To our customers,

Old Company Name in Catalogs and Other Documents

On April 1st, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

Renesas Electronics website: http://www.renesas.com

April 1st, 2010
Renesas Electronics Corporation

Issued by: Renesas Electronics Corporation (http://www.renesas.com)

Send any inquiries to http://www.renesas.com/inquiry.

Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
2. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
4. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
5. When exporting the products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations. You should not use Renesas Electronics products or the technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations.
6. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
7. Renesas Electronics products are classified according to the following three quality grades: “Standard”, “High Quality”, and “Specific”. The recommended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below. You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application categorized as “Specific” without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for an application categorized as “Specific” or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is “Standard” unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.

   “Standard”: Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots.

   “High Quality”: Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; safety equipment; and medical equipment not specifically designed for life support.

   “Specific”: Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.
8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

RENESAS

APPLICATION NOTE

78K/Ⅱ SERIES

8-BIT SINGLE-CHIP MICROCOMPUTER

FLOATING POINT OPERATION PROGRAM

μPD78214 SERIES
μPD78218A SERIES
μPD78224 SERIES
μPD78234 SERIES
μPD78244 SERIES

Document No. IEA-1273D
(O. D. No. IEA-686D)
Date Published December 1993 P
Printed in Japan

APPLICATION NOTE

NEC

78K/Ⅱ SERIES

8-BIT SINGLE-CHIP MICROCOMPUTER

FLOATING POINT OPERATION PROGRAM

μPD78214 SERIES
μPD78218A SERIES
μPD78224 SERIES
μPD78234 SERIES
μPD78244 SERIES

M7 92.6

Contents Updated in This Edition

Page	Contents
Throughout	Description regarding uPD78P244 has been deleted.

INTRODUCTION

Target readers:

This Application Note is intended for the engineers who understand the functions of the 78K/II series products and design floating-point computation programs using the 78K/II series products.

78K/II series products

- o uPD78214 series : uPD78212, 78213, 78214, 78P214
  uPD78212(A), 78213(A), 78214(A), 78P214(A)
- o uPD78218A series: uPD78217A, 78218A, 78P218A, 78218A(A)
- o uPD78224 series : uPD78220, 78224, 78P224
- o uPD78234 series : uPD78233, 78234, 78237, 78238, 78P238
  uPD78234(A), 78238(A)
- o uPD78244 series : uPD78243, 78244

Objectives:

This Application Note is intended to increase the users' understanding regarding the floating-point computation application programs in the 78K/II series. Note that the program examples included in this document are for reference only and are not for the purpose of mass-production of any application systems.

Configuration:

This Application Note describes the following topics.

- o Computation algorithms
- o Four-rule arithmetic operations
- o Functions (mathematical, coordinate-transformation, type transformation)
- o Program list

The following two application notes are also available:

o Basic part (IEA-607)
o Application part (IEA-700)

Quality grade:

The uPD78212(A) 78213(A), 78214(A) and 78P214(A) are " Special" quality grade versions of the uPD78212, 78213, 78214 and 78P214, respectively.
The uPD78218A(A) is "Special" quality grade version of the uPD78218A.
The uPD78234(A) and 78238(A) are "Special" quality grade versions of the uPD78234 and 78238, respectively.

Typical circuit examples described in this manual are designed for "Standard" quality grade products for general electronics devices. When using the circuits for applications requiring "Special" quality grade, the quality grade for each component and circuit actually used must be confirmed before using.

Please refer to "Quality grade on NEC Semiconductor Devices" (Document Number IEI-1209), published by NEC Corporation, to determine the quality grade specifications on the devices and their recommended applications.

Applications:

- Standard products
  - o Printers
  - o Cameras
  - o Typewriters
  - o PPC
  - o Electronic music instruments
  - o Air conditioners
- Special products
  - o Automotive electronics
  - o Combustion control

Related documents:

- Document commonly related to 78K/II series

Document name		Document number
User's manual Instruction part		IEU-754
SBI users manual		IEM-5040
Application note	Basic part	IEA-607
	Application part	IEA-700
	Floating point operation program part	This application note
Selection guide		IF-304
Instruction reference		IEM-5101
Instruction set		IEM-5102
Development tools Selection guide		EF-231

- Individual documents

o uPD78214 series

Document name \ Product	uPD78212	uPD78213	uPD78214	uPD78P214
Brochure	IB-5036			
Data sheet	IC-8149	IC-7649		IC-7732
User's manual Hardware part	IEM-5119			
Mode register reference	IEM-5100			

Document name \ Product	uPD78212(A)	uPD78213(A)	uPD78214(A)	uPD78P214A
Data sheet	IC-8147	IC-8234		IC-8589
User's manual Hardware part	IEM-5119			
Mode register reference	IEM-5110			

o uPD78218A series

Document name \ Product	uPD78217A	uPD78218A	uPD78P218A	uPD78218(A)
Brochure	IF-288			
Data sheet	IC-8132	IC-8131	IC-8133	IC-8685
User's manual Hardware part	IEU-755			
Special function register reference	IEM-5532			

o uPD78224 series

Document name \ Product	uPD78220	uPD78224	uPD78P224
Brochure	IB-5011		
Data sheet	IC-5457		IC-7757
User's manual Hardware part	IEU-5019		
Special function register reference	IEM-999		

o uPD78234 series

Document name \ Product	uPD78233	uPD78234	uPD78237	uPD78238	uPD78P238
Brochure	IF-207				
Data sheet	IC-7902		IC-8348	IC-8023	IC-8030
User's manual Hardware part	IEU-718				
Special function register reference	IEM-5515				

Document name \ Product	uPD78234(A)	uPD78238(A)
Brochure	-	
Data sheet	IC-8146	IC-8727
User's manual Hardware part	IEU-718	
Special function register reference	-	

o uPD78244 series

Document name \ Product	uPD78243	uPD78244
Brochure	IF-6339	
Data sheet	IC-8355	IC-8070
User's manual Hardware part	IEU-747	
Special function register reference	IEM-5528	

CONTENTS

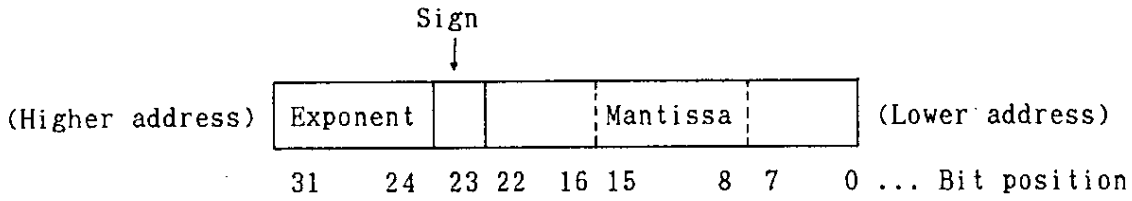
TABLE OF CONTENTS

CHAPTER 1 GENERAL

1.1 Floating Decimal Point Format

This computation program represents a floating-point number in 4 bytes, as follows:

- o Mantissa: 23 bits
- o Exponent: 8 bits
- o Sign : 1 bit



A number in this format is as follows:

$$(-1)^{(sign)} \times (mantissa) \times 2^{(exponent)}$$

The individual details FOR the mantissa, sign, and exponent are as follows:

(1) Mantissa

The mantissa is expressed as an absolute value. Bit positions 22 through 0 of the mantissa respectively correspond to binary numbers at the 2nd through 24th places below the decimal point.

The value of the exponent is adjusted so that the value of the mantissa is always in the 0.5 to 1 range, except when the value of the mantissa is 0. The first place below the decimal point (which signifies the value of 0.5) is always 1, which is represented in a simplified manner in this format.

Remarks:

An operation to always make the most significant bit (MSB) 1 is called normalization.

(2) Sign

The sign is 0 to indicate a positive number; it is 1 to indicate a negative number.

(3) Exponent

An exponent of base 2 is represented as a 1-byte integer (2's complement in the case of a negative number), to which a bias of 80H is further added. The following table shows the relations between the exponent and the value of the exponent.

Exponent (Hex.)	Value of exponent
FF	127
FE	126
:	:
81	1
80	0
7F	-1
:	:
01	-127

Note:

The value of a floating-point number represents 0, only when the exponent is 0. In this case, the mantissa and sign are ignored.

(4) Representation range of value

The value of floating-point number X can be represented in the following range and as "0":

Approx. $2.9387 \times 10^{-39} \leqq |X| \leqq$ approx. 1.7014×10^{38}

1.2 Supplied Functions

This computation program is supplied with the following four types of functions:

- o Arithmetic operation
- o Mathematical functions
- o Coordinate transformation functions
- o Numeric value type transformation functions

Functions	Purpose	Name
Arithmetic operation	Addition Subtraction Multiplication Division	LADD LSUB LMLT LDIV
Mathematical	Trigonometric { sin, cos, tan }	LSIN LCOS LTAN
	Natural logarithm (log) Common logarithm (log10)	LLOG LLOG10
	Exponent (base = e) Exponent (base = 10) Power (ab)	LEXP LEXP10 LPOW
	Square root	LSQRT
	Inverse trigonometric { arcsin, arccos, arctan }	LASIN LACOS LATAN
	Hyperbolic { sinh, cosh, tanh }	LHSIN LHCOS LHTAN
	Absolute value	LABS
	Inverse number	LRCPN
Coordinates transformation	Polar coordinates to rectangular coordinates Rectangular coordinates to polar coordinates	POTORA RATOPO
Numeric value type transformation	Character string to floating-point number Floating-point number to character string	ATOL LTOA

(to be cont'd)

(cont'd)

Functions	Purpose	Name
Numeric value type trans-formation	2-byte integer type to floating-point number Floating-point number to 2-byte integer type	FTOL LTOF
	Floating-point format transformation from IEEE-754 to 78K/II Floating-point format transformation from 78K/II to IEEE-754	LTO78 LTOIE

1.3 File Configuration

This computation program consists of the following four types of files:

- o Object source files* : 26
- o Common subroutine files* : 2
- o Common data file : 1
- o Include files : 4

*: The object source files are described in a structured assembly language.

(1) Object source files

Source file name	Supplied functions
LFLT1.SRC	Arithmetic operation
LSIN.SRC	sin function
LCOS.SRC	cos function
LTAN.SRC	tan function
LLOG.SRC	Natural logarithm function
LLOG10.SRC	Common logarithm function
LEXP.SRC	Exponent function (base = e)
LEXP10.SRC	Exponent function (base = 10)
LPOW.SRC	Power function
LSQRT.SRC	Square root function
LASIN.SRC	arcsin function
LACOS.SRC	arccos function
LATAN.SRC	arctan function
LHSIN.SRC	sinh function
LHCOS.SRC	cosh function
LHTAN.SRC	tanh function
LABS.SRC	Absolute value
LRCPN.SRC	Inverse number
POTORA.SRC	Polar coordinates-to-rectangular coordinates transformation
RATOPO.SRC	Rectangular coordinates-to-polar coordinates transformation
ATOL.SRC	Character string-to-floating-point number transformation
LTOA.SRC	Floating-point number-to-character string transformation
FTOL.SRC	2-byte integer type-to-floating-point number transformation
LTOF.SRC	Floating-point number-to-2-byte integer type transformation
LTO78.SRC	Floating-point format transformation from IEEE-754 to 78K/II
LTOIE.SRC	Floating-point format transformation from 78K/II to IEEE-754

(2) Common subroutine files

Source file name	Supplied functions
LFLT2.SRC	Polynomial computation function
LLD.SRC	Inter-floating-point register load/substitutional function

(3) Common data file

Source file name	Supplied functions
DFLT.SRC	Floating-point register definition

(2) Include files

Source file name	Supplied functions
EQU.INC	EQU definition
REF1.INC	Floating-point register reference declaration
REF2.INC	Inter-floating-point register load/substitutional function usage declaration
ASCII.INC	ASCII code definition

The object module files to be linked to use each function are described in the description for each function.

Remarks: NEC's 78K/II Series Assembler Package is supplied with a librarian, which can be used to create a library file. By registering all the above programs in the library file, necessary modules can be automatically linked, merely by the library file.
The programs can be registered in the library file in any sequence.

1.4 Program Characteristics

1.4.1 Program location address

All the work areas used by this computation program are defined in a short direct addressing area as floating-point registers (actually, they are merely global variables. However, as they can only be used for floating-point computation, they are called registers).
To reserve an area for these floating-point registers, DFLT (object modules to be linked) is used. The other modules externally reference the floating-point registers and, therefore, must be located in the ROM. However, they are not restricted in any other ways.

1.4.2 Reentrance

The program is not reentrant.
In a system that processes multiple tasks, the resources must be managed so that only one task can call this function.

1.4.3 Stack

The maximum size area occupied by each function is described in the description for each function. To use a function, it is necessary to prepare the specified size stack.

1.4.4 Registers and flags storage

General-purpose registers and flags are not stored. If necessary, save them to stack before calling this function.

1.4.5 Register bank

This computation program does not use an instruction that selects a register bank. The program uses a register bank, which has been selected when the program has been called.

1.4.6 Processing time

The processing time specified for each function is when uPD78213 operates on a 12-MHz external clock (internal system clock f_{CLK} = 6 MHz) and the stack area is from 0FE00H-0FE20H. If the high-speed fetch function for the ROM-contained model is used at the stack area location, the execution time changes.

1.5 Data Transfer/Reception

1.5.1 Parameter and return value

Data are transferred or received through the floating-point registers mentioned above.
Throughout this program, a value to be operated and a return value are stored in the same register. For this reason, the value to be operated is called a destination (d) and the operating value is called a source (s) throughout this manual.
Details regarding the setting of parameters and return values are described in the description of each function.
The floating-point register also serves as a work register for this computation program. Therefore, like the general-purpose register, its contents are not stored.

1.5.2 Report on operation result

The result of an operation can be classified into the following five types:

(1) Normal termination
(2) Underflow
(3) Overflow
(4) Mantissa
(5) Operation impossible (log (-1), etc.)

In case of an underflow, a value of 0 is returned as a value indicating the normal termination.
The report indicating the result of an operation does not distinguish error states in (3), (4), and (5) above, and it only indicates whether the result is normal or abnormal.

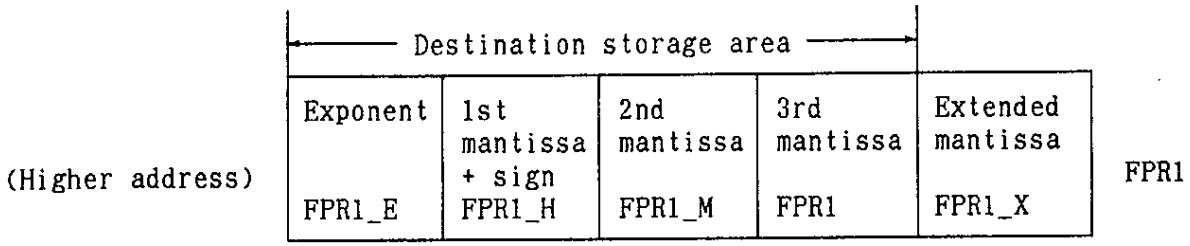
Termination state	A register contents	CY flag
Normal	0	0
Abnormal	81H	1

1.6 Floating-Point Registers

Throughout this Application Note, the work areas used for the computation program are called floating-point registers. This section briefly describes the role of each register.
In the figures shown below, one rectangle indicates 1 byte, and the leftmost position indicates the highest address (this rule of description also applies in the other parts of this Application Note).

1.6.1 Floating-point register 1 (FPR1)

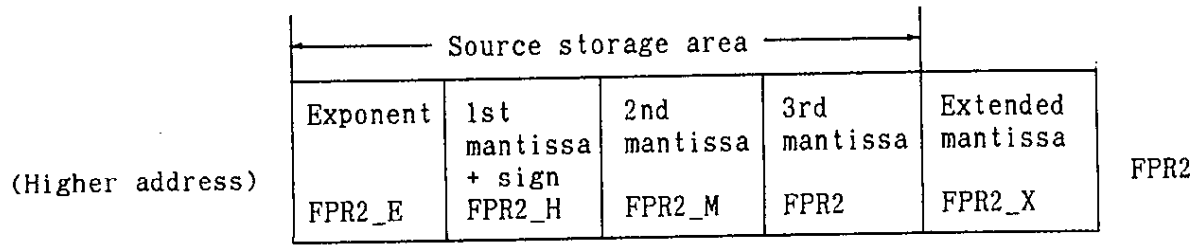
This register is used with almost all the functions to store the destination. It also plays the central role in computation.
The FPR1 register consists of five contiguous bytes, as follows:



Each of the five bytes making up the register is given an area name. The four bytes, FPR1-FPR1_E, are areas that store the destination.
FPR1_X is an area that internally extends the mantissa by 1 byte, in order to compute the 1 byte as the fourth mantissa (the 25th to 32nd places below the decimal point).

1.6.2 Floating-point register 2 (FPR2)

This register is used to store the source, and consists of five contiguous bytes, as follows:

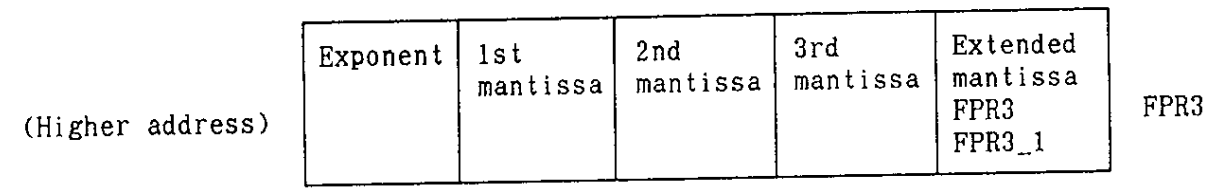


The configuration for this register is the same as that for FPR1.

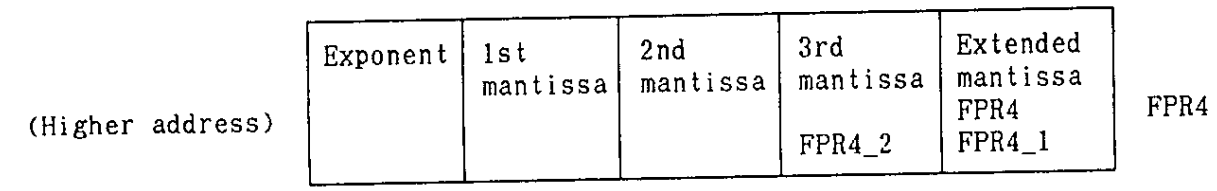
1.6.3 Floating-point registers 3 through 5 (FPR3, 4, and 5)

These registers are used as temporary work registers, when a mathematical function is executed.
Like FPR1 and FPR2, each of these registers consists of five contiguous bytes.

(1) Floating-point register 3 (FPR3)



(2) Floating-point register 4 (FPR4)



(3) Floating-point register 5 (FPR5)

(Higher address)

Exponent	1st mantissa	2nd mantissa	3rd mantissa FPR5_2	Extended mantissa FPR5 FPR5_1

FPR5

Note:

Registers FPR3, FPR4, and FPR5 are not used by all the functions. Which register is used by which function is described in the description for individual functions.

1.6.4 Inter-floating-register load/substitution

The mathematical functions and type transformation functions frequently load, store, or substitute registers. Therefore, the following functions that load, store, and substitute the floating-point registers are provided:

Function	Purpose
LLD21, LLD21X	Loads contents of first register to second register
LLD31, LLD31X	Loads contents of first register to third register
LLD41, LLD41X	Loads contents of first register to fourth register
LLD51, LLD51X	Loads contents of first register to fifth register
LLD52	Loads contents of second register to fifth register
LLD13	Loads contents of third register to first register
LLD23, LLD23X	Loads contents of third register to second register
LLD14, LLD14X	Loads contents of fourth register to first register
LLD24, LLD24X	Loads contents of fourth register to second register
LLD15	Loads contents of fifth register to first register
LLD25, LLD25X	Loads contents of fifth register to second register
LLD1C, LLD1CX	Loads constant data to first register
LLD2C, LLD2CX	Loads constant data to second register
LXC13, LXC13X	Substitutes contents of first register and third register
LXC14X	Substitutes contents of first register and fourth register
LXC15, LXC15X	Substitutes contents of first register and fifth register

Note:
Functions with suffix "X" also load or substitute the extended mantissa.

CHAPTER 2 COMPUTATION ALGORITHM

This chapter briefly describes the computation algorithm.

2.1 Functions Expansion

The following three expansion methods are used:

- o Square root : Newton-Raphson expansion
- o Inverse trigonometric: Best approximation
- o Other methods : Taylor expansion

2.2 Rounding off

Rounding off is executed toward 0.

2.3 Digit Drop Prevention

If an expansion polynomial expression for a Taylor expansion is a difference series, an extreme digit drop may occur, depending on the operand value range. To prevent such a digit drop, this computation program uses an expansion expression, with which the value of each of the first term to the Nth term monotonically and rapidly becomes close to 0. In addition, the mantissa is 32 bits long.

2.4 Error in Polynomial Addition/Multiplication

When addition of a polynomial expression with eight terms, which executes rounding off toward 0, is performed in a system with 24 bits of valid digits, a maximum of 4 bits of errors are included. A similar error is also included, when X^8 is rounded off by carrying out multiplication seven times.
To reduce the error in arithmetic operations (addition and multiplication), this computation program internally computes a mantissa as 32 bits (with an 8-bit extended mantissa added).

CHAPTER 3 ARITHMETIC OPERATION

The following four arithmetic operation functions are supplied:

(1) Floating-point addition (LADD)
Performs addition with the FPR1 value as the augend and the FPR2 value as the addend.

(2) Floating-point subtraction (LSUB)
Performs subtraction with the FPR1 value as the minuend and the FPR2 value as the subtrahend.

(3) Floating-point multiplication (LMLT)
Performs multiplication with the FPR1 value as the multiplicand and the FPR2 value as the multiplier.

(4) Floating-point division (LDIV)
Performs division with the FPR1 value as the dividend and the FPR2 value as the divisor.

The result of an excuted arithmetic operation is stored in FPR1.

3.1 Floating-Point Addition (LADD)

(1) Processing
Takes the FPR1 value as x and the FPR2 value as y, and returns x + y to FPR1.

(2) Object module files to be linked
DFLT, LFLT1

(3) Consumed stack size
2 (only return address from LADD)

(4) Registers
FPR1, FPR2

(5) Processing time (internal system clock: 6 MHz)
Average: 142 us
Maximum: 296 us (0.5 + (-0.500000060))

(6) Processing sequence

(a) If x or y is 0, takes the value which is not 0 as the result and terminates the operation.

(b) If the error in the exponent is greater than 32, takes whichever is greater as the result and terminates the operation.

(c) Adjusts a mantissa with the smaller exponent, in accordance with a mantissa with the greater exponent.

(d) Adds or subtracts the mantissa. If the original signs are the same, it performs addition; if not, it performs subtraction.

(e) Performs carry for addition, or zero judgment for subtraction.

(f) Performs normalization, and stores the operation result in FPR1.

(7) Work area

(a) As work areas for processing the mantissa, the DEHL register is used for FPR1, and the AXBC register is used for FPR2.

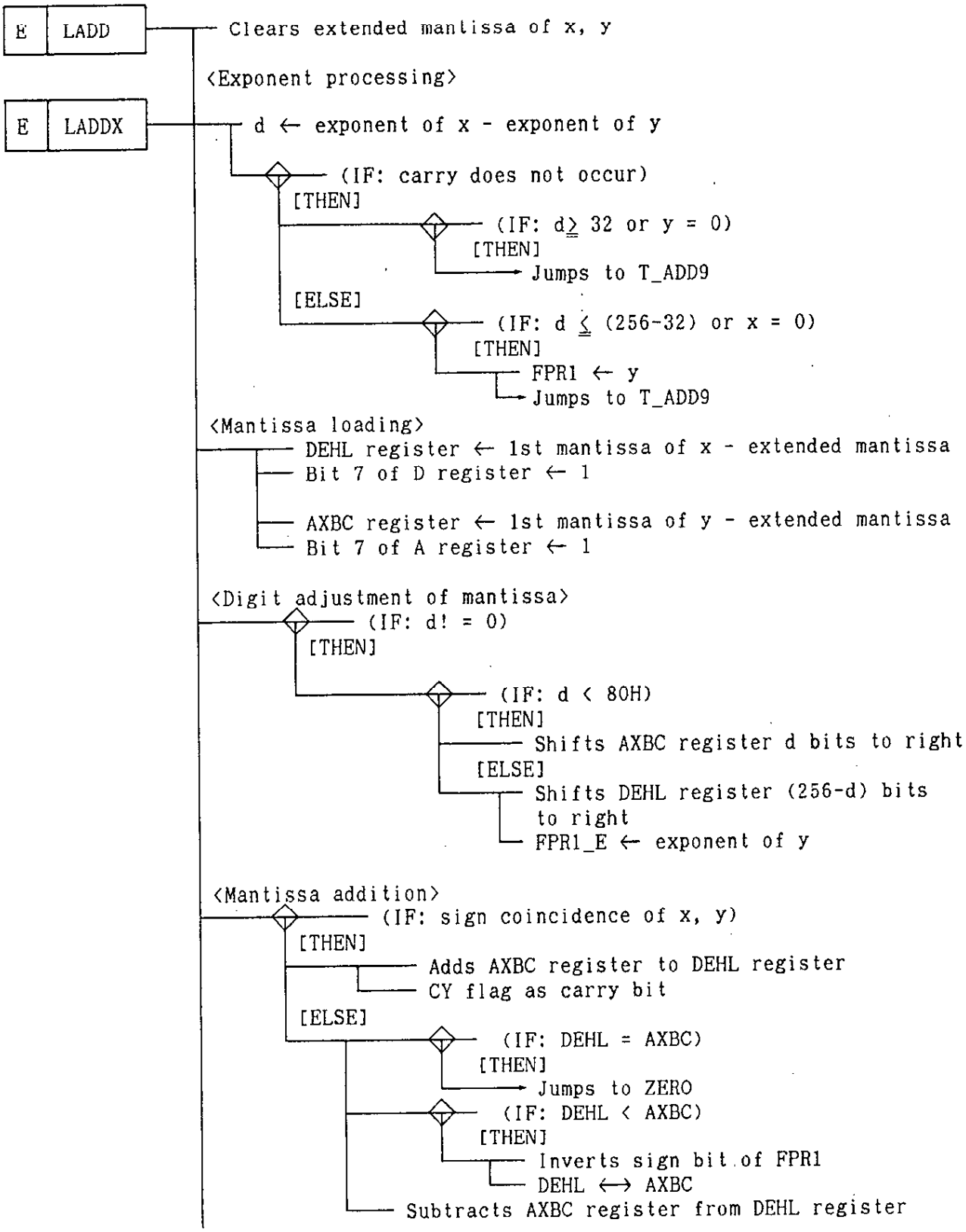
D ← FPR1_H, A ← FPR2_H
E ← FPR1_M, X ← FPR2_M
H ← FPR1 , B ← FPR2
L ← FPR1_X, C ← FPR2_X

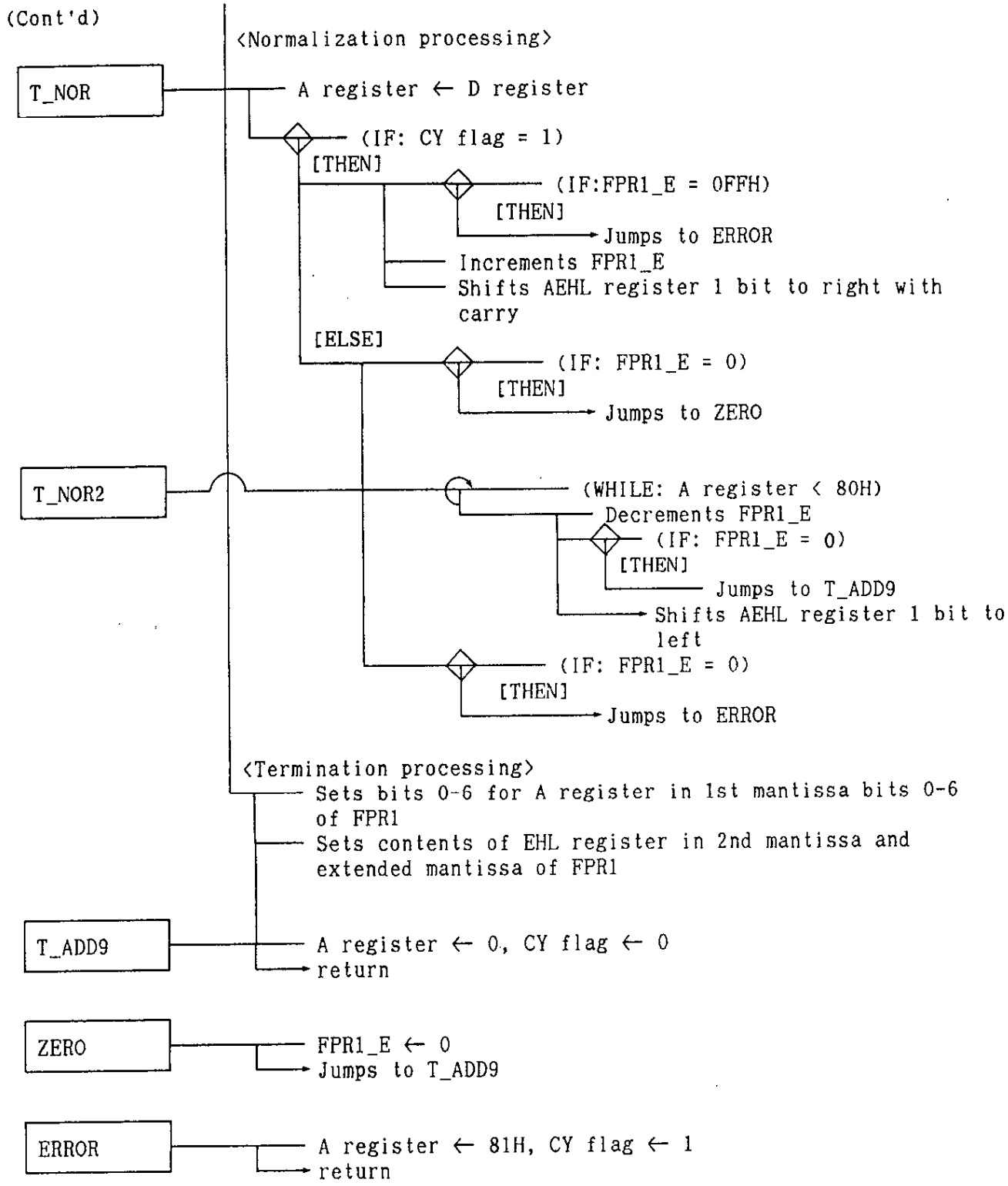
The 7 bits of the D and A registers are used as the MSBs, instead of the sign bits.

(b) The normalization processing uses the AEHL register for the first mantissa and extended mantissa.

(c) FPR1_X is used as an area to which an exponent error is to be stored (in the processing chart, a variable name d is used).

(8) Processing chart




(Cont'd)
<Normalization processing>
T_NOR
A register ← D register
(IF: CY flag = 1)
[THEN]
(IF:FPR1_E = 0FFH)
[THEN]
Jumps to ERROR
Increments FPR1_E
Shifts AEHL register 1 bit to right with carry
[ELSE]
(IF: FPR1_E = 0)
[THEN]
Jumps to ZERO
T_NOR2
(WHILE: A register < 80H)
Decrements FPR1_E
(IF: FPR1_E = 0)
[THEN]
Jumps to T_ADD9
Shifts AEHL register 1 bit to left
(IF: FPR1_E = 0)
[THEN]
Jumps to ERROR
<Termination processing>
Sets bits 0-6 for A register in 1st mantissa bits 0-6 of FPR1
Sets contents of EHL register in 2nd mantissa and extended mantissa of FPR1
T_ADD9
A register ← 0, CY flag ← 0
return
ZERO
FPR1_E ← 0
Jumps to T_ADD9
ERROR
A register ← 81H, CY flag ← 1
return

Remarks: 1. Label | E | | indicates an area name.

2. Labels | T_NOR | , | T_NOR2 | , | ZERO | , and | ERROR | are also referenced by LMLT, and LDIV functions.

3. Label | E | LADDX | is an internal area name, used to execute addition while implementing the extended mantissa by a mathematical function.

3.2 Floating-Point Subtraction (LSUB)

(1) Processing

Takes the FPR1 value as x and the FPR2 value as y, and returns x - y to FPR1.

(2) Object module files to be linked

DFLT, LFT1

(3) Consumed stack size

2 (only return address from LSUB)

(4) Registers

FPR1, FPR2

(5) Processing time (internal system clock: 6 MHz)

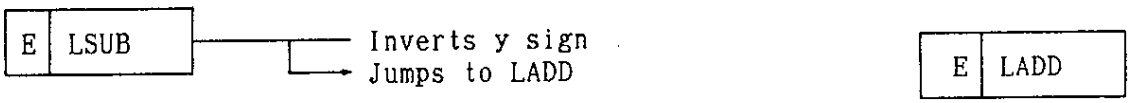
Average: 150 us

Maximum: 300 us (0.5 - 0.500000060)

(6) Processing sequence

Inverts the y sign and jumps to LADD.

(7) Processing chart



3.3 Floating-Point Multiplication (LMLT)

(1) Processing

Takes the FPR1 value as x and the FPR2 value as y, and returns x X y to FPR1.

(2) Object module files to be linked

DFLT, LFLT1

(3) Consumed stack size

2 (only return address from LMLT)

(4) Registers

FPR1, FPR2

(5) Processing time (internal system clock: 6 MHz)

Average: 125 us

Maximum: 133 us (1 x 0.5)

(6) Processing sequence

(a) Returns 0, if x or y is 0.

(b) Adds the exponent and detects an overflow and underflow.

(c) XORs the sign and takes the result as the sign.

(d) Multiplies the mantissa as follows:

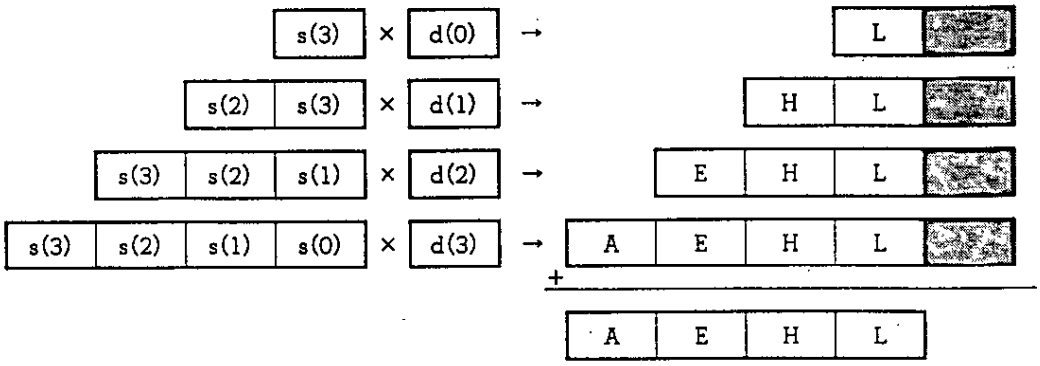
Regards the FPR1 and FPR2 contents, except the exponents, as 4-element BYTE type arrays d(4) and s(4), for processing the mantissa (the subscript takes the 0 to 3 value).

FPR1_H	FPR1_M	FPR1	FPR1_X	as d(4)

FPR2_H	FPR2_M	FPR2	FPR2_X	as s(4)

The bits 7 of d(3) and s(3) are used as the MSBs, instead of the sign bits.

The multiplication results are loaded to the AEHL registers in the sequence illustrated below (the blank portions are truncated).

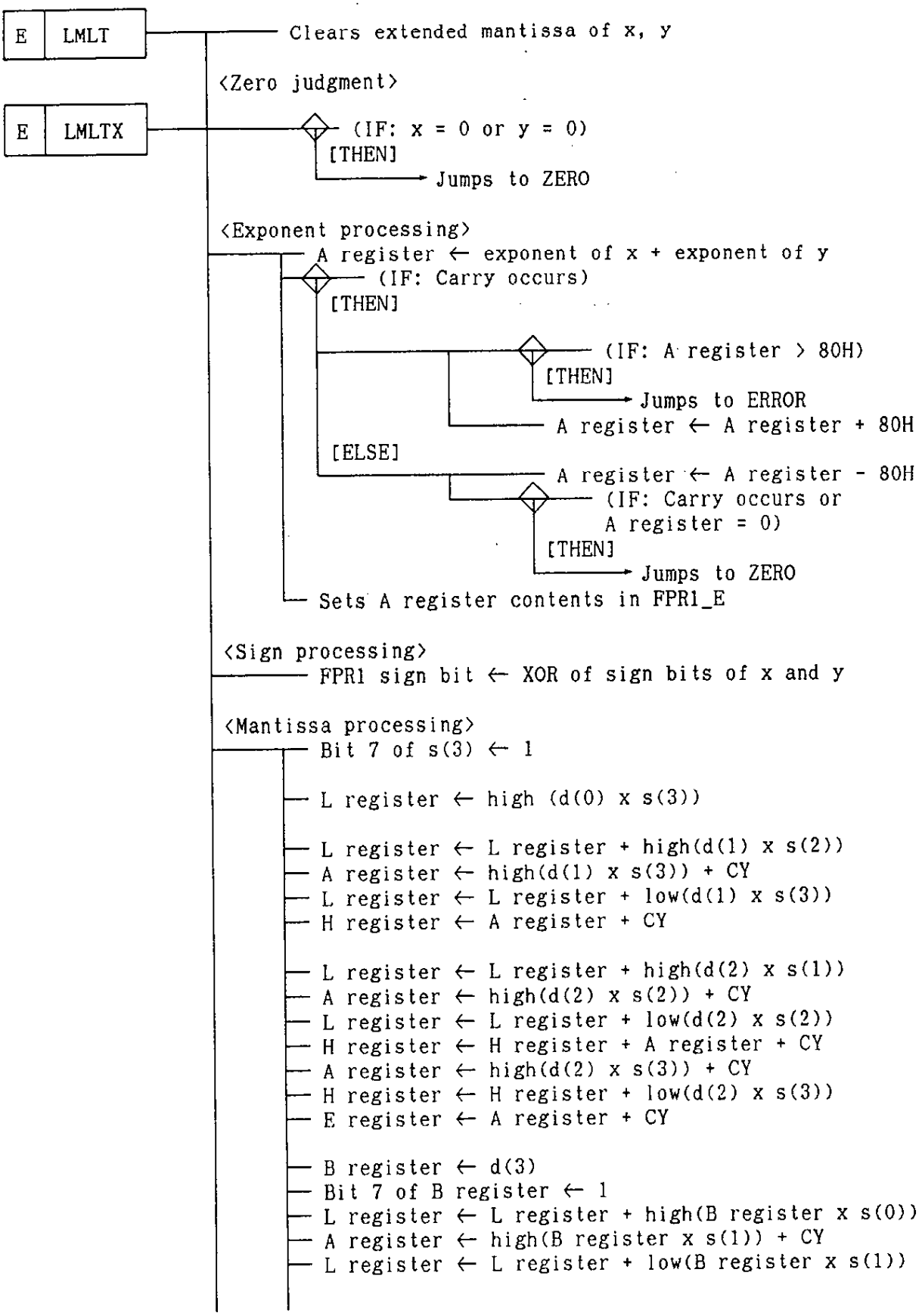


(e) Jumps to the normalization routine.

