8087 SUPPORT LIBRARY REFERENCE MANUAL

Order Number: 121725-001

intel®

8087 SUPPORT LIBRARY REFERENCE MANUAL

Order Number: 121725-001

A1094 / 1183 / 6.5K / DD / KH

REV.	REVISION HISTORY	DATE
-001	Original issue.	8/81

HOW TO USE THIS MANUAL

This manual is a programmer's guide to the 8087 Support Library. The Library is a package of software tools that enable the 8087 or the full 8087 emulator to provide a full range of floating point capabilities to ASM-86 and PL/M-86 programmers.

This manual is primarily a reference guide, to be used when you are writing code for ASM-86 and PL/M-86 programs. We have made the documentation for each procedure and function as self-contained as practical, to avoid excessive page-flipping.

To obtain an initial overview of the library, read Chapters 1 and 2 and the first parts of Chapters 3, 4, and 5. Read Chapter 6 if you are concerned with compliance to the IEEE Standard. Familiarize yourself with the contents of the appendices.

To find specific facts about individual procedures and functions, look in the reference pages, headed by large names, at the ends of Chapters 3, 4, and 5.

Note that the reference pages contain much material discussing unusual values such as denormals, unnormals, pseudo-zeroes, etc. You should remember that these values are very uncommon. For most applications they never occur, and you may ignore the paragraphs that discuss them.

For quick reference when you are an experienced user of the 8087 Support Library, consult Appendices D and E.

Related Publications

The following manuals will be useful to you when you are reading this manual. Particularly crucial is the documentation on the 8087 itself, to be found in the *Assembly Language Reference Manual* and the *8086 User's Manual Numerics Supplement*. These manuals all describe the Series-III versions (8086-based) of the software products.

- *iAPX 86,88 Family Utilities User's Guide*, Order Number 121616
- *PL/M-86 User's Guide*, Order Number 121636
- *8086/8087/8088 Macro Assembly Language Reference Manual*, Order Number 121627
- *8086/8087/8088 Macro Assembler Operating Instructions*, Order Number 121628
- *The 8086 Family User's Manual Numerics Supplement*, Order Number 121586

Nomenclature for the 8086 Family

Three naming conventions are used in this manual to describe the 8086 family of microprocessors:

1. *The iAPX Convention* describes the *system* containing one or more components of the family. It must contain an 8086 or 8088 microprocessor. The table below shows the components used in the most common iAPX systems.
2. *The 8086/8087/8088/8089 Convention* describes a specific *component* within an iAPX system.

3. *The NDP/NPX Convention* describes a *numeric functional unit* which can be realized by more than one hardware configuration.
   - NPX (Numeric Processor Extension) refers either to the 8087 component or the software 8087 full emulator.
   - NDP (Numeric Data Processor) refers to any iAPX system which contains an NPX.

System Name	8086	8087	8088	8089
iAPX 86/10	1			
iAPX 86/11	1			1
iAPX 86/12	1			2
iAPX 86/20	1	1		
iAPX 86/21	1	1		1
iAPX 86/22	1	1		2
iAPX 88/10			1	
iAPX 88/11			1	1
iAPX 88/12			1	2
iAPX 88/20		1	1	
iAPX 88/21		1	1	1
iAPX 88/22		1	1	2

Notational Conventions

The programming examples given in this manual are presented as chunks of code that can be inserted intact into a PL/M-86 or ASM-86 program. They contain no formulas of general syntax.

The examples of LINK86 and LOC86 invocations follow these conventions common to Intel manuals:

- black background is used to denote user input to the operating system.
- <cr> indicates a carriage return.

CONTENTS

CONTENTS (Cont'd.)

ILLUSTRATIONS

CHAPTER 1
INTRODUCTION

The 8087 Support Library greatly facilitates the use of floating point calculations in ASM-86 and PL/M-86. It adds to these languages many of the functions that are built into applications programming languages, such as Pascal-86 and FORTRAN-86.

The full 8087 emulator, E8087, and the interface libraries E8087.LIB, 8087.LIB, and 87NULL.LIB, allow a choice of run-time environments for modules containing floating point code. The emulator is the functional and programming equivalent to the 8087, implemented in 16K bytes of 8086 software.

The decimal conversion module, DCON87.LIB, aids the translation between human-readable numbers of many decimal formats, and binary numbers in all the formats supported by the NPX. It also provides translation between binary formats.

The common elementary function library, CEL87.LIB, provides a useful assortment of transcendental, rounding, and other common functions involving floating point numbers.

The error handler module, EH87.LIB, makes it easy for you to write interrupt procedures which recover from floating point error conditions.

Why the Support Library Is Needed

There are three areas in which many applications will require support software to overcome functional limitations of the NDP.

First, the NDP is exclusively a binary arithmetic machine. Although the NDP manipulates a variety of data types, it cannot operate directly on floating-point decimal quantities. Software must be invoked to convert decimal inputs to binary formats, and binary answers to decimal output.

Second, the 8087 instruction set is limited in its implementation of elementary functions. There is a tangent function, for example, but no sine, and the tangent is defined only for inputs between 0 and $\pi/4$. The implementation of a full set of elementary functions with complete domains is left to software.

Third, the handling of unmasked exceptions can complicate application software. Although the masked mode of error detection and recovery is adequate for most applications, there is sometimes need for customized error recovery. Deciphering NPX status information and restoring the NPX to a recovery state can be quite difficult. The task can be made easier with software.

The 8087 Support Library provides the software necessary to eliminate the above deficiencies. The Support Library makes the NDP a complete, accurate, easy-to-use floating point computer.

System Software Needed to Use the Library

Since the Support Library is called by ASM-86 and PL/M-86 programs, you will need either the ASM-86 assembler or the PL/M-86 compiler.

You will need to compile your PL/M-86 modules under the MEDIUM or LARGE models of computation, since all of the procedures of the Support Library are FAR procedures.

You will also need LINK86 and LOC86 to convert the object modules generated by your programs into programs that can be loaded or programmed into memory (or programmed into ROM) and executed.

All of the above programs come in versions that run on an 8080/8085-based development system under ISIS-II. They also come in versions that run on an 8086-based development system via the Series III RUN utility.

For consistency, we give all examples of LINK86 invocations as they appear running the 8080/8085 version of LINK86. LINK86 and the Support Library are assumed to be on drive 0; all user-supplied files are assumed to be on drive 1.

No run-time system software is needed for the Support Library. Once you have obtained the final executable program, the program will run on any NDP system.

CHAPTER 2 THE FULL 8087 EMULATOR AND INTERFACE LIBRARIES

Overview

The full 8087 emulator is a software module, executable on an iAPX86/10 or iAPX88/10, which completely and exactly duplicates all the functions of the 8087. You can use the emulator to develop prototypes for systems that will have an 8087; or you can use the emulator if you need the 8087's floating point capabilities but do not need its speed. The emulator is contained in the file E8087, which you include in a LINK86 statement with your program modules.

Included with the emulator are three interface libraries, E8087.LIB, 8087.LIB and 87NULL.LIB. By selecting one of these three libraries to appear in your LINK86 statement, you choose the run-time environment of your program: running with the emulator, with the 8087 component, or with no NPX capability.

The interface libraries also contain the procedures INIT87 and INITFP, which initialize the NPX under the various environments.

The full 8087 emulator is an expanded update of the partial 8087 emulator, PE8087, which we offer with PL/M-86. If you are adding E8087 to a LINK86 statement that includes PE8087, you must eliminate PE8087 from the statement.

Whenever we mention "the emulator" in this manual, we are referring to the full 8087 emulator.

How to Use the Emulator in Your Programs

To invoke the emulator in your source programs, you simply issue instructions as if you had an 8087. There is absolutely no difference between source programs that use the emulator and source programs that use the component.

Furthermore, there is no difference between unlinked object modules (or libraries) that use the emulator, and those that use the component. CEL87.LIB, DCON87.LIB and EH87.LIB can all be made to run either with the emulator or with the 8087 component.

The changes that you make in switching between the emulator and the component occur entirely in the invocation of LINK86. The files E8087 and E8087.LIB are linked to your object modules when you are using the emulator; the one file 8087.LIB is linked when you are using the component.

The source and object module compatibility between the emulator and the component is accomplished by a set of special external variables. These variables are issued automatically by all Intel translators that handle 8087 code, and are invisible to the programmer except in some LOC86 listings. The names of the variables are given in Appendix B.

Emulation of Unmasked Exceptions

When the 8087 detects an error, it checks its control word to see if the error is masked or unmasked. If unmasked, an interrupt signal is generated on one of the output pins of the 8087.

The emulator, in the same unmasked error situation, makes a call to interrupt 16 decimal. Thus, an INTERRUPT procedure is executed; the address of the procedure is given in a four-byte pointer at memory location 00040 hex.

Note that with the 8087 component you can configure your system and program your interrupt controller so that exactly the same interrupt is generated when the 8087 sees an unmasked error. Thus, the source code even for your exception handler can remain exactly the same.

Emulator Memory and Stack Requirements

The emulator, of course, occupies 8086 memory that is not needed when the 8087 component is used. The memory used comes in four segments:

1. The code of the emulator, which can be located in read-only memory, is in a segment called aqmCODE. It occupies about 16K bytes.
2. The emulator uses a non-reentrant data block, which must be in RAM. The block is about 150 bytes long, and is in a segment called aqmDATA.
3. The emulator uses 8086 software interrupt 16 decimal, and interrupts 20 through 31 decimal. The addresses of these interrupts are stored in memory locations 00040 hex through 00043 hex, and 00050 hex through 0007F hex, by INIT87 or INITFP. While you are still using the emulator, these locations cannot be changed.
4. The emulator uses about 60 bytes of the same 8086 stack as used by the calling routine. This stack must be in a segment called STACK.

How to Initialize the NPX

When power is first applied to an NDP system, the NPX must be brought to a known state. This is accomplished by calling one of the 8087 initialization procedures. If the emulator is being used, the addresses of interrupts 20 through 31 must be loaded before any floating point instructions are executed.

We have provided two 8087 initialization procedures, INIT87 and INITFP, in each of the three interface libraries. The 8087.LIB versions initialize the 8087 component and set the 8087 control word. The E8087.LIB versions load interrupts 20 through 31 before the 8087 initialization. The 87NULL.LIB versions satisfy 8087 externals with null procedures. 87NULL.LIB should only be used with software that contains no NPX instructions.

The difference between INIT87 and INITFP is the 8087 control word default settings. INIT87 masks all exceptions. INITFP masks all exceptions except "I". Although you may later modify the control word, you should first call either INIT87 or INITFP.

PL/M-86 Usage of INIT87 or INITFP

Following is the declaration of INIT87, which must appear at the top of your initialization module, and the call, which must be invoked before any floating point arithmetic is performed. The statements for INITFP are the same, with INITFP instead of INIT87.

Since INIT87 and INITFP are FAR procedures, you must compile the PL/M-86 module under the MEDIUM or LARGE controls.

```
INIT87: PROCEDURE EXTERNAL;
END;

CALL INIT87;
```

ASM-86 Usage of INIT87 or INITFP

Following is the declaration of INIT87, which must appear at the top of your initialization module, and the call instruction, which must appear within the initialization procedure. The statements for INITFP are the same, with INITFP instead of INIT87.

```
; The following line must appear outside of all SEGMENT-
; ENDS pairs
   EXTRN  INIT87: FAR

   CALL INIT87     ; Set up the NPX
```

Linkage of the Emulator and the Interface Libraries to User Programs

If your target system has an 8087 component, link the library 8087.LIB to your object modules. Neither the emulator (E8087) nor the partial emulator (PE8087) should appear in the LINK86 command.

If your target system does not have an 8087 component, link the emulator E8087 and the interface library E8087.LIB to your object modules. The partial emulator PE8087 should not appear in the LINK86 command.

If you issue a call to INIT87 or INITFP but do not have any other floating point instructions, you should link 87NULL.LIB to your object modules.

The module E8087 may appear anywhere in the sequence of input modules to LINK86. The interface libraries must appear after all modules that contain 8087 code.

Following is the suggested order for the object modules in your LINK86 command.

Your object modules
DCON87.LIB if you are using it
CEL87.LIB if you are using it
EH87.LIB if you are using it
8087.LIB if you are using the component, or
E8087,E8087.LIB if you are using the emulator.

Examples: Suppose your program consists of two modules, MYMOD1.OBJ and MYMOD2.OBJ, created with either ASM-86 or PL/M-86. If the program is to be executed on a system using the emulator, issue the command

```
-LINK86 :F1:MYMOD1.OBJ, :F1:MYMOD2.OBJ, &<cr>
>> :F0:E8087, :F0:E8087.LIB TO :F1:MYPROG.LNK
```

To modify the program so that it runs on a system using the 8087 component, you do not have to change your source files. Simply change the LINK86 command to

```
-LINK86 :F1:MYMOD1.OBJ, :F1:MYMOD2.OBJ, &<cr>
>> :F0:8087.LIB TO :F1:MYPROG.LNK
```

If your program calls INIT87 or INITFP, but contains no other floating point instructions, issue the command

```
-LINK86 :F1:MYMOD1.OBJ, :F1:MYMOD2.OBJ, &<cr>
>> :F0:87NULL.LIB TO :F1:MYPROG.LNK
```

CHAPTER 3
THE DECIMAL CONVERSION LIBRARY

This chapter describes the use of DCON87.LIB, a library of procedures which convert floating point numbers from one format of storage to another.

Overview of DCON87 Procedures

Intel supports three different binary formats for the internal representation of floating point numbers (see Appendix C). DCON87.LIB provides translation between these formats and an ASCII-encoded string of decimal digits.

The binary-to-decimal procedure mqcBIN_DECLOW accepts a binary number in any of the three formats. Because there are so many output formats for floating point numbers, mqcBIN_DECLOW does not attempt to provide a finished, formatted text string. Instead, it provides the "building blocks" for you to use to construct the output string which meets your exact format specification.

The decimal-to-binary procedure mqcDEC_BIN accepts a text string which consists of a decimal number with optional sign, decimal point, and/or power-of-ten exponent. It translates the string into the caller's choice of binary formats.

An alternate decimal-to-binary procedure mqcDECLOW_BIN provides an input format similar to the output of mqcBIN_DECLOW. The low-level input interface is provided for callers who have already broken the decimal number into its constituent parts.

The procedures mqcLONG_TEMP, mqcSHORT_TEMP, mqcTEMP_LONG, and mqcTEMP_SHORT convert floating point numbers between the longest binary format, TEMP_REAL, and the shorter formats. The conversions are performed so that the outputs are NaN's if and only if the inputs are NaN's. This improves upon the conversion algorithm used by the 8087 when it loads shorter formats to and from its stack.

All DCON87 procedure names begin with the three letters "mqc". We have added this prefix to reduce the chance that a DCON87 name will conflict with another name in your program. To make the names more readable, we have put the "mqc" prefix in lower case letters throughout this chapter.

DCON87 contains alternate public names for its procedures which are used by some Intel translators. These names are listed in Appendix F.

Declaring DCON87 Procedures in Your ASM-86 Programs

In each source program module which calls a DCON87 procedure, you must declare that procedure as external before it is used.

Following are the declarations you should make at the top of your ASM-86 program. You should include only those procedures you use.

All DCON87 procedures must be declared FAR.

This ASM-86 code also includes declarations which ease the construction of parameters. The ASM-86 examples given with the procedures later in this chapter assume that these declarations have been made.

To make reference easier, we have duplicated the procedure declarations in the examples for the individual functions. You should not duplicate the declarations in your programs.

```
; ASM-86 declarations for using the DCON decimal
; conversion library

; the following EXTRN statements must appear outside of
; all SEGMENT-ENDS pairs.

EXTRN mqcBIN_DECLOW: FAR
EXTRN mqcDEC_BIN: FAR
EXTRN mqcDECLOW_BIN: FAR
EXTRN mqcLONG_TEMP: FAR
EXTRN mqcSHORT_TEMP: FAR
EXTRN mqcTEMP_LONG: FAR
EXTRN mqcTEMP_SHORT: FAR

SHORT_REAL EQU 0
LONG_REAL  EQU 2
TEMP_REAL  EQU 3

BIN_DECLOW_BLOCK STRUC
  BIN_PTR      DD ?
  BIN_TYPE     DB ?
  DEC_LENGTH   DB ?
  DEC_PTR      DD ?
  DEC_EXPONENT DW ?
  DEC_SIGN     DB ?
BIN_DECLOW_BLOCK ENDS

DEC_BIN_BLOCK STRUC
  DD ?   ;  The names of these fields are the same as in
  DB ?   ;    BIN_DECLOW_BLOCK.
  DB ?
  DD ?
DEC_BIN_BLOCK ENDS

DECLOW_BIN_BLOCK STRUC
  DD ?     ; The names of these fields are the
  DB ?     ;   same as in BIN_DECLOW_BLOCK
  DB ?
  DD ?
  DW ?
  DB ?
DECLOW_BIN_BLOCK ENDS
```

Declaring DCON87 Procedures in Your PL/M-86 Programs

Following are the declarations you should make at the top of your PL/M-86 program. You should include only those procedures you use.

This PL/M-86 code also includes declarations which ease the construction of parameters. The PL/M-86 examples given with the procedures later in this chapter assume that these declarations have been made.

To make reference easier, we have duplicated the procedure declarations in the examples for the individual functions. You should not duplicate the declarations in your programs.

To use DCON87.LIB, you must compile your PL/M-86 modules under the MEDIUM or LARGE models of computation, since DCON's procedures are FAR procedures.

See the section entitled "An Alternate Method of DCON87 Error Detection" later in this chapter for a discussion of when you might want to declare the procedures as BYTE procedures (to receive an error status).

```
/* PL/M-86 declarations for using the DCON decimal
   conversion library */

mqcBIN_DECLOW: PROCEDURE (BLOCK_PTR) EXTERNAL;
  DECLARE BLOCK_PTR POINTER;
END mqcBIN_DECLOW;

mqcDEC_BIN: PROCEDURE (BLOCK_PTR) EXTERNAL;
  DECLARE BLOCK_PTR POINTER;
END mqcDEC_BIN;

mqcDECLOW_BIN: PROCEDURE (BLOCK_PTR) EXTERNAL;
  DECLARE BLOCK_PTR POINTER;
END mqcDECLOW_BIN;

mqcLONG_TEMP: PROCEDURE (LONG_REAL_PTR,TEMP_REAL_PTR)
  EXTERNAL;
  DECLARE (LONG_REAL_PTR,TEMP_REAL_PTR) POINTER;
END mqcLONG_TEMP;

mqcSHORT_TEMP: PROCEDURE (SHORT_REAL_PTR,TEMP_REAL_PTR)
  EXTERNAL;
  DECLARE (SHORT_REAL_PTR,TEMP_REAL_PTR) POINTER;
END mqcSHORT_TEMP;

mqcTEMP_LONG: PROCEDURE (TEMP_REAL_PTR,LONG_REAL_PTR)
  EXTERNAL;
  DECLARE (TEMP_REAL_PTR,LONG_REAL_PTR) POINTER;
END mqcTEMP_LONG;

mqcTEMP_SHORT: PROCEDURE (TEMP_REAL_PTR,SHORT_REAL_PTR)
  EXTERNAL;
  DECLARE (TEMP_REAL_PTR,SHORT_REAL_PTR) POINTER;
END mqcTEMP_SHORT;

DECLARE SHORT_REAL LITERALLY '0';
DECLARE LONG_REAL LITERALLY '2';
DECLARE TEMP_REAL LITERALLY '3';
DECLARE BIN_DECLOW_BLOCK STRUCTURE(
 BIN_PTR POINTER,
 BIN_TYPE BYTE,
 DEC_LENGTH BYTE,
 DEC_PTR POINTER,
 DEC_EXPONENT INTEGER,
 DEC_SIGN BYTE);

DECLARE DEC_BIN_BLOCK STRUCTURE (
 BIN_PTR POINTER,
 BIN_TYPE BYTE,
 DEC_LENGTH BYTE,
 DEC_PTR POINTER);
```

Stack Usage of DCON87 Procedures

DCON87 requires 176 bytes of the 8086 stack for its internal storage. This amount is allocated in the public STACK segment by the DCON87 modules.

All DCON87 procedures save the entire state of the 8087 upon entry and restore it upon exit.

The Accuracy of the Decimal Conversion Library

DCON87 guarantees that an 18-digit decimal number, when converted into TEMP__REAL and then back to decimal, will remain the same. However, DCON87 cannot guarantee that a TEMP__REAL number, when converted to decimal and then back to TEMP__REAL, will remain the same. Such accuracy would require arithmetic beyond the TEMP__REAL precision, which is too slow. The loss of accuracy is at worst less than 3 bits.

The IEEE Standard (described in Chapter 6) specifies the accuracy for decimal conversions of SHORT__REAL and LONG__REAL numbers. Since DCON87 uses TEMP__REAL arithmetic to perform its conversions, it exceeds the IEEE specifications.

Following is the range of SHORT__REAL and LONG__REAL numbers for which the IEEE standard specifies best possible conversion accuracy, for conversion both to and from the decimal format. The maximum number of decimal digits for best accuracy is given in the second column and the maximum magnitude for the decimal exponent is given in the third column. The exponents are given assuming the decimal point is to the right of the significand. For example, the largest number for totally accurate SHORT__REAL to decimal conversion is 999999999e13. The smallest positive number is 1e-13.

Conversion	**Significand**	**Exponent**
Decimal to SHORT__REAL	9 digits	13
Decimal to LONG__REAL	19 digits	27
SHORT__REAL to Decimal	9 digits	13
LONG__REAL to Decimal	17 digits	27

Following is a wider range of numbers for which the best possible conversion is not required. For numbers in the wider range, the IEEE standard allows an error of up to 0.47 times the unit in the last place.

Conversion	Significand	Exponent
Decimal to SHORT__REAL	9 digits	99
Decimal to LONG__REAL	19 digits	999
SHORT__REAL to Decimal	9 digits	54
LONG__REAL to Decimal	17 digits	341

Error Reporting in DCON87.LIB

DCON87 incorporates the 8087 error-reporting mechanism. Whenever an error is detected, an appropriate 8087 exception bit is set. Using 8087 instructions, you can set your choice of "unmasked" bits, and provide procedures which will be called whenever an unmasked exception is detected. This way, you do not have to test for errors every time you call a DCON procedure.

Whenever an unmasked exception is detected by any DCON87 procedure, it places certain quantities into 8087 storage before calling your specified trap handler.

- The opcode 00101101110 (16E hex) is placed into the 11-bit OPCODE register of the 8087. This is the opcode for the "FLDCW" instruction, which does not itself change the OPCODE register. You may therefore interpret the presence of the "FLDCW" opcode, together with an instruction address which is not all '1' bits, as meaning that the exception was discovered by DCON. (An all '1' bits instruction address signifies a CEL87 function. See Chapter 4.)
- The 20-bit absolute address of the DCON87 procedure which caused the exception is placed into the instruction address register of the 8087.
- The 20-bit absolute address of the input number is placed into the operand address register of the 8087. For mqcDEC__BIN and mqcDECLOW__BIN, this is a pointer into a DCON87 internal buffer on the 8086 stack, containing a string of decimal digits stripped for input to DECLOW__BIN. For all the other procedures, this is the pointer to the original input binary number.

An Alternate Method of DCON87 Error Detection

If you prefer to perform error testing yourself, an alternate method is possible. All DCON87 procedures return the 8087 exception byte in the AL register.

You can access the AL register in PL/M-86 programs by declaring the DCON87 procedures to be of type BYTE. For example, you could change the declaration of mqcBIN__DECLOW to:

```
mqcBIN_DECLOW: PROCEDURE (BLOCK_PTR) BYTE EXTERNAL;
  DECLARE BLOCK_PTR POINTER;
END mqcBIN_DECLOW;
```

The call to mqcBIN__DECLOW would then take the form:

```
STATUS = mqcBIN_DECLOW(@BIN_DECLOW_BLOCK);
```

After executing the above statement, you can test exception conditions by examining STATUS.

Format of Input Decimal Numbers to mqcDEC__BIN

The decimal number which is input to DEC__BIN takes the form of a string of consecutive ASCII bytes of memory, whose pointer is passed in the parameter block. The string may contain blanks after, but not before or within, the number.

The number may begin with a plus sign or a minus sign. Lack of a sign is equivalent to a plus sign.

After the optional sign is the significand. The significand is a sequence of one or more decimal digits, with an optional decimal point. The decimal point may appear before, within, or after the decimal digits. The significand must be present, with at least one decimal digit. The upper limit to the number of digits is determined by the limit of 255 bytes to the entire string; however, only the 20 most significant non-zero digits of a long significand will affect the output answer.

Following the significand is an optional exponent. This consists of one of the letters "E", "D", or "T", followed by an optional plus or minus sign, followed by a string of one or more decimal digits. The signed number indicates the power of ten which must be multiplied by the significand to yield the desired input number. The letters "E", "D", and "T" all have the same meaning (to identify the exponent). The letter can be in upper case or lower case.

Examples of Input Decimal Strings

The following strings all represent the number 100:

```
100
+100
100.
100.00
100.0000000000000000000006099216948
1E2
1T2
1d2
1.E2
.1E3
100000d-3
```

Here are some valid strings for other numbers:

```
3.14159265
-17
-34.25T-009
5.6E7
.001
-.4
-0
```

The following strings are invalid:

```
1 E2        ; embedded space not allowed
E2          ; missing significand
.E2         ; significand must contain at least one decimal digit
```

mqcBIN__DECLOW (Convert binary number to decimal string,low level interface)

Parameter: BIN__DECLOW__BLOCK__PTR, a double-word which points to a 13-byte block of memory. The block consists of six fields containing the following variables:

BIN__PTR is a double-word which points to the input binary number. The amount of memory space occupied by the number is determined by BIN__TYPE.

BIN__TYPE is a one-byte input code which indicates the type (SHORT__REAL, LONG__REAL, or TEMP__REAL) of the number pointed to by BIN__PTR. The codes for the types are 0, 2, and 3, respectively. The caller must insure that BIN__TYPE is set to the correct value—any other value will give an undefined result.

DEC__LENGTH is a one-byte input which indicates the number of decimal digits to be output at DEC__PTR. All one-byte values are legal—if zero is given, only the DEC__EXPONENT and SIGN are returned, with no string at DEC__PTR. A zero DEC__LENGTH combined with unusual inputs will yield an Invalid Result error, as described below.

DEC__PTR is a double-word which points to the memory location which will hold the output string of ASCII decimal digits. Other possible output values are shown in the table below DEC__SIGN.

DEC__EXPONENT is a word variable, output by mqcBIN__DECLOW. The decimal point is assumed to be to the right of the digits output at DEC__PTR. DEC__EXPONENT is then the power of ten which, when multiplied by the number at DEC__PTR, will yield the correct number. For NaN's and infinities, DEC__EXPONENT is 32767. For 0 and -0, DEC__EXPONENT is 0.

DEC__SIGN is a byte variable, output by mqcBIN__DECLOW. It is set to ASCII "+" (2B hex) if the input number is positive, and ASCII "-" (2D hex) if the number is negative. Other possible output values are shown below.

Decimal outputs for unusual input numbers:

Input number	DEC__SIGN	Decimal digits buffer
NaN	"."	"." followed by blanks
+ infinity	"+"	"+" followed by blanks
− infinity	"−"	"−" followed by blanks
+0	"0"	blanks
−0	"−"	"0" followed by blanks

Function: mqcBIN__DECLOW converts the floating point number of type BIN__TYPE, located at BIN__PTR, into a decimal representation stored in DEC__PTR, DEC__EXPONENT, and SIGN. The caller will use these quantities to construct the final output format of the number.

mqcBIN__DECLOW provides only 18 decimal digits. If DEC__LENGTH is greater than 18, the DEC__PTR buffer will be filled with 18 decimal digits, followed by ASCII underscore (5F hex) characters. The decimal point will be assumed to be to the right of the rightmost underscore. Thus, the underscore can be interpreted as "unknown decimal digit".

Errors: There are three possible errors for mqcBIN_DECLOW:

1. Invalid input. This error occurs only when the input DEC_LENGTH is given as 0, and the input number is one of: -0, any infinity, or NaN. In these cases, the empty decimal digits buffer which the caller requested would have contained information other than decimal digits. The outputs DEC_SIGN and DEC_EXPONENT are set appropriately. If the 8087 "I" exception is unmasked, the trap handler is called with the input number on the 8087 stack.

2. Denormalized input. This error occurs only when BIN_TYPE is TEMP_REAL, and the leading significand bit is zero. DEC_SIGN and DEC_EXPONENT are given the correct output values. The output decimal digit string is given leading zeroes to indicate the extent of denormalization. If the 8087 "D" exception is unmasked, the trap handler is called with the input number on the 8087 stack.

3. Inexact result. The correct answer is output. If the 8087 "P" exception is unmasked, the trap handler is called with the input number on the 8087 stack.

PL/M-86 usage example:

```
mqcBIN_DECLOW: PROCEDURE (BLOCK_PTR) EXTERNAL;
  DECLARE BLOCK_PTR POINTER;
END mqcBIN_DECLOW;

DECLARE REAL_VAR3 REAL;
DECLARE DIGITS_BUFFER(6) BYTE;
DECLARE FINAL_BUFFER(15) BYTE;

/* Assume that REAL_VAR3 contains the hex value
   C0E9A36D. */

BIN_DECLOW_BLOCK.BIN_PTR = @REAL_VAR3;
BIN_DECLOW_BLOCK.BIN_TYPE = SHORT_REAL;
BIN_DECLOW_BLOCK.DEC_LENGTH = 6;
BIN_DECLOW_BLOCK.DEC_PTR = @DIGITS_BUFFER;
CALL mqcBIN_DECLOW(@BIN_DECLOW_BLOCK);

/* The ''building blocks'' returned by mqcBIN_DECLOW are
   assembled into a six-digit scientific notation
   format */

FINAL_BUFFER(0) = BIN_DECLOW_BLOCK.DEC_SIGN;
FINAL_BUFFER(1) = DIGITS_BUFFER(0);
FINAL_BUFFER(2) = '.';
CALL MOVB( @DIGITS_BUFFER(1), @FINAL_BUFFER(3), 5 );
FINAL_BUFFER(8) = 'E';

/* Assume DECIMAL_WORD_OUT translates the first INTEGER
   parameter into a string of decimal digits placed at the
   second POINTER parameter */

CALL DECIMAL_WORD_OUT (BIN_DECLOW_BLOCK.DEC_EXPONENT + 5,
@FINAL_BUFFER(9));

/* FINAL_BUFFER now contains the string "-7.30120E0" */
```

ASM-86 usage example:

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs:
EXTRN mqcBIN_DECLOW: FAR

REAL_VAR3           DD            0C0E9A36DR
DIGITS_BUFFER_3     DB            6 DUP (?)
FINAL_BUFFER        DB            15 DUP (?)
BLOCK_3             BIN_DECLOW_BLOCK  < REAL_VAR3,
&                     SHORT_REAL,?,DIGITS_BUFFER_3,?,? >

     MOV BLOCK_3.DEC_LENGTH, 6    ; plug in the correct
                                  ; length
     MOV DX, SEG(BLOCK_3)         ; upper part of param
                                  ; address
     PUSH DX                      ; goes onto stack
     MOV DX, OFFSET(BLOCK_3)      ; lower part of param
                                  ; address
     PUSH DX                      ; goes onto stack
     CALL mqcBIN_DECLOW

; Now DIGITS_BUFFER contains the string "730120", DEC_SIGN
; equals "-", and DEC_EXPONENT equals -5. This
; information can be used to construct a number in any
; format desired.
```

mqcDEC_BIN (Convert decimal string to binary number)

Parameter: DEC_BIN_BLOCK_PTR, a double-word which points to a ten-byte block of memory. The block of memory contains four fields which hold the following variables:

BIN_PTR is a double-word which points to the output binary number. The amount of memory space occupied by the number is determined by BIN_TYPE.

BIN_TYPE is a one-byte input code which indicates the type (SHORT_REAL, LONG_REAL, or TEMP_REAL) of the number pointed to by BIN_PTR. The codes for the types are 0, 2, and 3, respectively. The caller must insure that BIN_TYPE is set to one of the indicated legal values—any other value will give an undefined result.

DEC_LENGTH is a one-byte input which indicates the length of the memory field pointed to by DEC_PTR. The length must be between 1 and 255; 0 will give an undefined result.

DEC_PTR is a double-word which points to the input string of ASCII decimal digits. The format of the input string is given earlier in this chapter. The caller should already have verified correct input syntax before calling mqcDEC_BIN.

Function: mqcDEC_BIN converts the string pointed to by DEC_PTR into a binary floating-point number of type BIN_TYPE, and leaves the number at the memory location pointed to by BIN_PTR.

Errors: There are six possible errors:

1. Illegal input. If the input string is not of the specified format, the output is undefined. No warning is issued.
2. Overflow beyond the TEMP_REAL format. If the 8087 "O" exception is masked, a correctly signed infinity is stored in the output buffer. If it is not masked, the 8087 trap handler is called with INDEFINITE on the 8087 stack, and nothing is written to the output buffer.
3. Overflow beyond the output format, but not the TEMP_REAL format. If the 8087 "O" exception is masked, the correctly signed infinity is stored in the output buffer. If the 8087 "O" exception is unmasked, the trap handler is called. In this case, the top of the 8087 stack contains the output number with the significand rounded to the output format's precision.
4. Underflow of the TEMP_REAL format. If the 8087 "U" exception is masked, then the output buffer contains the result of gradual underflow. If "U" is unmasked, INDEFINITE is left on the 8087 stack, the output buffer is unchanged and the 8087 trap handler is called.
5. Underflow of the output format, but not of the TEMP_REAL format. The output buffer contains the result of gradual underflow. If the 8087 "U" exception is unmasked, the trap handler is called. In this case, the top of the 8087 stack contains the output number with the significand rounded to the output format's precision.
6. An inexact result. The output buffer contains the result, rounded to the output format. If the 8087 "P" exception is unmasked, the trap handler is called with the TEMP_REAL result on the 8087 stack. Since this condition is not usually considered an error, the "P" exception is usually masked and the "P" error bit is ignored.

PL/M-86 usage example:

```
mqcDEC_BIN: PROCEDURE (BLOCK_PTR) EXTERNAL;
  DECLARE BLOCK_PTR POINTER;
END mqcDEC_BIN;

DECLARE REAL_VAR REAL;
DECLARE INPUT_BUFFER (120) BYTE;
DECLARE ACTUAL ADDRESS;

/* Assume that at this point a decimal string (e.g.
   "-3.4E2") has already been placed into INPUT_BUFFER,
   and a buffer length (6 for the above number) has been
   placed into ACTUAL */

DEC_BIN_BLOCK.BIN_PTR = @REAL_VAR;
DEC_BIN_BLOCK.BIN_TYPE = SHORT_REAL;
DEC_BIN_BLOCK.DEC_LENGTH = ACTUAL;
DEC_BIN_BLOCK.DEC_PTR = @INPUT_BUFFER;
CALL mqcDEC_BIN (@DEC_BIN_BLOCK);

/* REAL_VAR now equals the encoded SHORT_REAL value. For
   the sample string given, REAL_VAR = -340 */
```

ASM-86 usage example:

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs: EXTRN mqcDEC_BIN: FAR

INPUT_BUFFER DB           100 DUP (?)
ACTUAL       DW            0
REAL_VAR     DQ            0
BLOCK_1      DEC_BIN_BLOCK  < REAL_VAR, LONG_REAL, ?,
&              INPUT_BUFFER >

ENTRY:
  ; Assume that at this point a decimal string (e.g.
  ; "-3.4E2") has already been placed into INPUT_BUFFER,
  ; and a buffer length (6 for the above number) has been
  ; placed into ACTUAL.

     MOV AX, ACTUAL                ; get length of string
     MOV BLOCK_1.DEC_LENGTH, AL    ; place byte into param
                                   ; block
     MOV DX, SEG(BLOCK_1)          ; top half of param
                                   ; block ptr
     PUSH DX                       ; goes onto stack
     MOV DX, OFFSET(BLOCK_1)       ; bottom half of param
                                   ; block ptr
     PUSH DX                       ; goes onto stack
     CALL mqcDEC_BIN               ; convert string to
                                   ; binary

   ; REAL_VAR now equals the encoded LONG_REAL value.  For
   ; the sample given, REAL_VAR = -340.
```

mqcDECLOW_BIN (Convert decimal string, low-level interface, to binary number)

Parameter: DECLOW_BIN_BLOCK_PTR, a double-word which points to a 13-byte block of memory. The block of memory contains six fields which hold the following variables:

BIN_PTR is a double-word which points to the output binary number. The amount of memory space occupied by the number is determined by BIN_TYPE.

