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/III Series

16/8 Bit Single-Chip Microcontroller

Floating-Point Operation Program

μPD78322 Sub-Series
μPD78328 Sub-Series
μPD78334 Sub-Series
μPD78352A Sub-Series
μPD78356 Sub-Series
μPD78366A Sub-Series
μPD78372 Sub-Series

Document No. U12119EJ3V0AN00 (3ed edition)
(O.D.No. IEA-1291A)
Date Published June 1997 J

Phase-out/Discontinued

SUMMARY of Contents

Phase-out/Discontinued

[MEMO]

The export of these products from Japan is regulated by the Japanese government. The export of some or all of these products may be prohibited without governmental license. To export or re-export some or all of these products from a country other than Japan may also be prohibited without a license from that country. Please call an NEC sales representative.

Phase-out/Discontinued

Regional Information

Some information contained in this document may vary from country to country. Before using any NEC product in your application, please contact the NEC office in your country to obtain a list of authorized representatives and distributors. They will verify:

- Device availability
- Ordering information
- Product release schedule
- Availability of related technical literature
- Development environment specifications (for example, specifications for third-party tools and components, host computers, power plugs, AC supply voltages, and so forth)
- Network requirements

In addition, trademarks, registered trademarks, export restrictions, and other legal issues may also vary from country to country.

NEC Electronics Inc. (U.S.)
Santa Clara, California
Tel: 800-366-9782
Fax: 800-729-9288

NEC Electronics (Germany) GmbH
Duesseldorf, Germany
Tel: 0211-65 03 02
Fax: 0211-65 03 490

NEC Electronics (UK) Ltd.
Milton Keynes, UK
Tel: 01908-691-133
Fax: 01908-670-290

NEC Electronics Italiana s.r.1.
Milano, Italy
Tel: 02-66 75 41
Fax: 02-66 75 42 99

NEC Electronics (Germany) GmbH
Benelux Office
Eindhoven, The Netherlands
Tel: 040-2445845
Fax: 040-2444580

NEC Electronics (France) S.A.
Velizy-Villacoublay, France
Tel: 01-30-67 58 00
Fax: 01-30-67 58 99

NEC Electronics (France) S.A.
Spain Office
Madrid, Spain
Tel: 01-504-2787
Fax: 01-504-2860

NEC Electronics (Germany) GmbH
Scandinavia Office
Taeby, Sweden
Tel: 08-63 80 820
Fax: 08-63 80 388

NEC Electronics Hong Kong Ltd.
Hong Kong
Tel: 2886-9318
Fax: 2886-9022/9044

NEC Electronics Hong Kong Ltd.
Seoul Branch
Seoul, Korea
Tel: 02-528-0303
Fax: 02-528-4411

NEC Electronics Singapore Pte. Ltd.
United Square, Singapore 1130
Tel: 253-8311
Fax: 250-3583

NEC Electronics Taiwan Ltd.
Taipei, Taiwan
Tel: 02-719-2377
Fax: 02-719-5951

NEC do Brasil S.A.
Sao Paulo-SP, Brasil
Tel: 011-889-1680
Fax: 011-889-1689

Major Changes

Page	Description
Throughout	The following products have been added. μPD78356(A), μPD78P356(A), μPD78361A, μPD78362A, μPD78P364A, μPD78363A, μPD78365A, μPD78366A, μPD78368A, μPD78P368A, μPD78372(A), μPD78372(A1), μPD78372(A2), μPD78P372(A), μPD78P372(A1), μPD78P372(A2),
	The following products have been deleted. μPD78355A, μPD78356A, μPD78P356A, μPD78362, μPD78P364, μPD78365, μPD78366, μPD78P368, μPD78370, μPD78372, μPD78P372
	The following products have already been developed. μPD78P324, μPD78P324(A), μPD78P324(A1), μPD78P324(A2), μPD78350A, μPD78355, μPD78356, μPD78P356
P. 187	**Appendix** has been added.

The mark ⋆ shows major revised points.

PREFACE

Users

This application note is for engineers who intend to learn the capabilities of the 78K/III Series products and develop floating-point operation programs based on the 78K/III Series products.

* 78K/III Series products

• μPD78322 sub-series	μPD78320, μPD78322, μPD78P322, μPD78323, μPD78324, μPD78P324, μPD78320(A), μPD78320(A1), μPD78320(A2), μPD78322(A), μPD78322(A1), μPD78322(A2), μPD78323(A), μPD78323(A1), μPD78323(A2), μPD78324(A), μPD78324(A1), μPD78324(A2), μPD78P324(A), μPD78P324(A1), μPD78P324(A2)
• μPD78328 sub-series	μPD78327, μPD78328, μPD78P328, μPD78327(A), μPD78328(A)
• μPD78334 sub-series	μPD78330, μPD78334, μPD78P334, μPD78330(A), μPD78330(A1), μPD78330(A2), μPD78334(A), μPD78334(A1), μPD78334(A2), μPD78P334(A), μPD78P334(A1), μPD78P334(A2)
• μPD78352A sub-series	μPD78350, μPD78350A, μPD78352A, μPD78P352
• μPD78356 sub-series	μPD78355, μPD78356, μPD78P356, μPD78356(A), μPD78P356(A)
• μPD78366A sub-series	μPD78361A**Note**, μPD78362A, μPD78P364A, μPD78363A, μPD78365(A), μPD78366(A), μPD78368(A)**Note**, μPD78P368A
• μPD78372 sub-series	μPD78372(A), μPD78372(A1), μPD78372(A2), μPD78P372(A), μPD78P372(A1), μPD78P372(A2)

Note Under development

Purpose

The purpose of this application note is to help users understand floating-point operation application programs based on the 78K/III Series products. Note that the programs contained in this application note are examples, and are not designed for volume production.

Programs described in this document were created using the μPD78320 instruction set.

Organization

This application note covers the following topics:

- Calculation algorithms
- Four arithmetic operations
- Functions (arithmetic, coordinate conversion, data type conversion)
- Execution result
- Program list

The following application note is also available:

- 78K/III Series, Software Basic (U12118E)
- μPD78328, Hardware Basic (IEA-1287)

* **Quality grade**

- Standard μPD78320, μPD78322, μPD78P322, μPD78323, μPD78324, μPD78P324, μPD78327, μPD78328, μPD78P328, μPD78330, μPD78334, μPD78P334, μPD78350, μPD78350A, μPD78352A, μPD78P352, μPD78355, μPD78356, μPD78P356, μPD78361A, μPD78362A, μPD78P364A, μPD78363A, μPD78365A, μPD78366A, μPD78368A, μPD78P368A
- Special μPD78320(A), μPD78320(A1), μPD78320(A2), μPD78322(A), μPD78322(A1), μPD78322(A2), μPD78323(A), μPD78323(A1), μPD78323(A2), μPD78324(A), μPD78324(A1), μPD78324(A2), μPD78P324(A), μPD78P324(A1), μPD78P324(A2), μPD78327(A), μPD78328(A), μPD78330(A), μPD78330(A1), μPD78330(A2), μPD78334(A), μPD78334(A1), μPD78334(A2), μPD78P334(A), μPD78P334(A1), μPD78P334(A2), μPD78356(A), μPD78P356(A), μPD78372(A), μPD78372(A1), μPD78372(A2), μPD78P372(A), μPD78P372(A1), μPD78P372(A2)

The examples in this application note assume use of the "Standard" quality grade devices in general electronics applications. When considering use an example of this application note in an application where a "Special" quality grade device is required, closely study the required quality level of the parts and circuits actually used.

Please refer to **Quality Grades on NEC Semiconductor Devices** (Document number C11531E) published by NEC Corporation to know the specification of quality grade on the devices and its recommended applications.

Application

Standard products

- Applications that handle motor control units (μPD78322 sub-series, μPD78334 sub-series, μPD78352A sub-series, and μPD78356 sub-series)
- PWM inverter control application, inverter air-conditioner (μPD78328 sub-series and μPD78366A sub-series)

Special products

- Car electronics application (μPD78322 sub-series, μPD78328 sub-series, μPD78334 sub-series, μPD78356 sub-series, and μPD78372 sub-series)

* **Related documents** Note that "preliminary" is not indicated in this document, even though the related documents may be preliminary versions.

μPD78322 sub-series

Document name	Document number	
	Japanese	English
μPD78322 Product Letter	IF-6293	-
μPD78322(A) Product Letter	IF-6318	-
μPD78320, 78322 Data Sheet	U10455J	U10455E
μPD78P322 Data Sheet	U10435J	U10435E
μPD78323, 78324 Data Sheet	U10456J	U10456E
μPD78P324, 78P324(A) Data Sheet	IC-8315	IC-2857
μPD78320(A), 78322(A) Data Sheet	IC-8327	IC-2879
μPD78323(A), 78324(A) Data Sheet	IC-8712	IC-3211
μPD78322 User's Manual	IEU-619	IEU-1248
78K/III Series Application Note, Software Basic	U12118J	IEA-1272
78K/III Series Application Note, Floating-Point Operation Program	U12119J	This manual
μPD78322 Special Function Registers	IEM-5501	-
μPD78322 Instruction Set	IEM-601	-
μPD78322 Instruction List	IEM-602	-

Caution These related documents are subject to change without notice. Be sure to use the latest edition of the documents when you design your system.

Phase-out/Discontinued

μPD78328 sub-series

Document name	Document number	
	Japanese	English
μPD78327, 78328 Data Sheet	U10208J	U10208E
μPD78P328 Data Sheet	U10209J	U10209E
μPD78327(A), 78328(A) Data Sheet	IC-8291	IC-2858
μPD78328 User's Manual	IEU-693	IEU-1268
μPD78328 Application Note, Hardware Basic	IEA-716	IEA-1287
78K/III Series Application Note, Software Basic	U12118J	IEA-1272
78K/III Series Application Note, Floating-Point Operation Program	U12119J	This manual
μPD78328 Special Function Registers	IEM-5514	-
μPD78322 Instruction Set	IEM-601	-
μPD78322 Instruction List	IEM-602	-

μPD78334 sub-series

Document name	Document number	
	Japanese	English
μPD78334 Product Letter	IF-6340	-
μPD78334(A) Product Letter	IF-6292	IF-2027
μPD78330, 78334 Data Sheet	U10185J	U10185E
μPD78P334 Data Sheet	IC-8075	IC-2648
μPD78330(A), 78334(A) Data Sheet	IC-8494	IC-3364
μPD78P334(A), (A1), (A2) Data Sheet	IC-8804	IC-3264
μPD78334 User's Manual	IEU-729	IEU-1315
78K/III Series Application Note, Software Basic	U12118J	IEA-1272
78K/III Series Application Note, Floating-Point Operation Program	U12119J	This manual
μPD78334 Special Function Registers	IEM-5518	-
μPD78322 Instruction Set	IEM-601	-
μPD78322 Instruction List	IEM-602	-

Caution These related documents are subject to change without notice. Be sure to use the latest edition of the documents when you design your system.

Phase-out/Discontinued

μPD78352A sub-series

Document name	Document number	
	Japanese	English
μPD78352A Product Letter	IF-6335	IF-2036
μPD78350 Data Sheet	IC-8279	IC-2845
μPD78350A, 78352A Data Sheet	IC-8823	IC-3391
μPD78P352 Data Sheet	IC-8423	IC-2957
μPD78352A User's Manual, Hardware	IEU-781	IEU-1327
μPD78356 User's Manual, Instruction	U12117J	IEU-1395
78K/III Series Application Note, Software Basic	U12118J	IEA-1272
78K/III Series Application Note, Floating-Point Operation Program	U12119J	This manual
μPD78352A Special Function Registers	IEM-5540	IEM-1215
μPD78352A Instruction Set	U11955J	-

μPD78356 sub-series

Document name	Document number	
	Japanese	English
μPD78356 Product Letter	IF-6298	-
μPD78355, 78356 Data Sheet	U10155J	U10155E
μPD78P356 Data Sheet	U10325J	U10325E
μPD78356(A) Data Sheet	U11148J	U11148E
μPD78P356(A) Data Sheet	U11149J	U11149E
μPD78356 User's Manual, Hardware	U10669J	U10669E
μPD78356 User's Manual, Instruction	U12117J	IEU-1395
78K/III Series Application Note, Software Basic	U12118J	IEA-1272
78K/III Series Application Note, Floating-Point Operation Program	U12119J	This manual
μPD78356 Special Function Registers	IEM-5576	IEM-1214
μPD78352A Instruction Set	U11955J	-

Caution These related documents are subject to change without notice. Be sure to use the latest edition of the documents when you design your system.

Phase-out/Discontinued

μPD78366A sub-series

Document name	Document number	
	Japanese	English
μPD78362A Data Sheet	U10098J	U10098E
μPD78P364A Data Sheet	U10106J	U10106E
μPD78363A, 78365A, 78366A Data Sheet	U11109J	U11109E
μPD78P368A Data Sheet	U11373J	U11373E
μPD78362A User's Manual, Hardware	U10745J	U10745E
μPD78366A User's Manual, Hardware	U10205J	U10205E
μPD78356 User's Manual, Instruction	U12117J	IEU-1395
78K/III Series Application Note, Software Basic	U12118J	IEA-1272
78K/III Series Application Note, Floating-Point Operation Program	U12119J	This manual
μPD78362A Special Function Registers	U10210J	-
μPD78366A Special Function Registers	U10107J	-
μPD78352A Instruction Set	U11955J	-

μPD78372 sub-series

Document name	Document number	
	Japanese	English
μPD78372 Product Letter	IF-6351	-
μPD78370(A), 78372(A) Data Sheet	U10789J	U10789E
μPD78P372(A) Data Sheet	U12029J	U12029E
μPD78372 User's Manual, Hardware	U10642J	U10642E
μPD78356 User's Manual, Instruction	U12117J	IEU-1395
78K/III Series Application Note, Software Basic	U12118J	IEA-1272
78K/III Series Application Note, Floating-Point Operation Program	U12119J	This manual
μPD78372 Special Function Registers	U10631J	U10631E
μPD78352A Instruction Set	U11955J	-

Caution These related documents are subject to change without notice. Be sure to use the latest edition of the documents when you design your system.

Phase-out/Discontinued

[MEMO]

CONTENTS

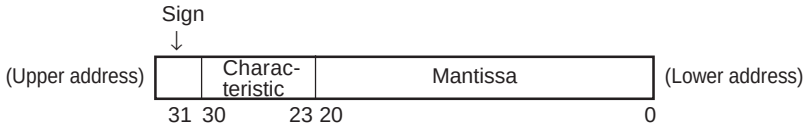
[MEMO]

CHAPTER 1 OVERVIEW

1.1 FLOATING-POINT FORMAT

The operation program represents a floating-point number by using four bytes as shown below.

- Mantissa : 23 bits
- Characteristic : 8 bits
- Sign : 1 bit



A number of this format is expressed as follows:

$$(-1)^{(\text{sign value})} \times (\text{mantissa value}) \times 2^{(\text{characteristic value})}$$

The mantissa, characteristic, and sign are detailed below.

(1) Mantissa

The mantissa is represented using an absolute value; bit positions 22 to 0 in the mantissa correspond to the first decimal place to twenty-third decimal place of a binary number.

A value in the characteristic is adjusted so that a value in the mantissa is always a number from 1 to 2 except when the floating-point number is 0. This means that the value in the ones digit (used to represent the value 1) is always 1, and is omitted in this format.

Remark Operation performed to have the most significant bit (MSB) set always to 1 is called normalization.

(2) Sign

For a positive number, the sign value is 0. For a negative number, the sign value is 1.

(3) Characteristic

An exponent for base 2 is represented using a one-byte integer (or two's complement for a negative value), then is added to a bias of 7FH as indicated in the table below.

Characteristic (hexadecimal)	Value in characteristic
FF	128
FE	127
⋮	⋮
81	2
80	1
7F	0
7E	-1
⋮	⋮
01	-126

Caution Only when the value in the characteristic is 0, the floating-point number is 0. In this case, the mantissa and sign are ignored.

(4) Range of values

The floating-point number x can represent 0 and the following range of values:

$$\text{About } 1.1755 \times 10^{-38} \leq |x| < \text{about } 6.8056 \times 10^{38}$$

1.2 DISTRIBUTION FUNCTIONS

The functions available with the operation program are outlined below. Four groups of functions are available:

- Four arithmetic operations
- Arithmetic functions
- Coordinate-conversion functions
- Functions for converting data types

Distribution function	Function	Function name
Four arithmetic operations	Addition Subtraction Multiplication Division	LADD LSUB LMLT LDIV
Arithmetic functions	Trigonometric functions Sine Cosine Tangent	 LSIN LCOS LTAN
	Natural logarithm (log) Common logarithm (log_{10})	LLOG LLOG10
	Exponential function (base: e) Exponential function (base: 10) Power (a^b)	LEXP LEXP10 LPOW
	Square root	LSQRT
	Inverse trigonometric functions Arcsine Arccosine Arctangent	 LASIN LACOS LATAN
	Hyperbolic functions Hyperbolic sine Hyperbolic cosine Hyperbolic tangent	 LHSIN LHCOS LHTAN
	Absolute function	LABS
	Inverse function	LRCPN
Coordinate-conversion functions	Conversion from polar coordinates to Cartesian coordinates Conversion from Cartesian coordinates to polar coordinates	POTORA RATOPO
Functions for converting data types	Conversion from character strings to floating-point type Conversion from floating-point type to character strings	ATOL LTOA
	Conversion from two-byte integer type to floating-point type Conversion from floating-point type to two-byte integer type	FTOL LTOF

1.3 FILE ORGANIZATION

The operation program includes four types of files:

- Object/source file[Note] : 24 files
- Common subroutine file[Note] : 2 files
- Common data file : 1 file
- Include file : 4 files

Note The object/source files are written in the structured assembler language.

(1) Object/source files

Source file name	Distribution function
LFLT1 .SRC	For arithmetic operations
LSIN .SRC	Sine
LCOS .SRC	Cosine
LTAN .SRC	Tangent
LLOG .SRC	Natural logarithm
LLOG10 .SRC	Common logarithm
LEXP .SRC	Exponential function (base: e)
LEXP10 .SRC	Exponential function (base: 10)
LPOW .SRC	Power
LSQRT .SRC	Square root
LASIN .SRC	Arcsine
LACOS .SRC	Arccosine
LATAN .SRC	Arctangent
LHSIN .SRC	Hyperbolic sine
LHCOS .SRC	Hyperbolic cosine
LHTAN .SRC	Hyperbolic tangent
LABS .SRC	Absolute function
LRCPN .SRC	Inverse function
POTORA .SRC	Conversion from polar coordinates to Cartesian coordinates
RATOPO .SRC	Conversion from Cartesian coordinates to polar coordinates
ATOL .SRC	Conversion from character strings to floating-point type
LTOA .SRC	Conversion from floating-point type to character strings
FTOL .SRC	Conversion from two-byte integer type to floating-point type
LTOF .SRC	Conversion from floating-point type to two-byte integer type

(2) Common subroutine files

Source file name	Distribution function
LFLT2 .SRC	Polynomial calculation function
LLD .SRC	Function for load/permutation between floating-point registers

(3) Common data file

Source file name	Definition
DFLT .SRC	Floating-point register definition

(4) Include files

Source file name	Definition
EQU .INC	EQU definition
REF1 .INC	Declaration of floating-point register reference
REF2 .INC	Declaration of use of function for load/permutation between floating-point registers
ASCII .INC	ASCII code definition

Information about the object module file to be linked in using a function is included in the description of each function.

Remark The assembler package for the 78K/III Series includes a librarian. With the librarian, a library file can be created. If all programs above are stored in a library file, a required module can be automatically linked just by specifying the library file at link time. The programs above can be stored in a library file in an arbitrary order.

1.4 PROGRAM FEATURES

1.4.1 Program Allocation Address

(1) Work area

A work area used with the operation program is called a floating-point register. (Actually, a work area is just a global variable; however, it is called a register because it is used only for floating-point operation.) The common data file DFLT.SRC is used to allocate areas for floating-point registers. Some floating-point registers are allocated in the short direct addressing area, and the other are allocated in a normal RAM area. Arbitrary allocation addresses are assigned in each area.

(2) Code area

A code area needs to be allocated in ROM. There are no other restrictions.

1.4.2 Reentrant

The program is not reentrant.

In multitasking, resource management is required to allow only a single task to call the function.

1.4.3 Stack

The description of each function includes an indication of a maximum required stack area size. When a function is used, a stack not smaller than the indicated maximum size needs to be allocated.

1.4.4 Register Bank

The operation program does not use a register bank selection instruction. The operation program uses a register bank that is already selected when a function is called.

1.4.5 Register Set Selection Flag and Register Names Used

With the operation program, the register set selection flag (RSS) is always set to 0. If an operation program function is called when RSS = 1, no correct results can be expected.

To specify a register, either its absolute name or functional name can be used. With the operation program, however, the functional names are usually used. (For R4 to R7 and RP4 that correspond to AX and BC when RSS = 1, the absolute names are used.)

Register (absolute name)	Functional name		
	Register pair	Upper	Lower
RP0	AX	A	X
RP1	BC	B	C
RP2	RP2	R5	R4
RP3	RP3	R7	R6
RP4	RP4	R9	R8
RP5	UP	-	-
RP6	DE	D	E
RP7	HL	H	L

Remark RP5 is not used for byte registers.

1.4.6 Saving of the Contents of Registers and Flags

No contents of the registers and flags are saved to or restored from stacks except for some functions for converting data types.

Each function uses different registers. (The description of each function includes information about registers used and whether their contents are saved or not.)

1.5 PASSING DATA

1.5.1 Parameters and Return Values

A floating-point register is used to pass data.

With the operation program, the same register is always used to hold a value that is operated on and a return value. In this sense, a value that is operated on is called a destination, and a source value is called a source in this application note.

For detailed information about parameter and return value setting, see the description of each function.

1.5.2 Reporting of the Result of Operation

When an operation is terminated, the result is one of the following:

(1) Normal termination
(2) Underflow
(3) Overflow
(4) Imaginary number generated
(5) Operation impossible (such as log(-1))

In the case of (2) underflow, the value 0 is returned as normal termination.

When the state of termination is reported, the type of error (whether the error is (3), (4), or (5)) is not identified, but the report just indicates whether the operation was terminated normally or abnormally.

State of termination	Contents of register A	CY flag
Normal termination	0	off
Abnormal termination	81H	on

1.6 FLOATING-POINT REGISTERS

In this application note, a work area used with the operation program is called a floating-point register. This section outlines the function of each register.

In the figures below, one box represents one byte, and the left side represents the upper address.

1.6.1 Floating-Point Register 1 (FPR1)

Most functions use this register to store a destination.

FPR1 consists of four contiguous bytes as shown below, and is allocated in a short direct addressing area (on a word boundary).

Characteristic + sign	1st mantissa + characteristic LSB	2nd mantissa	3rd mantissa
FPR1_4	FPR1_3 FPR1_HP	FPR1_2	FPR1_1 FPR1_LP

FPR1

↑
(on word boundary)

A global name is assigned to each byte of FPR1, the upper word (FPR1_HP), and the lower word (FPR1_LP).

1.6.2 Floating-Point Register 2 (FPR2)

This register is used to hold a source.

FPR2 consists of four contiguous bytes as shown below, and is allocated in a short direct addressing area (on a word boundary).

Characteristic + sign	1st mantissa + characteristic LSB	2nd mantissa	3rd mantissa
FPR2_4	FPR2_3 FPR2_HP	FPR2_2	FPR2_1 FPR2_LP

FPR2

↑
(on word boundary)

FPR2 has the same register organization as FPR1.

1.6.3 Floating-Point Register 3 to Floating-Point Register 5 (FPR3 to FPR5)

The arithmetic functions use these registers as temporary work areas.

FPR3 to FPR5 have an organization and global names similar to those of FPR1 or FPR2. However, FPR3 to FPR5 differ from FPR1 and FPR2 in that FPR3 to FPR5 are allocated in an ordinary RAM space.

1.6.4 Extended-Mantissa Registers

An extended-mantissa register is an area used to internally extend the mantissa by one byte for the fourth mantissa (the 24th to 31st decimal places).

The general registers R5 and R4 are used for FPR1 and FPR2. FPR3_X, FPR4_X, and FPR5_X are used for FPR3, FPR4, and FPR5.

FPR3_X, FPR4_X, and FPR5_X are allocated in an ordinary RAM space.

Remark All functions do not use these registers. The description of each function includes information about which registers are used for each function.

1.6.5 Load/Permutation between Floating-Point Registers

Arithmetic functions and functions for converting data types often perform register load, store, and permutation operations. So functions for load/store and permutation operations between floating-point registers are provided.

The table below lists these load/store functions and permutation functions.

Function name		Function
LLD21,	LLD21X	Loads the contents of register 1 into register 2.
	LLD31X	Loads the contents of register 1 into register 3.
LLD41,	LLD41X	Loads the contents of register 1 into register 4.
LLD51,	LLD51X	Loads the contents of register 1 into register 5.
	LLD32X	Loads the contents of register 2 into register 3.
LLD52		Loads the contents of register 2 into register 5.
	LLD13X	Loads the contents of register 3 into register 1.
	LLD23X	Loads the contents of register 3 into register 2.
LLD24,	LLD24X	Loads the contents of register 4 into register 2.
LLD15,	LLD15X	Loads the contents of register 5 into register 1.
LLD25,	LLD25X	Loads the contents of register 5 into register 2.
LLD1C,	LLD1CX	Loads constant data into register 1.
LLD2C,	LLD2CX	Loads constant data into register 2.
	LXC13X	Exchanges the contents of register 1 with the contents of register 3.
	LXC14X	Exchanges the contents of register 1 with the contents of register 4.
	LXC15X	Exchanges the contents of register 1 with the contents of register 5.

Caution A function with its name ending with X is a load/permutation function including an extended mantissa.

CHAPTER 2 CALCULATION ALGORITHMS

This chapter outlines the algorithms used as calculation bases.

2.1 FUNCTION EXPANSION METHODS

Three expansion methods are used:

- Root : Newton-Raphson method
- Inverse trigonometric function: Best Approximation method
- Others : Taylor expansion method

2.2 ROUNDING METHOD

A number is rounded to 0.

2.3 METHOD OF PREVENTING ROUNDING ERROR

If a polynomial subject to Taylor expansion results in a difference series, an extreme rounding error can occur, depending on the range of values used. To prevent such a rounding error, the operation program uses such an expansion expression that the values of the first term to the N-th term of the expansion expression reach 0 monotonously and rapidly.

2.4 ERROR IN POLYNOMIAL ADDITION/MULTIPLICATION

When an addition of an eight-term polynomial rounded to 0 is performed in a numbering system with 24 significant bits, the result includes an error of up to four bits. When x^8 is calculated by performing seven multiplications, the result includes the same error.

To minimize this type of cumulative error, the operation program internally uses a 31-bit mantissa (including an 8-bit extended mantissa).

[MEMO]

CHAPTER 3 FOUR ARITHMETIC OPERATIONS

Four arithmetic operation functions are available.

(1) Floating-point addition (LADD)
An addition is performed using the value in FPR1 as an augend, and the value in FPR2 as an addend.
(2) Floating-point subtraction (LSUB)
A subtraction is performed using the value in FPR1 as a minuend, and the value in FPR2 as a subtrahend.
(3) Floating-point multiplication (LMLT)
A multiplication is performed using the value in FPR1 as a multiplicand, and the value in FPR2 as a multiplier.
(4) Floating-point division (LDIV)
A division is performed using the value in FPR1 as a dividend, and the value in FPR2 as a divisor.

The result of an operation is held in FPR1.

3.1 FLOATING-POINT ADDITION (LADD)

(1) Description

This function returns (x + y) to FPR1, where x is the value in FPR1 before addition and y is the value in FPR2.

(2) Object module files to be linked

DFLT, LFLT1

(3) Required stack area size in bytes

2 (two bytes only for a return address from LADD)

(4) Registers to be used

AX, RP2, RP3, RP4

(5) Work areas to be used

FPR1, FPR2

(6) Execution time (Internal system clock: 8 MHz, no wait)

Average : 106 µs

Maximum: 221 µs (0.5 + (-0.50000006))

(7) Operation

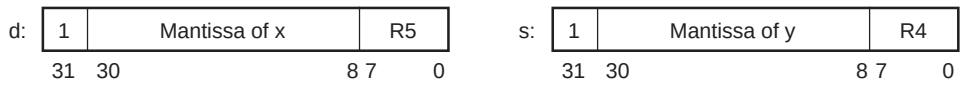
(a) The function terminates operation, when x or y is 0, with the non-zero value (x or y) being the solution.

(b) The function terminates operation, when a difference between the exponents is 32 or greater, with the larger value being the solution.

(c) The function memorizes the value of x or y, whichever has a greater exponent.

(d) The function performs a mantissa addition/subtraction as described below.

The function adds an MSB (1) to the mantissa of x, and assumes it as a double-word variable, d. The function adds an extended mantissa (8 bits) to the mantissa of y, and assumes it as a double-word variable, s.



The sum of the mantissas (or difference between the mantissas) is found according to the procedure below.

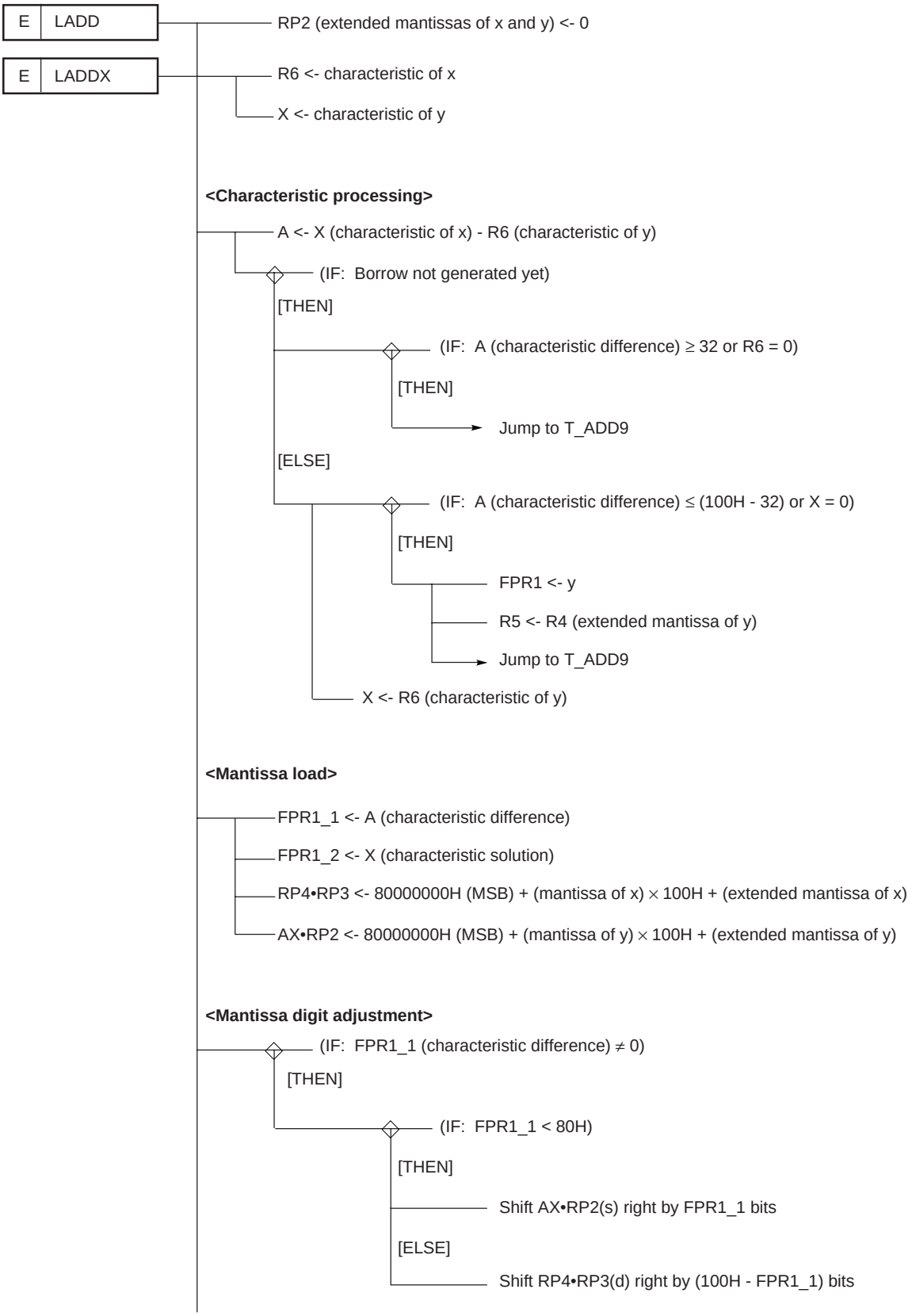
(i) When no match is found between the two characteristics, s is shifted right by the number of characteristic difference bits if the x side is greater; d is shifted right by the same number if the y side is greater.

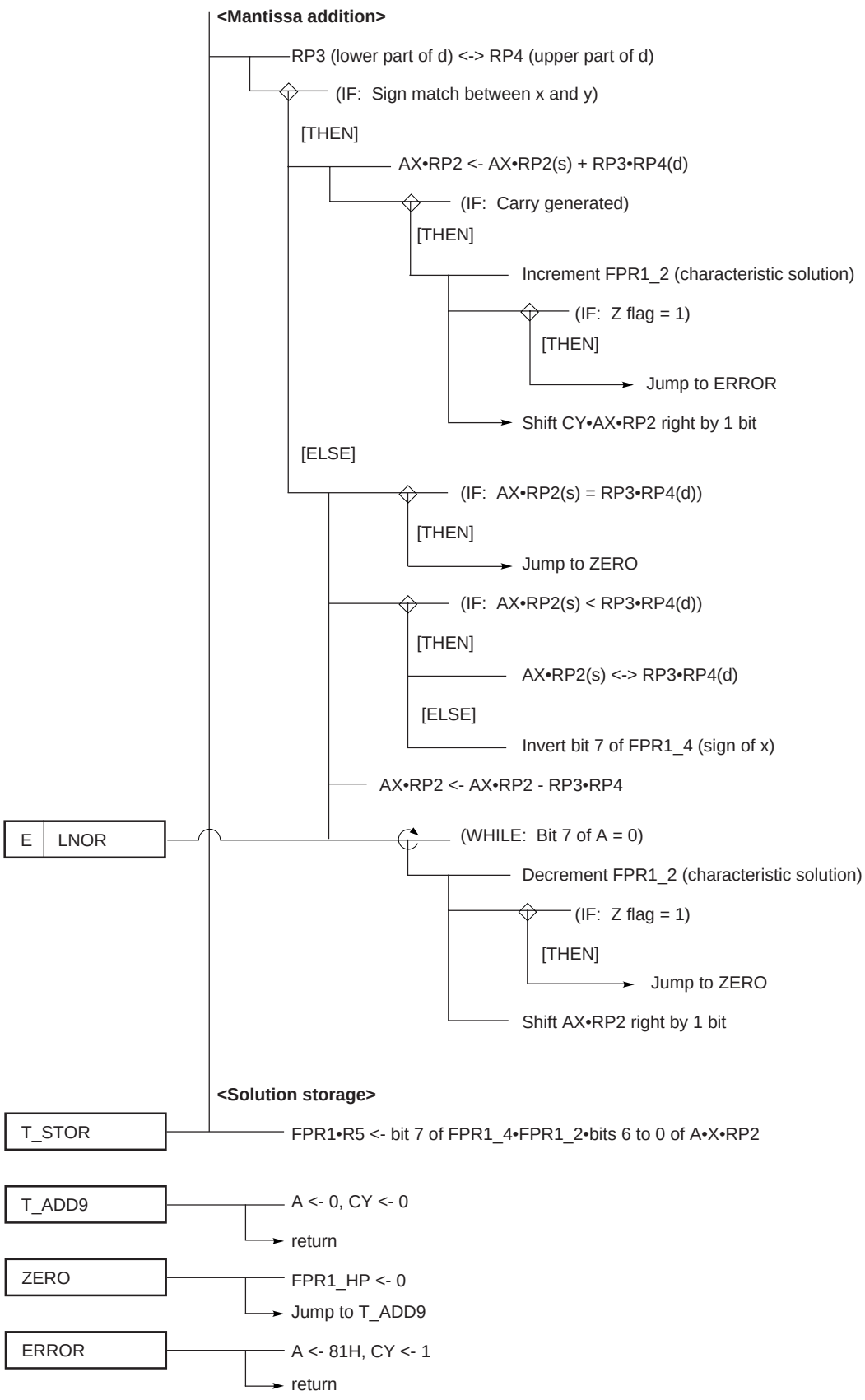
(ii) When the sign of x matches the sign of y, the sign of x is memorized. Then an addition of d and s is performed.

(iii) If the sign of x does not match the sign of y, the sign of d or s, whichever greater, is memorized. Then d or s, whichever smaller, is subtracted from d or s, whichever larger.

(e) The memorized characteristic and the solution of mantissa operation found in (d) above are normalized and loaded into FPR1 together with the sign bit.

(8) Flowchart





Remarks 1. The label E | | indicates a global name.

2. The label T_STOR, ZERO or ERROR is also referenced by the LMLT or LDIV function.

3. The label E | LADDX indicates an internal global name used with an arithmetic function to perform an addition involving extended mantissas.

4. The label E | LNOR indicates an internal global name used to perform normalization from a rounding error with a function for converting data types.

5. CY•AX•RP2 represents a 33-bit variable with MSB = CY. The other combinations in the flowchart have the same meaning. Similar representations are used in the flowcharts of other functions.

3.2 FLOATING-POINT SUBTRACTION (LSUB)

(1) Description

This function returns (x - y) to FPR1, where x is the value in FPR1 before subtraction and y is the value in FPR2.

(2) Object module files to be linked

DFLT, LFLT1

(3) Required stack area size in bytes

2 (two bytes only for a return address from LSUB)

(4) Registers to be used

AX, RP2, RP3, RP4

(5) Work areas to be used

FPR1, FPR2

(6) Execution time (Internal system clock: 8 MHz, no wait)

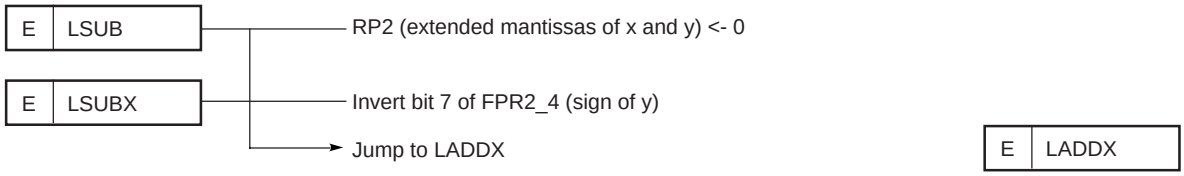
Average : 102 µs

Maximum: 224 µs (0.5 - 0.50000006)

(7) Operation

The function inverts the sign of y, and jumps to LADD.

(8) Flowchart



Remark The label E | LSUBX indicates an internal global name used with an arithmetic function to perform a subtraction involving extended mantissas.

3.3 FLOATING-POINT MULTIPLICATION (LMLT)

(1) Description

This function returns (x × y) to FPR1, where x is the value in FPR1 before multiplication and y is the value in FPR2 is.

(2) Object module files to be linked

DFLT, LFLT1

(3) Required stack area size in bytes

2 (two bytes only for a return address from LMLT)

(4) Registers to be used

AX, RP2, RP3, RP4

(5) Work areas to be used

FPR1, FPR2

(6) Execution time (Internal system clock: 8 MHz, no wait)

Average : 59.1 µs

Maximum: 61.0 µs (2 × 2)

(7) Operation

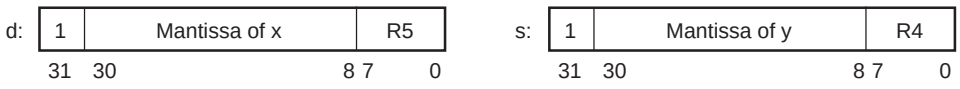
(a) The function returns 0 when x or y is 0.

(b) The function finds a characteristic solution by adding one characteristic to the other.

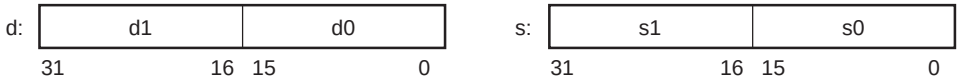
(c) The function XORs the sign of x with the sign of y to produce the resultant sign.

