HP 9000 Computer Systems

HEWLETT PACKARD

ASSEMBLY LANGUAGE

Reference Manual

READER COMMENT CARD

HP 9000 Series 800

Assembly Language Reference Manual

92432-90001 November 1988

A reader comment card helps us improve the readability and accuracy of the document. It is also a vehicle for recommending enhancements to the product or manual. Please use it to make improvements.

SERIOUS ERRORS such as technical inaccuracies that may render a program or a hardware device inoperative should be reported to your HP Response Center or directly to a Support Engineer. An engineer will enter the problem on HP's STARS (Software Tracking and Reporting System). This will ensure that critical and serious problems receive appropriate attention as soon as possible.

Editorial suggestions (please give page numbers involved): _____

_____

_____

_____

Recommended improvements (attach additional information if needed): _____

_____

_____

_____

Name: _____ Date: _____

Job Title: _____ Phone: _____

Company: _____

Address: _____

HP 1000 Series _____ (e.g., E-series, A400, A600, etc.)
HP 3000 Series _____ (e.g., 37, 68, 930, etc.)
HP 9000 Series _____ (e.g., 300, 840, etc.)

☐ Check here if you would like a reply.

Hewlett-Packard has the right to use submitted suggestions without obligation, with all such ideas becoming property of Hewlett-Packard.

Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 1070, CUPERTINO, CA 95014

- POSTAGE WILL BE PAID BY ADDRESSEE -

Languages Learning Products Manager
Hewlett-Packard Company
Systems Technology Division
19483 Pruneridge Avenue
Cupertino, California 95014-9786

Fold and Tape

HP 9000 Series 800

ASSEMBLY LANGUAGE

Reference Manual

HEWLETT PACKARD
19483 Pruneridge Ave. Cupertino, CA 95014

Part No. 92432-90001
E1188

Printed in U.S.A. November 1988

NOTICE

The information contained in this document is subject to change without notice.

HEWLETT-PACKARD MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance or use of this material.

Hewlett-Packard assumes no responsibility for the use or reliability of its software on equipment that is not furnished by Hewlett-Packard.

Printing History

New editions are complete revisions of the manual. Update packages, which are issued between editions, contain additional and replacement pages to be merged into the manual by the customer. The dates on the title page change only when a new edition or a new update is published. No information is incorporated into a reprinting unless it appears as a prior update; the edition does not change when an update is incorporated.

The software code printed alongside the date indicates the version level of the software product at the time the manual or update was issued. Many product updates and fixes do not require manual changes and, conversely, manual corrections may be done without accompanying product changes. Therefore, do not expect a one to one correspondence between product updates and manual updates.

First Edition	November 1986	
Update 1	March 1987	
Update 1 Incorporated	May 1987	
Second Edition	January 1988	92453-03A.00.03
Third Edition	November 1988	92453-03A.00.04

Preface

This manual describes the use of the HP Precision Architecture Assembler. Although the manual also summarizes the machine instruction set, you should refer to the *Precision Architecture and Instruction Reference Manual* for a complete description of these instructions. You need to be familiar with the machine instructions to use the Assembler.

Because the machine instruction set and Assembler directives were designed to optimize the new HP Precision Architecture, the resulting *assembly language* is tailored more to the needs of the computer rather than the programmer. Thus, we do not encourage using the Assembler for production programming purposes. For those cases, however, where efficiency or control require programming to be done in assembly language, this manual provides the necessary guidelines.

This manual is organized as follows:

Chapter 1 introduces the Assembler for HP 9000 Series 800 computers.

Chapter 2 explains programming the Assembler for HP-UX.

Chapter 3 describes the HP Precision Architecture Assembler control directives.

Chapter 4 summarizes the mnemonics and instruction format for the HP Precision Architecture machine instructions.

Chapter 5 contains several sample assembly language programs.

Chapter 6 describes the *assembly* (as) command and the ways to invoke the HP Precision Architecture Assembler under the HP-UX operating system.

Appendix A lists the error messages that the HP Precision Architecture Assembler may generate.

Appendix B lists the complete machine instruction set sorted alphabetically by mnemonic name.

This manual assumes that you are an experienced assembly language programmer. In addition, you should have detailed understanding of the HP Precision Architecture and hardware features, and a working knowledge of the HP-UX operating system, program structures, procedure calls, and stack unwind procedures. Consult the following manuals for additional details on specific subjects:

- *HP-UX Reference (09000-90009)* -- for information on HP's implementation of the UNIX* operating system.
- *Precision Architecture and Instruction Reference Manual (09740-90014)* -- for information on architecture and the instruction set.
- *Procedure Calling Conventions Manual (09740-90015)* -- for complete information about the use of the procedure calling convention.

---

*UNIX is a registered trademark of AT&T in the U.S. and other countries.

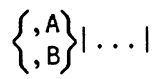
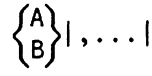
Conventions

NOTATION	DESCRIPTION
UPPERCASE	Within syntax statements, characters in uppercase must be entered in exactly the order shown, though you can enter them in either uppercase or lowercase. For example: SHOWJOB Valid entries are: showjob ShowJob SHOWJOB Invalid entries are: shojwob Shojob SHOW_JOB
italics	Within syntax statements, a word in italics represents a formal parameter or argument that you must replace with an actual value. In the following example, you must replace *filename* with the name of the file you want to release: RELEASE *filename*
punctuation	Within syntax statements, punctuation characters (other than brackets, braces, vertical parallel lines, and ellipses) must be entered exactly as shown.
{ }	Within syntax statements, when several elements within braces are stacked, you must select one. In the following equivalent examples, you must select ON or OFF: SETMSG {ON } {OFF} SETMSG {ON | OFF}
[ ]	Within syntax statements, brackets enclose optional elements. In the following example, brackets around ,TEMP indicate that the parameter and its delimiter are not required: PURGE *filename*[,TEMP] When several elements within brackets are stacked, you can select any one of the elements or none. In the following equivalent examples, you can select *devicename* or *deviceclass* or neither: SHOWDEV [*devicename*] [*deviceclass*] SHOWDEV [*devicename* | *deviceclass*]

Conventions (Continued)

NOTATION	DESCRIPTION

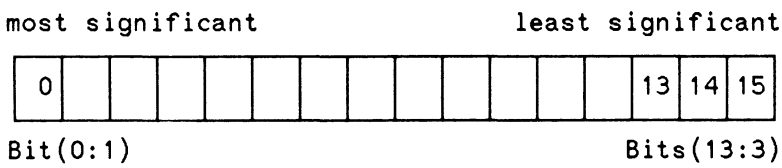
[...]	Within syntax statements, a horizontal ellipsis enclosed in brackets indicates that you can repeatedly select elements that appear within the immediately preceding pair of brackets or braces. In the following example, you can select *itemname* and its delimiter zero or more times, each instance of *itemname* preceded by a comma:
	
	`[,itemname][...]`
	
	If a punctuation character precedes the ellipsis, you must use that character as a delimiter to separate repeated elements. However, if you select only one element, the delimiter is not required. In the following example, the comma does not precede the first instance of *itemname*:
	
	`[itemname][,...]`

|...|	Within syntax statements, a horizontal ellipsis enclosed in parallel vertical lines indicates that you can select more than one element that appears within the immediately preceding pair of brackets or braces. However, each element can be selected only one time. In the following equivalent examples, you must select `,A` or `,B` or `,A,B` or `,B,A` :
	
	`{,A}`
	`{,B}|...|` 
	
	If a punctuation character precedes the ellipsis, you must use that character as a delimiter to separate repeated elements. However, if you select only one element, the delimiter is not required. In the following example, you must select `A` or `B` or `A,B` or `B,A` (the first element is not preceded by a comma):
	
	`{A}`
	`{B}|,...|` 

. . .	Within examples, horizontal or vertical ellipses indicate where portions of the example are omitted.
Δ	Within syntax statements, the space symbol Δ shows a required blank. In the following example, you must separate *modifier* and *variable* with a blank: `SET [(modifier)]Δ(variable);`
KEY	The symbol KEY indicates a key on the terminal's keyboard. For example, RETURN indicates the carriage return key.
CONTROL *char*	CONTROL *char* indicates a control character. For example, CONTROL Y means you must simultaneously press the control key and Y key on the terminal's keyboard.

Conventions (Continued)

NOTATION	DESCRIPTION
base prefixes	The prefixes %, #, and $ specify the numerical base of the value that follows: *%num* specifies an octal number *#num* specifies a decimal number *$num* specifies a hexadecimal number When no base is specified, decimal is assumed.
Bit (*bit:length*)	When a parameter contains more than one piece of data within its bit field, the different data fields are described in the format Bit (*bit:length*), where *bit* is the first bit in the field and *length* is the number of consecutive bits in the field. For example, Bits (13:3) indicates bits 13, 14, and 15:



Contents

Contents (continued)

Contents (continued)

Figures and Tables

TABLES

Figures and Tables (Continued)

FIGURES

Chapter 1
The Assembly Language

This chapter provides an introduction to the assembly language for the HP 9000 Series 800 computers.

The HP 9000 Series 800 Assembly Language represents machine language instructions symbolically, and permits declaration of addresses symbolically as well. The Assembler's function is to translate an assembly language program, stored in a *source file*, into machine language. The result of this translation resides in a *relocatable object file*. The object file is relocatable because it can still be combined with other relocatable object files and libraries. Thus, it is necessary to relocate any addresses that the Assembler chooses for the symbols in the source program. This process of combining object files and libraries is performed by the linker, *ld*. The linker's task is to transform one or more relocatable object files into an executable *program file*. Every program must be linked before it can be executed, even if the source file is complete within itself and does not need to be combined with other files.

Assembler Features

The Assembler provides a number of features to make assembly language programming convenient. These features include:

- **Mnemonic Instructions.** Each machine instruction is represented by a mnemonic operation code, which is easier to remember than the binary machine language operation code. The operation code, together with to output a binary machine instruction to the object file.

- **Symbolic Addresses.** You can select a symbol to refer to the address of a location in virtual memory. The address is often referred to as the *value* of the symbol, which should not be confused with the value of the memory locations at that address. These symbols are called *relocatable symbols* because the actual addresses represented by such symbols are subject to relocation by the linker.

- **Symbolic Constants.** A symbol can also be selected to stand for an integer constant. These symbols are called *absolute symbols* because the values of such symbols are not relocatable.

- **Expressions.** Arithmetic expressions can be formed from symbolic addresses and constants, integer constants, and arithmetic operators. Expressions involving only symbolic and integer constants defined in the current module, or the difference between two relocatable constants, are called *absolute expressions*. They can be used wherever an integer constant can be used. Expressions involving the sum or difference between a symbolic address and an absolute expression are called *relocatable expressions*, or *address expressions*. The constant part of an expression, the part that does not refer to relocatable expressions, may use parenthesized subexpressions to alter operator precedence.

- **Storage Allocation.** In addition to encoding machine language instructions symbolically, storage may be initialized to constant values or simply reserved. Symbolic addresses, or *labels*, can be associated with these memory locations.

- **Symbol Scope.** When two or more object files are to be combined by the linker, certain symbolic addresses can be defined in one module and used in another. Such symbols must be *exported* from the defining module and *imported* into the using module. In the defining module, the symbol has *universal* scope, while in the using module, the symbol is *unsatisfied*. Other symbols declared in the source program that are not exported have *local* scope.

- **Subspaces and Location Counters.** You can organize code and data into separate subspaces, and into separate location counters within each subspace. The programmer can move among the subspaces and location counters, while the Assembler changes the code and data into the correct order.

- **Macro Processing.** A *macro* is a user-defined word which calls a sequence of instructions. Including a macro in a source program causes the sequence of instructions to be inserted into the program wherever the macro appears.

Structure of the Source Program

An assembly language program is a sequence of *statements*. There are three classes of statements:

- Instructions
- Pseudo-operations
- Directives

Instructions represent a single machine instruction in symbolic form. Pseudo-operations cause the Assembler to initialize or reserve one or more words of storage for data, rather than machine instructions. Directives communicate information about the program to the Assembler, but do not generally cause the Assembler to output any machine instructions.

An assembly statement contains four *fields*:

- Label
- Opcode
- Operands
- Comments

Each of these fields is optional, with the exception of the operands field, which cannot appear without an opcode field.

The label field is used to associate a symbolic address with an instruction or data location, or to define a symbolic constant using the .EQU pseudo-operation. This field is optional for all but a few statement types; if present, the label must begin in column one of a source program line. If a label appears on a line by itself, or with a comment only, the label is associated with the next address within the same subspace and location counter.

When the label field begins with the "#" character, it is not treated as a label. If "#" is followed by white space and an integer, the Assembler's line number counter, used when reporting errors, is reset to the value of the integer. Otherwise, the line beginning with "#" is ignored.

The opcode field contains either a mnemonic machine instruction, a pseudo-operation code, or the name of an Assembler directive. It must be separated from the label field by a blank or tab. For certain machine instructions, the opcode field can also contain *completers*, separated from the instruction mnemonic by commas. The completers allowed for each instruction are described in Chapter 4.

The operands field follows the opcode field, separated by a blank or tab. The meaning of the operands depends on the specific statement type, determined by the opcode. Machine instructions require from zero to four operands, which can denote register numbers or memory addresses, depending on the specific instruction.

The comments field is introduced with a semicolon, and causes the Assembler to ignore the remainder of the source line. A comment can also appear on a line by itself.

Figure 1-1 contains several assembly language statements and identifies each of the four fields described above.

```
Label    Opcode    Operands                  Comments

JAN      .EQU      1                         ;declares a symbolic constant
SUM      .WORD     0                         ;reserve a word and set to zero
LOOP     LDW       4(%r1),%r2
         ADD       %r2,%r3,%r4
         STW       %r4,SUM-$global$(%dp)
         BL        LOOP,%r0
```

Figure 1-1. Assembly Language Statements

Statements are normally written on separate lines. It is sometimes useful, especially when using a macro preprocessor, to be able to write several statements on one line. This can be done by separating the statements with the " ! " character. When this feature is used, a label can be placed only on the first statement of the line, and a comment can only follow the last statement on the line. The .LABEL directive can override this condition by providing a means for declaring a label within a multi-statement line. %r is defined in "Registers and Register Mnemonics" later in this chapter.

Symbols and Constants

Both addresses and constants can be represented symbolically. Labels represent a symbolic address except when the label is on an .EQU directive. If the label is on an .EQU directive, the label represents a symbolic constant. These symbols are composed of uppercase and lowercase letters, digits, dollar signs, and underscores. Symbols cannot begin with a digit. The Assembler considers uppercase letters and lowercase letters in symbols to be distinct. The mnemonics for operation codes, directives, and pseudo-operations can be written in either case. There is no explicit limit on the length of a symbol. The following are examples of legal symbols:

```
$START$     _start      PROGRAM
M$3         $global$    $$mulI
main        P_WRITE     loop1
```

The following are examples of illegal symbols:

```
LOOP#1      Contains an illegal character
1st_time    Begins with a digit
```

Integer constants are written in either decimal, octal, or hexadecimal notation. Table 1-1 lists the ranges of these integer constants.

Table 1-1. Integer Constants

	Signed	Unsigned
Decimal	-2147483648 through 2147483647	0 through 4294967295
Octal	020000000000 through 017777777777	0 through 037777777777
Hexadecimal	0x80000000 through 0x7FFFFFFF	0 through 0xFFFFFFFF

The period (.) is a special symbol reserved to denote the current offset of the location counter. It is useful in address expressions to refer to a location relative to the current instruction or data word. This symbol is considered relocatable, and can be used anywhere a relocatable symbol can be used, with the exception of the label field.

NOTE

A symbol whose initial characters are "L$" will not be passed to the linker. Symbols beginning with "L$" may only be used for local code labels and absolute values.

Registers and Register Mnemonics

Series 800 processors have four sets of registers:

- General
- Floating-point
- Space
- Control

General registers are the focus of almost all activity. Data is loaded from memory into general registers and stored into memory from general registers. All arithmetic and logical operations are performed on the contents of the general registers. Each general register is 32 bits wide. There are 32 general registers, denoted %r0 through %r31. General register 0 is special because "writes" into it are ignored, and it always reads as zero. The remaining general registers can be used normally, with the caution that %r1 is the implicit target register for the ADDIL instruction, and %r31 is the implicit link register for the BLE instruction.

The floating-point registers are physically present only on systems with a floating-point coprocessor. On systems without the coprocessor, their presence and behavior is emulated by the HP-UX operating system. There are 16 floating-point registers, each capable of holding either a single or double-precision floating-point number in IEEE format. These registers are denoted %fr0 through %fr15. Registers %fr1, %fr2 and %fr3 are exception registers and are not available to the programmer. Floating-point register 0 contains a permanent floating-point zero when used in an arithmetic operation; when written or read with floating-point loads or stores, the floating-point status register is actually set.

The space registers form the basis of the virtual memory system. Each of the eight space registers can hold a 16 or 32-bit space identifier, depending on the hardware model (the Series 840 uses 16-bit space registers). The space registers are denoted %sr0 through %sr7. Space register 0 is set implicitly by the BLE instruction, and space registers 5 through 7 cannot be modified except by the operating system.

The control registers contain system-state information. There are 25 control registers, denoted %cr0 and %cr8 through %cr31. Of these registers, only %cr11, the shift amount register, and %cr16, the interval timer, are accessible to the user-level programmer.

Register operands are denoted as integer constants since the Assembler can differentiate between general registers, space registers, floating-point registers, and literal values from context. In order to make assembly code more readable, symbolic constants can be defined with the .EQU directive, and used as register operands. In addition, the Assembler has many predefined mnemonic register names that can be used instead of integers. These predefined registers have register types associated with them. The only way to obtain register type checking is to use the .REG directive to assign a predefined register in the operand field to a user-defined name in the label field. This permits type checking on user-defined register names. The following example demonstrates correct usage of the .REG directive:

```
tblptr      .REG      %r20
```

Predefined mnemonic registers are shown in the following tables. All of the mnemonics begin with the % character, so they do not conflict with any programmer-defined symbols.

Table 1-2. General Registers

%r0	%r8	%r16	%r24
%r1	%r9	%r17	%r25
%r2	%r10	%r18	%r26
%r3	%r11	%r19	%r27
%r4	%r12	%r20	%r28
%r5	%r13	%r21	%r29
%r6	%r14	%r22	%r30
%r7	%r15	%r23	%r31

Table 1-3. Floating-Point Registers

%fr0	%fr4	%fr8	%fr12
%fr1	%fr5	%fr9	%fr13
%fr2	%fr6	%fr10	%fr14
%fr3	%fr7	%fr11	%fr15

Table 1-4. Space Registers

%sr0	%sr2	%sr4	%sr6
%sr1	%sr3	%sr5	%sr7

Table 1-5. Control Registers

Registers	Synonyms	Registers	Synonyms
%cr0	%rctr	%cr20	%isr
%cr8	%pidr1	%cr21	%ior
%cr9	%pidr2	%cr22	%ipsw
%cr10	%ccr	%cr23	%eirr
%cr11	%sar	%cr24	%tr0 %ppda
%cr12	%pidr3	%cr25	%tr1 %hta
%cr13	%pidr4	%cr26	%tr2
%cr14	%iva	%cr27	%tr3
%cr15	%eiem	%cr28	%tr4
%cr16	%itmr	%cr29	%tr5
%cr17	%pcsq	%cr30	%tr6
%cr18	%pcoq	%cr31	%tr7
%cr19	%iir		

A few additional predefined register mnemonics are provided in Table 1-5 to match the standard procedure calling convention. This is discussed briefly in Chapter 2. For more detailed information see the *Procedure Calling Conventions Manual.*

Table 1-6. Procedure Calling Convention Registers

Register	Synonyms	Description
%r2	%rp	Return link
%r23	%arg3	Argument word 3
%r24	%arg2	Argument word 2
%r25	%arg1	Argument word 1
%r26	%arg0	Argument word 0
%r27	%dp	Data pointer
%r28	%ret0	Return value
%r29	%ret1 %s1	Return value, static link
%r30	%sp	Stack pointer
%r31	%mrp	Millicode return link
%sr1	%sret %sarg	Return value, argument

Expressions

Arithmetic expressions are often valuable in writing assembly code. The Assembler allows expressions involving integer constants, symbolic constants, and symbolic addresses. These terms can be combined with the standard arithmetic operators shown in Table 1-6.

Table 1-7. Standard Arithmetic Operators

Operator	Operation
+	Integer addition
-	Integer subtraction
*	Integer multiplication
/	Integer division (result is truncated)

The multiplication and division operators have *precedence* over addition and subtraction. That is, multiplications and divisions are performed first from left to right, then additions and subtractions are performed from left to right. Thus, the expression 2+3*4 evaluates to 14.

Expressions produce either an absolute or a relocatable result. Any operation involving only absolute terms yields an absolute result. Relocatable terms are allowed only for the + and - operators. The legal combinations involving relocatable terms are shown in Table 1-7.

Table 1-8. Legal Combinations For Relocatable Terms

Operation	Result
Absolute + Relocatable	Relocatable
Relocatable + Absolute	Relocatable
Relocatable - Absolute	Relocatable
Relocatable - Relocatable (defined locally)	Absolute

NOTE

The combination "relocatable-relocatable+relocatable" is not permitted.

For example, assume the symbols MONTH and YEAR are relocatable, and JANUARY and FEBRUARY are absolute. The expressions MONTH+JANUARY and MONTH+FEBRUARY-4 are relocatable, while the expressions YEAR-MONTH and FEBRUARY-4 are absolute. The expression MONTH+JANUARY*4 is also legal and produces a relocatable result, because JANUARY*4 is evaluated first, producing an absolute intermediate result that is added to MONTH. The expression MONTH+YEAR is illegal, because the sum of two relocatable terms is not permitted.

Because all Series 800 instructions are a single word in length, it is not possible to form a complete 32-bit address in a single instruction. Thus, it is likely that the Assembler (or linker) may not be able to insert the final address of a symbol into the instruction as desired. For example, to load the contents of a word into a register, the following instruction could be used:

```
LDW       START,%r2
```

Because the LDW provides only 14 bits for the address of START, the Assembler or linker prints an error message if the address of START requires more than 14 bits. There are two instructions, LDIL and ADDIL, whose function is to form the leftmost 21 bits of a 32-bit address. The succeeding instruction, by using the target of the LDIL or ADDIL as a base register, needs only 11 bits for the remainder of the address. The Assembler provides special operators, called *field selectors*, that extract the appropriate bits from the result of an expression. With the field selectors L% and R%, the previous example can be recoded as follows:

```
LDIL      L%START,%r1             ;put left part into r1
LDW       R%START(%r1),%r2        ;add r1 and right part
```

The field selectors are always applied to the final result of the expression. They cannot be used in the interior of an expression. The field selectors shown are the two most commonly used. Table 1-8 shows all the available field selectors and their meanings.

Table 1-9. Available Field Selectors

Field Selector	Meaning
F%	Full 32 bits (default)
L%	Right-justified, high-order 21 bits
R%	Low-order 11 bits
LS%	High-order 21 bits after rounding to nearest page
RS%	Low-order 11 bits, sign extended
LD%	Right-justified, high-order 21 bits after rounding to next page
RD%	Low-order 11 bits, with negative sign
LR%	L%value with constant rounded to nearest multiple of 8192
RR%	R%value with constant rounded to nearest multiple of 8192, plus the difference of the constant and the rounded constant.

Since a page is 2048 bytes long, the selectors L%, LS%, and LD% extract a page number, and the corresponding selectors R%, RS%, and RD% extract the offset relative to that page. The distinction is whether the offset is always positive and between 0 and 0x7ff (L%-R%), always negative and between -0x800 and -1 (LD%-RD%), or between -0x400 and 0x3ff (LS%-RS%). The LR% and RR% prefixes are used for accessing different fields of a structure, allowing the sharing of the LR% computation. The distinction is only important when using short addressing near a quadrant boundary, since only the left part is used to select a space register. Chapter 2 explains this further. Each pair is designed to work together just as L% and R% did in the previous example.

The field selectors may also be written F', L', R', LS', RS', LD', RD', LR', and RR'.

Parenthesized Sub-Expressions

The constant term of an expression may contain parenthesized sub-expressions that alter the order of evaluation from the precedence normally associated with arithmetic operators. For example:

```
LABEL1-LABEL2+((6765+(2048-1))/2048)*2048
```

contains a parenthesized sub-expression that rounds a value up to a multiple of 2048.

Absolute symbols may be equated to constant terms containing parenthesized sub-expressions as in the following sequence:

```
BASE    .EQU    0x200
N_EL    .EQU    24
SIZE    .EQU    (BASE+4)*N_EL
```

NOTE

The use of parentheses to group sub-expressions may cause ambiguities in statements where parenthesized register designators are also expected.

Operands and Completers

Machine instructions generally require one, two, or three operands that tell the processor what data to use and where to store the result. Operands can identify a register, a location in memory, or an immediate constant (that is, data that is coded into the instruction itself). The operation code determines how many, and what kinds of operands are required.

The most frequently used machine instructions are those that involve only general registers. Most of these instructions require three register operands that specify two source registers and one target register. Each register operand must be an absolute expression whose value is between zero and 31. Typically, the operand is just an integer constant, which is a symbolic constant that has been equated to a register number, or a register mnemonic.

Register operands may use *typing* by means of a predefined or user-defined register name. Users may define register names with the .REG directive as in the following example:

```
tblptr      .REG      %r20
```

Several instructions also provide access to space and control registers. When a space register is expected, a register mnemonic or absolute expression whose value is between zero and 7 must be given. For a control register, the value must be zero, or between 8 and 31. The following examples show a few machine instructions with register operands:

```
SCRATCH     .EQU      %r18
            ADD       %r3,%r7,%r4          ;r3 + r7 -> r4
            SUB       1,2,3                ;r1 - r2 -> r3
            OR        %r7,%r3,%r8          ;inclusive or of r7,r3 -> r8
            OR        SCRATCH,0,%r7        ;copy r18 to r7 (note: r0 = 0)
            MTCTL     %r2,%sar             ;set shift amount register (cr11)
            MFSP      %sr4,%r10            ;fetch contents of sr4
```

Operands designating memory locations usually consist of an expression and a general register used as a base register. Some instructions also require a space register designation. In general, such operands are written in the form `expr(sr,gr)` or `expr(gr)`, as in the following examples:

```
local_off   .EQU      -64
            LDW       4(%dp),%r2
            STW       %r0,local_off-4(0,%sp)
            LDW       0(%sr3,%r2),%r9
            BLE       $$mulI(%sr7,0)
```

Notice that the space register can be omitted on instructions that require it, as in the first LDW instruction shown in the previous example. If only one register is given, it is assumed to be the general register, and the space register field in the machine instruction is set to zero. Remember that the register mnemonics are equivalent to an integer constant. If the second LDW instruction is written as follows:

```
LDW     0(%r2,%sr3),%r9          ;wrong !
```

an error message is displayed.

The expression in a memory operand is either absolute or relocatable. Absolute expressions are meaningful when the base register contains the address of an array, record, or the stack pointer to which a constant offset is required. Relocatable expressions are meaningful when the base register is r0, or when

the base register contains the left part of a 32-bit address as illustrated in the following example:

```
LDIL        L%glob,r1                   ;set up r1 for STW
STW         %r9,R%glob(%r1)
```

Immediate operands provide data for the machine language instruction directly from the bits of the instruction word itself. A few instructions that use immediate operands are shown below:

```
ADDIL       L%var,%dp
LDIL        L%print,%r1
ADDI        4,%r3,%r5
SUBI        0x1C0,%r14,%ret0
```

Completers are special flags that modify an instruction's behavior. They are written in the opcode field, separated from the instruction mnemonic by a comma. The most common type of completer is a condition test. Many instructions can conditionally trap or nullify the following instruction, depending on the result of their normal operation. For example, notice the completers in the sequence below:

```
ADD,NSV     %r1,%r2,%r3
BL,N        handle_oflo,%r0
OR          %r3,%r4,%r5
```

The ,NSV in the ADD instruction nullifies the BL instruction if no overflow occurs in the addition operation, and execution proceeds with the OR instruction. If overflow does occur, the BL instruction is executed, but the ,N completer on the BL specifies that the OR instruction in its delay slot should not be executed.

Each class of machine instructions defines the set of completers that can be used. These are described with the individual instructions in Chapter 4.

Macro Processing

A macro is a user-defined word that calls a sequence of instructions. Including a macro in a source program causes the sequence of instructions to be inserted into the program wherever the macro appears.

A user may define a word as a macro by using the .MACRO directive.

Detailed information about macro arguments, placement and redefinition of macros, nested macro definitions, and nested macro calls is in Chapter 3 of this manual.

Defining New Instructions With Macros

If you are testing new CPU's or coprocessors, you may need to use opcodes that are unknown to the Assembler. A variant of a macro definition may be used to create a mnemonic for the instruction. After being defined, the new mnemonic instruction can be invoked as easily as a standard instruction.

Opcodes, subopcodes, completers, and operands are encoded into the instruction word in a bit intensive manner because all HP Precision Architecture instructions are one word, or 32-bits, in length. Bit fields do not usually fall on byte or nibble boundaries. A nibble is a half byte.

To write a macro, you must specify explicitly which bit fields are to contain constants and which are to contain macro arguments. The macroprocessor has no built-in knowledge of instruction formats. Defining new instructions through macros is only possible because a convenient way to delimit bit fields has been provided. It is up to the programmer to choose the correct bit field.

Bit positions within the 32-bit word are numbered from zero to 31, from left to right. A bit range is indicated by the starting bit position followed by the ending bit position. The two bit positions should be separated by two periods and enclosed in braces. The bit field beginning at bit position 6 and ending at bit position 10 is represented as:

```
{6..10}
```

If the bit field being assigned from is bigger than the bit field being assigned to, then a warning is issued and the *assigned from* bit field is truncated on the left. When no bit field is specified for the *assigned from* value, low-order bits are used until the value of the *assigned from* bit field becomes the same as the width of the *assigned to* bit field. The *assigned to* bit field must always be specified.

No sign extension is provided by the Macro-assembler when bit fields are generated.

Example

```
PACK    .MACRO              BASE,GREG,SREG,OFFSET
        {0..5}=0x3E{26..31}
        {6..10}=BASE{27..31}
        {11..15}=GREG{27..31}
        {16..17}=SREG{30..31}
        {18..31}=OFFSET{18..31}
       .ENDM
```

The above macro definition defines the macro PACK. The following explanation assumes that PACK is invoked:

```
PACK    %sp,%r19,%sr0,-52
```

Bit field 0 through 5 contains the six low-order bits of the new opcode *0x3E*, or binary 111110, entered as a constant in the macro definition. Bit field 6 through 10 contains general register 30, or binary 11110. These are the five low-order bits of the argument BASE in the macro definition. Bit field 11 through 15 contains general register 19, or binary 10011. These are the five low-order bits of the argument GREG in the macro definition. Bit field 16 through 17 contains space register 0 and represents the five low-order bits of the argument SREG in the macro definition. Bit field 18 through 31 contains binary 11111111001100, the OFFSET -52 which was entered as an argument to the macro definition.

Chapter 2
Programming for HP-UX

The Assembler is a flexible tool for writing programs, but every operating system imposes certain conventions and restrictions on the programs that are intended to run on that system. This chapter discusses the conventions that must be understood in order to write assembly language programs and procedures for the HP 9000 Series 800 HP-UX operating system. Several Assembler directives are mentioned in this chapter to place them in a meaningful context. A full discussion of these directives can be found in following chapters.

Spaces

Virtual addressing on the HP Precision Architecture is based on *spaces*. A virtual address is composed of a space identifier, which is either 16 or 32 bits long (depending on the hardware model), and a 32-bit offset within the space. Thus, each space can contain up to 4 gigabytes, and there is a large supply of spaces.

Every program on an HP-UX system is assigned two spaces when it is loaded for execution by the operating system: one for code, and one for data. The HP-UX operating system makes the code space *read only*, so that it can be shared whenever several processes are executing the same program. The data space is writable by the new process, and is private to that process; that is, every process has a unique data space.

The actual space identifiers assigned to these two spaces can vary from one execution of the program to the next; these numbers cannot be determined at compile time or link time. Generally, programmers do not need to be concerned with the space identifiers, since the operating system places them in two reserved space registers, where they remain for the duration of program execution. The identifier of the code space is placed in space register 4 (`sr4`) and the identifier of the data space is placed in `sr5`.

When writing an assembly language program, declare a space named `$TEXT$` for executable code, and a space named `$PRIVATE$` for modifiable data. Constant data, literals that you do not plan to modify during program execution, can be placed in either space. Placing constant data in the `$TEXT$` space decreases the size of the non-sharable part of your program and improves the overall efficiency of the operating system.

The particular space registers mentioned above play an important role in virtual addressing. While many of the branching instructions, such as `BL`, `BLR`, and `BV`, are capable of branching only within the currently executing code space (called PC-*space*), two of the branching instructions, `BE` and `BLE`, require that you specify a space register as well as an offset. These instructions allow you to branch to code executing in a different space. On HP-UX systems, all code for a program is contained in one space, so all `BE` and `BLE` instructions should be coded to use `sr4`.

In contrast, the memory reference instructions, such as LDW and STW, allow a choice between two forms of addressing: long and short. With long addressing, you can choose any of the space registers 1 through 3 for the space identifier part of the virtual address. The space offset is formed as the sum of an immediate displacement and the contents of a general register. With short addressing, one of the space registers

between 4 through 7 is chosen automatically, based on the high-order two bits of the general register. Each space addressed by these four space registers is effectively divided into four *quadrants*, with a different quadrant of each space accessible via short addressing.

On HP-UX systems, all of a program's code is placed in the first quadrant of the $TEXT$ space (space offsets from 0 through 0x3FFFFFFF). The data is placed in the second quadrant of the $PRIVATE$ space (space offsets from 0x40000000 through 0x7FFFFFFF). Thus, literal data in the code space and modifiable data in the data space can be addressed using the short addressing technique, without any concern for the space registers.

You can define spaces other than $TEXT$ and $PRIVATE$ in a program file by declaring a special kind of space called an *unloadable space*. Unloadable spaces are treated as normal spaces by the linker, but as the name implies, are not actually loaded when a program is executed. Unloadable spaces are typically used by compilers to store extra information within a program file. The most common example of an unloadable space is $DEBUG$, which is used to hold symbolic debugging information.

The *sort key* attribute allows the programmer to control the placement of a space relative to the other spaces. The linker places spaces with lower sort keys in front of spaces with higher sort keys.

The .SPACE directive is used to declare spaces. The assembly language programmer is not required to fill one space before beginning another. When a space is first declared, the Assembler begins filling that space. The .SPACE directive can also be used to return to a previously declared space, and the Assembler continues to fill it as if there had been no intervening spaces.

Subspaces

While a space is a fundamental concept of the architecture, a subspace is just a logical subdivision of a space. The Assembler places the program's code and data into subspaces rather than spaces. Each subspace *belongs* to the space that was current when the subspace was first declared. The linker groups subspaces into spaces as it builds an executable program file. For more details see the *ld* (UTIL) entry in the *HP-UX Reference* manual. When the linker combines several relocatable files, it groups the subspaces from each file by name, so that all subspaces with the same name are placed contiguously in the program.