BIN_TYPE is a one-byte input code which indicates the type (SHORT_REAL, LONG_REAL, or TEMP_REAL) of the number pointed to by BIN_PTR. The codes for the types are 0, 2, and 3, respectively. The caller must insure that BIN_TYPE is set to one of the indicated legal values—any other value will give an undefined result.

DEC_LENGTH is a one-byte input which indicates the length of the memory field pointed to by DEC_PTR. The length must be between 1 and 21, inclusive. Any other value gives an undefined result.

DEC_PTR is a double-word which points to the input string of ASCII decimal digits. The digits give the significand of the number to be converted. Leading zeroes, the sign, the decimal point, the exponent, and lagging zeroes have already been removed.

If DEC_LENGTH equals 21, then there were more than 20 digits in the original significand. In this case, the given significand consists of the first 20 digits only. The twenty-first digit is the least significant non-zero digit of the original significand.

DEC_EXPONENT is a word which gives the base 10 exponent of the number to be converted. It is assumed that the decimal point has been shifted to the immediate right of the last digit of the significand. DEC_EXPONENT is then the power of ten which, when multiplied by the significand, yields the number to be converted. Negative powers are represented in two's-complement form.

DEC_SIGN is a byte which gives the sign of the number to be converted. The possible values are ASCII "+" (2B hex) and ASCII "−" (2D hex).

Function: mqcDECLOW_BIN accepts a decimal number which has already been converted from a higher-level format to the lower-level format described under DEC_PTR, DEC_EXPONENT, and DEC_SIGN above. It converts the number into a binary floating-point number of type BIN_TYPE, and leaves the number at the memory location pointed to by BIN_PTR.

Errors: All errors (invalid input, overflow, underflow, and inexact result) are the same as for mqcDEC_BIN, and have the same consequences.

PL/M-86 usage example:

```
mqcDECLOW_BIN: PROCEDURE (BLOCK_PTR) EXTERNAL;
  DECLARE BLOCK_PTR POINTER;
END mqcDECLOW_BIN;

DECLARE REAL_VAR2 REAL;
DECLARE N_DIGITS BYTE;
DECLARE DIGITS_BUFFER(21) BYTE;
```

```
/* Assume that at this point the string "371" has been
   placed into DIGITS_BUFFER, and N_DIGITS has been set to
   the string length 3 */

/* The following code assumes DIGIT_BUFFER contains a
   decimal string representing an integer number of
   pennies. It places the number of dollars, in SHORT_REAL
   binary format, into REAL_VAR2. */

BIN_DECLOW_BLOCK.BIN_PTR = @REAL_VAR2;
BIN_DECLOW_BLOCK.BIN_TYPE = SHORT_REAL;
BIN_DECLOW_BLOCK.DEC_LENGTH = N_DIGITS;
BIN_DECLOW_BLOCK.DEC_PTR = @DIGITS_BUFFER;
BIN_DECLOW_BLOCK.DEC_EXPONENT = -2;
BIN_DECLOW_BLOCK.DEC_SIGN = '+';
CALL mqcDECLOW_BIN(@BIN_DECLOW_BLOCK);

/* REAL_VAR2 now equals the encoded SHORT_REAL value
   3.71 */
```

ASM-86 usage example:

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs:
EXTRN mqcDECLOW_BIN: FAR

N_DIGITS        DB              0
DIGITS_BUFFER   DB              21 DUP (?)
REAL_VAR2       DQ              ?
BLOCK_2         DECLOW_BIN_BLOCK < REAL_VAR2,
&                 LONG_REAL,?,DIGITS_BUFFER,-2,'+'>

; assume that the string ''371'' has been placed into
; DIGITS_BUFFER, and N_DIGITS has been set to the string
; length 3.

        MOV AL,N_DIGITS                ; load string length
        MOV BLOCK_2.DEC_LENGTH, AL     ; place into param
                                       ; block
        MOV DX, SEG(BLOCK_2)           ; top half of param
                                       ; block ptr
        PUSH DX                        ; goes onto stack
        MOV DX, OFFSET(BLOCK_2)        ; bottom half of
                                       ; param block ptr
        PUSH DX                        ; goes onto stack
        CALL mqcDECLOW_BIN             ; convert string to
                                       ; binary

; The encoded LONG_REAL value 3.71 has now been placed
; into REAL_VAR2.
```

mqcLONG_TEMP (Convert LONG_REAL to TEMP_REAL)

Parameters: LONG_REAL_PTR points to the eight bytes of memory which hold the input LONG_REAL binary floating point number.

TEMP_REAL_PTR points to the ten bytes of memory into which the TEMP_REAL answer is placed.

Function: The input LONG_REAL number is extended to the TEMP_REAL number which represents the same value. Unusual inputs (NaNs, infinities, −0) translate to the same unusual outputs, with zeroes padding the end of the output significand.

Errors: The only possible error is a denormalized input value. The corresponding denormalized output value is left in the output buffer. If the 8087 "D" exception is unmasked, the trap handler is called with the output on the 8087 stack.

PL/M-86 usage example:

```
mqcLONG_TEMP: PROCEDURE (LONG_REAL_PTR,TEMP_REAL_PTR)
              EXTERNAL;
  DECLARE (LONG_REAL_PTR,TEMP_REAL_PTR) POINTER;
END mqcLONG_TEMP;

DECLARE TEMP_REAL_ARRAY (8) STRUCTURE
       (SIGNIFICAND(8) BYTE, EXPO_SIGN(2) INTEGER);
DECLARE LONG_REAL_ARRAY (8) STRUCTURE (NUM(8) BYTE);
DECLARE I BYTE;

/* The following code expands an array of LONG_REAL
   variables into an array of TEMP_REAL variables */

  DO I = 0 TO 7;
   CALL mqcLONG_TEMP ( @LONG_REAL_ARRAY(I),
   @TEMP_REAL_ARRAY(I) );
  END;
```

ASM-86 usage example:

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs:
EXTRN mqcLONG_TEMP: FAR

TEMP_NUM  DT  ?
LONG_NUM DD  ?
```

```
; The following code transfers the LONG_REAL number at
; LONG_NUM to the TEMP_REAL number at TEMP_NUM

    MOV DX, SEG(LONG_NUM)          ; top half of parameter
                                   ; pointer 1
    PUSH DX                        ; goes onto stack
    MOV DX, OFFSET(LONG_NUM)       ; bottom half of
                                   ; parameter pointer 1
    PUSH DX                        ; goes onto stack
    MOV DX, SEG(TEMP_NUM)          ; top half of parameter
                                   ; pointer 2
    PUSH DX                        ; goes onto stack
    MOV DX, OFFSET(TEMP_NUM)       ; bottom half of
                                   ; parameter pointer 2
    PUSH DX                        ; goes onto stack
    CALL mqcLONG_TEMP
```

mqcSHORT_TEMP (Convert SHORT_REAL to TEMP_REAL)

Parameters: SHORT_REAL_PTR points to the four bytes of memory which hold the input SHORT_REAL binary floating point number.

TEMP_REAL_PTR points to the ten bytes of memory into which the TEMP_REAL answer is placed.

Function: The input SHORT_REAL number is extended to the TEMP_REAL number which represents the same value. Unusual inputs (NaNs, infinities, −0) translate to the same unusual outputs, with zeroes padding the end of the output significand.

Errors: The only possible error is a denormalized input value. The corresponding denormalized output value is left in the output buffer. If the 8087 "D" exception is unmasked, the trap handler is called with the output on the 8087 stack.

PL/M-86 usage example:

```
mqcSHORT_TEMP: PROCEDURE (SHORT_REAL_PTR,TEMP_REAL_PTR)
               EXTERNAL;
  DECLARE (SHORT_REAL_PTR,TEMP_REAL_PTR) POINTER;
END mqcSHORT_TEMP;

DECLARE TEMP_REAL_ARRAY (8) STRUCTURE (NUM(10) BYTE);
DECLARE SHORT_REAL_ARRAY (8) REAL;
DECLARE I BYTE;

/* The following loop expands an array of SHORT_REAL
   variables into an array of TEMP_REAL variables. */

  DO I = 0 TO 7;
    CALL mqcSHORT_TEMP ( @SHORT_REAL_ARRAY(I),
    @TEMP_REAL_ARRAY(I) );
  END;
```

ASM-86 usage example:

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs:
EXTRN mqcSHORT_TEMP: FAR

SHORT_X  DD  ?
TEMP_X   DD  ?
```

```
; The following code converts the SHORT_REAL number at
; SHORT_X into a TEMP_REAL value, and leaves it at TEMP_X.

; It is assumed that both numbers are in the DS segment.

        PUSH DS                    ; top half of first
                                   ; parameter
        MOV AX, OFFSET(SHORT_X)    ; bottom half of first
                                   ; parameter
        PUSH AX                    ; pushed onto stack
        PUSH DS                    ; top half of second
                                   ; parameter
        MOV AX, OFFSET(TEMP_X)     ; bottom half of second
                                   ; parameter
        PUSH AX                    ; pushed onto stack
        CALL mqcSHORT_TEMP         ; conversion is now
                                   ; complete
```

mqcTEMP_LONG (Convert TEMP_REAL to LONG_REAL)

Parameters: TEMP_REAL_PTR points to the ten bytes of memory which hold the input TEMP_REAL binary floating point number.

LONG_REAL_PTR points to the eight bytes of memory into which the LONG_REAL answer is placed.

Function: The TEMP_REAL number is rounded to the nearest LONG_REAL number. If there are two LONG_REAL numbers equally near, the one with a zero least significant bit is chosen.

Errors: There are five possible errors:

1. Overflow. If the exponent of the TEMP_REAL input exceeds the output capacity, the correctly signed infinity is placed in the output buffer. If the 8087 "O" exception is unmasked, the trap handler is called. In this case, the number left on the 8087 stack is the input number with the exponent unchanged and the significand rounded to the output's precision.
2. Underflow. The output buffer is filled with the result of gradual underflow, rounded to fit the output format. If the 8087 "U" exception is unmasked, the trap handler is called. In this case, the 8087 stack is left with the input number's significand rounded to the output precision; with the too-small exponent unchanged.
3. Inexact. This occurs when the input significand has non-zero bits outside the field of the output format. The correctly rounded answer is left in the output buffer. If the 8087 "P" exception is unmasked, the trap handler is called with the input number on the 8087 stack.
4. Unnormalized invalid input. This occurs when the input is unnormalized, there is no underflow and rounding to the output precision does not normalize the answer. INDEFINITE is left in the output buffer. If the 8087 "I" exception is unmasked, the trap handler is called with the input number on the 8087 stack.
5. Invalid nonzero NaN truncation. The only NaN inputs faithfully represented in the smaller format are those for which the significand bits being truncated are all zero. For other NaNs, invalid operation is signalled. The output buffer is filled with the truncated result only if the truncated significand is non-zero. If the significand would be zero, it is made non-zero by substituting the highest nonzero byte of the original significand into bits 5 through 12 of the new LONG_REAL significand. If the 8087 "I" exception is unmasked, the trap handler is called with the input number on the 8087 stack.

PL/M-86 usage example:

```
mqcTEMP_LONG: PROCEDURE (TEMP_REAL_PTR,LONG_REAL_PTR)
              EXTERNAL;
  DECLARE (TEMP_REAL_PTR,LONG_REAL_PTR) POINTER;
END mqcTEMP_LONG;

DECLARE TEMP_REAL_ARRAY (8) STRUCTURE (NUM(10) BYTE);
DECLARE LONG_REAL_ARRAY (8) STRUCTURE (NUM(8) BYTE);
DECLARE I BYTE;
```

```
/* The following code transfers an array of 8 TEMP_REAL
   numbers to an array of LONG_REAL numbers. */

 DO I = 0 TO 7;
  CALL mqcTEMP_LONG(@TEMP_REAL_ARRAY(I),
  @LONG_REAL_ARRAY(I)) ;
 END;
```

ASM-86 usage example:

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs:
EXTRN mqcTEMP_LONG: FAR

TEMP_REAL_ARRAY   DT    8 DUP (?)
LONG_REAL_ARRAY   DQ    8 DUP (?)

; The following code transfers the array of 8 TEMP_REAL
; numbers in TEMP_REAL_ARRAY to the 8 LONG_REAL numbers in
; LONG_REAL_ARRAY.

; It is assumed that both arrays are in the current DS
; segment.

     PUSH BP                          ; BP should always
                                      ; be preserved
     MOV BP, OFFSET(LONG_REAL_ARRAY)
     MOV DX, OFFSET(TEMP_REAL_ARRAY)  ; DX will not be
                                      ; preserved
     MOV CX, 8                        ; initialize loop
                                      ; counter
LOOP8:
     PUSH CX                          ; preserve counter
     PUSH DX                          ; preserve TEMP_REAL
                                      ; pointer
     PUSH DS                          ; top half of TEMP
                                      ; parameter
     PUSH DX                          ; bottom half of
                                      ; TEMP parameter
     PUSH DS                          ; top half of LONG
                                      ; parameter
     PUSH BP                          ; bottom half of
                                      ; LONG parameter
     CALL mqcTEMP_LONG                ; perform conversion
     POP DX                           ; restore TEMP_REAL
                                      ; pointer
     ADD DX, 10                       ; increment to next
                                      ; TEMP_REAL
     ADD BP, 8                        ; increment to next
                                      ; LONG_REAL
     POP CX                           ; restore counter
     DEC CX                           ; count down
     JNZ LOOP8                        ; loop back if more
                                      ; conversions
     POP BP                           ; restore the
                                      ; original BP
```

mqcTEMP__SHORT (Convert TEMP__REAL to SHORT__REAL)

Parameters: TEMP__REAL__PTR points to the ten bytes of memory which hold the input TEMP__REAL binary floating point number.

SHORT__REAL__PTR points to the four bytes of memory into which the SHORT__REAL answer is placed.

Function: The TEMP__REAL number is rounded to the nearest SHORT__REAL number. If there are two SHORT__REAL numbers equally near, the one with a zero least significant bit is chosen.

Errors: The errors (overflow, underflow, inexact, invalid unnormalized, and invalid nonzero NaN truncation) are the same as for mqcTEMP__LONG, with the same actions taken, except for the subcase of error 5 in which the highest nonzero byte of the original significand must be copied to the result significand to preserve its character as a NaN. In this case, that byte occupies bits 0 through 7 of the new SHORT__REAL significand.

PL/M-86 usage example:

```
mqcTEMP_SHORT: PROCEDURE (TEMP_REAL_PTR,SHORT_REAL_PTR)
               EXTERNAL;
  DECLARE (TEMP_REAL_PTR,SHORT_REAL_PTR) POINTER;
END mqcTEMP_SHORT;

DECLARE TEMP_REAL_ARRAY (8) STRUCTURE
       (SIGNIFICAND(8) BYTE, EXPO_SIGN(2) INTEGER);
DECLARE SHORT_REAL_ARRAY (8) REAL;
DECLARE I BYTE;

/* The following loop transfers an array of 8 TEMP_REAL
   numbers (which could, for example, represent an image
   of the 8087 stack) to a more compact array of
   SHORT_REAL numbers. */

  DO I = 0 TO 7;
    CALL mqcTEMP_SHORT ( @TEMP_REAL_ARRAY(I),
    @SHORT_REAL_ARRAY(I) );
  END;
```

ASM-86 usage example:

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs:
EXTRN mqcTEMP_SHORT: FAR

TEMP_VAR  DT  ?
SHORT_VAR DD  ?
```

```
; The following code transfers the TEMP_REAL number at
; TEMP_VAR to the SHORT_REAL number at SHORT_VAR

    MOV DX, SEG(TEMP_VAR)          ; top half of parameter
                                   ; pointer
    PUSH DX                        ; goes onto stack
    MOV DX, OFFSET(TEMP_VAR)       ; bottom half of
                                   ; parameter pointer
    PUSH DX                        ; goes onto stack
    MOV DX, SEG(SHORT_VAR)         ; top half of parameter
                                   ; pointer
    PUSH DX                        ; goes onto stack
    MOV DX, OFFSET(SHORT_VAR)      ; bottom half of
                                   ; parameter pointer
    PUSH DX                        ; goes onto stack
    CALL mqcTEMP_SHORT
```

Linkage of DCON87.LIB to Your Program Modules

The final action you must take to use DCON87.LIB in your programs is to insert the file name DCON87.LIB into a LINK86 command.

DCON87 requires either the 8087 component or the 8087 emulator. If the component is present, you must also link in 8087.LIB. If the emulator is used, you must link in both E8087 and E8087.LIB.

Following is the suggested order for the object modules in your LINK86 statement.

Your object modules
DCON87.LIB
CEL87.LIB if you are using it
EH87.LIB if you are using it
8087.LIB if you are using the component, or
E8087, E8087.LIB if you are using the emulator.

As an example, if you are linking your PL/M-86 modules MYMOD1.OBJ and MYMOD2.OBJ into a program using the 8087 emulator, issue the command

```
-LINK86 :F1:MYMOD1.OBJ, :F1:MYMOD2.OBJ, &<cr>
>> :F0:DCON87.LIB, :F0:E8087, :F0:E8087.LIB &<cr>
>> TO :F1:MYPROG.LNK<cr>
```

If you have a single ASM-86 generated object module :F1:MYPROG.OBJ to be executed in a system with an 8087 component, issue the command

```
-LINK86 :F1:MYPROG.OBJ, :F0:DCON87.LIB, :F0:8087.LIB &<cr>
>> TO :F1:MYPROG.LNK<cr>
```

CHAPTER 4 THE COMMON ELEMENTARY FUNCTION LIBRARY

Overview

This chapter describes the functions of CEL87.LIB, a library that contains commonly-used functions of floating point arithmetic.

The 8087 chip gives direct support for the most time-consuming part of the computation of every CEL87 function; but the forms of the functions and the ranges of the inputs are limited. CEL87 provides the software support necessary to let the 8087 provide a complete package of elementary functions, giving valid results for all appropriate inputs.

Following is a summary of CEL87 functions, grouped by functionality.

Rounding and Truncation Functions:

mqerIEX rounds a real number to the nearest integer; to the even integer if there is a tie. The answer returned is real.
mqerIE2 is as mqerIEX, except the answer is a 16-bit integer.
mqerIE4 is as mqerIEX, except the answer is a 32-bit integer.
mqerIAX rounds a real number to the nearest integer; to the integer away from zero if there is a tie. The answer returned is real.
mqerIA2 is as mqerIAX, except the answer is a 16-bit integer.
mqerIA4 is as mqerIAX, except the answer is a 32-bit integer.
mqerICX chops the fractional part of a real input. The answer is real.
mqerIC2 is as mqerICX, except the answer is a 16-bit integer.
mqerIC4 is as mqerICX, except the answer is a 32-bit integer.

Logarithmic and Exponential Functions:

mqerLGD computes decimal (base 10) logarithms.
mqerLGE computes natural (base e) lograithms.
mqerEXP computes exponentials of base e.
mqerY2X computes exponentials of any base.
mqerYI2 computes an input real to the AX integer power.
mqerYI4 is as mqerYI2, except the input integer is DXAX.
mqerYIS is as mqerYI2, except the input integer is on the 8086 stack.

Trigonometric and Hyperbolic Functions:

mqerSIN, mqerCOS, mqerTAN compute sine, cosine, and tangent.
mqerASN, mqerACS, mqerATN compute the corresponding inverse functions.
mqerSNH, mqerCSH, mqerTNH compute the corresponding hyperbolic functions.
mqerAT2 is a special version of the arc tangent function. It handles all the quadrants for rectangular-to-polar conversion in two dimensions.

Other Functions:

mqerDIM is FORTRAN's positive difference function.
mqerMAX computes the maximum of two real inputs.
mqerMIN computes the minimum of two real inputs.
mqerSGH combines the sign of one input with the magnitude of the other input.
mqerMOD computes a modulus, retaining the sign of the dividend.
mqerRMD computes a modulus, giving the value closest to zero.

All CEL87 procedure names being with the four letters "mqer". We have added this prefix to reduce the possibility that a CEL87 name will conflict with another name in your program. To make the names more readable, we have put the "mqer" prefix in lower case letters throughout this chapter.

Declaring CEL87 Procedures in Your ASM-86 Programs

In each source program module that calls a CEL87 procedure, you must declare that procedure as external before it is used.

Following are declarations you should make at the top of your ASM-86 program. You should include only those procedures you use.

To make reference easier, we have duplicated the EXTRN statements in the examples for the individual functions. You should not duplicate the EXTRN statements in your program.

All CEL87 procedures must be declared FAR.

```
; All of the following EXTRN statements should appear
; outside of all SEGMENT-ENDS pairs

EXTRN mqerACS: FAR
EXTRN mqerASN: FAR
EXTRN mqerAT2: FAR
EXTRN mqerATN: FAR
EXTRN mqerCOS: FAR
EXTRN mqerCSH: FAR
EXTRN mqerDIM: FAR
EXTRN mqerEXP: FAR
EXTRN mqerIA2: FAR
EXTRN mqerIA4: FAR
EXTRN mqerIAX: FAR
EXTRN mqerIC2: FAR
EXTRN mqerIC4: FAR
EXTRN mqerICX: FAR
EXTRN mqerIE2: FAR
EXTRN mqerIE4: FAR
EXTRN mqerIEX: FAR
EXTRN mqerLGD: FAR
EXTRN mqerLGE: FAR
EXTRN mqerMAX: FAR
EXTRN mqerMIN: FAR
EXTRN mqerMOD: FAR
EXTRN mqerRMD: FAR
EXTRN mqerSGN: FAR
EXTRN mqerSIN: FAR
EXTRN mqerSNH: FAR
EXTRN mqerTAN: FAR
EXTRN mqerTNH: FAR
EXTRN mqerY2X: FAR
EXTRN mqerYI2: FAR
EXTRN mqerYI4: FAR
EXTRN mqerYIS: FAR
```

Declaring CEL87 Procedures in Your PL/M-86 Programs

Following are the declarations you should make at the top of your PL/M-86 programs. You should include only those procedures you use.

To make reference easier, we have duplicated the procedure declarations in the examples for the individual functions. You should not duplicate the declarations in your programs.

To use CEL87.LIB, you must compile your PL/M-86 modules under the MEDIUM or LARGE models of computation, since CEL87's procedures are FAR procedures.

```
mqerACS: PROCEDURE (X) REAL EXTERNAL;
  DECLARE X REAL;
END mqerACS;

mqerASN: PROCEDURE (X) REAL EXTERNAL;
  DECLARE X REAL;
END mqerASN;

mqerAT2: PROCEDURE (Y,X) REAL EXTERNAL;
  DECLARE (Y,X) REAL;
END mqerAT2;

mqerATN: PROCEDURE (X) REAL EXTERNAL;
  DECLARE X REAL;
END mqerATN;

mqerCOS: PROCEDURE (THETA) REAL EXTERNAL;
  DECLARE THETA REAL;
END mqerCOS;

mqerCSH: PROCEDURE (THETA) REAL EXTERNAL;
  DECLARE THETA REAL;
END mqerCSH;

mqerDIM: PROCEDURE (Y,X) REAL EXTERNAL;
  DECLARE (Y,X) REAL;
END mqerDIM;

mqerEXP: PROCEDURE (X) REAL EXTERNAL;
  DECLARE X REAL;
END mqerEXP;

mqerIA2: PROCEDURE (X) INTEGER EXTERNAL;
  DECLARE X REAL;
END mqerIA2;

mqerIAX: PROCEDURE (X) REAL EXTERNAL;
  DECLARE X REAL;
END mqerIAX;

mqerIC2: PROCEDURE (X) INTEGER EXTERNAL;
  DECLARE X REAL;
END mqerIC2;

mqerICX: PROCEDURE (X) REAL EXTERNAL;
  DECLARE X REAL;
END mqerICX;
```

```
mqerIE2: PROCEDURE (X) INTEGER EXTERNAL;
  DECLARE X REAL;
END mqerIE2;

mqerIEX: PROCEDURE (X) REAL EXTERNAL;
  DECLARE X REAL;
END mqerIEX;

mqerLGD: PROCEDURE (X) REAL EXTERNAL;
  DECLARE X REAL;
END mqerLGD;

mqerLGE: PROCEDURE (X) REAL EXTERNAL;
  DECLARE X REAL;
END mqerLGE;

mqerMAX: PROCEDURE (Y,X) REAL EXTERNAL;
  DECLARE (Y,X) REAL;
END mqerMAX;

mqerMIN: PROCEDURE (Y,X) REAL EXTERNAL;
  DECLARE (Y,X) REAL;
END mqerMIN;

mqerMOD: PROCEDURE (Y,X) REAL EXTERNAL;
  DECLARE (Y,X) REAL;
END mqerMOD;

mqerRMD: PROCEDURE (Y,X) REAL EXTERNAL;
  DECLARE (Y,X) REAL;
END mqerRMD;

mqerSGN: PROCEDURE (Y,X) REAL EXTERNAL;
  DECLARE (Y,X) REAL;
END mqerSGN;

mqerSIN: PROCEDURE (THETA) REAL EXTERNAL;
  DECLARE THETA REAL;
END mqerSIN;

mqerSNH: PROCEDURE (THETA) REAL EXTERNAL;
  DECLARE THETA REAL;
END mqerSNH;

mqerTAN: PROCEDURE (THETA) REAL EXTERNAL;
  DECLARE THETA REAL;
END mqerTAN;

mqerTNH: PROCEDURE (THETA) REAL EXTERNAL;
  DECLARE THETA REAL;
END mqerTNH;

mqerY2X: PROCEDURE (Y,X) REAL EXTERNAL;
  DECLARE (Y,X) REAL;
END mqerY2X;

mqerYIS: PROCEDURE (Y,I) REAL EXTERNAL;
  DECLARE Y REAL, I INTEGER;
END mqerYIS;
```

How to Call CEL87 Functions in Your PL/M-86 Programs

CEL87 functions operate with numbers on the 8087 stack; however, this fact is invisible to you when you are programming in PL/M-86. There are no PL/M statements which explicitly load and unload numbers from the 8087 stack. The code for this is implicitly generated whenever arithmetic involving REAL numbers is programmed.

To invoke a CEL87 function after you have declared it, simply use the function in an assignment statement (or in an expression in any other context). All CEL87 functions, if declared as shown in the examples, will cause the PL/M compiler to load the input parameters when CEL87 is invoked, and to place the answer in the correct destination after the CEL87 procedure is complete.

PL/M-86 does not support the formats LONG_REAL and TEMP_REAL. Numeric constants and variables used by PL/M are 32-bit SHORT_REAL numbers, in 8086 memory, which are converted to TEMP_REAL when placed on the 8087 stack. When PL/M stores the result of a CEL87 function that returns a value on the 8087 stack, the function is converted from TEMP_REAL to SHORT_REAL. The conversion takes place with an FSTP instruction generated by the compiler.

It is possible for the FSTP instruction to generate an "O" overflow or a "U" underflow error. (Indeed, because TEMP_REAL is such a large format, this is where "O" and "U" errors are most likely to occur.) These errors are beyond the control of the CEL87 function; hence the outlines given in this chapter of what values are placed into 8087 registers do not apply for these errors.

Consult the *8086 Family User's Manual Numerics Supplement* for the error specifications of the FSTP instruction.

Stack Usage of CEL87 Procedures

CEL87 requires 50 bytes of the 8086 stack for its internal storage. If, however, there are situations in which CEL87 is interrupted by a program which itself calls CEL87 (for example, an error trap handler), then an additional 50 bytes of 8086 stack are needed for each recursion level possible. CEL87 declares a 50-byte STACK segment within its program modules, hence the first 50 bytes will be automatically allocated by LINK86 and LOC86.

CEL87 requires 4 numbers on the 8087 stack. Input parameters are counted in the four numbers. This means that if a CEL87 function has input parameters at the top two stack positions ST(0) and ST(1), the parameters will be destroyed by the function. Also, the numbers ST(6) and ST(7) must be empty for the function to execute correctly. If the 8087 stack overflows during a CEL87 function, the results are undefined.

All CEL87 functions are reentrant in that they use no fixed 8086 memory locations to store internal variables. However, since the 8087 stack contains only eight elements, it is advisable for an interrupt routine to explicitly preserve the 8087 stack upon entry and to restore it upon exit. Any procedure that interrupts a CEL87 function must save the entire 8087 stack, since it is impossible to tell which 4 stack elements are being used by CEL87.

Register Usage of CEL87 Procedures

CEL87 conforms to the register-saving conventions of all 8086 high level languages provided by Intel.

According to these conventions, the 8086 registers that can be destroyed by CEL87 functions are AX, BX, CX, DX, DI, SI, and ES. The 8086 registers that are unchanged by CEL87 functions are BP, SS, and DS. The SP register is changed only by mqerYIS, which accepts a parameter on the 8086 stack and returns with the parameter popped from the stack.

All CEL87 functions save the 8087 Control Word upon entry, and restore it upon return. During computation, all CEL87 functions except mqerDIM use TEMP__REAL precision, and a rounding mode of CEL87's choosing. mqerDIM uses the precision and rounding in effect at the time mqerDIM is called.

The 8087 exception flags (error bits) are treated by all CEL87 functions as "sticky". This means that the only change to error bits that CEL87 may make is from off to on. If an error bit is already on when CEL87 is called, it will stay on. This allows you to perform a sequence of 8087 calculations that involve CEL87 functions with some of the 8087 errors masked. At the end of the calculation, if the exception flag for a masked error is still zero, you can be sure that the error was never encountered.

Error Reporting in CEL87.LIB

CEL87 incorporates the 8087 error-reporting mechanism. Whenever an error is detected, an appropriate 8087 exception bit is set. Using 8087 instructions, you can set your choice of "unmasked" bits, and provide procedures that will be called whenever an unmasked exception is detected. This way, you do not have to test for errors every time you invoke a CEL87 function.

There are only 7 CEL87 functions which leave a meaningful value in the "P" precision error bit. These are mqerIC2, mqerIC4, mqerICX, mqerIE2, mqerIE4, mqerIEX, and mqerSGN. For all other CEL87 functions, the "P" exception should be masked, and the "P" exception bit returned should be ignored.

Whenever an unmasked exception is detected by any CEL87 function, the 8087 calls the trap handler. The values on the 8087 stack are decribed under each individual CEL87 function. In addition, CEL87 sets the following values whenever the trap handler is called:

- The return address from the CEL87 function, converted to 20 bits, is placed in the operand register of the 8087.
- The 8087 control word is restored to the value it had when the CEL87 function was called.
- The 8087 instruction address register is set to 0FFFFF hex.
- The 8087 opcode register is set to the code described under each individual CEL87 function. The description gives the code in the form of three hexadecimal digits. The first (most significant) digit gives the number of values remaining on the 8087 stack that relate to the CEL87 function that caused the trap. The bottom two digits identify the CEL87 function.

The trap handler can identify an interrupt as generated by CEL87 by examining the 8087 instruction address register to see that it is set to 0FFFFF hex. If it is, then the identity of the CEL87 function is determined by examining the 8087 opcode register.

mqerACS x = arc cosine(x)

Input parameter: (x) is the top number on the 8087 stack.

Function: mqerACS returns the number, in radians, whose cosine is equal to the input value. Results are in the range 0 to π. Valid inputs are from −1 to 1, inclusive. All zeroes, pseudo-zeroes, and denormals return the value π/2. Also, unnormals less than 2^{-63} return the value π/2.

Output: The answer replaces the input on the 8087 stack.

Errors: Since cosines exist only in the range −1 to 1, inclusive, all inputs to mqerACS outside of this range are invalid. Thus an "I" error is given for infinities, NaNs, values less than −1, and values greater than 1. An "I" error is given also for unnormals not less than 2^{-63}.

If "I" is unmasked, the trap handler is called with the input still on the stack, and the 8087 opcode register set to 175 hex. If "I" is masked, the answer is the input for NaNs; the answer is the value INDEFINITE for other invalid inputs.

PL/M-86 usage example:

```
mqerACS: PROCEDURE (X) REAL EXTERNAL;
  DECLARE X REAL;
END mqerACS;

DECLARE HYPOTENUSE REAL;  /* Longest side of a right
                             triangle */
DECLARE ADJACENT_SIDE REAL;  /* The other side, next to
                                angle computed */
DECLARE (THETA_RADIANS, THETA_DEGREES) REAL;  /* Two forms
                                                 of the
                                                 answer */
DECLARE PI LITERALLY '3.14159265358979';

HYPOTENUSE = 10.; ADJACENT_SIDE = 5.;  /* Test values */

/* The following lines calculate the value of an angle of
   a right triangle, given the length of the hypotenuse
   and the adjacent side. mqerACS returns the value in
   radians; the value is then converted to degrees. */

THETA_RADIANS = mqerACS( ADJACENT_SIDE / HYPOTENUSE);
THETA_DEGREES = (180./PI) * THETA_RADIANS;

/* Now THETA_DEGREES = 60 -- it is a 30-60-90 degree
   triangle */
```

ASM-86 usage example:

```
; This EXTRN must appear outside of all SEGMENT-ENDS
;pairs:
EXTRN mqerACS: FAR
```

```
HYPOTENUSE     DQ  10.0      ; longest side of a right
                             ; triangle
ADJACENT_SIDE  DQ  5.0       ; the other side, next to
                             ; angle computed
                   ; The above initial values are test
                   ; values
THETA_RADIANS  DQ  ?
THETA_DEGREES  DQ  ?
RAD_TO_DEG     DT  4004E52EE0D31E0FBDC3R  ; the constant
                                          ; 180/PI

  ; The following lines compute the angle of a right
  ; triangle, just as in the above PL/M example, except
  ; with LONG_REAL variables.