(d) The function performs a mantissa multiplication as described below.

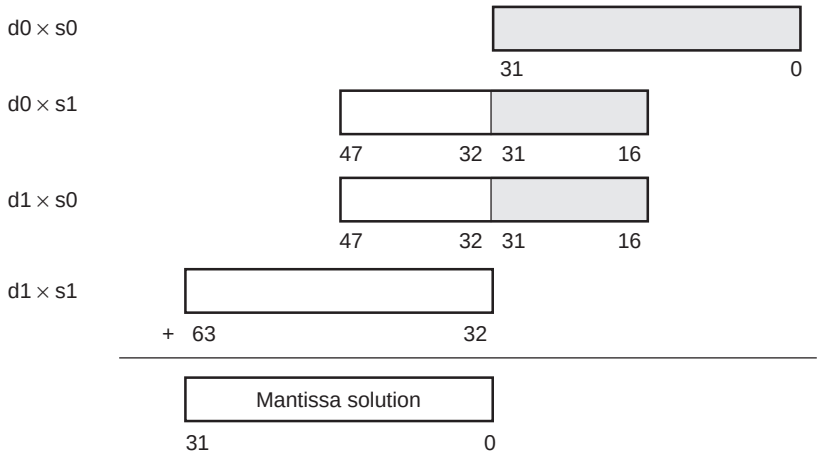
The function adds an MSB (1) to the mantissa of x, and assumes it as a double-word variable, d. The function adds an extended mantissa (8 bits) to the mantissa of y, and assumes it as a double-word variable, s.



Then the function divides d and s into an upper part and lower part, and assumes d1, d0, s1, and s0 as word variables as shown below.

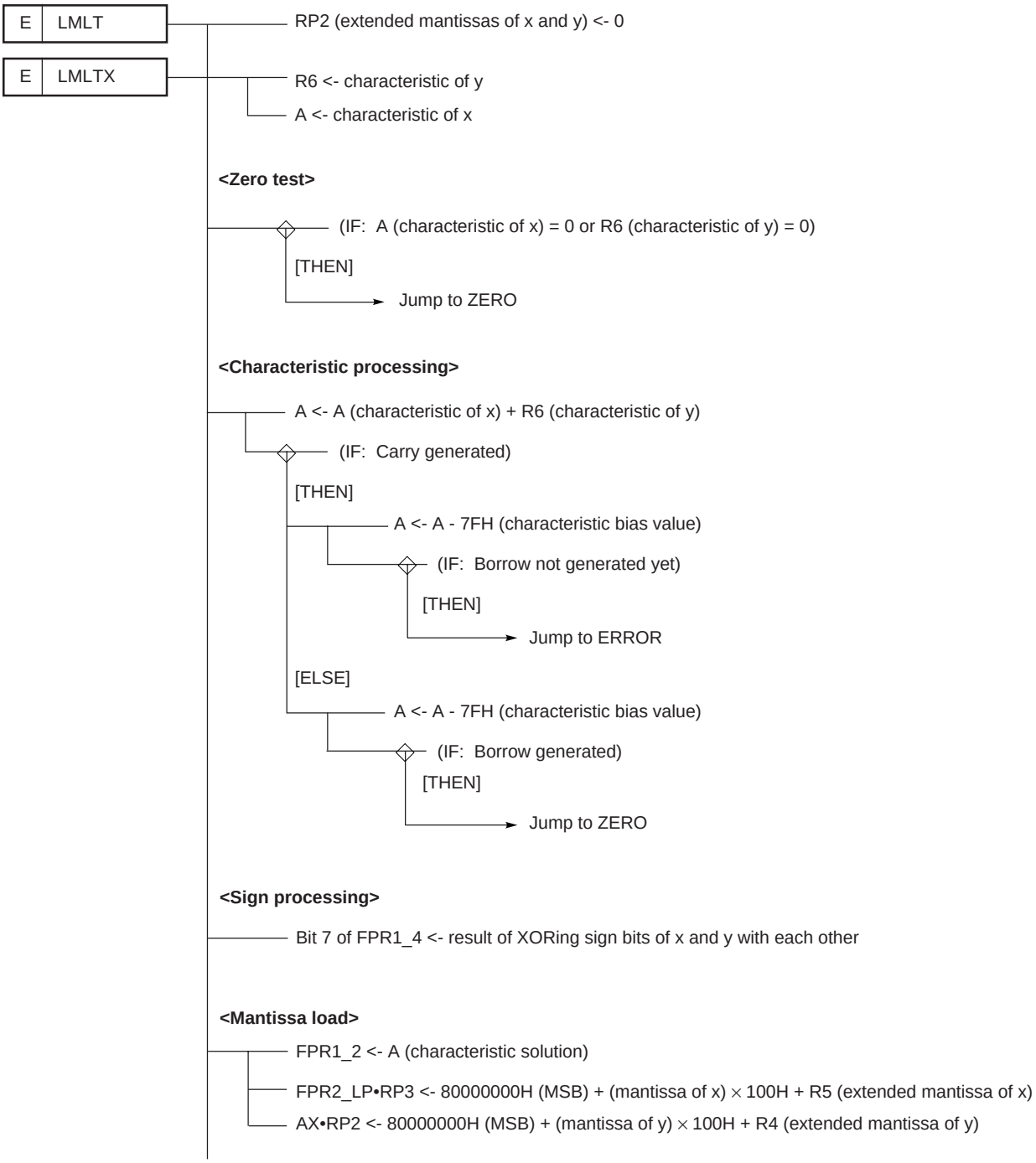


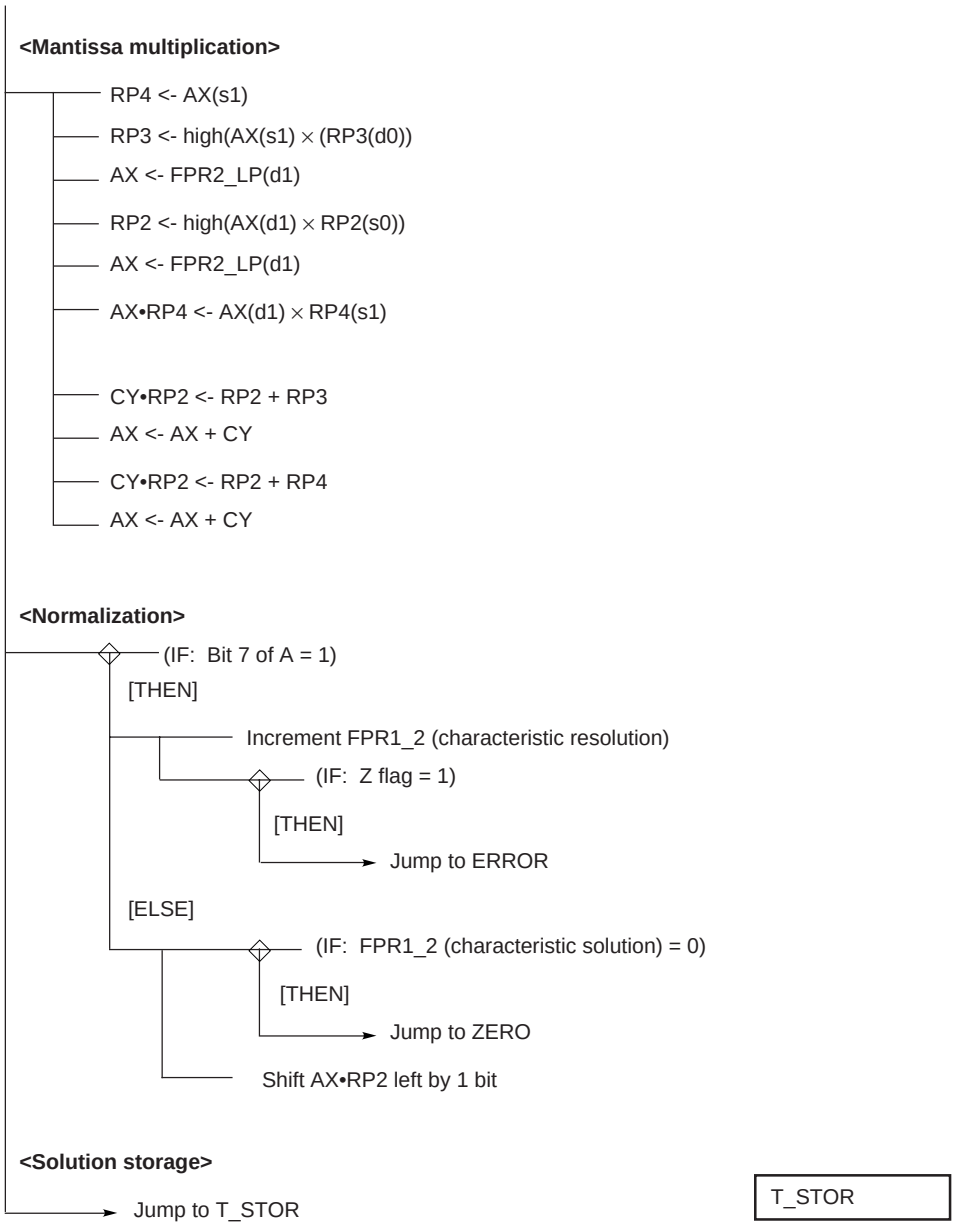
The upper 32 bits of the result of operation are found according to the procedure below. (The shaded portions are discarded.)



(e) The characteristic solution and the solution of mantissa operation found in (d) above are normalized and loaded into FPR1 together with the sign bit.

(8) Flowchart





Remarks 1. High ( ) indicates the upper 16 bits of the result of operation.

2. The label E LMLTX indicates an internal global name used with an arithmetic function to perform a multiplication involving extended mantissas.

3.4 FLOATING-POINT DIVISION (LDIV)

(1) Description

This function returns (x ÷ y) to FPR1, where x is the value in FPR1 before division and y is the value in FPR2.

(2) Object module files to be linked

DFLT, LFLT1

(3) Required stack area size in bytes

2 (two bytes only for a return address from LDIV)

(4) Registers to be used

AX, RP2, RP3, RP4, UP, DE

(5) Work areas to be used

FPR1, FPR2

(6) Execution time (Internal system clock: 8 MHz, no wait)

Average : 79.5 µs

Maximum: 84.0 µs (31377.404/(3.2481321e + 37))

(7) Operation

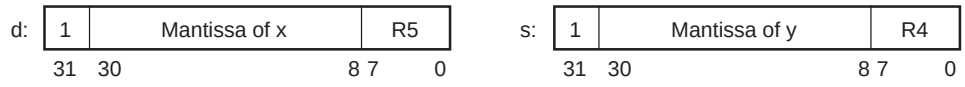
(a) The function terminates abnormally when y is 0. The function returns 0 when x is 0.

(b) The function finds a characteristic solution by subtracting the characteristic of y from the characteristic of x.

(c) The function XORs the sign of x with the sign of y to produce the resultant sign.

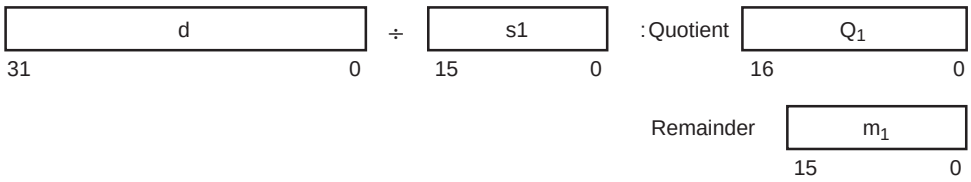
(d) The function performs a mantissa division as described below.

The function adds an MSB (1) to the mantissa of x, and assumes it as a double-word variable, d. The function adds an extended mantissa (8 bits) to the mantissa of y, and assumes it as a double-word variable, s.



Then the function divides d and s into an upper part and lower part, and assumes d1, d0, s1, and s0 as word variables as in the case of the multiplication function.

(i) By replacing (d ÷ s) with (d ÷ s1), an approximate value, Q_1, with 15 to 17 significant bits is found as shown below.



The error (F) included in Q_1 comes from the discarded part (m_1/s1) of the remainder (m_1) and a shortage in the number of divisor digits (-(d/s1 - d/s)). $F \times 2^{16}$ can be transformed as follows:

$$F \times 2^{16} = F' = \left\{ \frac{m_1}{s1} - \left(\frac{d}{s1} - \frac{d}{s} \right) \right\} \times 2^{16}$$

$$= \frac{m_1 \times s - d \times s2}{s1 \times s} \times 2^{16}$$

(Approximated with $s \fallingdotseq s1 \times 2^{16}$ and $d \fallingdotseq d1 \times 2^{16}$)

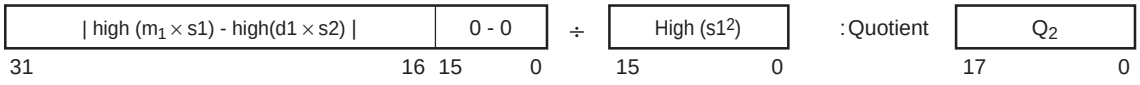
$$\fallingdotseq \frac{m_1 \times s1 - d1 \times s2}{s1^2 \times 2^{-16}}$$

(The lower part of the product of (word × word) is discarded; this discard operation is represented by high ( ).)

$$\fallingdotseq \frac{\{\text{high}\,(m_1 \times s1) - \text{high}\,(d1 \times s2)\} \times 2^{16}}{\text{high}\,(s1^2)}$$

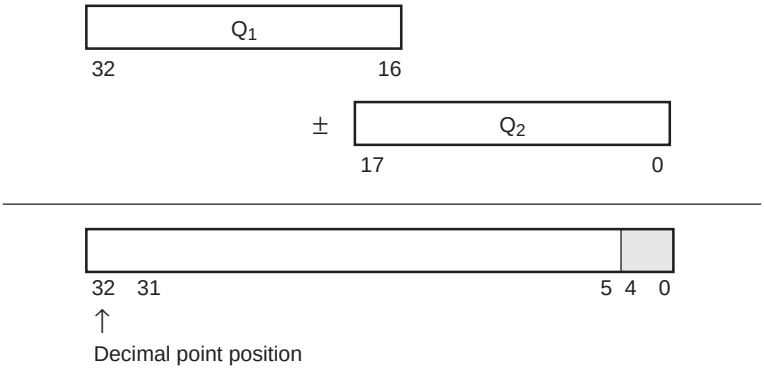
(ii) The denominator and numerator of F' are calculated.

(iii) Q_2, which is the Q_1 error (F) multiplied by 2^{16}, is found as shown below.



The error included in Q_2 is not greater than 18H.

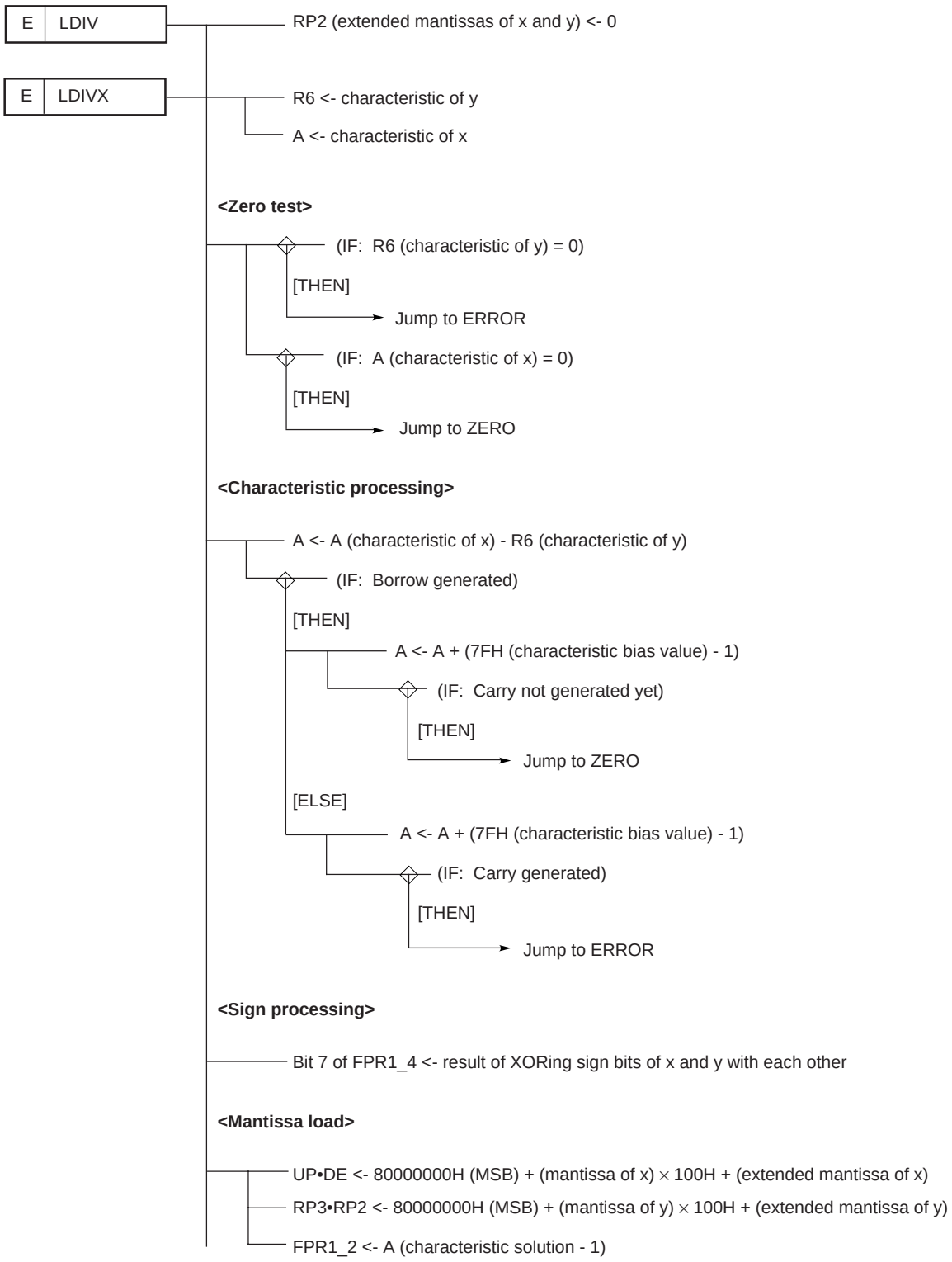
(iv) From Q_1 and Q_2, (d ÷ s) is obtained. (The shaded part indicates a maximum error.)



Remark When F is a positive value, an addition is performed. When F is a negative value, a subtraction is performed.

(e) The characteristic solution and the solution of mantissa operation found in (d) above are normalized and loaded into FPR1 together with the sign bit.

(8) Flowchart



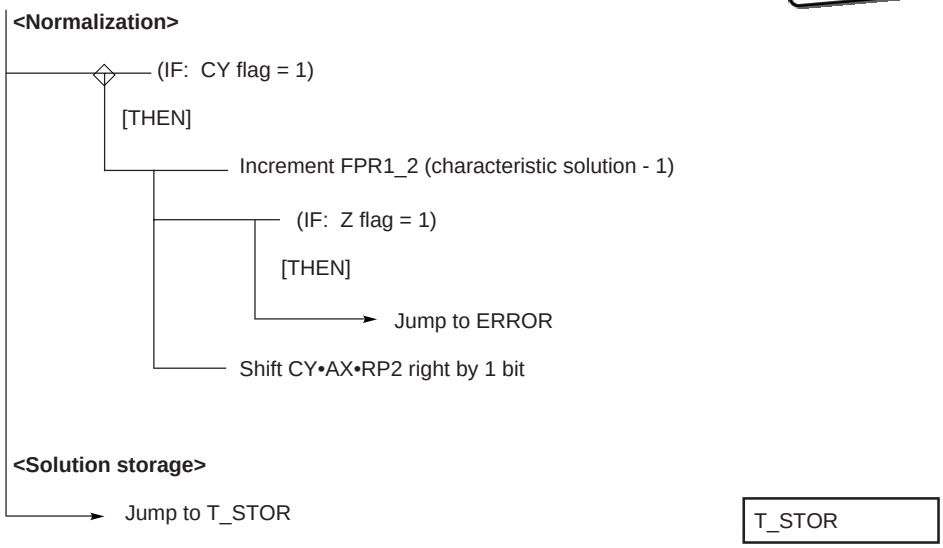
<Mantissa division>

- RP4 <- RP3(s1), AX <- UP(d1)
- AX•DE <- quotient of (AX•DE(d) ÷ RP3(s1))
- RP3 <- remainder of (AX•DE(d) ÷ RP3(s1))
- Bit 0 of FPR1_4 <- bit 0 of X (bit 16 of Q_1)
- AX <- RP4(s1)
- RP3 <- high(AX(s1) × RP3(m_1))
- AX <- UP(d1)
- AX <- high(AX(d1) × RP2(s2))
- RP3 <- RP3(high(m_1 × s1)) - AX(high(d1 × s2))
  - (IF: CY (borrow generated) = 1)

    [THEN]
    - RP3 <- two's complement of RP3
    - CY (sign of Q_2) <- 1
- AX <- RP4(s1)
- AX <- high(AX(s1) × RP4(s1))
- RP4 <- DE (bits 15 to 0 of Q_1)
- AX(high($s1^2$)) <-> RP3(|high(m1 × s1) - high(d1 × s2)|)
- DE <- 0
- AX•DE <- AX•DE(|high(m1 × s1) - high(d1 × s2)| × 2^{16}) ÷ RP3(high($s1^2$))
  - (IF: CY (sign of Q_2) = 0)

    [THEN]
    - CY•AX•RP2 <- AX•DE(Q_2) + RP4 (bits 15 to 0 of Q_1) × 10000H

    [ELSE]
    - AX•RP2 <- RP4 × 10000H - AXxDE, CY <- borrow
- CY <- CY (carry in addition ÷ borrow in subtraction) XOR bit 0 of FPR1_4 (bit 16 of Q_1)



Remark The label E | LDIVX indicates an internal global name used with an arithmetic function to perform a division involving extended mantissas.

CHAPTER 4 ARITHMETIC FUNCTIONS

The following arithmetic functions are available:

(1) Sine (LSIN)
This function finds the sine of the FPR1 value.
(2) Cosine (LCOS)
This function finds the cosine of the FPR1 value.
(3) Tangent (LTAN)
This function finds the tangent of the FPR1 value.
(4) Natural logarithm (LLOG)
This function finds the natural logarithm of the FPR1 value.
(5) Common logarithm (LLOG10)
This function finds the common logarithm of the FPR1 value.
(6) Exponential function (base e) (LEXP)
This function raises e to the power of the FPR1 value used as the exponent.
(7) Exponential function (base 10) (LEXP10)
This function raises 10 to the power of the FPR1 value used as the exponent.
(8) Power (LPOW)
This function raises the FPR1 value to the power of the FPR2 value.
(9) Square root (LSQRT)
This function finds a square root of the FPR1 value.
(10) Arcsine (LASIN)
This function finds the arcsine of the FPR1 value.
(11) Arccosine (LACOS)
This function finds the arccosine of the FPR1 value.
(12) Arctangent (LATAN)
This function finds the arctangent of the FPR1 value.
(13) Hyperbolic sine (LHSIN)
The function finds the hyperbolic sine of the FPR1 value.
(14) Hyperbolic cosine (LHCOS)
The function finds the hyperbolic cosine of the FPR1 value.
(15) Hyperbolic tangent (LHTAN)
This function finds the hyperbolic tangent of the FPR1 value.
(16) Absolute function (LABS)
This function finds the absolute value of the FPR1 value.
(17) Inverse function (LRCPN)
This function finds the reciprocal of the FPR1 value.

The result of each operation is loaded into FPR1.

4.1 COMMON SUBROUTINES (LPLY, LPLY2)

As described earlier, the arithmetic functions other than the function of square root are calculated using the Taylor approximation method or Best Approximation method. These approximation methods are expressed as polynomials of higher order, and have a common pattern.

So two types of polynomial calculation functions, LPLY and LPLY2, are provided as common subroutines.

(1) Description

Each subroutine returns the result of polynomial calculation to FPR1•R5 together with an extended mantissa.

(2) Object module files to be linked

DFLT, LFLT1, LFLT2, LLD

(3) Required stack area size in bytes

4 (including two bytes as a return address from LPLY or LPLY2)

(4) Registers to be used

AX, C, RP2, RP3, RP4, DE

(5) Work areas to be used

FPR1, FPR2, FPR3, FPR4, FPR3_X, FPR4_X (The contents of FPR4 and FPR4_X are preserved.)

(6) Algorithm

Two types of polynomial calculation are possible:

<1> $x + k_1xy + k_1k_2xy^2 + k_1k_2k_3xy^3 + \bullet\bullet\bullet + k_1k_2\bullet\bullet\bullet k_nxy^n$

<2> $z + k_1xy + k_1k_2xy^2 + k_1k_2k_3xy^3 + \bullet\bullet\bullet + k_1k_2\bullet\bullet\bullet k_nxy^n$

Remark x: First term of a polynomial

y: Multiplication constant (such as x^2) for the degree change of each term

(k_1, k_1k_2, •••, $k_1k_2\bullet\bullet\bullet k_n$): Coefficient of each term

Polynomial	Usage condition	Function
<1>	Each term has the common divisor x.	LPLY
<2>	The first term is a constant (z).	LPLY2

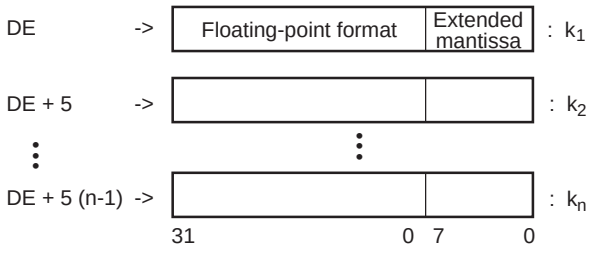
To minimize error, floating-point numbers with an extended mantissa are used for x, y, z, k_1, k_2, •••, k_n.

Caution The Taylor approximation requires that the following condition be satisfied:
|first term| > |second term| > ••• > |n-th term|

(7) Input conditions

Polynomial	x	y	z	n	Start address of coefficient string (k_1, k_2, k_3, •••, k_n)
<1>	FPR1•R5	FPR4•FPR4_X	-	C	DE
<2>	FPR3•FPR3_X	FPR4•FPR4_X	FPR1•R5	C	DE

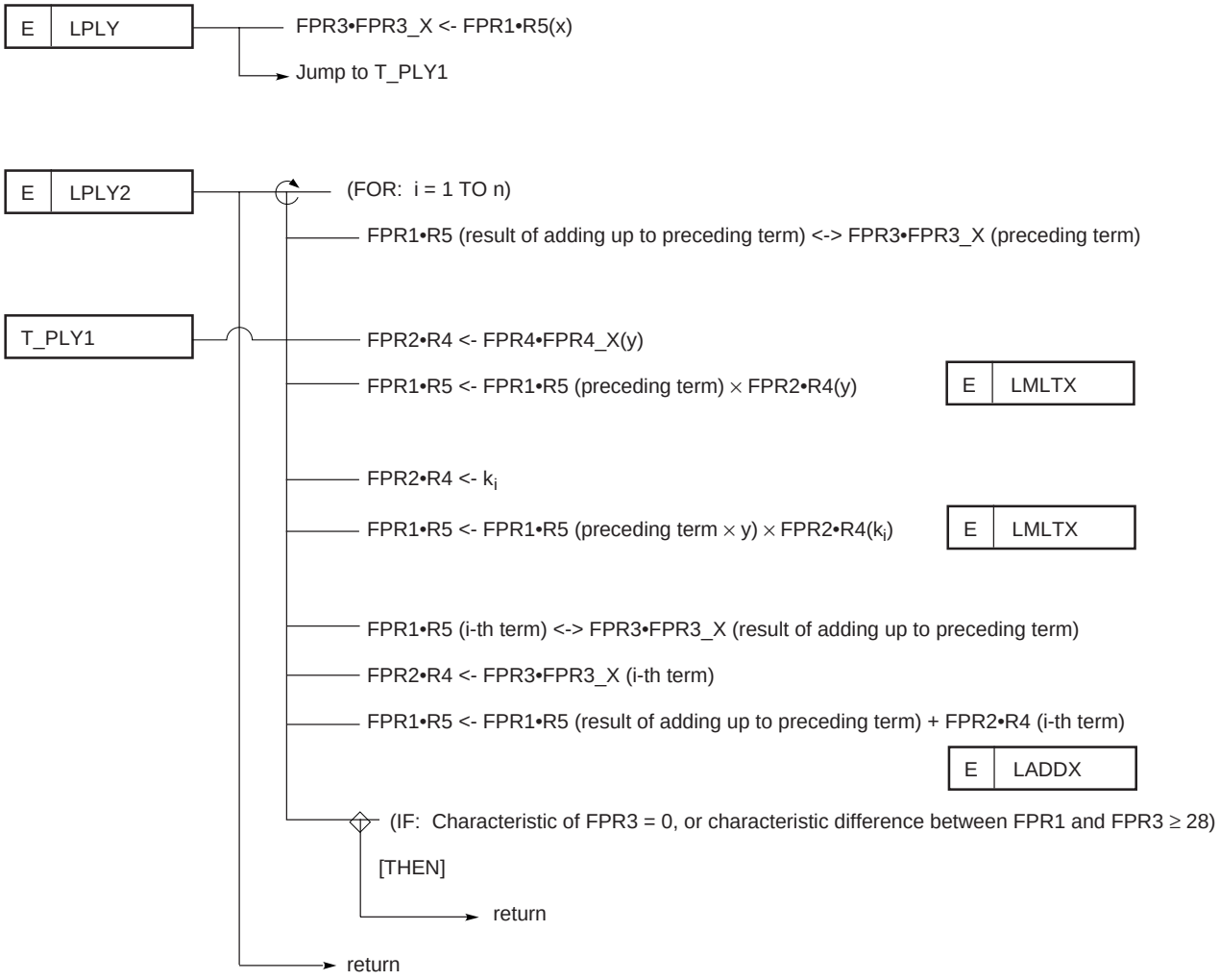
A coefficient string is held in the following format:



(8) Operation

(a) The function finds the i-th term by multiplying the (i-1)-th term with y, and k_i.
(b) The function adds the value of the i-th term to the sum of up to the (i-1)-th term.
(c) The function terminates the calculation when the i-th term is sufficiently smaller than the sum of up to the i-th term.
(d) The function repeats steps (a) to (c) up to the n-th term.

(9) Flowchart



4.2 SINE (LSIN)

(1) Description

This function returns sin(x) to FPR1, where x is the value of FPR1.

- Unit: Radian

(2) Object module files to be linked

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

(3) Required stack area size in bytes

6 (including two bytes as a return address from LSIN)

(4) Registers to be used

AX, C, RP2, RP3, RP4, DE, UF (user flag)

(5) Work areas to be used

FPR1, FPR2, FPR3, FPR4, FPR3_X, FPR4_X

(6) Execution time (Internal system clock: 8 MHz, no wait)

Average : 1.24 ms

Maximum: 4.68 ms (sin(6.806e + 38))

(7) Algorithm

The following Taylor approximate expression is used:

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

(8) Operation

(a) The function memorizes the sign of x, and finds the absolute value of x.

(b) The function scales x to the range $0 \leq x < \pi/2$.

Let x' be a value after scaling. Then the following expressions are given:

$$\sin(\pi/2 + x') = \sin(\pi/2 - x')$$
$$\sin(\pi + x') = \sin(-x')$$
$$\sin(3\pi/2 + x') = \sin(x' - \pi/2)$$

So sin(x) can be replaced as indicated below, where n is the quotient obtained by dividing x by π/2, and x' is the remainder.

Remainder of n/4	sin(x)	x"
0	sin(x')	x'
1	sin(π/2 - x')	π/2 - x'
2	sin(-x')	-x'
3	sin(x' - π/2)	x' - π/2

(c) The function multiplies x" by the preserved sign.

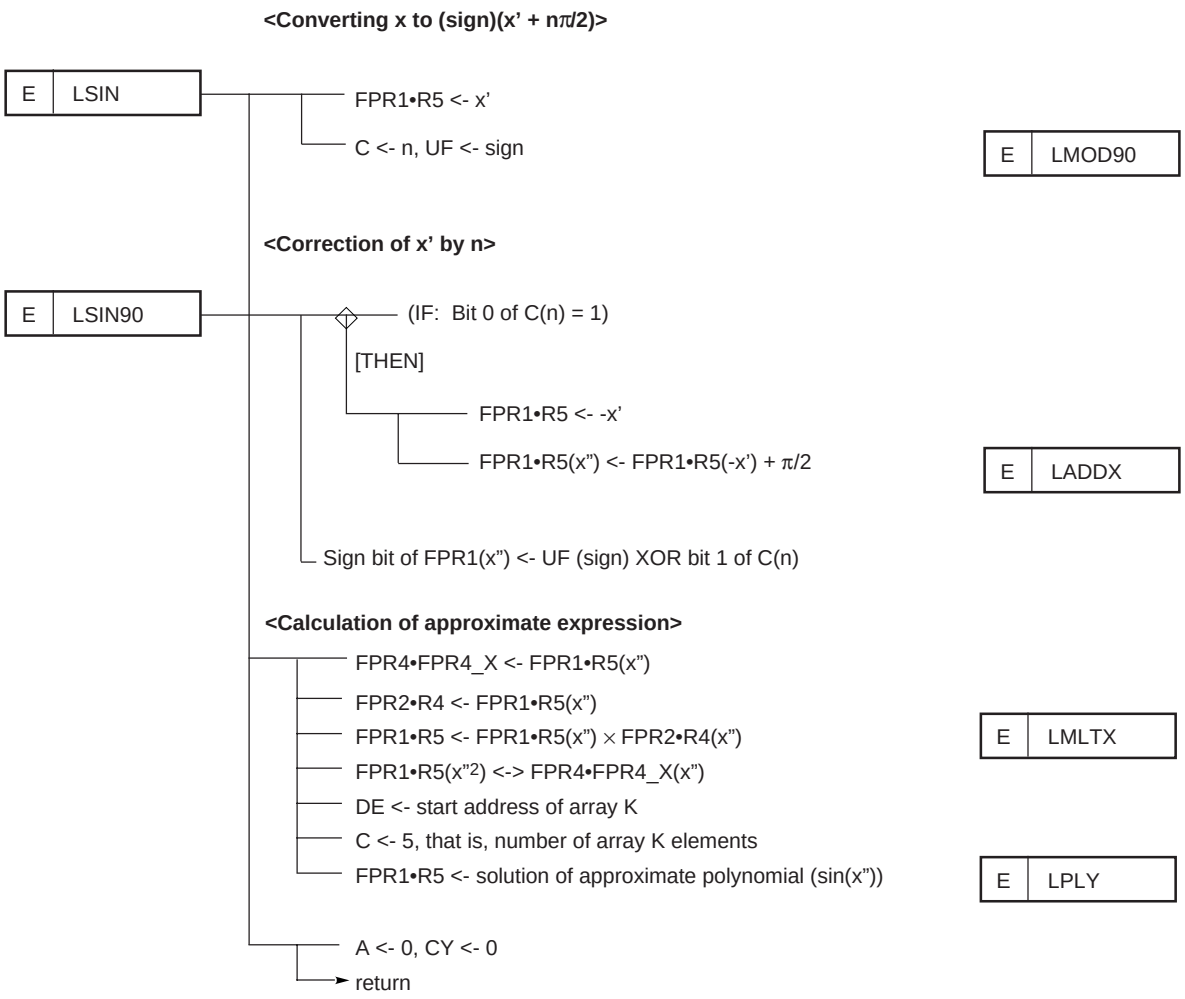
(d) The function finds the solution, sin(x"), according to the Taylor approximate expression.

(9) Floating-point constant

(a) The constants 2/π and π/2 with an extended mantissa are used.

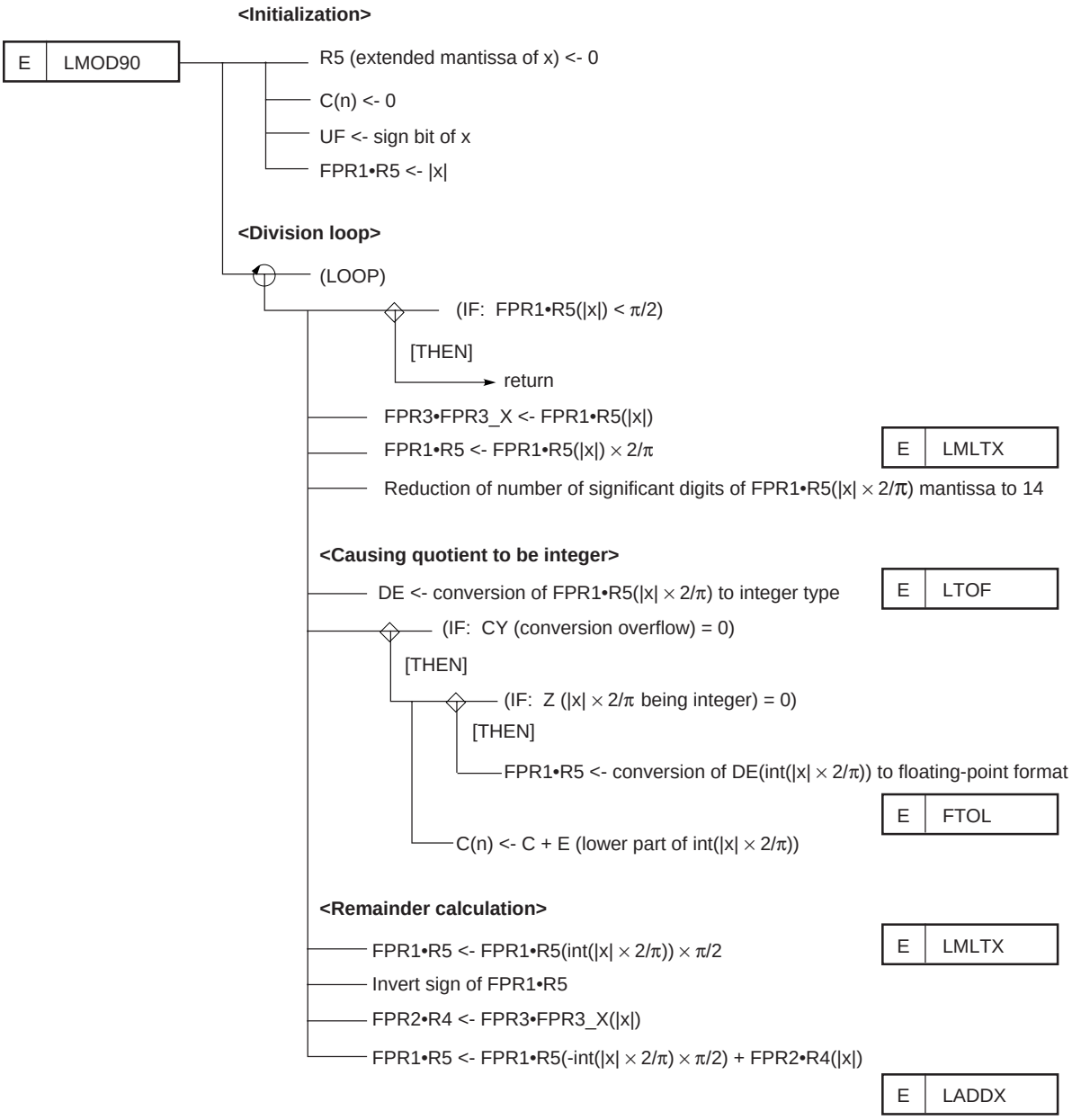
(b) For the coefficient string of an approximate polynomial, the constants -1/3!, -3!/5!, -5!/7!, -7!/9!, and -9!/11! with an extended mantissa are used as array K with five elements.

(10) Flowchart



Phase-out/Discontinued

(Subroutine to find the quotient and remainder of a division by π/2)



Remark int(x) represents the integer part of x.

4.3 COSINE (LCOS)

(1) Description

This function returns cos(x) to FPR1, where x is the value of FPR1.