Subspaces have several attributes. The *alignment* attribute specifies what memory alignment (in bytes) is required in the virtual address space. The alignment can be any power of two, from 1 through 2048, inclusive. Typically, the alignment is 4 or 8 to specify that the beginning of the subspace must be word or double-word aligned. Normally, the alignment attribute is computed automatically by the Assembler from the largest .ALIGN directive used within the subspace.

The *quadrant* attribute assigns the subspace to one of the four quadrants of its space. On HP-UX systems, all subspaces in the code space must be in quadrant 0, and all subspaces in the data space must be in quadrant 1.

The *access rights* attribute specifies the access rights that should be given to each physical page in the subspace. On HP-UX systems, all subspaces in the code space must have access rights of 0x2C (code page executable at any privilege level). All subspaces in the data space must have access rights of 0x1F (data page readable and writable at all privilege levels).

The *sort key* attribute allows the programmer to control the placement of a subspace relative to the other subspaces in its space. The linker places subspaces with lower sort keys in front of subspaces with higher sort keys.

The .SUBSPA directive is used to declare a subspace and its attributes. As with spaces, the assembly language programmer can switch from one subspace to another, and the Assembler will fill each subspace independently as if the source code had been presented one complete subspace at a time. When the .SPACE directive is used to switch spaces, the Assembler remembers the current subspace in each space.

Several additional Assembler directives are provided as shorthand to declare and switch to some standard spaces and subspaces. For example, the .CODE directive switches to the $TEXT$ space and the $CODE$ subspace, and the .DATA directive switches to the $PRIVATE$ space and the $DATA$ subspace.

You can declare as many subspaces as you can use, but the sort key attribute should be used carefully, because the stack unwind mechanism reserves a range of sort keys (56 through 88) for use with the $CODE$ subspace. Some of the standard subspaces and sort keys used by the compilers are shown in Table 2-1. Directives that generate commonly used spaces and subspaces are found in Table 3-4.

Table 2-1. Standard Subspaces and Sort Keys

Space	Subspace	Sort Key	Use
$TEXT$		8	
	$MILLICODE$	8	Millicode library routines
	LIT	16	Literals
	$CODE$	24	Normal code
	$UNWIND_START$	56	Stack unwind
	$UNWIND$MILLICODE$	62	Stack unwind
	$UNWIND$	64	Stack unwind
	$UNWIND_END$	72	Stack unwind
$PRIVATE$		16	
	$GLOBAL$	8	Pascal global variables
	$DATA$	24	Normal global and static data
	$COMMON$	24	FORTRAN BLOCK DATA
	BSS	80	Uninitialized data and common

NOTE

By linker convention, programs should avoid using sort keys less than 8 in either space.

Location Counters

Just as spaces can be divided into subspaces, subspaces can be further divided by using location counters. You can use up to four location counters in each subspace, and the Assembler fills a separate area for each location counter. When the assembly is complete, the subspace is formed by concatenating each of these areas. All references relative to a location counter are relocated so that they are relative to the complete subspace.

Unlike subspaces, however, the use of location counters is completely local to the Assembler. Once the subspace is formed at the end of the assembly, the distinction among the individual areas built by location counters disappears. No further reordering or grouping related to location counters is performed by the linker.

This facility allows you to assemble related data into disjoint pieces of a subspace while keeping the source code in a convenient order.

The `.LOCCT` directive is used to switch from one location counter to another. The Assembler automatically remembers the previous value of each location counter within each subspace. When the `.SUBSPA` directive is used to switch subspaces, the Assembler automatically begins using the location counter that was last in effect in the new subspace.

Compiler Conventions

In order to write assembly language procedures that can both call to and be called from high-level language procedures, it is necessary to understand the standard procedure calling convention and other compiler conventions.

On many computer systems, each high-level language has its own calling convention. Consequently, calls from one language to another are sometimes difficult to arrange, except through assembly code. The architecture generally prescribes very few operations that must be done to effect a procedure call, and there is often a pair of machine-language instructions to call a procedure and return from one. The HP Precision Architecture provides no special procedure call or return instructions. There is, however, a standard procedure calling convention for all high-level languages as well as the Assembler. It is tuned for the architecture, and is designed to make a procedure call with as few instructions as possible.

Besides defining a uniform call and return sequence for all languages, the calling convention is important for other reasons as well. In order to streamline the calling sequence, the return link is not saved on the stack unless necessary and the previous stack pointer is rarely saved on the stack. Thus, it is not usually possible to obtain a stack trace at an arbitrary point in the program without some additional static information about each procedure's stack frame size and usage. For example, you could not obtain a stack trace while debugging or analyzing a core dump, or using the try-recover feature in Pascal. Obtaining a stack trace is made possible by the *stack unwind* mechanism. It uses special *unwind descriptors* that contain the exact static information needed for each procedure. These descriptors are generated automatically by all high-level compilers as well as the Assembler. Each descriptor contains the starting and ending address of a procedure's object code, plus that procedure's stack frame size, and a few flags indicating, among other things, whether the return link is saved on the stack. Given the current program counter and stack pointer, the stack unwind mechanism can determine the calling procedure by finding the return link either in a register or on the stack. Also, it can determine the previous stack pointer by subtracting the current procedure's stack frame size.

The Assembler requires that you follow programming conventions to generate unwind descriptors. The beginning and end of each procedure must be noted with the `.PROC` and `.PROCEND` directives. The `.CALLINFO` directive supplies additional information about the procedure, including the stack frame size. With this information, the Assembler creates the unwind descriptor. It can also generate the standard entry and exit code to create and destroy the stack frame, save and restore the return link (if necessary), and save and restore any necessary registers. These code sequences are generated at the points indicated by the `.ENTER` and `.LEAVE` directives.

Arguments to procedures are loaded into general registers 26, 25, 24, and 23; these registers are named, respectively, `%arg0`, `%arg1`, `%arg2`, and `%arg3`. If more than four words of arguments are required, the remaining arguments are stored in the caller's stack frame in the variable argument list. The return value should be returned in general register 28, called `%ret0`. General register 29, called `%ret1`, is used for the low-order bits of a double-word return value, while `%ret0` contains the high order bits. In addition to the argument and return registers, the procedure can use registers 19 through 22 and registers 1 and 31 as scratch registers. Any other registers must be saved before use at entry and restored before exit.

Chapter 3 contains detailed descriptions of the Assembler directives described above. For a more thorough discussion of the procedure calling conventions, refer to the *Procedure Calling Conventions Manual.*

In order for an assembly language procedure to be callable from another language or another assembly language module, the name of the procedure must be *exported.* The `.EXPORT` directive does this. It also allows you to declare the *symbol type.* For procedure entry points, the symbol type should be `ENTRY`.

The Assembler and linker treat all symbols as case sensitive, while some compilers do not. By convention, compilers that are case insensitive uniformly convert all exported names to lower case. For example, it is possible to declare a procedure that cannot conflict with Pascal procedure names by using upper case letters. However, there is an *aliasing* mechanism in some compilers that allows you to declare a case-sensitive name for external use. See the appropriate language reference manual for more information.

Conversely, the .IMPORT directive allows you to reference a procedure name that is exported from another module, either from the Assembler or the compiler. Once a procedure name has been imported, it can be referenced exactly as if it were declared in the same module.

Data symbols can be exported and imported just like procedure names. However, not all compilers export the names of global variables, or provide a mechanism to reference data symbols exported from an assembly language module. For example, the HP Pascal/HP-UX compiler does not normally do this, while the C/HP-UX compiler does. FORTRAN 77/HP-UX named common blocks are exported, but the names of the variables within the common blocks are not.

It was mentioned before that data is allocated beginning from a virtual space offset 0x40000000. For convenience as well as compatibility with future releases of HP-UX systems, all data in the $PRIVATE$ space must be accessed relative to general register 27, called %dp. Standard run-time startup code, from the file /lib/crt0.o, must be linked with every program. This startup code declares a global symbol called $global$ in the $GLOBAL$ subspace. This code also loads the address of this symbol into the %dp register before beginning program execution. This register must not be changed during the execution of a program. Since the %dp register is known to contain the address of $global$, the following single instruction does the load as long as the displacement from $global$ to the desired location is less than 8 kilobytes:

```
LDW     var-$global$(%dp),%r3
```

If the desired location is not known to be close enough to $global$, the following sequence must be used:

```
ADDIL   L%var-$global$,%dp          ;result in r1
LDW     R%var-$global$(%r1),%r3
```

To access items in the $PRIVATE$ space (global data), the following does not work:

```
LDIL    L%var,%r1                   ;wrong
LDW     R%var(%r1),%r3              ;wrong
```

This assumes that the operating system always allocates data at the same virtual space offset 0x40000000.

Uninitialized areas in the data space can be requested with the .COMM (common) request. By convention, these requests should always be made in the BSS subspace in the $PRIVATE$ space. The BSS should not be used for anything else. Common requests are passed on to the linker, which matches up all requests with the same name and allocates a block of storage equal in size to the largest request. If, however, an exported data symbol is found with the same name, the linker treats the common requests as if they were imports. FORTRAN 77/HP-UX common blocks are naturally allocated in this way: if a BLOCK DATA subprogram initializes the common block, all common requests are linked to that initialized block. Otherwise, the linker allocates enough storage in BSS for the common block. The C/HP-UX compiler also allocates uninitialized global variables this way. In C, however, each uninitialized global is a separate common request.

System Calls

The HP-UX operating system defines a large set of system calls. Refer to the *HP-UX Reference* manual for more information. These system calls can be made indirectly by calling the interface routines in the C run-time library, or they can be made directly from assembly code. All system calls are funneled through a single entry point in the system space, which is identified by space register 7. Each system call is assigned a unique number, which must be loaded into general register 22. The arguments to the system call should be loaded into general registers 26, 25, 24, and 23, as necessary. When the system call returns, a status code is returned in register 22. If the status code is zero, the system call succeeded and the return value, if any, is in register 28. If the status code is nonzero, the system call failed and the error number is found in register 28. A list of the system call numbers as well as the location of the system call entry point is in the standard include file `/usr/include/sys/syscall.h`.

The following example of a code fragment shows a call to the *read* system call:

```
OR      %r0,%r0,%arg0                   ;file descriptor = 0
ADDIL   L%buf-$global$,%dp
LDO     R%buf-$global$(%r1),%arg1       ;buffer address
LDO     10,%arg2                        ;length = 10
LDIL    L%0xC0000004,%r1
BLE     R%0xC0000004(%sr7,%r1)          ;system call entry point
LDO     3,%r22                          ;read system call = 3
```

In the above code, the last instruction loads the constant 3 into register 22, and executes in the delay slot of the BLE instruction.

Assembly Listing

The Assembler's command line option, -l, causes an assembly listing to the standard output. For each line of source code, the listing provides the line number, the subspace offset, the hexadecimal representation of the assembled code (possibly flagged with an asterisk (*) to indicate address relocation), the source text, and any comments.

Following is a line of assembly language as it appears in the source file:

```
        SAVE        LDO        VAL(%r0),%r20        ;retain value
```

The above line would appear in the assembly listing as follows:

```
line no.   offset     hex representation  label opcode  operands        comment

16         0000004c   (341400A)           SAVE  LDO     VAL(%r0),%r20   ;retain value
```

The choice of line number 16 is arbitrary here. At the end of the assembly listing, a symbol table is printed showing the name and value of each symbol in the file. A type field for each symbol, indicating either absolute or relocatable, is included.

Certain types of source lines generate multiple instructions. Macro calls usually expand to several instructions. The .ENTER and .LEAVE directives each generate more than one HP Precision Architecture instruction. The predefined subspace directives, such as .CODE and .DATA, result in a space and a subspace declaration. Procedures in the $CODE$ subspace generate stack unwind descriptors in the $UNWIND$ subspace.

You have the choice of listing a section of assembled code in either the compressed or expanded form. The placement of the .LISTON and .LISTOFF directives determines which code will be expanded during listing. The directive .LISTON tells the Assembler to expand the listing of all subsequent source lines until a .LISTOFF directive is encountered. .LISTOFF stays in effect until the occurrence of a .LISTON directive. The default is .LISTOFF.

The directives .LISTON and .LISTOFF may be placed anywhere in the source text and always go into effect immediately. .LISTON and .LISTOFF may be used as often as desired.

Chapter 3
Assembler Directives and Pseudo-Operations

A set of Assembler directives allow you to take special programming actions during the assembly process. These Assembler directives begin with a period (.) to distinguish them from machine instruction opcodes or extended opcodes.

Table 3-1 lists the Assembler directives described in this chapter. These directives include those that establish the procedure calling convention, declare common, and define spaces and subspaces. Table 3-2 lists those directives that are compiler generated and, therefore, are not used by assembly language programmers. Table 3-3 lists the pseudo-operations that reserve and initialize data areas and generate entry and exit code.

This chapter also includes Table 3-4 under "Programming Aids" which lists the predefined directives for establishing standard spaces and subspaces.

Table 3-1. Assembler Directives

Directive	Function
.ALIGN	Forces location counter to the next larger multiple of the supplied alignment value.
.CALL	Specifies that the next statement is a procedure call.
.CALLINFO	Provides information for generating Entry/Exit code sequences and for creating stack unwind descriptors.
.COMM	Requests common storage for a specified number of bytes.
.COPYRIGHT	Inserts the specified string into the object module as a copyright notice.
.DOUBLE	Initializes a double-word to a floating-point value.
.END	Terminates an Assembly language program.
.ENDM	Marks the end cf a macro definition.
.ENTER	Marks a procedure's entry point and generates standard entry code.

(Continued on next page)

Table 3-1. Assembler Directives (Continued)

Directive	Function
.EQU	Assigns an expression to an identifier.
.EXPORT	Makes a specified symbol available to other modules.
.FLOAT	Initializes a double-word of storage to a floating-point value.
.IMPORT	Specifies that the definition of the given symbol occurs in another module.
.LABEL	Permits a label definition to appear within a sequence of directives that occur on a single line.
.LEAVE	Marks a procedure's exit point and generates standard exit code.
.LISTOFF	Controls listing of expanded Assembler instructions.
.LISTON	Controls listing of expanded Assembler instructions.
.LOCCT	Selects a location counter.
.MACRO	Marks the beginning of macro definitions.
.ORIGIN	Advances the location counter to a relative location from the beginning of the current subspace.
.PROC	Marks the first statement in a procedure.
.PROCEND	Marks the last statement in a procedure.
.REG	Attaches a type and number to a user-defined register name.
.SPACE	Declares a new space or switches back to a previous space.
.SPNUM	Reserves and initializes a word of storage.
.SUBSPA	Declares a new subspace or switches back to a previous subspace.
.VERSION	Inserts the specified string into the current object module as a user-defined version identification string.

Table 3-2. Compiler Generated Directives

Directive	Function
.ENTRY	Marks the entry point of the current procedure.
.EXIT	Marks the return point of the current procedure.

Table 3-3. Pseudo-Operations

Directive	Function
.BLOCK and .BLOCKZ	Reserves a block of data storage.
.BYTE	Reserves 8 bits (byte) of storage and initializes it to the given value.
.HALF	Reserves 16 bits (half word) of storage and initializes it to the given value.
.STRING and .STRINGZ	Reserves the appropriate amount of storage and initializes it to the given string.
.WORD	Reserves 32 bits (a word) of storage and initializes it to the given value.

Assembler Directives

The remainder of this chapter lists the Assembler directives and pseudo-operations in alphabetical order.

NOTE

The similar pseudo-operations, .BYTE, .HALF, and .WORD, are grouped together under "Byte". The .EXPORT and .IMPORT directives are also treated as one. Several of the descriptions include sample assembly code sequences. You can enter these short code sequences, assemble them using the -l option then inspect the offsets and field values to see how that particular directive controls the assembly environment.

The .ALIGN Directive

The .ALIGN directive advances the current location counter to the next specified "boundary."

Syntax

```
.ALIGN  [boundary]
```

Parameters

boundary — An integer value for the byte boundary to which you want to advance the location counter. The Assembler advances the location counter to that boundary. Permissible values must be a power of 2 and can range from one to 2048. The default value is 8 (doubleword aligned).

Example

```
                .CODE
                ADDIL L'$WORDMARK$-$global$,%dp
                B page
                NOP
                .ALIGN 2048
page
                ADDI    1,%r1,%r1
                .DATA
$WORDMARK$
                .WORD   0x0FFF
                .IMPORT $global$,DATA
```

This sample program adds a 21 bit field to the data pointer. Then a branch is taken to the label page that has been page aligned.

The .BLOCK and .BLOCKZ Pseudo-Operations

The .BLOCK and .BLOCKZ pseudo-operations reserve a block of storage.

Syntax

```
.BLOCK[Z]   [num_bytes]
```

Parameters

num_bytes — An integer value for the number of bytes you want to reserve. Permissible values range from zero to 0x7FFFFFFF, although the Assembler uses a default value of zero if you omit specifying a parameter.

Discussion

The .BLOCK pseudo-operation reserves a data storage area but does not perform any initialization. The .BLOCKZ pseudo-operation reserves a block of storage and initializes it to zero.

When you label a .BLOCK pseudo-operation, the label refers to the first byte of the storage area.

NOTE

Under the present implementation of the Assembler, the .BLOCK pseudo-operation also initializes the reserved area to zero.

Example

```
            .SPACE $TEXT$
            .SUBSPA $CODE$
            .BLOCK 64
swap        LDW 0(2),1
            STW 1,4(2)
            .END


            .DATA
word0       .BLOCK      0X20
word8       .WORD       0XFFFF
```

The first example requests the Assembler to reserve 64 bytes of memory in the $CODE$ subspace. This area is then followed by a "Load Word" and "Store Word" instruction. The second example reserves 32 bytes of memory in the $DATA$ subspace followed by one word intended as an end marker.

The .BYTE, .HALF, and .WORD Pseudo-Operations

The .BYTE, .HALF, and .WORD pseudo-operations reserve storage and initialize it to the given value.

Syntax

```
{ .BYTE }
{ .HALF } [init_value] [,init_value]...
{ .WORD }
```

Parameters

init_value	Either a decimal, octal, or hexadecimal number or a sequence of ASCII characters, surrounded by quotation marks. If you omit the initializing value, the Assembler initializes the area to zero.

Discussion

The .BYTE pseudo-operation requests 8 bits of storage; the .HALF pseudo-operation requests 16 bits of storage; and the .WORD pseudo-operation requests 32 bits of storage. If the location counter is not properly aligned on a boundary for a data item of that size, the Assembler advances the location counter to the next multiple of that item's size before reserving the area.

When you label one of these pseudo-operations, the label refers to the first byte of the storage area. Operands separated by commas initialize successive units of storage.

Example

```
E    .BYTE    "["
F    .WORD    -32
     .WORD    0X6eff1234
```

The first pseudo-operation allocates a byte labeled "E" and initializes it to the character "[". The next pseudo-operation advances the current subspace's location counter to a word boundary, allocates a word of storage labeled "F" and initializes that word to negative 32 (2's complement). The last pseudo-operation initializes a word of storage to the hexadecimal number 6EFF1234.

The .CALL Directive

The .CALL directive marks the next branch statement as a procedure call, and permits you to describe the location of arguments and the function return result.

Syntax

```
.CALL [argument_description]
```

Parameters

argument_description Allows you to communicate to the linker the types of registers used to pass floating point arguments and receive floating point return results in the succeeding procedure call. Similarly, this information can be communicated in the .EXPORT directive.

The linker requires this information because the Procedure Calling Convention allows floating point arguments and return values to reside in either general registers or floating point registers, depending on source language convention. At link time, the linker ensures that both the caller and called procedure agree on argument location. If not, the linker may insert code to relocate the arguments (or return result) before control is transferred to the called procedure or a procedure return is completed.

Up to 5 *argument-descriptions* may be present in the .CALL directive; one for each of the four arguments that may be passed in registers (arg0-arg3), and one for a return value (ret0).

The form of the *argument-description* is:

```
ARG  = location
```

where ARG may be:

ARGW0	The first word in the argument list.
ARGW1	The second word in the argument list.
ARGW2	The third word in the argument list.
ARGW3	The fourth word in the argument list.
RTNVAL	The return value for a procedure.

and location may be:

NO	The argument word cannot be relocated. This should be used for all non-floating point arguments; it is the default assumed when an *argument-description* is omitted.
GR	The argument word occurs in a general register.
FR	The argument word occurs in a floating point register.
FU	The argument word occurs in the upper half of a floating point register.

Example

```
;  This program calls printf() with four arguments
;  whose register locations are described in the .CALL directive.
;  The format string goes into arg0, not to be relocated.
;  The string "message" goes into arg1, specified as a general register.
;  The floating-point value 57005.57005 goes into farg2,
;  specified as a floating-point register.
;  The hexadecimal number 0xf00d goes into arg3,
;  specified as a general register.
;  The return value from printf() is not to be relocated.

          .LIT
          .ALIGN  8
          .WORD   1197387154          ; floating-point literal
          .BLOCKZ 12
fp2       .WORD   0
          .CODE
main
          .PROC
          .CALLINFO CALLER,FRAME=24,SAVE_RP
          .ENTER
          LDIL    L'fp2,%r1
          LDO     R'fp2(1),%r31       ; r31 < - floating-point literal address
          FLDWS   -16(0,%r31),%fr4
          LDO     -64(%sp),%r19
          FSTWS   %fr4,0(0,%r19)
          ADDIL   L'61453,0
          LDO     R'61453(%r1),%r20
          STW     %r20,-68(0,%sp)     ; end of stacking floating-point address
          ADDIL   L'string_area-$global$,%dp
          LDO     R'string_area-$global$(%r1),%r21  ; point to "message"
          STW     %r21,-60(0,%sp)     ; stack "message" address
          LDO     -64(%sp),%r22
          FLDWS   0(0,%r22),%fr5
          FCNVFF,SGL,DBL    %fr5,%fr6    ; convert floating-point value
          ADDIL   L'string_area-$global$+8,%dp
          LDO     R'string_area-$global$+8(%r1),%arg0 ;point to format string
          LDW     -60(0,%sp),%arg1    ; load "message" argument
          FSTDS   38,-16(0,%sp)
          FLDWS   -12(0,%sp),%fr6     ; load floating-point argument
          LDWS    -16(0,%sp),%arg3    ; load hexadecimal argument
          LDW     -68(0,%sp),%r1
          STW     %r1,-52(0,%sp)
          .CALL   argw0=no,argw1=no,argw2=fr,argw3=no,rtnval=no
          BL      printf,2
          NOP
          .LEAVE
          .PROCEND
          .EXPORT main,ENTRY
          .IMPORT printf,CODE

          .DATA
string_area
```

```
.ALIGN  8
.STRINGZ          "message"
.STRINGZ          "args = %s,%f,%x\n"
.IMPORT $global$,DATA
```

This example shows the use of the .CALL directive.

The .CALLINFO Directive

.CALLINFO is a required directive that describes the environment of the current procedure. The information it provides is available to the .ENTER and .LEAVE pseudo-operations to control the entry and exit code sequences they generate. Additional information is used by the Assembler to direct the creation of stack unwind descriptors.

Syntax

```
.CALLINFO  [FRAME=number]
           [,NO_UNWIND]  [,SAVE_SP]  [,SAVE_RP]
           [,ENTRY_GR=number]
           [,ENTRY_FR=number]
           [,ENTRY_SR=number]  [,CALLER | CALLS]  [,NO_CALLS]
           [,HPUX_INT]
```

NOTE

The first parameter in the syntax example is not preceded by a comma, but the following parameters are preceded by commas. This example uses FRAME as the first parameter which is an arbitrary choice.

Parameters

FRAME=*number*

Defines the combined size (in bytes) of the local variable area and variable argument area needed by the procedure. The .ENTER pseudo-operation allocates the desired space for local variables below the frame marker and the .LEAVE pseudo-operation deallocates that space.

The *number* parameter must be a multiple of eight bytes. If a .CALLINFO directive lacks this parameter, the Assembler assumes a default frame size of zero.

The stack frame includes space for local variables and the variable argument area. The size specified for the frame should not include space for the stack frame marker or the fixed argument area. Allocation of these areas is controlled by the CALLER and NO_CALLS parameters. The inclusion of the CALLER parameter always allocates space for the stack frame marker and the fixed argument area. (See Figure 3-1.)

A frame marker is required if the assembly routine calls another routine.

Because the frame marker contains 32 bytes and the fixed argument list contains 16 bytes, the frame area is offset from the Stack Pointer by 48 bytes if both of these areas are present. However, the Assembler does not allocate space for the frame marker and fixed argument list if the procedure does not call any other routines (see the NO_CALLS parameter).

When the total frame size for a procedure exceeds 8K bytes, the Assembler uses gr3 to locate the previous frame marker when it encounters an .ENTER or .LEAVE pseudo-operation. Under these circumstances, changing the value of gr3 can cause serious consequences.

NO_UNWIND

This is to be used only in the context of *stand alone code* or any procedure that does not need to be reliably *unwound*.

SAVE_SP

Specifies that the current routine saves the value of Previous_SP in its frame marker at SP-4. Because the Assembler does not automatically save the Stack Pointer when it generates Entry/Exit code sequences, you must explicitly save this value in your program when using this key word. (You can obtain the Previous_SP value from pseudo-register number 64.)

Programming languages, such as Pascal, typically use this value for up-level display pointers to reference local variables.

SAVE_RP

Specifies that the frame marker of the previous routine stores the value of the Return Pointer (RP). The Assembler automatically saves the Return Pointer when it encounters an .ENTER pseudo-operation, and it restores the RP value when it encounters a .LEAVE pseudo-operation. Generally, any procedure that calls other routines should save the RP value.

ENTRY_GR=*number*

Specifies the high end boundary of the Entry/Save register partition. The partition may extend over registers gr3 through gr18. If you omit this parameter, none of the registers are saved.

When a procedure uses these registers, the Assembler saves their values when it encounters an .ENTER pseudo-operation and restores these values when it encounters a .LEAVE pseudo-operation. The called routine saves these registers upon entry and restores them upon exit, so values in Entry/Save registers are preserved across a procedure call.

NOTE

See the previous description of the FRAME parameter regarding the use of gr3.

ENTRY_FR=*number*

Specifies the high end boundary of the Entry/Save floating-point register partition. The partition includes fr12 through fr15. The Assembler automatically saves these registers when it encounters an .ENTER pseudo-operation and restores them when it encounters a .LEAVE pseudo-operation.

ENTRY_SR=*number*

Specifies the high end boundary of the space register partition. The

partition currently contains only sr3. When the .CALLINFO directive includes this parameter, the Assembler automatically saves the Space Register when it encounters an .ENTER pseudo-operation and restores this register when it encounters a .LEAVE pseudo-operation.

CALLER
or
CALLS

Indicates that this procedure calls other routines so it requires space in the stack for a frame marker and a fixed argument list. (When a program is assembled using the -f option, this becomes the default case.) The Assembler allocates this space (48 bytes) when it encounters an .ENTER pseudo-operation and deallocates this space when it encounters a .LEAVE pseudo-operation.

The frame marker and fixed argument list area occur at the top of the stack so you must take this space into account when locating local variables on the stack. You must allocate an area (using FRAME=) for a variable argument list when this area is needed.

CALLER does not imply the existence of the parameter SAVE_RP.

The CALLER and CALLS parameters are equivalent.

NO_CALLS

Indicates that the procedure does not call other procedures and, therefore, does not require a frame marker on the stack. This is the default case unless the program is assembled using the -f option.

HPUX_INT

Specifies that this procedure is an interrupt procedure. This is necessary for the stack unwind mechanism.

A stack frame consists of a pointer to the top of the frame, a frame marker, a fixed argument list, and a variable argument list. Figure 3-1 illustrates these areas as an inverted stack.

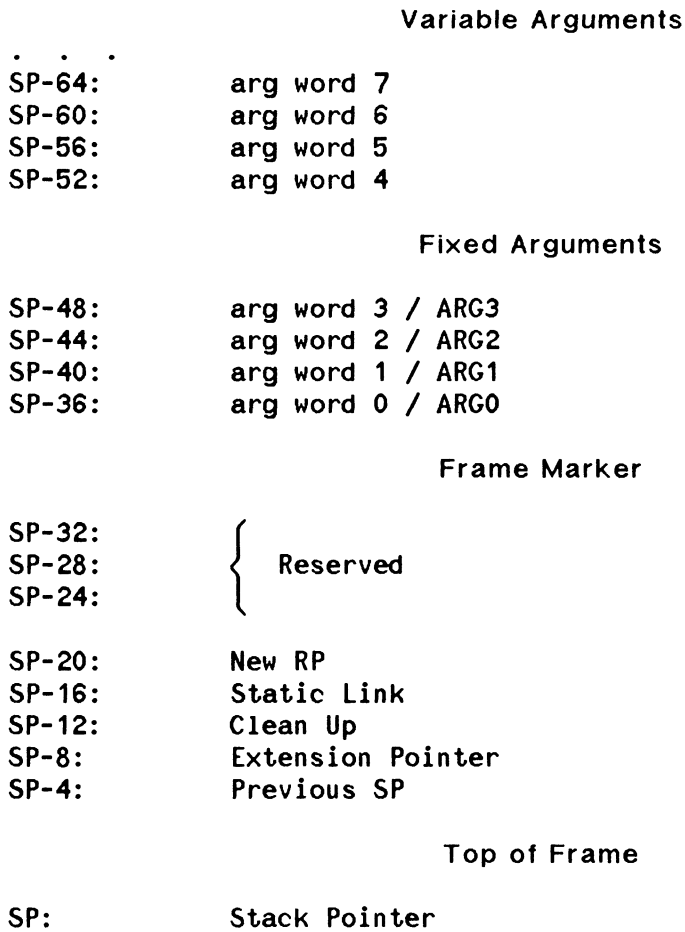


Figure 3-1. Stack Frame

Discussion

When a program uses the .CALLINFO directive, all entry and exit code must follow the Procedure Calling Convention and the Assembler automatically generates the necessary code. The parameters in the .CALLINFO directive govern the generation of the Entry/Exit code sequence (except for SAVE_SP).

Example

```
        .CODE                           ; declare space and subspace
main
        .PROC                           ; delimit procedure entry
        .CALLINFO CALLER,FRAME=0,SAVE_RP    ; no local variables, need return
        .ENTER                          ; insert entry code sequence
        ADDIL  L'stringinit-$global$,27    ; point to data to be printed
        LDO    R'stringinit-$global$(1),26 ; place argument to printf
        .CALL                           ; set up for procedure call
        BL     printf,2                 ; call printf, remembering from where
        NOP
        .LEAVE                          ; insert exit code sequence
        .PROCEND                        ; delimit procedure end

        .DATA                           ; declare space and subspace
stringinit                              ; mark use of global data subspace
        .STRINGZ      "hello world\n"   ; declare some data
        .IMPORT  $global$,DATA          ; get data reference point
        .CODE                           ; re-enter code subspace
        .EXPORT main,ENTRY              ; make routine known to linker
        .IMPORT printf,CODE             ; external procedure declaration
        .END
```

This example uses the C printf routine. It illustrates most of the directives to be used when assembly language programmers follow the standard procedure calling convention.

The .COMM Directive

The .COMM directive makes a storage request for a specified number of bytes.

Syntax

```
label   .COMM   [num_bytes]
```

Parameters

label	Labels the location of the reserved storage.
num_bytes	An integer value for the number of bytes you want to reserve. The Assembler uses a default value of 4 if the .COMM directive lacks a *num_bytes* parameter. Permissible values range from one to 0x7FFFFFFF.

Discussion

The .COMM directive declares a block of storage that may be thought of as a *common block*. You must label every .COMM directive. The linker associates the *label* with the subspace in which the .COMM directive is declared and allocates the necessary storage within that subspace. It is recommended that .COMM appear only in the BSS subspace of the $PRIVATE$ space. If the *label* of a .COMM directive appears in several object modules, the linker uses the maximum size specified in any module when it allocates the necessary storage in the current subspace.

Example

```
            .BSS
mydata      .COMM 16
```

This example reserves 16 bytes of storage for mydata.

The .COPYRIGHT Directive

The .COPYRIGHT directive inserts a company name and date into the object module as a copyright notice.

Syntax

```
.COPYRIGHT  "company name[, date]"
```

Parameters

company name, date — A sequence of ASCII characters, surrounded by quotation marks. The string can contain up to 256 characters. When a comma follows the company name, the next text is expected to be the date. The default date is 1985.

Example

```
            .COPYRIGHT "company name, date"
            .CODE
            .EXPORT main,ENTRY
main
            .PROC
            .CALLINFO
            .ENTER
            LDI     2,%r5
            ADDI    2,%r5,%r6
            .LEAVE
            .PROCEND
```

This program places a copyright notice in the object file. Once the copyright notice is in the object file, the HP-UX utility, *strings*, is used to access it.

NOTE

This directive can appear anywhere in the source file, but may appear only once.

The following is the standard copyright message placed in the copyright header of the object file:

> Copyright *company name, date*. All rights reserved. No part of this program may be photocopied, reproduced, or transmitted without prior written consent of *company name*.

The .DOUBLE Directive

The .DOUBLE directive initializes a double-word to a floating-point value calculated from the parameters provided. If the location counter is not aligned on a word double-word boundary, it is forced to the next multiple of eight. If the statement is labeled, the label refers to the first byte of the storage area.

Syntax

```
.DOUBLE  integer [ .'decimal][ E'[ -']power]
```

Parameters

integer	Specifies the whole number part of a decimal number.
decimal	Specifies the fractional part of a decimal number.
power	Specifies the power of ten to raise a decimal number. To raise the decimal number to a negative power of ten, place a minus sign (-) directly in front of the power specified.

Discussion

Each of the following examples initializes two words of memory to floating-point quantities: 0.00106 and 400000.0 respectively.

Example

```
dec_val1  .DOUBLE 10.6E-4

dec_val2  .DOUBLE 0.4E6
```

The .END Directive

The .END directive terminates an assembly language program.

Syntax

```
.END
```

Discussion

This directive is the last statement in an assembly language program. If a source file lacks an .END directive, the Assembler terminates the program when it encounters the end of the file.

Example

```
            .CODE
            .EXPORT double,ENTRY
            .PROC
double
            .CALLINFO
            .ENTER
            ADD     %arg0,%arg0,%ret0
            .LEAVE
            .PROCEND
            .END
```

A file that omitted the last line of this sample program would produce identical results.

The .ENDM directive

The .ENDM directive marks the end of a macro definition. The macro definition is entered into the macro table and the remaining source lines are read in and assembled. An .ENDM directive must always accompany a .MACRO directive.

Syntax

```
.ENDM
```

Example

```
QUADL   .MACRO  WD1,WD2,WD3,WD4
        .ALIGN  16
        .WORD   WD1
        .ALIGN  16
        .WORD   WD2
        .ALIGN  16
        .WORD   WD3
        .ALIGN  16
        .WORD   WD4
        .ENDM
```

This example defines the macro QUADL; it aligns the data specified in the macro parameters on quad word boundaries.

The .ENDM directive delimits the end of the definition of QUADL.