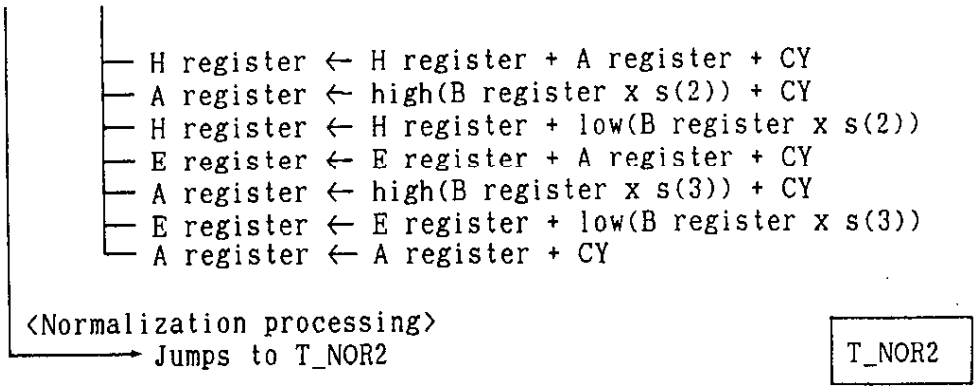
(7) Work area

The AEHL register is used to load the mantissa multiplication result.

(8) Processing chart



(Cont'd)



Remarks: 1. high( ) indicates the higher 1 byte of the multiplication result.

2. low( ) indicates the lower 1 byte of the multiplication result.

3. The label | E | LMLTX | is the name of an internal area, where multiplication using the mantissa is executed with a mathematical function.

3.4 Floating-Point Division (LDIV)

(1) Processing

Takes the FPR1 value as x and the FPR2 value as y, and returns x/y to FPR1.

(2) Object module files to be linked

DFLT, LFLT1

(3) Consumed stack size

2 (only return address from LDIV)

(4) Registers

FPR1, FPR2

(5) Processing time (internal system clock: 6 MHz)

Average: 510 us

Maximum: 609 us (-0.999999940/-0.5)

(6) Processing sequence

(a) Abnormally terminates, if y is 0.

(b) Returns 0, if x is 0.

(c) Subtracts the exponent of y from the exponent of x and detects an overflow and underflow.

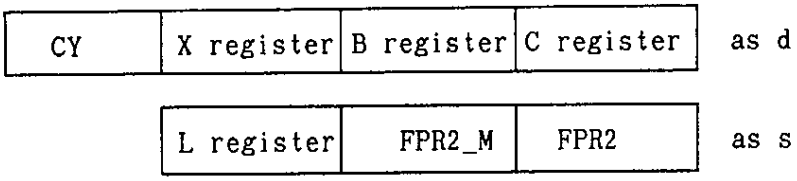
(d) XORs the signs and takes the result as the sign.

(e) Divides the mantissa, as follows:

Regards the mantissa of x substituted for by the CY flag and X, B, and C registers as 25-bit d and s, and the mantissa of y with the first mantissa substituted by the L register as a 24-bit integer (unsigned) type d and s.

```
CY ←       0,
X  ←  FPR1_H,    L  ←  FPR2_H
B  ←  FPR1_M,
C  ←  FPR1   ,
```

The bit 7 of the X and L register is used as the MSB, instead of the sign bit.



By the ordinary computation method, the quotient is loaded to the CY flag and DEH register.

(i) Initializes the CY flag, X, B, C, and L registers, and clears the H register.

(ii) Compares d and s. If d $\geq$ s, subtracts s from d, and sets the carry flag (1). If d < s, resets the carry flag (0).

(iii) Shifts the entire DEH register 1 bit to the left with the carry.

(iv) Shifts d 1 bit to the left.

(v) Repeats (ii) through (iv) 25 times (however, (iv) is not executed at the 25th time).

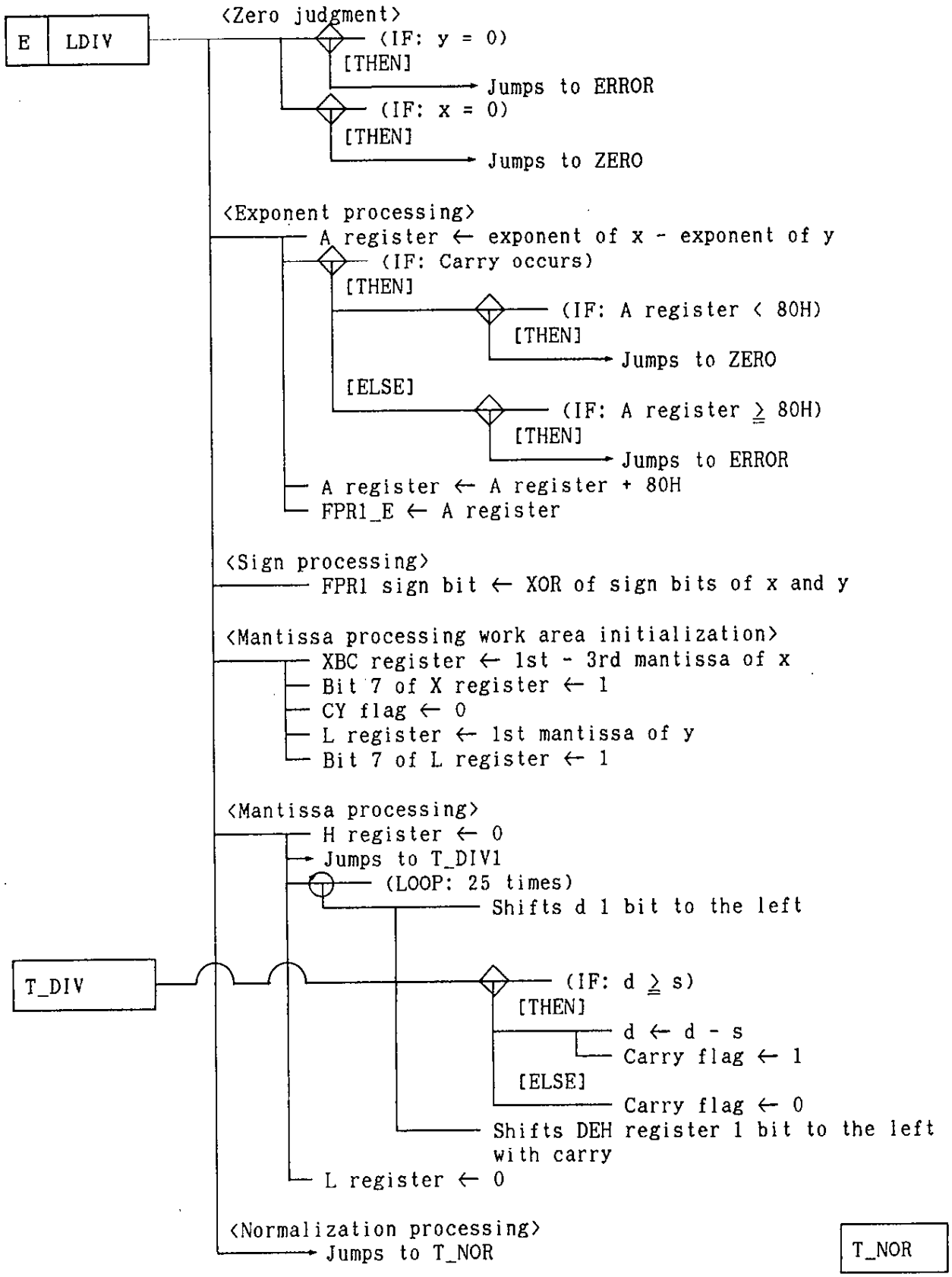
(vi) Takes the carry that has occurred in (iii) of the last processing as the carry bit of the mantissa.

(f) Jumps to the normalization routine.

(7) Work area

FPR1 (1 byte) is used as a loop counter for mantissa division.

(8) Processing chart



CHAPTER 4 MATHEMATICAL FUNCTIONS

The following mathematical functions are supplied:

(1) sin function (LSIN)
Solves for the sine of the FPR1 value.

(2) cos function (LCOS)
Solves for the cosine of the FPR1 value.

(3) tan function (LTAN)
Solves for the tangent of the FPR1 value.

(4) Natural logarithm (LLOG)
Solves for the natural logarithm of the FPR1 value.

(5) Common logarithm (LLOG10)
Solves for the command logarithm of the FPR1 value.

(6) Exponent function (base = e) (LEXP)
Solves for the exponent function with the FPR1 value as the exponent value and base e.

(7) Exponent function (base = 10) (LEXP10)
Solves for the exponent function with the FPR1 value as the exponent value and base 10.

(8) Power function (LPOW)
Solves for the power, which is raised to the FPR2 value, of the FPR1 value.

(9) Square root function (LSQRT)
Solves for the square root of the FPR1 value.

(10) arcsin function (LASIN)
Solves for the arc sine of the FPR1 value.

(11) arccos function (LACOS)
Solves for the arc cosine of the FPR1 value.

(12) arctan function (LATAN)
Solves for the arc tangent of the FPR1 value.

(13) sinh function (LHSIN)
Solves for the hyperbolic sine function of the FPR1 value.

(14) cosh function (LHCOS)
Solves for the hyperbolic cosine function of the FPR1 value.

(15) tanh function (LHTAN)
Solves for the hyperbolic tangent function of the FPR1 value.

(16) Absolute value function (LABS)
Solves for the absolute value of the FPR1 value.

(17) Inverse number function (LRCPN)
Solves for the inverse number of the FPR1 value.

The operation result is stored in FPR1.

4.1 Common Subroutine

As described earlier, the mathematical functions, except for the square root function, are computed by using Taylor approximation or best approximation expressions. These approximation expressions are expressed as polynomial expressions of higher degree, and have common patterns of Taylor expansion and best approximation.

Therefore, computation functions (LPLY and LPLY2) are provided as common subroutines for each of these two types of polynomial expressions.

(1) Processing

Returns the result of a polynomial expression to FPR1.

(2) Algorithm

The following two types of polynomial expressions can be computed:

①	$X + k_1XY + k_1k_2XY^2 + k_1k_2k_3XY^3 + \ldots + k_1k_2 \ldots k_nXY^n$
②	$Z + k_1XY + k_1k_2XY^2 + k_1k_2k_3XY^3 + \ldots + k_1k_2 \ldots k_nXY^n$

where,

X : first term of polynomial expression

Y : multiplication constant for change in degree of each term (X^2, etc.)

(k_1, k_1k_2, ..., $k_1k_2 \ldots k_n$): coefficient of each term

Expression	Conditions	Function
①	When each term has common measure X	LPLY
②	When first term is a constant	LPAY2

To minimize the error, X, Y, Z, k_1, k_2, ... k_n use a floating-point number having an extended mantissa.

Note: The Taylor approximation requires that the following condition be satisfied:

| 1st term | > | 2nd term | > ... | nth term |

(3) Processing sequence

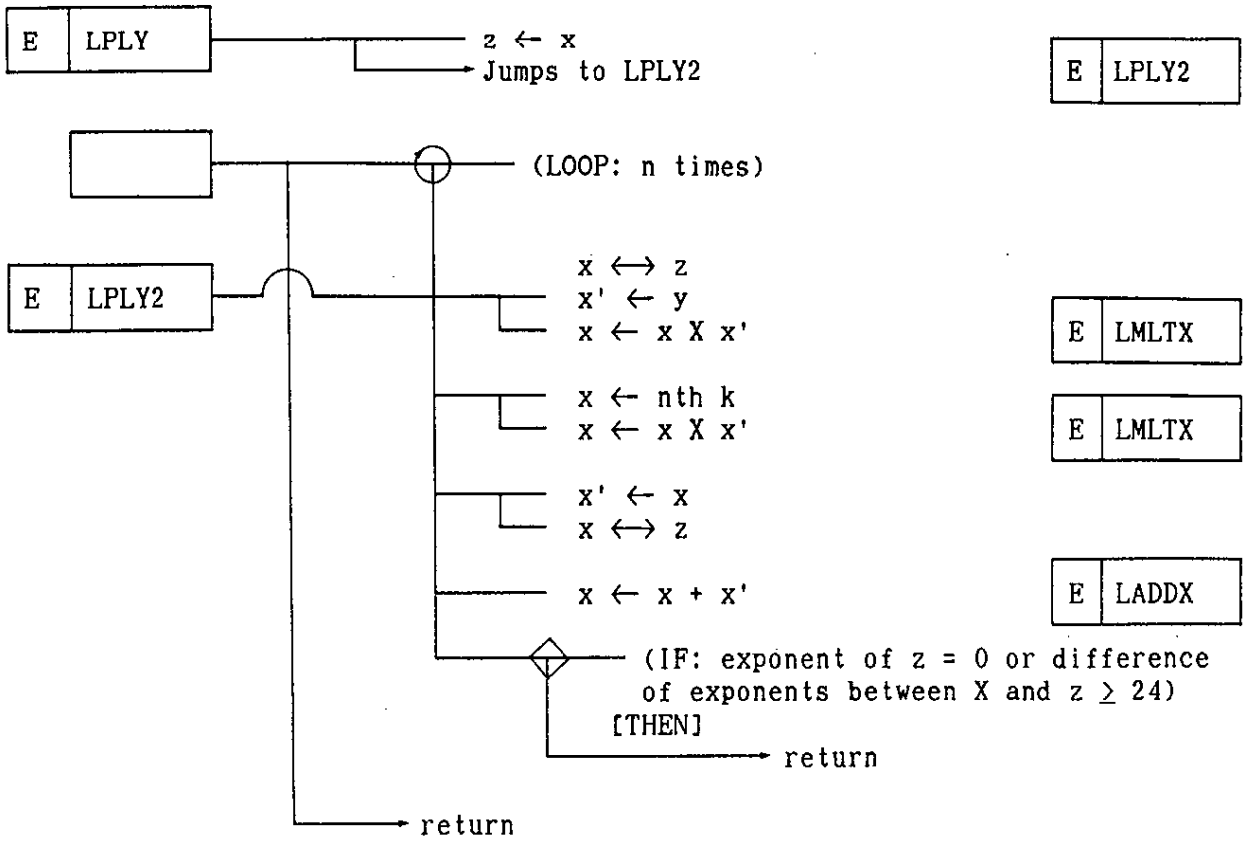
(a) Obtains the Nth term by the Nth term -1st term from, and multiplying Y by k_n.

(b) Adds the value of the Nth term to the sum of the Nth term - 1st term.

(c) Ends the computation, if the Nth term is much smaller than the sum of all terms up to the Nth term.

(d) Repeats (a) to (b), up to the nth term.

(4) Processing chart



Remarks: x : FPR1
x': FPR2
z : FPR3
y : FPR4
n : C register
The first address of k is transferred by the DE register.

4.2 sin Function (LSIN)

(1) Processing

Takes the FPR1 value as x and returns sin(x) to FPR1.

- o Unit: radian

(2) Object module files to be linked

DFLT, LFLT1, LFLT2, LLD, LSIN, FTOL, LTOF

(3) Consumed stack size

10 (including 2 bytes of return address from LSIN)

(4) Registers

FPR1, FPR2, FPR3, FPR4

(5) Processing time (internal system clock: 6 MHz)

Average: 2.69 ms

Maximum: 7.10 ms (sin(1.701412e + 38))

(6) Algorithm

The following Taylor expansion approximation expression is used:

$$SIN(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \frac{x^{11}}{11!}$$

(7) Processing sequence

(a) If x is more negative than SIN(-x) = -SIN(x), registers the sign and obtains the absolute value of x.

(b) Scales x to within the range of $0 \leqq x < \pi/2$.
Where $0 \leqq x' < \pi/2$,

$$\begin{aligned} SIN(\pi/2 + x') &= SIN(\pi/2 - x') \\ SIN(2\pi/2 + x') &= SIN(-x') \\ SIN(3\pi/2 + x') &= SIN(x' - \pi/2) \end{aligned}$$

Therefore, where the quotient of x divided by $\pi/2$ is n and the remainder is x', SIN(x) is:

Remainder of n/4	SIN(x)	x"
0	SIN(x')	x'
1	SIN($\pi/2$ - x')	$\pi/2$ - x'
2	SIN(-x')	-x'
3	SIN(x' - $\pi/2$)	x' - $\pi/2$

(c) Multiplies x" by the registered sign.

(d) Obtains SIN(x") by a Taylor approximation expression.
Calls a polynomial expression computation function with the first term (x") of the approximation polynomial expression, the ratio ($x"^2$), except for the coefficient of each term, and the ratio to the coefficient of the preceding term of the second to sixth terms (-1/3!, -3!/5!, -5!/7!, -7!/9!, -9!/11!) as arguments.

(8) Work area
The following work areas are used for the processing to obtain the quotient by $\pi/2$ and remainder:

(a) FPR4_1 is used to load the quotient (this is expressed as the variable name n in the processing chart).

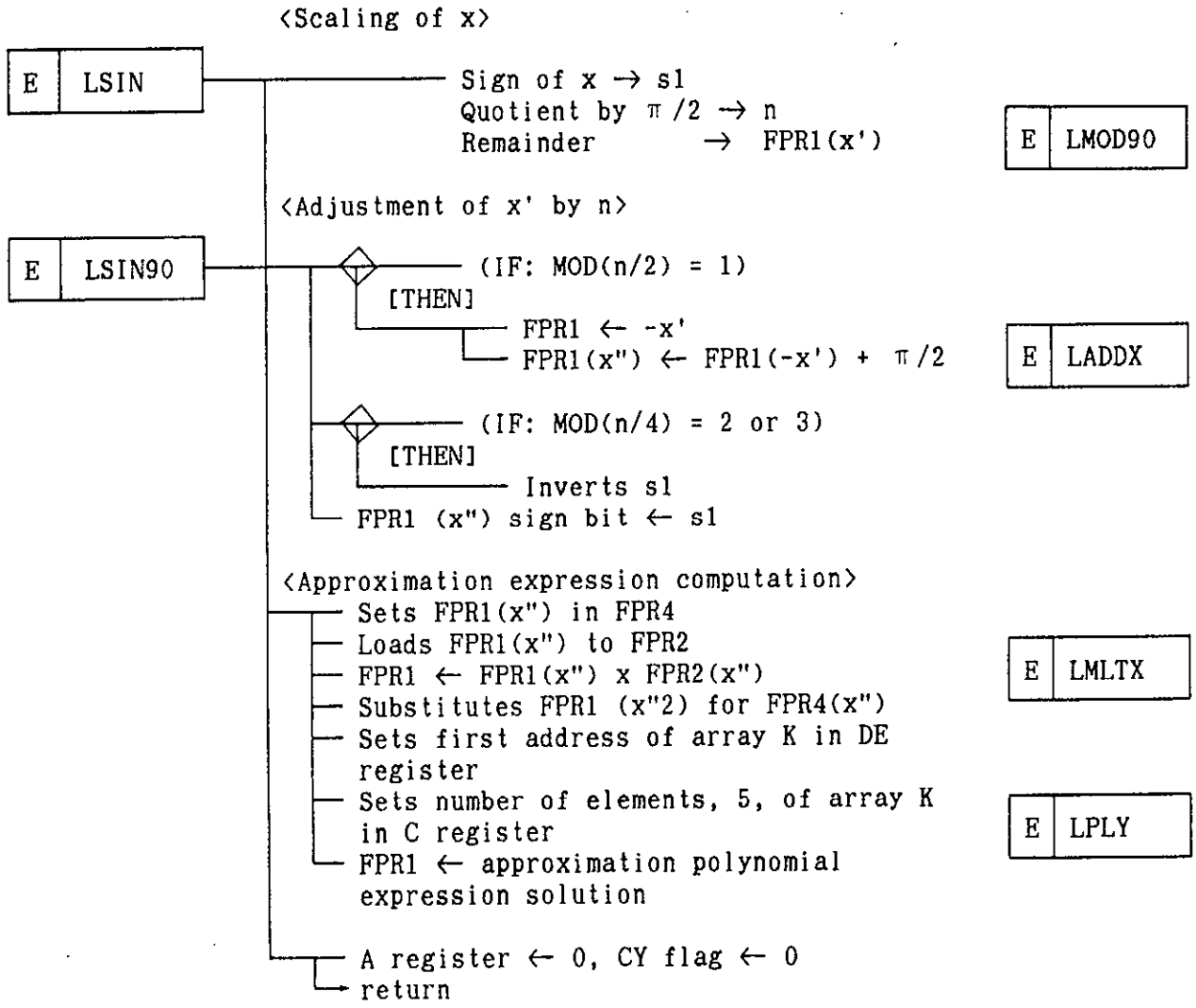
(b) FPR4_2 is used as a work area, to which the sign of x is to be saved (this is expressed as the variable name s1 in the processing chart).

(9) Floating-point constant data

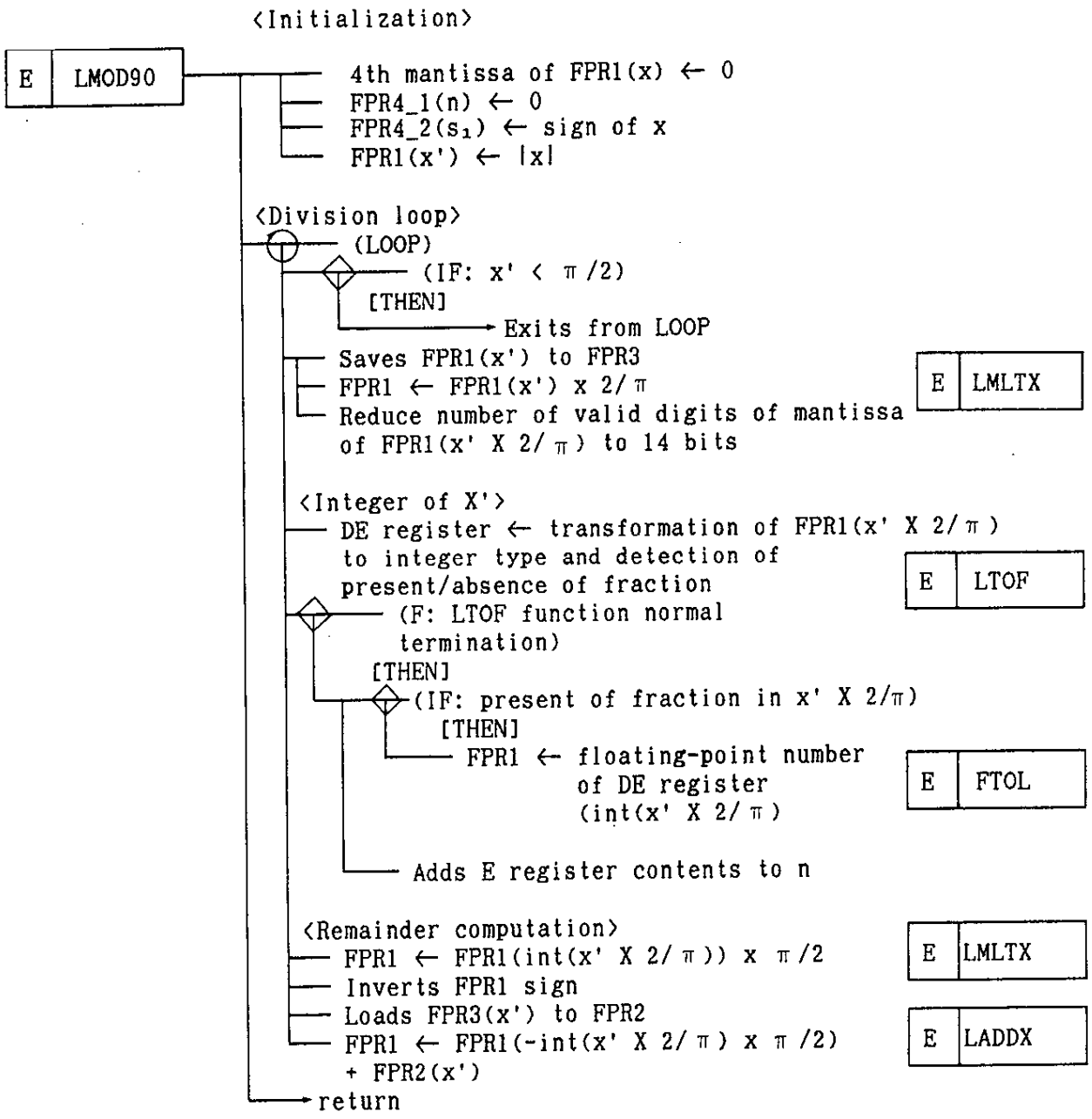
(a) Constant data $2/\pi$ and $\pi/2$ having an extended mantissa are used for scaling of $0-\pi/2$.

(b) Constant data $-1/3!$, $-3!/5!$, $-5!/7!$, $-7!/9!$, and $-9!/11!$ having an extended mantissa are used as a 5-element array K for the coefficient string of the Taylor approximation expression.

(10) Processing chart



(Subroutine computing quotient by $\pi/2$ and remainder)



Remarks: int(X) indicates the integer of X.

4.3 cos Function (LCOS)

(1) Processing

Takes the FPR1 value as x and returns cos(x) to FPR1.

- o Unit: radian

(2) Object module files to be linked

DFLT, LFLT1, LFLT2, LLD, LSIN, LCOS, FTOL, LTOF

(3) Consumed stack size

10 (including 2 bytes of return address from LCOS)

(4) Registers

FPR1, FPR2, FPR3, FPR4

(5) Processing time (internal system clock: 6 MHz)

Average: 2.67 ms

Maximum: 7.74 ms (cos(1.701412e + 38))

(6) Algorithm

SIN(X') is obtained by the following expression:

$$\cos(x) = \cos(|x|) = \sin(|x| + \pi/2)$$
$$x' = |x| + \pi/2$$

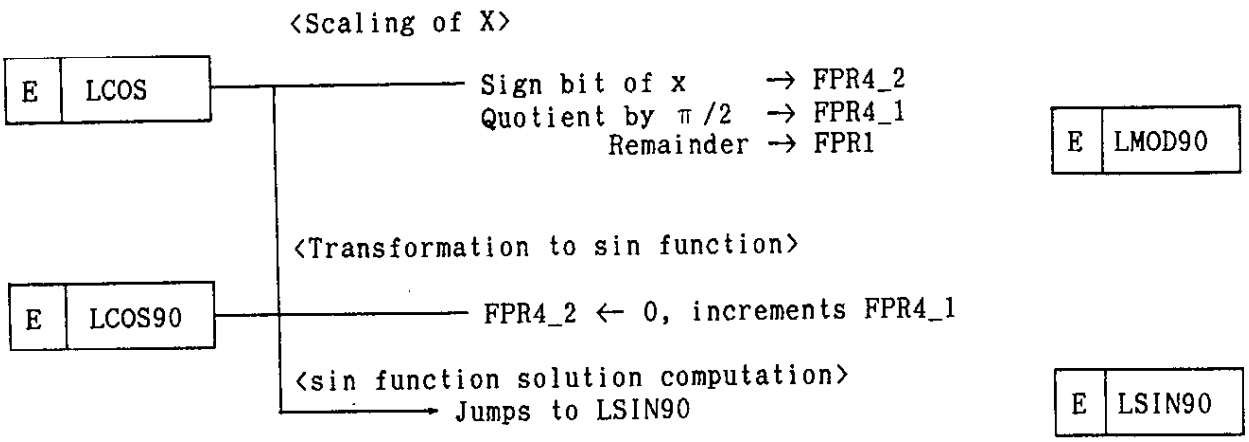
(7) Processing sequence

(a) The quotient by $\pi/2$ (FPR4_1) and remainder (FPR1) are obtained by the LMOD90 subroutine.

(b) Clears the sign (FPR4_2), saved by LMOD90, and adds 1 to FPR4_1.

(c) Jumps to LSIN90.

(8) Processing chart



4.4 tan Function (LTAN)

(1) Processing

Takes the FPR1 value as x and returns tan(x) to FPR1.

o Unit: radian

(2) Object module files to be linked

DFLT, LFLT1, LFLT2, LLD, LSIN, LCOS, LTAN, FTOL, LTOF

(3) Consumed stack size

14 (including 2 bytes of return address from LTAN)

(4) Registers

FPR1, FPR2, FPR3, FPR4, FPR5

(5) Processing time (internal system clock: 6 MHz)

Average: 5.54 ms

Maximum: 10.29 ms (tan(1.701412e + 38))

(6) Algorithm

The following expression is used:

$$TAN(x) = \frac{SIN(x)}{COS(x)}$$

(7) Processing sequence

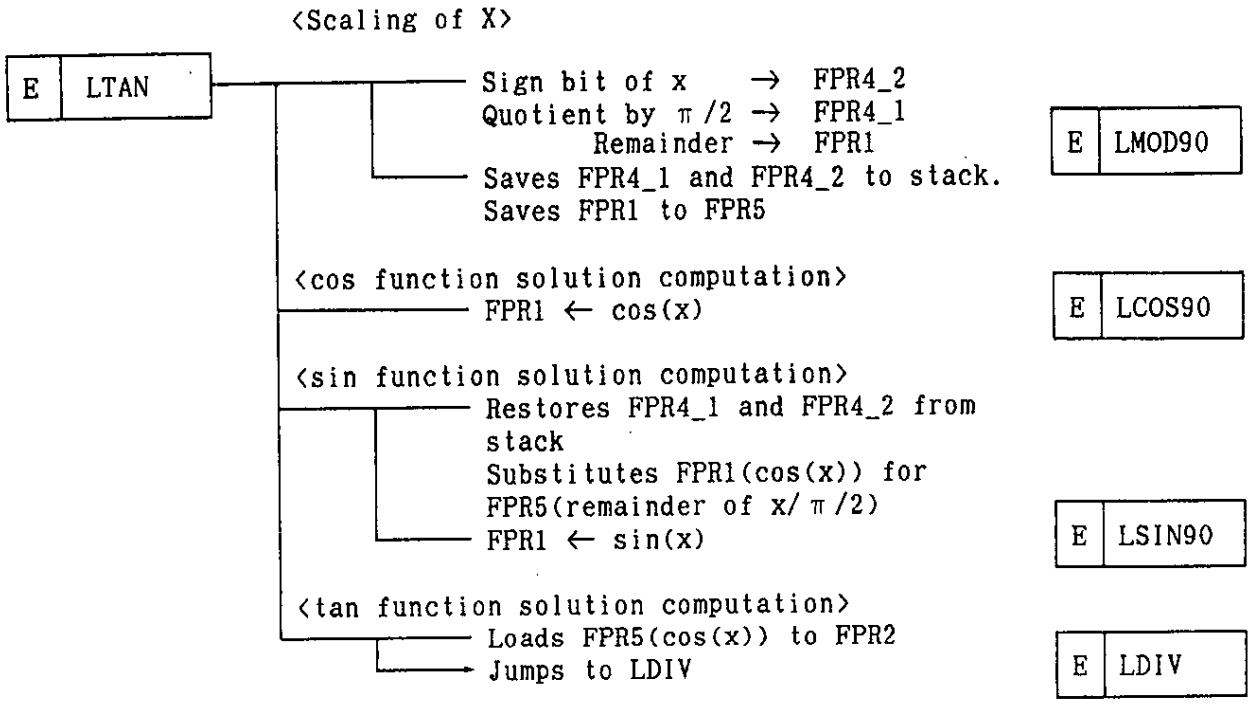
(a) The quotient by $\pi/2$ (FPR4_1), remainder (FPR1), and the sign (FPR4_2 of x are obtained by the LMOD90 subroutine.

(b) Obtains cos(x) by the LCOS90 subroutine.

(c) Obtains sin(x) by the LSIN90 subroutine.

(d) Solves sin(x)/cos(x).

(8) Processing chart



4.5 Natural Logarithm Function (LLOG)

(1) Processing

Takes the FPR1 value as x and returns log(x) to FPR1.

(2) Object module files to be linked

DFLT, LFLT1, LFLT2, LLD, LLOG, FTOL

(3) Consumed stack size

10 (including 2 bytes of return address from LLOG)

(4) Registers

FPR1, FPR2, FPR3, FPR4

(5) Processing time (internal system clock: 6 MHz)

Average: 3.08 ms

Maximum: 4.36 ms (log(3.27680039e + 10))

(6) Algorithm

The following expression is used:

o log(x) is as follows, where the exponent of X = X_e, and mantissa = X_f

$$\log(x) = X_e \times \log(2) + \log(X_f)$$

o $\log(X_f)$ is obtained as follows:

① $X' = \dfrac{X_f - 1}{X_f + 1}$

② This Taylor expansion approximation expression is used:

$\log(x_f) = \log(1 + X') - \log(1 - X')$

③ $\log(x) = X_e \times \log(2) + 2X' + \dfrac{2}{3}X'^3 + \dfrac{2}{5}X'^5 + \dfrac{2}{7}X'^7 + \dfrac{2}{9}X'^9 + \dfrac{2}{11}X'^{11} + \dfrac{2}{13}X'^{13}$

(a) If $x \leq 0$, returns an error.

(b) If x = 1, returns 0.

(c) Obtains X_e x log(2).

(d) Obtains X' from $(X_f - 1)/X_f + 1)$.

(e) Obtains log(x) from a Taylor approximation expression.

Calls a polynomial expression computation function with the sum of the logarithm of the exponent and the first term (X_e x log(2) + 2X'), the initial value of the second term 2X', the ratio except for the coefficient of each term (X'2), and the ratios to the coefficient of the preceding term for the second to seventh terms (1/3, 3/5, 5/7, 7/9, 9/11, 11/13) as arguments.

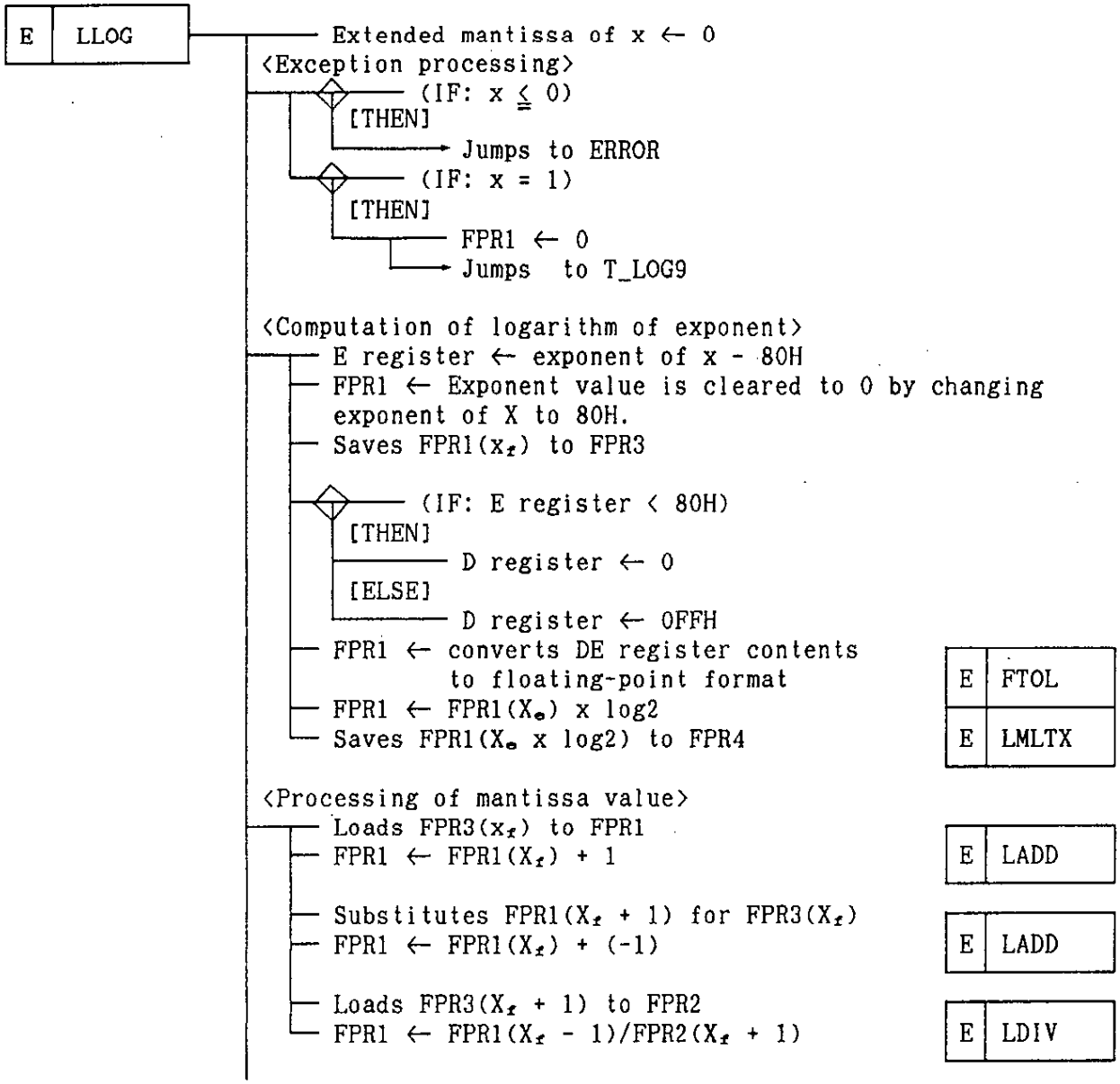
(7) Floating-point constant data

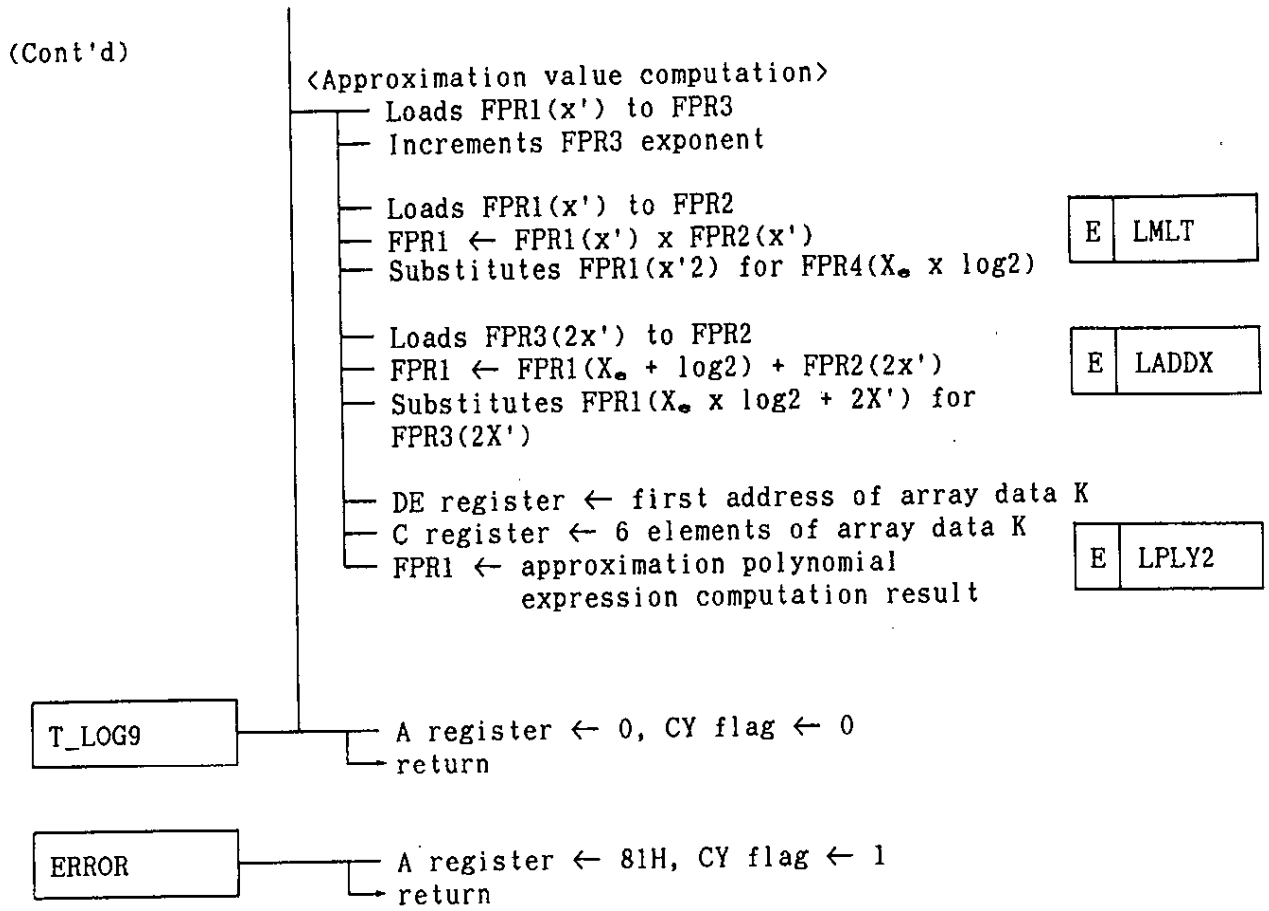
(a) Constant data 1 is used to judge if x = 1, and for the computation of X'.

