HP 64000
Logic Development
System

Pascal/64000
Compiler Supplement
6800

HEWLETT
PACKARD

CERTIFICATION

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

WARRANTY

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

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

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

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

LIMITATION OF WARRANTY

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

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

EXCLUSIVE REMEDIES

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

ASSISTANCE

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

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

NOTICE

Attached to this software notice is a summary of problems and solutions for the 6800 Pascal compiler that you may or may not encounter. Use this summary with the manual you received with the product. In the one-line description at the top of each problem and solution, there is a software topic or manual chapter reference.

KPR #: D200033670 Product: 6800 PASCAL M64811-90902 01.03

Keywords: TYPE CONVERSION

One-line description:
BYTE constants lose their "BYTENESS" when added. (See Chap 2, pg 2-32)

Problem:

```
$EXTENSIONS;RANGE$
CONST C1 = BYTE(80H);   (*-128*)
      C2 = BYTE (1H);   (* + 1*)
VAR   B1 : BYTE;

BEGIN
B1 := C1 + C2;
{IF $RANGE$ IS ON, THIS CAUSES A CALL TO A WORD-SIZED, BOUNDS-CHECKING
ROUTINE PASSING 0081H AND THE VALUE IN QUESTION.  THIS PASSED VALUE IS
WRONG BECAUSE IT IS NOT SIGN-EXTENDED AND THE UPPER-BOUNDS IS 007FH.}
```

Solution:
$RANGE OFF$ around the code; or better yet use:

```
B1 := BYTE(C1+C2);  {THIS CALLS THE BYTE-SIZED BOUNDS-CHECKER.}
```

KPR #: D200016717 Product: 6800 PASCAL M64811-90902 01.03

Keywords: LINKER

One-line description:
Error/Warning msgs not written to lnk list file. (See Ch 10, pg 10-13)

Problem:

```
"6800"
PROGRAM F1;
$EXTVAR$      {DECLARE THE FOLLOWING TO BE EXTERNAL}
004VAR A,B,C,D:INTEGER {ANY VARIABLES WILL DO}
BEGIN ; END.

"6800"
PROGRAM F2;
VAR A,B,C,D:INTEGER; {DON'T DECLARE THEM TO BE GLOBALS SO WE CAN
                      CREATE A LINKER ERROR.}
BEGIN ;END.
```

Compile these two files and link them (it doesn't matter what the load values are). A cryptic message that an error occured will appear on the screen, but no error messages appear in the linker listing file.

Solution:
To preserve the error messages in a file, do the following:

```
$ assign/user FS.ERR SYS$ERROR   !This will redirect the error output
     !to the file FS.ERR for the next single following command.
$ lnk /output FS.K     !"/OUTPUT" will cause an output listing file,
     !(see HELP LNK /OUTPUT). FS.K is the linker command file and must
     !be included. If a linker command file is not specified, the
     !normal interactive mode will not work with SYS$ERROR redirected.
```

KPR #: 5000084921 Product: 6800 PASCAL M64811-90902 01.03

Keywords: RANGE CHECKING

One-line description:
Parameter range err in LONGREAL_SQRT gives wrong err. (See Ch 4, pg 4-5)

Problem:
Parameter range error in LONGREAL_SQRT gives wrong error.

Page 4-5 of the 6800 Pascal compiler supplement states that the error trap routine REAL_OVERFLOW is called when a floating point operation would PRODUCE an Invalid number. However, the routine LONGREAL_SQRT generates an INVALID error when passed a negative number. It should generate an OVERFLOW (REAL_SQRT generates an OVERFLOW), as INVALID indicates that the parameter passed is not a properly represented floating point number (which it is).

Solution:
Both REAL_SQRT and LONGREAL_SQRT should produce an INVALID indication when given a negative number since this is an invalid operation. For example:

    LONGREAL_SQRT (-1) is invalid

KPR #: 5000084947 Product: 6800 PASCAL M64811-90902 01.03

Keywords: DEBUG LIBRARY

One-line description:
I/O can't output MININT using debug library. (See Chap 1, pg 1-4)

Problem:
Pascal I/O cannot output -32768 using debug library.

The value -32768 cannot be output using WRITE. An overflow error occurs in the Pascal file I/O library routine Pwrite_integer. This routine calls Zintneg to negate the value to be output. The debug library DLIB6800 routine Zintneg returns an overflow error if asked to negate -32768.

Note that is possible to read the value of -32768 using READ.

Solution:
Two possible work-arounds for the above problem are as follows:

1. The above problem is not present if user uses the non-debug libraries: LIB6800:L6800 or SLIB6800:S6800.
2. If the user must use the Debug Library (DLIB6800:D6800), then test integers for -32768 (if possible). Convert -32768 integers to "MININT".

KPR #: D200030551 Product: 6800 PASCAL M64811-90902 01.03

Keywords: MANUAL

One-line description:
Manual does not contain an index.

Problem:
The manual does not contain an index.

Solution:
At next revision, the manual will be updated with an index.

KPR #: D200037655 Product: 6800 PASCAL M64811-90902 01.03

Keywords: CONSTANTS

One-line description:
Compile-time CONSTants limited becuase of file I/O and real numbers.

Problem:
The use of FILE I/O and Real Numbers cause more limitations on the compile time constants.

Solution:
Rewrite program to minimize the number of compile time CONST required.

FOLD HERE

HEWLETT
PACKARD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD

FIRST CLASS PERMIT NO. 1303 COLORADO SPRINGS, COLORADO

POSTAGE WILL BE PAID BY ADDRESSEE

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

FOLD HERE

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

READER COMMENT SHEET

Operating Manual
Pascal/64000 Compiler Supplement
64811-90902, November 1985

Your comments are important to us. Please answer this questionnaire and return it to us. Circle the number that best describes your answer in questions 1 through 8. Thank you.

1. The information in this book is complete:		
Doesn't cover enough (what more do you need?)	1 2 3 4 5	Covers everything
2. The information in this book is accurate:		
Too many errors	1 2 3 4 5	Exactly right
3. The information in this book is:		
Difficult to find	1 2 3 4 5	Easy to find
4. The Index and Table of Contents are useful:		
Missing or inadequate	1 2 3 4 5	Helpful
5. What about the "how-to" procedures and examples:		
No help	1 2 3 4 5	Very Helpful
Not enough	1 2 3 4 5	Too many
6. What about the writing style:		
Confusing	1 2 3 4 5	Clear
7. What about organization of the book:		
Poor order	1 2 3 4 5	Good order
8. What about the size of the book:		
Too small	1 2 3 4 5	Too big

Comments: ____

____

____

____

____

____

Particular pages with errors? ______________________________
Name: ______________________________
Job title: ______________________________
Company: ______________________________
Address: ______________________________

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

FOLD HERE

HEWLETT
PACKARD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD

FIRST CLASS PERMIT NO. 1303 COLORADO SPRINGS, COLORADO

POSTAGE WILL BE PAID BY ADDRESSEE

HEWLETT-PACKARD

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

FOLD HERE

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

READER COMMENT SHEET

Operating Manual
Pascal/64000 Compiler Supplement
64811-90902, November 1985

Your comments are important to us. Please answer this questionnaire and return it to us. Circle the number that best describes your answer in questions 1 through 8. Thank you.

1. The information in this book is complete:

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

2. The information in this book is accurate:

Too many errors	1 2 3 4 5	Exactly right

3. The information in this book is:

Difficult to find	1 2 3 4 5	Easy to find

4. The Index and Table of Contents are useful:

Missing or inadequate	1 2 3 4 5	Helpful

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

No help	1 2 3 4 5	Very Helpful
Not enough	1 2 3 4 5	Too many

6. What about the writing style:

Confusing	1 2 3 4 5	Clear

7. What about organization of the book:

Poor order	1 2 3 4 5	Good order

8. What about the size of the book:

Too small	1 2 3 4 5	Too big

Comments: ______________________________

____

____

____

____

____

Particular pages with errors? ______________________________

Name: ______________________________

Job title: ______________________________

Company: ______________________________

Address: ______________________________

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

HEWLETT PACKARD

OPERATING MANUAL

Model 64811A
Pascal/HP 64000
Compiler Supplement
6800

Manual Part No. 64811-90902
E1185

PRINTED: **NOVEMBER 1985**

PRINTING HISTORY

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

The print date changes only when each new edition is published. Minor corrections or additions may be made as the manual is reprinted between editions. Vertical bars in a page margin indicate the location of reprint corrections.

First Edition .... January 1982 (P/N 64811-90902)
Second Edition .... November 1985 (P/N 64811-90902 E1185)

SOFTWARE VERSION NUMBER

Your HP 64000 software is identified with a version number of the form YY.XX. This manual applies to the following:

Model 64811A Version 1.XX (HP 64000 Hosted)

SOFTWARE MATERIALS SUBSCRIPTION

Hewlett-Packard offers a Software Materials Subscription (SMS) to provide timely and comprehensive information for the users of the Model 64000 Logic Development System. This service can maximize the productivity of your HP system by ensuring that the latest product enhancements and software revisions are being utilized.

For complete information about SMS, please contact your nearest Hewlett-Packard sales office.

BACK UP ANY MASTER COPY FLEXIBLE DISC(S)

Before using the flexible disc(s) provided with any HP product, make a work copy. Retain the original disc(s) as the master copy and use the work copy for daily use.

Specific rights to use one copy of the software product(s) are granted for use on a single, stand alone, development station or a cluster of development stations that boot from a single mass storage device.

If your master copy becomes damaged, replacement discs are available through your Hewlett-Packard sales and service office.

TABLE OF CONTENTS

Chapter 1: PASCAL/64000 COMPILER 6800

Chapter 2: PASCAL/64000 PROGRAMMING 6800

TABLE OF CONTENTS (Cont'd)

Chapter 3: RUN-TIME LIBRARY SPECIFICATIONS

TABLE OF CONTENTS (Cont'd)

LIST OF ILLUSTRATIONS

LIST OF TABLES

NOTES

Chapter 1

PASCAL/64000 COMPILIER 6800

INTRODUCTION

General

This compiler supplement is an extension of the Pascal/64000 Compiler Reference Manual. It contains all processor-dependent compiler information for use with the 6800 microprocessor.

This chapter describes compiler features, options and their uses. A brief discussion of the features, capabilities, and limitations of Pascal program development using the emulation is also provided. A more detailed description of Pascal/64000 features for the 6800 microprocessor is presented in Chapter 2.

Chapter 3 of this supplement is a detailed presentation of the run-time libraries required by the 6800 code generator. It is unnecessary reading for users who do not require knowledge of the run-time environment. It is at this level that examples are given which will allow the user to generate "target" code for the 6800 processor which is smaller, faster, etc.

PASCAL PROGRAM DESIGN

Pascal programs should be designed to be as processor and implementation independent as possible, yet certain concessions must be made when the processor has unique characteristics. Programs written to run on a large mainframe computer with megabytes of virtual memory may not run on a 6800 with a maximum of 64k-bytes of addressable memory. Most large mainframe computer implementations have enough memory to allocate a stack area and a heap for dynamic memory allocation with no prompting by the user. In a limited memory system these factors must be communicated to the compiler in some manner. For the 6800, the user must specify the location of the stack and, if needed, the location of a memory pool for dynamic allocation routines. The following sections describe subjects related to programming and compiling Pascal/64000 for the 6800 processor.

HOW TO IMPLEMENT A PROGRAM

The usual process of software generation is as follows:

a. Create a source program file using the editor.

b. Compile the source program.

c. Link the relocatable files.

d. Emulate the absolute file.

e. Debug as necessary.

This chapter will provide insight into each of these processes.

The Source File

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

```
"6800"
PROGRAM Name;
  .             {comments}
  .
CONST
    ...;
    ...;
TYPE
    ...;
    ...;
VAR
    ...;
    ...;
PROCEDURE Procedure_name(Parameter1 : Type);
    .
    .
    BEGIN
      .
      .
      .
    END;
BEGIN
  .
  .             {main program code}
  .
END.
```

When source file editing is complete, it is ready for compilation. Notice in the example form that the first line of the source program specifies the 6800 processor. This first line must be the special compiler directive indicating the processor for which the program was written.

Within a Pascal source program, the compiler only recognizes upper-case keywords, but identifiers may be lower case. When using a 64200 emulator, the global identifiers must begin with an upper-case letter if the user wishes to access these names symbolically during emulation. (During emulation, only emulation command keywords may start with a lower-case letter.)

A sample compiler command would appear as follows:

```
compile <FILE_name> listfile <FILE_list> options xref
```

The compiler output may be in two forms, a relocatable file and a listing file (if specified). Descriptions of these files are as follows:

Relocatable file:	If no errors were detected in the source file (called <FILE_name>:source), a relocatable file (called <FILE_name>:reloc) will be created. This file will be used by the linker to create an executable absolute file.
Listing file:	If a listfile is specified when the compiler is evoked, a file <FILE_list> containing source lines with line numbers, program counter, level numbers, errors and expanded code (if specified) will be generated.

Linking

After all program modules have been compiled (or assembled), the modules may be linked to form an executable absolute file. The compiler generates calls to a set of library routines for commonly used operations such as multiply, divide, comparisons, array referencing, etc. These routines must be linked with the program modules. There are three libraries which may be linked.

The first is a debug library file called DLIB6800:D6800. This library of relocatable procedures contains some extra code to detect errors such as division by 0, or overflow on multiplication. It is recommended that all program development be performed using this library before either of the other libraries is used.

The second library is called LIB6800:L6800. This library, which has only a limited set of error-detection code, should execute faster and take up less space in memory. This library may be linked in place of the debug library after reasonable assurance that the code is error free.

A third library, SLIB6800:S6800, is a special version which does not allow reentrant calls to the library. Programming considerations for the 6800 processor require significant additional run-time memory and time overhead to achieve pure code procedures. A pure code procedure uses no local statically allocated variables and must use safe stack storage for all local variable references not supplied by the user (e.g., EXTERNAL or ABSOLUTE variables).

The SLIB6800 will generally be faster and smaller than either of the first two libraries. When using this library, it is the user's responsibility to prevent any form of reentrant or parallel code execution which could cause a run time nested procedure call to occur. A single execution stream of a Pascal/64000 program will never cause this problem. Since only one logical processor stream is being executed at any time, only one call to a library routine can be active at any time. The typical programming techniques which will cause nested calls are interrupt programs calling library routines or a "parallel" processing stream which shares processor time among logically independent programs. Either of these techniques could cause the library to fail. It is left up to the user to ensure that this situation does not happen when using the static library.

It is important to realize the distinctions between the reentrant requirements of parallel processes encountered in a multi-task or interrupt-driven environment (which are not supported using the SLIB:S6800 run-time library) and the simpler requirements needed for direct recursion of Pascal programs. Direct or indirect recursion of Pascal programs (as may be required to write a recursive algorithm to compute N factorial) is supported by the recursive entry, exit and parameter passing routines in the library SLIB6800:S6800.

The linker is evoked and the questions asked should be answered as follows:

```
link ...

Object files:  MODULE0,MODULE1,MODULE2

Library files: DLIB6800:D6800
Load addresses: PROG,DATA,COMN = 00000H,00000H,00000H
.
.
.

Absolute file name:  PROGRAM
```

In the link listfile, the library routines that are referenced by the compiled code are linked at the end of the last user relocatable PROG and/or DATA areas. This fact must be considered for the proper choice of the stack pointer location, and PROG and DATA link addresses.

Linking with Real Numbers

When using real numbers for the 6800, the user must link with the real number support library:RealLIB:R6800. This library supports the Model 64000 Pascal implementation of the IEEE real number standard for both long and short floating point numbers (Pascal data types REAL and LONGREAL). To allow mixed REAL and LONGREAL expressions, all internal real operations are performed using an unpacked real number format with a 64-bit mantissa (fraction), a separate sign bit, and a 16-bit signed exponent.

RealLIG:R68000 will load subroutines in the PROG relocatable area and use the DATA relocatable area for local data, a default stack area, and a message buffer for error detection.

Since th use of floating point nubmers iwll require additional stack space for temporary computations, this library has a module, BIGSTACK, which will supply a defulat stack size of 1024 bytes (much larger that that supplied by the default stack in DLIB68000:D6800, LIB6800:L6800, and SLIB6800:S6800). If you have not defined your own stack area and you want to use the default stack, you should load the real library before loading the standard library of your choice.

If you do not supply your own versions of the real error reporting routines, INVALID and REAL_OVERFLOW, the real library will supply them plus a DATA relocatable buffer area for reporting the error condition. See the section on real number libraries in Chapter 4 for more informaiton on real number error detection.

Linking with Pascal File I/O

When using the Pascal FIle I/O features with the 6800, the user must link with the Pascal File I/O support library:PIOLIB:F6800.

If the simulated I/O feature of the emulation subsystem is used, the user should also link the simulated I/O support library SIMLIB:F6800.

The Pascal/64000 Reference Manual contains a ocmplete machine independent description of the routines in these libraries.

Both libraries are compiled with the options $SEPARATE ON,RECURSIVE OFF$. They will load subroutines in the PROG relocatable area and use the DATA relocatable area for local data and a message buffer for error detection.

See the section on Pascal File I/O in Chapter 5 for more information about the I/O support libraries.

Emulation of Pascal Programs

After all modules have been compiled (or assembled) and linked, the absolute file may be executed using the emulation facilities of the Model 64000. The emulator is initialized with the memory mapped in keeping with the target system and the stack pointer initialization in the code.

A program which is designed to run in read-only memory (ROM) should have been compiled with the $SEPARATE ON$ option. The memory should be mapped to have ROM and RAM as illustrated below.

```
           |=============|
           |     ROM     |  PROG relocatable area
           |     prog    |
           |=============|

           |=============|
           |     RAM     |  DATA relocatable area
           |  data area  |
 END_DATA_ |=============|
           |    heap     |
           |             |
           |=============|
           |        ?    |
           |        %    |
           |  stack |    |  (For use with the libraries:
    STACK_ |=============|  LIB6800:L6800 or
    SLIB6800:S6800)
```

For a program that has been compiled in $SEPARATE OFF$ mode, the program and data are allocated alternate blocks of storage in the PROG relocatable memory address space of the 6800. In memory mapping, RAM should appear as follows:

```
                |================|
                |       RAM      |  PROG relocatable area
                |  prog and data |
                |                |
END_DATA_       |================|
                |                |
                |      heap      |
                |                |
                |================|
                |        ?       |
                |        %       |  (For use with the library:
                |   stack |      |  DLIB6800:D6800)
STACK_          |================|
```

The transfer address will have been set by the linker so that simply loading the absolute file, and stepping or running the program is all that is required. Note that program execution does not start at address 0000H if the program contains local procedures or functions. However, the program NAME identifier in the program heading is a global symbol and the label of the program transfer address. This program may be executed within emulations by the command:

run from NAME

Debugging with DLIB6800:D6800 Library

When initializing the emulator, it is a good idea to answer yes to the "stop processor on illegal opcode?" question since execution errors may result in a jump into the error handler file, Derrors:D6800.

If, while watching the execution of the code, the status line should indicate "illegal opcode executed at address XXXXH", note the address and enter the command:

display local_symbols_in Derrors:D6800

The list will roll off the screen; do not stop it with the reset key, since the information which rolls off is not important. When the list has stopped, scan the upper portion of the list for the address at which the illegal opcode occurred. The error type will be listed at the left of this address. (Descriptions of run time errors are given in Appendix A.) The list will also be generated when using library LIB6800:L6800 by entering the following command:

display local_symbols_in Zerrors:L6800

or in library SLIB6800:S6800 by entering:ff

display local_symbols_in Zerrors:S6800

The display will now appear as follows:

NOTE

The addresses will change depending upon the link.

```
   Label               Address       Data

Z_END_PROGRAM          0CEFH         20H    Scan this portion
Z_ERR_CASE             0CBCH         00H    for the address
Z_ERR_DIV_BY_0         0CE6H         02H    where the illegal
Z_ERR_HEAP             0CC5H         03H    opcode occurred. The
Z_ERR_OVERFLOW         0CDAH         12H    data field in this
Z_ERR_RANGE            0CCEH         13H    portion is the
Z_ERR_SET_CONV         0CD4H         14H    illegal opcode for
Z_ERR_STRING           0CECH         15H    the error condition.
Z_ERR_UNDERFLOW        0CE0H         18H
Z_REG_A                0D24H         5FH    The data field in
Z_REG_B                0D25H         4EH    this portion may
Z_REG_X_H              0D26H         25H    contain useful
Z_REG_X_L              0D27H         93H    information.  The
Z_ZCALLER_H            0D29H         03H    addresses in this
Z_ZCALLER_L            0D2AH         04H    portion are not
Z_ZCC_FLAGS            0D28H         FFH    significant.
```

After some errors are detected at run time, the data field may contain useful information such as the contents of registers and the address in the user program which generated the error condition. Appendix A contains detailed information describing which items are useful for each error condition.

NOTE

It is important to remember that during emulation of Pascal/64000 programs, a Pascal program may be debugged symbolically (using global symbols in the source program) or by source program line numbers of the form: #1. This is a feature that provides a powerful tool for emulation.

NOTE

This compiler can generate duplicate symbols in the assembler symbol file for legal Pascal programs. These symbols can be generated by nested procedures with duplicate names or by procedures that conflict with lables generated by the compiler, i.e. E, R, C, and D procedure lables. Refer to the Pascal Compiler Reference Manual for a description of these lables.

These duplicate symbols can cause ambiguities with some HP Model 64000 logic analyzer measurements since a reference to a duplicated label may produce an incorrect result.

The compiler produces a warning message whenever it generates a duplicate label to warn the user that use of that symbol in an analysis product may result in an incorrect address being traced. This potential problem can be solved by changing one of the duplicate procedure names, or by moving one of the procedures to another file.

Example Warnings:

******WARNING ?? - Symbol: Y, is duplicated in the asmb_sym file.
******WARNING ?? - Symbol: RY, is duplicated in the asmb_sym file.

Chapter 2

PASCAL/64000 PROGRAMMING 6800

PROGRAMMING CONSIDERATIONS

Introduction

This chapter covers some important requirements of the run-time environment for 6800 Pascal/64000 programs. Although some requirements may not be necessary for every program, the programmer should become familiar with the information supplied in order to use it when the structure of a 6800 program requires it. The specific areas to be discussed are the 6800 direct addressing mode, stack pointer initialization, multiple module programs, heap initialization for use with dynamic memory routines (NEW, DISPOSE, MARK, and RELEASE), interrupt processing with Pascal programs, storage allocation for multibyte sets and strings, user defined operators, and pass 2 error messages.

CAUTION

PASCAL I/O cannot be performed to variables at address 0000H.

Direct Addressing Mode

If a data address is on the first page of memory (0000H through 00FFH), the 6800 can access the data using a two-byte direct address mode instruction instead of the normal three-byte extended address mode. For example, the instruction

```
LDAA    25H,D
```

will generate a two-byte instruction while

```
LDAA    25H,E
```

will generate a three-byte instruction.

The 6800 compiler will generate direct addressing instructions for any object known by the compiler to be located within the address range 00H and 0FFH. The user may inform the compiler that variables are to be on the base page with the ORG option. For the following Pascal variable declaration:

```
VAR
 $ORG = 20H$
  FLAG: BOOLEAN;
  INFORMATION: INTEGER;
 $END_ORG$
```

the 6800 compiler will generate direct addressing instructions to access variables FLAG and INFORMATION, since their addresses are known by the compiler to be between 00H and 0FFH.

Stack Pointer Initialization

The stack pointer is a hardware register maintained by the processor. Prior to use, however, it must be initialized by the user. A program that has a main code section must generate the following stack initialization statements in the relocatable file:

```
EXT STACK_
LDS #STACK_
```

Since the EXT statement implies that the label STACK_ has been declared global ($GLOBVAR$ in another Pascal program or GLB in the assembler) by another program module, the compiler will build a relocatable file, leaving assignment of the STACK_ value for the linker.

If the label STACK_ has not been declared global by any program module, the linker will search the applicable library for a default value. Depending upon which library has been selected by the user, one of the following default values will be selected:

a. If the DLIB6800:D6800 library is linked, the stack will be assigned 128 bytes in the program (PROG) area of the linked modules.

b. If the LIB6800:L6800 library is linked, the stack will be assigned 128 bytes in the data (DATA) area of the linked modules.

c. If the SLIB6800:S6800 library is linked, the stack will be assigned 128 bytes in the data (DATA) area of the linked modules.

NOTE

Whenever the LIB6800:L6800 or SLIB6800:S6800 libraries are linked, a DATA area location must be specified.

The user should allocate a larger stack when necessary. In particular, recursive programming will generally require a much larger stack than normal to run properly.

Another approach to stack pointer initialization is to define a global variable called STACK_ as shown in the following example:

```
(file MODULE1:source)
      .
      .
      .
    VAR
      ...;
      ...;
      $GLOBVAR ON$
      $ORG 3F80H
      STACK_AREA : ARRAY[1..128] of BYTE;
      STACK_ : BYTE;
      $END_ORG$
      $GLOBVAR OFF$

    BEGIN
      ...;
      ...;
      ...;
    END.
```

The compiler will generate relocatable code which sets the stack pointer to the address of STACK-(4000H in this example), and use an area of 129 bytes (3F80H..4000H) for the stack.

This technique will produce both a GLOBAL and an EXTERNAL reference for the symbol STACK_. The relocatable file will produce the proper results when linked. However, if the ASM_FILE option is in effect, the ASM6800:source file will produce an EG (external/global) error. The user should edit the ASM6800 file and delete the EXT STACK_ line before assembling the file.

The use of an absolute address for the stack as in the above example has the user convenience of assigning a fixed block of memory for the stack. It may be better, however, to allow the compiler to actually preserve a relocatable data area for the stack by leaving out the ORG and END_ORG options. This will help prevent accidental reuse of the assigned stack area by another module.

An approach when linking assembly language files is to include the initial stack pointer value or a stack area in an assembly file such as:

```
          "6800"
                    GLB STACK-
          STACK_    EQU 2000H    ;puts initial stack
                    .            ; pointer at 2000H
                    .
                    .
or:
          "6800"
                    GLB STACK-
                    DATA
          STACKBOT  RMB <stacksize>  ;puts stack
                                     ;storage in the
          STACK_:   RMB 1            ;DATA area of
                    .                ;the program
                    .
                    .
```

Note that the address of STACK_ will receive the first data byte being pushed. This file may then be linked with the other program modules generated by the compiler as follows:

Object files: ASMFILE1,MODULE1,MODULE2....

Multiple Module Programs

Only one module in an absolute program file should contain a Pascal program with a main code section. All other modules should contain procedures and functions only, with a period at the end of the procedure declarations to indicate an empty program block.
Example:

```
(file MODULE1:source)

     PROGRAM MODULE1; {this is the main module}

     CONST
         ...;
     TYPE
         ...;
     VAR
         ...;

     PROCEDURE X(Parameter : Type);EXTERNAL;
     PROCEDURE Y;EXTERNAL;

     BEGIN
         ...;
         ...;            {main code}
         ...;
     END.                {period signals end of program, main code
                          exists so stack initialization code is
                          generated}
```

NOTE

The transfer address is set to cause execution to begin in the main code section of the program module.

(file MODULE2:source)

```
PROGRAM MODULE2; {this module contains the procedures and
            functions used in MODULE1}

$GLOBPROC ON$
PROCEDURE X(Parameter : Type);
   BEGIN
     ...;
     ...;
   END;
PROCEDURE Y;
   BEGIN
     ...;
     ...;
   END;
          {The period signals the compiler that the
           program has ended.  Since no main code
           exists, the compiler does not generate any
           stack initialization code or linker
           transfer address}
```

Dynamic Allocation Heap Initialization

Before using standard procedures NEW and DISPOSE or MARK and RELEASE, the block of memory that you wish to have managed as a dynamic memory allocation pool must be initialized by calling the external library procedure:

INITHEAP(Start_address,Length_in_bytes : INTEGER)

The procedure must be declared EXTERNAL in the declaration section. The start address should be the smallest address of the memory block to be used. For example, if the block to be used is located from 4000H to 5FFFH, the initialization should appear as follows:

```
PROGRAM Test;

CONST
  .
TYPE
  .
VAR
  .
PROCEDURE
INITHEAP(Start_address,Length_in_bytes:INTEGER);EXTERNAL;
  .
  .
BEGIN  {main program block}
  INITHEAP(4000H,2000H);
  .
  .
  .
END.
```

If the desired location of the heap is at the end of the DATA area, the address of the external library variable END_DATA_ may be used as the start address and as part of an expression to give a length.

Example:

```
BEGIN
   INITHEAP(INTEGER(ADDR(END_DATA_)),
           INTEGER(ADDR(STACK_))-INTEGER(ADDR(END_DATA_)-40H));
        .
        .
END.
```

This example would reserve 41 hex (or 65 decimal) bytes for the stack and the remainder of the memory from the end of the DATA area to the initial stack pointer -41H for the dynamic allocation routines. This implies that the stack is in a contiguous block with the DATA area. For example, if END_DATA_ is address 1000H and STACK_ is address 2000H, then ADDR(STACK_) - ADDR(END_DATA_) -40H is equal to 0FC0H. The heap will be from address 1000H through 1FBFH (0F00H bytes), and the STACK will be from address 1FC0H through 2000H (see below).

```
                    prog and data

       END_DATA-
        (1000H)                      Begin_heap
                       heap

                                     End_heap (1FBFH)
                                     End_stack
                      stack
          STACK_                     Start_stack
         (2000H)
```

Six bytes are used each time the heap is initialized or marked. When an item of four bytes or less is to be allocated, four bytes will be removed from the free list even if less is needed. Likewise, when an item of four or less bytes in size is deallocated, four bytes will be returned to the free list.

Interrupt Vector Handling

The run-time programming environment of Pascal/64000 programs on the 6800 processor has been designed to impose a minimum amount of constraints on the user. In particular, the compiler does not normally generate code using any of the 6800 instructions relating to interrupts. Compiled code will not interfere with a properly designed user defined interrupt structure. As a result the code produced by the compiler is safely interruptable as long as the interrupt driven process restores the registers (which have been automatically pushed onto the stack when the 6800 recognized the interrupt) with a return from interrupt (RTI) instruction.

The 6800 processor supports four types of interrupts: a reset (or powerup) interrupt, a nonmaskable interrupt, a maskable interrupt, and a software interrupt. The first three of these are enabled by external control signals to the processor, while the last one is enabled by software program control. When the processor detects one of these interrupts it saves the current status of the processor and jumps to the address in the interrupt vector for that type of interrupt. These vectors are in the last 8 bytes of memory.

