For example a Real-Time program is to be loaded with a priority of 10, containing 3 segments, and the user wants to minimize Base Page links. A typical LOADR scenario might look like the following: (user responses are underlined).

```
RU,LOADR
     LOADR: RTIME
     LOADR: PRIORITY,10
     LOADR: SGMENTS,3
     LOADR: CPAGE
     LOADR: RE,%EXMPL
     LOADR: END
```

If this program is to be loaded in the same fashion EVERYTIME, then the user may wish to "hard-code" the LOADR commands into his program with the LOD command. See Figure 4-29 for examples. After coding in the LOADR command the user has only to do the following to accomplish the same tasks that were previously done interactively.

```
RU,LOADR,,%EXMPL
```

See the RTE-L RELOCATING LOADER REFERENCE MANUAL for specific details on LOADR commands.

```
ASMB,L,C
     NAM       EXMPL
     LOD       3,RTIME
     LOD       6,PRIORITY,10
     LOD       5,SGMENTS,3
     LOD       3,CPAGE
```

Figure 4-29. LOD Pseudo-Instruction Example

4-9. GEN STATEMENT

label	GEN	n,<ASCII Driver Defaults>	comments

GEN passes interface or device driver default parameters to the generator to be used in the process of system generation. The first operand, n specifies the number of words filled by the second operand. One character is right justified in each eight bits; the most significant bit is zero. n may be any expression resulting in an unsigned decimal value in the range 1 through 28. Symbols used in an expression must be previously defined. Anything in the operand field following 2n characters is treated as comments. If an odd number of characters is present or less than 2n characters are detected before the end of the operand field, the remaining characters are assumed to be spaces, and are stored as such. Examples are shown in Figure 4-30.

The GEN statement simplifies the process of DVT and IFT inputs to the generator. If the user is writing his own interface or device driver, in the actual driver code, defaults such as time outs, table extensions, device addresses, etc., can be directly specified via the GEN record. This process can simplify answer file preparation. See the RTE-L SYSTEM DESIGN manual for additional information on the use of the GEN statement.

```
From Driver: DD.30
    * Define Device Table Defaults for Discs
        GEN 9,EDD.30,TX:25,DX:8
    * 7902 Defaults (2 LUs)
        GEN 11,M7902:0,TO:500,DT:30B
        GEN 13,DP:2:0:0:0:3:134:DP:7:30:2
    *
        GEN 11,M7902:1,TO:500,DT:30B
        GEN 13,DP:2:1:0:0:3:134,DP:7:30:2
    *
    * 7906 Defaults (4 LUs)
        GEN 11,M7906:0,DT:32B,TO:1100
        GEN 13,DP:2:0:0:0:5:406,DP:7:48:1
                        :
                        :
```

Figure 4-30. GEN Examples

4-10. ASSEMBLY LISTING CONTROL

Assembly listing control pseudo instructions allow the user to control the assembly listing Output during pass 2 of the assembly process.

UNL	comments

List output is suppressed from the assembly listing, beginning with the UNL pseudo instruction and continuing for all instructions and comments until either an LST or END pseudo instruction is encountered. Diagnostic messages for errors encountered by the Assembler will be printed, however. The source statement sequence numbers (printed in columns 1-4 of the source program listing) are incremented for the instructions skipped.

LST	comments

The LST pseudo instruction causes the source program listing, terminated by a UNL, to be resumed.

A UNL following a UNL, an LST following an LST, and an LST not preceded by a UNL are not considered errors by the Assembler.

SUP	comments

The SUP pseudo instruction suppresses the output of additional code lines from the source program listing. Certain machine and pseudo instructions generate more than one line of coding. These additional code lines are suppressed by an SUP instruction until a UNS or the END pseudo instruction is encountered. SUP will suppress additional code lines in the following machine and pseudo instructions:

ADX	DJS	LAY	MLB	SBY
ADY	DLD	LBX	MPY	SJP
ASC	DST	LBY	MSA	SJS
BYT	FAD	LDX	MSB	STX
CBS	FDV	LDY	MVW	STY
CBT	FMP	MBT	OCT	TBS
CMW	FSB	MCA	SAX	UJP
DEC	JLY	MCB	SAY	UJS
DIV	JPY	MDB	SBS	XMM
DJP	LAX	MLA	SBX	XMS

The SUP pseudo instruction may be used to suppress the listing of literals at the end of the source program listing and also to suppress the printing of offset values for memory reference instructions which refer to external symbols with offsets.

UNS	comments

The UNS pseudo instruction causes the printing of additional coding lines, terminated by an SUP,to be resumed.

An SUP preceded by another SUP, UNS preceded by UNS, or UNS not preceded by an SUP are not considered errors by the Assembler.

SKP	comments

The SKP pseudo instruction causes the source program listing to be skipped to the top of the next page. The SKP instruction is not listed, but the source statement sequence number is incremented for the SKP.

SPC	n

The SPC pseudo instruction causes the source program listing to be skipped a specified number of lines. The list output is skipped n lines, or to the bottom of the page, whichever occurs first. The n may be any absolute expression. The SPC instruction is not listed but the source statement sequence number is incremented for the SPC.

HED	<heading>

The HED pseudo instruction allows the programmer to specify a heading to be printed at the top of each page of the source program listing. This header is printed in addition to the standard header printed by the Assembler.

The heading, a string of up to 56 ASCII characters, is printed at the top of each of the source program listings following the occurrence of the HED pseudo instruction. If HED is the first statement at the beginning of a program, the heading will be used on the first page of the source program listing. A HED instruction placed elsewhere in the program causes a skip to the top of the next page.

The heading specified in the HED pseudo instruction will be used on every page until it is changed by a succeeding HED instruction.

The source statement containing the HED will not be listed, but the source statement sequence number will be incremented.

4-11. ARITHMETIC SUBROUTINE CALLS

If an X appears in the control statement for the source program, the Assembler generates calls to arithmetic subroutines external to the source program for the following instructions: MPY, DIV, DLD, and DST. The instruction formats and functions are as described in paragraph 3-17 of Section III in this manual.

If an F *does not* appear in the control statement for the source program, the Assembler generates calls to arithmetic subroutines external to the source program for the following instructions: FMP, FDV, FAD, and FSB. The instruction formats and functions are as described in paragraph 3-18 of Section III in this manual.

Each use of a statement from this group except FIX and FLT generates two words of instructions. Symbolically, they could be represented as follows:

```
JSB    <.arithmetic pseudo operation>
DEF    m [,I]
```

An EXT <.arithmetic pseudo operation> is implied preceding the JSB operation.

In the above operations, the overflow bit is set when one of the following conditions occurs:

- Integer overflow
- Floating point overflow or underflow
- Division by zero.

Execution of any of the subroutines alter the contents of the E-Register.

4-12. DEFINE USER INSTRUCTION

	MIC	opcode,fcode,pnum	comments

This pseudo instruction provides the user the capability of defining his own instructions. *opcode* is a three-character alphabetic mnemonic, *fcode* is an instruction code, and *pnum* declares how many (0-7) parameter addresses are to be associated with the newly-defined instruction. Both *fcode* and *pnum* may be expressions which generate an absolute result. A user-defined instruction must not appear in the source program prior to the MIC pseudo instruction which defines it. When the user-defined mnemonic is used later in the source program, the specified number of parameter addresses (*pnum*) are supplied in the Operand field of the user-defined instruction separated from one another by spaces. The parameter addresses may be any addressable values, relocatable and/or indirect. The parameters may not be literals.

Note: All three operands (*opcode, fcode,* and *pnum*) must be supplied in the MIC pseudo instruction in order for the specified instruction to be defined. If *pnum* is zero, it must be expressly declared as such (*not* omitted).

4-13. "JUMP TO MICROPROGRAM" (HP 1000 M, E, F-SERIES ONLY)

The MIC pseudo instruction is primarily intended to facilitate the passing of control from an assembly language program to a user's microprogram residing in Read-Only-Memory (ROM) or Writable Control Store (WCS). Ordinarily, to do this the user must include an OCT 101*xxx* or OCT 105*xxx* statement (where *xxx* is 140 through 737) at the point in the source program where the jump is to occur. If parameters are to be passed, they are usually defined as constants (via OCT or DEF statements) immediately following the OCT 105*xxx* statement. With the MIC pseudo instruction, the user can define a source language instruction which passes control and a series of parameter addresses to a microprogram. If it is desired to pass additional parameters to a microprogram beyond those pointed to by the user-defined instruction, they must be defined as constants (via OCT or DEF statements) immediately following each use of the user-defined instruction.

4-14. EXAMPLE. Assume that the first two parameters to be passed from the assembly language program to the user's microprogram reside in the memory locations PARM1 and PARM2 and that the third parameter resides in the memory location pointed to by ADR. Also assume that the octal code for transferring control to the particular microprogram is 105240_8.

The following statement defines a source language instruction which passes control and three parameter addresses to the microprogram:

```
MIC    ABC,105240B,3
```

Whenever it is desired to pass control from the assembly language program to the microprogram, the following user-defined instruction may be used in the source program:

```
ABC    PARM1 PARM2 ADR,I
```

4-15. COMBINING MULTIPLE MNEMONICS

Another use of the MIC pseudo instruction is to assign a single mnemonic to a multiple instruction (shift-rotate or alter-skip) statement.

4-16. EXAMPLE. Instead of using the source statement:

```
ALR,CLE,SLA,RAL
```

the user may define a single mnemonic as follows:

```
MIC    XYZ,01472B,0
```

where 01472B is the octal instruction code for the four-mnemonic statement shown above. Whenever XYZ is subsequently used as an instruction mnemonic in the source program, it is the equivalent of using the source statement:

```
ALR,CLE,SLA,RAL
```

4-17. DEFINING CONSTANTS

The MIC pseudo instruction may also be used for defining constants (*opcode* = mnemonic, *fcode* = constant, and *pnum* = 0). Whenever the defined mnemonic is used as an instruction mnemonic in the source program the Assembler automatically replaces it with the specified constant.

4-18. EXAMPLE. The following statement defines the constant 10_{10} and assigns it the mnemonic TEN:

```
MIC    TEN,10,0
```

Whenever TEN appears as an instruction mnemonic later in the source program, the value 10_{10} is automatically inserted in that location by the Assembler.

4-19. ALTERNATE MICROCODE REFERENCE INSTRUCTION (21MX Series and 2100 Only)

	RAM	m	comments

An alternate but somewhat restricted way to access microprogrammed functions from the Assembler language is by employing the RAM (Random Access Memory) pseudo-instruction. The RAM pseudo-instruction will generate an executable machine instruction which when executed will cause a jump to microcode. The high order bits of the instruction will be 105 octal and the low order bits will be the octal value of m. m must evaluate to an absolute expression in the range 0 to 377 octal.

4-20. EXAMPLE. The following lines of assembly code:

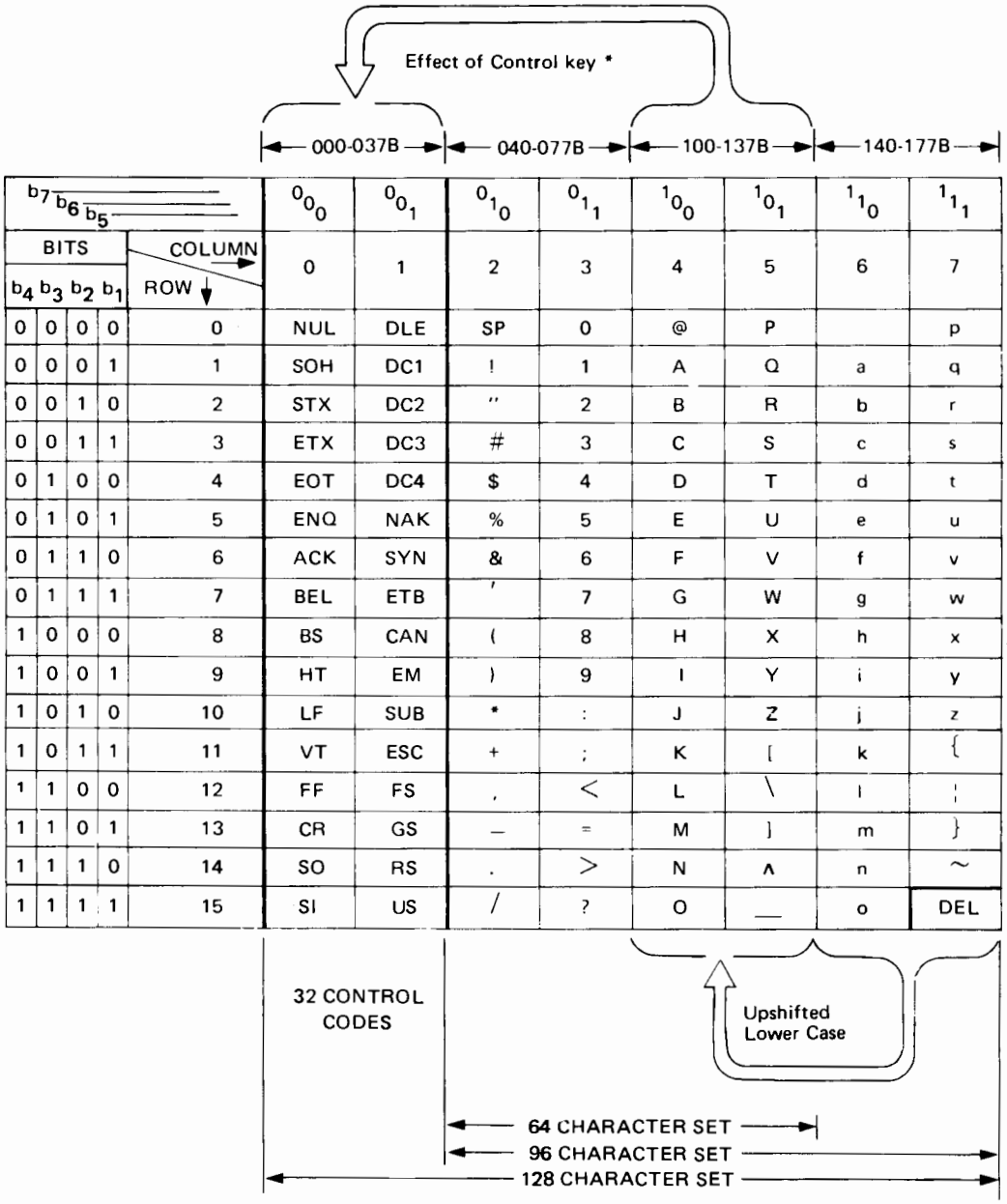
```
      RAM  B16
B16   EQU  16B
```

will generate this octal object code:

```
105016
```

HP CHARACTER SET FOR COMPUTER SYSTEMS

APPENDIX A



b_7 b_6 b_5				0_0	0_1	1_0	1_1	0_0	0_1	1_0	1_1
BITS				COLUMN →							
b_4	b_3	b_2	b_1	ROW ↓	0	1	2	3	4	5	6	7
0	0	0	0	0	NUL	DLE	SP	0	@	P		p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11	VT	ESC	+	;	K	[	k	{
1	1	0	0	12	FF	FS	,	<	L	\	l	¦
1	1	0	1	13	CR	GS	–	=	M	]	m	}
1	1	1	0	14	SO	RS	.	>	N	^	n	~
1	1	1	1	15	SI	US	/	?	O	_	o	DEL

EXAMPLE: The representation for the character "K" (column 4, row 11) is.

	b_7	b_6	b_5	b_4	b_3	b_2	b_1
BINARY	1	0	0	1	0	1	1
OCTAL	1		1		3		

* Depressing the Control key while typing an upper case letter produces the corresponding control code on most terminals. For example, Control-H is a backspace.

9206- 1A

HEWLETT-PACKARD CHARACTER SET FOR COMPUTER SYSTEMS

This table shows HP's implementation of ANS X3.4-1968 (USASCII) and ANS X3.32-1973. Some devices may substitute alternate characters from those shown in this chart (for example, Line Drawing Set or Scandanavian font). Consult the manual for your device.

The left and right byte columns show the octal patterns in a 16 bit word when the character occupies bits 8 to 14 (left byte) or 0 to 6 (right byte) and the rest of the bits are zero. To find the pattern of two characters in the same word, add the two values. For example, "AB" produces the octal pattern 040502. (The parity bits are zero in this chart.)

The octal values 0 through 37 and 177 are control codes. The octal values 40 through 176 are character codes.

Decimal Value	Octal Values		Mnemonic	Graphic[1]	Meaning
	Left Byte	Right Byte			
0	000000	000000	NUL	$^{N}_{U}$	Null
1	000400	000001	SOH	$^{S}_{H}$	Start of Heading
2	001000	000002	STX	$^{S}_{X}$	Start of Text
3	001400	000003	EXT	$^{E}_{X}$	End of Text
4	002000	000004	EOT	$^{E}_{T}$	End of Transmission
5	002400	000005	ENQ	$^{E}_{Q}$	Enquiry
6	003000	000006	ACK	$^{A}_{K}$	Acknowledge
7	003400	000007	BEL		Bell, Attention Signal
8	004000	000010	BS	$^{B}_{S}$	Backspace
9	004400	000011	HT	$^{H}_{T}$	Horizontal Tabulation
10	005000	000012	LF	$^{L}_{F}$	Line Feed
11	005400	000013	VT	$^{V}_{T}$	Vertical Tabulation
12	006000	000014	FF	$^{F}_{F}$	Form Feed
13	006400	000015	CR	$^{C}_{R}$	Carriage Return
14	007000	000016	SO	$^{S}_{O}$	Shift Out } Alternate Character Set
15	007400	000017	SI	$^{S}_{I}$	Shift In } Alternate Character Set
16	010000	000020	DLE	$^{D}_{L}$	Data Link Escape
17	010400	000021	DC1	D_1	Device Control 1 (X-ON)
18	011000	000022	DC2	D_2	Device Control 2 (TAPE)
19	011400	000023	DC3	D_3	Device Control 3 (X-OFF)
20	012000	000024	DC4	D_4	Device Control 4 (TAPE)
21	012400	000025	NAK	$^{N}_{K}$	Negative Acknowledge
22	013000	000026	SYN	$^{S}_{Y}$	Synchronous Idle
23	013400	000027	ETB	$^{E}_{B}$	End of Transmission Block
24	014000	000030	CAN	$^{C}_{N}$	Cancel
25	014400	000031	EM	$^{E}_{M}$	End of Medium
26	015000	000032	SUB	$^{S}_{B}$	Substitute
27	015400	000033	ESC	$^{E}_{C}$	Escape[2]
28	016000	000034	FS	$^{F}_{S}$	File Separator
29	016400	000035	GS	$^{G}_{S}$	Group Separator
30	017000	000036	RS	$^{R}_{S}$	Record Separator
31	017400	000037	US	$^{U}_{S}$	Unit Separator
127	077400	000177	DEL	▩	Delete, Rubout[3]

Decimal Value	Octal Values		Character	Meaning
	Left Byte	Right Byte		
32	020000	000040		Space, Blank
33	020400	000041	!	Exclamation Point
34	021000	000042	"	Quotation Mark
35	021400	000043	#	Number Sign, Pound Sign
36	022000	000044	$	Dollar Sign
37	022400	000045	%	Percent
38	023000	000046	&	Ampersand, And Sign
39	023400	000047	'	Apostrophe, Acute Accent
40	024000	000050	(	Left (opening) Parenthesis
41	024400	000051	)	Right (closing) Parenthesis
42	025000	000052	*	Asterisk, Star
43	025400	000053	+	Plus
44	026000	000054	,	Comma, Cedilla
45	026400	000055	-	Hyphen, Minus, Dash
46	027000	000056	.	Period, Decimal Point
47	027400	000057	/	Slash, Slant
48	030000	000060	0	} Digits, Numbers
49	030400	000061	1	
50	031000	000062	2	
51	031400	000063	3	
52	032000	000064	4	
53	032400	000065	5	
54	033000	000066	6	
55	033400	000067	7	
56	034000	000070	8	
57	034400	000071	9	
58	035000	000072	:	Colon
59	035400	000073	;	Semicolon
60	036000	000074	<	Less Than
61	036400	000075	=	Equals
62	037000	000076	>	Greater Than
63	037400	000077	?	Question Mark

9206-1B

Decimal Value	Octal Values		Character	Meaning
	Left Byte	Right Byte		
64	040000	000100	@	Commercial At
65	040400	000101	A	Upper Case Alphabet, Capital Letters
66	041000	000102	B	
67	041400	000103	C	
68	042000	000104	D	
69	042400	000105	E	
70	043000	000106	F	
71	043400	000107	G	
72	044000	000110	H	
73	044400	000111	I	
74	045000	000112	J	
75	045400	000113	K	
76	046000	000114	L	
77	046400	000115	M	
78	047000	000116	N	
79	047400	000117	O	
80	050000	000120	P	
81	050400	000121	Q	
82	051000	000122	R	
83	051400	000123	S	
84	052000	000124	T	
85	052400	000125	U	
86	053000	000126	V	
87	053400	000127	W	
88	054000	000130	X	
89	054400	000131	Y	
90	055000	000132	Z	
91	055400	000133	[	Left (opening) Bracket
92	056000	000134	\	Backslash, Reverse Slant
93	056400	000135	]	Right (closing) Bracket
94	057000	000136	^ ↑	Caret, Circumflex; Up Arrow[4]
95	057400	000137	_ ←	Underline; Back Arrow[4]

Decimal Value	Octal Values		Character	Meaning
	Left Byte	Right Byte		
96	060000	000140	`	Grave Accent[5]
97	060400	000141	a	Lower Case Letters[5]
98	061000	000142	b	
99	061400	000143	c	
100	062000	000144	d	
101	062400	000145	e	
102	063000	000146	f	
103	063400	000147	g	
104	064000	000150	h	
105	064400	000151	i	
106	065000	000152	j	
107	065400	000153	k	
108	066000	000154	l	
109	066400	000155	m	
110	067000	000156	n	
111	067400	000157	o	
112	070000	000160	p	
113	070400	000161	q	
114	071000	000162	r	
115	071400	000163	s	
116	072000	000164	t	
117	072400	000165	u	
118	073000	000166	v	
119	073400	000167	w	
120	074000	000170	x	
121	074400	000171	y	
122	075000	000172	z	
123	075400	000173	{	Left (opening) Brace[5]
124	076000	000174	|	Vertical Line[5]
125	076400	000175	}	Right (closing) Brace[5]
126	077000	000176	~	Tilde, Overline[5]

Notes: [1]This is the standard display representation. The software and hardware in your system determine if the control code is displayed, executed, or ignored. Some devices display all control codes as "|", "@", or space.

[2]Escape is the first character of a special control sequence. For example, ESC followed by "J" clears the display on a 2640 terminal.

[3]Delete may be displayed as "_", "@", or space.

[4]Normally, the caret and underline are displayed. Some devices substitute the up arrow and back arrow.

[5]Some devices upshift lower case letters and symbols (` through ~) to the corresponding upper case character (@ through ^). For example, the left brace would be converted to a left bracket.