- Unit: Radian

(2) Object module files to be linked

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

(3) Required stack area size in bytes

6 (including two bytes as a return address from LCOS)

(4) Registers to be used

AX, C, RP2, RP3, RP4, DE, UF (user flag)

(5) Work areas to be used

FPR1, FPR2, FPR3, FPR4, FPR3_X, FPR4_X

(6) Execution time (Internal system clock: 8 MHz, no wait)

Average : 1.72 ms

Maximum: 4.83 ms (cos(6.806e + 38))

(7) Algorithm

The following expression is used to find cos(x):

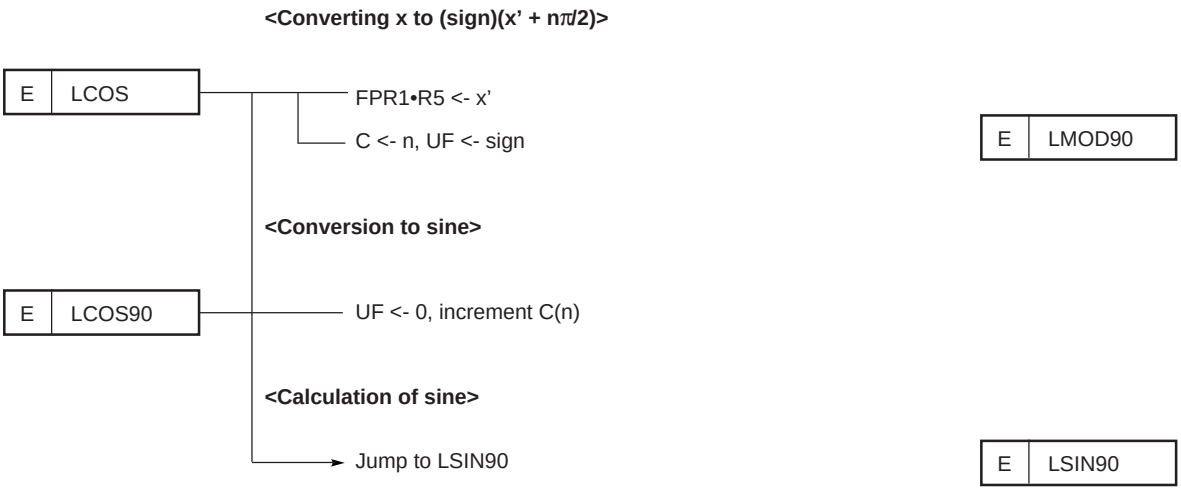
$$\cos(x) = \sin(|x| + \pi/2)$$

(8) Operation

(a) The function finds the quotient (n) and remainder (x') of a division of |x| by $\pi/2$, by using the LMOD90 subroutine.

(b) The function finds the solution, $\sin(x' + (n + 1)\ \pi/2)$, by using the LSIN90 subroutine.

(9) Flowchart



4.4 TANGENT (LTAN)

(1) Description

This function returns tan(x) to FPR1, where x is the value of FPR1.

- Unit: Radian

(2) Object module files to be linked

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

(3) Required stack area size in bytes

8 (including two bytes as a return address from LTAN)

(4) Registers to be used

AX, BC, RP2, RP3, RP4, DE, UF (user flag)

(5) Work areas to be used

FPR1, FPR2, FPR3, FPR4, FPR5, FPR3_X, FPR4_X, FPR5_X

(6) Execution time (Internal system clock: 8 MHz, no wait)

Average : 2.91 ms

Maximum: 6.53 ms (tan(6.806e + 38))

(7) Algorithm

The following expression is used:

$$\tan(x) = \frac{\sin(x)}{\cos(|x|)}$$

(8) Operation

(a) The function finds the quotient (n) and remainder (x') of a division of |x| by $\pi/2$, by using the LMOD90 subroutine. The function also finds the sign (s) of x.

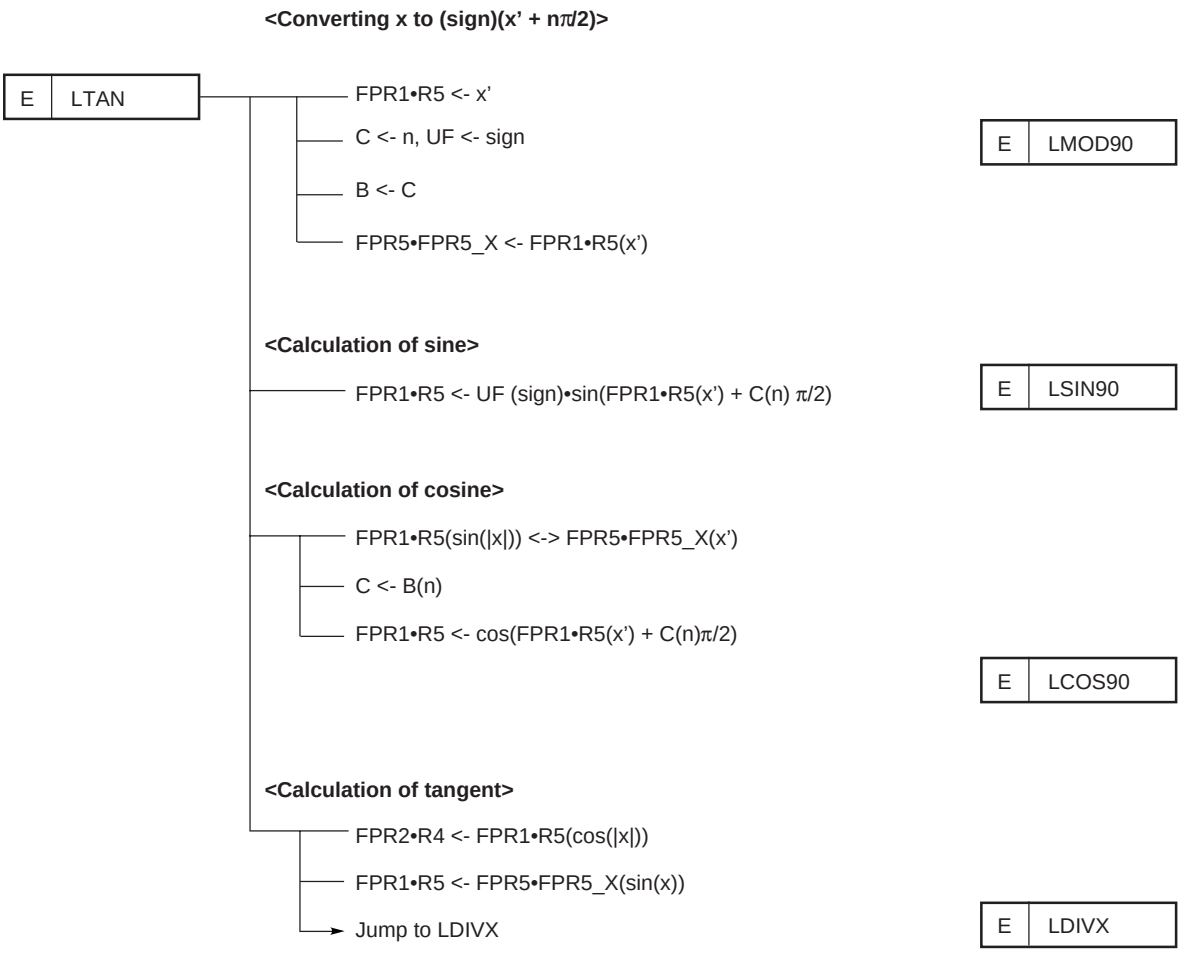
(b) The function finds (s)sin(x' + nπ/2) by using the LSIN90 subroutine.

(c) The function finds cos(x' + nπ/2) by using the LCOS90 subroutine.

(d) The function finds the solution, (s)sin(x' + nπ/2) ÷ cos(x' + nπ/2).

Phase-out/Discontinued

(9) Flowchart



4.5 NATURAL LOGARITHM (LLOG)

(1) Description

This function returns log(x) to FPR1, where x is the value of FPR1.

(2) Object module files to be linked

DFLT, LFLT1, LFLT2, LLD, LLOG, FTOL

(3) Required stack area size in bytes

6 (including two bytes as a return address from LLOG)

(4) Registers to be used

AX, C, RP2, RP3, RP4, UP, DE

(5) Work areas to be used

FPR1, FPR2, FPR3, FPR4, FPR3_X, FPR4_X

(6) Execution time (Internal system clock: 8 MHz, no wait)

Average : 1.54 ms

Maximum: 2.02 ms (log(12345.679))

(7) Algorithm

log(x) can be expanded with the Taylor expansion method as follows:

$$\text{Let } x' = \frac{x - 1}{x + 1}. \text{ Then}$$

$$\log(x) = \log(x' + 1) - \log(1 - x')$$

$$= 2x' + \frac{2}{3}x'^3 + \frac{2}{5}x'^5 + \frac{2}{7}x'^7 + \frac{2}{9}x'^9$$

This expression is theoretically valid for $0 < x < \infty$, but is practically useful only when x is near 1 due to slow convergence.

So with the method indicated below, a range of about -0.17 to 0.17 is used as the range of x' for an approximate expression.

- Let xe = characteristic of x, and xf = mantissa of x.
- When $xf < \sqrt{2}$, xe' = xe, and xf' = xf.
- When $xf \geq \sqrt{2}$, xe' = xe + 1, and xf' = xf/2.
- The following function conversion is performed:
  $\log(x) = xe' \times \log 2 + \log(xf')$

Phase-out/Discontinued

(8) Operation

(a) The function terminates abnormally when $x \leq 0$.

(b) When $xf < \sqrt{2}$, xe' = xe, and xf' = xf.

(c) When $xf \geq \sqrt{2}$, xe' = xe + 1, and xf' = xf/2.

(d) The function finds (xe' $\times$ log2).

(e) The function finds x' from (xf' - 1)/(xf' + 1).

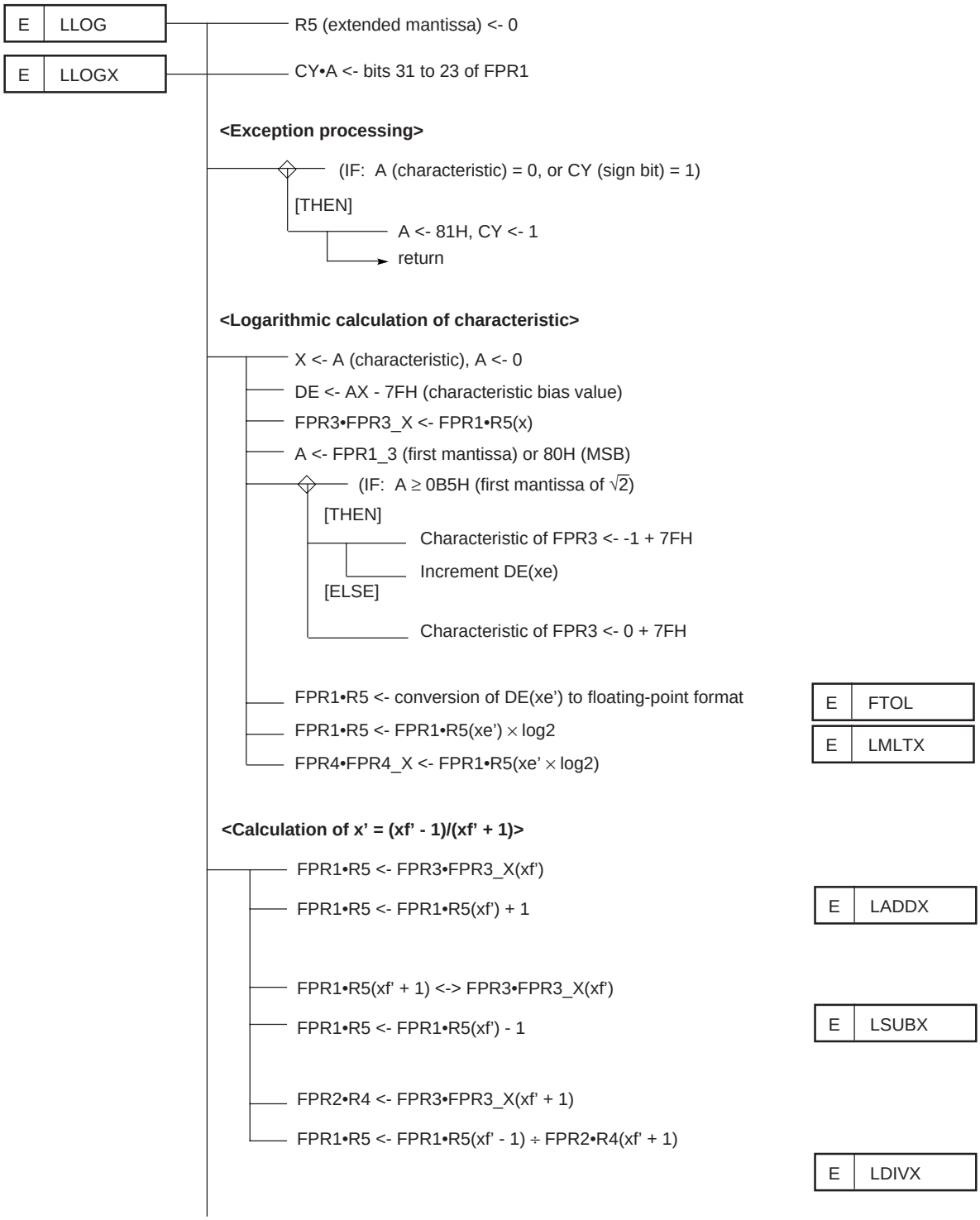
(f) The function replaces the first term (2x') of an approximate polynomial with (xe' $\times$ log2 + 2x') to calculate the approximate polynomial.

(9) Floating-point constant

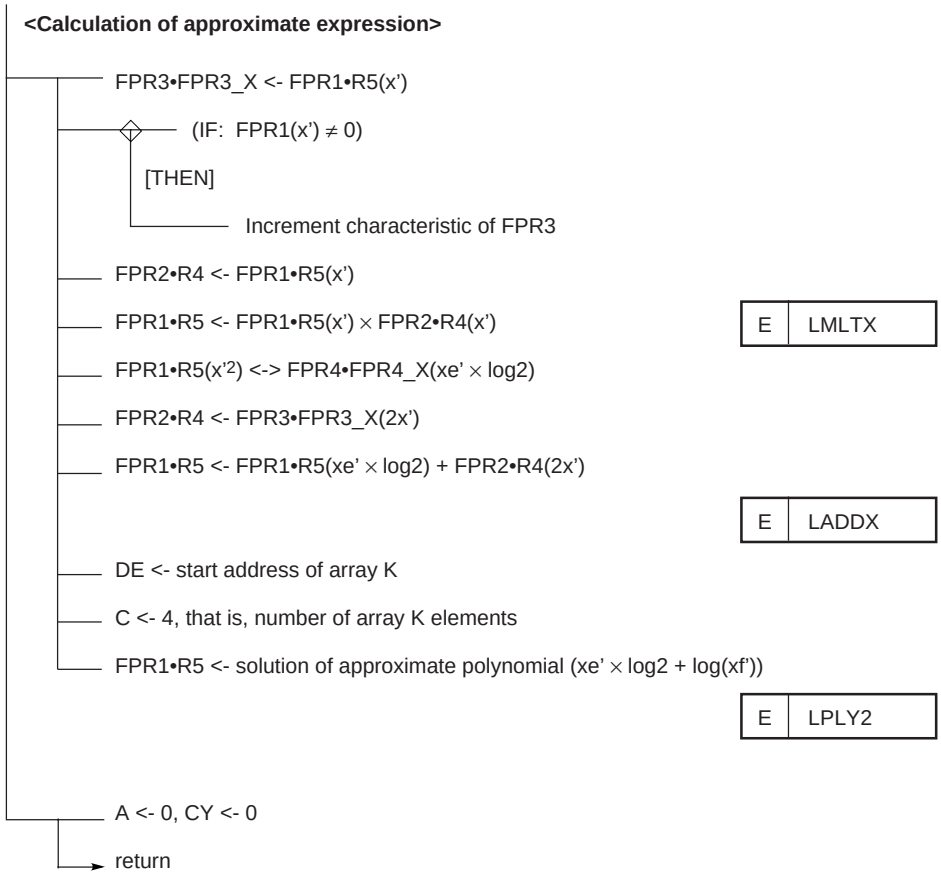
(a) The constants 1 and log2 with an extended mantissa are used.

(b) For the coefficient string of an approximate polynomial, the constants 1/3, 3/5, 5/7, and 7/9 with an extended mantissa are used as array K with four elements.

(10) Flowchart



Phase-out/Discontinued



Remark The label E LLOGX indicates an internal global name used with an arithmetic function to perform a logarithmic calculation involving an extended mantissa.

4.6 COMMON LOGARITHM (LLOG10)

(1) Description

This function returns $\log_{10}(x)$ to FPR1, where x is the value of FPR1.

(2) Object module files to be linked

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

(3) Required stack area size in bytes

8 (including two bytes as a return address from LLOG10)

(4) Registers to be used

AX, C, RP2, RP3, RP4, UP, DE

(5) Work areas to be used

FPR1, FPR2, FPR3, FPR4, FPR3_X, FPR4_X

(6) Execution time (Internal system clock: 8 MHz, no wait)

Average : 1.54 ms

Maximum: 2.08 ms ($\log_{10}$(3.6486432e - 26))

(7) Algorithm

The following expression is used:

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

(8) Operation

(a) The function finds log(x) by using the LLOG function.

(b) The function just terminates abnormally if the LLOG function terminates abnormally.

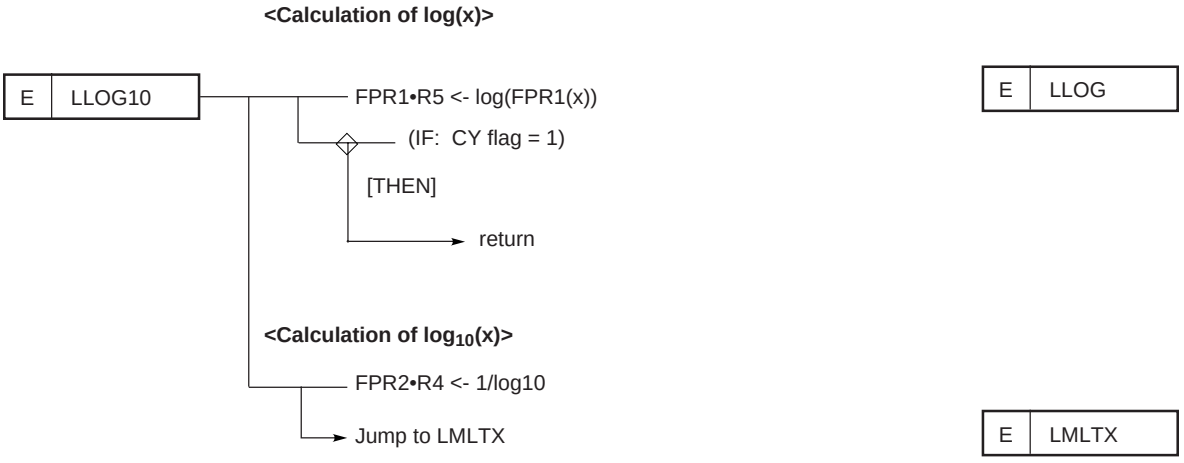
(c) The function finds the solution, log(x)/log10.

(9) Floating-point constant

The constant 1/log10 with an extended mantissa is used.

Phase-out/Discontinued

(10) Flowchart



4.7 EXPONENTIAL FUNCTION (BASE e) (LEXP)

(1) Description

This function returns e^x to FPR1, where x is the value of FPR1.

(2) Object module files to be linked

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

(3) Required stack area size in bytes

6 (including two bytes as a return address from LEXP)

(4) Registers to be used

AX, BC, RP2, RP3, RP4, DE, UF (user flag)

(5) Work areas to be used

FPR1, FPR2, FPR3, FPR4, FPR3_X, FPR4_X

(6) Execution time (Internal system clock: 8 MHz, no wait)

Average : 2.12 ms

Maximum: 2.48 ms ($e^{-0.82129019}$)

(7) Algorithm

The solution of the characteristic is first obtained by converting the base to 2.

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

Remarks 1. floor(x) represents the lower integer of x ($\mathrm{floor}(x) \le x < \mathrm{floor}(x) + 1$).

2. dec(x) represents the fractional part of x ($\mathrm{dec}(x) = x - \mathrm{floor}(x)$; $0 \le \mathrm{dec}(x) < 1$)

Since $1 \le 2^{\mathrm{dec}(x/\log 2)} < 2$, floor(x/log2) is the characteristic of the solution.

The mantissa is found according to the following Taylor approximate expression:

When $\mathrm{dec}(x/\log 2) < 1/2$, $x' = \mathrm{dec}(x/\log 2)$

When $\mathrm{dec}(x/\log 2) \ge 1/2$, $x' = \mathrm{dec}(x/\log 2) - 1$

$$2^{x'} = 1 + \frac{\log 2}{1!}x' + \frac{(\log 2)^2}{2!}x'^2 + \frac{(\log 2)^3}{3!}x'^3 + \bullet\bullet\bullet + \frac{(\log 2)^7}{7!}x'^7$$

Phase-out/Discontinued

Remarks 1. The same mantissa value is obtained even when $2^{x'}$ is multiplied by 1/2. So a range of $-1/2 \leq x' < 1/2$, which is most suitable for the approximate expression, is used.

2. When $dec(x/log2) \geq 1/2$, x' is less than 0 theoretically. However, x' can be equal to 0 due to calculation error. In such a case, $2^{x'} = 1$; an unexpected mantissa is obtained. To cope with this problem, the following expression is used to calculate x':

When $dec(x/log2) \geq 1/2$, $x' = dec(x/log2) - (1 + 2^{-30})$

(8) Operation

(a) The function finds x/log2.

(b) If an overflow occurs:

- When $x < 0$, the function terminates the operation, with 0 as the solution.
- When $x > 0$, the function just terminates abnormally.

(c) The function finds floor(x/log2).

(d) The function terminates the operation, with 0 as the solution, when $floor(x/log2) < -126$. The function terminates abnormally when $floor(x/log2) > 128$.

(e) The function finds a solution for $x' = dec(x/log2)$.

(f) $x' = x' - 1$ when $x' \geq 1/2$.

(g) The function calculates a Taylor approximate expression for $2^{x'}$.

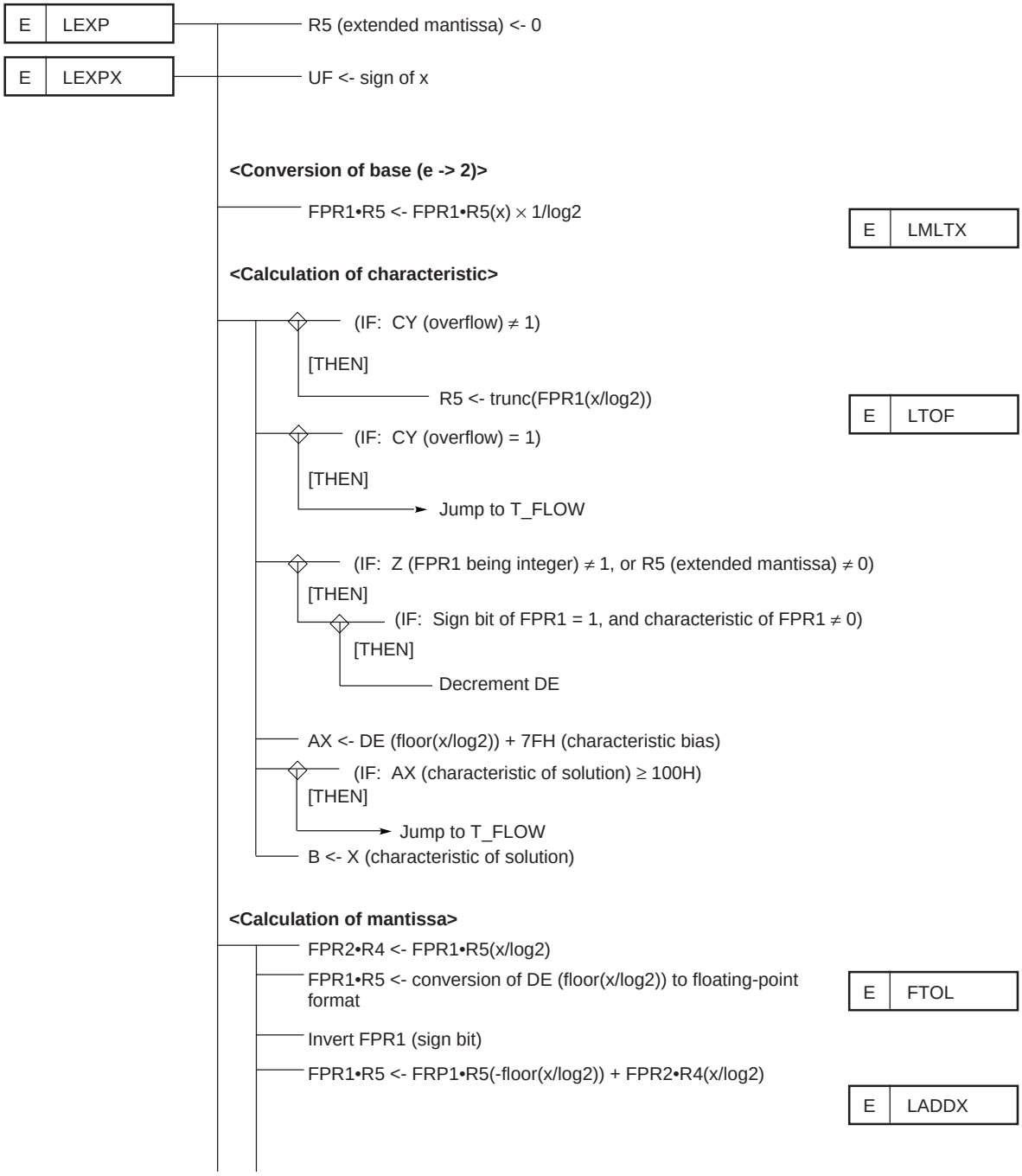
(h) The function includes the characteristic value, floor(x/log2), in the result of approximate expression calculation to obtain the solution.

(9) Floating-point constant

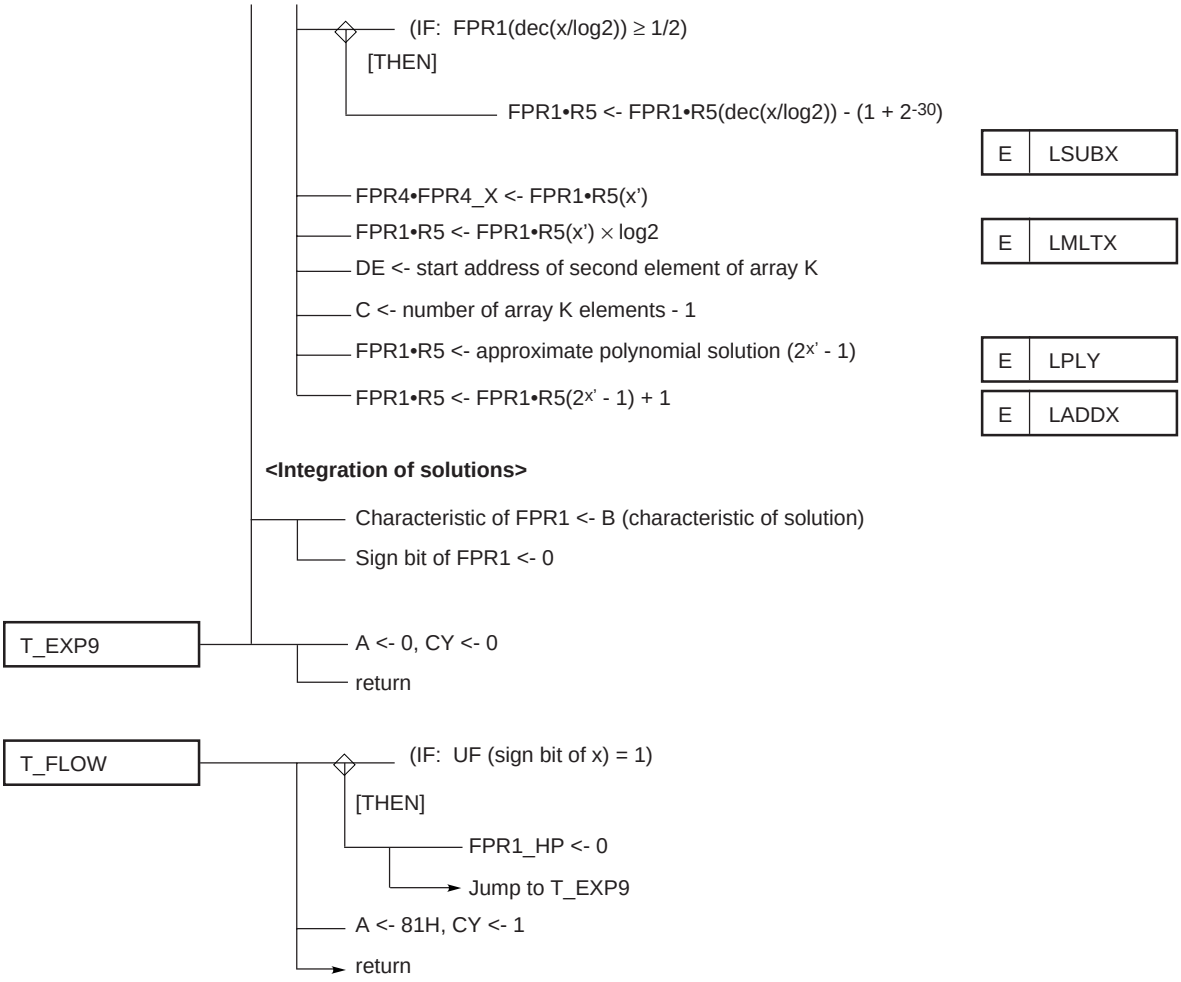
(a) The constants 1 and 1/log2 with an extended mantissa are used.

(b) For the coefficient string of an approximate polynomial, the constants log2, log2/2, log2/3, •••, log2/7 with an extended mantissa are used as array K with seven elements.

(10) Flowchart



Phase-out/Discontinued



Remarks 1. trunc(x) rounds the fractional part of x to 0.

$$\begin{pmatrix} \text{When } x \geq 0,\ \text{trunc}(x) \leq x < \text{trunc}(x) + 1 \\ \text{When } x < 0,\ \text{trunc}(x) - 1 < x \leq \text{trunc}(x) \end{pmatrix}$$

2. The label E | LEXPX indicates an internal global name used with arithmetic function to perform an exponential calculation involving an extended mantissa.

4.8 EXPONENTIAL FUNCTION (BASE 10) (LEXP10)

(1) Description

This function returns 10^x to FPR1, where x is the value of FPR1.

(2) Object module files to be linked

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

(3) Required stack area size in bytes

6 (including two bytes as a return address from LEXP10)

(4) Registers to be used

AX, BC, RP2, RP3, RP4, DE, UF (user flag)

(5) Work areas to be used

FPR1, FPR2, FPR3, FPR4, FPR3_X, FPR4_X

(6) Execution time (Internal system clock: 8 MHz, no wait)

Average : 2.27 ms

Maximum: 2.55 ms ($10^{-0.35975304}$)

(7) Algorithm

The following expression is used:

$$10^x = e^{x\log 10}$$

(8) Operation

(a) The function finds ($x \times \log 10$).

(b) If an overflow occurs as the result of multiplication:

When $x < 0$, the function terminates the operation, with 0 as the solution.

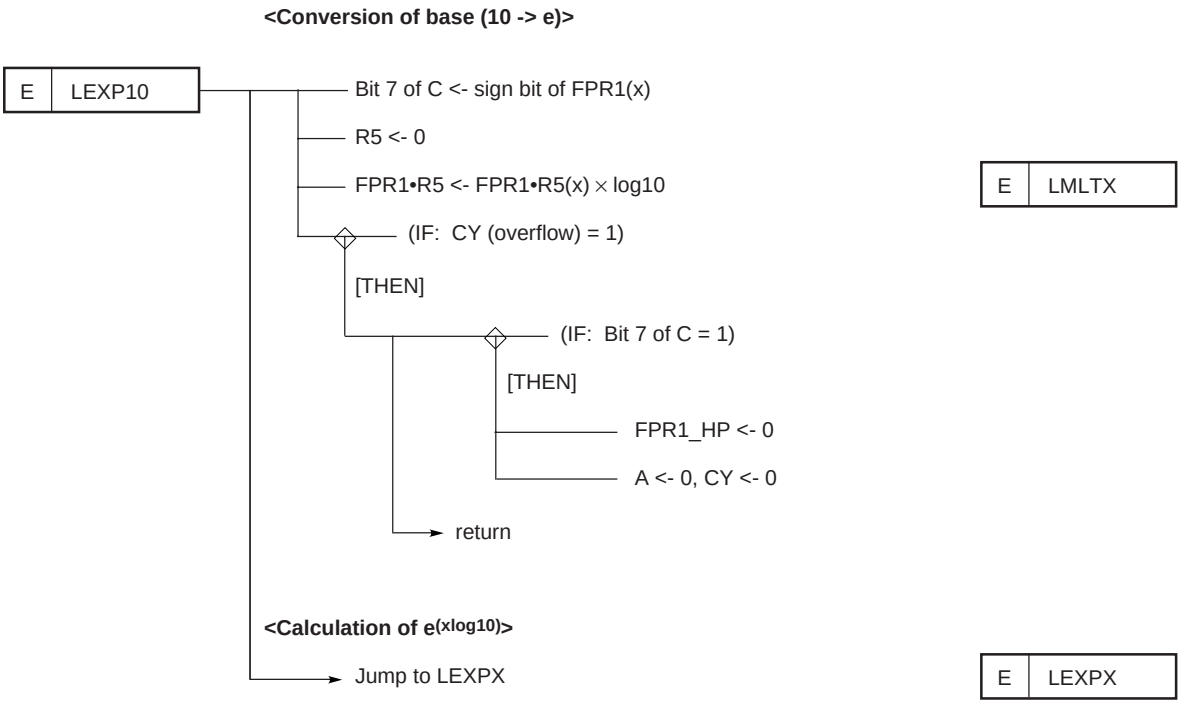
When $x > 0$, the function just terminates abnormally.

(c) The function jumps to the LEXP function.

(9) Floating-point constant

The constant log10 with an extended mantissa is used.

(10) Flowchart



4.9 POWER (LPOW)

(1) Description

This function returns a^b to FPR1, where a is the value of FPR1, and b is the value of FPR2.

(2) Object module files to be linked

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

(3) Required stack area size in bytes

8 (including two bytes as a return address from LPOW)

(4) Registers to be used

AX, BC, RP2, RP3, RP4, UP, DE, HL, UF (user flag)

(5) Work areas to be used

FPR1, FPR2, FPR3, FPR4, FPR5, FPR3_X, FPR4_X

(6) Execution time (Internal system clock: 8 MHz, no wait)

Average : 3.77 ms
Maximum: 4.46 ms (3^3)

(7) Algorithm

The method of calculation depends on the combination of the values of a and b.

a, b	a^b
$a = 0, b \leq 0$	Error
$a = 0, b > 0$	0
$a > 0$	$e^{blog(a)}$
$a < 0, b = 0$	1
$a < 0$, b is a nonzero integer.	
b: Even number	$e^{blog(\|a\|)}$
b: Odd number	$-e^{blog(\|a\|)}$
$a < 0$, b is not an integer.	Error

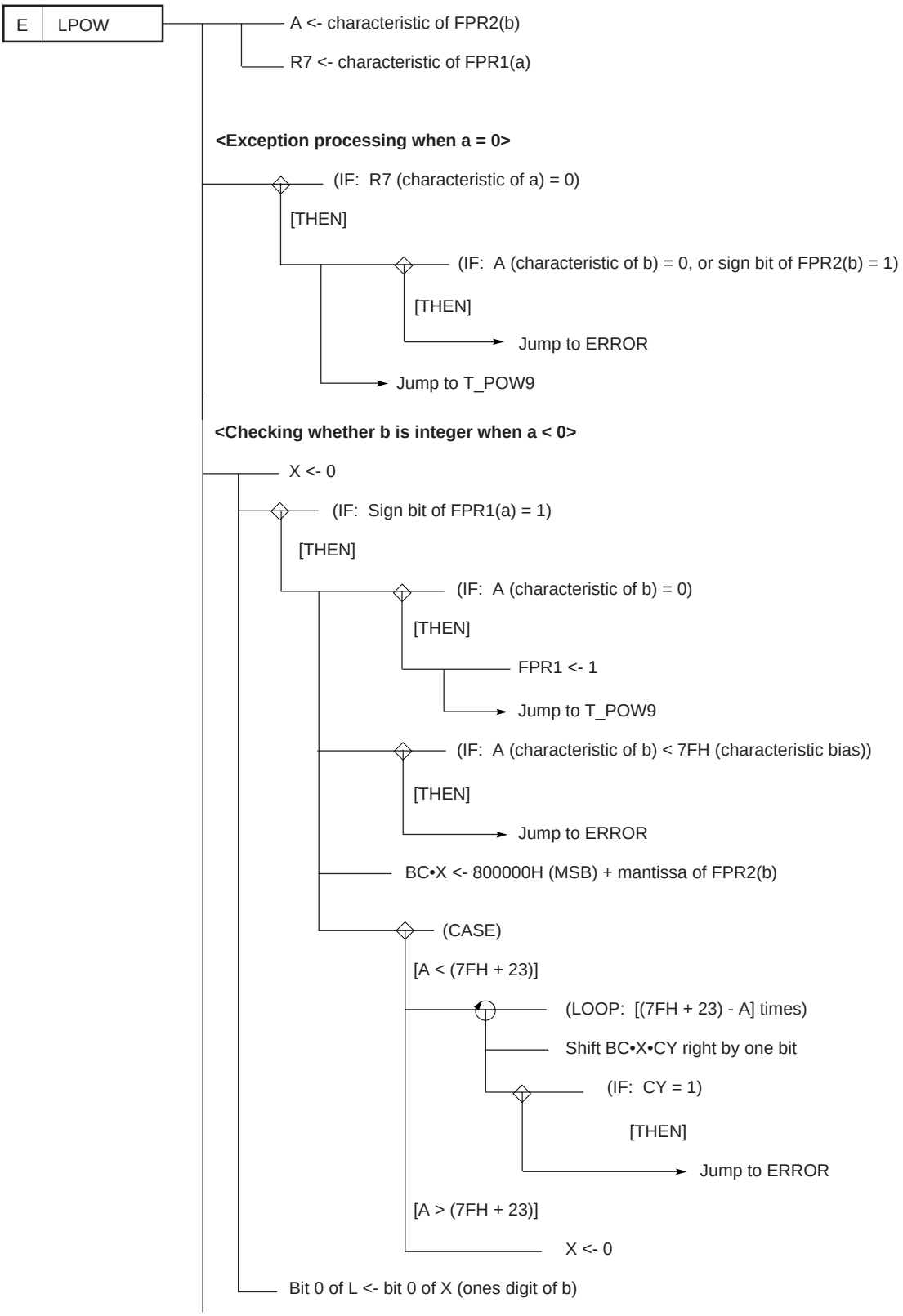
(8) Operation

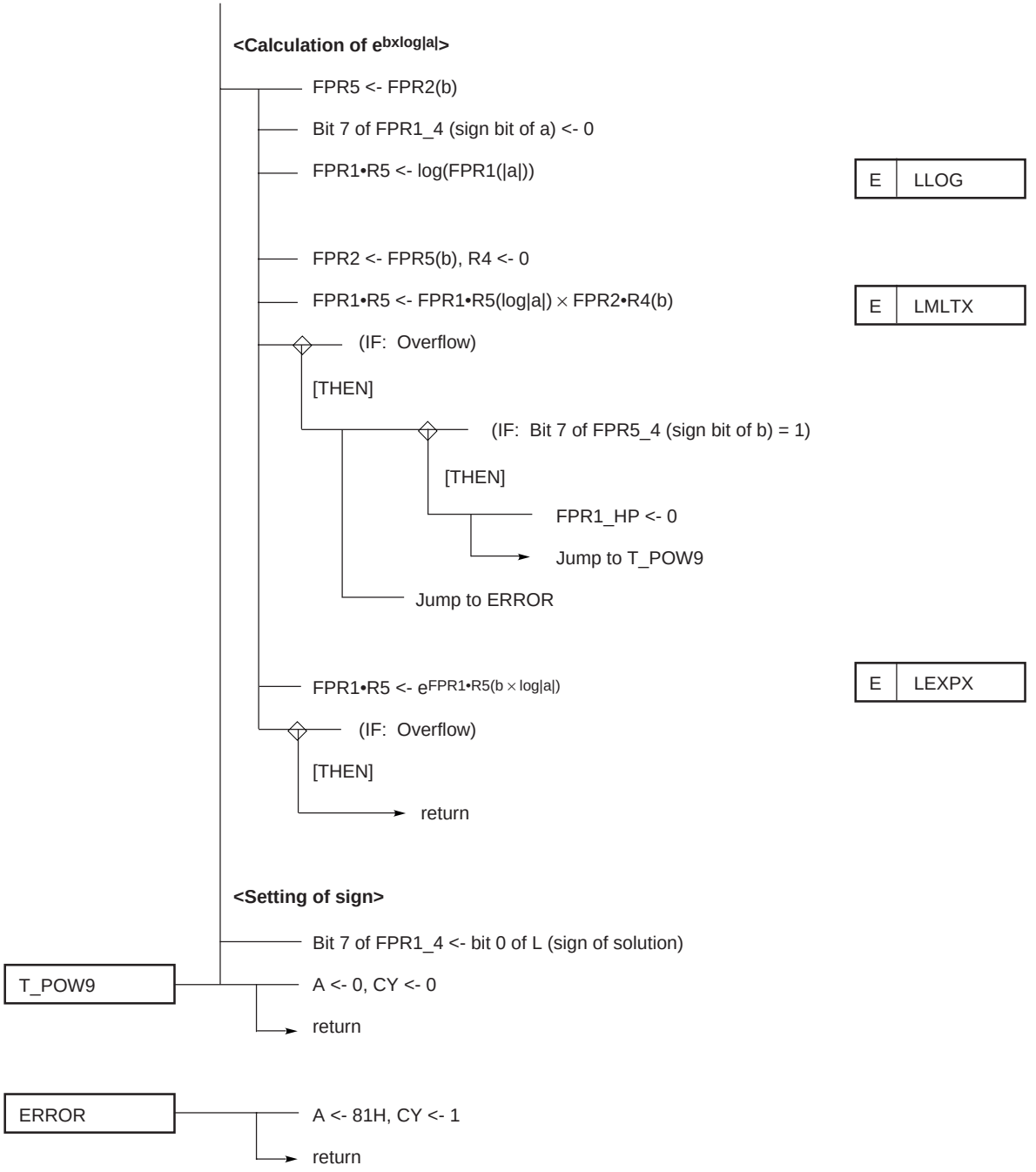
(a) The function terminates abnormally when $a = 0$, and $b \leq 0$.
(b) The function returns 0 as the result of operation when $a = 0$, and $b > 0$.
(c) The function returns 1 as the result of operation when $a < 0$, and $b = 0$.
(d) When $a < 0$, and $b \neq 0$, the function checks whether b is an integer, and checks whether b is an even number or odd number if b is an integer. When b is not an integer, the function terminates abnormally.
(e) The function finds $e^{blog(|a|)}$ by using the LLOG and LEXP functions.
(f) The function inverts the sign of the solution obtained in step (e) when $a < 0$, and b is an odd integer.