For the rest of this discussion assume that the following assembly

```
FILE: IRQ:C6800            HEWLETT-PACKARD: 6800 Assembler

LOCATION OBJECT CODE LINE     SOURCE LINE

                        1 "6800"
                        2  NAME "Interrupt Vector Definition"
                        3
                        4  EXT IRQ_ROUTINE, SOFT_ROUTINE
                        5  EXT NMI_ROUTINE, RESET_ROUTINE
                        6  ORG OFFF8H
                        7
 FFF8     0000          8  FDB IRQ_ROUTINE    ;IRQ interrupt
                                              ; routine
 FFFA     0000          9  FDB SOFT_ROUTINE   ;Software interrupt
                                              ; routine
 FFFC     0000         10  FDB NMI_ROUTINE    ;NMI interrupt
                                              ; routine
 FFFE     0000         11  FDB RESET_ROUTINE  ;RESET interrupt
                                              ; routine
                       12
```

It is possible to program routines for each of these interrupt types from Pascal/64000. A Pascal/64000 main program is suitable for use as the reset vector routine. The $INTERRUPT$ option is suitable for creating routines for use with the other three interrupt vectors.

A Pascal/64000 main program may logically be used as the RESET_ROUTINE to be called on RESET interrupt since it initializes the run time environment for Pascal program execution upon entry and performs a jump to external label Z_END_PROGRAM upon exit. A main program should be a "do forever" looping algorithm explicitly programmed by the user. Otherwise, it will end with a jump to a tight loop at Z_END_PROGRAM (generated by the compiler) at the end, thus fitting all the requirements of the RESET_ROUTINE. A "do forever" loop will loop in the user program and never reach the compiler generated jump to Z_END_PROGRAM.

Pascal/64000 allows the user to define procedures as routines to be called in the interrupt vector by using the $INTERRUPT ON$ option. The $INTERRUPT$ option is only recognized for procedures defined at the outer block of a program. An interrupt procedure needs to be declared global so its address can be available at link time to load into the proper interrupt vector. Nothing special is done upon entry to the $INTERRUPT$ procedure. At the end of the procedure the compiler generates a return from interrupt (RTI) instruction instead of a return from subroutine instruction (RTS). An $INTERRUPT$ procedure may not be called like a normal Pascal/64000 procedure because of the RTI return instruction.

The interrupt procedure can have no parameters but it may be compiled in either the $RECURSIVE ON$ or $RECURSIVE OFF$ modes. The $RECURSIVE ON$ mode is required if it is possible to be processing multiple interrupts at the same time. An interrupt handler for the IRQ interrupt which wants to allow an IRQ interrupt routine to be interrupted would require some assembly language modules, since the CLI instruction needed to enable the interrupt is not available in Pascal. The interrupt would normally not be enabled until the end of the IRQ ROUTINE when the RTI instruction would reset the interrupt mask bit.

With the previously defined interrupt vector definition the user should compile procedures IRQ_ROUTINE, SOFT_ROUTINE and NMI_ROUTINE with the $INTERRUPT ON$ option enabled. Care must be taken to turn off this option explicitly so that normal procedures and functions will not be compiled incorrectly. The RESET_ROUTINE should be compiled as a main program, i.e. PROGRAM RESET_ROUTINE.

Multi-byte Set Space Allocation

The 6800 compiler allocates sets by bytes, one bit per element. The bits are allocated low-order-bit to high-order-bit, from the lowest addressed byte to the highest. The Pascal statements:

```
PROGRAM TEST;
 VAR S1: SET OF 0..31;
```

will allocate four bytes of data to the set S1. The bits in the set will be numbered as follows:

```
        7   6   5   4   3   2   1   0
S1                  Byte #0

        15  14  13  12  11  10   9   8
                    Byte #1

        23  22  21  20  19  18  17  16
                    Byte #2

        31  30  29  28  27  26  25  24
                    Byte #3
```

String Space Allocation

The standard type STRING = PACKED ARRAY [0...255] OF CHAR is enabled with the $EXTENSIONS ON$ option. A string with a smaller maximum size, n, may be defined as a PACKED ARRAY [0...n] OF CHAR. It occupies n+1 bytes, and the 0th element contains the number of significant characters (0..255).

One string may be assigned to another, in which case the number of characters moved is the run-time length (number in the first byte) of the source. The run-time length of the destination becomes the run-time length of the source. Strings may be compared using =, <>, <, <=, >, >=. Equality means all significant characters and the length byte must be equal. One string is greater than another if the first character that differs is greater, or if they are identical up to the end of the shorter string.

A character is always compatible with a string and is treated as a string of length one when string compatibility is required.

USER DEFINED OPERATORS

General

Pascal/64000 allows the user to define his own special operators (user defined operators). User defined operators are created by using the option: $USER_DEFINED$ during the declaration of a user type. The option will apply to the declaration of one (the next) user type.

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

Operations

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

	OPERATION	SYMBOL	RUN-TIME ROUTINE
1.	Add	+	<typename>_ADD
2.	Negate	-	<typename>_NEG
3.	Subtract	-	<typename>_SUB
4.	Multiply	*	<typename>_MUL
5.	Divide	/ or DIV	<typename>_DIV
6.	Modulus	MOD	<typename>_MOD
7.	Equal Comparison	=	<typename>_EQU
8.	Not Equal Comparison	<>	<typename>_NEQ
9.	Less Than or Equal	<=	<typename>_LEQ to Comparison
10.	Greater Than or Equal	>=	<typename>_GEQ to Comparison
11.	Less Than Comparison	<	<typename>_LES
12.	Greater Than Comparison	>	<typename>_GTR

The compiler will provide the user with a Store routine. The 6800 compiler will use the multi-byte move routine (MBmove).

Parameters

The run-time routines to perform the $USER_DEFINED$ operations can be written in Pascal. For the binary operators (ADD, SUB, MUL, DIV, and MOD) with a Pascal expression of the form:

RESULT := LEFT <op> RIGHT;

the equivalent Pascal procedure definition is in the form:

PROCEDURE <typename>_<op> (VAR LEFT,RIGHT,RESULT: <typename>).

For the unary operator NEG with a Pascal expression of the form:

RESULT := - RIGHT;

the equivalent Pascal procedure definition is in the form:

PROCEDURE <typename>_NEG (VAR RIGHT,RESULT: <typename>) .

For the comparison operators (EQU, NEG, LEQ, GEQ, LES and GTR) with a Pascal expression of the form:

boolean_result := LEFT <op> RIGHT;

The equivalent Pascal definition is a function in the form:

FUNCTION <typename>_<op> (VAR LEFT,RIGHT: <typename>):BOOLEAN

The Boolean function call will cause a Boolean result (FALSE=0 or TRUE=1) to be loaded into the B register and the Z flag set accordingly upon exit.

Example:

The following program defines and uses the user type "REAL":

```
"6800"
PROGRAM USER_TYPE;
  TYPE
  $EXTENSIONS$
  $USER_DEFINED$
   REAL = RECORD
           MANTISSA: ARRAY[0..2] OF BYTE;
           EXPONENT: BYTE;
          END;
```

```
VAR  $EXTVAR$
 R1,R2,R3: REAL;
 SEMAPHORE: BOOLEAN;

BEGIN
 R1 := R2 - R3 * R1;
  { Compiler generated code for this statement:     }
  {    JSR   REAL_MUL                                }
  {    FDB   R3                 ;address of R3       }
  {    FDB   R1                 ;address of R1       }
  {    FDB   temp_result        ;compiler allocated  }
  {                               temporary result   }
  {    JSR   REAL_SUB                                }
  {    FDB   R2                 ;address of R2       }
  {    FDB   temp_result        ;compiler allocated  }
  {                               temporary result   }
  {    FDB   R1                 ;address of R1       }
  {                                                  }
 IF  -R1<R2 THEN R1 := R2;
  {  Compiler generated code for this statement:     }
  {     JSR  REAL_NEG                                }
  {     FDB  R1                                      }
  {     FDB  temp_result                             }
  {     JSR  REAL_LES                                }
  {     FDB  temp_result                             }
  {     FDB  R2                                      }
  {     BNE  then_label         ;boolean true result }
  {     JMP  else_label         ;boolean false result}
  {  then_label                                      }
  {     JSR  MBmove                                  }
  {     FDB  4                  ;# of bytes          }
  {     FDB  R2                 ;from address of R2  }
  {     FDB  R1                 ;to address of R1    }
  {  else_label                                      }
  {                                                  }
 SEMAPHORE := (R1 <= R2);
  {  Compiler generated code for this statement:     }
  {     JSR  REAL_LEQ                                }
  {     FDB  R1                                      }
  {     FDB  R2                                      }
  {     STB  SEMAPHORE   ;B has TRUE (=1) or         }
  {                             FALSE (=0)           }
END.
```

ROUTINE INTERNAL STRUCTURE

Programs, procedures, and functions are the basic blocks of Pascal program structure. Each of these routine types has a similar structure in the 6800 code generator. A routine is generally composed of a code area(including the entry point, code and an exit point), a data area and a constant area. The 6800 compiler allocates each of these areas as relocatable blocks of data normally assigned to the PROG relocation area. If the $SEPARATE$ option is in effect, the data area is assigned to the DATA relocation area and the code and constant blocks are assigned to the PROG relocation area.

The code area contains the entry point defined by a local or global label, followed by the code required to perform the routine's function. In Pascal a routine can have only one entry point and it will always return from one exit point.

The data area is the memory block where the routine's local variables and parameters are allocated. A function also needs to allocate room in the data area for the temporary copy of the function result. Finally the data area contains space for temporary values needed by the code generator to evaluate expressions which can not be computed in registers alone.

The constant area is a memory block where constants unique to a routine are specified. This area contains the dopevectors required for routines with parameters or compiled with the $RECURSIVE ON$ option and creating calls to the run time routines: PARAM_, RPARAM_, RENTRY_ and REXIT_.

An additional constant area, labeled CONST_prog, is allocated once in a compilation if certain global constant references are made. The CONST_prog area will contain the dope vectors for any array references requiring the run time library routines ARRAYand ARRAYN_. Constants being passed as value parameters will be defined in the CONST_prog area.

Compiler Internal Label Conventions

The construction of internal labels within the compiler gernerated code is discussed in Appendix C of the Pascal/64000 Compiler Manual. For the 6800 code generator, every procedure has associated with it each of the lables described in the above reference (i.e. the entry label, return label, data area label, and an end label). In addition a 6800 prodedure can have a constant area label marking the area needed for local constants and dope vectors.

In summary for a procedure named "test" the 6800 compiler would create the lables: test, Rtest, Ctest, and Etest. For the sample program listed in figure 2-1 the compiler generated lables are summarized as follows:

Compiler Generated Label	Program Counter	Label Description
Assign	0000H	Procedure entry
RAssign	002CH	Return label
CAssign	002DH	Constant area
DAssign	003DH	Data area
EAssign	004FH	End of procedure
SAME_function	0050H	Procedure entry
RSAME_function	008CH	Return label
CSAME_function	008DH	Constant area
DSAME_function	009DH	Data area
ESAME_function	00B1H	End of procedure
PF_sample	00B2H	Program entry
PF_samp00_2	00CAH	Compiler generated label
PF_samp00_1	00D9H	Compiler generated label
RPF_sample	00D9H	Return label
DPF_sample	00DCH	Data area
EPF_sample	00E6H	End of procedure

Figure 2-1(a) shows a source listing for a simple program. Figure 2-1(b) shows the expanded source listing for this program indicating the use of internal compiler lables.

```
FILE:  PF_sample:T6800      HP 64000 - Pascal     6800 code generator

 0000  1  "6800"
 0000  1  PROGRAM PF_sample;
 0000  1  $EXTENSIONS$
 0000  1  TYPE BIG_type= RECORD A,B,C,D:INTEGER; END;
 0000  1  VAR
 0000  1    Byte    :BYTE;
 0001  1    Integer :INTEGER;
 0003  1    Big_one :BIG_type;
 000B  1
 000B  1  PROCEDURE Assign(B1:BYTE;      VAR B2:BYTE;
 0000  2                  I1:INTEGER;  VAR I2:INTEGER;
 0000  2                  X1:BIG_type; VAR X2:BIG_type);
 0000  2   VAR DUMMY_local_var:INTEGER;
 0002  2   BEGIN
 0006  2    DUMMY_local_var:=0;
 000C  2    B2:= B1;
 0014  2    I2:= I1;
 0021  2    X2:= X1;
 002C  2   END;
 0000  1
 0000  1  FUNCTION SAME_function (B1:BYTE;      VAR B2:BYTE;
 0000  2                         I1:INTEGER;  VAR I2:INTEGER;
 0000  2                         X1:BIG_type; VAR X2:BIG_type)
                                 :BOOLEAN;
 0001  2   VAR DUMMY_local_var :INTEGER;
 0003  2   BEGIN
 0056  2    DUMMY_local_var:=1;
 005C  2    SAME_function:= (B2=B1) and (I2=I1) and (X2=X1);
 008C  2   END;
 0000  1
 0000  1  BEGIN   {Main program: PF_sample}
 00B5  1   IF NOT SAME_function (Byte,Byte,Integer,Integer,
                                Big_one,Big_one)
 00B5  1     THEN  Assign(Byte,Byte,Integer,Integer,Big_one,
                         Big_one);
 00D9  1  END

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

Figure 2-1 (a). Internal Structure Source Listing

```
FILE:  PF_sample:T6800      HP 64000 - Pascal     6800 code generator

   1  0000  1  "6800"
   2  0000  1  PROGRAM PF_sample;
   3  0000  1  $EXTENSIONS$
   4  0000  1  TYPE BIG_type= RECORD A,B,C,D:INTEGER; END;
   5  0000  1  VAR
   6  0000  1    Byte    :BYTE;
   7  0001  1    Integer :INTEGER;
   8  0003  1    Big_one :BIG_type;
   9  000B  1
  10  000B  1  PROCEDURE Assign(B1:BYTE;     VAR B2:BYTE;
  11  0000  2                   I1:INTEGER;  VAR I2:INTEGER;
  12  0000  2                   X1:BIG_type; VAR X2:BIG_type);
  13  0000  2   VAR DUMMY_local_var:INTEGER;
  14  0002  2   BEGIN
        0000          Assign
        0000              LDX  #CAssign
        0003              JSR  PARAM-
  15  0006  2    DUMMY_local_var:=0;
        0006              LDX  #00000H
        0009              STX  DAssign
  16  000C  2    B2:= B1;
        000C              LDAB DAssign+00002H
        000F              LDX  DAssign+00003H
        0012              STAB ,X
  17  0014  2    I2:= I1;
        0014              LDAA DAssign+00005H
        0017              LDAB DAssign+00006H
        001A              LDX  DAssign+00007H
        001D              STAA ,X
        001F              STAB 00001H,X
  18  0021  2    X2:= X1;
        0021              JSR  MBmove
        0024              FDB  00008H
        0026              FDB  DAssign+00009H
        0028              FDB  00000H
        002A              FDB  DAssign+00011H
  19  002C  2   END;
```

Figure 2-1 (b). Internal Structure Source Listing

```
FILE PF_sample:T6800   HP 64000 - Pascal   6800 code generator

        002C          RAssign
        002C              RTS
        002D          CAssign
        002D              FDB  DAssign+00002H
        002F              FDB  00006H
        0031              FDB  00001H
        0033              FDB  0FFFEH
        0035              FDB  00002H
        0037              FDB  0FFFEH
        0039              FDB  00008H
        003B              FDB  0FFFEH
        003D          DAssign
        003D              RMB  00013H
 20   0000   1

 21   0000   1  FUNCTION SAME_function (B1:BYTE;     VAR B2:BYTE;
 22   0000   2                          I1:INTEGER;  VAR I2:INTEGER;
 23   0000   2                          X1:BIG_type; VAR X2:BIG_type)
                                        :BOOLEAN;
 24   0001   2   VAR DUMMY_local_var :INTEGER;
 25   0003   2   BEGIN
        0050          EAssign          EQU $-1
        0050          SAME_function
        0050              LDX  #CSAME_function
        0053              JSR  PARAM-
 26   0056   2    DUMMY_local_var:=1;
        0056              LDX  #00001H
        0059              STX  DSAME_function+00001H
 27   005C   2    SAME_function:= (B2=B1) AND (I2=I1)  AND (X2=X1);
        005C              LDX  DSAME_function+00004H
        005F              LDAB ,X
        0061              CMPB DSAME_function+00003H
        0064              JSR  Zequ
        0067              STAB DSAME_function+00014H
        006A              LDX  DSAME_function+00008H
        006D              LDX  ,X
        006F              CPX  DSAME_function+00006H
        0072              JSR  Zequ
        0075              ANDB DSAME_function+00014H
        0078              STAB DSAME_function+00014H
        007B              JSR  MBequ
        007E              FDB  00008H
        0080              FDB  DSAME_function+0000AH
        0082              FDB  00000H
```

Figure 2-1 (b). Internal Structure Expanded Listing (Cont'd)

```
FILE: PF_sample:T6800   HP 64000 - Pascal   6800 code generator

        0084             FDB  DSAME_function+00012H
        0086             ANDB DSAME_function+00014H
        0089             STAB DSAME_function
 28   008C  2   END;
        008C         RSAME_function
        008C             RTS
        008D         CSAME_function
        008D             FDB  DSAME_function+00003H
        008F             FDB  00006H
        0091             FDB  00001H
        0093             FDB  0FFFEH
        0095             FDB  00002H
        0097             FDB  0FFFEH
        0099             FDB  00008H
        009B             FDB  0FFFEH
        009D         DSAME_function
        009D             RMB  00015H
 29   0000  1
 30   0000  1  BEGIN  {Main program:  PF_sample}
        00B2        ESAME_function EQU $-1
        00B2        PF_sample
        00B2             LDS  #STACK-
 31   00B5  1   IF NOT SAME_function (Byte,Byte,Integer,Integer,
                                    Big_one,Big_one)
 32   00B5  1     THEN  Assign(Byte,Byte,Integer,Integer,Big_one,
                           Big_one);
        00B5             BSR  SAME_function
        00B7             FDB  DPF_sample
        00B9             FDB  DPF_sample
        00BB             FDB  DPF_sample+00001H
        00BD             FDB  DPF_sample+00001H
        00BF             FDB  DPF_sample+00003H
        00C1             FDB  FDB_sample+00003H
        00C3             EORB #001H
        00C5             BNE  PF_samp00_2
        00C7             JMP  PF_samp00_1
```

Figure 2-1 (b). Internal Structure Expanded Listing (Cont'd)

```
FILE: PF_sample:T6800   HP 64000 - Pascal   6800 code generator

        00CA          PF_samp00_2
        00CA              JSR  Assign
        00CD              FDB  DPF_sample
        00CF              FDB  DPF_sample
        00D1              FDB  DPF_sample+00001H
        00D3              FDB  DPF_sample+00001H
        00D5              FDB  DPF_sample+00003H
        00D7              FDB  DPF_sample+00003H
        00D9          PF_samp00_1
  33  00D9  1  END.
        00D9          RPF_sample
        00D9              GLOBAL  RPF_sample
        00D9
        00D9              JSR  Z_END_PROGRAM
        00DC          DPF_sample
        00DC              RMB  0000BH
        00DC          EPF_sample    EQU $-1
        00DC
        00DC              GLOBAL  EPF_sample
        00DC
                          GLOBAL   PF_sample
                          EXTERNAL PARAM-
                          EXTERNAL Z_END_PROGRAM
                          EXTERNAL MBmove
                          EXTERNAL STACK-
                          EXTERNAL Zequ
                          EXTERNAL MBequ
                          END      PF_sample

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

Figure 2-(b). Internal Structure Expanded Listing (Cont'd)

Data Variable Allocation

The allocation of variables to the data area of a routine is always in the order: function result (if required) followed by local variables followed by parameters followed by temporary storage.

```
Data_area_label         {Low memory address}
[Dprocedure]           /-------------------\
                       |  Function result   | {Only for functions}
                       |-------------------|
                       |  Local variables   | {Declared variables}
                       |-------------------|
                       |  Parameters        | {Declared
                          parameters}
                       |-------------------|
                       |  Temporary values  | {Compiler
                          temporaries}
                       \-------------------/
                        {High memory address}
```

Procedures and functions pass parameters in the same way. For procedures and functions declared with the $RECURSIVE OFF$ option, the code generator will pass one parameter in a register if its size is 1 or 2 bytes. For one parameter with a size larger than 2 bytes or for more than one parameter and for all routines declared with the $RECURSIVE ON$ option, the code generator will pass parameters by the generalized parameter passing method using dope vectors described in detail in Chapter 3.

The expanded compiler listing in figure 2-1 is intended to show the memory allocation of data areas and the parameter passing method for procedures and functions. A descriptive summary of the data area for PROCEDURE Assign, FUNCTION SAME_function and main PROGRAM PF_sample is provided to help interpret the listing.

PROCEDURE Assign Data_area Description Summary

```
DAssign                RMB 00013H      ; 19 bytes
```

Program Counter {HEX}	Data_area Offset {HEX}	Size {Bytes}	Name Identifier	Description
003D	0000	2	DUMMY_local_var	Integer variable
003F	0002	1	B1	Byte parameter
0040	0003	2	B2	VAR byte parameter
0042	0005	2	I1	Integer parameter
0044	0007	2	I2	VAR integer parameter
0046	0009	8	X1	BIG_type parameter
004E	0011	2	X2	VAR BIG_type parameter

FUNCTION SAME_function Data_area Description Summary

```
DSAME_function            RMB 00015H      ; 21 bytes
```

Program Counter {HEX}	Data_area Offset {HEX}	Size {Bytes}	Name Identifier	Description
009D	0000	1	SAME_function	Boolean function return value
009E	0001	2	DUMMY_local_var	Integer variable
00A0	0003	1	B1	Byte parameter
00A1	0004	2	B2	VAR byte parameter
00A3	0006	2	I1	Integer parameter
00A5	0008	2	I2	VAR integer parameter
00A7	000A	8	X1	BIG_type parameter
00AF	0012	2	X2	VAR BIG_type parameter
00B1	0014	1	temporary	Boolean compiler temporary

PROGRAM PF_sample Data_area Description Summary

```
DPF_sample          RMB 0000BH      ; 11 bytes

Program   Data_area             Name
Counter   Offset     Size       Identifier       Description
 {HEX}     {HEX}    {Bytes}

 00DC      0000       1         Byte              Byte variable
 00DD      0001       2         Integer           Integer variable
 00DF      0003       8         Big_one           BIG_type variable
```

Large Function Results

The 6800 Pascal compiler allows user defined functions to return results of any size. Function results of data types which can be represented in one or two bytes are returned in registers. Function results of size three or more bytes are returned by adding an extra VAR parameter to the function's parameter list which tells the function where to store the result.

Function results which fit into one byte (eg. BOOLEAN, CHAR, BYTE, UNSIGNED_8, or scalar types with less than 256 values) return their result in the B register of the 6800. Function results which require two byte representations(eg. INTEGER or UNSIGNED_16)return their result in the X register of the 6800. The last statement of a one or two byte function will load the function result into the proper register.

Functions requiring the return of large function results (those which require 3 or more bytes) will have an extra VAR parameter added to the user defined parameter list to indicate the memory location where the calling routine wants to store the function result. During function execution the assigments to the function result are written into local storage within the function data area. At the end of the function prior to the return statement the function result is copied from the local data area into the result VAR parameter.

Figure 2-2 is an expanded listing of PROGRAM BIG_FUNC which shows the code generated for a procedure and a function which perform similar vector operations on an eight-byte array. The PROCEDURE BIG_type_ADD performs vector addition of two arrays storing the result into a third VAR parameter. The FUNCTION BIG_type_SUB performs vector subtraction of two arrays. Since the implementation of large function results requires the addition of the extra VAR parameter, note that the dope vectors of the two routines are similar. Each routine will pass three VAR parameters, and the result will be assigned to the third parameter.

Notice the significant difference between the internal operation of the procedure and the function. In the procedure, the result of the addition of each element is stored immediately into the VAR parameter RESULT. In the function, the result of each subtraction is stored first into the local function result area. The result of the function is only stored into the actual VAR parameter prior to the return from the function.

In a more complex algorithm where the parameters may be used more than once in an arbitrary order and where the result parameter may be the same as one of the inputs, the procedure implementation would allow one of the input parameters to be modified (as the result) before the computation was complete. A function implementation (by assigning the function result at the end of the function) would not affect any of the input parameters until the computation was complete.

Since the large function result for two inputs produces the same calling code required for a procedure with three large VAR parameters, a large function may also be used to satisfy the requirements of the user-defined operations of addition, subtraction, multiplication, division and modulus. User types of size one or two bytes may not be programmed as functions, since their results are returned immediately in registers.

```
FILE:  BIG_FUNC:T6800     HP 64000 - Pascal     6800 code generator

  1  0000  1  "6800"
  2  0000  1  PROGRAM BIG_FUNC;
  3  0000  1  $EXTENSIONS$
  4  0000  1  CONST       BIG_size = 7;
  5  0000  1  TYPE        BIG_type = ARRAY[0..BIG_size]OF BYTE;
  6  0000  1  VAR         U1,U2,U3,U4,U5:BIG_type;
  7  0028  1
  8  0028  1  PROCEDURE   BIG_type_ADD(VAR P1,P2,RESULT:BIG_type);
  9  0000  2  VAR         COUNT:BYTE;
 10  0001  2  BEGIN
        0000          BIG_type_ADD
        0000              LDX   #CBIG_type_ADD
        0003              JSR   PARAM-
 11  0006  2     FOR COUNT := 0 TO BIG_size DO
        0006              CLR   DBIG_type_ADD
        0009          BIG_typ01_2
 12  0009  2        RESULT[COUNT] := P1[COUNT] + P2[COUNT];
        0009              LDX   BIG_type_ADD+00005H
        000C              LDAB  DBIG_type_ADD
        000F              JSR   LEAX_B_X
        0012              STX   DBIG_type_ADD+00007H
        0015              LDX   DBIG_type_ADD+00001H
        0018              JSR   LEAX_B_X
        001B              STX   DBIG_type_ADD+00009H
        001E              LDX   DBIG_type_ADD+00003H
        0021              JSR   LEAX_B_X
        0024              STX   DBIG_type_ADD+0000BH
        0027              LDX   DBIG_type_ADD+00009H
        002A              LDAB  ,X
        002C              LDX   DBIG_type_ADD+0000BH
        002F              ADDB  ,X
        0031              LDX   DBIG_type_ADD+00007H
        0034              STAB  ,X
        0036              LDAB  DBIG_type_ADD
        0039              CMPB  #007H
        003B              BEQ   BIG_typ01_1
        003D              INC   DBIG_type_ADD
        0040              BRA   BIG_typ01_2
        0042          BIG_typ01_1
        0042          RBIG_type_ADD
        0042              RTS
        0043          CBIG_type_ADD
        0043              FDB  DBIG_type_ADD+00001H
        0045              FDB  00003H
        0047              FDB  0FFFEH
 13  0042  2  END;
```

Figure 2-2. Large Function Results

```
FILE:  BIG_FUNC:T6800    HP 64000 - Pascal    6800 code generator

        0049                FDB   0FFFEH
        004B                FDB   0FFFEH
        004D            BIG_type_ADD
        004D                RMB   0000DH
 14     0000   1
 15     0000   1  FUNCTION BIG_type_SUB(VAR P1,P2:BIG_type):BIG_type;
 16     0008   2  VAR
 17     0008   2      COUNT:BYTE;
 18     0009   2  BEGIN
        005A            EBIG_type_ADD  EQU $-1
        005A            BIG_type_SUB
        005A                LDX   #CBIG_type_SUB
        005D                JSR   PARAM-
 19     0060   2     FOR COUNT := 0 TO BIG_size DO
        0060                CLR   DBIG_type_SUB+00008H
        0063            BIG_typ02_4
 20     0063   2        BIG_type_SUB[COUNT] := P1[COUNT] - P2[COUNT];
        0063                LDX   #DBIG_type_SUB
        0066                LDAB  DBIG_type_SUB+00008H
        0069                JSR   LEAX_B_X
        006C                STX   DBIG_type_SUB+0000FH
        006F                LDX   DBIG_type_SUB+00009H
        0072                JSR   LEAX_B_X
        0075                STX   DBIG_type_SUB+00011H
        0078                LDX   DBIG_type_SUB+0000BH
        007B                JSR   LEAX_B_X
        007E                STX   DBIG_type_SUB+00013H
        0081                LDX   DBIG_type_SUB+00011H
        0084                LDAB  ,X
        0086                LDX   DBIG_type_SUB+00013H
        0089                SUBB  ,X
        008B                LDX   DBIG_type_SUB+0000FH
        008E                STAB  ,X
        0090                LDAB  DBIG_type_SUB+00008H
        0093                CMPB  #007H
        0095                BEQ   BIG_typ02_3
        0097                INC   DBIG_type_SUB+00008H
        009A                BRA   BIG_typ02_4
        009C            BIG_typ02_3
 21     009C   2  END;
        009C                JSR   MBmove
        009F                FDB   00008H
        00A1                FDB   DBIG_type_SUB
        00A3                FDB   00000H
        00A5                FDB   DBIG_type_SUB+0000DH
```

Figure 2-2. Large Function Results (Cont'd)

```
FILE:  BIG_FUNC:T6800     HP 64000 - Pascal     6800 code generator

         00A7          RBIG_type_SUB
         00A7              RTS
         00A8          CBIG_type_SUB
         00A8              FDB  DBIG_type_SUB+00009H
         00AA              FDB  00003H
         00AC              FDB  0FFFEH
         00AE              FDB  0FFFEH
         00B0              FDB  0FFFEH
         00B2          DBIG_type_SUB
         00B2              RMB 00015H
  22  0000  1
  23  0000  1
  24  0000  1  BEGIN
         00C7          EBIG_type_SUB  EQU $-1
         00C7          BIG_FUNC
         00C7              LDS  #STACK-
  25  00CA  1     BIG_type_ADD(U1,U2,U3);
         00CA              JSR BIG_type_ADD
         00CD              FDB DBIG_FUNC
         00CF              FDB DBIG_FUNC+00008H
         00D1              FDB DBIG_FUNC+00010H
  26  00D3  1     U4:=BIG_type_SUB(U3,U2);
         00D3              BSR BIG_type_SUB
         00D5              FDB DBIG_FUNC+00010H
         00D7              FDB DBIG_FUNC+00008H
         00D9              FDB DBIG_FUNC+00018H
  27  00DB  1  END.
         00DB          RBIG_FUNC
         00DB              GLOBAL  RBIG_FUNC
         00DB
         00DB              JSR  Z_END_PROGRAM
         00DE          DBIG_FUNC
         00DE              RMB  0028H
         00DE              EBIG_FUNC    EQU $-1
         00DE
         00DE              GLOBAL  EBIG_FUNC
         00DE
                           GLOBAL  BIG_FUNC
                           EXTERNAL PARM-
                           EXTERNAL Z_END_PROGRAM
                           EXTERNAL MBmove
                           EXTERNAL STACK-
                           EXTERNAL LEAX_B_X
                           END      BIG_FUNC

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

Figure 2-2. Large Function Results (Cont'd)

6800 COMPILER OPTIONS

ASM_FILE

Default OFF.
The compiler option ASM_FILE will produce a source file of the 6800 assembler code equivalent to the original program. This assembler source file will be created with the filename: ASM6800[:current_userid]. This file will generally be correct as an input source file for the 6800 assembler. External or global variables with the names A, B, X, D, or E will cause assembly errors because these are predefined symbols for the 6800 registers or for use in creating direct or extended memory accesses.

DEBUG

Default OFF.
The DEBUG option is used to check for overflow and underflow on arithmetic operations for the standard types: BYTE (or SIGNED_8), UNSIGNED_8, INTEGER (or SIGNED_16), and UNSIGNED_16. Operations which may normally be performed with in-line code (such as a BYTE add), will be performed using a subroutine call if the DEBUG option is on. The library routines in the debug library (DLIB6800:D6800) have checks to detect underflow or overflow of the arithmetic operation. The routines of the same name in the nondebug libraries (LIB6800:L6800 and SLIB6800:S6800) perform the same arithmetic operation but do not detect or report any overflow, underflow, or divide by zero error conditions.

The sample listing in figure 2-3 shows the different code generation sequences for $DEBUG ON$ and $DEBUG OFF$ for a simple BYTE addition.