      FLD ADJACENT_SIDE      ; (x) = ADJACENT_SIDE
      FDIV HYPOTENUSE        ; (x) = ADJACENT_SIDE/
                             ; HYPOTENUSE
      CALL mqerACS           ; angle is now in (x)
      FST THETA_RADIANS      ; radians result is saved
      FLD RAD_TO_DEG
      FMUL                   ; (x) converted to degrees
      FSTP THETA_DEGREES     ; degrees are saved, 8087
                              ; stack popped

        ; Now THETA_DEGREES = 60 -- It is a 30-60-90
        ; triangle.
```

mqerASN x = arc sine(x)

Input parameter: (x) is the top number on the 8087 stack.

Function: mqerASN returns the number, in radians, whose sine is equal to the input value. Results are in the range $-\pi/2$ to $\pi/2$. Valid inputs are from −1 to 1, inclusive. All zeroes, pseudo-zeroes, and denormals return the input value. Also, unnormals less than 2^{-63} return the input value.

Output: The answer replaces the input on the 8087 stack.

Errors: Since sines exist only in the range −1 to 1, inclusive, all inputs to mqerASN outside of this range are invalid. Thus an "I" error is given for infinities, NaNs, values less than −1, and values greater than 1. An "I" error is given also for unnormals not less than 2^{-63}.

If "I" is unmasked, the trap handler is called with the input still on the stack, and the 8087 opcode register set to 174 hex. If "I" is masked, the answer is the input for NaNs; the answer is the value INDEFINITE for other invalid inputs.

PL/M-86 usage example:

```
mqerASN: PROCEDURE (X) REAL EXTERNAL;
  DECLARE X REAL;
END mqerASN;

DECLARE HYPOTENUSE REAL;  /* Longest side of a right
                             triangle */
DECLARE OPPOSITE_SIDE REAL;  /* The other side, opposite
                                the angle computed */
DECLARE (THETA_RADIANS, THETA_DEGREES) REAL;  /* Two forms
                                                 of the
                                                 answer */
DECLARE PI LITERALLY '3.14159265358979';

HYPOTENUSE = 10.; OPPOSITE_SIDE = 5.;  /* Test values */

/* The following lines calculate the value of an angle of
   a right triangle, given the length of the hypotenuse
   and the opposite side.  mqerASN returns the value in
   radians; it is then converted to degrees. */

THETA_RADIANS = mqerACS( OPPOSITE_SIDE / HYPOTENUSE);
THETA_DEGREES = (180./PI) * THETA_RADIANS;

/* Now THETA_DEGREES = 30 -- it is a 30-60-90 degree
   triangle */
```

ASM-86 usage example:

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs:
EXTRN mqerASN: FAR
```

```
HYPOTENUSE     DQ  10.0      ; longest side of a right
                             ; triangle
OPPOSITE_SIDE  DQ  5.0       ; the other side, next to
                             ; angle computed
                   ; The above initial values are test
                   ; values
THETA_DEGREES  DQ  ?
RAD_TO_DEG     DT  4004E52EE0D31E0FBDC3R   ; the constant
                                           ; 180/PI

   ; The following lines compute an angle of a right
   ; triangle, just as in the above PL/M example; except
   ; with LONG_REAL variables.

      FLD OPPOSITE_SIDE      ; (x) := OPPOSITE_SIDE
      FDIV HYPOTENUSE        ; (x) = OPPOSITE_SIDE/
                             ; HYPOTENUSE
      CALL mqerASN           ; angle is now in (x)
      FLD RAD_TO_DEG
      FMUL                   ; (x) converted to degrees
      FSTP THETA_DEGREES     ; degrees are saved, 8087
                             ; stack popped

   ; With the test inputs, THETA_DEGREES = 30; it is a
   ; 30-60-90 triangle
```

mqerAT2 x = arc tangent(y/x)

Input parameters: (x) is the top number on the 8087 stack; (y) is the next number on the 8087 stack.

Function: mqerAT2 performs one half of a rectangular-to-polar coordinate conversion. If the inputs (x) and (y) are interpreted as the rectangular coordinates of a point in a plane, mqerAT2 returns the angle, in radians, which the point is displaced from the positive X-axis. See figure 4-1. The angle ranges from π (points just below the negative X-axis) to π (points on or just above the negative X-axis).

The formulas used to calculate the angle are as follows:

If $(x) > 0$, return (arc tangent(y/x)).
If $(x) = 0$, return mqerSGN ($\pi/2$, y).
If $(x) < 0$, return (arc tangent(y/x)) + mqerSGN (π, y).

It is legal for one (but not both) of the inputs to be an infinity. The point is then thought of as "infinitely far out" along one of the axes, and the angle returned is the angle of that axis. For points near the X-axis, the angle retains the sign of (y). Thus, x = +INFINITY returns 0 with the sign of (y); x = −INFINITY returns π with the sign of (y); y = +INFINITY returns $+\pi/2$; y = −INFINITY returns $-\pi/2$. In all these cases, it is legal for the non-infinite input to be unnormal. Also note that the distinction between +INFINITY and −INFINITY is made even when the 8087 is in projective (unsigned infinity) mode.

mqerAT2 accepts denormal inputs. Its action depends on whether the 8087 is in normalizing mode, as indicated by the setting of the "D" error masking bit. If the 8087 is in normalizing mode ("D" is unmasked), then any denormals are assumed to be zero. If the 8087 is in warning mode ("D" is masked), the denormals are replaced with unnormals with the same numeric value. Note that even though mqerAT2 checks the "D" masking bit, it does not set the "D" error bit; nor does it call the "D" trap handler.

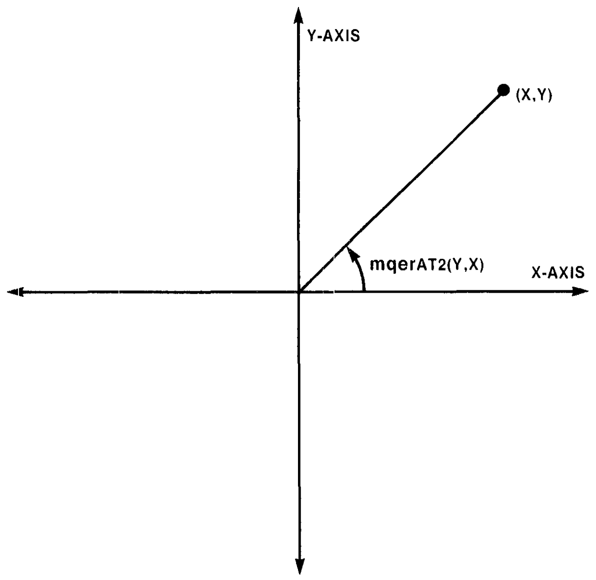

Figure 4-1. Rectangular-to-Polar Coordinate Conversion 121725-1

In some cases, mqerAT2 accepts an unnormal input with no error. This can only happen when (y) and (x) represent the coordinates of a point which is extremely close to one of the axes. A point is considered close enough to the X-axis if the absolute value of (y/x) is less than 2^{-63}. A point is considered close enough to the Y-axis if the absolute value of (x/y) is less than 2^{-63}. If the point is near the positive X-axis, the value returned is the tiny ratio (y/x) with the sign of (y). If the point is near one of the other three axes, the value returned is the angle of that axis: π with the sign of (y) for the negative X-axis, $\pi/2$ for the positive Y-axis, and $-\pi/2$ for the negative Y-axis. Under no circumstances is it legal for both inputs to be unnormal.

Output: The 8087 stack pops once, with the answer replacing the two inputs.

Errors: The first thing mqerAT2 does is check for input NaN's. If either input is a NaN, there is an "I" error. If "I" is masked, the input NaN is returned (if both inputs are NaN's, the larger NaN is returned).

An "I" error is given in a number of other cases. If both inputs are zero (i.e., the point on the coordinate plane is the origin), the angle cannot be defined; so an "I" error is given. If an unnormal input is given which does not fit any of the legal cases described above, an "I" error is given. For all these "I" errors, if "I" is masked, the value INDEFINITE is returned.

If "I" is unmasked for any "I" error, the trap handler is called with the original inputs on the 8087 stack, and the 8087 opcode register set to 277 hex.

If the magnitude of the result is too tiny to be represented in TEMP_REAL format, a "U" underflow error results. If "U" is masked, mqerAT2 returns a gradual-underflow denormal if possible; 0 if not possible. If "U" is unmasked, the trap handler is called with the 8087 opcode register set to 277 hex; but the inputs are not left on the 8087 stack. Instead, the correct answer is placed on the stack, with the too-small exponent presented in "wrapped" form. To obtain the true exponent, subtract decimal 24576 from the given exponent.

PL/M-86 usage example:

```
mqerAT2: PROCEDURE (Y,X) REAL EXTERNAL;
  DECLARE (Y,X) REAL;
END mqerAT2;

FSQRT87: PROCEDURE (X) REAL EXTERNAL;
  DECLARE X REAL;
END FSQRT87;

/* The above procedure is simply a PUBLIC ASM-86 procedure
   which consists of the FSQRT instruction, followed by a
   RET instruction. */

DECLARE REC_X, REC_Y REAL;
DECLARE (POLAR_THETA, POLAR_R) REAL;

REC_X = 3.;  REC_Y = 3. * FSQRT87(3.);    /* Test values */

/* The following statements convert the rectangular
   coordinates REC_X and REC_Y into the polar coordinates
   POLAR_R and POLAR_THETA.  */
POLAR_R = FSQRT87(REC_X * REC_X + REC_Y * REC_Y);
POLAR_THETA = mqerAT2(REC_Y,REC_X);

/* Now POLAR_R = 6 and POLAR_THETA = PI/3. */
```

ASM-86 usage example:

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs:
EXTRN mqerAT2: FAR

POLAR_THETA   DQ  ?
POLAR_R       DQ  ?
REC_X         DQ  3.0    ; rectangular X coordinate
                         ; initialized to test value
REC_Y         DQ  3.0    ; rectangular Y coordinate
                         ; initialized to test value

      FLD REC_Y          ; Y-coordinate parameter goes on
                         ; stack
      FLD REC_X          ; X-coordinate parameter goes on
                         ; stack
      CALL mqerAT2       ; angle is computed
      FSTP POLAR_THETA   ;           and stored

      FLD REC_Y          ; Y onto stack
      FMUL ST,ST         ; Y is squared
      FLD REC_X          ; X onto stack
      FMUL ST,ST         ; X is squared
      FADDP ST(1),ST     ; sum of squares
      FSQRT              ; radius is on stack
      FSTP POLAR_R       ; radius is stored and stack is
                         ; popped

      ; Now POLAR_THETA = PI/4 and
      ; POLAR_R = 3 * Squareroot(2).
```

mqerATN x = arc tangent(x)

Input parameter: (x) is the top number on the 8087 stack.

Function: mqerATN returns the number, in radians, whose tangent is equal to the input value. Numbers are chosen between $-\pi/2$ and $\pi/2$. All zeroes, pseodo-zeroes, and denormals return the input value. Also, unnormals less than 2^{-63} return the input value.

Infinite inputs are valid, whether the 8087 is in affine (signed infinity) of projective (unsigned infinity) mode. The input −INFINITY returns the value $-\pi/2$; the input +INFINITY returns the value $+\pi/2$.

Output: The answer replaces the input on the 8087 stack.

Errors: An input NaN gives an "I" error. Also, an unnormal not less than 2^{-63} gives an "I" error. If "I" is masked, the answer returned is the input for NaN's; the answer is the value INDEFINITE for illegal unnormals. If "I" is unmasked, the trap handler is called, with the input still on the 8087 stack and the value 176 hex in the 8087 opcode register.

PL/M-86 usage example:

```
mqerATN: PROCEDURE (X) REAL EXTERNAL;
  DECLARE X REAL;
END mqerATN;

DECLARE OPPOSITE_SIDE REAL;
DECLARE ADJACENT_SIDE REAL;     /* Shorter sides of a right
                                   triangle */
DECLARE THETA_RADIANS REAL;

OPPOSITE_SIDE = 1.; ADJACENT_SIDE = 1.;  /* Test values */

/* The following line computes the angle of a right
   triangle, given the lengths of the two shorter
   sides. */

THETA_RADIANS = mqerATN(OPPOSITE_SIDE/ADJACENT_SIDE);

/* THETA_RADIANS now equals the value of the angle:
   PI/4. */
```

ASM-86 usage example:

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs:
EXTRN mqerATN: FAR
```

```
OPPOSITE_SIDE  DQ  5.0       ; short side, opposite angle
                             ; computed
ADJACENT_SIDE  DQ  5.0       ; short side, next to angle
                             ; computed
                    ; The above initial values are test
                    ; values
THETA_RADIANS  DQ  ?

   ; The following lines compute an angle of a right
   ; triangle, just as in the above PL/M example, except
   ; with LONG_REAL variables.

         FLD OPPOSITE_SIDE       ; (x) := OPPOSITE_SIDE
         FDIV ADJACENT_SIDE      ; (x) = OPPOSITE_SIDE/
                                 ; ADJACENT_SIDE
         CALL mqerATN            ; angle is now in (x)
         FSTP THETA_RADIANS      ; radians are saved, 8087
                                 ; stack popped

   ; With the test inputs, THETA_RADIANS = PI/4.
   ; It is a 45-45-90 triangle.
```

mqerCOS x = cosine(x)

Input parameter: (x) is the top number on the 8087 stack.

Function: mqerCOS returns the trigonometric cosine of x, where x is an angle expressed in radians. All input zeroes, pseudo-zeroes, and denormals return the value 1. Also, unnormals whose value is less than 2^{-63} return the value 1.

Output: The answer replaces the input on the 8087 stack.

Errors: An "I" error is given for input infinities and NaN's. An "I" error is also given for unnormals which do not represent values less than 2^{-63}.

If "I" is unmasked, the trap handler is called with the input still on the stack, and the 8087 opcode register set to 172 hex. If "I" is masked, the answer is the input for NaNs; the answer is the value INDEFINITE for other invalid inputs.

PL/M-86 usage example:

```
mqerCOS: PROCEDURE (THETA) REAL EXTERNAL;
  DECLARE THETA REAL;
END mqerCOS;

DECLARE (POLAR_R, POLAR_THETA) REAL;
DECLARE REC_X REAL;
DECLARE PI LITERALLY '3.14159265358979';
DECLARE DEG_TO;RAD LITERALLY 'PI/180.';

POLAR_R = 2.; POLAR_THETA = 30.;   /* Test values */

/* The following line computes the X-coordinate of a
   polar-to-rectangular conversion. The input angle is in
   degrees, so it must be converted to radians. */

REC_X = POLAR_R * mqerCOS(POLAR_THETA * DEG_TO_RAD);

/* Now in the test case, REC_X = the square root of 3. */
```

ASM-86 usage example:

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs:
EXTRN mqerCOS: FAR
```

```
POLAR_THETA    DQ  30.0
POLAR_R        DQ  2.0
                   ; the above initializations are test
                   ; values.
REC_X          DQ  ?
DEG_TO_RAD     DT  3FF98EFA351294E9C8AER   ; the constant
                                           ; PI/180.

   ; The following lines compute the X-coordinate of a
   ; polar-to-rectangular conversion, as in the PL/M
   ; example above; except that the variables are
   ; LONG_REAL.

   FLD POLAR_THETA     ; degrees angle onto 8087 stack
   FLD DEG_TO_RAD
   FMUL                ; converted to radians
   CALL mqerCOS        ; cosine is taken
   FMUL POLAR_R        ; answer scaled to correct radius
   FSTP REC_X          ; X-coordinate stored and stack is
                       ; popped

   ; With the test case, REC_X is now the square root of
   ; 3.
```

mqerCSH x = hyperbolic cosine(x)

Input parameter: (x) is the top number on the 8087 stack.

Function: mqerCSH returns the hyperbolic cosine of x, where x is an angle expressed in radians. All input zeroes, pseudo-zeroes, and denormals return the value 1. Also, unnormals whose value is less than 2^{-63} return the value 1.

Either infinity returns the value +INFINITY with no error.

Output: The answer replaces the input on the 8087 stack.

Errors: An "I" error is given for input NaN's. An "I" error is also given for unnormals which do not represent values less than 2^{-63}.

If "I" is unmasked, the trap handler is called with the input still on the stack. If "I" is masked, the answer is the input for NaNs; the answer is the value INDEFINITE for invalid unnormals.

mqerCSH will give an "O" overflow error if the input is greater than about 11355, or less than about −11355. When "O" is masked, the value +INFINITY is returned. When "O" is unmasked, the trap handler is called with the input still on the 8087 stack.

When the trap handler is called, mqerSNH first sets the 8087 opcode register to 16F hex.

PL/M-86 usage example:

```
mqerCSH: PROCEDURE (THETA) REAL EXTERNAL;
  DECLARE THETA REAL;
END mqerCSH;

DECLARE (INPUT_VALUE, OUTPUT_VALUE) REAL;

INPUT_VALUE = 1.3;  /* Test value */

OUTPUT_VALUE = mqerCSH(INPUT_VALUE);

/* For the test case, OUTPUT_VALUE now is approximately
   1.97091 */
```

ASM-86 usage example:

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs:
EXTRN mqerCSH: FAR

INPUT_VALUE     DQ  1.56   ; test value
OUTPUT_VALUE    DQ  ?

   FLD INPUT_VALUE     ; input goes onto 8087 stack
   CALL mqerCSH        ; hyperbolic cosine is taken
   FSTP OUTPUT_VALUE   ; answer is stored in 8086 memory,
                       ; 8087 is popped

; For the test value, OUTPUT_VALUE now is about 2.48448
```

mqerDIM $x = \max(y-x, +0)$

Input parameters: (x) is the top number on the 8087 stack; (y) is the next number on the 8087 stack.

Function: mqerDIM first compares (y) and (x); if (x) is greater than (y), the answer +0 is returned. If (y) is greater than (x), the positive value (y − x) is returned.

When the 8087 chip is in affine (signed infinity) mode, and when both inputs are not the same, then infinite inputs are allowed. The results are as expected from the above comparison and subtraction. That is, if (x) is +INFINITY or (y) is −INFINITY the answer is +0; if (x) is −INFINITY or (y) is +INFINITY, the answer is +INFINITY.

mqerDIM uses the 8087 precision and rounding modes which were in effect at the time mqerDIM is called. Thus the results can vary, depending on the settings of these bits in the 8087 control word.

Output: The 8087 stack pops once, with the answer replacing the two inputs.

Errors: The first error which is detected is the "D" error, for unnormal or denormal inputs. If the 8087 is in warning mode ("D" is masked), the calculation continues; but the "D" error bit remains set. If the 8087 is in normalizing mode ("D" is unmasked), the trap handler is called. The trap handler is called directly from the interior of mqerDIM; the 8087 opcode register contains the code for the instruction which found the denormal: either FCOM or FSUB. The (y) and (x) inputs are left on the 8087 stack. Most trap handlers will replace the denormal input(s) with normalized numbers, reexecute the FCOM or FSUB instruction, and continue through mqerDIM. The trap handler provided by EH87.LIB (described in Chapter 5) will do this, and replace denormals with zero.

mqerDIM next checks for NaN inputs. If either input is a NaN, an "I" error is given. If "I" is masked, the answer returned is the input NaN (the larger NaN if both inputs are NaN's).

An "I" error is also given if the 8087 chip is in projective (unsigned infinity) mode, and either input is infinity; or if the 8087 chip is in affine (signed infinity) mode, and both inputs are the same infinity. In these cases a masked "I" yields the value INDEFINITE for an answer.

For all "I" errors, when "I" is unmasked, the trap handler is called with the inputs still on the 8087 stack, and the 8087 opcode register set to 265 hex.

In the case (y) > (x), when the answer is calculated by the subtraction (y − x), the subtraction can cause either an "O" overflow error or a "U" underflow error. If the error is masked, the answer returned is +INFINITY for overflow. For underflow, a gradual-underflow denormal is returned if possible; 0 is returned if not possible. If the error is unmasked, the trap handler is called, with the value 165 hex in the 8087 opcode register. The inputs are not on the 8087 stack. Instead, the correct answer appears on the 8087 stack, with the out-of-range exponent given "wrapped around". For underflow, the correct exponent is the given exponent minus the decimal value 24576; for overflow, it is the exponent plus 24576.

PL/M-86 usage example:

```
mqerDIM: PROCEDURE (Y,X) REAL EXTERNAL;
  DECLARE (Y,X) REAL;
END mqerDIM;

DECLARE (COSTS, RECEIPTS) REAL;
DECLARE (PROFIT, LOSS) REAL;

COSTS = 4700.00; RECEIPTS = 5300.00;  /* Test values */

/* The following lines return the profit or loss, given
   the costs and receipts. The positive difference goes
   into PROFIT or LOSS as appropriate, and the other value
   is set to zero. */

PROFIT = mqerDIM(RECEIPTS,COSTS);
LOSS = mqerDIM(COSTS,RECEIPTS);

/* For the test case, PROFIT is now 600.00, and LOSS
   is 0. */
```

ASM-86 usage example:

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs:
EXTRN mqerDIM: FAR

COSTS       DQ  800.00
RECEIPTS    DQ  650.00
                ; the above initializations are test cases
PROFIT      DQ  ?
LOSS        DQ  ?

   ; The following lines compute profit and loss, just as
   ; in the PL/M example.

   FLD RECEIPTS     ; first parameter to mqerDIM
   FLD COSTS        ; second parameter to mqerDIM
   CALL mqerDIM     ; positive difference is computed
   FSTP PROFIT      ; answer is stored and stack is popped
   FLD COSTS        ; now perform function the other way
   FLD RECEIPTS
   CALL mqerDIM
   FSTP LOSS        ; other answer goes to LOSS, and stack
                    ; is again popped

   ; For this test case, LOSS = 150.00 and PROFIT is 0.
```

mqerEXP $x = e^x$

Input parameters: x is the top number on the 8087 stack.

Function: mqerEXP raises the constant e (2.718281828459+) to the power of the input number. This is a standard function because it is used to derive other functions, notably the hyperbolics. Also, on many computers it is slightly easier to compute than exponentials based on other constants. The exponential is valid for both positive and negative inputs.

All input zeroes, pseudo-zeroes, denormals, and unnormals less than 2^{-63} yield an answer of 1.

If the 8087 is in affine (signed infinity) mode, infinite inputs are valid. +INFINITY returns itself as the answer; −INFINITY returns 0.

Output: The answer replaces the input on the 8087 stack.

Errors: All input NaN's, and input unnormals greater than 2^{-63}, cause an "I" error. Also, if the 8087 is in projective (unsigned infinity) mode, both infinite inputs give an "I" error. If "I" is masked, the value returned is the input if a NaN; the value INDEFINITE otherwise. If "I" is unmasked, the trap handler is called with the input still on the 8087 stack.

Since the largest number which can be stored in the 8087's temporary real format is about e^{11357}, an input greater than about 11357 causes an "O" overflow error. When "O" is masked, the value +INFINITY is returned. Likewise, an input less than about −11355 causes the "U" underflow error. If "U" is masked, the result is a gradual-underflow denormal if possible, zero otherwise. For unmasked "O" or "U" errors, the trap handler is called, with the input still on the 8087 stack.

When the trap handler is called, mqerEXP first places the value 16B hex into the 8087 opcode register.

PL/M-86 usage example:

```
mqerEXP: PROCEDURE (X) REAL EXTERNAL;
  DECLARE X REAL;
END mqerEXP;

DECLARE Y_VALUE REAL;
DECLARE X_VALUE REAL;
DECLARE NORM_CONSTANT LITERALLY '0.3989422803';
/* 1 / (SQRT(2*PI)) */

X_VALUE = -2.0;    /* Test value */

/* The following line gives the value for the graph of the
   normal distribution function, for mean 0 and variance
   1.  By plotting Y_VALUE against various different input
   X-values, one obtains the familiar ''bell-shaped
   curve''. */
```

```
Y_VALUE = NORM_CONSTANT
             * mqerEXP( - X_VALUE * X_VALUE / 2. );

/* For the test input, Y_VALUE is now approximately
   0.5399100 */
```

ASM-86 usage example:

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs:
EXTRN mqerEXP: FAR

MINUS_2         DQ  -2.  ; constant used in calculation
                         ; below
X_VALUE         DQ  0.4  ; initialization is a test value
Y_VALUE         DQ  ?
NORM_CONSTANT   DT  3FFDCC42299EA1B28468R  ; constant
                                           ; 1/sqrt(2*PI)

   ; The following lines compute the normal distribution
   ; just as in the PL/M example, except with LONG_REAL
   ; numbers.

   FLD X_VALUE           ; load input onto 8087 stack
   FMUL ST,ST            ; input is squared
   FIDIV MINUS_2         ; negate and divide by 2
   CALL mqerEXP          ; exponentiate
   FLD NORM_CONSTANT
   FMUL                  ; divide by the square root of
                         ; 2 * PI
   FSTP Y_VALUE          ; store the result and pop the
                         ; stack

   ; For the test case Y_VALUE is now about 0.3399562
```

mqerIA2 AX = roundaway(x)

Input parameter: (x) is the top number on the 8087 stack.

Function: If x is not normal it is first normalized. Then x is rounded to the nearest integer. If there are two equally near integers (i.e., the fractional part of x is .5), then the integer farthest from zero is selected. The answer is given in 16-bit two's complement form. For example, 3.1 returns hex 0003; 10.5 returns hex 000B; −2.5 returns hex FFFD, which is −3 in two's complement form.

Output: The input is popped from the 8087 stack, and the answer is left in the AX register.

Errors: The only numbers that will fit the 16-bit destination are those between, but not including, −32768.5 and +32767.5. Numbers not in this range, including infinities and NaN's, will cause an "I" error. If "I" is masked, the "indefinite integer" value 8000 hex is returned. If "I" is unmasked, the trap handler is called with the input still on the 8087 stack, and the 8087 opcode register set to 17E hex.

PL/M-86 usage example:

```
mqerIA2: PROCEDURE (X) INTEGER EXTERNAL;
  DECLARE X REAL;
END mqerIA2;

DECLARE REAL_VAR REAL;
DECLARE INTEGER_VAR INTEGER;

REAL_VAR = 4.5;    /* Test value */

INTEGER_VAR = mqerIA2(REAL_VAR);

/* Now in the test case, INTEGER_VAR = 0005 hex. */
```

ASM-86 usage example:

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs:
EXTRN mqerIA2: FAR

INTEGER_VAR    DW   ?
REAL_VAR       DQ   -8.5  ; Initialization is a test value

   FLD REAL_VAR          ; load the parameter
   CALL mqerIA2          ; round to nearest integer
   MOV INTEGER_VAR,AX    ; store the answer

   ; For the test case, INTEGER_VAR now equals -9, which
   ; is FFF7 hex.
```

mqerIA4 DXAX = roundaway(x)

Input parameter: (x) is the top number on the 8087 stack.

Function: If x is not normal it is first normalized. Then x is rounded to the nearest integer. If there are two equally near integers (i.e., the fractional part of x is .5), then the integer farthest from zero is selected. The answer is given in 32-bit two's complement form. For example, 3.1 returns hex 00000003; 10.5 returns hex 0000000B; −2.5 returns hex FFFFFFFD, which is −3 in two's complement form.

Output: The input is popped from the 8087 stack, and the answer is left in the DX and AX registers, with DH the highest byte and AL the lowest byte.

Errors: The only numbers that will fit the 32-bit destination are those between, but not including, −2147483648.5 and +2147483647.5. Numbers not in this range, including infinities and NaN's, will cause an "I" error. If "I" is masked, the "indefinite integer" value 80000000 hex is returned. If "I" is unmasked, the trap handler is called with the input still on the 8087 stack, and the 8087 opcode register set to 168 hex.

PL/M-86 usage example:

Since PL/M-86 does not support a 32-bit integer data type, mqerIA4 cannot be used by PL/M programs. You should use either mqerIA2 or mqerIAX.

ASM-86 usage example:

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs:
EXTRN mqerIA4: FAR

COUNT        DD  ?
REAL_COUNT   DQ  100000.5    ; initialized to a test value

; The following code assumes that the above variables are
; in the DS segment.

   FLD REAL_COUNT          ; load the parameter onto the
                           ; 8087 stack
   CALL mqerIA4            ; convert the number to 32 bits.
   MOV BX,OFFSET(COUNT)    ; point to the destination
   MOV [BX],AX             ; move lower 16 bits of answer
   MOV [BX+2],DX           ; move upper 16 bits of answer

   ; With the test input, COUNT is now 100001, which is
   ; 000186A1 hex.
```

mqerIAX x = roundaway(x)

Input parameter: (x) is the top number on the 8087 stack.

Function: If x is not normal it is first normalized. Then x is rounded to the nearest integer. If there are two equally near integers (i.e., the fractional part of x is .5), then the integer farthest from zero is selected. For example, 3.3 returns 3; 4.5 returns 5; −6.5 returns −7.

Infinite values are returned unchanged, with no error.

Output: The answer replaces the input on the 8087 stack.

Errors: The "I" error is set if the input is a NaN. The NaN is left unchanged. If the "I" error is unmasked, the trap handler is called, with the 8087 opcode register set to 167 hex.