HP 7970B BCD-ASCII CONVERSION

SYMBOL	BCD (OCTAL CODE)	ASCII EQUIVALENT (OCTAL CODE)	SYMBOL	BCD (OCTAL CODE)	ASCII EQUIVALENT (OCTAL CODE)
(space)	20	040	@	14	100
!	52	041	A	61	101
"	37	042	B	62	102
#	13	043	C	63	103
$	53	044	D	64	104
%	57	045	E	65	105
&	†	046	F	66	106
'	35	047	G	67	107
(	34	050	H	70	110
)	74	051	I	71	111
*	54	052	J	41	112
+	60	053	K	42	113
,	33	054	L	43	114
-	40	055	M	44	115
.	73	056	N	45	116
/	21	057	O	46	117
0	12	060	P	47	120
1	01	061	Q	50	121
2	02	062	R	51	122
3	03	063	S	22	123
4	04	064	T	23	124
5	05	065	U	24	125
6	06	066	V	25	126
7	07	067	W	26	127
8	10	070	X	27	130
9	11	071	Y	30	131
:	15	072	Z	31	132
;	56	073	[	75	133
<	76	074	\	36	134
=	17	075	]	55	135
>	16	076	↑	77	136
?	72	077	←	32	137

†The ASCII code 046 is converted to the BCD code for a space (20) when writing data onto a 7-track tape.

SUMMARY OF INSTRUCTIONS

APPENDIX B

Symbols	Meaning
label	Symbolic label, 1-5 alphanumeric characters and periods
m	Memory location represented by an expression
I	Indirect addressing indicator
C	Clear flag indicator
(m,m+1)	Two-word floating point value in m and m+1
comments	Optional comments
[ ]	Optional portion of field
{ }	One of set may be selected
P	Program Counter
( )	Contents of location
$\wedge$	Logical product
$\forall$	Exclusive "or"
$\vee$	Inclusive "or"
A	A-register
B	B-register
E	E-register
A_n	Bit n of A-register
B_n	Bit n of B-register
b	Bit positions in B- and A-register
$(\overline{A/B})$	Complement of contents of register A or B
(AB)	Two-word floating point value in register A and B
sc	Channel select code represented by an expression
d	Decimal constant
o	Octal constant
r	Repeat count
n	Integer constant
lit	Literal value
msb	Most significant bits
lsb	Least significant bits

NOTE

Instructions shaded in gray are implemented on the M, E, and F-Series computers and not on the L-Series computers.

For the HP 1000 L-Series computers: Instructions suffixed with an asterisk * are implemented in software under RTE-L/XL.

B-1. MACHINE INSTRUCTIONS

B-2. MEMORY REFERENCE

B-3. Jump and Increment-Skip

ISZ	m [,I]	(m) + 1 → m: then if (m) = 0, execute P + 2 otherwise execute P + 1
JMP	m [,I]	Jump to m; m → P
JSB	m [,I]	Jump subroutine to m: P + 1 → m; m + 1 → P

B-4. Add, Load and Store

ADA	{ m [,I] / lit }	(m) + (A) → A
ADB	{ m [,I] / lit }	(m) + (B) → B
LDA	{ m [,I] / lit }	(m) → A
LDB	{ m [,I] / lit }	(m) → B
STA	m [,I]	(A) → m
STB	m [,I]	(B) → m

B-5. Logical

AND	{ m [,I] / lit }	(m) ∧ (A) → A
XOR	{ m [,I] / lit }	(m) ∀ (A) → A
IOR	{ m [,I] / lit }	(m) ∨ (A) → A
CPA	{ m [,I] / lit }	If (m) ≠ (A), execute P + 2, otherwise execute P + 1
CPB	{ m [,I] / lit }	If (m) ≠ (B), execute P + 2, otherwise execute P + 1

B-6. Word Processing

MVW*	{ m [,I] / lit }	Move (m) words from array (A) → array (B)
CMW*	{ m [,I] / lit }	Compare (m) words of array (A) against (m) words of array (B); if the two arrays are equal, execute P + 3, if array (A) is less than array (B), execute P + 4, if array (A) is greater than array (B), execute P + 5

B-7. Byte Processing

LBT*		B contains a 16-bit byte address; ((B))→A_{0-7}; o's to A_{8-15}
SBT*		B contains a 16-bit byte address; (A_{0-7})→(B)
MBT*	{ m [,I] / lit }	A and B contain 16-bit byte addresses; move (m) bytes from array (A)→array (B)
CBT*	{ m [,I] / lit }	A and B contain 16-bit byte addresses; compare (m) bytes of array (A) against (m) bytes of array (B); if the two arrays are equal, execute P + 3; if array (A) is less than array (B), execute P + 4; if array (A) is greater than array (B), execute P + 5
SFB*		A_{0-7} contain the test byte, A_{8-15} contain the termination byte, and B contains a 16-bit byte address; scan array (B); if test byte found, execute P + 1, B contains address of test byte; if termination byte found, execute P + 2, B contains address of termination byte; if neither is found, execute P + 2, B contains zero

B-8. Bit Processing

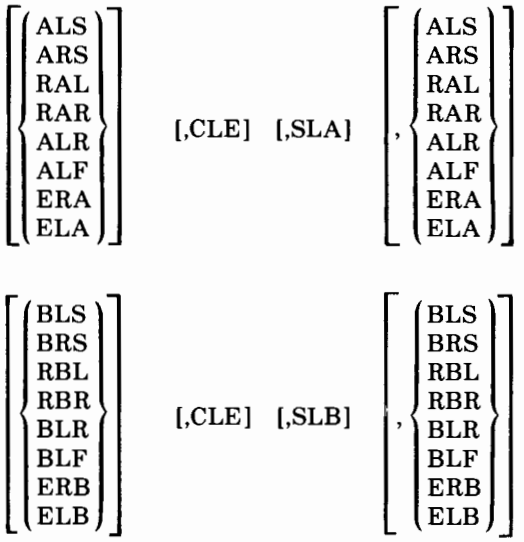
TBS*	{ m [,I] / lit }	n [,I]	Compare all "set" bits in (m) against corresponding bits in (n); if all bits tested are set, execute P + 3; if any of the bits tested are clear, execute P + 4
SBS*	{ m [,I] / lit }	n [,I]	Set all bits in (n) which correspond to "set" bits in (m)
CBS*	{ m [,I] / lit }	n [,I]	Clear all bits in (n) which correspond to "set" bits in (m)

B-9. REGISTER REFERENCE

B-10. Shift-Rotate

CLE	0→E
ALS	Shift (A) left one bit, 0→A_0, A_{15} unaltered
BLS	Shift (B) left one bit, 0→B_0, B_{15} unaltered
ARS	Shift (A) right one bit, (A_{15})→A_{14}
BRS	Shift (B) right one bit, (B_{15})→B_{14}
RAL	Rotate (A) left one bit
RBL	Rotate (B) left one bit
RAR	Rotate (A) right one bit
RBR	Rotate (B) right one bit
ALR	Shift (A) left one bit, 0→A_{15}
BLR	Shift (B) left one bit, 0→B_{15}
ERA	Rotate E and A right one bit
ERB	Rotate E and B right one bit
ELA	Rotate E and A left one bit
ELB	Rotate E and B left one bit
ALF	Rotate A left four bits
BLF	Rotate B left four bits
SLA	If (A_0) = 0, execute P + 2, otherwise execute P + 1
SLB	If (B_0) = 0, execute P + 2, otherwise execute P + 1

Shift-Rotate instructions can be combined as follows:



B-11. No-Operation

NOP	Execute P + 1

B-12. Alter-Skip

CLA	0's → A
CLB	0's → B
CMA	$\overline{(A)} \rightarrow A$
CMB	$\overline{(B)} \rightarrow B$
CCA	1's → A
CCB	1's → B
CLE	0 → E
CME	$\overline{(E)} \rightarrow E$
CCE	1 → E
SEZ	If (E) = 0, execute P + 2, otherwise execute P + 1
SSA	If $(A_{15}) = 0$, execute P + 2, otherwise execute P + 1
SSB	If $(B_{15}) = 0$, execute P + 2, otherwise execute P + 1
INA	(A) + 1 → A
INB	(B) + 1 → B
SZA	If (A) = 0, execute P + 2, otherwise execute P + 1
SZB	If (B) = 0, execute P + 2, otherwise execute P + 1
SLA	If $(A_0) = 0$, execute P + 2, otherwise execute P + 1
SLB	If $(B_0) = 0$, execute P + 2, otherwise execute P + 1
RSS	Reverse sense of skip instructions. If no skip instructions precede, execute P + 2

Alter-Skip instructions can be combined as follows:

$$\left[\left\{\begin{matrix}\text{CLA}\\\text{CMA}\\\text{CCA}\end{matrix}\right\}\right] \quad [,\text{SEZ}] \quad \left[,\left\{\begin{matrix}\text{CLE}\\\text{CME}\\\text{CCE}\end{matrix}\right\}\right] \quad [,\text{SSA}]\ [,\text{SLA}]\ [,\text{INA}]\ [,\text{SZA}]\ [,\text{RRS}]$$

$$\left[\left\{\begin{matrix}\text{CLB}\\\text{CMB}\\\text{CCB}\end{matrix}\right\}\right] \quad [,\text{SEZ}] \quad \left[,\left\{\begin{matrix}\text{CLE}\\\text{CME}\\\text{CCE}\end{matrix}\right\}\right] \quad [,\text{SSB}]\ [,\text{SLB}]\ [,\text{INB}]\ [,\text{SZB}]\ [,\text{RSS}]$$

B-13. Index Register

CAX*		(A)→X
CBX*		(B)→X
CAY*		(A)→Y
CBY*		(B)→Y
CXA*		(X)→A
CXB*		(X)→B
CYA*		(Y)→A
CYB*		(Y)→B
XAX*		(A)→X and (X)→A
XBX*		(B)→X and (X)→B
XAY*		(A)→Y and (Y)→A
XBY*		(B)→Y and (Y)→B
ISX*		(X) + 1→X, then test new (X); if (X) = 0, execute P + 2, otherwise execute P + 1
ISY*		(Y) + 1→Y, then test new (Y); if (Y) = 0, execute P + 2, otherwise execute P + 1
DSX*		(X) - 1→X, then test new (X); if (X) = 0, execute P + 2, otherwise execute P + 1
DSY*		(Y) - 1→Y, then test new (Y); if (Y) = 0, execute P + 2, otherwise execute P + 1
LDX*	m [,I] lit	(m)→X
LDY*	m [,I] lit	(m)→Y
STX*	m [,I]	(X)→m
STY*	m [,I]	(Y)→m
LAX*	m [,I]	(m + (X))→A
LBX*	m [,I]	(m + (X))→B
LAY*	m [,I]	(m + (Y))→A
LBY*	m [,I]	(m + (Y))→B
SAX*	m [,I]	(A)→m + (X)
SBX*	m [,I]	(B)→m + (X)
SAY*	m [,I]	(A)→m + (Y)
SBY*	m [,I]	(B)→m + (Y)
ADX*	m [,I] lit	(m) + (X)→X
ADY*	m [,I] lit	(m) + (Y)→Y
JLY*	m [,I]	Jump to m; P + 2→Y
JPY*	m	Jump to m + (Y)

B-14. INPUT/OUTPUT, OVERFLOW, AND HALT

B-15. Input/Output

STC	sc [,C]	Set control bit_{sc}, enable transfer of one element of data between $device_{sc}$ and $buffer_{sc}$
CLC	sc [,C]	Clear control bit_{sc}. If sc = 0 clear all control bits
LIA	sc [,C]	$(buffer_{sc}) \rightarrow A$
LIB	sc [,C]	$(buffer_{sc})$ V $(A) \rightarrow A$ Merge (inclusive or) the buffer into A.
MIA	sc [,C]	$(buffer_{sc})$ V $(B) \rightarrow B$ Merge (inclusive or) the buffer into B.
MIB	sc [,C]	$(buffer_{sc})$ $(B) \rightarrow B$
OTA	sc [,C]	$(A) \rightarrow buffer_{sc}$
OTB	sc [,C]	$(B) \rightarrow buffer_{sc}$
STF	sc	Set flag bit_{sc}. If sc = 0, enable interrupt system. sc = 1 sets overflow bit.
CLF	sc	Clear flag bit_{sc}. If sc = 0, disable interrupt system. If sc = 1, clear overflow bit.
SFC	sc	If (flag bit_{sc}) = 0, execute P + 2, otherwise execute P + 1. If sc = 1, test overflow bit.
SFS	sc	If (flag bit_{sc}) = 1, execute P + 2, otherwise execute P + 1. If sc = 1, test overflow bit.

B-16. Overflow

CLO		0→overflow bit
STO		1→overflow bit
SOC	[C]	If (overflow bit) = 0, execute P + 2, otherwise execute P + 1
SOS	[C]	If (overflow bit) = 0, execute P + 2, otherwise execute P + 1

B-17. Halt

HLT	[sc [,C]]	Halt computer

B-18. EXTENDED ARITHMETIC UNIT

MPY	$\left\{\begin{matrix} m\ [,I] \\ lit \end{matrix}\right\}$	(A) x (m)→($B_{\pm msb}$ and A_{lsb})
DIV	$\left\{\begin{matrix} m\ [,I] \\ lit \end{matrix}\right\}$	($B_{\pm msb}$ and A_{lsb})/(m)→A, remainder→B
DLD	$\left\{\begin{matrix} m\ [,I] \\ lit \end{matrix}\right\}$	(m) and (m + 1)→A and B
DST	$\left\{\begin{matrix} m\ [,I] \\ lit \end{matrix}\right\}$	(A) and (B)→m and m + 1
ASR	b	Arithmetically shift (BA) right b bits, B_{15} extended
ASL	b	Arithmetically shift (BA) left b bits, B_{15} unaltered, 0's to A_{lsb}

RRR	b	Rotate (BA) right b bits
RRL	b	Rotate (BA) left b bits
LSR	b	Logically shift (BA) right b bits, 0's to B_{msb}
LSL	b	Logically shift (BA) left b bits, o's to A_{lsb}
SWP		Swap the contents of the A and B registers

B-19. FLOATING POINT

FMP*	{ m [,I] / lit }	(AB) x (m, m + 1)→AB
FDV*	{ m [,I] / lit }	(AB)/(m, m + 1)→AB
FAD*	{ m [,I] / lit }	(m, m + 1) + (AB)→AB
FSB*	{ m [,I] / lit }	(AB) - (m, m + 1)→AB
FIX*		(AB) converted from floating-point to fixed-point; result →A
FLT*		(A) converted from fixed-point to floating-point; result →AB

B-20. MEMORY EXPANSION

DJP	m [,I]	Disable MEM and jump to m; m → P
DJS	m [,I]	Disable MEM and jump subroutine to m; P + 1 → m; m + 1 → P
JRS	m_1 [,I] m_2 [,I]	Jump and restore status
LFA		A → fence
LFB		B → fence
MBF*		Move bytes from alternate map. X← 0; A← A + no. bytes moved; B← B + no. bytes moved.
MBI*		Move bytes into alternate map. X← 0; A← A + no. bytes moved; B← B + no. bytes moved.
MBW*		Move bytes within alternate map. X← 0; A← A + no. bytes moved; B← B + no. bytes moved.
MWF*		Move words from alternate map. X← 0; A← A + no. words moved; B← B + no. words moved.
MWI*		Move words into alternate map. X← 0; A← A + no. words moved; B← B + no. words moved.
MWW*		Move words within alternate map. X← 0; A← A + no. words moved; B← B + no. words moved.

PAA		If A(15) = 0, Port A map ← memory; if A(15) = 1, Port A map → memory.
PAB		If B(15) = 0, Port A map ← memory; if B(15) = 1, Port A map → memory.
PBA		If A(15) = 0, Port B map ← memory; if A(15) = 1, Port B map → memory.
PBB		If B(15) = 0, Port B map ← memory; if B(15) = 1, Port B map → memory.
RSA		A ← status register
RSB		B ← status register
RVA		A ← violation register
RVB		B ← violation register
SJP	m [,I]	Enable System map and jump to m
SJS	m [,I]	Enable System map and jump subroutine to m
SSM	m [,I]	m ← status register
SYA		If A(15) = 0, System map ← memory; if A(15) = 1, System map → memory.
SYB		If B(15) = 0, System map ← memory; if B(15) = 1, System map → memory.
UJP	m [,I]	Enable User map and jump to m
UJS	m [,I]	Enable User map and jump subroutine to m
USA		If A(15) = 0, User map ← memory; if A(15) = 1, User map → memory.
USB		If B(15) = 0, User map ← memory; if B(15) = 1, User map → memory.
XCA*	m [,I]	Compare A with m; if A = m, execute P = 1; if A ≠ m, execute P + 2.
XCB*	m [,I]	Compare B with m; if B = m, execute P + 1; if B ≠ m, execute P + 2.
XLA*	m [,I]	A ← m
XLB*	m [,I]	B ← m
XMA		If A(15) = 0 and A(0) = 0, Port A map ← System map. If A(15) = 0 and A(0) = 1, Port B map ← system map. If A(15) = 1 and A(0) = 0, Port A map ← User map. If A(15) = 1 and A(0) = 1, Port B map ← User map.
XMB		If B(15) = 0 and B(0) = 0, Port A map ← System map. If B(15) = 0 and B(0) = 1, Port B map ← System map. If B(15) = = 1 and B(0) = 0, Port A map ← User map. If B(15) = 1 and B(0) = 1, Port B map ← User map.
XMM		A = register no., B = memory address, X = no. of registers. If X > 0, Maps ← memory; if X < 0, Memory ← maps.
XMS		A = first register no., B = first page no., X = positive no. of registers. First register is loaded with the page number indicated in B, the second register is loaded with that value + 1, and so forth.
XSA*	m [,I]	A → m
XSB*	m [,I]	B → m

B-21. PSEUDO INSTRUCTIONS

B-22. ASSEMBLER CONTROL

NAM	[name]	Specifies relocatable program and its name.
ORB		Gives relocatable program origin for the base page of relocatable program.
ORG	m	Gives absolute program origin or origin for a segment of relocatable or absolute program.
ORR		Reset main program location counter at value existing when first ORG or ORB of a string was encountered.
END	[m]	Terminates source language program. Produces transfer to program starting location, m, if given.
REP <statement>	r	Repeat immediately following statement r times.
IFN <statements> XIF		Include statements in program if control statement contains N.
IFZ <statements> XIF		Include statements in program if control statement contains Z.

B-23. OBJECT PROGRAM LINKAGE

COM $name_1[(size_1)][,name_2[(size_2)],...,name_n[(size_n)]]$

Reserves a block of common storage locations. $name_1$ identifies segments of block, each of length size.

ENT $name_1[,name_2,...,name_n]$

Defines entry points, $name_1$, that may be referred to by other programs.

EXT $name_1[,name_2, . . .,name_n]$

Defines external locations, name , which are labels of other programs, referenced by this program.

label RPL [m]

Defines the code replacement for [JSB label] external references.

B-24. ADDRESS AND SYMBOL DEFINITION

label	DEF	m [,I]	Generates a 15-bit address which may be referenced indirectly through the label.
label	ABS	m	Defines a 16-bit absolute value to be referenced by the label.
label	EQU	m	Equates the value, m, to the label.
label	DBL	m	Defines a 16-bit byte address (left half, bits 8-15, of word location *m*) to be referenced by the label.

label	DBR m	Defines a 16-bit byte address to be referenced by the label. The byte address is for the right half (bits 0-7) of word location m.

B-25. CONSTANT DEFINITION

ASC n, <2n characters>	Generates a string of 2n ASCII characters.
DEC d_1 [,d_2,...,d_n]	Records a string of decimal constants of the form: Integer: ±n Floating point: ±n.n, ±n., ±.n, ±nE±e, ±n.nE±e, ±n.E±e, ±.nE±e
DEX d_1 [,d_2,...,d_n]	Records a string of extended precision decimals constants of the form Floating point: ±n, ±n.m, ±n., ±.n, ±nE±e, ±n.nE±e, ±n.E±e, ±.nE±e
DEY d_1 [,d_2,...,d_n]	Records a string of four-word extended precision decimal constants in the same form as DEX.
OCT o_1 [,o_2,...,o_n]	Records a string of octal constants of the form: ±000000
BYT b[,b ,...,b_n]	Records a string of octal byte constants of the form: ±nnn (where *nnn* is 0 through 377_8).

B-26. STORAGE ALLOCATION

BSS	m	Reserves a storage area of length, m.
EMA	m_1,m_2	Extended Memory Area of size m_1m NSEG=m_2.

B-27. RTE-L PSEUDO INSTRUCTIONS

LOD n,<2n characters> Generates a string of 2n ASCII characters representing a RTE-L loader command.

GEN n,<2n characters> Generates a string of 2n ASCII characters representing a RTE-L generator command.

B-28. ASSEMBLY LISTING CONTROL

UNL		Suppress assembly listing output.
LST		Resume assembly listing output.
SKP		Skip listing to top of next page.
SPC	n	Skip n lines on listing.
SUP		Suppress listing of extended code lines (e.g., as produced by subroutine calls).
UNS		Resume listing of extended code lines.
HED	<heading>	Print <heading> at top of each page, where <heading> is up to 56 ASCII characters.

B-29. DEFINE USER INSTRUCTION

MIC	opcode,fcode,pnum	Defines a source language instruction. *opcode* = three-character alphabetic mnemonic, *fcode* = instruction code, and *pnum* declares how many parameter addresses are to be associated with the newly-defined instruction.

B-30. GENERATE AN EXECUTABLE MACHINE INSTRUCTION TO JUMP TO MICROCODE

RAM	m	Generates an executable machine instruction whose high order bits will be 105(octal), and whose low order bits will be the octal value of m. m must evaluate to an absolute expression in the range 0 to 377 octal.

ALPHABETIC LIST OF INSTRUCTIONS

APPENDIX C

Note: Instructions shaded in gray are implemented on the M, E, and F-Series computers and *not* on the L-Series computers.

For the HP 1000 L-Series computers: Instructions suffixed with an asterisk * are implemented in software under RTE-L/XL.