```
FILE:  DEBUG:T6800    HP Pascal - Pascal      Option DEBUG example


   4  0000   1  PROGRAM DEBUG;
   5  0000   1                            $EXTENSIONS$
   6  0000   1   VAR
   7  0000   1                            $EXTVAR$
   8  0000   1    FIRST,SECOND,THIRD:BYTE;
   9  0000   1    BEGIN
         0000          DEBUG
         0000              LDS  #STACK-

  10  0003   1                            $DEBUG OFF$
  11  0003   1     THIRD:= FIRST+SECOND;
         0003              LDAB FIRST
         0006              ADDB SECOND
         0009              STAB THIRD

  12  000C   1                            $DEBUG ON$
  13  000C   1     THIRD:= FIRST+SECOND;
         000C              LDAB FIRST
         000F              LDAA SECOND
         0012              JSR  Zbyteadd
         0015              STAB THIRD

  14  0018   1    END
         0018          RDEBUG
         0018              GLOBAL  RDBUG
         0018
         0018              JSR  Z_END_PROGRAM

  15  0000   1
         0000          EDEBUG      EQU $-1
         0000
         0000              GLOBAL   EDBUG
         0000
                           GLOBAL   DEBUG
                           EXTERNAL Z_END_PROGRAM
                           EXTERNAL STACK-
                           EXTERNAL Zbyteadd
                           END      DEBUG

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

Figure 2-3. Option $DEBUG$

OPTIMIZE

The 6800 instruction sequences for a particular Pascal construct have been written to minimize the size of the generated code while preserving the logical correctness of the function being performed. If OPTIMIZE is on, the compiler will generate more instructions for a given construct in an attempt to perform the function faster.

The OPTIMIZE option will generate both smaller and faster code for the particular situation of forward jumps. Since the code generator is logically a one-pass process, a branch or jump to a forward label must be able to branch an arbitrarily long distance. As a result, the 6800 is required to use a 2-byte branch followed by a 3-jump instruction to execute a conditional forward branch properly. In many cases, this is an unnecessary protection. With the OPTIMIZE option on, the compiler will create short branches to such undefined user or compiler generated forward labels. If the label turns out to be too far away, the compiler will report this as a Pass 2 Error #1200. If this error is not produced, the relative branch instructions have been successful. If this error is produced, the user should turn off the OPTIMIZE option for the offending line of code.

The sample listing in figure 2-4 shows the different code generation sequences for $OPTIMIZE ON$ and $OPTIMIZE OFF$ for a simple IF..THEN..ELSE statement.

```
FILE:  OPTIMIZE:T6800     HP 64000 - Pascal       Option OPTIMIZE example

   4  0000   1  PROGRAM OPTIMIZE;
   5  0000   1                               $EXTENSIONS$
   6  0000   1   VAR
   7  0000   1                               $EXTVAR$
   8  0000   1    FIRST,SECOND,LARGER:BYTE;
   9  0000   1    BEGIN
         0000          OPTIMIZE
         0000              LDS   #STACK-
  10  0003   1                               $OPTIMIZE OFF$
  11  0003   1     IF FIRST>SECOND THEN
         0003              LDAB FIRST
         0006              CMPB SECOND
         0009              BGT   OPTIMIZ00_5
         000B              JMP   OPTIMIZ00_1
         000E          OPTIMIZ00_5
  12  000E   1       LARGER:= FIRST
  13  000E   1      ELSE
         000E              STAB LARGER
         0011              JMP   OPTIMIZ00_2
         0014          OPTIMIZ00_1
  14  0014   1       LARGER:= SECOND;
         0014              LDAB SECOND
         0017              STAB LARGER
         001A          OPTIMIZ00_2
```

Figure 2-4 Option $OPTIMIZE$

```
FILE:  OPTIMIZE:T6800     HP 64000 - Pascal       Option OPTIMIZE example

  15  001A  1                                 $OPTIMIZE ON$
  16  001A  1     IF FIRST>SECOND THEN
        001A               LDAB FIRST
        001D               CMPB SECOND
        0020               BLE  OPTIMIZO0_3
  17  0022  1       LARGER:= FIRST
  18  0022  1      ELSE
        0022               STAB LARGER
        0025               BRA  OPTIMIZO0_4
        0027           OPTIMIZO0_3
  19  0027  1       LARGER:= SECOND;
        0027               LDAB SECOND
        002A               STAB LARGER
        002D           OPTIMIZO0_4
  20  002D  1    END
        002D           ROPTIMIZE
        002D                GLOBAL    ROPTIMIZE
        002D
        002D               JSR  Z_END_PROGRAM
  21  0000  1
        0000               EOPTIMIZE     EQU $-1
        0000
        0000               GLOBAL  EOPTIMIZE
        0000

                           GLOBAL OPTIMIZE
                           EXTERNAL Z_END_PROGRAM
                           EXTERNAL STACK-
                           END      OPTIMIZE

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

Figure 2-4. Option $OPTIMIZE$ (Cont'd)

RANGE

The RANGE option is used to check array index expressions, value parameters, and variable assignments for correct subrange values before performing the operation. If a variable has been defined as one of the standard predefined data types (BYTE or SIGNED_8, UNSIGNED_8, INTEGER or SIGNED_16 and UNSIGNED_16), there are no out-of-range values if the size of the expression (1 or 2 bytes) is appropriate. The assignment of these data types will create no range checking code. If the user desires to check for out-of-range values while performing arithmetic operations on standard predefined data types, the DEBUG option should be used.

Only if the user has defined a variable as a scalar data type or as a subrange data type will range-checking code be produced. The sample listing in figure 2-5 shows the different code generation sequences for $RANGE ON$ and $RANGE OFF$ for a simple BYTE subrange assignment.

```
FILE:  RANGE:T6800    HP 64000 - Pascal       Option RANGE example

   4  0000  1  PROGRAM RANGE;
   5  0000  1                             $EXTENSIONS$
   6  0000  1   VAR
   7  0000  1                             $EXTVAR$
   8  0000  1    FIRST,SECOND,THIRD:0..63;
   9  0000  1    BEGIN
         0000          RANGE
         0000              LDS  #STACK-
  10  0003  1                             $RANGE OFF$
  11  0003  1     THIRD:= FIRST+SECOND;
         0003              LDAB FIRST
         0006              ADDB SECOND
         0009              STAB THIRD
  12  000C  1                             $RANGE ON$
  13  000C  1     THIRD:= FIRST+SECOND;
         000C              LDAB FIRST
         000F              ADDB SECOND
         0012              TBA
         0013              LDX  #03F00H
         0016              JSR  Zbbounds
         0019              BNE  RANGE00_1
         001B              JSR  RANGE_ERROR
         001E          RANGE00_1
         001E              STAA THIRD
  14  0021  1    END
         0021          RRANGE
         0021              GLOBAL    RRANGE
         0021
         0021              JSR  Z_END_PROGRAM
  15  0000  1
        0000           ERANGE             EQU $-1
         0000
         0000              GLOBAL   ERANGE
         0000

                           GLOBAL   RANGE
                           EXTERNAL Z_END_PROGRAM
                           EXTERNAL STACK-
                           EXTERNAL Zbbounds
                           EXTERNAL RANGE_ERROR
                           END      RANGE

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

Figure 2-5. Option $RANGE$

PASS 2 ERRORS

Pass 2 errors will be diplayed on the screen with the message:

LINE # <line number>--PASS2 ERROR # <Pass 2 error number>

In addition, if a listing file has been indicated for the compilation it will indicate pass 2 errors where they occurred. It will also give you a listing of the meaning of each error.

Pass 2 errors numbers will always be >=1000. Errors with numbers between 1000 and 1099 are fatal errors. Errors with numbers >=1100 are nonfatal errors.

Pass 2 will stop generating code after a fatal pass 2 error. If a listing file has been indicated for the compilation, pass 3 will give you a listing with errors. Nonfatal errors are output to the display and to the listing file (if one exists), but compilation continues after appropriate action has been taken to correct the error. A list of pass 2 errors are given in Table 2-1.

Table 2-1. 6800 Pass 2 Errors

1000	"Out of memory" The 6800 code generator has run out of memory, break up your program and recompile.
1001	"Size not implemented" An integer larger than 16 bits has been detected.
1002	"Size error" A size larger than the maximum size allowed for a type has been detected.
1003	"Type not implemented" A real or other unimplemented type has been detected.
1004	"Type error" An operation with an incorrect type of operands has been detected; for example, a negation of an unsigned value.
1005	"Unimplemented feature" An attempt has been made at using a feature not implemented on the 6800 code generator.
1006	"Compiler error. Contact Hewlett-Packard." This is a compiler level error. Please report this error to Hewlett-Packard as soon as possible.
1007	"Expression too complicated" The compiler can not handle the level of complexity of this expression, simplify your expression.
1008	"Register needed but not available" The compiler can not generate more code without additional registers; add temporary results for your operations.
1100	"Bounds error" Compile time bounds check error when using the $RANGE ON$ option.
1103	"Interrupt procedure must not have parameters" An interrupt procedure can not have parameters. The compiler will ignore the parameters and continue to generate code.

Table 2-1. 6800 Pass 2 Errors (Cont'd)

1104	"Interrupt procedure call not allowed" An interrupt routine can only be accessed through an interrupt vector, since it will return with an RTI instead of an RTS. The compiler will ignore calls to interrupt routines.
1105	"Data size too large" More than 64K bytes of data have been allocated for this procedure.
1106	"Program counter overflow" The PROG section has become larger than 64K bytes. This error is detected in Pass 3.
1107	"Data counter overflow" The DATA section has become larger than 64K bytes. This error is detected in Pass 3.
1108	"Unimplemented feature" An attempt has been made to use a feature not implemented for the 6800 processor. The feature will be ignored.
1110	"Defined a static routine within a recursive one" Static routines can not be defined within recursive routines because of the difference in addressing. The compiler makes the routine recursive and continues to generate code.
1111	"Interrupt routines must be at level one" All interrupt routines must be at level one. For routines defined at levels greater than 1 with $INTERRUPT ON$, the compiler will ignore the option, i.e. it will generate a noninterrupt routine.
1113	"Program counters do not agree" The program counter for a label generated by Pass 2 does not agree with the program counter for that label in Pass 3. Please report the error to Hewlett-Packard as soon as possible. This error is detected in Pass 3.
1200	"Long range error; turn off OPTIMIZE for this line" The option $OPTIMIZE$ causes the code generator to use 2-byte branch instructions for forward branches. This error occurs when the label is too far away. Turning $OPTIMIZE OFF$ for this line of code will produce a long jump which will always work.

NOTES

Chapter 3

RUN-TIME LIBRARY SPECIFICATIONS

GENERAL

This chapter describes the run-time library routines needed to execute Pascal programs compiled by the Pascal/64000 compiler for the 6800 microprocessor. Each routine description includes the purpose, input requirements, and output results.

The library is logically divided into two groups of routines. One group contains the standard library procedures and functions. The second group supplies the elementary routines that supplement the standard 6800 instruction set. Tables 3-1 and 3-2 list the standard and supplemental routines for the 6800 microprocessor.

Table 3-1. Pascal Library Routines (Standard)

Name	Purpose
ARRAY_	Compute address of array element
ARRAYN_	Compute address of array vector
PARAM_	Pass parameters to procedures
RPARAM_	Pass parameters to recursive routines
RENTRY_	Recursive procedure entry
REXIT_	Recursive procedure exit
INITHEAP	Declares block of memory as memory pool
NEW	Dynamic memory allocation
DISPOSE	Dynamic memory deallocation
MARK	Save current status of dynamic memory heap
RELEASE	Restore prior status of dynamic memory heap

Table 3-2. Pascal Library Routines (for 6800)

8-BIT ARITHMETIC GROUP

NAME	PURPOSE
Zbyteabs	Byte absolute value
Zbyteneg	Byte negation
Zbyteadd	Byte addition
Zubyteadd	Unsigned byte addition
Zbytesub	Byte subtraction
Zubytesub	Unsigned byte subtraction
Zbytemul	Byte multiplication
Zubytemul	Unsigned byte multiplication
Zbytediv	Byte division
Zubytediv	Unsigned byte division
Dbytemul	Debug byte multiply
Dubytemul	Debug unsigned byte multiply

16-BIT ARITHMETIC GROUP

NAME	PURPOSE
Zintabs	Integer absolute value
Zintneg	Integer negation
Zintadd	Integer addition
Zuintadd	Unsigned integer addition
Zintsub	Integer subtraction
Zuintsub	Unsigned integer subtraction
Zintmul	Integer multiplication
Zuintmul	Unsigned integer multiplication
Zintdiv	Integer division
Zuintdiv	Unsigned integer division

Table 3-2. Pascal Library Routines (for 6800) (Cont'd)

BYTE AND WORD SHIFTS

NAME	PURPOSE
Zbshift	Byte shift logical with zero fill
Zbshiftc	Byte shift circular
Zwshift	Word shift logical with zero fill
Zwshiftc	Word shift circular

RYTE AND WORD SET OPERATIONS

NAME	PURPOSE
Zbinset8	Byte in 8-bit set
Zbinset16	Byte in 16-bit set
Zbtoset8	Byte to 8-bit set
Zbtoset16	Byte to 16-bit set
Zwinset8	Word in 8-bit set
Zwinset16	Word in 16-bit set
Zwtoset8	Word to 8-bit set
Zwtoset16	Word to 16-bit set
Zset16int	Intersection of 16-bit sets
Zset16uni	Union of 16-bit sets
Zset16geq	16-bit set greater than or equal
Zset16leq	16-bit set less than or equal
Zset16dif	Set difference of 16-bit sets

MULTIBYTE OPERATIONS

NAME	PURPOSE
MBmove	Multibyte assignment
MBequ	Multibyte equality test
MBneq	Multibyte inequality test
MBgeq	Multibyte greater than or equal test
MBgtr	Multibyte greater than test
MBleq	Multibyte less than or equal test
MBles	Multibyte less than test

Table 3-2. Pascal Library Routines (6800) (Cont'd)

MULTIBYTE SET OPERATIONS

NAME	PURPOSE
INSETmb	Multibyte set inclusion
TOSETmb	Multibyte set formation
SETmbINT	Multibyte set intersection
SETmbUNI	Multibyte set union
SETmbDIF	Multibyte set difference
SETmbGEQ	Multibyte set greater than or equal
SETmbLEQ	Multibyte set less than or equal

BYTE AND INTERGER COMPARISON AND BOUNDS CHECKING ROUTINES

NAME	PURPOSE
Zcc	Carry cleared test
Zequ	Byte and integer equality test
Zneq	Byte and integer inequality test
Zgeq	Byte greater than or equal test
Zgtr	Byte greater than test
Zleq	Byte less than or equal test
Zles	Byte less than test
Zugeq	Unsigned byte greater than or equal test
Zugtr	Unsigned byte greater than test
Zuleq	Unsigned byte less than or equal test
Zules	Unsigned byte less than test
Zintgeq	Integer greater than or equal test
Zintgtr	Integer greater than test
Zintleq	Integer less than or equal test
Zintles	Integer less than test
Zuintgeq	Unsigned integer greater than or equal test
Zuintgtr	Unsigned integer greater than test
Zuintleq	Unsigned integer less than or equal test
Zuintles	Unsigned integer less than test
Zbbounds	Byte bounds checking
Zubbounds	Unsigned byte bounds checking
Zwbounds	Integer bounds checking
Zuwbounds	Unsigned integer bounds checking

Table 3-2. Pascal Library Routines (for 6800) (Cont'd)

STRING OPERATIONS

NAME	PURPOSE
STmove	String assignment
STequ	String equality test
STneq	String inequality test
STgeq	String greater than or equal test
STgtr	String greater than test
STleq	String less than or equal test
STles	String less than test
CHequ	String-char equality test
CHneq	String-char inequality test
CHgeq	String-char greater than or equal test
CHgtr	String-char greater than test
CHleq	String-char less than or equal test
CHles	String-char less than test

UTILITY ROUTINES

NAME	PURPOSE
SEXtend	Signed byte to integer extension
ADD_BtoX	Unsigned byte and integer addition
ZBtoIadd	Byte to integer addition
ZBtoIsub	Byte to integer subtraction
ZuBtoIadd	Unsigned byte to integer addition
ZuBtoIsub	Unsigned byte to integer subtraction
TFR_DtoX	Transfer contents of D to X
TFR_XtoD	Transfer contents of X to D
LEAX_B_X	Signed byte and integer addition
LEAX_D_X	Integer addition
JMPI_B_X	Jump table indirect with index in B
JMPI_D_X	Jump table indirect with index in D(A,B)
PUSHX	Push register X onto stack

MISCELLANEOUS

NAME	PURPOSE
END_DATA_	Label at the end of the library that can be used to allocate the HEAP area.
Z_END_PROGRAM	Label called at the end of the main program.
EMPTY_SET_	The largest possible empty set for the 6800.

ARRAY REFERENCE ROUTINES

The Pascal/64000 compiler supports generalized array references with up to 10 indices. The array reference routines are called with the parameters:

DOPE_VECTOR - address of a record describing the array.

BASE_ADDRESS - address of the first element of the array. (May be indirected like a VAR parameter.)

Index_list - addresses of the actual index expressions (one for each formal index expression). Each index may be indirected like a VAR parameter.

The array reference routines return the computed memory address to the X register.

ARRAY-

The ARRAY_ routine returns the memory address of an n-dimensional array reference expression. An alternate form of array reference, ARRAYN_, is used if the array reference variable expression specifies less than the defined number of indices. In this case, the call is similar, but the number of actual index parameters is passed in register A of the 6800.

The array reference call for the 3-index array variable expression:

```
AA(I,J,7)
```

would be:

```
JSR   ARRAY-
FDB   DOPE_VECTOR    ;address of dope-vector for array
FDB   AA             ;base address of array AA
FDB   I              ;address of first index expression
FDB   J              ;address of second index expression
FDB   ADDR_CONST_7   ;address of the third index expression
```

The base address may be indirect; this would be indicated by a word of "0" in the Base_Address location. If indirect, the address pointing to the array would be found in the word following the word of "0". To illustrate the use of indirection for the base address, consider variable BB defined as a pointer to an array of the same type as AA in the above example. A reference to an element of BB% with the variable array expression:

```
BB%(6+Y,J,7)
```

would generate a call to ARRAY_ in the form:

```
JSR   ARRAY-
FDB   DOPE_VECTOR    ;address of dope-vector for array of
                     ;same type as AA
FDB   0              ;Indirect address indicator
FDB   BB             ;address of BB which points to array
FDB   D_TEMP         ;address where temp value (6+Y) stored
FDB   J              ;address of second index expression
FDB   ADDR_CONST_7   ;address of the third index expression
```

ARRAYN-

The routine ARRAYN_ is used to compute the address of an array "row" which has been referenced by an array variable expression with less than the defined number of formal indices. The actual number of indices is passed in register A of the 6800.

The ARRAYN_ call for a two-indexed array variable expression with a 3-dimensional array AA (used in the previous example), is as follows:

```
AA(I,J)
```

would be:

```
LDAA  #2             ;number of indices
JSR   ARRAYN-
FDB   DOPE_VECTOR    ;for array AA
FDB   AA             ;base address of array AA
FDB   I              ;address of second index
                     ; expression
FDB   J              ;address of second index
                     ; expression
```

Generalized Array DOPE_VECTOR

For the general Pascal array defined by the declaration:

```
                                                          [1]

A : ARRAY [I1L..I1H,I2L..I2H,..., InMINUS1L..InMUS1H,InL..InH]
         OF any_type;
```

the address of the array element defined by the variable expression A[I1,I2,...,InMINUS,In] is computed by the expression:

```
element_address := base_address                           [2]
              +(I1-I1L)*(D2*D3 ... *Dn*BPE)
              +(I2-I2L)*(D3* ... *Dn*BPE)
               .
               .
               .
              +(InMINUS_1-InMINUS_1L)*(Dn*BPE)
              +(In-InL)*(BPE)
```

where:

```
In  -  Actual index expression for index n
InL -  Formal index lower bound for index n
InH -  Formal index upper bound for index n
Dn  -  "Row" size := (InH-InL+1) for index n
BPE -  Bytes Per Element of array element
         (size of data type any_type)
```

The constant terms representing the product of the index lower bounds (InL) and the "row" widths of the form:

```
PROD1 := (D1PLUS_1* ... *DnMINUS_1*Dn*BPE)
```

may be combined into one constant called the OFFSET_CONSTANT. This constant is defined as:

```
OFFSET_CONSTANT := I1L*PROD1                              [3]
                 + I2L*PROD2
                    .
                    .
                    .
                 + InMINUS_1L*PRODnMINUS_1
                 + InL*PRODn
```

The resulting combined formula can now be written as:

```
ADDRESS := BASE_ADDRESS + (-OFFSET_CONSTANT)              [4]
        + I1*PROD1+I2*PROD2 + ... + InPRODn
```

Pascal defines the ARRAY type recursively as a single-dimensioned array of any declarable Pascal type. Thus multi-dimensioned arrays are simply defined as ARRAYS of arrays. An array may be referred to in its entirety (a so-called ENTIRE variable) by referring to the array by its name using no parameters. A variable expression allows the user to refer to an INDEXED element type by allowing from 1 to N index expressions to be used in an array reference. Pascal arrays are stored such that the right-most subscript changes fastest.

For the array defined as in [1], an array variable expression with N-1 expressions will access one element of the type:

```
ARRAY[InL..InH] OF any_type.
```

An individual element of this ROW type may be accessed by the address expression:

```
                                                    [5]
row_address_nMINUS1 := BASE_ADDRESS
                       +(I1-I1L)*(D2*D3:... Dn*BPE)
                       +(I2-I2L)*(D3*...*Dn*BPE)

                          .
                          .
                          .
                       +(InMINUS_1-InMINUS_1L)
                          *(DnMINUS_1*Dn*BPE)
```

When we compute a row address, we will compute the address as defined in [4], but omitting the unnecessary product terms. However, this expression has already incorporated the term:

```
ROWnMINUS_1_OFFSET := InL*PRODn
```

in the OFFSET_CONSTANT, so it must now be added back to the ADDRESS as defined in [5]. Thus, the computational expression used to compute the array row reference, using N-1 index expressions, is:

```
                                                       [6]
row_address:= BASE_ADDRESS+(-OFFSET_CONSTANT)+I1*PROD1+I2*PROD_2
                 +InMINUS_1*PRODnMINUS_1+ROWnMINUS_1_OFFSET
```

For each additional missing index expression there is one less multiplication but one more addition for the OFFSET_CONSTANT correction.

The form of the general array reference DOPE_VECTOR is equivalent to:

```
DOPE_VECTOR     DEFW  N                    ;number of dimensions
                DEFW  -(OFFSET_CONSTANT)   ;negative of constant
                DEFW  PROD_1
                DEFW  PROD_2
                 .
                 .
                 .
                DEFW  PROD_N               ;The information
                DEFW  ROW1_OFFSET          ;needed in the
```

```
DEFW   ROW2_OFFSET          ;dope_vector for
 .                          ;the ARRAY_ routine
 .                          ;would be complete
 .                          ;at this line.
DEFW   ROWnMINUS_1_OFFSET
```

NOTE

Users who write assembly language programs that define and use multi-dimension arrays to be used with the ARRAY_ and ARRAYN_ routines need to ensure that their use is consistent with the Pascal compiler. In order to accomplish this, it is recommended that the user write a simple Pascal program defining and using the arrays. The user can then use the expanded listing file or the ASM_FILE option to determine how the Pascal compiler accesses these arrays and defines the array dope vectors. It is important that the user's array dope vector be identical to that produced by the compiler.

PARAMETER PASSING

General

A procedure or function declaration may contain an optional parameter list in which the formal parameters are declared. A parameter may be passed by value or by reference. A value parameter is copied locally into the called routine where it may be changed (by assignment) without affecting the original actual parameter.

The keyword VAR preceding a parameter declaration indicates a parameter is to be passed by reference. This means that the address of the actual parameter is being passed to the called routine and that the called routine has (indirect) access to the actual parameter. Therefore, assignment to a VAR parameter within the body of a routine will affect the value of the original actual parameter.

A procedure using both reference and value parameters is shown in the following example:

```
PROCEDURE PROCA (VAR I: BYTE; VALUEP: BYTE);
  BEGIN
    I := I+VALUEP;
  END;
.
```

This example will also be used in the description of the parameter passing routine PARAM_.

PARAM-

PARAM_ is the routine evoked to transfer parameters from the calling routine to the called routine. The calling routine lists the parameters in order with an "FDB address" for each parameter immediately following the call. The called routine will return to the instruction immediately following the last parameter. The calling routine may indicate that one level of indirect address is required by inserting a zero word before the actual parameter. This tells PARAM_ that the indicated parameter is the address of a variable pointing to the actual parameter. (Note that this use of value 0 to indicate indirection prevents the passing of a variable at absolute address 0. The FDB 0 would always be interpreted as an indirection flag and not an actual data location address.)

The Pascal statement:

```
PROCA (FIRST,SECOND%);
```

will produce the following call sequence:

```
JSR  PROCA              ;Call procedure with 2 parameters
FDB  FIRST              ;Address of first parameter
FDB  0                  ;Indirect parameter flag
FDB  SECOND             ;Contains a pointer to the actual
     parameter
 ...                    ;Next instruction after call
```

The called procedure with parameters (PROCA in this example) upon entry will load the X register with the address of the parameter dope vector then call the parameter passing routine PARAM_, which will pass the parameter from the calling routine to the called routine. For the sample procedure PROCA, the code would be:

```
LDX #PROCA_C
JSR PARAM-
```

The parameter dope vector is a block of words containing the address of the data area, the number of parameters, and a description for each parameter.

The parameter descriptor is the number of bytes for a value parameter. If the descriptor has the value -2, it is a parameter passed by an address (a Pascal VAR parameter).

Parameter Dope Vector

A parameter dope vector has the form:

```
FDB   Address of parameter data area
FDB   Number of parameters
FDB   First parameter descriptor
FDB   Second parameter descriptor
            .
            .
            .
etc.
```

The parameter descriptors are integers that indicate the following:

```
DESCRIPTOR                  INDICATES
   >0           number of bytes to be transferred
                       for value parameter
   <0           pass address only for VAR parameter
                (A VAR descriptor will always be -2
                indicating either a 16-bit or a
                2-byte address.)
```

When PARAM_ is called, the Stack appears as follows:

```
                                 Stack
                         |---------------|
                         |    Previous   |
                         |     stack     |
                         |     info      |
                         |---------------|
                         |      ---      |
                         |---------------|  Return address (1st
                         |               |  parameter) of the
                         |               |  caller of PARAM_'s
                         |    RetAdd2    |  caller.  (Calling
                         |               |  routine)
  STACK                  |---------------|
  pointer--->    0,S     |    RetAdd1    |   Return address from
                         |               |   caller of PARAM_.
                         |_______________|   (Called routine)
```

An expanded listing of a sample procedure with $RECURSIVE OFF$ is as follows:

```
$RECURSIVE OFF$
PROCEDURE PROCA(VAR I: BYTE; VALUEP: BYTE);
 BEGIN

                  PROCA:
                             LDX  #PROCA_C
                             JSR  PARAM_          ;Pass
                                                  ;parameters
  I:= I+VALUEP;
                             LDAB PROCA_D+2
                             LDX  PROCA_D
                             ADDB 0,X
                             STAB 0,X
 END;
                             RTS
                  PROCA_C:                    ;Parameter dope
                                              ;vector
                             FDB PROCA_D      ;Address of
                             parameter
                                              ;data area
```

```
                FDB 2           ; 2 parameters
                FDB -2          ; Var address
                FDB 1           ; Byte value
PROCA_D:
                RMB 3           ;Local data area
```

RPARAM-

RPARAM_ is used for procedures or functions which have been declared with the $RECURSIVE ON$ option. For recursive routines the local data must be copied onto the stack, then the parameters must be passed leaving the stack in the form used by REXIT_ to restore the local data at the end of the routine. (Refer to the following section entitled "Recursive Entry and Exit".)

The parameter dope vector for a recursive routine requires three more words of information than the nonrecursive dope vector described previously for PARAM_. It must have two extra words in the dope vector indicating the starting address and the size of the local data area which must be saved upon entry like the procedure RENTRY_ and is to be restored on exit by REXIT_. Another word is added after the starting address of the parameter data area to indicate the number of bytes of parameter data to help in safely passing local data recursively. Since special care must be taken to enter recursive routines with parameters (even if RENTRY_ were called before calling RPARAM_), the functions of RENTRY_ and PARAM_ have been combined in the procedure RPARAM_.

Procedure Steps for RPARAM_:

(1) Local data is copied onto the stack (like RENTRY_).
This is necessary to preserve old values of local data which are, in fact, statically allocated.

(2) Pass parameters onto the stack.
If parameters are passed directly from the actual parameter address to the local data area, it is possible for a parameter being passed recursively to be written over with another value before the original value has been passed.

(3) Pass parameters from the stack into the local data area.
This restores the actual parameters into the local data for processing by the routine.