The .ENTER and .LEAVE Pseudo-Operations

The .ENTER and .LEAVE pseudo-operations mark a procedure's entry and exit points. They instruct the Assembler to generate procedure entry and exit code sequences based on the information provided in the .CALLINFO directive.

Syntax

```
.ENTER

.LEAVE
```

Discussion

The .ENTER pseudo-operation marks an entry point for the current procedure. Every procedure following the standard procedure calling convention must contain one .ENTER pseudo-operation. The .LEAVE pseudo-operation marks a procedure's exit point. Every procedure following the standard procedure calling convention must contain one .LEAVE pseudo-operation.

When the Assembler encounters an .ENTER pseudo-operation, it generates an entry code sequence according to the parameters in the .CALLINFO directive for that procedure. Similarly, when the Assembler encounters a .LEAVE pseudo-operation, it generates an exit code sequence according to the parameters in the .CALLINFO directive for that procedure.

Example

```
        .SPACE $TEXT$
        .SUBSPA $CODE$
    entrypt
        .PROC
        .CALLINFO
        .ENTER
        SH1ADD      %arg0,%arg1,%ret0
        .LEAVE
        .PROCEND
        .EXPORT entrypt,ENTRY
        .END
```

This example shows the placement of the .ENTER and .LEAVE pseudo-operations.

The .ENTRY and .EXIT Directives

.ENTRY and .EXIT are compiler generated directives that mark the entry point and return point of the current procedure.

Syntax

```
.ENTRY

.EXIT
```

Discussion

The .ENTRY directive signifies that the next statement is the beginning of an entry point for the current procedure. The .EXIT directive signifies that the next statement initiates a return from the current procedure. These directives are issued by compilers and are not used by assembly language programmers.

Example

```
        .ENTRY                                  ; proc entry code follows
        STW     2,-20(0,30)                     ; stack the return pointer
        LDO     48(30),30                       ; set up user stack pointer
        ADDIL   L'$THISMODULE$-$global$,27      ; point to printf data
        .CALL                                   ; set up for printf call
        BL      printf,2                        ; call printf thru RP
        LDO     R'$THISMODULE$-$global$(1),26   ; insert argument to printf
L$exit1                                         ; hide from linker
        LDW     -68(0,30),2                     ; get callee RP
        BV      0(2)                            ; exit thru RP
        .EXIT                                   ; end of exit sequence
```

This example shows a sequence of compiler-generated assembly code.

The .EQU Directive

The .EQU directive assigns an expression value to an identifier.

Syntax

```
symbolic_name  .EQU  value
```

Parameters

symbolic_name — Names the identifier to which the Assembler assigns the expression.

value — An integer expression. The Assembler evaluates the expression, which must be absolute, and assigns this value to *symbolic_name*. If the expression references other identifiers, each identifier must be defined before the .EQU directive attempts to evaluate the expression.

NOTE

The Assembler prohibits the use of offset values (instruction labels) and imported symbols as components of an expression.

Example

```
loc1 .EQU  0
loc2 .EQU  loc1+4
     .SPACE $TEXT$
     .SUBSPA $CODE$
     LDW   loc1,%r1
     STW   %r1,loc2
     .END
```

This is a valid assembly program because the definition of loc1 comes before the definition of loc2. Reversing the first two statements, however, produces an error condition.

The .EXPORT and .IMPORT Directives

The .EXPORT and .IMPORT directives allow symbols to be defined in one program but used in other programs.

Syntax

```
.EXPORT  symbol [,type] [,argument-description]
   or
.IMPORT  symbol [,type]
```

Parameters

symbol — The name of an identifier whose definition is being exported or imported.

type — A linker symbol type that can take one of the following values:

ABSOLUTE	Designates an absolute symbol.
DATA	Designates a data symbol.
CODE	Designates a code location. The location can not be a procedure entry.
ENTRY	Designates the entry point of a procedure.
MILLICODE	Locates code for the entry point of a millicode subroutine.
PLABEL	Locates a pointer to a procedure.
PRI_PROG	Designates the primary program entry point. The outerblock of Pascal and the main program in FORTRAN are type PRI_PROG.
SEC_PROG	Designates a secondary program entry point.

argument-description Allows you to communicate to the linker the types of registers used to receive floating point arguments and return floating point return results. Similarly, this information can be communicated in the .CALL directive.

The linker requires this information since the Procedure Calling Convention allows floating point arguments and return values to reside in either general registers or floating point registers depending on source language convention. At link time, the linker ensures that both the caller and called procedure agree on argument location. If not, the linker may insert code to relocate the arguments (or return result) before control is transferred to the called procedure or a procedure return is completed.

The form of the *argument-description* is described in the discussion of the .CALL directive. Refer to the .CALL directive in this chapter for more information.

Discussion

Both the .EXPORT and .IMPORT directives use a series of keywords to define a symbol to the linker. These keywords declare the symbol's type, and its argument relocation information if the symbol is the name of a procedure.

Because the .IMPORT directive specifies that another object module contains this symbol's formal definition, the Assembler does not associate an imported symbol with any particular subspace.

When an .IMPORT directive lacks a *type* parameter, the Assembler assigns the type of the current subspace (either CODE or DATA) to the symbol.

Example

```
.IMPORT symname,CODE
.CODE
LDIL    L'symname,%r1
BLE,n   R'symname(%sr4,%r1)
NOP
.END
```

This example imports the symbol symname, then loads the right part of symname with respect to gr1 into gr26.

The .FLOAT Directive

The .FLOAT directive initializes a double-word of storage to a floating-point value calculated from the parameters provided. If the location counter is not aligned on a word boundary, it is forced to the next multiple of four. If the statement is labeled, the label refers to the first byte of the storage area.

Syntax

```
.FLOAT  integer ['.'decimal]['E'['-']power]
```

Parameters

integer	Specifies the whole number part of a decimal number.
decimal	Specifies the fractional part of a decimal number.
power	Specifies the power of ten to raise a decimal number. To raise the decimal number to a negative power of ten, place a minus sign (-) directly in front of the power specified.

Discussion

Each of the following examples initializes one word of memory to floating-point quantities: 0.00096 and 3400000.0 respectively.

Example

```
factor1  .FLOAT  9.6E-4

factor2  .FLOAT  3.4E6
```

The .LABEL Directive

The .LABEL directive permits a label definition to appear within a sequence of instructions that occur on a single line.

Syntax

```
.LABEL  label_id
```

Parameters

label_id	Names the label identifier.

NOTE

The .LABEL directive is especially useful when using the M4 macroprocessor or the C pre-processor (cpp). You would normally use this directive in a DEFINE macro that includes multiple instructions. However, because the Assembler does not process M4 or cpp style macros, you must run programs that contain the .LABEL directive through the M4 pre-processor or the C pre-processor (cpp).

Example

```
#define Loop(xx) LDO xx(%r0),%r1 ! .LABEL Loop ! ADDI,= -1,%r1,%r1 \
! BL Loop,%r0  ! NOP  ! LDI 1,%ret0 ; macro
          .CODE
step_ten
          .PROC
          .CALLINFO
          .ENTER
          Loop(10)
          .LEAVE
          .PROCEND
          .EXPORT step_ten,ENTRY
```

This example defines a CPP macro named Loop.

The .LISTOFF and .LISTON Directives

The .LISTOFF and .LISTON directives control the expansion of instructions for all macro invocations, all predefined subspace declarations, and the .ENTER and .LEAVE pseudo-operations. .LISTOFF causes the Assembler to cease listing expanded instructions until a .LISTON directive is encountered. .LISTON causes the Assembler to list expanded instructions until a .LISTOFF directive is encountered. The default is .LISTOFF.

Syntax

```
.LISTOFF
.LISTON
```

The following is an example of .LISTON in an assembly listing of a procedure containing a macro invocation.

Example

```
line offset   hexcode  label    opcode            operands (comment)
1                                .LISTON
2                                .CODE
                                 .SPACE $TEXT$,    SPNUM=0,SORT=0
                                 .SUBSPA $CODE$,   QUAD=0,ALIGN=8,ACCESS=0x2c
3                                .PROC
4                      call_DECR                   ;proc label
5                                .CALLINFO         FRAME=0,SAVE_RP
6                                .ENTER
    00000000 (6BC23FD9)          STW               2,-0x14(0,0x1E)
    00000004 (37DE0060)          LDO               0x30(0x1E),0x1E
7   00000008 (2B600000)          ADDIL             L'count-$global$,%dp
8   0000000C (683A0000)          STW               %arg0,R'count-$global$(%r1)
9                                DECR              mark,count;
                                                   macro invocation
    00000010 (2B600000)          ADDIL             L'VAL-$global$,%dp
    00000014 (48340000)          LDW               R'VAL-$global$(%r1),%r20
                       LAB
    00000018 (AE9F3FF5)          ADDIBF,=          -1,%r20,LAB
    0000001C (08000240)          NOP
10                               .LEAVE
    00000020 (4BC23F79)          LDW               -0x44(0,0x1E),2
    00000024 (E840C000)          BV                0(2)
    00000028 (37C03FA1)          LDO               -0x30(0x1E),0
11                               .PROCEND
12                               .EXPORT           call_DECR,ENTRY

13                               DATA
                                 .SPACE $PRIVATE$, SPNUM=1,SORT=16
                                 .SUBSPA $DATA$,   QUAD=1,ALIGN=8
                                                   ACCESS=0x1f
14                               .IMPORT           $global$
15  40000000 (00000000) count    .WORD             0
16                               .LISTOFF
```

In the above example, if .LISTOFF had been used, the macro invocation DECR, and the directives .CODE, .DATA, .ENTER, and .LEAVE, would not have been expanded in the assembly listing.

The .LOCCT Directive

The .LOCCT directive specifies where subsequent code should occur in one of the four location counters of the current subspace. Note that the .LOCCT directive is not permitted within a procedure.

Syntax

```
.LOCCT  [loc_number]
```

Parameters

loc_number	A location counter number of the current subspace. The permissible values are 0, 1, 2, and 3. The default is zero.

Example

```
        .CODE
        .LOCCT  0
ldval1
        LDIL    L'val1,%r19
        LDO     R'val1(%r19),%r19
        .LOCCT  1
val1    .WORD   57005
        .LOCCT  0
ldval2
        LDIL    L'val2,%r20
        LDO     R'val2(%r20),%r20
        .LOCCT  1
val2    .WORD   61453
```

This example uses two location counters to separate code from data. In the assembled code, everything under location counter 0 comes first, followed by everything under location counter 1, and so on.

The .MACRO Directive

The .MACRO directive marks the beginning of macro definitions.

Syntax

```
label .MACRO  [formal_parameter]  [,formal_parameter]...
```

Parameters

formal_parameter — Specifies a string of characters treated as a positional parameter. The *i*th actual parameter in a macro invocation is substituted for the *i*th formal parameter in the macro declaration wherever the formal parameter appears in the body of the macro definition.

Normal Assembler syntax is observed within macro definitions except text substitution is assumed for formal parameters. The following line is an example of a macro declaration:

```
DECR    .MACRO LAB,VAL
```

LAB and VAL are formal parameters. Their actual values are determined by the first and second parameters on any invocation of the macro DECR. On the macro invocation, the parameters are delimited by commas. Successive commas indicate a null parameter, causing the expanded macro to substitute null for one of its formal parameters. When the number of formal parameters exceeds the number of actual parameters, null parameters are inserted for the excess parameter positions. When the number of actual parameters exceeds the number of formal parameters, a warning is issued and the excess parameters are ignored.

Example

```
DECR    .MACRO    LAB,VAL
SETP    ADDIL     L'VAL-$global$,%dp
        LDW       R'VAL-$global$(%r1),%r20
LAB
        ADDIBF,=  -1,%r20,LAB
        NOP
        .ENDM
```

The above macro definition defines a simple counter or timer call DECR. Following is an invocation to DECR:

```
DECR LOOP,COUNT
```

LOOP and COUNT are the actual parameters that are specific to this particular invocation of the macro DECR.

During macro expansion, textual substitution for positional parameters is performed in the body of the macro definition. Substitution is performed on strings of characters that are delimited by blanks, tabs, commas, or semicolons. If the string matches one of the formal parameters, it is replaced with the corresponding actual parameter.

When a macro definition contains a label, the expanded form of the macro adds a unique suffix to the label for each instance the macro is invoked. This unique suffix prevents duplicate symbols from occurring and prevents the label from being referenced from outside the body of the macro definition. This suffix also contains a number that is used as a counter by the Assembler.

Examples

```
PRINT   .MACRO   DATA_ADDR
        ADDIL    L'DATA_ADDR,%dp
        .CALL
        BL       print,%rp
        LDO      R'DATA_ADDR(%r1),%arg0
        .ENDM
```

The above example defines the macro PRINT to call *printf*. The macro parameter DATA_ADDR is used to set up the argument to be passed to *printf*.

```
STORE   .MACRO   REG,LOC
        LDIL     L'LOC-$global$,%r1
        STW      REG,R'LOC-$global$(%r1)
        .ENDM
```

The above example defines the macro STORE. STORE places the contents of the register REG, the first macro parameter, into the memory address LOC, the second parameter.

NOTE

Although there is no upper limit on the number of parameters or arguments in a macro definition, no single macro parameter may exceed 200 characters.

Discussion

Macro definitions may appear wherever and however often as necessary within source code. Macro definitions may occur inside or outside of spaces, subspaces, and procedures.

Since the Assembler always uses the most recently encountered definition, macros may be redefined as often as desired.

NOTE

A macro may not be defined within the body of another macro definition.

Although nested macro definitions are not allowed, nested macro calls are. A nested macro call occurs when one macro is invoked within the definition of another macro. A macro may not be invoked within its own definition. Macros can only be invoked after being defined.

The .ORIGIN Directive

The .ORIGIN directive advances the location counter to the specified location.

Syntax

```
.ORIGIN  [location]
```

Parameters

location

An integer value for the offset you want to advance the location counter. Permissible values range from zero to 0x7FFFFFFF. The default value is zero. The value specified cannot be less than the current value of the location counter.

When the Assembler encounters an .ORIGIN directive, it issues a .BLOCK pseudo-operation of a size calculated to advance the location counter to the requested origin. (See the discussion of the .BLOCK pseudo-operation.)

Example

```
        .CODE
        XOR     %r21,%r22,%r23
        B  idx
        NOP
        .ORIGIN 64
idx     LDWX    %r23(0,0),%r3
       .END
```

This sample program does an exclusive OR operation and advances the location counter to 64 bytes where the label idx is located as a branch target.

The .PROC and .PROCEND Directives

The .PROC and .PROCEND directives bracket the statements within a procedure.

Syntax

```
.PROC

.PROCEND
```

Discussion

The .PROC directive signifies that the next statement is the first statement of a procedure. The .PROCEND directive signifies that the previous statement was the last statement of the procedure. Switching spaces or subspaces within a procedure is not permitted.

Every procedure must contain a .CALLINFO directive and normally contains an .ENTER and .LEAVE pseudo-operation. The only exception to the latter rule occurs in procedures that are either compiler-generated or created by programmers who are writing their own entry and exit code sequences.

NOTE

Because the .ENTER and .LEAVE pseudo-operations guarantee that the stack unwind process works correctly, you should consistently use these directives rather than writing your own entry and exit code sequences.

Example

```
        .CODE
test
        .PROC
        .CALLINFO
        .ENTER
        COMCLR,=        %arg0,%arg1,%ret0
        LDI             1,%ret0
        .LEAVE
        .PROCEND
        .EXPORT test
```

This template shows a procedure that follows the standard procedure calling convention.

The .REG Directive

The .REG directive, which must be labeled, attaches a type and number to a user-defined register name. The new register name may optionally begin with %.

Syntax

```
label .REG [typed_register]
```

Parameters

typed_register	Must either be one of the predefined Assembler registers or a previously defined user-defined register name. All predefined Assembler registers begin with %.

Example

```
shift   .REG %SAR
```

The example above defines the register `shift` with *control* register type and register number eleven. %SAR is a synonym for control register eleven, the shift amount register.

The .SPACE Directive

The .SPACE directive starts a new space or switches back to an old space.

Syntax

```
.SPACE name [,SPNUM=value] [,UNLOADABLE]
            [,NOTDEFINED] [,PRIVATE] [,SORT=<value>]
```

Parameters

name	An identifier that names the new space.
SPNUM=*value*	A space number constant that provides a specific number for the current space. Its use is currently optional and is ignored by the linker. If the first parameter of the .SPACE directive is an integer, it will be interpreted as the space number and any remaining parameters will be ignored.
UNLOADABLE	Specifies that the space resides on disk and is not loadable into main memory. Debugger data is a typical example of an unloadable space.
NOTDEFINED	Specifies that the definition for this space occurs in another object module.
PRIVATE	Specifies that other programs cannot *share* the data in this space. Enforcement of this directive depends on the operating system.
SORT=value	Provides an integer value for the sort key. The linker orders the spaces in the output object module according to this key. It is suggested that the number "8" be used for space $TEXT$ and the number "16" be used for $PRIVATE$.

Discussion

The first time the Assembler encounters a .SPACE directive with a new *name*, it uses that name to declare a new space. As this is the defining occurrence of that space, additional keywords can describe attributes for that space.

If the Assembler encounters subsequent .SPACE directives with that *name*, it continues that space. In this case, where the program is re-entering a previously defined space, the .SPACE directive can only contain the space name; other keywords to describe the space are illegal.

A space can contain from one to four discrete quadrants (See the QUAD parameter of the .SUBSPA directive.) When you divide a space into multiple quadrants, you must define all the subspaces within each quadrant as a group. If subspaces for a quadrant are defined individually, program operation is unpredictable. The Assembler, however, does not check for this condition.

Example

```
.SPACE  $TEXT$
.SUBSPA $CODE$,     QUAD=0,ALIGN=8,ACCESS=0x2c,SORT=24
   .
   .
   .
.SPACE  $PRIVATE$, 1
.SUBSPA $DATA$,     QUAD=1,ALIGN=8,ACCESS=0x1f,SORT=24
.SUBSPA $BSS$,      QUAD=1,ALIGN=8,ACCESS=0x1f,SORT=80, ZERO
   .
   .
   .
.SPACE  $myspace$
.SUBSPA $myspace_ADDRESS$,      ALIGN=4,ACCESS=0x2c,UNLOADABLE
```

The above example shows some of the standard "space" definitions in a typical assembly language program.

The .SPNUM Directive

The .SPNUM directive reserves a word of storage and initializes it with the space number of the space named by the operand. Only one operand is allowed and any label present is offset at the first byte of the storage just initialized.

Syntax

```
.SPNUM name
```

Parameters

name	Specifies the name of a space whose space number is used to initialize a word of storage.

Example

```
            .SPACE    $PRIVATE$,SPNUM=1    SORT=16
            .SUBSPA   $DATA$,QUAD=1,       ALIGN=8,ACCESS=0x1f   SORT=24
data_ref
            .WORD                          0xFFFF
LOG         .SPNUM                         $PRIVATE$
```

In the above example, the space number of $PRIVATE$, 1, is stored as the address of the symbol LOG by the .SPNUM directive.

NOTE

Space numbers are ignored by the linker.

The .STRING and .STRINGZ Pseudo-Operations

The .STRING pseudo-operation reserves storage for a data area and initializes it to ASCII values. The .STRINGZ pseudo-operation reserves storage the same as .STRING, but appends a zero byte to the data. This creates a C language type string. If the statement is labeled, the label refers to the first byte of the storage area.

Syntax

```
.STRING[Z]  "init_value"
```

Parameters

init_value

A sequence of ASCII characters that are surrounded by quotation marks. A string can contain up to 256 characters.

Specifies a sequence of ASCII characters enclosed in quotation marks. The quotation marks are not generated into the storage area. The HP-UX style escape sequences \0, \n, \t, \b, \r, \f, \\, and \" are recognized. In addition, the escape sequences \X or \x followed by two hexadecimal digits can be used to represent any 8-bit character (the HP-UX sequence \0 becomes \X00).

Discussion

The .STRING pseudo-operation requests the required number of bytes to store the string (where each character is stored in a byte). The .STRINGZ pseudo-operation also requests the required storage for the quoted string but then appends a zero byte for compatibility with C language strings.

When you label one of these pseudo-operations, the label refers to the first byte of the storage area.

Examples

```
G   .STRING "A STRING"
```

This pseudo-operation allocates eight bytes, the first of which is labeled "G", then initializes this area with the characters: A, "space", S, T, R, I, N, and G.

```
G   .STRINGZ  "A STRING"
```

This pseudo-operation allocates eight bytes to hold A STRING, allocates an additional byte for the appended zero, and associates the label "G" with the first byte of this storage area.

The .SUBSPA Directive

The .SUBSPA directive declares a new subspace or switches back to an old subspace.

Syntax

```
.SUBSPA name [,QUAD=value]t ,ALIGN=value]
             [,ACCESS=value] [,SORT=value]
             [,FIRST] [,COMMON] [,ZERO] [,DUP_COMM]
             [,CODE_ONLY] [,UNLOADABLE]
```

Parameters

name	An identifier that names the current subspace.
QUAD=*value*	Specifies the quadrant (0 through 3) in which the Assembler should place this subspace. The default value is zero.
FIRST	Specifies that the subspace must be allocated exactly at the end of the specified space. The default is false.
ALIGN=*value*	Specifies a value (which must be a power of 2) on which the Assembler should align the beginning of the subspace. The default value is the largest alignment requested within that subspace, or one if no alignment requests exist.
ACCESS=*value*	Specifies the 7-bit value for the access rights field in the PDIR (Physical Page Directory for virtual address mapping). Must be 0X2C for code, and 0X1F for data subspaces.
SORT=*value*	Provides an integer value for the primary sort key. The linker orders the subspaces in the output object module according to this primary key. If several subspaces share the same primary key value, the linker lists these subspaces in the order in which it processes them. It is suggested that 24 be used for both code and data subspaces.
COMMON	Specifies that this subspace is a common block. The default setting is false.
ZERO	Specifies that this subspace contains all zeros and no data appears in the output file. The default setting is false.
DUP_COMM	Specifies that the initialized data symbols within this subspace may have duplicate names. When you include this parameter, multiple occurrences of a universal data symbol can exist and the linker does not report a "Duplicate Definition" error. The default setting is false.
CODE_ONLY	Specifies that this subspace contains only code. The default setting is false.

UNLOADABLE	Specifies that this subspace is not loadable into memory. The default setting is false. Loadable subspaces must reside in loadable spaces, and unloadable subspaces must reside in unloadable spaces.

Discussion

The first time the Assembler encounters a .SUBSPA directive with a new *name*, it uses that name to declare a new subspace. As this is the defining occurrence of that subspace, optional keywords describe attributes of that subspace.

When the Assembler encounters additional .SUBSPA directives with that *name*, it continues that subspace. In this case, the .SUBSPA directive can only contain the subspace name; other keywords to describe the subspace are illegal.

The $CODE$ subspace uses the stack unwind subspace called $UNWIND$. If other code subspaces have an associated stack unwind subspace, the Assembler identifies them with names of the form: $UNWIND$ *<code_subspace_name>*. For subspaces with names of the form $CODE$<mycode>, the Assembler creates an unwind subspace with the name $UNWIND$<mycode>.

Example

```
.SPACE  $TEXT$,     0
.SUBSPA $CODE$,     QUAD=0,ALIGN=8,ACCESS=0x2c,SORT=24
.SPACE  $PRIVATE$, 1
.SUBSPA $DATA$,     QUAD=1,ALIGN=8,ACCESS=0x1f,SORT=24
```

The above example shows some of the standard "subspace" definitions in a typical assembly language program.

The .VERSION Directive

The .VERSION directive places the designated string in the current object module for version identification.

Syntax

```
.VERSION  "info_string"
```

Parameters

info_string	A sequence of ASCII characters, surrounded by quotation marks. The string can contain up to 256 characters.

Discussion

The Assembler places this string in the current object module. A program may contain multiple .VERSION directives.

Example

```
.CODE
.VERSION "Version 1 of This Simple Sample Program"
SUB      %r19,%r20,%r19
DEP      %r19,14,5,%r22
.END
```

This program inserts version information into the object module, and performs subtract and deposit operations. Once the version information is in the object file, the HP-UX utility, *strings*, is used to access it.

NOTE

This directive can appear anywhere in the source file, and multiple occurrences are permitted.

Programming Aids

The Assembler provides a series of standard space and subspace definitions that you may use to simplify the writing of an assembly program. These definitions are duplicated in the system file *pcc_prefix.s*. Because these files are relatively large and may change with new releases of the Assembler, you can view the most recent version of these files on your terminal screen by typing the command:

```
more /lib/pcc_prefix.s.
```

Table 3-4 lists the predefined directives for establishing standard spaces and subspaces.

Table 3-4. Predefined Spaces and Subspaces

Directive	Space Name	Default Parameters
.FIRST	.space $TEXT$, .subspa $FIRST$,	SPNUM=0,SORT=8 QUAD=0,ALIGN=2048,ACCESS=0x2c, SORT=4,FIRST
.REAL	.space $TEXT$, .subspa $REAL$,	SPNUM=0,SORT=8 QUAD=0,ALIGN=8,ACCESS=0x2c, SORT=4,FIRST,LOCK
.MILLICODE	.space $TEXT$, .subspa $MILLICODE$,	SPNUM=0,SORT=8 QUAD=0,ALIGN=8,ACCESS=0x2c,SORT=8
.LIT	.space $TEXT$, .subspa LIT,	SPNUM=0,SORT=8 QUAD=0,ALIGN=8,ACCESS=0x2c,SORT=16
.CODE	.space $TEXT$, .subspa $CODE$,	SPNUM=0,SORT=8 QUAD=0,ALIGN=8,ACCESS=0x2c,SORT=24
.UNWIND	.space $TEXT$, .subspa $UNWIND$,	SPNUM=0,SORT=8 QUAD=0,ALIGN=4,ACCESS=0x2c,SORT=64
.RECOVER	.space $TEXT$, .subspa $RECOVER$,	SPNUM=0,SORT=8 QUAD=0,ALIGN=4,ACCESS=0x2c,SORT=80
.RESERVED	.space $TEXT$, .subspa $RESERVED$,	SPNUM=0,SORT=8 QUAD=0,ALIGN=8,ACCESS=0x73,SORT=82
.GATE	.space $TEXT$, .subspa $GATE$,	SPNUM=0,SORT=8 QUAD=0,ALIGN=8,ACCESS=0x4c, SORT=84,CODE_ONLY

(Continued on next page)

Table 3-4. Predefined Spaces and Subspaces (Continued)

Directive	Space Name	Default Parameters
.GLOBAL	.space $PRIVATE$, .subspa $GLOBAL$, .IMPORT $global$	PRIVATE,SPNUM=1,SORT=16 QUAD=1,ALIGN=8,ACCESS=0x1f,SORT=8
.SHORTDATA	.space $PRIVATE$, .subspa $SHORTDATA$,	PRIVATE,SPNUM=1,SORT=16 QUAD=1,ALIGN=8,ACCESS=0x1f,SORT=16
.DATA	.space $PRIVATE$, .subspa $DATA$,	PRIVATE,SPNUM=1,SORT=16 QUAD=1,ALIGN=8,ACCESS=0x1f,SORT=24
.PFA_COUNTER	.space $PRIVATE$, .subspa $PFA_COUNTER$,	PRIVATE,SPNUM=1,SORT=16 QUAD=1,ALIGN=4,ACCESS=0x1f,SORT=64
.BSS	.space $PRIVATE$, .subspa BSS,	PRIVATE,SPNUM=1,SORT=16 QUAD=1,ALIGN=8,ACCESS=0x1f, SORT=80,ZERO
.PCB	.space $PRIVATE$, .subspa PCB,	PRIVATE,SPNUM=1,SORT=16 QUAD=1,ALIGN=8,ACCESS=0x10,SORT=82
.STACK	.space $PRIVATE$, .subspa $STACK$,	PRIVATE,SPNUM=1,SORT=16 QUAD=1,ALIGN=8,ACCESS=0x1f,SORT=82
.HEAP	.space $PRIVATE$, .subspa $HEAP$,	PRIVATE,SPNUM=1,SORT=16 QUAD=1,ALIGN=8,ACCESS=0x1f,SORT=82
.PFA_ADDRESS	.space PFA, .subspa $PFA_ADDRESS$,	SPNUM=2,PRIVATE,UNLOADABLE,SORT=64 ALIGN=4,ACCESS=0x2c,UNLOADABLE
.HEADER	.space $DEBUG$, .subspa $HEADER$,	SPNUM=2,PRIVATE,UNLOADABLE,SORT=80 ALIGN=4,ACCESS=0,UNLOADABLE,FIRST
.GNTT	.space $DEBUG$, .subspa $GNTT$,	SPNUM=2,PRIVATE,UNLOADABLE,SORT=80 ALIGN=4,ACCESS=0,UNLOADABLE
.LNTT	.space $DEBUG$, .subspa $LNTT$,	SPNUM=2,PRIVATE,UNLOADABLE,SORT=80 ALIGN=4,ACCESS=0,UNLOADABLE
.SLT	.space $DEBUG$, .subspa SLT,	SPNUM=2,PRIVATE,UNLOADABLE,SORT=80 ALIGN=4,ACCESS=0,UNLOADABLE
.VT	.space $DEBUG$, .subspa VT,	SPNUM=2,PRIVATE,UNLOADABLE,SORT=80 ALIGN=4,ACCESS=0,UNLOADABLE

Chapter 4
The Instruction Set

The HP Precision Architecture instructions fall into the following major categories:

- Memory Reference
- Immediate
- Branch
- Computational
- System Control
- Assist (coprocessor)

In addition, pseudo-instructions are available to perform commonly used functions that are variations of the basic instructions.

Instruction Operands

Table 4-1 lists the instruction operands by class as well as meaning. The table is intended as a reference for the operands used in the instruction tables throughout this chapter.

Table 4-1. Instruction Operands

Operand Class	Meaning
b	base register (0 through 31)
d	14 bit displacement
i	5 or 11 bit immediate value
len	length of bit field
p	bit position indicator
count	shift count
r	source register: • general register (0 through 31) • floating point register (0 through 15) • control register (0 and 8 through 31)
r1 and r2	source register: • general register (0 through 31) • floating point register (0 through 15)
sr	space register (0 through 7)
s	space register (0 through 3)
t	destination register: • general register (0-31) • floating-point register (0-15) • control register (0 and 8-31)
wd	word displacement (12 bit immediate)
x	index register (general register 0 through 31)

Memory Reference Instructions

The memory reference instructions load values from memory into general registers and store values from general registers into memory. The displacement or index values can be used to modify the base offset in a general register.

Memory reference instructions include:

- Load and Store
- Load and Store With Base Register Modification
- Indexed Load
- Short Displacement Load and Store
- Store Bytes Short

Load and Store Instructions

A value is transferred between memory and a general register.

Table 4-2. Load and Store Instructions

Instruction Name	Mnemonic Name	Operands
Load Byte	LDB	d(s,b),t
Load Halfword	LDH	d(s,b),t
Load Word	LDW	d(s,b),t
Store Byte	STB	r,d(s,b)
Store Halfword	STH	r,d(s,b)
Store Word	STW	r,d(s,b)

Load and Store with Base Register Modification Instructions

A word is transferred between memory and a general register. The base register is pre-decremented if the displacement is negative; otherwise, the base register is post-incremented.

Table 4-3. Load and Store With Base Register Modification Instructions

Instruction Name	Mnemonic Name	Operands
Load Word and Modify	LDWM	d(s,b),t
Store Word and Modify	STWM	r,d(s,b)

Indexed Load Instructions

A value is read from memory into a general register. The address is formed by the addition of the index register to the base register. Optionally, the index value may be shifted to reflect an offset corresponding to the size of the value being read. The base register may be modified after the address has been formed.

Table 4-4. Indexed Load Instructions

Instruction Name	Mnemonic Name	Operands
Load Byte Indexed	LDBX,cmplt	x(s,b),t
Load Halfword Indexed	LDHX,cmplt	x(s,b),t
Load Word Indexed	LDWX,cmplt	x(s,b),t
Load Word Absolute Indexed *	LDWAX,cmplt	x(b),t
Load and Clear Word Indexed	LDCWX,cmplt	x(s,b),t

* This instruction may be executed only by code running at the most privileged level.

Table 4-5. Indexed Load Completers

cmplt	Description
,S	Shift Index by Data Size, No Base Register Modification
,M	No Index Shift, Base Register Modification
,S,M or ,SM	Shift Index by Data Size, Base Register Modification
none specified	No Index Shift, No Base Register Modification

Short Displacement Load and Store Instructions

A value is transferred between memory and a general register. The address is formed by adding the short displacement to the base register. A short displacement is one that is between -16 and +15 bytes. Optionally, the base register may be modified either before or after the address has been formed.

Table 4-6. Short Displacement Load and Store Instructions

Instruction Name	Mnemonic Name	Operands
Load Byte Short	LDBS,cmplt	d(s,b),t
Load Halfword Short	LDHS,cmplt	d(s,b),t
Load Word Short	LDWS,cmplt	d(s,b),t
Load Word Absolute Short *	LDWAS,cmplt	d(b),t
Load and Clear Word Short	LDCWS,cmplt	d(s,b),t
Store Byte Short	STBS,cmplt	r,d(s,b)
Store Halfword Short	STHS,cmplt	r,d(s,b)
Store Word Short	STWS,cmplt	r,d(s,b)
Store Word Absolute Short *	STWAS,cmplt	r,d(b)

* This instruction may be executed only by code running at the most privileged level.

Table 4-7. Short Displacement Load and Store Completers

cmplt	Description
,MB	Modify Before
,MA	Modify After
none specified	No Base Register Modification

Store Bytes Short Instruction

Byte values are written from a general register into memory. The address is formed using the short displacement, the base register, and the completer. The bytes can be aligned to the beginning or the end of the word. Optionally, the base register can be modified after the address has been formed.

Table 4-8. Store Bytes Short Instruction

Instruction Name	Mnemonic Name	Operands
Store Bytes Short	STBYS,cmplt	r,d(s,b)

Table 4-9. Store Bytes Short Completers

cmplt	Description
,B	Beginning case, no modify
,E	Ending case, no modify
,B,M	Beginning case, modify base
,E,M	Ending case, modify base

Immediate Instructions

An immediate value is loaded into a general register. Values are computed from either a 21-bit immediate value, a 21-bit immediate value plus a general register, or a 14-bit immediate value plus a base register.

Table 4-10. Immediate Instructions

Instruction Name	Mnemonic Name	Operands
Load Immediate Left	LDIL	i,t
Add Immediate Left *	ADDIL	i,r
Load Offset	LDO	d(b),t
Load Immediate **	LDI	i,t

* The result of the ADDIL operation is placed in general register one.

** The LDI pseudo-instruction generates an LDO i(0),t instruction.

Branch Instructions

Branch instructions can be conditional or unconditional. Unconditional branches are used to alter the control flow of a program. Conditional branches perform a function and then branch depending upon whether a specified condition has been met.

NOTE

When coding branch instructions, including those with nullification specified, attention should be given to the *following* instruction in the source code. All branch instructions consider the instruction that follows to be in its *delay slot*. An NOP pseudo-operation may be used to fill the delay slot when there is no other useful work to be performed.