(9) Floating-point constant

The constant 1 is used.

(10) Flowchart





4.10 SQUARE ROOT (LSQRT)

(1) Description

This function returns $\sqrt{a}$ to FPR1, where a is the value of FPR1.

(2) Object module files to be linked

DFLT, LFLT1, LLD, LSQRT

(3) Required stack area size in bytes

4 (including two bytes as a return address from LSQRT)

(4) Registers to be used

AX, BC, RP2, RP3, RP4, UP, DE

(5) Work areas to be used

FPR1, FPR2, FPR3, FPR4, FPR3_X, FPR4_X

(6) Execution time (Internal system clock: 8 MHz, no wait)

Average : 613 μs

Maximum: 625 μs ($\sqrt{(0.000001)}$)

(7) Algorithm

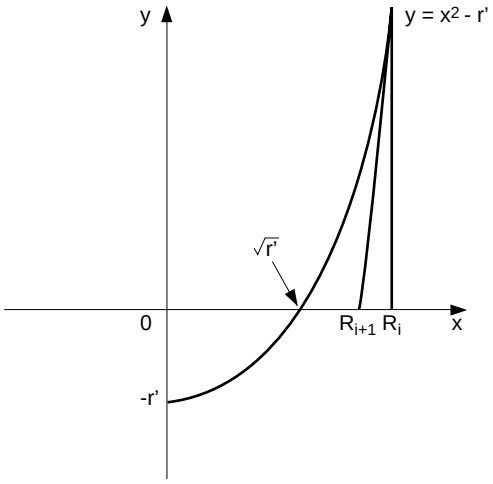
The characteristic, n, of $\sqrt{a}$ is found using the following expression:

$$\sqrt{a} = \sqrt{(r \times 2^{2n})} = \sqrt{r \times 2^{n}} \qquad (1 \leq r < 4)$$

The Newton-Raphson method is used to calculate $\sqrt{r}$.

As shown below, $\sqrt{r'}$ is the x coordinate of the intersection of the quadratic function, $y = x^2 - r'$, with the X-axis.

With R_i used as an approximate value of $\sqrt{r'}$, a tangent to $y = x^2 - r'$ is drawn from point $(R_i, R_i^2 - r')$. Let R_{i+1} be the x coordinate of the intersection of the tangent with the X-axis. Then R_{i+1} is a much closer approximate value of $\sqrt{r'}$ than R_i.



R_{i+1} is found from R_i according to the following expression:

$$R_{i+1} = \frac{R_i}{2} + \frac{r'}{2R_i}$$

With r' = r (when $1 \leq r < 2$) or r' = r/4 (when $2 \leq r < 4$), and the initial approximate value R1 = 1, this function calculates the fifth approximate value R5 to obtain the mantissa of $\sqrt{a}$.

(8) Operation

(a) The function terminates the operation, with 0 as the solution, when a = 0.

(b) The function terminates abnormally when a < 0.

(c) The function finds n' = n + 7FH and r'/2 from the following expressions for the characteristic of a, ae (including a bias of 7FH), and the mantissa of a, af:

- When ae is an odd number: n' = (ae - 7FH)/2 + 7FH = (ae - 1)/2 + 40H, r'/2 = af/2
- When ae is an even number: n' = (ae - 7FH - 1)/2 + 7FH = ae/2 + 3FH, r'/2 = ((af × 2)/4)/2

(d) The function calculates the second approximate value of $\sqrt{r'}$, that is, R2 = 1/2 + r'/2.

(e) The function finds the third, fourth, and fifth approximate values according to the approximate expression.

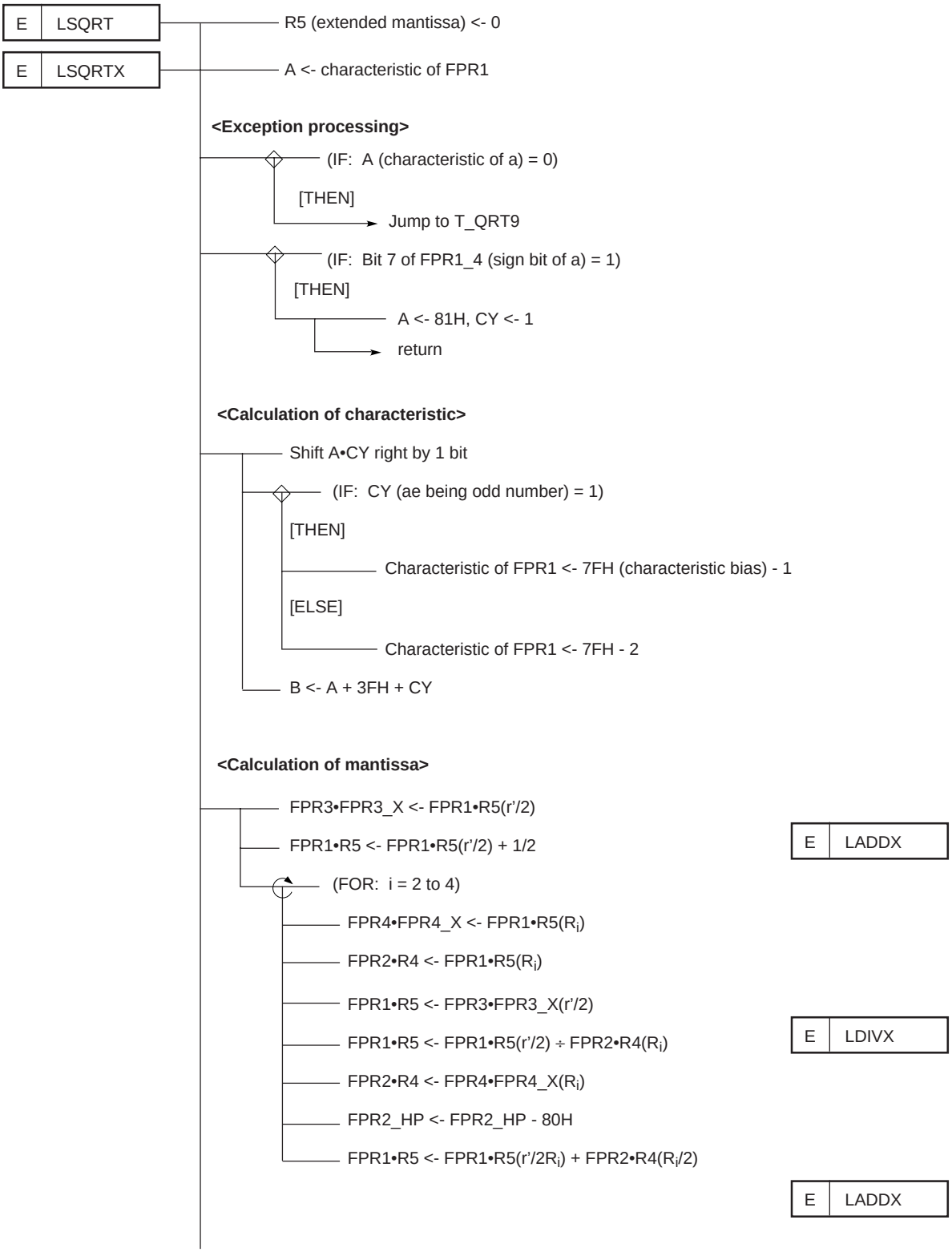
(f) The function obtains the solution by replacing the characteristic of the fifth approximate value with n'.

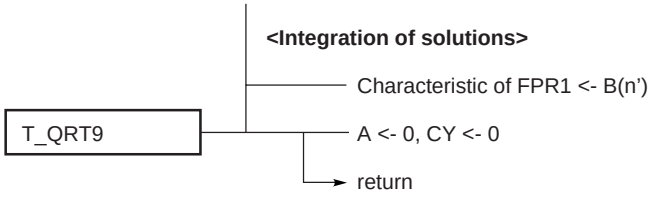
(9) Floating-point constant

The constant 1/2 with an extended mantissa is used.

Phase-out/Discontinued

(10) Flowchart





Remark The label E LSQRTX indicates an internal global name used with an arithmetic function to perform a square root calculation involving an extended mantissa.

Phase-out/Discontinued

4.11 ARCSINE (LASIN)

(1) Description

This function returns arcsin(x) to FPR1, where x is the value of FPR1.

- Valid range of the input value x : -1 to +1
- Range of return values : $-\pi/2$ to $+\pi/2$
- Unit : Radian

(2) Object module files to be linked

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

(3) Required stack area size in bytes

6 (including two bytes as a return address from LASIN)

(4) Registers to be used

AX, BC, RP2, RP3, RP4, UP, DE

(5) Work areas to be used

FPR1, FPR2, FPR3, FPR4, FPR5, FPR3_X, FPR4_X

(6) Execution time (Internal system clock: 8 MHz, no wait)

Average : 2.25 ms

Maximum: 2.74 ms (arcsin(0.98437494))

(7) Algorithm

The value of arcsine(x) is found by converting the function to the function of arctangent according to the following expression:

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

(8) Operation

(a) The function terminates the operation, with $\pi/2$ as the solution when $x = 1$, or $-\pi/2$ as the solution when $x = -1$.

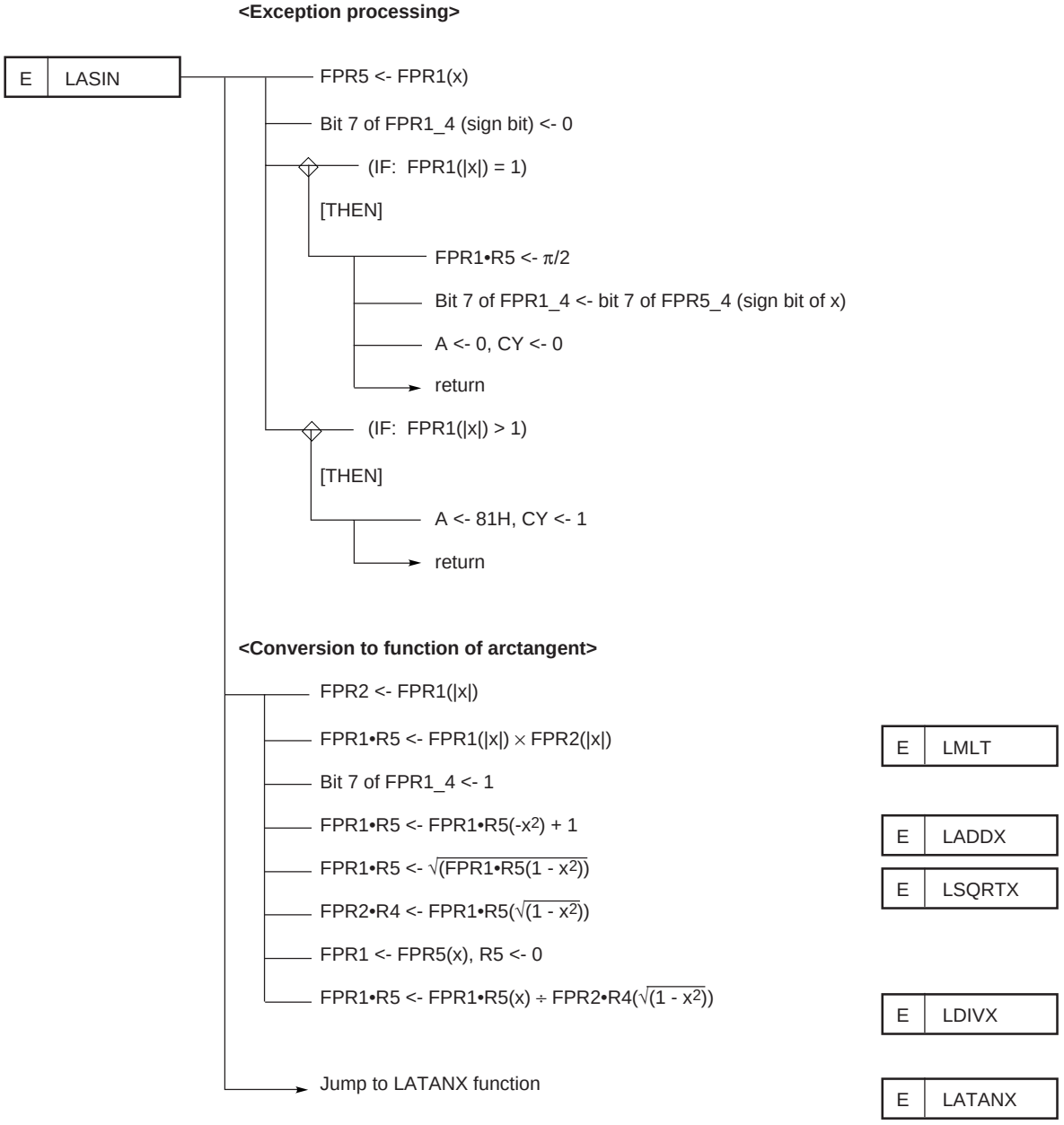
(b) The function terminates abnormally when $|x| > 1$.

(c) The function finds $x/\sqrt{(1 - x^2)}$, and jumps to the LATAN function.

(9) Floating-point constant

The constants $\pi/2$ and 1 with an extended mantissa are used.

(10) Flowchart



4.12 ARCCOSINE (LACOS)

(1) Description

This function returns arccos(x) to FPR1, where x is the value of FPR1.

- Valid range of the input value x : -1 to +1
- Range of return values : 0 to π
- Unit : Radian

(2) Object module files to be linked

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

(3) Required stack area size in bytes

8 (including two bytes as a return address from LACOS)

(4) Registers to be used

AX, BC, RP2, RP3, RP4, UP, DE

(5) Work areas to be used

FPR1, FPR2, FPR3, FPR4, FPR5, FPR3_X, FPR4_X

(6) Execution time (Internal system clock: 8 MHz, no wait)

Average : 2.26 ms

Maximum: 2.82 ms (arccos(0.98437494))

(7) Algorithm

The following expression is used:

$$\arccos(x) = \pi/2 - \arcsin(x)$$

(8) Operation

(a) The function finds arcsin(x) by using the LASIN function.

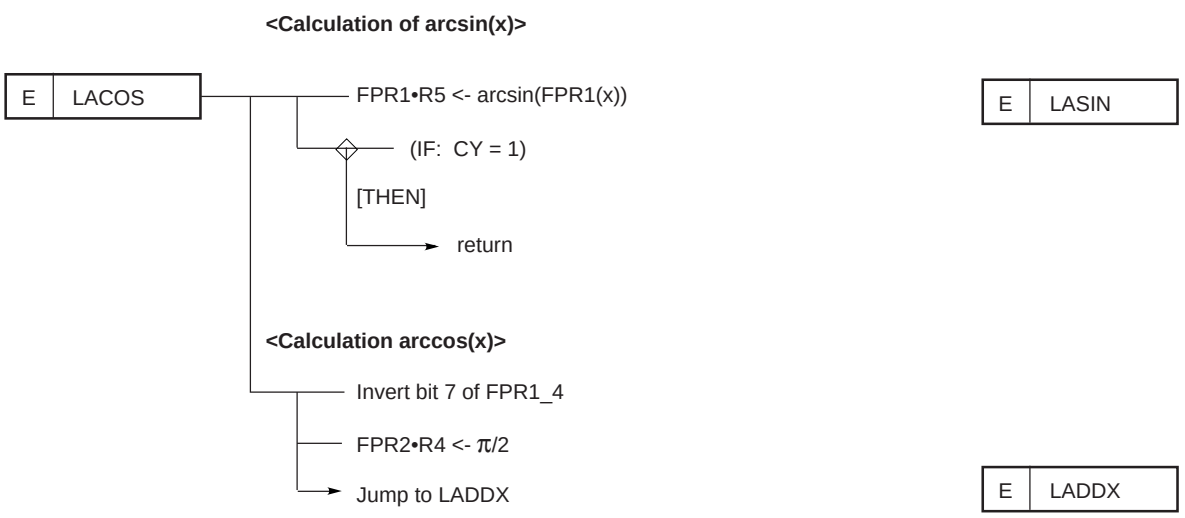
(b) The function just terminates abnormally if the LASIN terminates abnormally.

(c) The function finds the solution from (π/2 - arcsin(x)).

(9) Floating-point constant

The constant π/2 with an extended mantissa is used.

(10) Flowchart



Phase-out/Discontinued

4.13 ARCTANGENT (LATAN)

(1) Description

This function returns arctan(x) to FPR1, where x is the value of FPR1.

- Range of return values: $-\pi/2$ to $+\pi/2$
- Unit : Radian

(2) Object module files to be linked

DFLT, LFLT1, LFLT2, LLD, LATAN, LRCPN

(3) Required stack area size in bytes

6 (including two bytes as a return address from LATAN)

(4) Registers to be used

AX, BC, RP2, RP3, RP4, UP, DE

(5) Work areas to be used

FPR1, FPR2, FPR3, FPR4, FPR3_X, FPR4_X

(6) Execution time (Internal system clock: 8 MHz, no wait)

Average : 1.34 ms

Maximum: 1.85 ms (arctan(3.1415927))

(7) Algorithm

The Best Approximation method of Gouichi Shimauchi of Rikkyo (St. Paul's) University is used.

$$\arctan(x) \fallingdotseq \sum_{i=0}^{n} (a_i \times (4x)^{2i+1})$$

With this function, n = 3 is used to find the coefficients a_0, a_1, a_2, and a_3 as follows:

a_0 = 0.24999 99999 43

a_1 = -0.00520 83303 18

a_2 = 0.00019 52689 40

a_3 = -0.00000 84855 00

Caution The Best Approximation method of Gouichi Shimauchi can be used only when $|x| \leq 1/8$. When $|x| \geq 1/8$, the following method is used:

- When $|x| \geq 1$, let $x' = 1/|x|$. Then, $\arctan(|x|) = \pi/2 - \arctan(1/|x|)$
- Further, when $x' \geq 1/8$, let V and W be as follows:

$$V = \frac{x' - W}{1 + x' \times W}$$

W is a number among 1/8, 3/8, 5/8, and 7/8 which is the closest to x'.
Then $\arctan(x') = \arctan(W) + \arctan(V)$

Remark The addition theorem of the TAN function is used.

(8) Operation

(a) The function saves the sign bit of x, and finds the absolute value of x.
(b) The function finds $1/|x|$ when $|x| \geq 1$.
(c) Then the function finds W from the table below, and calculates V.

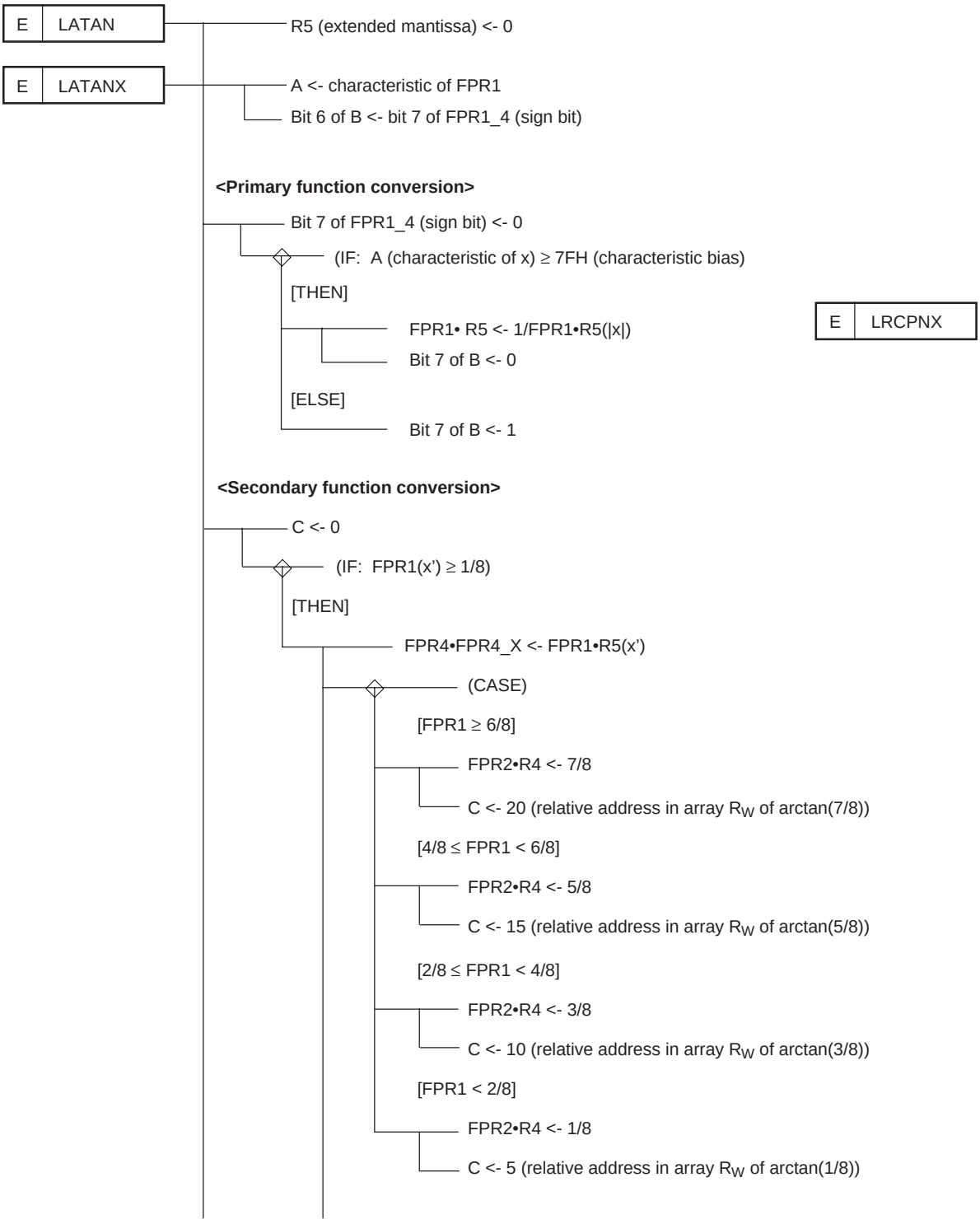
Range of x'	W
Less than 1/8	0
1/8 or greater, less than 1/4	1/8
1/4 or greater, less than 2/4	3/8
2/4 or greater, less than 3/4	5/8
3/4 or greater	7/8

(d) The function calculates the approximate polynomial by replacing the first term ($4a_0V$) of the approximate polynomial of arctan(V) with ($\arctan(W) + 4a_0V$).
(e) The function finds ($\pi/2$ - (solution of the approximate expression)) when $|x| \geq 1$, to obtain the solution of arctan(|x|).
(f) The function assigns the original sign bit of x to the solution of arctan(|x|).

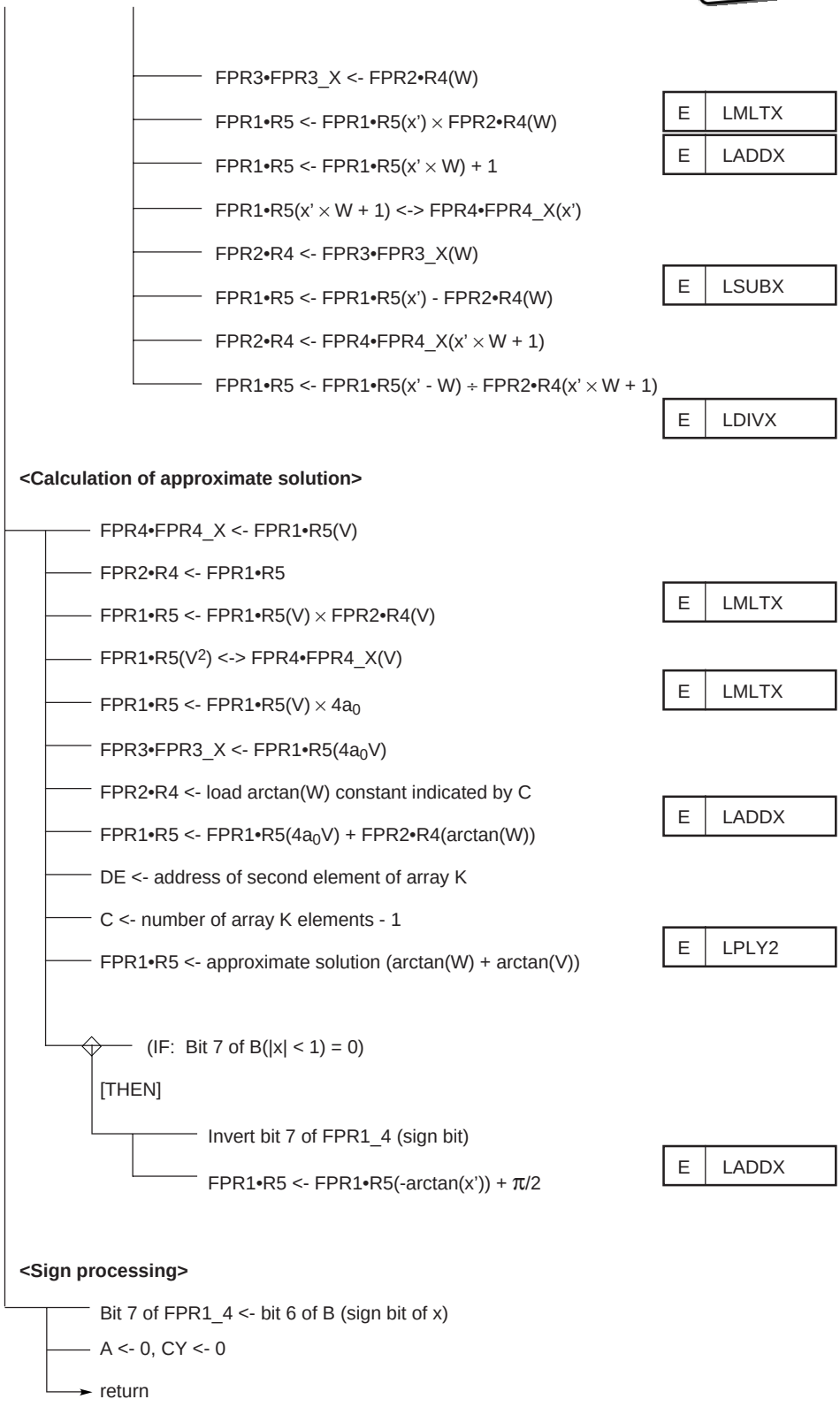
(9) Floating-point constant

(a) The constants 1 and $\pi/2$ with an extended mantissa are used.
(b) The constants arctan(0), arctan(1/8), arctan(3/8), arctan(5/8), and arctan(7/8) with an extended mantissa are used as array R_W with five elements.
(c) For the coefficient string of an approximate polynomial, the constants $4a_0$, $16a_1/a_0$, $16a_2/a_1$, and $16a_3/a_2$ with an extended mantissa are used as array K with four elements.

(10) Flowchart



Phase-out/Discontinued



Remark The label E LATANX indicates an internal global name used with an arithmetic function to perform an arctangent calculation involving an extended mantissa.

Phase-out/Discontinued

4.14 HYPERBOLIC SINE (LHSIN)

(1) Description

This function returns sinh(x) to FPR1, where x is the value of FPR1.

(2) Object module files to be linked

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

(3) Required stack area size in bytes

8 (including two bytes as a return address from LHSIN)

(4) Registers to be used

AX, BC, RP2, RP3, RP4, UP, DE, HL, UF (user flag)

(5) Work areas to be used

FPR1, FPR2, FPR3, FPR4, FPR3_X, FPR4_X

(6) Execution time (Internal system clock: 8 MHz, no wait)

Average : 1.77 ms

Maximum: 2.70 ms (sinh(3.351413))

(7) Algorithm

- When $|x| \geq 0.5$, the following expression is used:

$$\sinh(x) = (\text{sign of } x)\frac{e^{|x|} - e^{-|x|}}{2}$$

- When $|x| < 0.5$, the Taylor approximate expression is used.

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

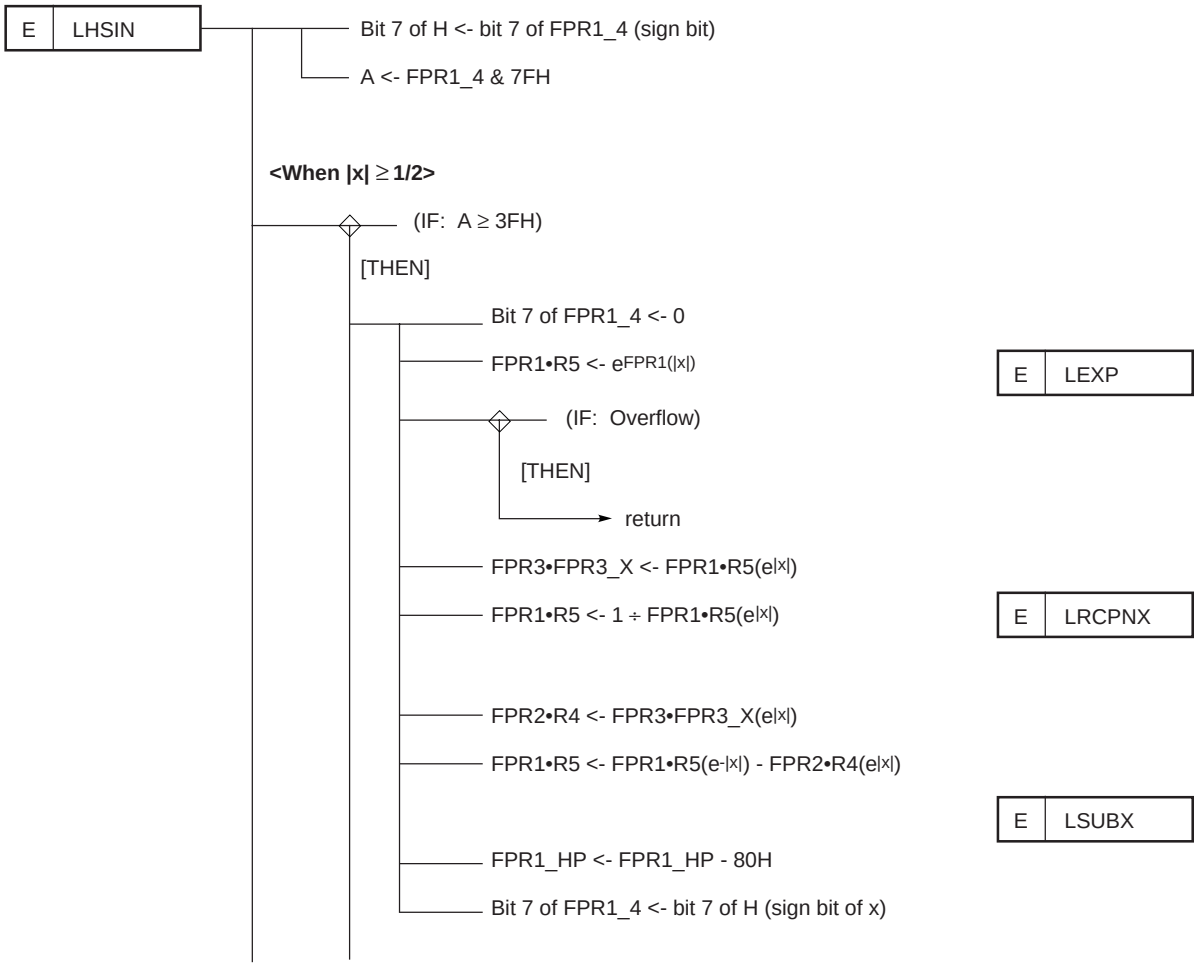
(8) Operation

- When $|x| \geq 0.5$
  (a) The function memorizes the sign of x, and finds the absolute value of x.
  (b) The function finds $e^{|x|}$ by using the LEXP function.
  (c) The function terminates abnormally if $e^{|x|}$ results in an overflow.
  (d) The function finds $(e^{|x|} - 1/e^{|x|})/2$, and assigns the original sign of x to obtain the solution.
- When $|x| < 0.5$
  The function finds sinh(x) by using the Taylor approximate expression.

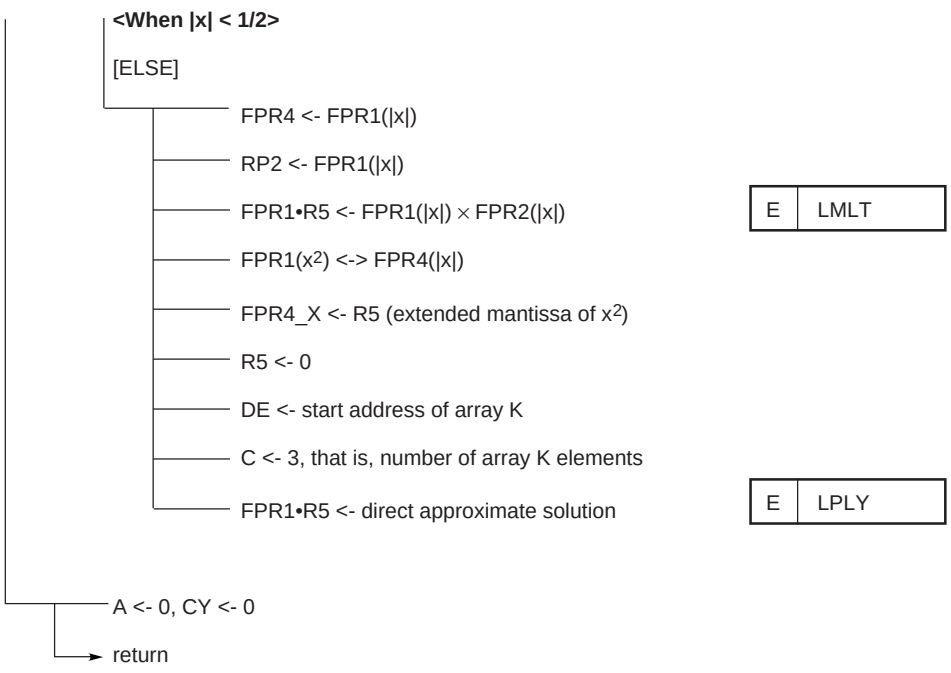
(9) Floating-point constant

For the coefficient string of the Taylor approximate expression, the constants 1/3!, 3!/5!, and 5!/7! with an extended mantissa are used as array K with three elements.

(10) Flowchart



Phase-out/Discontinued



4.15 HYPERBOLIC COSINE (LHCOS)

(1) Description

This function returns cosh(x) to FPR1, where x is the value of FPR1.

(2) Object module files to be linked

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

(3) Required stack area size in bytes

8 (including two bytes as a return address from LHCOS)

(4) Registers to be used

AX, BC, RP2, RP3, RP4, UP, DE, UF (user flag)

(5) Work areas to be used

FPR1, FPR2, FPR3, FPR4, FPR3_X, FPR4_X

(6) Execution time (Internal system clock: 8 MHz, no wait)

Average : 2.27 ms

Maximum: 2.68 ms (cosh(4.0319099))

(7) Algorithm

The following expression is used:

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

(8) Operation

(a) The function finds the absolute value of x.

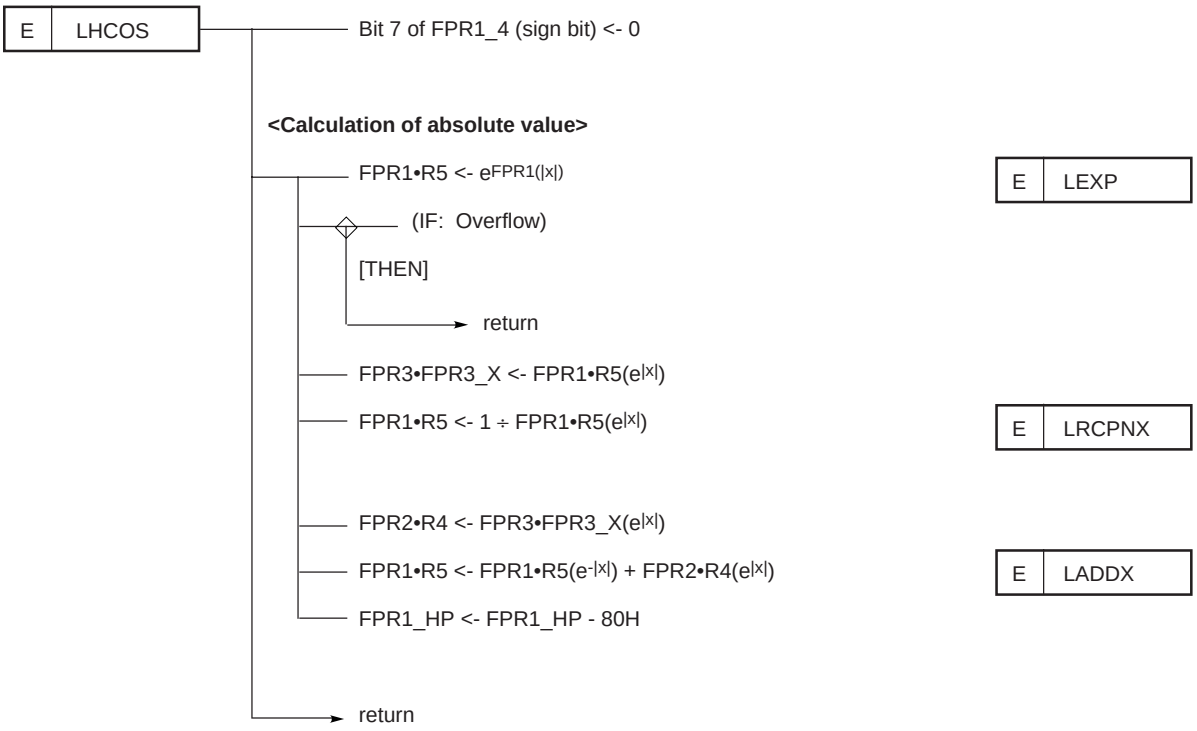
(b) The function finds $e^{|x|}$ by using the LEXP function.