ABS	Define absolute value
ADA	Add to A
ADB	Add to B
ADX*	Add memory to X
ADY*	Add memory to Y
ALF	Rotate A left 4
ALR	Shift A left 1, clear sign
ALS	Shift A left 1
AND	"And" to A
ARS	Shift A right 1, sign carry
ASC	Generate ASCII characters
ASL	Arithmetic long shift left
ASR	Arithmetic long shift right
BLF	Rotate B left 4
BLR	Shift B left 1, clear sign
BLS	Shift B left 1
BRS	Shift B right 1, carry sign
BSS	Reserve block of storage starting at symbol
BYT	Defines octal byte constants
CAX*	Copy A to X
CAY*	Copy A to Y
CBS*	Clear bits
CBT*	Compare bytes
CBX*	Copy B to X
CBY*	Copy B to Y
CCA	Clear and complement A (1's)
CCB	Clear and complement B (1's)
CCE	Clear and complement E (set E = 1)
CLA	Clear A
CLB	Clear B
CLC	Clear I/O control bit
CLE	Clear E
CLF	Clear I/O flag
CLO	Clear overflow bit
CMA	Complement A
CMB	Complement B
CME	Complement E
CMW*	Compare words
COM	Reserve block of common storage
CPA	Compare to A, skip if unequal
CPB	Compare to B, skip if unequal
CXA*	Copy X to A
CXB*	Copy X to B
CYA*	Copy Y to A
CYB*	Copy Y to B
DBL	Define left byte (bits 8-15) address
DBR	Define right byte (bits 0-7) address
DEC	Define decimal constant
DEF	Define address
DEX	Define extended precision constant
DEY	Define four-word extended precision constant
DIV	Divide
DJP	Disable MEM and jump
DJS	DISABLE MEM and jump to subroutine
DLD	Double load
DST	Double store
DSX*	Decrement X and skip if zero
DSY*	Decrement Y and skip if zero
ELA	Rotate E and A left 1
ELB	Rotate E and B left 1
EMA	Extended Memory Area
END	Terminate program
ENT	Entry point
ERA	Rotate E and A right 1
ERB	Rotate E and B right 1
EQU	Equate symbol
EXT	External reference
FAD*	Floating add
FDV*	Floating divide
FIX*	Convert floating-point be fixed-point
FLT*	Convert fixed-point to floating-point
FMP*	Floating multiply
FSB*	Floating subtract

HED	Print heading at top of each page
HLT	Halt
IFN	When N appears in Control statement, assemble ensuing instructions
IFZ	When Z appears in Control statement, assemble ensuing instructions
INA	Increment A by 1
INB	Increment B by 1
IOR	Inclusive "or" to A
ISX*	Increment X and skip if zero
ISY*	Increment Y and skip if zero
ISZ	Increment, then skip if zero
JLY*	Jump and load Y
JMP	Jump
JPY*	Jump indexed by Y
JRS	Jump and restore status
JSB	Jump to subroutine
LAX*	Load A from memory indexed by X
LAY*	Load A from memory indexed by Y
LBT*	Load byte
LBX*	Load B from memory indexed by X
LBY*	Load B from memory indexed by Y
LDA	Load into A
LDB	Load into B
LDX*	Load X from memory
LDY*	Load Y from memory
LFA	Load fence from A
LFB	Load fence from B
LIA	Load into A from I/O channel
LIB	Load into B from I/O channel
LSL	Logical long shift left
LSR	Logical long shift right
LST	Resume list output (follows a UNL)
MBF*	Move bytes from alternate map
MBI*	Move bytes into alternate map
MBT*	Move bytes
MBW*	Move bytes within alternate map
MIA	Merge (or) into A from I/O channel
MIB	Merge (or) into B from I/O channel
MIC	Define user instruction
MPY	Multiply
MVW*	Move words
MWF*	Move words from alternate map
MWI*	Move words into alternate map
MWW*	Move words within alternate map
NAM	Name relocatable program
NOP	No operation
OCT	Define octal constant
ORB	Establish origin in base page
ORG	Establish program origin
ORR	Reset program location counter
OTA	Output from A to I/O channel
OTB	Output from B to I/O channel
PAA	Load/store Port A map per A
PAB	Load/store Port A map per B
PBA	Load/store Port B map per A
PBB	Load/store Port B map per B
RAL	Rotate A left 1
RAM	Generate executable jump to microcode
RAR	Rotate A right 1
RBL	Rotate B left 1
RBR	Rotate B right 1
REP	Repeat next statement
RPL	Replace instruction definition
RRL	Rotate A and B left
RRR	Rotate A and B right
RSA	Read status register into A
RSB	Read status register into B
RSS	Reverse skip sense
RVA	Read violation register into A
RVB	Read violation register into B
SAX*	Store A into memory indexed by X
SAY*	Store A into memory indexed by Y
SBS*	Set bits
SBT*	Store byte
SBX*	Store B into memory indexed by X
SBY*	Store B into memory indexed by Y
SEZ	Skip if E = 0
SFB*	Scan for byte
SFC	Skip if I/O flag = 0 (clear)
SFS	Skip if I/O flag = 1 (set)

SJP	Enable System map and jump
SJS	Enable System map and jump to subroutine
SKP	Skip to top of next page
SLA	Skip if LSB of A = 0
SLB	Skip if LSB of B = 0
SOC	Skip if overflow bit = 0 (clear)
SOS	Skip if overflow bit = 1 (set)
SPC	Space n lines
SSA	Skip if sign A = 0
SSB	Skip if sign B = 0
SSM	Store status register in memory
STA	Store A
STB	Store B
STC	Set I/O control bit
STF	Set I/O flag
STO	Set overflow bit
STX*	Store X into memory
STY*	Store Y into memory
SUP	Suppress list output of additional code lines
SWP	Switch A and B
SYA	Load/store System map per A
SYB	Load/store System map per B
SZA	Skip if A = 0
SZB	Skip if B = 0
TBS*	Test bits
UJP	Enable User map and jump
UJS	Enable User map and jump to subroutine
UNL	Suppress list output
UNS	Resume list output
USA	Load/store User map per A
USB	Load/store User map per B
XAX*	Exchange A and X
XAY*	Exchange A and Y
XBX*	Exchange B and X
XBY*	Exchange B and Y
XCA*	Cross compare A
XCB*	Cross compare B
XIF	Terminate IFN or IFZ group of instructions
XLA*	Cross load A
XLB*	Cross load B
XMA	Transfer maps internally per A
XMB	Transfer maps internally per B
XMM	Transfer map or memory
XMS	Transfer maps sequentially
XOR	Exclusive "or" to A
XSA*	Cross store A
XSB*	Cross store B

CONSOLIDATED CODING SHEETS

APPENDIX D

Table D-1 presents the binary codes for the base set instructions while Table D-2 presents those for the extended instruction group.

Table D-1. Base Set Instruction Codes in Binary

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D/I	AND		001	0	Z/C	← Memory Address →									
D/I	XOR		010	0	Z/C										
D/I	IOR		011	0	Z/C										
D/I	JSB		001	1	Z/C										
D/I	JMP		010	1	Z/C										
D/I	ISZ		011	1	Z/C										
D/I	AD*		100	A/B	Z/C										
D/I	CP*		101	A/B	Z/C										
D/I	LD*		110	A/B	Z/C										
D/I	ST*		111	A/B	Z/C										

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	SRG		000	A/B	0	D/E	*LS		000	†CLE	D/E	‡SL*	*LS		000
				A/B	0	D/E	*RS		001		D/E		*RS		001
				A/B	0	D/E	R*L		010		D/E		R*L		010
				A/B	0	D/E	R*R		011		D/E		R*R		011
				A/B	0	D/E	*LR		100		D/E		*LR		100
				A/B	0	D/E	ER*		101		D/E		ER*		101
				A/B	0	D/E	EL*		110		D/E		EL*		110
				A/B	0	D/E	*LF		111		D/E		*LF		111
				NOP	000				000		000				000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	ASG		000	A/B	1	CL*	01	CLE	01	SEZ	SS*	SL*	IN*	SZ*	RSS
				A/B		CM*	10	CME	10						
				A/B		CC*	11	CCE	11						

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	IOG		000		1	H/C	HLT		000	← Select Code →					
					1	0	STF		001						
					1	1	CLF		001						
					1	0	SFC		010						
					1	0	SFS		011						
				A/B	1	H/C	MI*		100						
				A/B	1	H/C	LI*		101						
				A/B	1	H/C	OT*		110						
				0	1	H/C	STC		111						
				1	1	H/C	CLC		111						
					1	0	STO		001		000			001	
					1	1	CLO		001		000			001	
					1	H/C	SOC		010		000			001	
					1	H/C	SOS		011		000			001	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	EAG		000	MPY**		000		010			000			000	
				DIV**		000		100			000			000	
				DLD**		100		010			000			000	
				DST**		100		100			000			000	
				ASR		001		000		0	1				
				ASL		000		000		0	1				
				LSR		001		000		1	0		number		
				LSL		000		000		1	0	← of →			
				RRR		001		001		0	0		bits		
				RRL		000		001		0	0				

Notes: * = A or B, according to bit 11.
D/I, A/B, Z/C, D/E, H/C coded: 0/1.
**Second word is Memory Address.

†CLE: Only this bit is required.
‡SL*: Only this bit and bit 11 (A/B as applicable) are required.

Table D-2. Extended Instruction Group Codes in Binary

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SAX/SAY/SBX/SBY	1	0	0	0	A/B	0	1	1	1	1	1	0	X/Y	0	0	0
CAX/CAY/CBX/CBY	1	0	0	0	A/B	0	1	1	1	1	1	0	X/Y	0	0	1
LAX/LAY/LBX/LBY	1	0	0	0	A/B	0	1	1	1	1	1	0	X/Y	0	1	0
STX/STY	1	0	0	0	1	0	1	1	1	1	1	0	X/Y	0	1	1
CXA/CYA/CXB/CYB	1	0	0	0	A/B	0	1	1	1	1	1	0	X/Y	1	0	0
LDX/LDY	1	0	0	0	1	0	1	1	1	1	1	0	X/Y	1	0	1
ADX/ADY	1	0	0	0	1	0	1	1	1	1	1	0	X/Y	1	1	0
XAX/XAY/XBX/XBY	1	0	0	0	A/B	0	1	1	1	1	1	0	X/Y	1	1	1
ISX/ISY/DSX/DSY	1	0	0	0	1	0	1	1	1	1	1	1	X/Y	0	0	I/D
JUMP INSTRUCTIONS	1	0	0	0	1	0	1	1	1	1	1	1		0	1	0

JLY = 0
JPY = 1

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BYTE INSTRUCTIONS	1	0	0	0	1	0	1	1	1	1	1	1	0			

LBT = 0 1 1
SBT = 1 0 0
MBT = 1 0 1
CBT = 1 1 0
SFB = 1 1 1

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BIT INSTRUCTIONS	1	0	0	0	1	0	1	1	1	1	1	1	1			

SBS = 0 1 1
CBS = 1 0 0
TBS = 1 0 1

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WORD INSTRUCTIONS	1	0	0	0	1	0	1	1	1	1	1	1	1	1		

CMW = 0
MVW = 1

Table D-2. Extended Instruction Group Codes in Binary (Continued)

MEMORY EXPANSION

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DJP/DJS	1	0	0	0	1	0	1	1	1	1	0	1	1			

DJP = 0 1 0
DJS = 0 1 1

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SYB/USB/PAB/PBB/SSM/JRS	1	0	0	0	1	0	1	1	1	1	0	0	1			

SYB = 0 0 0
USB = 0 0 1
PAB = 0 1 0
PBB = 0 1 1
SSM = 1 0 0
JRS = 1 0 1

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
XMA/XLA/XSA/XCA/LFA	1	0	0	0	0	0	1	1	1	1	0	1	0			

XMA = 0 1 0
XLA = 1 0 0
XSA = 1 0 1
XCA = 1 1 0
LFA = 1 1 1

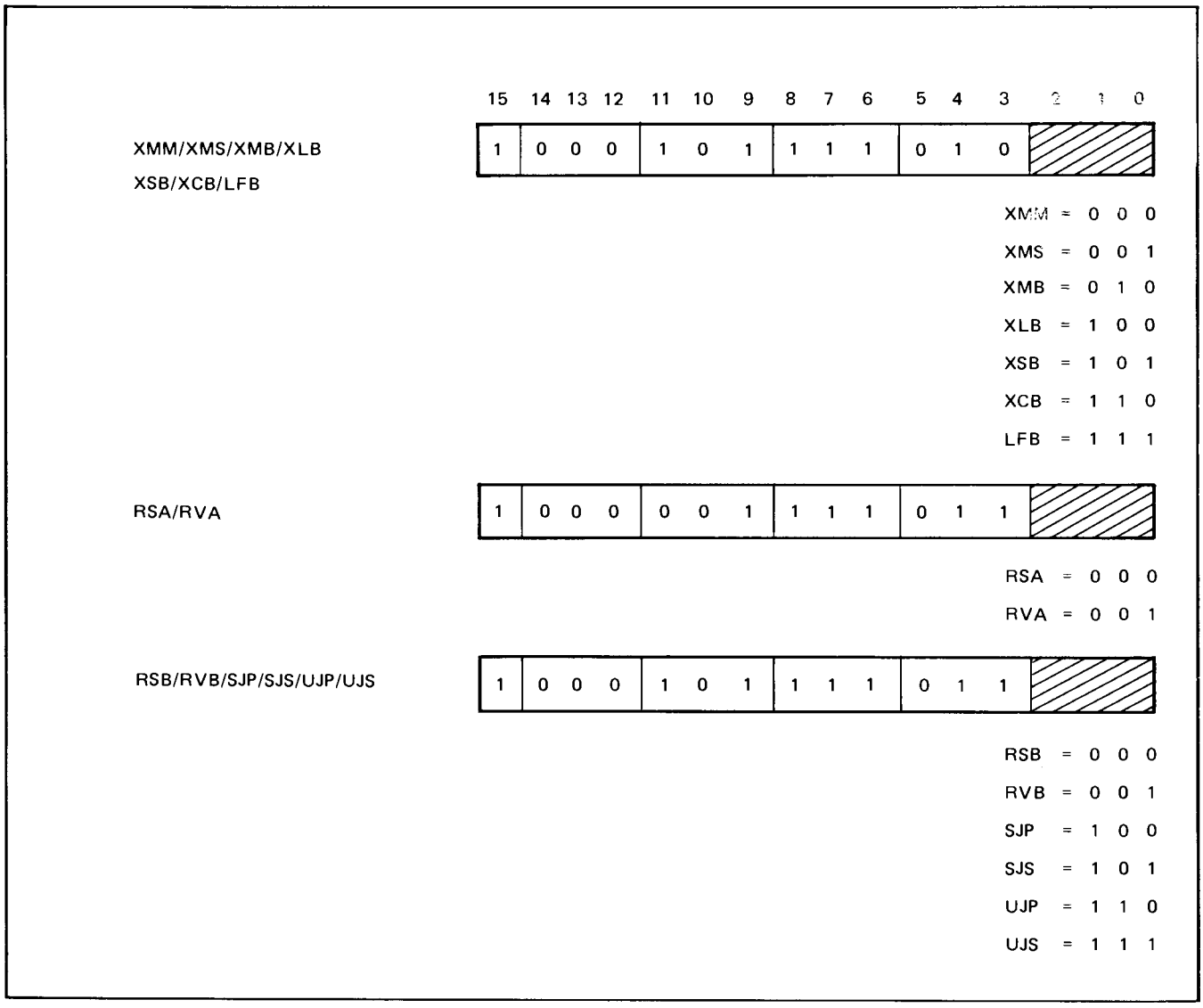
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MBI/MBF/MBW/MWI/MWF/MWW	1	0	0	0	1	0	1	1	1	1	0	0	0			

MBI = 0 1 0
MBF = 0 1 1
MBW = 1 0 0
MWI = 1 0 1
MWF = 1 1 0
MWW = 1 1 1

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SYA/USA/PAA/PBA	1	0	0	0	0	0	1	1	1	1	0	0	1			

SYA = 0 0 0
USA = 0 0 1
PAA = 0 1 0
PBA = 0 1 1

Table D-2. Extended Instruction Group Codes in Binary (Continued)



ASSEMBLER OPERATIONS

APPENDIX E

The Assembler is a segmented program that executes under control of RTE in the User Program Area of main memory. The Assembler consists of a main program (ASMB) and five segments (ASMB0, ASMB1, ASMB2, ASMB3, and ASMB4). It resides on disc, and is read into main memory when called by the RU directive.

Source programs, accepted from either an input device or a user file on the disc, are translated into absolute or relocatable object programs. ASMB will output the relocatable or absolute code to a disc file or device as specified by the binary output parameter when the assembler is invoked. If the source is on a device other than a disc file, it is stored on scratch tracks on the disc as it is being read. If there is insufficient space on the disc, the Assembler is suspended until more scratch tracks are available.

E-1. ON-LINE LOADING OF THE ASSEMBLER

The following example illustrates the on-line loading of the Assembler in an RTE-IV Operating System. The size of the program should be increased to at least nine pages, with twelve pages being a recommended size. The extra space is needed by the assembler for its symbol table.

```
:RU,LOADR
   /LOADR:  SZ,12              *increase size of program
   /LOADR:  RE,%4ASMB          *relocate main module
   /LOADR:  SE,%CLIB           *search compiler library
   /LOADR:  RE,%4ASB0          *relocate segment 0
   /LOADR:  SE,%CLIB           *search compiler library
   /LOADR:  RE,%4ASB1          *relocate segment 1
   /LOADR:  SE,%CLIB           *search compiler library
     .
     .
   /LOADR:  RE,%4ASB4          *relocate segment 4
   /LOADR:  SE,%CLIB1          *search compiler library
   /LOADR:  EN                 *end LOADR operation
```

The Assembler must be loaded as a type 3 program which is also the default type used by the LOADR.

RTE-L ON-LINE LOADING OF THE ASSEMBLER

The following example illustrates the on-line loading of the Assembler in an RTE-L Operating System.

```
:RU,LOADR
    LOADR:  LIB,$CLIB     *use $CLIB to do a search
    LOADR:  SGMENTS,5     *inform LOADER there is 5 segments
    LOADR:  REL,%ASMB     *relocate the main and five segments
                          *which are all in the same file.
    LOADR:  END           *end the load process, libraries are
                          *now searched.
```

For RTE-XL insert "SZ,12" as the first loader command.

E-2. ASSEMBLER OPERATION

The RTE Assembler is initiated with a RU directive in the following form:

:RU,ASMB, *source input* [*list ,output* [*binary ,output* [*line ,count* *,options*]]]

where:

source input

Name of an FMGR file or a logical unit number of the device containing the Assembly source code; this entry must conform to the format required by the FMGR namr parameter. The source input must always be specified.

If an interactive device is specified, the Assembler will print a right bracket (]) on the device as a prompt. It will then accept input a line at a time and output another prompt until an END statement is entered.

list output

Choose one of the following:

- (minus symbol)
FMGR file name
logical unit number
null (omitted)

If the minus symbol is specified, and the source file name begins with an ampersand (&), the list file name will consist of the source file name with the ampersand replaced by an apostrophe ('). For example:

&FIL1	source file name
'FIL1	list file name

The list file is always forced to reside on the same cartridge (cartridge reference code) as the source file. If an FMGR file by this name does not already exist it is created. The created list file is given the same file security code as that of the source if it was specified in the source namr of the run sequence.

If an FMGR file name is specified, it must conform to the format required by the FMGR namr parameter. The list file is created if it does not exist. If the file does exist, the first character in the file name must be an apostrophe ('); otherwise, an FMP −15 error will result indicating that the file name is illegal.

If a logical unit number is specified, the list output is directed to that logical device.

If this parameter is omitted, the logical unit of the interactive input device is assumed. Furthermore, if subsequent parameters are specified, the comma must be used as a parameter placeholder.

binary output

Choose one of the following:

- (minus symbol)
FMGR file name
logical unit number
null (omitted)

If the minus symbol is specified, and the source file name begins with an ampersand (&), the binary file name will consist of the source file name with the ampersand replaced by a percent sign (%). For example:

&FIL1	source file name
%FIL1	binary file name

This binary file is always forced to reside on the same cartridge (cartridge reference code) as the source file. If an FMGR file by this name does not already exist it is created. The created list file is given the same file security code as that of the source namr if it was specified in the source namr of the run sequence.

If an FMGR file name is specified, it must conform to the format required by the FMGR namr parameter. The binary file is created if it does not exist. If the file exists, it is necessary that:

a. the first character of the file's name be a percent sign (%) or !

b. the existing file be of the type specified in the namr parameter. If the file type is not declared in namr, the file's type must be type 5 or type 7, relocatable binary.

If the above conditions are not met, an FMP-15 error will result.

If a logical unit number is specified, the binary output is directed to that logical device.

If this parameter is omitted, binary output is not produced. Furthermore, if subsequent parameters are specified, a comma must be used as a parameter placeholder.

line count

A decimal number which defines the number of lines per page for the list device.

Specification of this parameter is optional. If it is omitted, 55 lines per page are assumed.

options

Up to six characters that select control function options. No commas are allowed within the option string. These characters are: A,R,B,L,Q,T,N,Z,C,F,X, and P (see Table 1-2 for an explanation of these options). If specified when the Assembler is run, these options replace (override) the options declared in the ASMB control statement.

The P option has a special meaning in this context. If P is specified by itself, the Assembler will output the object code (if the binary output parameter has been specified) and the error reports and take no further actions. The type of object code is determined by the source program control statement. If the P option is specified with any other option, the P option is ignored.

The R and A options cannot be overridden. Any attempt to do so will cause the Assembler to generate a CS error and abort. The R and A options are always determined by the source program control statement.

Examples:

*RU,ASMB,&PROGA,-,-

Schedules RTE ASMB to assemble the source code in file &PROGA. Listed output is directed to list file 'PROGA and binary relocatable code is directed to binary file %PROGA. The number of lines per list file defaults to 55.

:RU,ASMB,&FIL1,'LIST

Schedules RTE ASMB to assemble source file &FIL1. Listed output is directed to list file 'LIST. No binary relocatable code is generated. The number of lines per list file page defaults to 55.

:RU,ASMB,&ABCD

Schedules RTE ASMB to assemble source file &ABCD. Listed output defaults the user terminal. No binary relocatable code is generated. The number of lines per list file page defaults to 55.

:RU,ASMB,&AAAA,-,-,28,LZ

Schedules RTE ASMB to assemble source file &AAAA. Listed output is directed to list file 'AAAA. Binary relocatable code is directed to binary file %AAAA. The number of lines per list file page is 28.

A listing will be produced because the L option has been specified. The Assembler will perform a selective assembly with respect to the Z option (see section IV on the IFZ and IFN pseudo-instructions).

:RU,ASMB,&SFIL,-,-,,TQC

Schedules RTE ASMB to assemble source file &SFIL. Listed output is directed to list file 'SFIL. Binary relocatable code is directed to binary file %SFIL. T will cause a symbol table listing to be output to the list file.

Q will cause the memory reference instructions in the object code listing to appear as addresses only (the opcode will not appear in the listing). C will cause a cross-reference table to be output to the list file.

:RU,ASMB,&SFIL,−,−,,P

The P option will cause the Assembler to produce only object code and error messages. It overrides the control options specified in the assembly language source program.

E-3. MESSAGES DURING ASSEMBLY

a. When the end of a source tape is encountered, the following message is output to the system console:

RTE-IV:
I/O ET L #x E #y S #z

LU #x is unavailable until the operator declares the associated EQT up using the RTE UP command:

UP,y

RTE-L/XL:
I/O-ET @LUnn

LU #nn is unavailable until the operator declares it up using the UP,nn command.