Unconditional Branch Instructions

Unconditional branch instructions are used to make control transfers, procedure calls, and procedure returns. Targets for unconditional branches are either local or external. Unconditional *local* branch instructions are used for control transfers within a space (intraspace). Unconditional *external* branch instructions are used for control transfers into another space (interspace).

Table 4-11. Unconditional Branch Instructions

Instruction Name	Mnemonic Name	Operands
Branch External	BE,n	wd(sr,b)
Branch and Link External	BLE,n	wd(sr,b)
Branch and Link	BL,n	target,t
Branch and Link Register	BLR,n	x,t
Branch Vectored	BV,n	x(b)
Gateway	GATE,n	target,t
Unconditional Branch *	B,n	target

* The B pseudo-instruction generates a BL instruction with the link register t equal to GR0.

Conditional Branch Instructions

Local conditional branches include the following instructions:

- Move and Branch
- Compare and Branch
- Add and Branch
- Branch on Bit

A conditional branch is considered *taken* if the condition specified by the condition completer is met by the operands, and/or is met by the results of the function performed.

All conditional branches can be coded with a nullification completer. If nullification is specified, the instruction following either a *taken forward branch* or a *failing backward branch* is nullified. If nullification is not specified, the following instruction is executed before the branch target, especially in the case where a branch instruction occupies the delay slot. See the *Precision Architecture and Instruction Reference Manual* for the definition of "following instruction".

A forward branch is one to an address that is greater than the address of the instruction in the delay slot of the branch. A backward branch is one to an address that is less than or equal to the address of the delay slot. Figure 4-1 identifies which branches are forward branches and which branches are backward branches.

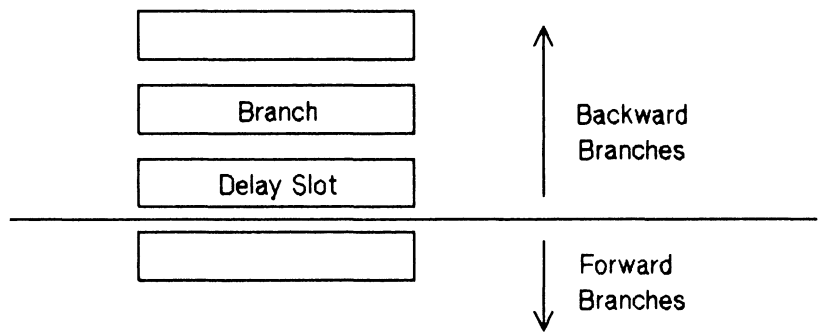


Figure 4-1. Branch Descriptions

Move and Branch Instructions

A general register or an immediate value is copied to a general register. The value copied is used to determine if the specified condition is met.

Table 4-12. Move and Branch Instructions

Instruction Name	Mnemonic Name	Operands
Move and Branch	MOVB, cond, n	r1,r2,target
Move Immediate and Branch	MOVIB, cond, n	i,r,target

Table 4-13. Move and Branch Conditions

cond	Description
	never
=	all bits are equal to 0
<	leftmost bit is equal to 1
OD	rightmost bit is equal to 1
TR	always
<>	at least one bit is equal to 1
>=	leftmost bit is equal to 0
EV	rightmost bit is equal to 0

Compare and Branch Instructions

A general register or an immediate value is compared against a general register. The operands and/or the result of the comparison are used to determine whether the specified condition is met. The COMBT, COMIBT, COMBF and COMIBF instructions use only the non-negated completer conditions (never, =, <, <=, <<, <<=, SV, OD). The COMB and COMIB pseudo-instructions use any of the condition completers.

Table 4-14. Compare and Branch Instructions

Instruction Name	Mnemonic Name	Operands
Compare and Branch If True	COMBT,cond,n	r1,r2,target
Compare Immediate and Branch If True	COMIBT,cond,n	i,r,target
Compare and Branch If False	COMBF,cond,n	r1,r2,target
Compare Immediate and Branch If False	COMIBF,cond,n	i,r,target
Compare and Branch *	COMB,cond,n	r1,r2,target
Compare Immediate and Branch **	COMIB,cond,n	i,r,target

* The COMB pseudo-instruction generates either a COMBT or COMBF instruction depending on the condition.

** The COMIB pseudo-instruction generates either a COMIBT or COMIBF instruction depending on the condition.

Table 4-15. Compare and Branch Conditions

cond	Description	
	never	Non-negated
=	opd1 is equal to opd2	
<	opd1 is less than opd2 (signed)	
<=	opd1 is less than or equal to opd2 (signed)	
<<	opd1 is less than opd2 (unsigned)	
<<=	opd1 is less than or equal to opd2 (unsigned)	
SV	opd1 minus opd2 results in overflow (signed)	
OD	result of opd1 minus opd2 is odd	
TR	always	Negated
<>	opd1 is less than or greater than opd2	
>=	opd1 is greater than or equal to opd2 (signed)	
>	opd1 is greater than opd2 (signed)	
>>=	opd1 is greater than or equal to opd2 (unsigned)	
>>	opd1 is greater than opd2 (unsigned)	
NSV	opd1 minus opd2 results in no overflow (signed)	
EV	result of opd1 minus opd2 is even	

Add and Branch Instructions

A general register or an immediate value is added to a general register. The operands and/or the *results* of the addition are used to determine whether the specified condition is met. The ADDBT, ADDIBT, ADDBF and ADDIBF instructions use only the non-negated completer conditions (never, =, <, <=, NUV, ZNV, SV, OD). The ADDB and ADDIB pseudo-instructions uses any of the condition completers.

Table 4-16. Add and Branch Instructions

Instruction Name	Mnemonic Name	Operands
Add and Branch If True	ADDBT,cond,n	r1,r2,target
Add Immediate and Branch If True	ADDIBT,cond,n	i,r,target
Add and Branch If False	ADDBF,cond,n	r1,r2,target
Add Immediate and Branch If False	ADDIBF,cond,n	i,r,target
Add and Branch *	ADDB,cond,n	r1,r2,target
Add Immediate and Branch **	ADDIB,cond,n	i,r,target

* The ADDB pseudo instruction generates either a ADDBT or ADDBF instruction, depending on the condition.

** The ADDIB pseudo-instruction generates either a ADDIBT or ADDIBF instruction, depending on the condition.

Table 4-17. Add and Branch Conditions

cond	Description	
	never	Non-negated
=	opd1 is equal to -opd2	
<	opd1 is less than -opd2 (signed)	
<=	opd1 is less than or equal to -opd2 (signed)	
NUV	opd1 + opd2 < 2^{32} (no unsigned overflow)	
ZNV	opd1 + opd2 < 2^{32} or opd1 + opd2 = 0	
SV	opd1 plus opd2 results in overflow (signed)	
OD	result of opd1 plus opd2 is odd	
TR	always	Negated
<>	opd1 is not equal to -opd2	
>=	opd1 is greater than or equal to -opd2 (signed)	
>	opd1 is greater than -opd2 (signed)	
UV	opd1 + opd2 > = 2^{32} (unsigned overflow)	
VNZ	opd1 + opd2 > 2^{32} and opd1 + opd2 not = 0	
NSV	opd1 plus opd2 results in no overflow (signed)	
EV	result of opd1 plus opd2 is even	

Branch on Bit Instructions

The bit specified, either as an immediate or in the shift-amount-register (SAR,CR11), of a general register is tested. The result of the test is used to determine whether the specified condition is met.

Table 4-18. Branch on Bit Instructions

Instruction Name	Mnemonic Name	Operands
Branch on Variable Bit	BVB,cond,n	r,target
Branch on Bit	BB,cond,n	r,p,target

Table 4-19. Branch on Bit Conditions

cond	Description
<	Bit specified is equal to 1
>=	Bit specified is equal to 0

Computational Instructions

Computational instructions are composed of the following:

- Add
- Shift and Add
- Subtract
- Compare and Clear
- Divide Step
- Logical
- Unit
- Shift
- Extract
- Deposit

The instructions in this group are organized by the type of condition that can be specified.

After the computational instructions perform a function, the operands and/or the "results" of the function being performed are used to determine whether the specified condition is met. If the condition is met, the *following* instruction is nullified.

Some of the computational instructions cause interrupt traps to occur based on overflow and/or condition detection.

Refer to the "Instruction Operands" section earlier in this chapter for a description of the instruction operands.

Add Instructions

Two general registers, or an immediate value and a general register, are added together and the result is stored in a general register. The operands and/or the *results* of the addition are used to determine whether the specified condition is met. If the specified condition is met, the following instruction is nullified, except for ADDIT and ADDITO where a trap is taken.

Table 4-20. Add Instructions

Instruction Name	Mnemonic Name	Operands
Add	ADD,cond	r1,r2,t
Add Logical	ADDL,cond	r1,r2,t
Add and Trap on Overflow	ADDO,cond	r1,r2,t
Add with Carry	ADDC,cond	r1,r2,t
Add with Carry and Trap on Overflow	ADDCO,cond	r1,r2,t
Add to Immediate and Trap on Condition	ADDIT,cond	i,r,t
Add to Immediate and Trap on Condition or Overflow	ADDITO,cond	i,r,t
Add to Immediate	ADDI,cond	i,r,t
Add to Immediate and Trap on Overflow	ADDIO,cond	i,r,t

Table 4-21. Add Conditions

cond	Description	
	never	Non-negated
=	opd1 is equal to -opd2	
<	opd1 is less than -opd2 (signed)	
<=	opd1 is less than or equal to -opd2 (signed)	
NUV	opd1 + opd2 < 2^{32} (no unsigned overflow)	
ZNV	opd1 + opd2 < 2^{32} or opd1 + opd2 = 0	
SV	opd1 plus opd2 results in overflow (signed)	
OD	result of opd1 plus opd2 is odd	
TR	always	Negated
<>	opd1 is not equal to -opd2	
>=	opd1 is greater than or equal to -opd2 (signed)	
>	opd1 is greater than -opd2 (signed)	
UV	opd1 + opd2 > = 2^{32} (unsigned overflow)	
VNZ	opd1 + opd2 > 2^{32} and opd1 + opd2 not = 0	
NSV	opd1 plus opd2 results in no overflow (signed)	
EV	result of opd1 plus opd2 is even	

Shift and Add Instructions

A general register is left-shifted and added to a general register, and the result is stored in a general register. The operands and/or the results are used to determine whether the specified condition is met. The following instruction is nullified if the specified condition is met. A signed overflow condition can be detected if any of the bits shifted out are different from the resulting left-most bit after the shift.

Table 4-22. Shift and Add Instructions

Instruction Name	Mnemonic Name	Operands
Shift One and Add	SH1ADD,cond	r1,r2,t
Shift One and Add Logical	SH1ADDL,cond	r1,r2,t
Shift One, Add and Trap on Overflow	SH1ADDO,cond	r1,r2,t
Shift Two and Add	SH2ADD,cond	r1,r2,t
Shift Two and Add Logical	SH2ADDL,cond	r1,r2,t
Shift Two, Add and Trap on Overflow	SH2ADDO,cond	r1,r2,t
Shift Three and Add	SH3ADD,cond	r1,r2,t
Shift Three and Add Logical	SH3ADDL,cond	r1,r2,t
Shift Three, Add and Trap on Overflow	SH3ADDO,cond	r1,r2,t

Table 4-23. Shift and Add Conditions

cond	Description	
	never	Non-negated
=	opd1 is equal to -opd2	
<	opd1 is less than -opd2 (signed)	
<=	opd1 is less than or equal to -opd2 (signed)	
NUV	opd1 + opd2 < 2^{32} (no unsigned overflow)	
ZNV	opd1 + opd2 < 2^{32} or opd1 + opd2 = 0	
SV	opd1 plus opd2 results in overflow (signed)	
OD	result of opd1 plus opd2 is odd	
TR	always	Negated
<>	opd1 is not equal to -opd2	
>=	opd1 is greater than or equal to -opd2 (signed)	
>	opd1 is greater than -opd2 (signed)	
UV	opd1 + opd2 >= 2^{32} (unsigned overflow)	
VNZ	opd1 + opd2 > 2^{32} and opd1 + opd2 not = 0	
NSV	opd1 plus opd2 results in no overflow (signed)	
EV	result of opd1 plus opd2 is even	

Subtract Instructions

A general register is subtracted from an immediate value or a general register and the result is stored in a general register. The operands and/or the results of the subtraction are used to determine whether the specified condition is met. If the specified condition is met, the following instruction is nullified, except for SUBT and SUBTO where a trap is taken.

Table 4-24. Subtract Instructions

Instruction Name	Mnemonic Name	Operands
Subtract	SUB,cond	r1,r2,t
Subtract and Trap on Condition	SUBT,cond	r1,r2,t
Subtract with Borrow	SUBB,cond	r1,r2,t
Subtract and Trap on Overflow	SUBO,cond	r1,r2,t
Subtract and Trap on Condition or Overflow	SUBTO,cond	r1,r2,t
Subtract with Borrow and Trap on Overflow	SUBBO,cond	r1,r2,t
Subtract from Immediate	SUBI,cond	i,r,t
Subtract from Immediate and Trap on Overflow	SUBIO,cond	i,r,t

Table 4-25. Subtract Conditions

cond	Description	
	never	Non-negated
=	opd1 is equal to opd2	
<	opd1 is less than opd2 (signed)	
<=	opd1 is less than or equal to opd2 (signed)	
<<	opd1 is less than opd2 (unsigned)	
<<=	opd1 is less than or equal to opd2 (unsigned)	
SV	opd1 minus opd2 results in overflow (signed)	
OD	result of opd1 minus opd2 is odd	
TR	always	Negated
<>	opd1 is less than or greater than opd2	
>=	opd1 is greater than or equal to opd2 (signed)	
>	opd1 is greater than opd2 (signed)	
>>=	opd1 is greater than or equal to opd2 (unsigned)	
>>	opd1 is greater than opd2 (unsigned)	
NSV	opd1 minus opd2 results in no overflow (signed)	
EV	result of opd1 minus opd2 is even	

Compare and Clear Instructions

An immediate value or a general register is compared against another general register and zero is loaded into a third general register. The operands of the comparison are used to determine whether the specified condition is met. If the specified condition is met, the following instruction is nullified.

Table 4-26. Compare and Clear Instructions

Instruction Name	Mnemonic Name	Operands
Compare and Clear	COMCLR,cond	r1,r2,t
Compare Immediate and Clear	COMICLR,cond	i,r,t

Table 4-27. Compare and Clear Conditions

cond	Description	
	never	Non-negated
=	opd1 is equal to opd2	
<	opd1 is less than opd2 (signed)	
<=	opd1 is less than or equal to opd2 (signed)	
<<	opd1 is less than opd2 (unsigned)	
<<=	opd1 is less than or equal to opd2 (unsigned)	
SV	opd1 minus opd2 results in overflow (signed)	
OD	result of opd1 minus opd2 is odd	
TR	always	Negated
<>	opd1 is less than or greater than opd2	
>=	opd1 is greater than or equal to opd2 (signed)	
>	opd1 is greater than opd2 (signed)	
>>=	opd1 is greater than or equal to opd2 (unsigned)	
>>	opd1 is greater than opd2 (unsigned)	
NSV	opd1 minus opd2 results in no overflow (signed)	
EV	result of opd1 minus opd2 is even	

Divide Step Instruction

A divide step operation is performed using two general registers; the intermediate result is stored in a general register. The operands and/or the results of the divide step operation are used to determine whether the specified condition is met. The divide step can be used with either the add or subtract conditions depending upon the stage of the operation.

Table 4-28. Divide Step Instruction

Instruction Name	Mnemonic Name	Operand
Divide Step	DS,cond	r1,r2,t

See Table 4-21 or 4-25 for the add or subtract conditions, respectively.

Logical Instructions

A logical operation is performed on two general registers and the result is stored in a general register. The result of the logical operation is used to determine whether the specified condition is met. The following instruction is nullified if the specified condition is met.

Table 4-29. Logical Instructions

Instruction Name	Mnemonic Name	Operands
And Complement	ANDCM,cond	r1,r2,t
And	AND,cond	r1,r2,t
Inclusive Or	OR,cond	r1,r2,t
Exclusive Or	XOR,cond	r1,r2,t
Copy *	COPY	r,t
No Operation **	NOP	

* The COPY pseudo-instruction generates the OR r,0,t instruction.

** The NOP pseudo-instruction generates the OR 0,0,0 instruction.

Table 4-30. Logical Conditions

cond	Description	
	never	Non-negated
=	all bits are equal to 0	
<	leftmost bit is equal to 1	
<=	leftmost bit is equal to 1, or all bits are equal to 0	
OD	rightmost bit is equal to 1	
tr	always	Negated
<>	at least one bit is equal to 1	
>=	leftmost bit is equal to 0	
>	leftmost bit is equal to 0, and at least one bit is equal to 1	
EV	rightmost bit is equal to 0	

Unit Instructions

One or two general registers are operated upon and the result is stored in a general register. The results of the operation are used to determine whether a specified condition is met. If the specified condition is met, the following instruction is nullified, except for UADDCMT where a trap is taken.

Table 4-31. Unit Instructions

Instruction Name	Mnemonic Name	Operands
Unit Exclusive Or	UXOR,cond	r1,r2,t
Unit Add Complement	UADDCM,cond	r1,r2,t
Unit Add Complement and Trap on Condition	UADDCMT,cond	r1,r2,t
Intermediate Decimal Correct	IDCOR,cond	r,t
Decimal Correct	DCOR,cond	r,t

Table 4-32. Unit Conditions

cond	Description	
	never	Non-negated
SBZ	some byte zero	
SHZ	some halfword zero	
SCD	some BCD digit carry	
SBC	some byte carry	
SHC	some halfword carry	
TR	always	Negated
NBZ	all bytes nonzero	
NHZ	all halfwords nonzero	
NDC	no BCD digit carry	
NBC	no byte carry	
NHC	no halfword carry	

Shift, Extract, and Deposit Instructions

A shift, extract, or deposit operation is performed and the result is stored in a general register. The results of the operation are used to determine whether a specified condition is met. A variable operation may be performed using the shift-amount-register (SAR, CR 11).

The shift instructions use two general registers to perform a double shift. The amount to be shifted comes either from the operand list or from the SAR.

The extract instructions isolate a field of the specified length in a general register, right shift to position the field, and store the result in a general register. The result may be stored as a sign extended value. The ending position of the field to be extracted may be specified as an operand or from the SAR.

The deposit instructions use an immediate value or a general register to isolate a field of the specified length, left shift to position the field, and merge the result into a general register. The result may be merged with the existing bits of the general register or zeros. The ending position of the result field may be specified as an operand or from the SAR.

Table 4-33. Shift, Extract, and Deposit Instructions

Instruction Name	Mnemonic Name	Operands
Variable Shift Double	VSHD,cond	r1,r2,t
Shift Double	SHD,cond	r1,r2,count,t
Variable Extract Unsigned	VEXTRU,cond	r,len,t
Variable Extract Signed	VEXTRS,cond	r,len,t
Extract Unsigned	EXTRU,cond	r,p,len,t
Extract Signed	EXTRS,cond	r,p,len,t
Variable Deposit	VDEP,cond	r,len,t
Zero and Deposit	ZDEP,cond	r,p,len,t
Deposit	DEP,cond	r,p,len,t
Zero and Variable Deposit	ZVDEP,cond	r,len,t
Zero and Variable Deposit Immediate	ZVDEPI,cond	i,len,t
Variable Deposit Immediate	VDEPI,cond	i,len,t
Zero and Deposit Immediate	ZDEPI,cond	i,p,len,t
Deposit Immediate	DEPI,cond	i,p,len,t

Table 4-34. Shift, Extract, and Deposit Conditions

cond	Description
	never
=	all bits are equal to 0
<	leftmost bit is equal to 1
OD	rightmost bit is equal to 1
TR	always
<>	at least one bit is equal to 1
>=	leftmost bit is equal to 0
EV	rightmost bit is equal to 0

System Control Instructions

System control instructions provide special register moves, system mask control, return from interruption, hash address computation, probe access rights, memory management operations, and implementation-dependent functions.

Some of the memory management instructions use an optional completer to allow the base register to be modified after the address has been formed.

Most of these instructions are executed by code that only runs at the most privileged level.

Table 4-35. System Control Completers

complt	Address Formation
,M	Base Register Modification
none specified	No Base Register Modification

Table 4-36. System Control Instructions

Instruction Name	Mnemonic Name	Operands
Break	BREAK	i1, i2
Return from Interruption	RFI	
Set System Mask	SSM	i, t
Reset System Mask	RSM	i, t
Move To System Mask	MTSM	r
Load Space Identifier	LDSID	(s, b), t
Move To Space Register	MTSP	r, sr
Move To Control Register	MTCTL	r, t
Move To Shift Amount Register *	MTSAR	r
Move From Space Register	MFSP	sr, t
Move From Control Register	MFCTL	r, t
Synchronize Caches	SYNC	
Probe Read Access	PROBER	(s, b), r, t
Probe Read Access Immediate	PROBERI	(s, b), i, t
Probe Write Access	PROBEW	(s, b), r, t
Probe Write Access Immediate	PROBEWI	(s,b), i, t
Load Physical Address	LPA, cmplt	x (s, b), t
Load Hash Address	LHA, cmplt	x (s, b), t
Purge Data Translation Lookaside Buffer	PDTLB, cmplt	x (s, b)
Purge Instruction Translation Lookaside Buffer	PITLB, cmplt	x (sr, b)
Purge Data Translation Lookaside Buffer Entry	PDTLBE, cmplt	x (s, b)

* The MTSAR pseudo-instruction generates an MTCTL r,11 instruction.

Table 4-36. System Control Instructions (Continued)

Instruction Name	Mnemonic Name	Operands
Purge Instruction Translation Lookaside Buffer Entry	PITLBE, cmplt	x (sr, b)
Insert Data Translation Lookaside Buffer Address	IDTLBA	r, (s, b)
Insert Instruction Translation Lookaside Buffer Address	IITLBA	r, (sr, b)
Insert Data Translation Lookaside Buffer Address	IDTLBP	r, (s, b)
Insert Instruction Translation Lookaside Buffer Protection	IITLBP	r, (sr, b)
Purge Data Cache	PDC, cmplt	x (s, b)
Flush Data Cache	FDC, cmplt	x (s, b)
Flush Instruction Cache	FIC, cmplt	x (sr, b)
Flush Data Cache Entry	FDCE, cmplt	x (s, b)
Flush Instruction Cache Entry	FICE, cmplt	x (sr, b)
Diagnose	DIAG	i

Assist (Coprocessor) Instructions

Coprocessor instructions are used to perform data manipulations on attached special purpose processors, while incurring minimum overhead in execution rate. Each instruction must specify a unit identifier between zero and seven to select a coprocessor. The coprocessor instructions include:

- Coprocessor Operation
- Indexed Load and Store
- Short Displacement Load and Store
- The Floating-Point Instruction Set

Coprocessor Operation Instruction

You must explicitly prescribe the binary value for the desired operation as well as the unit number of the coprocessor.

This instruction uses only completers. Operands are not accepted.

Table 4-37. Coprocessor Operation Instruction

Instruction Name	Mnemonic Name
Coprocessor Operation	COPR,uid,sop,n

Table 4-38. Coprocessor Operation Completers

	Completer Description
uid	(0 - 7) Select coprocessor unit number
sop	Provides a 22 bit value that the assembler encodes in the instruction
n	If the value is one and the coprocessor condition is satisfied, the following instruction is nullified.

NOTE

Identify coprocessor is encoded as COPR,uid,0.

Coprocessor Indexed Load and Store Instructions

A value is transferred between memory and a register. The address is formed by the addition of the index register to the base register. Optionally, the index value may be shifted to reflect an address corresponding to the size of the value being transferred and the base register may be modified after the address has been formed.

Table 4-39. Coprocessor Indexed Load and Store Instructions

Instruction Name	Mnemonic Name	Operands
Coprocessor Load Word Indexed	CLDWX,uid,complt	x(s,b),t
Coprocessor Load Doubleword Indexed	CLDDX,uid,complt	x(s,b),t
Coprocessor Store Word Indexed	CSTWX,uid,complt	r,x(s,b)
Coprocessor Store Doubleword Indexed	CSTDX,uid,complt	r,x(s,b)

Table 4-40. Coprocessor Indexed Load and Store Completers

complt	Description
,S	Shift Index by Data Size, No Base Register Modification
,M	No Index Shift Base Register Modification
,S,M or ,SM	Shift Index by Data Size Base Register Modification
none specified	No Index Shift, No Base Register Modification

Coprocessor Short Displacement Load and Store Instructions

A value is transferred between memory and a floating-point register. The address is formed by the addition of the short displacement to the base register. Optionally, the base register may be modified either before or after the address has been formed.

Table 4-41. Coprocessor Short Displacement Load and Store Instructions

Instruction Name	Mnemonic Name	Operands
Coprocessor Load Word Short	CLDWS,uid,complt	d(s,b),t
Coprocessor Load Doubleword Short	CLDDS,uid,complt	d(s,b),t
Coprocessor Store Word Short	CSTWS,uid,complt	r,d(s,b)
Coprocessor Store Doubleword Short	CSTDS,uid,complt	r,d(s,b)

Table 4-42. Coprocessor Short Displacement Load and Store Completers

complt	Description
,MB	Modify Before
,MA	Modify After
none specified	No Base Register Modification

Floating-Point Instructions

Floating-point instructions are used to perform floating-point arithmetic. The instructions are either executed directly by a floating-point assist coprocessor or by floating-point emulation software. The floating-point instructions include:

- Indexed Load and Store
- Short Displacement Load and Store
- Operations
- Compare and Test

Floating-Point Indexed Load and Store Instructions

A value is transferred between memory and a floating-point register. The address is formed by the addition of the index register to the base register. Optionally, the index value may be shifted to reflect an address corresponding to the size of the value being transferred and the base register may be modified after the address has been formed.

Table 4-43. Floating-Point Indexed Load and Store Instructions

Instruction Name	Mnemonic Name	Operands
Floating-point Load Word Indexed	FLDWX,cmplt	x(s,b),t
Floating-point Load Doubleword Indexed	FLDDX,complt	x(s,b),t
Floating-point Store Word Indexed	FSTWX,complt	r,x(s,b)
Floating-point Store Doubleword Indexed	FSTDX,complt	r,x(s,b)

Table 4-44. Floating-Point Indexed Load and Store Completers

complt	Description
,S	Shift Index by Data Size, No Base Register Modification
,M	No Index Shift Base Register Modification
",S,M" or ,SM	Shift Index by Data Size Base Register Modification
none specified	No Index Shift, No Base Register Modification

Floating-Point Short Displacement Load and Store Instructions

A value is transferred between memory and a floating-point register. The address is formed by the addition of the short displacement to the base register. Optionally, the base register may be modified either before or after the address has been formed.

Table 4-45. Floating-Point Short Displacement Load and Store Instructions

Instruction Name	Mnemonic Name	Operands
Floating-point Load Word Short	FLDWS,cmplt	d(s,b),t
Floating-point Load Doubleword Short	FLDDS,complt	d(s,b),t
Floating-point Store Word Short	FSTWS,cmplt	r,d(s,b)
Floating-point Store Doubleword Short	FSTDS,cmplt	r,d(s,b)

Table 4-46. Floating-Point Short Displacement Load and Store Completers

complt	Description
,MB	Modify Before
,MA	Modify After
none specified	No Base Register Modification

Floating-Point Operation Instructions

Floating-point operation instructions include math functions and data format conversions.

Values from one or two floating-point registers are used to perform an operation and the result is stored in a floating-point register.

Table 4-47. Floating-Point Operation Instructions

Instruction Name	Mnemonic Name	Operands
Floating-point Add	FADD,fmt	r1,r2,t
Floating-point Subtract	FSUB,fmt	r1,r2,t
Floating-point Multiply	FMPY fmt	r1,r2,t
Floating-point Divide	FDIV,fmt	r1,r2,t
Floating-point Square Root	FSQRT,fmt	r,t
Floating-point Absolute Value	FABS,fmt	r,t
Floating-point Remainder	FREM,fmt	r1,r2,t
Floating-point Round to Integer	FRND,fmt	r,t
Floating-point Copy	FCPY,fmt	r,t
Floating-point Convert from Fixed-point to Floating-point	FCNVXF,sf,df	r,t
Floating-point Convert from Floating-point to Fixed-point	FCNVFX,sf,df	r,t
Floating-point Convert from Floating-point to Floating-point	FCNVFF,sf,df	r,t
Floating-point Convert from Floating-point to Fixed-point and Truncate	FCNVFXT,sf,df	r,t

Table 4-48. Floating-Point Format Completers

fmt/df/sf	Format or Size Specified
,SGL or <none>	single (32-bit)
,DBL	double (64-bit)
,QUAD	quad (128-bit)

Floating-Point Compare and Test Instructions

A floating-point register is compared against a floating-point register using the format specified. The operands and/or the result of the comparison are used to determine whether the specified condition is met. If the condition is met, the C bit of the floating-point status register is set; otherwise, it is cleared.

A floating-point test instruction is used to nullify the following instruction based on the state of the C bit. If the C bit is set, the next instruction is skipped; otherwise, it is executed.

Table 4-49. Floating-Point Compare and Test Instructions

Instruction Name	Mnemonic Name	Operands
Floating-point Compare	FCMP,fmt,cond	r1,r2
Floating-point Test	FTEST	

For completer format, refer to Table 4-48.

Table 4-50. Floating-Point Compare Conditions

Cond	Relations				Invalid Operation Except if Unordered
	Greater Than	Less Than	Equal To	Unordered	
false? false ? !<=>	F F F F	F F F F	F F F F	F F T T	F T F T
= =T ?= !<>	F F F F	F F F F	T T T T	F F T T	F T F T
!?>= < ?< !>=	F F F F	T T T T	F F F F	F F T T	F T F T
!?> <= ?<= !>	F F F F	T T T T	T T T T	F F T T	F T F T
!?<= > ?> !<=	T T T T	F F F F	F F F F	F F T T	F T F T
!?< >= ?>= !=	T T T T	F F F F	T T T T	F F T T	F T F T
!?= <> != !=T	T T T T	T T T T	F F F F	F F T T	F T F T
!? <=> true? true	T T T T	T T T T	T T T T	F F T T	F T F T

Pseudo Instructions

Table 4-51 lists the commonly used pseudo instructions that are abbreviations for machine instructions. These pseudo instructions can be used instead of machine instructions.

Table 4-51. Pseudo-Instructions

Pseudo-Instruction Code		Standard Format		
B	w	BL	w, 0	
COMB,cond,n	r1,r2,target	COMBT,cond,n COMBF,cond,n	r1,r2,target r1,r2,target	*
COMIB,cond,n	r1,r2,target	COMIBF,cond,n COMIBT,cond,n	r1,r2,target r1,r2,target	*
ADDB,cond,n	r1,r2,target	ADDBT,cond,n ADDBF,cond,n	r1,r2,target r1,r2,target	*
ADDIB,cond,n	r1,r2,target	ADDIBT,cond,n ADDIBF,cond,n	r1,r2,target r1,r2,target	*
COPY	r,t	OR	r,0,t	
LDI	i,t	LDO	i(0),t	
NOP		OR	0,0,0	
MTSAR	r	MTCTL	r,11	

* The supplied completer determines the actual instruction that the Assembler uses in the conditional branch. You can use the completer for any arithmetic condition or its negation with the pseudo-instructions COMB, COMIB, ADDB, and ADDIB. The false form of the corresponding actual instruction is assembled if the completer specifies the negation of a condition. Otherwise, the true form is used.

Chapter 5
Programming Examples

This chapter consists of five programming examples in the assembly language. The first three examples show typical assembly language code sequences; the last two examples show the correspondence between C, a higher-level programming language, and assembly language.

Example 1 calculates the highest bit position set in a passed parameter. A binary search is used to enhance performance.

Example 2 copies bytes from a source location to a destination location. Both locations and the number of bytes to copy are passed in as parameters.

Example 3 uses Divide Step to divide a 64-bit signed dividend by a 32-bit signed divisor.

Example 4 uses a procedure call from a C program to the Assembler to verify that the program is passing the correct argument.

Example 5 shows a C program that generates assembly code to call `printf`.

Binary Search for Highest Bit Position

In Example 1, the Shift Double and Extract Unsigned instructions are used to implement a binary search. Bits shifted into general register 0 are effectively discarded.

Example 1

```
        .CODE
        .EXPORT post
;
; This procedure calculates the highest bit position
; set in the word passed in as the first argument.
; If passed parameter is non-zero, the algorithm
; starts by assuming it is one.
; A binary search for a set bit is then used
; to enhance performance.
;
; The calculated bit position is returned to the caller.
;
        .PROC
post
        .CALLINFO SAVE_RP
        .ENTER
        COMB,=,N    %r0,%arg0,all_zeros    ; No bits set
        LDI         31,%ret0               ; assume 2 to the 0 power
;
; if extracted bits non-zero, fall thru to change assumption
; else set up 16 low order bits and keep assumption
;
        EXTRU,<>    %arg0,15,16,%r0        ; check 16 high order bits
        SHD,TR      %arg0,%r0,16,%arg0     ; left shift arg0 16 bits
        ADDI        -16,%ret0,%ret0        ; assume 2 to the 16 power
;
; if extracted bits non-zero, fall thru to change assumption
; else set up 8 low order bits and keep assumption
;
        EXTRU,<>    %arg0,7,8,%r0          ; check next 8 high order bits
        SHD,TR      %arg0,%r0,24,%arg0     ; left shift arg0 8 bits
        ADDI        -8,%ret0,%ret0         ; assume 8 higher power of 2
;
; if extracted bits non-zero, fall thru to change assumption
; else set up 4 low order bits and keep assumption
;
        EXTRU,<>    %arg0,3,4,%r0          ; check next 4 high order bits
        SHD,TR      %arg0,%r0,28,%arg0     ; left shift arg0 4 bits
        ADDI        -4,%ret0,%ret0         ; assume 4 higher power of 2
```

```
;
; if extracted bits non-zero, fall thru to change assumption
; else set up 2 low order bits and keep assumption
;
       EXTRU,<>      %arg0,1,2,%r0          ; check next 2 high order bits
       SHD,TR        %arg0,%r0,30,%arg0     ; left shift arg0 2 bits
       ADDI          -2,%ret0,%ret0         ; assume 2 higher power of 2
;
; if extracted bit is zero, fall thru and keep assumption
; else make conclusion
;
       EXTRU,=       %arg0,0,1,%r0          ; check next bit
       ADDI          -1,%ret0,%ret0         ; next higher power of 2
       B,N           tally

all_zeros
       LDI           -1,%ret0

tally
       .LEAVE
       .PROCEND
```

Copying a String

Example 2 contains a section of assembly code that moves a byte string of arbitrary length to an arbitrary byte address.