(c) The function terminates abnormally if $e^{|x|}$ results in an overflow.

(d) The function finds the solution, $(e^{|x|} + 1/e^{|x|})/2$.

Phase-out/Discontinued

(9) Flowchart



4.16 HYPERBOLIC COSINE (LHTAN)

(1) Description

This function returns tanh(x) to FPR1, where x is the value of FPR1.

(2) Object module files to be linked

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

(3) Required stack area size in bytes

10 (including two bytes as a return address from LHTAN)

(4) Registers to be used

AX, BC, RP2, RP3, RP4, UP, DE, HL, UF (user flag)

(5) Work areas to be used

FPR1, FPR2, FPR3, FPR4, FPR5, FPR3_X, FPR4_X, FPR5_X

(6) Execution time (Internal system clock: 8 MHz, no wait)

Average : 4.09 ms

Maximum: 5.44 ms (tanh(9.4484739))

(7) Algorithm

The following expression is used:

$$\tanh(x) = \frac{\sinh(x)}{\cosh(x)}$$

(8) Operation

(a) The function finds cosh(x) by using the LHCOS function.

(b) If cos(x) results in an overflow:

- When x > 0, the function terminates the operation, with 1 as the solution.
- When x < 0, the function terminates the operation, with -1 as the solution.

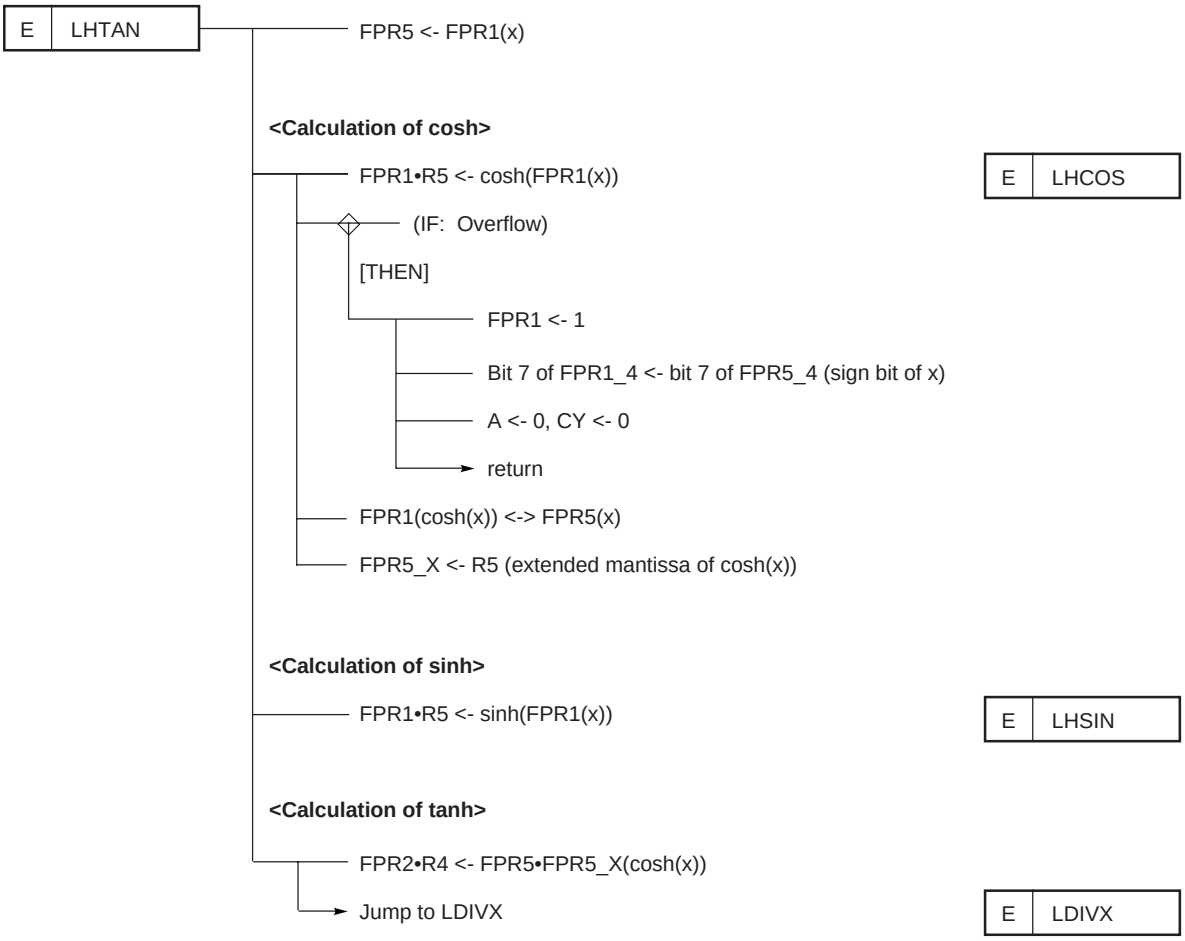
(c) The function finds sinh(x) by using the LHSIN function.

(d) The function finds the solution, sinh(x)/cosh(x).

(9) Floating-point constant

The constant 1 is used.

(10) Flowchart



4.17 ABSOLUTE FUNCTION (LABS)

(1) Description

This function finds the absolute value of FPR1, then returns the result to FPR1.

(2) Object module files to be linked

DFLT, LABS

(3) Required stack area size in bytes

2 (two bytes only for a return address from LABS)

(4) Registers to be used

A

(5) Work areas to be used

FPR1

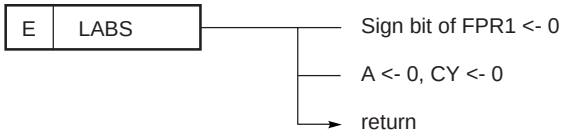
(6) Execution time (Internal system clock: 8 MHz, no wait)

6 μs

(7) Operation

The function sets the sign bit of FPR1 to 0.

(8) Flowchart



4.18 INVERSE FUNCTION (LRCPN)

(1) Description

This function finds the reciprocal of the value of FPR1, then returns the result to FPR1.

(2) Object module files to be linked

DFLT, LFLT1, LLD, LRCPN

(3) Required stack area size in bytes

4 (including two bytes as a return address from LRCPN)

(4) Registers to be used

AX, RP2, RP3, RP4, UP, DE

(5) Work areas to be used

FPR1, FPR2

(6) Execution time (Internal system clock: 8 MHz, no wait)

Average : 96.7 µs

Maximum: 100 µs (1/(-8.5070592e + 37))

(7) Operation

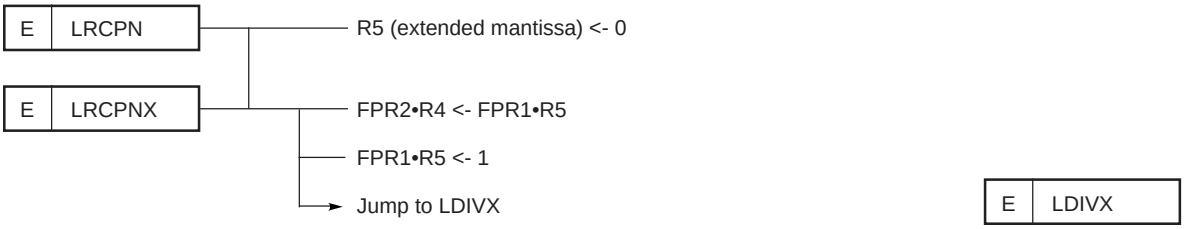
(a) The function transfers the value of FPR1 to FPR2, and sets the constant 1 in FPR1.

(b) The function jumps to the LDIV function.

(8) Floating-point constant

The constant 1 with an extended mantissa is used.

(9) Flowchart



[MEMO]

CHAPTER 5 COORDINATE-CONVERSION FUNCTIONS

The following functions are provided for converting coordinates:

(1) Function for converting polar coordinates to Cartesian coordinates (POTORA)
Polar coordinates (r, θ) are converted to Cartesian coordinates (x, y).
Register FPR1 is used for temporarily storing coordinate r or x. Register FPR2 is used for temporarily storing coordinate θ or y.

(2) Function for converting Cartesian coordinates to polar coordinates (RATOPO)
Cartesian coordinates (x, y) are converted to Polar coordinates (r, θ).
Register FPR1 is used for temporarily storing coordinate x or r. Register FPR2 is used for temporarily storing coordinate y or θ.

77

5.1 FUNCTION FOR CONVERTING POLAR COORDINATES TO CARTESIAN COORDINATES (POTORA)

(1) Description

This function converts polar coordinates r stored in FPR1 and θ stored in FPR2 to Cartesian coordinates and returns coordinate x to FPR1 and coordinate y to FPR2.

- Unit for coordinate θ: Radians

(2) Object module files to be linked

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

(3) Required stack area size in bytes

8 (including an area of two bytes for the return address from POTORA)

(4) Registers to be used

AX, BC, RP2, RP3, RP4, UP, DE, HL, UF (user flag)

(5) Work areas to be used

FPR1, FPR2, FPR3, FPR4, FPR5, FPR3_X, FPR4_X, FPR5_X

(6) Execution time (internal system clock: 8 MHz, no wait)

Average : 3.19 ms

Maximum: 6.62 ms (r = 0.5, θ = 6.806e + 38)

(7) Algorithm

The following equations are used for the conversion.

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

(8) Operation

(a) If r is 0, this function sends back coordinates (0, 0).

(b) If r is negative, the function terminates abnormally.

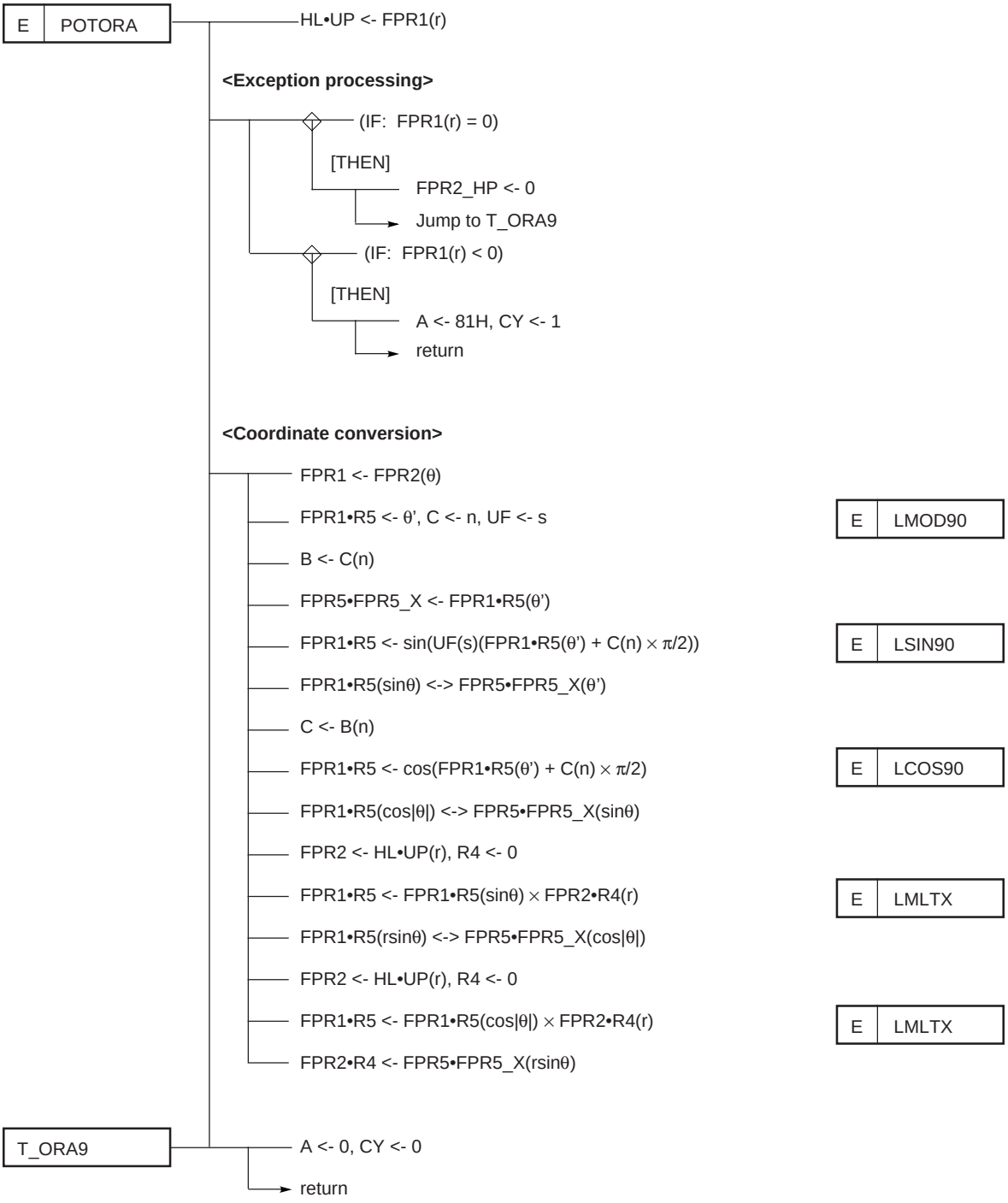
(c) The notation of θ is converted to $s(\theta' + n\pi/2)$ using the LMOD90 function,
where
s: sign of θ
n: integer
$0 \leq \theta' < \pi/2$

(d) The value of $\sin(s(\theta' + n\pi/2))$ is calculated using the LSIN90 function. The value of $\cos(\theta' + n\pi/2)$ is calculated using the LCOS90 function.

(e) The value of $r \times \cos(\theta' + n\pi/2)$ is calculated and the result is x. The value of $r \times \sin(s(\theta' + n\pi/2))$ is calculated and the result is y.

Phase-out/Discontinued

(9) Flowchart



5.2 FUNCTION FOR CONVERTING CARTESIAN COORDINATES TO POLAR COORDINATES (RATOPO)

(1) Description

This function converts Cartesian coordinates x stored in FPR1 and y stored in FPR2 to polar coordinates and returns coordinate r to FPR1 and coordinate θ to FPR2.

- Range of θ : $-\pi$ to $+\pi$
- Unit for coordinate θ: Radians

(2) Object module files to be linked

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

(3) Required stack area size in bytes

8 (including an area of two bytes for the return address from RATOPO)

(4) Registers to be used

AX, BC, RP2, RP3, RP4, UP, DE, H, UF (user flag)

(5) Work areas to be used

FPR1, FPR2, FPR3, FPR4, FPR5, FPR3_X, FPR4_X, FPR5_X

(6) Execution time (internal system clock: 8 MHz, no wait)

Average : 2.39 ms

Maximum: 2.92 ms (x = -38.408036, y = -12.227133)

(7) Algorithm

The following equations are used for the conversion.

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

(8) Operation

(a) If both x and y are 0, this function returns immediately.

(b) The value of $x^2 + y^2$ is calculated.

(c) If the value calculated above overflows, the function terminates abnormally.

(d) The value of y/x is calculated.

(e) If the value calculated above overflows, θ is assumed to be as follows:

- When y is positive, $\theta = \pi/2$
- When y is negative, $\theta = -\pi/2$

(f) When the value of y/x is within the limits and x is not negative, the value of arctan(y/x), θ, is calculated using the LATAN function.

(g) When the value of y/x is within the limits and x is negative, the value of arctan(y/x) is calculated. Value θ is assumed to be as follows:

- When y is not negative, θ is assumed to be the calculated result plus π.
- When y is negative, θ is assumed to be the calculated result minus π.

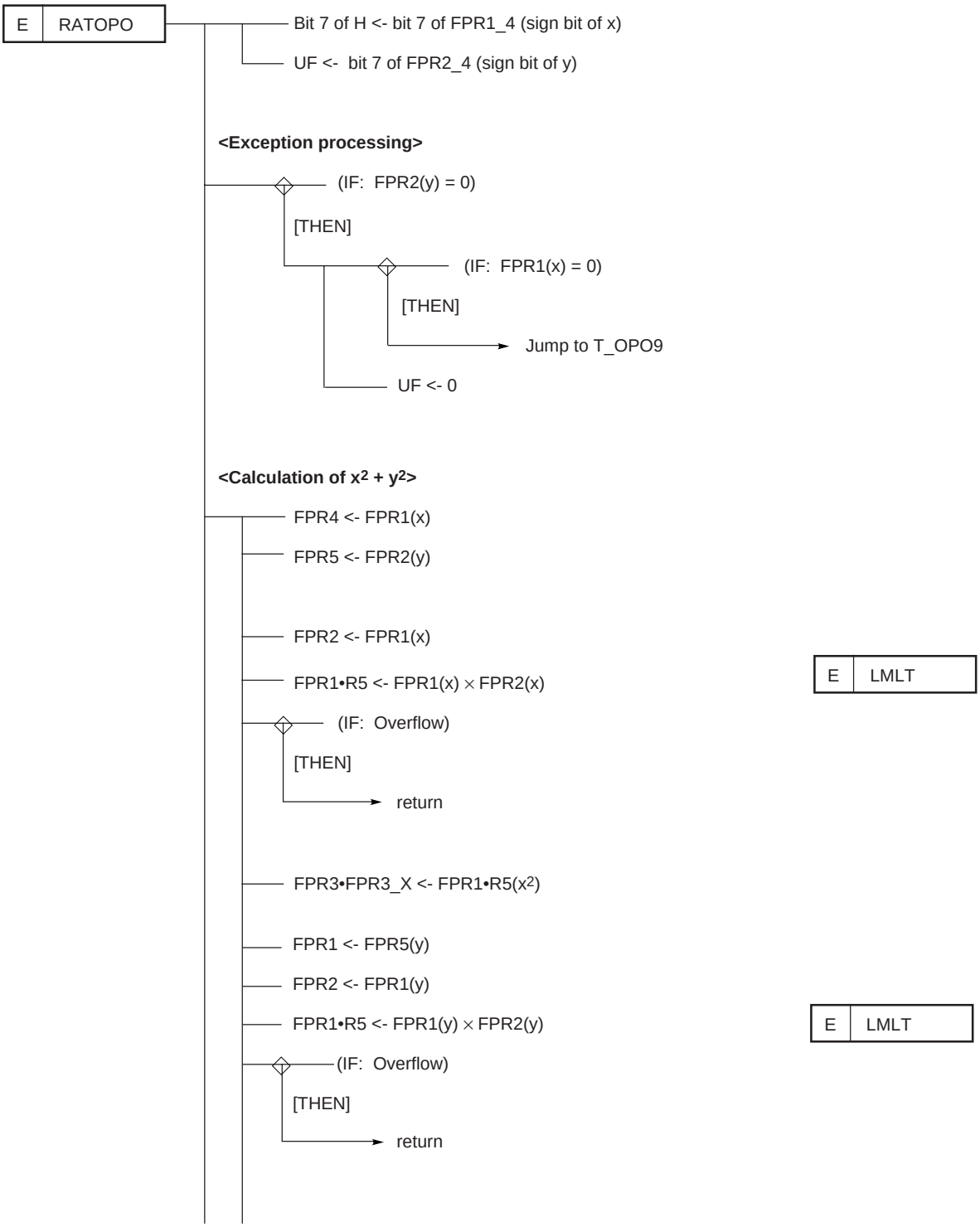
(h) The value of $\sqrt{(x^2 + y^2)}$, r, is calculated.

Phase-out/Discontinued

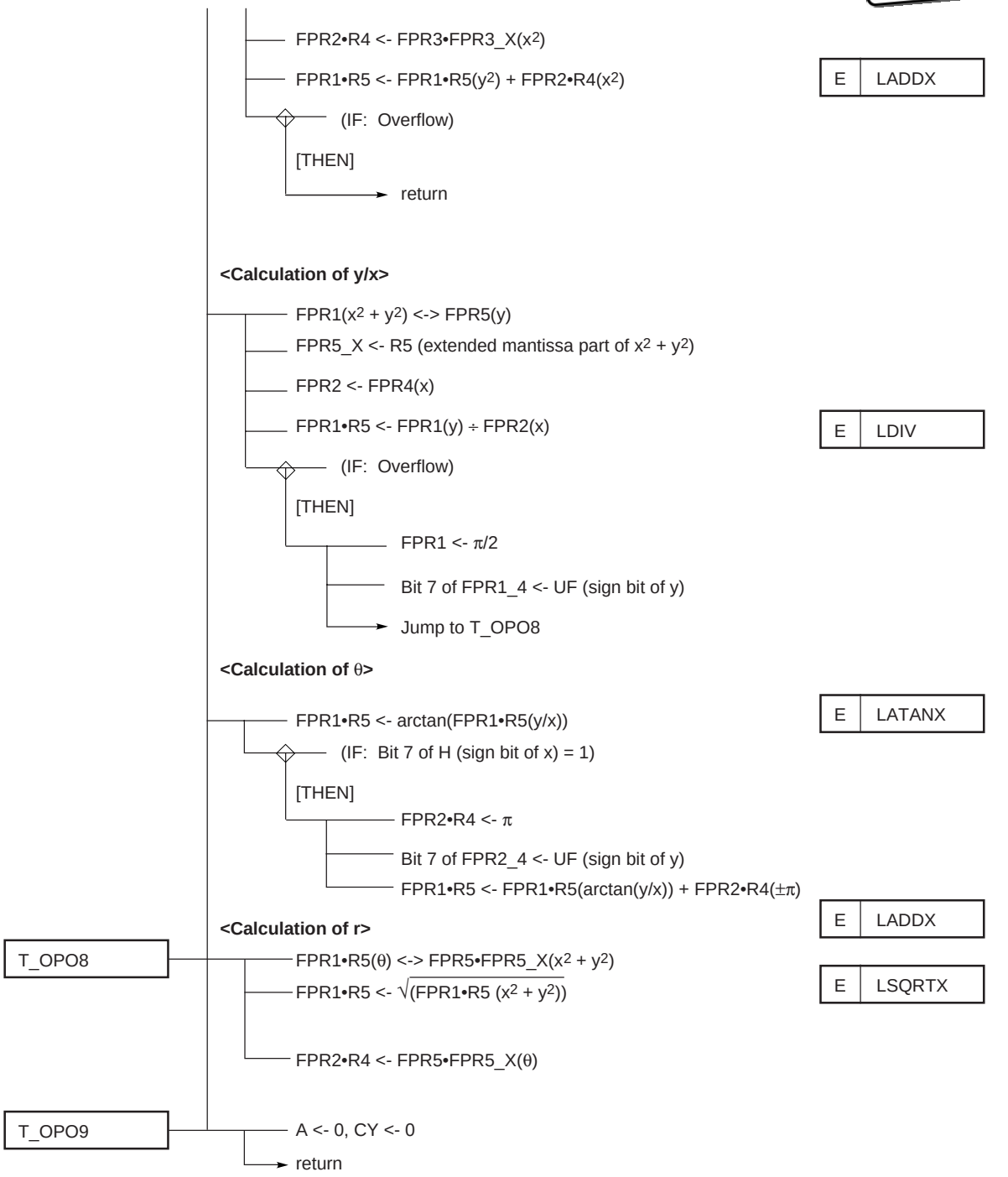
(9) Floating-point constant

The values of π/2 and π (π is expressed in the expended format) are used as floating-point constants.

(10) Flowchart



Phase-out/Discontinued



CHAPTER 6 FUNCTIONS FOR CONVERTING DATA TYPES

The following functions are provided for converting data types.

(1) Function for converting character strings to floating-point codes (ATOL)

The character string whose starting address is stored in the HL register is converted to floating-point code. The result is stored in FPR1.

(2) Function for converting floating-point codes to character strings (LTOA)

The value stored in the FPR1 is converted to a character string. The result is stored in the areas starting from the address indicated by the HL register.

(3) Function for converting two-byte integers to floating-point codes (FTOL)

The value stored in the DE register as a two-byte integer with a sign is converted to a floating-point code. The result is stored in FPR1.

(4) Function for converting floating-point codes to two-byte integers (LTOF)

The value stored in FPR1 is converted to a signed two-byte integer. The result is stored in the DE register.

6.1 FUNCTION FOR CONVERTING CHARACTER STRINGS TO FLOATING-POINT CODES (ATOL)

(1) Description

This function converts the character string whose starting address is stored in the HL register to floating-point code. The result is stored in FPR1.

(2) Object module files to be linked

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

(3) Required stack area size in bytes

10 (including an area of two bytes for the return address from ATOL)

(4) Registers to be used

AX, BC, RP2, RP3, RP4, UP, DE, HL, UF (user flag)
(The value stored in the HL register is retained.)

(5) Work areas to be used

FPR1, FPR2, FPR3, FPR4, FPR5, FPR3_X, FPR4_X, FPR5_X

(6) Execution time (internal system clock: 8 MHz, no wait)

Average : 3.20 ms
Maximum: 3.95 ms (“0.00000000000000000117549428E - 20”)

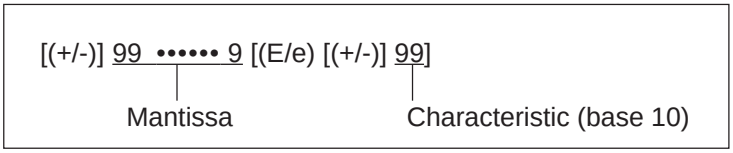
(7) Characters used in character strings

The following characters can be used in character strings.

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

(8) Character-string format

The format of a character string is as follows:



Remark [ ] : Can be omitted
9 : Numbers from 0 to 9
(/) : Either is selected

Example "-99.8" = -99.8
".007e0" = .007
"0998e-03" = 998×10^{-3}

(9) Restrictions on character strings

If this function converts character strings which do not conform to the following rules, an error occurs.

(a) The character string must end with Δ (space) or NUL.
(b) Characters other than those specified in Item (7) cannot be used.
(c) Character strings used for the mantissa may contain one "." and must contain at least one numeral. The maximum character length is 27.
(d) The length of character strings used for a characteristic must be one or two.
(e) Converted values must range from 2^{129} to -2^{129}.

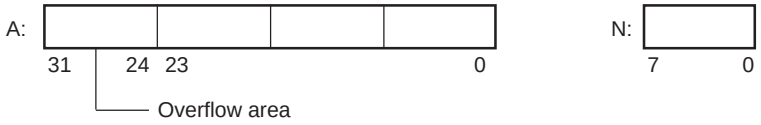
Remark When the mantissa is 0, the character string for the characteristic is ignored and the converted result is 0.

(10) Operation

(a) When the first character is "-", this function regards the sign of the character string as minus. When the first character is not "-", the function regards the sign as plus.
(b) The decimal places of the string are calculated from the character string for the mantissa. Let F represent the result.
(c) Mantissa A is calculated in the four-byte integer type from the character string, A_1, A_2, ••••••, A_n, which expresses the mantissa without the decimal point.

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

The calculation method is as follows:



(i) Let the initial values of A and N be 0.
(ii) A_1 is added to A. If A_1 does not exist, the function terminates abnormally.
(iii) When the overflow bits represent 0, A is multiplied by 10 and A_k is added.
(iv) If the overflow bits do not represent 0, the value of A_k is ignored and 1 is added to N.
(v) Steps (iii) to (iv) are repeated for k (= 2 to n) times.
(vi) If n exceeds 27, the function terminates abnormally.

(d) Mantissa A calculated in step (c) is normalized in the floating-point notation and the stored sign is prefixed to A. Let the result be A'.

(e) Characteristic B is calculated from the character string for the characteristic $(+/-)B_1B_2$.

(f) The number of decimal places F and the ignored number of places in the mantissa N are added to characteristic B to obtain the actual characteristic B' (= B - F + N).

(g) The converted result is calculated as follows using the values of A' and B'.

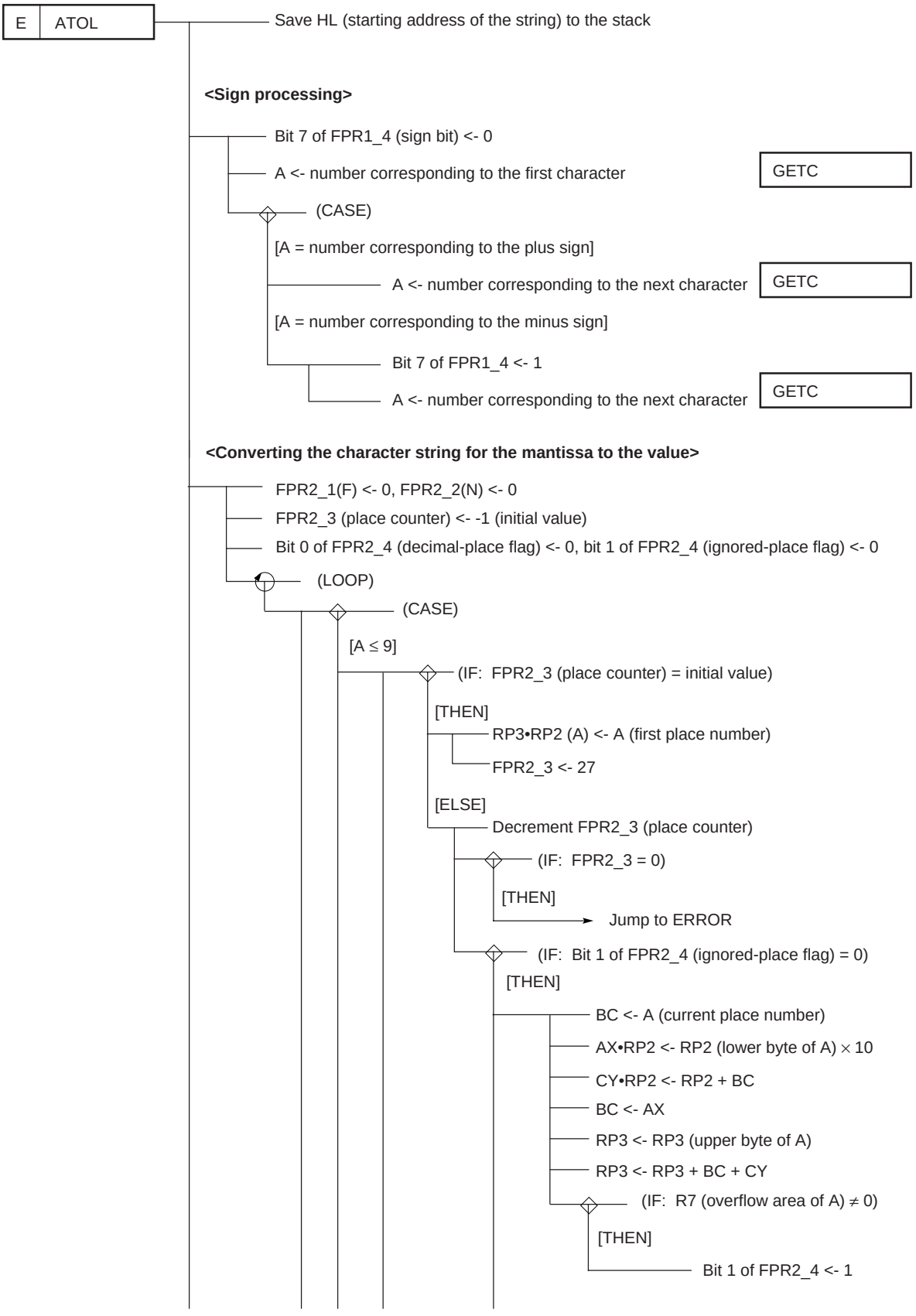
$$
\begin{aligned}
& A' \times 10^{B'} \\
= \; & A' \times 2^{\log_2 10 \times B'} \\
= \; & A' \times 2^{\mathrm{dec}(\log_2 10 \times B')} \times 2^{\mathrm{int}(\log_2 10 \times B')} \\
= \; & A' \times e^{\log 2 \times \mathrm{dec}(\log_2 10 \times B')} \times 2^{\mathrm{int}(\log_2 10 \times B')}
\end{aligned}
$$

Remark Function dec(x) represents the decimal fraction part of x. Function int(x) represents the integer part of x.

(11) Floating-point constant

The values of $\log_2 10$ with the extended mantissa part and log2 are used as constants.

(12) Flowchart



[ELSE]

Increment FPR2_2(N)

(IF: Bit 0 of FPR2_4 (decimal-place flag) = 1)

[THEN]

Increment FPR2_1(F)

[A = number corresponding to a period]

(IF: Bit 0 of FPR2_4 (decimal-place flag) = 0)

[THEN]

Bit 0 of FPR2_4 <- 1

[ELSE]

Jump to ERROR

[OTHERS]

Break the loop

A <- number corresponding to the next character

GETC

<Exception processing>

(IF: FPR2_3 (place counter) = initial value)

[THEN]

Jump to ERROR

(IF: RP3•RP2(A) = 0)

[THEN]

FPR1_HP <- 0

Jump to T_TOL9

(IF: A (number corresponding to the next character) ≥ number corresponding to the minus sign)

[THEN]

Jump to ERROR

FPR2_1 <- FPR2_1(F) - FPR2_2(N)

<Normalizing mantissa A>

FPR2_2 <- A (number corresponding to the next character)

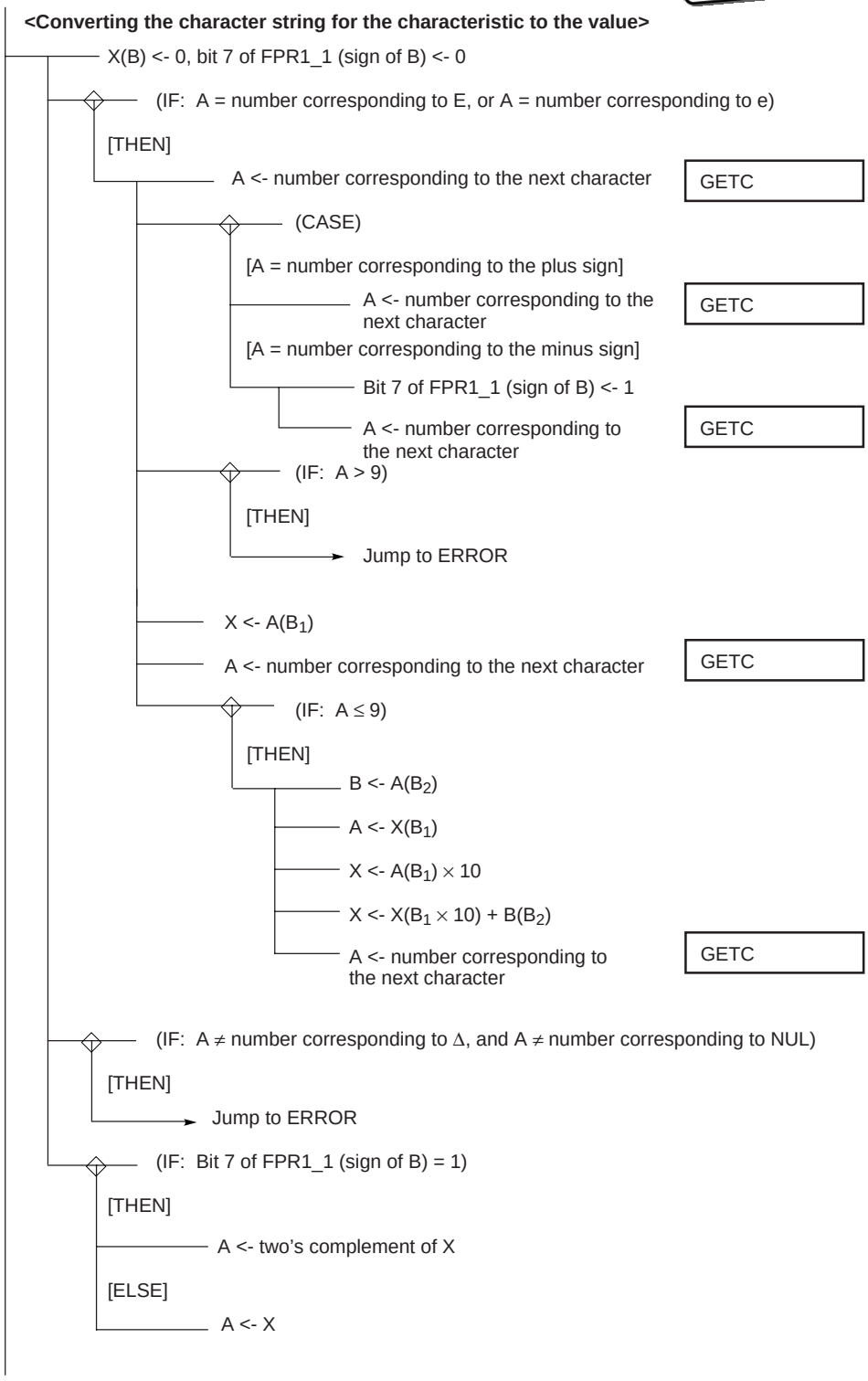
AX•RP2 <- RP3•RP2(A), FPR1_2 <- 31 + 7FH (bias for the characteristic)

FPR1•R5 <- normalizing bit 7 of FPR1_4 (sign), FPR1_2 (characteristic), and AX•RP2 (mantissa)

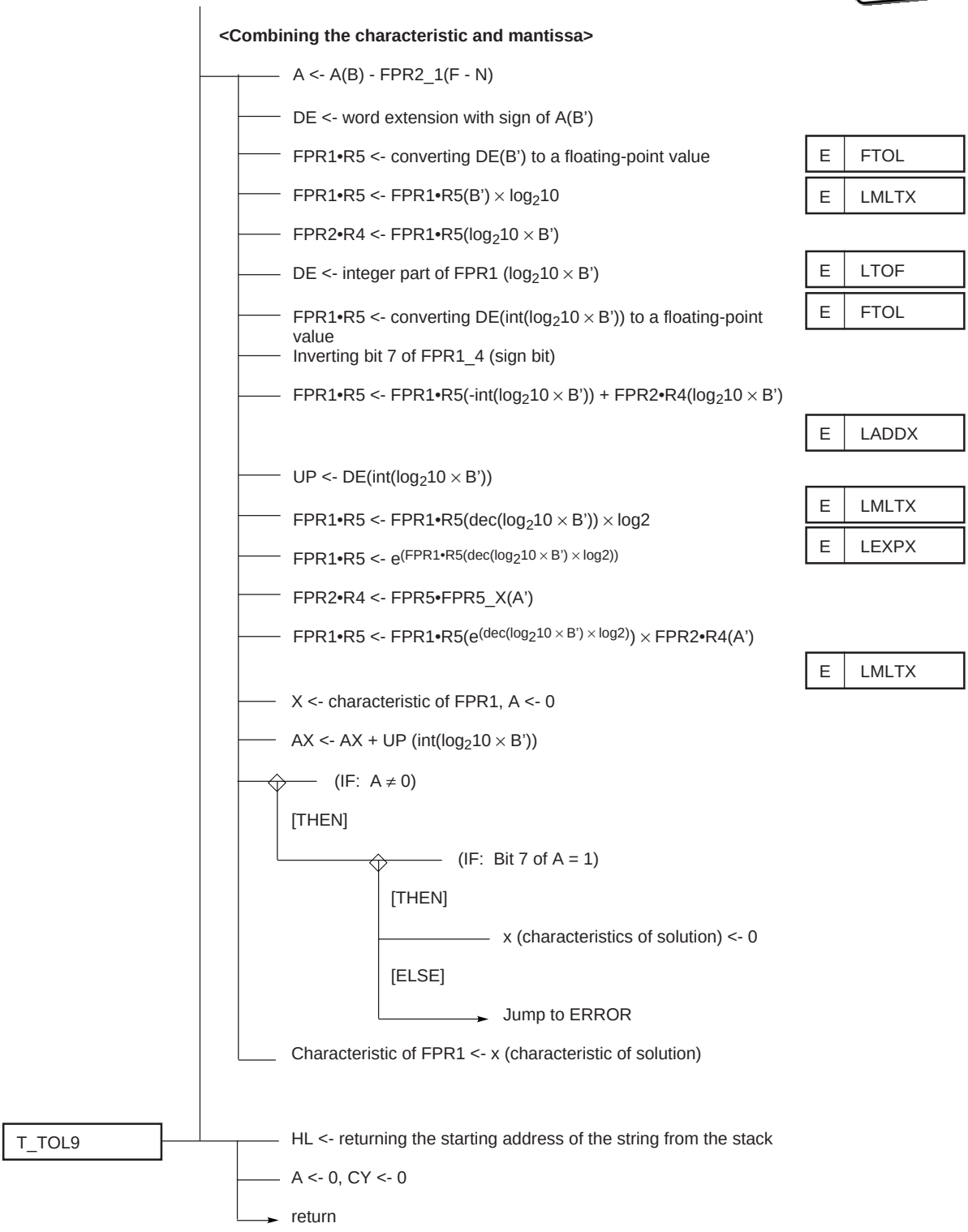
E	LNOR

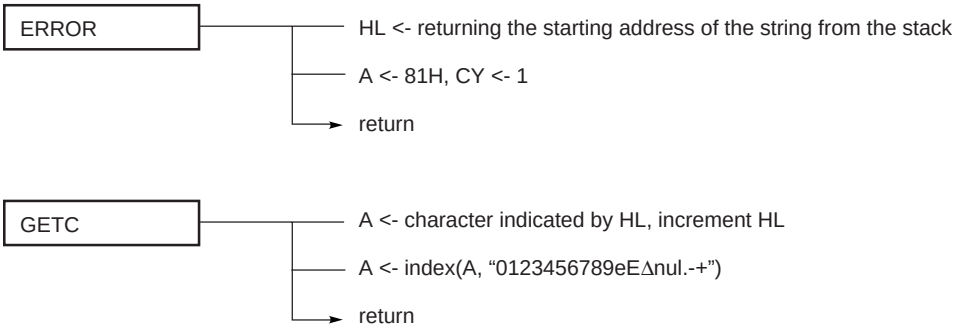
FPR5•FPR5_X <- FPR1•R5(A')

A <- FPR2_2 (number corresponding to the next character)



Phase-out/Discontinued





Remark Function index(character, string) represents the position (0 to 16) of the character in the string. If the character does not exist in the string, the value of 0FFH is returned.