(b) Constant data log(2), having an extended mantissa, is used for computing the logarithm of the exponent.

(c) Defines floating-point constant data 1/3, 3/5, 5/7, 7/9, 9/11, and 11/13, having an extended mantissa, as a 6-element array K for the coefficient string of the polynomial expression.

(8) Processing chart




(Cont'd)
<Approximation value computation>
Loads FPR1(x') to FPR3
Increments FPR3 exponent
Loads FPR1(x') to FPR2
FPR1 ← FPR1(x') x FPR2(x')
Substitutes FPR1(x'2) for FPR4(X₀ x log2)
E
LMLT
Loads FPR3(2x') to FPR2
FPR1 ← FPR1(X₀ + log2) + FPR2(2x')
Substitutes FPR1(X₀ x log2 + 2X') for FPR3(2X')
E
LADDX
DE register ← first address of array data K
C register ← 6 elements of array data K
FPR1 ← approximation polynomial expression computation result
E
LPLY2
T_LOG9
A register ← 0, CY flag ← 0
return
ERROR
A register ← 81H, CY flag ← 1
return

4.6 Common Logarithm Function (LLOG10)

(1) Processing

Takes the FPR1 value as x and returns log10(x) to FPR1.

(2) Object module files to be linked

DFLT, LFLT1, LFLT2, LLD, LLOG, LLOG10, FTOL

(3) Consumed stack size

12 (including 2 bytes of return address from LLOG10)

(4) Registers

FPR1, FPR2, FPR3, FPR4

(5) Processing time (internal system clock: 6 MHz)

Average: 3.24 ms

Maximum: 4.47 ms (log10(3.27680039e + 04))

(6) Algorithm

The following expression is used:

$$\log10(x) = \frac{\log(x)}{\log(10)}$$

(7) Processing sequence

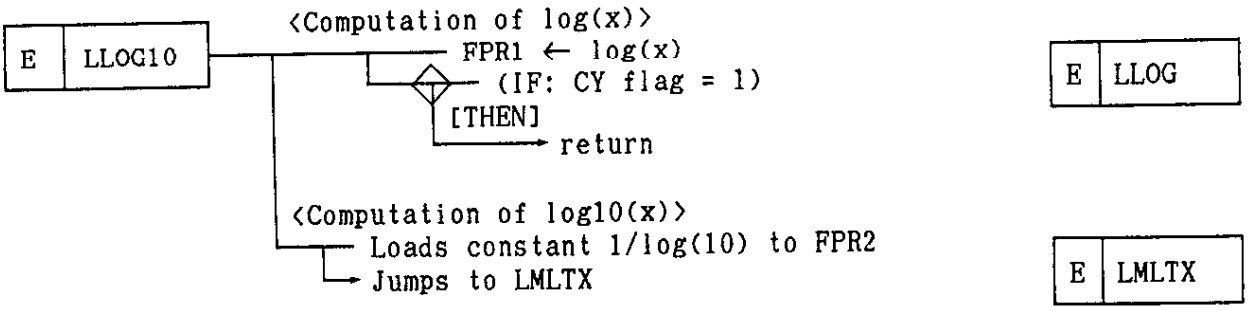
(a) Obtains log(x) by the LLOG function.

(b) If the LLOG function results in an error, returns the error as is.

(8) Floating-point constant data

Constant data 1/log(10) having an extended mantissa is used.

(9) Processing chart



4.7 Exponent Function (base = e) (LEXP)

(1) Processing

Takes the FPR1 value as x and returns e^x to FPR1.

(2) Object module files to be linked

DFLT, LFLT1, LFLT2, LLD, LEXP, FTOL, LTOF

(3) Consumed stack size

10 (including 2 bytes of return address from LEXP)

(4) Registers

FPR1, FPR2, FPR3, FPR4, FPR5

(5) Processing time (internal system clock: 6 MHz)

Average: 3.43 ms

Maximum: 4.54 ms (exp(-5.67776156))

(6) Algorithm

The exponent of the solution is first obtained by converting the exponent base to 2. The following expression is used:

$$e^x = 2^{x/\log 2} = 2^{\mathrm{int}(x/\log 2)} \times 2^{\mathrm{dec}(x/\log 2)}$$

Remarks: 1. int(X) indicates the higher integer of X (int(X) - 1 $\leqq$ X < int(X)).

2. dec(x) indicates the fraction of X (dec(X) = X - int(X); -1 $\leqq$ dec(X) < 0).

Since $2^{\mathrm{dec}(x/\log 2)}$ is in the 0.5 to 1 range, the solution for int(x/log2) is the exponent itself.

The mantissa is obtained by the following Taylor approximation expression, where $2^{\mathrm{dec}(x/\log 2)+1}$ (the range of 0 to 1, which is the most advantageous for the following expression is used, because the mantissa that is obtained,

even if multiplied by 2, is the same):

Where X' = dec (x/log2) + 1,

$$2X' = 1 + \frac{\log 2}{1!}X' + \frac{(\log 2)^2}{2!}X'^2 + \frac{(\log 2)^3}{3!}X'^3 + \ldots + \frac{(\log 2)^9}{9!}X'^9$$

Although 2X' is in the 1 to 2 range, it may become less than 1, because the rounding off is executed toward zero. In this case, int(X/log2) - 1 is assumed as the exponent of the solution.

(7) Processing sequence

(a) Solves for x/log2

(b) If an overflow occurs:

- o Returns 0 as the solution, if x < 0.
- o Returns an error as the solution, if x > 0.

(c) Solves for int(x/log2)

(d) Returns 0 as the solution, if int(x/log2) < -7FH.
Returns an error as the solution, if int (x/log2) > 7FH.

(e) Solves for dec(x/log2)+1 and takes the result as X'.

(f) Solves for $2^{x'}$ with a Taylor approximation expression. Calls a polynomial expression computation function with the sum up to the second term of the approximation polynomial expression (1 + log2 X'), log2 X' as the initial computation value of the third term, ratio except for the coefficient of each term (X'), and the ratios to the coefficient of the preceding term from the coefficients of the third to the tenth terms (log2/2, log2/3, log2/4, log2/5, log2/6, log2/7, log2/8, log2/9) as arguments.

(g) If $2^{x'} < 1$, subtracts 1 from the exponent (int(x/log2)) of the solution.

(h) Embeds the exponent value int(x/log2) into the result of the computation of the approximation expression, and takes it as the solution.

(8) Work area

(a) Bit 7 of FPR3_1 is used to save the sign of x (this is expressed as symbol S in the processing chart).

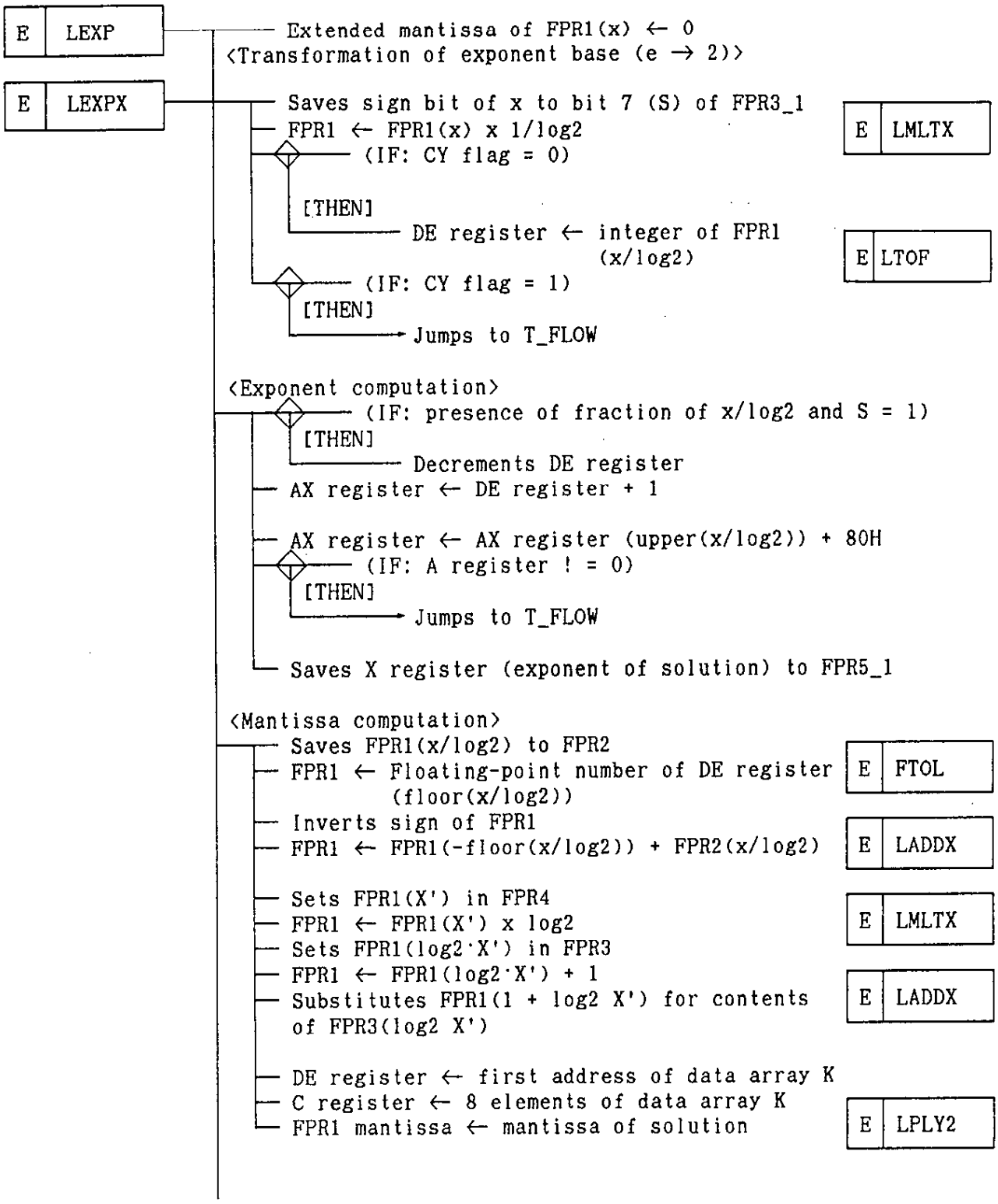
(b) FPR5_1 is used as an area to save the exponent value int(x/log2).

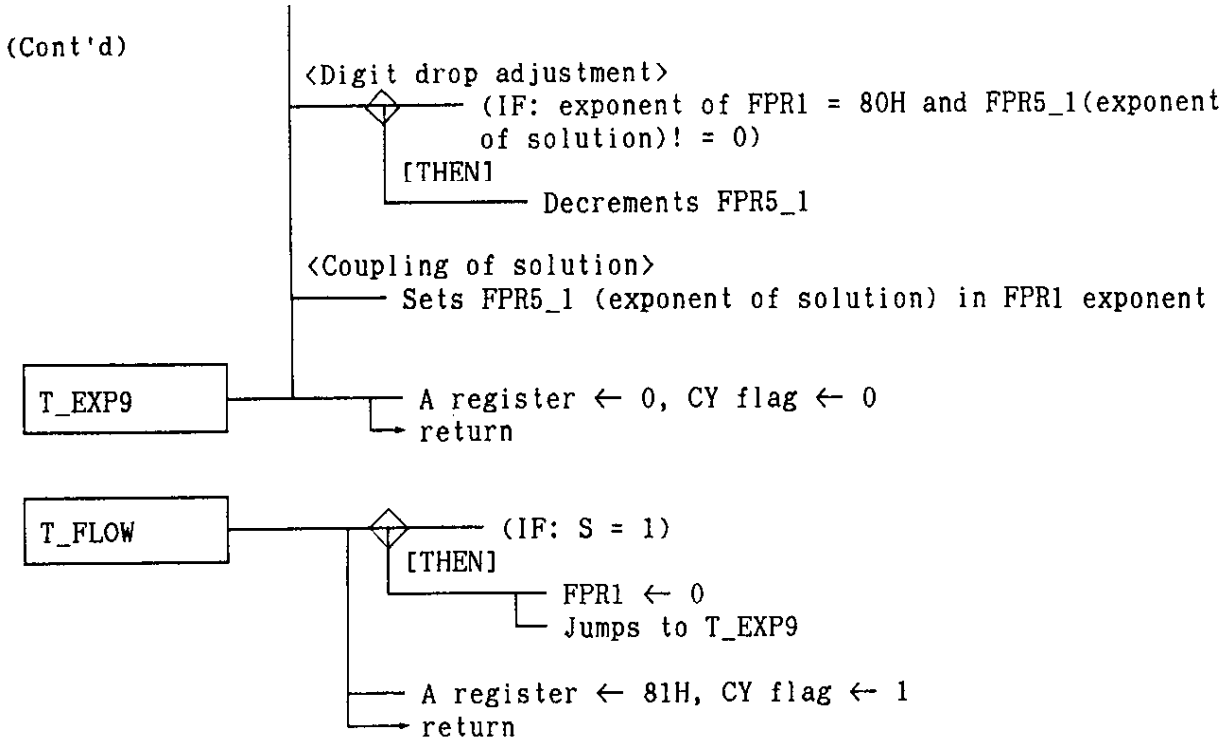
(9) Floating-point constant data

(a) Constant data having an extended mantissa 1/log2, log2, and 1 are used.

(b) Constant data having an extended mantissa log2/2, log2/3, log2/4, log2/5, log2/6, log2/7, log2/8, and log2/9 is used as an 8-element array K for the coefficient string of the approximation polynomial expression.

(10) Processing chart





Remarks: 1. upper(X) indicates the higher integer of X (upper(X) - 1 $\leq$ X < upper(X)).

2. floor(X) indicates the lower integer of X (floor(X) $\leq$ X < floor(X) + 1).

3. Label | E | LEXPX | is the name of an internal area for executing exponent computation using the extended mantissa with the other mathematical functions.

4.8 Exponent Function (base = 10) (LEXP10)

(1) Processing

Takes the FPR1 value as x and returns 10^x to FPR1.

(2) Object module files to be linked

DFLT, LFLT1, LFLT2, LLD, LEXP, LEXP10, FTOL, LTOF

(3) Consumed stack size

10 (including 2 bytes of return address from LEXP10)

(4) Registers

FPR1, FPR2, FPR3, FPR4, FPR5

(5) Processing time (internal system clock: 6 MHz)

Average: 3.53 ms

Maximum: 4.76 ms (exp10(-0.979693294))

(6) Algorithm

The following expression is used:

$$10^x = e^{x \times \log(10)}$$

(7) Processing sequence

(a) Solves for x X log(10).

(b) If the result of the multiplication in (a) is an error, returns the error as is.

(c) Jumps to the LEXP function.

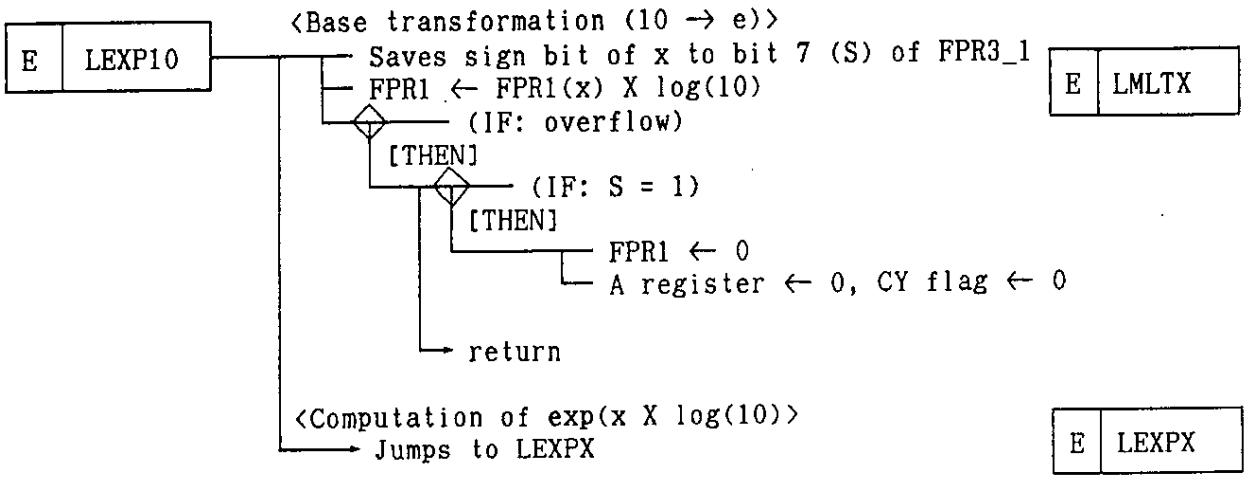
(8) Work area

FPR3_1 is used as an area to save the sign bit of x (this is expressed as symbol S in the processing chart).

(9) Floating-point constant data

Constant data log(10), having an extended mantissa, is used to convert the exponent base.

(10) Processing chart



4.9 Power Function (LPOW)

(1) Processing

Takes the FPR1 value as a and the FPR2 value as b, and returns a^b to FPR1.

(2) Object module files to be linked

DFLT, LFLT1, LFLT2, LLD, LLOG, LEXP, LPOW, FTOL, LTOF

(3) Consumed stack size

14 (including 2 bytes of return address from LPOW)

(4) Registers

FPR1, FPR2, FPR3, FPR4, FPR5

(5) Processing time (internal system clock: 6 MHz)

Average: 6.50 ms

Maximum: 8.72 ms (-0.5^-3)

(6) Algorithm

The method of computation differs, depending on the combination of values a and b, as follows:

a, b	a^b
$a = 0, b \leq 0$	Error
$a = 0, b > 0$	0
$a > 0$	$e^{b\log(a)}$
$a < 0$, b = 0	1
$a < 0$, b is integer other than 0. b is even:	$e^{b\log(\|a\|)}$
b is odd :	$-e^{b\log(\|a\|)}$
$a < 0$, b is not integer	Error

(7) Processing sequence

(a) If a = 0 and $b \leq 0$, returns an error as an operation

result.

(b) If a = 0 and b > 0, returns 0 as an operation result.

(c) If a < 0 and b = 0, returns 1 as an operation result.

(d) If a < 0 and b ≠ 0, judges whether b is an integer: if it is an integer, judges whether b is an odd or even number. If b is not an integer, returns an error as an operation result.

(e) Solves for $e^{b \log(|a|)}$ with the LLOG and LEXP functions.

(f) If a is negative and b is an odd integer, sets the sign bit.

(8) Work area

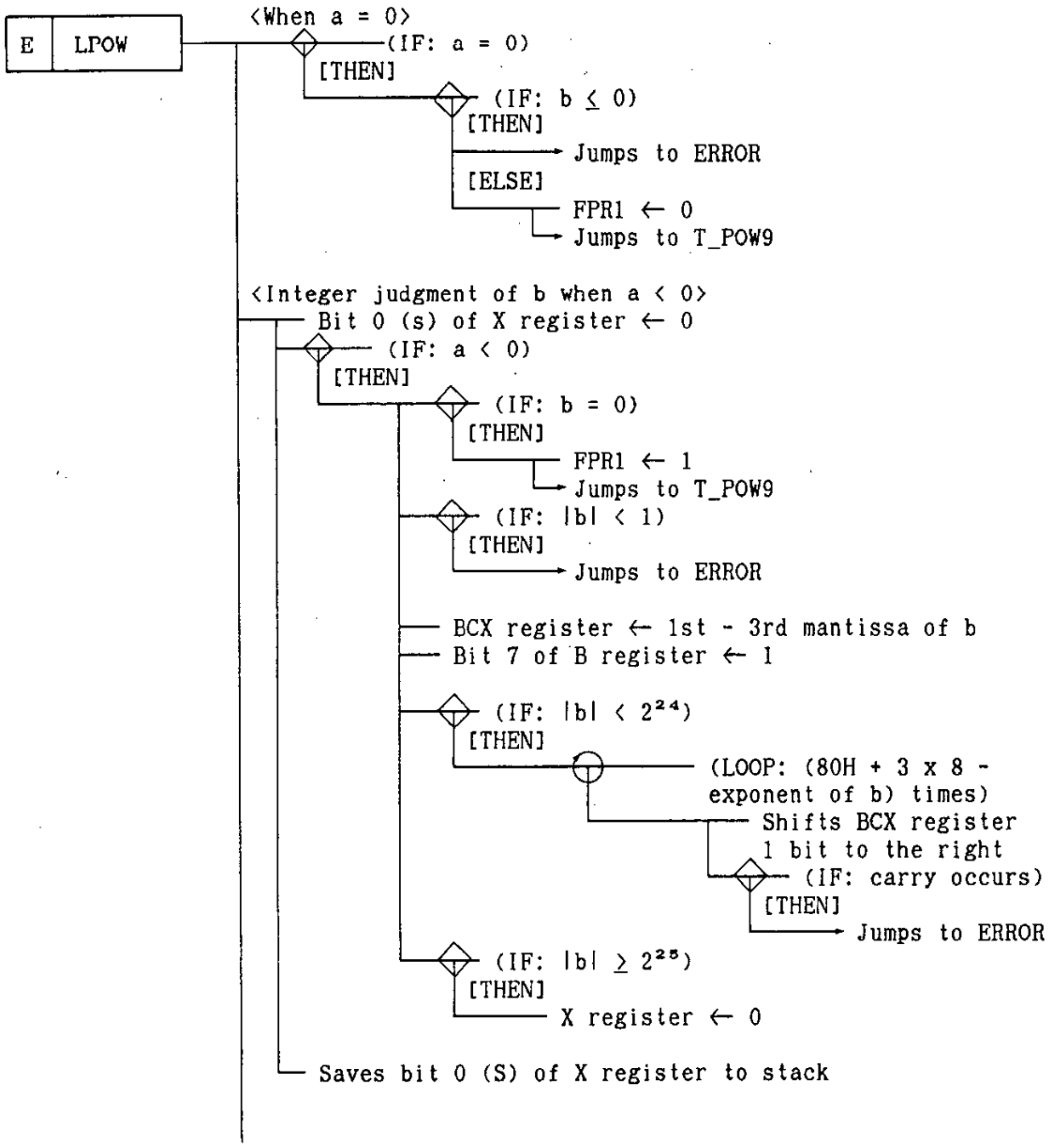
(a) Bit 0 of X register is used as the work area for the sign (this is expressed as the variable name s in the processing chart).

(b) The BCX register is used as a work area for processing the mantissa of b for judging whether b is an integer or not.

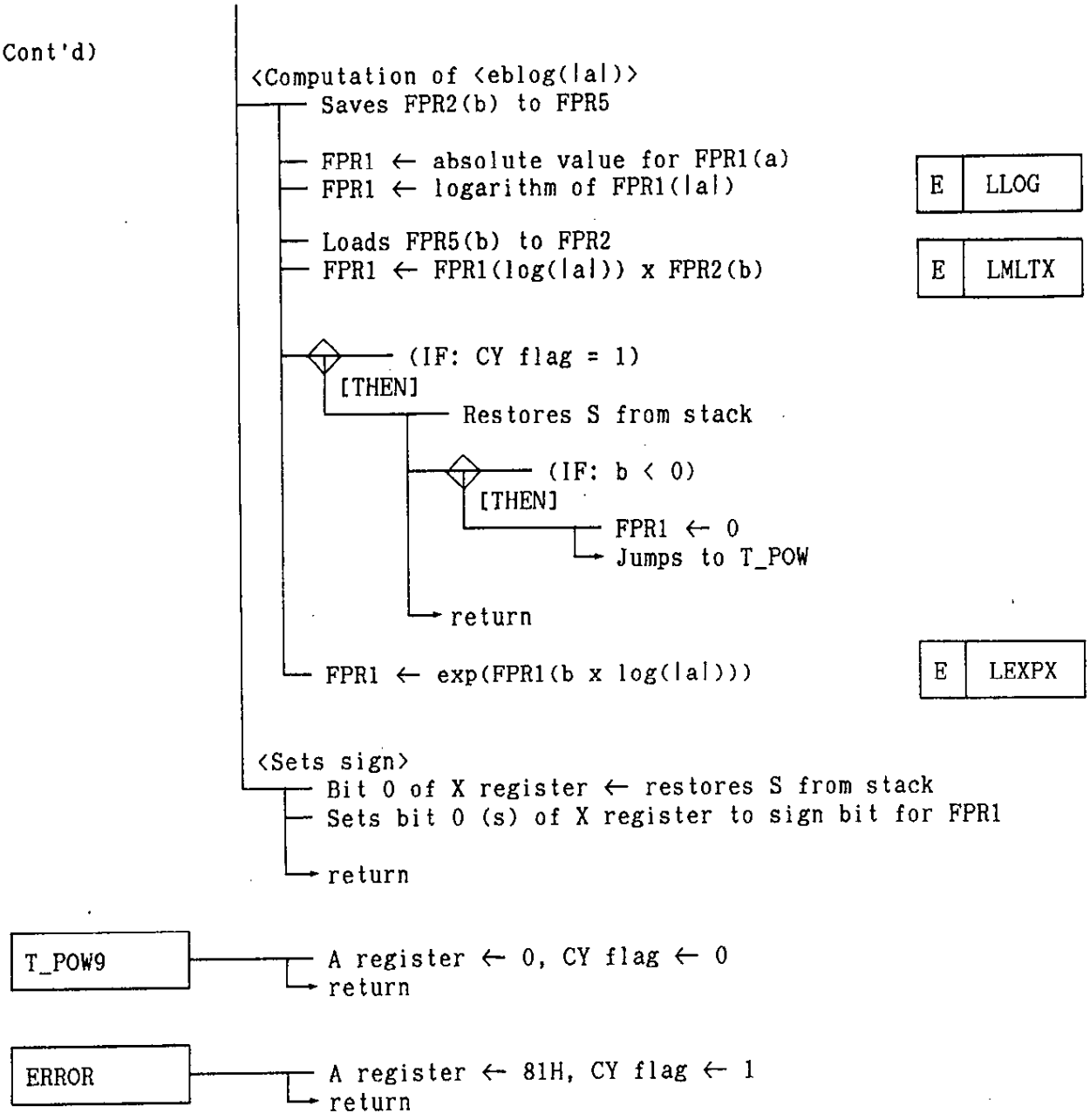
(9) Floating-point data

Constant data 1 is used as a solution, if a < 0 and b = 0.

(10) Processing chart



(Cont'd)


<Computation of <eblog(|a|)>
Saves FPR2(b) to FPR5
FPR1 ← absolute value for FPR1(a)
FPR1 ← logarithm of FPR1(|a|)
E
LLOG
Loads FPR5(b) to FPR2
FPR1 ← FPR1(log(|a|)) x FPR2(b)
E
LMLTX
(IF: CY flag = 1)
[THEN]
Restores S from stack
(IF: b < 0)
[THEN]
FPR1 ← 0
Jumps to T_POW
return
FPR1 ← exp(FPR1(b x log(|a|)))
E
LEXPX
<Sets sign>
Bit 0 of X register ← restores S from stack
Sets bit 0 (s) of X register to sign bit for FPR1
return
T_POW9
A register ← 0, CY flag ← 0
return
ERROR
A register ← 81H, CY flag ← 1
return

4.10 Square Root Function (LSQRT)

(1) Processing

Takes the FPR1 value as a and returns the square root of a to FPR1.

(2) Object module files to be linked

DFLT, LFLT1, LLD, LSQRT

(3) Consumed stack size

4 (including 2 bytes of return address from LSQRT)

(4) Registers

FPR1, FPR2, FPR3, FPR4

(5) Processing time (internal system clock: 6 MHz)

Average: 2.56 ms

Maximum: 2.76 ms ($\sqrt{(6.30915472e + 36)}$)

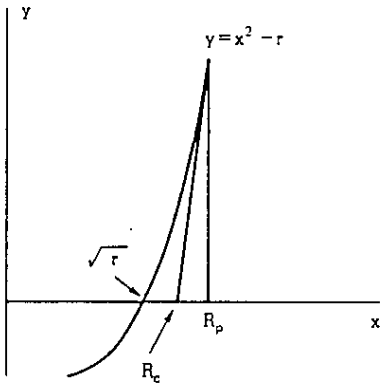
(6) Algorithm

$$\sqrt{a} = \sqrt{(r \times 2^{2n})} = 2^n \times \sqrt{r} \quad (0.25 \leq r < 1)$$

By using the above expression, the exponent (2^n) and mantissa ($\sqrt{r}$) are separated from the beginning.

To compute the square root of r, the Newton-Raphson method is used.

The square root of r is at coordinates x, at the point where the second degree function $y = x^2 - r$ intersects with the X axis, as shown below.



Taking R_p as the upper approximate value of the square root of r, a tangent of $y = x^2 - r$ is drawn from the intersection of straight line $x = R_p$ and $y = x^2 - r$.
Where coordinate x, at the point where the tangent and the X axis intersect with each other, is R_c, R_c is an approximate value of the square root of r closer than R_p. R_c can be obtained from R_p by the following expression:

$$R_c = \frac{R_p}{2} + \frac{r}{2R_p}$$

This function solves for an approximate value, with R_6 as the maximum value, from $\sqrt{r} < 1$, where the initial approximate value $R_1 = 1$.

(7) Processing sequence

(a) If $a < 0$, returns an error. If $a = 0$, returns 0 as an operation result.

(b) Solves for $(ae + 1)/2$ for the exponent ae of a and takes the root of the exponent as n.

(c) Adjusts the exponent of a for n, and takes r (=a/n).

(d) Solves for the second-degree approximate value (R_2) $r/2 + 1/2$.

(e) Takes the second-degree approximate value as the first R_p, and computes up to the six-degree approximate value. However, if $R_c \geqq R_p$ in the midpoint, takes the approximate value at that point as the solution.

(f) Lastly, embeds the root (n) of the exponent into the exponent of the obtained approximate solution for the mantissa.

(8) Work area

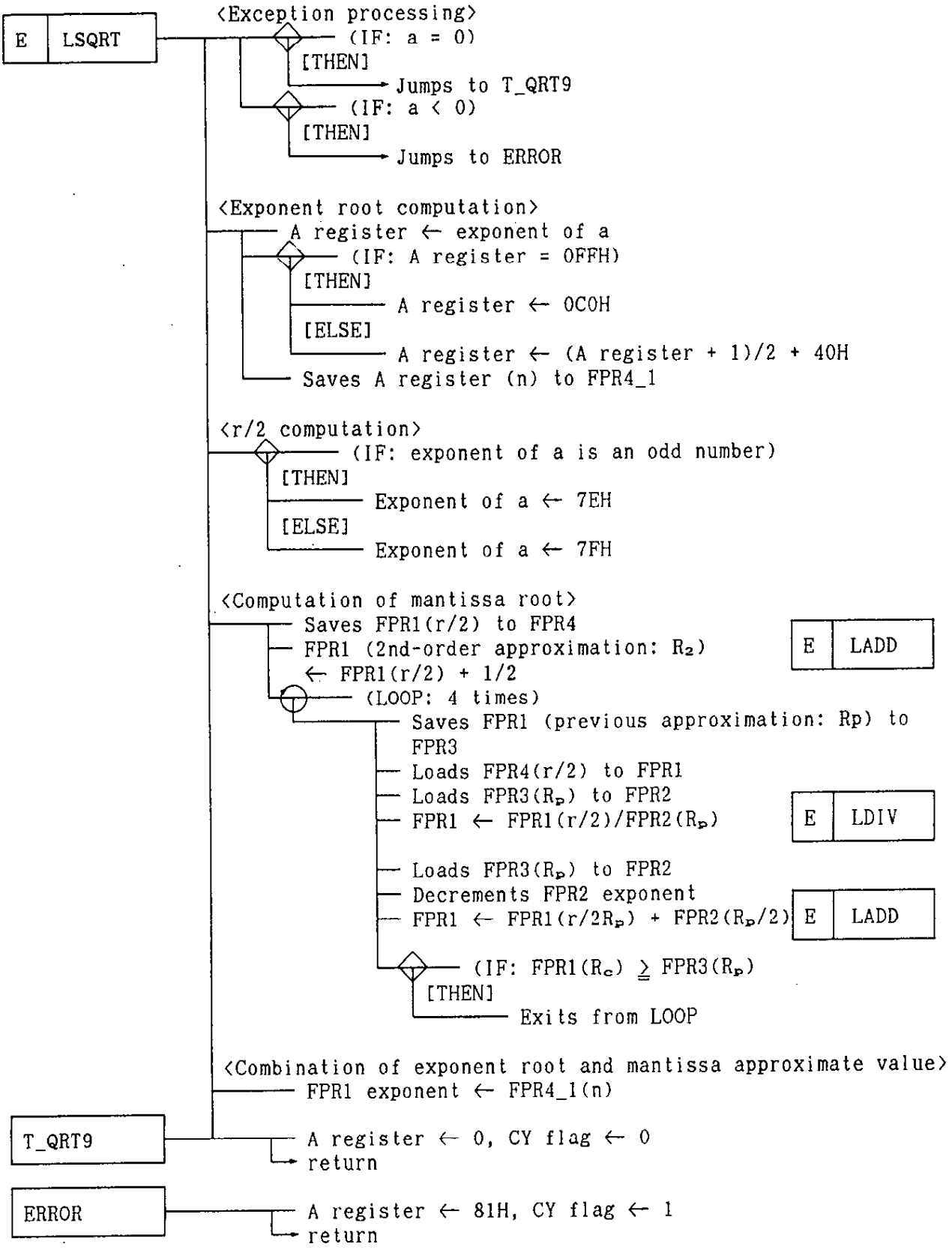
(a) FPR4_1 is used as an area to save the root (n) for the exponent.

(b) FPR3_1 is used as a loop counter for computing the approximate value of the mantissa.

(9) Floating-point constant data

Constant data 1/2 is used in the process of computing the first-degree approximate value.

(10) Processing chart



4.11 arcsin Function (LASIN)

(1) Processing

Takes the FPR1 value as x and returns ARCSIN(x) to FPR1.

- o Valid input value x range : -1 to 1
- o Return value range : $-\pi/2$ to $\pi/2$
- o Unit : Radian

(2) Object module files to be linked

DFLT, LFLT1, LFLT2, LLD, LSQRT, LASIN, LATAN

(3) Consumed stack size

11 (including 2 bytes of return address from LASIN)

(4) Registers

FPR1, FPR2, FPR3, FPR4, FPR5

(5) Processing time (internal system clock: 6 MHz)

Average: 4.41 ms

Maximum: 7.95 ms (arcsin(0.999999940))

(6) Algorithm

The inverse sine (arc sine) is solved by the following expression, by being converted into an inverse tangent:

$$\arcsin(x) = \arctan\left(X/\sqrt{(1 - x^2)}\right)$$

(7) Processing sequence

(a) If x = 0, returns 1 as an operation result.

(b) If x = 1, returns $\pi/2$ as the solution: if x = -1, returns $-\pi/2$.