Example 2

```
;
;
; The routine rflect copies bytes from the source location
; to the destination location.
;
; The first parameter is the source address and the second
; parameter is the destination address.
;
; The third parameter is the number of bytes to copy.
;
; For performance, larger chunks of bytes are handled differently.
;
;
        .CODE
        .EXPORT     rflect,ENTRY
rflect
        .PROC
        .CALLINFO   ENTRY_GR=6,SAVE_RP
        .ENTER
        COMB,=,N    %arg2,%r0,fallout       ; done, count is zero
        COMB,<,N    %arg2,%r0,choke         ; caller error, neg count
        OR          %arg0,%arg1,%r6         ; source and dest
        OR          %r6,%arg2,%r6           ; count
        EXTRU,=     %r6,31,2,%r0            ; 2 low order bits = 0?
        B,N         onebyte                 ; yes, skip this branch
        ADDIBT,<,N  -16,%arg2,chekchunk     ; no, skip chunkify if count<0

chunkify
        LDWN        16(%arg0),%r6           ; word 1- > temp1
                                            ; point ahead 4 words  in source
        LDW         -12(%arg0),%r5          ; place mark 3 wds back- >temp2
        LDW         -8(%arg0),%r4           ; place mark 2 wds back- >temp3
        LDW         -4(%arg0),%r3           ; place mark 1 wds back- >temp4
        STW         %r5,4(%arg1)            ; dest wd 2 <-temp2
        STWM        %r6,16(%arg1)           ; dest wd 1 <-temp1
                                            ; point ahead 4 words in dest
        STW         %r4,-8(%arg1)           ; dest wd 3 <-temp3
        ADDIBF,<    -16,%arg2,chunkify      ; loop if count > 0
        STW         %r3,-4(%arg1)           ; dest wd 2 <-temp1
chekchunk
        ADDIBT,<    12,%arg2,exeunt         ; go if count < -12
        COPY        %r0,%ret0               ; clear rtnval
```

```
subchunk
        LDWS,MA     4(%arg0),%r6          ; word- >temp1
                                          ; point ahead 4 bytes in source
        ADDIBF,<    -4,%arg2,subchunk     ; go if count<4
        STWS,MA     %r6,4(%arg1)          ; dest< -temp1
                                          ; point ahead 4 bytes in dest

        B           exeunt                ; all done
        COPY        %r0,%ret0             ; clear rtnval

onebyte
        LDBS,MA     1(%arg0),%r6          ; temp1 < -byte,bump src pointer
onemore
        STBS,MA     %r6,1(%arg1)          ; dest<-temp1,bump dest pointer
        ADDIBF,=,N  -1,%arg2,onemore      ; decrement count
                                          ; compare for 0.
        LDBS,MA     1(%arg0),%r6          ; delay slot
                                          ; temp1 <-byte,bump src pointer
fallout
        B           exeunt
        COPY        %r0,%ret0

choke
        B           exeunt
        LDI         14,%ret0

exeunt
        .LEAVE
        .PROCEND
```

Dividing a Double-Word Dividend

Example 3 contains the code sequence to divide a 64-bit signed dividend by a 32-bit signed divisor using the DS (Divide Step) instruction. Table 5-1 lists the registers that this program uses.

Example 3

```
start
            MOVB,>=         dvdu,rem,check_mag  ; Move upper dividend
                                                ;   check for < 0
            ADD             0,dvdl,quo          ; Move lower dividend always
            SUB             0,quo,quo           ; Get absolute value of
            SUBB            0,rem,rem           ;   the dividend in rem,quo
check_mag
            SUBT,=          0,dvr,tp            ; Check 0, clear carry,
                                                ;   negate the divisor
                                                ;   and trap if dvr = 0
            DS              0,tp,0;             ; Set V-bit to the complement
                                                ;   of the divisor sign
            ADD             quo,quo,quo         ; Shift msb bit into carry
            DS,<<           rem,dvr,rem         , 1st divide step, if carry
                                                ;   out msb of quotient = 0
            B,n             min_ovfl            ; Abs(quotient) > 2**31
                                                ;   deal with elsewhere
            ADDC            quo,quo,quo         ; Shift quo with/into carry
            DS              rem,dvr,rem         ; 2nd divide step
            ; ...
            ;repeat divide step sequence
            ; ...
            ADDC            quo,quo,quo         ; Shift quo with/into carry
            DS              rem,dvr,rem         ; 31st divide step
            ADDC            quo,quo,quo         ; Shift quo with/into carry
            DS              rem,dvr,rem         ; 32nd divide step,
            ADDC            quo,quo,quo         ; Shift last quo bit into quo
            ADDB,>=,n       rem,0,finish        ; Branch if pos. rem
            ADD,<           dvr,0,0             ; If dvr > 0, add dvr
            ADD,tr          rem,dvr,rem         ;   for correcting rem.
            ADDL            rem,tp,rem          ; Else add absolute value dvr
finish
            ADD,>=          dvdu,0,0            ; Set sign of rem
            SUB             0,rem,rem           ;   to sign of dividend
            XOR,>=          dvdu,dvr,0          ; Get correct sign of quo
            SUB             0,quo,quo           ;   based on operand signs
```

Table 5-1. Register Designations

Register Designations	Purpose
dvr	Register holding divisor.
dvdu, dvdl	Pair of registers holding dividend.
tp	Temporary register.
quo	Register holding quotient.
rem	Register holding remainder.

Demonstrating the Procedure Calling Convention

In Example 4, a C program calls an assembly language program to test if .ENTER and .LEAVE are working correctly. The assembly language program checks to see if the C program has passed the value zero in ARG0. The assembly language program then returns the value RET0 to the calling program.

Example 4

C Program Listing

```
#include <stdio.h>
int errorcount = 0;
main ()
{
     int toterr = 0;
     printf("TESTING FEATURE  000");
     fflush(stdout);
     if( feat000(000) != -9 ) ++errorcount;
     printf("         %d errors\n",errorcount);
     toterr += errorcount;
     errorcount = 0;
}
```

Assembly Program Listing

```
; Assembler Module that passes results back to
; the C driver module
myfeat   .EQU    000
success  .EQU    -9
         .CODE
         .IMPORT errorcount,DATA
         .SUBSPA $CODE$
         .EXPORT feat000,ENTRY
         .PROC
         .CALLINFO
feat000 .ENTER
         COMIB,<>  myfeat,arg0,exit
         LDI       0,ret0
         LDI       success,ret0
exit     .LEAVE
         .PROCEND
         .END
```

Output of the cc -S Command

Example 5 shows how a simple C program generates assembly language code. The program calls the `printf` routine. To run the assembled code, you need to link to the file `/lib/crt0.o` and the C library file. Remember that the `ld` command requires that you link the `crt0.o` file first.

Example 5

C Program Listing

```
main ()
{
     printf ("Hello World\n");
}
```

Assembly Program Listing From the C Compiler

```
        .SPACE  $TEXT$
        .SUPSPA $CODE$,QUAD=0,ALIGN=8,ACCESS=44,CODE_ONLY
main                                            ; C runtime interface
        .PROC                                   ; delimit procedure
        .CALLINFO CALLER,FRAME=0,SAVE_RP        ; no locals, need return
        .ENTRY                                  ; proc entry code follows
        STW     2,-20(0,30)                     ; stack the return pointer
        LDO     48(30),30                       ; set up user stack pointer
        ADDIL   L'$THISMODULE$-$global$,27      ; point to printf data
        .CALL   ARGW0=GR                        ; set up for printf call
        BL      printf,2                        ; call printf thru ret pointer
        LDO     R'$THISMODULE$-$global$(1),26   ; insert argument to printf
L$exit1                                         ; hide from linker
        LDW     -68(0,30),2                     ; get callee return pointer
        BV      0(2)                            ; exit thru return pointer
        .EXIT                                   ; end of exit sequence
        LDO     -48(30),30                      ; stack pointer -> stack frame
        .PROCEND

        .SPACE  $TEXT$
        .SUBSPA $LIT$,QUAD=0,ALIGN=8,ACCESS=44
        .SUBSPA $CODE$,QUAD=0,ALIGN=8,ACCESS=44,CODE_ONLY
        .SUBSPA $UNWIND$,QUAD=0,ALIGN=8,ACCESS=44
        .SUBSPA $CODE$
        .SPACE  $PRIVATE$
        .SUBSPA $DATA$,QUAD=1,ALIGN=8,ACCESS=31
```

```
$THISMODULE$                                    ; demarks local data
        .ALIGN  8
        .STRINGZ        "Hello World\n"         ; local data
        .SUBSPA $BSS$,QUAD=1,ALIGN=8,ACCESS=31,ZERO
        .IMPORT $global$,DATA                   ; global data reference point
        .SPACE  $TEXT$
        .SUBSPA $CODE$
        .EXPORT main,PRIV_LEV=3,RTNVAL=GR       ; for linking this routine
        .IMPORT printf,CODE                     ; external proc declaration
        .END
```

Chapter 6
Assembling Your Program

This chapter describes two different ways you can invoke the Assembler and the various command line options controlling its behavior. It also contains a brief description of the interface between the Assembler and linker, and things you should remember to facilitate the running of an assembly program.

Invoking the Assembler

You may invoke the Assembler directly by using the **as** command. Or, you may invoke the Assembler through the **cc** command which processes the assembly source using the C pre-processor. The next two sections describe these pathways.

Using the as Command

The **as** command is the standard command for invoking the Assembler on the HP Precision Architecture system.

Syntax

```
as [ [option] ... [file] ... ] ...
```

The **as** command resides in the `/bin` directory. If your programming environment does not establish a path to this directory, you must include the pathname as the first part of the **as** command. For example:

```
/bin/as -l line.s box.s draw.s
```

Parameters

option Serves as a flag to the Assembler to take some special action. The **as** command supports the following flags:

`-e` Specifies that the Assembler should tolerate one million errors before terminating the assembly process. Without this option, the Assembler terminates a program after 100 errors.

`-f`	Specifies that procedures can call other procedures as the default condition. Normally, the Assembler assumes that procedures do not call other procedures. (See the CALLS and NO_CALLS parameters for the .CALLINFO directive.).
`-l`	Lists the assembled program on the standard output device. This listing shows instruction offsets and the values stored in each field.
`-o` *filename*	Assigns the specified name to the output file.
`-u`	Prevents the Assembler from creating stack unwind descriptors. This option precludes the use of the .ENTER and .LEAVE directives within a program.
`-v` *filename*	Names a file to which the Assembler writes cross-reference information; this includes the source file and the line number for each appearance of all symbols.

file

Names an input or output file. You can include multiple input files to the **as** command. The Assembler converts this stream of input files to a single output file.

Assembler input files must include the suffix **.s**. The name for the output file corresponds to the name of the last input file, except the Assembler changes the suffix of that file's name to **.o**.

Using the cc Command

You can also use the `cc` command to run the Assembler on files that end with a `.s` suffix. The `cc` command appends the system file `pcc_prefix.s` to the front of the file and pipes the file through the C pre-processor before passing the file to the Assembler.

The system file `pcc_prefix.s` contains the following configuration files:

```
hard_reg.h
soft_reg.h
std_space.h
```

These files are in the directory `/usr/include`. Table 6-1 describes the contents of each file.

NOTE

The Assembler automatically supports the different definitions that the `pcc_prefix.s` file contains and provides them as programming aids without requiring that you include this file as an input file to the `as` command. Refer to "Registers and Register Mnemonics" in Chapter 1 and "Programming Aids" in Chapter 3 for more information.

Table 6-1. PCC_PREFIX.S Definition Files

File Name	Contents
hard_reg.h	Set of .EQU's for hardware registers.
soft_reg.h	Set of register definitions that follows the Procedure Calling Convention.
std_space.h	Set of space and subspace definitions that most Assembly programs use.

The cc command normally strips all as command options from the command line. Therefore, when you want to retain one of these options, you must include the flag -wa,<opt>,... on the cc command line (where <opt> names the Assembler option you want to preserve).

You can use the C pre-processor (cpp) with assembly language programs to include C-type macros, including directives. You can use an exclamation point (!) as a statement terminator to include multiple statements in the body of one macro definition. Furthermore, you can use the .LABEL directive to declare labels within a macro definition.

NOTE

If you use cpp, C-style comments should only be used on separate lines.

Error Message Catalog

The operating system stores the text for the Assembler error messages in a file with the pathname:

/lib/as_msgs.cat.

Linking an Assembly Program

The relocatable object file produced by the Assembler must be processed by the linker, ld, before it can be executed. The linker merges relocatable object files, searches libraries for any routines that are referenced by the user's code, assigns final addresses to all program symbols, and produces an executable program file.

For the simplest assembly program, the command:

```
ld myfile.o
```

is all that is required when myfile.o is the name of the relocatable file produced by the Assembler. The linker produces a program file called a.out, provided that myfile.o did not contain any references to imported symbols. To select a program file name other than a.out, the command:

```
ld myfile.o -o myfile
```

can be used. The -o option specifies the name myfile for the program file.

In practice, however, a single stand-alone assembly program is extremely uncommon. Most assembly language code declares procedures that are called from high-level language programs. In addition, assembly language code often calls procedures in high-level languages. In these cases, the link command must include a special start-up file called /lib/crt0.o, as well as additional relocatable object files and the high-level language libraries. For example, a C program that calls some assembly language code should be linked as follows:

```
ld /lib/crt0.o prog.o asm.o -lc -o myprog
```

In this example, prog.o is the output of the C/HP-UX compiler, asm.o is the output of the Assembler, the -lc option tells the linker to search the C library, and the -o option names the program file myprog.

When the assembly language procedures are written to be called from a high-level language, the high-level language compiler is usually the easiest interface to the linker. The above example could also have been linked with the command:

```
cc prog.o asm.o -o myprog
```

The cc command always runs the linker with the appropriate files and options for linking a C program. Similarly, the pc and f77 commands should be used when the main program is in Pascal or FORTRAN 77 respectively.

Appendix A
Error Messages

This appendix lists all error messages that originate from the HP Precision Architecture Assembler.

The Assembler error messages are divided into the following categories:

- **Warning Messages** -- flag conditions that cause errors in program execution.
- **Error Messages** -- cause the assembly process to terminate abnormally.
- **Panic Messages** -- cause the assembly process to abort immediately.
- **User Warnings** -- cause the assembler to produce an object file and possibly to take a specified corrective action.
- **Limit Errors** -- caused by running into assembler limits or running out of memory.
- **Branching Errors** -- prevent the assembler from creating an object file.

The symbol "<operand>" used in the error messages designates the Assembler source element that is the subject of the error. When an error message is printed during assembly operation, the "<operand>" designation is replaced by the appropriate source element.

Warning Messages

1	MESSAGE	`-L option is obsolete and ignored`
	CAUSE	-L appeared as a command line argument.
	ACTION	Remove -L option from command line argument list.
2	MESSAGE	`Value of <operand> for 5-bit field not in [0..31] (truncated)`
	CAUSE	Second operand of EXTRS, BB, or DEP instruction not between zero and 31 inclusive. The Assembler truncates the value to be the <operand> value MOD 32.
	ACTION	To avoid truncation, specify legal value.
3	MESSAGE	`Extract/Deposit field size of <operand> not in [1..32] (set to 32)`
	CAUSE	Operand field 3 of EXTRS or DEP instruction not between 1 and 32 inclusive.
	ACTION	To override the default value of 32, use legal value in operand field 3.
4	MESSAGE	`Value of <operand> for short signed immed. < -16 (set to -16)`
	CAUSE	This message may appear for one of three reasons: 1) First operand of a deposit immediate instruction (DEPI, VDEPI, ZDEPI, or ZVDEPI) or a conditional branch immediate instruction (ADDIBF, ADDIBT, COMIBF, COMIBT, or MOVIB) is less than -16. 2) Address offset of first operand of a load short instruction (LDBS, LDCWS, LDHS, LDWAS, or LDWS) is less than -16. 3) Address offset of second operand of a store short instruction (STBS, STBYS, STHS, STWAS, or STWS) is less than -16.
	ACTION	Change indicated operand so its value falls between -16 and +15 inclusive.

---

5 **MESSAGE** `Value of <operand> for short signed immed. > 15 (set to 15)`

CAUSE This message may appear for one of three reasons:

1) First operand of a deposit immediate instruction (DEPI, VDEPI, ZDEPI, or ZVDEPI) or a conditional branch immediate instruction (ADDIBF, ADDIBT, COMIBF, COMIBT, or MOVIB) is greater than 15.

2) Address offset of first operand of a load short instruction (LDBS, LDCWS, LDHS, LDWAS, or LDWS) is greater than 15.

3) Address offset of second operand of a store short instruction (STBS, STBYS, STHS, STWAS, or STWS) is greater than 15.

ACTION Change above operand to value between -16 and +15 inclusive.

---

6 **MESSAGE** `Keyword <operand> is obsolete and ignored`

CAUSE Keyword DUMMY used with .SPACE directive or keywords ALONE or ATTEND used with .SUBSPACE directive are obsolete.

ACTION Remove obsolete keywords.

---

7 **MESSAGE** `Space characteristics may not be changed after first declaration`

CAUSE New values assigned to keywords for a space previously declared with a .SPACE directive. This message appears whenever a keyword is assigned a value, even if the values remain the same.

ACTION Use desired values for keywords on first declaration of space. Specify keyword values on first declaration of space only.

---

8 **MESSAGE** `Subspace characteristics may not be changed after first definition`

CAUSE New values assigned to keywords for a subspace previously declared with a .SUBSPACE directive.

ACTION Use desired values for keywords on first declaration of subspace. Specify keyword values on first declaration of space only.

9	MESSAGE	`Size omitted - zero assumed`
	CAUSE	`.COMM` directive used without an integer argument.
	ACTION	Use the `.COMM` directive to issue a storage request for the number of bytes desired.
10	MESSAGE	`Alignment omitted - 8 assumed`
	CAUSE	`.ALIGN` directive used without a power of two integer argument.
	ACTION	Use a power of two integer argument with `.ALIGN`.
11	MESSAGE	`Missing value - zero assumed`
	CAUSE	`.ORIGIN` or `.EQU` directive used without an integer argument.
	ACTION	Specify the actual value of the location offset or identifier desired.
12	MESSAGE	`Size omitted - 4 assumed`
	CAUSE	`.BLOCK` directive used without an integer argument.
	ACTION	Specify the actual number of bytes to be reserved for uninitialized storage.
13	MESSAGE	`Use of GR3 when frame>=8192 may cause conflict`
	CAUSE	General register 3 used in an instruction within a procedure where keyword frame set above 8192. Note: `gr3` is used as a base register for large frames.
	ACTION	Use general register other than `gr3`.
14	MESSAGE	`KEEP should not be in force for this statement`
	CAUSE	`.KEEP` directive outside of a procedure.
	ACTION	Remove `.KEEP` directive.

15	MESSAGE	`Procedure makes calls but is not flagged as CALLER in .CALLINFO`
	CAUSE	Missing `CALLER` keyword in a procedure containing `.CALL`.
	ACTION	Add `CALLER` keyword to `.CALLINFO` directive.
16	MESSAGE	`Value for <operand> must be in range 0..3 (set to 3)`
	CAUSE	Illegal privilege level specified.
	ACTION	Change the privilege level specification.
17	MESSAGE	`Existing register name, number, and type are being overwritten`
	CAUSE	Name (label) used with `.REG` directive was previously defined.
	ACTION	Use a different name in the label part to avoid overwriting previous definition.
18	MESSAGE	`Error message file cannot be located`
	CAUSE	Error message catalog is not in the user path.
	ACTION	Ensure `as_msgs.cat` is in the user path or make `/lib/as_msgs.cat` accessible.
19	MESSAGE	`Defining register missing or defining register has no type`
	CAUSE	Parameter to `.REG` is not one of the predefined Assembler registers nor is it a previously defined (with the `.REG` directive) register.
	ACTION	Either use one of the predefined Assembler registers or define register type using the `.REG` directive.

20	MESSAGE	General register expected in this field - <operand>
	CAUSE	Wrong register type used.
	ACTION	Use a general register.
21	MESSAGE	Space register expected in this field - <operand>
	CAUSE	Wrong register type used.
	ACTION	Use a space register.
22	MESSAGE	Control register expected in this field - <operand>
	CAUSE	Wrong register type used.
	ACTION	Use a control register.
23	MESSAGE	Floating point register expected in this field - <operand>
	CAUSE	Wrong register type used.
	ACTION	Use a floating point register.
24	MESSAGE	Location counter must be in range 0..3 (set to 0) - <argument>
	CAUSE	Argument to .LOCCT is not within 0..3 range.
	ACTION	Change argument to .LOCCT to be within 0..3 range.

Error Messages

1000	MESSAGE	`Unterminated quoted string`
	CAUSE	String specified as in `.STRING`, without trailing quotes.
	ACTION	Add trailing quotes to string.

1001	MESSAGE	`Undefined register symbol`
	CAUSE	Parenthesized instruction operand does not have a corresponding `.EQU` directive.
	ACTION	Equate register symbols to values zero through 31.

1002	MESSAGE	`Undefined completer`
	CAUSE	Invalid value in instruction completer field.
	ACTION	Use only those completers specified in the *Precision Architecture and Instruction Reference Manual.*

1003	MESSAGE	`Improper completer specified`
	CAUSE	Completer used with an instruction which does not take completers.
	ACTION	Use completers only with instructions specified for them in the *Precision Architecture and Instruction Reference Manual.*

1004	MESSAGE	`Output file name missing`
	CAUSE	-o option given to Assembler but not followed by a filename.
	ACTION	Place desired output filename after -o option.

1005	MESSAGE	`Unable to open xref file: <operand>`
	CAUSE	A file specified with the -v option is not writable.
	ACTION	Use a different filename.

1006	MESSAGE	`XREF file name missing after -v`
	CAUSE	Filename omitted from command line.
	ACTION	Provide a filename after -v to dump the cross-reference information.

1007	MESSAGE	`Label not allowed here in this expr`
	CAUSE	1) Register specified by identifier. 2) An expression contains the sequence immediate-operator-label.
	ACTION	1) Use integer values for registers. 2) Use labels only as first term when combining with immediate values.

1008	MESSAGE	`Illegal symbol in expression`
	CAUSE	An expression contains a sequence other than label-operator term or term-operator-term.
	ACTION	Place operators +, -, *, / between a label and a term or a term and a term.

1009	MESSAGE	`Field selector not allowed in pc_relative expression`
	CAUSE	Field selector, such as L' or R' used on an expression contained in a branch instruction.
	ACTION	Omit field selector from branch instruction.

1010	MESSAGE	`String not allowed in pc_relative expression`
	CAUSE	String used as the target of a branch instruction.
	ACTION	Target branch to an expression beginning with a label or "."
1011	MESSAGE	`"." allowed in pc_rel expression only`
	CAUSE	"." used as an operand in a nonbranch, external branch or vectored branch.
	ACTION	Use "." only in branch instructions other than branch external or branch vectored.
1012	MESSAGE	`PC_relative expression must begin with . or label`
	CAUSE	Branch target poorly formed.
	ACTION	Use label or "." as first term of branch target expression.
1013	MESSAGE	`Second label not allowed in pc_relative expression`
	CAUSE	Branch target poorly formed: label-operator-label.
	ACTION	Use an offset in place of second label.
1014	MESSAGE	`Labels may not be added - only subtracted`
	CAUSE	Attempt to form an expression using sum of two labels.
	ACTION	Use an offset in place of second label.
1015	MESSAGE	`Unexpected end of expression`
	CAUSE	Nothing follows a +, -, /, or * in an expression.
	ACTION	Place meaningful terms, integers, or labels after operator.

1016	MESSAGE	General register <operand> is out of range
	CAUSE	Register number specified greater than 31.
	ACTION	Use register number between zero and 31.
1017	MESSAGE	Value of <operand> for space register not in [0..3]
	CAUSE	Load or privileged store instruction references a space register greater than 3.
	ACTION	Change space register number to a legal value.
1018	MESSAGE	Value of <operand> for space register not in [0..7]
	CAUSE	Branch or privileged instruction uses space register greater than 7.
	ACTION	Change space register number to a legal value.
1019	MESSAGE	Opcode not defined
	CAUSE	Characters in opcode field do not comprise legal machine instruction or directive.
	ACTION	Starting in column 2, use only defined opcodes and directives.
1020	MESSAGE	Number required for keyword value
	CAUSE	A .CALLINFO keyword is set equal to a nonnumeric argument.
	ACTION	Ensure .CALLINFO keywords are assigned numeric values.
1021	MESSAGE	Unrecognized value for keyword
	CAUSE	Illegal assignment to an ARGW or RTNVAL keyword in .CALL or .EXPORT directive.
	ACTION	Use "NO", "GR", "FR", or "FU" appropriately.

1022	MESSAGE	`This statement must occur within a declared subspace`
	CAUSE	Instruction or directive present before .SUBSPA directive.
	ACTION	Use .SUBSPA directive before issuing instruction or data.

1023	MESSAGE	`Directive not allowed inside a procedure`
	CAUSE	Use of .LOCCT, .SPACE, or .SUBSPA within a procedure.
	ACTION	Do not attempt to change location counter, space or subspace within a procedure.

1024	MESSAGE	`Space name required`
	CAUSE	.SPACE directive not followed by a valid name.
	ACTION	Follow .SPACE directive with a name starting with an alphabetic character.

1025	MESSAGE	`Unrecognized keyword`
	CAUSE	Directive, such as .SPACE, .SUBSPA, or .CALLINFO, followed by a keyword not specified in Assembler manual.
	ACTION	Follow directives with legal keywords separated by commas.

1026	MESSAGE	`Name previously defined`
	CAUSE	Subspace name matches a space name.
	ACTION	Choose subspace names different from space names.

1027	MESSAGE	`This item must be declared within a space`
	CAUSE	A directive, such as .SUBSPA, is used before the first .SPACE directive.
	ACTION	Place .SPACE directive prior to offending directive.

1028	MESSAGE	`Subspace name required`
	CAUSE	.SUBSPA directive parameter list does not begin with a valid name.
	ACTION	.SUBSPA must be followed by name beginning with an alphabetic character or "$".
1029	MESSAGE	`This statement must appear within a procedure`
	CAUSE	Directive, such as .CALLINFO, .ENTER, or .LEAVE, is used outside of a procedure.
	ACTION	Use procedure related directives only within a procedure.
1030	MESSAGE	`Only one .CALLINFO per procedure`
	CAUSE	Multiple .CALLINFO directives between successive .PROC directives.
	ACTION	Place all desired keywords in one .CALLINFO directive for the procedure.
1031	MESSAGE	`Value for <operand> must be >=0`
	CAUSE	In a .CALLINFO directive, FRAME is assigned an identifier equated to a negative value.
	ACTION	Use only nonnegative FRAME values.
1032	MESSAGE	`Value for <operand> must be in range 3..19`
	CAUSE	In a .CALLINFO directive, ENTRY_GR is assigned a value less than 3 or greater than 19.
	ACTION	Assign ENTRY_GR a value between 3 and 19 inclusive. Note: Use of value zero is also legal here.

1033	MESSAGE	`Value for <operand> must be in range 12..31`
	CAUSE	In a .CALLINFO directive, ENTRY_FR is assigned a value less than 12 or greater than 31.
	ACTION	Set ENTRY_FR to a value between 12 and 31 inclusive. Note: Use of value zero is also legal here.
1034	MESSAGE	`ENTRY_SR must be 3 or not specified`
	CAUSE	.CALLINFO contains keyword specifying saved space register other than three.
	ACTION	Omit ENTRY_SR keyword as other space registers do not require saving.
1037	MESSAGE	`Nested .PROC`
	CAUSE	.PROC directive present within .PROC - .PROCEND sequence.
	ACTION	Insert .PROCEND between successive .PROCs or remove any unnecessary .PROCs.
1038	MESSAGE	`Variable name missing`
	CAUSE	Label omitted in .COMM or .EQU directive.
	ACTION	Use label with all .COMM or .EQU directives.
1039	MESSAGE	`Missing string constant`
	CAUSE	.STRING, .STRINGZ, .VERSION, or .COPYRIGHT directive present without a string operand.
	ACTION	Add missing quoted string after directive.

1040	MESSAGE	Only one copyright message permitted
	CAUSE	Multiple copyright directives present.
	ACTION	Combine messages and use one copyright directive.

1041	MESSAGE	Export name required
	CAUSE	.EXPORT directive not followed by an identifier.
	ACTION	Follow .EXPORT directive with a legal identifier that must begin with an alphabetic character, an underscore, or a dollar sign.

1042	MESSAGE	Import name required
	CAUSE	.IMPORT directive not followed by a legal identifier.
	ACTION	Add missing identifier after .IMPORT directive.

1043	MESSAGE	<operand> not permitted for export
	CAUSE	.EXPORT directive not followed by a legal identifier.
	ACTION	Provide appropriate identifier after .EXPORT directive.

1044	MESSAGE	Name required for label definition
	CAUSE	.LABEL directive not followed by legal identifier.
	ACTION	Add missing identifier after .LABEL directive

1045	MESSAGE	Name to be defined by .LABEL must appear as operand.
	CAUSE	Identifier present in column one in a .LABEL directive.
	ACTION	Place identifier after .LABEL, not before it.

1046	MESSAGE	`Duplicate definition of symbol`
	CAUSE	A label was used more than once.
	ACTION	Give labels unique names.

1047	MESSAGE	`Unmatched .PROCEND`
	CAUSE	Two `.PROCEND` directives present without `.PROC` directive in between.
	ACTION	Each procedure should begin with a single `.PROC` and end with a `.PROCEND`.

1048	MESSAGE	`A procedure may not be empty`
	CAUSE	`.PROC` followed by `.PROCEND` with no executable code between.
	ACTION	Use at least `.CALLINFO`, `.ENTER`, and `.LEAVE` within procedure body.

1049	MESSAGE	`Procedure does not have .CALLINFO`
	CAUSE	Missing `.CALLINFO` directive between `.PROC` and `.PROCEND`.
	ACTION	Insert `.CALLINFO` following `.PROC`.

1050	MESSAGE	`Illegal symbol in label position`
	CAUSE	Character present in column one which is not alphabetic, underscore, or dollar sign.
	ACTION	Use only legal identifiers in label field.

1051	MESSAGE	`Illegal symbol in opcode position`
	CAUSE	A sequence of characters starting in column two or beyond does not begin with an alphabetic character or a period.
	ACTION	Use only prescribed opcodes and directives starting in column two or beyond.

1052	MESSAGE	`Directive name not recognized`
	CAUSE	A sequence of characters starting in column two or beyond, beginning with a period, does not correspond to a prescribed directive.
	ACTION	When beginning with a period, use only prescribed directives starting in column two or beyond.
1054	MESSAGE	`Unexpected items at end of line`
	CAUSE	Legal operands are followed by trailing characters or operators.
	ACTION	Examine entire sequence of operands for syntactic integrity.
1055	MESSAGE	`Label must be defined within a declared subspace`
	CAUSE	Label present prior to a .SUBSPA directive.
	ACTION	Place label after issuing .SUBSPA directive.
1056	MESSAGE	`Empty source file(s)`
	CAUSE	No space has been declared.
	ACTION	Use at least one .SPACE directive.
1057	MESSAGE	`Missing .PROCEND`
	CAUSE	Last procedure in program does not end with .PROCEND.
	ACTION	Add .PROCEND directive to last procedure in program.
1059	MESSAGE	`Divide by zero`
	CAUSE	Attempt to perform a division operation with a zero divisor.
	ACTION	Be sure value for divisor can not become equal to zero.

1060	MESSAGE	`Argument 1 or 3 in FARG upper`
	CAUSE	Using the FU value with the ARGW1 or ARGW3 keywords.
	ACTION	Only use the FU value with ARGW0 or ARGW2 keywords.
1061	MESSAGE	`Closing parenthesis is missing in expression`
	CAUSE	Mismatched parenthesis.
	ACTION	Insert closing parenthesis at the end of the expression.
1062	MESSAGE	`Macro parameters must be separated by commas`
	CAUSE	Formal parameters to `.MACRO` or actual parameters to a macro call are not separated by commas.
	ACTION	Insert commas between parameters.
1063	MESSAGE	`Unterminated macro definition`
	CAUSE	A `.MACRO` directive is not matched with an `.ENDM` directive.
	ACTION	Terminate the macro definition with an `.ENDM` directive.
1064	MESSAGE	`Poorly formed macro parameter`
	CAUSE	Formal parameter to the macro definition is not in an Assembler-accepted form.
	ACTION	Change the form of the formal parameter to an acceptable form.
1065	MESSAGE	`Poorly formed .FLOAT or .DOUBLE argument`
	CAUSE	The floating point number that was used as the argument to `.FLOAT` or `.DOUBLE` is not in the right format.
	ACTION	Use the correct floating point format for the argument.

1066	MESSAGE	`Poorly formed bit field specifier`
	CAUSE	Bit field is not being specified in the form {x..y} where x and y are natural numbers.
	ACTION	Specify bit fields in the correct form.
1067	MESSAGE	`Bit field too wide for instruction field`
	CAUSE	Mismatched bit field declaration and use.
	ACTION	Use the same length for both the bit field being assigned to and the bit field being assigned from.
1068	MESSAGE	`Brace outside of macro definition`
	CAUSE	Opening or closing braces are being used outside a macro definition.
	ACTION	Remove opening or closing braces or use them before the `.ENDM`.
1069	MESSAGE	`Equal sign required in bit field assignment`
	CAUSE	Missing operator = for assigning one bit field to another.
	ACTION	Insert equal sign (=) for bit field assignment.
1070	MESSAGE	`Bit range must be within {0..31}`
	CAUSE	Range specified in bit field is not 0..31.
	ACTION	Ensure bit field range is within 0..31.

1071	MESSAGE	`Opening brace expected in bit range designator`
	CAUSE	Missing opening brace to specify a bit field.
	ACTION	Use correct format for bit field specification.
1072	MESSAGE	`Ending brace expected in bit range designator`
	CAUSE	Missing closing brace to specify a bit field.
	ACTION	Use correct format for bit field specification.
1073	MESSAGE	`Unmatched .ENDM`
	CAUSE	No `.MACRO` was recognized as corresponding to the `.ENDM`.
	ACTION	Either remove the unmatched `.ENDM` or insert a `.MACRO` in appropriate position preceding the `.ENDM`.
1074	MESSAGE	`Illegal expression type for plabel`
	CAUSE	More than one label was found in a plabel expression.
	ACTION	Use only one label in a procedure label expression.
1075	MESSAGE	`Undefined field selector`
	CAUSE	Illegal field selector is being used.
	ACTION	Use correct field selector.

Panic Messages

2000	MESSAGE	`Exceeded maximum error count`
	CAUSE	More than 100 errors were detected and the -e option was not invoked.
	ACTION	Use -e option to permit up to a million errors.
2001	MESSAGE	`Bad option - <operand>`
	CAUSE	An option flag was used on the command line that does not correspond to an option in the Assembler manual.
	ACTION	Use only prescribed options.
2002	MESSAGE	`Unable to open input file: <operand>`
	CAUSE	Requested input file is either nonexistent or unreadable.
	ACTION	Check for presence of requested input file and examine read-write permissions.
2003	MESSAGE	`Unable to open output file: <operand>`
	CAUSE	1) Output file exists and is not writable. 2) File system error.
	ACTION	1) Make output file writable. 2) Contact HP-UX system administrator.
2004	MESSAGE	`Free storage exhausted`
	CAUSE	Assembler cannot allocate memory for its internal structures.
	ACTION	Break up the program into smaller modules. If this doesn't work, contact HP-UX system administrator.