PL/M-86 usage example:

```
mqerIAX: PROCEDURE (X) REAL EXTERNAL;
  DECLARE X REAL;
END mqerIAX;

DECLARE (HOURS, MINUTES) REAL;

HOURS = 2.71;  /* Test value */

/* The following statement converts HOURS into an integer
   number of MINUTES */

MINUTES = mqerIAX(HOURS*60.);

/* Now MINUTES = 163. */
```

ASM-86 usage example:

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs:
EXTRN mqerIAX: FAR

HOURS      DQ  2.71    ; initialized to test value
MINUTES    DQ  ?
SIXTY      DD  60.0

; The following lines convert HOURS into an integer number
; of MINUTES. It does the same thing as the above PL/M
; example, only with LONG_REAL numbers.

   FLD HOURS        ; put HOURS onto 8087 stack
   FMUL SIXTY       ; convert to a real number of MINUTES
   CALL mqerIAX     ; round to the nearest integer
   FSTP MINUTES     ; store the integer in LONG_REAL format

; With the test case, MINUTES is now 163.0
```

mqerIC2 AX = chop(x)

Input parameter: (x) is the top number on the 8087 stack.

Function: If x is not normal it is first normalized. Then if x is an integer, it is returned unchanged. If x is not an integer, the fractional part of x is chopped. Thus the nearest integer in the direction of 0 is returned. The answer is given in 16-bit two's complement form. For example, 4 returns hex 0004; 11.7 returns hex 000B; −6.9 returns hex FFFA, which is −6 in two's complement form.

Output: The input is popped from the 8087 stack, and the answer is left in the AX register.

Errors: The only numbers that will fit the 16-bit destination are those between, but not including, −32769 and +32768. Numbers not in this range, including infinities and NaN's, will cause an "I" error. If "I" is masked, the "indefinite integer" value 8000 hex is returned. If "I" is unmasked, the trap handler is called with the input still on the 8087 stack.

Note that "indefinite integer" can also represent the legal output value −32768. The two outputs can be distinguished only by the setting of the "I" exception bit of the 8087.

If a truncation does take place, the "P" error is set. If "P" is masked the answer is returned as usual, since this is not usually considered an error. If "P" is unmasked, the trap handler is called. Since the output in the AX register is likely to be lost before the trap handler can use it, we recommend that you do not unmask the "P" exception.

If either trap handler is called, the 8087 opcode register is first set to 17E hex.

PL/M-86 usage example:

```
mqerIC2: PROCEDURE (X) INTEGER EXTERNAL;
  DECLARE X REAL;
END mqerIC2;

DECLARE CONTROL_SETTING INTEGER;
DECLARE REAL_INPUT REAL;

REAL_INPUT = 37.885;    /* Test value */

/* The following line translates REAL_INPUT, which could
   have been calculated using floating point arithmetic,
   into an INTEGER value CONTROL_SETTING, which might be
   output as up to 16 logic lines to a physical device. */

CONTROL_SETTING = mqerIC2(REAL_INPUT);

/* For the test input, CONTROL_SETTING is now 37; which is
0025 hex. */
```

ASM-86 usage example:

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs:
EXTRN mqerIC2: FAR

REAL_INPUT         DQ  65.7    ; initialized to a test
                               ; value
CONTROL_SETTING    DW  0

; The following lines convert the REAL_INPUT into the
; 16-bit integer value CONTROL_SETTING, just as in the
; PL/M example above, except that REAL_INPUT is a
; LONG_REAL number.

   FLD REAL_INPUT            ; load the input onto the 8087
                             ; stack
   CALL mqerIC2              ; chop to the integer part
   MOV CONTROL_SETTING,AX    ; store the 16-bit answer

; For the input test value, CONTROL_SETTING is now 65, or
; 41 hex.
```

mqerIC4 DXAX = chop(x)

Input parameters: x is the top number on the 8087 stack.

Function: If x is not normal it is first normalized. Then if x is an integer, it is returned unchanged. If x is not an integer, the fractional part of x is chopped. Thus the nearest integer in the direction of 0 is returned. The answer is given in 32-bit two's complement form. For example, 4 returns hex 00000004; 11.7 returns hex 0000000B; −6.9 returns hex FFFFFFFA, which is −6 in two's complement form.

Output: The input is popped from the 8087 stack, and the answer is left in the DX and AX registers, with DH the highest byte and AL the lowest byte.

Errors: The only numbers that will fit the 32-bit destination are those between, but not including, −2147483649 and +2147483648. Numbers not in this range, including infinities and NaN's, will cause an "I" error. If "I" is masked, the "indefinite integer" value 80000000 hex is returned. If "I" is unmasked, the trap handler is called with the input still on the 8087 stack.

Note that "indefinite integer" can also represent the legal output value −2147483648. The two outputs can be distinguished only by the setting of the "I" exception bit of the 8087.

If a truncation does take place, the "P" error is set. If "P" is masked the answer is returned as usual, since this is not usually considered an error. If "P" is unmasked, the trap handler is called. Since the output in the DXAX registers is likely to be lost before the trap handler can use it, we recommend that you do not unmask the "P" exception.

If either trap handler is called, the 8087 opcode register is first set to 179 hex.

PL/M-86 usage example:

Since PL/M-86 does not support a 32-bit integer data type, mqerIC4 cannot be used by PL/M programs. You should use either mqerIC2 or mqerICX.

ASM-86 usage example:

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs:
EXTRN mqerIC4: FAR

POPULATION     DQ  5306279.0    ; total number of voters
SUPPORT_SHARE  DQ  .39          ; proportion of population
                                ; supporting us
                  ; above numbers are initialized to test
                  ; values
SUPPORT_VOTES  DD  ?
```

```
; The following lines calculate the number of voters
; supporting an issue, given the total population, and the
; fractional share of the population which supports the
; issue.  This is simply the share multiplied by the
; total, then chopped to a 32-bit integer SUPPORT_VOTES.

  FLD POPULATION                       ; load total onto
                                       ; 8087 stack
  FMUL SUPPORT_SHARE                   ; multiply by share
  CALL mqerIC4                         ; chop to 32 bits
  MOV WORD PTR SUPPORT_VOTES, AX       ; store bottom 16
                                       ; bits
  MOV WORD PTR (SUPPORT_VOTES+2), DX ; store top 16 bits

; With the test inputs, SUPPORT_VOTES is now 2069448,
; which is 001F93C8 hex.
```

mqerICX   x = chop(x)

Input parameter: (x) is the top number on the 8087 stack.

Function: If x is not normal it is first normalized. Then if x is an integer, it is returned unchanged. If x is not an integer, the fractional part of x is chopped. Thus the nearest integer in the direction of 0 is returned. For example, 4 returns 4; −3.9 returns −3; 1.7 returns 1.

Infinite values are returned unchanged, with no error.

Output: The answer replaces the input on the 8087 stack.

Errors: If a truncation does take place, the "P" error is set. The correct answer is on the 8087 stack. If "P" is unmasked (this is rarely done), the trap handler is then called.

The "I" error is set if the input is a NaN. The NaN is left unchanged, and if the "I" error is unmasked, the trap handler is called.

If either trap handler is called, the 8087 opcode register is first set to 166 hex.

PL/M-86 usage example:

```
mqerICX: PROCEDURE (X) REAL EXTERNAL;
  DECLARE X REAL;
END mqerICX;

DECLARE PRICE_DOLLARS REAL;
PRICE_DOLLARS = 37.596;   /* Test value */

/* The following statement chops PRICE_DOLLARS to an even
   number of pennies.  First, PRICE_DOLLARS is multiplied
   by 100 to get a number of pennies; second, the pennies
   are chopped by mqerICX; third, the answer is divided by
   100 to convert back into dollars. */

PRICE_DOLLARS = ( mqerICX(PRICE_DOLLARS * 100.) / 100. ) ;

/* Now PRICE_DOLLARS = 37.59 */
```

ASM-86 usage example:

```
; This EXTRN must appear outside of all SEGMENT-ENDS
pairs:
EXTRN mqerICX: FAR

PRICE_DOLLARS    DQ  37.596    ; initialized to a test
                               ;value
ONE_HUNDRED      DD  100.00    ; constant which is used
                               ;twice below
```

```
; The following lines chop PRICE_DOLLARS to an even number
; of pennies, just as in the PL|M example above, except
; that PRICE_DOLLARS is here a LONG_REAL variable.

   FLD PRICE_DOLLARS      ; amount is loaded onto 8087
                          ; stack
   FMUL ONE_HUNDRED       ; amount is converted to pennies
   CALL mqerICX           ; pennies are chopped
   FDIV ONE_HUNDRED       ; amount is converted back to
                          ; dollars
   FSTP PRICE_DOLLARS     ; converted number is stored,
                          ; 8087 stack is popped

; With the above test input, PRICE_DOLLARS is now 37.59
```

mqerIE2 AX = roundeven(x)

Input parameter: (x) is the top number on the 8087 stack.

Function: If x is not normal it is first normalized. Then x is rounded to the nearest integer. If the are two integers equally near (i.e., the fractional part of x is .5), then the even integer is selected. The answer is given in 16-bit two's complement form. For example, 3.1 returns hex 0003; 10.5 returns hex 000A; −2.5 returns hex FFFE, which is −2 in two's complement form.

Output: The input is popped from the 8087 stack, and the answer is left in the AX register.

Errors: The only numbers that will fit the 16-bit destination are those greater than or equal to −32768.5 and less than +32767.5. Numbers not in this range, including infinities and NaN's, will cause an "I" error. If "I" is masked, the "indefinite integer" value 8000 hex is returned. If "I" is unmasked, the trap handler is called with the input still on the 8087 stack.

If a rounding does take place, the "P" error is set. If "P" is masked the answer is returned as usual, since this is not usually considered an error. If "P" is unmasked, the trap handler is called. Since the output in the AX register is likely to be lost before the trap handler can use it, we recommend that you do not unmask the "P" exception.

When the trap handler is called, the 8087 opcode register is first set to 180 hex.

PL/M-86 usage example:

```
mqerIE2: PROCEDURE (X) INTEGER EXTERNAL;
  DECLARE X REAL;
END mqerIE2;

DECLARE REAL_VAR REAL;
DECLARE INTEGER_VAR INTEGER;

REAL_VAR = 4.5;    /* Test value */

INTEGER_VAR = mqerIE2(REAL_VAR);

/* Now in the test case, INTEGER_VAR = 0004 hex. */
```

ASM-86 usage example:

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs:
EXTRN mqerIE2: FAR
```

```
INTEGER_VAR   DW  ?
REAL_VAR      DQ  -8.5  ; Initialization is a test value

   FLD REAL_VAR         ; load the parameter
   CALL mqerIE2         ; round to nearest integer
   MOV INTEGER_VAR,AX   ; store the answer

   ; For the test case, INTEGER_VAR now equals -8, which
   ; is FFF8 hex.
```

mqerIE4 DXAX = roundeven(x)

Input parameter: (x) is the top number on the 8087 stack.

Function: If x is not normal it is first normalized. Then x is rounded to the nearest integer. If the are two integers equally near (i.e., the fractional part of x is .5), then the even integer is selected. The answer is given in 32-bit two's complement form. For example, 3.1 returns hex 00000003; 10.5 returns hex 0000000A; −2.5 returns hex FFFFFFFE, which is −2 in two's complement form.

Output: The input is popped from the 8087 stack, and the answer is left in the DX and AX registers, with DH the highest byte and AL the lowest byte.

Errors: The only numbers that will fit the 32-bit destination are those greater than or equal to −2147483648.5 and less than +2147483647.5. Numbers not in this range, including infinities and NaN's, will cause an "I" error. If "I" is masked, the "indefinite integer" value 80000000 hex is returned. If "I" is unmasked, the trap handler is called with the input still on the 8087 stack.

If a rounding does take place, the "P" error is set. If "P" is masked the answer is returned as usual, since this is not usually considered an error. If "P" is unmasked, the trap handler is called. Since the output in the DXAX registers is likely to be lost before the trap handler can use it, we recommend that you do not unmask the "P" exception.

When the trap handler is called, the 8087 opcode register is first set to 17B hex.

PL/M-86 usage example:

Since PL/M-86 does not support a 32-bit integer data type, mqerIE4 cannot be used by PL/M programs. You should use either mqerIE2 or mqerIEX.

ASM-86 usage example:

```
; This EXTRN must appear outside of all SEGMENT-ENDS
pairs:
EXTRN mqerIE4: FAR

COUNT        DD  ?
REAL_COUNT   DQ  100000.5    ; initialized to a test value

; The following code assumes that the above variables are
; in the DS segment.

   FLD REAL_COUNT               ; load the parameter onto
                                ; the 8087 stack
   CALL mqerIE4                 ; convert the number to 32
                                ; bits.
   MOV WORD PTR COUNT, AX       ; move lower 16 bits of
                                ; answer
   MOV WORD PTR (COUNT+2), DX   ; move upper 16 bits of
                                ; answer

   ; With the test input, COUNT is now 100000, which is
   ; 000186A0 hex.
```

mqerIEX x = roundeven(x)

Input parameter: (x) is the top number on the 8087 stack.

Function: If x is not normal it is first normalized. Then x is rounded to the nearest integer. If the are two integers equally near (i.e., the fractional part of x is .5), then the even integer is selected. For example, 3.3 returns 3; 4.5 returns 4; −6.5 returns −6.

Infinite values are returned unchanged, with no error.

Output: The answer replaces the input on the 8087 stack.

Errors: If a rounding does take place, the "P" error is set. The correct answer is on the 8087 stack. If "P" is unmasked (this is rarely done), the trap handler is then called.

The "I" error is set if the input is a NaN. The NaN is left unchanged, and if the "I" error is unmasked, the trap handler is called.

If either trap handler is called, the 8087 opcode register is first set to 178 hex.

PL/M-86 usage example:

```
mqerIEX: PROCEDURE (X) REAL EXTERNAL;
  DECLARE X REAL;
END mqerIEX;

DECLARE UNITS REAL;
DECLARE THOUSANDS REAL;

UNITS = 4500.00;  /* Test value */

/* The following line computes an integer number of
   thousands, given an input number of units. */

THOUSANDS = mqerIEX(UNITS/1000.);

/* With the test input value, THOUSANDS now equals 4.00 */
```

ASM-86 usage example:

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs:
EXTRN mqerIEX: FAR

THOUSANDS      DQ  ?
UNITS          DQ  5890.14    ; initialization is a test
                              ; value
ONE_GRAND      DD  1000.00    ; constant for the division
                              ; below
; The following lines compute an integer number of
; thousands, just as in the PL|M example above, except the
; LONG_REAL variables are used.

   FLD UNITS            ; load the input onto the 8087 stack
   FDIV ONE_GRAND       ; convert to thousands
   CALL mqerIEX         ; round to an integer value
   FSTP THOUSANDS       ; store it into 8086 memory and pop
                        ; the 8087 stack

   ; With the test value, THOUSANDS is now 6.00
```

mqerLGD x = common log(x)

Input parameter: (x) is the top number on the 8087 stack.

Function: mqerLGD returns the number L such that 10^L equals the input number. For positive inputs this is a well-defined number. For example, an input 10 gives 1; 100 gives 2; .001 gives −3. Inputs between 10 and 100 give outputs between 1 and 2.

If the 8087 processor is in affine (signed infinity) mode, the input +INFINITY is also valid, returning itself as an answer.

Output: The answer replaces the input on the 8087 stack.

Errors: Negative numbers (including −INFINITY), all NaN's, and all unnormals give an "I" error. Also, when the 8087 is in projective (unsigned infinity) mode, both values of INFINITY give an "I" error. If "I" is masked, the result is the input for NaN's; the result is the value INDEFINITE for other invalid inputs. If "I" is unmasked, the trap handler is called with the input number still on the 8087 stack.

Zero input, of either sign, gives a "Z" error. If "Z" is masked, the result is −INFINITY. If "Z" is unmasked, the trap handler is called with the input number still on the 8087 stack.

If the input is a denormal, mqerLGD tests the 8087 "D" exception bit to see if the 8087 is in normalizing mode. If in normalizing mode ("D" unmasked), the input is treated as a zero, and a "Z" error is issued. If not in normalizing mode ("D" masked), the input is treated as an unnormal, and an "I" error is issued. Note that even though the "D" masking bit is tested, the "D" error is never issued by mqerLGD, and the "D" trap handler is never called during mqerLGD.

Whenever the trap handler is called, the number 16D hex is first placed into the 8087 opcode register.

PL/M-86 usage example:

```
mqerLGD: PROCEDURE (X) REAL EXTERNAL;
  DECLARE X REAL;
END mqerLGD;

DECLARE QUANTITY REAL;
DECLARE TENS_POWER REAL;

QUANTITY = 1900.00;    /* Test value */

TENS_POWER = mqerLGD(QUANTITY);

/* Since the test value QUANTITY is between 10 ** 3 and
   10 ** 4, the answer TEST_POWER is between 3 and 4. It
   is about 3.27875.  */
```

ASM-86 usage example:

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs:
EXTRN mqerLGD: FAR

QUANTITY       DQ  .0001     ; initialized to a test value
TENS_POWER     DQ  ?

; The following code implements the above PL/M call in
; assembly language, with LONG_REAL variables.

   FLD QUANTITY            ; load input onto 8087 stack
   CALL mqerLGD            ; take the base 10 logarithm
   FSTP TENS_POWER         ; store the answer and pop the
                           ; 8087 stack

; Since the test input was 10 ** -4, the output should be
; -4.00.  Due to accumulated rounding errors, it may not
; be the exact integer.
```

mqerLGE x = natural log(x)

Input parameter: (x) is the top number on the 8087 stack.

Function: mqerLGE returns the number L such that e^L equals the input number. The constant e is 2.718281828459+. This logarithm is called "natural" because it occurs in calculus as the inverse derivative of 1/X; and on many computers it is slightly easier to compute than other logarithms. For positive inputs, the logarithm is a well-defined number. If the 8087 processor is in affine (signed infinity) mode, the input +INFINITY is also valid, returning itself as an answer.

Output: The answer replaces the input on the 8087 stack.

Errors: Negative numbers (including −INFINITY), all NaN's, and all unnormals give an "I" error. Also, when the 8087 is in projective (unsigned infinity) mode, both values of INFINITY give an "I" error. If "I" is masked, the result is the input for NaN's; the result is the value INDEFINITE for other invalid inputs. If "I" is unmasked, the trap handler is called with the input number still on the 8087 stack.

Zero input, of either sign, gives a "Z" error. If "Z" is masked, the result is −INFINITY. If "Z" is unmasked, the trap handler is called with the input number still on the 8087 stack.

If the input is a denormal, mqerLGE tests the 8087 "D" exception bit to see if the 8087 is in normalizing mode. If in normalizing mode ("D" unmasked), the input is treated as a zero, and a "Z" error is issued. If not in normalizing mode ("D" masked), the input is treated as an unnormal, and an "I" error is issued. Note that even though the "D" masking bit is tested, the "D" error is never issued by mqerLGE, and the "D" trap handler is never called during mqerLGE.

Whenever the trap handler is called, the number 16C hex is first placed into the 8087 opcode register.

PL/M-86 usage example:

```
mqerLGE: PROCEDURE (X) REAL EXTERNAL;
  DECLARE X REAL;
END mqerLGE;

DECLARE X REAL;
DECLARE THETA REAL;

X = 3.0;    /* Test value */

/* The following code calculates the inverse hyperbolic
   sine of X.  That is, THETA is set to the number whose
   hyperbolic sine is X. */

THETA = mqerLGE( X + FSQRT87(X*X + 1.));

/* For the test input, THETA now equals about 1.81845 */
```

ASM-86 usage example:

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs:
EXTRN mqerLGE: FAR

X           DQ  -3.0     ; initialized to a test value
THETA       DQ  ?
ONE         DD  1.00     ; constant used below

; The following code calculates the inverse hyperbolic
; sine, just as in the PL/M example above, except with
; LONG_REAL inputs.

   FLD X                 ; input parameter onto 8087 stack
   FMUL ST,ST            ; X squared
   FADD ONE              ; X squared + 1
   FSQRT
   FADD X                ; X + SQRT(X squared + 1)
   CALL mqerLGE          ; take the natural logarithm -- this
                         ; is the answer
   FSTP THETA            ; store it and pop the 8087 stack

; With the test input THETA is now about -1.81845
```

mqerMAX x = max(x,y)

Input parameters: (x) is the top number on the 8087 stack; (y) is the next number on the 8087 stack.

Function: mqerMAX returns the greater of the numbers (x) and (y). That is, if the 8087 FCOM instruction indicates that (x) > (y), then (x) is returned. If (x) < (y), then (y) is returned. When (x) and (y) test as equal even though they may be given in different formats (for example, +0 and −0), (x) is returned.

If the 8087 chip is in affine (signed infinity) mode, either or both inputs can be infinite. If the 8087 is in projective (unsigned infinity) mode, then if either input is infinite, both must be infinite. In that case, the values test as equal, and (x) is the answer.

Output: The 8087 stack pops once, with the answer replacing the two inputs.

Errors: The first error which is detected is the "D" error, for unnormal or denormal inputs. If the 8087 is in warning mode ("D" is masked), the calculation continues; but the "D" error bit remains set. If the 8087 is in normalizing mode ("D" is unmasked), the "D" trap handler is called. The trap handler is called directly from the interior of mqerMAX; the 8087 opcode register contains the code for the FCOM instruction which caused the "D" error. The (y) and (x) inputs are left on the 8087 stack. Most trap handlers will replace the denormal input(s) with normalized numbers, reexecute the FCOM or FSUB instruction, and continue through mqerMAX. The trap handler provided by EH87.LIB, described in Chapter 5, will do this and replace denormals with zero.

mqerMAX next checks for NaN inputs. If either input is a NaN, an "I" error is given. If "I" is masked, the answer returned is the input NaN (the larger NaN if both inputs are NaN's).

An "I" error is also given if the 8087 chip is in projective (unsigned infinity) mode, and exactly one of the inputs is infinite. In this case a masked 'I' yields the value INDEFINITE for an answer.

For all "I" errors, when "I" is unmasked, the trap handler is called with the inputs still on the 8087 stack, and the 8087 opcode register set to 265 hex.

PL/M-86 usage example:

```
mqerMAX: PROCEDURE (Y,X) REAL EXTERNAL;
  DECLARE (Y,X) REAL;
END mqerMAX;

DECLARE POPULATIONS(10) REAL;
DECLARE LARGEST REAL;
DECLARE N BYTE;

/* The following code sets LARGEST to the greatest of the
   ten values in the array POPULATIONS. */

LARGEST = POPULATIONS(0);
DO N = 1 TO 9;
  LARGEST = mqerMAX(LARGEST,POPULATIONS(N));
END;
```

ASM-86 usage example:

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs:
EXTRN mqerMAX: FAR

VAR1          DQ  -5.3
VAR2          DQ  -7.9
                  ; the above initializations are test
                  ; values
LARGEST       DQ  ?

; The following code sets LARGEST to the maximum of VAR1
; and VAR2.

     FLD VAR1            ; load the first parameter onto
                         ; the 8087 stack
     FLD VAR2            ; load the second parameter
     CALL mqerMAX        ; stack now contains the maximum
     FSTP LARGEST        ; maximum is stored, and 8087
                         ; stack is popped

; With the test inputs, LARGEST is now -5.3
```

mqerMIN x = min(x,y)

Input parameters: (x) is the top number on the 8087 stack; (y) is the next number on the 8087 stack.

Function: mqerMIN returns the lesser of the numbers (x) and (y). That is, if the 8087 FCOM instruction indicates that (x) < (y), then (x) is returned. If (x) > (y), then (y) is returned. When (x) and (y) test as equal even though they may be given in different formats (for example, +0 and −0), (x) is returned.

If the 8087 chip is in affine (signed infinity) mode, either or both inputs can be infinite. If the 8087 is in projective (unsigned infinity) mode, then if either input is infinite, both must be infinite. In that case, the values test as equal, and (x) is the answer.

Output: The 8087 stack pops once, with the answer replacing the two inputs.

Errors: The first error which is detected is the "D" error, for unnormal or denormal inputs. If the 8087 is in warning mode ("D" is masked), the calculation continues; but the "D" error bit remains set. If the 8087 is in normalizing mode ("D" is unmasked), the "D" trap handler is called. The trap handler is called directly from the interior of mqerMIN; the 8087 opcode register contains the code for the FCOM instruction which caused the "D" error. The (y) and (x) inputs are left on the 8087 stack. Most trap handlers will replace the denormal input(s) with normalized numbers, reexecute the FCOM or FSUB instruction, and continue through mqerMIN. The trap handler provided by EH87.LIB, described in Chapter 5, will do this and replace denormals with zero.

mqerMIN next checks for NaN inputs. If either input is a NaN, an "I" error is given. If "I" is masked, the answer returned is the input NaN (the larger NaN if both inputs are NaN's).

An "I" error is also given if the 8087 chip is in projective (unsigned infinity) mode, and exactly one of the inputs is infinite. In this case a masked 'I' yields the value INDEFINITE for an answer.

For all "I" errors, when "I" is unmasked, the trap handler is called with the inputs still on the 8087 stack, and the 8087 opcode register set to 265 hex.

PL/M-86 usage example:

```
mqerMIN: PROCEDURE (Y,X) REAL EXTERNAL;
  DECLARE (Y,X) REAL;
END mqerMIN;

DECLARE POPULATIONS(10) REAL;
DECLARE SMALLEST REAL;
DECLARE N BYTE;

/* The following code sets SMALLEST to the smallest of the
   ten values in the array POPULATIONS. */

SMALLEST = POPULATIONS(0);
DO N = 1 TO 9;
  SMALLEST = mqerMIN(SMALLEST,POPULATIONS(N));
END;
```

ASM-86 usage example:

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs:
EXTRN mqerMIN: FAR

VAR1          DQ  -5.3
VAR2          DQ  -7.9
                    ; the above initializations are test
                    ; values
SMALLEST       DQ  ?

; The following code sets SMALLEST to the minimum of VAR1
; and VAR2.

     FLD VAR1            ; load the first parameter onto
                         ; the 8087 stack
     FLD VAR2            ; load the second parameter
     CALL mqerMIN        ; stack now contains the minimum
     FSTP SMALLEST       ; minimum is stored, and 8087
                         ; stack is popped

; With the test inputs, SMALLEST is now -7.9
```

mqerMOD x = (y mod x), same sign as (y)

Input parameters: (x) is the top number on the 8087 stack; (y) is the next number on the 8087 stack.

Function: mqerMOD returns the answer (y - (x * mqerICX(y/x)). In more intuitive terms, this is the "remainder" left when (y) is divided by (x). The answer is always exact; there is no roundoff error.

mqerMOD always returns the same values when (x) is negative as for the corresponding positive input (−x). Whenever (x) is mentioned in the following paragraphs, we are really talking about the absolute value of (x).

mqerMOD is calculated by subtracting an integer multiple of (x) from the input (y), to bring it down to within (x) units of zero. The choice of which integer multiple to subtract determines the range of possible values the function can yield. For mqerMOD, the integer is mqerICX(y/x), which is the value (y/x) chopped towards zero. If (y) is postive, the answer is greater than or equal to 0, and less than (x). If (y) is negative, the answer is greater than (−x), and less than or equal to 0.

For example, suppose (x) equals either −5 or 5. Then for negative y mqerMOD(y,5) gives values in the range between −5 and 0. mqerMOD(−7,5) is −2; mqerMOD(−10,5) is 0; mqerMOD(−19.99,5) is −4.99. For positive y, mqerMOD(y,5) gives values in the range between 0 and 5. mqerMOD(2,5) is 2; mqerMOD(45,5) is 0; mqerMOD(44.75,5) is 4.75.

It is legal to have infinite (x) inputs. In that case, mqerMOD simply returns (y), if (y) is finite and normal. If (y) is unnormal, the normalized (y) is returned. If (y) is denormal, the result depends on the setting of the 8087's normalization mode, as determined by the "D" error masking flag. If in normalizing mode ("D" unmasked), the result is 0 with no error. If in warning mode ("D" masked), the result is the unchanged denormal (y), also with no error.

It is often legal to have unnormal and denormal (y) inputs. The cases with infinite (x) are discussed in the above paragraph. If (y) is unnormal, and if the normalization of (y) does not produce a denormal, then (y) is legal, and the normalized input is used.

Output: The 8087 stack pops once, with the answer replacing the two inputs.

Errors: First, the inputs are checked to see if either is a NaN. If so, an "I" error is given. If "I" is masked, the input NaN is returned (if both inputs are NaN's the larger NaN is returned).

If (x) is unnormal, denormal, or any zero value, an "I" error is given. Also, if (y) is infinite, an "I" error is given. If "I" is masked, the value INDEFINITE is returned.

If "I" is unmasked for any of the above errors, the trap handler is called with the inputs still on the 8087 stack, and the number 269 hex in the 8087 opcode register.

A "U" error is given when (y) is unnormal, and the normalization of (y) produces a denormal. A "U" error is also given if (y) is denormal. If "U" is masked, an answer of 0 is returned. If "U" is unmasked, the trap handler is called, with the value 169 hex placed in the 8087 opcode register; but the inputs are not on the 8087 stack. Instead, the correct answer is given, with a "wrapped" exponent. To obtain the correct exponent, subtract the decimal number 24576 from the given exponent.

PL/M-86 usage example:

```
mqerMOD: PROCEDURE (Y,X) REAL EXTERNAL;
  DECLARE (Y,X) REAL;
END mqerMOD;

DECLARE Y REAL;
DECLARE LAST_THREE_DIGITS REAL;

Y = 456789.00;  /* Test value * /

/* The following line sets LAST_THREE_DIGITS to Y MOD
   1000.  If Y is a positive integer, then the answer is
   the number formed by the last three decimal digits of
   Y. */

LAST_THREE_DIGITS = mqerMOD(y,1000.);

/* With the test value, LAST_THREE_DIGITS is now 789.00 */
```

ASM-86 usage example:

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs:
EXTRN mqerMOD: FAR

LAST_THREE_DIGITS   DQ  ?
Y                   DQ  181137.00  ; initialization is a
                                   ; test value
ONE_GRAND           DD  1000.00    ; constant for
                                   ; calculation below

; The following code calculates Y MOD 1000, as in the PL/M
; example above, except with LONG_REAL variables

   FLD Y               ; load first parameter onto 8087
                       ; stack
   FLD ONE_GRAND       ; load modulus 1000 onto 8087 stack
   CALL mqerMOD        ; take the modulus
   FSTP LAST_THREE_DIGITS   ; store answer and pop the
                            ; 8087 stack

; With the test value, LAST_THREE_DIGITS is 137.00
```

mqerRMD x = (y mod x), close to 0

Input parameters: (x) is the top number on the 8087 stack; (y) is the next number on the 8087 stack.

Function: mqerRMD returns the answer (y - (x * mqerIEX(y/x)). In more intuitive terms, this is the "remainder" left when (y) is divided by (x). The answer is always exact; there is no roundoff error.

mqerRMD always returns the same values when (x) is negative as for the corresponding positive input (−x). Whenever (x) is mentioned in the following paragraphs, we are really talking about the absolute value of (x).

mqerRMD is calculated by subtracting an integer multiple of (x) from the input (y), to bring it down to within (x) units of zero. The choice of which integer multiple to subtract determines the range of possible values the function can yield. For mqerRMD, the integer is mqerIEX(y/x), which is the value (y/x) rounded to the nearest integer. Thus the range of answers is from (−x/2) to (x/2).

For example, suppose (x) equals either −5 or 5. Then the value of mqerRMD(y,5) ranges from −2.5 to 2.5. mqerRMD(−7,5) is −2; mqerRMD(−10,5) is 0; mqerRMD(−19.99,5) is +0.01; mqerRMD(2,5) is 2; mqerRMD(4,5) is −1; mqerRMD(44.75,5) is −0.25.

When the input (y) is an odd integer multiple of (x/2), the answer returned by mqerRMD can be either (x/2) or (−x/2). The number chosen is determined by the convention of mqerIEX, which rounds to the even integer in case of a tie. This results in values alternating between (−x/2) and (x/2). Inputs (y) which yield (x/2) form the series {... −7x/2, −3x/2, x/2, 5x/2, 9x/2, ... }. Inputs (y) which yield (−x/2) form the series {... −5x/2, −x/2, 3x/2, 7x/2, 11x/2, ... }. For example, mqerRMD(2.5,5) is 2.5; mqerRMD(7.5,5) is −2.5.

It is legal to have infinite (x) inputs. In that case, mqerRMD simply returns (y), if (y) is finite and normal. If (y) is unnormal, the normalized (y) is returned. If (y) is denormal, the result depends on the setting of the 8087's normalization mode, as determined by the "D" error masking flag. If in normalizing mode ("D" unmasked), the result is 0 with no error. If in warning mode ("D" masked), the result is the unchanged denormal (y), also with no error.

It is often legal to have unnormal and denormal (y) inputs. The cases with infinite (x) are discussed in the above paragraph. If (y) is unnormal, and if the normalization of (y) does not produce a denormal, then (y) is legal, and the normalized input is used.

Output: The 8087 stack pops once, with the answer replacing the two inputs.

Errors: First, the inputs are checked to see if either is a NaN. If so, an "I" error is given. If "I" is masked, the input NaN is returned (if both inputs are NaN's the larger NaN is returned).

If (x) is unnormal, denormal, or any zero value, an "I" error is given. Also, if (y) is infinite, an "I" error is given. If "I" is masked, the value INDEFINITE is returned.

If "I" is unmasked for any of the above errors, the trap handler is called with the inputs still on the 8087 stack, and the number 27A hex in the 8087 opcode register.