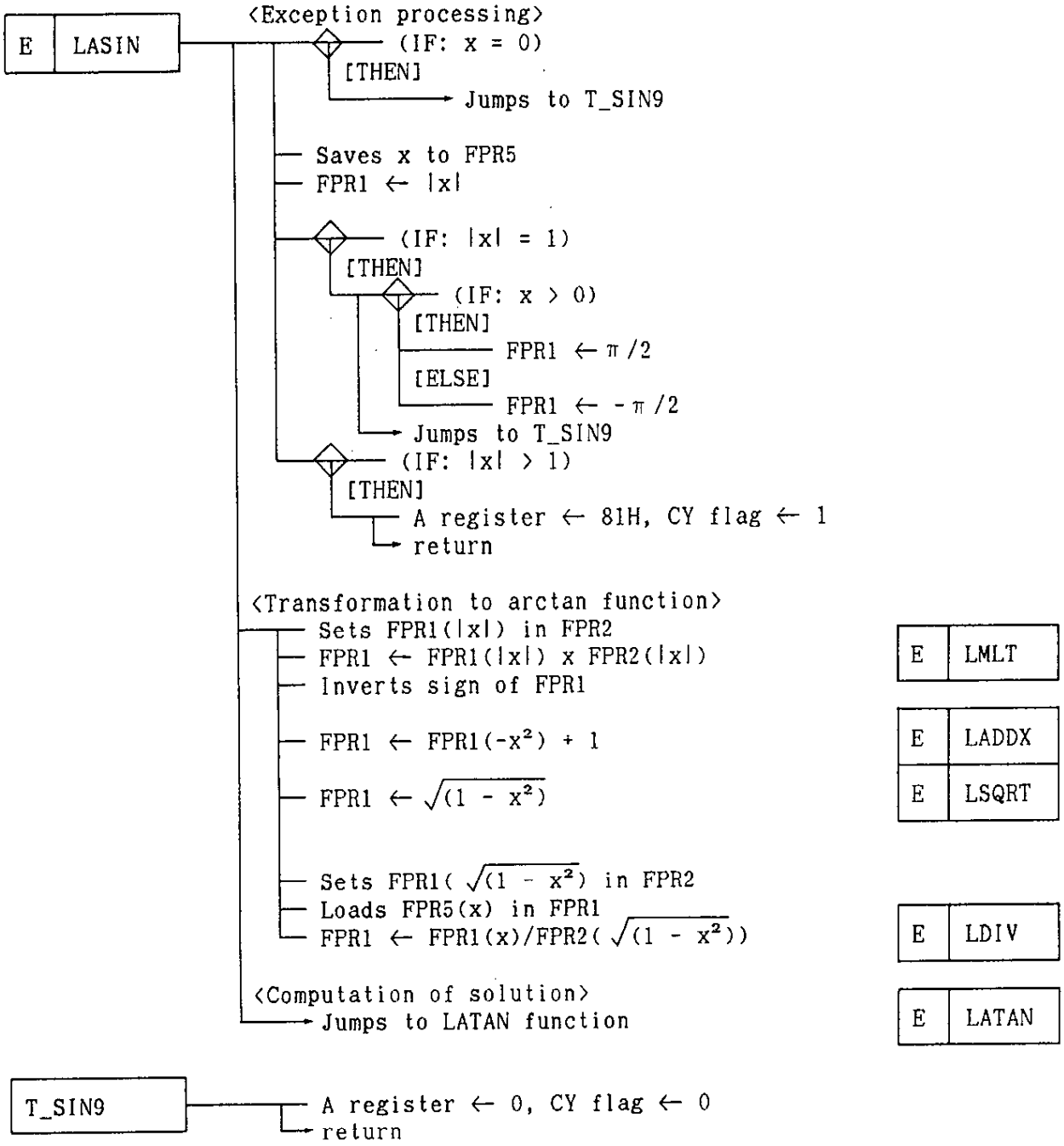
(c) If |x| > 1, returns an error as an operation result.

(d) Solves for $x/\sqrt{(1 - X^2)}$, and jumps to the LATAN function.

(8) Floating-point constant data

Constant data 1 and $\pi/2$ having an extended mantissa are used.

(9) Processing chart



4.12 arccos Function (LACOS)

(1) Processing

Takes the FPR1 value as x and returns ARCCOS(x) to FPR1.

- o Valid input value x range : -1 to 1
- o Return value range : 0 to π
- o Unit : Radian

(2) Object module files to be linked

DFLT, LFLT1, LFLT2, LLD, LSQRT, LASIN, LACOS, LATAN

(3) Consumed stack size

13 (including 2 bytes of return address from LACOS)

(4) Registers

FPR1, FPR2, FPR3, FPR4, FPR5

(5) Processing time (internal system clock: 6 MHz)

Average: 4.50 ms

Maximum: 8.16 ms (arccos(0.999999940))

(6) Algorithm

The following expression is used:

$$ARCCOS(x) = \pi/2 - ARCSIN(x)$$

(7) Processing sequence

(a) Solves for arcsin(x) by the LASIN function.

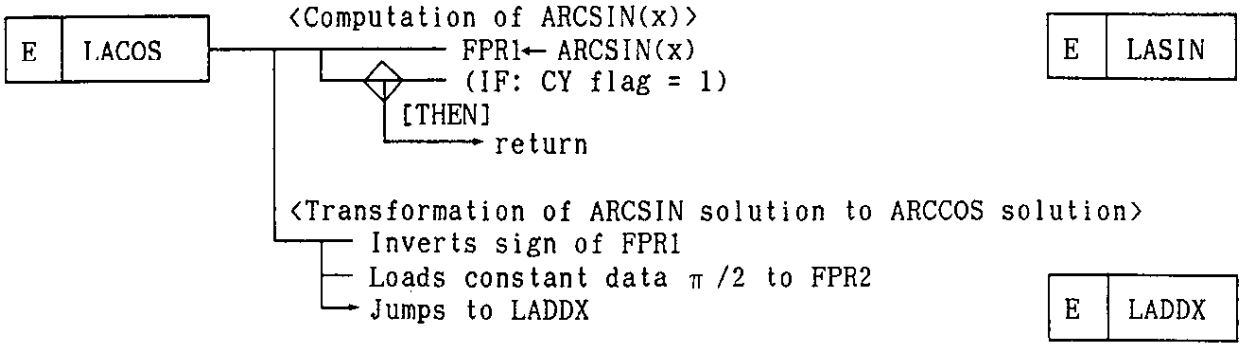
(b) If an error occurs as a result of the LASIN function, returns the error as is.

(c) Computes π/2 - ARCSIN(x) to obtain the solution.

(8) Floating-point constant data

Constant data π/2 having an extended mantissa are used.

(9) Processing chart



4.13 arctan Function (LATAN)

(1) Processing

Takes the FPR1 value as x and returns ARCTAN(x) to FPR1.

- o Return value range : $-\pi/2$ to $\pi/2$
- o Unit : Radian

(2) Object module files to be linked

DFLT, LFLT1, LFLT2, LLD, LATAN

(3) Consumed stack size

11 (including 2 bytes of return address from LATAN)

(4) Registers

FPR1, FPR2, FPR3, FPR4

(5) Processing time (internal system clock: 6 MHz)

Average: 3.33 ms

Maximum: 4.47 ms (arctan(4))

(6) Algorithm

The following Hastings best approximation expression is used:

$$\arctan(x) \fallingdotseq \sum_{i=0}^{n} (A_i \times x^{2i+1})$$

This function defines n = 7 and coefficient strings A_0 through A_7, as follows:

$A_0 = 0.9999993329$ $A_4 = 0.0964200441$

$A_1 = -0.3332985605$ $A_5 = -0.0559098861$

$A_2 = 0.1994653599$ $A_6 = 0.0218612288$

A_3 = -0.1390853351 A_7 = -0.0040540580

Note:

The Hastings best approximation expression can only be used in the range of $|x| \leq 1$. Therefore, if $|x| \geq 1$, the following expression is used:

x is substituted, where $x' = \dfrac{|x| - 1}{|x| + 1}$

$\arctan(|x|) = \pi/4 + \arctan(x')$

(7) Processing sequence

(a) If x = 0, returns 0 as an operation result.

(b) Saves the sign of x and the exponent and obtains the absolute value of x.

(c) If $|x| \geq 1$, converts X to $(|x| - 1)/(|x| + 1)$.

(d) Computes arctan(x') with the best approximation polynomial expression.

Calls the polynomial computation function with the first term of the approximate polynomial expression to be (A_0 X x'), x' as the initial value of the second term, the ratios except the coefficient of each term (x'2), and the ratio of the second term coefficient and the third through eighth term coefficient to the preceding coefficient (A_1, A_2/A_1, A_3/A_2, A_4/A_3, A_5/A_4, A_6/A_5, A_7/A_6) as arguments.

(e) If $|x| > 1$, adds $\pi/4$ to the solution of the approximation expression.

(f) Embeds the original sign of x into the operation result of arctan(|x|).

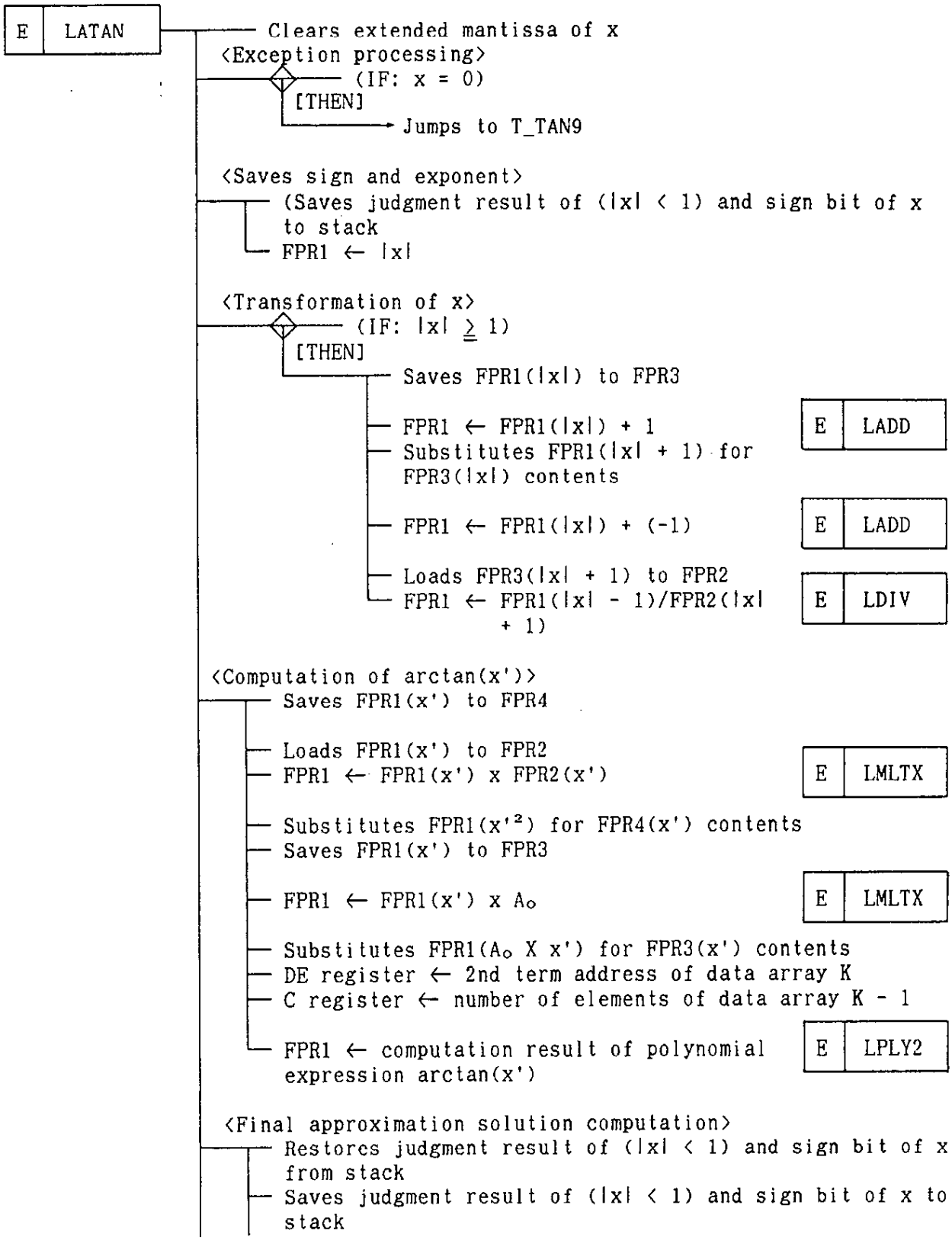
(8) Floating-point constant data

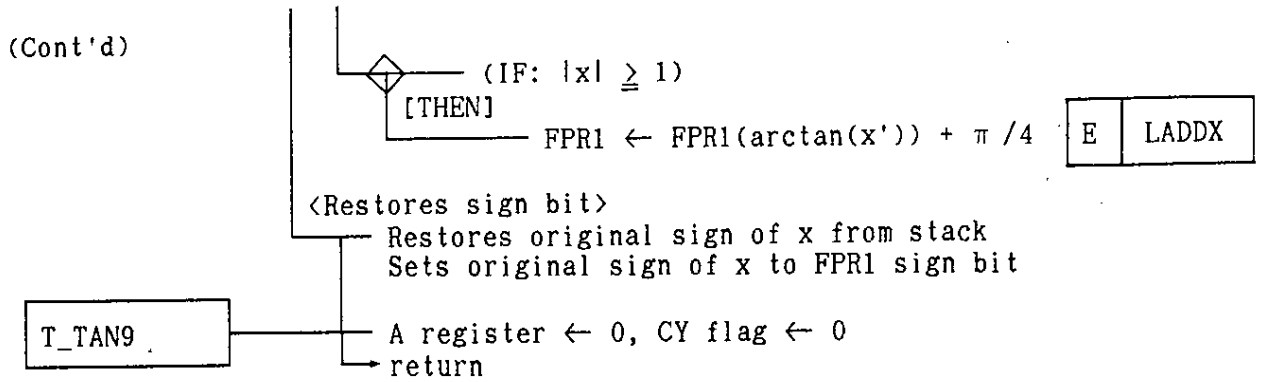
(a) Constant data 1 is used for transformation from x to x'.

(b) Constant data $\pi/4$ having an extended mantissa is used for transformation from arctan(x') to arctan(|x|).

(c) Constant data A_0, A_1, A_2/A_1, A_3/A_2, A_4/A_3, A_5/A_4, A_6/A_5, and A_7/A_6 having an extended mantissa are used as an 8-element array K for the coefficient string of the approximation polynomial expression.

(9) Processing chart




(Cont'd)
(IF: |x| ≧ 1)
[THEN]
FPR1 ← FPR1(arctan(x')) + π /4
E
LADDX
<Restores sign bit>
Restores original sign of x from stack
Sets original sign of x to FPR1 sign bit
T_TAN9
A register ← 0, CY flag ← 0
return

4.14 sinh Function (LHSIN)

(1) Processing

Takes the FPR1 value as x and returns SINH(x) to FPR1.

(2) Object module files to be linked

DFLT, LFLT1, LFLT2, LLD, LEXP, LHSIN, LRCPN, FTOL, LTOF

(3) Consumed stack size

14 (including 2 bytes of return address from LHSIN)

(4) Registers

FPR1, FPR2, FPR3, FPR4, FPR5

(5) Processing time (internal system clock: 6 MHz)

Average: 3.09 ms

Maximum: 5.03 ms (sinh(-86.6433868))

(6) Algorithm

o Where $|x| \geq 0.5$

The following expression is used:

$$SINH(x) = \frac{e^{x} - e^{-x}}{2}$$

o Where $|x| < 0.5$

The following Taylor expansion approximation expression is used:

$$SINH(x) = x + \frac{1}{3!}x^{3} + \frac{1}{5!}x^{5} + \frac{1}{7!}x^{7}$$

Remarks: That SINH(-x) = -SINH(x) is used.

(7) Processing sequence

o Where $|x| \geq 0.5$

(a) Stores the sign of x and obtains the absolute value of x.

(b) Solves for $e^{|x|}$ with the LEXP function.

(c) In case of $e^{|x}$ overflow, returns the error as is.

(d) Solves for $(e^{|x|} - 1/e^{|x|})/2$, embeds the original sign of x, and obtains the solution.

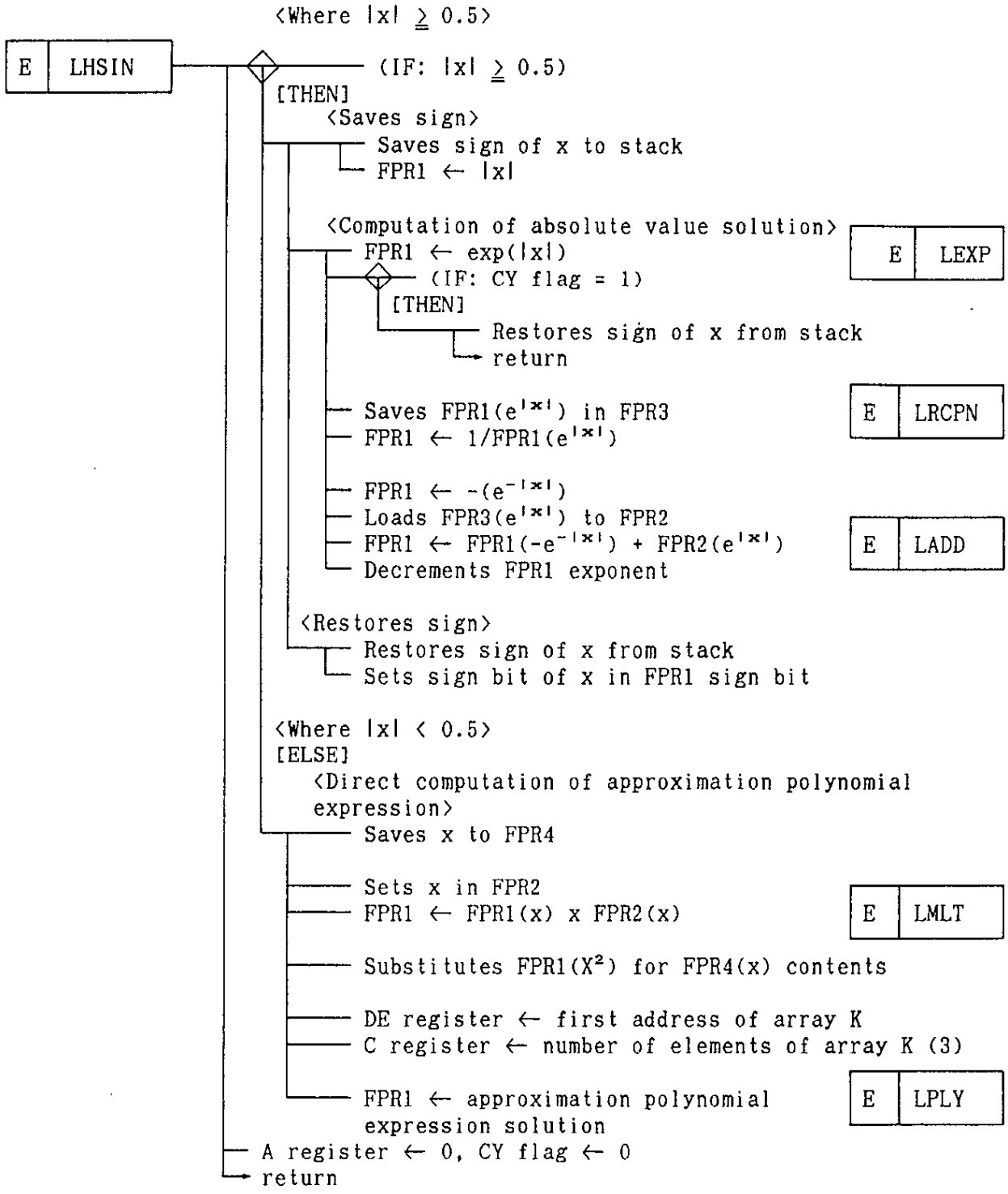
o Where $|x| < 0.5$

Solves for SINH(x) with a Taylor approximation expression. Calls the polynomial computation function with the first term of the approximation polynomial expression (x), the ratios, except for the coefficient of each term (x^2), and the ratio of the preceding coefficient to the coefficients of the second through the fourth terms (1/3!, 3!/5!, and 5!/7!) as arguments.

(8) Floating-point constant data

Constant data 1/3!, 3!/5!, and 5!/7! having an extended mantissa are used as a 3-element array K for the coefficient string of the Taylor approximation expression.

(9) Processing chart



4.15 cosh Function (LHCOS)

(1) Processing

Takes the FPR1 value as x and returns COSH(x) to FPR1.

(2) Object module files to be linked

DFLT, LFLT1, LFLT2, LLD, LEXP, LHCOS, LRCPN, FTOL, LTOF

(3) Consumed stack size

12 (including 2 bytes of return address from LHCOS)

(4) Registers

FPR1, FPR2, FPR3, FPR4, FPR5

(5) Processing time (internal system clock: 6 MHz)

Average: 3.56 ms

Maximum: 5.02 ms (cosh(86.6433868))

(6) Algorithm

The following expression is used:

$$COSH(x) = \frac{e^{x} + e^{-x}}{2}$$

Remarks: That COSH(-x) = COSH(x) is used.

(7) Processing sequence

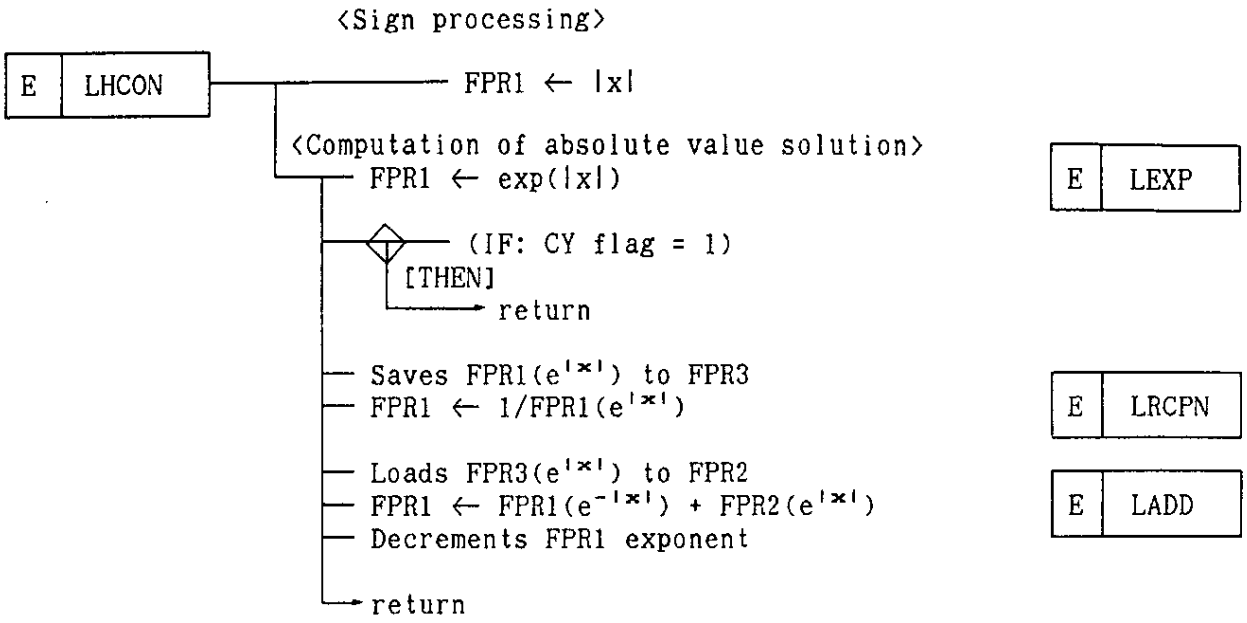
(a) Obtains the absolute value of X.

(b) Solves $e^{|x|}$ with the LEXP function.

(c) In case of $e^{|x|}$) overflow, returns the error as is.

(d) Solves for $(e^{|x|} + 1/e^{|x|})/2$ to obtain the solution.

(8) Processing chart



4.16 tanh Function (LHTAN)

(1) Processing

Takes the FPR1 value as x and returns TANH(x) to FPR1.

(2) Object module files to be linked

DFLT, LFLT1, LFLT2, LLD, LEXP, LHSIN, LHCOS, LHTAN, LRCPN, FTOL, LTOF

(3) Consumed stack size

16 (including 2 bytes of return address from LHTAN)

(4) Registers

FPR1, FPR2, FPR3, FPR4, FPR5

(5) Processing time (internal system clock: 6 MHz)

Average: 6.26 ms

Maximum: 10.53 ms (tanh(86.6433868))

(6) Algorithm

The following expression is used:

$$TANH(x) = \frac{SINH(x)}{COSH(x)}$$

(7) Processing sequence

(a) Solves for COSH(x) with the LHCOS function.

(b) In case of overflow

- Returns 1 as the solution, if the original sign of x is positive.
- Returns -1 as the solution, if the original sign of x is negative.

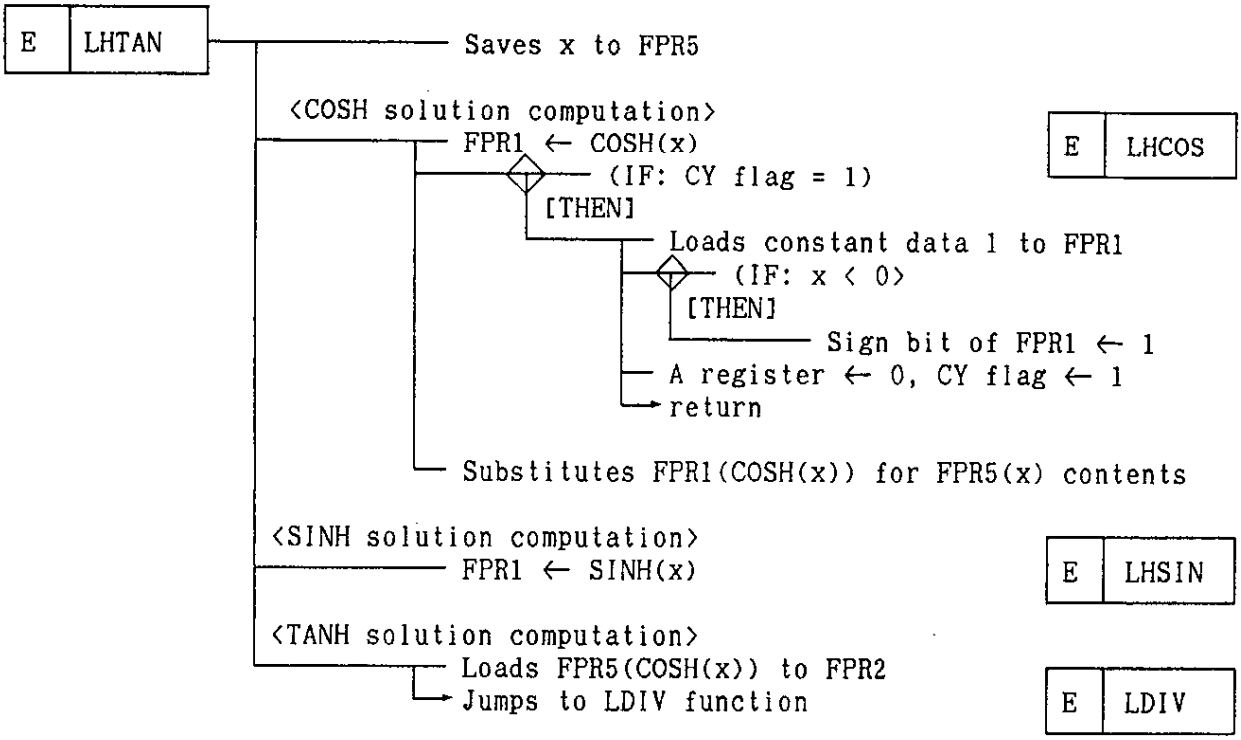
(c) Solves for SINH(x) with the LHSIN function.

(d) Takes SINH(x)/COSH(x) as the solution.

(8) Floating-point constant data

Constant data 1 is used as the solution, in case of an overflow.

(9) Processing chart



4.17 Absolute Value Function (LABS)

(1) Processing

Obtains the FPR1 absolute value and returns it to FPR1.

(2) Object module files to be linked

DFLT, LABS

(3) Consumed stack size

2 (only 2 bytes of return address from LABS)

(4) Registers

FPR1

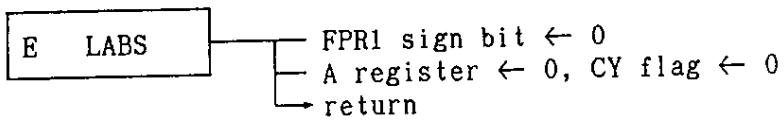
(5) Processing time (internal system clock: 6 MHz)

7.1 us

(6) Processing sequence

Clears the FPR1 sign bit to 0.

(7) Processing chart



4.18 Inverse Number Function (LRCPN)

(1) Processing

Obtains the inverse number for the FPR1 value and returns it to FPR1.

(2) Object module files to be linked

DFLT, LFLT1, LLD, LRCPN

(3) Consumed stack size

4 (including 2 bytes of return address from LRCPN)

(4) Registers

FPR1, FPR2

(5) Processing time (internal system clock: 6 MHz)

Average: 534 us

Maximum: 581 us (1/6.62484413e + 26)

(6) Processing sequence

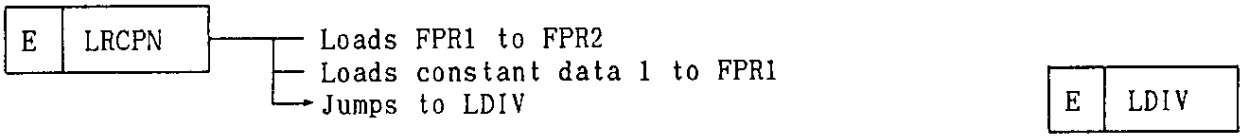
(a) Transfers the FPR1 value to FPR2 and sets constant 1 in FPR1.

(B) Jumps to the LDIV function.

(7) Floating-point constant data

Constant data 1 is used.

(8) Processing chart



CHAPTER 5 COORDINATE TRANSFORMATION FUNCTIONS

The following coordinate transformation functions are available:

(1) Function for transforming polar coordinates to rectangular coordinates (POTORA)

Transforms polar coordinates (r, θ) to rectangular coordinates (x, y).

The values r and x are given from FPR1, and θ and y, are given from FPR2.

(2) Function for transforming rectangular coordinates to polar coordinates (RATOPO)

Transforms rectangular coordinates (x, y) to polar coordinates (r, θ).

The values x and r are given from FPR1, and y and θ are given from FPR2.

5.1 Polar Coordinate-to-Rectangular Coordinate Transformation Function (POTORA)

(1) Processing

Taking the FPR1 value as r and the FPR2 value as θ, transforms polar coordinates (r, θ) to rectangular coordinates (x, y), and returns x to FPR1 and y to FPR2.

o Units of θ : Radian

(2) Object module files to be linked

DFLT, LFLT1, LFLT2, LLD, LSIN, LCOS, POTORA, FTOL, LTOF

(3) Consumed stack size

16 (including 2 bytes of return address from POTORA)

(4) Registers

FPR1, FPR2, FPR3, FPR4, FPR5

(5) Processing time (internal system clock: 6 MHz)

Average: 4.74 ms

Maximum: 10.12 ms (r = 0.5, θ = 1.70141173e + 38)

(6) Algorithm

The following expression is used:

$$x = r \times \cos(\theta),\ y = r \times \sin(\theta)$$

(7) Processing sequence

(a) If r = 0, returns coordinates (0, 0).

(b) If r < 0, returns an error.

(c) Transforms θ to $\theta = \theta' + n \times \pi/2$ (where n is an integer, $0 \leq \theta' < \pi/2$) with the LMOD90 function.

(d) Solves for $\sin(\theta)$ with the LSIN90 function and $\cos(\theta)$ with the LCOS90 function.

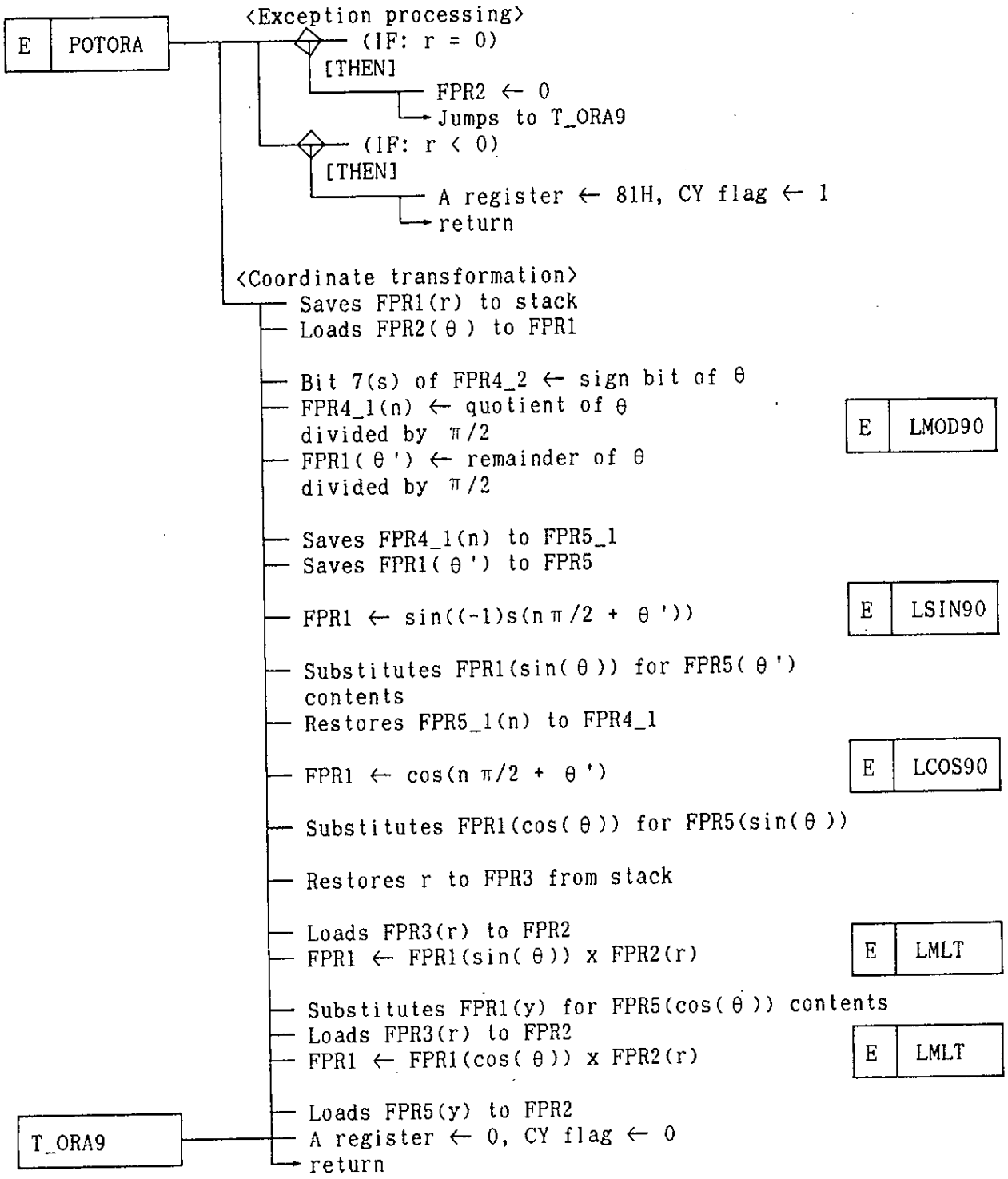
(e) Takes $r \times \cos(\theta)$ as x and $r \times \sin(\theta)$ as y.

(8) Work area

(a) FPR5_1 is used as an area to save quotient n resulting from dividing θ by π/2.

(b) r is saved to the stack.

(9) Processing chart



5.2 Rectangular Coordinate-to-Polar Coordinate Transformation Function (RATOPO)

(1) Processing

Taking the FPR1 value as x and the FPR2 value as y, transforms rectangular coordinates (x, y) to polar coordinates (r, θ), and returns r to FPR1 and θ to FPR2.

o Return value range : $-\pi$ to π

o Unit : Radian

(2) Object module files to be linked

DFLT, LFLT1, LFLT2, LLD, LSQRT, LATAN, RATOPO

(3) Consumed stack size

13 (including 2 bytes of return address from RATOPO)

(4) Registers

FPR1, FPR2, FPR3, FPR4, FPR5

(5) Processing time (internal system clock: 6 MHz)

Average: 5.33 ms

Maximum: 7.90 ms (x = -0.5, y = 1)

(6) Algorithm

The following two expressions are used:

$$r = \sqrt{(x^2 + y^2)}$$
$$\theta = \arctan(y/x)$$

(7) Processing sequence

(a) If x = y = 0, executes nothing, only returns.

(b) Solves for $x^2 + y^2$.

(c) If an overflow occurs as a result of $x^2 + y^2$, returns an error.

(d) Solves for y/x.

(e) If an overflow occurs as a result of y/x,

- where $y > 0$, $\theta = \pi/2$
- where $y < 0$, $\theta = -\pi/2$

(f) If y/x terminates normally, solves ARCTAN(y/x) with the LATAN function, and takes the result as θ .

(g) If $x < 0$ in (f),

- where $y \geq 0$, adds π to θ .
- where $y < 0$, subtracts π from θ .

(h) Solves for $\sqrt{(x^2 + y^2)}$, and takes the result as r.