2005	MESSAGE	`Internal instruction parsing error on <operand>`
	CAUSE	Assembler has an internal defect.
	ACTION	Contact HP-UX system administrator.
2006	MESSAGE	`Unable to regain access to source file for listing`
	CAUSE	Not able to access source file for reading.
	ACTION	Check for existence of source file and permission to read it.
2007	MESSAGE	`Unable to access temporary file to build listing`
	CAUSE	Not able to write to the temporary listing file. Could be a file system error.
	ACTION	Call HP-UX system administrator.
2008	MESSAGE	`Unterminated macro definition`
	CAUSE	Macro definition is not complete until a `.ENDM` is encountered.
	ACTION	Insert `.ENDM` at the end of the macro definition.

User Warnings

Errors 7100 through 7199 are user warnings. The assembler will proceed, and produce an object file, in some cases taking the described form of corrective action.

7100

MESSAGE `code subspace has no unwind subspace`

CAUSE No unwind subspace was specified for the code subspace.

ACTION Contact HP-UX system administrator.

7101

MESSAGE `Improper completer, <completer>, given for opcode [opcode] - ignored`

CAUSE The completer given is not valid for the opcode.

The assembler generates object code as if the completer were not given.

ACTION You should either remove the completer or give a correct completer.

7102

MESSAGE `Immediate value of <constant> for 5-bit field in <opcode> not in [0..31] (truncated)`

CAUSE The constant was given as an operand for the opcode, and is larger than the 5-bit field allows.

The lower five bits of the given constant are used.

ACTION You should change the value to within the limits (0 through 31) or use a different instruction.

7103

MESSAGE `Extract/deposit of <constant> for field size in [opcode] not in [1..32] (set to 32)`

CAUSE The constant was given as a field size for the extract or deposit instruction with the specified opcode, and is larger than the 5-bit field allows.

The constant is set to 32 unless you take some action.

ACTION You should change the value to within the limits (1 through 32), or use a different instruction.

7104	MESSAGE	Immediate value of <constant> for <opcode> is less than -16 (set to -16)
	CAUSE	The constant was given as an operand for the opcode, and is smaller than the 5-bit signed field allows.
	ACTION	You should change the value to within the limits (-16 through 15), or use a different instruction.

7105	MESSAGE	Immediate value of <constant> for <opcode> is greater than 15 (set to 15)
	CAUSE	The constant was given as an operand for the opcode, and is larger than the 5-bit signed field allows.
	ACTION	You should change the value to within the limits (-16 through 15), or use a different instruction.

7106	MESSAGE	DSR value of <constant> for <opcode> not in [0..3] - truncated
	CAUSE	A data space register value other than 0,1,2, or 3 was given for the specified opcode.
	ACTION	You should change the value to within the limits (0 through 3).

7107	MESSAGE	CSR value of <constant> for <opcode> not in [0..7] - truncated
	CAUSE	A data space register value (constant) other than zero through 7 was given for the specified opcode.
	ACTION	You should change the value to within the limits (0 through 7).

7108	MESSAGE	The value <constant> did not fit into a <field size> bit field at offset <offset> (op code - [opcode])
	CAUSE	During object file generation, a symbol reference (such as branch label or load offset) being resolved to a constant could not be resolved within the field of the referencing instruction. The value of the actual constant being placed into the instruction is given, as well as the size of the field, the offset from the beginning of the subspace of the offending instruction, and the opcode of the offending instruction.
	ACTION	You must remove the reference, or use a different instruction sequence that can accommodate the size of the constant.

7109	MESSAGE	Tried to define the value of non-absolute symbol <symbol-name>
	CAUSE	You used a .EQU directive to define a symbol that was already defined as a non-absolute symbol, such as DATA or ENTRY.
	ACTION	You must either remove or change the symbol name in the .EQU directive, or resolve the inconsistency with the other uses of that symbol.

Limit Errors

Errors 7200 through 7299 are fatal errors that you may be able to work around. They involve running into assembler limits or running out of memory.

7200	MESSAGE	`internal table overflow`
	CAUSE	Too many labels in the object file.
	ACTION	Split the file. Make global variable and procedure names shorter. This should rarely happen.

7201	MESSAGE	`new_slc_block: out of memory`
	CAUSE	The assembler attempted to allocate some dynamic memory, and the system was unable to provide the memory.
	ACTION	1) Check the system limits because other processes that allocate dynamic memory might also be running. More memory might be available at another time. 2) Break your assembly file into smaller pieces, and assemble them separately.

7202	MESSAGE	`init_link: Out of memory`
	CAUSE	The assembler attempted to allocate some dynamic memory, and the system was unable to provide the memory.
	ACTION	1) Check the system limits because other processes might be running that also allocate dynamic memory. More memory might be available at another time. 2) Break up your assembly file into smaller pieces and assemble them separately.

7203	MESSAGE	`allocate_bytes: Out of memory`
	CAUSE	The assembler attempted to allocate some dynamic memory, and the system was unable to provide the memory.
	ACTION	1) Check the system limits because other processes might be running that also allocate dynamic memory. More memory might be available at another time. 2) Break up your assembly file into smaller pieces, and assemble them separately.

7204	MESSAGE	`error in writing to output file.`
	CAUSE	The assembler cannot write to the object file.
	ACTION	Contact HP-UX system administrator to check for file system errors.

Branching Errors

Errors 7400 through 7405 are branching errors that prevent the assembler from creating an object file. You must correct these errors to assemble your program.

7400	MESSAGE	`Procedure number <number> has no label known to linker`
	CAUSE	You used the .PROC directive to specify the beginning of a procedure, but have not exported a procedure label. This is necessary for creating unwind tables.
	ACTION	You need to export a label for the procedure.

7401	MESSAGE	`Attempt to set location counter backward with .ORIGIN value of <constant>`
	CAUSE	You specified a value for the .ORIGIN directive that causes the location of the following item to be earlier than the previous item.
	ACTION	You must either delete items before the .ORIGIN directive to adjust the offset of the immediately preceding item, or adjust the value on the .ORIGIN directive.

7402	MESSAGE	`Procedure call to non entry point: <label name>`
	CAUSE	You used the .CALL directive to mark a procedure call to a label that is not an entry point. It must be marked with either .ENTER or .ENTRY.
	ACTION	You must either remove the .CALL directive, change the target on the .CALL directive, or place a .ENTER or .ENTRY at the target.

7403	MESSAGE	`undefined label - <label name>`
	CAUSE	You used the given label in an expression, such as a branch instruction or a .WORD directive, but the label was never defined.
	ACTION	Remove the reference to the label. Or define it by exporting it or using it as a label on an instruction or data item.

7404	MESSAGE	branch target .-<constant> unresolvable, instruction number <number>
	CAUSE	You specified a negative displacement from the current location, and there is no instruction there.
	ACTION	You must place an instruction at the target of the branch, or change the constant in the branch.

7405	MESSAGE	branch target .+<constant> unresolvable, instruction number <number>
	CAUSE	You specified a branch to the current location plus a constant, but there is no instruction there. The sequence number of the branch instruction within its subspace is given.
	ACTION	You must place an instruction at the target of the branch, or change the constant in the branch.

Appendix B
Instruction Summaries

Table B-1 lists the different Assembler machine instructions, alphabetically, by their mnemonic names.

Table B-1. Instructions Arranged by Mnemonic Name

Mnemonic Name	Operands	Instruction Name
ADD,cond	r1,r2,t	Add
ADDB,cond,n	r1,r2,target	Add and Branch *
ADDBF,cond,n	r1,r2,target	Add and Branch If False
ADDBT,cond,n	r1,r2,target	Add and Branch If True
ADDC,cond	r1,r2,t	Add With Carry
ADDCO,cond	r1,r2,t	Add With Carry and Trap On Overflow
ADDI,cond	i,r,t	Add To Immediate
ADDIB,cond,n	i,r2,target	Add Immediate and Branch *
ADDIBF,cond,n	i,r2,target	Add Immediate and Branch If False
ADDIBT,cond,n	i,r2,target	Add Immediate and Branch If True
ADDIL	i,r	Add Immediate Left
ADDIO,cond	i,r,t	Add To Immediate and Trap On Overflow
ADDIT,cond	i,r,t	Add To Immediate and Trap On Condition
ADDITO,cond	i,r,t	Add to Immediate and Trap On Condition or Overflow
ADDL,cond	r1,r2,t	Add Logical
ADDO,cond	r1,r2,t	Add and Trap On Overflow

*Pseudo instruction.

(continued on next page)

Table B-1. Instructions Arranged by Mnemonic Name (continued)

Mnemonic Name	Operands	Instruction Name
AND,cond	r1,r2,t	And
ANDCM,cond	r1,r2,t	And Complement
B,n	target	Branch *
BB,cond,n	r1,p,target	Branch On Bit
BE,n	wd(sr,b)	Branch External
BL,n	target,t	Branch and Link
BLE,n	wd(sr,b)	Branch and Link External
BLR,n	x,t	Branch and Link Register
BREAK	i1,i2	Break
BV,n	x(b)	Branch Vectored
BVB,cond,n	r1,target	Branch On Variable Bit
CLDDS,uid,cmplt	i(s,b),t	Coprocessor Load Doubleword Short
CLDDX,uid,cmplt	x(s,b),t	Coprocessor Load Doubleword Indexed
CLDWS,uid,cmplt	d(s,b),t	Coprocessor Load Word Short
CLDWX,uid,cmplt	x(s,b),t	Coprocessor Load Word Indexed
COMB,cond,n	r1,r2,target	Compare and Branch *
COMBF,cond,n	r1,r2,target	Compare and Branch If False
COMBT,cond,n	r1,r2,target	Compare and Branch If True
COMCLR,cond	r1,r2,t	Compare and Clear
COMIB,cond,n	i,r2,target	Compare Immediate and Branch *
COMIBF,cond,n	i,r2,target	Compare Immediate and Branch If False
COMIBT,cond,n	i,r2,target	Compare Immediate and Branch If True
COMICLR,cond	i,r,t	Compare Immediate and Clear

*Pseudo instruction.

(continued on next page)

Table B-1. Instructions Arranged by Mnemonic Name (continued)

Mnemonic Name	Operands	Instruction Name
COPR,uid,sop,n		Coprocessor Operation
COPY	r,t	Copy *
CSTDS,uid,cmplt	r,d(s,b)	Coprocessor Store Doubleword Short
CSTDX,uid,cmplt	r,x(s,b)	Coprocessor Store Doubleword Indexed
CSTWS,uid,cmplt	r,d(s,b)	Coprocessor Store Word Short
CSTWX,uid,cmplt	r,x(s,b)	Coprocessor Store Word Indexed
DCOR,cond	r,t	Decimal Correct
DEP,cond	r,p,len,t	Deposit
DEPI,cond	i,p,len,t	Deposit Immediate
DIAG	i	Diagnose
DS,cond	r1,r2,t	Divide Step
EXTRS,cond	r,p,len,t	Extract Signed
EXTRU,cond	r,p,len,t	Extract Unsigned
FABS,fmt	r,t	Floating-point Absolute Value
FADD,fmt	r1,r2,t	Floating-point Add
FCMP,fmt,cond	r1,r2	Floating-point Compare
FCNVFF,sf,df	r,t	Floating-point Convert from Floating-point to Floating-point
FCNVFX,sf,df	r,t	Floating-point Convert from Floating-point to fixed-Point
FCNVFXT,sf,df	r,t	Floating-point Convert from Floating-point to Fixed-point and Truncate
FCNVXF,sf,df	r,t	Floating-point Convert from Fixed-point to Floating-point
FCPY,fmt	r,t	Floating-point Copy

*Pseudo instruction.

(continued on next page)

Table B-1. Instructions Arranged by Mnemonic Name (continued)

Mnemonic Name	Operands	Instruction Name
FDC,cmplt	x(s,b)	Flush Data Cache
FDCE,cmplt	x(s,b)	Flush Data Cache Entry
FDIV,fmt	r1,r2,t	Floating-point Divide
FIC,cmplt	x(sr,b)	Flush Instruction Cache
FICE,cmplt	x(sr,b)	Flush Instruction Cache Entry
FLDDS,cmplt	d(s,b),t	Floating-point Load Doubleword Short
FLDDX,cmplt	x(s,b),t	Floating-point Load Doubleword Indexed
FLDWS,cmplt	d(s,b),t	Floating-point Load Word Short
FLDWX,cmplt	x(s,b),t	Floating-point Load Word Indexed
FMPY,fmt	r1,r2,t	Floating-point Multiply
FREM,fmt	r1,r2,t	Floating-point Remainder
FRND,fmt	r,t	Floating-point Round to Integer
FSQRT,fmt	r,t	Floating-point Square Root
FSTDS,cmplt	r,d(s,b)	Floating-point Store Doubleword Short
FSTDX,cmplt	r,x(s,b)	Floating-point Store Doubleword Indexed
FSTWS,cmplt	r,d(s,b)	Floating-point Store Word Short
FSTWX,cmplt	r,x(s,b)	Floating-point Store Word Indexed
FSUB,fmt	r1,r2,t	Floating-point Subtract
FTEST		Floating-point Test
GATE,n	target,t	Gateway
IDCOR,cond	r,t	Intermediate Decimal Correct
IDTLBA	r,(s,b)	Insert Data Translation Lookaside Buffer Address

(continued on next page)

Table B-1. Instructions Arranged by Mnemonic Name (continued)

Mnemonic Name	Operands	Instruction Name
IDTLBP	r,(s,b)	Insert Data Translation Lookaside Buffer Protection
IITLBA	r,(sr,b)	Insert Instruction Translation Lookaside Buffer Address
IITLBP	r,(sr,b)	Insert Instruction Translation Lookaside Buffer Protection
LDB	d(s,b),t	Load Byte
LDBS,cmplt	d(s,b),t	Load Byte Short
LDBX,cmplt	x(s,b),t	Load Byte Indexed
LDCWS,cmplt	d(s,b),t	Load and Clear Word Short
LDCWX,cmplt	x(s,b),t	Load and Clear Word Indexed
LDH	d(s,b),t	Load Halfword
LDHS,cmplt	d(s,b),t	Load Halfword Short
LDHX,cmplt	x(s,b),t	Load Halfword Indexed
LDI	i,t	Load Immediate *
LDIL	i,t	Load Immediate Left
LDO	d(b),t	Load Offset
LDSID	(s,b),t	Load Space Identifier
LDW	d(s,b),t	Load Word
LDWAS,cmplt	d(b),t	Load Word Absolute Short
LDWAX,cmplt	x(b),t	Load Word Absolute Indexed
LDWM	d(s,b),t	Load Word and Modify
LDWS,cmplt	d(s,b),t	Load Word Short
LDWX,cmplt	x(s,b),t	Load Word Indexed

*Pseudo instruction.

(continued on next page)

Table B-1. Instructions Arranged by Mnemonic Name (continued)

Mnemonic Name	Operands	Instruction Name
LHA,cmplt	x(s,b),t	Load Hash Address
LPA,cmplt	x(s,b),t	Load Physical Address
MFCTL	r,t	Move From Control Register
MFSP	sr,t	Move From Space Register
MOVB,cond,n	r1,r2,target	Move and Branch
MOVIB,cond,n	i,r2,target	Move Immediate and Branch
MTCTL	r,t	Move To Control Register
MTSAR	r	Move To Shift Amount Register *
MTSM	r	Move To System Mask
MTSP	r,sr	Move To Space Register
NOP		No Operation *
OR,cond	r1,r2,t	Inclusive OR
PDC,cmplt	x(s,b)	Purge Data Cache
PDTLB,cmplt	x(s,b)	Purge Data Translation Lookaside Buffer
PDTLBE,cmplt	x(s,b)	Purge Data Translation Lookaside Buffer Entry
PITLB,cmplt	x(sr,b)	Purge Instruction Translation Lookaside Buffer
PITLBE,cmplt	x(sr,b)	Purge Instruction Translation Lookaside Buffer Entry
PROBER	(s,b),r,t	Probe Read Access
PROBERI	(s,b),i,t	Probe Read Access Immediate
PROBEW	(s,b),r,t	Probe Write Access
PROBEWI	(s,b),i,t	Probe Write Access Immediate
RFI		Return From Interruption

*Pseudo instruction.

(continued on next page)

Table B-1. Instructions Arranged by Mnemonic Name (continued)

Mnemonic Name	Operands	Instruction Name
RSM	i,t	Reset System Mask
SH1ADD,cond	r1,r2,t	Shift One and Add
SH1ADDL,cond	r1,r2,t	Shift One and Add Logical
SH1ADDO,cond	r1,r2,t	Shift One, Add and Trap On Overflow
SH2ADD,cond	r1,r2,t	Shift Two and Add
SH2ADDL,cond	r1,r2,t	Shift Two and Add Logical
SH2ADDO,cond	r1,r2,t	Shift Two, Add and Trap On Overflow
SH3ADD,cond	r1,r2,t	Shift Three and Add
SH3ADDL,cond	r1,r2,t	Shift Three and Add Logical
SH3ADDO,cond	r1,r2,t	Shift Three, Add and Trap On Overflow
SHD,cond	r1,r2,count,t	Shift Double
SSM	i,t	Set System Mask
STB	r,d(s,b)	Store Byte
STBS,cmplt	r,d(s,b)	Store Byte Short
STBYS,cmplt	r,d(s,b)	Store Bytes Short
STH	r,d(s,b)	Store Halfword
STHS,cmplt	r,d(s,b)	Store Halfword Short
STW	r,d(s,b)	Store Word
STWAS,cmplt	r,d(b)	Store Word Absolute Short
STWM	r,d(s,b)	Store Word and Modify
STWS,cmplt	r,d(s,b)	Store Word Short
SUB,cond	r1,r2,t	Subtract
SUBB,cond	r1,r2,t	Subtract With Borrow
SUBBO,cond	r1,r2,t	Subtract With Borrow and Trap On Overflow

(continued on next page)

Table B-1. Instructions Arranged by Mnemonic Name (continued)

Mnemonic Name	Operands	Instruction Name
SUBI,cond	i,r,t	Subtract from Immediate
SUBIO,cond	i,r,t	Subtract from Immediate and Trap On Overflow
SUBO,cond	r1,r2,t	Subtract and Trap On Overflow
SUBT,cond	r1,r2,t	Subtract and Trap On Condition
SUBTO,cond	r1,r2,t	Subtract and Trap On Condition or Overflow
SYNC		Synchronize Caches
UADDCM,cond	r1,r2,t	Unit Add Complement
UADDCMT,cond	r1,r2,t	Unit Add Complement and Trap On Condition
UXOR,cond	r1,r2,t	Unit Exclusive Or
VDEP,cond	r,len,t	Variable Deposit
VDEPI,cond	i,len,t	Variable Deposit Immediate
VEXTRS,cond	r,len,t	Variable Extract Signed
VEXTRU,cond	r,len,t	Variable Extract Unsigned
VSHD,cond	r1,r2,t	Variable Shift Double
XOR,cond	r1,r2,t	Exclusive Or
ZDEP,cond	r,p,len,t	Zero and Deposit
ZDEPI,cond	i,p,len,t	Zero and Deposit Immediate
ZVDEP,cond	r,len,t	Zero and Variable Deposit
ZVDEPI,cond	i,len,t	Zero and Variable Deposit Immediate

Index

SPECIAL CHARACTERS

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W

SALES & SUPPORT OFFICES
Arranged alphabetically by country

Product Line Sales/Support Key

Key Product Line
- **A** Analytical
- **CM** Components
- **C** Computer Systems
- **E** Electronic Instruments & Measurement Systems
- **M** Medical Products
- **P** Personal Computation 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.

HEADQUARTERS OFFICES

If there is no sales office listed for your area, contact one of these headquarters offices.

ASIA
Hewlett-Packard Asia Ltd.
47/F, 26 Harbour Rd.,
Wanchai, **HONG KONG**
G.P.O. Box 863, Hong Kong
Tel: 5-8330833
Telex: 76793 HPA HX
Cable: HPASIAL TD

CANADA
Hewlett-Packard (Canada) Ltd.
6877 Goreway Drive
MISSISSAUGA, Ontario L4V 1M8
Tel: (416) 678-9430
Telex: 069-8644

EASTERN EUROPE
Hewlett-Packard Ges.m.b.h.
Lieblgasse 1
P.O.Box 72
A-1222 **VIENNA**, Austria
Tel: (222) 2500-0
Telex: 1 3 4425 HEPA A

NORTHERN EUROPE
Hewlett-Packard S.A.
V. D. Hooplaan 241
P.O.Box 999
NL-118 LN 15 **AMSTELVEEN**
The Netherlands
Tel: 20 5479999
Telex: 18919 hpner

SOUTH EAST EUROPE
Hewlett-Packard S.A.
World Trade Center
110 Avenue Louis-Casai
1215 Cointrin, **GENEVA**, Switzerland
Tel: (022) 98 96 51
Telex: 27225 hpser
Mail Address:
P.O. Box
CH-1217 Meyrin 1
GENEVA
Switzerland

MIDDLE EAST AND CENTRAL AFRICA
Hewlett-Packard S.A.
Middle East/Central
Africa Sales H.Q.
7, rue du Bois-du-Lan
P.O. Box 364
CH-1217 Meyrin 1
GENEVA
Switzerland
Tel: (022) 83 12 12
Telex: 27835 hmea ch
Telefax: (022) 83 15 35

UNITED KINGDOM
Hewlett-Packard Ltd.
Nine Mile Ride
WOKINGHAM
Berkshire, RG113LL
Tel: 0344 773100
Telex: 848805/848814/848912

UNITED STATES OF AMERICA
Customer Information Center
(800) 752-0900
6:00 AM to 5 PM Pacific Time

EASTERN USA
Hewlett-Packard Co.
4 Choke Cherry Road
ROCKVILLE, MD 20850
Tel: (301) 948-6370

MIDWESTERN USA
Hewlett-Packard Co.
5201 Tollview Drive
ROLLING MEADOWS, IL 60008
Tel: (312) 255-9800

SOUTHERN USA
Hewlett-Packard Co.
2000 South Park Place
ATLANTA, GA 30339
Tel: (404) 955-1500

WESTERN USA
Hewlett-Packard Co.
5161 Lankershim Blvd.
NORTH HOLLYWOOD, CA 91601
Tel: (818) 505-5600

OTHER INTERNATIONAL AREAS
Hewlett-Packard Co.
Intercontinental Headquarters
3495 Deer Creek Road
PALO ALTO, CA 94304
Tel: (415) 857-1501
Telex: 034-8300
Cable: HEWPACK

ALGERIA
Hewlett-Packard Trading S.A.
Bureau de Liaison Alger
Villa des Lions
9, Hai Galloul
DZ-BORDJ EL BAHRI
Tel: 76 03 36
Telex: 63343 dlion dz

ANGOLA
Telectra Angola LDA
Empresa Técnica de Equipamentos
16 rue Cons. Julio de Vilhema
LUANDA
Tel: 35515,35516
Telex: 3134
E,P

ARGENTINA
Hewlett-Packard Argentina S.A.
Montaneses 2140/50
1428 **BUENOS AIRES**
Tel: 541-11-1441
Telex: 22796 HEW PAC-AR
A,C,E,P
Biotron S.A.C.I.M.e.I.
Av. Paso Colon 221, Piso 9
1399 **BUENOS AIRES**
Tel: 541-333-490,
541-322-587
Telex: 17595 BIONAR
M
Laboratorio Rodriguez
Corswant S.R.L.
Misiones, 1156 - 1876
Bernal, Oeste
BUENOS AIRES
Tel: 252-3958, 252-4991
A
Intermaco S.R.L.
Florida 537/71
Galeria Jardin - Local 28
1005 **BUENOS AIRES**
Tel: 393-4471/1928
Telex. 22796 HEW PAC-AR
P (Calculators)
Argentina Esanco S.R.L.
A/ASCO 2328
1416 **BUENOS AIRES**
Tel: 541-58-1981, 541-59-2767
Telex: 22796 HEW PAC-AR
A
All Computers S.A.
Montaneses 2140/50 5 Piso
1428 **BUENOS AIRES**
Tel: 781-4030/4039/783-4886
Telex: 18148 Ocme
P

AUSTRALIA

Adelaide, South Australia Office
Hewlett-Packard Australia Ltd.
153 Greenhill Road
PARKSIDE, S.A. 5063
Tel: 61-8-272-5911
Telex: 82536
Cable: HEWPARD Adelaide
A*,C,CM,E,P

Brisbane, Queensland Office
Hewlett-Packard Australia Ltd.
10 Payne Road
THE GAP, Queensland 4061
Tel: 61-7-300-4133
Telex: 42133
Cable: HEWPARD Brisbane
A,C,CM,E,M,P

Canberra, Australia Capital Territory Office
Hewlett-Packard Australia Ltd.
Thynne Street, Fern Hill Park
BRUCE, A.C.T. 2617
P.O. Box 257,
JAMISON, A.C.T. 2614
Tel: 61-62-80-4244
Telex: 62650
Cable: HEWPARD Canberra
C,CM,E,P

Melbourne, Victoria Office
Hewlett-Packard Australia Ltd.
31-41 Joseph Street
P.O. Box 221
BLACKBURN, Victoria 3130
Tel: 61-3-895-2895
Telex: 31-024
Cable: HEWPARD Melbourne
A,C,CM,E,M,P

Perth, Western Australia Office
Hewlett-Packard Australia Ltd.
Herdsman Business Park
CLAREMONT, W.A. 6010
Tel: 61-9-383-2188
Telex: 93859
Cable: HEWPARD Perth
C,CM,E,P

Sydney, New South Wales Office
Hewlett-Packard Australia Ltd.
17-23 Talavera Road
P.O. Box 308
NORTH RYDE, N.S.W. 2113
Tel: 61-2-888-4444
Telex: 21561
Cable: HEWPARD Sydney
A,C,CM,E,M,P

AUSTRIA
Hewlett-Packard Ges.m.b.h.
Verkaufsbuero Graz
Grottenhofstrasse 94
A-8052 **GRAZ**
Tel: 43-316-291-5660
Telex: 312375
C,E

Hewlett-Packard Ges.m.b.h.
Lieblgasse 1
P.O. Box 72
A-1222 **VIENNA**
Tel: 43-222-2500
Telex: 134425 HEPA A
A,C,CM,E,M,P

BAHRAIN
Green Salon
P.O. Box 557
MANAMA
Tel: 255503-250950
Telex: 84419
P

Wael Pharmacy
P.O. Box 648
MANAMA
Tel: 256123
Telex: 8550 WAEL BN
E,M
Zayani Computer Systems
218 Shaik Mubarak Building
Government Avenue
P.O. Box 5918
MANAMA
Tel: 276278
Telex: 9015 plans bn
P

BELGIUM
Hewlett-Packard Belgium S.A./N.V.
Blvd de la Woluwe, 100
Woluwedal
B-1200 **BRUSSELS**
Tel: (02) 32-2-761-31-11
Telex: 23494 hewpac
A,C,CM,E,M,P

BERMUDA
Applied Computer Technologies
Atlantic House Building
P.O. Box HM 2091
Par-La-Ville Road
HAMILTON 5
Tel: 295-1616
Telex: 380 3589/ACT BA
P

BOLIVIA
Arrellano Ltda
Av. 20 de Octubre #2125
Casilla 1383
LA PAZ
Tel: 368541
M

BRAZIL
Hewlett-Packard do Brasil S.A.
Alameda Rio Negro, 750-I. AND.
ALPHAVILLE
06400 Barueri SP
Tel: (011) 421.1311
Telex: (011) 71351 HPBR BR
Cable: HEWPACK Sao Paulo
CM,E
Hewlett-Packard do Brasil S.A.
Praia de Botafago 228-A-614
6. AND.-CONJ. 601
Edificio Argentina - Ala A
22250 **RIO DE JANEIRO, RJ**
Tel: (02I) 552-6422
Telex: 21905 HPBR BR
Cable: HEWPACK Rio de Janeiro
E
Van Den Cientifica Ltda.
Rua Jose Bonifacio, 458
Todos os Santos
20771 **RIO DE JANEIRO, RJ**
Tel: (021) 593-8223
Telex: 33487 EGLB BR
A
ANAMED I.C.E.I. Ltda.
Rua Vergueiro, 360
04012 **SAO PAULO, SP**
Tel: (011) 572-1106
Telex: 24720 HPBR BR
M

Datatronix Electronica Ltda.
Av. Pacaembu 746-C11
SAO PAULO, SP
Tel: (118) 260111
CM

BRUNEI

Komputer Wisman Sdn Bhd
G6, Chandrawaseh Cmplx,
Jalan Tutong
P.O. Box 1297,
BANDAR SERI BEGAWAN
NEGARA BRUNI DARUSSALAM
Tel: 673-2-2000-70/26711
C,E,P

CAMEROON

Beriac
B. P. 23
DOUALA
Tel: 420153
Telex: 5351
C,P

CANADA

Alberta

Hewlett-Packard (Canada) Ltd.
3030 3rd Avenue N.E.
CALGARY, Alberta T2A 6T7
Tel: (403) 235-3100
A,C,CM,E*,M,P*

Hewlett-Packard (Canada) Ltd.
11120-178th Street
EDMONTON, Alberta T5S 1P2
Tel: (403) 486-6666
A,C,CM,E,M,P

British Columbia

Hewlett-Packard (Canada) Ltd.
10691 Shellbridge Way
RICHMOND,
British Columbia V6X 2W8
Tel: (604) 270-2277
Telex: 610-922-5059
A,C,CM,E*,M,P*

Hewlett-Packard (Canada) Ltd.
121 - 3350 Douglas Street
VICTORIA, British Columbia V8Z 3L1
Tel: (604) 381-6616
C

Manitoba

Hewlett-Packard (Canada) Ltd.
1825 Inkster Blvd.
WINNIPEG, Manitoba R2X 1R3
Tel: (204) 694-2777
A,C,CM,E,M,P*

New Brunswick

Hewlett-Packard (Canada) Ltd.
814 Main Street
MONCTON, New Brunswick E1C 1E6
Tel: (506) 855-2841
C

Nova Scotia

Hewlett-Packard (Canada) Ltd.
Suite 111
900 Windmill Road
DARTMOUTH, Nova Scotia B3B 1P7
Tel: (902) 469-7820
C,CM,E*,M,P*

Ontario

Hewlett-Packard (Canada) Ltd.
3325 N. Service Rd., Unit W03
BURLINGTON, Ontario L7N 3G2
Tel: (416) 335-8644
C,M*

Hewlett-Packard (Canada) Ltd.
552 Newbold Street
LONDON, Ontario N6E 2S5
Tel: (519) 686-9181
A,C,CM,E*,M,P*

Hewlett-Packard (Canada) Ltd.
6877 Goreway Drive
MISSISSAUGA, Ontario L4V 1M8
Tel: (416) 678-9430
Telex: 069-83644
A,C,CM,E,M,P

Hewlett-Packard (Canada) Ltd.
2670 Queensview Dr.
OTTAWA, Ontario K2B 8K1
Tel: (613) 820-6483
A,C,CM,E*,M,P*

Hewlett-Packard (Canada) Ltd.
3790 Victoria Park Ave.
WILLOWDALE, Ontario M2H 3H7
Tel: (416) 499-2550
C,E

Quebec

Hewlett-Packard (Canada) Ltd.
17500 Trans Canada Highway
South Service Road
KIRKLAND, Quebec H9J 2X8
Tel: (514) 697-4232
Telex: 058-21521
A,C,CM,E,M,P*

Hewlett-Packard (Canada) Ltd.
1150 rue Claire Fontaine
QUEBEC CITY, Quebec G1R 5G4
Tel: (418) 648-0726
C

Hewlett-Packard (Canada) Ltd.
130 Robin Crescent
SASKATOON, Saskatchewan S7L 6M7
Tel: (306) 242-3702
C

CHILE

ASC Ltda.
Austria 2041
SANTIAGO
Tel: 223-5946, 223-6148
Telex: 392-340192 ASC CK
C,P

Jorge Calcagni y Cia
Av. Italia 634 Santiago
Casilla 16475
SANTIAGO 9
Tel: 9-011-562-222-0222
Telex: 392440283 JCYCL CZ
CM,E,M

Metrolab S.A.
Monjitas 454 of. 206
SANTIAGO
Tel: 395752, 398296
Telex: 340866 METLAB CK
A

Olympia (Chile) Ltda.
Av. Rodrigo de Araya 1045
Casilla 256-V
SANTIAGO 21
Tel: 225-5044
Telex: 340892 OLYMP
Cable: Olympiachile Santiagochile
C,P

CHINA, People's Republic of

China Hewlett-Packard Co., Ltd.
47/F China Resources Bldg.
26 Harbour Road
HONG KONG
Tel: 5-8330833
Telex: 76793 HPA HX
Cable: HP ASIA LTD
A*,M*

China Hewlett-Packard Co., Ltd.
P.O. Box 9610, Beijing
4th Floor, 2nd Watch Factory Main
Shuang Yu Shou, Bei San Huan Road
Hai Dian District
BEIJING
Tel: 33-1947 33-7426
Telex: 22601 CTSHP CN
Cable: 1920 Beijing
A,C,CM,E,M,P

China Hewlett-Packard Co., Ltd.
CHP Shanghai Branch
23/F Shanghai Union Building
100 Yan An Rd. East
SHANG-HAI
Tel: 265550
Telex: 33571 CHPSB CN
Cable: 3416 Shanghai
A,C,CM,E,M,P

COLOMBIA

Instrumentación
H. A. Langebaek & Kier S.A.
Carrerra 4A No. 52A-26
Apartado Aereo 6287
BOGOTA 1, D.E.
Tel: 212-1466
Telex: 44400 INST CO
Cable: AARIS Bogota
CM,E,M

Nefromedicas Ltda.
Calle 123 No. 9B-31
Apartado Aereo 100-958
BOGOTA D.E., 10
Tel: 213-5267, 213-1615
Telex: 43415 HEGAS CO
A

Compumundo
Avenida 15 # 107-80
BOGOTA D.E.
Tel: 57-214-4458
Telex: 39645466 MARCO
P

Carvajal, S.A.
Calle 29 Norte No. 6A-40
Apartado Aereo 46
CALI
Tel: 9-011-57-3-621888
Telex: 39655650 CUJCL CO
C,E,P

CONGO

Seric-Congo
B. P. 2105
BRAZZAVILLE
Tel: 815034
Telex: 5262

COSTA RICA

Cientifica Costarricense S.A.
Avenida 2, Calle 5
San Pedro de Montes de Oca
Apartado 10159
SAN JOSE
Tel: 9-011-506-243-820
Telex: 3032367 GALGUR CR
CM,E,M

O. Fischel R. Y. Cia. S.A.
Apartados 434-10174
SAN JOSE
Tel: 23-72-44
Telex: 2379
Cable: OFIR
A

CYPRUS

Telerexa Ltd.
P.O. Box 1152
Valentine House
8 Stassandrou St.
NICOSIA
Tel: 45 628, 62 698
Telex: 5845 tlrx cy
E,M,P