A "U" error is given when (y) is unnormal, and the normalization of (y) produces a denormal. A "U" error is also given if (y) is already denormal. If "U" is masked, an answer of 0 is returned. If "U" is unmasked, the trap handler is called, with the value 17A hex placed in the 8087 opcode register; but the inputs are not on the 8087 stack. Instead, the correct answer is given, with a "wrapped" exponent. To obtain the correct exponent, subtract the decimal number 24576 from the given exponent.

PL/M-86 usage example:

```
mqerRMD: PROCEDURE (Y,X) REAL EXTERNAL;
  DECLARE (Y,X) REAL;
END mqerRMD;

DECLARE TWO_PI LITERALLY '6.28318530717958647692S';
     /* 2 * PI -- a full circle expressed in radians */

DECLARE THETA REAL;  /* angle to be reduced */

THETA = 6.;  /* Test value */

/* The following line reduces THETA to a principal value
   -- a value between -PI and PI.  */

THETA = mqerRMD(THETA, TWO_PI);

/* Now THETA is 6 radians, reduced to the principal value:
        about -0.2831853 */
```

ASM-86 usage example:

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs:
EXTRN mqerRMD: FAR

THETA        DQ  -6.00    ; initialization is a test value

; The following code performs the same reduction of an
; angle to a principal value as the PL/M code above,
; except with a LONG_REAL variable.

   FLD THETA             ; angle parameter onto 8087 stack
   FLDPI                 ; constant PI onto stack
   FADD ST,ST            ; 2 * PI
   CALL mqerRMD          ; modulus is taken
   FSTP THETA            ; principal value is stored, 8087
                         ; stack is popped

; With the test value, THETA is now about 0.2831853
```

mqerSGN x = (y with x's sign)

Input parameters: (x) is the top number on the 8087 stack; (y) is the next number on the 8087 stack.

Function: If (x) is greater than or equal to zero, mqerSGN returns the absolute value of (y). If (x) is less than zero, mqerSGN returns the negative of the absolute value of (y).

The positive absolute value of (y) is returned for all values of (x) which are zeroes or pseudo-zeroes; even if (x) is equivalent to −0.

Unnormal values of (x) are legal. If (x) is not a pseudo-zero, only the sign of (x) is relevant to the final answer.

Infinite values of (x) are allowed. The sign of the infinity determines the sign on the answer, even when the 8087 is in projective (unsigned infinity) mode.

Any input (y) is legal, including NaN's, unnormals, denormals, and infinities. The only part of (y) which might be changed upon output is the sign.

Output: The 8087 stack pops once, with the answer replacing the two inputs.

Errors: If (x) is a denormal, the "D" error is given by an FTST instruction within the interior of mqerSGN. If the 8087 is in warning mode ("D" is masked), mqerSGN will use the denormal to determine the sign of the answer. If the 8087 is in normalizing mode ("D" is unmasked), the "D" trap handler will be called with the input still on the 8087 stack. Most trap handlers will normalize the argument, reperform the FTST instruction, and continue with the computation of mqerSGN. The trap handler provided by EH87.LIB, described in Chapter 5, will replace the denormal with 0. Thus, the absolute value of (y) will be returned by mqerSGN.

If (x) is a NaN, an "I" error results. If "I" is masked, (x) is returned. If "I" is unmasked, the trap handler is called with the inputs still on the 8087 stack; and the 8087 opcode register set to 264 hex.

PL/M-86 usage example:

```
mqerSGN: PROCEDURE (Y,X) REAL EXTERNAL;
  DECLARE (Y,X) REAL;
END mqerSGN;

DECLARE THETA REAL;
DECLARE Y_COOR REAL;
DECLARE PI LITERALLY '3.14159265358979323';

Y_COOR = -0.0000001;  /* Test value */

/* The following code returns either the value PI or -PI.
   If Y_COOR is positive, it returns PI.  If Y_COOR is
   negative, it returns -PI. */

THETA = mqerSGN(PI,Y_COOR);

/* With the test value, THETA now is -PI. */
```

ASM-86 usage example:

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs:
EXTRN mqerSGN: FAR

THETA          DQ  ?
Y_COOR         DQ  -0.001    ; initialized to a test value

; The following code  -0.001    ; initialized to a test
; value

; The following code returns PI with the sign of Y_COOR,
; just as in the PL/M example above.

   FLDPI           ; first parameter PI onto 8087 stack
   FLD Y_COOR      ; second parameter Y_COOR onto 8087
                   ; stack
   CALL mqerSGN    ; combine sign of Y_COOR with magnitude
                   ; of PI
   FSTP THETA      ; store answer and pop the 8087 stack

; With the test case, THETA is now PI.
```

mqerSIN x = sine(x)

Input parameter: (x) is the top number on the 8087 stack.

Function: mqerSIN returns the trigonometric sine of x, where x is an angle expressed in radians. All input zeroes, pseudo-zeroes, and denormals return the input value. Also, unnormals whose value is less than 2^{-63} return the input value.

Output: The answer replaces the input on the 8087 stack.

Errors: An "I" error is given for input infinities and NaN's. An "I" error is also given for unnormals which do not represent values less than 2^{-63}.

If "I" is unmasked, the trap handler is called with the input still on the stack, and the 8087 opcode register set to 171 hex. If "I" is masked, the answer is the input for NaNs; the answer is the value INDEFINITE for other invalid inputs.

PL/M-86 usage example:

```
mqerSIN: PROCEDURE (THETA) REAL EXTERNAL;
  DECLARE THETA REAL;
END mqerSIN;

DECLARE (POLAR_R, POLAR_THETA) REAL;
DECLARE REC_Y REAL;
DECLARE PI LITERALLY '3.14159265358979';
DECLARE DEG_TO_RAD LITERALLY 'PI/180.';

POLAR_R = 2.; POLAR_THETA = 30.;    /* Test values */

/* The following line computes the Y-coordinate of a
   polar-to-rectangular conversion.  The input angle is in
   degrees, so it must be converted to radians. */

REC_Y = POLAR_R * mqerSIN(POLAR_THETA * DEG_TO_RAD);

/* Now in the test case, REC_Y = 1. */
```

ASM-86 usage example:

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs:
EXTRN mqerSIN: FAR

POLAR_THETA    DQ  30.0
POLAR_R        DQ  2.0
                    ; the above initializations are test
                    ; values.
REC_Y          DQ  ?
DEG_TO_RAD     DT  3FF98EFA351294E9C8AER    ; the constant
                                            ; PI/180.
; The following lines compute the Y-coordinate of a
; polar-to-rectangular conversion, as in the PL/M
; example above; except that the variables are
; LONG_REAL.

FLD POLAR_THETA     ; degrees angle onto 8087 stack
FLD DEG_TO_RAD
FMUL                ; converted to radians
CALL mqerSIN        ; sine is taken
FMUL POLAR_R        ; answer scaled to correct radius
FSTP REC_Y          ; Y-coordinate stored and stack is
                    ; popped

; With the test case, REC_Y is now 1.
```

mqerSNH x = hyperbolic sine(x)

Input parameter: (x) is the top number on the 8087 stack.

Function: mqerSNH returns the hyperbolic sine of x, where x is an angle expressed in radians. All input zeroes, pseudo-zeroes, and denormals return the input value. Also, unnormals whose value is less than 2^{-63} return the input value.

Infinite inputs are legal, and return the input value.

Output: The answer replaces the input on the 8087 stack.

Errors: An "I" error is given for input NaN's. An "I" error is also given for unnormals which do not represent values less than 2^{-63}.

If "I" is unmasked, the trap handler is called with the input still on the stack. If "I" is masked, the answer is the input for NaNs; the answer is the value INDEFINITE for other invalid inputs.

mqerSNH will give an "O" overflow error if the input is greater than about 11355. When "O" is masked, the value +INFINITY is returned. Likewise, "O" is given for inputs less than about −11355, with −INFINITY returned for masked "O". When "O" is unmasked, the trap handler is called with the input still on the 8087 stack.

When either trap handler is called, mqerSNH first sets the 8087 opcode register to 16E hex.

PL/M-86 usage example:

```
mqerSNH: PROCEDURE (THETA) REAL EXTERNAL;
  DECLARE THETA REAL;
END mqerSNH;

DECLARE (INPUT_VALUE, OUTPUT_VALUE) REAL;

INPUT_VALUE = 2.7;    /* Test value */

OUTPUT_VALUE = mqerSNH(INPUT_VALUE);

/* Now with the test input, OUTPUT_VALUE is about
   14.812526 */
```

ASM-86 usage example:

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs:
EXTRN mqerSNH: FAR

INPUT_VALUE     DQ  -2.7    ; initialization is a test
                            ; value
OUTPUT_VALUE    DQ  ?
```

```
    ; The following code duplicates the above PL|M
    ; assignment statement,
    ;   except with LONG_REAL variables.

    FLD INPUT_VALUE       ; load the parameter onto the 8087
                          ; stack
    CALL mqerSNH          ; take the hyperbolic sine
    FSTP OUTPUT_VALUE     ; store the answer and pop the
                          ; 8087 stack

; With the test input, OUTPUT_VALUE is now about
; -14.812526
```

mqerTAN x = tangent(x)

Input parameter: (x) is the top number on the 8087 stack.

Function: mqerTAN returns the trigonometric tangent of x, where x is an angle expressed in radians. All input zeroes, pseudo-zeroes, and denormals return the input value. Also, unnormals whose value is less than 2^{-63} return the input value.

Output: The answer replaces the input on the 8087 stack.

Errors: An "I" error is given for input infinities and NaN's. An "I" error is also given for unnormals which do not represent values less than 2^{-63}.

If "I" is unmasked, the trap handler is called with the input still on the stack, and the 8087 opcode register set to 173 hex. If "I" is masked, the answer is the input for NaNs; the answer is the value INDEFINITE for other invalid inputs.

A "Z" error is given when the input number is an exact odd multiple of the closest TEMP__REAL number to $\pi/2$. When "Z" is masked, the answer +INFINITY is returned. When "Z" is unmasked, the trap handler is called with the input still on the 8087 stack, and the 8087 opcode register set to 173 hex.

PL/M-86 usage example:

```
mqerTAN: PROCEDURE (THETA) REAL EXTERNAL;
  DECLARE THETA REAL;
END mqerTAN;

DECLARE PI LITERALLY '3.14159265358979';
DECLARE DEG_TO_RAD LITERALLY 'PI/180.';
DECLARE THETA_DEGREES REAL;
DECLARE SLOPE REAL;

THETA_DEGREES = 135.0;    /* Test value */

/* The following line computes the tangent of the angle
   THETA_DEGREES. The answer is called SLOPE because it is
   the slope of a line which is displaced by THETA_DEGREES
   from the X-axis. */

SLOPE = mqerTAN(THETA_DEGREES * DEG_TO_RAD);

/* Now with the test value, SLOPE = -1. */
```

ASM-86 usage example:

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs:
EXTRN mqerTAN: FAR

THETA_DEGREES      DQ  45.00      ; initialization is a test
                                  ; value
SLOPE              DQ  ?
DEG_TO_RAD    DT  3FF98EFA351294E9C8AER   ; the constant
                                          ; PI/180.
```

```
    ; The following code computes the tangent just as in
    ; the PL/M example above, except with LONG_REAL
    ; variables.

    FLD THETA_DEGREES       ; load the first parameter onto
                            ; the 8087 stack
    FLD DEG_TO_RAD
    FMUL                    ; convert from degrees to radians
    CALL mqerTAN            ; take the tangent of the radians
                            ; value
    FSTP SLOPE              ; store the answer and pop the
                            ; 8087 stack

; With the test input, SLOPE is now 1.00
```

mqerTNH x = hyperbolic tangent(x)

Input parameter: (x) is the top number on the 8087 stack.

Function: mqerTNH returns the hyperbolic tangent of x, where x is an angle expressed in radians. All input zeroes, pseudo-zeroes, and denormals return the input value. Also, unnormals whose value is less than 2^{-63} return the input value.

Infinite inputs are allowed. Input +INFINITY gives an answer of +1; −INFINITY gives −1. The sign of the infinity is significant even if the 8087 is in projective (unsigned infinity) mode.

Output: The answer replaces the input on the 8087 stack.

Errors: An "I" error is given for input NaN's. An "I" error is also given for unnormals which do not represent values less than 2^{-63}.

If "I" is unmasked, the trap handler is called with the input still on the stack, and the 8087 opcode register set to 170 hex. If "I" is masked, the answer is the input for NaNs; the answer is the value INDEFINITE for illegal unnormals.

PL/M-86 usage example:

```
mqerTNH: PROCEDURE (THETA) REAL EXTERNAL;
  DECLARE THETA REAL;
END mqerTNH;

DECLARE (INPUT_VALUE, OUTPUT_VALUE) REAL;

INPUT_VALUE = 0.62;    /* Test value */

OUTPUT_VALUE = mqerTNH(INPUT_VALUE);

/* Now with the test input, OUTPUT_VALUE is about
   0.55112803 */
```

ASM-86 usage example:

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs:
EXTRN mqerTNH: FAR

INPUT_VALUE     DQ  -0.62    ; initialization is a test
                             ; value
OUTPUT_VALUE    DQ  ?

   ; The following code duplicates the above PL/M
   ; assignment statement, except with LONG_REAL
   ; variables.
```

```
    FLD INPUT_VALUE       ; load the parameter onto the 8087
                          ; stack
    CALL mqerTNH          ; take the hyperbolic tangent
    FSTP OUTPUT_VALUE     ; store the answer and pop the
                          ; 8087 stack

; With the test input, OUTPUT_VALUE is now about
; -0.55112803
```

mqerY2X $x = y^x$

Input parameters: (x) is the top number on the 8087 stack; (y) is the next number on the 8087 stack.

Function: mqerY2X computes (y) to the (x) power; neither (y) nor (x) is required to be an integer. For most inputs, the formula used is $2^{(x*LG2(y))}$.

The base (y) must be positive for the logarithmic formula to have any meaning. There are some cases, however, in which (y) is allowed to be non-positive.

- If (x) is positive, a zero (y) gives a zero answer, with no error.
- If (x) is zero, (y) can be negative; the answer is 1.
- If (x) is an integer, (y) can be negative. The function is evaluated using the absolute value of (y). The result is the answer if (x) is even; it is the negative of the answer if (x) is odd.

Inputs which are zero, infinite, unnormal, or denormal are accepted under certain conditions.

When either input is denormal, it is replaced with an alternate value. The value selected depends on the setting of the 8087 "D" masking bit. If the 8087 is in normalizing mode ("D" is unmasked), the input is replaced by 0. If the 8087 is in warning mode ("D" is masked), the input is replaced by the equivalent unnormal. Note that even though mqerY2X references the "D" masking bit, it never gives a "D" error, and it never calls the "D" trap handler.

An unnormal (y) input is legal only if (x) is a normal integer or infinite value. If (x) is infinite, the function is evaluated as if (y) were the equivalent normal value. If (x) is zero, (y) must be nonzero; in that case, the answer is 1. If (x) is any other integer, it must fit into 32 bits; in that case, the function mqerYI4 is called to obtain the answer.

An unnormal (x) input is legal only if it is non-zero, and if (y) is infinite or zero. In those cases, (x) is replaced by its normal equivalent.

When the 8087 is in affine (signed infinity) mode, there are a number of cases in which mqerY2X allows infinite inputs:

- If (y) is −INFINITY, then (x) must be a non-zero integer. The magnitude of the answer is then INFINITY if (x) is positive; zero if (x) is negative. The sign of the answer is positive if (x) is even; negative if (x) is odd.
- If (y) is +INFINITY, then any non-zero (x) is legal. The answer is +INFINITY if (x) is positive; zero if (x) is negative.
- If (x) is +INFINITY, then (y) must be positive or any zero, and not equal to 1. The answer is zero if (y) is less than 1, and +INFINITY if (y) is greater than 1.
- If (x) is −INFINITY, then (y) must likewise be positive or any zero, and not equal to 1. The answer is +INFINITY if (y) is less than 1, and −INFINITY if (y) is greater than 1.

When the 8087 chip is in projective (unsigned infinity) mode, there is only one case in which any infinite input is allowed. This occurs when (y) is infinite, and (x) is a non-zero integer. The result is the same as if the 8087 were in affine mode.

Output: The 8087 stack pops once, with the answer replacing the two inputs.

Errors: mqerY2X first checks for NaN inputs. If either input is a NaN, an "I" error is given. If "I" is masked, the input NaN is returned (the larger NaN is returned if both inputs are NaN's).

The legal cases involving unnormal inputs, infinite inputs, and negative (y) inputs are described above. Illegal cases yield an "I" error. If "I" is masked, the value INDEFINITE is returned. The case (y = 1) and (x infinite), among others, falls into this category.

It is illegal for both (x) and (y) to have zero values. This too gives an "I" error, with an INDEFINITE answer if "I" is masked.

If (y) is any zero and (x) is any negative value (including negative infinity), then a "Z" error is given. If "Z" is masked, the value +INFINITY is returned.

The "O" overflow or "U" underflow error occurs when (y^x) cannot be represented by the TEMP_REAL exponent. If "O" is masked, an overflow will return the answer +INFINITY. In underflow cases, if "U" is masked, the correct answer is given if it can be represented by a denormal; otherwise, 0 is given.

All of the errors, "I", "Z", "O", and "U", will cause the trap handler to be called when the corresponding exception bit is unmasked. mqerY2X leaves the input numbers on the 8087 stack, and places the value 26A hex into the 8087 opcode register before calling the trap handler.

PL/M-86 usage example:

```
mqerY2X: PROCEDURE (Y,X) REAL EXTERNAL;
  DECLARE (Y,X) REAL;
END mqerY2X;

DECLARE CUBE_ROOT REAL;
DECLARE INPUT_VALUE REAL;

INPUT_VALUE = 17.00;    /* Test value */

/* The following line takes the cube root of the positive
   value INPUT_VALUE. */

CUBE_ROOT = mqerY2X(INPUT_VALUE, 1./3.);

/* With the test input, INPUT_VALUE is now about
   2.5712816 */
```

ASM-86 usage example:

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs:
EXTRN mqerY2X: FAR

INPUT_VALUE     DQ  64.00    ; initialization is a test
                             ; value
CUBE_ROOT       DQ  ?
ONE_THIRD       DT  0.3333333333333333333  ; constant
                                           ; used below
; The following lines take the cube root just as in the
; PL/M example above, except with LONG_REAL variables.

   FLD INPUT_VALUE          ; load first parameter onto
                            ; 8087 stack
   FLD ONE_THIRD            ; load second parameter onto
                            ; 8087 stack
   CALL mqerY2X             ; exponentiate
   FSTP CUBE_ROOT           ; store the answer and pop the
                            ; 8087 stack

; With the test input, CUBE_ROOT is now about 4.00
```

mqerYI2 x = x ** AX

Input parameters: (x) is the top number on the 8087 stack. The power to which (x) is raised is the 8086 AX register, interpreted as a twos-complement signed integer.

Function: mqerYI2 raises the real input (x) to an integer power. If the integer is zero, the answer is 1. In this case, no error is given, even if (x) is a NaN.

If AX is not zero, the input (x) is first checked for unusual values.

If (x) is +INFINITY, the answer is +INFINITY for positive AX; +0 for negative AX. There is no error.

If (x) is −INFINITY, the magnitude of the answer is INFINITY for positive AX; 0 for negative AX. The sign of the answer is positive for even AX; negative for odd AX.

Zero (x) input is legal if AX is positive. The answer is −0 if (x) is −0 and AX is odd; the answer is +0 in all other cases.

If (x) is denormal, no error is given. However, the 8087 "D" error masking bit is checked to see what action to take. If the 8087 is in normalizing mode ("D" is unmasked), (x) is replaced by zero. If the 8087 is in warning mode ("D" is masked), (x) is replaced by the unnormal number with the same numeric value as the denormal. The evaluation of mqerYI2 proceeds with the new (x).

If (x) is unnormal and AX is negative, then (1/x) is computed, preserving the number of bits of unnormalization. Then the positive power is computed by successive squaring and multiplying by (x).

If (x) is unnormal and AX is positive, then the power is computed by successive squaring and multiplying by (x).

For normal, non-zero values of (x), computation of the power proceeds according to the value of the integer power AX.

If the integer power is 64 or greater, or −64 or less, the answer is computed with logarithms. The answer is 2 ** (AX * LG2(x)).

If the integer power is from 1 to 63, the answer is computed by successively squaring and multiplying by (x) to achieve the correct power.

If the integer power is from −63 to −1, mqerYI2 determines if any exceptions would occur if the expression 1 / (x * x * ... x) were evaluated. If not, the expression is evaluated and the answer is returned. If so, then the expression (1/x) * (1/x) * ... * (1/x) is evaluated. If the second expression causes exceptions, the trap handler is called.

The maximum number of multiplications performed for any of the above squaring-and-multiplying algorithms is 9.

Output: The answer replaces the input (x) on the 8087 stack. The AX input is destroyed, as allowed by PL/M-86 procedure conventions.

Errors: As stated above, there can be no errors if AX is 0. Otherwise, errors occur in the following cases:

If (x) is a NaN, an "I" error is given. If "I" is masked, the input NaN is returned as the answer.

A "U" underflow error occurs when the answer computed is too close to zero to be represented by the exponent. If "U" is masked, the answer is replaced by the equivalent denormal if it exists; 0 otherwise.

An "O" overflow error occurs when the magnitude of the answer is too great to be represented by the exponent. If "O" is masked, the answer is INFINITY with the appropriate sign.

If any of the "I", "O", or "U" errors occurs with the error unmasked, the trap handler is called. Before calling the trap handler, mqerYI2 sets the 8087 opcode register to 27C hex and leaves the inputs on the 8087 stack. The integer power is converted to TEMP__REAL and pushed onto the top of the 8087 stack. The base (the original (x)) becomes the second stack element.

PL/M-86 usage example:

For PL/M-86, the 16-bit input parameter should be on the 8086 stack rather than in the AX register. Therefore, use mqerYIS (instead of mqerYI2) in PL/M-86 programs.

ASM-86 usage example:

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs:
EXTRN mqerYI2: FAR

POWER           DW  9     ; exponent which will be used
REAL_BASE       DQ  1.3   ; number which will be raised to
                          ; POWER
                   ; the above initializations are test
                   ; values
REAL_OUTPUT     DQ  ?

; The following code multiplies POWER copies of REAL_BASE
; together, and stores the answer in REAL_OUTPUT.

   FLD REAL_BASE        ; base parameter goes onto the 8087
                        ; stack
   MOV AX,POWER         ; exponent goes into the AX
                        ; register
   CALL mqerYI2         ; REAL_BASE ** POWER is now on 8087
                        ; stack
   FSTP REAL_OUTPUT     ; store the answer and pop the 8087
                        ; stack

; With the test inputs, REAL_OUTPUT is now about 10.604499
```

mqerYI4 x = x ** DXAX

Input parameters: (x) is the top number on the 8087 stack. The power to which (x) is raised is a 32-bit twos complement value in the 8086 DX and AX registers. DX is the most significant half, and AX is the least significant half.

Function: mqerYI2 raises the real input (x) to an integer power. The input integer is presented in a 32-bit format. Note, however, that 32 bits are rarely necessary to represent an integer exponent. Raising a number to a power greater than 32768 rarely gives meaningful results. Thus mqerYI2 is sufficient for almost every application in which mqerYI4 might be used. We have provided mqerYI4 mainly for compatibility with the 32-bit integer types found in Pascal-86 and FORTRAN-86.

If the input integer power is zero, the answer is 1, no matter what (x) is. In this case, no error is given, even if (x) is a NaN.

If DXAX is not zero, the input (x) is first checked for unusual values.

If (x) is +INFINITY, the answer is +INFINITY for positive DXAX; +0 for negative DXAX. There is no error.

If (x) is −INFINITY, the magnitude of the answer is INFINITY for positive DXAX; 0 for negative DXAX. The sign of the answer is positive for even DXAX; negative for odd DXAX.

Zero (x) input is legal if DXAX is positive. The answer is −0 if (x) is −0 and DXAX is odd; the answer is +0 in all other cases.

If (x) is denormal, no error is given. However, the 8087 "D" error masking bit is checked to see what action to take. If the 8087 is in normalizing mode ("D" is unmasked), (x) is replaced by zero. If the 8087 is in warning mode ("D" is masked), (x) is replaced by the unnormal number with the same numeric value as the denormal. The evaluation of mqerYI2 proceeds with the new (x).

If (x) is unnormal and DXAX is negative, then (1/x) is computed, preserving the number of bits of unnormalization. Then the positive power is computed by successive squaring and multiplying by (x).

If (x) is unnormal and DXAX is positive, then the power is computed by successive squaring and multiplying by (x).

For normal, non-zero values of (x), computation of the power proceeds according to the value of the integer power DXAX.

If the integer power is 64 or greater, or −64 or less, the answer is computed with logarithms. The answer is 2 ** (DXAX * LG2(x)).

If the integer power is from 1 to 63, the answer is computed by successively squaring and multiplying by (x) to achieve the correct power.

If the integer power is from −63 to −1, mqerYI4 determines if any exceptions would occur if the expression 1 / (x * x * ... x) were evaluated. If not, the expression is evaluated and the answer is returned. If so, then the expression (1/x) * (1/x) * ... * (1/x) is evaluated. If the second expression causes exceptions, the trap handler is called.

The maximum number of multiplications performed for any of the above squaring-and-multiplying algorithms is 9.

Output: The answer replaces the input (x) on the 8087 stack. The DXAX input is destroyed, as allowed by PL/M-86 procedure conventions.

Errors: As stated above, there can be no errors if DXAX is 0. Otherwise, errors occur in the following cases:

If (x) is a NaN, an "I" error is given. If "I" is masked, the input NaN is returned as the answer.

A "U" underflow error occurs when the answer computed is too close to zero to be represented by the exponent. If "U" is masked, the answer is replaced by the equivalent denormal if it exists; 0 otherwise.

A "O" overflow error occurs when the magnitude of the answer is too great to be represented by the exponent. If "O" is masked, the answer is INFINITY with the appropriate sign.

If any of the "I", "O", or "U" errors occurs with the error unmasked, the trap handler is called. Before calling the trap handler, mqerYI2 sets the 8087 opcode register to 27C hex and leaves the inputs on the 8087 stack. The integer power is converted to TEMP_REAL and pushed onto the top of the 8087 stack. The base (the original (x)) becomes the second stack element.

PL/M-86 usage example:

Since PL/M-86 does not support a 32-bit integer data type, mqerYI4 cannot be used by PL/M programs. You should use either mqerYIS or mqerY2X.

ASM-86 usage example:

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs:
EXTRN mqerYI4: FAR

POWER          DD  9     ; exponent which will be used
REAL_BASE      DQ  1.3   ; number which will be raised to
                         ; POWER
                         ; the above initializations are
                         ; test values
REAL_OUTPUT    DQ  ?

; The following code multiplies POWER copies of REAL_BASE
; together, and stores the answer in REAL_OUTPUT.

   FLD REAL_BASE                  ; base parameter goes onto
                                  ; the 8087 stack
   MOV AX, WORD PTR POWER         ; low 16 bits of exponent
                                  ; to AX
   MOV DX, WORD PTR (POWER+2)     ; ---high 16 bits to DX
   CALL mqerYI4                   ; REAL_BASE ** POWER is
                                  ; now on 8087 stack
   FSTP REAL_OUTPUT               ; store the answer and pop
                                  ; the 8087 stack

; With the test inputs, REAL_OUTPUT is now about 10.604499
```

mqerYIS x = x ** STK

Input parameters: (x) is the top number on the 8087 stack. The power STK to which (x) is to be raised is a 16-bit twos-complement integer that is pushed onto the stack before mqerYIS is called.

Function: mqerYIS raises the real input (x) to an integer power. If the integer is zero, the answer is 1. In this case, no error is given, even if (x) is a NaN.

If STK is not zero, the input (x) is first checked for unusual values.

If (x) is +INFINITY, the answer is +INFINITY for positive STK; +0 for negative STK. There is no error.

If (x) is −INFINITY, the magnitude of the answer is INFINITY for positive STK; 0 for negative STK. The sign of the answer is positive for even STK; negative for odd STK.

Zero (x) input is legal if STK is positive. The answer is −0 if (x) is −0 and STK is odd; the answer is +0 in all other cases.

If (x) is denormal, no error is given. However, the 8087 "D" error masking bit is checked to see what action to take. If the 8087 is in normalizing mode ("D" is unmasked), (x) is replaced by zero. If the 8087 is in warning mode ("D" is masked), (x) is replaced by the unnormal number with the same numeric value as the denormal. The evaluation of mqerYIS proceeds with the new (x).

If (x) is unnormal and STK is negative, then (1/x) is computed, preserving the number of bits of unnormalization. Then the positive power is computed by successive squaring and multiplying by (x).

If (x) is unnormal and STK is positive, then the power is computed by successive squaring and multiplying by (x).

For normal, non-zero values of (x), computation of the power proceeds according to the value of the integer power STK.

If the integer power is 64 or greater, or −64 or less, the answer is computed with logarithms. The answer is 2 ** (STK * LG2(x)).

If the integer power is from 1 to 63, the answer is computed by successively squaring and multiplying by (x) to achieve the correct power.

If the integer power is from −63 to −1, mqerYIS determines if any exceptions would occur if the expression 1 / (x * x * ... x) were evaluated. If not, the expression is evaluated and the answer is returned. If so, then the expression (1/x) * (1/x) * ... * (1/x) is evaluated. If the second expression causes exceptions, the trap handler is called.

The maximum number of multiplications performed for any of the above squaring-and-multiplying algorithms is 9.

Output: The answer replaces the input (x) on the 8087 stack. mqerYIS returns with the value STK popped off the 8086 stack, so it no longer exists.

Errors: As stated above, there can be no errors if STK is 0. Otherwise, errors occur in the following cases:

If (x) is a NaN, an "I" error is given. If "I" is masked, the input NaN is returned as the answer.

A "U" underflow error occurs when the answer computed is too close to zero to be represented by the exponent. If "U" is masked, the answer is replaced by the equivalent denormal if it exists; the answer is 0 otherwise.

An "O" overflow error occurs when the magnitude of the answer is too great to be represented by the exponent. If "O" is masked, the answer is INFINITY with the appropriate sign.

If any of the "I", "O", or "U" errors occurs with the error unmasked, the trap handler is called. Before calling the trap handler, mqerYIS sets the 8087 opcode register to 27C hex and leaves the inputs on the 8087 stack. The integer power is converted to TEMP_REAL and pushed onto the top of the 8087 stack. The base (the original (x)) becomes the second stack element.

PL/M-86 usage example:

```
mqerYIS: PROCEDURE (Y,I) REAL EXTERNAL;
  DECLARE Y REAL, I INTEGER;
END mqerYIS;

DECLARE INTEREST_RATE REAL;
DECLARE NUMBER_OF_PERIODS INTEGER;
DECLARE START_AMOUNT REAL;
DECLARE FINISH_AMOUNT REAL;

INTEREST_RATE = 0.015;   /* Test value */
NUMBER_OF_PERIODS = 12;   /* Test value */
START_AMOUNT = 1000.00;  /* Test value */

/* The following line calculates compound interest for the
   given NUMBER_OF_PERIODS, given the rate INTEREST_RATE
   for each period. INTEREST_RATE is presented as a
   fraction of 1; for example, the value 0.015 represents
   1.5 percent interest for each time period. */

FINISH_AMOUNT = START_AMOUNT* mqerYIS
                (1.+INTEREST_RATE, NUMBER_OF_PERIODS);

/* With the test inputs, FINISH_AMOUNT is now about
   1195.62. This is the balance of an unpaid loan of
   $1000.00 after one year, if the loan accumulates 1.5
   percent interest every month. */
```

ASM-86 usage example:

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs:
EXTRN mqerYIS: FAR
```

```
INTEREST_RATE       DQ  0.015
NUMBER_OF_PERIODS   DW  12
START_AMOUNT        DQ  1000.00
                      ; the above initializations are test
                      ; values
FINISH_AMOUNT       DQ  ?

; The following code implements the above PL/M example in
; assembly language, with LONG_REAL variables.

   FLD1                       ; 1 onto 8087 stack
   FADD INTEREST_RATE         ; (1 + I) is on stack
   PUSH NUMBER_OF_PERIODS     ; exponent parameter goes onto
                              ; 8086 stack
   CALL mqerYIS               ; (1 + I) ** N is on 8087 stack
   FMUL START_AMOUNT          ; scaled up by the amount of
                              ; money
   FSTP FINISH_AMOUNT         ; store result and pop the 8087
                              ; stack
; NOTE: Do not explicitly POP the NUMBER_OF_PERIODS from
; the 8086 stack --  mqerYIS does that for you.
```

Linkage of CEL87.LIB to Your Program Modules

The final action you must take to use CEL87.LIB in your programs is to include the file name CEL87.LIB into an appropriate LINK86 command.

CEL87 requires either the 8087 component or the 8087 emulator. If the component is present, you must also link in 8087.LIB. If the emulator is used, you must link in both E8087 and E8087.LIB.

If you are also using EH87.LIB, you must give the name EH87.LIB after CEL87.LIB. If you put EH87.LIB before CEL87.LIB, the program will link with no error messages, but it will halt after the first CEL87 function is called.