Assembly continues after the UP. More than one source tape can be assembled into one program by loading the next tape before giving the UP.

b. If an FMP error occurs during the assembly, the Assembler will print the following message on the system console.

/ASMB: FMP−*nn* { SOURCE / LIST / BINARY }

where:

−*nn* is the FMP error number

either SOURCE,LIST, or BINARY will be printed according to which file caused the error. The current assembly is aborted.

c. The following message on the system console signifies the end of the assembly:

/ASMB: $END

d. If an error is found in the Assembler control statement, the following message is output to the system console.

/ASMB: $END CS

and the current assembly stops.

e. If an end-of-file condition on the source input occurs before an END statement is found, the system console signals:

/ASMB:
$END
XEND

and the current assembly stops.

f. If the source input file does not exist, the system console signals:

/ASMB: $END NPRG

and the current assembly stops.

g. During pass 1, the Assembler will output error messages for each error it finds. Immediately above the error message, the number of the tape containing the error will be printed in the following form:

#nnn

The tape counter starts with one and increments by one whenever an end-of-tape condition occurs (using paper tape), or a blank card is encountered. When the counter increments, the numbering of the source statements starts over at one.

h. Each error diagnostic printed in the program listing during pass two of the assembly is associated with a different message (printed on a separate line just above each diagnostic):

PG *ppp*

ppp is the page number (in the listing) of the previous error diagnostic. PG 000 is associated with the first error found in the program.

i. At the end of pass 2, the Assembler will display the total error count on the system console in the following form:

/ASMB: *xxxx* ERRORS TOTAL

where *xxxx* is "NO" if 0 errors occurred, or the number of errors otherwise. The Assembler will also return the number of errors that occurred to the program that scheduled it as the first return parameter. This parameter may be retrieved using a call to the library subroutine RMPAR.

MACHINE INSTRUCTION SET SUMMARY

APPENDIX F

The following alphabetic list details the instructions that are available on the various HP 1000 computers. An asterisk in the L-SERIES column indicates that the instruction is implemented in software in RTE-L/XL systems (see Section 3 paragraph 3-28). An "NA" indicates that the instruction is not available on that model of computer. It is assumed that the M, E and F-Series computers contain the Dynamic Mapping System instructions.

Table F-1. M, E, F and L-Series Instruction Sets

INSTRUCTION		M/E/F-SERIES	L-SERIES
ABS	Define absolute value		
ADA	Add to A		
ADB	Add to B		
ADX	Add memory to X		*
ADY	Add memory to Y		*
ALF	Rotate A left 4		
ALR	Shift A left 1, clear sign		
ALS	Shift A left 1		
AND	"And" to A		
ARS	Shift A right 1, sign carry		
ASC	Generate ASCII characters		
ASL	Arithmetic long shift left		
ASR	Arithmetic long shift right		
BLF	Rotate B left 4		
BLR	Shift B left 1, clear sign		
BLS	Shift B left 1		
BRS	Shift B right 1, carry sign		
BSS	Reserve block of storage starting at symbol		
BYT	Defines octal byte constants		
CAX	Copy A to X		*
CAY	Copy A to Y		*
CBS	Clear bits		*
CBT	Compare bytes		*
CBX	Copy B to X		*
CBY	Copy B to Y		*
CCA	Clear and complement A (1's)		
CCB	Clear and complement B (1's)		
CCE	Clear and complement E (set E = 1)		
CLA	Clear A		
CLB	Clear B		

Table F-1. M, E, F and L-Series Instruction Sets (Continued)

INSTRUCTION		M/E/F-SERIES	L-SERIES
CLC	Clear I/O control bit		
CLE	Clear E		
CLF	Clear I/O flag		
CLO	Clear overflow bit		
CMA	Complement A		
CMB	Complement B		
CME	Complement E		
CMW	Compare words		*
COM	Reserve block of common storage		
CPA	Compare to A, skip if unequal		
CPB	Compare to B, skip if unequal		
CXA	Copy X to A		*
CXB	Copy X to B		*
CYA	Copy Y to A		*
CYB	Copy Y to B		*
DBL	Define left byte (bits 8-15) address		
DBR	Define right byte (bits 0-7) address		
DEC	Define decimal constant		
DEF	Define address		
DEX	Define extended precision constant (3 word constant)		
DEY	Define extended precision constant (4 word constant)		
DIV	Divide		
DJP	Disable MEM and jump		NA
DJS	Disable MEM and jump to subroutine		NA
DLD	Double load		
DST	Double store		
DSX	Decrement X and skip if zero		*
DSY	Decrement Y and skip if zero		*
ELA	Rotate E and A left 1		
ELB	Rotate E and B left 1		
EMA	Extended Memory Area		NA
END	Terminate program		
ENT	Entry point		
ERA	Rotate E and A right 1		
ERB	Rotate E and B right 1		
EQU	Equate symbol		
EXT	External reference		
FAD	Floating add		*
FDV	Floating divide		*
FIX	Convert floating-point to fixed-point		*

Table F-1. M, E, F and L-Series Instruction Sets (Continued)

INSTRUCTION		M/E/F-SERIES	L-SERIES
FLT	Convert fixed-point to floating-point		*
FMP	Floating multiply		*
FSB	Floating subtract		*
HED	Print heading at top of each page		
HLT	Halt		
IFN	When N appears in Control statement, assemble ensuing instructions		
IFZ	When Z appears in Control statement, assemble ensuing instructions		
INA	Increment A by 1		
INB	Increment B by 1		
IOR	Inclusive "or" to A		
ISX	Increment X and skip if zero		*
ISY	Increment Y and skip if zero		*
ISZ	Increment, then skip if zero		
JLY	Jump and load Y		*
JMP	Jump		
JPY	Jump indexed by Y		*
JRS	Jump and restore status		NA
JSB	Jump to subroutine		
LAX	Load A from memory indexed by X		*
LAY	Load A from memory indexed by Y		*
LBT	Load byte		*
LBX	Load B from memory indexed by X		*
LBY	Load B from memory indexed by Y		*
LDA	Load into A		
LDB	Load into B		
LDX	Load X from memory		*
LDY	Load Y from memory		*
LFA	Load fence from A		NA
LFB	Load fence from B		NA
LIA	Load into A from I/O channel		
LIB	Load into B from I/O channel		
LSL	Logical long shift left		
LSR	Logical long shift right		
LST	Resume list output (follows a UNL)		
MBF	Move bytes from alternate map		*
MBI	Move bytes into alternate map		*
MBT	Move byte		*

Table F-1. M, E, F and L-Series Instruction Sets (Continued)

INSTRUCTION		M/E/F-SERIES	L-SERIES
MBW	Move bytes within alternate map		*
MIA	Merge (or) into A from I/O channel		
MIB	Merge (or) into B from I/O channel		
MIC	Define user instruction		
MPY	Multiply		
MVW	Move words		*
MWF	Move words from alternate map		*
MWI	Move words into alternate map		*
MWW	Move words within alternate map		*
NAM	Name relocatable program		
NOP	No operation		
OCT	Define octal constant		
ORB	Establish origin in base page		
ORG	Establish program origin		
ORR	Reset program location counter		
OTA	Output from A to I/O channel		
OTB	Output from B to I/O channel		
PAA	Load/store Port A map per A		NA
PAB	Load/store Port A map per B		NA
PBA	Load/store Port B map per A		NA
PBB	Load/store Port B map per B		NA
RAL	Rotate A left 1		
RAM	Generate executable jump to microcode		NA
RAR	Rotate A right 1		
RBL	Rotate B left 1		
RBR	Rotate B right 1		
REP	Repeat next statement		
RPL	Replace instruction definition		
RRL	Rotate A and B left		
RRR	Rotate A and B right		
RSA	Read status register in A		NA
RSB	Read status register into B		NA
RSS	Reverse skip sense		
RVA	Read violation register into A		NA
RVB	Read violation register into B		NA
SAX	Store A into memory indexed by X		*
SAY	Store A into memory indexed by Y		*

Table F-1. M, E, F and L-Series Instruction Sets (Continued)

INSTRUCTION		M/E/F-SERIES	L-SERIES
SBS	Set Bits		*
SBT	Store byte		*
SBX	Store B into memory indexed by X		*
SBY	Store B into memory indexed by Y		*
SEZ	Skip if E = 0		
SFB	Scan for byte		*
SFC	Skip if I/O flag = 0 (clear)		
SFS	Skip if I/O flag = 1 (set)		
SJP	Enable System map and jump		NA
SJS	Enable System map and jump to subroutine		NA
SKP	Skip to top of next page		
SLA	Skip if LSB of A = 0		
SLB	Skip if LSB of B = 0		
SOC	Skip if overflow bit = 0 (clear)		
SOS	Skip if overflow bit = 1 (set)		
SPC	Space n lines		
SSA	Skip if sign A = 0		
SSB	Skip if sign B = 0		
SSM	Store status register in memory		NA
STA	Store A		
STB	Store B		
STC	Set I/O control bit		
STF	Set I/O flag		
STO	Set overflow bit		
STX	Store X into memory		*
STY	Store Y into memory		*
SUP	Suppress list output of additional code lines		
SWP	Switch A and B		
SYA	Load/store System map per A		NA
SYB	Load/store System map per B		NA
SZA	Skip if A = 0		
SZB	Skip if B = 0		
TBS	Test bits		*
UJP	Enable User map and jump		NA
UJS	Enable User map and jump to subroutine		NA
UNL	Suppress list output		
UNS	Resume list output		
USA	Load/store User map per A		NA
USB	Load/store User map per B		NA

Table F-1. M, E, F and L-Series Instruction Sets (Continued)

INSTRUCTION		M/E/F-SERIES	L-SERIES
XAX	Exchange A and X		*
XAY	Exchange A and Y		*
XBX	Exchange B and X		*
XBY	Exchange B and Y		*
XCA	Cross compare A		*
XCB	Cross compare B		*
XIF	Terminate IFN or IFZ group of instructions		
XLA	Cross load A		*
XLB	Cross load B		*
XMA	Transfer maps internally per A		NA
XMB	Transfer maps internally per B		NA
XMM	Transfer map or memory		NA
XMS	Transfer maps sequentially		NA
XOR	Exclusive "or" to A		
XSA	Cross store A		*
XSB	Cross store B		*

ASSEMBLER ERROR MESSAGES

APPENDIX G

Errors detected in the source program are indicated by a 1- or 2-letter mnemonic followed by the sequence number and the first 62 characters of the statement in error. The messages are printed on the list output device during the passes indicated. A message specifying the number of errors detected is printed on the system console device at the end of each pass.

Error listings produced during Pass 1 are preceded by a number which identifies the source input file where the error was found. Pass 2 error messages are preceded by a reference to the previous page of the listing where an error message was written. The first error will refer to page "0". The error count at the end of Pass 2 is preceded by the page number in the listing where the final error was encountered.

Error Code	Pass	Description
CS	1	Control statement error: a. The control statement contained a parameter other than the legal set. b. Both A and R were specified. c. Both F and X were specified.
DD	1	Doubly defined symbol: A name defined in the symbol table appears more than once as: a. A label of a machine instruction. b. A label of one of the pseudo operations: BSS DBL EMA DBR BYT EQU ASC ABS DEC OCT DEF Arithmetic subroutine call DEX c. A name in the Operand field of a COM or EXT statement. d. A label in an instruction following a REP pseudo operation. e. Any combination of the above. An arithmetic subroutine call symbol appears in a program both as a pseudo instruction and as a label.
EN	1	The symbol specified in an ENT statement has already been defined in an EXT statement, or is a label for an EMA pseudo-instruction.
EN UNDEF <symbol>	2	The entry point specified in an ENT statement does not appear in the label field of a machine or BSS instruction. The entry point has been defined in the Operand field of an EXT statement.
IF	1	An IFZ or an IFN follows either an IFZ or an IFN without an intervening XIF. The second pseudo instruction is ignored.

Error Code	Pass	Description
IL	1	Illegal instruction: a. Instruction mnemonic cannot be used with type of assembly requested in control statement. The following are illegal in an absolute assembly. NAM EXT EMA ENT COM Arithmetic subroutine calls b. The ASMB statement has an R parameter, and NAM has been detected after the first valid Opcode. c. An EMA pseudo-instruction is encountered more than once.
	1 or 2	Illegal character: A numeric term used in the Operand field contains an illegal character (e.g., an octal constant contains other than +, −, or 0-7). This code may also appear following an M error for missing operands.
LB	1	Missing label in an EQU, RPL or EMA pseudo-instruction.
M	1 or 2	Illegal operand: a. Operand is missing for an Opcode requiring one. b. Operands are optional and omitted but comments are included for: END HLT c. Operand is an external symbol or an indirect address for: DBL DBR d. An absolute expression in one of the following instructions from a relocatable program is greater than 1777_8. Instructions referencing memory locations: DEF, DBL, and DBR Arithmetic subroutine calls e. A negative operand is used with an Opcode other than ABS, DEX, DEC, OCT, and BYT. f. A character other than I follows a comma with operands which can be indirect. g. Operand is an indirect address when used with JPY. h. Using a literal as the second operand in the following instructions: TBS SBS CBS i. A character other than C follows a comma in certain I/O instructions. j. A relocatable expression in the Operand field of one of the following: ABS ASR RRL REP ASL LSR SPC RRR LSL k. An ORG statement appearing in a relocatable program includes an expression that is common relocatable or absolute. l. A relocatable expression contains a mixture of program and common relocatable terms. m. The literal, literal code, or type of literal is illegal for the operation code used (e.g., STA = B7).

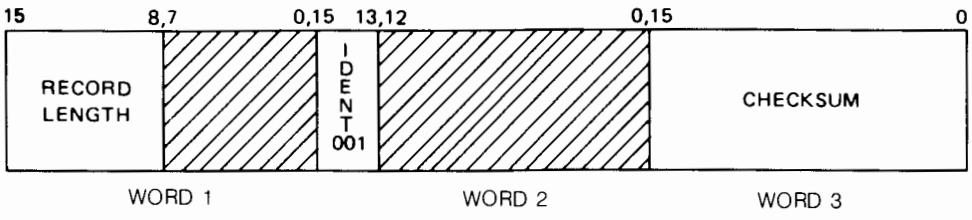
Error Code	Pass	Description
		n. An integer expression in one of the following instructions does not meet the condition $1 \leq n \leq 16$. The integer is evaluated modulo 2^4. ASR RRR LSR ASL RRL LSL
		o. The value of an 'L' type literal is relocatable.
		p. The number of words, n, specified for an ASCII string definition, ASC n, exceeds 28 decimal words.
		q. There is no comma after the first operand in an EMA or MIC pseudo-instruction.
		r. One or both of the operands m_1 and m_2 for the EMA instruction do not conform to the bounds specifications.
NO	1 or 2	No origin definition: The first statement in the assembly containing a valid opcode following the ASMB control statement (and remarks and/or HED, if present) is neither an ORG nor a NAM statement. If absolute, the program is assembled starting at 2000; if relocatable, the program is assembled starting at zero.
OP	1 or 2	Illegal Opcode preceding first valid Opcode. The statement being processed does not contain an asterisk in position one. The statement is assumed to contain an illegal Opcode; it is treated as a remarks statement: Illegal Opcode: A mnemonic appears in the Opcode field which is not valid. A word is generated in the object program.
OV	1 or 2	Numeric operand overflow: The numeric value of a term or expression has overflowed its limit: $1 \geqslant N \geqslant 16$ Shift-Rotate Set 2^2-1 Input/Output, Overflow, Halt $2^{10}-1$ Memory Reference (in absolute assembly) 2^{15} Data generated by DEC or DEX $2^{15}-1$ DEF and ABS operands and expressions concerned with program location counter. $2^{16}-1$ OCT
SO		There are more symbols defined in the program than the symbol table can handle.
SY	1 or 2	Illegal Symbol: A Label field contains an illegal character or is greater than 5 characters. A label with illegal characters may result in an erroneous assembly if not corrected. A long label is truncated on the right to 5 characters. Illegal Symbol: A symbolic term in the Operand field is greater than five characters; the symbol is truncated on the right to 5 characters. Too many control statements: The source file contains more than one control statement. The Assembler assumes that the second control statement is a label, since it begins in column 1. Thus, the commas are considered as illegal characters and the "label" is too long. The binary object program is not affected by this error. The first control statement processed is the one used by the Assembler.
UN	1 or 2	Undefined Symbol: a. A symbolic term in an Operand field is not defined in the Label field of an instruction or is not defined in the Operand field of a COM or EXT statement. b. A symbol appearing in the Operand field of one of the following pseudo operations was not defined previously in the source program: BSS ASC EQU ORG END EMA

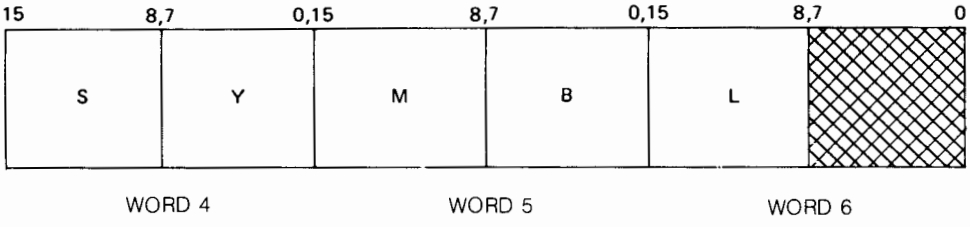
OUTPUT DATA FORMATS

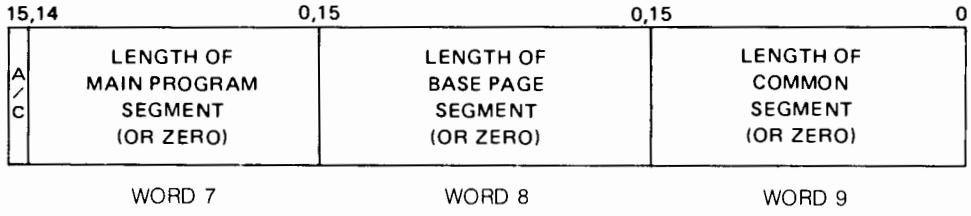
APPENDIX H

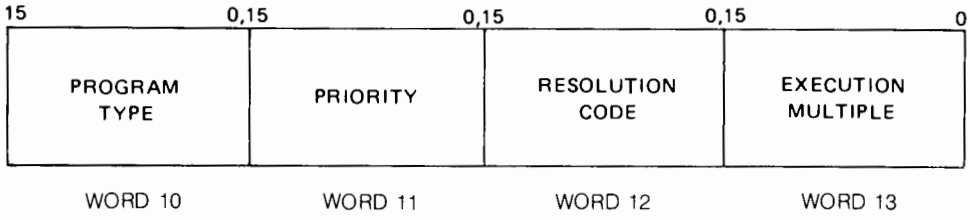
NAM RECORD

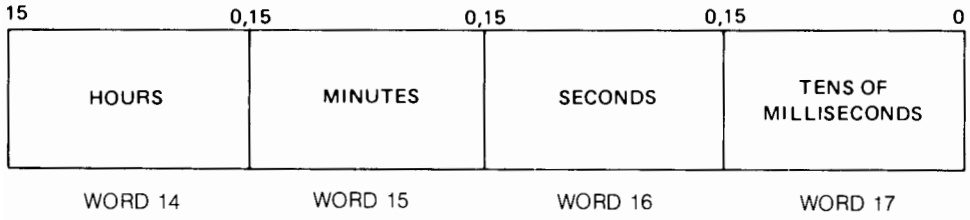
CONTENT

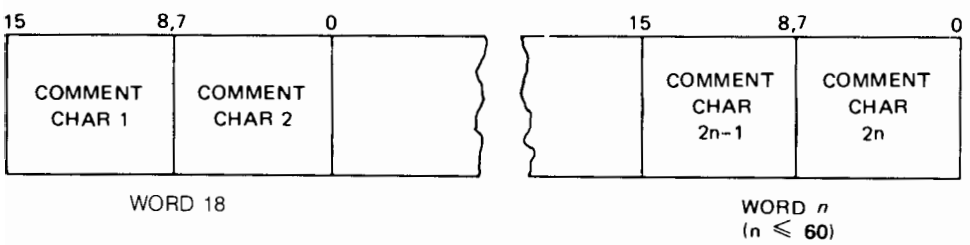












EXPLANATION

RECORD LENGTH = 9-60 WORDS

IDENT = 001

CHECKSUM: ARITHMETIC TOTAL OF ALL WORDS IN RECORD EXCLUDING WORDS 1 AND 3.

SYMBL: FIVE CHARACTER NAME OF PROGRAM

A/C: BINARY TAPE PRECESSION

= 0 IF ASSEMBLER PRODUCED OR LENGTH IS EXACT.

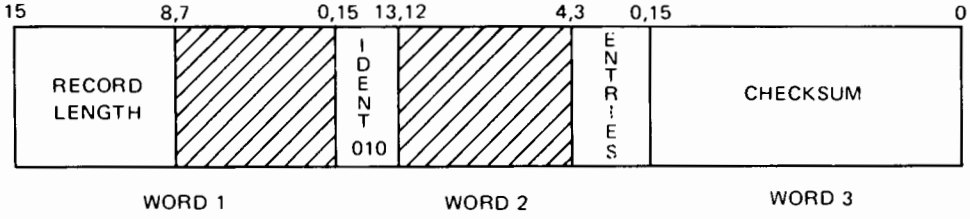
= 1 IF COMPILER PRODUCED AND LENGTH IS UNKNOWN.

HATCH-MARKED AREAS SHOULD BE ZERO-FILLED WHEN THE RECORDS ARE GENERATED

CROSS-HATCH-MARKED AREAS SHOULD BE SPACE FILLED WHEN THE RECORDS ARE GENERATED

ENT RECORD

CONTENT

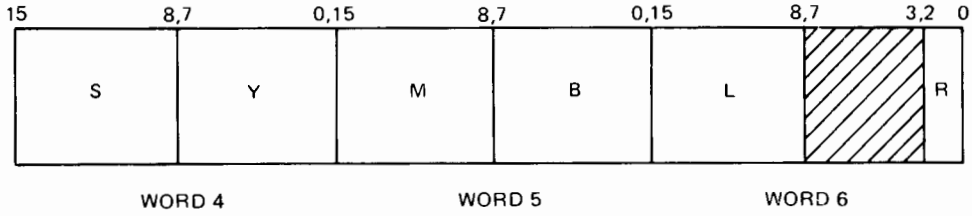


EXPLANATION

RECORD LENGTH = 7-59 WORDS

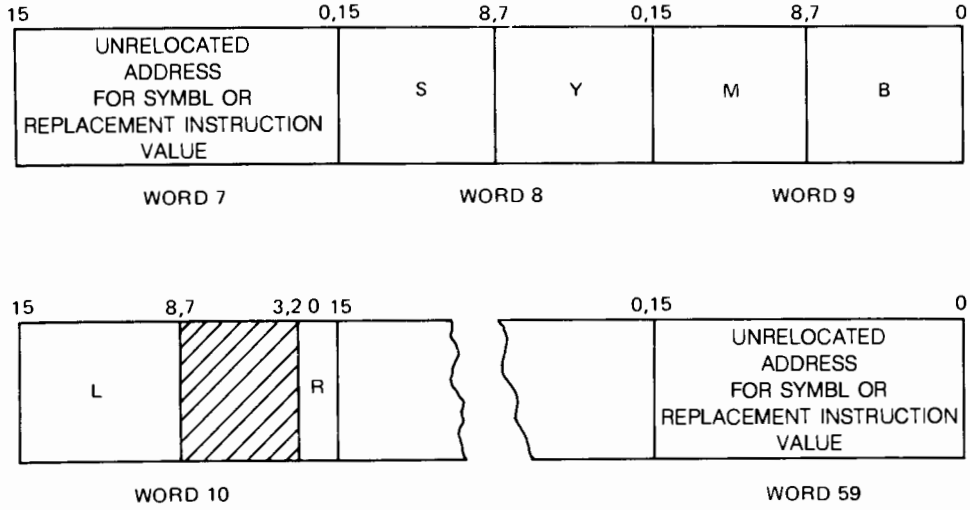
IDENT = 010

ENTRIES: 1 TO 14 ENTRIES PER RECORD; EACH ENTRY IS FOUR WORDS LONG.