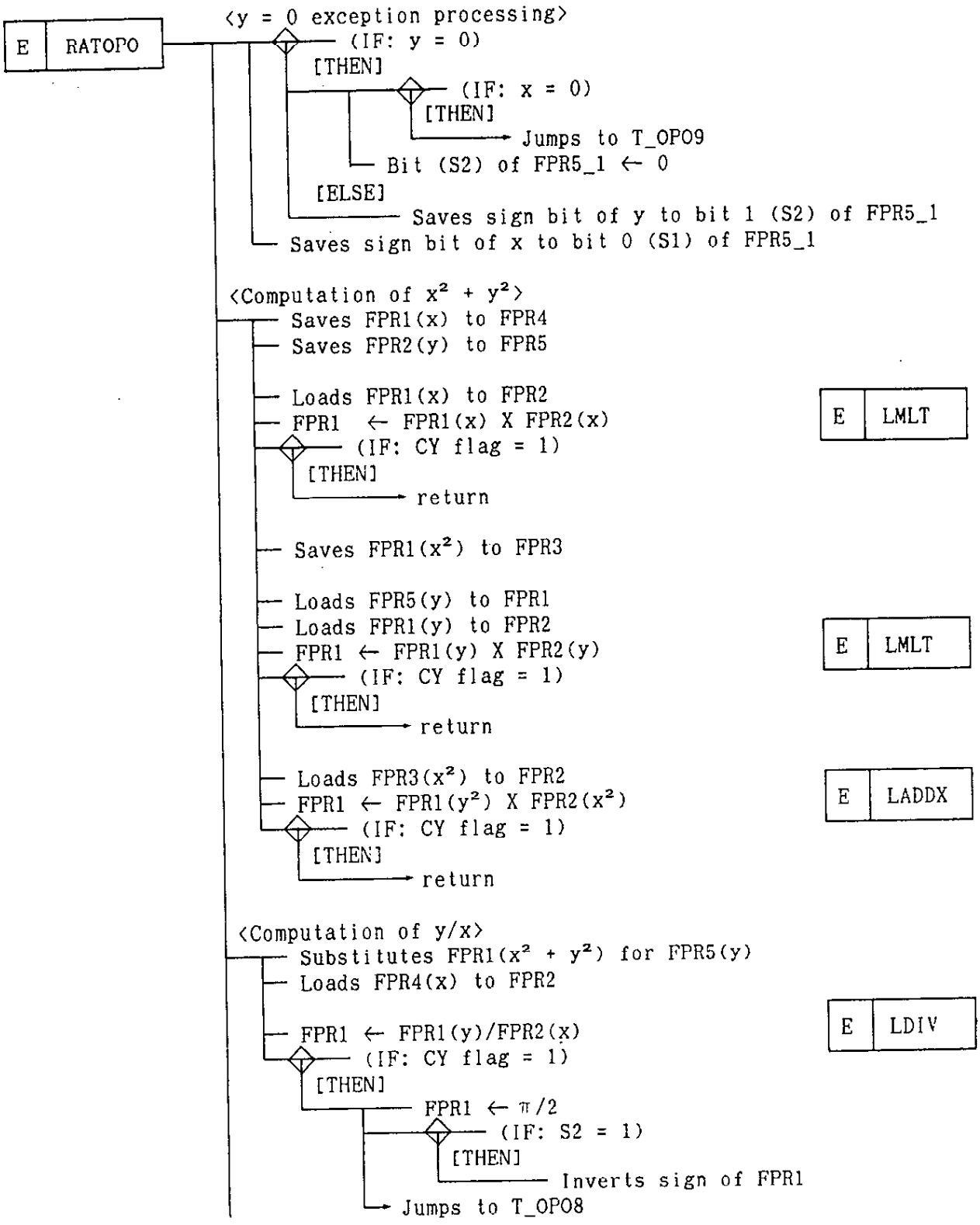
(8) Work area

Bit 0 of FPR5_1 is used to store the sign bit of x, and bit 1 is used to store that for y (these are expressed as variable names S1 and S2 in the processing chart).

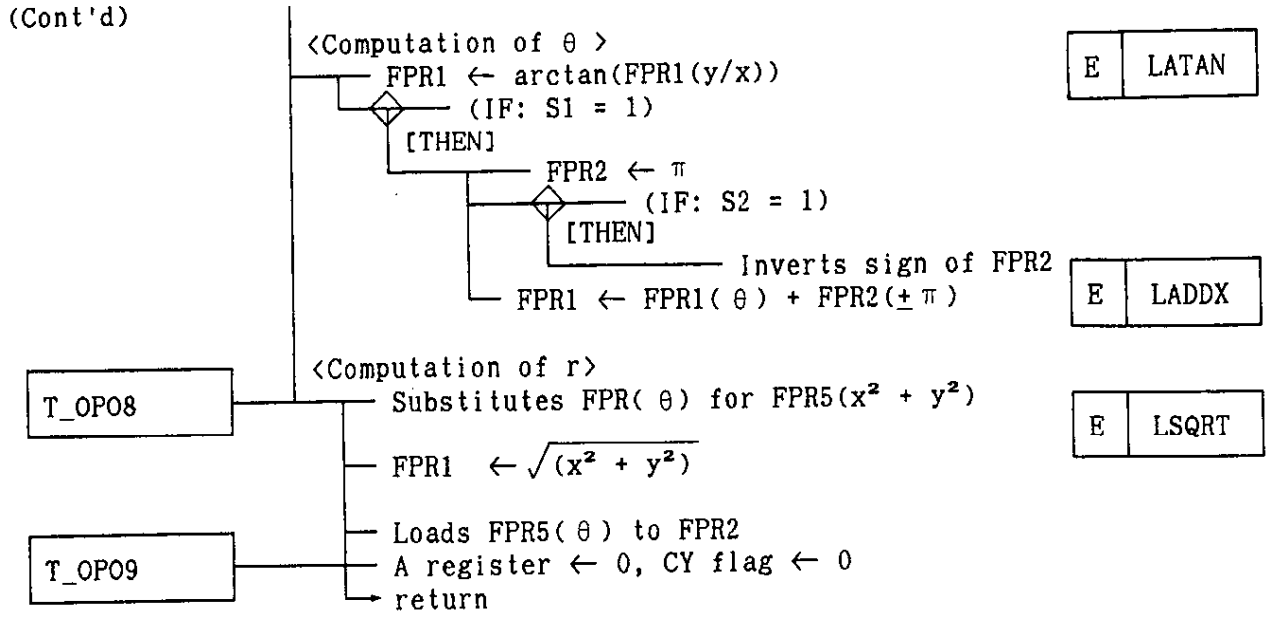
(9) Floating-point constant data

$\pi/2$ and π (π is in extended format) are used as constant data.

(10) Processing chart



(Cont'd)


<Computation of θ >
FPR1 ← arctan(FPR1(y/x))
(IF: S1 = 1)
[THEN]
FPR2 ← π
(IF: S2 = 1)
[THEN]
Inverts sign of FPR2
FPR1 ← FPR1(θ) + FPR2(± π)
E
LATAN
E
LADDX
<Computation of r>
T_OPO8
Substitutes FPR(θ) for FPR5(x² + y²)
FPR1 ← √(x² + y²)
E
LSQRT
T_OPO9
Loads FPR5(θ) to FPR2
A register ← 0, CY flag ← 0
return

CHAPTER 6 TYPE TRANSFORMATION FUNCTIONS

The following type transformation functions are available:

(1) Character string-to-floating-point format transformation function (ATOL)

Transforms a character string, whose first address is indicated by the HL register, into the floating-point format, and stores the result in FPR1.

(2) Floating-point format-to-character string transformation function (LTOA)

Transforms the FPR1 value to a character string and stores it from an address indicated by the HL register.

(3) 2-byte integer type-to-floating-point format transformation function (FTOL)

Transforms the DE register contents as signed 2-byte integer type to floating-point format, and stores the result in FPR1.

(4) Floating-point format-to-2-byte integer type transformation function (LTOF)

Transforms the FPR1 value to signed 2-byte integer type and stores the result in the DE register.

(5) IEEE-754 format-to-78K/II format floating-point format transformation function (LTO78)

Transforms the FPR1 contents from the IEEE-754 format to 78K/II format.

(6) 78K/II format-to-IEEE-754 format floating-point format transformation function (LTOIE)

Transforms the FPR1 contents from the 78K/II format to IEEE-754 format.

6.1 Character String-to-Floating-Point Format Transformation Function (ATOL)

(1) Processing

Transforms the character string, whose first address is indicated by the HL register, to the floating-point format, and returns the result to FPR1.

(2) Object module files to be linked

DFLT, LFLT1, LFLT2, LLD, LEXP, ATOL, FTOL, LTOF

(3) Consumed stack size

14 (including 2 bytes of return address from ATOL)

(4) Registers

FPR1, FPR2, FPR3, FPR4, FPR5

(5) Processing time (internal system clock: 6 MHz)

Average: 5.57 ms

Maximum: 9.34 ms ("1.00000000000000000000000000e - 37")

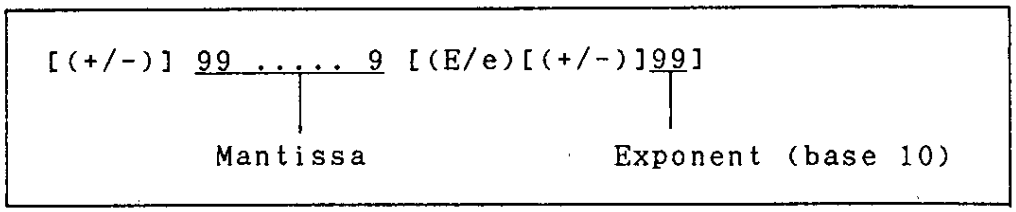
(6) Character string constituents

A character string consists of the following 17 types of characters:

Character	ASCII code
0-9	30H-39H
+	2BH
-	2DH
.	2EH
E	45H
e	65H
△(space)	20H
NUL	00H

(7) Character string format

The character string format is as follows:



Remarks: [ ]: can be omitted
9: 0-9
(/): either one

Example: "-99.8" = -99.8
".007e0" = .007
"0998E-03" = 998 x 10^{-3}

(8) Character string rules

Character strings that do not satisfy the following rules cause an error:

(a) The end of a character sting is identified by a Δ (space) or NUL.

(b) Characters not specified in the above table must not be included in character strings.

(c) The maximum length of the mantissa character string is 27, with one or more '.' included.
However, more than one numeral must be included.

(d) The length of the exponent character string must be 2 or 1.

(e) If the value exceeds 2^{127}, or falls below -2^{127}, an error occurs.

(9) Processing sequence

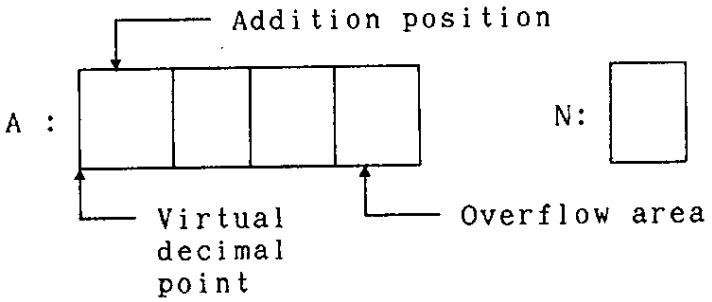
(a) If the first character is '-', the character string is stored as a negative value: if not, it is stored as a positive value.

(b) Obtains the number of digits for the fraction from the mantissa character string and takes it as F.

(c) From the mantissa character string A_1, A_2 ... A_n with the decimal point removed, obtains the value of A in 4-byte integer type.

$$A = (...((A_1 \cdot 10 + A_2) \cdot 10 + A_3) \cdot 10 ... A_{n-1}) \cdot 10 + A_n$$

The computation is carried out as follows:



(i) Takes A = 0 and N = 0 as initial values.

(ii) Adds A_1 to the addition position.
If A_1 does not exist, returns an error.

(iii) Where the overflow area = 0,
multiplies A by 10 and adds A_k.

(iv) Where the overflow area is not equal to 0,
adds 1 to N and ignores A_k.

(v) Repeats (iii) to (iv) until k = 2 - n.

(vi) If n > 27, returns an error.

(d) Normalizes the mantissa value A obtained in (c) to the floating-point format, and takes Y_1 as the exponent, and X_1 as the mantissa (with the exponent as 80H).

(e) Obtains exponent value B from the exponent character

(e) Obtains exponent value B from the exponent character string' (+/-) B_1B_2'.

(f) Obtains real exponent value B' (= B - F + N), which is exponent value B to which is added the number of digits for the fraction and the ignored number of digits for the mantissa.

(g) From X_1, Y_1, and B' obtained above, the solution is obtained with the following expression:

$$
\begin{aligned}
&(-1)^{(sign)} \times X_1 \times 2^{Y_1} \times 10^{B'} \\
&= (-1)^{(sign)} \times X_1 \times 2^{\log_2 10 \times B' + Y_1} \\
&= (-1)^{(sign)} \times X_1 \times 2^{dec(\log_2 10 \times B')} \times 2^{int(\log_2 10 \times B') + Y_1} \\
&= (-1)^{(sign)} \times X_1 \times e^{\log 2 \times dec(\log_2 10 \times B')} \times 2^{int(\log_2 10 \times B') + Y_1}
\end{aligned}
$$

Remarks: dec(x) indicates the x fraction and int(x) indicates the x integer.

(10) Work area

(a) Bit 7 of FPR5_1 is used to as an area to save the sign (this is expressed as symbol s in the processing chart).

(b) The registers are used as follows for computation of the mantissa value:

- o FPR4_1 is used as a counter for the number of digits F in the fraction.
- o FPR4_2 is used as a counter for the ignored number of digits N.
- o X register is used as a counter for the number of digits in the mantissa.

(These counters are expressed as symbols DC1, DC2, and DC3, respectively, in the processing chart.)

DEHL register is used as a work register A, to convert the mantissa character string into a binary number.

(c) After the mantissa normalization, the registers are used as follows:

- o FPR4_1 is used to save F_N (the negative number is in the 2's complement formal).
- o FPR4_2 is used to save the exponent Y_1 of the normalized decimal mantissa value.

(d) The INDEX string, used to identify characters, is as follows:

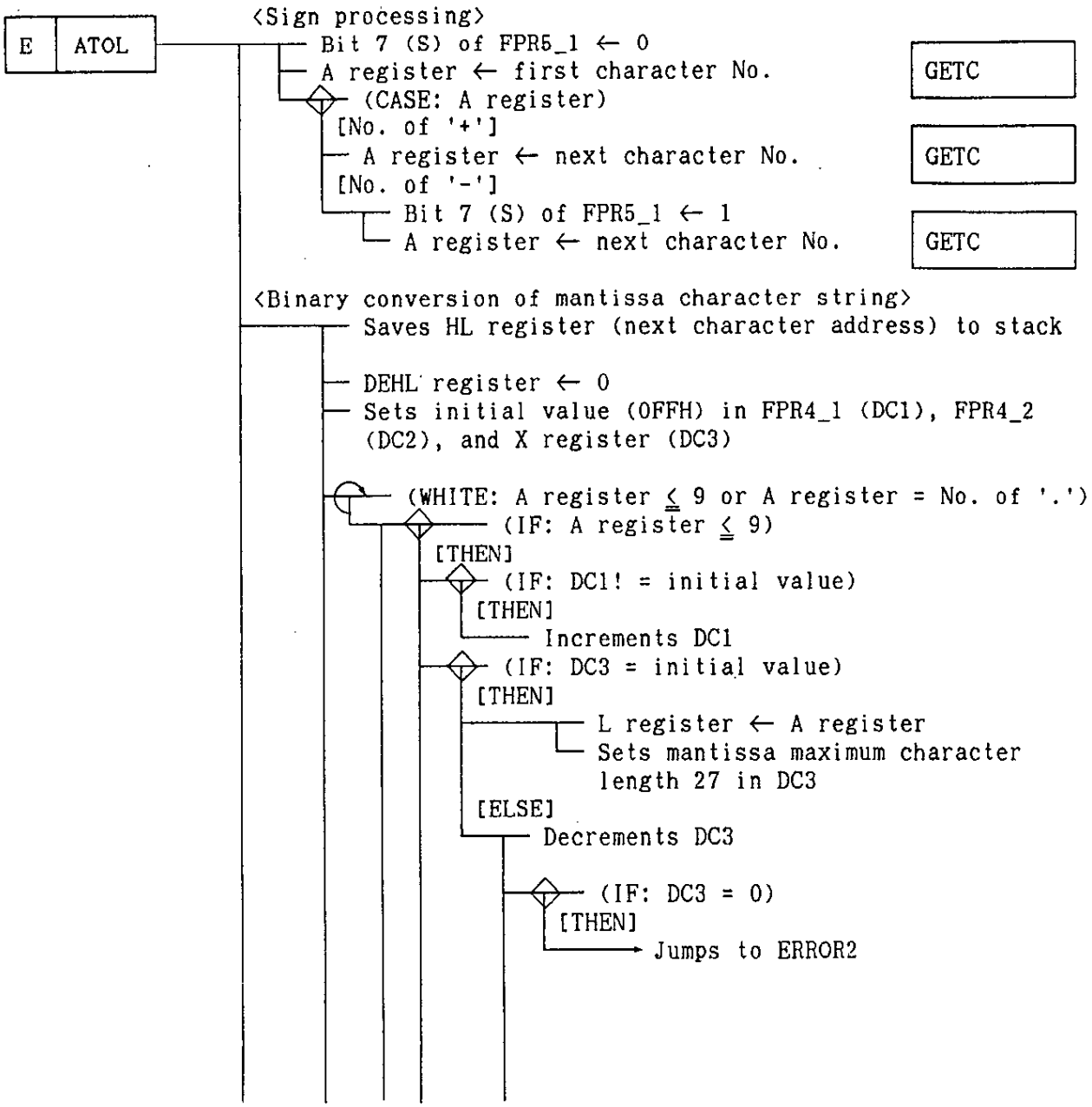
```
'0123456789eE△NUL, -+'
```

Internal subroutine GETC is a function to obtain the position of an input character in the INDEX string, and returns the position numbers 0 through 16 from the leftmost position of the INDEX string and, in case of INDEX error, 0FFH.

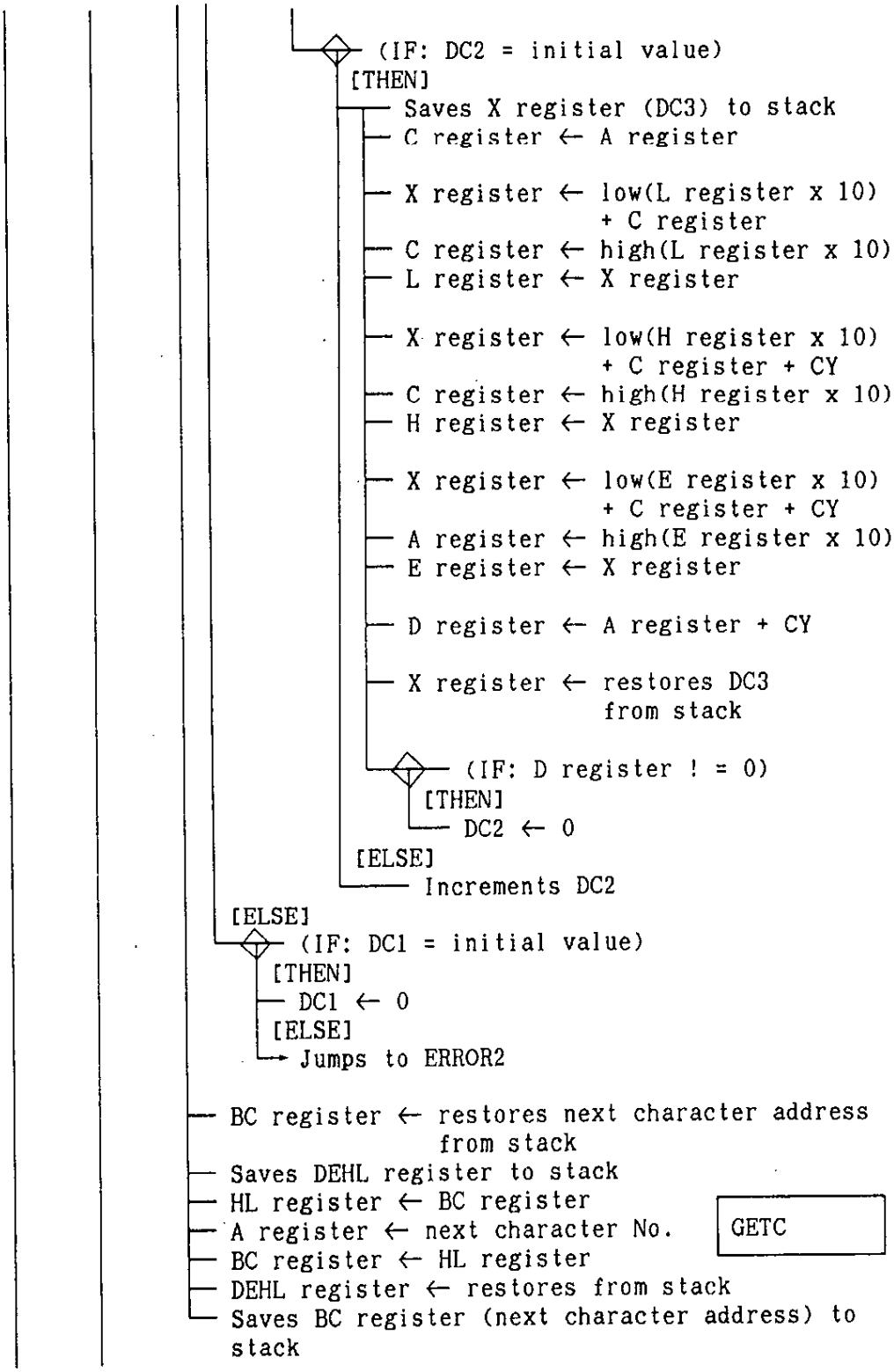
(11) Floating-point constant data

Constant data $\log_2 10$ and log2, having an extended mantissa, are used.

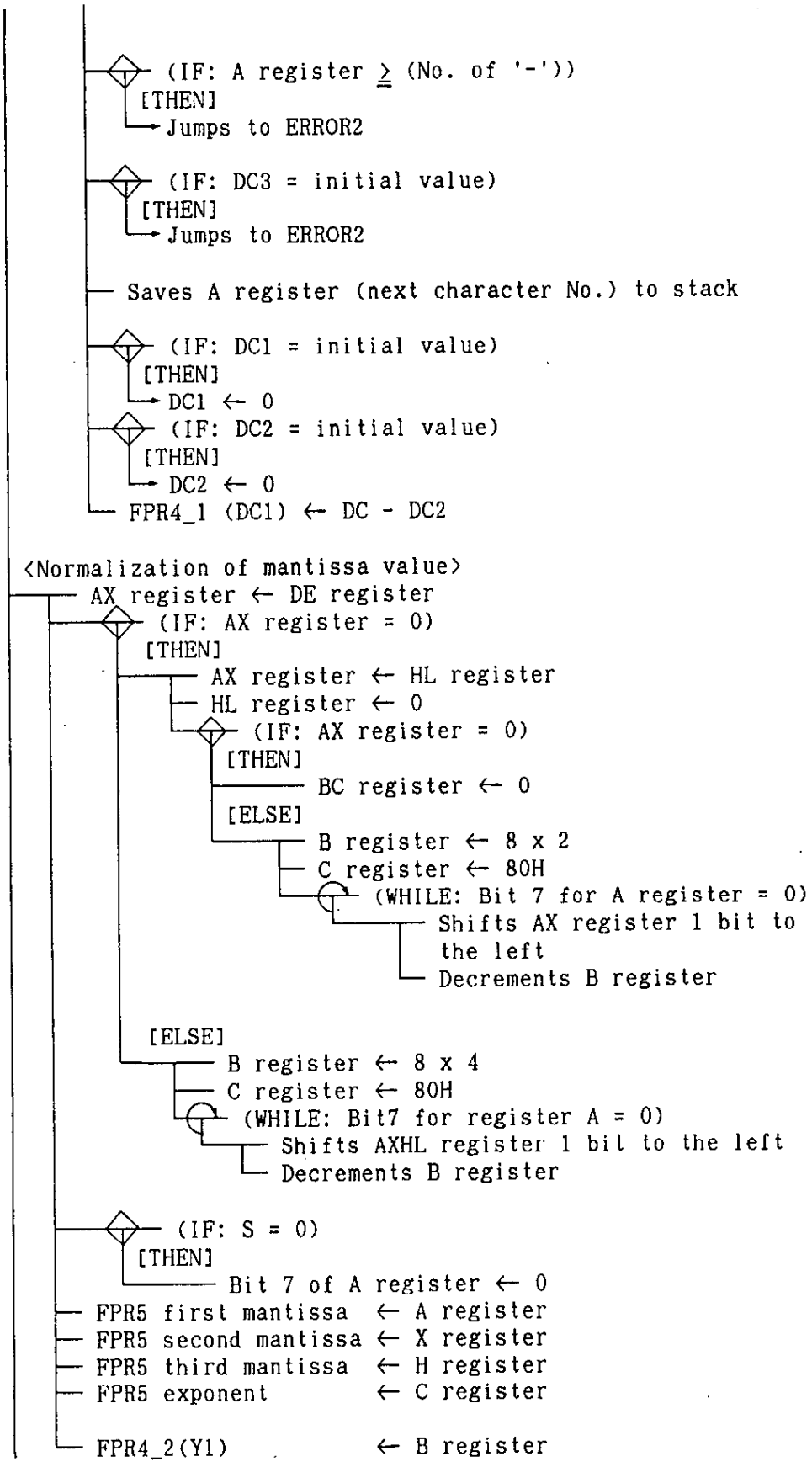
(12) Processing chart



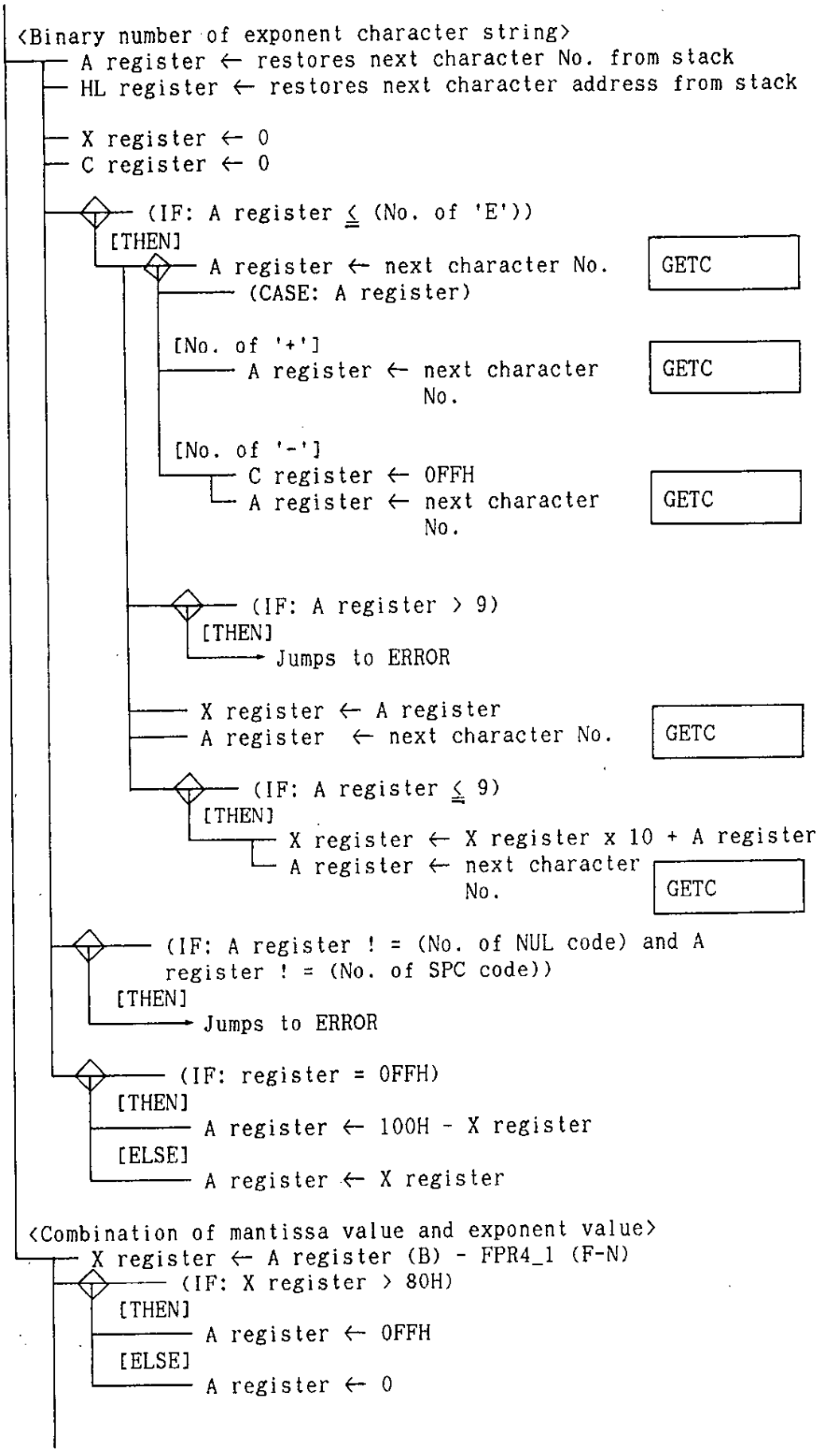
(Cont'd)


(IF: DC2 = initial value)
[THEN]
Saves X register (DC3) to stack
C register ← A register
X register ← low(L register x 10)
+ C register
C register ← high(L register x 10)
L register ← X register
X register ← low(H register x 10)
+ C register + CY
C register ← high(H register x 10)
H register ← X register
X register ← low(E register x 10)
+ C register + CY
A register ← high(E register x 10)
E register ← X register
D register ← A register + CY
X register ← restores DC3
from stack
(IF: D register ! = 0)
[THEN]
DC2 ← 0
[ELSE]
Increments DC2
[ELSE]
(IF: DC1 = initial value)
[THEN]
DC1 ← 0
[ELSE]
Jumps to ERROR2
BC register ← restores next character address
from stack
Saves DEHL register to stack
HL register ← BC register
A register ← next character No.
BC register ← HL register
DEHL register ← restores from stack
Saves BC register (next character address) to
stack
GETC

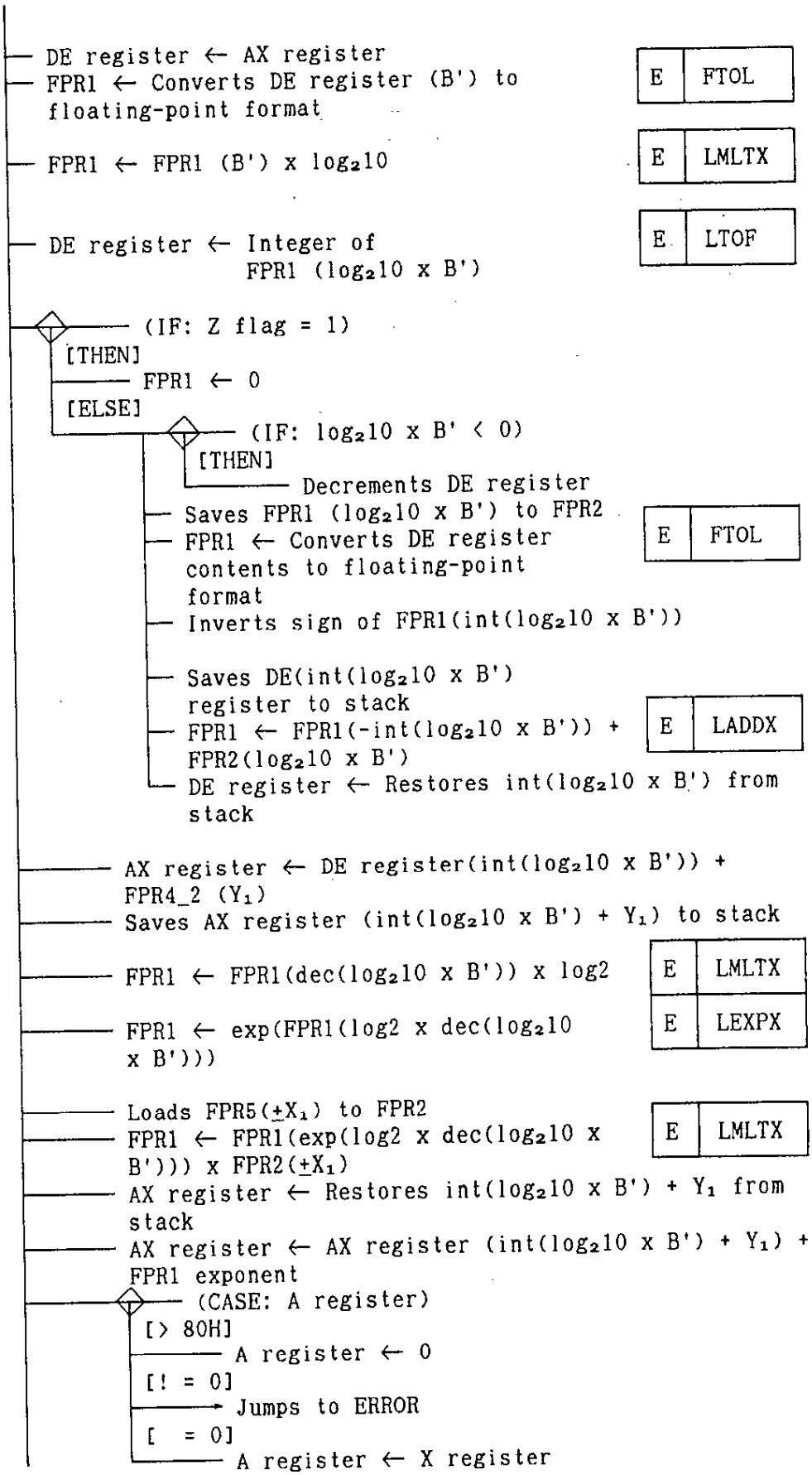
(Cont'd)


(IF: A register ≥ (No. of '-'))
[THEN]
Jumps to ERROR2
(IF: DC3 = initial value)
[THEN]
Jumps to ERROR2
Saves A register (next character No.) to stack
(IF: DC1 = initial value)
[THEN]
DC1 ← 0
(IF: DC2 = initial value)
[THEN]
DC2 ← 0
FPR4_1 (DC1) ← DC - DC2
<Normalization of mantissa value>
AX register ← DE register
(IF: AX register = 0)
[THEN]
AX register ← HL register
HL register ← 0
(IF: AX register = 0)
[THEN]
BC register ← 0
[ELSE]
B register ← 8 x 2
C register ← 80H
(WHILE: Bit 7 for A register = 0)
Shifts AX register 1 bit to the left
Decrements B register
[ELSE]
B register ← 8 x 4
C register ← 80H
(WHILE: Bit7 for register A = 0)
Shifts AXHL register 1 bit to the left
Decrements B register
(IF: S = 0)
[THEN]
Bit 7 of A register ← 0
FPR5 first mantissa ← A register
FPR5 second mantissa ← X register
FPR5 third mantissa ← H register
FPR5 exponent ← C register
FPR4_2(Y1) ← B register

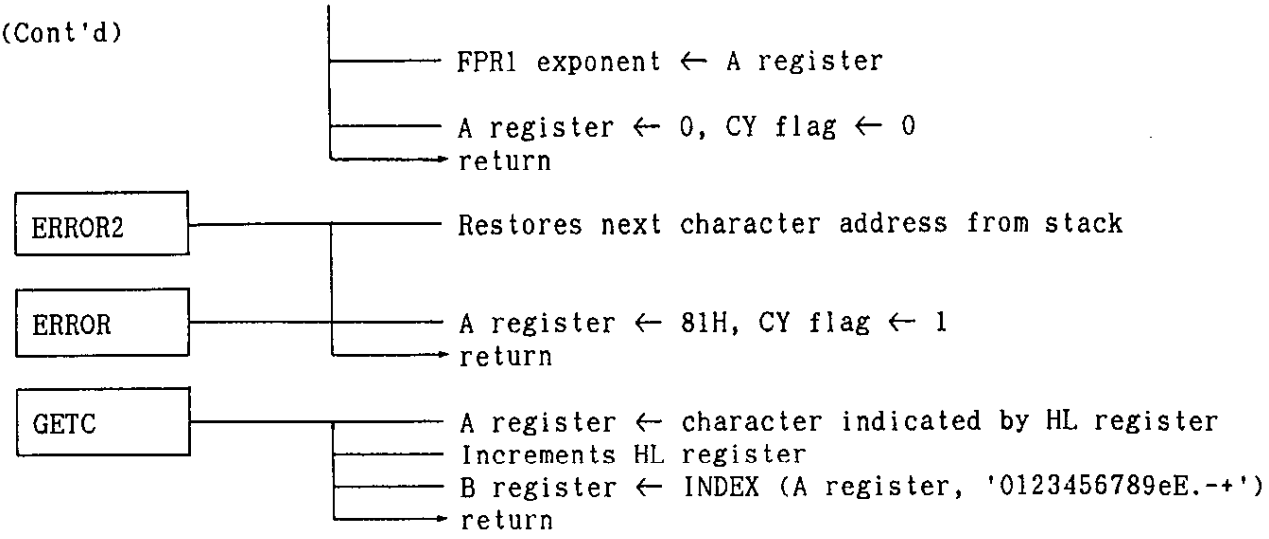
(cont'd)


<Binary number of exponent character string>
A register ← restores next character No. from stack
HL register ← restores next character address from stack
X register ← 0
C register ← 0
(IF: A register ≦ (No. of 'E'))
[THEN]
A register ← next character No.
GETC
(CASE: A register)
[No. of '+']
A register ← next character No.
GETC
[No. of '-']
C register ← 0FFH
A register ← next character No.
GETC
(IF: A register > 9)
[THEN]
Jumps to ERROR
X register ← A register
A register ← next character No.
GETC
(IF: A register ≦ 9)
[THEN]
X register ← X register x 10 + A register
A register ← next character No.
GETC
(IF: A register ! = (No. of NUL code) and A register ! = (No. of SPC code))
[THEN]
Jumps to ERROR
(IF: register = 0FFH)
[THEN]
A register ← 100H - X register
[ELSE]
A register ← X register
<Combination of mantissa value and exponent value>
X register ← A register (B) - FPR4_1 (F-N)
(IF: X register > 80H)
[THEN]
A register ← 0FFH
[ELSE]
A register ← 0

(Cont'd)


DE register ← AX register
FPR1 ← Converts DE register (B') to floating-point format
E FTOL
FPR1 ← FPR1 (B') x log₂10
E LMLTX
DE register ← Integer of FPR1 (log₂10 x B')
E LTOF
(IF: Z flag = 1)
[THEN]
FPR1 ← 0
[ELSE]
(IF: log₂10 x B' < 0)
[THEN]
Decrements DE register
Saves FPR1 (log₂10 x B') to FPR2
FPR1 ← Converts DE register contents to floating-point format
E FTOL
Inverts sign of FPR1(int(log₂10 x B'))
Saves DE(int(log₂10 x B') register to stack
FPR1 ← FPR1(-int(log₂10 x B')) + FPR2(log₂10 x B')
E LADDX
DE register ← Restores int(log₂10 x B') from stack
AX register ← DE register(int(log₂10 x B')) + FPR4_2 (Y₁)
Saves AX register (int(log₂10 x B') + Y₁) to stack
FPR1 ← FPR1(dec(log₂10 x B')) x log2
E LMLTX
FPR1 ← exp(FPR1(log2 x dec(log₂10 x B')))
E LEXPX
Loads FPR5(±X₁) to FPR2
FPR1 ← FPR1(exp(log2 x dec(log₂10 x B'))) x FPR2(±X₁)
E LMLTX
AX register ← Restores int(log₂10 x B') + Y₁ from stack
AX register ← AX register (int(log₂10 x B') + Y₁) + FPR1 exponent
(CASE: A register)
[> 80H]
A register ← 0
[! = 0]
Jumps to ERROR
[= 0]
A register ← X register

(Cont'd)



Remarks:
1. high ( ) indicates the higher byte of the multiplication result.
2. low ( ) indicates the lower byte of the multiplication result.