Following is the suggested order for object modules in your LINK86 statement.

Your object modules
DCON87.LIB if you are using it
CEL87.LIB
EH87.LIB if you are using it
8087.LIB if you are using the component, or
E8087, E8087.LIB if you are using the emulator

As an example, if you are linking your PL/M-86 modules MYMOD1.OBJ and MYMOD2.OBJ into a program using the 8087 emulator and the error handler, issue the command

```
-LINK86 :F1:MYMOD1.OBJ, :F1:MYMOD2.OBJ, &<cr>
>> :F0:CEL87.LIB, :F0:EH87.LIB, :F0:E8087, &<cr>
>> :F0:E8087.LIB TO :F1:MYPROG.LNK
```

If you have a single ASM-86-generated object module :F1:MYPROG.OBJ to be executed in a system with an 8087 chip, issue the command

```
-LINK86 :F1:MYPROG.OBJ, :F0:CEL87.LIB, &<cr>
>> :F0:8087.LIB TO :F1:MYPROG.LNK
```

CHAPTER 5
THE ERROR HANDLER MODULE

Overview

This chapter describes EH87.LIB, a library of five utility procedures which you can use to write trap handlers. Trap handlers are procedures which are called when an unmasked 8087 error occurs.

EH87.LIB also contains a set of "dummy" public symbols for all the functions of CEL87.LIB. These symbols save you code when you do not use all the functions of CEL87.LIB. Their presence, however, makes it absolutely necessary that you link EH87.LIB and CEL87.LIB in the correct order. The last section of this chapter tells you how to do so.

EH87.LIB also contains a set of alternate public names for its procedures, which are used by some Intel translators. They are listed in Appendix F.

The 8087 error reporting mechanism can be used not only to report error conditions, but also to let software implement modes and functions not directly supported by the chip. This chapter defines three such extensions to the 8087: normalizing mode, non-trapping NaN's, and non-ordered comparison. The utility procedures support these extra features.

DECODE is called near the beginning of the trap handler. It preserves the complete state of the 8087, and also identifies what function called the trap handler, with what arguments and/or results. DECODE eliminates much of the effort needed to determine what error caused the trap handler to be called.

NORMAL provides the "normalizing mode" capability for handling the "D" exception (described in the following section). By calling NORMAL in your trap handler, you eliminate the need to write code in your application program which tests for non-normal inputs.

SIEVE provides two capabilities for handling the "I" exception. It implements non-trapping NaN's and non-ordered comparisons (both described in the following sections). These two IEEE standard features reduce the incidence of multiple error reports for a single bad input.

ENCODE is called near the end of the trap handler. It restores the state of the 8087 saved by DECODE, and performs a choice of concluding actions, by either retrying the offending function or returning a specified result. ENCODE provides a common path for exiting the trap handler and resuming execution of the user program.

FILTER calls each of the above four procedures. If your error handler does nothing more than detect fatal errors and implement the features supported by SIEVE and NORMAL, then your interface to EH87.LIB can be accomplished with a single call to FILTER.

Normalizing Mode

Normalizing mode allows you to perform floating point operations without having to worry about whether the operands are in normal form. All denormal inputs will be normalized before the operation, without any user intervention.

The 8087 provides the "D" error, which warns you that an operand is not normal. You can implement normalizing mode in software by unmasking the "D" error, and providing a "D" trap handler. The handler should perform the needed normalization, and then retry the operation. We have provided NORMAL, which gives normalizations adequate for a majority of applications.

Non-Trapping NaN's

The large number of representations for NaN's gives you the chance to put diagnostic information into a NaN. The information can be passed along as the NaN undergoes multiple floating point operations. The information can not only tell where the NaN came from, but also control further error action.

EH87.LIB adopts the convention that the top bit of the fractional part of a NaN is a control bit, to be used in the following way: if the bit is 1, the NaN is to be considered a "non-trapping NaN", for which no further error need be explicitly reported. You can return a non-trapping NaN as the result of an invalid operation. Then when the NaN passes through more arithmetic, there will be no more errors reported. You thus avoid multiple error messages which really come from only one error.

The 8087 does not distinguish between trapping and non-trapping NaN's. All NaN's will generate an "I" error when they are used. However, you can provide an "I" trap handler which distinguishes between trapping and non-trapping NaN's. The procedure SIEVE does this for you.

Non-Ordered Comparisons

When you are testing two floating point numbers for equality, you may or may not want an error to be reported if those numbers are NaN's. The 8087 provides the FCOM and FTST instructions, which report an "I" error if they are given a NaN input.

To suppress error reporting for NaN's in FCOM and FTST, we recommend the following convention: if either FCOM or FTST is followed by a MOV AX,AX instruction (8BC0 hex), then the "I" trap handler should treat non-trapping NaN's as legal inputs. It should return the answer NON-ORDERED (C3 = C0 = 1), even if the two inputs are the same NaN, and act as if no "I" error had occurred. The procedure SIEVE follows this suggested convention.

Note that comparisons coded in PL/M-86 generate FCOM and FTST instructions which are not followed by a MOV AX,AX instruction. Therefore, according to the EH87.LIB convention, PL/M-86 comparisons of non-trapping NaN's are not considered legal. There is no way to cause PL/M-86 to insert a MOV AX,AX instruction after a comparison.

The ESTATE87 Data Structure

ESTATE87 is a 144-byte data structure created by DECODE, and used by the other EH87.LIB utility procedures. It contains most of the information your trap handler will need to provide customized error recovery: the state of the 8087, the identity of the offending operation, the values and formats of the operands, and possible already-calculated results.

You will typically receive an ESTATE87 structure from DECODE, and pass it back to ENCODE mostly unchanged. You do not need to become familiar with those parts of ESTATE87 which you do not change.

Following is a description of each of the fields of the structure ESTATE87. The offsets mentioned are decimal numbers which give the number of bytes from the beginning of ESTATE87 to the beginning of the field.

OPERATION is a WORD, offset 0, which contains an error code identifying which procedure or 8087 instruction caused the error. The error codes for 8087 instructions are given in Appendix D. The error codes for CEL87 functions are the last two digits of the codes given in Appendix E. The error code for all DCON87 functions is 0C8 hex.

ARGUMENT is a BYTE, offset 2, which identifies the types and locations of the arguments of the interrupted operation. See figure 5-1 for the layout of the bit fields within the ARGUMENT byte. The 3-bit fields ATYPE1 and ATYPE2 indicate the types of ARG1 and ARG2, according to the following codes:

0	no operand
1	top of 8087 stack: ST(0)
2	next element on 8087 stack: ST(1)
3	the element of the 8087 stack specified by REGISTER
4	a number in 8086 memory of a type given by FORMAT
5	a TEMP__REAL operand
6	a 64-bit integer operand
7	a binary-coded decimal operand

The PUSH ONCE bit is 1 if the result is pushed onto the 8087 stack (rather than replacing one or two of the input arguments).

For example, ARGUMENT would equal 21 hex if the instruction FPREM caused the error, since the arguments to FPREM are the top two elements on the 8087 stack, and the result replaces the inputs rather than causing the 8087 stack to be pushed.

ARG1(5) is a WORD array, offset 3, which gives the first argument of the operation, in the format specified by ARGUMENT.

ARG1__FULL is a boolean BYTE, offset 13, whose bottom bit is 1 if ARG1 is present. If the error handler is called after the offending operation is done, the argument may no longer exist.

ARG2(5) is a WORD array, offset 14, which gives the second argument of the operation, in the format specified by ARGUMENT.

ARG2__FULL is a boolean BYTE, offset 24, whose bottom bit is 1 if ARG2 is present. If there is only one argument, or if the error handler is called after the offending operation is done, then this argument may not exist.

RESULT is a BYTE, offset 25, which identifies the types and locations of the results of the interrupted operation. See figure 5-2. The 3-bit fields RTYPE1 and RTYPE2 use the same codes as the corresponding fields in the ARGUMENT byte. The POP ONCE bit is set if the operation causes the 8087 stack to pop exactly once; the POP TWICE bit is set if the operation causes the 8087 stack to pop twice (i.e., the operation is FCOMPP.)

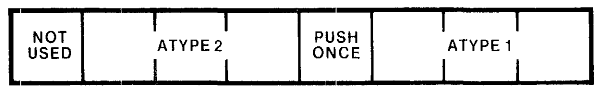

Figure 5-1. Bit Fields of the ARGUMENT Byte in ESTATE87 121725-2

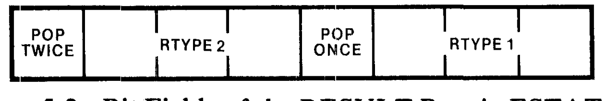

Figure 5-2. Bit Fields of the RESULT Byte in ESTATE87 121725-3

RES1(5) is a WORD array, offset 26, which gives the first result of the operation, in the format specified by RESULT.

RES1_FULL is a boolean BYTE, offset 36, whose bottom bit is 1 if RES1 is present. If the error handler is called before the offending operation is done, the result does not yet exist.

RES2(5) is a WORD array, offset 37, which gives the second result of the operation, in the format specified by RESULT.

RES2_FULL is a boolean BYTE, offset 47, whose bottom bit is 1 if RES2 is present. If there is only one result, or if the error handler is called before the offending operation is done, then this result does not exist.

FORMAT is a BYTE, offset 48, which specifies the memory data type when a field of ARGUMENT or RESULT has value 4. (There is never more than one such field.) The possible values of FORMAT are as follows:

0	for SHORT_REAL (32 bits)
1	for a 32-bit integer
2	for LONG_REAL (64 bits)
3	for a 16-bit integer

REGISTER is a BYTE, offset 49, which specifies the 8087 stack element number when a field of ARGUMENT or RESULT has the value 3. (No more than one such field can have this value.) The values of REGISTER range from 0 for the top stack element to 7 from the bottom-most stack element.

SAVE87(47) is a WORD array, offset 50, which contains the state of the 8087, as defined by the 8087 FSAVE instruction. Since DECODE is called after the 8087 exceptions have been cleared, the 8087 status word stored into SAVE87 differs from the status word as it existed when the trap handler was invoked. This former value must be maintained separately from ESTATE87; it appears as the parameter ERRORS87 in all the EH87.LIB routines.

How To Write an Exception Handler in ASM-86 Using EH87.LIB

By using EH87.LIB, you eliminate the difficult aspects of interfacing to the 8087 for your error recovery. However, there remains a strict protocol which must be followed. Following is a template of coding structure for an 8087 error handler written in ASM-86. If your customized error recovery is limited to errors which do not examine the ESTATE87 structure, you can follow the simpler template given under FILTER.

1. The handler procedure must be an INTERRUPT procedure. The 4-byte pointer is placed into the fixed slot for interrupt 16 (00040 hex).
2. If you expect to return to the code which caused the handler to be called, you must preserve all 8086 registers. The 8086 flags are automatically saved and restored because this is an interrupt routine.
3. The first floating point instruction of the handler should be an FNSTSW instruction, which stores the 8087 status word into a 16-bit memory location. This status word is the parameter ERRORS87 given to the EH87.LIB procedures.
4. To insure synchronization with the 8087 at this point, there must be an arbitrary access to the 8086 memory space. We recommend a PUSH AX instruction followed by a POP AX instruction.

5. There should follow an FNCLEX instruction, to clear the 8087 exceptions. The exceptions must be cleared for the following EH87.LIB calls to work properly.
6. The parameters to DECODE should be pushed onto the 8086 stack, and DECODE should be called.
7. If you intend to use NORMAL and SIEVE, their calls should come immediately after the call to DECODE. You should not have intervening code that alters ESTATE87. NORMAL and SIEVE require ESTATE87 to be defined by DECODE. If ESTATE87 does not have values which could be output by DECODE, the results of NORMAL and SIEVE are undefined.
8. If you have any customized code in your error handler, it should appear here.
9. If the handler is returning to the calling environment, the parameters to ENCODE should be pushed onto the 8086 stack, and ENCODE should be called.
10. The 8086 registers which were saved at the beginning of the exception handler should now be restored.
11. The exception handler should be exited using an IRET instruction. A simple RET instruction will not do because the 8086 pushes its flags onto its stack when the interrupt is executed.

You should also remember that if there is the possibility of a recursive call to the error handler (which can happen if ENCODE retries an instruction with exceptions unmasked), then all data storage used by your handler must be allocated on the 8086 stack. This includes the ERRORS87 word, and the entire ESTATE87 structure.

An Example of an Exception Handler Written in ASM-86

```
NAME HANDLER_87

EXTRN DECODE: FAR
EXTRN SIEVE: FAR
EXTRN ENCODE: FAR
EXTRN NORMAL: FAR

INTERRUPT_TABLE SEGMENT AT 0
  ORG 4 * 16
  DD TRAP_HANDLER
INTERRUPT_TABLE ENDS

CODE  SEGMENT PUBLIC
ASSUME CS:CODE

I_MASK         EQU    0001H              ; Position of "I"
                                         ; error bit and "I"
                                         ; mask

STACK_LAYOUT STRUC                       ; Pointed at by BP
                                         ; during TRAP_HANDLER

 OPERATION           DW ?                ; ESTATE87 template
                                         ; begins with this
                                         ; line

 ARGUMENT            DB ?
 ARG1                DW 5 DUP(?)
 ARG1_FULL           DB ?
```

```
ARG2                DW 5 DUP(?)
ARG2_FULL           DB ?
RESULT              DB ?
RES1                DW 5 DUP(?)
RES1_FULL           DB ?
RES2                DW 5 DUP(?)
RES2_FULL           DB ?
FORMAT              DB ?
REGISTER            DB ?
CONTROL_WORD        DW ?                ; Start of 94-byte
                                        ; FSAVE template
STATUS_WORD         DW ?                ; As it exists after
                                        ; clearing exceptions
TAG_WORD            DW ?
ERROR_POINTERS      DW 4 DUP(?)
STACK87             DT 8 DUP(?)         ; Last line of FSAVE
                                        ; and of ESTATE87
RETRY_CONTROL       DW ?                ; Retry 8087 Control
                                        ; setting for ENCODE
RETRY_FLAG          DB ?                ; Boolean parameter to
                                        ; ENCODE
ERRORS87            DW ?                ; 8087 Status Word
                                        ; before clearing

; You can place additional stack-allocated variables here,
; for your custom code. You refer to them as
; [BP].your_var_name. If the number of bytes of inserted
; space is odd, you should eliminate the following dummy
; variable.

 DUMMY_EVEN_ALIGN   DB ?                ; Not used; filler to
                                        ; keep SP even
 REGISTERS_86       DW 8 DUP(?)         ; Pushed by
                                        ; PUSH_REGISTERS
 FLAGS_86           DW ?                ; Pushed by 8086 when
                                        ; interrupt called
 RET_ADDRESS        DD ?                ; From TRAP_HANDLER

STACK_LAYOUT ENDS

; TRAP_HANDLER is a functioning 8087 exception handler
; written in ASM-86, using the procedures of EH87.LIB. It
; assumes that the only unmasked exceptions are "I" and
; "D", though we indicate below where you could insert
; code to handle other exceptions.

TRAP_HANDLER PROC FAR

  CALL PUSH_REGISTERS                 ; This interrupt will
                                      ; preserve 8086 registers
  SUB SP,OFFSET REGISTERS_86          ; Allocate room for
                                      ; STACK_LAYOUT
  MOV BP,SP                           ; Set up indexing into
                                      ; STACK_LAYOUT
  FNSTSW [BP].ERRORS87                ; Save the errors that
                                      ; caused the exception
  PUSH AX                             ; Arbitrary memory access
                                      ; to synchronize
  POP AX                              ;    with the 8087
  FNCLEX                              ; Clear the exceptions so
```

```
                                   ; handler can use 8087
   CALL PUSH_ESTATE_ERRORS         ; Push parameters to
                                   ; DECODE onto stack
   CALL DECODE                     ; ESTATE87 is now filled
                                   ; with valid data
   MOV AX,[BP].CONTROL_WORD        ; Existing control word is
                                   ; the default for
   MOV [BP].RETRY_CONTROL,AX       ;   any ENCODE retry
                                   ;   attempt
   MOV [BP].RETRY_FLAG,0FFH        ; Default setting: retry
                                   ; will take place
   CALL PUSH_ESTATE_ERRORS         ; Push parameters to
                                   ; NORMAL onto stack
   CALL NORMAL                     ; Handle "D" errors
   RCR AL,1                        ; Test boolean answer: was
                                   ; "D" the only error?
   JC ENCODE_EXIT                  ; If so then no other
                                   ; checking is necessary

   OR [BP].RETRY_CONTROL,I_MASK    ; Mask "I" exception
   CALL PUSH_ESTATE_ERRORS         ; Push parameters to SIEVE
   CALL SIEVE                      ; Check for legal non-
                                   ; Trapping NaN
   RCR AL,1                        ; Test boolean answer: was
                                   ; there such a NaN?
   JC ENCODE_EXIT                  ; If so then we can exit;
                                   ; "I" has been cleared

   TEST [BP].ERRORS87,I_MASK       ; Was "I" error bit set?
   JNZ ACTUAL_I_ERROR

   ; Here you could insert code to examine [BP].ERRORS87 to
   ; detect exceptions other than "I" or "D". If those
   ; other exceptions are detected, you can provide your
   ; own code to handle them.

   JMP ENCODE_EXIT

ACTUAL_I_ERROR:

   ; Here you may place your own customized code to deal
   ; with "I" exceptions other than legal non-trapping
   ; NaN's and denormalized inputs. The following lines set
   ; the "I" error bit, mask the "I" exception, and drop
   ; through to ENCODE_EXIT. This simulates the masked "I"
   ; exception, but with non-trapping NaN's implemented.
   ; The user program must unmask the "I" exception when it
   ; tests and clears the "I" error bit.

   OR [BP].STATUS_WORD,I_MASK      ; Set the "I" error bit
   OR [BP].CONTROL_WORD,I_MASK     ; Mask the "I" exception

ENCODE_EXIT:
   CALL PUSH_ESTATE_ERRORS         ; Push first two ENCODE
                                   ; parameters
   PUSH [BP].RETRY_CONTROL         ; Push third ENCODE
                                   ; parameter
   PUSH WORD PTR [BP].RETRY_FLAG   ; Push fourth ENCODE
                                   ; parameter
   CALL ENCODE                     ; Restore post-exception
```

```
                                     ; 8087 state
    ADD SP,OFFSET REGISTERS_86       ; Release the STACK_LAYOUT
                                     ; storage
    CALL POP_REGISTERS               ; Restore the eight 8086
                                     ; registers
    IRET                             ; Restore 8086 flags;
                                     ; long-return to caller

  TRAP_HANDLER ENDP

  ; PUSH_ESTATE_ERRORS causes the two parameters
  ; ESTATE87_PTR and ERRORS87 to be pushed onto the 8086
  ; stack, prior to a call to one of the EH87.LIB procedures
  ; which call for these parameters.

  PUSH_ESTATE_ERRORS PROC NEAR
    POP DX                         ; save the return address
    PUSH SS                        ; push the segment half of
                                   ; ESTATE87_PTR
    LEA AX,[BP].OPERATION          ; push the offset half of
                                   ; ESTATE87_PTR
    PUSH AX                        ;    --push complete
    PUSH [BP].ERRORS87             ; ERRORS87 is the bottom
                                   ; half of this byte
    JMP DX                         ; this is the RETURN from
                                   ; PUSH_ESTATE_ERRORS
  PUSH_ESTATE_ERRORS ENDP

  ; PUSH_REGISTERS causes the eight 8086 registers
  ; SI,DI,ES,BP,DX,CX,BX,AX to be pushed onto the 8086
  ; stack. The registers can be popped off with a call to
  ; POP_REGISTERS.

  PUSH_REGISTERS PROC NEAR
    PUSH DI
    PUSH ES
    PUSH BP
    PUSH DX
    PUSH CX
    PUSH BX
    PUSH AX
    MOV BP,SP              ; Get stack pointer into an index
                           ; register
    XCHG SI,[BP+14]        ; Exchange the return address with
                           ; SI which is being saved
    JMP SI                 ; This is the RETURN of the
                           ; procedure
  PUSH_REGISTERS ENDP

  ; POP_REGISTERS causes the eight registers which were
  ; pushed by PUSH_REGISTERS to be popped, restoring them to
  ; their original values.

  POP_REGISTERS PROC NEAR
    POP SI                 ; Hold this call's return address
                           ; temprarily in SI
    MOV BP,SP              ; Get the stack pointer into an
                           ; index register
    XCHG SI,[BP+14]        ; Restore SI, and position the
                           ; return address
    POP  AX
    POP  BX
    POP  CX
    POP  DX
    POP  BP
    POP  ES
    POP  DI
    RET                         ; Return address is in position due
                                ; to above XCHG
POP_REGISTERS ENDP

CODE  ENDS

END
```

How To Write an 8087 Exception Handler in PL/M-86 Using EH87.LIB

Following is a template of coding structure for any 8087 error handler written in PL/M-86. Note that many of the protocols required for an ASM-86 error handler are not needed in PL/M-86. They are provided automatically by PL/M-86 and its built-in functions.

If you customized error recovery is limited to errors which do not examine the ESTATE87 structure, you can follow the simpler template given under FILTER.

1. Your error handler procedure must have the attribute INTERRUPT 16.
2. You must declare the structure ESTATE87, and the WORD variable ERRORS87.
3. If you use any other variables in your error handler, they should also be declared within the procedure. All data must be declared within the procedure for the error handler to be reentrant.
4. You should make the assignment ERRORS87 = GET$REAL$ERROR.
5. You should call DECODE with the approriate parameters.
6. If you intend to use NORMAL and SIEVE, their calls should come immeditely after the call to DECODE. NORMAL and SIEVE require ESTATE87 to be defined by DECODE. If ESTATE87 does not have values which could be output by DECODE, the results of NORMAL and SIEVE are undefined.
7. If you have any customized code in your error handler, it should appear here.
8. If the handler is returning to the calling environment, you should call ENCODE with the appropriate parameters.

An Example of an 8087 Exception Handler Written in PL/M-86

```
HANDLER_MODULE: DO;

DECODE: PROCEDURE (ESTATE87_PTR,ERRORS87) EXTERNAL;
  DECLARE (ESTATE87_PTR POINTER, ERRORS87) WORD;
END;

ENCODE: PROCEDURE (ESTATE87_PTR,ERRORS87,CONTROL87,
                   RETRY_FLAG) EXTERNAL;
  DECLARE ESTATE87_PTR POINTER;
  DECLARE ERRORS87 WORD;
  DECLARE CONTROL87 WORD;
  DECLARE RETRY_FLAG BYTE;
END;

NORMAL: PROCEDURE (ESTATE87_PTR,ERRORS87) BYTE EXTERNAL;
  DECLARE (ESTATE87_PTR POINTER, ERRORS87) WORD;
END;

SIEVE: PROCEDURE (ESTATE87_PTR,ERRORS87) BYTE EXTERNAL;
  DECLARE (ESTATE87_PTR POINTER, ERRORS87) WORD;
END;

/* TRAP_HANDLER is a functioning 8087 exception handler
   written in PL/M-86, using the procedures of EH87.LIB.
   It assumes that the only unmasked exceptions are "I"
   and "D", though we indicate below where you could
   insert code to handle other exceptions. */

DECLARE I$ERROR$BIT LITERALLY '0001H';  /* To set the "I"
                                           error bit */
DECLARE TRUE LITERALLY '0FFH';

TRAP_HANDLER: PROCEDURE INTERRUPT 16;

  DECLARE ESTATE87 STRUCTURE (
     OPERATION WORD,
     ARGUMENT BYTE,
     ARG1(5) WORD, ARG1_FULL BYTE,
     ARG2(5) WORD, ARG2_FULL BYTE,
     RESULT BYTE,
     RES1(5) WORD, RES1_FULL BYTE,
     RES2(5) WORD, RES2_FULL BYTE,
     FORMAT BYTE,
     REGISTER BYTE,
     CONTROL_WORD WORD,
     STATUS_WORD WORD,
     TAG_WORD WORD,
     ERROR_POINTERS(4) WORD,
     STACK_87(40) WORD);
  DECLARE ERRORS87 BYTE;
  DECLARE CONTROL87 WORD;
  DECLARE RETRY_FLAG BYTE;

  ERRORS87 = GET$REAL$ERROR;
  CALL DECODE(@ESTATE87,ERRORS87);
  CONTROL87 = ESTATE87.CONTROL_WORD;
  RETRY_FLAG = TRUE;

  IF NOT NORMAL(@ESTATE87,ERRORS87)
   THEN DO;
    CONTROL87 = CONTROL87 OR I$ERROR$BIT;
    IF NOT SIEVE(@ESTATE87,ERRORS87)
     THEN DO;
      IF ERRORS87
       THEN DO;
        /* Here you may place your own customized code to
           deal with "I" exceptions other than non-
           trapping NaN's and denormalized inputs. The
           following lines set the "I" error bit, mask the
           "I" exception, and drop through to the RETRY
```

```
              exit. This simulates the masked "I" exception,
              but with non-trapping NaN's implemented. The
              user program must unmask the "I" exception when
              it tests and clears the "I" error bit. */
           ESTATE87.STATUS_WORD = ESTATE87.STATUS_WORD OR
                                  I$ERROR$BIT;
           ESTATE87.CONTROL_WORD = CONTROL87;
           END;
          ELSE DO;
           /* Here you could insert code to examine ERRORS87
              to detect exceptions other than "I" and "D". If
              those other exceptions are detected, you can
              provide your own code. */
           END;
         END;
       END;

     CALL ENCODE(@ESTATE87,ERRORS87,CONTROL87,RETRY_FLAG);

   END TRAP_HANDLER;

   END HANDLER_MODULE;
```

DECODE Decode trapped operation and save 8087 status

Input: Two parameters, totalling six bytes, must be pushed onto the 8086 stack before calling DECODE.

First, the four-byte pointer ESTATE87_PTR is pushed. As usual, the segment of ESTATE87_PTR is pushed first, followed by the offset. DECODE will send its output to the 144-byte memory buffer pointed to by ESTATE87_PTR.

Second, a two-byte quantity whose bottom byte is ERRORS87 is pushed onto the stack. ERRORS87 is the bottom byte of the 8087 status word as it existed when the trap handler was called, before the exceptions were cleared. The top byte of the two-byte quantity is ignored.

Function: DECODE provides, in a standardized format, all the information an exception handler might need to deal with the error condition. This includes the complete state of the 8087, the identity of the offending operation, and the existence, formats, and values of any arguments or results.

We have programmed into DECODE the error calling specifications of all 8087 instructions and all CEL87.LIB functions. Once DECODE has identified the interrupted operation, it uses these specifications to fill ESTATE87 with the correct arguments and results.

An exception to the programming specifications occurs for the CEL87 functions that return values in 8086 registers. These functions are mqerIA2, mqerIA4, mqerIC2, mqerIC4, mqerIE2, and mqerIE4. The results are specified by DECODE to go to the 8087 stack, not to 8086 registers.

DECODE identifies DCON87 errors by setting OPERATION to 0C8 hex, but no information about DCON87 arguments or results is available, because they reside in a location on the 8086 stack unknown to DECODE.

Note that for FCOMP and FCOMPP instructions with denormal arguments, the trap handler is called with the inputs already popped off the 8087 stack. DECODE recovers these arguments from the bottom of the 8087 stack, and pushes them back onto the stack top.

Output: All output is performed through the 144-byte memory buffer pointed at by the input parameter ESTATE87_PTR. The fields of the ESTATE87 structure are described in the section "The ESTATE87 Data Structure", at the beginning of this chapter.

The 8087 itself is left completely cleared to its initialized state. This is the input state expected by the other procedures of EH87.LIB.

PL/M-86 declaration and calling sequence:

```
DECODE: PROCEDURE (ESTATE87_PTR,ERRORS87) EXTERNAL;
  DECLARE ESTATE87_PTR POINTER, ERRORS87 WORD;
END;

/* Assume ESTATE87 is already declared as presented
   earlier in this chapter */
```

```
DECLARE ERRORS87 WORD;

/* Assume ERRORS87 has been set to the value of the bottom
   part of the 8087 status word before exceptions were
   cleared. */

CALL DECODE( @ESTATE87, ERRORS87 );
```

ASM-86 declaration and calling sequence:

```
; the following line must occur outside of all SEGEMENT-
; ENDS pairs

    EXTRN  DECODE: FAR

ESTATE_TEMPLATE STRUC
   DB  144 DUP(?)       ; The individual fields of ESTATE87
                        ; can be declared
                        ; as in STACK_LAYOUT at the
                        ; beginning of this chapter.
ESTATE_TEMPLATE ENDS

DATA SEGMENT
    ESTATE_87  ESTATE_TEMPLATE
    ERRORS87  DW  ?
DATA ENDS

; the following code assumes that DS contains the segment
; of ESTATE87

    ASSUME DS:DATA
    PUSH DS                      ; push the segment of
                                 ; ESTATE87 onto the stack
    MOV AX,OFFSET(ESTATE87)
    PUSH AX                      ; 4-byte pointer is now
                                 ; pushed onto stack
    MOV AL,ERRORS87              ; second parameter
    PUSH AX                      ;   -- pushed onto stack
    CALL DECODE                  ; ESTATE87 is now filled
                                 ; with information
```

ENCODE Restore 8087 status and operation environment

Input: ENCODE expects the 8087 to be completely cleared to its initialized state. Information about the 8087 is obtained from ESTATE87, not from the entry state of the 8087 itself.

Four parameters, totalling ten bytes, must be pushed onto the 8086 stack before calling ENCODE. The first parameter is four bytes, and each of the other three parameters is two bytes.

The first parameter, ESTATE87_PTR, points to the 144-byte ESTATE87 structure which was filled by a previous call to DECODE, and possibly modified by intervening code. ESTATE87_PTR is a four-byte pointer, with the segment pushed first and the offset pushed next.

The second parameter pushed onto the stack is ERRORS87. ERRORS87 is a WORD parameter whose bottom byte is the bottom byte of the 8087 status word as it existed when the trap handler was called, before the exceptions were cleared. The top byte of ERRORS87 is ignored.

The third parameter, CONTROL87, is a WORD parameter. It contains the value which the caller wants to place into the 8087 control word before retrying the function when RETRY_FLAG is true. If RETRY_FLAG is false, CONTROL87 is ignored.

The fourth and last parameter pushed onto the stack is RETRY_FLAG. RETRY_FLAG is a BYTE parameter, the bottom half of the WORD which is pushed onto the 8086 stack (the top half is ignored). RETRY_FLAG is a boolean control input which tells ENCODE what action to take after restoring the 8087 environment, as described below.

Function: ENCODE restores the 8087 to a state representing the completion of the operation which caused an error trap. This is accomplished by restoring the 8087 to the state stored in the ESTATE87 structure, after performing one of two actions, depending on RETRY_FLAG.

If the bottom bit of RETRY_FLAG is 0, the operation is assumed to have already been performed, and the results are assumed to be in ESTATE87. The results are copied to their destination. If insufficient results are available in ESTATE87, the value INDEFINITE may be used.

If the bottom bit of RETRY_FLAG is 1, the operation is identified from ESTATE87, and reperformed using operands as specified in ESTATE87. The parameter CONTROL87 is used as the 8087 control word for the retry; you can thus select which exceptions will be unmasked. After the retry is complete, the 8087 control word as it exists in ESTATE87 is restored.

If the operation being retried is a load or store that merely moves data without transforming it, ENCODE will perform the operation with exceptions masked, using the ARGUMENT and RESULT information of ESTATE87. This allows you to call ENCODE with true RETRY_FLAG in all cases when NORMAL returns TRUE, without fear that the retry will repeat the "D" error.

Note that ENCODE cannot be called with RETRY_FLAG true if the error came from DCON87, since ESTATE87 does not contain enough information about DCON87. For the same reason, ENCODE cannot retry the CEL87 functions mqerIC2, mqerIC4, mqerIE2, or mqerIE4 after a "P" precision error.

ENCODE can perform the retry for "I" errors coming from the above four CEL87 functions, as well as mqerIA2 and mqerIA4, but the results will go to the 8087 stack and not to the correct 8086 registers. Your code must identify these functions and pop the answer from the 8087 to the correct destination in order to resume execution of the trapped program.

If your CONTROL87 parameter contains unmasked exceptions, it is possible for an exception handler to be called during the retry, and for ENCODE to be invoked recursively. If you unmask the exception that caused the error, you should have modified the original arguments or results to avoid an infinite recursion.

Output: ENCODE returns with the 8087 restored to the appropriate post-exception state, as indicated by ESTATE87 and the effects of either the retry or of the movement of results to their destination. Also, the input parameters are popped from the 8086 stack.