SYMBL: 5 CHARACTER ENTRY POINT SYMBOL

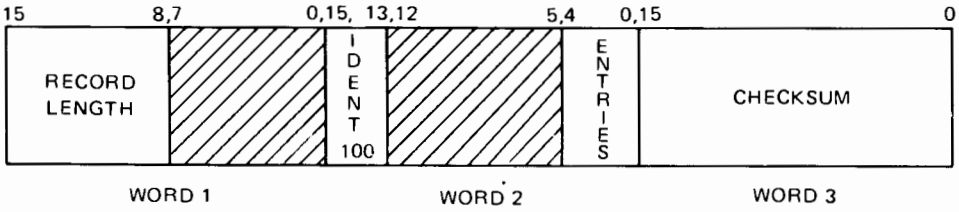
R: RELOCATION INDICATOR
= 0 IN PROGRAM RELOCATABLE
= 1 IF BASE PAGE RELOCATABLE
= 2 IF COMMON RELOCATABLE
= 3 IF ABSOLUTE
= 4 INSTRUCTION REPLACEMENT

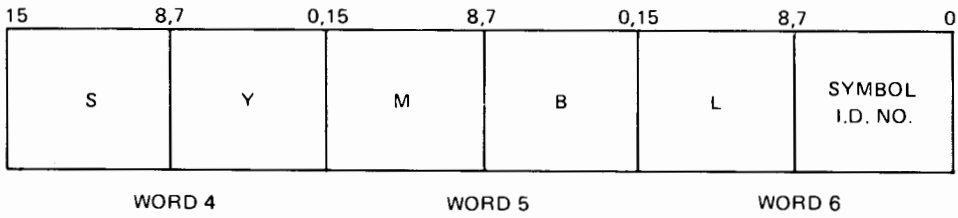


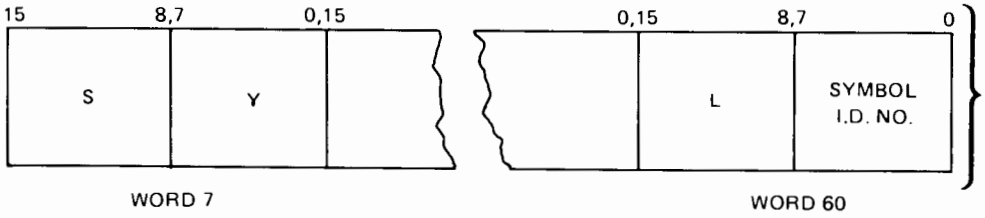
WORDS 4 THROUGH 7 ARE REPEATED FOR EACH ENTRY POINT SYMBOL.

EXT RECORD

CONTENT







EXPLANATION

RECORD LENGTH = 6-60 WORDS

IDENT = 100

ENTRIES: 1 TO 19 PER RECORD; EACH ENTRY IS THREE WORDS LONG

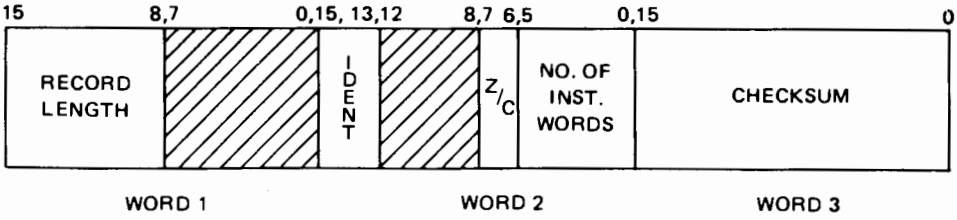
SYMBL: 5 CHARACTER EXTERNAL SYMBOL

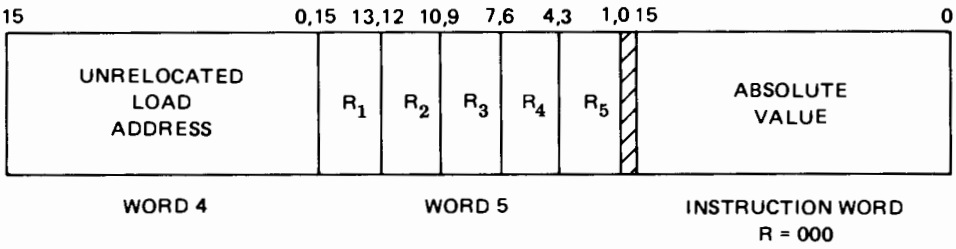
SYMBOL ID. NO.: NUMBER ASSIGNED TO SYMBL FOR USE IN LOCATING REFERENCE IN BODY OF PROGRAM.

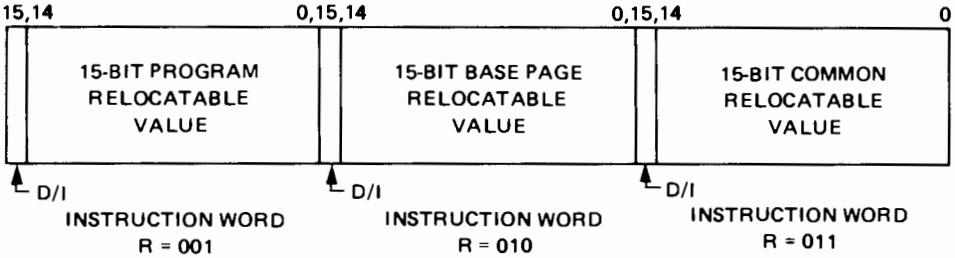
WORDS 4 THROUGH 6 REPEATED FOR EACH EXTERNAL SYMBOL (MAXIMUM OF 19 PER RECORD).

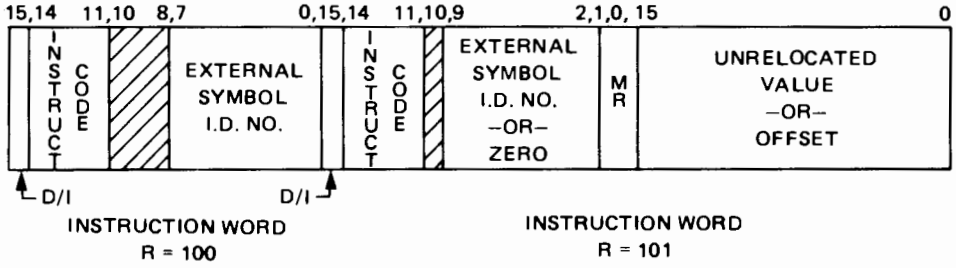
DBL RECORD

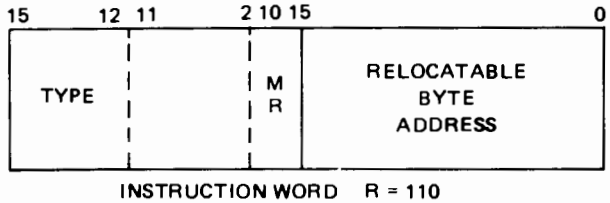
CONTENT











EXPLANATION

RECORD LENGTH = 6-60 WORDS
IDENT = 011
Z/C: RELOCATION OF LOAD ADDRESS
= 0 FOR BASE PAGE
= 1 FOR PROGRAM
= 2 FOR ABSOLUTE
= 3 FOR COMMON
NO. OF INST. WORDS: 1 TO 45 LOADABLE INSTRUCTION WORDS PER RECORD

RELOCATABLE LOAD ADDRESS: STARTING ADDRESS FOR LOADING THE INSTRUCTIONS WHICH FOLLOW;

R's: RELOCATION INDICATORS:
000 = ABSOLUTE
001 = 15-BIT PROGRAM RELOCATABLE
010 = 15-BIT BASE PAGE RELOCATABLE
011 = 15-BIT COMMON RELOCATABLE
100 = EXTERNAL REFERENCE
101 = MEMORY REFERENCE
110 = BYTE ADDRESS

R_1 IS RELOCATION INDICATOR FOR INSTRUCTION $WORD_1$; R_2, FOR INSTRUCTION $WORD_2$; ETC.

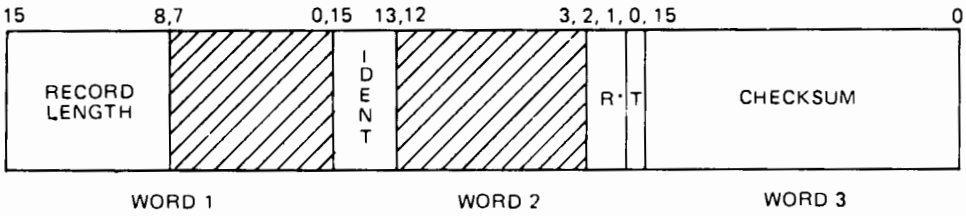
D/I: INDIRECT ADDRESSING

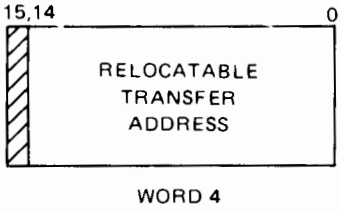
0 = DIRECT
1 = INDIRECT

MEMORY REFERENCE INSTRUCTIONS USE TWO WORDS, WITHIN THE TWO-WORD GROUP "MR" INDICATES RELOCATABILITY OF OPERAND SPECIFIED IN SECOND WORDS:
00 = PROGRAM RELOCATABLE
01 = BASE PAGE RELOCATABLE
10 = COMMON RELOCATABLE
11 = ABSOLUTE

END RECORD

CONTENT





EXPLANATION

RECORD LENGTH = 4 WORDS
IDENT = 101

R: RELOCATION INDICATOR FOR TRANSFER ADDRESS

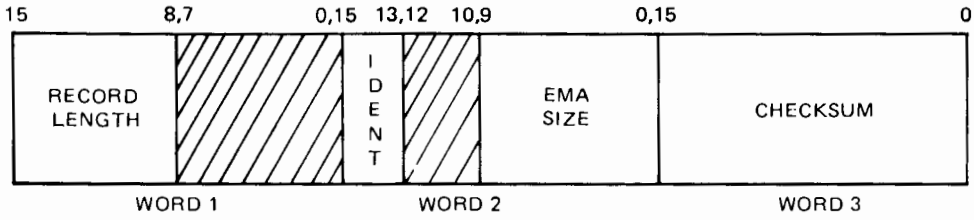
= 0 IF PROGRAM RELOCATABLE
= 1 IF BASE PAGE RELOCATABLE
= 2 IF COMMON RELOCATABLE
= 3 IF ABSOLUTE

T: TRANSFER ADDRESS INDICATOR

= 0 IF NO TRANSFER ADDRESS IN RECORD

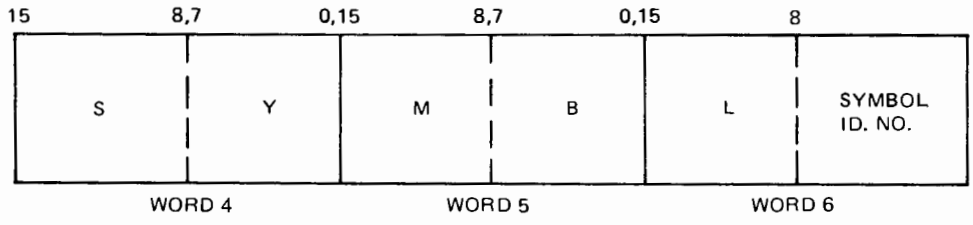
= 1 IF TRANSFER ADDRESS PRESENT

EMA RECORD

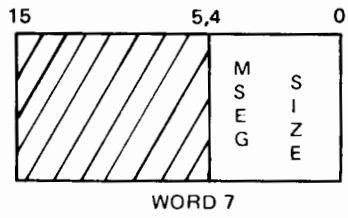


EXPLANATION

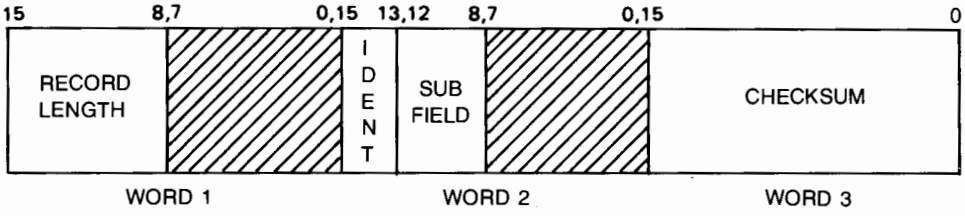
RECORD LENGTH = 7 WORD
IDENT = 110



SYMBOL ID. NO.: NUMBER ASSIGNED TO SYMBL FOR USE IN LOCATING REFERENCE IN BODY OF PROGRAM.



LOADR/GENERATOR INFORMATION RECORD

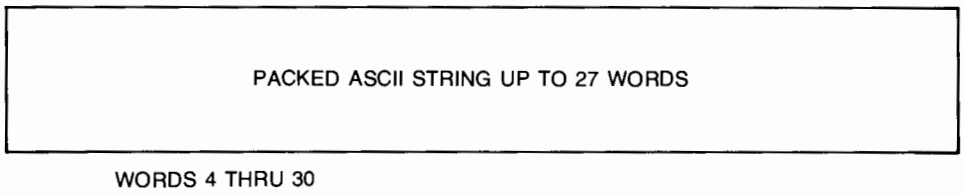


EXPLANATION

RECORD LENGTH = 4-60 WORDS

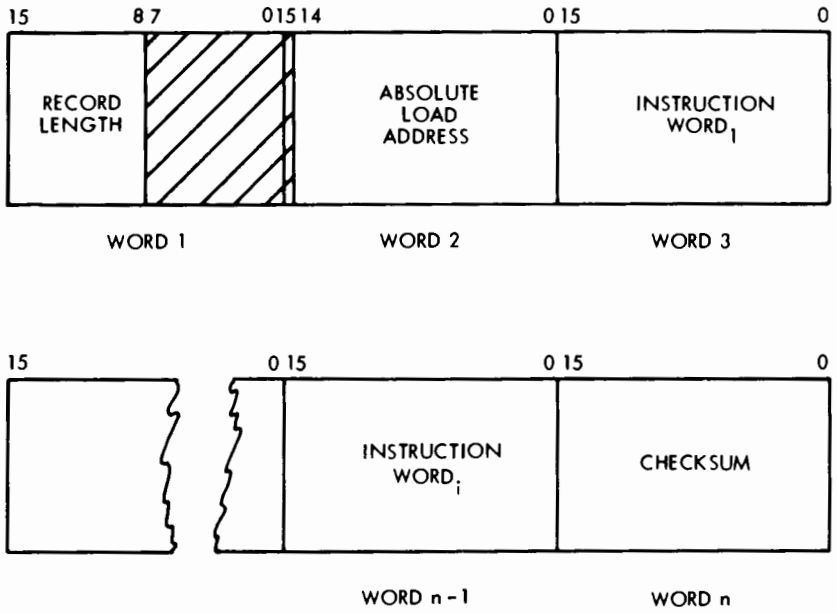
IDENT = 7

SUB FIELD = 0 GEN RECORD
= 5 LOD RECORD



ABSOLUTE FORMAT

CONTENT



EXPLANATION

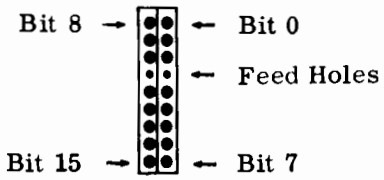
RECORD LENGTH = NUMBER OF WORDS IN RECORD EXCLUDING WORDS 1 AND 2 AND THE LAST WORD.

ABSOLUTE LOAD ADDRESS: STARTING ADDRESS FOR LOADING THE INSTRUCTIONS WHICH FOLLOW

INSTRUCTION WORDS: ABSOLUTE INSTRUCTIONS OR DATA

CHECKSUM: ARITHMETIC TOTAL OF ALL WORDS EXCEPT FIRST AND LAST

On paper tape, each word represents two frames arranged as follows:



RTE CROSS REFERENCE TABLE GENERATOR

APPENDIX I

This Real-Time Executive Operating System Cross Reference Table Generator routine (XREF) processes an assembler source program and provides a list of all symbols and symbol references used within the program.

I-1. COMPUTER CONFIGURATION

The routine requires a Real-Time Executive Operating System with logical unit 1 as the system console and a standard list device.

XREF can be loaded on-line using the RTE LOADR. The LOADR SZ command should be used to increase the program size to reflect the number of symbols in the programs to be cross-referenced. The minimum recommended size is eight pages, with fourteen pages or more preferred to handle programs with large symbol tables. Refer also to the discussion on Bounds for a method to cross-reference only parts of the symbol table at a time.

The following example presents a typical on-line loading of program XREF:

```
:RU,LOADR
    /LOADR:  SZ,12          *size increased to 12 pages
    /LOADR:  RE,%4XREF      *relocate module
    /LOADR:  SE,%CLIB       *search compiler library
    /LOADR:  EN             :end LOADR operations
```

On-line loading of the program XREF under the RTE-L operating system is accomplished by the following:

```
:RU,LOADR
    LOADR:  LIB,$CLIB   *use $CLIB to do a search
    LOADR:  RE,%XREF    *relocate module
    LOADR:  END         *end the load process,
                        *libraries are now searched
```

For RTE-XL, insert "SZ,12" as the first loader command.

I-2. FUNCTIONAL AND OPERATIONAL CHARACTERISTICS

Source program input may come from:

a. An input unit specified by a logical unit number,

b. A disc file.

c. The temporary work area of the disc which was set up by the Assembler in a previous assembly.

I-3. OUTPUT FORMAT

The general format of the output list is:

```
SSSSS DDDDD/TT RRRRR /TT  ... RRRRR /TT
      RRRRR ;TT           ... RRRRR /TT
```

where:

SSSSS
is the symbol which may be any legal label to the assembler.

DDDDD
is the statement number in decimal in which the label was defined. It has a maximum value of 32767, when using the no tape number option, and a maximum value of 2047 when using tape numbers.

TT
is the decimal tape or file number (following a zero length record) with a maximum value of 16.

RRRRR
is the statement number in decimal in which the label was referenced. It has the same physical limits as the defining statement numbers.

Note: The defined format and meaning of SSSSS, DDDDD, and TT are used in the following paragraphs.

I-4. PSEUDO PROCESSING

ORG, ORB, ORR, IFN, IFZ, XIF, MIC, and MIC-defined OPCODES are listed as:

```
**ORG  ********  RRRRR/TT,  ...  RRRRR/TT
**ORB      "         "              "
**ORR      "         "              "
**IFN      "         "              "
**XIF      "         "              "
**MIC      "         "              "
**XYZ      "         "              "
```

(where 'XYZ' is a 'MIC'-defined opcode.)

The defining statement number is replaced by a string of asterisks.

I-5. DOUBLE DEFINED PROCESSING

If a symbol has been defined more than once, it is listed in the symbol list in the following format:

```
SSSSS  ########  RRRRR/TT .... RRRRR/TT
```

where:

SSSSS
is the symbol.

I-6. UNDEFINED LABEL PROCESSING

A symbol is referenced but not defined. The entry in the symbol list has the following format:

SSSSS ???????? RRRRR/TT, ... RRRRR/TT

The defining statement number is replaced by question marks.

I-7. UNUSED LABEL PROCESSING

If a symbol is defined but not referenced by a statement, the entry in the symbol list has the following format:

@SSSSS DDDDD/TT

The symbol is preceded by a "@".

I-8. LITERAL PROCESSING

If a literal of the type =L is referenced by a statement, the characters following the =L are handled as a normal symbol.

If a literal of the type =A, =B, =D, or =F is referenced by a statement, the symbol list has the following format:

LLLLL ..... RRRRR/TT, ... RRRRR/TT

where:

LLLLL
is an exact copy of the literal. DDDDD, the defining statement number, is replaced by dots.

If the literal is seven or more characters long: LLLLL is a maximum length of seven characters, the defining statement number does not have the first dot, and only the first seven characters are used. For example, =B12345 and =B123456 would be considered as the same literal =B12345 and would have the format.

=B12345 ..... RRRRR/TT, ... RRRRR/TT

I-9. OPERATION DIRECTIVE

The operation directive for the cross-reference symbol table is:

*RU,XREF,$^{source}_{namr}$ [$^{list}_{,namr}$ [,A[,B[,C]]]]

where:

source namr
is an FMGR file or logical unit number containing the source code.

If an interactive device is specified, XREF will print a right bracket (]) on the device as a prompt. It will then accept input a line at a time until an END statement is entered.

list namr
is an FMGR file or logical unit number to which XREF will output its listing. If not given, logical unit 6 is used. A minus symbol (−) may also be specified, in which case the same considerations apply to the list file as in the Assembler invocation (see Appendix E).

A
is the bounds specification.

A=0 to specify no bounds.

A=any non-zero character to request the entering of bounds from the system console to allow multiple passes of the cross-reference routine (see section I-10) for further explanation of bounds parameter).

B
is tape number or no tape number specification corresponding to the mode used in the assembly.

B=0 to specify use of tape numbers and a tape length of less than 2038 statements.

Note: A blank card inserted into a card deck before statement 2038 indicates an end-of-tape.

B=N to specify no tape numbers are to be used (sequence numbers can be as large as 32767).

B=−N to cause XREF to number pages consecutively from the last RTE-ASMB page number (−N = −page number).

C
is the number of lines per page.

C=0 to print 55 lines per page.

C=N to print N lines per page.

Note: The cross-reference routine can also be requested to run immediately after an assembly. XREF can be specified via a C parameter in the ASMB control statement. In this case, the following options are assumed: A=0,B=0,C=55.

