APPLICATION NOTE

NEC

78K/0 SERIES

8-BIT SINGLE-CHIP MICROCOMPUTER

FLOATING-POINT ARITHMETIC PROGRAMS

μPD78014 SERIES
μPD78014Y SERIES
μPD78044 SERIES
μPD78054 SERIES
μPD78064 SERIES

Document No. IEA-1289A
(O. D. No. IEA-718A)
Date Published October 1993 P
Printed in Japan

APPLICATION NOTE

NEC

78K/0 SERIES

8-BIT SINGLE-CHIP MICROCOMPUTER

FLOATING-POINT ARITHMETIC PROGRAMS

μPD78014 SERIES
μPD78014Y SERIES
μPD78044 SERIES
μPD78054 SERIES
μPD78064 SERIES

NOTE: Purchase of NEC I^2C components conveys a license under the Philips I^2C Patent Rights to use these components in an I^2C system, provided that the system conforms to the I^2C Standard Specification as defined by Philips.

M7 92.6

Major Revisions in This Version

Section	Description
Whole manual	Change of name of products concerned . uPD78011, 78012 to uPD78011B, 78012B Addition of products concerned . uPD78011BY, 78012BY, 78013Y, 78014Y . uPD78042, 78043, 78044, 78P044 . uPD78052, 78053, 78054, 78P054, 78056, 78058 . uPD78062, 78063, 78064, 78P064
A-1	Addition of Appendix "Explanation of SPD Charts"

PREFACE

Intended Readership

This Application Note is intended for users' engineers who have an understanding of the functions of 78K/0 series products and wish to design floating point operation programs using these products.

78K/0 series products

- uPD78014 series : uPD78011B, 78012B, 78013, 78014, 78P014
- uPD78014Y series : uPD78011BY, 78012BY, 78013Y, 78014Y, 78P014Y
- uPD78044 series : uPD78042, 78043, 78044, 78P044
- uPD78054 series : uPD78052, 78053, 78054, 78P054, 78056*, 78058*
- uPD78064 series : uPD78062*, 78063, 78064, 78P064*

*: Under development

Purpose

The purpose of this Application Note is to give users an understanding of 78K/0 series product floating point operation application programs. The programs shown in this document are given as examples only, and are not intended for mass production design.

Organization

This Application Note covers the following topics:

- Calculation algorithms
- Four rules operations
- Functions (mathematics, coordinate conversion, type conversion)
- Execution results
- Program listings

The following Application Note is also available separately:

- Introductory Volume I (IEA-715)
- Introductory Volume II (IEA-740)

Quality Grade

Standard

Please refer to "Quality grade on NEC Semiconductor Devices" (Document number IEI-1209) published by NEC Corporation to know the specification of quality grade on the devices and its recommended applications.

Application Area

- Consumer products

Related Documentation

o 78K/0 Series Common Documentation

Document Name		Document No.
Application note	Introductory volume I	IEA-715
	Introductory volume II	IEA-740
	Floating point operation program volume	This application note
Selection guide		IF-375
Instruction table		IEM-5522
Instruction set		IEM-5521

o Individual Documents

● uPD78014 Series Documentation

Document Name \ Product	uPD78011B	uPD78012B	uPD78013	uPD78014	uPD78P014
Data sheet	In creation		IC-8201		IC-8111
User's manual	IEU-780				
Special function register table	IEM-5527				

● uPD78014Y Series Documentation

Document Name \ Product	uPD78011BY	uPD78012BY	uPD78013Y	uPD78014Y	uPD78P014Y
Data sheet	In creation				IC-8572
User's manual	IEU-780				
Special function register table	IEM-5527				

● uPD78044 Series Documentation

Product / Document Name	uPD78042	uPD780043	uPD78044	uPD78P044
Data sheet	IC-8497			IC-8499
User's manual	IEU-801			
Special function register table	IEM-5556			

● uPD78054 Series Documentation

Product / Document Name	uPD78052	uPD78053	uPD78054	uPD78P054	uPD78056	uPD78058
Data sheet	In creation					
User's manual	IEU-824					
Special function register table	IEM-5574					

● uPD78064 Series Documentation

Product / Document Name	uPD78062	uPD780063	uPD78064	uPD78P064
Data sheet	In creation			IP-8636
User's manual	IEU-817			
Special function register table	IEM-5568			

NOTE: The information in these related documents is subject to change without notice. For design purpose, etc., check if your documents are the latest ones and be sure to use the latest ones.