PL/M-86 declaration and calling sequence:

```
ENCODE: PROCEDURE (ESTATE87_PTR,ERRORS87,CONTROL87,
    RETRY_FLAG) EXTERNAL;
  DECLARE ESTATE87_PTR POINTER;
  DECLARE ERRORS87 WORD;
  DECLARE CONTROL87 WORD;
  DECLARE RETRY_FLAG BYTE;
END;

/* Assume ESTATE87 is already declared as presented
   earlier in this chapter. */

DECLARE ERRORS87 WORD;
DECLARE CONTROL87 WORD;
DECLARE RETRY_FLAG BYTE;

/* Assume that all the parameters have been filled with
   values. */

CALL ENCODE( @ESTATE87, ERRORS87, CONTROL87, RETRY_FLAG );

/* The 8087 has now been returned to an appropriate post-
   exception state. */
```

ASM-86 declaration and calling sequence:

```
; the following line must occur outside of all SEGEMENT-
; ENDS pairs

    EXTRN  ENCODE: FAR

DATA SEGMENT
    ESTATE_87  ESTATE_TEMPLATE
    ERRORS87  DW  ?
DATA ENDS

; The following code assumes that DS contains the segment
; of ESTATE87. It also assumes that the parameters to
; ENCODE have been set to appropriate values.
```

```
        ASSUME DS:DATA
        PUSH DS                       ; push the segment of
                                      ; ESTATE87 onto the stack
        MOV AX,OFFSET(ESTATE87)
        PUSH AX                       ; 4-byte pointer is now
                                      ; pushed onto stack
        MOV AX,ERRORS87
        PUSH AX                       ; second parameter is
                                      ; pushed onto stack
        MOV AX,CONTROL87              ; load third parameter
        PUSH AX                       ; push onto stack
        MOV AL, RETRY_FLAG            ; last parameter
        PUSH AX                       ;  -- pushed onto stack
        CALL ENCODE

    ; the 8087 is now restored to an appropriate post-
    ; exception state
```

FILTER Filter denormals and non-trapping NaN's from user error handling

Input: The two-byte quantity whose bottom byte is ERRORS87 is pushed onto the 8086 stack before calling FILTER. ERRORS87 is the bottom byte of the 8087 status word as it existed when the trap handler was called, before the exceptions were cleared. The top byte of the two-byte quantity is ignored.

Function: FILTER provides a single interface to EH87.LIB which calls all of the other four functions DECODE, SIEVE, NORMAL, and ENCODE. If ERRORS87 indicates that a "D" error caused the trap handler to be called, the denormal argument is found and normalized. If an "I" error caused the trap, FILTER indicates whether the exception was caused by a legal non-trapping NaN.

FILTER is a boolean function. It returns TRUE if the trap was indeed caused by one of the above-mentioned "I" or "D" cases. TRUE means that you may return immediately from the trap handler. FILTER returns FALSE if it has not handled the error. If FILTER returns FALSE, your trap handler should execute appropriate customized error recovery.

Output: The boolean value of FILTER is output as the bottom bit of the AL register. The input word ERRORS87 is popped from the 8086 stack upon return.

If FILTER returns FALSE, the 8087 is returned to the same state that it was prior to the call to FILTER.

If FILTER returns TRUE, the 8087 is changed to reflect a retry of the instruction with normal operands (in case of "D" error) or "I" masked (in case of "I" error).

PL/M-86 programming example:

```
FILTER_MODULE: DO;

FILTER: PROCEDURE (ERRORS87) BYTE EXTERNAL;
  DECLARE ERRORS87 WORD;
END;

/* SIMPLE_TRAP_HANDLER is an floating point exception
   handler which provides a bare minimum interface to
   EH87.LIB.  If the error which caused this interrupt has
   been handled by FILTER, then the interrupt returns to
   the user program immediately.  Otherwise, your inserted
   custom code is executed. */

SIMPLE_TRAP_HANDLER: PROCEDURE INTERRUPT 16;

  DECLARE ERRORS87 WORD;

  IF FILTER (ERRORS87 := GET$REAL$ERROR) THEN RETURN;

  /* Here you could place code to handle exceptions other
     than the denormals and non-trapping NaN's handled by
     FILTER. */

END SIMPLE_TRAP_HANDLER;

END FILTER_MODULE;
```

ASM-86 programming example:

```
NAME FILTER_MODULE

EXTRN FILTER: FAR

INTERRUPT_TABLE SEGMENT AT 0
  ORG 4 * 16
  DD SIMPLE_TRAP_HANDLER
INTERRUPT_TABLE ENDS

CODE  SEGMENT PUBLIC
ASSUME CS:CODE

STACK_LAYOUT STRUC              ; Pointed at by BP during
                                ; SIMPLE_TRAP_HANDLER

; You can place additional stack-allocated variables here,
; for your custom code.  You refer to them as
; BP.your_var_name.

ERRORS87      DW ?            ; 8087 status word before
                              ; clearing
REGISTERS_86  DW 8 DUP(?)     ; Pushed by PUSH_REGISTERS
FLAGS_86      DW ?            ; Pushed by 8086 when interrupt
                              ; called
RET_ADDRESS   DD ?            ; From SIMPLE_TRAP_HANDLER

STACK_LAYOUT ENDS

; SIMPLE_TRAP_HANDLER is an floating point exception
; handler which provides a bare minimum interface to
; EH87.LIB. If the error which caused this interrupt has
; been handled by FILTER, then the interrupt returns to
; the user program immediately. Otherwise, your inserted
; custom code is executed.

SIMPLE_TRAP_HANDLER PROC FAR

 CALL PUSH_REGISTERS            ; Save the 8086 registers
 SUB SP,OFFSET REGISTERS_86     ; Allocate room for stack
                                ; layout
 MOV BP,SP                      ; Set up indexing into
                                ; STACK_LAYOUT
 FNSTSW [BP].ERRORS87           ; Save the errors that caused
                                ; the exception
 MOV AX,[BP].ERRORS87           ; Fetch the status word ...
 PUSH AX                        ;   ... and store it as FILTER
                                ;       parameter
 FNCLEX                         ; Clear the exceptions so
                                ; handler can use 8087
 CALL FILTER                    ; Check for denormal or non-
                                ; trapping NaN
 RCR AL,1                       ; Was error handled?
 JC TRAP_EXIT                   ; If so then do nothing more

; Here you could place code to handle exceptions other
; than the denormals and non-trapping NaN's handled by
; FILTER.
```

```
TRAP_EXIT:
  CALL POP_REGISTERS          ; Restore the 8086 registers
  IRET

SIMPLE_TRAP_HANDLER ENDP

; PUSH_REGISTERS causes the eight 8086 registers
; SI,DI,ES,BP,DX,CX,BX,AX to be pushed onto the 8086
; stack. The registers can be popped off with a call to
; POP_REGISTERS.

PUSH_REGISTERS PROC NEAR
 PUSH DI
 PUSH ES
 PUSH BP
 PUSH DX
 PUSH CX
 PUSH BX
 PUSH AX
 MOV BP,SP          ; Get stack pointer into an index
                    ; register
 XCHG SI,[BP+14]    ; Exchange the return address with SI
                    ; which is being saved
 JMP SI             ; This is the RETURN of the procedure
PUSH_REGISTERS ENDP

; POP_REGISTERS causes the eight registers which were
; pushed by PUSH_REGISTERS to be popped, restoring them to
; their original values.

POP_REGISTERS PROC NEAR
 POP SI             ; Hold this call's return address
                    ; temporarily in SI
 MOV BP,SP          ; Get the stack pointer into an index
                    ; register
 XCHG SI,[BP+14]    ; Restore SI, and position the return
                    ; address
 POP AX
 POP BX
 POP CX
 POP DX
 POP BP
 POP ES
 POP DI
 RET        ; Return address is in position due to above
            ; XCHG
POP_REGISTERS ENDP

CODE  ENDS

END
```

NORMAL Detect and normalize "D" error non-normal arguments

Input: NORMAL expects the 8087 to be completely cleared to its initialized state. Information about the 8087 is obtained from ESTATE87, not from the entry state of the 8087 itself.

Two parameters, totalling six bytes, must be pushed onto the 8086 stack before calling NORMAL.

First, the four-byte pointer ESTATE87_PTR is pushed. The segment of ESTATE87_PTR is pushed first, followed by the offset. ESTATE87_PTR points to the 144-byte ESTATE87 structure which was filled by a previous call to DECODE.

Second, a two-byte quantity whose bottom byte is ERRORS87 is pushed onto the stack. ERRORS87 is the bottom byte of the 8087 status word as it existed when the trap handler was called, before the exceptions were cleared. The top byte of the two-byte quantity is ignored.

Function: NORMAL first checks ERRORS87 to see if the "D" error bit is the only unmasked error bit which is set. If this is not the case, NORMAL immediately returns FALSE, indicating that no normalization retry is to take place.

If "D" is set, and no other unmasked error bits are set, NORMAL returns TRUE. If the operation that caused the "D" error was a load operation, the non-normal arguments are left unchanged. If the operation was not a load operation, NORMAL modifies the denormal arguments. Denormal SHORT_REAL and LONG_REAL arguments are normalized; denormal TEMP_REAL arguments are converted to zero. Only the copies of arguments in ESTATE87's ARG1 and ARG2 fields are modified.

Whenever NORMAL returns TRUE, you should call ENCODE with RETRY_FLAG set to TRUE. You may leave "D" unmasked, using the same control word in effect when the trap handler was called. If the operation was a load instruction, ENCODE will perform the load with "D" masked, and not cause a repeat of the "D" error.

Note that NORMAL always returns FALSE if the operation that caused the trap is DCON87 or a CEL87 function. Only individual 8087 instructions can cause a "D" error.

Output: NORMAL returns normalized arguments by modifying ESTATE87. In addition, it returns a boolean value in the AL register indicating further action to be taken. If the bottom bit of AL is 1, then a normalization has taken place, and NORMAL is requesting the caller to retry the offending operation by calling ENCODE with a true RETRY_FLAG. If the bottom bit of AL is 0, no such retry is requested.

NORMAL returns the 8087 itself to the same cleared state it had upon entry.

PL/M-86 declaration and calling sequence:

```
NORMAL: PROCEDURE (ESTATE87_PTR,ERRORS87) BYTE EXTERNAL;
  DECLARE ESTATE87_PTR POINTER, ERRORS87 WORD;
END;

/* Assume ESTATE87 is already declared as presented
   earlier in this chapter */

DECLARE ERRORS87 WORD;
DECLARE NORM_RETRY BYTE;

NORM_RETRY = NORMAL( @ESTATE87, ERRORS87 );

/* Now NORM_RETRY is true if a retry of the operation
   should be made */
```

ASM-86 declaration and calling sequence:

```
; the following line must occur outside of all SEGEMENT-
; ENDS pairs

    EXTRN  NORMAL: FAR

DATA SEGMENT
    ESTATE_87        ESTATE_TEMPLATE
    ERRORS87         DW   ?
    NORM_RETRY       DB   ?
DATA ENDS

; the following code assumes that DS contains the segment
; of ESTATE87

 ASSUME DS:DATA
 PUSH DS                ; push the segment of ESTATE87 onto
                        ; the stack
 MOV AX,OFFSET(ESTATE87)
 PUSH AX                ; 4-byte pointer is now pushed onto
                        ; stack
 MOV AL,ERRORS87        ; second parameter
 PUSH AX                ;  -- pushed onto stack
 CALL NORMAL            ; AL now tells whether to retry
 MOV NORM_RETRY,AL      ; boolean answer stored in NORM_RETRY
```

SIEVE Detect non-trapping NaN's which should be ignored

Input: SIEVE expects the 8087 to be completely cleared to its initialized state. Information about the 8087 is obtained from ESTATE87, not from the entry state of the 8087 itself.

Two parameters, totalling six bytes, must be pushed onto the 8086 stack before calling SIEVE.

First, the four-byte pointer ESTATE87_PTR is pushed. The segment of ESTATE87_PTR is pushed first, followed by the offset. ESTATE87_PTR points to the 144-byte ESTATE87 structure which was filled by a previous call to DECODE.

Second, a two-byte quantity whose bottom byte is ERRORS87 is pushed onto the stack. ERRORS87 is the bottom byte of the 8087 status word as it existed when the trap handler was called, before the exceptions were cleared. The top byte of the two-byte quantity is ignored.

Function: SIEVE signals those cases in which the "I" exception should not have been given because the argument was a legal non-trapping NaN. This detection applies to all arithmetic operations which check for NaN inputs.

SIEVE follows the conventions described in the sections titled "Non-Trapping NaN's" and "Non-Ordered Comparisons" towards the beginning of this chapter. If it is determined that this "I" error should not have taken place because the NaN arguments are non-trapping and the operation is not an ordered comparison, then SIEVE returns TRUE. In this case, the caller should retry the offending operation by calling ENCODE with a true RETRY_FLAG, and CONTROL87 modified so that "I" is masked. ENCODE will restore the original control word, with "I" unmasked, after the retry is executed.

If it is determined that this "I" error should still be flagged (for example, a stack error has occurred), SIEVE returns FALSE. Some appropriate "I" error recovery action should be performed.

Note that if the operation which caused the trap is DCON87 or a CEL87 function, SIEVE will always return FALSE. All legal non-trapping NaN situations arise from individual 8087 instructions.

Output: The boolean answer is returned as the bottom bit of the AL register. If this bit is 1, the "I" error should not have taken place, according to the non-trapping NaN conventions described earlier in this chapter.

SIEVE leaves the 8087 in the same cleared state it had upon entry. The structure ESTATE87 is also unchanged by SIEVE.

PL/M-86 declaration and calling sequence:

```
SIEVE: PROCEDURE (ESTATE87_PTR,ERRORS87) BYTE EXTERNAL;
  DECLARE ESTATE87_PTR POINTER, ERRORS87 WORD;
END;

/* Assume ESTATE87 is already declared as presented
   earlier in this chapter */
```

```
DECLARE ERRORS87 WORD;
DECLARE LEGAL_NAN BYTE;

LEGAL_NAN = SIEVE( @ESTATE87, ERRORS87 );

/* Now LEGAL_NAN is true if there should not have been an
   ''I'' error */
```

ASM-86 declaration and calling sequence:

```
; the following line must occur outside of all SEGEMENT-
; ENDS pairs

    EXTRN  SIEVE: FAR

DATA SEGMENT
    ESTATE_87  ESTATE_TEMPLATE
    ERRORS87   DW  ?
DATA ENDS

; the following code assumes that DS contains the segment
; of ESTATE87

    ASSUME DS:DATA
    PUSH DS              ; push the segment of ESTATE87 onto
                         ; the stack
    MOV AX,OFFSET(ESTATE87)
    PUSH AX              ; 4-byte pointer is now pushed onto
                         ; stack
    MOV AL,ERRORS87      ; second parameter
    PUSH AX              ;   -- pushed onto stack
    CALL SIEVE           ; AL now contains boolean answer
    MOV LEGAL_NAN,AL     ; LEGAL_NAN true if there is no
                         ; ''I'' error
```

Linkage of EH87.LIB to Your Program Modules

This section tells how to link EH87.LIB into your program using a LINK86 command.

If you are linking both CEL87.LIB and EH87.LIB, it is absolutely necessary that EH87.LIB appear after CEL87.LIB in the LINK86 command for this reason: EH87.LIB contains references to all the functions of CEL87.LIB. However, if there are any unused CEL87 functions, they do not need to be linked for EH87.LIB to work correctly. We have therefore provided public symbols for CEL87.LIB in EH87.LIB, all of which point to an 8086 HLT instruction. CEL87.LIB must appear first in the LINK86 invocation to pick up the actual references to CEL87 functions; then EH87.LIB will supply HLT references (which do not add to the code size) for all unused CEL7 symbols.

If you mistakenly put CEL87.LIB after EH87.LIB in your link invocation, the link will be performed without any visible problems. But all your calls to CEL87.LIB will go to a HLT instruction. When you attempt to execute your program, the NDP will halt the first time a CEL87 function is called.

EH87.LIB requires either the 8087 component or the 8087 emulator. If the component is present, you must link in 8087.LIB. If the emulator is used, you must instead link in E8087 and E8087.LIB.

Following is the recommended ordering of modules in the LINK86 statement:

Your object modules
DCON87.LIB if your program uses it
CEL87.LIB if your program uses it
EH87.LIB
8087.LIB if you are using the component, or
E8087, E8087.LIB if you are using the emulator

Examples:

If your modules are MYMOD1.OBJ and MYMOD2.OBJ and you are using the 8087 emulator, issue the command

```
-LINK86 :F1:MYMOD1.OBJ, :F1:MYMOD2.OBJ, &<cr>
>> :F0:DCON87.LIB, :F0:CEL87.LIB, :F0:EH87.LIB, &<cr>
>> :F0:E8087, :F0:E8087.LIB TO :F1:MYPROG.LNK
```

If you have a single module MYPROG.OBJ, are using the 8087 component, and do not use any libraries other than EH87.LIB, issue the command

```
-LINK86 :F1:MYPROG.OBJ, :F0:EH87.LIB, &<cr>
>> :F0:8087.LIB TO :F1:MYPROG.LNK
```

CHAPTER 6
IMPLEMENTING THE IEEE STANDARD

The NDP, together with the 8087 Support Library, provides an implementation of "A Proposed Standard for Binary Floating Point Arithmetic", Draft 8.0 of IEEE Task P754, *Computer*, March 1981, pp. 51-62.

This chapter describes the relationship between the NDP and the IEEE Standard. It gives the choices we have made in the places where the Standard has options. It pinpoints two areas in which we do not conform to the Standard. If your application requires it, you can provide software to meet the Standard in these areas. We indicate how to write this software.

This chapter contains many terms with precise technical meanings, as specified by the Standard. We follow the Standard's convention of emphasizing the precision of our meaning by capitalizing those terms. The glossary in Appendix A provides the definitions for all capitalized phrases in this chapter.

Options We Have Chosen

Our SHORT_REAL and LONG_REAL formats conform perfectly to the Standard's Single and Double Floating Point Numbers, respectively. Our TEMP_REAL format is the same as the Standard's Double Extended format. The Standard allows a choice of Bias in representing the exponent; we have chosen the Bias 16383 decimal.

For the Double Extended format, the Standard contains an option for the meaning of the minimum exponent combined with a non-zero significand. The Bias for this special case can be either 16383, as in all the other cases, or 16382, making the smallest exponent equivalent to the second-smallest exponent. We have chosen the Bias 16382 for this case. This allows us to distinguish between Denormal numbers (integer part is zero, fraction is non-zero, Biased Exponent is 0) and Unnormal numbers of the same value (same as the denormal except the Biased Exponent is 1.)

The Standard endorses the support of only one extended format, and that is all we provide.

The Standard allows us to specify which NaN's are Trapping and which are non-Trapping. EH87.LIB, which provides a software implementation of non-Trapping NaN's, defines the distinction. If the most significant bit of the fractional part of a NaN is 1, the NaN is non-Trapping. If it is 0, the NaN is Trapping.

When a masked "I" error involves two NaN inputs, the Standard lets us set the rule for which NaN is output. We choose the NaN whose absolute value is greatest.

Areas Which Need the Support Library to Meet the Standard

There are five features of the Standard which are not implemented directly on the 8087.

1. The Standard recommends (but does not absolutely require) that a Normalizing Mode be provided, in which any non-normal operands to functions are automatically normalized before the function is performed. The NPX instead has a "D" exception, not mentioned in the Standard, which gives the exception

handler the opportunity to perform the normalization specified by the Standard. The "D" exception handler provided by EH87.LIB implements the Standard's Normalizing Mode completely for Single and Double precision arguments. Normalizing mode for Double Extended operands is implemented with one non-Standard feature, mentioned in the next section.

2. The Standard specifies that the "=" comparison test yield an answer of FALSE, with no "I" error, if the relation between the input operands is "unordered". The 8087 FCOM and FTST instructions issue an "I" error for this case. The error handler EH87.LIB filters out the "I" error, using the following convention: whenever an FCOM or FTST instruction is followed by a MOV AX,AX instruction (8BC0 hex), and neither argument is a trapping NaN, the error handler will assume that a Standard "=" comparison was intended, and return the correct answer with "I" erased. Note that "I" must be unmasked for this action to occur.
3. The Standard requires that two kinds of NaN's be provided, Trapping and non-Trapping. Non-Trapping NaN's will not cause further "I" errors when they occur as operands to calculations. The NPX directly supports only Trapping NaN's. The error handler module EH87.LIB implements non-Trapping NaN's by returning the correct answer with "I" erased. Note that "I" must be unmasked for this action to occur.
4. The Standard requires that all functions which convert real numbers to integer formats automatically normalize the inputs if necessary. The 8087 FIST instruction does not do so. CEL87.LIB's integer conversion functions fully meet the Standard in this aspect.
5. The Standard specifies the remainder function which is provided by mqerRMD in CEL87.LIB. The 8087 FPREM instruction returns answers in a different range.

Further Software Required to Fully Meet the Standard

There are two cases in which you will need to provide further software to meet the Standard. We have not provided this software because we feel that a vast majority of applications will never encounter these cases.

1. Non-Trapping NaN's are not implemented when "I" is masked. Likewise, the Standard's "=" function is not implemented when "I" is masked. You can simulate the Standard's concept of a masked "I" exception by unmasking the 8087 "I" bit, and providing an "I" trap handler which supports non-Trapping NaN's and the "=" function, but otherwise acts just as if "I" were masked. Chapter 5 contains examples in both ASM-86 and PL/M-86 for doing this.
2. Denormal operands in the TEMP_REAL format are converted to 0 by the EH87.LIB Normalizing Mode, giving sharp Underflow to 0. The Standard specifies that the operation be performed on the real numbers represented by the denormals, giving gradual Underflow. To correctly perform such arithmetic, you will have to normalize the operands into a format identical to TEMP_REAL except for two extra exponent bits, then perform the operation on those numbers. Thus your software must manage the 17-bit exponent explicitly.

We felt that it would be disadvantageous to most users to increase the size of the Normalizing routine by the amount necessary to provide this expanded arithmetic. Remember that the TEMP_REAL exponent field is so much larger than the LONG_REAL exponent field that it is extremely unlikely that you will encounter TEMP_REAL Underflow. Remember further that the Standard does not require

Normalizing mode. If meeting the Standard is a more important criterion to you than the choice between Normalizing and warning modes, you can select warning mode (''D'' masked), which fully meets the Standard.

If you do wish to implement the TEMP_REAL arithmetic with extra exponent bits, we list below some useful pointers about when the ''D'' error occurs:

a. TEMP_REAL numbers are considered Denormal by the NPX whenever the Biased Exponent is 0 (minimum exponent). This is true even if the explicit integer bit of the significand is 1. Such numbers can occur as the result of Underflow.

b. The 8087 FLD instruction can cause a ''D'' error if a number is being loaded from memory. It cannot cause a ''D'' error if it is coming from elsewhere in the 8087 stack.

c. The 8087 FCOM and FTST instructions will cause a ''D'' error for unnormal operands as well as denormal operands.

d. In cases where both ''D'' and ''I'' errors occur, you will want to know which is signalled first. When a comparison instruction is issued between a non-existent stack element and a denormal number in 8086 memory, the ''D'' and ''I'' errors are issued simultaneously. In all other situations, a stack ''I'' error takes precedence over a ''D'' error, and a ''D'' error takes precedence over a non-stack ''I'' error.

APPENDIX A GLOSSARY OF 8087 AND FLOATING-POINT TERMINOLOGY

We continue in this appendix the convention of Chapter 6, capitalizing terms which have precise technical meanings. Such terms appear as entries in this glossary. Thus you may interpret any non-standard capitalization as a cross-reference.

Affine Mode: a state of the 8087, selected in the 8087 Control Word, in which infinities are treated as having a sign. Thus the values +INFINITY and −INFINITY are considered different; and they can be compared with finite numbers and with each other.

Base: (1) a term used in logarithms and exponentials. In both contexts, it is a number which is being raised to a power. The two equations (y = log base b of x) and ($b^y = x$) are the same.

Base: (2) a number which defines the representation being used for a string of digits. Base 2 is the binary representation; Base 10 is the decimal representation; Base 16 is the hexadecimal representation. In each case, the Base is the factor of increased significance for each succeeding digit (working up from the bottom).

Bias: the difference between the unsigned Integer which appears in the Exponent field of a Floating Point Number and the true Exponent that it represents. To obtain the true Exponent, you must subtract the Bias from the given Exponent. For example, the Short Real format has a Bias of 127 whenever the given Exponent is non-zero. If the 8-bit Exponent field contains 10000011, which is 131, the true Exponent is 131−127, or +4.

Biased Exponent: the Exponent as it appears in a Floating Point Number, interpreted as an unsigned, positive number. In the above example, 131 is the Biased Exponent.

Binary Coded Decimal: a method of storing numbers which retains a base 10 representation. Each decimal digit occupies 4 full bits (one hexadecimal digit). The hex values A through F (1010 through 1111) are not used. The 8087 supports a "Packed Decimal" format which consists of 9 bytes of Binary Coded Decimal (18 decimal digits), and one sign byte.

Binary Point: an entity just like a decimal point, except that it exists in binary numbers. Each binary digit to the right of the Binary Point is multiplied by an increasing negative power of two.

C3—C0: the four "condition code" bits of the 8087 Status Word. These bits are set to certain values by the compare, test, examine, and remainder functions of the 8087.

Characteristic: a term used for some non-Intel computers, meaning the Exponent field of a Floating Point Number.

Chop: to set the fractional part of a real number to zero, yielding the nearest integer in the direction of zero.

Control Word: a 16-bit 8087 register that the user can set, to determine the modes of computation the 8087 will use, and the error interrupts that will be enabled.

Denormal: a special form of Floating Point Number, produced when an Underflow occurs. On the 8087, a Denormal is defined as a number with a Biased Exponent which is zero. By providing a Significand with leading zeros, the range of possible negative Exponents can be extended by the number of bits in the Significand. Each leading zero is a bit of lost accuracy, so the extended Exponent range is obtained by reducing significance.

Double Extended: the Standard's term for the 8087 Temporary Real format, with more Exponent and Significand bits than the Double (Long Real) format, and an explicit Integer bit in the Significand.

Double Floating Point Number: the Standard's term for the 8087's 64-bit Long Real format.

Environment: the 14 bytes of 8087 registers affected by the FSTENV and FLDENV instructions. It encompasses the entire state of the 8087, except for the 8 Temporary Real numbers of the 8087 stack. Included are the Control Word, Status Word, Tag Word, and the instruction, opcode, and operand information provided by interrupts.

Exception: any of the six error conditions (I, D, O, U, Z, P) signalled by the 8087.

Exponent: (1) any power which is raised by an exponential function. For example, the operand to the function mqerEXP is an Exponent. The Integer operand to mqerYI2 is an Exponent.

Exponent: (2) the field of a Floating Point Number which indicates the magnitude of the number. This would fall under the above more general definition (1), except that a Bias sometimes needs to be subtracted to obtain the correct power.

Floating Point Number: a sequence of data bytes which, when interpreted in a standardized way, represents a Real number. Floating Point Numbers are more versatile than Integer representations in two ways. First, they include fractions. Second, their Exponent parts allow a much wider range of magnitude than possible with fixed-length Integer representations.

Gradual Underflow: a method of handling the Underflow error condition which minimizes the loss of accuracy in the result. If there is a Denormal number which represents the correct result, that Denormal is returned. Thus digits are lost only to the extent of denormalization. Most computers return zero when Underflow occurs, losing all significant digits.

Implicit Integer Bit: a part of the Significand in the Short Real and Long Real formats which is not explicitly given. In these formats, the entire given Significand is considered to be to the right of the Binary Point. A single Implicit Integer Bit to the left of the Binary Point is always 1, except in one case. When the Exponent is the minimum (Biased Exponent is 0), the Implicit Integer Bit is 0.

Indefinite: a special value that is returned by functions when the inputs are such that no other sensible answer is possible. For each Floating Point format there exists one non-Trapping NaN which is designated as the Indefinite value. For binary Integer formats, the negative number furthest from zero is often considered the Indefinite value. For the 8087 Packed Decimal format, the Indefinite value contains all 1's in the sign byte and the uppermost digits byte.

Infinity: a value which has greater magnitude than any Integer, or any Real number. The existence of Infinity is subject to heated philosophical debate. However, it is often useful to consider Infinity as another number, subject to special rules of arithmetic. All three Intel Floating Point formats provide representations for +INFINITY and −INFINITY. They support two ways of dealing with Infinity: Projective (unsigned) and Affine (signed).

Integer: a number (positive, negative, or zero) which is finite and has no fractional part. "Integer" can also mean the computer representation for such a number: a sequence of data bytes, interpreted in a standard way. It is perfectly reasonable for Integers to be represented in a Floating Point format; this is what the 8087 does whenever an Integer is pushed onto the 8087 stack.

Invalid Operation: the error condition for the 8087 which covers all cases not covered by other errors. Included are 8087 stack overflow and underflow, NaN inputs, illegal infinite inputs, out-of-range inputs, and illegal unnormal inputs.

Long Integer: an Integer format supported by the 8087 which consists of a 64-bit Two's Complement quantity.

Long Real: a Floating Point Format supported by the 8087, which consists of a sign, an 11-bit Biased Exponent, an Implicit Integer Bit, and a 52-bit Significand; a total of 64 explicit bits.

Mantissa: a term used for some non-Intel computers, meaning the Significand of a Floating Point Number.

Masked: a term which applies to each of the six 8087 Exceptions I,D,Z,O,U,P. An exception is Masked if a corresponding bit in the 8087 Control Word is set to 1. If an exception is Masked, the 8087 will not generate an interrupt when the error condition occurs; it will instead provide its own error recovery.

NaN: an abbreviation for "Not a Number"; a Floating Point quantity which does not represent any numeric or infinite quantity. NaN's should be returned by functions which encounter serious errors. If created during a sequence of calculations, they are transmitted to the final answer, and can contain information about where the error occurred.

NDP: Numeric Data Processor. This is any iAPX86 or iAPX88 system that contains an 8087 or the full 8087 emulator.

Non-Trapping NaN: a NaN in which the most significant bit of the fractional part of the Significand is 1. By convention, these NaN's can undergo certain operations without visible error. Non-Trapping NaN's are implemented for the 8087 via the software in EH87.LIB.

Normal: the representation of a number in a Floating Point format in which the Significand has an Integer bit 1 (either explicit or Implicit).

Normalizing Mode: a state, recommended by the Standard, in which non-normal inputs are automatically converted to normal inputs whenever they are used in arithmetic. Normalizing Mode is implemented for the 8087 via the software in EH87.LIB.

NPX: Numeric Processor Extension. This is either the 8087 or the full 8087 emulator.

Overflow: an error condition in which the correct answer is finite, but has magnitude too great to be represented in the destination format.

Packed Decimal: an Integer format supported by the 8087. A Packed Decimal number is a 10-byte quantity, with nine bytes of 18 Binary Coded Decimal digits, and one byte for the sign.

Pop: to remove from a stack the last item that was placed on the stack.

Precision Control: an option, programmed through the 8087 Control Word, which allows all 8087 arithmetic to be performed with reduced precision. Since no speed advantage results from this option, its only use is for strict compatibility with the Standard, and with other computer systems.

Precision Exception: an 8087 error condition which results when a calculation does not return an exact answer. This exception is usually Masked and ignored; it is used only in extremely critical applications, when the user must know if the results are exact.

Projective Mode: a state of the 8087, selected in the 8087 Control Word, in which infinities are treated as not having a sign. Thus the values +INFINITY and −INFINITY are considered the same. Certain operations, such as comparison to finite numbers, are illegal in Projective Mode but legal in Affine Mode. Thus Projective Mode gives you a greater degree of error control over infinite inputs.

Pseudo Zero: a special value of the Temporary Real format. It is a number with a zero significand and an Exponent which is neither all zeroes or all ones. Pseudo-zeroes can come about as the result of multiplication of two Unnormal numbers; but they are very rare.

Real: any finite value (negative, positive, or zero), which can be represented by a decimal expansion. The fractional part of the decimal expansion can contain an infinite number of digits. Reals can be represented as the points of a line marked off like a ruler. The term "Real" can also refer to a Floating Point Number which represents a Real value.

Short Integer: an Integer format supported by the 8087 which consists of a 32-bit Two's Complement quantity. Short Integer is not the shortest 8087 Integer format—there is the 16-bit Word Integer.

Short Real: a Floating Point Format supported by the 8087, which consists of a sign, an 8-bit Biased Exponent, an Implicit Integer Bit, and a 23-bit Significand; a total of 32 explicit bits.

Significand: the part of a Floating Point Number which consists of the most significant non-zero bits of the number, if the number were written out in an unlimited binary format. The Significand alone is considered to have a Binary Point after the first (possibly Implicit) bit; the Binary Point is then moved according to the value of the Exponent.

Single Extended: a Floating Point format, required by the Standard, which provides greater precision than Single; it also provides an explicit Integer Significand bit. The 8087's Temporary Real format meets the Single Extended requirement as well as the Double Extended requirement.

Single Floating Point Number: the Standard's term for the 8087's 32-bit Short Real format.

Standard: "a Proposed Standard for Binary Floating-Point Arithmetic," Draft 8.0 of IEEE Task P754, *Computer*, March 1981, pp. 51-62.

Status Word: A 16-bit 8087 register which can be manually set, but which is usually controlled by side effects to 8087 instructions. It contains condition codes, the 8087 stack pointer, busy and interrupt bits, and error flags.

Tag Word: a 16-bit 8087 register which is automatically maintained by the 8087. For each space in the 8087 stack, it tells if the space is occupied by a number; if so, it gives information about what kind of number.

Temporary Real: the main Floating Point Format used by the 8087. It consists of a sign, a 15-bit Biased Exponent, and a Significand with an explicit Integer bit and 63 fractional-part bits.