A dope vector for recursive procedures which invoke RPARAM_ is as follows:

```
FDB address        ;address for beginning of local data area
FDB NumBytes       ;total number of bytes of local data to be
                   ;copied
FDB param_area     ;address of parameter data area
FDB #ofBytes       ;number of bytes of parameters
FDB #ofParam       ;number of parameters to be passed
FDB descript1      ;1st parameter descriptor
FDB descript2      ;2nd parameter descriptor
   ...
FDB descriptlast ;description of last parameter
```

An expanded list of a sample program with $RECURSIVE ON$ is as follows:

```
$RECURSIVE ON$
PROCEDURE PROCA(VAR I: BYTE; VALUEP: BYTE);
 BEGIN
                  PROCA:
                              LDX  #PROCA_C
                              JSR  RPARAM_          ;Pass
                                                    ;parameters
  I:= I+VALUEP;
                              LDAB PROCA_D+2
                              LDX  PROCA_D
                              ADDB 0,X
                              STAB 0,X
 END;
                              LDX #PROCA_C      ;Restore
                                                ;local data
                              JSR  REXIT-
                              RTS
                  PROCA_C:                      ;Parameter dope
                                                ;vector
                              FDB PROCA_D       ;Address of data
                                                ;area
                              FDB 3             ;Total bytes
                                                ;of local data
                              FDB PROCA_D
                              FDB 3             ;bytes of
                                                ;parameters
                              FDB 2             ;2 parameters
                              FDB -2            ;Var address
                              FDB 1             ;Byte value
                  PROCA_D:
                              RMB 3             ;Local data area
```

NOTE

Users who write assembly language programs that define and use procedures and functions, particularly with parameters, need to ensure that their use is consistent with the Pascal compiler. In order to accomplish this, it is recommended that the user write a simple Pascal program defining the procedure or function with the desired parameter list and an empty BEGIN END block for code. The user can then use the expanded listing file or the ASM_FILE option to determine how the Pascal compiler enters and exits the equivalent do-nothing procedure and how the parameter dope vector is defined. It is important that the user's assembly language routines follow the same entry, parameter passing, and exit code produced by the compiler. In particular, it is important that the parameter dope vector be identical to that produced by the compiler and that recursive or static mode declarations (and use) be consistent.

RECURSIVE ENTRY AND EXIT

Pascal/64000 supports recursive and reentrant calling sequences for procedures compiled for the 6800 with the $RECURSIVE ON$ option by additional run-time entry and exit code. This code causes the local data area of a procedure to be copied onto the stack before entry to the procedure and to be restored from the stack upon exit from the procedure. These functions are performed by procedures RENTRY_ and REXIT_.

RENTRY-

RENTRY_ is called at the entry point of a procedure or function which has been declared with the option $RECURSIVE ON$. RENTRYwill copy the local data area of a procedure onto the stack so that this status may be restored upon exit (by REXIT_). RENTRYis only called for routines with no parameters. For routines with parameters the procedure RPARAM_ will perform this local parameter saving before passing the parameters.

Before calling RENTRY_ the X register must contain the address of a data block containing the starting address of the local data area to be saved as the first word and the total number of bytes of data in the local area as the second word. The calling sequence for RENTRY_ would be as follows:

```
LDX #LOCAL_DATA
JSR RENTRY-
```

For a procedure with 5 bytes of data starting at address DATA_AREA, the LOCAL_DATA block would be as follows:

```
LOCAL_DATA     FDB DATA_AREA    ;Address of data area
               FDB 5            ;Total bytes local data
```

RENTRY_ is called upon entry to a recursive Pascal procedure or function and will change the values in all registers.

```
                        Stack upon entrance
                         |---------------|
                         |   Previous    |
                         |     stack     |
                         |     info      |
                         |---------------|
                         |      ...      |
                         |---------------|
                         |               |  Return address of the
                         |    RetAdd2    |  caller of RENTRY_'s
                         |               |  caller.
STACK                    |---------------|
pointer---> 0,S          |    RetAdd1    |  Return address from
                         |_______________|  caller of RENTRY_.
```

Assuming five bytes of data are saved, upon exit the stack would appear as follows:

```
                        |---------------|
                        |    RetAdd2    |  Same as above
                        |---------------|
   STACK                |    5 bytes    |
   pointer---> 0,S      |    of data    |
                        |---------------|
```

REXIT-

REXIT_ is called at the exit point of a procedure or function which has been declared with the option $RECURSIVE ON$. REXITwill copy the local data area of a procedure from the stack back

to the local data area and leave the return address of the calling procedure on top of the stack so that a normal return from subroutine may be executed.

Before calling REXIT_ the X register must contain the address of a data block containing the starting address of the local data area to be saved as the first word and the total number of bytes of data in the local area as the second word. This is the same data block used in the call to RENTRY_ above or in the call to RPARAM_ if the procedure has parameters. The calling sequence is:

```
LDX #LOCAL_DATA
JSR REXIT-
```

REXIT_ is called before exit from a recursive Pascal procedure or function. In order to allow the saving of function return values REXIT_ will preserve the values in the A and B registers and copy them as the D register into the X register.

```
                      Stack upon entrance:
                        |---------------|
                        |    RetAdd2    |  Return address of the
                        |               |  caller of RENTRY_'s
                        |               |  caller.
                        |---------------|
                        |    5 bytes    |
                        |    of data    |
   STACK                |---------------|
   pointer--->  0.S     |    RetAdd1    |  Return address from
                        |_______________|  caller
```

After restoring the five bytes of data and returning to the caller of RENTRY_'s caller (RetAdd2), the stack would appear as follows:

```
                        |---------------|
                        |   Previous    |
                        |    stack      |
   STACK                |     info      |
   pointer--->  0,S     |_______________|
```

DYNAMIC MEMORY ALLOCATION

Pascal/64000 supports dynamic allocation and deallocation of storage space through the procedures NEW, DISPOSE, MARK, RELEASE, and INITHEAP.

INITHEAP

The user declares a block of memory to be used as the memory pool or heap by calling: INITHEAP (Start_address, Length_in_bytes . INTEGER). The procedure, INITHEAP, must be declared EXTERNAL in the declaration block of a program. The resultant heap will be six bytes smaller than length_in_bytes.

NEW

The procedure NEW (Pointer : Pointer_to_type) is used to allocate space. The procedure, NEW, searches for available space in a free-list of ascending size blocks. When a block is found that is the proper size or larger, it is allocated and any space left over is returned to the free-list in a new place corresponding to the size of the leftover block. If the referenced block is four or less bytes in size, four bytes will be allocated.

DISPOSE

The procedure DISPOSE is exactly the reverse of the procedure NEW. It indicates that storage occupied by the indicated variable is no longer required.

MARK

This procedure marks the state of the heap in the designated variable that may be of any pointer type. The variable must not be subsequently altered by assignment.

RELEASE

The procedure RELEASE restores the state of the heap to the value in the indicated variable. This will have the effect of disposing all heap objects created by the NEW procedure since the variable was marked. The variable must contain a value returned by a previous call to MARK; this value may not have been passed previously as a parameter to RELEASE.

STANDARD BYTE ROUTINES

For standard byte routines, parameter values are passed using specific registers. The operands are 8-bit signed bytes. There are two groups of byte operations: the unary byte operation, and the binary byte operation. These operations are discussed in the following paragraphs.

Unary Byte Operations

Zbyteabs	Byte absolute value
Zbyteneg	Byte negation

The unary byte operation is of the form:

```
RESULT :=  <op>  B1
```

where:

```
B1 is loaded in register B
```

The library routine is called after loading B1 into the B register. The byte RESULT is returned in the B register.

REGISTER ALLOCATION SUMMARY :: UNARY BYTE OPERATIONS

```
Input:  B contains byte parameter B1

Output: B contains byte RESULT

Registers:
    Modified:  B,CC
    Unchanged: A,X
```

Binary Byte Operations

Zbyteadd	Byte addition
Zubyteadd	Unsigned byte addition
Zbytesub	Byte subtraction
Zubytesub	Unsigned byte subtraction
Zbytemul	Byte multiplication
Zubytemul	Unsigned byte multiplication
Zbytediv	Byte division
Zubytediv	Unsigned byte division
Dbytemul	Debug byte multiply
Dubytemul	Debug unsigned byte multiply

The binary byte operations compute the byte (8-bit) result of arithmetic expressions of the form:

```
RESULT := B1 <op> B2
```

where:

```
B1 is loaded in register A
B2 is loaded in register B
```

The library routine is called after loading the operands, B1 and B2, into the A and B registers. The RESULT is returned in the B register.

REGISTER ALLOCATION SUMMARY :: BINARY BYTE OPERATIONS

```
Input:  A contains byte parameter B1
        B contains byte parameter B2

 Output: B contains byte RESULT

Registers:
    Modified:  A,B,CC
    Unchanged: X
```

NOTE

For each of the binary byte operations the A register contains useful information upon return. Specifically:

for **Zbyteadd,Zubytadd, Zbytesub,Zubytesub, Zbytemul,Zubytemul, Dbytemul, Dubytemul**	A contains the upper 8 bits of the 16-bit result
for **Zbytediv,Zubytediv**	A contains the MODULUS value

The routines Dbytemul and Dubytemul are used in the debug library to check for overflow and underflow for byte multiplication. Since the normal routines, Zbytemul and Zubytemul, allow a 16-bit answer, there are no overflow or underflow conditions. The debug versions will check to insure that a byte result is in the range -128..127 and that an unsigned byte result is in the range 0..255. These routines will only be called when the $DEBUG ON$ option is enabled and an expression has two byte operands with a byte result.

STANDARD INTEGER ROUTINES

The integer operations require 16-bit operands. The two 8-bit accumulators (A,B) are normally used as a 16-bit register (called D) for these routines. As a register pair, the high-order byte is always stored in the A register and the low-order byte is stored in the B register. The X register is a 16-bit register and is used for binary operations and for returning some results. There are two groups of integer operations: the unary integer operation and the binary integer operation. These operations are discussed in the following paragraphs.

Unary Integer Operations

Zintabs Integer absolute value
Zintneg Integer negation

The unary integer operation is of the form:

```
            RESULT := <op> I1
where:
            I1 is loaded in register pair D(A,B).
```

The library routine is called after loading I1 into the D register. The integer RESULT is returned in the D register.

REGISTER ALLOCATION SUMMARY :: UNARY INTEGER OPERATIONS

```
Input:  D(A,B) contains integer parameter I1

Output: D(A,B) contains integer RESULT

Registers:     (for Zintabs and Zintneg)
    Modified:  A,B,CC
    Unchanged: X
```

Binary Integer Operations

Zintadd Integer addition
Zuintadd Unsigned integer addition
Zintsub Integer subtraction
Zuintsub Unsigned integer subtraction
Zintmul Integer multiplication
Zuintmul Unsigned integer multiplication
Zintdiv Integer division
Zuintdiv Unsigned integer division

The binary integer operations compute the integer result of arithmetic expressions in the form:

```
            RESULT := I1 <op> I2    where: I1 is loaded in register X I2 is loaded in regist-
er D(A,B)
```

The library routine is called after loading the operands, I1 and I2, into the X and D(A,B) registers. The RESULT is returned in the D(A,B) register.

REGISTER ALLOCATION SUMMARY :: BINARY INTERGER OPERATIONS

```
Input:  X contains integer parameter I1
        D(A,B) contains integer parameter I2

Output: D(A,B) contains integer RESULT

Registers:     (for Zintmul, Zuintmul, Zintdiv, Zuintdiv
    Modified:  A,B,CC,X
    Unchanged: none

Registers:     (for Zintadd Zuintadd, Zintsub,
                   and Zuintsub)
```

```
Modified:  A,B,CC
Unchanged: X
```

NOTE

For some of the binary integer operations, the X register contains useful information upon return. Specifically:

for **Zintmul, Zuintmul**	X contains the upper 16 bits of the 32-bit result
for **Zintdiv, Zuintdiv**	X contains the MODULUS value

BYTE AND WORDS SHIFTS

Pascal/64000 supports logical and circular shifting of both bytes (8-bit) and words (16-bit) quantities using the predefined functions SHIFT and ROTATE. These functions are available when compiling with the $EXTENSIONS ON$ option of the compiler. The DIV operator using powers of 2 may be used to accomplish an arithmetic right shift (i.e., with sign extension). For example, X DIV 2 is equivalent to a one-bit right shift with sign extension.

SHIFT

Logical shifting with zero fill will shift the quantity left or right placing a zero in the most (right shift) or least (left shift) significant bit for each shift. The function is called with two parameters: the quantity to be shifted and the number of bit positions to shift. The function call in Pascal is of the form:

```
variable:= SHIFT(expression,n);
```

where:

```
expression    is any expression, variable or constant

         n    is the number of bits to be shifted
```

where:

```
       n>0    results in a left shift
       n<0    results in a right shift
```

ROTATE

Circular shifting rotates the quantity left or right and fills the vacated position with the bit shifted out of the other end. The function is called with two parameters: the quantity to be shifted and the number of bit positions to shift. The function call in Pascal is of the form:

```
variable:= ROTATE(expression,n);
```

where:

```
expression    is any expression, variable or constant

         n    is the number of bits to be shifted
```

where:

```
       n>0    results in a left circular shift
       n<0    results in a right cicular shift
```

Pascal/64000 determines the size (1 or 2 bytes) of the data being shifted by the type of the first parameter expression. The type of result returned by the function SHIFT or ROTATE is the same type as the type of the first parameter expression.

Byte Shifts

Zbshift Byte shift logical with zero fill
Zbshiftc Byte shift circular

The byte shift operations compute the byte result of shift expressions of the form:

```
  RESULT := SHIFT(B1,B2);

or

  RESULT := ROTATE(B1,B2);
```

where:

```
B1 is loaded in register A
B2 is loaded in register B
```

The library routine is called after loading B1 into the A register and B2 into the B register. The byte RESULT is returned in the B register.

REGISTER ALLOCATION SUMMARY :: BYTE SHIFT OPERATIONS

```
Input:  A contains byte to be shifted, B1
        B contains number of positions to shift, B2

Output: B contains byte RESULT

Registers:
    Modified:  B,CC
    Unchanged: A,X
```

Word Shifts

Zwshift Word shift logical with zero fill
Zwshiftc Word shift circular

The word shift operations compute the word result of shift expressions of the form:

```
RESULT := SHIFT(I1,I2);
```

```
or

  RESULT := ROTATE(I1,I2);
```

where:

```
I1 is loaded in register X
I2 is loaded in register B
```

The library routine is called after loading I1 into the X register and I2 into the B register. The word RESULT is returned in the D(A,B) register.

REGISTER ALLOCATION SUMMARY :: INTEGER SHIFT OPERATIONS

```
Input:  X contains word to be shifted, I1
        B contains the number of positions to shift, I2

Output: D(A,B) contains word RESULT

Registers:
    Modified:  A,B,CC
    Unchanged: X
```

BYTE AND WORD SET OPERATIONS

Pascal/64000 supports 8-bit and 16-bit sets and are called bytesets and wordsets, respectively for the following discussion. For these sets all Pascal set operations are available. In the following descriptions of the set routines assume the following definition of set types and data variables:

```
TYPE
 SET8 = SET OF 0..7;
 SET16= SET OF 0..15;
VAR
 BYTESET: SET8;
 WORDSET: SET16;
 B1,B2: BYTE;
 I1,I2: INTEGER;
 W1,W2: SET16;
```

Byte Set Operations

Zbinset8	Byte in 8-bit set
Zbtoset8	Byte to 8-bit set
Zwinset8	Word in 8-bit set
Zwtoset8	Word to 8-bit set

Zbinset8. This routine is used to test the set membership of a byte value in a specified byte set. For example, the Pascal/64000 expression:

```
B1 IN BYTESET
```

is a Boolean expression whose value is TRUE if bit B1 of the set BYTESET is set and FALSE if bit B1 of BYTESET is reset.

REGISTER ALLOCATION SUMMARY :: ZBINSET8

```
Input:  B contains the byte set being compared
        A contains byte value to be tested

Output: B set to 0, Z flag set if value not in set
        B set to 1, Z flag reset if value in set

Registers:
    Modified:  B,CC,X
    Unchanged: A
```

Zbtoset8. This routine converts a byte into an 8-bit set. The only valid input values are 0 through 7. Out of range values are detected in the debug library, DLIB6800:D6800, but are not detected in either LIB6800:L6800 or SLIB6800:S6800 and may produce out of range results. The Pascal statements:

```
B1:= 7;
BYTESET:= SET8[B1]
```

will assign to BYTESET a byte with the most significant bit set and all the others reset. (BYTESET will contain the hex value 80H.)

REGISTER ALLOCATION SUMMARY :: ZBTOSET8

```
Input:  B contains byte value to be converted

Output: B contains the byteset result

Registers:
    Modified:  B,CC,X
    Unchanged: A
```

Zwinset8. This routine is used to test the set membership of a byte value in a specified word set. For example, the Pascal/64000 expression:

```
W1 IN BYTESET
```

is a Boolean expression whose value is TRUE if bit W1 of the set BYTESET is set and FALSE if bit W1 of BYTESET is reset.

REGISTER ALLOCATION SUMMARY :: ZWINSET8

```
Input:  B contains the byte set being compared
        X contains the word value to be tested

Output: B set to 0, Z flag set if value not in set
        B set to 1, Z flag reset if value in set
```

```
Registers:
    Modified:  B,CC
    Unchanged: A,X
```

Zwtoset8. This routine converts a word into an 8-bit set. The only valid input values are 0 through 7. Out of range values are detected in the debug library, DLIB6800:D6800, but are not detected in either LIB6800:L6800 or SLIB6800:S6800 and may produce out of range results. The Pascal statements:

```
I1:= 7;
BYTESET:= SET8[I1]
```

will assign to BYTESET a byte with the most significant bit set and all the others reset. (BYTESET will contain the hex value 8000H.)

REGISTER ALLOCATION SUMMARY :: ZWTOSET8

```
Input:  D(A,B) contains word value to be converted
Output: B contains the byteset result

 Registers:
    Modified:  B,X
    Unchanged: A
```

Word Set Operations

Zbinset16	Byte in 16-bit set
Zbtoset16	Byte to 16-bit set
Zwinset16	Word in 16-bit set
Zwtoset16	Word to 16-bit set

Zbinset16. This routine is used to test the set membership of a byte value in a specified word set. For example, the Pascal/64000 expression:

```
B1 IN WORDSET
```

is a Boolean expression whose value is TRUE if bit B1 of the set WORDSET is set and FALSE if bit B1 of WORDSET is reset.

REGISTER ALLOCATION SUMMARY :: ZBINSET16

```
Input:  B contains byte value to be tested
        X contains the word set being compared

Output: B set to 0, Z flag set if value not in set
        B set to 1, Z flag reset if value in set

Registers:
    Modified:  B,CC
    Unchanged: A,X
```

Zbtoset16. This routine converts a byte into a 16-bit set. The only valid input values are 0 through 15. Out of range values are detected in the debug library, DLIB6800:D6800, but are not detected in either LIB6800:L6800 or SLIB6800:S6800 and may produce out of range results. The Pascal statements:

```
B1:= 15;
WORDSET:= SET16[B1]
```

will assign to WORDSET a byte with the most significant bit set and all the others reset. (WORDSET will contain the hex value 8000H.)

REGISTER ALLOCATION SUMMARY :: ZBTOSET16

```
Input:  B contains byte value to be converted
Output: D(A,B) contains the wordset result

  Registers:
      Modified:  A,B,CC,X
      Unchanged: none
```

Zwinset16. This routine is used to test the set membership of a word value in a specified word set. For example, the Pascal/64000 expression:

```
I1 IN WORDSET
```

is a Boolean expression whose value is TRUE if bit I1 of the set WORDSET is set and FALSE if bit I1 of WORDSET is reset.

REGISTER ALLOCATION SUMMARY :: ZWINSET16

```
Input:  D(A,B) contains word value to be tested
        X contains the word set being compared

Output: B set to 0, Z flag set if value not in set
        B set to 1, Z flag reset if value in set

Registers:
    Modified:  A,B,CC,X
    Unchanged: none
```

Zwtoset16. This routine converts a word into a 16-bit set. The only valid input values are 0 through 15. Out of range values are detected in the debug library, DLIB6800:D6800, but are not detected in either LIB6800:L6800 or SLIB6800:S6800 and may produce out of range results. The Pascal statements:

```
I1:= 15;
WORDSET:= SET16[I1]
```

will assign to WORDSET a byte with the most significant bit set and all the others reset. (WORDSET will contain the hex value 8000H.)

REGISTER ALLOCATION SUMMARY :: ZWINSET16

```
Input:  D(A,B) contains word value to be
converted
Output: D(A,B) contains the wordset result

  Registers:
      Modified:  A,B,CC,X
      Unchanged: none
```

Binary Word Set Operations

Zset16int Intersection of 16-bit sets
Zset16uni Union of 16-bit sets
Zset16dif Set difference of 16-bit sets
Zset16geq 16-bit set greater than or equal
Zset16leq 16-bit set less than or equal

Zset16int. This routine is used to compute the set intersection of two wordsets. For expressions in the form:

```
W1 * W2
```

the set intersection is the wordset containing only the elements contained in both wordset W1 and wordset W2.

REGISTER ALLOCATION SUMMARY :: ZSET16INT

```
Input :  X contains the wordset W1
         D(A,B) contains the wordset W2
Output:  D(A,B) contains the wordset result

Registers:
    Modified : A,B,CC
    Unchanged: X
```

Zset16uni. This routine is used to compute the set union of two wordsets. For expressions in the form:

```
W1 + W2
```

the set union is the wordset containing all the elements in both wordset W1 and wordset W2.

REGISTER ALLOCATION SUMMARY :: ZSET16UNI

```
Input :  X contains the wordset W1
         D(A,B) contains the wordset W2
Output:  D(A,B) contains the wordset result

Registers:
    Modified : A,B,CC
    Unchanged: X
```

Zset16dif. This routine is used to compute the set difference of two wordsets. For expressions in the form:

```
W1 - W2
```

the set difference is a set containing all the elements of wordset W1 which are not contained in wordset W2.

REGISTER ALLOCATION SUMMARY :: ZSET16DIF

```
Input :  X contains the wordset W1
         D(A,B) contains the wordset W2
Output:  D(A,B) contains the wordset result

Registers:
    Modified : A,B,CC
    Unchanged: X
```

Zset16geq. This routine is used to test the set inclusion of wordsets. For example , the Pascal/64000 expression:

```
WORDSET >= SET16[0,1,7]
```

is a Boolean expression whose value is TRUE if bits 0,1, and 7 of WORDSET are all set; otherwise the value is FALSE. This is equivalent to asking if the set with bits 0,1, and 7 set is a subset of WORDSET. For expressions of the form:

```
W1 >= W2
```

the Boolean result indicates whether W2 is a proper subset of W1.

REGISTER ALLOCATION SUMMARY :: ZSET16GEQ

```
Input:  X contains the wordset of superset W1
        D(A,B) contains the wordset of subset W2

Output: B set to 1, Z flag reset if W2 subset of W1
        B set to 0, Z flag set otherwise

Registers:
    Modified:  A,B,CC
    Unchanged: X
```

Zset16leq. This routine is used to test the set inclusion of wordsets. For example , the Pascal/64000 expression:

```
SET16[0,1,7] <= WORDSET
```

is a Boolean expression whose value is TRUE if bits 0,1, and 7 of WORDSET are all set; otherwise the value is FALSE. This is equivalent to asking if the set with bits 0,1, and 7 set is a subset of WORDSET. For expressions of the form:

```
W1 <= W2
```

the Boolean result indicates whether W1 is a proper subset of W2.

REGISTER ALLOCATION SUMMARY :: ZSET16LEQ

```
Input:  X contains the wordset of subset W1
        D(A,B) contains the wordset of superset W2
Output: B set to 1, Z flag reset if W1 subset of W2
        B set to 0, Z flag set otherwise
Registers:
    Modified:  A,B,CC
    Unchanged: X
```

MULTIBYTE OPERATIONS

The multibyte routines are used by the compiler to operate on multibyte records (or arrays) of the same type. Consider a record defined by the Pascal source:

```
TYPE
  PERSON=RECORD
           NAME : ARRAY[1..LENGTH] OF CHAR
           ADDRESS : ARRAY[1..LENGTH] OF CHAR
         END;
VAR
  SALESPERSON,TOP_SALESPERSON:PERSON;
```

Each of the variables, SALESPERSON and TOP_SALESPERSON, is a multibyte data structure containing 2*LENGTH bytes of information. Pascal only allows assignment and tests for equality or inequality of such data structures. Pascal will allow multibyte assignments of the form:

```
SALESPERSON := TOP_SALESPERSON;
```

and tests for equality and inequality of the form:

```
IF SALESPERSON = TOP_SALESPERSON THEN...   IF SALESPERSON <> TOP_SALESPERSON
THEN...
```

Pascal/64000 does not accept <=, <, >= or > comparisons for arrays or records. Therefore, the 6800 code generator will never generate calls to routines MBgeq, MBgtr, MBleq or MBles. However, these routines are included in the library for consistency and possible future extensions to the compiler.

Since there are not enough registers in the 6800 processor to pass all the necessary parameters for multibyte routines (two address operands and the number of bytes of the multibyte type), parameters are passed in a parameter list after the subroutine call in a manner similar to that used for user Pascal procedures with parameters.

For expression of the form:

```
LEFT <op> RIGHT
```

where LEFT and RIGHT are multibyte types, the calling sequence would be:

```
JSR  MB<op>
FDB  SIZE                  ;# Bytes of data to move
FDB  RIGHT_pointer         ;address RIGHT operand
FDB  LEFT                  ;address LEFT operand
```

If the RIGHT operand were a pointer to a multibyte type, an expression of the form:

```
LEFT <op> RIGHT_pointer%
```

would produce a call as follows:

```
JSR  MB<op>
FDB  SIZE
FDB  0                     ;Indication of indirection
FDB  RIGHT_pointer         ;address of address of right
     operand
FDB  LEFT                  ;address of left operand
```

MBmove

This routine is used to copy a multibyte record from one location to another. Expressions in the form: `TOP_SALESPERSON:= SALESPERSON`

will copy the entire data record for SALESPERSON (of length 2*LENGTH) into the data record for TOP_SALESPERSON.

REGISTER ALLOCATION SUMMARY :: MBMOVE

```
Input : no registers parameter list
          after call

Registers:
    Modified : A,B,CC,X
    Unchanged: None
```

Multibyte Comparisons

```
MBequ        Multibyte equality test
MBneq        Multibyte inequality test
MBgeq        Multibyte greater than or
               equal test
MBgtr        Multibyte greater than test
MBleq        Multibyte less than or equal
               test
MBles        Multibyte less than test
```

These routines are used to compare multibyte records. For expressions in the form:

```
TOP_SALESPERSON = SALESPERSON
```

or

```
TOP_SALESPERSON <> SALESPERSON
```

the compiler will compare each byte of the two data structures until all bytes have been found equal or until the first nonequal bytes are encountered. (For the <=, <, >= and > comparisons, the comparison is determined by the sense of the inequality of these first unequal bytes or the result equality applies when all bytes are equal.)

REGISTER ALLOCATION SUMMARY :: MBCOMPARISONS

```
Input : parameter list after call instruction

Output: IF condition is TRUE then
             B=1(TRUE), Z is reset
        IF condition is FALSE then
             B=0(FALSE), Z is set

Registers:
    Modified : A,B,CC,X
    Unchanged: none
```

MULTIBYTE SET OPERATIONS

Pascal/64000 supports 8-bit and 16-bit sets as well as larger sets with up to 256 elements. These larger sets requiring three or more bytes are referred to as multibyte sets. For multibyte sets all Pascal set operations are available. In the following descriptions of the multibyte set routines assume that some scalar and set types and data variables have been defined as follows:

```
TYPE
 LARGE_SET = SET OF 0..63; (This set uses 8 bytes of
                              storage)
VAR
 S1,S2: LARGE_SET;
 I1: INTEGER;
```

Since there are not enough registers in the 6800 processor to pass all the necessary parameters for multibyte routines (three address operands for the two operands and the result operand and the number of bytes of the multibyte set), parameters are passed in a parameter list after the subroutine call in a manner similar to that used for user Pascal procedures with parameters. For expression of the form:

```
RESULT := LEFT <op> RIGHT
```

where RESULT, LEFT and RIGHT are multibyte sets and the <op> is equality or inequality (<op>"=" or"<>"), the multibyte routine MBequ or MBneq is called with the calling sequence defined previously for these routines.

For the other operators ("+", "-", "*", "<=" ,">="), the RESULT address and the number of bytes in the set (<=31) must also be passed. Since the number of bytes in a multibyte set is in the range 0..31, it is passed in the B register. The calling sequence for these routines has the form:

```
LDA  B,#SIZE
JSR  SETmb<op>
FDB  RIGHT          ;address of RIGHT set
FDB  LEFT           ;address of LEFT set
FDB  RESULT         ;address of RESULT set
```

The calling sequence for routines INSETmb and TOSETmb is described within each routine description below.

Multibyte Set Routines

INSETmb	Multibyte set inclusion
TOSETmb	Multibyte set formation
SETmbINT	Multibyte set intersection
SETmbUNI	Multibyte set union
SETmbDIF	Multibyte set difference
SETmbGEQ	Multibyte set inclusion of sets
SETmbLEQ	Multibyte subset inclusion

INSETmb. This routine is used to test the set membership of an integer value in a multibyte set. For example the Pascal/64000 expression: I1 IN S1

is a Boolean expression whose value is TRUE if bit I1 of the multibyte set S1 is set and FALSE if bit I1 of S1 is reset.

Before the call to INSETmb, the integer value, I1, is loaded into the D(A,B) register and the address of the multibyte set is loaded into the X register. Upon return the B register contains the Boolean value result of the inclusion and the Z flag corresponds to the value in B.

REGISTER ALLOCATION SUMMARY :: INSETMB

```
Input : X contains address of the multibyte set
        D(A,B) contains the integer value I1

Output: IF I1 is an contained in S1
         THEN
           B set to 1 (TRUE), Z flag reset
         ELSE
           B set to 0 (FALSE), Z flag set

Registers:
    Modified : A,B,CC,X
    Unchanged: none
```

TOSETmb. This routine is used to convert an integer value into a multibyte set. For example the Pascal/64000 statements:

```
I1:= 63;
S1:= LARGE_SET[I1];
```

will assign to multibyte set S1 an 8 byte record with the most significant bit of the eighth byte set and all others reset.

The run-time libraries do not check for out of range values in the set conversion. The user should have the $RANGE ON$ option enabled if it is possible to convert illegal values. The $RANGE ON$ option will check to insure that the range of I1 is within the subrange of the multibyte set, LARGE_SET, before calling the set conversion routine.

In order to create the proper multibyte set, the entire contents of the set, S1, must be reset to 0 before setting the one bit representing I1. To do this, the routine needs to know the actual size of the set S1. This value is passed in the B register. The parameters I1 and S1 are passed using a parameter list after the call instruction.

```
LDAB  #SIZE     ;#bytes in set
JSR   TOSETmb
FDB   I1
FDB   S1
```

REGISTER ALLOCATION SUMMARY :: TOSETMB

```
Input : B contains the number of bytes in the
        multibyte set.
        Address of I1 and S1 passed in parameter
        list following the JSR instruction.
Output:

Registers:
    Modified : A,B,X,CC
    Unchanged: none
```

SETmbINT. This routine is used to compute the set intersection of two multibyte sets. For expressions in the form:

```
RESULT := S1 * S2
```

the set intersection is the set containing only the elements contained in both multibyte set S1 and multibyte set S2.

Before calling the multibyte set union routine, the B register is loaded with the number of bytes in the multibyte set. The other parameters are passed in a parameter list following the call as described previously.