6.2 FUNCTION FOR CONVERTING FLOATING-POINT CODES TO CHARACTER STRINGS (LTOA)

(1) Description

This function converts the value stored in FPR1 to a character string and stores it in the areas whose starting address is indicated by the HL register.

(2) Object module files to be linked

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

(3) Required stack area size in bytes

12 (including an area of two bytes for the return address from LTOA)

(4) Registers to be used

AX, BC, RP2, RP3, RP4, UP, DE, HL, UF (user flag)
(The value stored in the HL register is retained.)

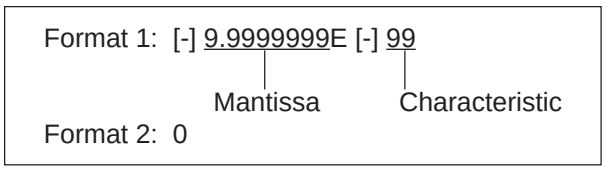
(5) Work areas to be used

FPR1, FPR2, FPR3, FPR4, FPR5, FPR3_X, FPR4_X, FPR5_X

(6) Execution time (internal system clock: 8 MHz, no wait)

Average : 5.88 ms
Maximum: 6.64 ms (9.9999992e+13)

(7) Format of output character strings



Rules

(a) When the converted value is 0, format 2 is used. In other cases, format 1 is used.
(b) The NUL code is attached to the end of the converted character string.
(c) In format 1, the number of characters in the mantissa is nine including the decimal point, and that for the characteristic is 2.
(d) In format 1, the minus code is attached to the mantissa and characteristic only when the converted value is negative.
(e) The number of characters including the NUL code is up to 15.

(8) Operation

(a) When floating-point value x is 0, this function outputs character string ONUL and terminates.

(b) Value x is converted to expression $a \times 10^b$ as follows:

$b = \text{floor}(\log_{10}(|x|))$, $b \neq 38 \rightarrow a = x \times 10^{-b}$
$b = 38 \rightarrow a = x/10^{38}$

where floor(x) represents the nearest integer from x in the negative direction.
Because value 10^{-38} underflows in the floating-point system of the 78K/III series, value 10^{-b} is not directly calculated when b is 38.

(c) When value a is negative, the minus code is output. Then the absolute value of a is set to a.

(d) Theoretically, although value a ranges from $1 \leq a < 10$, it may be less than 1, or 10 or more due to a calculation error. If it occurs, the following compensation is made.

When $a \geq 10$, a <- a/10, b <- b + 1
When $a < 1$, $a \leftarrow a \times 10$, b <- b - 1

(e) The location of the decimal point is fixed because value a ranges from $1 \leq a < 10$. The function calculates the following values and outputs character string A_1, A_2, ••••••, A_8.

$a_1 = a$
$A_1 = \text{int}(a_1)$, $a_2 = \text{dec}(a_1) \times 10$
$A_2 = \text{int}(a_2)$, $a_3 = \text{dec}(a_2) \times 10$

• • • • • • • • • • • • •

$A_8 = \text{int}(a_8)$

where, function int(a) represents the integer part of value a, and function dec(a) represents the decimal fraction part of a.

(f) Value E is output.

(g) When value b is negative, the minus code is output. Then the absolute value of b is set to b.

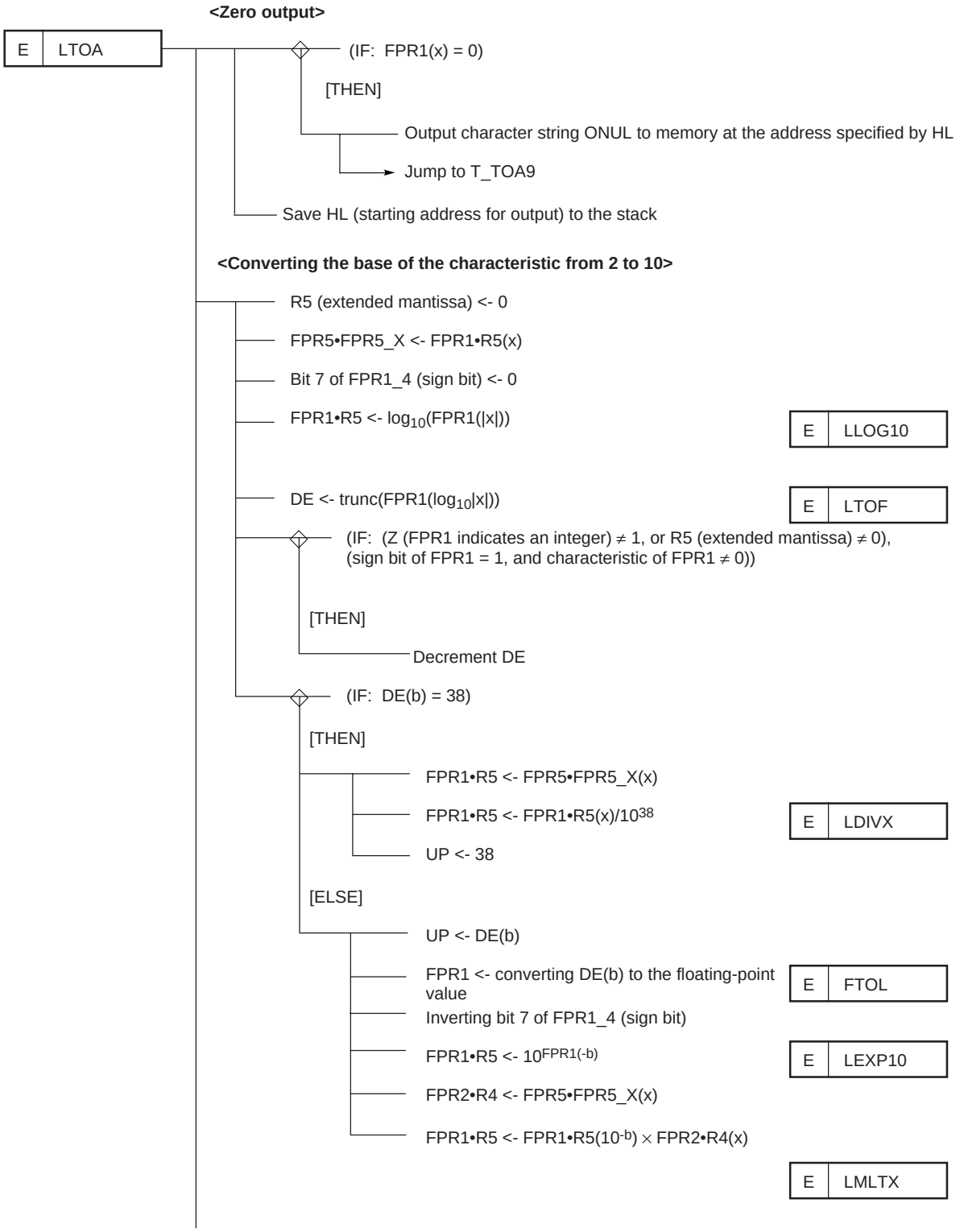
(h) Character string B_1B_2 ($B_1 = b/10$, $B_2 = b - B_1 \times 10$) which corresponds to value b is output.

(i) The NUL code is output.

(9) Floating-point constant

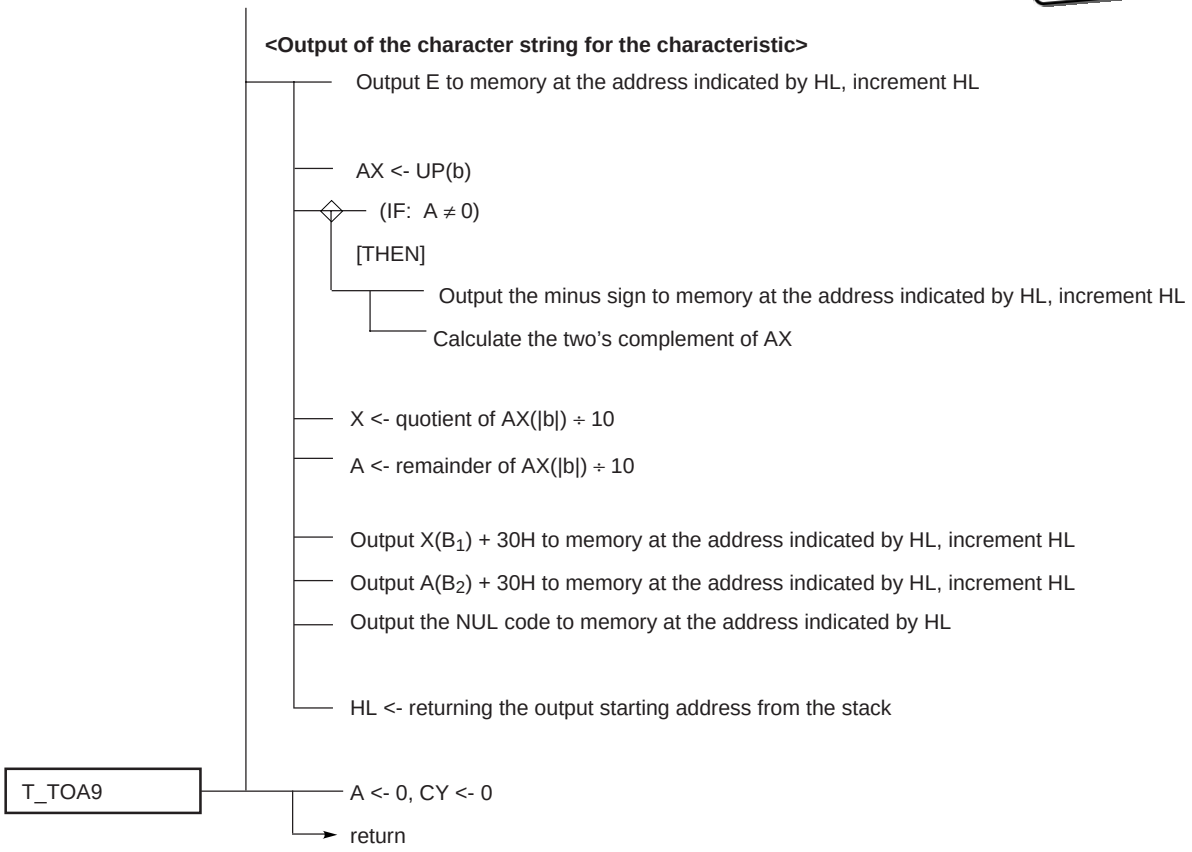
The values of 10^{38} with the extended mantissa part, 10 and 1/10 are used as constants.

(10) Flowchart



<Output of the character string for the mantissa>

- (IF: Bit 7 of FPR1_4 (sign of a) = 1)
  - [THEN]
    - Output the minus sign to memory at the address indicated by HL, increment HL
    - Bit 7 of FPR1_4 <- 0
- (IF: FPR1•R5(|a|) ≥ 10)
  - [THEN]
    - FPR1•R5 <- FPR1•R5(|a|) + 1/10 — E | LMLTX
    - Increment UP(b)
- (IF: FPR1•R5 (|a|) <1)
  - [THEN]
    - FPR1•R5 <- FPR1•R5(|a|) × 10 — E | LMLTX
    - Decrement UP
- DE <- int(FPR1(a_1)) — E | LTOF
- Output E(A_1) + 30H to memory at the address indicated by HL, increment HL
- Output a period to memory at the address indicated by HL, increment HL
- (FOR: i = 1 TO 7)
  - FPR2•R4 <- FPR1•R5(a_i)
  - FPR1•R5 <- converting DE(int(a_i)) to a floating-point value — E | FTOL
  - Bit 7 of FPR1_4 (sign bit) <- 1
  - FPR1•R5 <- FPR1•R5(-int(a_i)) + FPR2•R4(a_i) — E | LADDX
  - FPR1•R5 <- FPR1•R5(dec(a_i)) × 10 — E | LMLTX
  - DE <- int(FPR1(a_{i+1})) — E | LTOF
  - Output E(A_{i+1}) + 30H to memory at the address indicated by HL, increment HL



Remark Function trunc(x) rounds down the decimal fraction part of x toward 0. When $x \geq 0$, $\text{trunc}(x) \leq x < \text{trunc}(x)$. When $x < 0$, $\text{trunc}(x) - 1 < x \leq \text{trunc}(x)$.

6.3 FUNCTION FOR CONVERTING TWO-BYTE INTEGERS TO FLOATING-POINT CODES (FTOL)

(1) Description

This function converts the value stored in the DE register as a signed two-byte integer to a floating-point code and returns the code to FPR1.

(2) Object module files to be linked

DFLT, FTOL

(3) Required stack area size in bytes

2 (an area of two bytes for the return address from FTOL)

(4) Registers to be used

AX, R5, DE (The value stored in the DE register is retained.)

(5) Work areas to be used

FPR1

(6) Execution time (internal system clock: 8 MHz, no wait)

Average : 33.6 µs

Maximum: 56.0 µs (-1)

(7) Format of two-byte integers

The least significant bit represents a sign. Negative values are expressed in two's complement. Two-byte integer F ranges from -8000H to +7FFFH.

(8) Operation

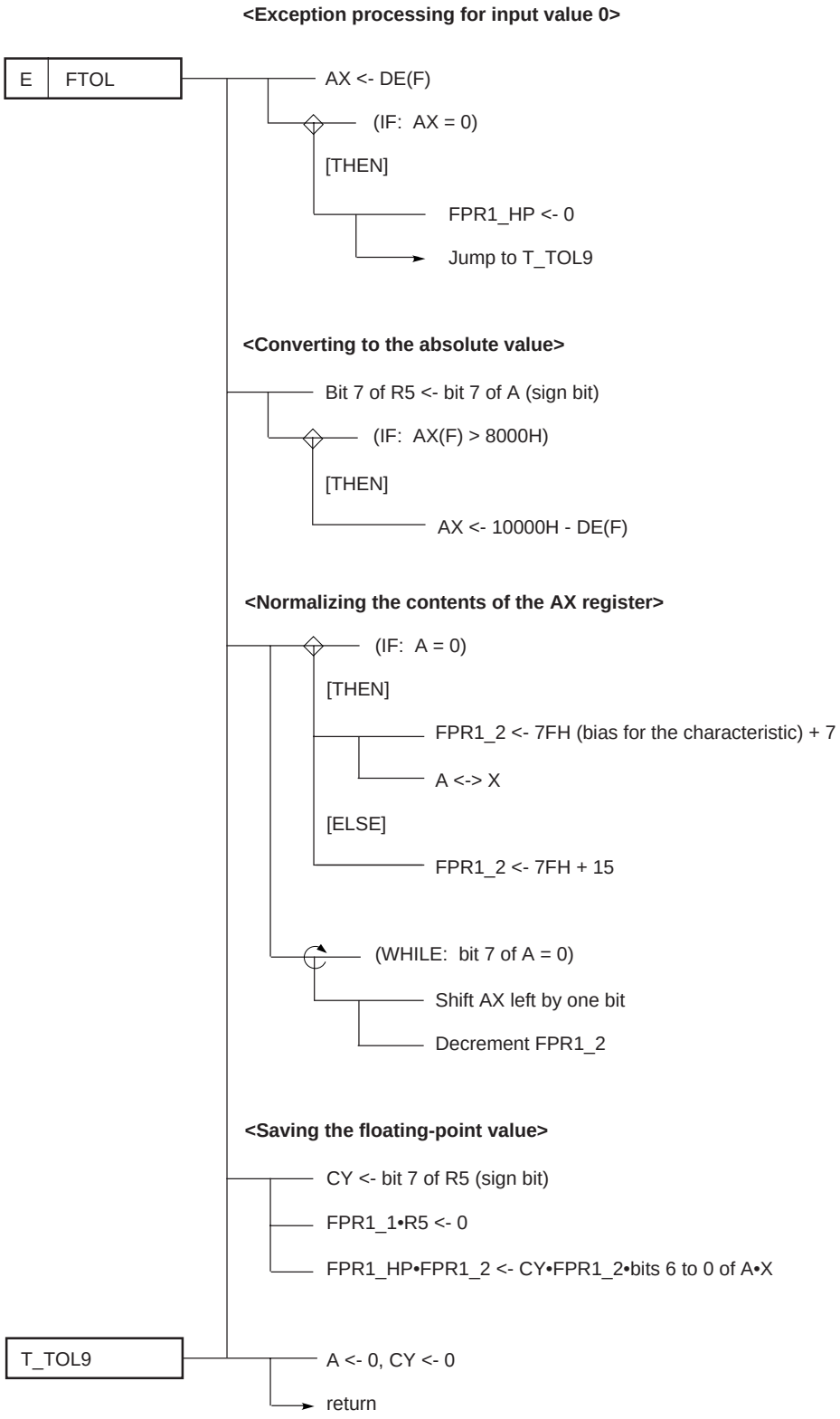
(a) If two-byte integer F is 0, this function returns value 0.

(b) If integer F ranges from -7FFFH ≤ F < 0, its two's complement is calculated.

(c) The mantissa and characteristic are assumed to be the following initial conditions. They are normalized and converted to floating-point codes.

(d) The mantissa, characteristic, and sign code of value F obtained in step (c) are stored in FPR1.

(9) Flowchart



6.4 FUNCTION FOR CONVERTING FLOATING-POINT CODES TO TWO-BYTE INTEGERS (LTOF)

(1) Description

This function converts the value stored in the FPR1 register to a signed two-byte integer and returns the integer to the DE register. When the value stored in FPR1 does not have a decimal fraction part, the function returns 1 to the Z flag. When the decimal fraction part is rounded down in the conversion, the function returns 0 to the Z flag.

(2) Object module files to be linked

DFLT, LTOF

(3) Required stack area size in bytes

2 (an area of two bytes for the return address from LTOF)

(4) Registers to be used

AX, C, RP3, DE

(5) Work areas to be used

FPR1 (The value stored in FPR1 is retained.)

(6) Execution time (internal system clock: 8 MHz, no wait)

Average : 45.1 μs

Maximum: 61.0 μs (-1.8251824)

(7) Operation

(a) When the characteristic of the code is 0, this function returns value 0 and sets 1 in the Z flag. When the characteristic of the code is less than 7FH, the function returns value 0 and sets 0 in the Z flag. If the characteristic of the code is 8FH or more, the function returns an error code.

(b) The integer part of the code is extracted in the unsigned integer format. The most significant bit is set. When the characteristic is less than 87H, the first mantissa is shifted right by (86H - the characteristic) bits to make it an integer. When the characteristic is 87H or more, the first and second mantissa is shifted right by (8EH - the characteristic) bits to make it an integer.

(c) The unsigned integer is converted to the signed integer.

Depending on the integer obtained in step (b) and the sign of the FPR1, the unsigned integer is converted as follows:

When the integer is more than 8000H and the sign is negative, an error occurs.

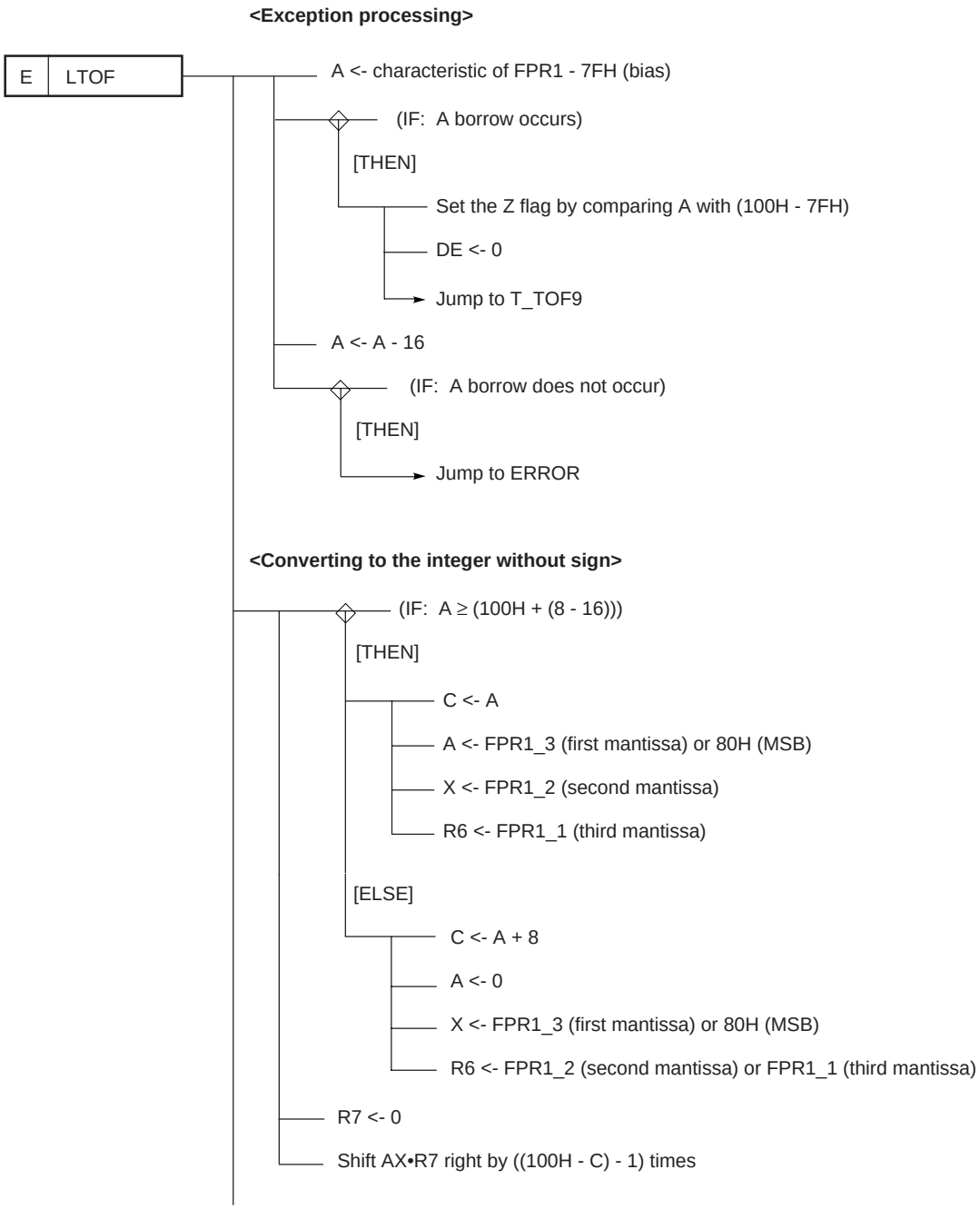
When the integer is less than or equal to 8000H and the sign is negative, the two's complement of the integer is calculated.

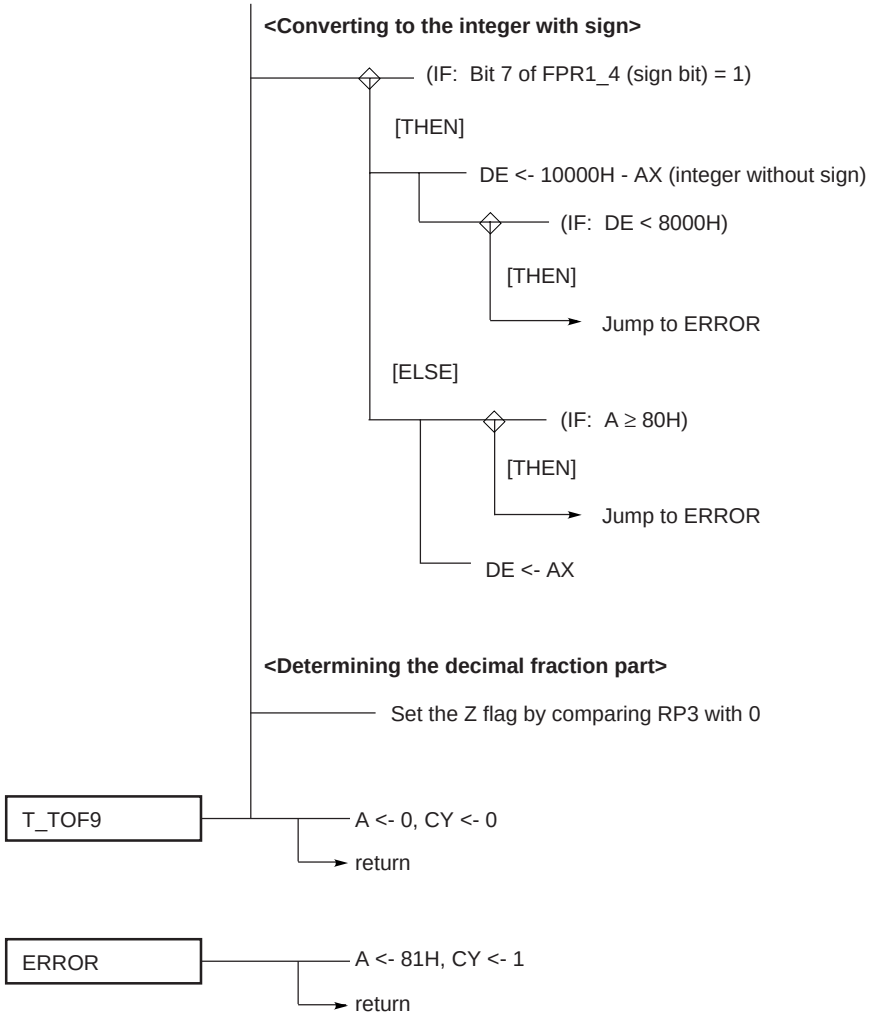
When the integer is 8000H or more, and the sign is positive, an error occurs.

When the integer is less than 8000H, and the sign is positive, the integer is not converted.

(d) • Value 1 is returned to the Z flag in step (b) in the following cases:
  • When the characteristic is less than 87H, all bits of the second and third mantissa are 0, and a carry is not generated in the right-shift process for the first mantissa,
  • When the characteristic is 87H or more, all the bits of the third mantissa are 0, and a carry is not generated in the right-shift process for the first and second mantissa
• Value 0 is returned to the Z flag in cases other than those described above.

(8) Flowchart





[MEMO]

CHAPTER 7 EXECUTION RESULTS

This chapter describes the results of calculation for each function and the execution time measured under the following conditions:

CPU: μPD78320
Clock: An external clock of 16 MHz
(internal system clock, f_{CLK} = 8 MHz)
Code area: External RAM
Work area: Internal RAM
Stack area: 0FE00H to 0FE1FH
Programmable wait: No wait

The execution time is measured by timer 0 in the μPD78320. The measured time includes the time required for reading the timer, which is equivalent to about 20 clocks.

The results of calculation include rounding errors generated when values are converted between the internal floating-point notation and decimal notation.

The results of calculation performed by a PC-9801 personal computer (MSC version 5.1) are also shown for reference.

Programs described in this document were created using the μPD78320 instruction set.

7.1 FLOATING-POINT ADDITION (LADD)

0 + 0 = 0 (18 µs)
1 + 0 = 1 (17 µs)
0 + 1 =1 (22 µs)
1234 + 98765 = 99999 (74 µs)
4.5567831e + 09 + 2.1447790e + 06 = 4.5589279e + 09 (97 µs)
1.3e + 20 + 4e - 30 = 1.3e + 20 (18 µs)
223 + 0.111111 = 223.11111 (97 µs)
2.1474836e + 09 + 0.5 = 2.1474836e + 09 (18 µs)
3.4028237e + 38 + 3.4028237e + 38 = Abnormal termination (40 µs)
0.5 + (-0.5) = 0 (42 µs)
1.7632415e - 38 + (-1.1754944e - 38) = 0 (50 µs)
1.0737418e + 09 + 0.5 = 1.0737418e + 09 (190 µs)
0.5 + (-0.50000006) = -5.9604645e - 08 (221 µs)

7.2 FLOATING-POINT SUBTRACTION (LSUB)

0 - 0 = 0 (22 µs)
1 - 0 = 1 (21 µs)
0 - 1= -1 (25 µs)
1.7014110e + 38 - 1e + 32 = 1.70141e + 38 (148 µs)
2.3352e - 05 - 9.99999e - 21 = 2.3352e - 05 (20 µs)
3.7634668e - 24 - 1.2 = -1.2 (25 µs)
9.8999999e - 38 - 2.2000001e - 38 = 7.6999998e - 38 (77 µs)
12356565 - 45876 = 12310689 (93 µs)
1.2345679e + 08 - 1.2345679e + 08 = 0 (45 µs)
125654 - 988656 = -863002 (68 µs)
0.5 - 0.50000006 = -5.9604645e - 08 (224 µs)

7.3 FLOATING-POINT MULTIPLICATION (LMLT)

0 × 0 = 0 (18 μs)
1 × 0 = 0 (19 μs)
0 × 1 = 0 (18 μs)
1.701411e + 38 × 0.1 = 1.701411e + 37 (59 μs)
6.5436653e + 08 × 12345 = 8.0781548e + 12 (61 μs)
1.2345679e + 08 × 1.2345679e + 08 = 1.5241579e + 16 (58 μs)
1e + 30 × 0 = 0 (19 μs)
1e + 30 × 1e - 10 = 1e + 20 (58 μs)
1.234e + 20 × 2.34e + 02 = 2.8875600e + 22 (59 μs)
5.1042355e + 38 × 1.5 = Abnormal termination (50 μs)
2.5521178e + 38 × 1.5 = 3.8281766e + 38 (59 μs)
1 × 1.1754944e - 38 = 1.1754944e - 38 (60 μs)
2 × 2 = 4 (61 μs)

7.4 FLOATING-POINT DIVISION (LDIV)

0 ÷ 1 = 0 (20 μs)
1.701411e + 38 ÷ 2 = 8.5070551e + 37 (79 μs)
1 ÷ 0 = Abnormal termination (15 μs)
9.9999997e + 37 ÷ 1e + 08 = 9.9999997e + 29 (75 μs)
12 ÷ 21 = 0.57142857(79 μs)
1.1754944e - 38 ÷ 2 = 0 (24 μs)
3.4028237e + 38 ÷ 0.25 = Abnormal termination (21 μs)
(-2.3509887e - 38) ÷ 2 = -1.1754944e - 38 (79 μs)
1 ÷ 8.5070592e + 37 = 1.1754944e - 38 (79 μs)
31377.404 ÷ 3.2481321e + 37 = 9.6601381e - 34 (84 μs)

7.5 SINE (LSIN)

	μPD78320		PC-9801
sin(3.1415927)	-8.7544322e - 08	(793 μs)	-8.7422780e - 08
sin(1.5707964)	0.99999995	(2.23 ms)	1
sin(100)	-0.50636563	(2.01 ms)	-0.50636564
sin(9999999)	0.99015793	(2.41 ms)	0.99066465
sin(-100)	0.50636563	(2.01 ms)	0.50636564
sin(0)	0	(199 μs)	0
sin(0.2)	0.19866933	(1.30 ms)	0.19866933
sin(-32.967228)	-0.99980993	(1.92 ms)	-0.99980998
sin(33.333332)	0.94053001	(2.04 ms)	0.94053001
sin(6.2831593)	-2.6050024e - 05	(853 μs)	-2.6051198e - 05
sin(9.424778)	-2.6077032e - 08	(819 μs)	-2.3849761e - 08
sin(-6.8056469e + 38)	-0.5389122	(4.68 ms)	Rounding error

7.6 COSINE (LCOS)

	μPD78320		PC-9801
cos(0)	0.99999994	(1.62 ms)	1
cos(3.1415927)	-0.99999995	(2.23 ms)	-1
cos(1.5707964)	-4.3772161e - 08	(802 μs)	-4.3711390e - 08
cos(-10000)	-0.95215494	(2.05 ms)	-0.95215537
cos(8.3775806)	-0.5000002	(1.97 ms)	-0.5000002
cos(-9424.7783)	0.99999988	(2.16 ms)	0.99999994
cos(162.31561)	0.49999341	(1.94 ms)	0.49999338
cos(6.8056469e + 38)	0.84236194	(4.83 ms)	Rounding error

7.7 TANGENT (LTAN)

	μPD78320		PC-9801
tan(0)	0	(1.88 ms)	0
tan(3.1415927)	8.7544327e - 08	(2.67 ms)	8.7422780e - 08
tan(1.5707964)	-22845569	(2.68 ms)	-22877332
tan(-1.0471976)	-1.7320509	(3.44 ms)	-1.7320509
tan(2.0999999)	-1.7098469	(3.80 ms)	-1.7098469
tan(1000)	1.4703252	(3.81 ms)	1.4703242
tan(500)	0.52924408	(3.83 ms)	0.52924386
tan(157.07964)	2.9802324e - 06	(2.63 ms)	2.9406275e - 06
tan(6.8056469e + 38)	0.63976323	(6.53 ms)	Rounding error

7.8 NATURAL LOGARITHM (LLOG)

	μPD78320		PC-9801
log(2.7182817)	0.99999997	(1.88 ms)	0.99999997
log(9.9999996e + 35)	82.893063	(1.67 ms)	82.893063
log(1)	0	(465 μs)	0
log(0)	Abnormal termination	(10.0 μs)	Invalid argument
log(-0.1)	Abnormal termination	(10.0 μs)	Invalid argument
log(12345.679)	9.4210614	(2.02 ms)	9.4210614
log(59874.141)	11	(1.54 ms)	11
log(20.085537)	3	(1.79 ms)	3
log(4.5399931e - 05)	-10	(1.97 ms)	-10
log(6.8056469e + 38)	89.415986	(1.07 ms)	89.415986
log(1.1754944e - 38)	-87.336545	(524 μs)	-87.336545

7.9 COMMON LOGARITHM (LLOG10)

	μPD78320		PC-9801
$\log_{10}(0)$	Abnormal termination	(14.0 μs)	Invalid argument
$\log_{10}(-1)$	Abnormal termination	(14.0 μs)	Invalid argument
$\log_{10}(1)$	0	(496 μs)	0
$\log_{10}(10)$	1	(1.86 ms)	1
$\log_{10}(1.2345679e + 08)$	8.091515	(1.41 ms)	8.091515
$\log_{10}(9.8765434e + 08)$	8.994605	(1.41 ms)	8.994605
$\log_{10}(0.44400001)$	-0.35261702	(1.70 ms)	-0.35261702
$\log_{10}(100000)$	5	(1.91 ms)	5
$\log_{10}(6.8056469e + 38)$	38.832869	(1.15 ms)	38.832869
$\log_{10}(1.1754944e - 38)$	-37.929779	(596 μs)	-37.929779
$\log_{10}(3.6486432e - 26)$	-25.437869	(2.08 ms)	-25.437869

7.10 EXPONENTIAL FUNCTION (BASE e) (LEXP)

	μPD78320		PC-9801
$e^{(0)}$	1	(293 μs)	1
$e^{(1)}$	2.7182818	(2.23 ms)	2.7182818
$e^{(-1)}$	0.36787944	(2.34 ms)	0.36787944
$e^{(0.98765433)}$	2.6849291	(2.23 ms)	2.6849291
$e^{(20)}$	4.8516519e + 08	(2.32 ms)	4.851652e + 08
$e^{(11)}$	59874.142	(2.33 ms)	59874.142
$e^{(89.415993)}$	Abnormal termination	(122 μs)	Overflow
$e^{(89.415985)}$	6.8056389 + 38	(1.28 ms)	6.8056393 + 38
$e^{(-87.336548)}$	0	(1.26 ms)	1.1754907e - 38
$e^{(-87.33654)}$	1.1754997e - 38	(1.16 ms)	1.1754997e - 38
$e^{(-0.82129019)}$	0.43986378	(2.48 ms)	0.43986378

7.11 EXPONENTIAL FUNCTION (BASE 10) (LEXP10)

	μPD78320		PC-9801
$10^{(38.832863)}$	6.805543e + 38	(1.29 ms)	6.8055441e + 38
$10^{(-37.929775)}$	1.1755059e - 38	(1.22 ms)	1.1755058e - 38
$10^{(89.415993)}$	Abnormal termination	(226 μs)	Overflow
$10^{(-87.336548)}$	0	(228 μs)	4.60736e - 88
$10^{(0)}$	1	(326 μs)	1
$10^{(0.56666666)}$	3.686945	(2.21 ms)	3.686945
$10^{(0.96666664)}$	9.2611867	(2.41 ms)	9.2611867
$10^{(1.7333332)}$	54.11694	(2.49 ms)	54.11694
$10^{(0.34659675)}$	2.2212465	(2.29 ms)	2.2212465
$10^{(0.33924761)}$	2.1839748	(2.32 ms)	2.1839748
$10^{(0.35975304)}$	0.43676412	(2.55 ms)	0.43676412

7.12 POWER (LPOW)

	μPD78320		PC-9801
$(0)^{(0)}$	Abnormal termination	(15 μs)	Overflow
$(0)^{(1)}$	0	(16 μs)	0
$(1)^{(0)}$	1	(813 μs)	1
$(1)^{(1)}$	1	(813 μs)	1
$(1)^{(-1)}$	1	(813 μs)	1
$(2)^{(-2)}$	0.25	(1.79 ms)	0.25
$(50)^{(2)}$	2500	(4.38 ms)	2500
$(2050)^{(2)}$	4202499.9	(2.34 ms)	4202500
$(-1)^{(2.5669999)}$	Abnormal termination	(34 μs)	Invalid argument
$(0)^{(-9.8765001)}$	Abnormal termination	(15 μs)	Overflow
$(1.3038405e + 19)^{(2)}$	1.6999997e + 38	(3.55 ms)	1.7e + 38
$(9.876543)^{(1.2345679)}$	16.900803	(3.97 ms)	16.900803
$(9)^{(1.2345001)}$	15.066502	(3.73 ms)	15.066502
$(2.1900001)^{(-9.1199999)}$	7.8550620e - 04	(3.95 ms)	7.8550618e - 04
$(4)^{(6.8056469e + 38)}$	Abnormal termination	(635 μs)	Overflow
$(3)^{(3)}$	27	(4.46 ms)	27

7.13 SQUARE ROOT (LSQRT)

	μPD78320		PC-9801
√(0)	0	(11.0 μs)	0
√(1)	1	(612 μs)	1
√(2)	1.4142136	(602 μs)	1.4142136
√(121)	11	(611 μs)	11
√(2500)	50	(606 μs)	50
√(1e - 06)	1e - 03	(625 μs)	1e - 03
√(-9.9999998e - 03)	Abnormal termination	(10 μs)	Invalid argument
√(30.863079)	5.5554549	(625 μs)	5.5554549
√(11.111111)	3.3333333	(602 μs)	3.3333333

7.14 ARCSINE (LASIN)

	μPD78320		PC-9801
arcsin(0)	0	(991 μs)	0
arcsin(1)	1.5707963	(35.0 μs)	1.5707963
arcsin(-1)	-1.5707963	(35.0 μs)	-1.5707963
arcsin(-0.5)	-0.52359878	(2.40 ms)	-0.52359878
arcsin(3.1415927)	Abnormal termination	(19.0 μs)	Invalid argument
arcsin(0.78539819)	0.90333915	(2.53 ms)	0.90333915
arcsin(-0.86602539)	-1.0471975	(2.57 ms)	-1.0471975
arcsin(0.98437494)	1.3937883	(2.74 ms)	1.3937883