6.2 Floating-Point Format-to-Character String Transformation Function (LTOA)

(1) Processing

Transforms the FPR1 value to a character string and stores it from the address indicated by the HL register.

(2) Object module files to be linked

DFLT, LFLT1, LFLT2, LLD, LLOG, LLOG10, LEXP, LEXP10, LTOA, FTOL, LTOF

(3) Consumed stack size

16 (including 2 bytes of return address from LTOA)

(4) Registers

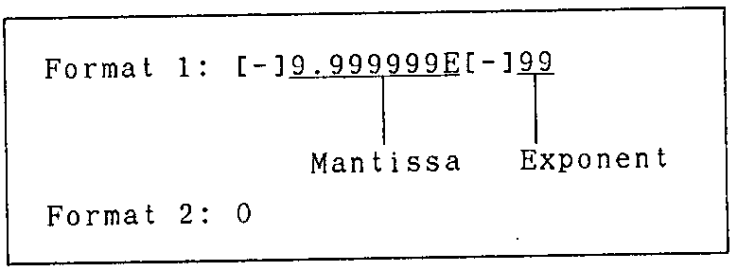
FPR1, FPR2, FPR3, FPR4, FPR5

(5) Processing time (internal system clock: 6 MHz)

Average: 9.87 ms

Maximum: 11.97 ms (6E120377)

(6) Output character string format



Rules:

(a) Outputs in format 2, only when 0; otherwise, format 1 will be used.

(b) 'NUL' code is suffixed to the character string.

(c) The character length of the mantissa and exponent in format 1 are fixed to 8 and 2, respectively.

(d) The sign is appended to the mantissa and exponent in format 1, only when the mantissa and exponent are negative.

(e) The maximum character length is 14 characters, including the 'NUL' code.

(7) Processing sequence

(a) If floating-point value X = 0, outputs character string 'ONUL' and terminates.

(b) Converts X to a x 10^b with the following expression:

$b = \text{floor}(\log_{10}(|X|))$, $b > -39 \rightarrow a = X \times 10^{-b}$
$b = -39$ or $-40 \rightarrow a = (X \times 10^2) \times 10^{-b-2}$

In the above expression, floor(X) indicates the integer closest to X in the negative direction.
10^{-b} is not directly computed, when b is less than -39, because 10^{39} overflows with the 78K/II floating-point system.

(c) If $a < 0$, outputs '-', and $a \leftarrow |a|$.

(d) Although a is mathematically $1 \leqq a < 10$, actually $a < 1$ or $a \geqq 10$, due to calculation error. In this case, the following adjustment is made:

If $a < 1$, $a \leftarrow a \times 10$, $b \leftarrow b - 1$
If $a \geqq 10$, $a \leftarrow a/10$, $b \leftarrow b + 1$

(e) The position of the decimal point is fixed, because $1 \leqq a < 10$.
From the result of the following calculation, outputs character string' A_1, A_2 ... A_7':

```
A1 = int(a),   a = dec(a) x 10
A2 = int(a),   a = dec(a) x 10
     ...................
A7 = int(a)
```

where int(a) indicates the integer of a and dec(a) indicates the fraction of a.

(f) Outputs 'E'.

(g) If $b < 0$, outputs '-', and $b \leftarrow |b|$.

(h) Outputs the character string 'B_1B_2' (B_1 = b/10, B_2 = b - B_1x10) of b.

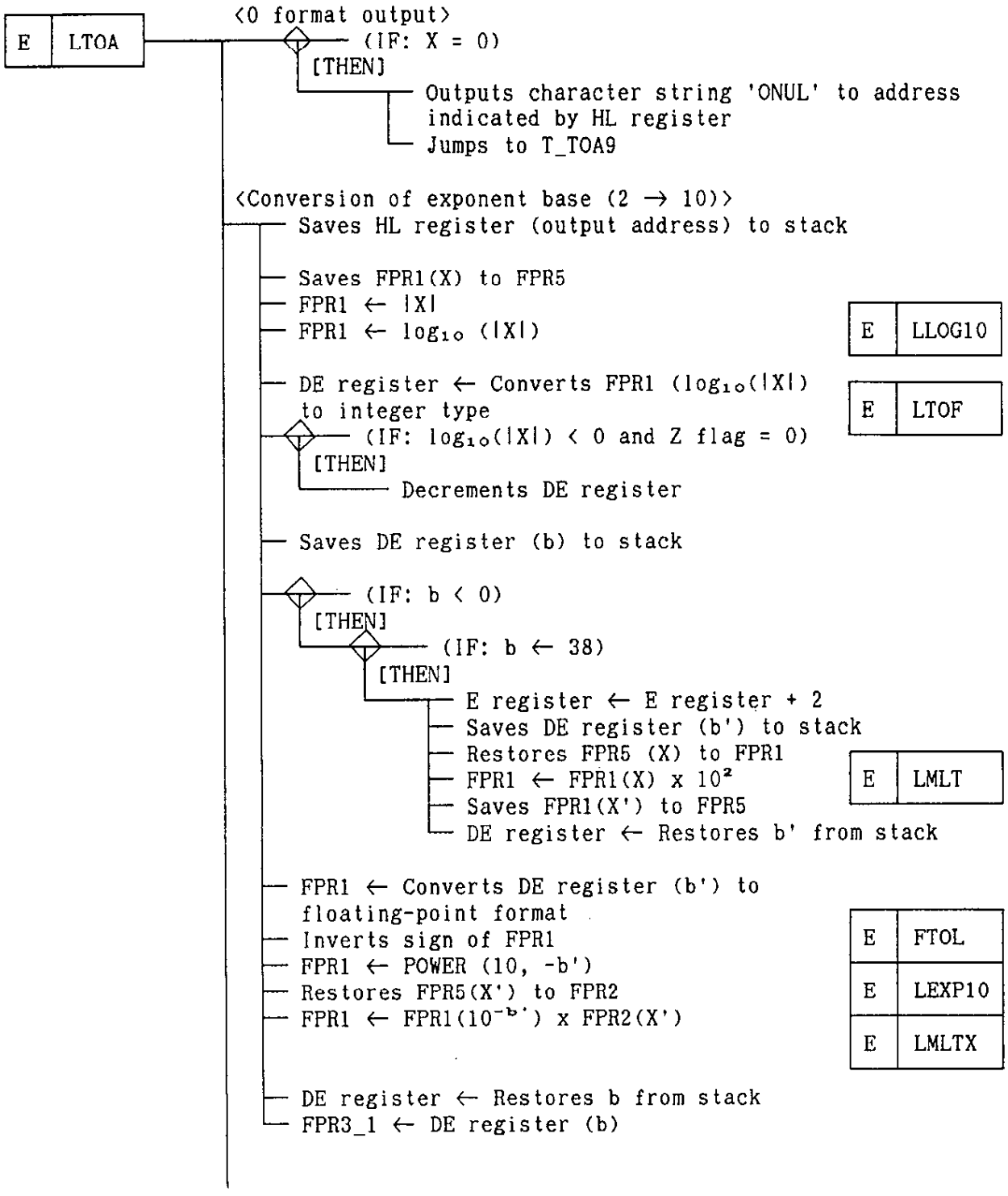
(i) Outputs 'NUL' code.

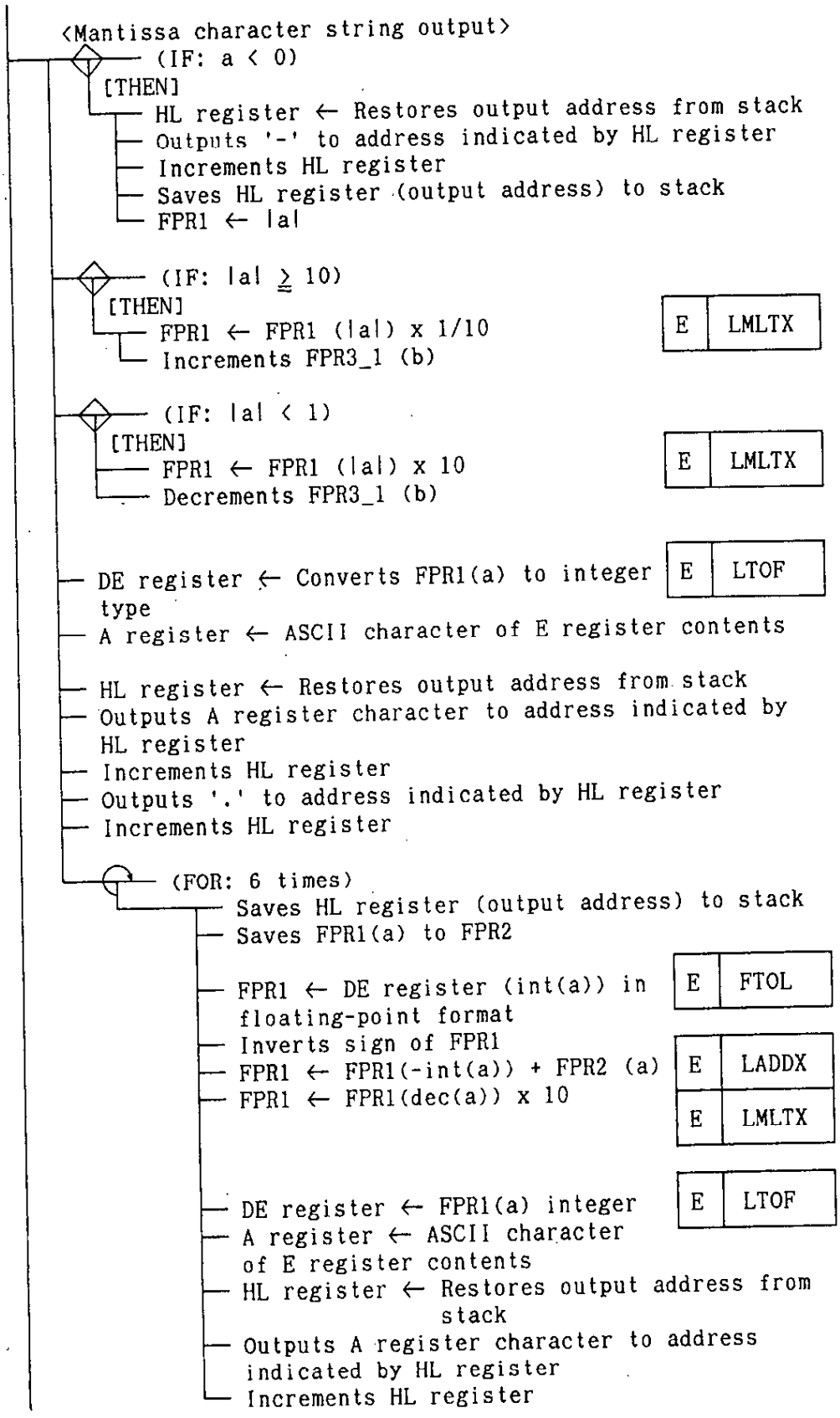
(8) Floating-point constant data

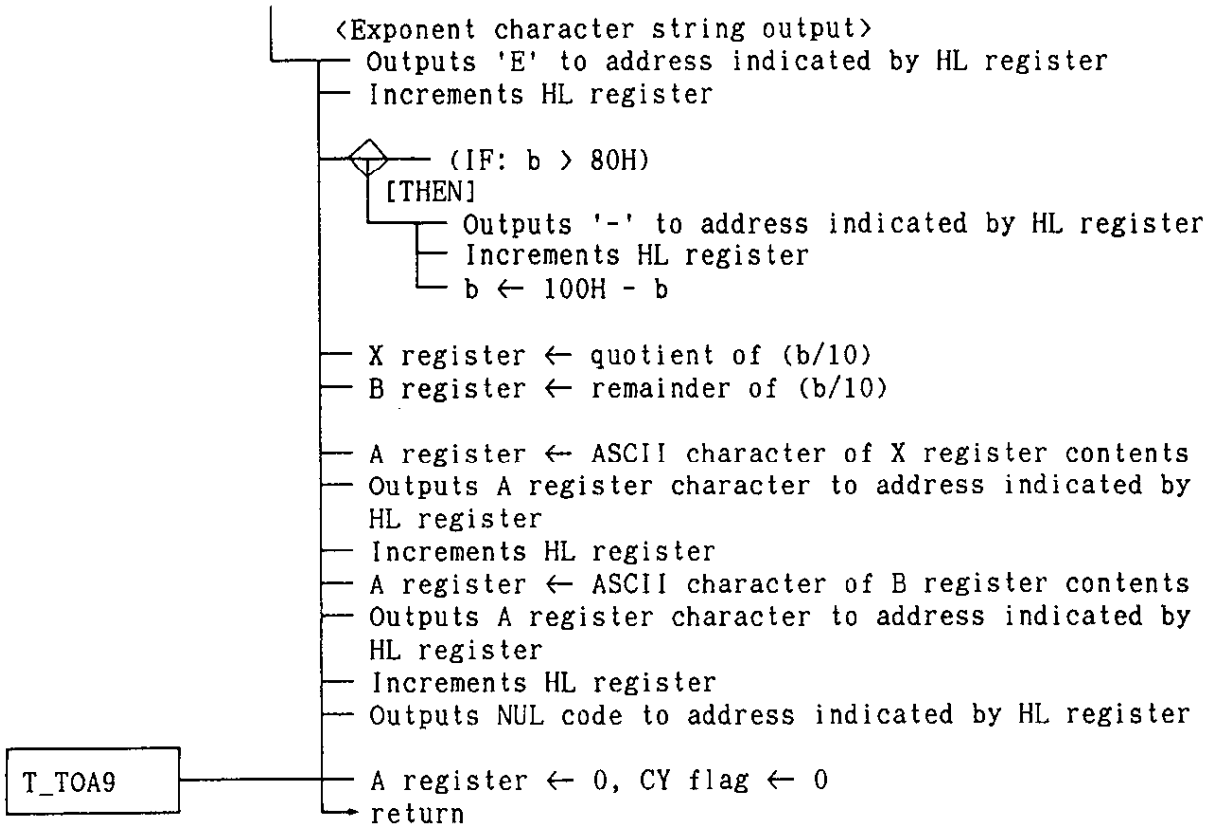
(a) Constant data 100 is used.

(b) Constant data 10 and 1/10 having an extended mantissa are used.

(9) Processing chart




<Mantissa character string output>
(IF: a < 0)
[THEN]
HL register ← Restores output address from stack
Outputs '-' to address indicated by HL register
Increments HL register
Saves HL register (output address) to stack
FPR1 ← |a|
(IF: |a| ≥ 10)
[THEN]
FPR1 ← FPR1 (|a|) x 1/10
Increments FPR3_1 (b)
E
LMLTX
(IF: |a| < 1)
[THEN]
FPR1 ← FPR1 (|a|) x 10
Decrements FPR3_1 (b)
E
LMLTX
DE register ← Converts FPR1(a) to integer type
E
LTOF
A register ← ASCII character of E register contents
HL register ← Restores output address from stack
Outputs A register character to address indicated by HL register
Increments HL register
Outputs '.' to address indicated by HL register
Increments HL register
(FOR: 6 times)
Saves HL register (output address) to stack
Saves FPR1(a) to FPR2
FPR1 ← DE register (int(a)) in floating-point format
E
FTOL
Inverts sign of FPR1
FPR1 ← FPR1(-int(a)) + FPR2 (a)
E
LADDX
FPR1 ← FPR1(dec(a)) x 10
E
LMLTX
DE register ← FPR1(a) integer
E
LTOF
A register ← ASCII character of E register contents
HL register ← Restores output address from stack
Outputs A register character to address indicated by HL register
Increments HL register


T_TOA9
<Exponent character string output>
Outputs 'E' to address indicated by HL register
Increments HL register
(IF: b > 80H)
[THEN]
Outputs '-' to address indicated by HL register
Increments HL register
b ← 100H - b
X register ← quotient of (b/10)
B register ← remainder of (b/10)
A register ← ASCII character of X register contents
Outputs A register character to address indicated by HL register
Increments HL register
A register ← ASCII character of B register contents
Outputs A register character to address indicated by HL register
Increments HL register
Outputs NUL code to address indicated by HL register
A register ← 0, CY flag ← 0
return

6.3 2-Byte Integer Type-to-Floating-Point Format Transformation Function (FTOL)

(1) Processing

Taking the DE register pair contents as signed 2-byte integer type, transforms them to floating-point format, and returns them to FPR1 (the DE register contents are preserved).

(2) Object module files to be linked

DFLT, FTOL

(3) Consumed stack size

2 (only 2 bytes of return address from FTOL)

(4) Registers

FPR1

(5) Processing time (internal system clock: 6 MHz)

Average: 41 us

Maximum: 72 us (-1)

(6) 2-byte integer type format

The most significant bit serves as the sign bit. The negative value is represented as a 2's complement.

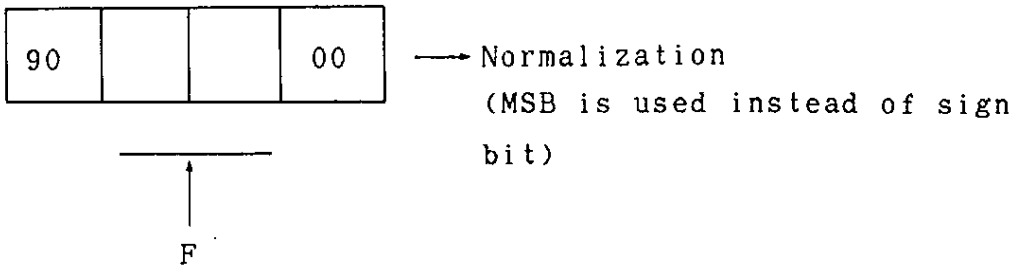
2-byte integer type F takes an integer value in the -8000H to + 7FFFH range.

(7) Processing sequence

(a) If 2-byte integer value F = 0, returns 0.

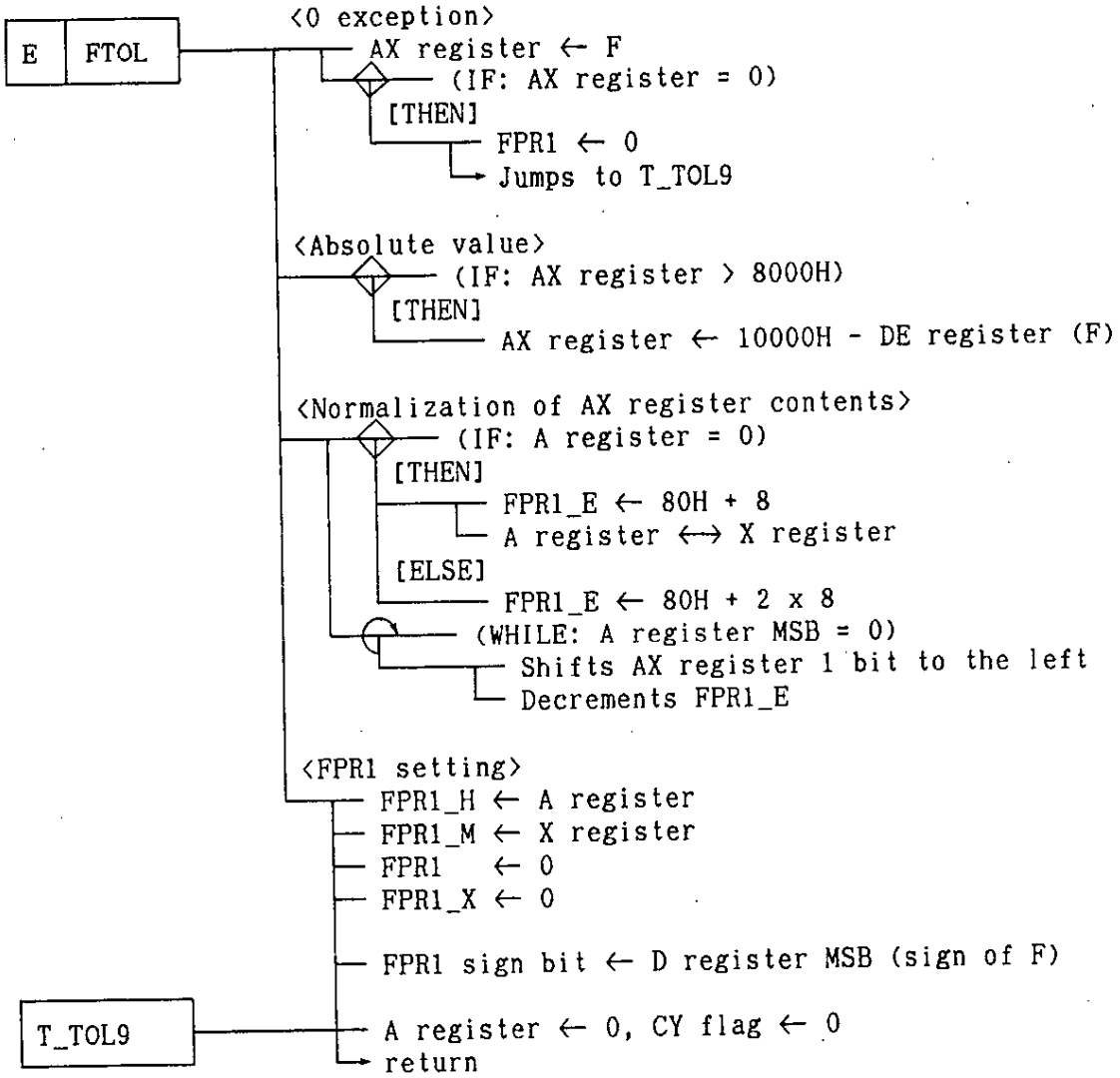
(b) If $-7FFFH \leq F < 0$, takes 2's complement of F.

(c) Transforms F to the floating-point format by the following method:



(d) Embeds the original sign of F into the floating-point format.

(8) Processing chart



6.4 Floating-Point Format-to-2-Byte Integer Type Transformation Function (LTOF)

(1) Processing

Transforms the FPR1 value to signed 2-byte integer type and returns it to the DE register.

If the faction is not included in FPR1, returns Z flag = 1: if the fraction is truncated in the process of the transformation, returns Z flag = 0 (the FPR1 contents are preserved).

(2) Object module files to be linked

DFLT, LTOF

(3) Consumed stack size

2 (only 2 bytes of return address from LTOF)

(4) Registers

FPR1

(5) Processing time (internal system clock: 6 MHz)

Average: 61 us

Maximum: 77 us (-1)

(6) Processing sequence

(a) If the exponent = 0, returns integer value 0 and Z flag = 1.

If the exponent $\leq$ 80H, returns integer value 0 and Z flag = 0.

If the exponent > 90H, returns an error.

(b) Extracts the integer in the unsigned integer format.

Sets the MSB.

If the exponent $\leq$ 88H, shifts the first mantissa to the right by (88H - exponent) bits and obtains an integer value.

If the exponent > 88H, shifts the first and second mantissas to the right (90H - exponent) bits and

obtains an integer value.

(c) Transforms the unsigned integer format to integer format.

The integer value obtained in (b) is treated as follows:

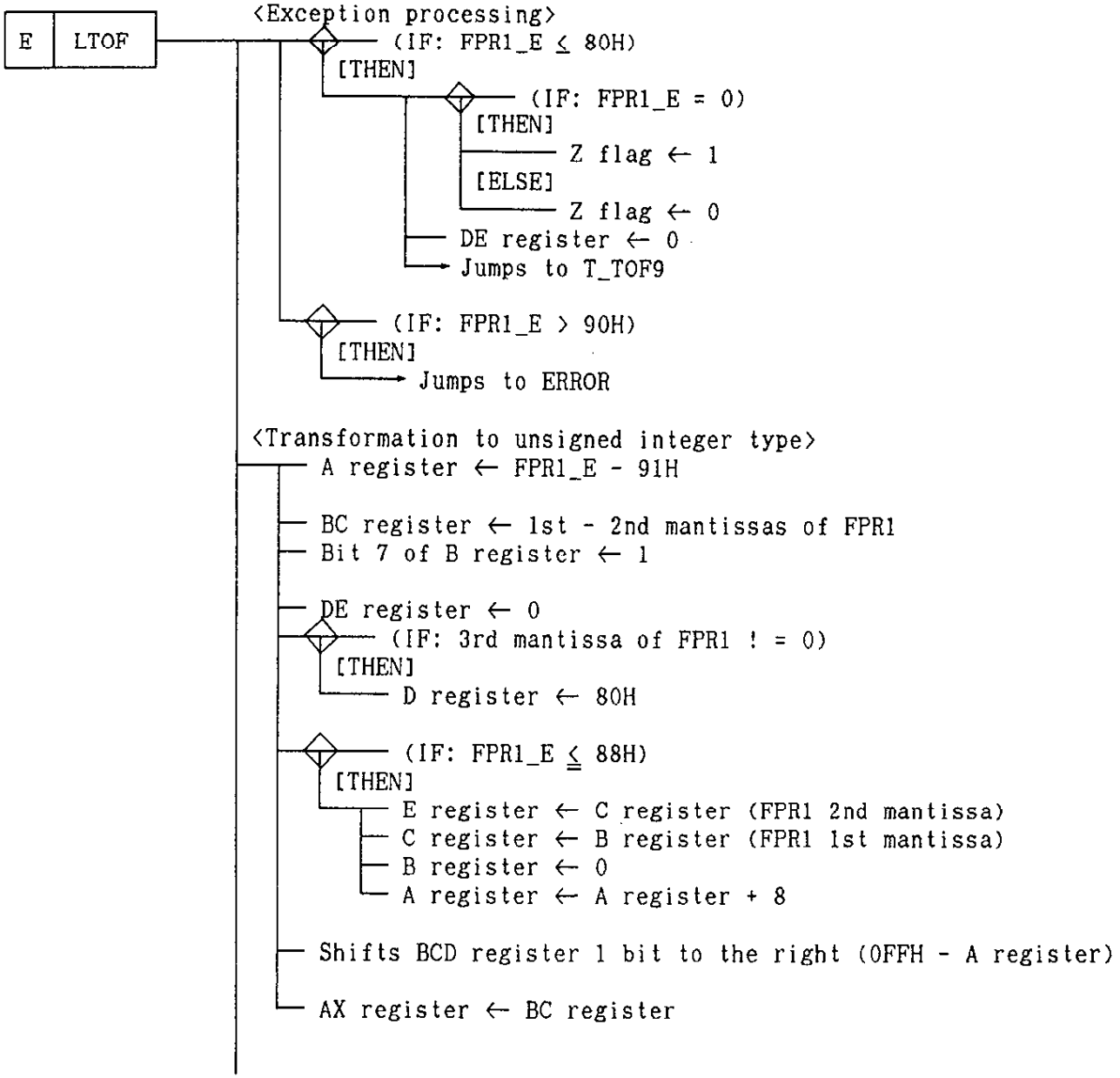
Greater than 8000H	→	Error
8000H, positive	→	Error
8000H, negative	→	As is
Less than 8000H, positive	→	As is
Less than 8000H, negative	→	Takes 2's complement as solution

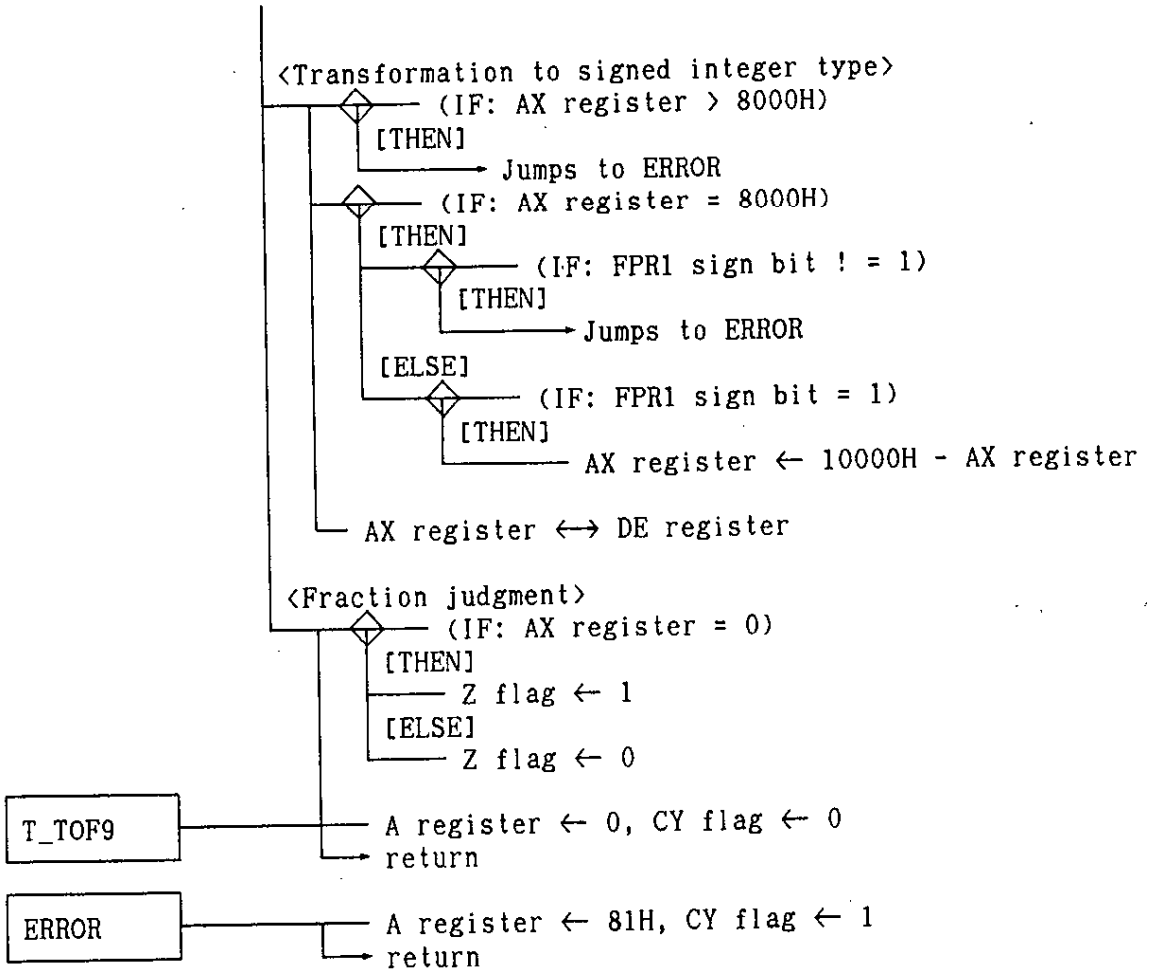
(d) In (b),

- To return Z flag = 1
  - If the exponent $\leq$ 88H, if all the bits of the second and third mantissas are 0, and if no carry is generated in the process of the right shift.
  - If the exponent > 88H, if all the bits of the third mantissa are 0, and if no carry is generated in the process of the right shift of the first and second mantissas.
- To return Z flag = 0

  Other than above

(7) Processing chart




<Transformation to signed integer type>
(IF: AX register > 8000H)
[THEN]
Jumps to ERROR
(IF: AX register = 8000H)
[THEN]
(IF: FPR1 sign bit ! = 1)
[THEN]
Jumps to ERROR
[ELSE]
(IF: FPR1 sign bit = 1)
[THEN]
AX register ← 10000H - AX register
AX register ⟷ DE register
<Fraction judgment>
(IF: AX register = 0)
[THEN]
Z flag ← 1
[ELSE]
Z flag ← 0
T_TOF9
A register ← 0, CY flag ← 0
return
ERROR
A register ← 81H, CY flag ← 1
return

6.5 IEEE-754 Format-to-78K/II Format Transformation Function with Floating-Point Format (LTO78)

(1) Processing

Transforms the FPR1 contents from the IEEE-754 format to 78K/II format.

(2) Object module files to be linked

DFLT, LTO78

(3) Consumed stack size

2 (only 2 bytes of return address from LTO78)

(4) Registers

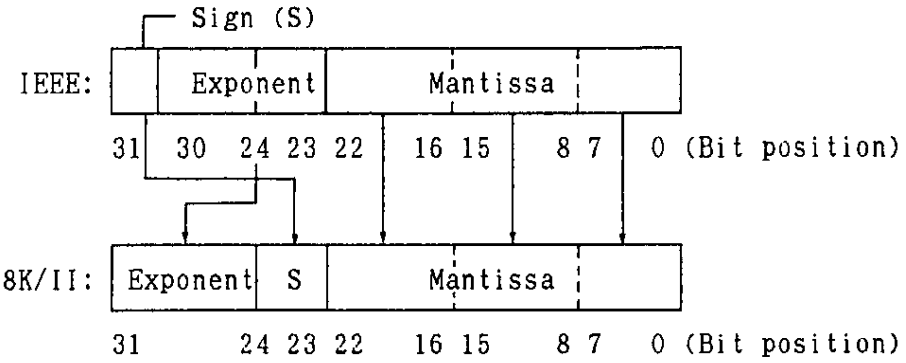
FPR1

(5) Processing time (internal system clock: 6 MHz)

17.8 us

(6) Processing sequence

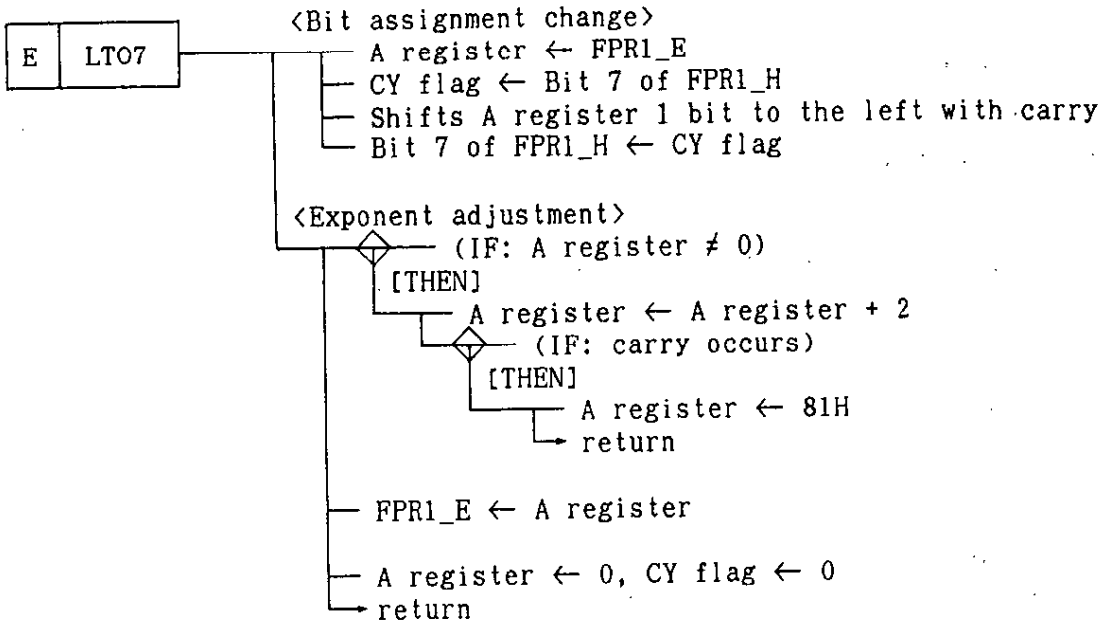
(a) Transforms the bit assignments for the sign, exponent, and mantissa in the IEEE format to the bit assignments in the 78K/II format.



(b) If the exponent $\geq$ 0FEH, returns an error.

(c) If the exponent is not equal to 0, adds 2 to the exponent.

(7) Processing chart



6.6 78K/II Format-to-IEEE-754 Format Transformation Function with Floating-Point Format (LTOIE)

(1) Processing

Transforms the FPR1 contents from the 78K/II format to the IEEE-754 format.

(2) Object module files to be linked

DFLT, LTOIE

(3) Consumed stack size

2 (only 2 bytes of return address from LTOIE)

(4) Registers

FPR1

(5) Processing time (internal system clock: 6 MHz)

17.8 us

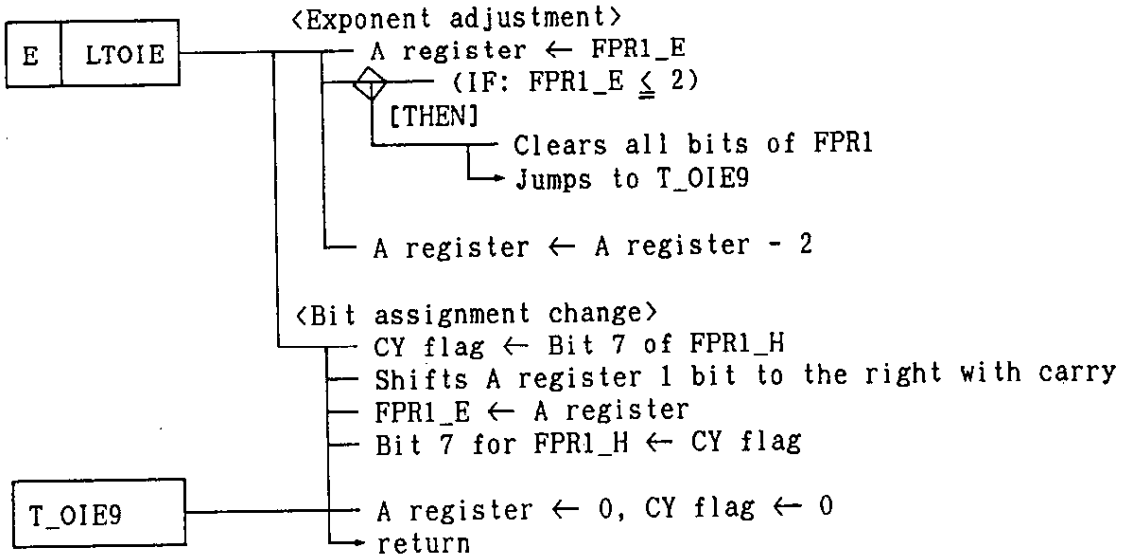
(6) Processing sequence

(a) If the exponent $\leqq$ 2, clears all the bits of FPR1 and terminates.

(b) Subtracts 2 from the exponent.

(c) Changes the bit assignment in the reverse order to the LTO78 function.