SETmbUNI. This routine is used to compute the set union of two multibyte sets. For expressions in the form:

```
RESULT := S1 + S2
```

the set union is the set containing all the elements in both multibyte set S1 and multibyte set S2.

Before calling the multibyte set union routine, the B register is loaded with the number of bytes in the multibyte set. The other parameters are passed in a parameter list after the call as described previously.

SETmbDIF. This routine is used to compute the set difference of two multibyte sets. For expressions in the form:

```
RESULT := S1 - S2
```

the set difference is the set containing all the elements of multibyte set S1 which are not contained in multibyte set S2.

Before calling the multibyte set difference routine, the B register is loaded with the number of bytes in the multibyte set. The other parameters are passed in a parameter list following the call as described previously.

REGISTER ALLOCATION SUMMARY :: SETMBINTNT

SETmbUNI
SETmbDIF

```
Input : B contains the number of bytes in the
        multibyte set.
        S2, S1, and RESULT are passed in
        parameter list following the call
        instruction.
Registers:
    Modified : A,B,CC,X
    Unchanged: none
```

SETmbGEQ. This routine is used to compute the set inclusion of two multibyte sets. For example, the Pascal/64000 expression:

```
S1 >= LARGE_SET[0,7,63]
```

is a Boolean expression whose value is TRUE if bits 0, 7, and 63 of S1 are all set; otherwise the value is FALSE. This is equivalent to asking if the set with bits 0, 7, and 63 set is a subset of S1. Before calling the multibyte set inclusion routine, the number of bytes in the multibyte set is loaded into the B register.

SETmbLEQ. This routine is used to compute the set inclusion of two multibyte sets. For example, the Pascal/64000 expression:

```
LARGE_SET[0,7,63] <= S1
```

is a Boolean expression whose value is TRUE if bits 0, 7, and 63 of S1 are all set; otherwise the value is FALSE. This is equivalent to asking if the set with bits 0, 7, and 63 set is a subset of S1.

For expressions of the form:

```
S1 >= S2
```

the Boolean result indicates whether S2 is a proper subset of S1. Before calling these multibyte set inclusion routines, the number of bytes in the multibyte set is loaded into the B register. The other parameters are passed in a parameter list following the call. A sample calling sequence is as follows:

```
LDAB  #SIZE
JSR   SETmbGEQ
FDB   S2            ;address of RIGHT operand
FDB   S1            ;address of LEFT operand
```

The Boolean result is returned in the B register.

**REGISTER ALLOCATION SUMMARY :: SETMBGEQEQ
SETmbLEQ**

```
Input : B contains the number of bytes in the
          multibyte set.
        S1 and S2 are passed in parameter list
          following call instruction.

Output: IF S2 is a subset of S1
         THEN
           B set to 1 (TRUE), Z flag reset
         ELSE
           B set to 0 (FALSE), Z flag set

Registers:
    Modified : A,B,CC,X
    Unchanged: none
```

BYTE AND INTEGER COMPARISON AND BOUNDS CHECKING ROUTINES

The comparison (=,<>,>=,>,<=,<) of byte and integer variables produces a Boolean result (FALSE or TRUE) based on the signed or unsigned sequences of byte or word scalar types. In many cases where the comparison is being used as the condition for an IF, REPEAT, or WHILE statement, a branch is taken based on the result of the comparison. However, if the Boolean result is being assigned to a variable or if the expression has multiple comparisons (using AND and OR) an actual Boolean result is required. The byte and word comparison subroutines are used specifically in these situations where the Boolean result is necessary for further computations.

When the $RANGE ON$ option is enabled, all assignment statements and parameter passing of byte and word variables are checked to assure that they are within the bounds of the declared type. The range checking routines for byte and word variables are also described in this section.

Byte and Word Comparisons

Zcc	Carry cleared test
Zequ	Byte and integer equality test
Zneq	Byte and integer inequality test
Zgeq	Byte greater than or equal test
Zgtr	Byte greater than test
Zleq	Byte less than or equal test
Zles	Byte less than test
Zugeq	Unsigned byte greater than or equal test
Zugtr	Unsigned byte greater than test
Zuleq	Unsigned byte less than or equal test
Zules	Unsigned byte less than test

Library subroutines are called when the Boolean result is required of a byte comparison expression of the form:

```
RESULT := B1 <op> B2
```

where:

```
B1 and B2 are bytes
```

or an integer comparison for equality or inequality of the form;

```
RESULT := I1 <op> I2
```

where:

```
I1 and I2 are words.
```

For bytes, the calling sequence for comparison is:

```
LDB B1
CMP B2
JSR compare_routine
```

For integers, the calling sequence for equality and inequality comparisons is:

```
LDX B1
CPX B2
JSR compare_routine
```

The library routine is called after performing the comparison of bytes or words as indicated. The Boolean RESULT is returned in the B register.

REGISTER ALLOCATION SUMMARY :: BYTE COMPARISON

```
Input:  Condition codes resulting from comparison

Output: B set to 0, Z flag set result is FALSE
        B set to 1, Z flag reset if result is TRUE

Registers:
    Modified:  B,CC
    Unchanged: A,X
```

Word Comparison

Zintgeq Integer greater than or equal test
Zintgtr Integer greater than test
Zintleq Integer less than or equal test
Zintles Integer less than test
Zuintgeq Unsigned integer greater than or equal test
Zuintgtr Unsigned integer greater than test
Zuintleq Unsigned integer less than or equal test
Zuintles Unsigned integer less than test

For the comparison operators (>=, >, <=, <), the CPX instruction does not set all the condition codes correctly for the full 16-bit comparison. For signed and unsigned integers, these comparisons are done by run-time library routines which return a Boolean result representing the truth value of the comparison. For comparisons of the form:

```
I1 <op> I2
```

the calling sequence is:

```
JSR   word-compare_routine
FDB   I2
FDB   I1
```

The Boolean result of the comparison is returned in the B register.

REGISTER ALLOCATION SUMMARY :: WORD COMPARISON

```
Input:  Parameters I1 and I2 are passed as a
parameter
        list following the JSR instruction.

Output: B set to 0, Z flag set if result is FALSE.
        B set to 1, Z flag reset if result is TRUE.


Registers:
    Modified:  A,B,CC,X
    Unchanged: none
```

Byte Bounds Checking

Zbbounds Byte bounds checking
Zubbounds Unsigned byte bounds checking

The bounds checking for signed and unsigned byte variables use the same calling sequence and return the same results. The data being checked is loaded into the A register. The upper limit is loaded into the upper byte of the X resgister and the lower limit is loaded into the lower byte of the X register. Upon return, the B register contains the Boolean result (FALSE or TRUE) of the bounds check and the Z flag will be set according to the Boolean value in B. If B=FALSE (0) then Z is set. If B=TRUE (1) then Z value in B. If B=FALSE (0) then Z is set. If B=TRUE (1) then Z is reset.

If Byte_result and Byte value are defined to be the subrange type 0..15, a sample calling sequence for the Pascal assignment statement:

```
Byte_result:=Byte_value
```

compiled with $RANGE ON$ would appear as follows:

```
          LDAA  Byte_value
          LDX   #0F00H          ;15 in upper byte
                                ;0 in lower byte
          JSR   Zbbounds
          BNE   Range_OK
          JSR   RANGE_ERROR
Range_OK
          STAA  Byte_result
```

REGISTER ALLOCATION SUMMARY :: BYTE BOUNDS CHECK

```
Input:  A contains byte data
        X upper byte contains upper bound
        X lower byte contains lower bound

Output: B set to 0, Z flag set if value not in range
        B set to 1, Z flag reset if value in range

Registers:
    Modified:  B,CC
    Unchanged: A,X
```

Word Bounds Checking

Zwbounds Integer bounds checking
Zuwbounds Unsigned integer bounds checking

The bounds checking for signed and unsigned word variables use the same calling sequence and return the same results. If Int_result:= Int_value are defined to be the subrange type 0..511, a sample calling sequence for the Pascal assignment statement:

```
Int_result:= Int_value
```

compiled with $RANGE ON$, would appear as follows:

```
          LDX   Int_value
          JSR   Zwbounds
          FDB   0                ;lower bound
          FDB   511              ;upper bound
          BNE   Range_OK
          JSR   RANGE_ERROR
Range_OK
          STX   Int_result
```

REGISTER ALLOCATION SUMMARY ::WORD BOUNDS CHECK

```
Input:  X contains word data
        lower and upper bound constants follow call

Output: B set to 0, Z flag set if value not in range
        B set to 1, Z flag reset if value in range

Registers:
    Modified:  B,CC
    Unchanged: A,X
```

STRING OPERATIONS

Pascal/64000 supports variable length character strings as a special interpretation of packed arrays of type char. In particular the type STRING defined by the following Pascal source:

```
TYPE
    STRING = PACKED ARRAY [0..LENGTH] OF CHAR;
```

is interpreted to be a special string type. The length byte is located at array element 0 and contains the current run-time length of the string as shown in the following example (next page):

```
VAR ST1: PACKED ARRAY[0..n1] OF CHAR;
     (ST1[0] contains the run-time length of ST1)

    ST2: PACKED ARRAY[0..n2] OF CHAR;
     (ST2[0] contains the run-time length of ST2)

    CH: CHAR;
```

The number of bytes allocated for a particular string is determined at compile time and is fixed during execution time. Normal Pascal rules for type compatibility are relaxed for string types. In particular assignments and comparisons of strings are compatible even if the declared maximum string sizes are different. It is left to the run-time string handling routines to insure that strings are not assigned beyond the actual limits of a particular string.

String Routines.

STmove	String assignment
STequ	String equality test
STneq	String inequality test
STgeq	String greater than or equal test
STgtr	String greater than test
STleq	String less than or equal test
STles	String less than test
CHequ	String-char equality test
CHneq	String-char inequality test
CHgeq	String-char greater than or equal test
CHgtr	String-char greater than test
CHleq	String-char less than or equal test
CHles	String-char less than test

STmove. The routine STmove is used to copy a string from one location to another. The Pascal statement:

```
                 ST1:=ST2;
where:
          ST1 is a string variable or constant
          ST2 is a string variable or constant
```

will cause string, ST2, to be copied into ST1 if ST1 is large enough to contain the string ST2. Only the current length of string ST2 is used to check the validity of the assignment. The maximum possible length of string, ST1, is passed to the STmove routine. As long as the current length of ST2 is less than or equal to the maximum possible length of ST1 the assigment is performed.

Before calling the string move routine, the maximum size of ST1 is loaded into the B register. The address of the from string and the to string appear as a parameter list following the call. For the Pascal assignment statement:

```
ST1:= ST2
```

the calling sequence would appear as follows:

```
LDAB   #n1          ;maximum length of ST1
JSR    STmove
FDB    ST2          ;address of ST2
FDB    ST1          ;address of ST1
```

REGISTER ALLOCATION SUMMARY :: STMOVE

```
Input : B contains maximum length of ST1
        other parameters passed as parameter
        list following the JSR instruction.

Registers:
    Modified : A,B,CC,X
    Unchanged: none
```

String Comparisons. The STequ, STneq, STgeq, STgtr, STleq and STles routines are used to compare character strings. For expressions in the form:

```
ST1  <op>  ST2
```

where:

```
ST1  is a string variable or constant
ST2  is a string variable or constant
<op> is a comparison operator (=, <>, >=, >,
       <= or <)
```

String equality or inequality is determined by the following rules:

a. Two strings are equal if and only if their lengths are equal and they are equal character by character.

b. The inequality of two strings is determined by comparing the two strings character by character until either

1) one character is different - then the inequality is that of the differing character.

or

2) all characters are the same up to the length of the shorter of the two strings in which case the longer string is the larger.

For a Pascal expression of the form:

```
ST1 <op> ST2
```

the following code would be generated:

```
JSR  ST<op>       ;where <op> is equ, neg, geq,
                  ;qtr, leq, or les
FDB  ST2          ;right operand
FDB  ST1          ;left operand
```

REGISTER ALLOCATION SUMMARY :: ST COMPARES

```
Input : parameters passed as parameter list
        following the JSR instruction

Output: IF condition is TRUE then
             B=1 (TRUE), Z is reset
        IF condition is FALSE then
             B=0 (FALSE), Z is set

Registers:
    Modified : A,B,C,XX
    Unchanged: none
```

String-character Comparisons. The CHequ, CHneq, CHgeq, CHgtr, CHleq and CHles routines are used to compare strings with a single character variable. For expressions in the form:

```
             ST1 <op> CH
or
             CH <op> ST1
```

where:

```
ST1 is a string variable or constant
CH  is a char variable or constant
<op> is a comparison operator (=, <>, >=, >,
        <= or <)
```

The equality or inequality of a character to string comparison is determined logically by converting the character to a string of length 1 and following the string comparison rules defined above.

Before calling a record compare routine, the string address is loaded into the X register and the character is loaded into the B register. Upon return the B register contains the Boolean result and the X register still contains the address of the string variable.

REGISTER ALLOCATION SUMMARY :: STRING-CHARACTER

```
                              :: comparisons

Input : X contains address of the string
        B contains the character

Output: IF condition is TRUE then
             B=1 (TRUE), Z is reset

        IF condition is FALSE then
             B=0 (FALSE), Z is set

Registers:

    Modified : A,B,CC
    Unchanged: X
```

Utility Routines

SEXtend Signed byte to signed integer extension
ZBtoIadd Byte to integer addition
ZBtoIsub Byte to integer subtraction
ZuBtoIadd Unsigned byte to integer addition
ZuBtoIsub Unsigned byte to integer subtraction
TFR_DtoX Transfer contents of D to X
TFR_XtoD Transfer contents of X to D
ADD_BtoX Unsigned byte and unsigned integer addition
LEAX_B_X Signed byte and integer addition
LEAX_D_X Integer addition
JMPI_B_X Jump table indirect with index in B
JMPI_D_X Jump table indirect with index in D(A,B)
PUSHX Push register X onto stack
PshXSavD Push register X onto stack

SEXtend. This routine extends a signed Byte to become a signed Integer.

```
RESULT:= B1
```

where:

```
B1 is loaded in register B
```

The library routine is called after loading B1 into the B register. The integer RESULT is returned tin the D(A,B) register.

REGISTER ALLOCATION SUMMARY :: SEXTEND

```
Input : B contains byte parameter B1

Output: D(A,B) contains integer RESULT

Registers:
    Modified : A,CC
    Unchanged: B,X
```

ZBtoIadd, ZBtoIsub. ZBtoIadd and ZBtoIsub perform an operation (add or subtract) on two signed byte operands giving a signed integer result.

ZuBtoIadd, ZuBtoIsub. ZuBtoIadd and ZuBtoIsub perform an operation (add or subtract) on two unsigned byte operands giving an unsigned integer result.

```
RESULT:= A1<op>B1
```

where:

```
    B1 is loaded in Register B
and A1 is loaded in register A
```

The library routines are called after loading B1 and A1 into the proper registers. The RESULT is returned in the D(A,B) register.

REGISTER ALLOCATION SUMMARY :: ZBTOLADD AND ZBTOISUB
ZUBTOIADD AND ZUTOISUB

```
Input : B contains byte parameter B1
        A contains byte parameter A1

Output: D(A,B) contains integer RESULT

Registers:
    Modified : A,B,CC
    Unchanged: X
```

Register Transfer Routines

TFR_DtoX Transfer contents of D to X
TFR_XtoD Transfer contents of X to D

These routines transfer the contents of one Integer register into another.

```
RESULT: = I1
```

where:

```
I1 is loaded in register X or D(A,B).
```

The library routines are called after loading I1 into the register that will be transferred. The RESULT is returned in the remaining integer register.

REGISTER ALLOCATION SUMMARY :: TFR_DTOX

```
Input : D(A,B) contains integer parameter I1
Output: X contains integer RESULT

Registers
    Modified : X,CC
    Unchanged: A,B
```

REGISTER ALLOCATION SUMMARY :: TFR-XTOD

```
Input : X contains integer parameter I1
Output: D(A,B) contains integer RESULT

Registers:
    Modified : A,B,CC
    Unchanged: X
```

ADD_BtoX. This routine adds the Unsigned values of a byte and integer.

```
RESULT := B1 + X1
```

where:

```
    B1 is loaded in register B
and X1 is loaded in register X
```

The library routine is called after loading B1 and X1 into proper registers. The integer RESULT is returned in the X register.

REGISTER ALLOCATION SUMMARY :: ADD-BTOX

```
Input : B contains byte parameter B1
        X contains integer parameter X1

Output: X contains integer RESULT

Registers:
    Modified : X,CC
    Unchanged: A,B
```

LEAX_B_X, LEAX_D_X. These routines add the signed values of a specified byte or integer register to an integer in the X register leaving the result in the X register.

```
RESULT: = C1 + X1
```

where:

```
C1 is (a byte) loaded in register B
   or (an integer) loaded in register D(A,B)
```

The RESULT is returned in register X.

```
REGISTER ALLOCATION SUMMARY :: ADDITION TO X

 Input : B contains byte parameter C1
     or  D(AB)  contains integer parameter C1
 Output: X contains integer RESULT

 Registers:
     Modified : X,CC
     Unchanged: A,B
```

Indirect Table Jumps

JMPI_B_X Jump table indirect with index in B
JMPI_D_X Jump table indirect with index D(A,B)

These routines are used to perform table indirect jumps required by the CASE statement. When a CASE statement has many case values for branching, the compiler generates a table of program label addresses with one entry for each of the specified case labels. This table of addresses is then indexed by the actual case value expression in the case satement, and the program is transferred the proper case element code.

These routines are called with the base address of the case label table loaded into the X register and the actual case value expression loaded into the B register if it is a byte or into the D(A,B) register if it is an integer. Since the case label table is an array of 16-bit words containing addresses, the index value is multiplied by two then added to the value in X. This produces the address containing the 16-bit case label value where the program wishes to transfer. This label is then loaded into the X register and a JMP 0,X is executed resulting in the proper switch into the case statement.

Sample calling sequence:

```
        LDAB Case_value          ; CASE Case_value OF ...
        LDX  #CASE_TABLE         ; Load X with base address
                                 ; case table
        JMP  JMPI_B_X            ; Routine to perform indirect
                                 ; jump
        ...
CASE_TABLE    FDB CASE_0         ;address of 0: statement
              FDB CASE_1         ;address of 1: statement
              FDB CASE_2         ;address of 2: statement
              FDB CASE_3         ;address of 3: statement
              etc.
```

REGISTER ALLOCATION SUMMARY :: INDIRECT TABLE JUMPS

```
Input :X contains the address of the jump table
 and either
       B contains the byte case value for JMPI_B_X
 or
       D(A,B) contains the word case value for JMP_ID_X

Output: No register output but control transferred to
        indirect table value

Registers:
    Modified : A,B,CC,X
    Unchanged: None
```

PUSHX, PshXSavD. These routines push the contents of register X onto the stack without destroying the contents of the memory location used as an intermediate. PshXSavD does this operation without destroying registers A or B. They are created especially for use in the reentrant and debug libraries.

REGISTER ALLOCATION SUMMARY :: PUSHX AND PSHXSAVDf

```
Input : X register to be pushed onto the stack

Output: X register has been pushed onto the stack


Registers (for PUSHX)
    Modified : A,B,CC
    Unchanged: X


Registers (for PshXSavD)
    Modified: CC
    Unchanged; A,B,X
```

REAL NUMBER LIBRARIES (PRI)

The Pascal/64000 implementation of the IEEE floating point standard for the 6800 microprocessor is supported by two real libraries: RealLIB:R6800 (for Pascal data types: LONGREAL and REAL) and ShortReal:R6800 (for Pascal data type REAL only).

The user interface to these libraries is similar to that described for "User Defined Operators" (see Chapter 2). Each library routine name is a global symbol composed of the symbol REAL or LONGREAL followed by the operation mnemonic (such as REAL_ADD or LONGREAL_MUL). where op is the mnemonic for one of the supported operations. Since the compiler performs some automatic type conversions, there are some additional operations to convert between INTEGER, REAL and LONGREAL data types. Each of the library routines is defined by the equivalent Pascal procedure heading for its declaration.

Table 3-3 summarizes the floating point routines supported by the Pascal/64000 real number libraries. The text describes in more detail the external calling sequence used by the 6800 code generator to invoke these routines. For each routine the Pascal procedure or function heading is given which describes the logical interface for passing parameters and receiving results.

Table 3-3. Pascal Real Number Library Routines

Name	Purpose
REAL_ADD	Real addition
REAL_SUB	Real subtraction
REAL_MUL	Real multiplication
REAL_DIV	Real division
REAL_ABS	Real absolute value
REAL_NEG	Real negation
REAL_SQRT	Real square root
REAL_EXP	Real exponentiation(e to the X)
REAL_LN	Real natural logarithm
REAL_SIN	Real sine
REAL_COS	Real cosine
REAL_ATAN	Real arctangent
REAL_EQU	Real equality test
REAL_NEQ	Real inequality test
REAL_LES	Real less than test
REAL_GTR	Real greater than test
REAL_LEQ	Real less than or equal test
REAL_GEQ	Real greater than or equal test
REAL_FLOAT	Integer to real conversion
REAL_ROUND	Real to integer conversion with rounding
REAL_TRUNC	Real to integer conversion with truncation
LONGREAL_ADD	Longreal addition
LONGREAL_SUB	Longreal subtraction
LONGREAL_MUL	Longreal multiplication
LONGREAL_DIV	Longreal division
LONGREAL_ABS	Longreal absolute value
LONGREAL_NEG	Longreal negation
LONGREAL_SQRT	Longreal square root
LONGREAL_EXP	Longreal exponentiation(e to the X)
LONGREAL_LN	Longreal natural logarithm
LONGREAL_SIN	Longreal sine
LONGREAL_COS	Longreal cosine
LONGREAL_ATAN	Longreal arctangent
LONGREAL_EQU	Longreal equality test
LONGREAL_NEQ	Longreal inequality test
LONGREAL_LES	Longreal less than test
LONGREAL_GTR	Longreal greater than test
LONGREAL_LEQ	Longreal less than or equal test
LONGREAL_GEQ	Longreal greater than or equal test
LONGREAL_FLOAT	Integer to longreal conversion
LONGREAL_ROUND	Longreal to integer conversion with rounding
LONGREAL_TRUNC	Longreal to integer conversion with truncation
REAL_CONTRACT	Longreal to real conversion
REAL_EXTEND	Real to longreal conversion

Floating Point BINARY Operations (sec)

For binary floating point operations of the form:

```
RESULT:= LEFT <op> RIGHT
```

the equivalent Pascal procedure heading is in the form:

```
PROCEDURE REAL_<op> (VAR LEFT,RIGHT,RESULT:REAL)
```

or

```
PROCEDURE LONGREAL_<op> (VAR LEFT,RIGHT,RESULT:LONGREAL).
```

Binary operations supported in both floating point libraries (RealLIB:R6800 and ShortReal:R6800) are as follows:

```
PROCEDURE REAL_ADD (VAR LEFT,RIGHT,RESULT:REAL)
PROCEDURE REAL_SUB (VAR LEFT,RIGHT,RESULT:REAL)
PROCEDURE REAL_MUL (VAR LEFT,RIGHT,RESULT:REAL)
PROCEDURE REAL_DIV (VAR LEFT,RIGHT,RESULT:REAL)
```

Binary operations for LONGREAL (supported only in the library RealLIB:R6800) are as follows:

```
PROCEDURE LONGREAL_ADD (VAR LEFT,RIGHT,RESULT:LONGREAL)
PROCEDURE LONGREAL_SUB (VAR LEFT,RIGHT,RESULT:LONGREAL)
PROCEDURE LONGREAL_MUL (VAR LEFT,RIGHT,RESULT:LONGREAL)
PROCEDURE LONGREAL_DIV (VAR LEFT,RIGHT,RESULT:LONGREAL)
```

Floating Point UNARY Operations (sec)

For unary floating point operations of the form:

```
RESULT:=  <op> RIGHT
```

the equivalent Pascal procedure heading is in the form:

```
PROCEDURE REAL_<op> (VAR RIGHT,RESULT:REAL)
```

or

```
PROCEDURE LONGREAL_<op> (VAR RIGHT,RESULT:LONGREAL).
```

Unary operations supported in both floating point libraries (RealLIB:R6800 and ShortReal:R6800) are as follows:

```
PROCEDURE REAL_ABS  (VAR RIGHT,RESULT:REAL)
PROCEDURE REAL_NEG  (VAR RIGHT,RESULT:REAL)
PROCEDURE REAL_SQRT (VAR RIGHT,RESULT:REAL)
PROCEDURE REAL_EXP  (VAR RIGHT,RESULT:REAL)
PROCEDURE REAL_LN   (VAR RIGHT,RESULT:REAL)
PROCEDURE REAL_SIN  (VAR RIGHT,RESULT:REAL)
PROCEDURE REAL_COS  (VAR RIGHT,RESULT:REAL)
PROCEDURE REAL_ATAN (VAR RIGHT,RESULT:REAL)
```

Unary operations for LONGREAL (supported only in the library RealLIB:R6800) are as follows:

```
PROCEDURE LONGREAL_ABS  (VAR RIGHT,RESULT:LONGREAL)
PROCEDURE LONGREAL_NEG  (VAR RIGHT,RESULT:LONGREAL)
PROCEDURE LONGREAL_SQRT (VAR RIGHT,RESULT:LONGREAL)
PROCEDURE LONGREAL_EXP  (VAR RIGHT,RESULT:LONGREAL)
PROCEDURE LONGREAL_LN   (VAR RIGHT,RESULT:LONGREAL)
PROCEDURE LONGREAL_SIN  (VAR RIGHT,RESULT:LONGREAL)
PROCEDURE LONGREAL_COS  (VAR RIGHT,RESULT:LONGREAL)
PROCEDURE LONGREAL_ATAN (VAR RIGHT,RESULT:LONGREAL)
```

Floating Point Comparison Operations (sec)

For floating point comparison operations of the form:

```
BOOLEAN:= LEFT <op> RIGHT
```

the equivalent Pascal procedure heading is in the form:

```
FUNCTION REAL_<op> (VAR LEFT,RIGHT:REAL):BOOLEAN;
```

or

```
FUNCTION LONGREAL_<op> (VAR LEFT,RIGHT:LONGREAL):BOOLEAN;
```

Comparison operations supported in both floating point libraries (RealLIB:R6800 and ShortReal:R6800) are as follows:

```
FUNCTION REAL_EQU (VAR LEFT,RIGHT:REAL):BOOLEAN
FUNCTION REAL_NEQ (VAR LEFT,RIGHT:REAL):BOOLEAN
FUNCTION REAL_LES (VAR LEFT,RIGHT:REAL):BOOLEAN
FUNCTION REAL_GTR (VAR LEFT,RIGHT:REAL):BOOLEAN
FUNCTION REAL_LEQ (VAR LEFT,RIGHT:REAL):BOOLEAN
FUNCTION REAL_GEQ (VAR LEFT,RIGHT:REAL):BOOLEAN
```

Comparison operations for LONGREAL (supported only in the library RealLIB:R6800) are as follows:

```
FUNCTION LONGREAL_EQU (VAR LEFT,RIGHT:LONGREAL):BOOLEAN
FUNCTION LONGREAL_NEQ (VAR LEFT,RIGHT:LONGREAL):BOOLEAN
FUNCTION LONGREAL_LES (VAR LEFT,RIGHT:LONGREAL):BOOLEAN
FUNCTION LONGREAL_GTR (VAR LEFT,RIGHT:LONGREAL):BOOLEAN
FUNCTION LONGREAL_LEQ (VAR LEFT,RIGHT:LONGREAL):BOOLEAN
FUNCTION LONGREAL_GEQ (VAR LEFT,RIGHT:LONGREAL):BOOLEAN
```

Floating Point Conversion Operations (sec)

For floating point conversion operations of the form:

```
RESULT:=  <op> RIGHT
```

the equivalent Pascal procedure heading is in the form:

```
PROCEDURE REAL_<op> (VAR RIGHT:RIGHTtype;VAR RESULT:RESULTtype)
```

or

```
PROCEDURE LONGREAL_<op> (VAR RIGHT:RIGHTtype;VAR RESULT:RESULTtype)
```

Conversion operations supported in both floating point libraries (RealLIB:R6800 and ShortReal:R6800) are as follows:

```
PROCEDURE REAL_FLOAT  (VAR RIGHT:INTEGER;VAR RESULT:REAL)
PROCEDURE REAL_ROUND  (VAR RIGHT:REAL;VAR RESULT:INTEGER)
PROCEDURE REAL_TRUNC  (VAR RIGHT:REAL;VAR RESULT:INTEGER)
```

Conversion operations for LONGREAL (supported only in the library RealLIB:R6800) are as follows:

```
PROCEDURE LONGREAL_FLOAT (VAR RIGHT:INTEGER;VAR RESULT:LONGREAL)
PROCEDURE LONGREAL_ROUND (VAR RIGHT:LONGREAL;VAR RESULT:INTEGER)
PROCEDURE LONGREAL_TRUNC (VAR RIGHT:LONGREAL;VAR RESULT:INTEGER)
PROCEDURE REAL_CONTRACT  (VAR RIGHT:LONGREAL;VAR RESULT:REAL)
PROCEDURE REAL_EXTEND    (VAR RIGHT:REAL;VAR RESULT:LONGREAL)
```

Floating Point Error Detection (sec)

The floating point libraries have two error conditions which, when detected, cause the execution of one of two global routines. These routine names are OVERFLOW and INVALID. OVERFLOW is called when an operation would produce an invalid number. INVALID is called when an invalid floating point number is passed as a parameter to one of the floating point routines.

The user may replace either of these routines with an error recovery routine of his own. In particular, defining either of these routines as a simple return from subroutine instruction (RTS) will cause the program to continue with an invalid number returned as a result.

If the user does not supply his own version of these routines, the libraries will supply one which will cause illegal opcode 1EH for oveflow errors and illegal opcode 1FH for invalid operation errors.

If either of the illegal opcodes is detected by the emulator, the user can get information describing the error by entering the emulation command:

```
display memory REALerror
```

which will produce a memory display indicating the error condition.