I-10. BOUNDS

If the symbol table overflows when cross-referencing a program, XREF should be invoked using the bounds parameter. This capability allows the cross-reference table to be generated for only a specified set of symbols at a time. By entering repeated bounds specifications the entire cross-reference table may be listed.

If the bounds parameter is specified, XREF will display the following query on the system console.

/XREF: ENTER LIMITS <LH> or </E>?]

The response to the limits query should be made by specifying either:

LH

or:

/E

where:

L	indicates the low bound
H	indicates the high bound
/E	terminates XREF

If LH is specified, XREF will output to the list namr a cross-reference table for all symbols which are alphabetically between the values of L and H, inclusive. The standard ASCII ordering scheme is used, with blank being the lowest character and left arrow being the highest character. Following its output, XREF will repeat the query.

If /E is entered, XREF will terminate.

I-11. XREF ERROR MESSAGES

Table I-1 contains a list of messages (and their meanings) the user may receive using the ON,XREF directive. The messages are printed as follows:

/XREF: *<message>*

Table I-1. XREF Messages

MESSAGE	MEANING
END OF FILE	End of a user specified source file is reached before an END instruction is found. The XREF routine terminates.
TABLE OVERFLOW	XREF routine does not have enough core space for the table entries. The XREF routine can be made to run with the option for specifying lower and upper bounds and use of multiple passes.
/XREF: $END <*****>	Termination Message. Absolute assembly sources appear as shown. Relocatable sources contain NAM symbols: <NAME>.
ENTER LIMITS OR /E	A request is made to enter XREF bound limits from the system teleprinter.
NO SOURCE	The Logical Source is not defined, or for RTE: A = 0. The XREF routine terminates.
> 16 TAPES!!	More than 16 tapes or zero-length records have been encountered. XREF terminates.

I-12. SAMPLE CROSS-REFERENCE GENERATION

The following pages show a sample Assembler program using cross-reference generation.

```
PAGE 0001 #01                          1:30 PM  TUE.,  6  DEC., 1977

0001                ASMB,R,L,T,Z,C

 # 002
DO  0019  THREE OCT 3
START R 000000
AGAIN R 000002
LOOP  R 000005
NEXT  R 000011
NOUSE R 000012
ADD   R 000013
ADDR  R 000016
TIMES R 000017
THREE B 000000
TWO   B 000001
INIT  R 000020
COUNT R 000026
ONE   R 000027
DNUM  B 000002
NUM   R 000031
HERE  R 000043
**0001 ERRORS PASS#1 **RTE ASMB 92067-16011**
```

```
PAGE 0002 #01                          1:30 PM  TUE.,  6  DEC., 1977

0001                    ASMB,R,L,T,Z,C
0002  00000                    NAM EXAMP     DO-NOTHING USEFUL
0003  00000 000000  START NOP
0004  00001 016020R       JSB INIT
0005  00002 062017R AGAIN LDA TIMES
0006  00003 072026R       STA COUNT     SET COUNTER FOR LOOP.
0007  00004 062043R       LDA =D123     INITIALIZE FIRST VALUE
0008  00005 172016R LOOP  STA ADDR,I    SAVE VALUE
0009  00006 042044R       ADA =B123456 CALCULATE NEXT VALUE
0010  00007 036016R       ISZ ADDR
0011  00010 036026R       ISZ COUNT     BUMP COUNT
0012  00011 026005R NEXT  JMP LOOP      REPEAT UNTIL DONE
0013  00012 062017R NOUSE LDA TIMES
0014                      IFN
0015                 ADD  ADA ONE       ONE IF BY 'N'
0016                      XIF
0017                      IFZ
0018  00013 040001B ADD   ADA TWO       TWO IF BY 'Z'
0019                      XIF
0020  00014 072017R       STA TIMES
0001  00015 026002R       JMP AGAIN     SECOND TAPE
0002  00011                ORG NEXT
0003  00011 026002R       JMP AGAIN
0004  00016                ORR
0005  00016 000000  ADDR  NOP
0006  00017 000000  TIMES NOP
0007  00000                ORB
0008  00000 000003  THREE DEC 3
0009  00001 000002  TWO   DEC 2
0010  00020                ORR
0011  00020 000000  INIT  NOP
0012  00021 060002B       LDA DNUM
0013  00022 072016R       STA ADDR

PG 000
UN  0014         LDA NEG10
0014  00023 062000       LDA NEG10
0015  00024 072017R       STA TIMES
0016  00025 126020R       JMP INIT,I
0017  00026 000000  COUNT NOP
0018  00027 000001  ONE   OCT 1
0019  00030 000003  THREE OCT 3
0020  00002                ORB
0021  00002 000031R DNUM  DEF NUM
0022  00031                ORR
0023  00031 000000  NUM   BSS 10
0024  00043          HERE  EQU *
      00043 000173
      00044 123456
0025                       END START

PG 002
**0002 ERRORS *TOTAL **RTE ASMB 92067-16011**
```

```
PAGE 0003                          EXAMP       DO-NOTHING USEFUL
CROSS-REFERENCE SYMBOL TABLE              1:30 PM  TUE.,  6  DEC., 1977

 **IFN  ******** 00014/01

 **IFZ  ******** 00017/01

 **ORB  ******** 00007/02 00020/02

 **ORG  ******** 00002/02

 **ORR  ******** 00004/02 00010/02 00022/02

 **XIF  ******** 00016/01 00019/01

 =B12345 ....... 00009/01

 =D123  ........ 00007/01

 ADD    ######## 00015/01 00018/01

 ADDR   00005/02 00008/01 00010/01 00013/02

 AGAIN  00005/01 00001/02 00003/02

 COUNT  00017/02 00006/01 00011/01

 DNUM   00021/02 00012/02

@HERE   00024/02

 INIT   00011/02 00004/01 00016/02

 LOOP   00008/01 00012/01

 NEG10  ???????? 00014/02

 NEXT   00012/01 00002/02

@NOUSE  00013/01

 NUM    00023/02 00021/02

 ONE    00018/02 00015/01

 START  00003/01 00025/02

 THREE  ######## 00008/02 00019/02

 TIMES  00006/02 00005/01 00013/01 00020/01 00015/02

 TWO    00009/02 00018/01
```

INDEX

READER COMMENT SHEET

RTE-IV ASSEMBLER
Reference Manual

92067-90003 **October 1980**

Update No. _____
(If Applicable)

We welcome your evaluation of this manual. Your comments and suggestions help us improve our publications. Please use additional pages if necessary.

FROM:

Name _____

Company _____

Address _____

_____

_____

FOLD FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 141 CUPERTINO, CA.

— POSTAGE WILL BE PAID BY —

Hewlett-Packard Company
Data Systems Division
11000 Wolfe Road
Cupertino, California 95014
ATTN: Technical Marketing Dept.

FOLD FOLD

SALES & SUPPORT OFFICES

Arranged alphabetically by country

Product Line Sales/Support Key

Key	Product Line
A	Analytical
CM	Components
C	Computer Systems
CP	Computer Systems Primary Service Responsible Office (SRO)
CS	Computer Systems Secondary SRO
E	Electronic Instruments & Measurement Systems
M	Medical Products
MP	Medical Products Primary SRO
MS	Medical Products Secondary SRO
P	Personal Computing Products
*	Sales only for specific product line
**	Support only for specific product line

IMPORTANT: These symbols designate general product line capability. They do not insure sales or support availability for all products within a line, at all locations. Contact your local sales office for information regarding locations where HP support is available for specific products.

HP distributors are printed in italics.

ANGOLA

Telectra
Empresa Técnica de Equipamentos Eléctricos, S.A.R.L.
R. Barbosa Rodrigues, 41-I DT.
Caixa Postal 6487
LUANDA
Tel: 35515,35516
A,E,M,P*

ARGENTINA

Hewlett-Packard Argentina S.A.
Avenida Santa Fe 2035
Martinez 1640 **BUENOS AIRES**
Tel: 798-5735, 792-1293
Telex: 122443 AR CIGY
Cable: HEWPACKARG
A,E,CP,P

Biotron S.A.C.I.y.M
Avenida Paseo Colon 221
9 Piso
*1399 **BUENOS AIRES***
Tel: 30-4846, 30-1851, 30-8384
Telex: (33)17595 BIONAR
Cable: BIOTRON Argentina
M

Fate S.A. Electronica
Bartolomeu Mitre 833
*1036 **BUENOS AIRES***
Tel: 74-41011, 74-49277, 74-43459
Telex: 18137, 22754
P

AUSTRALIA

Adelaide, South Australia Pty. Ltd.

Hewlett-Packard Australia Pty.Ltd.
153 Greenhill Road
PARKSIDE, S.A. 5063
Tel: 272-5911
Telex: 82536
Cable: HEWPARD Adelaide
A*,CM,CS,E,MS,P

Brisbane, Queensland Office

Hewlett-Packard Australia Pty.Ltd.
5th Floor
Teachers Union Building
495-499 Boundary Street
SPRING HILL, Queensland 4000
Tel: 229-1544
Telex: 42133
Cable: HEWPARD Brisbane
A,CM,CS,E,MS,P

Canberra, Australia Capital Office

Hewlett-Packard Australia Pty.Ltd.
121 Wollongong Street
FYSHWICK, A.C.T. 2069
Tel: 804244
Telex: 62650
Cable: HEWPARD Canberra
A*,CM,CS,E,MS,P

Melbourne, Victoria Office

Hewlett-Packard Australia Pty.Ltd.
31-41 Joseph Street
BLACKBURN, Victoria 3130
Tel: 89-6351
Telex: 31-024
Cable: HEWPARD Melbourne
A,CM,CP,E,MS,P

Perth, Western Australia Office

Hewlett-Packard Australia Pty.Ltd.
141 Stirling Highway
NEDLANDS, W.A. 6009
Tel: 386-5455
Telex: 93859
Cable: HEWPARD Perth
A,CM,CS,E,MS,P

Sydney, New South Wales Office

Hewlett-Packard Australia Pty.Ltd.
17-23 Talavera Road
NORTH RYDE, N.S.W. 2113
P.O. Box 308
Tel: 887-1611
Telex: 21561
Cable: HEWPARD Sydney
A,CM,CP,E,MS,P

AUSTRIA

Hewlett-Packard Ges.m.b.h.
Grottenhofstrasse 94
Verkaufsburo Graz
8052 **GRAZ**
Tel: 21-5-66
Telex: 32375
CM,C*,E*

Hewlett-Packard Ges.m.b.h.
Wehlistrasse 29
P.O. Box 7
A-1205 **VIENNA**
Tel: (222) 35-16-210
Telex: 135823/135066
A,CM,CP,E,MS,P

BAHRAIN

Green Salon
P.O. Box 557
BAHRAIN
Tel: 5503
Telex: 88419
P

Wael Pharmacy
P.O. Box 648
BAHRAIN
Tel: 54886, 56123
Telex: 8550 WAEL GJ
M

BELGIUM

Hewlett-Packard Belgium S.A./N.V.
Blvd de la Woluwe, 100
Woluwedal
B-1200 ***BRUSSELS***
Tel: (02) 762-32-00
Telex: 23-494 paloben bru
A,CM,CP,E,MP,P

BRAZIL

Hewlett-Packard do Brasil I.e.C. Ltda.
Alameda Rio Negro, 750
ALPHAVILLE 06400 Barueri SP
Tel: 421-1311
Telex: 011 23602 HPBR-BR
Cable: HEWPACK Sao Paulo
A,CM,CP,E,MS

Hewlett-Packard do Brasil I.e.C. Ltda.
Avenida Epitacio Pessoa, 4664
22471 **RIO DE JANEIRO**-RJ
Tel: 286-0237
Telex: 021-21905 HPBR-BR
Cable: HEWPACK Rio de Janeiro
A,CM,E,MS,P*

BURUNDI

Typomeca S.P.R.L.
B.P. 553
BUJUMBURA
Tel: 2659
P

CANADA

Alberta

Hewlett-Packard (Canada) Ltd.
210, 7220 Fisher Street S.E.
CALGARY, Alberta T2H 2H8
Tel: (403) 253-2713
Telex: 610-821-6141
A,CM,CP,E*,MS,P*

Hewlett-Packard (Canada) Ltd.
11620A-168th Street
EDMONTON, Alberta T5M 3T9
Tel: (403) 452-3670
Telex: 610-831-2431
A,CM,CP,E,MS,P*

British Columbia

Hewlett-Packard (Canada) Ltd.
10691 Shellbridge Way
RICHMOND, British Columbia V6X 2W7
Tel: (604) 270-2277
Telex: 610-922-5059
A,CM,CP,E*,MS,P*

Manitoba

Hewlett-Packard (Canada) Ltd.
380-550 Century Street
WINNIPEG, Manitoba R3H 0Y1
Tel: (204) 786-6701
A,CM,CS,E,MS,P*

Nova Scotia

Hewlett-Packard (Canada) Ltd.
P.O. Box 931
900 Windmill Road
DARTMOUTH, Nova Scotia B2Y 3Z6
Tel: (902) 469-7820
Telex: 610-271-4482
CM,CP,E*,MS,P*

Ontario

Hewlett-Packard (Canada) Ltd.
552 Newbold Street
LONDON, Ontario N6E 2S5
Tel: (519) 686-9181
Telex: 610-352-1201
A,CM,CS,E*,MS,P*

Hewlett-Packard (Canada) Ltd.
6877 Goreway Drive
MISSISSAUGA, Ontario L4V 1M8
Tel: (416) 678-9430
Telex: 610-492-4246
A,CM,CP,E,MP,P

Hewlett-Packard (Canada) Ltd.
1020 Morrison Drive
OTTAWA, Ontario K2H 8K7
Tel: (613) 820-6483
Telex: 610-563-1636
A,CM,CP,E*,MS,P*

Quebec

Hewlett-Packard (Canada) Ltd.
17500 South Service Road
Trans-Canada Highway
KIRKLAND, Quebec H9J 2M5
Tel: (514) 697-4232
Telex: 610-422-3022
A,CM,CP,E,MP,P*

CHILE

Jorge Calcagni y Cia. Ltda.
Arturo Burhle 065
Casilla 16475
SANTIAGO 9
Tel: 220222
Telex: JCALCAGNI
A,CM,E,M

Olympia (Chile) Ltd.
Rodrico de Araya 1045
Casilla 256-V
SANTIAGO 21
Tel: 25-50-44
Telex: 40-565
C,P

COLOMBIA

Instrumentación
H. A. Langebaek & Kier S.A.
Apartado Aéreo 6287
BOGOTÁ 1, D.E.
Carrera 7 No. 48-75
BOGOTA, 2 D.E.
Tel: 287-8877
Telex: 44400
Cable: AARIS Bogota
A,CM,E,M,P

COSTA RICA

Cientifica Costarricense S.A.
Avenida 2, Calle 5
San Pedro de Montes de Oca
Apartado 10159
SAN JOSÉ
Tel: 24-38-20, 24-08-19
Telex: 2367 GALGUR
Cable: GALGUR
CM,E,M

CYPRUS

Telerexa Ltd.
P.O. Box 4809
14C Stassinos Avenue
NICOSIA
Tel: 45628
Telex: 2894
E,M,P

CZECHOSLOVAKIA

Hewlett-Packard
Obchodni Zastupitelstvi v CSSR
Posl. schranka 27
*CS-118 01 **PRAHA 011***
Tel: 66-296
Telex: 121353 IHC

DENMARK

Hewlett-Packard A/S
Datavej 52
DK-3460 **BIRKEROD**
Tel: (02) 81-66-40
Telex: 37409 hpas dk
A,CM,CP,E,MS,P

Hewlett-Packard A/S
Navervej 1
DK-8600 **SILKEBORG**
Tel: (06) 82-71-66
Telex: 37409 hpas dk
CM,CS,E

ECUADOR

CYEDE Cia. Ltda.
P.O. Box 6423 CCI
Avenida Eloy Alfaro 1749
QUITO
Tel: 450-975, 243-052
Telex: 2548 CYEDE ED
Cable: CYEDE-Quito
A,CM,E,P

Hospitalar S.A.
Casilla 3590
Robles 625
QUITO
Tel: 545-250, 545-122
Cable: HOSPITALAR-Quito
M

EGYPT

Samitro
Sami Amin Trading Office
18 Abdel Aziz Gawish
ABDINE-CAIRO
Tel: 24-932
P

International Engineering Associates
24 Hussein Hegazi Street
Kasr-el-Aini
CAIRO
Tel: 23-829
Telex: 93830
E,M

Informatic For Computer Systems
22 Talaat Harb Street
CAIRO
Tel: 759006
Telex: 93938 FRANK UN
C

EL SALVADOR

IPESA
Boulevard de los Heroes
Edificio Sarah 1148
SAN SALVADOR
Tel: 252787
A,C,CM,E,P

FINLAND

Hewlett-Packard Oy
Revontulentie 7
SF-02100 **ESPOO** 10
Tel: (90) 455-0211
Telex: 121563 hewpa sf
A,CM,CP,E,MS,P

FRANCE

Hewlett-Packard France
Le Ligoures
Bureau de Vente de
Aix-en-Provence
Place Romée de Villeneuve
F-13090 **AIX-EN-PROVENCE**
Tel: (42) 59-41-02
Telex: 410770F
A,CM,CS,E,MS,P*

SALES & SUPPORT OFFICES

Arranged alphabetically by country

FRANCE (Cont.)

Hewlett-Packard France
Boite Postale No. 503
F-25026 **BESANCON**
28 Rue de la Republique
F-25000 **BESANCON**
Tel: (81) 83-16-22
C,M

Hewlett-Packard France
Bureau de Vente de Lyon
Chemin des Mouilles
Boite Postale No. 162
F-69130 **ECULLY** Cédex
Tel: (78) 33-81-25
Telex: 310617F
A,CM,CP,E,MP

Hewlett-Packard France
Immeuble France Evry
Tour Lorraine
Boulevard de France
F-91035 **EVRY** Cédex
Tel: (60) 77-96-60
Telex: 692315F
CM,E

Hewlett-Packard France
5th Avenue Raymond Chanas
F-38320 **EYBENS**
Tel: (76) 25-81-41
Telex: 980124 HP GRENOB EYBE
CM,CS

Hewlett-Packard France
Bâtiment Ampère
Rue de la Commune de Paris
Boite Postale 300
F-93153 **LE BLANC MESNIL**
Tel: (01) 865-44-52
Telex: 211032F
CM,CP,E,MS

Hewlett-Packard France
Le Montesquieu
Avenue du President JF Kennedy
F-33700 **MERIGNAC**
Tel: (56) 34-00-84
Telex: 550105F
CM,CP,E,MS

Hewlett-Packard France
32 Rue Lothaire
F-57000 **METZ**
Tel: (87) 65-53-50
CM,CS

Hewlett-Packard France
F-91947 Les Ulis Cédex**ORSAY**
Tel: (1) 907-78-25
Telex: 600048F
A,CM,CP,E,MP,P

Hewlett-Packard France
Paris Porte-Maillot 13, 15 25
Boulevard De L'Amiral Bruix
F-75782 **PARIS** Cédex 16
Tel: (01) 502-12-20
Telex: 613663F
CM,CP,MS,P

Hewlett-Packard France
2 Allee de la Bourgonette
F-35100 **RENNES**
Tel: (99) 51-42-44
Telex: 740912F
CM,CS,E,MS,P*

Hewlett-Packard France
4 Rue Thomas Mann
Boite Postale 56
F-67200 **STRASBOURG**
Tel: (88) 28-56-46
Telex: 890141F
CM,CS,E,MS,P*

Hewlett-Packard France
20 Chemin de la Cépière
F-31081 **TOULOUSE** Cédex
Tel: (61) 40-11-12
Telex: 531639F
A,CM,CS,E,P*

Hewlett-Packard France
Bureau de Vente de Lille
Immeuble Péricentre
Rue Van Gogh
F-59650 **VILLENEUVE D'ASQ**
Tel: (20) 91-41-25
Telex: 160124F
CM,CS,E,MS,P*

GERMAN FEDERAL REPUBLIC

Hewlett-Packard GmbH
Technisches Büro Berlin
Keithstrasse 2-4
D-1000 **BERLIN** 30
Tel: (030) 24-90-86
Telex: 018 3405 hpbln d
A,CM,CS,E,X,M,P

Hewlett-Packard GmbH
Technisches Büro Böblingen
Herrenberger Strasse 110
D-7030 **BÖBLINGEN**
Tel: (07031) 667-1
Telex: 07265739 bbn or 07265743
A,CM,CP,E,MP,P

Hewlett-Packard GmbH
Technisches Büro Dusseldorf
Emanuel-Leutze-Strasse 1
D-4000 **DUSSELDORF**
Tel: (0211) 5971-1
Telex: 085/86 533 hpdd d
A,CM,CP,E,MS,P

Hewlett-Packard GmbH
Vertriebszentrale Frankfurt
Berner Strasse 117
Postfach 560 140
D-6000 **FRANKFURT** 56
Tel: (0611) 50-04-1
Telex: 04 13249 hpffm d
A,CM,CP,E,MP,P

Hewlett-Packard GmbH
Technisches Büro Hamburg
Kapstadtring 5
D-2000 **HAMBURG** 60
Tel: (040) 63804-1
Telex: 21 63 032 hphh d
A,CM,CP,E,MS,P

Hewlett-Packard GmbH
Technisches Büro Hannover
Am Grossmarkt 6
D-3000 **HANNOVER** 91
Tel: (0511) 46-60-01
Telex: 092 3259
A,CM,CS,E,MS,P

Hewlett-Packard GmbH
Technisches Büro Mannheim
Rosslauer Weg 2-4
D-6800 **MANNHEIM**
Tel: (621) 70050
Telex: 0462105
A,C,E

Hewlett-Packard GmbH
Technisches Büro Neu Ulm
Messerschmittstrasse 7
D-7910 **NEU ULM**
Tel:
Telex:
C,E

Hewlett-Packard GmbH
Technisches Büro Nürnberg
Neumeyerstrasse 90
D-8500 **NÜRNBERG**
Tel: (0911) 56-30-83
Telex: 0623 860
CM,CS,E,MS,P

Hewlett-Packard GmbH
Technisches Büro München
Eschenstrasse 5
D-8021 **TAUFKIRCHEN**
Tel: (089) 6117-1
Telex: 0524985
A,CM,CP,E,MS,P

GREAT BRITAIN

Hewlett-Packard Ltd.
Trafalgar House
Navigation Road
ALTRINCHAM
Chesire WA14 1NU
Tel: (061) 928-6422
Telex: 668068
A,C,E,M

Hewlett-Packard Ltd.
Oakfield House, Oakfield Grove
Clifton
BRISTOL BS8 2BN
Tel: 36806
Telex: 444302
P

Hewlett-Packard Ltd.
14 Wesley Street
CASTLEFORD
Yorkshire WF10 1AE
Tel: (0977) 550016
Telex: 5557355
C

Hewlett-Packard Ltd.
Fourier House
257-263 High Street
LONDON COLNEY
Herts., AL2 1HA
Tel: (0727) 24400
Telex: 1-8952716
C,E