DENMARK

Hewlett-Packard A/S
Kongevejen 25
DK-3460 **BIRKEROD**
Tel: 45-02-81-6640
Telex: 37409 hpas dk
A,C,CM,E,M,P

Hewlett-Packard A/S
Rolighedsvej 32
DK-8240 **RISSKOV,** Aarhus
Tel: 45-06-17-6000
Telex: 37409 hpas dk
C,E

DOMINICAN REPUBLIC

Microprog S.A.
Juan Tomás Mejía y Cotes No. 60
Arroyo Hondo
SANTO DOMINGO
Tel: 565-6268
Telex: 4510 ARENTA DR (RCA)
P

ECUADOR

CYEDE Cia. Ltda.
Avenida Eloy Alfaro 1749
y Belgica
Casilla 6423 CCI
QUITO
Tel: 9-011-593-2-450975
Telex: 39322548 CYEDE ED
E,P

Medtronics
Valladolid 524 Madrid
P.O. 9171, **QUITO**
Tel: 2-238-951
Telex: 2298 ECUAME ED
A

Hospitalar S.A.
Robles 625
Casilla 3590
QUITO
Tel: 545-250, 545-122
Telex: 2485 HOSPTL ED
Cable: HOSPITALAR-Quito
M

Ecuador Overseas Agencies C.A.
Calle 9 de Octubre #818
P.O. Box 1296, Guayaquil
QUITO
Tel: 306022
Telex: 3361 PBCGYE ED
M

EGYPT

Sakrco Enterprises
P.O. Box 259
ALEXANDRIA
Tel: 802908, 808020, 805302
Telex: 54333
C

International Engineering Associates
6 El Gamea Street
Agouza
CAIRO
Tel: 71-21-68134-80-940
Telex: 93830 IEA UN
Cable: INTEGASSO
E

Sakrco Enterprises
70 Mossadak Street
Dokki, Giza
CAIRO
Tel: 706 440, 701 087
Telex: 9337
C

S.S.C. Medical
40 Gezerat El Arab Street
Mohandessin
CAIRO
Tel: 803844, 805998, 810263
Telex: 20503 SSC UN
M*

EL SALVADOR

IPESA de El Salvador S.A.
29 Avenida Norte 1223
SAN SALVADOR
Tel: 9-011-503-266-858
Telex: 301 20539 IPESA SAL
A,C,CM,E,P

ETHIOPIA

Seric-Ethiopia
P.O. Box 2764
ADDIS ABABA
Tel: 185114
Telex: 21150
C,P

FINLAND

Hewlett-Packard Finland
Field Oy
Niittylanpolku IO
00620 **HELSINKI**
Tel: (90) 757-1011
Telex: 122022 Field SF
CM

Hewlett-Packard Oy
Piispankalliontie 17
02200 **ESPOO**
Tel: (90) 887-21
Telex: 121563 HEWPA SF
A, C, E, M, P

FRANCE

Hewlett-Packard France
Z.I. Mercure B
Rue Berthelot
13763 Les Milles Cedex
AIX-EN-PROVENCE
Tel: 33-42-59-4102
Telex: 410770F
A,C,E,M

Hewlett-Packard France
64, Rue Marchand Saillant
F-61000 **ALENCON**
Tel: (33) 29 04 42
C**

Hewlett-Packard France
Batiment Levitan
2585, route de Grasse
Bretelle Autoroute
06600 **ANTIBES**
Tel: (93) 74-59-19
C

FRANCE (Cont'd)

Hewlett-Packard France
28 Rue de la République
Boite Postale 503
25026 **BESANÇON CEDEX, FRANCE**
Tel: (81) 83-16-22
Telex: 361157
C,E*

Hewlett-Packard France
ZA Kergaradec
Rue Fernand Forest
F-29239 **GOUEESNOU**
Tel: (98) 41-87-90
E

Hewlett-Packard France
Chemin des Mouilles
Boite Postale 162
69131 **ECULLY** Cedex (Lyon)
Tel: 33-78-33-8125
Telex: 310617F
A,C,E,M,P*

Hewlett-Packard France
Parc d'activités du Bois Briard
2 Avenue du Lac
F-91040 **EVRY** Cedex
Tel: 3311/6077 9660
Telex: 692315F
C

Hewlett-Packard France
Application Center
5, avenue Raymond Chanas
38320 **EYBENS** (Grenoble)
Tel: (76) 62-57-98
Telex: 980124 HP GRENOB EYBE
C

Hewlett-Packard France
Rue Fernand. Forest
Z.A. Kergaradec
29239 **GOUESNOU**
Tel: (98) 41-87-90

Hewlett-Packard France
Parc Club des Tanneries
Batiment B4
4, Rue de la Faisanderie
67381 **LINCOLSHEIM**
(Strasbourg)
Tel: (88) 76-15-00
Telex: 890141F
C,E*,M*,P*

Hewlett-Packard France
Centre d'affaires Paris-Nord
Bâtiment Ampère
Rue de la Commune de Paris
Boite Postale 300
93153 **LE BLANC-MESNIL**
Tel: (1) 865-44-52
Telex: 211032F
C,E,M

Hewlett-Packard France
Parc d'activités Cadéra
Quartier Jean-Mermoz
Avenue du Président JF Kennedy
33700 **MÉRIGNAC** (Bordeaux)
Tel: 33-56-34-0084
Telex: 550105F
C,E,M

Hewlett-Packard France
3, Rue Graham Bell
BP 5149
57074 **METZ** Cedex
Tel: (87) 36-13-31
Telex: 860602F
C,E

Hewlett-Packard France
Miniparc-ZIRST
Chemin du Vieux Chêne
38240 **MEYLAN** (Grenoble)
Tel: (76) 90-38-40
980124 HP Grenobe
C

Hewlett-Packard France
Bureau vert du Bois Briand
Cheman de la Garde
- CP 212 212
44085 **NANTES** Cedex
Tel: (40) 50-32-22
Telex: 711085F
A,C,E,CM*,P

Hewlett-Packard France
125, Rue du Faubourg Bannier
45000 **ORLÉANS**
Tel: 33-38-62-2031
E,P*

Hewlett-Packard France
Zone Industrielle de Courtaboeuf
Avenue des Tropiques
91947 **LES ULIS** Cedex (Orsay)
Tel: 33-6-907 7825
Telex: 600048F
A,C,CM,E,M,P**

Hewlett-Packard France
15, Avenue de L'Amiral-Bruix
75782 **PARIS** Cedex 16
Tel: 33-15-02-1220
Telex: 613663F
C,P*

Hewlett-Packard France
242 Ter, Ave J Mermoz
64000 **PAU**
Tel: 33-59-80-3802
Telex: 550365F
C,E*

Hewlett-Packard France
6, Place Sainte Croix
86000 **POITIERS**
Tel: 33-49-41-2707
Telex: 792335F
C, E*

Hewlett-Packard France
47, Rue de Chativesle
51100 **REIMS**
Tel: 33-26-88-6919
C, P*

Hewlett-Packard France
Parc d'activités de la Poterie
Rue Louis Kerautel-Botmel
35000 **RENNES**
Tel: 33-99-51-4244
Telex: 740912F
A*,C,E,M,P*

Hewlett-Packard France
98 Avenue de Bretagne
76100 **ROUEN**
Tel: 33-35-63-5766
Telex: 770035F
C,E

Hewlett-Packard France
4, Rue Thomas-Mann
Boite Postale 56
67033 **STRASBOURG** Cedex
Tel: (88) 28-56-46
Telex: 890141F
C,E,M,P*

Hewlett-Packard France
Le Péripole III
3, Chemin du Pigeonnier de la Cépière
31081 **TOULOUSE** Cedex
Tel: 33-61-40-1112
Telex: 531639F
A,C,E,M,P*

Hewlett-Packard France
Les Cardoulines
Batiment B2
Route des Dolines
Parc d'activite de Valbonne
Sophia Antipolis
06560 **VALBONNE** (Nice)
Tel: (93) 65-39-40
C

Hewlett-Packard France
9, Rue Baudin
26000 **VALENCE**
Tel: 33-75-42-7616
C**

Hewlett-Packard France
Carolor
ZAC de Bois Briand
57640 **VIGY** (Metz)
Tel: (8) 771 20 22
C

Hewlett-Packard France
Parc d'activité des Prés
1, Rue Papin Cedex
59658 **VILLENEUVE D'ASCQ**
Tel: 33-20-91-4125
Telex: 160124F
C,E,M,P

Hewlett-Packard France
Parc d'activités Paris-Nord 11
Boite Postale 60020
95971 Roissy Charles de Gaulle
VILLEPINTE
Tel: (1) 48 63 80 80
Telex: 211032F
C,E,M,P*

GABON

Sho Gabon
P.O. Box 89
LIBREVILLE
Tel: 721 484
Telex: 5230

GERMAN FEDERAL REPUBLIC

Hewlett-Packard GmbH
Vertriebszentrum Mitte
Hewlett-Packard-Strasse
D-6380 **BAD HOMBURG**
Tel: (06172) 400-0
Telex: 410 844 hpbhg
A,C,E,M,P

Hewlett-Packard GmbH
Geschäftsstelle
Keithstrasse 2-4
D-1000 **BERLIN** 30
Tel: (030) 21 99 04-0
Telex: 018 3405 hpbln d
A,C,E,M,P

Hewlett-Packard GmbH
Verbindungsstelle Bonn
Friedrich-Ebert-Allee 26
5300 **BONN**
Tel: (0228) 234001
Telex: 8869421

Hewlett-Packard GmbH
Vertriebszentrun Südwest
Schickardstrasse 2
D-7030 **BOBLINGEN**
Postfach 1427
Tel: (07031) 645-0
Telex: 7265 743 hep
A,C,CM,E,M,P

Hewlett-Packard GmbH
Zeneralbereich Mktg
Herrenberger Strasse 130
D-7030 **BOBLINGEN**
Tel: (07031) 14-0
Telex: 7265739 hep

Hewlett-Packard GmbH
Geschäftsstelle
Schleefstr. 28a
D-4600 **DORTMUND**-41
Tel: (0231) 45001
Telex: 822858 hepdod
A,C,E

Hewlett-Packard gmbH
Reparaturzentrum Frankfurt
Berner Strasse 117
6000 **FRANKFURT/MAIN 60**
Tel: (069) 500001-0
Telex: 413249 hpffm

Hewlett-Packard GmbH
Vertriebszentrum Nord
Kapstadtring 5
D-2000 **HAMBURG** 60
Tel: 49-40-63-804-0
Telex: 021 63 032 hphh d
A,C,E,M,P

Hewlett-Packard GmbH
Geschäftsstelle
Heidering 37-39
D-3000 **HANNOVER** 61
Tel: (0511) 5706-0
Telex: 092 3259 hphan
A,C,CM,E,M,P

Hewlett-Packard GmbH
Geschäftsstelle
Rosslauer Weg 2-4
D-6800 **MANNHEIM**
Tel: 49-0621-70-05-0
Telex: 0462105 hpmhm
A,C,E

Hewlett-Packard GmbH
Geschäftsstelle
Messerschmittstrasse 7
D-7910 **NEU ULM**
Tel: 49-0731-70-73-0
Telex: 0712816 HP ULM-D
A,C,E*

Hewlett-Packard GmbH
Geschäftsstelle
Emmericher Strasse 13
D-8500 **NÜRNBERG** 10
Tel: (0911) 5205-0
Telex: 0623 860 hpnbg
C,CM,E,M,P

Hewlett-Packard GmbH
Vertriebszentrum Ratingen
Berliner Strasse 111
D-4030 **RATINGEN** 4
Postfach 31 12
Tel: (02102) 494-0
Telex: 589 070 hprad
A,C,E,M,P

Hewlett-Packard GmbH
Vertriebszentrum Muchen
Eschenstrasse 5
D-8028 **TAUFKIRCHEN**
Tel: 49-89-61-2070
Telex: 0524985 hpmch
A,C,CM,E,M,P

Hewlett-Packard GmbH
Geschäftsstelle
Ermlisallee
7517 **WALDBRONN** 2
Postfach 1251
Tel: (07243) 602-0
Telex: 782 838 hepk
A,C,E

GREAT BRITAIN
See United Kingdom

GREECE

Hewlett-Packard A.E.
178, Kifissias Avenue
6th Floor
Halandri-**ATHENS**
Greece
Tel: 301116473 360, 301116726 090
Telex: 221 286 HPHLGR
A,C,CM**,E,M,P

Kostas Karaynnis S.A.
8, Omirou Street
ATHENS 133
Tel: 32 30 303, 32 37 371
Telex: 215962 RKAR GR
A,C*,CM,E

Impexin
Intelect Div
209 Mesogion
11525 **ATHENS**
Tel: 6474481/2
Telex: 216286
P

Haril Company
38, Mihalakopoulou
ATHENS 612
Tel: 7236071
Telex: 218767
M*

Hellamco
P.O. Box 87528
18507 **PIRAEUS**
Tel: 4827049
Telex: 241441
A

GUATEMALA

IPESA DE GUATEMALA
Avenida Reforma 3-48, Zona 9
GUATEMALA CITY
Tel: 316627, 317853,66471/5
9-011-502-2-316627
Telex: 3055765 IPESA 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, Wan Chai
HONG KONG
Tel: 852-5-832-3211
Telex: 66678 HEWPA HX
Cable: HEWPACK HONG KONG
E,C,P

CET Ltd.
10th Floor, Hua Asia Bldg.
64-66 Gloucester Road
HONG KONG
Tel: (5) 200922
Telex: 85148 CET HX
CM

Schmidt & Co. (Hong Kong) Ltd.
18th Floor, Great Eagle Centre
23 Harbour Road, Wanchai
HONG KONG
Tel: 5-8330222
Telex: 74766 SCHMC HX
A,M

ICELAND

Hewlett-Packard Iceland
Hoefdabakka 9
112 **REYKJAVIK**
Tel: 354-1-67-1000
Telex: 37409
A,C,CM,E,M,P

INDIA

Computer products are sold through Blue Star Ltd.All computer repairs and maintenance service is done through Computer Maintenance Corp.

Blue Star Ltd.
B. D. Patel House
Near Sardar Patel Colony
AHMEDABAD 380 014
Tel: 403531, 403532
Telex: 0121-234
Cable: BLUE FROST
A,C,CM,E

Blue Star Ltd.
40/4 Lavelle Road
BANGALORE 560 001
Tel: 57881, 867780
Telex: 0845-430 BSLBIN
Cable: BLUESTAR
A,C*,CM,E

Blue Star Ltd.
Band Box House
Prabhadevi
BOMBAY 400 025
Tel: 4933101, 4933222
Telex: 011-71051
Cable: BLUESTAR
A,M

Blue Star Ltd.
Sahas
414/2 Vir Savarkar Marg
Prabhadevi
BOMBAY 400 025
Tel: 422-6155
Telex: 011-71193 BSSS IN
Cable: FROSTBLUE
A,CM,E,M

Blue Star Ltd.
Kothari, 19 Vishwas Colony
Alkapuri, **BORODA,** 390 005
Tel: 65235, 65236
Cable: BLUE STAR
A

Blue Star Ltd.
7 Hare Street
P.O. Box 506
CALCUTTA 700 001
Tel: 230131, 230132
Telex: 031-61120 BSNF IN
Cable: BLUESTAR
A,M,C,E

Blue Star Ltd.
133 Kodambakkam High Road
MADRAS 600 034
Tel: 472056, 470238
Telex: 041-379
Cable: BLUESTAR
A,M

Blue Star Ltd.
13 Community Center
New Friends Colony
NEW DELHI 110 065
Tel: 682547
Telex: 031-2463
Cable: BLUEFROST
A,C*,CM,E,M

Blue Star Ltd.
15/16 C Wellesley Rd.
PUNE 411 011
Tel: 22775
Cable: BLUE STAR
A

Blue Star Ltd.
2-2-47/1108 Bolarum Rd.
SECUNDERABAD 500 003
Tel: 72057, 72058
Telex: 0155-459
Cable: BLUEFROST
A,C,E

Blue Star Ltd.
T.C. 7/603 Poornima
Maruthunkuzhi
TRIVANDRUM 695 013
Tel: 65799, 65820
Telex: 0884-259
Cable: BLUESTAR
E

Computer Maintenance Corporation Ltd.
115, Sarojini Devi Road
SECUNDERABAD 500 003
Tel: 310-184, 345-774
Telex: 031-2960
C**

INDONESIA

BERCA Indonesia P.T.
P.O.Box 496/Jkt.
Jl. Abdul Muis 62
JAKARTA
Tel: 21-373009
Telex: 46748 BERSAL IA
Cable: BERSAL JAKARTA
P

BERCA Indonesia P.T.
P.O.Box 2497/Jkt
Antara Bldg., 12th Floor
Jl. Medan Merdeka Selatan 17
JAKARTA-PUSAT
Tel: 21-340417
Telex: 46748 BERSAL IA
A,C,E,M,P

BERCA Indonesia P.T.
Jalan Kutai 24
SURABAYA
Tel: 67118
Telex: 31146 BERSAL SB
Cable: BERSAL-SURABAYA
A*,E,M,P

IRAQ

Hewlett-Packard Trading S.A.
Service Operation
Al Mansoor City 9B/3/7
BAGHDAD
Tel: 551-49-73
Telex: 212-455 HEPAIRAQ IK
C

IRELAND

Hewlett-Packard Ireland Ltd.
Temple House, Temple Road
Blackrock, Co. **DUBLIN**
Tel: 88/333/99
Telex: 30439
C,E,P

Hewlett-Packard Ltd.
75 Belfast Rd, Carrickfergus
Belfast BT38 8PH
NORTHERN IRELAND
Tel: 09603-67333
Telex: 747626
M

ISRAEL

Eldan Electronic Instrument Ltd.
P.O.Box 1270
JERUSALEM 91000
16, Ohaliav St.
JERUSALEM 94467
Tel: 533 221, 553 242
Telex: 25231 AB/PAKRD IL
A,M

Computation and Measurement Systems (CMS) Ltd.
11 Masad Street
67060
TEL-AVIV
Tel: 388 388
Telex: 33569 Motil IL
C,CM,E,P

ITALY

Hewlett-Packard Italiana S.p.A
Traversa 99C
Via Giulio Petroni, 19
I-70124 **BARI**
Tel: (080) 41-07-44
C,M

Hewlett-Packard Italiana S.p.A.
Via Emilia, 51/C
I-40011 **BOLOGNA** Anzola Dell'Emilia
Tel: 39-051-731061
Telex: 511630
C,E,M

Hewlett-Packard Italiana S.p.A.
Via Principe Nicola 43G/C
I-95126 **CATANIA**
Tel: (095) 37-10-87
Telex: 970291
C

Hewlett-Packard Italiana S.p.A.
Via G. di Vittorio 10
20094 **CORSICO** (Milano)
Tel: 39-02-4408351

Hewlett-Packard Italiana S.p.A.
Viale Brigata Bisagno 2
16129 **GENOVA**
Tel: 39-10-541141
Telex: 215238

Hewlett-Packard Italiana S.p.A.
Viale G. Modugno 33
I-16156 **GENOVA PEGLI**
Tel: (010) 68-37-07
Telex: 215238
C,E

Hewlett-Packard Italiana S.p.A.
Via G. di Vittorio 9
I-20063 **CERNUSCO SUL NAVIGLIO**
(Milano)
Tel: (02) 923691
Telex: 334632
A,C,CM,E,M,P

Hewlett-Packard Italiana S.p.A.
Via Nuova Rivoltana 95
20090 **LIMITO** (Milano)
Tel: 02-92761

Hewlett-Packard Italiana S.p.A.
Via Nuova San Rocco a
Capodimonte, 62/A
I-80131 **NAPOLI**
Tel: (081) 7413544
Telex: 710698
A**,C,E,M

Hewlett-Packard Italiana S.p.A.
Via Orazio 16
80122 **NAPOLI**
Tel: (081) 7611444
Telex: 710698

Hewlett-Packard Italiana S.p.A.
Via Pellizzo 15
35128 **PADOVA**
Tel: 39-49-664-888
Telex: 430315
A,C,E,M

Hewlett-Packard Italiana S.p.A.
Viale C. Pavese 340
I-00144 **ROMA EUR**
Tel: 39-65-48-31
Telex: 610514
A,C,E,M,P*

Hewlett-Packard Italiana S.p.A.
Via di Casellina 57/C
500518 **SCANDICCI-FIRENZE**
Tel: 39-55-753863
C,E,M

Hewlett-Packard Italiana S.p.A.
Corso Svizzera, 185
I-10144 **TORINO**
Tel: 39-11-74-4044
Telex: 221079
A*,C,E

IVORY COAST

S.I.T.E.L.
Societe Ivoirienne de
Telecommunications
Bd. Giscard d'Estaing
Carrefour Marcory
Zone 4.A.
Boite postale 2580
ABIDJAN 01
Tel: 353600
Telex: 43175
E

S.I.T.I.
Immeuble "Le General"
Av. du General de Gaulle
01 BP 161
ABIDJAN 01
Tel: 321227
Telex: 22149
C,P

JAPAN

Yokogawa-Hewlett-Packard Ltd.
152-1, Onna
ATSUGI, Kanagawa, 243
Tel: (0462) 25-0031
C,CM,E

Yokogawa-Hewlett-Packard Ltd.
Meiji-Seimei Bldg. 6F
3-1 Motochiba-Cho
CHIBA, 280
Tel: (0472) 25 7701
C,E

Yokogawa-Hewlett-Packard Ltd.
Yasuda-Seimei Hiroshima Bldg.
6-11, Hon-dori, Naka-ku
HIROSHIMA, 730
Tel: (082) 241-0611

Yokogawa-Hewlett-Packard Ltd.
Towa Building
2-2-3 Kaigan-dori, Chuo-ku
KOBE, 650
Tel: (078) 392-4791
C,E

Yokogawa-Hewlett-Packard Ltd.
Kumagaya Asahi 82 Bldg.
3-4 Tsukuba
KUMAGAYA, Saitama 360
Tel: (0485) 24-6563
C,CM,E

Yokogawa-Hewlett-Packard Ltd.
Asahi Shinbun Daiichi Seimei Bldg.
4-7, Hanabata-cho
KUMAMOTO, 860
Tel: 96-354-7311
C,E

Yokogawa-Hewlett-Packard Ltd.
Shin-Kyoto Center Bldg.
614, Higashi-Shiokoji-cho
Karasuma-Nishiiru
KYOTO, 600
Tel: 075-343-0921
C,E

Yokogawa-Hewlett-Packard Ltd.
Mito Mitsui Bldg.
1-4-73, Sanno-maru
MITO, Ibaraki 310
Tel: (0292) 25-7470
C,CM,E

Yokogawa-Hewlett-Packard Ltd.
Meiji-Seimei Kokubun Bldg.
7-8 Kokubun, 1 Chome, Sendai
MIYAGI, 980
Tel: (0222) 25-1011
C,E

Yokogawa-Hewlett-Packard Ltd.
Gohda Bldg. 2F
1-2-10 Gohda Okaya-Shi
Okaya-Shi
NAGANO, 394
Tel: (0266) 23 0851
C,E

Yokogawa-Hewlett-Packard Ltd.
Nagoya Kokusai Center Building
1-47-1, Nagono, Nakamura-ku
NAGOYA, AICHI 450
Tel: (052) 571-5171
C,CM,E,M

Yokogawa-Hewlett-Packard Ltd.
Sai-Kyo-Ren Building
1-2 Dote-cho
OOMIYA-SHI SAITAMA 330
Tel: (0486) 45-8031

JAPAN (Cont'd)

Yokogawa-Hewlett-Packard Ltd.
Chuo Bldg., 5-4-20 Nishi-Nakajima
4-20 Nishinakajima, 5 Chome,
Yodogawa-ku
OSAKA, 532
Tel: (06) 304-6021
Telex: YHPOSA 523-3624
C,CM,E,M,P*

Yokogawa-Hewlett-Packard Ltd.
1-27-15, Yabe
SAGAMIHARA Kanagawa, 229
Tel: 0427 59-1311

Yokogawa-Hewlett-Packard Ltd.
Hamamtsu Motoshiro-Cho Daichi
Seimei Bldg 219-21, Motoshiro-Cho
Hamamatsu-shi
SHIZUOKA, 430
Tel: (0534) 56 1771
C,E

Yokogawa-Hewlett-Packard Ltd.
Shinjuku Daiichi Seimei Bldg.
2-7-1, Nishi Shinjuku
Shinjuku-ku,**TOKYO** 163
Tel: 03-348-4611
C,E,M

Yokogawa Hewlett-Packard Ltd.
9-1, Takakura-cho
Hachioji-shi, **TOKYO,** 192
Tel: 81-426-42-1231
C,E

Yokogawa-Hewlett-Packard Ltd.
3-29-21 Takaido-Higashi, 3 Chome
Suginami-ku **TOKYO** 168
Tel: (03) 331-6111
Telex: 232-2024 YHPTOK
C,CM,E,P*

Yokogawa Hokushin Electric
Corporation
Shinjuku-NS Bldg. 10F
4-1 Nishi-Shinjuku 2-Chome
Shinjuku-ku
TOKYO, 163
Tel: (03) 349-1859
Telex: J27584
A

Yokogawa Hokushin Electric Corp.
9-32 Nokacho 2 Chome
Musashino-shi
TOKYO, 180
Tel: (0422) 54-1111
Telex: 02822-421 YEW MTK J
A

Yokogawa-Hewlett-Packard Ltd.
Meiji-Seimei
Utsunomiya Odori Building
1-5 Odori, 2 Chome
UTSUNOMIYA, Tochigi 320
Tel: (0286) 33-1153
C,E

Yokogawa-Hewlett-Packard Ltd.
Yasuda Seimei Nishiguchi Bldg.
30-4 Tsuruya-cho, 3 Chome
Kanagawa-ku, **YOKOHAMA** 221
Tel: (045) 312-1252
C,CM,E

JORDAN

Scientific and Medical Supplies Co.
P.O. Box 1387
AMMAN
Tel: 24907, 39907
Telex: 21456 SABCO JO
C,E,M,P

KENYA

ADCOM Ltd., Inc., Kenya
P.O.Box 30070
NAIROBI
Tel: 331955
Telex: 22639
E,M

KOREA

Samsung Hewlett-Packard Co. Ltd.
Dongbang Yeoeuido Building
12-16th Floors
36-1 Yeoeuido-Dong
Youngdeungpo-Ku
SEOUL
Tel: 784-4666, 784-2666
Telex: 25166 SAMSAN K
C,CM,E,M,P
Young In Scientific Co., Ltd.
Youngwha Building
547 Shinsa Dong, Kangnam-Ku
SEOUL 135
Tel: 546-7771
Telex: K23457 GINSCO
A

Dongbang Healthcare
Products Co. Ltd.
Suite 301 Medical Supply Center
Bldg. 1-31 Dongsungdong
Jong Ro-gu, **SEOUL**
Tel: 764-1171, 741-1641
Telex: K25706 TKBKO
Cable: TKBEEPKO
M

KUWAIT

Al-Khaldiya Trading & Contracting
P.O. Box 830
SAFAT
Tel: 424910, 411726
Telex: 22481 AREEG KT
Cable: VISCOUNT
E,M,A

Gulf Computing Systems
P.O. Box 25125
SAFAT
Tel: 435969
Telex: 23648
P

Photo & Cine Equipment
P.O. Box 270
SAFAT
Tel: 2445111
Telex: 22247 MATIN KT
Cable: MATIN KUWAIT
P

W.J. Towell Computer Services
P.O. Box 5897
SAFAT
Tel: 2462640/1
Telex: 30336 TOWELL KT
C

LEBANON

Computer Information Systems S.A.L.
Chammas Building
P.O. Box 11-6274 Dora
BEIRUT
Tel: 89 40 73
Telex: 42309 chacis le
C,E,M,P

LIBERIA

Unichemicals Inc.
P.O. Box 4509
MONROVIA
Tel: 224282
Telex: 4509
E

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,C,CM,E,M,P

MADAGASCAR

Technique et Precision
12, rue de Nice
P.O. Box 1227
101 **ANTANANARIVO**
Tel: 22090
Telex: 22255
P

MALAYSIA

Hewlett-Packard Sales (Malaysia)
Sdn. Bhd.
9th Floor
Chung Khiaw Bank Building
46, Jalan Raja Laut
50736 **KUALA LUMPUR, MALAYSIA**
Tel: 03-2986555
Telex: 31011 HPSM MA
A,C,E,M,P*

Protel Engineering
P.O.Box 1917
Lot 6624, Section 64
23/4 Pending Road
Kuching, **SARAWAK**
Tel: 36299
Telex: 70904 PROMAL MA
Cable: PROTELENG
A,E,M

MALTA

Philip Toledo Ltd.
Kirkirkara P.O. Box 11
Notabile Rd.
MRIEHEL
Tel: 447 47, 455 66, 4915 25
Telex: Media MW 649
E,M,P

MAURITIUS

Blanche Birger Co. Ltd.
18, Jules Koenig Street
PORT LOUIS
Tel: 20828
Telex: 4296
P

MEXICO

Hewlett-Packard de Mexico,
S.A. de C.V.
Rio Nio No. 4049 Desp. 12
Fracc. Cordoba
JUAREZ
Tel: 161-3-15-62
P

Hewlett-Packard de Mexico,
S.A. de C.V.
Condominio Kadereyta
Circuito del Mezon No. 186 Desp. 6
COL. DEL PRADO - 76030 Qro.
Tel: 463-6-02-71
P

Hewlett-Packard de Mexico,
S.A. de C.V.
Monti Morelos No. 299
Fraccionamiento Loma Bonita 45060
GUADALAJARA, Jalisco
Tel: 36-31-48-00
Telex: 0684 186 ECOME
P

Microcomputadoras
Hewlett-Packard, S.A.
Monti Pelvoux 115
LOS LOMAS, Mexico, D.F.
Tel: 520-9127
P

Microcomputadoras Hewlett-Packard,
S.A. de C.V.
Monte Pelvoux No. 115
Lomas de Chapultepec, 11000
MEXICO, D.F.
Tel: 520-9127
P

Hewlett-Packard de Mexico,
S.A. de C.V.
Monte Pelvoux No. 111
Lomas de Chapultepec
11000 **MEXICO, D.F.**
Tel: 5-40-62-28, 72-66, 50-25
Telex: 17-74-507 HEWPACK MEX
A,C,CM,E,M,P

Hewlett-Packard De Mexico (Polanco)
Avenida Ejercito Nacional #579
2day3^{er} piso
Colonia Granada 11560
MEXICO D.F.
Tel: 254-4433
P

Hewlett-Packard de Mexico,
S.A. de C.V.
Czda. del Valle
409 Ote. 4th Piso
Colonia del Valle
Municipio de Garza
Garcia Nuevo Leon
66220 **MONTERREY,** Nuevo León
Tel: 83-78-42-40
Telex: 382410 HPMY
C

Infograficas y Sistemas
del Noreste, S.A.
Rio Orinoco #171 Oriente
Despacho 2001
Colonia Del Valle
MONTERREY
Tel: 559-4415, 575-3837
Telex: 483164
A,E

Hewlett-Packard de Mexico,
S.A. de C.V.
Blvd. Independencia No. 2000 Ote.
Col. Estrella
TORREON, COAH.
Tel: 171-18-21-99
P

MOROCCO

Etablissement Hubert Dolbeau & Fils
81 rue Karatchi
B.P. 11133
CASABLANCA
Tel: 3041-82, 3068-38
Telex: 23051, 22822
E

Gerep
2, rue Agadir
Boite Postale 156
CASABLANCA 01
Tel: 272093, 272095
Telex: 23 739
P

Sema-Maroc
Dept. Seric
6, rue Lapebie
CASABLANCA
Tel: 260980
Telex: 21641
C,P

NETHERLANDS

Hewlett-Packard Nederland B.V.
Startbaan 16
NL-1187 XR **AMSTELVEEN**
P.O. Box 667
NL-1180 AR **AMSTELVEEN**
Tel: (020) 547-6911
Telex: 13 216 HEPA NL
A,C,CM,E,M,P

Hewlett-Packard Nederland B.V.
Bongerd 2
P.O. Box 41
NL 2900AA **CAPELLE A/D IJSSEL**
Tel: 31-20-51-6444
Telex: 21261 HEPAC NL
C,E

Hewlett-Packard Nederland B.V.
Pastoor Petersstraat 134-136
P.O. Box 2342
NL 5600 CH **EINDHOVEN**
Tel: 31-40-32-6911
Telex: 51484 hepae nl
C,E,P

NEW ZEALAND

Hewlett-Packard (N.Z.) Ltd.
5 Owens Road
P.O. Box 26-189
Epsom, **AUCKLAND**
Tel: 64-9-687-159
Cable: HEWPAK Auckland
C,CM,E,P*

Hewlett-Packard (N.Z.) Ltd.
184-190 Willis Street
WELLINGTON
P.O. Box 9443
Courtenay Place, **WELLINGTON 3**
Tel: 64-4-887-199
Cable: HEWPACK Wellington
C,CM,E,P

Northrop Instruments & Systems Ltd.
369 Khyber Pass Road
P.O. Box 8602
AUCKLAND
Tel: 794-091
Telex: 60605
A,M

Northrop Instruments & Systems Ltd.
110 Mandeville St.
P.O. Box 8388
CHRISTCHURCH
Tel: 488-873
Telex: 4203
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

Elmeco Nigeria Ltd.
45 Saka Tirubu St.
Victoria Island
LAGOS
Tel: 61-98-94
Telex: 20-117
E

NORTHERN IRELAND
See United Kingdom

NORWAY

Hewlett-Packard Norge A/S
Folke Bernadottes vei 50
P.O. Box 3558
N-5033 **FYLLINGSDALEN** (Bergen)
Tel: 0047/5/16 55 40
Telex: 76621 hpnas n
C,E,M

Hewlett-Packard Norge A/S
Osterndalen 16-18
P.O. Box 34
N-1345 **OESTERAAS**
Tel: 47-2-17-1180
Telex: 76621 hpnas n
A,C,CM,E,M,P

Hewlett-Packard Norge A/S
Boehmergt. 42
Box 2470
N-5037 **SOLHEIMSVIK**
Tel: 0047/5/29 00 90

OMAN

Khimjil Ramdas
P.O. Box 19
MUSCAT/SULTANATE OF OMAN
Tel: 795 901
Telex: 3489 BROKER MB MUSCAT
P

Suhail & Saud Bahwan
P.O.Box 169
MUSCAT/SULTANATE OF OMAN
Tel: 734 201-3
Telex: 5274 BAHWAN MB
E

Imtac LLC
P.O. Box 9196
MINA AL FAHAL/SULTANATE OF OMAN
Tel: 70-77-27, 70-77-23
Telex: 3865 Tawoos On
A,C,M

PAKISTAN

Mushko & Company Ltd.
House No. 16, Street No. 16
Sector F-6/3
ISLAMABAD
Tel: 824545
Telex: 54001 Muski Pk
Cable: FEMUS Islamabad
A,E,P*

Mushko & Company Ltd.
Oosman Chambers
Abdullah Haroon Road
KARACHI 0302
Tel: 524131, 524132
Telex: 2894 MUSKO PK
Cable: COOPERATOR Karachi
A,E,P*