Transcendental: one of a class of functions for which polynomial formulas are always approximate, never exact for more than isolated values. The 8087 supports trigonometric, exponential, and logarithmic functions; all are Transcendental.

Trapping NaN: a NaN which causes an "I" error whenever it enters into a calculation or comparison, even a non-ordered comparison.

Two's Complement: a method of representing Integers. If the uppermost bit is 0, the number is considered positive, with the value given by the rest of the bits. If the uppermost bit is 1, the number is negative, with the value obtained by subtracting ($2^{\text{bit count}}$) from all the given bits. For example, the 8-bit number 11111100 is −4, obtained by subtracting 2^8 from 252.

Unbiased Exponent: the true value that tells how far and in which direction to move the Binary Point of the Significand of a Floating Point Number. For example, if a Short Real Exponent is 131, we subtract the Bias 127 to obtain the Unbiased Exponent +4. Thus the Real number being represented is the Significand with the Binary Point shifted 4 bits to the right.

Underflow: an error condition in which the correct answer is non-zero, but has a magnitude too small to be represented as a Normal number in the destination Floating Point format. The Standard specifies that an attempt be made to represent the number as a Denormal.

Unmasked: a term which applies to each of the six 8087 Exceptions I,D,Z,O,U,P. An exception is Unmasked if a corresponding bit in the 8087 Control Word is set to 0. If an exception is Unmasked, the 8087 will generate an interrupt when the error condition occurs. You can provide an interrupt routine that customizes your error recovery.

Unnormal: a Temporary Real representation in which the explicit Integer bit of the Significand is zero, and the exponent is non-zero. We consider Unnormal numbers as distinct from Denormal numbers.

Word Integer: an Integer format supported by both the 8086 and the 8087 that consists of a 16-bit Two's Complement quantity.

Zerodivide: an error condition in which the inputs are finite, but the correct answer, even with an unlimited exponent, has infinite magnitude.

APPENDIX B EXTERNAL SYMBOLS USED BY THE 8087 EMULATOR

Intel translators put into generated object modules a number of special external symbols when the NPX is used by programs. These symbols are explained and listed in this appendix, so that you can recognize them in listings issued by LINK86 and LOC86.

The machine code generated by an 8087 instruction for the 8087 component is usually the WAIT opcode, followed by an ESCAPE opcode, followed by further code which identifies the operands and/or the specific instruction. (See the *8086/8087/8088 Macro Assembly Language Reference Manual* for the details of these opcodes).

The emulator can be called using the same amount of code. Instead of the WAIT and ESCAPE opcodes, a two-byte call to an 8086 interrupt routine is used. The interrupt routines are written to decode the remaining instruction bytes in the same way that the 8086/8087 decodes them.

Thus, for each 8087 instruction, there are two opcode bytes which differ when the emulator is used instead of the chip. Each two-byte pattern for the 8087 chip is stored under a special name in 8087.LIB. The corresponding two-byte pattern for the emulator is stored under the same name in E8087.LIB. The special names are built into all Intel translators that support the 8087. Instead of issuing the opcode for an 8087 instruction, the translator issues the appropriate external name. LINK86 then supplies the correct opcodes when the externals are satisfied, either by 8087.LIB or E8087.LIB.

87NULL.LIB satisfies the external names with the same opcodes as 8087.LIB. It is up to the user program to insure that any resulting 8087 instructions are never executed.

Here are the special names. The colon given is part of the name; it exists to make it impossible for the name to be generated by any Intel translators.

```
M:_NCS
M:_NDS
M:_NES
M:_NSS
M:_NST
M:_WCS
M:_WDS
M:_WES
M:_WSS
M:_WST
M:_WT
```

APPENDIX C SUMMARY OF 8087 FLOATING POINT FORMATS

The 8087 supports three formats for real numbers. All formats are in reverse order when stored in 8086 memory. That is, the first byte (lowest memory address) is the least significant byte of the Significand; the last byte (highest memory address) contains the sign and the top seven bits of the Exponent.

The three formats supported are the 32-bit Short Real, the 64-bit Long Real, and the 80-bit Temporary Real.

Short Real:

1-bit Sign	
8-bit Exponent:	Bias 126 if Exponent zero; 127 if non-zero
0-bit Implicit Integer Bit:	0 if Exponent zero; 1 if non-zero
23-bit Fractional Part:	digits to the right of the Binary Point

Long Real:

1-bit Sign	
11-bit Exponent:	Bias 1022 if Exponent zero; 1023 if non-zero
0-bit Implicit Integer Bit:	0 if Exponent zero; 1 if non-zero
52-bit Fractional Part:	digits to the right of the Binary Point

Temporary Real:

1-bit Sign	
15-bit Exponent:	Bias 16382 if Exponent zero; 16383 if non-zero
1-bit Explicit Integer Bit	
63-bit Fractional Part:	digits to the right of the Binary Point

Special Values in All Three Formats:

Infinity:	maximum exponent and zero Significand
NaN:	maximum exponent and non-zero Significand
Zero:	minimum exponent and zero Significand
Denormal:	minimum exponent and non-zero Significand

APPENDIX D SUMMARY OF 8087 INSTRUCTIONS

On the following pages you will find a reference chart of NPX instructions. Here is an explanation of each column of the chart.

Opcode: This column gives the machine codes generated by ASM86 and LINK86 for the given instruction. Digits and upper-case letters "A" through "F" are hexadecimal digits. In addition, the following special codes are used:

- "i" denotes a 4-bit quantity whose top bit is 0, and whose bottom three bits give the 8087 stack element number for the instruction.
- "j" denotes "i" plus 8. It is a 4-bit quantity whose top bit is 1, and whose bottom three bits give the 8087 stack element number for the instruction.
- "/" followed by a digit denotes a MODRM byte, as described in the ASM86 language manual. The digit gives the value of the middle REG field of the byte (bits 5,4,3). The MOD field (bits 7,6) can be any of the values 00, 01, or 10. The R/M field (bits 2,1,0) can be any of the 8 possible values. For some values of MOD and R/M, there will be one or two immediate displacement bytes following the MODRM byte, as defined by the 8086 architecture.

The machine codes are those for the 8087 component, produced by linking 8087.LIB to your ASM-86 object modules. If there is a segment override byte, it goes between the first (WAIT or NOP) byte and the second (ESCAPE) byte.

The code for the 8087 emulator, produced by linking E8087.LIB, differs as follows: The FWAIT instruction produces a NOP (90 hex) byte instead of the 8086 WAIT (9B hex) byte. All other instructions produce a first byte of CD hex, the 8086 INT instruction, instead of the WAIT or NOP byte produced for the 8087. If there is no segment override byte, the second byte's top hexadecimal digit is "1" instead of "D", giving an 8086 interrupt number between 18 hex and 1F hex. The remaining parts of the instruction are the same. If there is a segment override byte, the byte is replaced by an interrupt number between 14 hex and 17 hex, as shown below. The third ESCAPE byte remains the same, and is interpreted by the emulator.

Interrupt	Override
14	ES
15	CS
16	SS
17	DS

Instruction: This column gives the 8087 instruction just as it appears in an ASM-86 program. Operands which begin with "mem" can be replaced with any memory operand denoting a data area of the correct number of bytes. The number following "mem" gives the decimal number of memory bytes acted upon. A trailing "r" in the name denotes a REAL format, "i" denotes an INTEGER format, and "d" denotes a Binary Coded Decimal format.

Function: This column gives a concise description of the function performed by the instruction.

Clocks: This column gives the typical number of clock cycles used by the 8087 chip to execute the instruction. It is not an exact number. If a MODRM byte is involved, a typical time of 9 cycles is added for calculating the effective address of the memory operand.

ErrID: This column gives the hexadecimal value returned by the procedure DECODE in the library EH87.LIB, described in Chapter 5. The value indicates the type of instruction which caused an error, and is returned in the OPERATION field of the structure ESTATE87.

Errors: This column lists the possible exceptions which can occur if the instruction is executed.

Opcode	Instruction	Function	Clocks	ErrID	Errors
9B D9 F0	F2XM1	ST ←(2 ** ST) − 1	500	19	UP
9B D9 E1	FABS	ST ← |ST|	14	01	I
9B DE C1	FADD	ST(1) ←ST(1) + ST, pop	90	05	IDOUP
9B DC Ci	FADD ST(i),ST	ST(i) ←ST(i) + ST	85	05	IDOUP
9B D8 Ci	FADD ST,ST(i)	ST ←ST(i) + ST	85	05	IDOUP
9B D8 /0	FADD mem4r	ST ←ST + mem4r	114	05	IDOUP
9B DC /0	FADD mem8r	ST ←ST + mem8r	119	05	IDOUP
9B DE Ci	FADDP ST(i),ST	ST(i) ←ST(i) + ST, pop	90	05	IDOUP
9B DF /4	FBLD mem10d	push, ST ←mem10d	309	10	I
9B DF /6	FBSTP mem10d	mem10d ←ST, pop	539	0F	I
9B D9 E0	FCHS	ST ← − ST	15	02	I
9B DB E2	FCLEX	clear exceptions	5		
9B D8 D1	FCOM	compare ST − ST(1)	45	03	ID
9B D8 Di	FCOM ST(i)	compare ST − ST(i)	45	03	ID
9B D8 /2	FCOM mem4r	compare ST − mem4r	74	03	ID
9B DC /2	FCOM mem8r	compare ST − mem8r	79	03	ID
9B D8 D9	FCOMP	compare ST − ST(1), pop	47	03	ID
9B D8 Dj	FCOMP ST(i)	compare ST − ST(i), pop	47	03	ID
9B D8 /3	FCOMP mem4r	compare ST − mem4r , pop	77	03	ID
9B DC /3	FCOMP mem8r	compare ST − mem8r , pop	81	03	ID
9B DE D9	FCOMPP	compare ST − ST(1), pop 2	50	03	ID
9B D9 F6	FDECSTP	decrement stack pointer	9		
9B DB E1	FDISI	set interrupt mask	5		
9B DE F1	FDIV	ST(1) ←ST(1) / ST, pop	202	09	IDZOUP
9B DC Fj	FDIV ST(i),ST	ST(i) ← ST(i) / ST	198	09	IDZOUP
9B D8 Fi	FDIV ST,ST(i)	ST ← ST / ST(i)	198	09	IDZOUP
9B D8 /6	FDIV mem4r	ST ←ST / mem4r	229	09	IDZOUP
9B DC /6	FDIV mem8r	ST ←ST / mem8r	234	09	IDZOUP
9B DE Fj	FDIVP ST(i),ST	ST(i) ←ST(i) / ST, pop	202	09	IDZOUP
9B DE F1	FDIVR	ST(1) ←ST / ST(1), pop	203	0A	IDZOUP
9B DC Fj	FDIVR ST(i),ST	ST(i) ←ST / ST(i)	199	0A	IDZOUP
9B D8 Fj	FDIVR ST,ST(i)	ST ←ST(i) / ST	199	0A	IDZOUP
9B D8 /7	FDIVR mem4r	ST ←mem4r / ST	230	0A	IDZOUP
9B DC /7	FDIVR mem8r	ST ←mem8r / ST	235	0A	IDZOUP
9B DE Fi	FDIVRP ST(i),ST	ST(i) ←ST / ST(i), pop	203	0A	IDZOUP
9B DB E0	FENI	clear interrupt mask	5		
9B DD Ci	FFREE ST(i)	empty ST(i)	11		
9B DE /0	FIADD mem2i	ST ← ST + mem2i	129	05	IDOP
9B DA /0	FIADD mem4i	ST ← ST + mem4i	134	05	IDOP
9B DE /2	FICOM mem2i	compare ST − mem2i	89	03	ID
9B DA /2	FICOM mem4i	compare ST − mem4i	94	03	ID
9B DE /3	FICOMP mem2i	compare ST − mem2i , pop	91	03	ID
9B DA /3	FICOMP mem4i	compare ST − mem4i , pop	96	03	ID
9B DE /6	FIDIV mem2i	ST ←ST − mem2i	239	09	IDZOUP
9B DA /6	FIDIV mem4i	ST ←ST − mem4i	245	09	IDZOUP
9B DE /7	FIDIVR mem2i	ST ← mem2i / ST	239	0A	IDZOUP
9B DA /7	FIDIVR mem4i	ST ← mem4i / ST	246	0A	IDZOUP
9B DF /0	FILD mem2i	push, ST ← mem2i	59	10	I
9B DB /0	FILD mem4i	push, ST ← mem4i	65	10	I
9B DF /5	FILD mem8i	push, ST ← mem8i	73	10	I
9B DE /1	FIMUL mem2i	ST ← ST * mem2i	139	08	IDOP
9B DA /1	FIMUL mem4i	ST ← ST * mem4i	145	08	IDOP
9B D9 F7	FINCSTP	increment stack pointer	9		
9B DB E3	FINIT	initialize 8087	5		
9B DF /2	FIST mem2i	mem2i ← ST	95	0F	IP
9B DB /2	FIST mem4i	mem4i ← ST	97	0F	IP
9B DF /3	FISTP mem2i	mem2i ← ST, pop	97	0F	IP
9B DB /3	FISTP mem4i	mem4i ← ST, pop	99	0F	IP
9B DF /7	FISTP mem8i	mem8i ← ST, pop	109	0F	IP
9B DE /4	FISUB mem2i	ST ← ST − mem2i	129	06	IDOP

Opcode	Instruction	Function	Clocks	ErrID	Errors
9B DA /4	FISUB mem4i	ST ← ST − mem4i	134	06	IDOP
9B DE /5	FISUBR mem2i	ST ← mem2i − ST	129	07	IDOP
9B DA /5	FISUBR mem4i	ST ← mem4i − ST	134	07	IDOP
9B D9 Ci	FLD ST(i)	push, ST ← old ST(i)	20	10	I
9B DB /5	FLD mem10r	push, ST ← mem10r	66	10	ID
9B D9 /0	FLD mem4r	push, ST ← mem4r	52	10	ID
9B DD /0	FLD mem8r	push, ST ← mem8r	55	10	ID
9B D9 E8	FLD1	push, ST ← 1.0	18	11	I
9B D9 /5	FLDCW mem2i	control word ← mem2i	19		
9B D9 /4	FLDENV mem14	environment ← mem14	49		
9B D9 EA	FLDL2E	push, ST ← log base 2 of e	18	13	I
9B D9 E9	FLDL2T	push, ST ← log base 2 of 10	19	12	I
9B D9 EC	FLDLG2	push, ST ← log base 10 of 2	21	15	I
9B D9 ED	FLDLN2	push, ST ← log base e of 2	20	16	I
9B D9 EB	FLDPI	push, ST ← Pi	19	14	I
9B D9 EE	FLDZ	push, ST ← +0.0	14	17	I
9B DE C9	FMUL	ST(1) ← ST(1) * ST, pop	142	08	IDOUP
9B DC Cj	FMUL ST(i),ST	ST(i) ← ST(i) * ST	138	08	IDOUP
9B D8 Cj	FMUL ST,ST(i)	ST ← ST * ST(i)	138	08	IDOUP
9B D8 /1	FMUL mem4r	ST ← ST * mem4r	127	08	IDOUP
9B DC /1	FMUL mem8r	ST ← ST * mem8r	170	08	IDOUP
9B DE Cj	FMULP ST(i),ST	ST(i) ← ST(i) * ST, pop	142	08	IDOUP
90 DB E2	FNCLEX	nowait clear exceptions	5		
90 DB E1	FNDISI	nowait set interrupt mask	5		
90 DB E0	FNENI	nowait clear interrupt mask	5		
90 DB E3	FNINIT	nowait initialize 8087	5		
9B D9 D0	FNOP	no operation	13		
90 DD /6	FNSAVE mem94	nowait mem94 ← 8087 state	219		
90 D9 /7	FNSTCW mem2i	nowait mem2i ← control word	24		
90 D9 /6	FNSTENV mem14	nowait mem14 ← environment	54		
90 DD /7	FNSTSW mem2i	nowait mem2i ← status word	24		
9B D9 F3	FPATAN	ST ← arctan(ST(1)°ST), pop	650	1D	UP
9B D9 F8	FPREM	ST ← REPEAT(ST − ST(1))	125	1E	IDU
9B D9 F2	FPTAN	push, ST(1)/ST ← tan(old ST)	400	1C	IP
9B D9 FC	FRNDINT	ST ← round(ST)	45	1F	IP
9B DD /4	FRSTOR mem94	8087 state ← mem94	219		
9B DD /6	FSAVE mem94	mem94 ← 8087 state	219		
9B D9 FD	FSCALE	ST ← ST * 2 ** ST(1)	35	18	IOU
9B D9 FA	FSQRT	ST ← square root of ST	183	0C	IDP
9B DD Di	FST ST(i)	ST(i) ← ST	18	0F	I
9B D9 /2	FST mem4r	mem4r ← ST	96	0F	IOUP
9B DD /2	FST mem8r	mem8r ← ST	109	0F	IOUP
9B D9 /7	FSTCW mem2i	mem2i ← control word	24		
9B D9 /6	FSTENV mem14	mem14 ← environment	54		
9B DD Dj	FSTP ST(i)	ST(i) ← ST, pop	20	0F	I
9B DB /7	FSTP mem10r	mem10r ← ST, pop	64	0F	I
9B D9 /3	FSTP mem4r	mem4r ← ST, pop	98	0F	IOUP
9B DD /3	FSTP mem8r	mem8r ← ST, pop	111	0F	IOUP
9B DD /7	FSTSW mem2i	mem2i ← status word	24		
9B DE E9	FSUB	ST(1) ← ST(1) − ST, pop	90	06	IDOUP
9B DC Ej	FSUB ST(i),ST	ST(i) ← ST(i) − ST	85	06	IDOUP
9B D8 Ei	FSUB ST,ST(i)	ST ← ST − ST(i)	85	06	IDOUP
9B D8 /4	FSUB mem4r	ST ← ST − mem4r	114	06	IDOUP
9B DC /4	FSUB mem8r	ST ← ST − mem8r	119	06	IDOUP
9B DE Ej	FSUBP ST(i),ST	ST(i) ← ST(i) − ST, pop	90	06	IDOUP
9B DE E1	FSUBR	ST(1) ← ST − ST(1), pop	90	07	IDOUP
9B DC Ei	FSUBR ST(i),ST	ST(i) ← ST − ST(i)	87	07	IDOUP
9B D8 Ej	FSUBR ST,ST(i)	ST ← ST(i) − ST	87	07	IDOUP
9B D8 /5	FSUBR mem4r	ST ← mem4r − ST	114	07	IDOUP
9B DC /5	FSUBR mem8r	ST ← mem8r − ST	119	07	IDOUP
9B DE Ei	FSUBRP ST(i),ST	ST(i) ← ST − ST(i), pop	90	07	IDOUP
9B D9 E4	FTST	compare ST − 0.0	42	04	ID
9B	FWAIT	wait for 8087 ready			
9B D9 E5	FXAM	C3—C0 ← type of ST	17		
9B D9 C9	FXCH	exchange ST and ST(1)	12	0E	I
9B D9 Cj	FXCH ST(i)	exchange ST and ST(i)	12	0E	I
9B D9 F4	FXTRACT	push, ST(1) ← expo, ST ← sig	50	0B	I
9B D9 F1	FYL2X	ST ← ST(1) * log2(ST), pop	950	1A	P
9B D9 F9	FYL2XP1	ST ← ST(1) * log2(ST+1), pop	850	1B	P

APPENDIX E SUMMARY OF SUPPORT LIBRARY FUNCTIONS

The following pages give condensed summaries of the procedures and functions of the three libraries DCON87.LIB, CEL87.LIB, and EH87.LIB. You can use this appendix as a quick reference guide after you have assimilated the material in the main part of this manual.

The 8087 Emulator is summarized in Appendix D, the table of 8087 instructions.

DCON87.LIB

All input parameters to DCON87.LIB are 4-byte long pointers to 8086 memory buffers which contain the inputs and outputs. The pointer(s) are pushed onto the 8086 stack, high byte first, before calling the procedure. The procedure returns with the pointer(s) popped off the 8086 stack.

The first three procedures have one input pointer. The last four procedures have two input pointers.

mqcBIN__DECLOW (Convert binary number to decimal string)

Input: BIN__DECLOW__BLOCK__PTR →
4-byte BIN__PTR → input binary number.
1-byte BIN__TYPE: 0 for SHORT__REAL
1 for LONG__REAL
2 for TEMP__REAL
1-byte DEC__LENGTH: length of output field
4-byte DEC__PTR → output decimal significand, decimal point at right
2-byte DEC__EXPONENT: base ten exponent of output
1-byte DEC__SIGN: sign of output, in ASCII

Sign and digits output for unusual inputs:
NaN = ".."
+INFINITY = "++"
−INFINITY = "−−"
+0 = "0 "
−0 = "−0"

Errors: I,D,P

mqcDEC__BIN (Convert decimal string to binary number)

Input: DEC__BIN__BLOCK__PTR →
4-byte BIN__PTR → output binary number.
1-byte BIN__TYPE: 0 for SHORT__REAL
1 for LONG__REAL
2 for TEMP__REAL
1-byte DEC__LENGTH: length of input field
4-byte DEC__PTR → input string.

Errors: O,U,P

mqcDECLOW__BIN (Convert decimal string, low-level interface, to binary number)

Input: DECCLOW__BIN__BLOCK__PTR →
4-byte BIN__PTR → output binary number.
1-byte BIN__TYPE: 0 for SHORT__REAL
1 for LONG__REAL
2 for TEMP__REAL
1-byte DEC__LENGTH: length of input field
4-byte DEC__PTR → input string, stripped down to decimal digits
2-byte DEC__EXPONENT: base ten exponent, with decimal point to right of input
1-byte DEC__SIGN: sign of input, in ASCII

Errors: O,U,P

mqcLONG__TEMP (Convert LONG__REAL to TEMP__REAL)

Inputs: LONG__REAL__PTR → input number
TEMP__REAL__PTR → output number

Error: D

mqcSHORT__TEMP (Convert SHORT__REAL to TEMP__REAL)

Inputs: SHORT__REAL__PTR → input number
TEMP__REAL__PTR → output number

Error: D

mqcTEMP__LONG (Convert TEMP__REAL to LONG__REAL)

Inputs: TEMP__REAL__PTR → input number
LONG__REAL__PTR → output number

Errors: I,O,U,P

mqcTEMP__SHORT (Convert TEMP__REAL to SHORT__REAL)

Inputs: TEMP__REAL__PTR → input number
SHORT__REAL__PTR → output number

Errors: I,O,U,P

CEL87.LIB

"x" denotes the 8087 stack top ST.

"y" denotes the 8087 next stack element ST(1).

"STK" denotes a 2-byte integer pushed onto the 8086 stack.

All 8086 and 8087 stack inputs are popped on successful return.

The errors columns give the hexadecimal code left in the 11-bit 8087 opcode register, along with the possible errors the function can produce. Unmasked errors will trap with the input(s) on the 8087 stack if under the "Inputs" column; with the output(s) on the 8087 stack if under the "Outputs" column. The first of the three hex digits tells how many numbers are on the 8087 stack when a trap handler is called.

		Errors, trap with:				
Name	Function	Inputs		Outputs		
mqerACS	x = arc cosine(x)	175	I			
mqerASN	x = arc sine(x)	174	I			
mqerAT2	x = arc tangent(y/x)	277	I	277	U	
mqerATN	x = arc tangent(x)	176	I			
mqerCOS	x = cosine(x)	172	I			
mqerCSH	x = hyperbolic cosine(x)	16F	IO			
mqerDIM	x = max(y-x,+0)	265	I	265	OU	*
mqerEXP	x = e ** x	16B	IOU			
mqerIA2	AX = roundaway(x)	17E	I			
mqerIA4	DXAX = roundaway(x)	168	I			
mqerIAX	x = roundaway(x)	167	I			
mqerIC2	AX = chop(x)	17E	I	17E	P	
mqerIC4	DXAX = chop(x)	179	I	179	P	
mqerICX	x = chop(x)	166	I	166	P	
mqerIE2	AX = roundeven(x)	180	I	180	P	
mqerIE4	DXAX = roundeven(x)	17B	I	17B	P	
mqerIEX	x = roundeven(x)	178	I	178	P	
mqerLGD	x = common log(x)	16D	IZ			
mqerLGE	x = natural log(x)	16C	IZ			
mqerMAX	x = max(x,y)	282	I			*
mqerMIN	x = min(x,y)	281	I			*
mqerMOD	x = (y mod x), has sign(y)	269	I	269	U	
mqerRMD	x = (y mod x), close to 0	27A	I	27A	U	
mqerSGN	x = (y with x's sign)	264	I			*
mqerSIN	x = sine(x)	171	I			
mqerSNH	x = hyperbolic sine(x)	16E	IO			
mqerTAN	x = tangent(x)	173	IZ			
mqerTNH	x = hyperbolic tangent(x)	170	I			
mqerY2X	x = y ** x	26A	IZOU			
mqerYI2	x = x ** AX	27C	IOU			
mqerYI4	x = x ** DXAX	27C	IOU			
mqerYIS	x = x ** STK	27C	IOU			

* mqerDIM, mqerMAX, mqerMIN and mqerSGN can produce "D" errors from within their interiors.

All EH87.LIB procedures operate on a structure ESTATE87, summarized below.

Elements of ESTATE87:

OPERATION WORD	instruction or procedure which caused error
ARGUMENT BYTE	two nibbles, each coded: 0=no operand 1=ST(0) 2=ST(1) 3=ST(REGISTER) 4=see FORMAT 5=TEMP REAL 6=64-bit integer 7=BCD bit 3 means push once
ARG1(5) WORD	value of bottom-nibble argument
ARG1_FULL BYTE	true if ARG1 contains a value
ARG2(5) WORD	value of top-nibble argument
ARG2_FULL BYTE	true if ARG2 contains a value
RESULT BYTE	formats of result, as in ARGUMENT; except bit 3 on means pop once, bit 7 on means pop twice
RES1(5) WORD	value of bottom-nibble result
RES1_FULL BYTE	true if RES1 contains a value
RES2(5) WORD	value of top-nibble result
RES2_FULL BYTE	true if RES2 contains a value
FORMAT BYTE	format of type 4 ARGUMENT or RESULT: 0=32-bit real 1=32-bit integer 2=64-bit real 3=16-bit integer
REGISTER BYTE	8087 stack register number for type 3 ARGUMENT or RESULT
CONTROL_WORD WORD	8087 control word
STATUS_WORD WORD	8087 status word
TAG_WORD WORD	8087 tag word
ERROR_POINTERS(5) WORD	8087 instruction pointer, opcode, operand pointer
STACK_87(40) WORD	8087 stack of 8 temporary real values

DECODE: Push 4-byte ESTATE87_PTR and 2-byte ERRORS87 before calling. DECODE fills ESTATE87 with information about the 8087 error that caused the exception.

ENCODE: Push 4-byte ESTATE87_PTR, 2-byte ERRORS87, 2-byte CONTROL_WORD, 1-byte RETRY_FLAG before calling. ENCODE restores the 8087 to the state indicated by ESTATE87; if RETRY_FLAG is true it retries the error operation using CONTROL_WORD.

FILTER: Push 2-byte ERRORS87 before calling. FILTER calls DECODE, NORMAL, SIEVE, and ENCODE. FILTER returns AL TRUE if either NORMAL or SIEVE returned TRUE.

NORMAL: Push 4-byte ESTATE87_PTR and 2-byte ERRORS87 before calling. NORMAL returns AL TRUE if "D" was the only error in ERRORS87; it also normalizes arguments if operation was not a load operation.

SIEVE: Push 4-byte ESTATE87_PTR and 2-byte ERRORS87 before calling. SIEVE returns AL TRUE if there was a non-trapping NaN which should not have caused an "I" error.

APPENDIX F PUBLIC SYMBOLS IN THE SUPPORT LIBRARY

Each of the three libraries DCON87.LIB, CEL87.LIB, and EH87.LIB contain public symbols other than the names of the procedures documented in this manual. They are internal names, used either within the libraries or by Intel translators.

You should not use any of these names in your programs.

There are no extra public names in the 8087 emulator or interface libraries, other than those listed in Appendix B. The names in Appendix B cannot be generated by Intel translators, so there is no possibility of conflict.

Following is a list of all public names, both documented and undocumented, for each library.

DCON87.LIB

- CHK_UNMSKD_O_U_ERR
- MQCBINDEC
- MQCBIN_DECLOW
- MQCDBX
- MQCDBXDB
- MQCDECBIN
- MQCDECBINLO
- MQCDECLOW_BIN
- MQCDEC_BIN
- MQCLONG_TEMP
- MQCSHORT_TEMP
- MQCSNGXDB
- MQCSNX
- MQCTEMP_LONG
- MQCTEMP_SHORT
- MQCXDB
- MQCXDBDB
- MQCXDBSNG
- MQCXSN
- POWER_OF_10
- UNMSKD_OV_OR_UN
- XCPTN_RTRN

CEL87.LIB

- MQERACS
- MQERAIN
- MQERANT
- MQERASN
- MQERAT2
- MQERATN
- MQERCI2
- MQERCOS
- MQERCSH
- MQERDIM
- MQEREXP
- MQERIA2
- MQERIA4
- MQERIAX
- MQERIC2
- MQERIC4
- MQERICX
- MQERIE2
- MQERIE4
- MQERIEX
- MQERINT
- MQERIRT
- MQERLGD
- MQERLGE
- MQERMAX
- MQERMIN
- MQERMOD
- MQERNI2
- MQERNIN
- MQERRI2
- MQERRMD
- MQERRNT
- MQERSGN
- MQERSIN
- MQERSNH
- MQERTAN
- MQERTNH
- MQERY2X
- MQERYI2
- MQERYI4
- MQERYIS
- MQ_1
- MQ_2XM1
- MQ_63U
- MQ_63U1
- MQ_63UPI2
- MQ_AT2
- MQ_CONST
- MQ_COS
- MQ_CP2N63
- MQ_DECIDE
- MQ_EXIT
- MQ_EXM1
- MQ_I
- MQ_IRCHK
- MQ_LOG
- MQ_LOG10
- MQ_LOGDN
- MQ_MQRPI
- MQ_NAN
- MQ_NOF
- MQ_NORM
- MQ_NQ
- MQ_OF
- MQ_P0
- MQ_PI2
- MQ_PII
- MQ_Q
- MQ_RAD
- MQ_RERR
- MQ_SIN
- MQ_TXAM
- MQ_U0
- MQ_YL2X

EH87.LIB

DECODE
ENCODE
FILTER
FQFORTRANSTATUSCHECK
MQERACS
MQERAIN
MQERANT
MQERASN
MQERAT2
MQERATN
MQERCI2
MQERCOS
MQERCSH
MQERDIM
MQEREXP
MQERINT
MQERIRT
MQERLGD
MQERLGE
MQERMAX
MQERMIN
MQERMOD
MQERN12
MQERNIN
MQERR12
MQERRMD
MQERRNT
MQERSGN
MQERSIN
MQERSNH
MQERTAN
MQERTNH
MQERY2X
MQERY14
NORMAL
SIEVE
TQDECODE87
TQENCODE87
TQFETCH_AND_STORE
TQINSTRUCTION_RETRY
TQNANFILTER
TQNORM87
TQNORMALIZE
TQPOP_THE_TOP
TQREALMATHFILTER
TQRESTORE_PTRS
TQSAVE_PTRS
TQUNPOP_THE_TOP
TQ_312
TQ_320
TQ_322
TQ_324
TQ_326

INDEX

REQUEST FOR READER'S COMMENTS

Intel's Technical Publications Departments attempt to provide documents that meet the needs of al product users. This form lets you participate directly in the documentation process. Your commen help us correct and improve our manuals. Please take a few minutes to respond.

Please restrict your comments to the usability, accuracy, readability, organization, and completen this document. If you have any comments on the equipment software itself, please contact you representative. If you wish to order manuals contact the Intel Literature Department (see page ii manual).

1. Please describe any errors you found in this manual (include page number).

____
____
____
____
____
____

2. Does the document cover the information you expected or required? Please make suggestio improvement.

____
____
____
____
____

3. Is this the right type of document for your needs? Is it at the right level? What other typ documents are needed?

____
____
____
____
____
____

4. Did you have any difficulty understanding descriptions or wording? Where?

____
____
____
____

5. Please rate this publication on a scale of 1 to 5 (5 being the best rating). ____________

NAME ______________________ DATE ____________

TITLE ______________________

COMPANY NAME/DEPARTMENT ______________________

ADDRESS ______________________

CITY ____________ STATE ____________ ZIP CODE ____________
(COUNTRY)

Please check here if you require a written reply. ☐

'D LIKE YOUR COMMENTS ...

document is one of a series describing Intel products. Your comments on the back of this form will us produce better manuals. Each reply will be carefully reviewed by the responsible person. All ments and suggestions become the property of Intel Corporation.

NO POSTAGE
NECESSARY
IF MAILED
IN U.S.A.

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 1040 SANTA CLARA, CA

POSTAGE WILL BE PAID BY ADDRESSEE

Intel Corporation
Attn: Technical Publications M/S 6-2000
3065 Bowers Avenue
Santa Clara, CA 95051

INTEL CORPORATION, 3065 Bowers Avenue, Santa Clara, California 95051 (408) **987-8080**

Printed in U.S.A.