Hewlett-Packard Ltd
Tradax House, St. Mary's Walk
MAIDENHEAD
Berkshire, SL6 1ST
Tel: (0628) 39151
E,P

Hewlett-Packard Ltd.
308/314 Kings Road
READING, Berkshire
Tel: 61022
Telex: 84-80-68
CM,P

Hewlett-Packard Ltd.
Quadrangle
106-118 Station Road
REDHILL, Surrey
Tel: (0737) 68655
Telex: 947234 C,E

Hewlett-Packard Ltd.
Westminster House
190 Stratford Road
SHIRLEY, Solihull
West Midlands B90 3BJ
Tel: (021) 7458800
Telex: 339105
C

Hewlett-Packard Ltd.
King Street Lane
WINNERSH, Wokingham
Berkshire RG11 5AR
Tel: (0734) 784774
Telex: 847178
A,C,E,M

GREECE

Kostas Karaynnis
8 Omirou Street
***ATHENS** 133*
Tel: 32-30-303, 32-37-371
Telex: 21 59 62 RKAR GR
E,M,P

"Plaiso"
G. Gerados
24 Stournara Street
ATHENS
Tel: 36-11-160
Telex: 21 9492
P

GUATEMALA

IPESA
Avenida Reforma 3-48
Zona 9
GUATEMALA CITY
Tel: 316627, 314786, 664715
Telex: 4192 Teletro Gu
A,C,CM,E,M,P

HONG KONG

Hewlett-Packard Hong Kong, Ltd.
G.P.O. Box 795
5th Floor, Sun Hung Kai Centre
30 Harbour Road
HONG KONG
Tel: 5-8323211
Telex: 66678 HEWPA HX
Cable: HP ASIA LTD Hong Kong
E,CP,P

Schmidt & Co. (Hong Kong) Ltd.
Wing On Centre, 28th Floor
Connaught Road, C.
HONG KONG
Tel: 5-455644
Telex: 74766 SCHMX HX
A,M

ICELAND

Elding Trading Company Inc.
Hafnarnvoli-Tryggvagolu
P.O. Box 895
IS-REYKJAVIK
Tel: 1-58-20, 1-63-03
M

INDIA

Blue Star Ltd.
Bhavdeep
Stadium Road
***AHMEDABAD** 380 014*
Tel: 42932
Telex: 012-234
Cable: BLUEFROST
E

Blue Star Ltd.
11 Magarath Road
***BANGALORE** 560 025*
Tel: 55668
Telex: 0845-430
Cable: BLUESTAR
A,CM,C,E

Blue Star Ltd.
Band Box House
Prabhadevi
***BOMBAY** 400 025*
Tel: 422-3101
Telex: 011-3751
Cable: BLUESTAR
A,M

Blue Star Ltd.
Sahas
414/2 Vir Savarkar Marg
Prabhadevi
***BOMBAY** 400 025*
Tel: 422-6155
Telex: 011-4093
Cable: FROSTBLUE
A,CM,C,E,M

Blue Star Ltd.
7 Hare Street
***CALCUTTA** 700 001*
Tel: 12-01-31
Telex: 021-7655
Cable: BLUESTAR
A,M

Blue Star Ltd.
Meenakshi Mandiram
XXXXV/1379-2 M. G. Road
***COCHIN** 682-016*
Tel: 32069
Telex: 085-514
Cable: BLUESTAR
*A**

Blue Star Ltd.
133 Kodambakkam High Road
***MADRAS** 600 034*
Tel: 82057
Telex: 041-379
Cable: BLUESTAR
A,M

Blue Star Ltd.
Bhandari House, 7th/8th Floors
91 Nehru Place
***NEW DELHI** 110 024*
Tel: 682547
Telex: 031-2463
Cable: BLUESTAR
A,CM,C,E,M

Blue Star Ltd.
1-1-117/1 Sarojini Devi Road
***SECUNDERABAD** 500 033*
Tel: 70126
Telex: 0155-459
Cable: BLUEFROST
A,E

Blue Star Ltd.
T.C. 7/603 Poornima
Maruthankuzhi
***TRIVANDRUM** 695 013*
Tel: 65799
Telex: 0884-259
Cable: BLUESTAR
E

INDONESIA

BERCA Indonesia P.T.
P.O.Box 496/Jkt.
Jln. Abdul Muis 62
JAKARTA
Tel: 373009
Telex: 31146 BERSAL IA
Cable: BERSAL JAKARTA
A,C,E,M,P

BERCA Indonesia P.T.
P.O. Box 174/Sby.
J.L. Kulei No. 11
SUBAEE-SURABAYA
Tel: 68172
Telex: 31146 BERSAL SD
Cable: BERSAL-SURABAYA
A,E,M,P*

IRAQ

Hewlett-Packard Trading S.A.
Mansoor City 9B/3/7
BAGHDAD
Tel: 551-49-73
Telex: 2455 HEPAIRAQ IK
CP

IRELAND

Hewlett-Packard Ireland Ltd.
Kestrel House
Clanwilliam Court
Lower Mount Street
DUBLIN 2, Eire
Tel: 680424, 680426
Telex: 30439
A,C,CM,E,M,P

Cardiac Services Ltd.
Kilmore Road
Artane
***DUBLIN** 5, Eire*
Tel: (01) 351820
Telex: 30439
M

SALES & SUPPORT OFFICES

Arranged alphabetically by country

ISRAEL

Electronics Engineering Division
Motorola Israel Ltd.
16 Kremenetski Street
P.O. Box 25D16
TEL-AVIV 67899
Tel: 338973
Telex: 33569 Motil IL
Cable: BASTEL Tel-Aviv
A,CM,C,E,M,P

ITALY

Hewlett-Packard Italiana S.p.A.
Traversa 99C
Giulio Petrone, 19
I-70124 **BARI**
Tel: (080) 41-07-44
M

Hewlett-Packard Italiana S.p.A.
Via Martin Luther King, 38/111
I-40132 **BOLOGNA**
Tel: (051) 402394
Telex: 511630
CM,CS,E,MS

Hewlett-Packard Italiana S.p.A.
Via Principe Nicola 43G/C
I-95126 **CATANIA**
Tel: (095) 37-10-87
Telex: 970291
C,P

Hewlett-Packard Italiana S.p.A.
Via G. Di Vittorio 9
I-20063 **CERNUSCO SUL NAVLIGLIO**
Tel: (2) 903691
Telex: 334632
A,CM,CP,E,MP,P

Hewlett-Packard Italiana S.p.A.
Via Nuova san Rocco A
Capodimonte, 62/A
I-80131 **NAPOLI**
Tel: (081) 7413544
A,CM,CS,E

Hewlett-Packard Italiana S.p.A.
Viale G. Modugno 33
I-16156 **GENOVA PEGLI**
Tel: (010) 68-37-07 E,C

Hewlett-Packard Italiana S.p.A.
Via Turazza 14
I-35100 **PADOVA**
Tel: (49) 664888
Telex: 430315
A,CM,CS,E,MS

Hewlett-Packard Italiana S.p.A.
Viale C. Pavese 340
I-00144 **ROMA**
Tel: (06) 54831
Telex: 610514
A,CM,CS,E,MS,P*

Hewlett-Packard Italiana S.p.A.
Corso Giovanni Lanza 94
I-10133 **TORINO**
Tel: (011) 682245, 659308
Telex: 221079
CM,CS,E

JAPAN

Yokogawa-Hewlett-Packard Ltd.
Inoue Building
1348-3, Asahi-cho
ATSUGI, Kanagawa 243
Tel: (0462) 24-0451
CM,C*,E

Yokogawa-Hewlett-Packard Ltd.
3-30-18 Tsuruya-cho
Kanagawa-ku, Yokohama-Shi
KANAGAWA, 221
Tel: (045) 312-1252
Telex: 382-3204 YHP YOK
CM,CS,E

Yokogawa-Hewlett-Packard Ltd.
Sannomiya-Daiichi Seimei-Bldg. 5F
69 Kyo-Machi Ikuta-Ku
KOBE CITY 650 Japan
Tel: (078) 392-4791
C,E

Yokogawa-Hewlett-Packard Ltd.
Kumagaya Asahi Yasoji Bldg 4F
4-3 Chome Tsukuba
KUMAGAYA, Saitama 360
Tel: (0485) 24-6563
CM,CS,E

Yokogawa-Hewlett-Packard Ltd.
Mito Mitsui Building
4-73, San-no-maru, 1-chome
MITO, Ibaragi 310
Tel: (0292) 25-7470
CM,CS,E

Yokogawa-Hewlett-Packard Ltd.
Sumitomo Seimei Bldg.
11-2 Shimo-sasajima-cho
Nakamura-ku
NAGOYA, Aichi 450
Tel: (052) 581-1850
CM,CS,E,MS

Yokogawa-Hewlett-Packard Ltd.
Chuo Bldg., 4th Floor
5-4-20 Nishinakajima, 5-chome
Yodogawa-ku, Osaka-shi
OSAKA, 532
Tel: (06) 304-6021
Telex: YHPOSA 523-3624
A,CM,CP,E,MP,P*

Yokogawa-Hewlett-Packard Ltd.
29-21 Takaido-Higashi 3-chome
Suginami-ku **TOKYO** 168
Tel: (03) 331-6111
Telex: 232-2024 YHPTOK
A,CM,CP,E,MP,P*

JORDAN

Mouasher Cousins Company
P.O. Box 1387
AMMAN
Tel: 24907, 39907
Telex: 21456 SABCO JO
E,M,P

KOREA

Samsung Electronics
4759 Shinkil, 6 Dong
Youngdeungpo-Ku,
SEOUL
Tel: 8334311, 8334312
Telex: SAMSAN 27364
A,C,E,M,P

KUWAIT

Al-Khalidya Trading & Contracting
P.O. Box 830 Safat
KUWAIT
Tel: 42-4910, 41-1726
Telex: 2481 Areeg kt
A,E,M

Photo & Cine Equipment
P.O. Box 270 Safat
KUWAIT
Tel: 42-2846, 42-3801
Telex: 2247 Matin
P

LUXEMBOURG

Hewlett-Packard Belgium S.A./N.V.
Blvd de la Woluwe, 100
Woluwedal
B-1200 ***BRUSSELS***
Tel: (02) 762-32-00
Telex: 23-494 paloben bru
A,CM,CP,E,MP,P

MALAYSIA

Hewlett-Packard Sales (Malaysia)
Sdn. Bhd.
Suite 2.21/2.22
Bangunan Angkasa Raya
Jalan Ampang
KUALA LUMPUR
Tel: 483544
Telex: MA31011
A,CP,E,M,P*

Protel Engineering
Lot 319, Satok Rd.
P.O. Box 1917
KUCHING, SARAWAK
Tel: 535-44
Telex: MA 70904 Promal
Cable: Proteleng
A,E,M

MEXICO

Hewlett-Packard Mexicana, S.A. de C.V.
Avenida Periferico Sur No. 6501
Tepepan, Xochimilco
MEXICO CITY 23, D.F.
Tel: (905) 676-4600
Telex: 017-74-507
A,CP,E,MS,P

Hewlett-Packard Mexicana, S.A. de C.V.
Rio Volga 600
Colonia del Valle
MONTERREY, N.L.
Tel: 78-42-93, 78-42-40, 78-42-41
Telex: 038-410
CS

MOROCCO

Dolbeau
81 rue Karatchi
CASABLANCA
Tel: 3041-82, 3068-38
Telex: 23051, 22822
E

Gerep
2 rue d'Agadir
Boite Postale 156
CASABLANCA
Tel: 272093, 272095
Telex: 23 739
P

NETHERLANDS

Hewlett-Packard Nederland B.V.
Van Heuven Goedhartlaan 121
NL 1181KK **AMSTELVEEN**
P.O. Box 667
NL 1080 AR **AMSTELVEEN**
Tel: (20) 47-20-21
Telex: 13 216
A,CM,CP,E,MP,P

Hewlett-Packard Nederland B.V.
Bongerd 2
NL 2906VK **CAPPELLE**, A/D Ijessel
P.O. Box 41
NL2900 AA **CAPELLE**, Ijssel
Tel: (10) 51-64-44
Telex: 21261 HEPAC NL
A,CM,CP

NEW ZEALAND

Hewlett-Packard (N.Z.) Ltd.
169 Manukau Road
P.O. Box 26-189
Epsom, **AUCKLAND**
Tel: 68-7159
Cable: HEWPACK Auckland
CM,CS,E,P*

Hewlett-Packard (N.Z.) Ltd.
4-12 Cruickshank Street
P.O. Box 9443
Kilbirnie, **WELLINGTON** 3
Tel: 877-199
Cable: HEWPACK Wellington
CM,CP,E,P

Northrop Instruments & Systems Ltd.
Eden House, 44 Khyber Pass Road
P.O. Box 9682
Newmarket, ***AUCKLAND***
Tel: 794-091
A,M

Northrop Instruments & Systems Ltd.
Terrace House, 4 Oxford Terrace
P.O. Box 8388
CHRISTCHURCH
Tel: 64-165
A,M

Northrop Instruments & Systems Ltd.
Sturdee House
85-87 Ghuznee Street
P.O. Box 2406
WELLINGTON
Tel: 850-091
Telex: NZ 3380
A,M

NIGERIA

The Electronics Instrumentations Ltd.
N6B/S70 Oyo Road
Oluseun House
P.M.B. 5402
IBADAN
Tel: 461577
Telex: 31231 TEIL NG
A,E,M,P

The Electronics Instrumentations Ltd.
144 Agege Motor Road, Mushin
P.O. Box 6645
Mushin, ***LAGOS***
A,E,M,P

NORTHERN IRELAND

Cardiac Services Company
95A Finaghy Road South
BELFAST *BT 10 0BY*
Tel: (0232) 625-566
Telex: 747626
M

NORWAY

Hewlett-Packard Norge A/S
Folke Bernadottesvei 50
P.O. Box 3558
N-5033 **FYLLINGSDALEN** (BERGEN)
Tel: (05) 16-55-40
Telex: 16621 hpnas n
CM,CS,E

Hewlett-Packard Norge A/S
Oesterndalen 18
P.O. Box 34
N-1345 **OESTERAAS**
Tel: (02) 17-11-80
Telex: 16621 hpnas n
A*,CM,CP,E,MS,P

OMAN

Khimji Ramdas
P.O. Box 19
MUSCAT
Tel: 72-22-17, 72-22-25
Telex: 3289 BROKER MB MUSCAT
P

PAKISTAN

Mushko & Company Ltd.
10, Bazar Road
Sector G-6/4
ISLAMABAD
Tel: 28624
Cable: FEMUS Rawalpindi
A,E,M

Mushko & Company Ltd.
Oosman Chambers
Abdullah Haroon Road
KARACHI *0302*
Tel: 511027, 512927
Telex: 2894 MUSHKO PW
Cable: COOPERATOR Karachi
*A,E,M,P**

PANAMA

Electrónico Balboa, S.A.
Apartado 4929
Panama 5
Calle Samuel Lewis
Edificio "Alfa" No. 2
CIUDAD DE PANAMA
Tel: 64-2700
Telex: 3480380
Cable: ELECTRON Panama
A,CM,E,M,P

Foto Internacional, S.A.
P.O. Box 2068
Free Zone of Colon
COLON 3
Tel: 45-2333
Telex: 3485126
Cable: IMPORT COLON/Panama
P

PERU

Cómpania Electro Médica S.A.
Los Flamencos 145, San Isidro
Casilla 1030
LIMA *1*
Tel: 41-4325
Telex: Pub. Booth 25424 SISIDRO
Cable: ELMED Lima
A,CM,E,M,P

PHILIPPINES

The Online Advanced Systems Corporation
Rico House, Amorsolo Cor. Herrera Street
Legaspi Village, Makati
P.O. Box 1510
Metro ***MANILA***
Tel: 85-35-81, 85-34-91, 85-32-21
Telex: 3274 ONLINE
A,C,E,M

Electronic Specialists and Proponents Inc.
690-B Epifanio de los Santos Avenue
Cubao, ***QUEZON CITY***
P.O. Box 2649 Manila
Tel: 98-96-81, 98-96-82, 98-96-83
Telex: 742-40287
P

POLAND

Buro Informasji Technicznej
Hewlett-Packard
Ul Stawki 2, 6P
PL00-950 ***WARSZAWA***
Tel: 39-59-62, 39-67-43
Telex: 812453 hepa pl

SALES & SUPPORT OFFICES

Arranged alphabetically by country

PORTUGAL

Telectra-Empresa Técnica de Equipmentos Eléctricos S.a.r.l.
Rua Rodrigo da Fonseca 103
P.O. Box 2531
***P-LISBON** 1*
Tel: (19) 68-60-72
Telex: 12598
A,C,E,P

Mundinter
Intercambio Mundial de Comércio S.a.r.l
P.O. Box 2761
Avenida Antonio Augusto de Aguiar 138
P-LISBON
Tel: (19) 53-21-31, 53-21-37
Telex: 16691 munter p
M

PUERTO RICO

Hewlett-Packard Puerto Rico
P.O. Box 4407
CAROLINA, Puerto Rico 00630
Calle 272 Edificio 203
Urb. Country Club
RIO PIEDRAS, Puerto Rico 00924
Tel: (809) 762-7255
Telex: 345 0514
A,CP

QATAR

Nasser Trading & Contracting
P.O. Box 1563
DOHA
Tel: 22170
Telex: 4439 NASSER
M

Scitecharabia
P.O. Box 2750
I'DOHA
Tel: 329515
Telex: 4806 CMPARB
P

ROMANIA

Hewlett-Packard Reprezentanta
Boulevard Nicolae Balcescu 16
BUCURESTI
Tel: 130725
Telex: 10440

SAUDI ARABIA

Modern Electronic Establishment
P.O. Box 193
AL-KHOBAR
Tel: 44-678, 44-813
Telex: 670136
Cable: ELECTA AL-KHOBAR
C,E,M,P

Modern Electronic Establishment
P.O. Box 1228, Baghdadiah Street
JEDDAH
Tel: 27-798
Telex: 401035
Cable: ELECTA JEDDAH
C,E,M,P

Modern Electronic Establishment
P.O. Box 2728
RIYADH
Tel: 62-596, 66-232
Telex: 202049
C,E,M,P

SCOTLAND

Hewlett-Packard Ltd.
Royal Bank Buildings
Swan Street
BRECHIN, Angus, Scotland
Tel: 3101, 3102
CM,CS

Hewlett-Packard Ltd.
SOUTH QUEENSFERRY
West Lothian, EH30 9TG
GB-Scotland
Tel: (031) 3311000
Telex: 72682
A,CM,E,M

SINGAPORE

Hewlett-Packard Singapore (Pty.) Ltd.
P.O. Box 58 Alexandra Post Office
SINGAPORE, 9115
6th Floor, Inchcape House
450-452 Alexandra Road
SINGAPORE 0511
Tel: 631788
Telex: HPSGSO RS 34209
Cable: HEWPACK, Singapore
A,CP,E,MS,P

SOUTH AFRICA

Hewlett-Packard South Africa (Pty.) Ltd.
P.O. Box 120
Howard Place
Pine Park Center, Forest Drive, Pinelands
CAPE PROVINCE 7450
Tel: 53-7955, 53-7956, 53-7957
Telex: 57-0006
A,CM,CS,E,MS,P

Hewlett-Packard South Africa (Pty.) Ltd.
P.O. Box 37066
Overport
DURBAN 4067
Tel: 28-4178, 28-4179, 28-4110
CM,CS

Hewlett-Packard South Africa (Pty.) Ltd.
P.O. Box 33345
Glenstantia 0010 **TRANSVAAL**
1st Floor East
Constantia Park Ridge Shopping Centre
Constantia Park
PRETORIA Tel: 98-1126 or 98-1220
Telex: 32163
C,E

Hewlett-Packard South Africa (Pty.) Ltd.
Daphny Street
Private Bag Wendywood
SANDTON 2144
Tel: 802-5111, 802-5125
Telex: 89-84782
Cable: HEWPACK Johannesburg
A,CM,CP,E,MS,P

SPAIN

Hewlett-Packard Española S.A.
c/Entenza, 321
E-BARCELONA 29
Tel: (3) 322-24-51, 321-73-54
Telex: 52603 hpbee
A,CM,CP,E,MS,P

Hewlett-Packard Española S.A.
c/San Vicente S/N
Edificio Albia II,7 B
E-BILBAO 1
Tel: (944) 423-8306, 423-8206
A,CM,E,MS

Hewlett-Packard Española S.A.
Calle Jerez 3
E-MADRID 16
Tel: 458-2600
Telex: 23515 hpe
A,CM,E,MP,P

Hewlett-Packard Española S.A.
Colonia Mirasierra
Edificio Juban
c/o Costa Brava 13, 2.
E-MADRID 34
Tel: 734-8061, 734-1162
CM,CP

Hewlett-Packard Española S.A.
Av Ramón y Cajal 1-9
Edificio Sevilla 1,
E-SEVILLA 5
Tel: 64-44-54, 64-44-58
Telex: 72933
A,CM,CS,MS,P

Hewlett-Packard Española S.A.
C/Ramon Gordillo, 1 (Entlo.3)
E-VALENCIA 10
Tel: 361-1354, 361-1358
CM,CS,P

SWEDEN

Hewlett-Packard Sverige AB
Enighetsvägen 3, Fack
P.O. Box 20502
S-16120 **BROMMA**
Tel: (08) 730-0550
Telex: (854) 10721 MESSAGES
Cable: MEASUREMENTS STOCKHOLM
A,CM,CP,E,MS,P

Hewlett-Packard Sverige AB
Sunnanvagen 14K
S-22226 **LUND**
Tel: (46) 13-69-79
Telex: (854) 10721 (via BROMMA office)
CM,CS

Hewlett-Packard Sverige AB
Vastra Vintergatan 9
S-70344 **OREBRO**
Tel: (19) 10-48-80
Telex: (854) 10721 (via BROMMA office)
CM,CS

Hewlett-Packard Sverige AB
Frötallisgatan 30
S-42132 **VÄSTRA-FRÖLUNDA**
Tel: (031) 49-09-50
Telex: (854) 10721 (via BROMMA office)
CM,CS,E,P

SWITZERLAND

Hewlett-Packard (Schweiz) AG
Clarastrasse 12
CH-4058 **BASLE**
Tel: (61) 33-59-20
A,CM

Hewlett-Packard (Schweiz) AG
47 Avenue Blanc
CH-1202 **GENEVA**
Tel: (022) 32-30-05, 32-48-00
CM,CP

Hewlett-Packard (Schweiz) AG
29 Chemin Château Bloc
CH-1219 **LE LIGNON**-Geneva
Tel: (022) 96-03-22
Telex: 27333 hpag ch
Cable: HEWPACKAG Geneva
A,CM,E,MS,P

Hewlett-Packard (Schweiz) AG
ZUrcherstrasse 20
Allmend 2
CH-8967 **WIDEN**
Tel: (57) 50-111
Telex: 59933 hpag ch
Cable: HPAG CH
A,CM,CP,E,MS,P