7.15 ARCCOSINE (LACOS)

	μPD78320		PC-9801
arccos(0)	1.5707963	(1.03 ms)	1.5707963
arccos(1)	0	(92 μs)	0
arccos(-1)	3.1415927	(101 μs)	3.1415927
arccos(0.52359879)	1.0197267	(2.14 ms)	1.0197267
arccos(-0.5)	2.0943951	(2.47 ms)	2.0943951
arccos(-0.86602539)	2.6179938	(2.63 ms)	2.6179938
arccos(0.1)	1.4706289	(2.14 ms)	1.4706289
arccos(-0.1)	1.6709637	(2.14 ms)	1.6709637
arccos(0.98437494)	0.17700803	(2.83 ms)	0.17700802

7.16 ARCTANGENT (LATAN)

	μPD78320		PC-9801
arctan(0)	0	(273 μs)	0
arctan(1)	0.78539816	(1.83 ms)	0.78539816
arctan(-1)	-0.78539816	(1.83 ms)	-0.78539816
arctan(3.1415927)	1.2626273	(1.85 ms)	1.2626273
arctan(1.5707964)	1.0038848	(1.39 ms)	1.0038848
arctan(1.7014110e + 38)	1.5707963	(397 μs)	1.5707963
arctan(10000000)	1.5707962	(709 μs)	1.5707962
arctan(0.001)	9.9999971e - 04	(798 μs)	9.9999971e - 04
arctan(10)	1.4711277	(1.36 ms)	1.4711277
arctan(-10)	-1.4711277	(1.36 ms)	-1.4711277

7.17 HYPERBOLIC SINE (LHSIN)

	μPD78320		PC-9801
sinh(89.415993)	Abnormal termination	(130 μs)	3.4028456e + 38
sinh(-89.415993)	Abnormal termination	(130 μs)	-3.4028456e + 38
sinh(89.415985)	3.4028194e + 38	(1.38 ms)	3.4028196e + 38
sinh(-89.415985)	-3.4028194e + 38	(1.38 ms)	-3.4028196e + 38
sinh(0)	0	(186 μs)	0
sinh(0.49999997)	0.52109527	(991 μs)	0.52109527
sinh(0.125)	0.12532578	(1.11 ms)	0.12532578
sinh(1.0842022e - 19)	1.0842022e - 19	(233 μs)	1.0842022e - 19
sinh(0.76666665)	0.84400998	(2.26 ms)	0.84400998
sinh(1.0666666)	1.2807619	(2.53 ms)	1.2807619
sinh(1.8666666)	3.1560329	(2.59 ms)	3.1560329
sinh(3.351413)	14.254001	(2.70 ms)	14.254001

7.18 HYPERBOLIC COSINE (LHCOS)

	μPD78320		PC-9801
cosh(-89.415993)	Abnormal termination	(127 μs)	3.4028456e + 38
cosh(-89.415985)	3.4028194e + 38	(1.37 μs)	3.4028196e + 38
cosh(0)	1	(471 μs)	1
cosh(1.0842022e - 19)	1	(605 μs)	1
cosh(0.76666665)	1.308569	(2.23 ms)	1.308569
cosh(1.0666666)	1.6249157	(2.51 ms)	1.6249157
cosh(1.8666666)	3.3106712	(2.57 ms)	3.3106712
cosh(4.0319099)	28.193105	(2.68 ms)	28.193105

7.19 HYPERBOLIC TANGENT (LHTAN)

	μPD78320		PC-9801
tanh(89.415993)	1.0000001	(156 μs)	1
tanh(-89.415993)	-1.0000001	(156 μs)	-1
tanh(0)	0	(717 μs)	0
tanh(0.4998779)	0.46202112	(3.56 ms)	0.46202113
tanh(0.125)	0.12435300	(3.64 ms)	0.124353
tanh(1.0842022e - 19)	1.0842022e - 19	(959 μs)	1.0842022e - 19
tanh(0.76666665)	0.64498699	(4.61 ms)	0.64498699
tanh(1.0666666)	0.78820205	(5.16 ms)	0.78820205
tanh(1.8666666)	0.95329096	(5.27 ms)	0.95329096
tanh(9.4484739)	0.99999999	(5.44 ms)	0.99999999

7.20 ABSOLUTE FUNCTION (LABS)

| 0 | = 0 (6 μs)
| -2.1290744e - 19 | = 2.1290744e - 19 (6 μs)
| 1.6415355e + 27 | = 1.6415355e + 27 (6 μs)

7.21 INVERSE FUNCTION (LRCPN)

1/0 = Abnormal termination (35 μs)
1/ (-1.1754944e - 38) = -8.5070592e + 37 (99 μs)
1/0.99999994 = 1.0000001 (94 μs)
1/ (-1.0000001) = -0.99999988 (98 μs)
1/ (8.5070602e + 37) = 0 (99 μs)
1/ (8.5070592e + 37) = 1.1754944e - 38 (99 μs)
1/ (2.8545976e - 18) = 3.5031207e + 17 (99 μs)
1/ (-2.9092885e + 21) = -3.4372665e - 22 (95 μs)
1/ (-8.5070592e + 37) = -1.1754944e - 38 (100 μs)

7.22 FUNCTION FOR CONVERTING POLAR COORDINATES TO CARTESIAN COORDINATES (POTORA)

(r, θ)	(x, y)	
(1, 1.5707964)	(-4.3772161e - 08, 0.99999995) (-4.3711390e - 08, 1)	(2.77 ms)
(1, 0.52359879)	(0.8660254, 0.50000001) (0.8660254, 0.50000001)	(3.53 ms)
(7, 2.6179938)	(-6.0621777, 3.5000003) (-6.0621777, 3.5000003)	(3.88 ms)
(99, -2.0943952)	(-49.500005, -85.736512) (-49.500005, -85.736512)	(3.89 ms)
(8.8888798, 2.3561945)	(-6.2853872, 6.2853871) (-6.2853872, 6.2853871)	(4.04 ms)
(4.4443998, 3.1415927)	(-4.4443996, -3.8908197e - 07) (-4.4443998, -3.8854179e - 07)	(2.76 ms)
(0.5, 6.8056469e + 38)	(0.42118097, 0.2694561) Rouding error	(6.62 ms)

Remark The coordinates converted by the μPD78320 are shown on the upper lines and those converted by the PC-9801 personal computer are shown on the lower lines.

7.23 FUNCTION FOR CONVERTING CARTESIAN COORDINAES TO POLAR COORDINATES (RATOPO)

(x, y)	(r, θ)	
(0, 0)	(0, 0) (0, 0)	(15.0 μs)
(1, 1)	(1.4142136, 0.78539816) (1.4142136, 0.78539816)	(2.82 ms)
(0, 1)	(1, 1.5707963) (1, 1.5707963)	(871 μs)
(1, -1)	(1.4142136, -0.78539816) (1.4142136, -0.78539816)	(2.82 ms)
(-1, 1)	(1.4142136, 2.3561945) (1.4142136, 2.3561945)	(2.89 ms)
(-1, -1)	(1.4142136, -2.3561945) (1.4142136, -2.3561945)	(2.89 ms)
(0, -1)	(1, -1.5707963) (1, -1.5707963)	(871 μs)
(1, 0)	(1, 0) (1, 0)	(1.14 ms)
(-1, 0)	(1, 3.1415927) (1, 3.1415927)	(1.18 ms)
(11111, 11111)	(15713.327, 0.78539816) (15713.327, 0.78539816)	(2.82 ms)
(1, -9)	(9.0553851, -1.4601391) (9.0553851, -1.4601391)	(2.38 ms)
(-3.8408036, 12.227133)	(12.816183, -1.8751576) (12.816183, -1.8751576)	(2.92 ms)

Remark The coordinates converted by the μPD78320 are shown on the upper lines and those converted by the PC-9801 personal computer are shown on the lower lines.

7.24 FUNCTION FOR CONVERTING CHARACTER STRINGS TO FLOATING-POINT CODE (ATOL)

"1234567.890123456789012345678"	= Abnormal termination	(782 μs)
"0Q"	= 0	(104 μs)
"E12"	= Abnormal termination	(69 μs)
"1e"	= Abnormal termination	(396 μs)
"1E + 123"	= Abnormal termination	(409 μs)
"1.17549427E - 38"	= 0	(3.31 ms)
"1.17549428E - 38"	= 1.1754944e - 38	(3.31 ms)
"6.8056476E + 38"	= Abnormal termination	(2.72 ms)
"6.8056475E + 38"	= 6.8056473e + 38	(2.72 ms)
"655361"	= 655361	(877 μs)
"1e - 20"	= 1e - 20	(3.18 ms)
"12345678901234567890123456 7E - 32"	= 1.2345679e - 06	(3.54 ms)
"1.00000000000000000000000000E - 9"	= 1e - 09	(3.69 ms)
"+1.2030646E + 22"	= 1.2030646e + 22	(3.32 ms)
"-4.6231684E - 18"	= -4.6231685e - 18	(3.04 ms)
"0.00000000000000000117549428E - 20"	= 1.1754944e - 38	(3.95 ms)

7.25 FUNCTION FOR CONVERTING FLOATING-POINT CODE TO CHARACTER STRINGS (LTOA)

0	= "0"	(12.0 μs)
1.0000002e - 37	= "1.0000000E - 37"	(4.86 ms)
9.9999999e - 38	= "9.9999987E - 38"	(5.55 ms)
1.0000001e - 05	= "1.0000000E - 05 "	(5.73 ms)
1	= "1.0000000E00"	(1.64 ms)
100000	= "1.0000000E05"	(5.66 ms)
9.9999993e + 19	= "9.9999997E19"	(6.11 ms)
9.9999987e + 37	= "9.9999998E37"	(5.71 ms)
1.0000001e + 38	= "1.0000001E38"	(5.21 ms)
6.8056469e + 38	= "6.8056469E38"	(3.05 ms)
-6.8056469e + 38	= "-6.8056469E38"	(3.05 ms)
9.9999992e + 13	= "9.9999994E13"	(6.64 ms)

7.26 FUNCTION FOR CONVERTING TWO-BYTE INTEGERS TO FLOATING-POINT CODE (FTOL)

0	(10.0 μs)
-1	(56.0 μs)
-32768	(24.0 μs)
1	(55.0 μs)
511	(53.0 μs)
-511	(55.0 μs)
255	(25.0 μs)
32767	(29.0 μs)

Phase-out/Discontinued

7.27 FUNCTION FOR CONVERTING FLOATING-POINT CODE TO TWO-BYTE INTEGERS (LTOF)

-0.99999994	=	0 (Flag Z=0)	(12.0 µs)
0	=	0 (Flag Z=1)	(12.0 µs)
65536	=	Abnormal termination	(12.0 µs)
-32769	=	Abnormal termination	(26.0 µs)
-32768	=	-32768 (Flag Z=1)	(29.0 µs)
32767.5	=	32767 (Flag Z=0)	(32.0 µs)
1	=	1 (Flag Z=1)	(60.0 µs)
1.5	=	1 (Flag Z=0)	(60.0 µs)
-1.8251824	=	-1 (Flag Z=0)	(61.0 µs)

CHAPTER 8 PROGRAM LIST

(1) EQU.INC

```
$      NOLIST
;*************************************************
;*
;*  78K3 COMMON NAME DEFINE
;*
;*
;*
;*************************************************
SHORT  EQU    4       ;size of real type
INTEGR EQU    2       ;size of integer type
BYTE   EQU    8       ;bit figures of byte
ZEROEX EQU    7FH     ;exponent bias for 0
R_OK   EQU    0       ;normal return code
R_ERR  EQU    81H     ;abnormal return code
$      LIST
```

(2) REF1.INC

```
$      NOLIST
;*************************************************
;*
;*  78K3 FLOATING POINT REGISTER REFFERENCE DEFINE
;*
;*
;*
;*************************************************
       EXTRN  FPR1
       EXTRN  FPR1_LP,FPR1_HP
       EXTRN  FPR1_1,FPR1_2,FPR1_3,FPR1_4

       EXTRN  FPR2
       EXTRN  FPR2_LP,FPR2_HP
       EXTRN  FPR2_1,FPR2_2,FPR2_3,FPR2_4

       EXTRN  FPR3
       EXTRN  FPR3_LP,FPR3_HP
       EXTRN  FPR3_1,FPR3_2,FPR3_3,FPR3_4
       EXTRN  FPR4
       EXTRN  FPR4_LP,FPR4_HP
       EXTRN  FPR4_1,FPR4_2,FPR4_3,FPR4_4
       EXTRN  FPR5
       EXTRN  FPR5_LP,FPR5_HP
       EXTRN  FPR5_1,FPR5_2,FPR5_3,FPR5_4

       EXTRN  FPR3_X,FPR4_X,FPR5_X
$      LIST
```

(3) REF2.INC

```
$      NOLIST
;*************************************************************
;*
;*  78K3 FLOATING POINT REGISTER LOAD FUNCTION REFFERENCE
;*
;*
;*
;*************************************************************
       EXTRN LLD21,LLD21X
       EXTRN       LLD31X
       EXTRN LLD41,LLD41X
       EXTRN LLD51,LLD51X
       EXTRN       LLD32X
       EXTRN LLD52
       EXTRN       LLD13X
       EXTRN       LLD23X
       EXTRN LLD24,LLD24X
       EXTRN LLD15,LLD15X
       EXTRN LLD25,LLD25X
       EXTRN LLD1C,LLD1CX
       EXTRN LLD2C,LLD2CX
       EXTRN       LXC13X
       EXTRN       LXC14X
       EXTRN       LXC15X
$      LIST
```

(4) ASCII.INC

```
$     NOLIST
;*************************************************************
;
; 78K3 ASCII CODE DEFINE
;
;
;
;*************************************************************
A_PL  EQU   02BH    ;'+'
A_MN  EQU   02DH    ;'-'
A_PD  EQU   02EH    ;'.'
A_NL  EQU   000H    ;nul
A_BL  EQU   020H    ;blank
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_BL  EQU   12
N_E   EQU   11
N_9   EQU   9

S_INDX EQU   7

@_INDX MACRO
       DB A_PL,A_MN,A_PD,A_NL
       DB A_BL,A_E ,A_E2
       ENDM
$     LIST
```

(5) DFLT.SRC

```
$       TITLE   ('FLOATING POINT REGISTERS')
        NAME    M_DFLT
;************************************************
;*
;*  78K3 FLOATING POINT REGISTERS
;*
;*
;*
;************************************************

        PUBLIC  FPR1
        PUBLIC  FPR1_LP,FPR1_HP
        PUBLIC  FPR1_1,FPR1_2,FPR1_3,FPR1_4

        PUBLIC  FPR2
        PUBLIC  FPR2_LP,FPR2_HP
        PUBLIC  FPR2_1,FPR2_2,FPR2_3,FPR2_4

        PUBLIC  FPR3
        PUBLIC  FPR3_LP,FPR3_HP
        PUBLIC  FPR3_1,FPR3_2,FPR3_3,FPR3_4
        PUBLIC  FPR4
        PUBLIC  FPR4_LP,FPR4_HP
        PUBLIC  FPR4_1,FPR4_2,FPR4_3,FPR4_4
        PUBLIC  FPR5
        PUBLIC  FPR5_LP,FPR5_HP
        PUBLIC  FPR5_1,FPR5_2,FPR5_3,FPR5_4

        PUBLIC  FPR3_X,FPR4_X,FPR5_X

        DSEG    SADDRP
;******** FLOATING POINT REGISTER 1 **
FPR1:
FPR1_LP:
FPR1_1:
        DS      1
FPR1_2:
        DS      1
FPR1_HP:
FPR1_3:
        DS      1
FPR1_4:
        DS      1
```

```
;******** FLOATING POINT REGISTER 2 **
FPR2:
FPR2_LP:
FPR2_1:
        DS      1
FPR2_2:
        DS      1
FPR2_HP:
FPR2_3:
        DS      1
FPR2_4:
        DS      1


        DSEG

;******** FLOATING POINT REGISTER 3 **
FPR3:
FPR3_LP:
FPR3_1:
        DS      1
FPR3_2:
        DS      1
FPR3_HP:
FPR3_3:
        DS      1
FPR3_4:
        DS      1

;******** FLOATING POINT REGISTER 4 **
FPR4:
FPR4_LP:
FPR4_1:
        DS      1
FPR4_2:
        DS      1
FPR4_HP:
FPR4_3:
        DS      1
FPR4_4:
        DS      1

;******** FLOATING POINT REGISTER 5 **
FPR5:
FPR5_LP:
FPR5_1:
        DS      1
FPR5_2:
        DS      1
FPR5_HP:
FPR5_3:
        DS      1
FPR5_4:
        DS      1
```

```
;******** FLOATING POINT REGISTER 4th MANTISSA **
FPR3_X:
        DS      1
FPR4_X:
        DS      1
FPR5_X:
        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,LSUBX,LMLTX,LDIVX
        PUBLIC  LNOR

        CSEG
        RSS     0
;************************************************
;*
;*  78K3 FLOATING POINT ADDITION FUNCTION
;*
;*    DESTINATION REGISTER :FPR1
;*    SOURCE REGISTER      :FPR2
;*
;*    RESULT : FPR1 += FPR2
;*             ERROR then set CY
;*
;************************************************

LADD:
        RP2 = #0              ;clear 4th mantissa
LADDX:
        AX  = FPR2_HP
        AX += AX
        R6  = A               ;FPR2 exp.

        AX  = FPR1_HP
        AX += AX
        X   = A               ;FPR1 exp.
```

```
;******* CHECK DIFFERENCE OF EXPONENT **

        A -= R6                 ;difference of exp.

        R7 = #0
        if_bit (!CY)
          if (A >= #SHORT*BYTE || R6 == R7)
            goto T_ADD9
          endif
        else
          if (A <= #LOW(-(SHORT*BYTE)) || X == R7)
            FPR1_HP = FPR2_HP
            FPR1_LP = FPR2_LP
            R5 = R4
            goto T_ADD9
          endif
          X = R6
        endif

        A <-> X

;******* LOAD MANTISSA **
                        ;* RP4.RP3 <- FPR1 mantissa
                        ;* AX.RP2  <- FPR2 mantissa
        AX <-> FPR1_LP          ;STORE : FPR1_1 <- diff.
                                ;      : FPR1_2 <- exp.
        R7 = X
        R8 = A

        A  = FPR1_3
        A |= #80H               ;(set mantissa MSB)
        R9 = A
        R6 = R5

        AX = FPR2_LP
        R5 = X
        X  = A

        A  = FPR2_3
        A |= #80H               ;(set mantissa MSB)
```

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

        if (FPR1_1 != #0)       ;exp. agree?
          if_bit (!FPR1_1.7)            ;destination is higher
            repeat
              SHRW AX,1
              RORC R5,1
              RORC R4,1
              FPR1_1--
            until_bit (Z)
          else                          ;source is higher
            repeat
              SHRW RP4,1
              RORC R7,1
              RORC R6,1
              FPR1_1++
            until_bit (Z)
          endif
        endif

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

        RP3 <-> RP4

        FPR2_4 ^= FPR1_4
        if_bit (!FPR2_4.7)      ;sign agree?
          ADDW RP2,RP4
          ADDC X,R6
          ADDC A,R7

          if_bit (CY)
            FPR1_2++            ;normalize mantissa overflow
            if_bit (Z)
              goto ERROR
            endif
            RORC A,1
            RORC X,1
            RORC R5,1
            RORC R4,1
          endif

        else
```

```
          if (AX == RP3)
            if (RP2 == RP4)
              goto ZERO
            endif
          endif
          if_bit (CY)           ;abs(source) < abs(destination)
            RP2 <-> RP4
            AX  <-> RP3
          else
            FPR1_4 ^= #80H      ;turn sign bit
          endif

          SUBW RP2,RP4
          SUBC X,R6
          SUBC A,R7
LNOR:
          while_bit (!A.7)      ;normalize catastrophic cancellation
            FPR1_2--
            if_bit (Z)
              goto ZERO
            endif
            ADDW RP2,RP2
            ADDC X,X
            ADDC A,A
          endw
        endif

;******* STORE FPR1 **

T_STOR:
        A <-> X
        A <-> FPR1_2     ;2nd mantissa

        CY = FPR1_4.7
        RORC A,1
        X.7 = CY
        FPR1_HP = AX     ;sign,exponent & 1st mantissa
        FPR1_1 = R5 (A)  ;3rd mantissa
        R5 = R4          :4th mantissa

T_ADD9:
        A = #R_OK
        CLR1 CY
        RET
ZERO:
        FPR1_HP = #0
        goto T_ADD9
```

```
ERROR:
        A = #R_ERR
        SET1 CY
        RET


;************************************************
;*
;*  78K3 FLOATING POINT SUBTRACTION FUNCTION
;*
;*    DESTINATION REGISTER : FPR1
;*    SOURCE REGISTER      : FPR2
;*
;*    RESULT : FPR1 -= FPR2
;*             ERROR then set CY
;*
;************************************************

LSUB:
        RP2 = #0             ;clear 4th mantissa
LSUBX:
        FPR2_4 ^= #80H
        goto LADDX


;************************************************
;*
;*  78K3 FLOATING POINT MULTIPLICATION FUNCTION
;*
;*    DESTINATION REGISTER : FPR1
;*    SOURCE REGISTER      : FPR2
;*
;*    RESULT : FPR1 *= FPR2
;*             ERROR then set CY
;*
;************************************************

LMLT:
        RP2 = #0             ;clear 4th mantissa
LMLTX:
        AX  = FPR2_HP
        AX += AX
        R6  = A              ;FPR2 exp.

        AX  = FPR1_HP
        AX += AX             ;FPR1 exp.
```

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

        X = #0
        if (A == #0 || R6 == X)
          goto ZERO
        endif

;******* MULTIPLE EXPONENT **

        A += R6
        if_bit (CY)
          A -= #ZEROEX
          if_bit (!CY)          ;exp. >= 100H
            goto ERROR
          endif
        else
          A -= #ZEROEX
          if_bit (CY)           ;exp. < 0
            goto ZERO
          endif
        endif

        FPR1_4 ^= FPR2_4        ;FPR1_4.7 <- sign

;******* LOAD MANTISSA **
                                ;d: FPR2_LP.RP3 <- FPR1 mantissa
                                ;s: AX•RP2      <- FPR2 mantissa
                                ;store exp. to FPR1_2
        AX <-> FPR1_LP
        R7  =  X
        R6  =  R5
        X   =  A
        A   =  FPR1_3
        A  |=  #80H             ;(set mantissa MSB)

        AX <-> FPR2_LP
        R5  =  X
        X   =  A
        A   =  FPR2_3
        A  |=  #80H             ;(set mantissa MSB)

;******* CALC. MANTISSA (set result to AX•RP2)**

        RP4 = AX                ;HIGH(s)

        MULUW RP3
        RP3 = AX                ;HIGH(LOW(d)*HIGH(s))

        AX = FPR2_LP
        MULUW RP2
        RP2 = AX                ;HIGH(HIGH(d)*LOW(s))

        AX = FPR2_LP
        MULUW RP4               ;HIGH(d)*HIGH(s)
```

```
        RP2 += RP3
        R6 = #0
        ADDC X,R6
        ADDC A,#0

        RP2 += RP4
        ADDC X,R6
        ADDC A,#0

;******* NORMALIZE **

        if_bit (A.7)
          FPR1_2++              ;2 <= mantissa < 4
          if_bit (Z)
            goto ERROR          ;exp. = 100H
          endif
        else
          if (FPR1_2 == #0)     ;1 <= mantissa < 2
            goto ZERO
          endif
          ADDW RP2,RP2
          ADDC X,X
          ADDC A,A
        endif

        goto T_STOR


;************************************************
;*
;*  78K3 FLOATING POINT DIVISION FUNCTION
;*
;*   DESTINATION REGISTER : FPR1
;*   SOURCE REGISTER      : FPR2
;*
;*   RESULT : FPR1 /= FPR2
;*            ERROR then set CY
;*
;************************************************

LDIV:
        RP2 = #0
LDIVX:
        AX  = FPR2_HP
        AX += AX
        R6  = A             ; FPR2 exp.

        AX  = FPR1_HP
        AX += AX            ; FPR1 exp.
```

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

        X = #0
        if (R6 == X)
          goto ERROR
        endif
        if (A == #0)
          goto ZERO
        endif

;******* DIVIDE EXPONENT **

        A -= R6
        if_bit (CY)
          A += #ZEROEX-1
          if_bit (!CY)                ;exp. <= 0
            goto ZERO
          endif
        else
          A += #ZEROEX-1
          if_bit (CY)                 ;exp. > 100H
            goto ERROR
          endif
        endif

        FPR1_4 ^= FPR2_4              ;set SIGN(FPR1_4.7)

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

        AX <-> FPR1_LP                ;STORE : FPR1_2 <- (exp.- 1)
        D   =  X
        E   =  R5
        X   =  A
        A   =  FPR1_3
        A  |=  #80H                   ;(set mantissa MSB)
        UP  =  AX                     ;h•j : UP•DE <- FPR1 mantissa

        AX  =  FPR2_LP
        R6  =  A
        R5  =  X
        A   =  FPR2_3
        A  |=  #80H                   ;(set mantissa MSB)
        R7  =  A                      ;k•1 : RP3•RP2 <- FPR2 mantissa

;******* DIVIDE MANTISSA (set quotient to RP2•XA•CY) **

        RP4 = RP3
        AX  = UP

        DIVUX RP3                     ;s1 : RP3        <-  surplus(h•j/k)
        FPR1_4.0 = X.0 (CY)           ;q1 : FPR1_4.0•DE <- quotient(h•j/k)

        AX = RP4
        MULUW RP3
        RP3 = AX                      ;high(s1*k)
```

```
        AX = UP
        MULUW RP2               ;high(h*1)

        RP3 -= AX               ;|high(s1*k)-high(h*1)|
        if_bit (CY)
          AX  = #0
          AX -= RP3             ;CY : sign of high(s1*k)-high(h*1)
          RP3 = AX
        endif

        AX = RP4
        MULUW RP4               ;high(k*k)

        RP4 =  DE
        AX <-> RP3
        DE  = #0

        DIVUX RP3       ;AX•DE <- quotient(|high(s1*k)-high(h*1)|*10000H
                        ;                 / high(k*k))
        if_bit (!CY)
          RP2 = DE
          AX += RP4
        else
          RP2  = #0
          RP2 -= DE
          SUBC R8,X
          SUBC R9,A
          AX   = RP4
        endif
        CY ^= FPR1_4.0

;******* NORMALIZE **

        if_bit (CY)
          FPR1_2++              ;1 <= mantissa < 2
          if_bit (Z)
            goto ERROR
          endif
          RORC A,1
          RORC X,1
          RORC R5,1
          RORC R4,1
        endif

        goto T_STOR

        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
        RSS     0
;*****************************************************************
;*
;*  78K3 FLOATING POINT FUNCTION THAT
;*
;*     CALC. A POLYNOMIAL EXPRESSION by EXTENDED FORMAT
;*
;*                                  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:
;*        FPR3 <- x , FPR4 <- y, FPR1 <- 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 T_PLY1
LPLY2:
        repeat
        CALL !LXC13X
T_PLY1:
        CALL !LLD24X
        CALL !LMLTX

        CALL !LLD2CX
        CALL !LMLTX
```

```
  CALL !LXC13X

  CALL !LLD23X
  CALL !LADDX

  AX  = !FPR3_HP
  AX += AX
  if (A == #0)
    RET
  endif

  R6  = A
  AX  = FPR1_HP
  AX += AX

  A  -= R6
  if_bit (!CY)
    if (A >= #(SHORT-1)*BYTE+4)
      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)

        PUBLIC  LLD21, LLD21X
        PUBLIC         LLD31X
        PUBLIC  LLD41, LLD41X
        PUBLIC  LLD51, LLD51X
        PUBLIC         LLD32X
        PUBLIC  LLD52
        PUBLIC         LLD13X
        PUBLIC         LLD23X
        PUBLIC  LLD24, LLD24X
        PUBLIC  LLD15, LLD15X
        PUBLIC  LLD25, LLD25X
        PUBLIC  LLD1C, LLD1CX
        PUBLIC  LLD2C, LLD2CX
        PUBLIC         LXC13X
        PUBLIC         LXC14X
        PUBLIC         LXC15X

        CSEG
        RSS     0
;************************************************
;*
;*  78K3 FLOATING POINT REGISTER LOAD FUNCTIONS
;*
;*
;*
;************************************************

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

LLD21X:
        R4      = R5
LLD21:
        FPR2_LP = FPR1_LP
        FPR2_HP = FPR1_HP
        RET

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

LLD31X:
        !FPR3_X  = R5 (A)
        !FPR3_LP = FPR1_LP (AX)
        !FPR3_HP = FPR1_HP (AX)
        RET
```

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

LLD41X:
        !FPR4_X  = R5 (A)
LLD41:
        !FPR4_LP = FPR1_LP (AX)
        !FPR4_HP = FPR1_HP (AX)
        RET

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

LLD51X:
        !FPR5_X  = R5 (A)
LLD51:
        !FPR5_LP = FPR1_LP (AX)
        !FPR5_HP = FPR1_HP (AX)
        RET

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

LLD32X:
        !FPR3_X  = R4 (A)
        !FPR3_LP = FPR2_LP (AX)
        !FPR3_HP = FPR2_HP (AX)
        RET

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

LLD52:
        !FPR5_LP = FPR2_LP (AX)
        !FPR5_HP = FPR2_HP (AX)
        RET

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

LLD13X:
        R5      = !FPR3_X (A)
        FPR1_LP = !FPR3_LP (AX)
        FPR1_HP = !FPR3_HP (AX)
        RET

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

LLD23X:
        R4      = !FPR3_X (A)
        FPR2_LP = !FPR3_LP (AX)
        FPR2_HP = !FPR3_HP (AX)
        RET
```

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

LLD24X:
        R4      = !FPR4_X (A)
LLD24:
        FPR2_LP = !FPR4_LP (AX)
        FPR2_HP = !FPR4_HP (AX)
        RET

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

LLD15X:
        R5      = !FPR5_X (A)
LLD15:
        FPR1_LP = !FPR5_LP (AX)
        FPR1_HP = !FPR5_HP (AX)
        RET

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

LLD25X:
        R4      = !FPR5_X (A)
LLD25:
        FPR2_LP = !FPR5_LP (AX)
        FPR2_HP = !FPR5_HP (AX)
        RET

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

LLD1CX:
        R5      = [DE+] (A)
LLD1C:
        FPR1_LP = [DE+] (AX)
        FPR1_HP = [DE+] (AX)
        RET

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

LLD2CX:
        R4      = [DE+] (A)
LLD2C:
        FPR2_LP = [DE+] (AX)
        FPR2_HP = [DE+] (AX)
        RET
```

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

LXC13X:
        A        = !FPR3_X
        A       <-> R5
        !FPR3_X  =  A
        AX       = !FPR3_LP
        AX      <-> FPR1_LP
        !FPR3_LP =  AX
        AX       = !FPR3_HP
        AX      <-> FPR1_HP
        !FPR3_HP =  AX
        RET

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

LXC14X:
        A        = !FPR4_X
        A       <-> R5
        !FPR4_X  =  A
        AX       = !FPR4_LP
        AX      <-> FPR1_LP
        !FPR4_LP =  AX
        AX       = !FPR4_HP
        AX      <-> FPR1_HP
        !FPR4_HP =  AX
        RET

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

LXC15X:
        A        = !FPR5_X
        A       <-> R5
        !FPR5_X  =  A
        AX       = !FPR5_LP
        AX      <-> FPR1_LP
        !FPR5_LP =  AX
        AX       = !FPR5_HP
        AX      <-> FPR1_HP
        !FPR5_HP =  AX
        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

C1_X    EQU     0A2H
C1_1    EQU     0DAH
C1_2    EQU     00FH
C1_3    EQU     0C9H
C1_4    EQU     03FH

        CSEG
        RSS     0
;*******************************************************
;*
;*  78K3 FLOATING POINT SINE FUNCTION
;*
;*     input  condition : FPR1 <- x
;*
;*     output conditions: FPR1 <- sin(x)
;*
;*******************************************************
LSIN:

;******* TRANS. sin(x) to (sign)sin(x’ +nπ/2) : 0<=x’<π/2 **

        CALL !LMOD90

;******* TRANS. to (sign’)sin(x’’) by n : 0<=x’’<π/2 **
LSIN90:
        C >>= 1
        if_bit (CY)             ;(n = odd)?
          SET1 FPR1_4.7
          DE = #C1

          CALL !LLD2CX
          CALL !LADDX           ; π/2 -x’
        endif
```

```
        C >>= 1                 ;(n/2 = odd)?
        CY ^= PSWH.7            ;  turn sign bit

        FPR1_4.7 = CY           ;set sign bit

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

          CALL !LLD41X          ;set x'' to FPR4•FPR4_X

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

          CALL !LXC14X          ;set x''*x'' to FPR4•FPR4_X
                                ;set x'' to FPR1•R5
          DE = #CK
          C = #S_PLY
          CALL !LPLY

          A = #R_OK
          CLR1 CY
          RET

;*********************************************
;*       GET MOD by π/2
;*
;* input  conditions : FPR1 <- x
;* output conditions : FPR1•R5 = x % π/2
;*                     C.(0,1bit) <- quotient
;*                     UF <- sign of x
;*********************************************
LMOD90:
        R5 = #0                 ;clear 4th mantissa
        C  = #0
        PSWH.7 = FPR1_4.7 (CY)  ;set sign bit
        CLR1 FPR1_4.7           ;|x|

        while (forever)
          if (FPR1_HP == #C1_4*100H+C1_3)
            if (FPR1_LP == #C1_2*100H+C1_1)
              A = R5
              CMP A,#C1_X
            endif
          endif

          if_bit (CY)
            RET
          endif

          CALL !LLD31X  ;esc. x to FPR3
```

```
          DE = #C2
          CALL !LLD2CX
          CALL !LMLTX                  ;x / (π/2) : quotient
          R5  = #0
          FPR1_1 = #0
          FPR1_2 &= #0FCH              ;valid digit -> 14bit

          CALL !LTOF
          if_bit (!CY)
            if_bit (!Z)                ;if (include decimal digit)
              CALL !FTOL               ;  cut decimal digit
            endif
            AX = DE
            C += X                     ;add last 2bit of quotient
          endif

          DE = #C1
          CALL !LLD2CX
          CALL !LMLTX                  ; int(x/(π/2)) * π/2

          SET1 FPR1_4.7
          CALL !LLD23X
          CALL !LADDX                  ; x - int(x/(π/2))* π/2
        endw

C1:
        DB C1_X,C1_1,C1_2,C1_3,C1_4    ; const π/2
C2:
        DB 06EH,083H,0F9H,022H,03FH    ;       2/π

CK:                                    ; coefficient array of LPLY
        DB 0AAH,0AAH,0AAH,02AH,0BEH    ; const  -1/6
        DB 0CCH,0CCH,0CCH,04CH,0BDH    ;        -1/20
        DB 0C3H,030H,00CH,0C3H,0BCH    ;        -1/42
        DB 0E3H,038H,08EH,063H,0BCH    ;        -1/72
        DB 04FH,009H,0F2H,014H,0BCH    ;        -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
        RSS     0
;*******************************************************
;*
;*  78K3 FLOATING POINT COSINE FUNCTION
;*
;*     input  condition  : FPR1 <- x
;*
;*     output conditions : FPR1 <- cos(x)
;*
;*******************************************************
LCOS:

;******* TRANS. cos(x) to cos(x’ +nπ/2) : 0<=x’<π/2 **

        CALL !LMOD90

;******* TRANS. to sin(x’ +(n+1)π/2) **

LCOS90:
        CLR1 PSWH.7    ;clear sign bit
        C++            ;n++

        goto LSIN90

        END
```

(11) LTAN.SRC

```
$       TITLE   (‘TANGENT FUNCTION’)
        NAME    M_LTAN

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

        EXTRN   LDIVX
        EXTRN   LMOD90,LSIN90,LCOS90

        PUBLIC  LTAN

        CSEG
        RSS     0
;*****************************************************
;*
;*  78K3 FLOATING POINT TANGENT FUNCTION
;*
;*    input  condition  : FPR1 <- x
;*
;*    output conditions : FPR1 <- tan(x)
;*                        ERROR then set CY
;*
;*****************************************************
LTAN:

;******* TRANS. sin(x) to (sign)sin(x’ +nπ/2)
;              cos(x) to       cos(x’ +nπ/2) : 0<=x’<π/2 **

        CALL !LMOD90

        B = C           ;esc. n
        CALL !LLD51X    ;esc. x’

;******* GET (sign)sin(x’ +nπ/2) / cos(x’ +nπ/2) **

        CALL !LSIN90
        CALL !LXC15X

        C = B
        CALL !LCOS90

        CALL !LLD21X
        CALL !LLD15X
        goto LDIVX

          END
```

(12) LLOG.SRC

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

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

        EXTRN   LADDX,LSUBX,LMLTX,LDIVX
        EXTRN   LPLY2
        EXTRN   FTOL

        PUBLIC  LLOG,LLOGX

C0_3    EQU     0B5H   ;1st mantissa of √2

S_PLY   EQU     4

        CSEG
        RSS     0
;*******************************************************
;*
;*  78K3 FLOATING POINT LOGARITHMIC FUNCTION
;*
;*    input  condition  : FPR1 <- x
;*
;*    output conditions : FPR1 <- log(x)
;*                        ERROR then set CY
;*
;*******************************************************
LLOG:

        R5 = #0
LLOGX:
        AX = FPR1_HP
        X += X
        ADDC A,A              ;x exponent(xe + exponent  bias)

;******* EXCEPTION **

        if_bit (CY || Z)
          A = #R_ERR           ;zero,negative exception
          SET1 CY
          RET
        endif
```

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

        X = A
        A = #0
        AX -= #ZEROEX
        DE  = AX                ;exponent value (:xe)

        !FPR3_X  = R5 (A)       ;set mantissa value (:xf) to FPR3•FPR3_X
        !FPR3_LP = FPR1_LP (AX)
        AX       = FPR1_HP

        A  = #ZEROEX/2
        A <-> X
        A |= #80H
        if (A >= #C0_3)
          A &= #7FH             ;xf/2 (:xf')
          DE++                  ;xe+1 (:xe')
        endif
        A <-> X
        !FPR3_HP = AX

        CALL !FTOL              ;real value of xe'

        DE = #C1
        CALL !LLD2CX
        CALL !LMLTX             ;exponent part log (xe'*log2)
        CALL !LLD41X            ;STORE xe'*log2 to FPR4•FPR4_X

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

        CALL !LLD13X
        DE = #C2
        CALL !LLD2CX
        CALL !LADDX             ;xf' +1

        CALL !LXC13X
        DE = #C2
        CALL !LLD2CX
        CALL !LSUBX             ;xf' -1

        CALL !LLD23X
        CALL !LDIVX             ;(xf' -1)/(xf' +1)  :x'

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

        !FPR3_X  = R5 (A)
        !FPR3_LP = FPR1_LP (AX)
        RP3 = FPR1_HP (AX)

        ADD  R6,R6
        ADDC R7,R7
        if_bit (!Z)
          AX += #80H            ;set 2x' to FPR3•FPR3_X
        endif
        !FPR3_HP = AX
```

```
        CALL !LLD21X
        CALL !LMLTX             ; x'*x'
        CALL !LXC14X            ;set x'*x' to FPR4•FPR4_X

        CALL !LLD23X
        CALL !LADDX             ;set xe'*log2+2x' to FPR1•R5

        DE = #CK
        C  = #S_PLY
        CALL !LPLY2

        A = #R_OK
        CLR1 CY
        RET
C1:
        DB 0F7H,017H,072H,031H,03FH     ; const  log2
C2:
        DB 000H,000H,000H,080H,03FH     ;        1

CK:                                     ; coefficient array of LPLY
        DB 0AAH,0AAH,0AAH,0AAH,03EH     ; const  1/3
        DB 099H,099H,099H,019H,03FH     ;        3/5
        DB 0B6H,06DH,0DBH,036H,03FH     ;        5/7
        DB 0C7H,071H,01CH,047H,03FH     ;        7/9

        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
        RSS     0
;*******************************************************
;*
;*  78K3 FLOATING POINT LOGARITHMIC FUNCTION 2
;*
;*     input  condition : FPR1 <- x
;*
;*     output conditions: FPR1 <- log10(x)
;*                        ERROR then set CY
;*
;*******************************************************
LLOG10:

;******* log(x) / log10 **

        CALL !LLOG
        if_bit (CY)
          RET
        endif

        DE = #C1
        CALL !LLD2CX
        goto LMLTX

C1:
        DB 0A9H,0D8H,05BH,0DEH,03EH ; const 1/log10

        END
```

(14) LEXP.SRC

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

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

        EXTRN   LADDX,LSUBX,LMLTX
        EXTRN   LPLY
        EXTRN   LTOF,FTOL

        PUBLIC  LEXP,LEXPX

S_PLY   EQU     6

        CSEG
        RSS     0
;*******************************************************
;*
;*  78K3 FLOATING POINT EXPONENTIAL FUNCTION
;*
;*    input  condition  : FPR1 <- x
;*
;*    output conditions : FPR1 <- e^x
;*                        ERROR then set CY
;*
;*******************************************************
LEXP:
        R5 = #0
LEXPX:
        PSWH.7 = FPR1_4.7 (CY)  ;esc sign bit

;******* TRANS. to 2^(x/log2) **

        DE = #C1
        CALL !LLD2CX
        CALL !LMLTX             ;1/log2 * x

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

        if_bit (!CY)
          CALL !LTOF            ;trunc(x/log2)
        endif
        if_bit (CY)
          goto T_FLOW
        endif
```

```
        if_bit (Z)
          A = R5
          CMP A,#0
        endif
        if_bit (!Z)             ;if ((dec(x/log2) != 0) &&
          if (FPR1_HP >= #8080H) ;   (negative))
            DE--                ;  floor(x/log2) = trunc(x/log2)-1
          endif
        endif

        AX = DE
        AX += #ZEROEX

        if (A != #0)
          goto T_FLOW
        endif

        B = X                   ;esc exp.