If no error has occurred, the display will appear as follows:

```
MEMORY
 Adr   -------Data(hex)------      -(ASCII)-
---------------------------------------------------
A2EA  4E 6F 20 65  72 72 6F 72     No error
A2F2  20 20 20 20  20 20 20 20
A2FA  20 20 20 20  20 20 20 20
A302  20 20 20 20  20 20 20 20
A30A  20 20 20 20  20 20 20 20
A312  20 20 20 20  20 20 20 20
A31A  20 20 20 20  20 20 20 20
A322  20 20 20 20  20 20 20 20
A32A  20 20 20 20  20 20 20 20
A332  20 20 20 20  20 20 20 20
A33A  20 20 20 20  20 20 20 20
A342  20 20 20 20  20 20 20 20
A34A  20 20 20 20  20 20 20 20
A352  20 20 20 20  20 20 20 20
A35A  20 20 20 20  20 20 20 20
A362  20 20 20 20  20 20 20 20
```

If an OVERFLOW error has occurred, the display will appear as follows:

```
MEMORY
 Adr   -------Data(hex)------      -(ASCII)-
---------------------------------------------------
A2EA  52 65 61 6C  20 20 20 20     Real
A2F2  65 72 72 6F  72 20 20 20     error
A2FA  4F 56 45 52  46 4C 4F 57     OVER FLOW
A302  61 74 20 20  20 20 20 20     at
A30A  20 20 20 20  37 41 33 38         7A38
A312  20 20 20 20  20 20 20 20
A31A  4C 4F 4E 47  52 45 41 4C     LONG REAL
A322  5F 41 44 44  20 20 20 20     _MUL
A32A  63 61 6C 6C  65 64 20 20     called
A332  66 72 6F 6D  20 20 20 20     from
A33A  20 20 20 20  33 32 42 36         32B6
A342  20 20 20 20  20 20 20 20
A34A  20 20 20 20  20 20 20 20
A352  20 20 20 20  20 20 20 20
A35A  20 20 20 20  20 20 20 20
A362  20 20 20 20  20 20 20 20
```

If an INVALID operation error has occurred, the display will appear as follows:

```
MEMORY
 Adr  -------Data(hex)------      -(ASCII)-
-------------------------------------------------
A2EA  52 65 61 6C  20 20 20 20    Real
A2F2  65 72 72 6F  72 20 20 20    error
A2FA  49 4E 56 41  4C 49 44 20    INVALID
A302  61 74 20 20  20 20 20 20    at
A30A  20 20 20 20  37 33 45 46        73EF
A312  20 20 20 20  20 20 20 20
A31A  4C 4F 4E 47  52 45 41 4C    LONG REAL
A322  5F 41 44 44  20 20 20 20    _ADD
A32A  63 61 6C 6C  65 64 20 20    called
A332  66 72 6F 6D  20 20 20 20    from
A33A  20 20 20 20  32 41 31 37        2A17
A342  20 20 20 20  20 20 20 20
A34A  20 20 20 20  20 20 20 20
A352  20 20 20 20  20 20 20 20
A35A  20 20 20 20  20 20 20 20
A362  20 20 20 20  20 20 20 20
```

With this display, the user can determine which type of error has been detected, which floating point library was called, and where the floating point library detected the error.

NOTES

Chapter 4

REAL NUMBER LIBRARY

INTRODUCTION

THE PASCAL/64000 implementation of the IEEE floating point standard for the 6800 microprocessor is supported by real library RealLib:R6800 (for Pascal data types: LONGREAL and REAL).

The user interface to these libraries is similar to that described for "User Defined Operators" (see Chapter 2). Each library routine name is a global symbol composed of the symbol REAL or LONGREAL followed by the operation mnemonic (such as REAL_ADD or LONGREAL_MUL) for one of the supported operations. Since the compiler performs some automatic type conversions, there are some additional operations to convert between INTERGER, REAL and LONGREAL data types.

Table 4-1 summarizes the floating point routines suported by the Pascal/64000 real number libraries. The text describes in more detail the external calling sequence used by the 6800 code generator to invoke these routines. For each routine the Pascal procedure or function heading is given which describes the logical interface for passing parameters and receiving results.

Table 4-1. Pascal Real Number Library Routines

NAME	PURPOSE
REAL_ADD	Real addition
REAL_SUB	Real subtraction
REAL_MUL	Real multiplication
REAL_DIV	Real division
REAL_ABS	Real absolute value
REAL_NEG	Real negation
REAL_SQRT	Real square root
REAL_EXP	Real exponentiation (e to the X)
REAL_LN	Real natural logarithm
REAL_SIN	Real sine
REAL_COS	Real cosine
REAL_ATAN	Real arctangent
REAL_EQU	Real equlity test
REAL_NEG	Real inequality test
REAL_LES	Real less than test
REAL_GTR	Real greater than test
REAL_LEQ	Real less than or equal test
REAL_GEQ	Real greater than or equal test
REAL_FLOAT	Integer to real conversion
REAL_ROUND	Real to integer conversion with rounding
REAL_TRUNC	Real to integer conversion with truncation
LONGREAL_ADD	Longreal addition
LONGREAL_SUB	Longreal subtraction
LONGREAL_MUL	Longreal multiplication
LONGREAL_DIV	Longreal division
LONGREAL_ABS	Longreal absolute value
LONGREAL_NEG	Longreal negation
LONGREAL_SQRT	Longreal square root
LONGREAL_EXP	Longreal exponentiation (e to the X)
LONGREAL_LN	Longreal natural logarithm
LONGREAL_SIN	Longreal sine
LONGREAL_COS	Longreal cosine
LONGREAL_ATAN	Longreal arctangent
LONGREAL_EQU	Longreal equality test
LONGREAL_NEQ	Longreal inequality test
LONGREAL_LES	Longreal less than test
LONGREAL_GTR	Longreal greater than test
LONGREAL_LEQ	Longreal less than or equal test
LONGREAL_GEQ	Longreal greater than or equal test
LONGREAL_FLOAT	Interger to longreal conversion
LONGREAL_ROUND	Longreal to interger conversion with rounding
LONGREAL_TRUNC	Longreal to integer conversion with truncation
LONG_CONTRACT	Longreal to real conversion
REAL_EXTENDED	Real to longreal conversion

Floating Point BINARY Operations

For binary floating point operations of the form:

```
RESULTS:= LEFT <op> RIGHT
```

the equivalent Pascal procedure heading is in the form:

```
PROCEDURE REAL_<op> (VAR LEFT,RIGHT,RESULTS:REAL)
```

or

```
PROCEDURE LONGREAL_<op> (VAR LEFT,RIGHT,RESULT:LONGREAL
```

Binary operations supported in RealLIB:R6800 are as follows:

```
PROCEDURE REAL_ADD (VAR LEFT,RIGHT,RESULTS:REAL)
PROCEDURE REAL_SUB (VAR LEFT,RIGHT,RESULT:REAL)
PROCEDURE REAL_MUL (VAR LEFT,RIGHT,RESULT:REAL)
PROCEDURE REAL_DIV (VAR LEFT,RIGHT,RESULT:REAL)
PROCEDURE LONGREAL_ADD (VAR LEFT,RIGHT,RESULT:LONGREAL)
PROCEDURE LONGREAL_SUB (VAR LEFT,RIGHT,RESULT:LONGREAL)
PROCEDURE LONGREAL_MUL (VAR LEFT,RIGHT,RESULT:LONGREAL)
PROCEDURE LONGREAL_DIV (VAR LEFT,RIGHT,RESULTS:LONGREAL
```

Floating Point UNARY Operations

For unary floating point operations of the form:

```
RESULT:= <op> RIGHT
```

the equvalent Pascal procedure heading is in the form:

```
PROCEDURE REAL_<op> (VAR RIGHT,RESULT:REAL)
```

or

```
PROCEDURE LONGREAL_<op> (VAR RIGHT,RESULT:LONGREAL
```

Unary operations supported in RealLIB:R6800 are as follows:

```
PROCEDURE REAL_ABS   (VAR RIGHT,RESULT:REAL)
PROCEDURE REAL_NEG   (VAR RIGHT,RESULT:REAL)
PROCEDURE REAL_SQRT  (VAR RIGHT,RESULT:REAL)
PROCEDURE REAL_EXP   (VAR RIGHT,RESULT:REAL)
PROCEDURE REAL_LN    (VAR RIGHT,RESULT:REAL)
PROCEDURE REAL_SIN   (VAR RIGHT,RESULT:REAL)
PROCEDURE REAL_COS   (VAR RIGHT,RESULT:REAL)
PROCEDURE REAL_ATAN      (VAR RIGHT,RESULT:LONGREAL)
PROCEDURE LONGREAL_ABS   (VAR RIGHT,RESULT:LONGREAL)
PROCEDURE LONGREAL_NEG   (VAR RIGHT,RESULT:LONGREAL)
PROCEDURE LONGREAL_SQRT  (VAR RIGHT,RESULT:LONGREAL)
PROCEDURE LONGREAL_EXP   (VAR RIGHT,RESULT:LONGREAL)
PROCEDURE LONGREAL_LN    (VAR RIGHT,RESULT:LONGREAL)
PROCEDURE LONGREAL_SIN   (VAR RIGHT,RESULT:LONGREAL)
PROCEDURE LONGREAL COS   (VAR RIGHT,RESULT:LONGREAL)
PROCEDURE LONGREAL_ATAN  (VAR RIGHT,RESULT:LONGREAL)
```

Floating Point Comparison Operations

For floating point comparison operations of the form:

```
BOOLEAN:= LEFT <op> RIGHT
```

the equivalent Pascal procedure heading is in the form:

```
FUNCTION REAL_<op> (VAR LEFT,RIGHT:REAL):BOOLEAN;
```

or

```
FUNCTION LONGREAL_<op> (VAR LEFT,RIGHT:LONGREAL):BOOLEAN;
```

Comparison operations supported in RealLIB:R6800 are as follows:

```
FUNCTION REAL_EQU   (VAR LEFT,RIGHT:REAL):BOOLEAN
FUNCTION REAL_NEG   (VAR LEFT,RIGHT:REAL):BOOLEAN
FUNCTION REAL_LES   (VAR LEFT,RIGHT:REAL):BOOLEAN
FUNCTION REAL_GTR   (VAR LEFT,RIGHT:REAL):BOOLEAN
FUNCTION REAL_LEQ   (VAR LEFT,RIGHT:REAL):BOOLEAN
FUNCTION REAL_GEQ   (VAR LEFT,RIGHT:REAL):BOOLEAN
FUNCTION LONGREAL_EQU   (VAR LEFT,RIGHT:LONGREAL):BOOLEAN
FUNCTION LONGREAL_NEQ   (VAR LEFT,RIGHT:LONGREAL):BOOLEAN
FUNCTION LONGREAL_LES   (VAR LEFT,RIGHT:LONGREAL):BOOLEAN
FUNCTION LONGREAL_GTR   (VAR LEFT,RIGHT:LONGREAL):BOOLEAN
FUNCTION LONGREAL_LEQ   (VAR LEFT,RIGHT:LONGREAL):BOOLEAN
FUNCTION LONGREAL_GEQ   (VAR LEFT,RIGHT:LONGREAL):BOOLEAN
```

Floating Point Conversion Operations

For floating point conversion operations of the form:

```
RESULT:= <op> RIGHT
```

the equivalent Pascal procedure heading is in the form:

```
PROCEDURE REAL_<op> (VAR RIGHT:RIGHTtype;
                     VAR RESULT:RESULTtype)
```

or

```
PROCEDURE LONGREAL_<op> (VAR RIGHT:RIGHTtype;
                         VAR RESULT:RESULTtype)
```

Conversion operations supported in RealLIB:R6800 are as follows:

```
PROCEDURE REAL_FLOAT   (VAR RIGHT:INTEGER;VAR RESULT:REAL)
PROCEDURE REAL_ROUND   (VAR RIGHT:REAL;VAR RESULT:INTEGER)
PROCEDURE REAL_TRUNC   (VAR RIGHT:REAL;VAR RESULT:INTEGER)
PROCEDURE LONGREAL_FLOAT   (VAR RIGHT:INTEGER)
                       VAR RESULTS:LONGREAL)
PROCEDURE LONGREAL_ROUND   (VAR RIGHT:LONGREAL;
                       VAR RESULT:INTEGER)
PROCEDURE LONGREAL_TRUNC   (VAR RIGHT:LONGREAL;
                       VAR RESULT:INTERGER)
PROCEDURE REAL_CONTRACT   (VAR RIGHT:LONGREAL;
                      VAR RESULT:REAL)

PROCEDURE REAL_RXTENDED   (VAR RIGHT:REAL;
                       VAR RESULT:LONGREAL)
```

Floating Point Error Detection

The floating point libraries have two error conditions which, when detected, cause the execution of one of two global routines. These routine names are REAL_OVERFLOW and INVALID. REAL_OVERFLOW is called when an operation would produce an invalid number. INVALID is called when an invalid floating point number is passed as a parameter to one of the floating point routines.

Users may replace either if these routines with an error recovery routine of their own. In particular, defining either of these routines as a simple return from subroutine instruction (RTS) will cause the program to continue with an invalid number returned as a result.

The error routines provided by the library will write a status message to the buffer, ERROR_MESSAGE, indicating the type of error and where it occured. They will then return and continue normal operation.

The user can get information describing the error by entering the emulation command:

display memory ERROR_MESSAGE blocked word

which will produce a memory display indicating the error condition.

If no error has occured, the display will appear as follows:

```
Memory     :words  :blocked
address         data              :hex                    :ascii
-----------------------------------------------------------------
   8086-95      4E6F 2065 7272 6F72 2020 2020 2020 2020   No error
   8096-A5      2020 2020 2020 2020 2020 2020 2020 2020
   80A6-B5      2020 2020 2020 2020 2020 2020 2020 2020
   80B6-C5      2020 2020 2020 2020 2020 2020 2020 2020
   80C6-D5      2020 2020 2020 2020 2020 2020 2020 2020
   80D6-E5      2020 2020 2020 2020 2020 2020 2020 2020
   80E6-F5      2020 2020 2020 2020 2020 2020 2020 2020
   80F6-05      2020 2020 2020 2020 2020 2020 2020 2020
   8106-15      2020 2020 2020 2020 2020 2020 2020 2020
   8116-25      2020 2020 2020 2020 2020 2020 2020 2020
   8126-35      2020 2020 2020 2020 2020 2020 2020 2020
   8136-45      2020 2020 2020 2020 2020 2020 2020 2020
   8146-55      2020 2020 2020 2020 2020 2020 2020 2020
   8156-65      2020 2020 2020 2020 2020 2020 2020 2020
   8166-75      2020 2020 2020 2020 2020 2020 2020 2020
   8176-85      2020 2020 2020 2020 2020 2020 2020 202E
```

If in OVERFLOW error has occured, the display will appeart as follows:

```
Memory     :words  :blocked
address        data              :hex                   :ascii
-----------------------------------------------------------------
   8086-95      5265 616C 2020 2020 6572 726F 7220 2020    Real error
   8096-A5      4F56 4552 464C 4F57 2020 6174 2020 2020    OVERFLOW at
   80A6-B5      3137 4431 4820 2020 2020 2020 2020 2020    17D1H
   80B6-C5      4C4F 4E47 5245 414C 5F45 5850 2020 2020    LONGREAL _EXP
   80C6-D5      526F 7574 696E 6520 6361 6C6C 6564 2020    Routine called
   80D6-E5      6279 2020 2020 2020 7573 6572 2020 2020    by user
   80E6-F5      6672 6F6D 2020 2020 6164 6472 6573 7320    from address
   80F6-05      3031 3934 4820 2020 2020 2020 2020 2020    0194H
   8106-15      2020 2020 2020 2020 2020 2020 2020 2020
   8116-25      2020 2020 2020 2020 2020 2020 2020 2020
   8126-35      2020 2020 2020 2020 2020 2020 2020 2020
   8136-45      2020 2020 2020 2020 2020 2020 2020 2020
   8146-55      2020 2020 2020 2020 2020 2020 2020 2020
   8156-65      2020 2020 2020 2020 2020 2020 2020 2020
   8166-75      2020 2020 2020 2020 2020 2020 2020 2020
   8176-85      2020 2020 2020 2020 2020 2020 2020 202E
```

If an INVALID operation error has occurred, the display will appear as follows:

```
Memory     :words  :blocked
address        data              :hex                   :ascii
-----------------------------------------------------------------
   8086-95      5265 616C 2020 2020 6572 726F 7220 2020    Real error
   8096-A5      494E 5641 4C49 4420 2020 6174 2020 2020    INVALID at
   80A6-B5      3137 3130 4820 2020 2020 2020 2020 2020    1710H
   80B6-C5      5245 414C 5F41 4444 2020 2020 2020 2020    REAL_ADD
   80C6-D5      526F 7574 696E 6520 6361 6C6C 6564 2020    Routine called
   80D6-E5      6279 2020 2020 2020 7573 6572 2020 2020    by user
   80E6-F5      6672 6F6D 2020 2020 6164 6472 6573 7320    from address
   80F6-05      3030 3731 4820 2020 2020 2020 2020 2020    0071H
   8106-15      2020 2020 2020 2020 2020 2020 2020 2020
   8116-25      2020 2020 2020 2020 2020 2020 2020 2020
   8126-35      2020 2020 2020 2020 2020 2020 2020 2020
   8136-45      2020 2020 2020 2020 2020 2020 2020 2020
   8146-55      2020 2020 2020 2020 2020 2020 2020 2020
   8156-65      2020 2020 2020 2020 2020 2020 2020 2020
   8166-75      2020 2020 2020 2020 2020 2020 2020 2020
   8176-85      2020 2020 2020 2020 2020 2020 2020 202E
```

With this display, the user can determine which type of error has been detected, which floating point library was called, and where the floating point library detected the error.

Floating Point Number Internal Format

The floating point numbers use the IEEE standard for the two packed formats (single precision (REAL) and double precision (LONGREAL)). The two formats are described in the following paragraphs.

SINGLE PRECISION FORMAT. The single precision floating point number used for the type REAL is a 32-bit binary value packed as follows:

s	e	f
0	8	31

where:

s is the sign bit.

e is the exponent.

f is the 23-bit fraction.

The value (v) of a single precision floating point number (x) can be computed as follows:

(a) If $e=255$ and $f<>0$, then v=not a number.
(b) If $e=255$ and $f=0$, then $v=(-1)^s\infty$.
(c) If $0<e<255$, then $v=(-1)^s 2^{e-127}(1.f)$.
(d) If $e=0$ and $f<>0$, then $v=(-1)^s 2^{-126}(0.f)$.
(e) If $e=0$ and $f=0$, then $v=(-1)^s 0$, (zero).

The range of REAL numbers is approximately +/- 10^{38}.

DOUBLE PRECISION FORMAT. A double precision floating point number used for the type LONGREAL is a 64-bit binary value packed as follows:

s	e	f
0	11	63

where:

s is the sign bit.

e is the exponent.

f is the 52-bit fraction.

The value (v) of a double precision floating point number (x) can be computed as follows:

(a) If e=2047 and f<>0, then v=not a number.
(b) If e=2047 and f=0, then $v=(-1)^{S}\infty$.
(c) If 0<e<2047, then $v=(-1)^{S}2^{e-1023}(1.f)$.
(d) If e=0 and f<>0, then $v=(-)^{S}2^{-1022}(0.f)$.
(e) If e=0 and f=0, then $v=(-1)^{S}0$,(zero).

The range of LONGREAL numbers is approximately +/- 10^{308}.

NOTES

Chapter 5

PASCAL FILE I/O LIBRARIES

INTRODUCTION

The Pascal I/O features are provided by the Pascal I/O support library: PIOLIB:F6800. The simulated I/O features of the emulation subsystem are provided by the support library: SIMLIB:F6800.

Chapter 6 of the Pascal/64000 Reference Manual contains a comlete machine independent description of the routines in these libraries.

Both libraries are compiled with the options $SEPARATE ON,RECURSIVE OFF$. They will load subroutines in the PROG relocatable area and use the DATA relocatable area for local data and a message buffer for error detection.

File Error Detection

The Pascal I/O libraries support error detection as described in the Pascal/64000 Reference Manual.

If the file operations are compiled with the option,$1OCHECK OFF$, each file will set a global variable to indicate the result code. The user should follow each file operation with a call to the function IORESULT, defined by the Pascal I/O library, to obtain the result code of the most recent I/O operation. It is the user's responsibility to ensure the correct processing of any I/O error so that the program continues properly.

If the operations are compiled with the option, $IOCHECK ON$ (defaullt case), any error detected by the I/O libraries will cause an error message to be written into 6800 memory at the location FILE_ERROR. The program will wait in the library module FMON_6800 in a loop executing the illegal opcode, 1FM. Since this mode of operation cannot assume the correct response to any arbitrary I/O error, it effectively stops the operation of the program so no further errors will occur.

When running programs compiled with option $IOCHECK ON$, in the emulation subsystem, it is recommended that the user answer the emulation configuration questin "Stop processor on illegal opcodes?" in the affirmative. If a file is then detected, the emulation status message will display:

"ERROR: 6800/2--Reset in background illegal opcode 1FH at XXXXH"

The user can then see the file error number and the location where the file routine was called by entering the command.

display memory FILE_ERROR blocked word

If no error has occured, the display will appear as follows:

```
Memory     :words   :blocked

address          data             :hex        :ascii
-----------------------------------------------------------------
67C8-D7         4E6F 2065 7272 6F72 2020 2020 2020 2020  No error
67D8-E7         2020 2020 2020 2020 2020 2020 2020 2020
67E8-F7         2020 2020 2020 2020 2020 2020 2020 2020
67F8-07         2020 2020 2020 2020 2020 2020 2020 2020
6808-17         2020 2020 2020 2020 2020 2020 2020 2020
6818-27         2020 2020 2020 2020 2020 2020 2020 2020
6828-37         2020 2020 2020 2020 2020 2020 2020 2020
6838-46         2020 2020 2020 2020 2020 2020 2020 2020
6848-57         2020 2020 2020 2020 2020 2020 2020 2020
6858-67         2020 2020 2020 2020 2020 2020 2020 2020
6868-77         2020 2020 2020 2020 2020 2020 2020 2020
6878-87         2020 2020 2020 2020 2020 2020 2020 2020
6888-97         2020 2020 2020 2020 2020 2020 2020 2020
6898-A7         2020 2020 2020 2020 2020 2020 2020 2020
68A8-B7         2020 2020 2020 2020 2020 2020 2020 2020
68B8-C7         2020 2020 2020 2020 2020 2020 2020 202E
```

If a file has been detected the display will appear as follows:

```
Memory     :words   :blocked

address          data             :hex        :ascii
-----------------------------------------------------------------
67C8-D7         2020 492F 4F20 2020 2065 7272 6F72 2020  I/O    error
67D8-E7         2020 2020 2020 2020 2020 2020 2020 3031
67E8-F7         4669 6C65 2049 4F20 726F 7574 696E 6520  file IO  routine
67F8-07         2063 616C 6C65 6420 2020 2062 7920 2020   called   by
6808-17         2020 7573 6572 2020 2020 6672 6F6d 2020   user   from
6818-27         2061 6464 7265 7373 2002 3132 3536 4820  address  1256H
6828-37         2020 2020 2020 2020 2020 2020 2020 2020
6838-47         2020 2020 2020 2020 2020 2020 2020 2020
6848-57         2020 2020 2020 2020 2020 2020 2020 2020
6858-67         2020 2020 2020 2020 2020 2020 2020 2020
6868-77         2020 2020 2020 2020 2020 2020 2020 2020
6878-87         2020 2020 2020 2020 2020 2020 2020 2020
6888-97         2020 2020 2020 2020 2020 2020 2020 2020
6898-A7         2020 2020 2020 2020 2020 2020 2020 2020
68A8-87         2020 2020 2020 2020 2020 2020 2020 2020
68B8-C7         2020 2020 2020 2020 2020 2020 2020 2020
```

With this display, the user can determine the number of the error which has occured. The description of the function, IORESULT, in Chapter 6 of the Pascal/64000 reference manual contains the explanation for each error number.

If the error number is 1 and the simulated I/O library, SIMLIB:F6800, is being used, then the global variable, errno, will contain the simulated I/O error number. These errors are summarized in the Pascal/64000 Reference Manual, Chapter 6, in the section describing error reporting for the Simulated I/O library.

NOTES

Appendix A

RUN-TIME ERROR DESCRIPTIONS

This appendix contains descriptions of run-time errors that may occur.

ERROR UTILITIES

NAME	PURPOSE
Derrors:D6800	Debugging library error handler
Zerrors:L6800	Normal library error handler
Zerrors:S6800	Static library error handler

Derrors

Derrors contains the run-time routines which store user information at the time an error occurs during debugging.

1) The following errors may occur in the indicated library routines:

OPCODE	ERROR	ROUTINES
00H	Case_error	User programs
02H	Div_by_Zero	Zbytediv, Zubytediv, Zintdiv, Zuintdiv
03H	Heap_error	INITHEAP, NEW, DISPOSE, MARK, RELEASE

12H	Overflow	Dbytemul, Dubytemul, Zintmul, Zuintmul Zbytediv, Zintdiv Zbyteadd, Zubyteadd, Zintadd, Zuintadd Zbytesub, Zubytesub, Zintsub, Zuintsub Zbyteneg, Zintneg Zbyteabs, Zintabs
13H	Range_error	In user routines after calling bound check routines: Zbbounds, Zubbounds, Zwbounds, Zuwbounds
14H	Set_conversion_error	Zbtoset8, Zwtoset8 Zbtoset16, Zwtoset16
15H	String_error	STmove
18H	Underflow	Dbytemul, Zintmul, Zbyteadd, Zintadd, Zbytesub, Zubytesub, Zintsub, Zuintsub

2) When an error is detected, a jump to Derrors is generated and valid register information is saved. The labels for the stored information are described below:

LABEL	DESCRIPTION
Z_ZCALLER_H Z_ZCALLER_L	Contain the high byte (CALLER_H) and the low byte (CALLER_L) of the address of the statement which called the routine where the actual error occurred. This information will usually be the address of the next executable statement following the library routine call.
Z_ZCC_FLAGS Z_REG_A Z_REG_B Z_REG_X_H Z_REG_X_L	Contain the contents of the registers at the time the error occurred. Only registers with information relevant to the error are saved - the indicated contents of the other registers is garbage.

NOTE

The CC register which is displayed is that which was present when the error occurred in the Debug Library routine. The CC register which was present when the Debug routine was called is not retrievable.

---

3) The following is a description of the errors that may occur and the information that is accessible when they do occur.

ERROR MSG.	DESCRIPTION	INFORMATION
AVAILABLE		
Z_ERR_CASE	Jump to error occurs when the test variable of CASE statement is out of range and no OTHERWISE label exists.	Z_ZCALLER_H Z_ZCALLER_L Z_REG_A Z_REG_B
Z_ERR_DIV_BY_0	Jump to error occurs if division by zero is attempted by byte or integer division routines.	Z_ZCALLER_H Z_ZCALLER_L Z_ZCC_FLAGS Z_REG_X_H Z_REG_X_L
Z_ERR_HEAP	Jump to error occurs when some misuse of the dynamic allocation routines NEW, DISPOSE,MARK, or RELEASE takes place.	Z_ZCALLER_H Z_ZCALLER_L Z_ZCC_FLAGS ?
Z_ERR_OVERFLOW	Jump to error occurs when results of multiplication, addition, subtraction, negation, or the absolute value is too positive (i.e. INTEGERS: result > 32767 BYTES: result > 127)	Z_ZCALLER_H Z_ZCALLER_L Z_ZCC_FLAGS Z_REG_A Z_REG_B Z_REG_X_H Z_REG_X_L

ERROR MSG. AVAILABLE	DESCRIPTION	INFORMATION
Z_ERR_RANGE	Jump to error occurs if a range declaration has been violated (i.e.: a variable does not fall within its assigned range)	Z_ZCALLER_H Z_ZCALLER_L Z_ZCC_FLAGS Z_REG_A Z_REG_B
Z_ERR_SET_CONV	Jump to error occurs if operand is not legal ordinal value for a set of the base type.	Z_ZCALLER_H Z_ZCALLER_L Z_ZCC_FLAGS Z_REG_A Z_REG_B Z_REG_X_H Z_REG_X_L
Z_ERR_STRING	Jump to error occurs on a string assignment, when the run-time size of the string being assigned is larger than that of which is it is being assigned to.	Z_ZCALLER_H Z_ZCALLER_L
Z_ERR_UNDERFLOW	Jump to error occurs if results of addition, subtraction, or multiplication were too negative (i.e. INTEGERS result < -32768 BYTES result < -128)	Z_ZCALLER_H Z_ZCALLER_L Z_ZCC_FLAGS Z_REG_A Z_REG_B Z_REG_X_H Z_REG_X_L
Z_END_PROGRAM	Jump to this label when the program completes execution of the main body code.	

4) The illegal opcodes associated with the various errors are as follows:

OPCODE	ERROR
00H	Case_error
02H	Div_by_0
03H	Heap_error
12H	Overflow
13H	Range_error
14H	Set_conversion_error
15H	String_size_assignment_error
18H	Underflow

Zerrors

Zerrors contains the run-time routines which store user information at the time an error occurs during execution in the non-debug libraries (LIB6800:L6800 and SLIB6800:S6800).

The following errors may occur in the indicated library routines:

OPCODE	ERROR	ROUTINES
00H	Case_error	User programs
03H	Heap_error	INITHEAP, NEW, DISPOSE, MARK, RELEASE
13H	Range_error	User programs with range checks.
15H	String_error	STmove

When an error is detected, a jump to Zerrors is generated and valid register information is saved. The stored information, the routines and the illegal opcodes for this errors are as described in Derrors.

NOTES

INDEX

a

b

c

d

INDEX (Cont'd)

e

f

g

h

i

INDEX (Cont'd)

l

m

n

o

p

INDEX (Cont'd)

p

r

s

t

INDEX (Cont'd)

u

v

w

z

6800

NOTES

SALES & SUPPORT OFFICES
Arranged alphabetically by country

Product Line Sales/Support Key

Key Product Line
A Analytical
CM Components
C Computer Systems Sales only
CH Computer Systems Hardware Sales and Services
CS Computer Systems Software Sales and Services
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.

HP distributors are printed in italics.

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

AFRICA AND MIDDLE EAST
Hewlett-Packard S.A.
Mediterranean and Middle East Operations
Atrina Centre
32 Kifissias Ave.
Paradissos-Amarousion, **ATHENS**
Greece
Tel: 682 88 11
Telex: 21-6588 HPAT GR
Cable: HEWPACKSA Athens

NORTH/CENTRAL AFRICA
Hewlett-Packard S.A.
7, Rue du Bois-du-Lan
CH-1217 **MEYRIN** 2, Switzerland
Tel: (022) 83 12 12
Telex: 27835 hpse
Cable: HEWPACKSA Geneve

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: 610-492-4246

EASTERN EUROPE
Hewlett-Packard Ges.m.b.h.
Lieblgasse 1
P.O.Box 72
A-1222 **VIENNA,** Austria
Tel: (222) 2365110
Telex: 1 3 4425 HEPA A

NORTHERN EUROPE
Hewlett-Packard S.A.
Uilenstede 475
P.O.Box 999
NL-1180 AZ **AMSTELVEEN**
The Netherlands
Tel: 20 437771

SOUTH EAST EUROPE
Hewlett-Packard S.A.
World Trade Center
110 Avenue Louis Carol
1215 Cointrin, **GENEVA,** Switzerland
Tel: (022) 98 96 51
Telex: 27225 hpse.