SYRIA

General Electronic Inc.
Nuri Basha-Ahnat Ebn Kays Street
P.O. Box 5781
DAMASCUS
Tel: 33-24-87
Telex: 11215 ITIKAL
Cable: ELECTROBOR DAMASCUS
E

Sawah & Co.
Place Azmé
Boite Postale 2308
DAMASCUS
Tel: 16-367, 19-697, 14-268
Telex: 11304 SATACO SY
Cable: SAWAH, DAMASCUS
M

TAIWAN

Hewlett-Packard Far East Ltd.
Kaohsiung Branch
68-2, Chung Cheng 3rd Road
Shin Shin, Chu
KAOHSIUNG
Tel: 24-2318, 26-3253
CS,E,MS,P

Hewlett-Packard Far East Ltd.
Taiwan Branch
5th Floor
205 Tun Hwa North Road
TAIPEI
Tel:(02) 751-0404
Cable:HEWPACK Taipei
A,CP,E,MS,P

Hewlett-Packard Far East Ltd.
Taichung Branch
#33, Cheng Yih Street
10th Floor, Room 5
TAICHUNG
Tel: 289274

Ing Lih Trading Co.
3rd Floor 18, Po-la Road
TAIPEI
Tel:
Telex:
Cable: INGLIH TAIPEI
A

THAILAND

UNIMESA Co. Ltd.
Elcom Research Building
2538 Sukhumvit Ave.
*Bangchak, **BANGKOK***
Tel: 393-2387, 393-0338
Telex: TH81160, 82938, 81038
Cable: UNIMESA Bangkok
A,C,E,M

Bangkok Business Equipment Ltd.
5/5-6 Dejo Road
BANGKOK
Tel: 234-8670, 234-8671, 234-8672
Cable: BUSIQUIPT Bangkok
P

TRINIDAD & TOBAGO

Caribbean Telecoms Ltd.
P.O. Box 732
50/A Jerningham Avenue
PORT-OF-SPAIN
Tel: 624-4213, 624-4214
A,CM,E,M,P

TUNISIA

Tunisie Electronique
31 Avenue de la Liberte
TUNIS
Tel: 280-144
E,P

Corema
1 ter. Av. de Carthage
TUNIS
Tel: 253-821
Telex: 12319 CABAM TN
M

TURKEY

Teknim Company Ltd.
Riza Sah Pehievi
Caddesi No. 7
*Kavaklidere, **ANKARA***
Tel: 275800
Telex: 42155
E

EMA, Muhendislik Kollektif Sirketi
Mediha Eldem
Sokak 41/6
*Yüksel Caddesi, **ANKARA***
Tel: 17-56-22
Cable: Ematrade
M

UNITED ARAB EMIRATES

Emitac Ltd.
P.O. Box 1641
SHARJAH
Tel: 354121, 354123
Telex: 68136
E,M,P,C

UNITED KINGDOM

see: GREAT BRITAIN NORTHERN IRELAND SCOTLAND

UNITED STATES

Alabama

Hewlett-Packard Co.
700 Century Park South
Suite 128
BIRMINGHAM, AL 35226
Tel: (205) 822-6802
CM,CS,MP

Hewlett-Packard Co.
P.O. Box 4207
8290 Whitesburg Drive, S.E.
HUNTSVILLE, AL 35802
Tel: (205) 881-4591
CM,CP,E,M*

Alaska

Hewlett-Packard Co.
1577 "C" Street, Suite 252
ANCHORAGE, AK 99510
Tel: (206) 454-3971
CM,CS**

Arizona

Hewlett-Packard Co.
2336 East Magnolia Street
PHOENIX, AZ 85034
Tel: (602) 273-8000
A,CM,CP,E,MS

Hewlett-Packard Co.
2424 East Aragon Road
TUCSON, AZ 85702
Tel: (602) 889-4631
CM,CS,E,MS**

Arkansas

Hewlett-Packard Co.
P.O. Box 5646
Brady Station
LITTLE ROCK, AR 72215
Tel: (501) 376-1844, (501) 664-8773
CM,MS

UNITED STATES (Cont.)

California

Hewlett-Packard Co.
7621 Canoga Avenue
CANOGA PARK, CA 91304
Tel: (213) 702-8300
A,CM,CP,E,P

Hewlett-Packard Co.
1579 W. Shaw Avenue
FRESNO, CA 93771
Tel: (209) 224-0582
CM,MS

Hewlett-Packard Co.
1430 East Orangethorpe
FULLERTON, CA 92631
Tel: (714) 870-1000
CM,CP,E,MP

Hewlett-Packard Co.
5400 W. Rosecrans Boulevard
LAWNDALE, CA 90260
P.O. Box 92105
LOS ANGELES, CA 90009
Tel: (213) 970-7500
CM,CP,MP

Hewlett-Packard Co.
3939 Lankershim Blvd.
NORTH HOLLYWOOD, CA 91604
Tel: (213) 877-1282
Regional Headquarters

Hewlett-Packard Co.
3200 Hillview Avenue
PALO ALTO, CA 94304
Tel: (415) 857-8000
CM,CP,E

Hewlett-Packard Co.
646 W. North Market Boulevard
SACRAMENTO, CA 95834
Tel: (916) 929-7222
A*,CM,CP,E,MS

Hewlett-Packard Co.
9606 Aero Drive
P.O. Box 23333
SAN DIEGO, CA 92123
Tel: (714) 279-3200
CM,CP,E,MP

Hewlett-Packard Co.
3003 Scott Boulevard
SANTA CLARA, CA 95050
Tel: (408) 988-7000
A,CM,CP,E,MP

Hewlett-Packard Co.
454 Carlton Court
SO. SAN FRANCISCO, CA 94080
Tel: (415) 877-0772
CM,CP

Colorado

Hewlett-Packard Co.
24 Inverness Place, East
ENGLEWOOD, CO 80112
Tel: (303) 771-3455
A,CM,CP,E,MS

Connecticut

Hewlett-Packard Co.
47 Barnes Industrial Road South
P.O. Box 5007
WALLINGFORD, CT 06492
Tel: (203) 265-7801
A,CM,CP,E,MS

Florida

Hewlett-Packard Co.
P.O. Box 24210
2727 N.W. 62nd Street
FORT LAUDERDALE, FL 33309
Tel: (305) 973-2600
CM,CP,E,MP

Hewlett-Packard Co.
4080 Woodcock Drive, #132
Brownett Building
JACKSONVILLE, FL 32207
Tel: (904) 398-0663
CM,C*,E*,MS**

Hewlett-Packard Co.
P.O. Box 13910
6177 Lake Ellenor Drive
ORLANDO, FL 32809
Tel: (305) 859-2900
A,CM,CP,E,MS

Hewlett-Packard Co.
6425 N. Pensacola Blvd.
Suite 4, Building 1
PENSACOLA, FL 32575
Tel: (904) 476-8422
A,CM,MS

Hewlett-Packard Co.
110 South Hoover, Suite 120
Vanguard Bldg.
TAMPA, FL 33609
Tel: (813) 872-0900
A*,CM,CS,E*,M*

Georgia

Hewlett-Packard Co.
P.O. Box 105005
2000 South Park Place
ATLANTA, GA 30339
Tel: (404) 955-1500
Telex: 810-766-4890
A,CM,CP,E,MP

Hewlett-Packard Co.
Executive Park Suite 306
P.O. Box 816
AUGUSTA, GA 30907
Tel: (404) 736-0592
CM,MS

Hewlett-Packard Co.
P.O. Box 2103
1172 N. Davis Drive
WARNER ROBINS, GA 31098
Tel: (912) 922-0449
CM,E

Hawaii

Hewlett-Packard Co.
Kawaiahao Plaza, Suite 190
567 South King Street
HONOLULU, HI 96813
Tel: (808) 526-1555
A,CM,CS,E,MS

Idaho

Hewlett-Packard Co.
11311 Chinden Boulevard
BOISE, ID 83707
Tel: (208) 376-6000
CM,CS,M*

Illinois

Hewlett-Packard Co.
211 Prospect Road
BLOOMINGTON, IL 61701
Tel: (309) 663-0383
CM,CS,MS**

Hewlett-Packard Co.
1100 31st Street
DOWNERS GROVE, IL 60515
Tel: (312) 960-5760
CM,CP

Hewlett-Packard Co.
5201 Tollview Drive
ROLLING MEADOWS, IL 60008
Tel: (312) 255-9800
A,CM,CP,E,MP

Indiana

Hewlett-Packard Co.
P.O. Box 50807
7301 No. Shadeland Avenue
INDIANAPOLIS, IN 46250
Tel: (317) 842-1000
A,CM,CS,E,MS

Iowa

Hewlett-Packard Co.
2415 Heinz Road
IOWA CITY, IA 52240
Tel: (319) 351-1020
CM,CS,E*,MS

Kansas

Hewlett-Packard Co.
1644 S. Rock
WICHITA, KA 67207
Tel: (316) 265-5200
CM,CS

Kentucky

Hewlett-Packard Co.
10170 Linn Station Road
Suite 525
LOUISVILLE, KY 40223
Tel: (502) 426-0100
A,CM,CS,MS

Louisiana

Hewlett-Packard Co.
P.O. Box 1449
3229 Williams Boulevard
KENNER, LA 70062
Tel: (504) 443-6201
A,CM,CS,E,MS

Maryland

Hewlett-Packard Co.
7121 Standard Drive
HANOVER, MD 21076
Tel: (301) 796-7700
A,CM,CP,E,MS

Hewlett-Packard Co.
2 Choke Cherry Road
ROCKVILLE, MD 20850
Tel: (301) 948-6370
Telex: 710-828-9685
A,CM,CP,E,MP

Massachusetts

Hewlett-Packard Co.
32 Hartwell Avenue
LEXINGTON, MA 02173
Tel: (617) 861-8960
A,CM,CP,E,MP

Michigan

Hewlett-Packard Co.
23855 Research Drive
FARMINGTON HILLS, MI 48024
Tel: (313) 476-6400
A,CM,CP,E,MP

Hewlett-Packard Co.
4326 Cascade Road S.E.
GRAND RAPIDS, MI 49506
Tel: (616) 957-1970
CM,CS,MS

Minnesota

Hewlett-Packard Co.
2025 W. Larpenteur Ave.
ST. PAUL, MN 55113
Tel: (612) 644-1100
A,CM,CP,E,MP

Mississippi

Hewlett-Packard Co.
P.O. Box 5028
322 N. Mart Plaza
JACKSON, MS 39216
Tel: (601) 982-9363
CM,MS

Missouri

Hewlett-Packard Co.
11131 Colorado Avenue
KANSAS CITY, MO 64137
Tel: (816) 763-8000
Telex: 910-771-2087
A,CM,CS,E,MS

Hewlett-Packard Co.
1024 Executive Parkway
ST. LOUIS, MO 63141
Tel: (314) 878-0200
A,CM,CP,E,MP

Nebraska

Hewlett-Packard
7101 Mercy Road
Suite 101, IBX Building
OMAHA, NE 68106
Tel: (402) 392-0948
CM,MS

Nevada

Hewlett-Packard Co.
Suite D-130
5030 Paradise Blvd.
LAS VEGAS, NV 89119
Tel: (702) 736-6610
CM,MS**

New Jersey

Hewlett-Packard Co.
Crystal Brook Professional Building
Route 35
EATONTOWN, NJ 07724
Tel: (201) 542-1384
A*,CM,C*,E*,P*

Hewlett-Packard Co.
W120 Century Road
PARAMUS, NJ 07652
Tel: (201) 265-5000
A,CM,CP,E,MP

Hewlett-Packard Co.
60 New England Avenue West
PISCATAWAY, NJ 08854
Tel: (201) 981-1199
A,CM,CP,E

New Mexico

Hewlett-Packard Co.
P.O. Box 11634
11300 Lomas Blvd.,N.E.
ALBUQUERQUE, NM 87123
Tel: (505) 292-1330
Telex: 910-989-1185
CM,CP,E,MS

New York

Hewlett-Packard Co.
5 Computer Drive South
ALBANY, NY 12205
Tel: (518) 458-1550
Telex: 710-444-4691
A,CM,CS,E,MS

Hewlett-Packard Co.
9600 Main Street
CLARENCE, NY 14031
Tel: (716) 759-8621
Telex: 710-523-1893

Hewlett-Packard Co.
200 Cross Keys Office
FAIRPORT, NY 14450
Tel: (716) 223-9950
Telex: 510-253-0092
CM,CP,E,MS

Hewlett-Packard Co.
No. 1 Pennsylvania Plaza
55th Floor
34th Street & 8th Avenue
NEW YORK, NY 10119
Tel: (212) 971-0800
CM,CP,E*,M*

Hewlett-Packard Co.
5858 East Molloy Road
SYRACUSE NY 13211
Tel: (315) 455-2486
A,CM,CS,E,MS

Hewlett-Packard Co.
3 Crossways Park West
WOODBURY, NY 11797
Tel: (516) 921-0300
Telex: 510-221-2183
A,CM,CP,E,MS

North Carolina

Hewlett-Packard Co.
P.O. Box 15579
2905 Guess Road (27705)
DURHAM, NC 27704
Tel: (919) 471-8466
C,M

Hewlett-Packard Co.
5605 Roanne Way
GREENSBORO, NC 27409
Tel: (919) 852-1800
A,CM,CP,E,MS

Ohio

Hewlett-Packard Co.
9920 Carver Road
CINCINNATI, OH 45242
Tel: (513) 891-9870
CM,CP,MS

Hewlett-Packard Co.
16500 Sprague Road
CLEVELAND, OH 44130
Tel: (216) 243-7300
Telex: 810-423-9430
A,CM,CP,E,MS

Hewlett-Packard Co.
962 Crupper Ave.
COLUMBUS, OH 43229
Tel: (614) 436-1041
CM,CP,E*

Hewlett-Packard Co.
330 Progress Rd.
DAYTON, OH 45449
Tel: (513) 859-8202
A,CM,CP,E*,MS

Oklahoma

Hewlett-Packard Co.
P.O. Box 366
1503 W. Gore Blvd., Suite #2
LAWTON, OK 73502
Tel: (405) 248-4248
C

Hewlett-Packard Co.
P.O. Box 32008
304 N. Meridan Avenue, Suite A
OKLAHOMA CITY, OK 73107
Tel: (405) 946-9499
A*,CM,CP,E*,MS

Hewlett-Packard Co.
Suite 121
9920 E. 42nd Street
TULSA, OK 74145
Tel: (918) 665-3300
A**,CM,CS,M*

Oregon

Hewlett-Packard Co.
1500 Valley River Drive, Suite 330
EUGENE, OR 97401
Tel: (503) 683-8075
C

Hewlett-Packard Co.
9255 S. W. Pioneer Court
WILSONVILLE, OR 97070
Tel: (503) 682-8000
A,CM,CP,E*,MS

Pennsylvania

Hewlett-Packard Co.
1021 8th Avenue
King of Prussia Industrial Park
KING OF PRUSSIA, PA 19406
Tel: (215) 265-7000
Telex: 510-660-2670
A,CM,CP,E,MP

Hewlett-Packard Co.
111 Zeta Drive
PITTSBURGH, PA 15238
Tel: (412) 782-0400
A,CM,CP,E,MP

South Carolina

Hewlett-Packard Co.
P.O. Box 6442
6941-0 N. Trenholm Road
COLUMBIA, SC 29260
Tel: (803) 782-6493
CM,CS,E,MS

6

SALES & SUPPORT OFFICES

Arranged alphabetically by country

UNITED STATES (Cont.)

South Carolina (Cont.)
Hewlett-Packard Co.
814 Wade Hampton Blvd.
Suite 10
GREENVILLE, SC 29609
Tel: (803) 232-0917
C

Tennessee
Hewlett-Packard Co.
P.O. Box 22490
224 Peters Road
Suite 102
KNOXVILLE, TN 37922
Tel: (615) 691-2371
A*,CM,MS

Hewlett-Packard Co.
3070 Directors Row
MEMPHIS, TN 38131
Tel: (901) 346-8370
A,CM,CS,MS

Hewlett-Packard Co.
Suite 103
478 Craighead Street
NASHVILLE, TN 37204
Tel: (615) 383-9136
CM,MS**

Texas
Hewlett-Packard Co.
Suite 310W
7800 Shoalcreek Blvd.
AUSTIN, TX 78757
Tel: (512) 459-3143
CM,E

Hewlett-Packard Co.
Suite C-110
4171 North Mesa
EL PASO, TX 79902
Tel: (915) 533-3555
CM,CS,E*,MS**

Hewlett-Packard Co.
5020 Mark IV Parkway
FORT WORTH, TX 76106
Tel: (817) 625-6361
CM,C*

Hewlett-Packard Co.
P.O. Box 42816
10535 Harwin Street
HOUSTON, TX 77036
Tel: (713) 776-6400
A,CM,CP,E,MP

Hewlett-Packard Co.
3309 67th Street
Suite 24
LUBBOCK, TX 79413
Tel: (806) 799-4472
M

Hewlett-Packard Co.
P.O. Box 1270
930 E. Campbell Rd.
RICHARDSON, TX 75081
Tel: (214) 231-6101
A,CM,CP,E,MP

Hewlett-Packard Co.
205 Billy Mitchell Road
SAN ANTONIO, TX 78226
Tel: (512) 434-8241
CM,CS,E,MS

Utah
Hewlett-Packard Co.
3530 W. 2100 South Street
SALT LAKE CITY, UT 84119
Tel: (801) 974-1700
A,CM,CP,E,MS

Virginia
Hewlett-Packard Co.
P.O. Box 9669
2914 Hungary Spring Road
RICHMOND, VA 23228
Tel: (804) 285-3431
A,CM,CP,E,MS

Hewlett-Packard Co.
P.O. Box 4786
3110 Peters Creek Road, N.W.
ROANOKE, VA 24015
Tel: (703) 563-2205
CM,CS,E**

Hewlett-Packard Co.
P.O. Box 12778
5700 Thurston Avenue
Suite 111
VIRGINIA BEACH, VA 23455
Tel: (804) 460-2471
CM,CS,MS

Washington
Hewlett-Packard Co.
15815 S.E. 37th Street
BELLEVUE, WA 98006
Tel: (206) 643-4000
A,CM,CP,E,MP

Hewlett-Packard Co.
Suite A
708 North Argonne Road
SPOKANE, WA 99206
Tel: (509) 922-7000
CM,CS

West Virginia
Hewlett-Packard Co.
4604 MacCorkle Ave., S.E.
CHARLESTON, WV 25304
Tel: (304) 925-0492
A,CM,MS

Wisconsin
Hewlett-Packard Co.
150 S. Sunny Slope Road
BROOKFIELD, WI 53005
Tel: (414) 784-8800
A,CM,CS,E*,MP

URUGUAY

Pablo Ferrando S.A.C. e.I.
Avenida Italia 2877
Casilla de Correo 370
MONTEVIDEO
Tel: 403102
Telex: 901 Public Booth Para Pablo Ferrando 919520
Cable: RADIUM Montevideo
A,CM,E,M

Guillermo Kraft del Uruguay S.A.
Avda. Libertador Brig. Gral.
Lavalleja 2083
MONTEVIDEO
Tel: 234588, 234808, 208830
Telex: 6245 ACTOUR UY
P

U.S.S.R.

Hewlett-Packard Co.
Representative Office
Pokrovsky Blvd. 4/17 KV12
MOSCOW 101000 Tel: 294-2024
Telex: 7825 HEWPACK SU

VENEZUELA

Hewlett-Packard de Venezuela C.A.
Apartado 50933
3A Transversal Los Ruices Norte
Edificio Segre 2Y3
CARACAS 1071
Tel: 239-4133, 239-4777, 239-4244
Telex: 25146 HEWPACK
Cable: HEWPACK Caracas
A,CP,E,MS,P

YUGOSLAVIA

Iskra-Commerce-Representation of Hewlett-Packard
Sava Centar Delegacija 30
Milentija Popovica 9
*11170 **BEOGRAD***
Tel: 638-762
Telex: 12042, 12322 YU SAV CEN

Iskra-Commerce-Representation of Hewlett-Packard
Koprska 46
*61000 **LJUBLJANA***
Tel: 321674, 315879
Telex:

ZAMBIA

R. J. Tilbury (Zambia) Ltd.
P.O. Box 2792
LUSAKA
Tel: 81243
A,E,M,P

ZIMBABWE

Field Technical Sales
45 Kelvin Road, North
P.B. 3458
SALISBURY
Tel:
C,E,M,P

FOR COUNTRIES AND AREAS NOT LISTED:

CANADA

Ontario
Hewlett-Packard (Canada) Ltd.
6877 Goreway Drive
MISSISAUGA, Ontario L4V 1M8
Tel: (416) 678-9430
Telex: 610-492-4246

EASTERN USA

Maryland
Hewlett-Packard Co.
4 Choke Cherry Road
Rockville, MD 20850
Tel: (301) 258-2000

MIDWESTERN USA

Illinois
Hewlett-Packard Co.
5201 Tollview Drive
ROLLING MEADOWS, IL 60008
Tel: (312) 255-9800

SOUTHERN USA

Georgia
Hewlett-Packard Co.
P.O. Box 105005
450 Interstate N. Parkway
ATLANTA, GA 30339
Tel: (404) 955-1500

WESTERN USA

California
Hewlett-Packard Co.
3939 Lankersim Blvd.
LOS ANGELES, CA 91604
Tel: (213) 877-1282

EUROPEAN AREAS NOT LISTED, CONTACT

SWITZERLAND

Hewlett-Packard S.A.
7 Rue du Bois-du-Lan
CH-1217 **MEYRIN 2**, Switzerland
Tel: (022) 83-81-11
Telex: 27835 hpse
Cable: HEWPACKSA Geneve

EAST EUROPEAN AREAS NOT LISTED, CONTACT

AUSTRIA

Hewlett-Packard Ges.m.b.h.
Wehlistrasse 29
P.O. Box 7
A-1205 **VIENNA**
Tel: (222) 35-16-210
Telex: 135823/135066

MEDITERRANEAN AND MIDDLE EAST AREAS NOT LISTED, CONTACT

GREECE

Hewlett-Packard S.A.
Mediterranean & Middle East Operations
35, Kolokotroni Street
Platia Kefallariou
GR-Kifissia, **ATHENS**, Greece
Tel: 808-0359, 808-0429
Telex: 21-6588
Cable: HEWPACKSA Athens

INTERNATIONAL AREAS NOT LISTED, CONTACT

OTHER AREAS

Hewlett-Packard Co.
Intercontinental Headquarters
3495 Deer Creek Road
PALO ALTO, CA 94304
Tel: (415) 857-1501
Telex: 034-8300
Cable: HEWPACK

MANUAL PART NO. 92067-90003
Printed in U.S.A. October 1980

HEWLETT-PACKARD COMPANY
Data Systems Division
11000 Wolfe Road
Cupertino, California 95014