(7) Processing chart



CHAPTER 7 PROGRAM LIST

(1) EQU. INC

```
SHORT   EQU     4       ;size of real type
INTEGR  EQU     2       ;size of integer type
BYTE    EQU     8       ;bit figures of byte
ZEROEX  EQU     080H    ;expornent bias for 0
R_OK    EQU     0       ;normal return code
R_ERR   EQU     81H     ;abnormal return code
```

(2) REF1. INC

```
EXTRN FPR1,FPR2

EXTRN FPR1_X,FPR1_M,FPR1_H,FPR1_E
EXTRN FPR2_X,FPR2_M,FPR2_H,FPR2_E

EXTRN FPR3,FPR3_1
EXTRN FPR4,FPR4_1,FPR4_2

EXTRN FPR5,FPR5_1,FPR5_2
```

(3) REF2. INC

```
EXTRN    LLD21,LLD21X
EXTRN    LLD31,LLD31X
EXTRN    LLD41,LLD41X
EXTRN    LLD51,LLD51X
EXTRN    LLD52
EXTRN    LLD13
EXTRN    LLD23,LLD23X
EXTRN    LLD14,LLD14X
EXTRN    LLD24,LLD24X
EXTRN    LLD15
EXTRN    LLD25,LLD25X
EXTRN    LLD1C,LLD1CX
EXTRN    LLD2C,LLD2CX

EXTRN    LXC13,LXC13X
EXTRN          LXC14X
EXTRN    LXC15,LXC15X
```

(4) ASCII. INC

```
A_PL    EQU     02BH        ;'+'
A_MN    EQU     02DH        ;'-'
A_PD    EQU     02EH        ;'.'
A_NL    EQU     000H        ;nul
A_SP    EQU     020H        ;space
A_E     EQU     045H        ;'E'
A_E2    EQU     065H        ;'e'
A_0     EQU     030H        ;'0'
A_9     EQU     039H        ;'9'

N_PL    EQU     16
N_MN    EQU     15
N_PD    EQU     14
N_NL    EQU     13
N_SP    EQU     12
N_E     EQU     11
N_9     EQU     9

S_INDX  EQU     17

@_INDX  MACRO
        DB A_PL, A_MN, A_PD, A_NL
        DB A_SP, A_E,  A_E2, A_9
        DB A_9-1,A_9-2,A_9-3,A_9-4
        DB A_9-5,A_9-6,A_9-7,A_9-8
        DB A_9-9
        ENDM
```

(5) DFLT. SRC

```
$       TITLE   ('FLOATING POINT REGISTERS')
        NAME    M_DFLT
;*****************************************************
;*
;*  FLOATING POINT REGISTERS
;*
;*              (ADDRESSING IN SADDR)
;*
;*****************************************************

        PUBLIC FPR1,FPR2

        PUBLIC FPR1_X,FPR1_M,FPR1_H,FPR1_E
        PUBLIC FPR2_X,FPR2_M,FPR2_H,FPR2_E

        PUBLIC FPR3,FPR3_1
        PUBLIC FPR4,FPR4_1,FPR4_2
        PUBLIC FPR5,FPR5_1,FPR5_2

        DSEG   SADDR
;******** FLOATING POINT REGISTER 1 **
FPR1_X:
        DS     1
FPR1:
        DS     1
FPR1_M:
        DS     1
FPR1_H:
        DS     1
FPR1_E:
        DS     1

;******** FLOATING POINT REGISTER 2 **
FPR2_X:
        DS     1
FPR2:
        DS     1
FPR2_M:
        DS     1
FPR2_H:
        DS     1
FPR2_E:
        DS     1
```

```
;******** FLOATING POINT REGISTER 3 **
FPR3:
FPR3_1:
        DS      1
        DS      1
        DS      1
        DS      1
        DS      1

;******** FLOATING POINT REGISTER 4 **
FPR4:
FPR4_1:
        DS      1
FPR4_2:
        DS      1
        DS      1
        DS      1
        DS      1

;******** FLOATING POINT REGISTER 5 **
FPR5:
FPR5_1:
        DS      1
FPR5_2:
        DS      1
        DS      1
        DS      1
        DS      1

        END
```

(6) LFLT1. SRC

```
$       TITLE   ('THE 4 RULES FUNCTIONS')
        NAME    M_LFLT1

$ INCLUDE(EQU.INC)
$ INCLUDE(REF1.INC)

        PUBLIC  LADD,LSUB,LMLT,LDIV
        PUBLIC  LADDX,LMLTX

        CSEG

;*************************************************
;*
;*  FLOATING POINT ADDITION FUNCTION
;*
;*    DESTINATION REGISTER: FPR1
;*    SOURCE REGISTER     : FPR2
;*
;*    RESULT : FPR1 += FPR2
;*             ERROR then set CY
;*
;*************************************************
LADD:
        FPR1_X = #0
        FPR2_X = #0

;******** CHECK DIFFERENCE OF EXPORNENT **

LADDX:

        A = FPR1_E
        A -= FPR2_E
        if_bit (!CY)
          if (A >= #SHORT*BYTE || FPR2_E == #0)
            goto T_ADD9
          endif
        else
```

```
      if (A <= #LOW(-(SHORT*BYTE)) || FPR1_E == #0)
        FPR1_X = FPR2_X
        FPR1   = FPR2
        FPR1_M = FPR2_M
        FPR1_H = FPR2_H
        FPR1_E = FPR2_E
        goto T_ADD9
      endif
    endif


;******** LOAD MANTISSA TO GENERAL reg. **

    ;* LHED reg. <- FPR1 *
    ;* FPR1_X    <- A (DIFFERENCE of EXP) *
    A <-> FPR1_X
    A <-> L

    A = FPR1
    A <-> H

    A = FPR1_M
    A <-> E

    A = FPR1_H
    A |= #80H       ;set MSB
    A <-> D

    ;* CBXA reg. <- FPR2 *
    A = FPR2_X
    A <-> C
    A = FPR2
    A <-> B
    A = FPR2_M
    A <-> X
    A = FPR2_H
    A |= #80H         ;set MSB
```

```
;******** BE AGREED MANTISSA POTENTIAL **

        if (FPR1_X != #0)      ;exp. agree?
          if_bit (!FPR1_X.7)    ;destination is higher
            repeat
              SHRW AX,1
              RORC B,1
              RORC C,1
              FPR1_X--
            until_bit (Z)
          else                  ;sourse is higher
            repeat
              SHRW DE,1
              RORC H,1
              RORC L,1
              FPR1_X++
            until_bit (Z)
            FPR1_E = FPR2_E
          endif
        endif

;******** CALC. MANTISSA **

        FPR2_H ^= FPR1_H
        if_bit (!FPR2_H.7)      ;sign agree?
          ADD  L,C
          ADDC H,B
          ADDC E,X
          ADDC D,A          ;CY : mantissa overflow bit
        else
          if (AX == DE)
            if (B == H)
              if (C == L)
                goto ZERO
              endif
            endif
          endif
```

```
          if_bit (!CY)          ; source > destination
            NOT1 FPR1_H.7
            A <-> D
            X <-> E
            B <-> H
            C <-> L
          endif

          SUB  L,C
          SUBC H,B
          SUBC E,X
          SUBC D,A
        endif

;*********************
;*        NORMALIZE **
;*********************
T_NOR:
        A = D

        if_bit (CY)             ; mantissa overflow
          FPR1_E++
          if_bit (Z)
            goto ERROR
          endif
          RORC A,1
          RORC E,1
          RORC H,1
          RORC L,1
        else
          if (FPR1_E == #0)
            goto ZERO
          endif

T_NOR2:
          while_bit (!A.7)      ;catastrophic cancellation
            FPR1_E--
            if_bit (Z)
              goto T_ADD9
            endif
            SHLW HL,1
            ROLC E,1
            ROLC A,1
          endw
```

```
          if (FPR1_E == #0)
            goto ERROR              ; exp=100H
          endif
        endif

        ;* return mantissa *
        if_bit (!FPR1_H.7)
          A &= #7FH
        endif
        FPR1_H = A
        FPR1_M = E (A)
        FPR1   = H (A)
        FPR1_X = L (A)
T_ADD9:
        A = #R_OK
        CLR1 CY
        RET
ZERO:
        FPR1_E = #0
        goto T_ADD9
ERROR:
        A = #R_ERR
        SET1 CY
        RET

;*****************************************************
;*
;*  FLOATING POINT SUBTRACTION FUNCTION
;*
;*    DESTINATION REGISTER: FPR1
;*    SOURCE REGISTER     : FPR2
;*
;*    RESULT : FPR1 -= FPR2
;*             ERROR then set CY
;*
;*****************************************************
LSUB:
        NOT1 FPR2_H.7
        goto LADD
```

```
;*************************************************
;*
;*  FLOATING POINT MULTIPLICATION FUNCTION
;*
;*    DESTINATION REGISTER: FPR1
;*    SOURCE REGISTER     : FPR2
;*
;*    RESULT : FPR1 *= FPR2
;*             ERROR then set CY
;*
;*************************************************
LMLT:
        FPR1_X = #0
        FPR2_X = #0

;******** ZERO EXCEPTION **
LMLTX:
        if (FPR1_E == #0 || FPR2_E == #0)
          goto ZERO
        endif

;******** MULTIPLE EXPORNENT **

        A = FPR1_E
        A += FPR2_E

        if_bit (CY)
          if (A > #ZEROEX)      ;exp. > 100H
            goto ERROR
          endif
          A += #ZEROEX
        else
          A -= #ZEROEX
          if_bit (Z || CY)      ;exp. <= 0
            goto ZERO
          endif
        endif

        FPR1_E = A              ;set expornent

        CY = FPR1_H.7
        CY ^= FPR2_H.7
        FPR1_H.7 = CY           ;set sign
```

```
;******** CALC. MANTISSA (result set to AEHLreg.)**

        SET1 FPR2_H.7           ;set MSB

        ;* FPR1_X *
        A = FPR1_X      ;FPR1_X * FPR2_H
        A <-> B
        A = FPR2_H
        MULU B
        A <-> L


        ;* FPR1   *
        A = FPR1        ;FPR1   * FPR2_M
        A <-> B
        A = FPR2_M
        MULU B
        L += A           ;->CY

        A = FPR2_H      ;FPR1   * FPR2_H
        MULU B
        ADDC A,#0        ;<-CY
        L += X           ;->CY
        ADDC A,#0        ;<-CY
        A <-> H

        ;* FPR1_M *
        A = FPR1_M      ;FPR1_M * FPR2
        A <-> B
        A = FPR2
        MULU B
        L += A           ;->CY

        A = FPR2_M      ;FPR1_M * FPR2_M
        MULU B
        ADDC A,#0        ;<-CY
        L += X           ;->CY
        ADDC H,A         ;<-CY,->CY
```

```
        A = FPR2_H      ;FPR1_M * FPR2_H
        MULU B
        ADDC A,#0       ;<-CY
        H += X          ;->CY
        ADDC A,#0       ;<-CY
        A <-> E

        ;* FPR1_H *
        A = FPR1_H      ;FPR1_H * FPR2_X
        A |= #80H       ;set MSB
        A <-> B
        A = FPR2_X
        MULU B
        L += A          ;->CY

        A = FPR2        ;FPR1_H * FPR2
        MULU B
        ADDC A,#0       ;<-CY
        L += X          ;->CY
        ADDC H,A        ;<-CY,->CY

        A = FPR2_M      ;FPR1_H * FPR2_M
        MULU B
        ADDC A,#0       ;<-CY
        H += X          ;->CY
        ADDC E,A        ;<-CY,->CY


        A = FPR2_H      ;FPR1_H * FPR2_H
        MULU B
        ADDC A,#0       ;<-CY
        E += X          ;->CY
        ADDC A,#0       ;<-CY

;******** NORMALIZE **

        goto T_NOR2
```

```
;*************************************************
;*
;*  FLOATING POINT DIVISION FUNCTION
;*
;*    DESTINATION REGISTER: FPR1
;*    SOURCE REGISTER     : FPR2
;*
;*    RESULT : FPR1 /=FPR2
;*             ERROR then set CY
;*
;*************************************************
LDIV:

;******** ZERO EXCEPTION **

        if (FPR2_E == #0)
          goto ERROR
        endif
        if (FPR1_E == #0)
          goto ZERO
        endif

;******** DIVIDE EXPORNENT **

        A = FPR1_E
        A -= FPR2_E

        if_bit (CY)
          if_bit (!A.7)          ;exp. < 0
            goto ZERO
          endif
        else
          if_bit (A.7)           ;exp. >= 100H
            goto ERROR
          endif
        endif
        A += #ZEROEX

        FPR1_E = A               ;set expornent
```

```
        CY = FPR1_H.7               ;make sign
        CY ^= FPR2_H.7
        FPR1_H.7 = CY               ;set sign

;******** LOAD MANTISSA **

        ;* XBCreg. <- FPR1(1st-3rd mantissa) *
        A = #(SHORT-1)*BYTE+1       ;loop counter
        A <-> FPR1
        A <-> C

        A = FPR1_M
        A <-> B

        A = FPR1_H
        A |= #80H           ;set MSB
        A <-> X

        CLR1 CY

        ;* Lreg. <- FPR2(1st mantissa) *
        A = FPR2_H
        A |= #80H           ;set MSB
        A <-> L

;******** DIVIDE MANTISSA **

        H = #0

        goto T_DIV1

        ;* DEHreg. <- quotient *
        repeat
          SHLW BC,1     ;b*2
          ROLC X,1
```

```
T_DIV1:
          NOT1 CY
          if_bit (CY)
            if (X == L)
              A = B
              if (A == FPR2_M)
                A = C
                CMP A,FPR2
              endif
            endif
          endif


          if_bit (!CY)      ;if d > S
            A = FPR2        ;  then d -= S
            SUB C,A
            A = FPR2_M
            SUBC B,A
            SUBC X,L
            SET1 CY
          else
            CLR1 CY
          endif

          ROLC H,1          ;set quotient
          ROLC E,1
          ROLC D,1          ;CY then mantissa overflow

          FPR1--
        until_bit (Z)

;******** NORMALIZE **

        L = #0          ;clr 4th mantissa

        goto T_NOR

        END
```

(7) LFLT2. SRC

```
$       TITLE     ('FLOATING POINT COMMON FUNCTIONS 1')
        NAME      M_LFLT2

$ INCLUDE(EQU.INC)
$ INCLUDE(REF1.INC)
$ INCLUDE(REF2.INC)

        EXTRN     LADDX,LMLTX

        PUBLIC    LPLY,LPLY2

        CSEG

;*****************************************************************
;*
;*   FLOATING POINT FUNCTION THAT
;*
;*     CALC. A POLYNOMIAL EXPRESSION by EXTENDED
;*
;*                                2                   n
;*  polynomial.1: x + k1xy + k1k2xy  + ... + k1k2..knxy
;*
;*     input  conditions:
;*        FPR1 <- X , FPR4 <- y
;*        DE <- head address of coefficient array (k1,,kn)
;*        C <- n
;*
;*                                  2                 n
;*  polynomial.2: z + k1xy + k1k2xy  + ... + k1k2..knxy
;*
;*     input  conditions:
;*        FPR1 <- x , FPR4 <- y, FPR3 <- z
;*        DE <- head address of coefficient array (k1,,kn)
;*        C <- n
;*
;*     output conditions(common to both):
;*        FPR1 <- result of polynomial expression
;*        FPR4 : keep
;*
;*****************************************************************
```

```
LPLY:
        CALL !LLD31X
        goto LPLY2

        repeat

          CALL !LXC13X

LPLY2:
          CALL !LLD24X

          PUSH BC
          PUSH DE

          CALL !LMLTX

          POP DE
          CALL !LLD2CX
          PUSH DE

          CALL !LMLTX

          CALL !LLD21X
          CALL !LXC13X

          CALL !LADDX

          POP DE
          POP BC

          if (FPR3+SHORT == #0)
            RET
          endif

          A = FPR1_E
          A -= FPR3+SHORT
          if_bit (!CY)
            if (A >= #(SHORT-1)*BYTE)
              RET
            endif
          endif
```

```
  C--
until_bit (Z)
RET

END
```

(8) LLD. SRC

```
$       TITLE   ('FPR LOAD FUNCTIONS')
        NAME    M_LLD

$ INCLUDE(EQU.INC)
$ INCLUDE(REF1.INC)

;*****************************************************
;*
;*  FLOATING POINT REGISTER LOAD FUNCTIONS
;*
;*****************************************************

        PUBLIC  LLD21,LLD21X
        PUBLIC  LLD31,LLD31X
        PUBLIC  LLD41,LLD41X
        PUBLIC  LLD51,LLD51X
        PUBLIC  LLD52
        PUBLIC  LLD13
        PUBLIC  LLD23,LLD23X
        PUBLIC  LLD14,LLD14X
        PUBLIC  LLD24,LLD24X
        PUBLIC  LLD15
        PUBLIC  LLD25,LLD25X
        PUBLIC  LLD1C,LLD1CX
        PUBLIC  LLD2C,LLD2CX

        PUBLIC  LXC13,LXC13X
        PUBLIC        LXC14X
        PUBLIC  LXC15,LXC15X

        CSEG

;******* LOAD FPR2,FPR1

LLD21X:
        FPR2_X = FPR1_X
LLD21:
        FPR2   = FPR1
        FPR2_M = FPR1_M
        FPR2_H = FPR1_H
        FPR2_E = FPR1_E
        RET
```

```
;******* LOAD FPR3,FPR1

LLD31X:
        FPR3   = FPR1_X
LLD31:
        FPR3+1 = FPR1
        FPR3+2 = FPR1_M
        FPR3+3 = FPR1_H
        FPR3+4 = FPR1_E
        RET

;******* LOAD FPR4,FPR1

LLD41X:
        FPR4   = FPR1_X
LLD41:
        FPR4+1 = FPR1
        FPR4+2 = FPR1_M
        FPR4+3 = FPR1_H
        FPR4+4 = FPR1_E
        RET

;******* LOAD FPR5,FPR1

LLD51X:
        FPR5   = FPR1_X
LLD51:
        FPR5+1 = FPR1
        FPR5+2 = FPR1_M
        FPR5+3 = FPR1_H
        FPR5+4 = FPR1_E
        RET

;******* LOAD FPR5,FPR2

LLD52:
        FPR5+1 = FPR2
        FPR5+2 = FPR2_M
        FPR5+3 = FPR2_H
        FPR5+4 = FPR2_E
        RET
```

```
;******* LOAD FPR1,FPR3

LLD13:
        FPR1   = FPR3+1
        FPR1_M = FPR3+2
        FPR1_H = FPR3+3
        FPR1_E = FPR3+4
        RET

;******* LOAD FPR2,FPR3

LLD23X:
        FPR2_X = FPR3
LLD23:
        FPR2   = FPR3+1
        FPR2_M = FPR3+2
        FPR2_H = FPR3+3
        FPR2_E = FPR3+4
        RET

;******* LOAD FPR1,FPR4

LLD14X:
        FPR1_X = FPR4
LLD14:
        FPR1   = FPR4+1
        FPR1_M = FPR4+2
        FPR1_H = FPR4+3
        FPR1_E = FPR4+4
        RET

;******* LOAD FPR2,FPR4

LLD24X:
        FPR2_X = FPR4
LLD24:
        FPR2   = FPR4+1
        FPR2_M = FPR4+2
        FPR2_H = FPR4+3
        FPR2_E = FPR4+4
        RET
```

```
;******* LOAD FPR1,FPR5

LLD15:
        FPR1   = FPR5+1
        FPR1_M = FPR5+2
        FPR1_H = FPR5+3
        FPR1_E = FPR5+4
        RET

;******* LOAD FPR2,FPR5

LLD25X:
        FPR2_X = FPR5
LLD25:
        FPR2   = FPR5+1
        FPR2_M = FPR5+2
        FPR2_H = FPR5+3
        FPR2_E = FPR5+4
        RET
        RET

;******* LOAD FPR1,constant

LLD1CX:
        FPR1_X = [DE+] (A)
LLD1C:
        FPR1   = [DE+] (A)
        FPR1_M = [DE+] (A)
        FPR1_H = [DE+] (A)
        FPR1_E = [DE+] (A)
        RET

;******* LOAD FPR2,constant

LLD2CX:
        FPR2_X = [DE+] (A)
LLD2C:
        FPR2   = [DE+] (A)
        FPR2_M = [DE+] (A)
        FPR2_H = [DE+] (A)
        FPR2_E = [DE+] (A)
        RET
```

```
;******* XCHANGE FPR1,FPR3

LXC13X:
        FPR1_X <-> FPR3
LXC13:
        FPR1   <-> FPR3+1
        FPR1_M <-> FPR3+2
        FPR1_H <-> FPR3+3
        FPR1_E <-> FPR3+4
        RET

;******* XCHANGE FPR1,FPR4

LXC14X:
        FPR1_X <-> FPR4
        FPR1   <-> FPR4+1
        FPR1_M <-> FPR4+2
        FPR1_H <-> FPR4+3
        FPR1_E <-> FPR4+4
        RET

;******* XCHANGE FPR1,FPR5

LXC15X:
        FPR1_X <-> FPR5
LXC15:
        FPR1   <-> FPR5+1
        FPR1_M <-> FPR5+2
        FPR1_H <-> FPR5+3
        FPR1_E <-> FPR5+4
        RET

        END
```

(9) LSIN. SRC

```
$       TITLE     ('SINE FUNCTION')
        NAME      M_LSIN

$ INCLUDE(EQU.INC)
$ INCLUDE(REF1.INC)
$ INCLUDE(REF2.INC)

        EXTRN     LPLY
        EXTRN     LADDX,LMLTX

        EXTRN     FTOL,LTOF

        PUBLIC    LSIN
        PUBLIC    LMOD90,LSIN90

S_PLY   EQU       5

C_1X    EQU       0A2H
C_1L    EQU       0DAH
C_1M    EQU       00FH
C_1H    EQU       049H
C_1E    EQU       081H

        CSEG

;*******************************************************
;*
;*   FLOATING POINT SINE FUNCTION
;*
;*     input  condition : FPR1 <- x
;*
;*     output conditions: FPR1 <- sin(x)
;*
;*******************************************************
LSIN:

;********* SCALING of X **

          CALL !LMOD90
```

```
;********** GET SIN(x' + n * pai/2)  , 0 <= x' < pai/2 **

LSIN90:
          if_bit (FPR4_1.0)    ;mod(n/2)=1?
            SET1 FPR1_H.7
            DE = #C_1
            CALL !LLD2CX
            CALL !LADDX        ; x' <- pai/2 - x'
          endif

          CY = FPR4_1.1        ; mod(n/4)=2 or 3?
          CY ^= FPR4_2.7       ;  then turn sign bit

          FPR1_H.7 = CY        ;set SIGN

;********** CALC. POLYNOMIAL EXPRESSION **

          CALL !LLD41X         ;set x'' to FPR4

          CALL !LLD21X
          CALL !LMLTX          ;x''*x''

          CALL !LXC14X         ;set x''*x'' to FPR4
                               ;set x'' to FPR1
          DE = #C_K
          C = #S_PLY
          CALL !LPLY

          A = #R_OK
          CLR1 CY
          RET
```

```
;*************************************
;**        GET MOD by pai/2 (->FPR1) **
;*************************************

LMOD90:
          FPR1_X = #0
          FPR4_1 = #0              ;quotient keeper
          FPR4_2 = FPR1_H          ;sign keep
          CLR1 FPR1_H.7

          while (forever)
            ;* CMP FPR1,pai/2
               if (FPR1_E == #C_1E)
                 if (FPR1_H == #C_1H)
                   if (FPR1_M == #C_1M)
                     if (FPR1 == #C_1L)
                       CMP FPR1_X,#C_1X
                     endif
                   endif
                 endif
               endif

               if_bit (CY)
                 break
               endif

               CALL !LLD31X        ;esc. x' to FPR3

               ;* *2/pai *
               DE = #C_2
               CALL !LLD2CX
               CALL !LMLTX         ;x' / pai/2
               FPR1_X  = #0
               FPR1    = #0
               FPR1_M &= #0FCH     ;valid digit → 14

               ;* FPR1 -> integer *
               CALL !LTOF
               if_bit (!CY)
                 if_bit (!Z)
                   CALL !FTOL
                 endif
```

```
    A = E
    A += FPR4_1     ;add last 2bit of quotient
    FPR4_1 = A
  endif

  DE = #C_1
  CALL !LLD2CX
  CALL !LMLTX          ; int(x' / pai/2) * pai/2

  SET1 FPR1_H.7
  CALL !LLD23X
  CALL !LADDX          ; x' - int(x' / pai/2) * pai/2
endw

RET

C_1:
        DB C_1X,C_1L,C_1M,C_1H,C_1E ; const pai/2
C_2:
        DB 06EH,083H,0F9H,022H,080H ;       2/pai

C_K:                                ;coefficient array for LPLY
        DB 0AAH,0AAH,0AAH,0AAH,07EH ;const -1/6
        DB 0CCH,0CCH,0CCH,0CCH,07CH ;      -1/20
        DB 0C3H,030H,00CH,0C3H,07BH ;      -1/42
        DB 0E3H,038H,08EH,0E3H,07AH ;      -1/72
        DB 04FH,009H,0F2H,094H,07AH ;      -1/110

        END
```

(10) LCOS. SRC

```
$       TITLE     ('COSINE FUNCTION')
        NAME      M_LCOS

$ INCLUDE(EQU.INC)
$ INCLUDE(REF1.INC)

        EXTRN     LMOD90,LSIN90

        PUBLIC    LCOS,LCOS90

        CSEG
;*********************************************************
;*
;*   FLOATING POINT COSINE FUNCTION
;*
;*     input  condition : FPR1 <- x
;*
;*     output conditions: FPR1 <- cos(x)
;*
;*********************************************************
LCOS:

;********* SCALING of X **

          CALL !LMOD90

;********* GET COS(X' + n * pai/2)  , -pai/2 < x' < pai/2 **

LCOS90:
          CLR1 FPR4_2.7   ; x' <- |x'|
          FPR4_1++        ; n <- n+1
          goto LSIN90

          END
```

(11) LTAN. SRC

```
$       TITLE     ('TANGENT FUNCTION')
        NAME      M_LTAN

$ INCLUDE(EQU.INC)
$ INCLUDE(REF1.INC)
$ INCLUDE(REF2.INC)

        EXTRN     LDIV
        EXTRN     LMOD90,LSIN90,LCOS90

        PUBLIC    LTAN

        CSEG

;*********************************************************
;*
;*   FLOATING POINT TANGENT FUNCTION
;*
;*     input  condition : FPR1 <- x
;*
;*     output conditions: FPR1 <- tan(x)
;*                        ERROR then set CY
;*
;*********************************************************
LTAN:

          CALL !LMOD90        ;get mod by pai/2

          A = FPR4_2
          A <-> X             ;X: esc. sign bit
          A = FPR4_1          ;A: esc. quotient by pai/2
          PUSH AX

          CALL !LLD51X        ;esc. mod by pai/2 to FPR5

          CALL !LCOS90        ;cos(x)
          POP AX

          FPR4_1 = A
          FPR4_2 = X (A)

          CALL !LXC15X        ;esc. cos(x) to FPR5
          CALL !LSIN90        ;sin(x)

          CALL !LLD25         ;ret. cos(x) to FPR2
          goto LDIV           ;sin(x)/cos(x)

          END
```

(12) LLOG. SRC

```
$       TITLE       ('LOGARITHMIC FUNCTION')
        NAME        M_LLOG

$ INCLUDE(EQU.INC)
$ INCLUDE(REF1.INC)
$ INCLUDE(REF2.INC)

        EXTRN       LADD,LADDX,LMLT,LMLTX,LDIV
        EXTRN       LPLY2
        EXTRN       FTOL

        PUBLIC      LLOG

S_PLY   EQU         6

C_2L    EQU         000H
C_2M    EQU         000H
C_2H    EQU         000H
C_2E    EQU         081H

        CSEG

;*********************************************************
;*
;*   FLOATING POINT LOGARITHMIC FUNCTION
;*
;*     input  condition : FPR1 <- x
;*
;*     output conditions: FPR1 <- log(x)
;*                        ERROR then set CY
;*
;*********************************************************
LLOG:
          FPR1_X = #0
```

```
;********* EXCEPTION **

          if (FPR1_E == #0)
            goto ERROR
          endif
          if_bit (FPR1_H.7)
            goto ERROR
          endif

          ;* FPR1 = 1? *
          if (FPR1_E == #C_2E)
            if (FPR1_H == #C_2H)
              if (FPR1_M == #C_2M)
                if (FPR1 == #C_2L)
                  FPR1_E = #0
                  goto T_LOG9
                endif
              endif
            endif
          endif

;********* CALC. EXP.PART LOG **

          A = #ZEROEX
          A <-> FPR1_E
          CALL !LLD31           ;esc. mantissa(Xf) to FPR3

          A -= #ZEROEX
          if_bit (!CY)
            D = #0
          else
            D = #0FFH
          endif
          A <->E                ;DE: integer value of exp.part
          CALL !FTOL            ;real value of exp.part

          DE = #C_1
          CALL !LLD2CX
          CALL !LMLTX           ;Xe * log(2)
          CALL !LLD41X          ;esc. exp.part LOG to FPR4
```

```
;********** TRANS. MANTISSA FOR TAYLOR APPROXIMATE **

          CALL !LLD13
          DE = #C_2
          CALL !LLD2C
          CALL !LADD                    ; Xf+1

          CALL !LXC13
          DE = #C_2
          CALL !LLD2C
          SET1 FPR2_H.7
          CALL !LADD                    ;Xf-1

          CALL !LLD23
          CALL !LDIV                    ;(Xf-1)/(Xf+1)  :x'

;********** CALC. MANTISSA LOG **

          CALL !LLD31X          ;esc X' to FOR3
          FPR3+4++              ;set 2X' to FPR3

          CALL !LLD21X
          CALL !LMLT            ; X'*X'
          CALL !LXC14X          ;set X'*X' to FPR4


          CALL !LLD23X
          CALL !LADDX           ;set Xe*log(2)+2X' to FPR1
          CALL !LXC13X

          DE = #C_K
          C = #S_PLY
          CALL !LPLY2

T_LOG9:
          A = #R_OK
          CLR1 CY
          RET
ERROR:
          A = #R_ERR
          SET1 CY
          RET
```

```
C_1:
        DB 0F7H,017H,072H,031H,080H ; const log(2)
C_2:
        DB      C_2L,C_2M,C_2H,C_2E ;       1

C_K:                                ; coefficient array of LPLY
        DB 0AAH,0AAH,0AAH,02AH,07FH ; const 1/3
        DB 099H,099H,099H,019H,080H ;       3/5
        DB 0B6H,06DH,0DBH,036H,080H ;       5/7
        DB 0C7H,071H,01CH,047H,080H ;       7/9
        DB 017H,05DH,074H,051H,080H ;       9/11
        DB 0D8H,089H,09DH,058H,080H ;      11/13

        END
```

(13) LLOG10. SRC

```
$       TITLE     ('LOGARITHMIC FUNCTION 2')
        NAME      M_LLOG10

$ INCLUDE(EQU.INC)
$ INCLUDE(REF1.INC)
$ INCLUDE(REF2.INC)

        EXTRN     LMLTX
        EXTRN     LLOG

        PUBLIC    LLOG10

        CSEG

;*******************************************************
;*
;*   FLOATING POINT LOGARITHMIC FUNCTION 2
;*
;*     input  condition : FPR1 <- x
;*
;*     output conditions: FPR1 <- log10(x)
;*                        ERROR then set CY
;*
;*******************************************************
LLOG10:

;********* log(x) / log(10) **

          CALL !LLOG
          if_bit (CY)
            RET
          endif

          DE = #C_1
          CALL !LLD2CX
          goto LMLTX

C_1:
          DB 0A9H,0D8H,05BH,05EH,07FH ; const 1/log(10)

          END
```

(14) LEXP. SRC

```
$       TITLE     ('EXPONENTIAL FUNCTION')
        NAME      M_LEXP

$ INCLUDE(EQU.INC)
$ INCLUDE(REF1.INC)
$ INCLUDE(REF2.INC)

        EXTRN     LADDX,LMLTX
        EXTRN     LPLY2
        EXTRN     LTOF,FTOL

        PUBLIC    LEXP,LEXPX

S_PLY   EQU       8

        CSEG

;*********************************************************
;*
;*   FLOATING POINT EXPONENTIAL FUNCTION
;*
;*     input  condition : FPR1 <- x
;*
;*     output conditions: FPR1 <- exp(x)
;*                        ERROR then set CY
;*
;*********************************************************
LEXP:
          FPR1_X = #0

;********* TRANSLATE to EXP2(X) **
LEXPX:
          FPR3_1 = FPR1_H      ;esc sign bit

          DE = #C_1
          CALL !LLD2CX
          CALL !LMLTX          ;1/log2*X
          if_bit (!CY)
            CALL !LTOF
          endif
          if_bit (CY)
            goto T_FLOW
          endif
```

```
;********* CALC. EXP **

          if_bit (!Z && FPR3_1.7)
            DE--                ;DE <- floor integer(X/log2)
          endif

          AX = DE
          AX += #ZEROEX+1       ;AX <- upper integer(X/log2)

          if (A != #0)
            goto T_FLOW
          endif

          FPR5_1 = X (A)        ;esc exp

;********* CALC. MANTISSA **

          CALL !LLD21X
          CALL !FTOL            ; floor integer(X/log2)
          NOT1 FPR1_H.7
          CALL !LADDX           ; X/log2 -(floor integer(X/log2))

          CALL !LLD41X          ;set X'

          DE = #C_2
          CALL !LLD2CX
          CALL !LMLTX           ;log2·X'

          CALL !LLD31X          ;set log2·X'

          DE = #C_3
          CALL !LLD2CX
          CALL !LADDX           ;1+log2·X'

          CALL !LXC13X

          DE = #C_K
          C = #S_PLY
          CALL !LPLY2
```

```
        if (FPR1_E == #80H)   ;if cancellation happen then
          if (FPR5_1 != #0)   ;
            FPR5_1--          ;  adjust expornent
          endif
        endif

;********* RETURN EXP.PART **

        FPR1_E = FPR5_1
T_EXP9:
        A = #R_OK
        CLR1 CY
        RET
T_FLOW:
        if_bit (FPR3_1.7)
          FPR1_E = #0
          goto T_EXP9
        endif

        A = #R_ERR
        SET1 CY
        RET

C_1:
        DB 029H,03BH,0AAH,038H,081H  ; const  1/log2
C_2:
        DB 0F7H,017H,072H,031H,080H  ;        log2
C_3:
        DB 000H,000H,000H,000H,081H  ;        1

C_K:                                 ; coefficient array of LPLY2
        DB 0F7H,017H,072H,031H,07FH  ; const log2/2
        DB 0F5H,01FH,098H,06CH,07EH  ;       log2/3
        DB 0F7H,017H,072H,031H,07EH  ;       log2/4
        DB 0F9H,0DFH,0F4H,00DH,07EH  ;       log2/5
        DB 0F5H,01FH,098H,06CH,07DH  ;       log2/6
        DB 01BH,089H,0CBH,04AH,07DH  ;       log2/7
        DB 0F7H,017H,072H,031H,07DH  ;       log2/8
        DB 0F8H,0BFH,0BAH,01DH,07DH  ;       log2/9

        END
```

(15) LEXP10. SRC

```
$       TITLE     ('EXPONENTIAL FUNCTION 2')
        NAME      M_LEXP10

$ INCLUDE(EQU.INC)
$ INCLUDE(REF1.INC)
$ INCLUDE(REF2.INC)

        EXTRN     LMLTX
        EXTRN     LEXPX

        PUBLIC    LEXP10

        CSEG

;*********************************************************
;*
;*   FLOATING POINT EXPONENTIAL FUNCTION 2
;*
;*     input  condition : FPR1 <- x
;*
;*     output conditions: FPR1 <- exp10(x)
;*                        ERROR then set CY
;*
;*********************************************************
LEXP10:
         FPR1_X = #0

;********* TRANSLATE EXP10 TO EXP **

         FPR3_1 = FPR1_H

         DE = #C_1
         CALL !LLD2CX
         CALL !LMLTX
         if_bit (CY)
           if_bit (FPR3_1.7)
             FPR1_E = #0
             A = #R_OK
             CLR1 CY
           endif
           RET
         endif
```

```
;********* CALC. exp(X*log(10)) **

          goto LEXPX

C_1:
          DB ODDH,08DH,05DH,013H,082H ;const log(10)

          END
```

(16) LPOW. SRC

```
$       TITLE     ('POWER FUNCTION')
        NAME      M_LPOW

$ INCLUDE(EQU.INC)
$ INCLUDE(REF1.INC)
$ INCLUDE(REF2.INC)

        EXTRN     LMLTX
        EXTRN     LLOG,LEXPX

        PUBLIC    LPOW

        CSEG

;*********************************************************
;*
;*   FLOATING POINT POWER FUNCTION
;*
;*     input  condition : FPR1 <- a , FPR2 <- b
;*
;*     output conditions: FPR1 <- a^b
;*                        ERROR then set CY
;*
;*********************************************************
LPOW:

;********* a=0 EXCEPTION **

          if (FPR1_E == #0)
            CMP FPR2_E,#0
            if_bit (Z || FPR2_H.7)
              goto ERROR          ;0^0,0^(negative)=overflow
            else
              FPR1_E = #0
              goto T_POW9         ;0^(positive)=0
            endif
          endif
```

```
;********* a<0 EXCEPTION **

          X = #0                ;X.0 :sign of result
          if_bit (FPR1_H.7)
            A = FPR2_E
            if (A == #0)
              DE = #C_1
              CALL !LLD1C
              goto T_POW9           ;x^0=1
            endif

;**      ** b: DECIMAL PART = 0 ? **

            if (A <= #ZEROEX)
              goto ERROR            ;(negative)^(decimal)=error
            endif

            A <-> X

            A = FPR2_H
            A |= #80H
            A <-> B
            A = FPR2_M
            A <-> C
            A = FPR2
            A <-> X

            A -= #ZEROEX+BYTE*(SHORT-1)
            if_bit (CY)
              repeat
                SHRW BC,1
                RORC X,1
                if_bit (CY)
                  goto ERROR    ;include decimal digit
                endif
                A++
              until_bit (Z)
            endif
```

```
;**     ** b: ODD or EVEN ? **

          if_bit (!Z)          ;not include UNIT1
            X = #0
          endif
        endif

        PUSH AX                ;esc UNIT1(X.0)

;********* CALC. exp(b * log(|a|) **

        CALL !LLD52          ;esc b to FPR5
        CLR1 FPR1_H.7
        CALL !LLOG           ;log(|a|)
        CALL !LLD25          ;ret. b to FPR2
        FPR2_X = #0
        CALL !LMLTX          ;b * log(|a|)

        if_bit (CY)          ;overflow
          POP BC
          if_bit (FPR5+(SHORT-1).7)
            FPR1_E = #0
            goto T_POW9
          endif
          RET
        endif

        CALL !LEXPX

;********* RETURN SIGN BIT **

        POP BC
        X = C

        if_bit (X.0)         ;if b = odd integer & a<0
          SET1 FPR1_H.7      ; then set sign bit
        endif

        RET
```

```
T_POW9:
        A = #R_OK
        CLR1 CY
        RET
ERROR:
        A = #R_ERR
        SET1 CY
        RET

C_1:
        DB 000H,000H,000H,081H ;const 1

        END
```