EASTERN USA
Hewlett-Packard Co.
4 Choke Cherry Road
ROCKVILLE, MD 20850
Tel: (301) 258-2000

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
P.O. Box 105005
ATLANTA, GA 30348
Tel: (404) 955-1500

WESTERN USA
Hewlett-Packard Co.
3939 Lankershim Blvd.
P.O. Box 3919
LOS ANGELES, CA 91604
Tel: (213) 506-3700

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

ANGOLA
Telectra
Empresa TAEcnica de Equipamentos
R. Barbosa Rodrigues, 41-I DT.
Caixa Postal 6487
LUANDA
Tel: 35515,35516
E,P

ARGENTINA
Hewlett-Packard Argentina S.A.
Avenida Santa Fe 2035
Martinez 1640 **BUENOS AIRES**
Tel: 798-5735, 792-1293
Cable: HEWPACKARG
A,E,CH,CS,P

AUSTRALIA

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

Brisbane, Queensland Office
Hewlett-Packard Australia Ltd.
10 Payne Road
THE GAP, Queensland 4061
Tel: 30-4133
Telex: 42133
Cable: HEWPARD Brisbane
A,CH,CS,CM,E,M,P

Canberra, Australia Capital Territory Office
Hewlett-Packard Australia Ltd.
121 Wollongong Street
FYSHWICK, A.C.T. 2609
Tel: 80 4244
Telex: 62650
Cable: HEWPARD Canberra
C,CH,CM,CS,E,P

Melbourne, Victoria Office
Hewlett-Packard Australia Ltd.
31-41 Joseph Street
BLACKBURN, Victoria 3130
Tel: 895-2895
Telex: 31-024
Cable: HEWPARD Melbourne
A,CH,CM,CS,E,M,P

Perth, Western Australia Office
Hewlett-Packard Australia Ltd.
261 Stirling Highway
CLAREMONT, W.A. 6010
Tel: 383-2188
Telex: 93859
Cable: HEWPARD Perth
A,CH,CM,CS,E,M,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: 888-4444
Telex: 21561
Cable: HEWPARD Sydney
A,CH,CM,CS,E,M,P

AUSTRIA
Hewlett-Packard Ges.m.b.h.
Grottenhofstrasse 94
A-8052 **GRAZ**
Tel: (0316) 291 5 66
Telex: 32375
CH,E

Hewlett-Packard Ges.m.b.h.
Lieblgasse 1
P.O. Box 72
A-1222 **VIENNA**
Tel: (0222) 23 65 11-0
Telex: 134425 HEPA A
A,CH,CM,CS,E,M,P

BAHRAIN
Green Salon
P.O. Box 557
Manama
BAHRAIN
Tel: 255503-255950
Telex: 84419
P

Wael Pharmacy
P.O. Box 648
BAHRAIN
Tel: 256123
Telex: 8550 WAEL BN
E,M

BELGIUM
Hewlett-Packard Belgium S.A./N.V.
Blvd de la Woluwe, 100
Woluwedal
B-1200 **BRUSSELS**
Tel: (02) 762-32-00
Telex: 23-494 paloben bru
A,CH,CM,CS,E,M,P

BERMUDA
Applied Computer Technologies
Atlantic House Building
Par-La-Ville Road
Hamilton 5
Tel: 295-1616
P

BRAZIL
Hewlett-Packard do Brasil
I.e.C. Ltda.
Alameda Rio Negro, 750
Alphaville
06400 **BARUERI SP**
Tel: (011) 421.1311
Telex: (011) 33872 HPBR-BR
Cable: HEWPACK Sao Paulo
A,CH,CM,CS,E,M,P

SALES & SUPPORT OFFICES
Arranged alphabetically by country

BRAZIL (Cont'd)
Hewlett-Packard do Brasil
I.e.C. Ltda.
Avenida Epitacio Pessoa, 4664
22471 **RIO DE JANEIRO**-RJ
Tel: (021) 286.0237
Telex: 021-21905 HPBR-BR
Cable: HEWPACK Rio de Janeiro
A,CH,CM,E,M,P*

Convex/Van Den
Rua Jose Bonifacio
458 Todos Os Santos
CEP 20771
***RIO DE JANEIRO**, RJ*
Tel: 249-7121, 591-4946
Telex: 33487

ANAMED I.C.E.I. Ltda.
Rua Bage, 103
*04012 **SAO PAULO***
Tel: (011) 570-5726
Telex: 021-21905 HPBR-BR
M

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

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

British Columbia
Hewlett-Packard (Canada) Ltd.
10691 Shellbridge Way
RICHMOND,
British Columbia V6X 2W7
Tel: (604) 270-2277
Telex: 610-922-5059
A,CH,CM,CS,E*,M,P*

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

Manitoba
Hewlett-Packard (Canada) Ltd.
1825 Inkster Blvd.
WINNIPEG, Manitoba R3H 0Y1
Tel: (204) 786-6701
A,CH,CM,E,M,P*

New Brunswick
Hewlett-Packard (Canada) Ltd.
37 Shediac Road
MONCTON, New Brunswick E1A 2R6
Tel: (506) 855-2841
CH,CS

Nova Scotia
Hewlett-Packard (Canada) Ltd.
Suite 111
900 Windmill Road
DARTMOUTH, Nova Scotia B2Y 3Z6
Tel: (902) 469-7820
CH,CM,CS,E*,M,P*

Ontario
Hewlett-Packard (Canada) Ltd.
3325 N. Service Rd., Unit 6
BURLINGTON, Ontario P3A 2A3
Tel: (416) 335-8644
CS,M*

Hewlett-Packard (Canada) Ltd.
496 Days Road
KINGSTON, Ontario K7M 5R4
Tel: (613) 384-2088
CH,CS

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

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

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

Hewlett-Packard (Canada) Ltd.
1855 Lasalle Boulevard
SUDBURY, Ontario, P3A 2A3
Tel: (705) 560-5450
CH

Hewlett-Packard (Canada) Ltd.
220 Yorkland Blvd. Unit #11
WILLOWDALE, Ontario M2J 1R5
Tel: (416) 499-9333
CH

Quebec
Hewlett-Packard (Canada) Ltd.
17500 South Service Road
Trans-Canada Highway
KIRKLAND, Quebec H9J 2M5
Tel: (514) 697-4232
A,CH,CM,CS,E,M,P*

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

Hewlett-Packard (Canada) Ltd.
#7-130 Robin Crescent
SASKATOON, Saskatchewan S7L 6M7
Tel: (306) 242-3702
CH,CS

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

Jorge Calcagni y Cia. Ltda.
Av. Italia 634 Santiago
Casilla 16475
***SANTIAGO** 9*
Tel: 222-0222
Telex: 440283 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
CH,CS,P

CHINA, People's Republic of
China Hewlett-Packard Co., Ltd.
6th Floor, Sun Hung Kai Centre
30 Harbour Road
HONG KONG
Tel: 5-8323211
Telex: 36678 HEWPA HX
A,C,CH,CS,E,M,P

China Hewlett-Packard Rep. Office
P.O. Box 418
1A Lane 2, Luchang St.
Beiwei Rd., Xuanwu District
BEIJING
Tel: 33-1947, 33-7426
Telex: 22601 CTSHP CN
Cable: 1920
A,CH,CM,CS,E,P

COLOMBIA
InstrumentaciAOn
H. A. Langebaek & Kier S.A.
Carrera 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

Procesa, S.A.
CRA 7 No. 24-89 Piso 25
Torre Colpatria
Apartado Aereo No. 49667
***BOGOTA** D.E.*
Tel: 2344925, 2344958, 2344742
Telex: 43127 COVER CO
C,P

Compumundo
Avenida 15 # 107-80
***BOGOTA** D.E.*
Tel: 214-4458
Telex: 45466 MARICO
P

COSTA RICA
Cientifica Costarricense S.A.
Avenida 2, Calle 5
San Pedro de Montes de Oca
Apartado 10159
SAN JOSÉ
Tel: 24-38-20, 24-08-19
Telex: 2367 GALGUR CR
CM,E,M

CYPRUS
Telerexa Ltd.
P.O. Box 4809
14C Stassinos Avenue
NICOSIA
Tel: 62698
Telex: 2894 LEVIDO CY
E,M,P

DENMARK
Hewlett-Packard A/S
Datavej 52
DK-3460 **BIRKEROD**
Tel: (02) 81-66-40
Telex: 37409 hpas dk
A,CH,CM,CS,E,M,P

Hewlett-Packard A/S
Rolighedsvej 32
DK-8240 **RISSKOV,** Aarhus
Tel: (06) 17-60-00
Telex: 37409 hpas dk
CH,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: 450-975, 243-052
Telex: 2548 CYEDE ED
CM,E,P

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

QUITO
Tel: 2-238-951
Telex: 2298 ECUAME ED

EGYPT
Egyptian International
Office for Foreign Trade
P.O. Box 2558
42 El-Zahraa Street
*Dokki, **CAIRO,***
Tel: 712230
Telex: 93337 EGPOR UN
Cable: EGYPOR
P,A

EGYPT (Cont'd)

INFORMATIC FOR SYSTEMS
22 Talaat Harb Street
CAIRO,
Tel: 759006
Telex: 93697 SAFLM UN
CS

International Engineering Associates
24 Hussein Hegazi Street
Kasr-el-Aini
CAIRO,
Tel: 23829, 21641
Telex: 93830 IEA UN
Cable: INTEGASSO
E

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 1216
SAN SALVADOR
Tel: 26-6858, 26-6868
Telex: 20539 IPESASAL
A,CH,CM,CS,E,P

FINLAND

Hewlett-Packard Oy
Piispankalliontie 17
02200 **ESPOO**
Tel: 00358-0-88721
Telex: 121563 HEWPA SF
CH,CM,SS,P

Hewlett-Packard Oy
(Olarinluoma 7)
PL 24
02101 **ESPOO** 10
Tel: (90) 4521022
A,E,M

Hewlett-Packard Oy
Aatoksenkatv 10-C
SF-40720-72 **JYVASKYLA**
Tel: (941) 216318
CH

Hewlett-Packard Oy
Kainvuntie 1-C
SF-90140-14 **OULU**
Tel: (981) 338785
CH

FRANCE

Hewlett-Packard France
Z.I. Mercure B
Rue Berthelot
F-13763 Les Milles Cedex
AIX-EN-PROVENCE
Tel: (42) 59-41-02
Telex: 410770F
A,CH,E,M,P*

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

Hewlett-Packard France
Boite Postale 503
F-25026 **BESANCON**
28 rue de la Republique
F-25000 **BESANCON**
Tel: (81) 83-16-22
Telex: 361157
CH,M

Hewlett-Packard France
13, Place Napoleon III
F-29000 **BREST**
Tel: (98) 03-38-35

Hewlett-Packard France
Chemin des Mouilles
Boite Postale 162
F-69130 **ECULLY** Cedex (Lyon)
Tel: (78) 833-81-25
Telex: 310617F
A,CH,CS,E,M

Hewlett-Packard France
Parc d'Activite du Bois Briard
Ave. du Lac
F-91040 **EVRY** Cedex
Tel: 6 077-8383
Telex: 692315F
E

Hewlett-Packard France
5, Avenue Raymond Chanas
F-38320 **EYBENS** (Grenoble)
Tel: (76) 62-67-98
Telex: 980124 HP GRENOB EYBE
CH

Hewlett-Packard France
Centre d'Affaire Paris-Nord
Bâtiment Ampère 5 étage
Rue de la Commune de Paris
Boite Postale 300
F-93153 **LE BLANC MESNIL**
Tel: (1) 865-44-52
Telex: 211032F
CH,CS,E,M

Hewlett-Packard France
Parc d'Activités Cadera
Quartier Jean Mermoz
Avenue du Président JF Kennedy
F-33700 **MERIGNAC** (Bordeaux)
Tel: (56) 34-00-84
Telex: 550105F
CH,E,M

Hewlett-Packard France
Immueble "Les 3 B"
Nouveau Chemin de la Garde
ZAC de Bois Briand
F-44085 **NANTES** Cedex
Tel: (40) 50-32-22
Telex: 711085F
CH**

Hewlett-Packard France
125, rue du Faubourg Bannier
F-45000 **ORLEANS**
Tel: (38) 68 01 63

Hewlett-Packard France
Zone Industrielle de Courtaboeuf
Avenue des Tropiques
F-91947 Les Ulis Cedex **ORSAY**
Tel: (6) 907-78-25
Telex: 600048F
A,CH,CM,CS,E,M,P

Hewlett-Packard France
Paris Porte-Maillot
15, Avenue de L'Amiral Bruix
F-75782 **PARIS** CEDEX 16
Tel: (1) 502-12-20
Telex: 613663F
CH,M,P

Hewlett-Packard France
124, Boulevard Tourasse
F-64000 **PAU**
Tel: (59) 80 38 02

Hewlett-Packard France
2 AllAEe de la Bourgonnette
F-35100 **RENNES**
Tel: (99) 51-42-44
Telex: 740912F
CH,CM,E,M,P*

Hewlett-Packard France
98 Avenue de Bretagne
F-76100 **ROUEN**
Tel: (35) 63-57-66
Telex: 770035F
CH**,CS

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

Hewlett-Packard France
Le PAEripole
20, Chemin du Pigeonnier de la CAEpiGEere
F-31083 **TOULOUSE** Cedex
Tel: (61) 40-11-12
Telex: 531639F
A,CH,CS,E,P*

Hewlett-Packard France
9, rue Baudin
F-26000 **VALENCE**
Tel: (75) 42 76 16

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

Hewlett-Packard France
Immeuble PEricentre
F-59658 **VILLENEUVE D'ASCQ** Cedex
Tel: (20) 91-41-25
Telex: 160124F
CH,E,M,P*

GERMAN FEDERAL REPUBLIC

Hewlett-Packard GmbH
Geschäftsstelle
Keithstrasse 2-4
D-1000 **BERLIN** 30
Tel: (030) 24-90-86
Telex: 018 3405 hpbln d
A,CH,E,M,P

Hewlett-Packard GmbH
Geschäftsstelle
Herrenberger Strasse 130
D-7030 **BÖBLINGEN**
Tel: (7031) 14-0
Telex: 07265739
A,CH,CM,CS,E,M,P

Hewlett-Packard GmbH
Geschäftsstelle
Emanuel-Leutze-Strasse 1
D-4000 **DUSSELDORF**
Tel: (0211) 5971-1
Telex: 085/86 533 hpdd d
A,CH,CS,E,M,P

Hewlett-Packard GmbH
Geschäftsstelle
Schleefstr. 28a
D-4600 **DORTMUND**-Aplerbeck
Tel: (0231) 45001

Hewlett-Packard GmbH
Vertriebszentrale Frankfurt
Berner Strasse 117
Postfach 560 140
D-6000 **FRANKFURT** 56
Tel: (0611) 50-04-1
Telex: 04 13249 hpffm d
A,CH,CM,CS,E,M,P

Hewlett-Packard GmbH
Geschäftsstelle
Aussenstelle Bad Homburg
Louisenstrasse 115
D-6380 **BAD HOMBURG**
Tel: (06172) 109-0

Hewlett-Packard GmbH
Geschäftsstelle
Kapstadtring 5
D-2000 **HAMBURG** 60
Tel: (040) 63804-1
Telex: 021 63 032 hphh d
A,CH,CS,E,M,P

Hewlett-Packard GmbH
Geschäftsstelle
Heidering 37-39
D-3000 **HANNOVER** 61
Tel: (0511) 5706-0
Telex: 092 3259
A,CH,CM,E,M,P

Hewlett-Packard GmbH
Geschäftsstelle
Rosslauer Weg 2-4
D-6800 **MANNHEIM**
Tel: (0621) 70050
Telex: 0462105
A,C,E

Hewlett-Packard GmbH
Geschäftsstelle
Messerschmittstrasse 7
D-7910 **NEU ULM**
Tel: 0731-70241
Telex: 0712816 HP ULM-D
A,C,E*

Hewlett-Packard GmbH
Geschäftsstelle
Ehhericherstr. 13
D-8500 **NÜRNBERG** 10
Tel: (0911) 5205-0
Telex: 0623 860
CH,CM,E,M,P

Hewlett-Packard GmbH
Geschäftsstelle
Eschenstrasse 5
D-8028 **TAUFKIRCHEN**
Tel: (089) 6117-1
Telex: 0524985
A,CH,CM,E,M,P

GREAT BRITAIN

See United Kingdom

SALES & SUPPORT OFFICES
Arranged alphabetically by country

GREECE
Hewlett-Packard A.E.
178, Kifissias Avenue
6th Floor
Halandri-**ATHENS**
Greece
Tel: 6471673, 6471543, 6472971
A,CH,CM**,CS**,E,M,P

Kostas Karaynnis S.A.
8 Omirou Street
***ATHENS** 133*
Tel: 32 30 303, 32 37 371
Telex: 215962 RKAR GR
A,CH,CM,CS,E,M,P

PLAISIO S.A.
Eliopoulos Brohers Ltd.
11854
ATHENS
Tel: 34-51-911
Telex: 216286
P

GUATEMALA
IPESA
Avenida Reforma 3-48, Zona 9
GUATEMALA CITY
Tel: 316627, 314786
Telex: 4192 TELTRO GU
A,CH,CM,CS,E,M,P

HONG KONG
Hewlett-Packard Hong Kong, Ltd.
G.P.O. Box 795
5th Floor, Sun Hung Kai Centre
30 Harbour Road
HONG KONG
Tel: 5-8323211
Telex: 66678 HEWPA HX
Cable: HEWPACK HONG KONG
E,CH,CS,P

CET Ltd.
10th Floor, Hua Asia
Bldg. Gloucester
64-66 Gloulester 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
Elding Trading Company Inc.
Hafnarnvoli-Tryggvagotu
P.O. Box 895
*IS-***REYKJAVIK**
Tel: 1-58-20, 1-63-03
M

INDIA
Computer products are sold through Blue Star Ltd.All computer repairs and maintenance service is done through Computer Maintenance Corp.

Blue Star Ltd.
Sabri Complex II Floor
24 Residency Rd.
BANGALORE *560 025*
Tel: 55660
Telex: 0845-430
Cable: BLUESTAR
A,CH,CM,CS*,E*

Blue Star Ltd.
Band Box House
Prabhadevi
BOMBAY *400 025*
Tel: 422-3101
Telex: 011-3751
Cable: BLUESTAR
A,M

Blue Star Ltd.
Sahas
414/2 Vir Savarkar Marg
Prabhadevi
BOMBAY *400 025*
Tel: 422-6155
Telex: 011-71193
Cable: FROSTBLUE
A,CH,CM,CS*,E,M*

Blue Star Ltd.
Kalyan, 19 Vishwas Colony
Alkapuri, **BORODA,** *390 005*
Tel: 65235
Cable: BLUE STAR
A

Blue Star Ltd.
7 Hare Street
***CALCUTTA** 700 001*
Tel: 12-01-31
Telex: 021-7655
Cable: BLUESTAR
A,M

Blue Star Ltd.
133 Kodambakkam High Road
MADRAS *600 034*
Tel: 82057
Telex: 041-379
Cable: BLUESTAR
A,M

Blue Star Ltd.
Bhandari House, 7th/8th Floors
91 Nehru Place
NEW DELHI *110 024*
Tel: 682547
Telex: 031-2463
Cable: BLUESTAR
A,CH,CM,CS*,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
Telex: 0155-459
Cable: BLUEFROST
A,E

Blue Star Ltd.
T.C. 7/603 Poornima
Maruthankuzhi
TRIVANDRUM *695 013*
Tel: 65799
Telex: 0884-259
Cable: BLUESTAR
E

Computer Maintenance Corporation Ltd.
115, Sarojini Devi Road
SECUNDERABAD *500 003*
Tel: 310-184, 345-774
Telex: 031-2960
*CH***

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., 17th Floor
Jl. Medan Merdeka Selatan 17
JAKARTA-PUSAT
Tel: 21-344-181
Telex: BERSAL IA
A,CS,E,M

BERCA Indonesia P.T.
P.O. Box 174/SBY.
Jl. Kutei No. 11
SURABAYA
Tel: 68172
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
CH,CS

IRELAND
Hewlett-Packard Ireland Ltd.
82/83 Lower Leeson Street
DUBLIN 2
Tel: 0001 608800
Telex: 30439
A,CH,CM,CS,E,M,P

Cardiac Services Ltd.
Kilmore Road
Artane
DUBLIN *5*
Tel: (01) 351820
Telex: 30439
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
CH,CM,CS,E,P

ITALY
Hewlett-Packard Italiana S.p.A
Traversa 99C
Via Giulio Petroni, 19
I-70124 **BARI**
Tel: (080) 41-07-44
M,CH

Hewlett-Packard Italiana S.p.A.
Via Martin Luther King, 38/III
I-40132 **BOLOGNA**
Tel: (051) 402394
Telex: 511630
CH,CS,E,M

Hewlett-Packard Italiana S.p.A.
Via Principe Nicola 43G/C
I-95126 **CATANIA**
Tel: (095) 37-10-87
Telex: 970291
CH

Hewlett-Packard Italiana S.p.A.
Via G. Di Vittorio 9
I-20063 **CERNUSCO SUL NAVIGLIO**
(Milano)
Tel: (02) 923691
Telex: 334632
A,CH,CM,CS,E,M,P

Hewlett-Packard Italiana S.p.A.
Via C. Colombo 49
I-20090 **TREZZANO SUL NAVIGLIO**
(Milano)
Tel: (02) 4459041
Telex: 322116
CH,CS

Hewlett-Packard Italiana S.p.A.
Via Nuova San Rocco a
Capodimonte, 62/A
I-80131 **NAPOLI**
Tel: (081) 7413544
Telex: 710698
A**,CH,CS,E,M

Hewlett-Packard Italiana S.p.A.
Viale G. Modugno 33
I-16156 **GENOVA PEGLI**
Tel: (010) 68-37-07
Telex: 215238
E,C

Hewlett-Packard Italiana S.p.A.
Via Pelizzo 15
I-35128 **PADOVA**
Tel: (049) 664888
Telex: 430315
A,CH,CS,E,M

Hewlett-Packard Italiana S.p.A.
Viale C. Pavese 340
I-00144 **ROMA EUR**
Tel: (06) 54831
Telex: 610514
A,CH,CS,E,M,P*

ITALY (Cont'd)

Hewlett-Packard Italiana S.p.A.
Via di Casellina 57/C
I-50018 **SCANDICCI-FIRENZE**
Tel: (055) 753863
CH,E,M

Hewlett-Packard Italiana S.p.A.
Corso Svizzera, 185
I-10144 **TORINO**
Tel: (011) 74 4044
Telex: 221079
A*,CS,CH,E

JAPAN

Yokogawa-Hewlett-Packard Ltd.
152-1, Onna
ATSUGI, Kanagawa, 243
Tel: (0462) 28-0451
CM,C*,E

Yokogawa-Helwett-Packard Ltd.
Meiji-Seimei Bldg. 6F
3-1 Hon Chiba-Cho
CHIBA, 280
Tel: 472 25 7701
E,CH,CS

Yokogawa-Hewlett-Packard Ltd.
Yasuda-Seimei Hiroshima Bldg.
6-11, Hon-dori, Naka-ku
HIROSHIMA, 730
Tel: 82-241-0611

Yokogawa-Hewlett-Packard Ltd.
Towa Building
2-3, Kaigan-dori, 2 Chome 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
CH,CM,E

Yokogawa-Hewlett-Packard Ltd.
Asahi Shinbun Daiichi Seimei Bldg.
4-7, Hanabata-cho
KUMAMOTO, 860
Tel: (0963) 54-7311
CH,E

Yokogawa-Hewlett-Packard Ltd.
Shin-Kyoto Center Bldg.
614, Higashi-Shiokoji-cho
Karasuma-Nishiiru
Shiokoji-dori, Shimogyo-ku
KYOTO, 600
Tel: 075-343-0921
CH,E

Yokogawa-Hewlett-Packard Ltd.
Mito Mitsui Bldg
4-73, Sanno-maru, 1 Chome
MITO, Ibaraki 310
Tel: (0292) 25-7470
CH,CM,E

Yokogawa-Hewlett-Packard Ltd.
Meiji-Seimei Kokubun Bldg. 7-8
Kokubun, 1 Chome, Sendai
MIYAGI, 980
Tel: (0222) 25-1011
Telex:
C,E

Yokogawa-Hewlett-Packard Ltd.
Sumitomo Seimei 14-9 Bldg.
Meieki-Minami, 2 Chome
Nakamura-ku
NAGOYA, 450
Tel: (052) 571-5171
CH,CM,CS,E,M

Yokogawa-Hewlett-Packard Ltd.
Chuo Bldg.,
4-20 Nishinakajima, 5 Chome
Yodogawa-ku
OSAKA, 532
Tel: (06) 304-6021
Telex: YHPOSA 523-3624
A,CH,CM,CS,E,M,P*

Yokogawa-Hewlett-Packard Ltd.
27-15, Yabe, 1 Chome
SAGAMIHARA Kanagawa, 229
Tel: 0427 59-1311

Yokogawa-Hewlett-Packard Ltd.
Daiichi Seimei Bldg.
7-1, Nishi Shinjuku, 2 Chome
Shinjuku-ku,**TOKYO** 160
Tel: 03-348-4611
CH,E

Yokogawa-Hewlett-Packard Ltd.
29-21 Takaido-Higashi, 3 Chome
Suginami-ku **TOKYO** 168
Tel: (03) 331-6111
Telex: 232-2024 YHPTOK
A,CH,CM,CS,E,M,P*

Yokogawa-Hewlett-Packard Ltd.
Daiichi Asano Building
2-8, Odori, 5 Chome
UTSUNOMIYA, Tochigi 320
Tel: (0286) 25-7155
CH,CS,E

Yokogawa-Hewlett-Packard Ltd.
Yasuda Seimei Nishiguchi Bldg.
30-4 Tsuruya-cho, 3 Chome
YOKOHAMA 221
Tel: (045) 312-1252
CH,CM,E

JORDAN

Scientific and Medical Supplies Co.
P.O. Box 1387
AMMAN
Tel: 24907, 39907
Telex: 21456 SABCO JO
CH,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.
12 Fl. Kinam Bldg.
San 75-31, Yeoksam-Dong
Kangnam-Ku
Yeongdong P.O. Box 72
SEOUL
Tel: 555-7555, 555-5447
Telex: K27364 SAMSAN
A,CH,CM,CS,E,M,P

KUWAIT

Al-Khaldiya Trading & Contracting
P.O. Box 830
SAFAT
Tel: 424910, 411726
Telex: 22481 AREEG KT
Cable: VISCOUNT
E,M,A

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 75
SAFAT
Tel: 2462640/1
Telex: 30336 TOWELL KT
C

LEBANON

Computer Information Systems
P.O. Box 11-6274
BEIRUT
Tel: 89 40 73
Telex: 42309
C,E,M,P

LUXEMBOURG

Hewlett-Packard Belgium S.A./N.V.
Blvd de la Woluwe, 100
Woluwedal
B-1200 **BRUSSELS**
Tel: (02) 762-32-00
Telex: 23-494 paloben bru
A,CH,CM,CS,E,M,P

MALAYSIA

Hewlett-Packard Sales (Malaysia)
Sdn. Bhd.
1st Floor, Bangunan British
American
Jalan Semantan, Damansara Heights
KUALA LUMPUR 23-03
Tel: 943022
Telex: MA31011
A,CH,E,M,P*

Protel Engineering
P.O.Box 1917
Lot 6624, Section 64
23/4 Pending Road
Kuching, **SARAWAK**
Tel: 36299
Telex: MA 70904 PROMAL
Cable: PROTELENG
A,E,M

MALTA

Philip Toledo Ltd.
Notabile Rd.
MRIEHEL
Tel: 447 47, 455 66
Telex: Media MW 649
E,P,M

MEXICO

Hewlett-Packard Mexicana, S.A.
de C.V.
Av. Periferico Sur No. 6501
Tepepan, Xochimilco
16020 **MEXICO D.F.**
Tel: 6-76-46-00
Telex: 17-74-507 HEWPACK MEX
A,CH,CS,E,M,P

Hewlett-Packard Mexicana, S.A.
de C.V.
Czda. del Valle
409 Ote. 1 °Piso
Colonia del Valle
Municipio de Garza Garciá
66220 **MONTERREY,** Nuevo LeAOn
Tel: 78 42 41
Telex: 038 410
CH

Equipos Cientificos de Occidente, S.A.
Av. Lazaro Cardenas 3540
GUADALAJARA
Tel: 21-66-91
Telex: 0684186 ECOME
A

Infograficas y Sistemas del Noreste, S.A.
Rio Orinoco #171 Oriente
Despacho 2001
Colonia Del Valle
MONTERREY
Tel: 782499, 781259A
A

MOROCCO

Dolbeau
81 rue Karatchi
CASABLANCA
Tel: 3041-82, 3068-38
Telex: 23051, 22822
E

Gerep
2 rue d'Agadir
Boite Postale 156
CASABLANCA
Tel: 272093, 272095
Telex: 23 739
P

Sema-Maroc
Rue Lapebie
CASABLANCA
Tel: 26.09.80
CH,CS,P

NETHERLANDS

Hewlett-Packard Nederland B.V.
Van Heuven Goedhartlaan 121
NL 1181KK **AMSTELVEEN**
P.O. Box 667
NL1180 AR **AMSTELVEEN**
Tel: (020) 47-20-21
Telex: 13 216 HEPA NL
A,CH,CM,CS,E,M,P

Hewlett-Packard Nederland B.V.
Bongerd 2
NL 2906VK **CAPELLE A/D IJSSEL**
P.O. Box 41
NL 2900AA **CAPELLE A/D IJSSEL**
Tel: (10) 51-64-44
Telex: 21261 HEPAC NL
A,CH,CS,E

Hewlett-Packard Nederland B.V.
Pastoor Petersstraat 134-136
NL 5612 LV **EINDHOVEN**
P.O. Box 2342
NL 5600 CH **EINDHOVEN**
Tel: (040) 326911
Telex: 51484 hepae nl
A,CH**,E,M

SALES & SUPPORT OFFICES
Arranged alphabetically by country

NEW ZEALAND
Hewlett-Packard (N.Z.) Ltd.
5 Owens Road
P.O. Box 26-189
Epsom, **AUCKLAND**
Tel: 687-159
Cable: HEWPAK Auckland
CH,CS,CM,E,P*

Hewlett-Packard (N.Z.) Ltd.
4-12 Cruickshank Street
Kilbirnie, **WELLINGTON 3**
P.O. Box 9443
Courtenay Place, **WELLINGTON 3**
Tel: 877-199
Cable: HEWPACK Wellington
CH,CS,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

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: 16621 hpnas n
CH,CS,E,M

Hewlett-Packard Norge A/S
UCOsterndalen 16-18
P.O. Box 34
N-1345 **OCUSTERÅS**
Tel: 0047/2/17 11 80
Telex: 16621 hpnas n
A,CH,CM,CS,E,M,P

OMAN
Khimjil Ramdas
P.O. Box 19
MUSCAT
Tel: 722225, 745601
Telex: 3289 BROKER MB MUSCAT
P