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

        CALL !LLD21X
        CALL !FTOL      ;floor(x/log2)
        NOT1 FPR1_4.7
        CALL !LADDX     ;x' : x/log2 -floor(x/log2)

        if (FPR1_4 == #3FH)     ;(1/2<=x'<1)
          DE = #C2+1
          CALL !LLD2C
          R4 = #02H     ;(power adjust to (x'<1) for boundary)
          CALL !LSUBX   ;x' :  decimal(x/log2)-1
        endif

        CALL !LLD41X    ;set x'

        DE = #CK
        CALL !LLD2CX
        CALL !LMLTX     ;set log2*x'

        C = #S_PLY
        CALL !LPLY

        DE = #C2
        CALL !LLD2CX
        CALL !LADDX     ;2^(x')
```

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

        B >>= 1
        FPR1_3.7 = CY
        FPR1_4 = B (A)
T_EXP9:
        A = #R_OK
        CLR1 CY
        RET
T_FLOW:
        if_bit (PSWH.7)
          FPR1_HP = #0
          goto T_EXP9
        endif

        A = #R_ERR
        SET1 CY
        RET

C1:
        DB 029H,03BH,0AAH,0B8H,03FH     ; const  1/log2
C2:
        DB 000H,000H,000H,080H,03FH     ;        1

CK:                                     ; coefficient array of LPLY2
        DB 0F7H,017H,072H,031H,03FH     ; const. log2
        DB 0F7H,017H,072H,0B1H,03EH     ;        log2/2
        DB 0F5H,01FH,098H,06CH,03EH     ;        log2/3
        DB 0F7H,017H,072H,031H,03EH     ;        log2/4
        DB 0F9H,0DFH,0F4H,00DH,03EH     ;        log2/5
        DB 0F5H,01FH,098H,0ECH,03DH     ;        log2/6
        DB 01BH,089H,0CBH,0CAH,03DH     ;        log2/7

        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
        RSS     0
;*******************************************************
;*
;*  78K3 FLOATING POINT EXPONENTIAL FUNCTION 2
;*
;*    input  condition :  FPR1 <- x
;*
;*    output conditions:  FPR1 <- 10^x
;*                        ERROR then set CY
;*
;*******************************************************
LEXP10:
        R5 = #0                 ;clear 4th mantissa

;******* TRANSLATE TO e^(log10*x) **

        C = FPR1_4 (A)

        DE = #C1
        CALL !LLD2CX
        CALL !LMLTX
        if_bit (CY)                     ;overflow
          C += C                        ;(CY <- sign bit)
          NOT1 CY
          if_bit (!CY)                  ;x<0
            FPR1_HP = #0
            A = #R_OK
          endif
          RET
        endif

        goto LEXPX

C1:
        DB 0DDH,08DH,05DH,013H,040H ;const log10

        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
        RSS     0
;*****************************************************
;*
;*  78K3 FLOATING POINT POWER FUNCTION
;*
;*    input  condition : FPR1 <- a , FPR2 <- b
;*
;*    output conditions: FPR1 <- a^b
;*                       ERROR then set CY
;*
;*****************************************************
LPOW:
        RP3 = FPR1_HP (AX)

        AX  = FPR2_HP
        AX += AX        ;exp. of b

        ADD  R6,R6
        ADDC R7,R7      ;exp. of a

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

        if_bit (Z)
          CMP A,#0
          if_bit (Z || FPR2_4.7)
            goto ERROR          ;0^0,0^(negative)=overflow
          endif
          goto T_POW9           ;0^(positive)=0
        endif

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

        X = #0                  ;X.0 :sign of result
        if_bit (CY)
          if (A == #0)
            DE = #C1
            CALL !LLD1C
            goto T_POW9                 ;x^0=1
          endif
```

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

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

          X = A

          A = FPR2_3
          A |= #80H
          B = A
          C = FPR2_2 (A)
          A = FPR2_1

          A <-> X

          A -= #ZEROEX+BYTE*(SHORT-1)-1
          if_bit (CY)
            repeat
              SHRW BC,1
              RORC X,1
              if_bit (CY)
                goto ERROR              ;include decimal digit
              endif
              A++
            until_bit (Z)               ;X.0 = UNIT1
          endif
          if_bit (!Z)                   ;if (exp.of b > 23)
            X = #0                      ;  {b = even}
          endif
        endif

        HL = AX                         ;esc. UNIT1(X.0)

;***** CALC. e^(b * log|a|) **

        CALL !LLD52                     ;esc. b to FPR5
        CLR1 FPR1_4.7
        CALL !LLOG                      ;log|a|
        CALL !LLD25                     ;ret. b to FPR2
        R4 = #0
        CALL !LMLTX                     ;b * log|a|

        if_bit (CY)                     ;overflow
          if (!FPR5_4 >= #80H) (A)
            FPR1_HP = #0
            goto T_POW9
          endif
          goto ERROR
        endif

        CALL !LEXPX
```

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

        if_bit (CY)
          RET
        endif

        AX = HL
        if_bit (X.0)                    ;if (b == odd integer && a<0)
          SET1 FPR1_4.7                 ;  {set sign bit}
        endif
T_POW9:
        A = #R_OK
        CLR1 CY
        RET
ERROR:
        A = #R_ERR
        SET1 CY
        RET
C1:
        DB 000H,000H,080H,03FH ;const 1

        END
```

(17) LSQRT.SRC

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

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

        EXTRN   LADDX,LDIVX

        PUBLIC  LSQRT
        PUBLIC  LSQRTX

C_LIM   EQU     5      ;limiter of approximate

        CSEG
        RSS     0
;*****************************************************
;*
;*  78K3 FLOATING POINT SQUARE ROOT FUNCTION
;*
;*   input  condition  : FPR1 <- x
;*
;*   output conditions : FPR1 <- √(x)
;*                       ERROR then set CY
;*
;*****************************************************
LSQRT:
        R5  = #0        ;clear 4th mantissa
LSQRTX:
        AX  = FPR1_HP
        ADD X,X
        ADDC A,A

;******* EXCEPTION **

        if_bit (Z)
          goto T_QRT9   ;zero
        endif
        if_bit (CY)
          A = #R_ERR    ;negative
          RET
        endif
```

```
;******* TRANS. √‾(a) TO √‾(r)*2^n (1≤r<4) **

        A >>= 1
        if_bit (CY)
          FPR1_4 = #(ZEROEX-1)/2        ;r'/2 (r'=r)
        else
          FPR1_4 = #(ZEROEX-2)/2        ;r'/2 (r'=r/4)
        endif
        ADDC A,#ZEROEX/2
        NOT1 FPR1_3.7
        B = A           ;escape exp.part root (n)

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

        CALL !LLD31X    ;esc. r'/2 to FPR3•FPR3_X

        DE = #C1
        CALL !LLD2CX
        CALL !LADDX     ; r'/2 + .5  : 2ndary approximate (R2)

        C = #C_LIM-2
        repeat
          CALL !LLD41X  ; esc.previous approximate(Ri) to FPR4•FPR4_X
          CALL !LLD21X
          CALL !LLD13X
          CALL !LDIVX   ; r'/2 / Ri

          CALL !LLD24X
          FPR2_HP -= #80H       ; Ri/2

          CALL !LADDX   ; Ri/2 + r'/(2Ri) : next approximate

          C--

        until_bit (Z)

        A = B
        A >>= 1
        FPR1_4 = A              ;ret. exp.part root (n)
        FPR1_3.7 = CY
T_QRT9:
        A = #R_OK
        CLR1 CY
        RET
C1:
        DB 000H,000H,000H,000H,03FH ;const .5

        END
```

(18) LASIN.SRC

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

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

        EXTRN   LMLT
        EXTRN   LADDX,LDIVX
        EXTRN   LSQRTX,LATANX

        PUBLIC  LASIN

C1_4    EQU     03FH
C1_3    EQU     080H
C1_2    EQU     000H
C1_1    EQU     000H

        CSEG
        RSS     0
;*******************************************************
;*
;*  78K3 FLOATING POINT ARCSINE FUNCTION
;*
;*    input  condition  : FPR1 <- x
;*
;*    output conditions : FPR1 <- arcsin(x)
;*                        ERROR then set CY
;*
;*******************************************************
LASIN:

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

        CALL !LLD51             ;store x to FPR5

        CLR1 FPR1_4.7           ;x <- |x|

        ;* |x| = 1? *
        if (FPR1_HP == #C1_4*100H+C1_3)
          if (FPR1_LP == #C1_2*100H+C1_1)
            DE = #C2            ;|x|=1 exception
            CALL !LLD1CX

            A  = !FPR5_4        ;FPR1 <- π/2
            A += A
            FPR1_4.7 = CY       ;return sign bit
            A = #R_OK
            CLR1 CY
            RET
          endif
        endif
```

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

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

        CALL !LLD21
        CALL !LMLT      ;x*x
        SET1 FPR1_4.7

        DE = #C1
        CALL !LLD2CX
        CALL !LADDX     ;1-x*x
        CALL !LSQRTX    ;√‾(1-x*x)

        CALL !LLD21X
        CALL !LLD15
        R5 = #0
        CALL !LDIVX     ;x/√‾(1-x*x)

        goto LATANX

C1:
        DB 000H,C1_1,C1_2,C1_3,C1_4 ;const  1
C2:
        DB 0A2H,0DAH,00FH,0C9H,03FH ;       π/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
        RSS     0
;*******************************************************
;*
;*  78K3 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_4.7

        DE = #C1
        CALL !LLD2CX
        goto LADDX              ;π/2 -arcsin(x)

C1:
        DB 0A2H,0DAH,00FH,0C9H,03FH ;const π/2

        END
```

(20) LATAN.SRC

```
$       TITLE   (‘ARCTANGENT FUNCTION’)
        NAME    M_LATAN

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

        EXTRN   LADDX,LSUBX,LMLTX,LDIVX
        EXTRN   LPLY2
        EXTRN   LRCPNX

        PUBLIC  LATAN
        PUBLIC  LATANX

S_PLY   EQU     3

        CSEG
        RSS     0
;*******************************************************
;*
;*  78K3 FLOATING POINT ARCTANGENT FUNCTION
;*
;*    input  condition  : FPR1 <- x
;*
;*    output conditions : FPR1 <- arctan(x)
;*
;*******************************************************
LATAN:
         R5  = #0        ;clear 4th mantissa
LATANX:
        AX  = FPR1_HP
        AX += AX

        X.6 = CY        ; sign bit

;******* TRANS. arctan(x) to (sign)(arctan(x’))   or
;                           (sign)(π/2 -arctan(x’)) **

                        ;x’ = |x|       case |x|< 1
                        ;x’ = 1/|x|     case |x|>=1

        CLR1 FPR1_4.7   ;|x|

        CMP A,#ZEROEX

        X.7 = CY        ;(|x|<1)
        B   = X

        if_bit (!CY)
          CALL !LRCPNX  ;1/|x|
        endif
```

```
;******* TRANS. arctan(x') to arctan(W)+arctan(V) case if x'>=1/8 **

                        ;W : 1/8,3/8,5/8 or 7/8
                        ;   (-1/8 <= x'-W <= 1/8)
                        :V = (x'-W)/(1+x'*W)

        C = #0
        if (FPR1_4 >= #3EH)     ;1/8
          CALL !LLD41X

          FPR2_4 = FPR1_4
          if (FPR1_HP >= #3F40H)        ;6/8
                FPR2_3 = #060H          ;W=7/8
                C = #(SHORT+1)*4
          elseif (FPR1_4 >= #3FH)       ;4/8
                FPR2_3 = #020H          ;W=5/8
                C = #(SHORT+1)*3
          elseif (FPR1_3 >= #80H)       ;2/8
                FPR2_3 = #0C0H          ;W=3/8
                C = #(SHORT+1)*2
          else
                FPR2_3 = #000H          ;W=1/8
                C = #(SHORT+1)*1
          endif

          FPR2_LP = #0
          R4 = #0
          CALL !LLD32X

          CALL !LMLTX   ;x'*W

          DE = #C1
          CALL !LLD2CX
          CALL !LADDX   ;x'*W +1

          CALL !LXC14X
          CALL !LLD23X
          CALL !LSUBX   ;x' -W

          CALL !LLD24X
          CALL !LDIVX   ;(x'-W)/(1+x'*W)
        endif

;******* CALC. APPROXIMATE POLYNOMIAL FUNCTION **

        CALL !LLD41X

        CALL !LLD21X
        CALL !LMLTX     ;V*V

        CALL !LXC14X    ;set V*V to FPR4•FPR4_X
```

```
        DE = #CK0
        CALL !LLD2CX
        CALL !LMLTX     ;4a0*V

        CALL !LLD31X    ;set 4a0*V to FPR3•FPR3_X

        X   = C
        A   = #0
        DE  = #CW
        DE += AX
        CALL !LLD2CX
        CALL !LADDX     ;4a0*V +arctan(W)

        DE = #CK1
        C  = #S_PLY
        CALL !LPLY2     ;arctan(x')

        B += B          ;CY <- (|x|<1)

        if_bit (!CY)    ;|x|>=1?
          NOT1 FPR1_4.7

          DE = #C2
          CALL !LLD2CX
          CALL !LADDX   ;π/2 -arctan(x')
        endif

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

        B += B          ;CY <- sign bit
        FPR1_4.7 = CY

        A = #R_OK
        CLR1 CY
        RET
C1:
        DB 000H,000H,000H,080H,03FH  ;const 1
C2:
        DB 0A2H,0DAH,00FH,0C9H,03FH  ;      π/2

CW:
        DB 000H,000H,000H,000H,000H  ;      0
        DB 0D5H,0D4H,0ADH,0FEH,03DH  ;      arctan(1/8)
        DB 00FH,0CAH,0B0H,0B7H,03EH  ;      arctan(3/8)
        DB 05FH,05DH,000H,00FH,03FH  ;      arctan(5/8)
        DB 02CH,03EH,005H,038H,03FH  ;      arctan(7/8)

                                     ;coefficient array(an)
                                     ;      of approximate
CK0:
         DB 0FEH,0FFH,0FFH,07FH,03FH ;cof.  a0    * 4
CK1:     DB 032H,0A4H,0AAH,0AAH,0BEH ;      a1/a0 *16
         DB 05CH,0DAH,090H,019H,0BFH ;      a2/a1 *16
         DB 001H,058H,0FEH,031H,OBFH ;      a3/a2 *16

         END
```

(21) LHSIN.SRC

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

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

        EXTRN   LMLT
        EXTRN   LSUBX
        EXTRN   LPLY
        EXTRN   LEXP
        EXTRN   LRCPNX

        PUBLIC  LHSIN

S_PLY   EQU     3

        CSEG
        RSS     0
;*******************************************************
;*
;*  78K3 FLOATING POINT HYPERBOLICSINE FUNCTION
;*
;*    input  condition  : FPR1 <- x
;*
;*    output conditions : FPR1 <- sinh(x)
;*                        ERROR then set CY
;*
;*******************************************************
LHSIN:

        HL = FPR1_HP (AX)

;******* CALC. (e^|x|-e^(-|x|))/2  case if |x| >= 0.5 **

        A &= #7FH
        if (A >= #ZEROEX/2)     ;|x| >= 0.5

          CLR1 FPR1_4.7         ;x <- |x|

          CALL !LEXP            ;e^|x|
          if_bit (CY)
            RET                 ;overflow
          endif

          CALL !LLD31X
          CALL !LRCPNX          ;e^(-|x|)
          CALL !LLD23X

          CALL !LSUBX           ;e^(-|x|)-e^|x|
          FPR1_HP -= #80H       ;(e^(-|x|)-e^|x|)/2
```

```
          HL += HL              ;CY <- sign bit
          FPR1_4.7 = CY

;******** CALC. DIRECT APPROXIMATE  case if |x| < 0.5 **

          else          ;|x| < 0.5

          CALL !LLD41

          CALL !LLD21
          CALL !LMLT       ;x*x

          CALL !LXC14X
          R5 = #0

          DE = #CK
          C  = #S_PLY
          CALL !LPLY
        endif

        A = #R_OK
        CLR1 CY
        RET
CK:                                ; coefficient array of LPLY
        DB 0AAH,0AAH,0AAH,02AH,03EH ; const 1/6
        DB 0CCH,0CCH,0CCH,04CH,03DH ;       1/20
        DB 0C3H,030H,00CH,0C3H,03CH ;       1/42

        END
```

(22) LHCOS.SRC

```
$       TITLE   (‘HYPERBOLICCOSINE FUNCTION’)
        NAME    M_LHCOS

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

        EXTRN   LADDX
        EXTRN   LRCPNX
        EXTRN   LEXP

        PUBLIC  LHCOS

        CSEG
        RSS     0
;*******************************************************
;*
;*  78K3 FLOATING POINT HYPERBOLICCOSINE FUNCTION
;*
;*    input  condition  : FPR1 <- x
;*
;*    output conditions : FPR1 <- cosh(x)
;*                        ERROR then set CY
;*
;*******************************************************
LHCOS:

        CLR1 FPR1_4.7

;******** CALC. (e^|X|+e^(-|X|))/2 **

        CALL !LEXP      ;e^|x|
        if_bit (CY)
          RET           ;overflow
        endif

        CALL !LLD31X
        CALL !LRCPNX    ;e^(-|x|)
        CALL !LLD23X

        CALL !LADDX     ;e^|x| +e^(-|x|)
        FPR1_HP -= #80H ;(e^|x| +e^(-|x|))/2
        RET

        END
```

(23) LHTAN.SRC

```
$       TITLE   ('HYPERBOLICTANGENT FUNCTION')
        NAME    M_LHTAN

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

        EXTRN   LDIVX
        EXTRN   LHSIN,LHCOS

        PUBLIC  LHTAN

        CSEG
        RSS     0
;*****************************************************
;*
;*  78K3 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 = #C1
          CALL !LLD1C           ;tanh(positive infinity)=1
          if (!FPR5_4 >= #80H) (A)
            SET1 FPR1_4.7       ;tanh(negative infinity)=-1
          endif
          A = #R_OK
          CLR1 CY
          RET
        endif

        CALL !LXC15X    ;esc. cosh(x) to FPR5
```

```
;******* CALC. LHSIN **

        CALL !LHSIN     ;sinh(x)

;******* CALC. LHTAN **

        CALL !LLD25X    ;ret. cosh(x)
        goto  LDIVX     ;sinh(x)/cosh(x)
C1:
        DB 000H,000H,080H,03FH ; const 1

        END
```

(24) LABS.SRC

```
$       TITLE   (‘ABSOLUTE FUNCTION’)
        NAME    M_LABS

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

        PUBLIC  LABS

        CSEG
        RSS     0
;*******************************************************
;*
;*  78K3 FLOATING POINT ABSOLUTE FUNCTION
;*
;*    input  condition  : FPR1 <- x
;*
;*    output conditions: FPR1 <-  |x|
;*
;*******************************************************
LABS:

        CLR1 FPR1_4.7

        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   LDIVX

        PUBLIC  LRCPN
        PUBLIC  LRCPNX

        CSEG
        RSS     0
;************************************************
;*
;*  78K3 FLOATING POINT FUNCTION
;*                      GET RECIPROCAL NUMBER
;*
;*    input condition   : FPR1 <- x
;*
;*    output conditions : FPR1 <- 1/x
;*                         ERROR then set CY
;*
;************************************************
LRCPN:
        R5 = #0         ;clear 4th mantissa
LRCPNX:
        CALL !LLD21X

        DE = #C1
        CALL !LLD1CX

        goto LDIVX


C1:
        DB 000H,000H,000H,080H,03FH

        END
```

(26) POTORA.SRC

```
$       TITLE   ('TRANS. TO RIGHT ANGLE COORDINATES')
        NAME    M_POTORA

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

        EXTRN   LMLTX
        EXTRN   LMOD90,LSIN90,LCOS90

        PUBLIC  POTORA

        CSEG
        RSS     0
;*****************************************************
;*
;*  78K3 FLOATING POINT FUNCTION THAT
;*         TRANS. COORDINATES FROM POLE TO RIGHT ANGLE
;*
;*    input  condition : FPR1 <- r, FPR2 <- θ
;*
;*    output conditions: FPR1 <- x, FPR2 <- y
;*                       ERROR then set CY
;*
;*****************************************************
POTORA:
        HL  = FPR1_HP (AX)
        ADD X,X
        ADDC A,A

        UP  = FPR1_LP (AX)

;******* EXCEPTION **

        if_bit (Z)              ;r = 0
          FPR2_HP = #0
          goto T_ORA9
        endif

        if_bit (CY)             ;r < 0
          A = #R_ERR
          RET
        endif
```

```
;******* TRANSLATE **

        FPR1_HP = FPR2_HP
        FPR1_LP = FPR2_LP

        CALL !LMOD90            ; θ -> (sign)(θ‘+ nπ/2) (0 ≤ θ‘< π/2)

        B = C                   ;esc. n
        CALL !LLD51X            ;esc. θ‘

        CALL !LSIN90            ;sin((sign)(θ‘+nπ/2))

        CALL !LXC15X            ;esc. sin((sign)(θ‘+nπ/2))
        C = B                   ;ret. n

        CALL !LCOS90            ;cos(θ‘+np/2)

        CALL !LXC15X

        FPR2_HP = HL (AX)
        FPR2_LP = UP (AX)
        R4 = #0

        CALL !LMLTX             ;rsinθ

        CALL !LXC15X

        FPR2_HP = HL (AX)
        FPR2_LP = UP (AX)
        R4 = #0

        CALL !LMLTX             ;rcosθ

        CALL !LLD25X

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   LADDX,LMLT,LDIV
        EXTRN   LSQRTX,LATANX

        PUBLIC  RATOPO

        CSEG
        RSS     0
;*****************************************************
;*
;*  78K3 FLOATING POINT FUNCTION THAT
;*       TRANS. COORDINATES FROM RIGHT ANGLE TO POLE
;*
;*    input  condition  : FPR1 <- x, FPR2 <- y
;*
;*    output conditions : FPR1 <- r, FPR2 <- θ
;*                        ERROR then set CY
;*
;*****************************************************
RATOPO:
        AX  = FPR1_HP
        H   = A                 ;H.7 <- sign of x
        AX += AX
        R6  = A                 ;R6 <- exp. of x

        AX  = FPR2_HP
        AX += AX                ;A <- exp. of y
        PSWH.7 = CY             ;PSWH.7 <- sign of y

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

        if (A == #0)
          if (R6 == A)          ;if (x==0 && y==0)
            goto T_OPO9         ;  {(r,θ) = (0,0)}
          endif
          CLR1 PSWH.7           ;if (y==0) clear sign bit of y
        endif
```

```
;******** CALC. x*x+y*y  **

        CALL !LLD41             ;FPR4 <- x
        CALL !LLD52             ;FPR5 <- y

        CALL !LLD21
        CALL !LMLT              ;x*x
        if_bit (CY)
          RET
        endif

        CALL !LLD31X

        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 !LXC15X
        CALL !LLD24
        CALL !LDIV
        if_bit (CY)
          DE = #C1
          CALL !LLD1C                   ;arctan(infinity)=π/2 or -π/2
          FPR1_4.7 = PSWH.7 (CY)        ;sign of θ <- sign of y
          goto T_OPO8
        endif

;******** CALC. θ **

        CALL !LATANX
        AX = HL
        if_bit (A.7)
          DE = #C2
          CALL !LLD2CX                  ;x<0, y≥0 : θ=arctan(y/x)+π
          CY = PSWH.7
          FPR2_4.7 = CY                 ;x<0, y<0 : θ=arctan(y/x)-π
          CALL !LADDX
        endif
```

```
;******** CALC. r  **

T_OPO8:
        CALL !LXC15X
        CALL !LSQRTX            ;√‾(x*x+y*y)

        CALL !LLD25X            ; θ

T_OPO9:
        A = #R_OK
        CLR1 CY
        RET
C1:
        DB  0DAH,00FH,0C9H,03FH     ;const π/2
C2:
        DB 0A2H,0DAH,00FH,049H,040H ;      π

        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   LNOR
        EXTRN   LEXPX

        PUBLIC  ATOL

S_VIRT  EQU     27    ;maximum length of mantissa

        CSEG
        RSS     0
;*****************************************************
;*
;*  78K3 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
;*                        HL keep
;*****************************************************
ATOL:
        PUSH HL

;******* TRANS. SIGN **

        FPR1_4 = #0     ;sign keeper

        CALL !GETC
        if     (A == #N_PL)
          CALL !GETC
        elseif (A == #N_MN)
          SET1 FPR1_4.7
          CALL !GETC
        endif
```

```
;******* TRANS. MANTISSA TO BINARY **

        FPR2_LP = #0    ;work FPR2_1 : decimal digit counter (F)
                        ;     FPR2_2 : neglect digit counter (N)
        FPR2_HP = #80H  ;     FPR2_3 : mantissa digit counter
                        ;     FPR2_4.0 : decimal digit flag
                        ;     FPR2_4.1 : neglect digit flag
        while (forever)
          if (A < #N_9+1)
            if_bit (FPR2_3.7)
              R4 = A                ;initial digit
              R5 = #0
              RP3 = #0
              FPR2_3 = #S_VIRT-1+1

            else

              FPR2_3- -
              if_bit (Z)
                goto ERROR              ;mantissa length over
              endif

              if_bit (!FPR2_4.1)            ;(not neglect) ?
                B = #0
                C = A
                AX = #10
                MULUW RP2
                RP2 += BC
                BC = AX
                AX = #10
                MULUW RP3
                ADDC R6,C
                ADDC R7,B       ;RP3•RP2 <- mantissa val.

                if_bit (!Z)     ;if (work area fill)
                  SET1 FPR2_4.1;  {neglect forward digit}
                endif
              else
                FPR2_2++           ;neglect digit count up
              endif
            endif

            if_bit (FPR2_4.0)
              FPR2_1++          ;decimal digit count up
            endif
```

```
          elseif (A == #N_PD)
            if_bit (!FPR2_4.0)
              SET1 FPR2_4.0            ;begin count of decimal digit
            else
              goto ERROR               ;duplicate
            endif
          else
            break
          endif

          CALL !GETC
        endw

;******* EXCEPTION **

        if_bit (FPR2_3.7)              ;no mantissa digit
          goto ERROR
        endif

        BC = #0
        if (RP2 == BC && RP3 == BC)
          FPR1_HP = #0          ;ZERO EXCEPTION
          goto T_TOL9
        endif

        if (A >= #N_MN)
          goto ERROR
        endif

        FPR2_1 -= FPR2_2        ;decimal digit - neglect digit (F-N)

;******* NORMALIZE MANTISSA val. **

        FPR2_2 = A

        AX  = RP3
        FPR1_2 = #ZEROEX+SHORT*BYTE-1
        CALL !LNOR              ;normalize with sign bit

        CALL !LLD51X            ;esc. mantissa val(A')
        A = FPR2_2

;******* TRANS. EXP_PART **

        X = #0          ;work exp val
        CLR1 FPR1_1.7   ;     sign of exp

        if (A < #N_BL)          ;'E' or 'e'
          CALL !GETC
          if    (A == #N_PL)
            CALL !GETC
          elseif(A == #N_MN)
            SET1 FPR1_1.7
            CALL !GETC
          endif
```

```
          if (A >= #N_9+1)
            goto ERROR
          endif

          X = A         ;1st. digit
          CALL !GETC
          if (A < #N_9+1)
            A <-> X
            B = X
            X = #10
            MULU X
            X += B
            CALL !GETC
          endif
        endif

        if (A != #N_NL && A != #N_BL)
          goto ERROR
        endif

        if_bit (FPR1_1.7)
          A = #0
          A -= X
        else
          A = X         ;exp.part value (:B)
        endif

;******* UNITE MANTISSA.val & EXP.val **

        A -= FPR2_1             ; B - (F-N) (:B')
        CVTBW
        DE = AX
        CALL !FTOL              ; B' -> real

        DE = #C1
        CALL !LLD2CX
        CALL !LMLTX             ;log2(10)*B'
        CALL !LLD21X

        CALL !LTOF
        CALL !FTOL
        NOT1 FPR1_4.7
        CALL !LADDX     ;dec(log2(10)*B')

        UP = DE         ;int(log2(10)*B')

        DE = #C2
        CALL !LLD2CX
        CALL !LMLTX     ;dec(log2(10)*B')*log2
        CALL !LEXPX

        CALL !LLD25X    ;ret. A'
        CALL !LMLTX     ;A' * e^(dec(log2(10)*B')*log2)
```

```
        AX  = FPR1_HP
        AX += AX

        X = A
        A = #0
        AX += UP        ;exp.part RESULT
        if_bit (A.7)
          X = #0                ;underflow
        elseif (A != #0)
          goto ERROR            ;overflow
        endif

        CY = FPR1_4.7
        RORC X,1
        FPR1_4 = X (A)
        FPR1_3.7 = CY
T_TOL9:
        POP HL
        A = #R_OK
        CLR1 CY
        RET
ERROR:
        POP HL
        A = #R_ERR
        SET1 CY
        RET

GETC:
        A = [HL+]
        if (A >= #A_0 && A < #A_9+1)
          A -= #A_0
          RET
        endif

        DE = #INDEX
        C = #S_INDX
        repeat
          if (A == [DE+])
            A = C
            A += #N_9
            RET
          endif
          C--
        until_bit (Z)

        A = #0FFH
        RET

INDEX:
        @_INDX
C1:
        DB 04BH,078H,09AH,054H,040H ;const. log2(10)
C2:
        DB 0F7H,017H,072H,031H,03FH ;       log2

        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   LADDX,LMLTX,LDIVX
        EXTRN   LLOG10,LEXP10
        EXTRN   LTOF,FTOL

        PUBLIC  LTOA

S_VIRT  EQU     8       ;string length of mantissa

C2_4    EQU     41H
C2_3    EQU     20H

        CSEG
        RSS     0
;*****************************************************
;*
;*  78K3 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
;*                      HL keep
;*****************************************************
LTOA:
        AX  = FPR1_HP
        AX += AX

;******* ZERO FORMAT **

        if (A == #0)
          [HL] = #A_0+A_NL*100H (AX)
          goto T_TOA9
        endif

        PUSH HL

;******* TRANS. to a * 10^b (1<= a <10) **

        R5 = #0         ;clear 4th mantissa

        CALL !LLD51X                    ;esc. x to FPR5•FPR5_X
        CLR1 FPR1_4.7                   ;x <- |x|
        CALL !LLOG10                    ;log10(|x|)
```

```
        CALL !LTOF              ;trunc integer(log10(|x|))
        if_bit (Z)
          A = R5
          CMP A,#0
        endif
        if_bit (!Z)             ;include decimal digit &&
          if (FPR1_HP >= #8080H) ;negative ?
            DE--                ;floor integer(log10(|x|))  : b
          endif
        endif

        AX  = DE
        if (AX == #38)
          CALL !LLD15X
          DE = #C1
          CALL !LLD2CX
          CALL !LDIVX           ;x / (10^38)
          UP = #38              ;esc. b
        else
          UP = DE               ;esc. b
          CALL !FTOL
          NOT1 FPR1_4.7
          CALL !LEXP10          ;10^(-b)
          CALL !LLD25X
          CALL !LMLTX           ;x * (10^(-b))
        endif

;******* OUTPUT MANTISSA **

        if_bit (FPR1_4.7)       ; a<0 ?
          [HL+] = #A_MN (A)
          CLR1 FPR1_4.7           ; a <- |a|
        endif

        if (FPR1_HP >= #C2_4*100H+C2_3) ;if (limit |a|<10 over)
          DE = #C3                      ;  {normalize}
          CALL !LLD2CX
          CALL !LMLTX
          UP++
        endif

        if (FPR1_HP < #3F80H)           ;if (limit |a|>=1 over)
          DE = #C2                      ;  {normalize}
          CALL !LLD2CX
          CALL !LMLTX
          UP--
        endif

        CALL !LTOF      ;integer(a)
        AX = DE
        AX += #A_PD*100H+A_0
        [HL+] = AX
```

```
        B = #S_VIRT-1
        repeat
          CALL !LLD21X
          CALL !FTOL
          SET1 FPR1_4.7
          CALL !LADDX           ;a - integer(a)
          DE = #C2
          CALL !LLD2CX
          CALL !LMLTX           ;(a - integer(a))*10

          CALL !LTOF
          AX = DE
          A  = X
          A += #A_0
          [HL+] = A

          B--
        until_bit (Z)

;******* OUTPUT EXP.PART **

        [HL+] = #A_E (A)

        AX = UP
        if (A != #0)
          [HL+] = #A_MN (A)
          AX = #0
          AX -= UP
        endif

        B = #10
        DIVUW B
        A = B

        AX += #A_0*100H+A_0
        [HL+] = AX

        [HL] = #A_NL (A)
        POP HL
T_TOA9:
        A = #R_OK
        CLR1 CY
        RET

C1:
        DB 051H,099H,076H,096H,07EH ;const 10^38
C2:
        DB 000H,000H,000H,C2_3,C2_4 ;const 10
C3:
        DB 0CDH,0CCH,0CCH,0CCH,03DH ;const 1/10

        END
```

(30) FTOL.SRC

```
$       TITLE   ('TRANS. FIXED TO REAL')
        NAME    M_FTOL

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

        PUBLIC  FTOL

        CSEG
        RSS     0
;*****************************************************
;*
;*  78K3 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_HP = AX
          goto T_TOL9
        endif

;******* GET ABSOLUTE VALUE **

        R5 = A                  ;R5.7 <- sign bit
        if (AX >= #8000H+1)
          AX = #0
          AX -= DE
        endif

;******* TRANSLATE **

        if (A == #0)
          FPR1_2 = #ZEROEX+BYTE-1
          A <-> X
        else
          FPR1_2 = #ZEROEX+BYTE*INTEGR-1
        endif

        while_bit (!A.7)
          AX += AX
          FPR1_2--
        endw
```

```
;******* STORE **

        A <-> X
        R5 += R5        ;CY <- sign bit

        R5 = #0         ;4th mantissa
        FPR1_1 = #0     ;3rd mantissa
        A <-> FPR1_2    ;2nd mantissa
        RORC A,1
        FPR1_HP = AX    ;sign,exponent,1st mantissa

        FPR1_3.7 = CY   ;exponent LSB

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
        RSS     0
;*******************************************************
;*
;*  78K3 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:
        AX  = FPR1_HP
        AX += AX
        A  -= #ZEROEX

;******* EXCEPTION **

        if_bit (CY)             ;integer(FPR1)=0 ?
          CMP A,#LOW(-ZEROEX)
          DE = #0
          goto T_TOF9
        endif

        A -= #BYTE*INTEGR
        if_bit (!CY)
          goto ERROR            ;overflow
        endif

;******* GET UNSIGNED INTEGER **

        A  <-> X
        RP3 = !FPR1_LP
        A   =  FPR1_3
        A  |= #80H      ;(set mantissa MSB)

        if_bit (!X.3)
          SET1   X.3
          R6 |=  R7
          A  <-> R7
          A   = #0
        endif
```

```
        C  =  X
        X  =  R7
        R7 = #0

        C++
        while_bit (!Z)
          SHRW AX,1
          ADDC R7,R7
          C++
        endw

;******* UNSIGNED -> SIGNED INTEGER **

        if_bit (FPR1_4.7)
          DE  = #0
          DE -= AX
          BP $ERROR
        else
          if (A >= #80H)
            goto ERROR
          endif
          DE = AX
        endif

;******* DECIMAL PART JUDGE **

        AX = RP3
        CMPW AX,#0
T_TOF9:
        A = #R_OK
        CLR1 CY
        RET
ERROR:
        A = #R_ERR
        SET1 CY
        RET

        END
```

[MEMO]

* APPENDIX REVISION HISTORY

The revision history is shown below. The chapters described in the revised-chapter column indicate those for the corresponding edition.

Edition	Major changes	Revised chapter
Second	The following products have been added. μPD78323(A), μPD78323(A1), μPD78323(A2), μPD78324(A), μPD78324(A1), μPD78324(A2), μPD78P324(A), μPD78P324(A1), μPD78P324(A2), μPD78350, μPD78350A, μPD78352A, μPD78P352, μPD78355, μPD78356, μPD78P356, μPD78355A, μPD78356A, μPD78P356A, μPD78362, μPD78P364, μPD78365, μPD78366, μPD78P368, μPD78370, μPD78372, μPD78P372	Throughout
	The following products have already been developed. μPD78P334, μPD78P334(A), μPD78P334(A1), μPD78P334(A2)	
Third	The following products have been added. μPD78356(A), μPD78P356(A), μPD78361A, μPD78362A, μPD78P364A, μPD78363A, μPD78365A, μPD78366A, μPD78368A, μPD78P368A, μPD78372(A), μPD78372(A1), μPD78372(A2), μPD78P372(A), μPD78P372(A1), μPD78P372(A2)	Throughout
	The following products have been deleted. μPD78355A, μPD78356A, μPD78P356A, μPD78362, μPD78P364, μPD78365, μPD78366, μPD78P368, μPD78370, μPD78372, μPD78P372	
	The following products have already been developed. μPD78P324, μPD78P324(A), μPD78P324(A1), μPD78P324(A2), μPD78350A, μPD78355, μPD78356, μPD78P356	

[MEMO]

Phase-out/Discontinued

Facsimile Message

From:

Name

Company

Tel. FAX

Address

Although NEC has taken all possible steps to ensure that the documentation supplied to our customers is complete, bug free and up-to-date, we readily accept that errors may occur. Despite all the care and precautions we've taken, you may encounter problems in the documentation. Please complete this form whenever you'd like to report errors or suggest improvements to us.

Thank you for your kind support.

North America
NEC Electronics Inc.
Corporate Communications Dept.
Fax: 1-800-729-9288
1-408-588-6130

Europe
NEC Electronics (Europe) GmbH
Technical Documentation Dept.
Fax: +49-211-6503-274

South America
NEC do Brasil S.A.
Fax: +55-11-889-1689

Hong Kong, Philippines, Oceania
NEC Electronics Hong Kong Ltd.
Fax: +852-2886-9022/9044

Korea
NEC Electronics Hong Kong Ltd.
Seoul Branch
Fax: 02-528-4411

Taiwan
NEC Electronics Taiwan Ltd.
Fax: 02-719-5951

Asian Nations except Philippines
NEC Electronics Singapore Pte. Ltd.
Fax: +65-250-3583

Japan
NEC Corporation
Semiconductor Solution Engineering Division
Technical Information Support Dept.
Fax: 044-548-7900

I would like to report the following error/make the following suggestion:

Document title: _____

Document number: _____ Page number: _____

If possible, please fax the referenced page or drawing.

Document Rating	Excellent	Good	Acceptable	Poor
Clarity	❑	❑	❑	❑
Technical Accuracy	❑	❑	❑	❑
Organization	❑	❑	❑	❑

Phase-out/Discontinued