CONTENTS

CHAPTER 1. GENERAL DESCRIPTION 1-1

1.1 Floating point format 1-1
1.2 Function provided 1-3
1.3 File configuration 1-5
1.4 Program characteristics 1-8
1.4.1 Program location address 1-8
1.4.2 Reentrancy 1-8
1.4.3 Stack 1-8
1.4.4 Register banks 1-9
1.4.5 Saving registers, flags, etc.....................1-9
1.5 Data transfer method 1-9
1.5.1 Parameters and returned values 1-9
1.5.2 Operation result notification 1-9
1.6 Floating point registers 1-10
1.6.1 Floating point register 1 (FPR1).................1-10
1.6.2 Floating point register 2 (FPR2) 1-11
1.6.3 Floating point register 3 to 5 (FPR3 to FPR5)....1-11
1.6.4 Mantissa extension registers 1-12
1.6.5 Inter-floating point register load/exchange 1-12

CHAPTER 2. CALCULATION ALGORITHMS.............................2-1

2.1 Function expansion methods 2-1
2.2 Rounding method 2-1
2.3 Prevention of dropped digits 2-1
2.4 Errors due to polynomial addition/multiplication 2-1

CHAPTER 3. FOUR RULES OPERATIONS 3-1

3.1 Floating point addition operation (LADD) 3-2
3.2 Floating point subtraction operation (LSUB) 3-6
3.3 Floating point multiplication operation (LMLT) 3-7
3.4 Floating point division operation (LDIV) 3-12

CHAPTER 4. MATHEMATICAL FUNCTIONS 4-1

4.1 Common subroutines (LPLY, LPLY2) 4-3
4.2 sin function (LSIN) 4-7
4.3 cos function (LCOS) 4-11
4.4 tan function (LTAN) 4-13
4.5 Natural logarithm function (LLOG) 4-15
4.6 Common logarithm function (LLOG10)..................4-19
4.7 Exponent function (base = e)(LEXP) 4-21
4.8 Exponent function (base =10)(LEXP10) 4-26
4.9 Power function (LPOW) 4-28
4.10 Square root function (LSQRT) 4-32
4.11 arcsin function (LASIN) 4-36
4.12 arccos function (LACOS) 4-39
4.13 arctan function (LATAN) 4-42
4.14 sinh function (LHSIN) 4-47
4.15 cosh function (LHCOS) 4-50
4.16 tanh function (LHTAN) 4-52
4.17 Absolute value function (LABS) 4-55
4.18 Reciprocal function (LRCPN) 4-56

CHAPTER 5. COORDINATE CONVERSION FUNCTIONS 5-1

5.1 Polar coordinate → rectangular coordinate conversion function (POTORA) 5-1
5.2 Rectangular coordinate → polar coordinate conversion function (RATOPO) 5-4

CHAPTER 6. TYPE CONVERSION FUNCTIONS 6-1

6.1 Character string → floating point format conversion function (ATOL) 6-2
6.2 Floating point format → character string conversion function (LTOA).......................................6-11
6.3 2-byte integer type → floating point format conversion function (FTOL) 6-17
6.4 Floating point format → 2-byte integer type conversion function (LTOF) 6-20

CHAPTER 7. EXECUTION RESULTS 7-1

7.1 Floating point addition (LADD) 7-1
7.2 Floating point subtraction (LSUB) 7-2
7.3 Floating point multiplication (LMLT) 7-2
7.4 Floating point division (LDIV) 7-3
7.5 sin function (LSIN) 7-3
7.6 cos function (LCOS) 7-3
7.7 tan function (LTAN) 7-4
7.8 Natural logarithm function (LLOG) 7-4
7.9 Common logarithm function (LLOG10) 7-5
7.10 Exponent function (base = e)(LEXP) 7-5
7.11 Exponent function (base = 10)(LEXP10) 7-6
7.12 Power function (LPOW) 7-6
7.13 Square root function (LSQRT) 7-7
7.14 arcsin function (LASIN) 7-7
7.15 arccos function (LACOS) 7-8
7.16 arctan function (LATAN) 7-8
7.17 sinh function (LHSIN) 7-9
7.18 cosh function (LHCOS) 7-9
7.19 tanh function (LHTAN) 7-10
7.20 Absolute value function (LABS) 7-10
7.21 Reciprocal function (LRCPN) 7-10
7.22 Polar coordinate → rectangular coordinate conversion function (POTORA) 7-11
7.23 Rectangular coordinate → polar coordinate conversion function (RATOPO) 7-12
7.24 Character string → floating point format conversion function (ATOL) 7-13
7.25 Floating point format → character string conversion function (LTOA) 7-13
7.26 2-byte integer type → floating point format conversion function (FTOL) 7-14
7.27 Floating point format → 2-byte integer type conversion function (LTOF) 7-14