PANAMA

Electronico Balboa, S.A.
Calle Samuel Lewis, Ed. Alfa
Apartado 4929
PANAMA CITY
Tel: 9-011-507-636613
Telex: 368 3483 ELECTRON PG
CM,E,M,P

PERU

Cia Electro Médica S.A.
Los Flamencos 145, Ofc. 301/2
San Isidro
Casilla 1030
LIMA 1
Tel: 9-011-511-4-414325, 41-3705
Telex: 39425257 PE PB SIS
CM,E,M,P

SAMS S.A.
Arenida Republica de Panama 3534
San Isidro, **LIMA**
Tel: 9-011-511-4-229332/413984/413226
Telex: 39420450 PE LIBERTAD
A,C,P

PHILIPPINES

The Online Advanced Systems Corp.
2nd Floor, Electra House
115-117 Esteban Street
P.O. Box 1510
Legaspi Village, Makati
Metro **MANILA**
Tel: 815-38-10 (up to 16)
Telex: 63274 ONLINE PN
A,C,E,M,P

PORTUGAL

Mundinter Intercambio
Mundial de Comércio S.A.R.L.
Av. Antonio Augusto Aguiar 138
Apartado 2761
LISBON
Tel: (19) 53-21-31, 53-21-37
Telex: 16691 munter p
M

Soquimica
Av. da Liberdade, 220-2
1298 **LISBOA** Codex
Tel: 56-21-82
Telex: 13316 SABASA
A

Telectra-Empresa Técnica de
Equipmentos Eléctricos S.A.R.L.
Rua Rodrigo da Fonseca 103
P.O. Box 2531
LISBON 1
Tel: (19) 68-60-72
Telex: 12598
CM,E

C.P.C.S.I.
Rua de Costa Cabral 575
4200 **PORTO**
Tel: 499174/495173
Telex: 26054
C,P

PUERTO RICO

Hewlett-Packard Puerto Rico
101 Muñoz Rivera Av
Esu. Calle Ochoa
HATO REY, Puerto Rico 00918
Tel: (809) 754-7800
A,C,CM,M,E,P

QATAR

Computer Arabia
P.O. Box 2750
DOHA
Tel: 428555
Telex: 4806 CHPARB
P

Nasser Trading & Contracting
P.O.Box 1563
DOHA
Tel: 422170
Telex: 4439 NASSER DH
M

SAUDI ARABIA

Modern Electronics Establishment
Hewlett-Packard Division
P.O. Box 281
Thuobah
AL-KHOBAR 31952
Tel: 895-1760, 895-1764
Telex: 671 106 HPMEEK SJ
Cable: ELECTA AL-KHOBAR
C,E,M

Modern Electronics Establishment
Hewlett-Packard Division
P.O. Box 1228
Redec Plaza, 6th Floor
JEDDAH
Tel: 644 96 28
Telex: 4027 12 FARNAS SJ
Cable: ELECTA JEDDAH
A,C,CM,E,M,P

Modern Electronics Establishment
Hewlett-Packard Division
P.O.Box 22015
RIYADH 11495
Tel: 491-97 15, 491-63 87
Telex: 202049 MEERYD SJ
C,E,M

Abdul Ghani El Ajou Corp.
P.O. Box 78
RIYADH
Tel: 40 41 717
Telex: 200 932 EL AJOU
P

SCOTLAND
See United Kingdom

SENEGAL

Societe Hussein Ayad & Cie.
76, Avenue Georges Pompidou
B.P. 305
DAKAR
Tel: 32339
Cable: AYAD-Dakar
E

Moneger Distribution S.A.
1, Rue Parent
B.P. 148
DAKAR
Tel: 215 671
Telex: 587
P

Systeme Service Conseil (SSC)
14, Avenue du Parachois
DAKAR ETOILE
Tel: 219976
Telex: 577
C,P

SINGAPORE

Hewlett-Packard Singapore (Sales)
Pte. Ltd.
1150 Depot Road
SINGAPORE, 0410
Tel: 4731788
Telex: 34209 HPSGSO RS
Cable: HEWPACK, Singapore
A,C,E,M,P

Dynamar International Ltd.
Unit 05-11 Block 6
Kolam Ayer Industrial Estate
SINGAPORE 1334
Tel: 747-6188
Telex: 26283 RS
CM

SOUTH AFRICA

Hewlett-Packard So Africa (Pty.) Ltd.
P.O. Box 120
Howard Place, **CAPE PROVINCE**
7450 South Africa
Tel: 27 121153-7954
Telex: 57-20006
A,C,CM,E,M,P

Hewlett-Packard So Africa (Pty.) Ltd.
2nd Floor Juniper House
92 Overport Drive
DURBAN 4067
Tel: 27-31-28-4178
Telex: 6-22954
C

Hewlett-Packard So Africa (Pty.) Ltd.
Shop 6 Linton Arcade
511 Cape Road
Linton Grange
PORT ELIZABETH 6001
Tel: 27141130 1201
Telex: 24-2916
C

Hewlett-Packard So Africa (Pty.) Ltd.
Fountain Center
Kalkoen Str.
Monument Park Ext 2
PRETORIA 0105
Tel: (012) 45 5725
Telex: 32163
C,E

Hewlett-Packard So Africa (Pty.) Ltd.
Private Bag Wendywood
SANDTON 2144
Tel: 27-11-802-5111, 27-11-802-5125
Telex: 4-20877 SA
Cable: HEWPACK Johannesburg
A,C,CM,E,M,P

SPAIN

Hewlett-Packard Española, S.A.
Calle Entenza, 321
E.-BARCELONA 29
Tel: 3/322 24 51, 321 73 54
Telex: 52603 hpbee
A,C,E,M,P

Hewlett-Packard Española, S.A.
Calle San Vicente S/N
Edificio Albia II-7B
48001 **BILBAO**
Tel: 4/423 83 06
A,C,E,M

Hewlett-Packard Española, S.A.
Crta. N-VI, Km. 16, 400
Las Rozas
E-MADRID
Tel: (1) 637.00.11
Telex: 23515 HPE
C,M

Hewlett-Packard Española, S.A.
Avda. S. Francisco Javier, S/N
Planta 10. Edificio Sevilla 2
E-SEVILLA 5, SPAIN
Tel: 54/64 44 54
Telex: 72933
A,C,M,P

Hewlett-Packard Española, S.A.
Isabel La Catolica, 8
E-46004 **VALENCIA**
Tel: 34-6-361 1354
Telex: 63435
C,P

Hewlett-Packard Española, S.A.
Av. de Zugazarte, 8
Las Arenas-Guecho
E-48930 **VIZCAYA**
VIZCAYA
Tel: 34-423-83 06
Telex: 33032

SWEDEN

Hewlett-Packard Sverige AB
Östra Tullgatan 3
S-20011 **MALMÖ**
Box 6132
Tel: 46-40-702-70
Telex: (854) 17886 (via Spånga office)
C,P

Hewlett-Packard Sverige AB
Elementvagen 16
S-7022 7 **ÖREBRO**
Tel: 49-019-10-4820
Telex: (854) 17886 (via Spånga office)
C

Hewlett-Packard Sverige AB
Skalholtsgatan 9, Kista
P.O. Box 19
S-16393 **SPÅNGA**
Tel: (08) 750-2000
Telex: (854) 17886
Telefax: (08) 7527781
A,C,CM,E,M,P

Hewlett-Packard Sverige AB
Box 266
Topasgatan 1A
S-42123 **VÄSTRA-FRÖLUNDA** (Gothenburg)
Tel: 46-031-89-1000
Telex: (854) 17886 (via Spånga office)
A,C,CM,E,M,P

SUDAN

Mediterranean Engineering
& Trading Co. Ltd.
P.O. Box 1025
KHARTOUM
Tel: 41184
Telex: 24052
C,P

SWITZERLAND

Hewlett-Packard (Schweiz) AG
Clarastrasse 12
CH-4058 **BASEL**
Tel: 41-61-33-5920
A,C,E,P

Hewlett-Packard (Schweiz) AG
7, rue du Bois-du-Lan
Case postale 365-1366
CH-1217 **MEYRIN** 1
Tel: (0041) 22-83-11-11
Telex:27333 HPAG CH
A,C,CM,E,M,P

SWITZERLAND (Cont'd)

Hewlett-Packard (Schweiz) AG
Allmend 2
CH-8967 **WIDEN**
Tel: 41-57-31-2111
Telex: 53933 hpag ch
Cable: HPAG CH
A,C,CM,E,M,P

Hewlett-Packard (Schweiz) AG
Schwamendingenstrasse 10
CH-8050 **ZURICH**
Tel: 41-1-315-8181
Telex: 823 537 HPAG CH
C,P

SYRIA

General Electronic Inc.
Nuri Basha Ahnaf Ebn Kays Street
P.O. Box 5781
DAMASCUS
Tel: 33-24-87
Telex: 44-19-88
Cable: ELECTROBOR DAMASCUS
E

Middle East Electronics
P.O.Box 2308
Abu Rumaneh
DAMASCUS
Tel: 33 45 92
Telex: 411 771 Meesy
M

TAIWAN

Hewlett-Packard Taiwan Ltd.
THM Office
2, Huan Nan Road
CHUNG LI, Taoyuan
Tel: (034) 929-666
C

Hewlett-Packard Taiwan Ltd.
Kaohsiung Office
11/F, 456, Chung Hsiao 1st Road
KAOHSIUNG
Tel: (07) 2412318
C,E

Hewlett-Packard Taiwan Ltd.
8th Floor, Hewlett-Packard Building
337 Fu Hsing North Road
TAIPEI
Tel: (02) 712-0404
Telex: 24439 HEWPACK
Cable:HEWPACK Taipei
A,C,CM,E,M,P

Ing Lih Trading Co.
3rd Floor, No. 7, Sect. 2
Jen Ai Road
TAIPEI 100
Tel: (02) 394-8191
Telex: 22894 SANKWANG
A

THAILAND

Unimesa Co. Ltd.
30 Patpong Ave., Suriwong
BANGKOK 5,
Tel: 235-5727, 234-0991/3
Telex: 84439 Simonco TH
Cable: UNIMESA Bangkok
A,C,E,M

Bangkok Business Equipment Ltd.
5/3-6 Dejo Road
BANGKOK
Tel: 234-8670, 234-8671
Telex: 87699-BEQUIPT TH
Cable: BUSIQUIPT Bangkok
P

TOGO

Societe Africaine De Promotion
Immeuble Sageb
Rue d'Atakpame
P.O. Box 4150
LOME
Tel: 21-62-88
Telex: 5357
P

TRINIDAD & TOBAGO

Caribbean Telecoms Ltd.
Corner McAllister Street &
Eastern Main Road, Laventille
P.O. Box 732
PORT-OF-SPAIN
Tel: 624-4213
Telex: 22561 CARTEL WG
Cable: CARTEL, PORT OF SPAIN
CM,E,M,P

Computer and Controls Ltd.
P.O. Box 51
1 Taylor Street
PORT-OF-SPAIN
Tel: (809) 622-7719/622-7985
Telex: 38722798 COMCON WG
LOOGO AGENCY 1264
A,P

Feral Assoc.
8 Fitzgerald Lane
PORT-OF-SPAIN
Tel: 62-36864, 62-39255
Telex: 22432 FERALCO
Cable: FERALCO
M

TUNISIA

Tunisie Electronique S.A.R.L.
31 Avenue de la Liberte
TUNIS
Tel: 280-144
C,E,P

Tunisie Electronique S.A.R.L.
94, Av. Jugurtha, Mutuelleville
1002 **TUNIS-BELVEDERE**
Tel: 280144
Telex: 13238
C,E,P

Corema S.A.
1 ter. Av. de Carthage
TUNIS
Tel: 253-821
Telex: 12319 CABAM TN
M

TURKEY

E.M.A
Mediha Eldem Sokak No. 41/6
Yenisehir
ANKARA
Tel: 319175
Telex: 42321 KTX TR
Cable: EMATRADE ANKARA
M

Teknim Company Ltd.
Iran Caddesi No. 7
Karaklidere
ANKARA
Tel: 275800
Telex: 42155 TKNM TR
C,E

Kurt & Kurt A.S.
Mithatpasa Caddesi No. 75
Kat 4 Kizilay
ANKARA
Tel: 318875/6/7/8
Telex: 42490 MESR TR
A

Saniva Bilgisayar Sistemleri A.S.
Buyukdere Caddesi 103/6
Gayrettepe
ISTANBUL
Tel: 1673180
Telex: 26345 SANI TR
C,P

Best Inc.
Esentepe, Gazeteciler Sitesi
Keskin Kalem
Sokak 6/3, Gayrettepe
ISTANBUL
Tel: 172 1328, 173 3344
Telex: 42490
A

UNITED ARAB EMIRATES

Emitac Ltd.
P.O. Box 1641
SHARJAH
Tel: 591181
Telex: 68136 EMITAC EM
Cable: EMITAC SHARJAH
E,C,M,P,A

Emitac Ltd.
P.O. Box 2711
ABU DHABI
Tel: 820419-20
Cable: EMITACH ABUDHABI

Emitac Ltd.
P.O. Box 8391
DUBAI,
Tel: 377591

Emitac Ltd.
P.O. Box 473
RAS AL KHAIMAH
Tel: 28133, 21270

UNITED KINGDOM
ENGLAND

Hewlett-Packard Ltd.
Miller House
The Ring, **BRACKNELL**
Berks RG12 1XN
Tel: 44/344/424-898
Telex: 848733
E

Hewlett-Packard Ltd.
Elstree House, Elstree Way
BOREHAMWOOD, Herts WD6 1SG
Tel: 01 207 5000
Telex: 8952716
C,E

Hewlett-Packard Ltd.
Oakfield House, Oakfield Grove
Clifton **BRISTOL**, Avon BS8 2BN
Tel: 44-272-736 806
Telex: 444302
C,E,P

Hewlett-Packard Ltd.
9 Bridewell Place
LONDON EC4V 6BS
Tel: 44-01-583-6565
Telex: 298163
C,P

Hewlett-Packard Ltd.
Pontefract Road
NORMANTON, West Yorkshire WF6 1RN
Tel: 44/924/895 566
Telex: 557355
C,P

Hewlett-Packard Ltd.
The Quadrangle
106-118 Station Road
REDHILL, Surrey RH1 1PS
Tel: 44-737-686-55
Telex: 947234
C,E,P

Hewlett-Packard Ltd.
Avon House
435 Stratford Road
Shirley, **SOLIHULL**, West Midlands
B90 4BL
Tel: 44-21-745-8800
Telex: 339105
C,E,P

Hewlett-Packard Ltd.
Heathside Park Road
Cheadle Heath, Stockport
SK3 ORB, United Kingdom
Tel: 44-061-428-0828
Telex: 668068
A,C,E,M,P

Hewlett-Packard Ltd.
Harmon House
No. 1 George Street
UXBRIDGE, Middlesex UX8 1YH
Tel: 895 720 20
Telex: 893134/5
C,CM,E,M,P

Hewlett-Packard Ltd.
King Street Lane
Winnersh, **WOKINGHAM**
Berkshire RG11 5AR
Tel: 44/734/784774
Telex: 8471789
A,C,E,M,P

NORTHERN IRELAND

Hewlett-Packard (Ireland) Ltd.
Carrickfergus Industrial Centre
75 Belfast Road, Carrickfergus
CO. ANTRIM BT38 8PM
Tel: 09603 67333
C,E

Cardiac Services Company
95A Finaghy Road South
BELFAST, BT10 OBY
Tel: 0232-625566
Telex: 747626
M

SCOTLAND

Hewlett-Packard Ltd.
1/3 Springburn Place
College Milton North
EAST KILBRIDE, G74 5NU
Tel: 041-332-6232
Telex: 779615
C,E

Hewlett-Packard Ltd.
SOUTH QUEENSFERRY
West Lothian, EH30 9TG
Tel: 031 331 1188
Telex: 72682 HPSQFYG
C,CM,E,M,P

UNITED STATES

Hewlett-Packard Co.
Customer Information Center
Tel: (800) 752-0900
Hours: 6:00 AM to 5:00 PM
Pacific Time

Alabama

Hewlett-Packard Co.
2100 Riverchase Center
Building 100 - Suite 118
BIRMINGHAM, AL 35244
Tel: (205) 988-0547
A,C,M,P*

Hewlett-Packard Co.
420 Wynn Drive
HUNTSVILLE, AL 35805
Tel: (205) 830-2000
C,CM,E,M*

Alaska

Hewlett-Packard Co.
4000 Old Seward Highway
Suite 101
ANCHORAGE, AK 99503
Tel: (907) 563-8855
C,E

Arizona

Hewlett-Packard Co.
8080 Pointe Parkway West
PHOENIX, AZ 85044
Tel: (602) 273-8000
A,C,CM,E,M,P

Hewlett-Packard Co.
3400 East Britannia Dr.
Bldg. C, Suite 124
TUCSON, AZ 85706
Tel: (602) 573-7400
C,E,M**

California

Hewlett-Packard Co.
99 South Hill Dr.
BRISBANE, CA 94005
Tel: (415) 330-2500
C

Hewlett-Packard Co.
1907 North Gateway Blvd.
FRESNO, CA 93727
Tel: (209) 252-9652
C,M

Hewlett-Packard Co.
1421 S. Manhattan Av.
FULLERTON, CA 92631
Tel: (714) 999-6700
C,CM,E,M

Hewlett-Packard Co.
7408 Hollister Ave. #A
GOLETA, CA 93117
Tel: (805) 685-6100
C,E

Hewlett-Packard Co.
2525 Grand Avenue
LONG BEACH, CA 90815
Tel: (213) 498-1111
C

Hewlett-Packard Co.
5651 West Manchester Ave.
LOS ANGELES, CA 90045
Tel: (213) 337-8000

Hewlett-Packard Co.
3155 Porter Drive
PALO ALTO, CA 94304
Tel: (415) 857-8000
C,E

Hewlett-Packard Co.
5725 W. Las Positas Blvd.
PLEASANTON, CA 94566
Tel: (415) 460-0282
C

Hewlett-Packard Co.
4244 So. Market Court, Suite A
SACRAMENTO, CA 95834
Tel: (916) 929-7222
A*,C,E,M

Hewlett-Packard Co.
9606 Aero Drive
SAN DIEGO, CA 92123
Tel: (619) 279-3200
C,CM,E,M

Hewlett-Packard Co.
3003 Scott Boulevard
SANTA CLARA, CA 95054
Tel: (408) 988-7000
Telex: 910-338-0586
A,C,CM,E

Hewlett-Packard Co.
2150 W. Hillcrest Dr.
THOUSAND OAKS, CA 91320
(805) 373-7000
C,CM,E

Colorado

Hewlett-Packard Co.
2945 Center Green Court South
Suite A
BOULDER, CO 80301
Tel: (303) 499-6655
A,C,E

Hewlett-Packard Co.
24 Inverness Place, East
ENGLEWOOD, CO 80112
Tel: (303) 649-5000
A,C,CM,E,M

Connecticut

Hewlett-Packard Co.
500 Sylvan Av.
BRIDGEPORT, CT 06606
Tel: (203) 371-6454
C,E

Hewlett-Packard Co.
47 Barnes Industrial Road South
WALLINGFORD, CT 06492
Tel: (203) 265-7801
A,C,CM,E,M

Florida

Hewlett-Packard Co.
2901 N.W. 62nd Street
FORT LAUDERDALE, FL 33309
Tel: (305) 973-2600
C,E,M,P*

Hewlett-Packard Co.
6800 South Point Parkway
Suite 301
JACKSONVILLE, FL 32216
Tel: (904) 636-9955
C*,M**

Hewlett-Packard Co.
255 East Drive, Suite B
MELBOURNE, FL 32901
Tel: (305) 729-0704
CM,E

Hewlett-Packard Co.
6177 Lake Ellenor Drive
ORLANDO, FL 32809
Tel: (305) 859-2900
A,C,CM,E,P*

Hewlett-Packard Co.
4700 Bayou Blvd.
Building 5
PENSACOLA, FL 32503
Tel: (904) 476-8422
A,C,M

Hewlett-Packard Co.
5550 W. Idlewild, #150
TAMPA, FL 33614
Tel: (813) 884-3282
C,E,M,P

Georgia

Hewlett-Packard Co.
2015 South Park Place
ATLANTA, GA 30339
Tel: (404) 955-1500
Telex: 810-766-4890
A,C,CM,E,M,P*

Hewlett-Packard Co.
3607 Parkway Lane
Suite 300
NORCROSS, GA 30092
Tel: (404) 448-1894
C,E,P

Hawaii

Hewlett-Packard Co.
Pacific Tower
1001 Bishop St.
Suite 2400
HONOLULU, HI 96813
Tel: (808) 526-1555
A,C,E,M

Idaho

Hewlett-Packard Co.
11309 Chinden Blvd.
BOISE, ID 83714
Tel: (208) 323-2700
C

Illinois

Hewlett-Packard Co.
2205 E. Empire St.
P.O. Box 1607
BLOOMINGTON, IL 61702-1607
Tel: (309) 662-9411
A,C,E,M**

Hewlett-Packard Co.
525 W. Monroe, #1308
CHICAGO, IL 60606
Tel: (312) 930-0010
C

Hewlett-Packard Co.
1200 East Diehl Road
NAPERVILLE, IL 60566
Tel: (312) 357-8800
C

Hewlett-Packard Co.
5201 Tollview Drive
ROLLING MEADOWS, IL 60008
Tel: (312) 255-9800
Telex: 910-687-1066
A,C,CM,E,M

Indiana

Hewlett-Packard Co.
11911 N. Meridian St.
CARMEL, IN 46032
Tel: (317) 844-4100
A,C,CM,E,M

Hewlett-Packard Co.
111 E. Ludwig Road
Suite 108
FT. WAYNE, IN 46825
Tel: (219) 482-4283
C,E

Iowa

Hewlett-Packard Co.
4070 22nd Av. SW
CEDAR RAPIDS, IA 52404
Tel: (319) 390-4250
C,E,M

Hewlett-Packard Co.
4201 Corporate Dr.
WEST DES MOINES, IA 50265
Tel: (515) 224-1435
A**,C,M**

Kansas

Hewlett-Packard Co.
North Rock Business Park
3450 N. Rock Rd.
Suite 300
WICHITA, KS 67226
Tel: (316) 684-8491
C,E

Kentucky

Hewlett-Packard Co.
305 N. Hurstbourne Lane,
Suite 100
LOUISVILLE, KY 40223
Tel: (502) 426-0100
A,C,M

Louisiana

Hewlett-Packard Co.
160 James Drive East
ST. ROSE, LA 70087
P.O. Box 1449
KENNER, LA 70063
Tel: (504) 467-4100
A,C,E,M,P

Maryland

Hewlett-Packard Co.
3701 Koppers Street
BALTIMORE, MD 21227
Tel: (301) 644-5800
Telex: 710-862-1943
A,C,CM,E,M

Hewlett-Packard Co.
2 Choke Cherry Road
ROCKVILLE, MD 20850
Tel: (301) 948-6370
A,C,CM,E,M

Massachusetts

Hewlett-Packard Co.
1775 Minuteman Road
ANDOVER, MA 01810
Tel: (617) 682-1500
A,C,CM,E,M,P*

Hewlett-Packard Co.
29 Burlington Mall Rd
BURLINGTON, MA 01803-4514
Tel: (617) 270-7000
C,E

Michigan

Hewlett-Packard Co.
4326 Cascade Road S.E.
GRAND RAPIDS, MI 49506
Tel: (616) 957-1970
C,M

Hewlett-Packard Co.
39550 Orchard Hill Place Drive
NOVI, MI 48050
Tel: (313) 349-9200
A,C,E,M

Hewlett-Packard Co.
560 Kirts Rd.
Suite 101
TROY, MI 48084
Tel: (313) 362-5180
C

Minnesota

Hewlett-Packard Co.
2025 W. Larpenteur Ave.
ST. PAUL, MN 55113
Tel: (612) 644-1100
A,C,CM,E,M

Missouri

Hewlett-Packard Co.
1001 E. 101st Terrace Suite 120
KANSAS CITY, MO 64131-3368
Tel: (816) 941-0411
A,C,CM,E,M

Hewlett-Packard Co.
13001 Hollenberg Drive
BRIDGETON, MO 63044
Tel: (314) 344-5100
A,C,E,M

Nebraska

Hewlett-Packard
11626 Nicholas St.
OMAHA, NE 68154
Tel: (402) 493-0300
C,E,M

New Jersey

Hewlett-Packard Co.
120 W. Century Road
PARAMUS, NJ 07652
Tel: (201) 265-5000
A,C,CM,E,M

Hewlett-Packard Co.
20 New England Av. West
PISCATAWAY, NJ 08854
Tel: (201) 562-6100
A,C,CM,E

New Mexico

Hewlett-Packard Co.
7801 Jefferson N.E.
ALBUQUERQUE, NM 87109
Tel: (505) 823-6100
C,E,M

Hewlett-Packard Co.
1362-C Trinity Dr.
LOS ALAMOS, NM 87544
Tel: (505) 662-6700
C,E

New York

Hewlett-Packard Co.
5 Computer Drive South
ALBANY, NY 12205
Tel: (518) 458-1550
A,C,E,M

Hewlett-Packard Co.
9600 Main Street
CLARENCE, NY 14031
Tel: (716) 759-8621
C,E,M

Hewlett-Packard Co.
200 Cross Keys Office Park
FAIRPORT, NY 14450
Tel: (716) 223-9950
A,C,CM,E,M

Hewlett-Packard Co.
7641 Henry Clay Blvd.
LIVERPOOL, NY 13088
Tel: (315) 451-1820
A,C,CM,E,M

Hewlett-Packard Co.
No. 1 Pennsylvania Plaza
55th Floor
34th Street & 7th Avenue
MANHATTAN NY 10119
Tel: (212) 971-0800
C,M*

Hewlett-Packard Co.
15 Myers Corner Rd.
Hollowbrook Park, Suite 2D
WAPPINGERS FALLS, NY 12590
Tel: (914) 298-9125
CM,E

Hewlett-Packard Co.
2975 Westchester Ave
PURCHASE, NY 10577
Tel: (914) 935-6300
C,CM,E

Hewlett-Packard Co.
3 Crossways Park West
WOODBURY, NY 11797
Tel: (516) 682-7800
A,C,CM,E,M

North Carolina

Hewlett-Packard Co.
305 Gregson Dr.
CARY, NC 27511
Tel: (919) 467-6600
C,CM,E,M,P*

Hewlett-Packard Co.
9401 Arrow Point Blvd
Suite 100
CHARLOTTE, NC 28217
Tel: (704) 527-8780
C*

Hewlett-Packard Co.
5605 Roanne Way
GREENSBORO, NC 27420
Tel: (919) 852-1800
A,C,CM,E,M,P*

Ohio

Hewlett-Packard Co.
2717 S. Arlington Road
AKRON, OH 44312
Tel: (216) 644-2270
C,E

Hewlett-Packard Co.
4501 Erskine Road
CINCINNATI, OH 45242
Tel: (513) 891-9870
C,M

Hewlett-Packard Co.
15885 Sprague Road
CLEVELAND, OH 44136
Tel: (216) 243-7300
A,C,CM,E,M

Hewlett-Packard Co.
9080 Springboro Pike
MIAMISBURG, OH 45342
Tel: (513) 433-2223
A,C,CM,E*,M

Hewlett-Packard Co.
One Maritime Plaza, 5th Floor
720 Water Street
TOLEDO, OH 43604
Tel: (419) 242-2200
C

Hewlett-Packard Co.
675 Brooksedge Blvd.
WESTERVILLE, OH 43081
Tel: (614) 891-3344
C,CM,E*

Oklahoma

Hewlett-Packard Co.
3525 N.W. 56th St.
Suite C-100
OKLAHOMA CITY, OK 73112
Tel: (405) 946-9499
C,E*,M

SALES & SUPPORT OFFICES
Arranged alphabetically by country

UNITED STATES (Cont'd)

Hewlett-Packard Co.
6655 South Lewis,
Suite 105
TULSA, OK 74136
Tel: (918) 481-6700
A**,C,E,M*,P*

Oregon

Hewlett-Packard Co.
9255 S. W. Pioneer Court
WILSONVILLE, OR 97070
Tel: (503) 682-8000
A,C,E*,M

Pennsylvania

Hewlett-Packard Co.
Heatherwood Industrial Park
50 Dorchester Rd.
Route 22
HARRISBURG, PA 17112-2799
Tel: (717) 657-5900
C

Hewlett-Packard Co.
111 Zeta Drive
PITTSBURGH, PA 15238
Tel: (412) 782-0400
A,C,E,M

Hewlett-Packard Co.
2750 Monroe Boulevard
VALLEY FORGE, PA 19482
Tel: (215) 666-9000
A,C,CM,E,M

South Carolina

Hewlett-Packard Co.
Brookside Park, Suite 122
1 Harbison Way
COLUMBIA, SC 29212
Tel: (803) 732-0400
C,M

Hewlett-Packard Co.
545 N. Pleasantburg Dr.
Suite 100
GREENVILLE, SC 29607
Tel: (803) 232-8002
C

Tennessee

Hewlett-Packard Co.
One Energy Centr. Suite 200
Pellissippi Pkwy.
KNOXVILLE, TN 37932
Tel: (615) 966-4747
A,C,E,M,P

Hewlett-Packard Co.
3070 Directors Row
Directors Square
MEMPHIS, TN 38131
Tel: (901) 346-8370
A,C,E,M

Hewlett-Packard Co.
44 Vantage Way,
Suite 160
NASHVILLE, TN 37228
Tel: (615) 255-1271
A,C,E,M,P

Texas

Hewlett-Packard Co.
1826-P Kramer Lane
AUSTIN, TX 78758
Tel: (512) 835-6771
C,E,P*

Hewlett-Packard Co.
5700 Cromo Dr
EL PASO, TX 79912
Tel: (915) 833-4400
C,E*,M**

Hewlett-Packard Co.
3952 Sandshell Drive
FORT WORTH, TX 76137
Tel: (817) 232-9500
C

Hewlett-Packard Co.
10535 Harwin Drive
HOUSTON, TX 77036
Tel: (713) 776-6400
A,C,E,M,P*

Hewlett-Packard Co.
3301 West Royal Lane
IRVING, TX 75063
Tel: (214) 869-3377
C,E

Hewlett-Packard Co.
109 E. Toronto, Suite 100
McALLEN, TX 78501
Tel: (512) 630-3030
C

Hewlett-Packard Co.
930 E. Campbell Rd.
RICHARDSON, TX 75081
Tel: (214) 231-6101
A,C,CM,E,M,P*

Hewlett-Packard Co.
1020 Central Parkway South
SAN ANTONIO, TX 78232
Tel: (512) 494-9336
A,C,E,M,P*

Utah

Hewlett-Packard Co.
3530 W. 2100 South St.
SALT LAKE CITY, UT 84119
Tel: (801) 974-1700
A,C,E,M

Virginia

Hewlett-Packard Co.
840 Greenbrier Circle
Suite 101
CHESAPEAKE, VA 23320
Tel: (804) 424-7105
C,E,M

Hewlett-Packard Co.
4305 Cox Road
GLEN ALLEN, VA 23060
Tel: (804) 747-7750
A,C,E,M,P*

Hewlett-Packard Co.
Tanglewood West Bldg.
Suite 240
3959 Electric Road
ROANOKE, VA 24018
Tel: (703) 774-3444
C,E,P

Washington

Hewlett-Packard Co.
15815 S.E. 37th Street
BELLEVUE, WA 98006
Tel: (206) 643-4004
A,C,CM,E,M

Hewlett-Packard Co.
1225 Argonne Rd
SPOKANE, WA 99212
Tel: (509) 922-7000
C

West Virginia

Hewlett-Packard Co.
501 56th Street
CHARLESTON, WV 25304
Tel: (304) 925-0492
A,C,M

Wisconsin

Hewlett-Packard Co.
275 N. Corporate Dr.
BROOKFIELD, WI 53005
Tel: (414) 784-8800
A,C,E*,M

URUGUAY

Pablo Ferrando S.A.C. e I.
Avenida Italia 2877
Casilla de Correo 370
MONTEVIDEO
Tel: 59-82-802-586
Telex: 398802586
A,CM,E,M

Olympia de Uruguay S.A.
Maquines de Oficina
Avda. del Libertador 1997
Casilla de Correos 6644
MONTEVIDEO
Tel: 91-1809, 98-3807
Telex: 6342 OROU UY
P

VENEZUELA

Hewlett-Packard de Venezuela C.A.
3A Transversal Los Ruices Norte
Edificio Segre 2 & 3
Apartado 50933
CARACAS 1050
Tel: (582) 239-4133
Telex: 251046 HEWPACK
A,C,CM,E,M,P

Hewlett-Packard de Venezuela, C.A.
Centro Ciudad Comercial Tamanaco
Nivel C-2 (Nueva Etapa)
Local 53H05
Chuao, **CARACAS**
Tel: 928291
P

Albis Venezolana S.R.L.
Av. Las Marias, Ota. Alix,
El Pedregal
Apartado 81025
CARACAS 1080A
Tel: 747984, 742146
Telex: 24009 ALBIS VC
A

Tecnologica Medica del Caribe, C.A.
Multicentro Empresarial del Este
Ave. Libertador
Edif. Libertador
Nucleo "C" - Oficina 51-52
CARACAS
Tel: 339867/333780
M

Hewlett-Packard de Venezuela C.A.
Residencias Tia Betty Local 1
Avenida 3 y con Calle 75
MARACAIBO, Estado Zulia
Apartado 2646
Tel: 58-2-617-5669
Telex: 62464 HPMAR
C,E*

Hewlett-Packard de Venezuela C.A.
Urb. Lomas de Este
Torre Trebol — Piso 11
VALENCIA, Estado Carabobo
Apartado 3347
Tel: (5841) 222992
C,P

YUGOSLAVIA

Do Hermes
General Zdanova 4
YU-11000 **BEOGRAD**
Tel: (011) 342 641
Telex: 11433
A,C,E,M,P

Do Hermes
Celovska 73
YU-61000 **LJUBLJANA**
Tel: (061) 553 170
Telex: 31583
A,C,E,M,P

Elektrotehna
Titova 51
YU-61000 **LJUBLJANA**
CM

Do Hermes
Kralja Tomislava 1
YU-71000 **SARAJEVO**
Tel: (071) 35 859
Telex: 41634
C**,P

ZAIRE

Computer & Industrial Engineering
25, Avenue de la Justice
B.P. 12797
KINSHASA, Gombe
Tel: 32063
Telex: 21552
C,P

ZAMBIA

R.J. Tilbury (Zambia) Ltd.
P.O. Box 32792
LUSAKA
Tel: 215590
Telex: 40128
E

ZIMBABWE

Field Technical Sales (Private) Limited
45, Kelvin Road North
P.O. Box 3458
SALISBURY
Tel: 705 231
Telex: 4-122 RH
E,P

September 1987

Part No. 92432-90001
Printed in U.S.A. November 1988
E1188

HEWLETT PACKARD