(17) LSQRT. SRC

```
$       TITLE       ('SQUARE ROOT FUNCTION')
        NAME        M_LSQRT

$ INCLUDE(EQU.INC)
$ INCLUDE(REF1.INC)
$ INCLUDE(REF2.INC)

        EXTRN       LADD,LDIV

        PUBLIC      LSQRT

C_LIM   EQU         5       ;limiter of approximate

        CSEG

;*********************************************************
;*
;*  FLOATING POINT SQUARE ROOT FUNCTION
;*
;*   input  condition : FPR1 <- x
;*
;*   output conditions: FPR1 <- root(x)
;*                      ERROR then set CY
;*
;*********************************************************
LSQRT:

;********* EXCEPTION **

        if (FPR1_E == #0)
          goto T_QRT9
        endif
        if_bit (FPR1_H.7)
          goto ERROR
        endif
```

```
;********* GET EXP.PART ROOT **

          A = FPR1_E
          A++
          CY = PSW.6  ; CY <- Z
          RORC A,1
          A += #ZEROEX/2

          FPR4_1 = A              ;escape exp.part root

;*********  a -> r/2 (.125 - .5) **

          if_bit (FPR1_E.0)
            FPR1_E = #ZEROEX-2
          else
            FPR1_E = #ZEROEX-1
          endif

;********* CALC. VIRT.PART ROOT **

          CALL !LLD41         ;esc. r/2 to FPR4

          DE = #C_1
          CALL !LLD2C
          CALL !LADD        ; r/2 + .5  : 2ndary approximate

          FPR3_1 = #C_LIM-1
          repeat
            CALL !LLD31     ;esc.previous approximate(Rp) to FPR3
            CALL !LLD14
            CALL !LLD23
            CALL !LDIV      ; r/2 / Rp

            CALL !LLD23
            FPR2_E--        ; Rp/2
            CALL !LADD      ; Rp/2 + R/(2Rp) : next approximate

            if (FPR1_E == FPR3+4)
              if (FPR1_H == FPR3+3)
                if (FPR1_M == FPR3+2)
                  CMP FPR1,FPR3+1
                endif
              endif
            endif
```

```
            if_bit (!CY)
              break
            endif

            FPR3_1--
          until_bit (Z)

          FPR1_E = FPR4_1        ;ret. exp part ROOT
T_QRT9:
          A = #R_OK
          CLR1 CY
          RET
ERROR:
          A = #R_ERR
          SET1 CY
          RET

C_1:
          DB 000H,000H,000H,080H ;const .5

          END
```

(18) LASIN. SRC

```
$       TITLE     ('ARCSINE FUNCTION')
        NAME      M_LASIN

$ INCLUDE(EQU.INC)
$ INCLUDE(REF1.INC)
$ INCLUDE(REF2.INC)

        EXTRN     LMLT,LDIV
        EXTRN     LADDX
        EXTRN     LSQRT,LATAN

        PUBLIC    LASIN

C_1X    EQU       000H
C_1L    EQU       000H
C_1M    EQU       000H
C_1H    EQU       000H
C_1E    EQU       081H

        CSEG

;************************************************
;*
;*   FLOATING POINT ARCSINE FUNCTION
;*
;*     input  condition : FPR1 <- x
;*
;*     output conditions: FPR1 <- arcsin(x)
;*                        ERROR then set CY
;*
;************************************************
LASIN:

;******** EXCEPTION **

          if (FPR1_E == #0)
            goto T_SIN9
          endif

          CALL !LLD51          ; esc X to FPR5

          CLR1 FPR1_H.7        ; x <- |x|
```

```
        ;* |x| = 1? *
        if (FPR1_E == #C_1E)
          if (FPR1_H == #C_1H)
            if (FPR1_M == #C_1M)
              if (FPR1 == #C_1L)
                DE = #C_2
                CALL !LLD1CX
                if_bit (FPR5+1+2.7) ; X < 0 ?
                  SET1 FPR1_H.7
                endif
                goto T_SIN9
              endif
            endif
          endif
        endif

        if_bit (!CY)            ; |x| > 1
          A = #R_ERR            ; then error
          SET1 CY
          RET
        endif

;******** TRANS. to ARCTAN  **

        CALL !LLD21
        CALL !LMLT          ;X*X
        SET1 FPR1_H.7       ;-X*X

        DE = #C_1
        CALL !LLD2CX
        CALL !LADDX         ;1-X*X
        CALL !LSQRT         ;root(1-X*X)

        CALL !LLD21
        CALL !LLD15
        CALL !LDIV          ;X/root(1-X*X)

        goto LATAN

T_SIN9:
        A = #R_OK
        CLR1 CY
        RET
```

```
C_1:
        DB C_1X,C_1L,C_1M,C_1H,C_1E ;const 1
C_2:
        DB 0A2H,0DAH,00FH,049H,081H ;       pai/2

        END
```

(19) LACOS. SRC

```
$       TITLE       ('ARCCOSINE FUNCTION')
        NAME        M_LACOS

$ INCLUDE(EQU.INC)
$ INCLUDE(REF1.INC)
$ INCLUDE(REF2.INC)

        EXTRN       LADDX
        EXTRN       LASIN

        PUBLIC      LACOS

        CSEG

;*************************************************************
;*
;*   FLOATING POINT ARCCOSINE FUNCTION
;*
;*     input  condition : FPR1 <- x
;*
;*     output conditions: FPR1 <- arccos(x)
;*                        ERROR then set CY
;*
;*************************************************************
LACOS:

        CALL !LASIN
        if_bit (CY)
          RET
        endif

        NOT1 FPR1_H.7

        DE = #C_1
        CALL !LLD2CX
        goto LADDX      ;pai/2 -sin(x)

C_1:
        DB 0A2H,0DAH,00FH,049H,081H ;const pai/2

        END
```

(20) LATAN. SRC

```
$       TITLE       ('ARCTANGENT FUNCTION')
        NAME        M_LATAN

$ INCLUDE(EQU.INC)
$ INCLUDE(REF1.INC)
$ INCLUDE(REF2.INC)

        EXTRN       LADD,LDIV
        EXTRN       LADDX,LMLTX
        EXTRN       LPLY2

        PUBLIC      LATAN

S_PLY   EQU         7

        CSEG

;*********************************************************
;*
;*   FLOATING POINT ARCTANGENT FUNCTION
;*
;*     input  condition : FPR1 <- x
;*
;*     output conditions: FPR1 <- arctan(x)
;*
;*********************************************************
LATAN:
         FPR1_X = #0

;******** ZERO EXCEPTION **

         if (FPR1_E == #0)
           goto T_TAN9
         endif
```

```
;******** ESCAPE SIGN & EXPORNENT **

        CMP FPR1_E,#ZEROEX+1    ;CY(if |X|<1)

        A = FPR1_H
        A &= #80H               ;Z (rev. sign bit)

        PUSH PSW

        CLR1 FPR1_H.7

;******** TRANS. X TO (|X|-1)/(|X|+1) case if |X| >= 1 **

        if_bit (!CY)
          CALL !LLD31

          DE = #C_1
          CALL !LLD2C
          CALL !LADD
          CALL !LXC13       ;esc. |X|+1 to FPR3

          DE = #C_1
          CALL !LLD2C
          SET1 FPR2_H.7
          CALL !LADD

          CALL !LLD23       ;ret. |X|+1
          CALL !LDIV        ;(|X|-1)/(|X|+1) : X'
        endif

;******** CALC POLINOMIAL FUNCTION **

        CALL !LLD41X      ;esc. x' to FPR4
        CALL !LLD21X

        CALL !LMLTX       ; x'*x'
        CALL !LXC14X      ;set x'*x' to FPR4
```

```
        CALL !LLD31X      ;esc. x' to FPR3
        DE = #C_K
        CALL !LLD2CX
        PUSH DE
        CALL !LMLTX       ; A0*x'
        POP DE

        CALL !LXC13X      ;set A0*x' to FPR3
                          ;set x' to FPR1
        C = #S_PLY
        CALL !LPLY2

;******** CALC. APPROXIMATE **

        POP PSW
        PUSH PSW

        if_bit (!CY)      ; X >= 1 ?
          DE = #C_2
          CALL !LLD2CX
          CALL !LADDX     ; pai/4 + arctan(x')
        endif

;******** RETURN SIGN BIT **

        POP PSW
        if_bit (!Z)       ; X < 0 ?
          SET1 FPR1_H.7
        endif
T_TAN9:
        A = #R_OK
        CLR1 CY
        RET

C_1:
        DB      000H,000H,000H,081H ;const 1
C_2:
        DB 0A2H,0DAH,00FH,049H,080H ;       pai/4
```

```
C_K:                                    ;coefficient array(An)
                                        ;     of Hastings approximate
        DB 0CEH,0F4H,0FFH,07FH,080H ;cof. A0
        DB 0E2H,01BH,0A6H,0AAH,07FH ;     A1
        DB 0B0H,093H,034H,099H,080H ;     A2/A1
        DB 03DH,0A4H,081H,0B2H,080H ;     A3/A2
        DB 0B7H,06CH,078H,0B1H,080H ;     A4/A3
        DB 097H,08AH,071H,094H,080H ;     A5/A4
        DB 0FBH,03CH,032H,0C8H,07FH ;     A6/A5
        DB 0BAH,052H,0E5H,0BDH,07EH ;     A7/A6

        END
```

(21) LHSIN. SRC

```
$       TITLE     ('HYPERBOLICSINE FUNCTION')
        NAME      M_LHSIN

$ INCLUDE(EQU.INC)
$ INCLUDE(REF1.INC)
$ INCLUDE(REF2.INC)

        EXTRN     LADD,LMLT
        EXTRN     LPLY
        EXTRN     LEXP,LRCPN

        PUBLIC    LHSIN

S_PLY   EQU       3

        CSEG

;*********************************************************
;*
;*   FLOATING POINT HYPERBOLICSINE FUNCTION
;*
;*     input  condition : FPR1 <- x
;*
;*     output conditions: FPR1 <- sinh(x)
;*                        ERROR then set CY
;*
;*********************************************************
LHSIN:

;******** CALC. (exp(|X|)-exp(-|X|))/2  case if |X| >= 0.5 **

          if_bit (FPR1_E.7)  ; |X| >= 0.5

            A = FPR1_H
            PUSH AX          ;esc sign bit

            CLR1 FPR1_H.7    ; x <- |x|
```

```
          CALL !LEXP        ; exp(|x|)
          if_bit (CY)
            POP BC
            RET             ;overflow
          endif

          CALL !LLD31       ;esc. exp(|x|) to FPR3
          CALL !LRCPN       ; exp(-|x|)

          SET1 FPR1_H.7
          CALL !LLD23
          CALL !LADD        ; exp(|x|)-exp(-|x|)
          FPR1_E--          ; (exp(|x|)-exp(-|x|))/2

          POP AX
          if_bit (A.7)
            SET1 FPR1_H.7
          endif

;******** CALC. DIRECT APPROXIMATE      case if |X| <  0.5 **

        else                ; |X| < 0.5

          CALL !LLD41

          CALL !LLD21
          CALL !LMLT        ;X*X

          CALL !LXC14X
          FPR1_X = #0

          DE = #C_K
          C  = #S_PLY
          CALL !LPLY
        endif

        A = #R_OK
        CLR1 CY
        RET
```

```
C_K:                                ; coefficient array of LPLY
        DB 0AAH,0AAH,0AAH,02AH,07EH ; const 1/6
        DB 0CCH,0CCH,0CCH,04CH,07CH ;       1/20
        DB 0C3H,030H,00CH,043H,07BH ;       1/42

        END
```

(22) LHCOS. SRC

```
$       TITLE     ('HYPERBOLICCOSINE FUNCTION')
        NAME      M_LHCOS

$ INCLUDE(EQU.INC)
$ INCLUDE(REF1.INC)
$ INCLUDE(REF2.INC)

        EXTRN     LADD
        EXTRN     LRCPN
        EXTRN     LEXP

        PUBLIC    LHCOS

        CSEG

;*********************************************************
;*
;*   FLOATING POINT HYPERBOLICCOSINE FUNCTION
;*
;*     input  condition : FPR1 <- x
;*
;*     output conditions: FPR1 <- cosh(x)
;*                        ERROR then set CY
;*
;*********************************************************
LHCOS:

        CLR1 FPR1_H.7

;******** CALC. (exp(|X|)+exp(-|X|))/2 **

        CALL !LEXP        ; exp(|x|)
        if_bit (CY)
          RET             ;overflow
        endif

        CALL !LLD31
        CALL !LRCPN       ; exp(-|x|)

        CALL !LLD23
        CALL !LADD        ; exp(|x|)+exp(-|x|)
        FPR1_E--          ; (exp(|x|)+exp(-|x|))/2
        RET

        END
```

(23) LHTAN. SRC

```
$       TITLE     ('HYPERBOLICTANGENT FUNCTION')
        NAME      M_LHTAN

$ INCLUDE(EQU.INC)
$ INCLUDE(REF1.INC)
$ INCLUDE(REF2.INC)

        EXTRN     LDIV
        EXTRN     LHSIN,LHCOS

        PUBLIC    LHTAN

        CSEG

;*******************************************************
;*
;*   FLOATING POINT HYPERBOLICTANGENT FUNCTION
;*
;*     input  condition : FPR1 <- x
;*
;*     output conditions: FPR1 <- tanh(x)
;*
;*******************************************************
LHTAN:

        CALL !LLD51         ; esc X to FPR5

;******** CALC. LHCOS **

        CALL !LHCOS
        if_bit (CY)
          DE = #C_1
          CALL !LLD1C       ; tanh(positive infinity)=1
          if_bit (FPR5+1+2.7)
            SET1 FPR1_H.7   ; tanh(negative infinity)=-1
          endif
          A = #R_OK
          CLR1 CY
          RET
        endif

        CALL !LXC15         ; esc cosh(X) to FPR5
```

(23) LHTAN. SRC

```
$       TITLE     ('HYPERBOLICTANGENT FUNCTION')
        NAME      M_LHTAN

$ INCLUDE(EQU.INC)
$ INCLUDE(REF1.INC)
$ INCLUDE(REF2.INC)

        EXTRN     LDIV
        EXTRN     LHSIN,LHCOS

        PUBLIC    LHTAN

        CSEG

;*********************************************************
;*
;*   FLOATING POINT HYPERBOLICTANGENT FUNCTION
;*
;*     input  condition : FPR1 <- x
;*
;*     output conditions: FPR1 <- tanh(x)
;*
;*********************************************************
LHTAN:

        CALL !LLD51         ; esc X to FPR5

;******** CALC. LHCOS **

        CALL !LHCOS
        if_bit (CY)
          DE = #C_1
          CALL !LLD1C       ; tanh(positive infinity)=1
          if_bit (FPR5+1+2.7)
            SET1 FPR1_H.7   ; tanh(negative infinity)=-1
          endif
          A = #R_OK
          CLR1 CY
          RET
        endif

        CALL !LXC15         ; esc cosh(X) to FPR5
```

```
;******** CALC. LHSIN **

        CALL !LHSIN         ; sinh(X)

;******** CALC. LHTAN **

        CALL !LLD25         ; ret cosh(X)
        goto  LDIV

C_1:
        DB 000H,000H,000H,081H ; const 1

        END
```

(24) LABS. SRC

```
$       TITLE       ('ABSOLUTE FUNCTION')
        NAME        M_LABS

$ INCLUDE(EQU.INC)
$ INCLUDE(REF1.INC)

        PUBLIC      LABS

        CSEG

;*********************************************************
;*
;*   FLOATING POINT ABSOLUTE FUNCTION
;*
;*     input  condition : FPR1 <- x
;*
;*     output conditions: FPR1 <-  |x|
;*
;*********************************************************
LABS:
        CLR1 FPR1_H.7

        A = #R_OK
        CLR1 CY
        RET

        END
```

(25) LRCPN. SRC

```
$       TITLE     ('RECIPROCAL NUMBER FUNCTION')
        NAME      M_LRCPN

$ INCLUDE(EQU.INC)
$ INCLUDE(REF1.INC)
$ INCLUDE(REF2.INC)

        EXTRN     LDIV

        PUBLIC    LRCPN

        CSEG

;*************************************************************
;*
;*   FLOATING POINT FUNCTION THAT
;*                GET RECIPROCAL NUMBER
;*
;*     input  condition : FPR1 <- x
;*
;*     output conditions: FPR1 <- 1/x
;*                        ERROR then set CY
;*
;*************************************************************
LRCPN:
        CALL !LLD21
        DE = #C_1
        CALL !LLD1C

        goto LDIV

C_1:
        DB 000H,000H,000H,081H ;const 1

        END
```

(26) POTORA. SRC

```
$       TITLE       ('TRANS. TO RIGHT ANGLE COORDINATES')
        NAME        M_POTORA

$ INCLUDE(EQU.INC)
$ INCLUDE(REF1.INC)
$ INCLUDE(REF2.INC)

        EXTRN       LMLT
        EXTRN       LMOD90,LSIN90,LCOS90

        PUBLIC      POTORA

        CSEG

;*********************************************************
;*
;*   FLOATING POINT FUNCTION THAT
;*      TRANS. COORDINATES FROM POLE TO RIGHT ANGLE
;*
;*     input  condition : FPR1 <- r, FPR2 <- sheeta
;*
;*     output conditions: FPR1 <- X, FPR2 <- Y
;*                        ERROR then set CY
;*
;*********************************************************
POTORA:

;********* EXCEPTION **

          if (FPR1_E == #0)     ; r=0
            FPR2_E = #0         ;  then (x,y)=(0,0)
            goto T_ORA9
          endif

          if_bit (FPR1_H.7)     ; r<0 then error
            A = #R_ERR
            SET1 CY
            RET
          endif
```

```
        CALL !LLD23
        CALL !LMLT          ;Y

        CALL !LXC15         ;esc Y & set cos(sheeta) to FPR1
        CALL !LLD23         ;        set r           to FPR2
        CALL !LMLT          ;X

        CALL !LLD25         ;ret Y
T_ORA9:
        A = #R_OK
        CLR1 CY
        RET

        END
```

(27) RATOPO. SRC

```
$       TITLE       ('TRANS. TO POLE COORDINATES')
        NAME        M_RATOPO

$ INCLUDE(EQU.INC)
$ INCLUDE(REF1.INC)
$ INCLUDE(REF2.INC)

        EXTRN       LMLT,LDIV
        EXTRN       LADDX
        EXTRN       LSQRT,LATAN

        PUBLIC      RATOPO

        CSEG

;*********************************************************
;*
;*   FLOATING POINT FUNCTION THAT
;*       TRANS. COORDINATES FROM RIGHT ANGLE TO POLE
;*
;*     input  condition : FPR1 <- X, FPR2 <- Y
;*
;*     output conditions: FPR1 <- r, FPR2 <- sheeta
;*                        ERROR then set CY
;*
;*********************************************************
RATOPO:

;********* Y=0 EXCEPTION & KEEP SIGN BIT**

          if (FPR2_E == #0)
            if (FPR1_E == #0)
              goto T_OP09
            endif
            CLR1 FPR5_1.1
          else
            CY = FPR2_H.7
            FPR5_1.1 = CY   ;esc sign of Y to FPR5_1.1
          endif

          CY = FPR1_H.7
          FPR5_1.0 = CY     ;esc sign of X to FPR5_1.0
```

```
;********* CALC. X*X+Y*Y **

          CALL !LLD41        ;esc. x to FPR4
          CALL !LLD52        ;esc. y to FPR5

          CALL !LLD21
          CALL !LMLT         ; x*x
          if_bit (CY)
            RET
          endif

          CALL !LLD31X       ;esc. x*x to FPR3

          CALL !LLD15
          CALL !LLD21
          CALL !LMLT         ; y*y
          if_bit (CY)
            RET
          endif

          CALL !LLD23X
          CALL !LADDX        ; x*x + y*y
          if_bit (CY)
            RET
          endif

;********* CALC. Y/X **

          CALL !LXC15        ;esc. X*X+Y*Y to FPR5
          CALL !LLD24

          CALL !LDIV                ; y/x
          if_bit (CY)
            DE = #C_1
            CALL !LLD1C             ; sheeta=pai/2
            if_bit (FPR5_1.1)
              SET1 FPR1_H.7         ; sheeta=-pai/2
            endif
            goto T_OP08
          endif
```

```
;********* CALC. sheeta **

          CALL !LATAN

          if_bit (FPR5_1.0)       ; x<0 ?
            DE = #C_2
            CALL !LLD2CX          ; sheeta = arctan(y/x) + pai
            if_bit (FPR5_1.1)
              SET1 FPR2_H.7       ; sheeta = arctan(y/x) - pai
            endif
            CALL !LADDX
          endif

;********* CALC. r **

T_OP08:
          CALL !LXC15
          CALL !LSQRT

          CALL !LLD25             ;ret sheeta
T_OP09:
          A = #R_OK
          CLR1 CY
          RET

C_1:


          DB      0DAH,00FH,049H,081H ;const pai/2
C_2:
          DB 0A2H,0DAH,00FH,049H,082H ;       pai

          END
```

(28) ATOL. SRC

```
$         TITLE     ('TRANSLATE ASCII STRING')
          NAME      M_ATOL

$ INCLUDE(EQU.INC)
$ INCLUDE(ASCII.INC)
$ INCLUDE(REF1.INC)
$ INCLUDE(REF2.INC)

          EXTRN     FTOL,LTOF
          EXTRN     LADDX,LMLTX
          EXTRN     LEXPX

          PUBLIC    ATOL

S_VIRT    EQU       27          ; maximum length of mantissa

          CSEG

;*********************************************************
;*
;*   FLOATING POINT FUNCTION THAT
;*            TRANSLATE ASCII STRING
;*
;*     input  condition : HL <- HEAD ADDRESS of STRING
;*
;*     output conditions: FPR1 <- (REAL VALUE MEANING STRING)
;*                        ERROR then set CY
;*
;*********************************************************
ATOL:

;********* TRANS. SIGN **

          FPR5_1 = #0               ;sign keeper

          CALL !GETC
          if     (A == #N_PL)
            CALL !GETC
          elseif (A == #N_MN)
            SET1 FPR5_1.7
            CALL !GETC
          endif
```

```
;********* TRANS. MANTISSA TO BINARY **

          PUSH HL

          DE = #0
          HL = DE

          SET1 FPR4_1.7      ;work: decimal digit of mantissa
          SET1 FPR4_2.7      ;    : negrect digit of ""

          SET1 X.7           ;    : mantissa digit counter

          while (A <= #N_9 || A == #N_PD) ;'0'-'9','.'
            if (A <= #N_9)
              if_bit (!FPR4_1.7)
                FPR4_1++              ;decimal digit count up
              endif

              if_bit (X.7)
                A <-> L               ;initial digit
                X = #S_VIRT-1+1

              else

                X--
                if_bit (Z)
                  goto ERROR2         ;mantissa length over
                endif
                if_bit (FPR4_2.7)
                  PUSH AX
                  C = A

                  ; *10 +A
                  B = #10
                  A = L
                  MULU B
                  X += C        ;->CY
                  L = X
                  A <-> C

                  A = H
                  MULU B
                  ADDC X,C      ;<-CY,->CY
                  H = X
                  A <-> C
```

```
      A = E
      MULU B
      ADDC X,C      ;<-CY,->CY
      E = X

      ADDC A,#0     ;<-CY
      A <->D

      POP AX

      if_bit (!Z)           ;work area fill
        FPR4_2 = #0         ; then negrect forword digit
      endif
    else
      FPR4_2++              ;negrect digit count up
    endif

  endif

 else       ; '.'

  if_bit (FPR4_1.7)
    FPR4_1 = #0             ; begin count of decimal digit
  else
    goto ERROR2             ; duplicate
  endif
 endif

 POP BC
 PUSH DE
 PUSH HL
 HL = BC
 CALL !GETC
 BC = HL
 POP HL
 POP DE
 PUSH BC
endw

if (A >= #N_MN)
  goto ERROR2
endif

if_bit (X.7)            ;no mantissa digit
  goto ERROR2
endif
```

```
        PUSH AX

        A = FPR4_2
        if_bit (A.7)
          A = #0
        endif

        A <-> X
        A = FPR4_1
        if_bit (A.7)
          A = #0
        endif

        A -= X              ;decimal digit - negrect digit
        FPR4_1 = A          ;esc. decimal digit (:F-N)

;********* NORMALIZE MANTISSA val. **

        AX = DE
        if (AX == #0)
          A <-> H
          X <-> L
          if (AX == #0)
            BC = AX
          else
            BC = #BYTE*(SHORT-2)*100H+ZEROEX
            while_bit (!A.7)
              SHLW AX,1
              B--
            endw
          endif
        else
          BC = #BYTE*SHORT*100H+ZEROEX
          while_bit (!A.7)
            SHLW HL,1
            ROLC X,1
            ROLC A,1
            B--
          endw
        endif

        if_bit (!FPR5_1.7)
          A &= #7FH         ;ret sign bit
        endif
        FPR5+1+2 = A        ;esc mantissa val(x1: .5<=x1<1)
        FPR5+1+1 = X (A)
```

```
        FPR5+1   = H (A)
        FPR5+1+3 = C (A)
        FPR4_2   = B (A)    ;esc mantissa exp of 2 (y1)

;********* TRANS. EXP_PART **

        POP AX
        POP HL
        X = #0              ;work exp val
        C = #0              ;     sign of exp

        if (A <= #N_E)      ;'E' or 'e'
          CALL !GETC
          if     (A == #N_PL)
            CALL !GETC
          elseif (A == #N_MN)
            C = #0FFH
            CALL !GETC
          endif
          if (A > #N_9)
            goto ERROR
          endif

          A <-> X             ;1st. digit
          CALL !GETC
          if (A <= #N_9)
            A <-> D
            A = X
            E = #10
            MULU E
            X += D          ;1st.digit * 10 + 2nd.digit
            CALL !GETC
          endif
        endif

        if (A != #N_NL && A != #N_SP)
          goto ERROR
        endif

        A = C
        A <-> X             ;A:exp-part value (:B)
        if_bit (X.7)
          A ^= #0FFH
          A++
        endif
```

```
;********* UNITE MANTISSA.val & EXP.val **

          A -= FPR4_1          ; B - (F-N) (:B')
          X = #0
          if_bit (A.7)
            X = #0FFH
          endif

          A <-> X
          DE = AX
          CALL !FTOL           ; B' → real
          DE = #C_1
          CALL !LLD2CX
          CALL !LMLTX          ;log2(10) * B'

          CALL !LTOF
          if_bit (Z)
            FPR1_E = #0          ; dec(log2(10)*B') ← 0
          else
            if_bit (FPR1_H.7)
              DE--
            endif
            CALL !LLD21X
            CALL !FTOL
            NOT1 FPR1_H.7
            PUSH DE
            CALL !LADDX        ;dec(log2(10)*B')
            POP DE
          endif

          A = FPR4_2
          X = #0
          A <-> X
          AX += DE
          PUSH AX              ;int(log2(10)*B')+Y1

          DE = #C_2
          CALL !LLD2CX
          CALL !LMLTX          ;dec(log2(10)*B')*log(2)
          CALL !LEXPX

          CALL !LLD25          ;ret. ±X1
          FPR2_X = #0
          CALL !LMLTX          ;±X1 * exp(dec(log2(10)*B')*log(2))
```

```
        A = FPR1_E
        X = #0
        A <-> X

        POP BC
        AX += BC            ;exp.part result
        if_bit (A.7)
          A = #0
        elseif (A != #0)
          goto ERROR
        else
          A = X
        endif

        FPR1_E = A

        A = #R_OK
        CLR1 CY

        RET
ERROR2:
        POP HL
ERROR:
        A = #R_ERR
        SET1 CY
        RET

GETC:
        A = [HL+]
        DE = #INDEX
        B = #S_INDX
        repeat
          if (A == [DE+])
            break
          endif
          B--
        until_bit (Z)

        B--
        A = B
        RET

INDEX:
        @_INDX
```

```
C_1:
        DB 04BH,078H,09AH,054H,082H ;const. log2(10)
C_2:
        DB 0F7H,017H,072H,031H,080H ;       log(2)

        END
```

(29) LTOA. SRC

```
$         TITLE     ('TRANSLATE TO ASCII STRING')
          NAME      M_LTOA

$ INCLUDE(EQU.INC)
$ INCLUDE(ASCII.INC)
$ INCLUDE(REF1.INC)
$ INCLUDE(REF2.INC)

          EXTRN     LADD,LMLT
          EXTRN     LADDX,LMLTX
          EXTRN     LTOF,FTOL
          EXTRN     LLOG10,LEXP10

          PUBLIC    LTOA

S_VIRT    EQU       7         ; string length of mantissa

C_LIM     EQU       38        ; max expression of exp10

C_2H      EQU       20H
C_2E      EQU       84H

          CSEG

;*********************************************************
;*
;*   FLOATING POINT FUNCTION THAT
;*           TRANSLATE TO ASCII STRING
;*
;*     input  condition : FPR1 <- x
;*                       HL <- STORE ADDRESS of STRING
;*
;*     output conditions: STRING HEAD IS APPOINTED TO HL
;*
;*********************************************************
LTOA:
```

```
;********* ZERO FORMAT **

          if (FPR1_E == #0)
            [HL+] = #A_0 (A)
            [HL]  = #A_NL (A)
            goto T_TOA9
          endif

;********* TRANS. to a * exp10(b) (1<= a <10) **

          PUSH HL

          CALL !LLD51               ;esc x to FPR5
          CLR1 FPR1_H.7             ; x <- |x|
          CALL !LLOG10              ; log10(|x|)

          CALL !LTOF                ; trunc integer(log10(|x|)
          if_bit (FPR1_H.7 && !Z)   ; include decimal digit ?
            DE--                    ; floor integer(log10(|x|)  : b
          endif

          PUSH DE                   ;esc. b to STACK

          A = E
          if_bit (A.7)              ; b<0 ?
            A += #C_LIM
            if_bit (!CY)            ; b<(-C_LIM) ?
              E++
              E++                   ;  then coordinate b to caluculatable range
              PUSH DE               ; b'
              CALL !LLD15
              DE = #C_1
              CALL !LLD2C
              CALL !LMLT            ; x*100 :X'
              CALL !LLD51
              POP DE
            endif
          endif
```

```
        CALL !FTOL              ; real(b')
        NOT1 FPR1_H.7
        CALL !LEXP10            ; exp10(-b')
        CALL !LLD25
        FPR2_X = #0
        CALL !LMLTX             ; X' * exp10(-b')   : a

        POP DE
        FPR3_1 = E (A)          ;esc b to FPR3_1

;********* OUTPUT MANTISSA OF EXP10 **

        if_bit (FPR1_H.7)       ; a<0 ?
          POP HL
          [HL+] = #A_MN (A)
          PUSH HL
          CLR1 FPR1_H.7         ; a <- |a|
        endif

        if (FPR1_E == #C_2E)
          CMP FPR1_H,#C_2H
        endif

        if_bit (!CY)            ;limit |a|<10 over then
          DE = #C_3             ;  normalize
          CALL !LLD2CX
          CALL !LMLTX
          FPR3_1++
        endif

        if (FPR1_E < #ZEROEX+1) ;limit |a|>=1 over then
          DE = #C_2             ;  normalize
          CALL !LLD2CX
          CALL !LMLTX
          FPR3_1--
        endif

        CALL !LTOF              ; integer(a)
        A = E
        A += #A_0
        POP HL
```

```
        [HL+] = A
        [HL+] = #A_PD (A)

        B = #S_VIRT-1
        repeat
          PUSH BC
          PUSH HL

          CALL !LLD21X
          CALL !FTOL
          SET1 FPR1_H.7
          CALL !LADDX           ; a - integer(a)
          DE = #C_2
          CALL !LLD2CX
          CALL !LMLTX           ; (a - integer(a))*10

          CALL !LTOF            ; integer(a)
          A = E
          A += #A_0

          POP HL
          [HL+] = A

          POP BC
          B--
        until_bit (Z)

;********* OUTPUT EXP.PART OF EXP10 **

        [HL+] = #A_E (A)

        if_bit (FPR3_1.7)
          [HL+] = #A_MN (A)
          A = #0
          A -= FPR3_1
        else
          A = FPR3_1
        endif
```

```
        X = #0
        A <-> X

        B = #10
        DIVUW B
        A = X

        A += #A_0
        [HL+] = A
        A = B
        A += #A_0
        [HL+] = A
        [HL] =#A_NL (A)
T_TOA9:
        A = #R_OK
        CLR1 CY
        RET

C_1:
        DB      000H,000H,048H,087H ;const 100
C_2:
        DB 000H,000H,000H,C_2H,C_2E ;const 10
C_3:
        DB 0CDH,0CCH,0CCH,04CH,07DH ;const 1/10

        END
```

(30) FTOL. SRC

```
$       TITLE       ('TRANS. FIXED TO REAL')
        NAME        M_FTOL

$ INCLUDE(EQU.INC)
$ INCLUDE(REF1.INC)

        PUBLIC      FTOL

        CSEG

;*********************************************************
;*
;*  FUNCTION THAT TRANSLATE FIXED TO REAL
;*
;*    input  condition : DE <- (integer with sign bit)
;*
;*    output condition : FPR1 <- (real value meaning DE)
;*                       DE keep
;*
;*********************************************************
FTOL:

;********* ZERO EXCEPTION **

          AX = DE
          if (AX == #0)
            FPR1_E = #0
            goto T_TOL9
          endif

;********* GET ABSOLUTE VALUE **

          if (AX >= #8000H+1)
            AX = #0
            AX -= DE
          endif
```

```
;********* TRANSLATE **

          if (A == #0)
            FPR1_E = #ZEROEX+BYTE
            A <-> X
          else
            FPR1_E = #ZEROEX+BYTE*INTEGR
          endif

          while_bit (!A.7)
            SHLW AX,1
            FPR1_E--
          endw

;********* SET VIRT.PART & SIGN BIT **

          FPR1_H = A
          FPR1_M = X (A)
          FPR1   = #0
          FPR1_X = #0

          A = D
          if_bit (!A.7)
            CLR1 FPR1_H.7
          endif

T_TOL9:
          A = #R_OK
          CLR1 CY
          RET

          END
```

(31) LTOF. SRC

```
$       TITLE       ('TRANS. REAL TO FIXED')
        NAME        M_LTOF

$ INCLUDE(EQU.INC)
$ INCLUDE(REF1.INC)

        PUBLIC      LTOF

        CSEG

;*********************************************************
;*
;*  FUNCTION THAT TRANSLATE
;*                REAL TO FIXED
;*
;*    input  condition : FPR1 <- (real value)
;*
;*    output condition : DE <- (integer value meaning FPR1)
;*                       ERROR then set CY
;*                       not INCLUDE DECIMAL PART then set Z
;*                keep : FPR1
;*
;*********************************************************
LTOF:

          A = FPR1_E
          A -= #ZEROEX+1

;********* EXCEPTION **

          if_bit (CY)                ; integer(FPR1)=0 ?
            CMP A,#LOW(-(ZEROEX+1))
            DE = #0
            goto T_TOF9
          endif

          A -= #BYTE*INTEGR
          if_bit (!CY)
            goto ERROR               ;overflow
          endif
```

```
;********* GET UNSIGNED INTEGER **

          A <-> X
          A = FPR1_H
          A |= #80H
          A <-> B
          A = FPR1_M
          A <-> C
          A <-> X

          DE = #0
          if (FPR1 != #0)
            D = #80H
          endif

          if_bit (!A.3)
            C <-> E
            B <-> C
            A += #BYTE
          endif

          A++
          if_bit (!Z)
            repeat
              SHRW BC,1
              RORC D,1
              A++
            until_bit (Z)
          endif

          X = C
          A = B

;********* UNSIGNED --> SIGNED INTEGER **

          if (AX >= #8001H)
            goto ERROR
          endif
          if_bit (A.7)          ;(AX == #8000H)
            if_bit (!FPR1_H.7)
              goto ERROR
            endif
          else
```

```
            if_bit (FPR1_H.7)
              A ^= #0FFH
              A <-> X
              A ^= #0FFH
              A <-> X
              AX++
            endif
          endif

          A <-> D
          X <-> E

;********** DECIMAL PART JUDGE **

          CMPW AX,#0

T_TOF9:
          A = #R_OK
          CLR1 CY
          RET
ERROR:
          A = #R_ERR
          SET1 CY
          RET

          END
```

(32) LTO78. SRC

```
$       TITLE       ('TRANS. FORMAT TO 78K/2')
        NAME        M_LTO78

$ INCLUDE(EQU.INC)
$ INCLUDE(REF1.INC)

        PUBLIC      LTO78

        CSEG

;*********************************************************
;*
;*  FUNCTION THAT TRANSLATE FORMAT
;*                IEEE-754 TO 78K/2
;*
;*    input  condition : FPR1 <- (real:format by iEEE-754)
;*
;*    output condition : FPR1 <- (real:format by 78k/2)
;*                       ERROR then set CY
;*
;*********************************************************
LTO78:

;********* TRANS. BIT FORMAT **

          A = FPR1_E
          CY = FPR1_H.7
          ROLC A,1
          FPR1_H.7 = CY

;********* ADJUST EXP.PART **

          if (A != #0)
            A += #2
           .if_bit (CY)
              A = #R_ERR
              RET
            endif
          endif

          FPR1_E = A

          A = #R_OK
          RET

          END
```

(33) LTOIE. SRC

```
$       TITLE       ('TRANS. FORMAT TO iEEE-754')
        NAME        M_LTOIE

$ INCLUDE(EQU.INC)
$ INCLUDE(REF1.INC)

        PUBLIC      LTOIE

        CSEG

;*********************************************************
;*
;*  FUNCTION THAT TRANSLATE FORMAT
;*                78K/2 TO IEEE-754
;*
;*    input  condition : FPR1 <- (real:format by 78K/2)
;*
;*    output condition : FPR1 <- (real:format by iEEE-754)
;*
;*********************************************************
LTOIE:

;********* ADJUST EXP.PART **

          A = FPR1_E
          A -= #3
          if_bit (CY)
            FPR1   = #0
            FPR1_M = #0
            FPR1_H = #0
            FPR1_E = #0
            goto T_OIE9
          endif

          A++

;********* TRANS. BIT FORMAT **

          CY = FPR1_H.7
          RORC A,1
          FPR1_E = A
          FPR1_H.7 = CY
T_OIE9:
          A = #R_OK
          CLR1 CY
          RET

          END
```