Suhail & Saud Bahwan
P.O.Box 169
MUSCAT
Tel: 734 201-3
Telex: 3274 BAHWAN MB
E

Imtac LLC
P.O. Box 8676
MUTRAH
Tel: 601695
Telex: 5741 Tawoos On
A,C,M

PAKISTAN
Mushko & Company Ltd.
House No. 16, Street No. 16
Sector F-6/3
ISLAMABAD
Tel: 824545
Cable: FEMUS Islamabad
*A,E,M,P**

Mushko & Company Ltd.
Oosman Chambers
Abdullah Haroon Road
KARACHI *0302*
Tel: 524131, 524132
Telex: 2894 MUSKO PK
Cable: COOPERATOR Karachi
*A,E,M,P**

PANAMA
ElectrOnico Balboa, S.A.
Calle Samuel Lewis, Ed. Alfa
Apartado 4929
PANAMA *5*
Tel: 63-6613, 63-6748
Telex: 3483 ELECTRON PG
A,CM,E,M,P

PERU
Cía Electro Médica S.A.
Los Flamencos 145, San Isidro
Casilla 1030
LIMA *1*
Tel: 41-4325, 41-3703
Telex: Pub. Booth 25306
CM,E,M,P

SAMS
Rio De La Plata 305
SAN ISIDRO
Tel: 419928
Telex: 394 20450 PELIBERTAD
P

PHILIPPINES
The Online Advanced Systems Corporation
Rico House, Amorsolo Cor. Herrera Street
Legaspi Village, Makati
P.O. Box 1510
Metro **MANILA**
Tel: 815-38-11 (up to 16)
Telex: 63274 Online PN
A,CH,CS,E,M

Electronic Specialists and Proponents Inc.
690-B Epifanio de los Santos Avenue
Cubao, **QUEZON CITY**
P.O. Box 2649 Manila
Tel: 98-96-81, 98-96-82, 98-96-83
Telex: 40018, 42000 ITT GLOBE MACKAY BOOTH
P

PORTUGAL
Mundinter
Intercambio Mundial de ComAErcio S.A.R.L.
P.O. Box 2761
Av. Antonio Augusto de Aguiar 138
*P-***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 81/2/3
Telex: 13316 SABASA
P

Telectra-Empresa Técnica de Equipmentos Eléctricos S.A.R.L.
Rua Rodrigo da Fonseca 103
P.O. Box 2531
*P-***LISBON** *1*
Tel: (19) 68-60-72
Telex: 12598
CM,E

Rarcentro Ltda
R. Costa Cabral 575
4200 **PORTO**
Tel: 499174/495173
Telex: 26054
CH,CS

PUERTO RICO
Hewlett-Packard Puerto Rico
101 MuANoz Rivera Av
Esu. Calle Ochoa
HATO REY, Puerto Rico 00918
Tel: (809) 754-7800
A,CH,CS,CM,M,E,P

QATAR
Computer Arabia
P.O. Box 2750
DOHA
Tel: 883555
Telex: 4806 CHPARB
P

Nasser Trading & Contracting
P.O.Box 1563
DOHA
Tel: 422170
Telex: 4439 NASSER DH
M

SAUDI ARABIA
Modern Electronic Establishment
Hewlett-Packard Division
P.O. Box 281
Thuobah
AL-KHOBAR
Tel: 895-1760, 895-1764
Telex: 671 106 HPMEEK SJ
Cable: ELECTA AL-KHOBAR
CH,CS,E,M

Modern Electronic Establishment
Hewlett-Packard Division
P.O. Box 1228
Redec Plaza, 6th Floor
JEDDAH
Tel: 644 38 48
Telex: 4027 12 FARNAS SJ
Cable: ELECTA JEDDAH
A,CH,CS,CM,E,M,P

Modern Electronic Establishment
Hewlett-Packard Division
P.O.Box 22015
RIYADH
Tel: 491-97 15, 491-63 87
Telex: 202049 MEERYD SJ
CH,CS,E,M

Abdul Ghani El Ajou
P.O. Box 78
RIYADH
Tel: 40 41 717
Telex: 200 932 EL AJOU
P

SCOTLAND
See United Kingdom

SINGAPORE
Hewlett-Packard Singapore (Sales) Pte. Ltd.
#08-00 Inchcape House
450-2 Alexandra Road
P.O. Box 58 Alexandra Rd. Post Office
SINGAPORE, 9115
Tel: 631788
Telex: HPSGSO RS 34209
Cable: HEWPACK, Singapore
A,CH,CS,E,MS,P

Dynamar International Ltd.
Unit 05-11Block 6
Kolam Ayer Industrial Estate
SINGAPORE *1334*
Tel: 747-6188
Telex: RS 26283
CM

SOUTH AFRICA
Hewlett-Packard So Africa (Pty.) Ltd.
P.O. Box 120
Howard Place **CAPE PROVINCE** 7450
Pine Park Center, Forest Drive, Pinelands
CAPE PROVINCE 7405
Tel: 53-7954
Telex: 57-20006
A,CH,CM,E,M,P

Hewlett-Packard So Africa (Pty.) Ltd.
P.O. Box 37099
Overport Drive 92
DURBAN 4067
Tel: 28-4178
Telex: 6-22954
CH,CM

Hewlett-Packard So Africa (Pty.) Ltd.
6 Linton Arcade
511 Cape Road
Linton Grange
PORT ELIZABETH 6001
Tel: 041-301201
CH

Hewlett-Packard So Africa (Pty.) Ltd.
Fountain Center
Kalkden Str.
Monument Park
Ext 2
PRETORIA 0105
Tel: 45-5723
Telex: 32163
CH,E

SOUTH AFRICA (Cont'd)

Hewlett-Packard So Africa (Pty.) Ltd.
Private Bag Wendywood
SANDTON 2144
Tel: 802-5111, 802-5125
Telex: 4-20877
Cable: HEWPACK Johannesburg
A,CH,CM,CS,E,M,P

SPAIN

Hewlett-Packard Española S.A.
Calle Entenza, 321
E-**BARCELONA** 29
Tel: 322.24.51, 321.73.54
Telex: 52603 hpbee
A,CH,CS,E,M,P

Hewlett-Packard Española S.A.
Calle San Vicente S/No
Edificio Albia II 7B
E-**BILBAO** 1
Tel: 423.83.06
A,CH,E,M

Hewlett-Packard Española S.A.
Crta. de la Coruña, Km. 16, 400
Las Rozas
E-**MADRID**
Tel: (1) 637.00.11
Telex: 23515 HPE
CH,CS,M

Hewlett-Packard Española S.A.
Avda. S. Francisco Javier, S/no
Planta 10. Edificio Sevilla 2,
E-**SEVILLA** 5
Tel: 64.44.54
Telex: 72933
A,CS,M,P

Hewlett-Packard Española S.A.
C/Isabel La Catolica, 8
E-46004 **VALENCIA**
Tel: 0034/6/351 59 44
CH,P

SWEDEN

Hewlett-Packard Sverige AB
Sunnanvagen 14K
S-22226 **LUND**
Tel: (046) 13-69-79
Telex: (854) 17886 (via Spånga office)
CH

Hewlett-Packard Sverige AB
Östra Tullgatan 3
S-21128 **MALMÖ**
Tel: (040) 70270
Telex: (854) 17886 (via Spånga office)

Hewlett-Packard Sverige AB
Våstra Vintergatan 9
S-70344 **ÖREBRO**
Tel: (19) 10-48-80
Telex: (854) 17886 (via Spånga office)
CH

Hewlett-Packard Sverige AB
Skalholtsgatan 9, Kista
Box 19
S-16393 **SPÅNGA**
Tel: (08) 750-2000
Telex: (854) 17886
Telefax: (08) 7527781
A,CH,CM,CS,E,M,P

Hewlett-Packard Sverige AB
Frötallisgatan 30
S-42132 **VÄSTRA-FRÖLUNDA**
Tel: (031) 49-09-50
Telex: (854) 17886 (via Spånga office)
CH,E,P

SWITZERLAND

Hewlett-Packard (Schweiz) AG
Clarastrasse 12
CH-4058 **BASEL**
Tel: (61) 33-59-20
A

Hewlett-Packard (Schweiz) AG
7, rue du Bois-du-Lan
Case Postale 365
CH-1217 **MEYRIN** 2
Tel: (0041) 22-83-11-11
Telex:27333 HPAG CH
CH,CM,CS

Hewlett-Packard (Schweiz) AG
Allmend 2
CH-8967 **WIDEN**
Tel: (0041) 57 31 21 11
Telex: 53933 hpag ch
Cable: HPAG CH
A,CH,CM,CS,E,M,P

SYRIA

General Electronic Inc.
Nuri Basha Ahnaf Ebn Kays Street
P.O. Box 5781
DAMASCUS
Tel: 33-24-87
Telex: 411 215
Cable: ELECTROBOR DAMASCUS
E

Middle East Electronics
P.O.Box 2308
Abu Rumnaneh
DAMASCUS
Tel: 33 45 92
Telex: 411 304
M

TAIWAN

Hewlett-Packard Taiwan
Kaohsiung Office
11/F 456, Chung Hsiao 1st Road
KAOHSIUNG
Tel: (07) 2412318
CH,CS,E

Hewlett-Packard Taiwan
8th Floor Hewlett-Packard Building
337 Fu Hsing North Road
TAIPEI
Tel: (02) 712-0404
Telex: 24439 HEWPACK
Cable:HEWPACK Taipei
A,CH,CM,CS,E,M,P

Ing Lih Trading Co.
3rd Floor, 7 Jen-Ai Road, Sec. 2
TAIPEI 100
Tel: (02) 3948191
Cable: INGLIH TAIPEI
A

THAILAND

Unimesa
30 Patpong Ave., Suriwong
BANGKOK *5*
Tel: 235-5727
Telex: 84439 Simonco TH
Cable: UNIMESA Bangkok
A,CH,CS,E,M

Bangkok Business Equipment Ltd.
5/5-6 Dejo Road
BANGKOK
Tel: 234-8670, 234-8671
Telex: 87669-BEQUIPT TH
Cable: BUSIQUIPT Bangkok
P

TOGO

Societe Africaine De Promotion
B.P. 12271
LOME
Tel: 21-62-88
Telex: 5304
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
66 Independence Square
PORT-OF-SPAIN
Tel: 623-4472
Telex: 3000 POSTLX WG
P

TUNISIA

Tunisie Electronique
31 Avenue de la Liberte
TUNIS
Tel: 280-144
CH,CS,E,P

Corema
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

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

Teknim Company Ltd.
Iran Caddesi No. 7
Kavaklidere
ANKARA
Tel: 275800
Telex: 42155 TKNM TR
E,CM

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: 377951

Emitac Ltd.
P.O. Box 473
RAS AL KHAIMAH,
Tel: 28133, 21270

UNITED KINGDOM

GREAT BRITAIN

Hewlett-Packard Ltd.
Trafalgar House
Navigation Road
ALTRINCHAM
Cheshire WA14 1NU
Tel: 061 928 6422
Telex: 668068
A,CH,CS,E,M,M,P

Hewlett-Packard Ltd.
Miller House
The Ring, **BRACKNELL**
Berks RG12 1XN
Tel: 44344 424898
Telex: 848733
E

Hewlett-Packard Ltd.
Elstree House, Elstree Way
BOREHAMWOOD, Herts WD6 1SG
Tel: 01 207 5000
Telex: 8952716
E,CH,CS,P

Hewlett-Packard Ltd.
Oakfield House, Oakfield Grove
Clifton **BRISTOL,** Avon BS8 2BN
Tel: 0272 736806
Telex: 444302
CH,CS,E,P

SALES & SUPPORT OFFICES
Arranged alphabetically by country

GREAT BRITAIN (Cont'd)
Hewlett-Packard Ltd.
Bridewell House
Bridewell Place
LONDON EC4V 6BS
Tel: 01 583 6565
Telex: 298163
CH,CS,P

Hewlett-Packard Ltd.
Fourier House
257-263 High Street
LONDON COLNEY
Herts. AL2 1HA, St. Albans
Tel: 0727 24400
Telex: 1-8952716
CH,CS

Hewlett-Packard Ltd.
Pontefract Road
NORMANTON, West Yorkshire WF6 1RN
Tel: 0924 895566
Telex: 557355
CH,CS,P

Hewlett-Packard Ltd.
The Quadrangle
106-118 Station Road
REDHILL, Surrey RH1 1PS
Tel: 0737 68655
Telex: 947234
CH,CS,E,P

Hewlett-Packard Ltd.
Avon House
435 Stratford Road
Shirley, **SOLIHULL,** West Midlands
B90 4BL
Tel: 021 745 8800
Telex: 339105
CH,CS,E,P

Hewlett-Packard Ltd.
West End House
41 High Street, West End
SOUTHAMPTON
Hampshire S03 3DQ
Tel: 04218 6767
Telex: 477138
CH,CS,P

Hewlett-Packard Ltd.
King Street Lane
Winnersh, **WOKINGHAM**
Berkshire RG11 5AR
Tel: 0734 784774
Telex: 847178
A,CH,CS,E,M,P

Hewlett-Packard Ltd.
Nine Mile Ride
Easthampstead, **WOKINGHAM**
Berkshire, 3RG11 3LL
Tel: 0344 773100
Telex: 848805
CH,CS,E,P

IRELAND

NORTHERN IRELAND
Hewlett-Packard Ltd.
Cardiac Services Building
95A Finaghy Road South
BELFAST BT10 OBY
Tel: 0232 625-566
Telex: 747626
CH,CS

SCOTLAND
Hewlett-Packard Ltd.
SOUTH QUEENSFERRY
West Lothian, EH30 9TG
Tel: 031 331 1188
Telex: 72682
CH,CM,CS,E,M,P

UNITED STATES

Alabama
Hewlett-Packard Co.
700 Century Park South, Suite 128
BIRMINGHAM, AL 35226
Tel: (205) 822-6802
C,CH,CS,P*

Hewlett-Packard Co.
420 Wynn Drive
P.O. Box 7700
HUNTSVILLE, AL 35807
Tel: (205) 830-2000
C,CH,CM,CS,E,M*

Alaska
Hewlett-Packard Co.
3601 C St., Suite 1234
ANCHORAGE, AK 99503
Tel: (907) 563-8855
CH,CS,E

Arizona
Hewlett-Packard Co.
8080 Pointe Parkway West
PHOENIX, AZ 85044
Tel: (602) 273-8000
A,CH,CM,CS,E,M

Hewlett-Packard Co.
2424 East Aragon Road
TUCSON, AZ 85706
Tel: (602) 573-7400
CH,E,M**

California
Hewlett-Packard Co.
99 South Hill Dr.
BRISBANE, CA 94005
Tel: (415) 330-2500
CH,CS

Hewlett-Packard Co.
P.O. Box 7830 (93747)
5060 E. Clinton Avenue, Suite 102
FRESNO, CA 93727
Tel: (209) 252-9652
CH,CS,M

Hewlett-Packard Co.
1421 S. Manhattan Av.
FULLERTON, CA 92631
Tel: (714) 999-6700
CH,CM,CS,E,M

Hewlett-Packard Co.
320 S. Kellogg, Suite B
GOLETA, CA 93117
Tel: (805) 967-3405
CH

Hewlett-Packard Co.
5400 W. Rosecrans Blvd.
LAWNDALE, CA 90260
P.O. Box 92105
LOS ANGELES, CA 90009
Tel: (213) 643-7500
Telex: 910-325-6608
CH,CM,CS,M

Hewlett-Packard Co.
3155 Porter Drive
PALO ALTO, CA 94304
Tel: (415) 857-8000
CH,CS,E

Hewlett-Packard Co.
4244 So. Market Court, Suite A
P.O. Box 15976
SACRAMENTO, CA 95813
Tel: (916) 929-7222
A*,CH,CS,E,M

Hewlett-Packard Co.
9606 Aero Drive
P.O. Box 23333
SAN DIEGO, CA 92123
Tel: (619) 279-3200
CH,CM,CS,E,M

Hewlett-Packard Co.
2305 Camino Ramon 'C'
SAN RAMON, CA 94583
Tel: (415) 838-5900
CH,CS

Hewlett-Packard Co.
3005 Scott Boulevard
SANTA CLARA, CA 95050
Tel: (408) 988-7000
Telex: 910-338-0586
A,CH,CM,CS,E,M

Hewlett-Packard Co.
5703 Corsa Avenue
WESTLAKE VILLAGE, CA 91362
Tel: (213) 706-6800
E*,CH*,CS*

Colorado
Hewlett-Packard Co.
24 Inverness Place, East
ENGLEWOOD, CO 80112
Tel: (303) 649-5000
A,CH,CM,CS,E,M

Connecticut
Eff. Dec. 1, 1984
Hewlett-Packard Co.
500 Sylvan Av.
BRIDGEPORT, CT 06606
Tel: (203) 371-6454
CH,CS,E

Hewlett-Packard Co.
47 Barnes Industrial Road South
P.O. Box 5007
WALLINGFORD, CT 06492
Tel: (203) 265-7801
A,CH,CM,CS,E,M

Florida
Hewlett-Packard Co.
2901 N.W. 62nd Street
P.O. Box 24210
FORT LAUDERDALE, FL 33307
Tel: (305) 973-2600
CH,CS,E,M,P*

Hewlett-Packard Co.
4080 Woodcock Drive, Suite 132
JACKSONVILLE, FL 32207
Tel: (904) 398-0663
C*,CH*,M**

Hewlett-Packard Co.
6177 Lake Ellenor Drive
P.O. Box 13910
ORLANDO, FL 32859
Tel: (305) 859-2900
A,C,CH,CM,CS,E,P*

Hewlett-Packard Co.
4700 Bayoue Blvd.
Building 5
PENSACOLA, FL 32505
Tel: (904) 476-8422
A,C,CH,CM,CS,M

Hewlett-Packard Co.
5550 Idlewild, #150
P.O. Box 15200
TAMPA, FL 33684
Tel: (813) 884-3282
A*,C,CH,CS,E*,M*,P*

Georgia
Hewlett-Packard Co.
2000 South Park Place
P.O. Box 105005
ATLANTA, GA 30348
Tel: (404) 955-1500
Telex: 810-766-4890
A,C,CH,CM,CS,E,M,P*

Hawaii
Hewlett-Packard Co.
Kawaiahao Plaza, Suite 190
567 South King Street
HONOLULU, HI 96813
Tel: (808) 526-1555
A,CH,E,M

Illinois
Hewlett-Packard Co.
304 Eldorado Road
P.O. Box 1607
BLOOMINGTON, IL 61701
Tel: (309) 662-9411
CH,M**

Hewlett-Packard Co.
525 W. Monroe, #1300
CHICAGO, IL 60606
Tel: (312) 930-0010
CH,CS

Hewlett-Packard Co.
1200 Diehl
NAPERVILLE, IL 60566
Tel: (312) 357-8800
CH*,CS

Hewlett-Packard Co.
5201 Tollview Drive
ROLLING MEADOWS, IL 60008
Tel: (312) 255-9800
Telex: 910-687-1066
A,CH,CM,CS,E,M

Indiana
Hewlett-Packard Co.
11911 N. Meridian St.
CARMEL, IN 46032
Tel: (317) 844-4100
A,CH,CM,CS,E,M

Iowa
Hewlett-Packard Co.
4070 22nd Av. SW
CEDAR RAPIDS, IA 52404
Tel: (319) 390-4250
CH,CS,E,M

UNITED STATES (Cont'd)

Hewlett-Packard Co.
4201 Corporate Dr.
WEST DES MOINES, IA 50265
Tel: (515) 224-1435
A**,CH,M**

Kentucky

Hewlett-Packard Co.
10300 Linn Station Road, #100
LOUISVILLE, KY 40223
Tel: (502) 426-0100
A,CH,CS,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,CH,E,M,P*

Maryland

Hewlett-Packard Co.
3701 Koppers Street
BALTIMORE, MD 21227
Tel: (301) 644-5800
Telex: 710-862-1943
A,CH,CM,CS,E,M

Hewlett-Packard Co.
2 Choke Cherry Road
ROCKVILLE, MD 20850
Tel: (301) 948-6370
A,CH,CM,CS,E,M

Massachusetts

Hewlett-Packard Co.
1775 Minuteman Road
ANDOVER, MA 01810
Tel: (617) 682-1500
A,C,CH,CS,CM,E,M,P*

Hewlett-Packard Co.
32 Hartwell Avenue
LEXINGTON, MA 02173
Tel: (617) 861-8960
CH,CS,E

Michigan

Hewlett-Packard Co.
4326 Cascade Road S.E.
GRAND RAPIDS, MI 49506
Tel: (616) 957-1970
CH,CS,M

Hewlett-Packard Co.
39550 Orchard Hill Place Drive
NOVI, MI 48050
Tel: (313) 349-9200
A,CH,CS,E,M

Hewlett-Packard Co.
1771 W. Big Beaver Road
TROY, MI 48084
Tel: (313) 643-6474
CH,CS

Minnesota

Hewlett-Packard Co.
2025 W. Larpenteur Ave.
ST. PAUL, MN 55113
Tel: (612) 644-1100
A,CH,CM,CS,E,M

Missouri

Hewlett-Packard Co.
1001 E. 101st Terrace
KANSAS CITY, MO 64131
Tel: (816) 941-0411
A,CH,CM,CS,E,M

Hewlett-Packard Co.
13001 Hollenberg Drive
BRIDGETON, MO 63044
Tel: (314) 344-5100
A,CH,CS,E,M

Nebraska

Hewlett-Packard
10824 Old Mill Rd., Suite 3
OMAHA, NE 68154
Tel: (402) 334-1813
CM,M

New Jersey

Hewlett-Packard Co.
120 W. Century Road
PARAMUS, NJ 07652
Tel: (201) 265-5000
A,CH,CM,CS,E,M

Hewlett-Packard Co.
20 New England Av. West
PISCATAWAY, NJ 08854
Tel: (201) 981-1199
A,CH,CM,CS,E

New Mexico

Hewlett-Packard Co.
11300 Lomas Blvd.,N.E.
P.O. Box 11634
ALBUQUERQUE, NM 87112
Tel: (505) 292-1330
CH,CS,E,M

New York

Hewlett-Packard Co.
5 Computer Drive South
ALBANY, NY 12205
Tel: (518) 458-1550
A,CH,E,M

Hewlett-Packard Co.
9600 Main Street
P.O. Box AC
CLARENCE, NY 14031
Tel: (716) 759-8621
CH,CS,E

Hewlett-Packard Co.
200 Cross Keys Office Park
FAIRPORT, NY 14450
Tel: (716) 223-9950
A,CH,CM,CS,E,M

Hewlett-Packard Co.
7641 Henry Clay Blvd.
LIVERPOOL, NY 13088
Tel: (315) 451-1820
A,CH,CM,CS,E,M

Hewlett-Packard Co.
No. 1 Pennsylvania Plaza
55th Floor
34th Street & 8th Avenue
MANHATTAN NY 10119
Tel: (212) 971-0800
CH,CS,M*

Hewlett-Packard Co.
15 Myers Corner Rd.
WAPPINGER FALLS, NY 12590
CM,E

Hewlett-Packard Co.
250 Westchester Avenue
WHITE PLAINS, NY 10604
Tel: (914) 684-6100
CM,CH,CS,E

Hewlett-Packard Co.
3 Crossways Park West
WOODBURY, NY 11797
Tel: (516) 921-0300
A,CH,CM,CS,E,M

North Carolina

Hewlett-Packard Co.
305 Gregson Dr.
CARY, NC 27511
Tel: (919) 467-6600
C,CH,CM,CS,E,M,P*

Hewlett-Packard Co.
9600-H Southern Pine Blvd.
CHARLOTTE, NC 28210
Tel: (704) 527-8780
CH*,CS*

Hewlett-Packard Co.
5605 Roanne Way
P.O. Box 26500
GREENSBORO, NC 27420
Tel: (919) 852-1800
A,C,CH,CM,CS,E,M,P*

Ohio

Hewlett-Packard Co.
9920 Carver Road
CINCINNATI, OH 45242
Tel: (513) 891-9870
CH,CS,M

Hewlett-Packard Co.
16500 Sprague Road
CLEVELAND, OH 44130
Tel: (216) 243-7300
A,CH,CM,CS,E,M

Hewlett-Packard Co.
980 Springboro Pike
MIAMISBURG, OH 45343
Tel: (513) 859-8202
A,CH,CM,E*,M

Hewlett-Packard Co.
675 Brooksedge Blvd.
WESTERVILLE, OH 43081
Tel: (614) 436-1041
CH,CM,CS,E*

Oklahoma

Hewlett-Packard Co.
304 N. Meridian, Suite A
P.O. Box 75609
OKLAHOMA CITY, OK 73147
Tel: (405) 946-9499
C,CH,CS,E*,M

Hewlett-Packard Co.
3840 S. 103rd E. Ave., #100
P.O. Box 35747
TULSA, OK 74153
Tel: (918) 665-3300
A**,C,CH,CS,M*,E,P*

Oregon

Hewlett-Packard Co.
9255 S. W. Pioneer Court
P.O. Box 328
WILSONVILLE, OR 97070
Tel: (503) 682-8000
A,CH,CS,E*,M

Pennsylvania

Hewlett-Packard Co.
50 Dorchester Rd.
P.O. Box 6080
HARRISBURG, PA 17111
Tel: (717) 657-5900
C

Hewlett-Packard Co.
111 Zeta Drive
PITTSBURGH, PA 15238
Tel: (412) 782-0400
A,CH,CS,E,M

Hewlett-Packard Co.
2750 Monroe Boulevard
P.O. Box 713
VALLEY FORGE, PA 19482
Tel: (215) 666-9000
A,CH,CM,CS,E,M

South Carolina

Hewlett-Packard Co.
Brookside Park, Suite 122
1 Harbison Way
P.O. Box 21708
COLUMBIA, SC 29221
Tel: (803) 732-0400
A,C,CH,CS,M

Hewlett-Packard Co.
100 Executive Cntr. Dr.
Koger Executive Center
Chesterfield Bldg., Suite 124
GREENVILLE, SC 29615
Tel: (803) 297-4120
C

Tennessee

Hewlett-Packard Co.
One Energy Centr. #200
Pellissippi Pkwy.
P.O. Box 22490
KNOXVILLE, TN 37933
Tel: (615) 966-4747
A,C,CH,CS,M

Hewlett-Packard Co.
3070 Directors Row
MEMPHIS, TN 38131
Tel: (901) 346-8370
A,C,M

Hewlett-Packard Co.
220 Great Circle Road, Suite 116
NASHVILLE, TN 37228
Tel: (615) 255-1271
C,M,P*

Texas

Hewlett-Packard Co.
11002-B Metric Boulevard
AUSTIN, TX 78758
Tel: (512) 835-6771
C,CM,E,P*

Hewlett-Packard Co.
5700 Cromo Dr
P.O. Box 12903
EL PASO, TX 79913
Tel: (915) 833-4400
CH,E*,M**

SALES & SUPPORT OFFICES
Arranged alphabetically by country

UNITED STATES (Cont'd)

Hewlett-Packard Co.
3952 Sand Shell St
FORT WORTH, TX 76137
Tel: (817) 232-9500
A,C,CH,E,M

Hewlett-Packard Co.
10535 Harwin Drive
P.O. Box 42816
HOUSTON, TX 77042
Tel: (713) 776-6400
A,C,CH,CS,E,M,P*

Hewlett-Packard Co.
511 W. John W. Carpenter Fwy.
Royal Tech. Center #100
IRVINE, TX 75062
Tel: (214) 556-1950
C,CH,CS,E

Hewlett-Packard Co.
930 E. Campbell Rd.
P.O. Box 83/1270
RICHARDSON, TX 75083-1270
Tel: (214) 231-6101
A,CH,CM,CS,E,M,P*

Hewlett-Packard Co.
1020 Central Parkway South
P.O. Box 32993
SAN ANTONIO, TX 78232
Tel: (512) 494-9336
A,C,CH,CS,E,M,P*

Utah

Hewlett-Packard Co.
3530 W. 2100 South
P.O. Box 26626
SALT LAKE CITY, UT 84126
Tel: (801) 974-1700
A,CH,CS,E,M

Virginia

Hewlett-Packard Co.
4305 Cox Road
GLEN ALLEN, VA 23060
P.O. Box 9669
RICHMOND, VA 23228
Tel: (804) 747-7750
A,C,CH,CS,E,M,P*

Washington

Hewlett-Packard Co.
15815 S.E. 37th Street
BELLEVUE, WA 98006
Tel: (206) 643-4000
A,CH,CM,CS,E,M

Hewlett-Packard Co.
708 North Argonne Road
P.O. Box 3808
SPOKANE, WA 99220-3808
Tel: (509) 922-7000
CH,CS

West Virginia

Hewlett-Packard Co.
4604 MacCorkle Ave.
CHARLESTON, WV 25304
Tel: (304) 925-0492
A,M

Wisconsin

Hewlett-Packard Co.
275 N. Corporate Dr.
BROOKFIELD, WI 53005
Tel: (414) 784-8800
A,CH,CS,E*,M

URUGUAY

Pablo Ferrando S.A.C. e I.
Avenida Italia 2877
Casilla de Correo 370
MONTEVIDEO
Tel: 80-2586
Telex: Public Booth 901
A,CM,E,M

Mini Computadores, Ltda.
Avda. del Libertador Brig
Gral Lavalleja 2071
Local 007
MONTEVIDEO
Tel: 29-55-22
Telex: 901 P BOOTH UY
P

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.
3RA Transversal Los Ruices Norte
Edificio Segre 1, 2 & 3
Apartado 50933
CARACAS 1071
Tel: 239-4133
Telex: 251046 HEWPACK
A,CH,CS,E,M,P

Hewlett-Packard de Venezuela C.A.
Residencias Tia Betty Local 1
Avenida 3 y con calfe 75
MARACAIBO, Estado Zulia
Apartado 2646
Tel: (061) 75801-75805-75806-80304
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: (041) 222992/223024
CH,CS,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

CIZUCA
Cientifica Zulia C.A.
Calle 70, Los Olivos
No. 66-86
Apartado 1843
MARACAIBO
Tel: 54-64-37, 54-63-85, 54-64-94
Telex: 62144
A

YUGOSLAVIA

Do Hermes
General Zdanova 4
Telex: YU-11000 ***BEOGRAD***
A,CH,E,P

Hermes
Titova 50
Telex: YU-61000 ***LJUBLJANA***
CH,CS,E,M,P

Elektrotehna
Titova 51
Telex: YU-61000 ***LJUBLJANA***
CM

ZAMBIA

R.J. Tilbury (Zambia) Ltd.
P.O. Box 32792
LUSAKA
Tel: 215590
Telex: 40128
E

ZIMBABWE

Field Technical Sales
45 Kelvin Road, North
P.B. 3458
SALISBURY
Tel: 705 231
Telex: 4-122 RH
E,P

August 1984

HP distributors are printed in italics.

64811-90902, NOVEMBER 1985
E1185
Replaces 64811-90902, January 1982

HEWLETT PACKARD

PRINTED IN U.S.A.