CHAPTER 8. PROGRAM LISTINGS 8-1

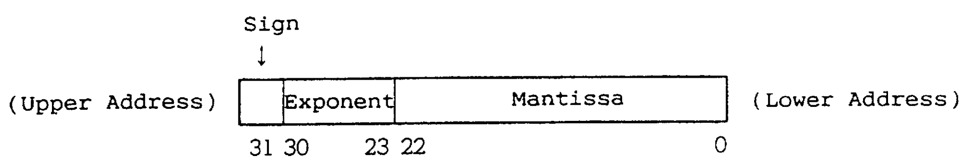
APPENDIX. EXPLANATION OF SPD CHARTS A-1

CHAPTER 1. GENERAL DESCRIPTION

1.1 FLOATING POINT FORMAT

In these operation programs, floating point numbers are represented in 4-byte format. The breakdown is as follows (see figure below):

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



Using this format, a number is expressed as shown below.

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

Each of the component parts is described in detail below.

(1) Mantissa

The mantissa is expressed as an absolute value, and bit positions 22 to 0 of the mantissa correspond to places 1 to 23 after the binary point.
The value of the exponent is adjusted so that the value of the mantissa is always in the range between 1 and 2 except when the floating point value is 0. As a

result, the first binary place (meaning the value 1) is always 1, and in this format it is possible to express a number in abbreviated form.

Remarks : The operation whereby the most significant bit (MSB) always has a value of 1 is called "normalization".

(2) Sign

This bit is 0 for a positive number and 1 for a negative number.

(3) Exponent

A base 2 exponent is expressed in the form of a one-byte integer (two's complement representation is used for a negative number), and the value obtained by adding a bias of 7FH to this value is used.
The relation between these values is shown in concrete terms below.

Exponent (Hexadecimal)	Exponent Value
FF	128
FE	127
:	:
81	2
80	1
7F	0
7E	-1
:	:
01	-126

NOTE : The floating point value indicates 0 only when the exponent is 0. In this case, the mantissa and sign are ignored.

(4) Numeric representation range

A floating point value x can be represent a value in the following range and "0".

$$\text{Approx. } 1.1755 \times 10^{-38} \leq |x| < \text{approx. } 6.8056 \times 10^{38}$$

1.2 FUNCTIONS PROVIDED

This section outlines the functions provided in these operation programs.
The following four kinds of functions are provided:

- Four rules operations
- Mathematical functions
- Coordinate conversion functions
- Numeric type conversion functions

Functions Provided	Functions		Function Names
Four rules operations	Addition		LADD
	Subtraction		LSUB
	Multiplication		LMLT
	Division		LDIV
Mathematical functions	Trigonometric functions	sin function	LSIN
		cos function	LCOS
		tan function	LTAN
	Natural logarithm function (log)		LLOG
	Common logarithm function ($\log_{10}$)		LLOG10
	Exponent function (base = e)		LEXP
	Exponent function (base = 10)		LEXP10
	Power (a^b)		LPOW
	Square root		LSQRT
	Inverse trigonometric functions	arcsin function	LASIN
		arccos function	LACOS
		arctan function	LATAN
	Hyperbolic functions	sinh function	LHSIN
		cosh function	LHCOS
		tanh function	LHTAN
	Absolute value		LABS
	Reciprocal		LRCPN

(to be continued)

(cont'd)

Functions Provided	Functions	Function Names
Coordinate conversion functions	Polar coordinate → rectangular coordinate conversion	POTORA
	Rectangular coordinate → polar coordinate conversion	RATOPO
Numeric type conversion functions	Character string → floating point format conversion	ATOL
	Floating point format → character string conversion	LTOA
	2-byte integer type → floating point format conversion	FTOL
	Floating point format → 2-byte integer type conversion	LTOF

1.3 FILE CONFIGURATION

These operation programs comprise the following four kinds of files:

- Object source files* : 24
- Common subroutine files* : 2
- Common data file : 1
- Include files : 4

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

(1) Object source files

Source File Name	Function Provided
LFLT1 .SRC	Four rules operations
LSIN .SRC	sin function
LCOS .SRC	cos function
LTAN .SRC	tan function
LLOG .SRC	Natural logarithm function
LLOG10 .SRC	Common logarithm function
LEXP .SRC	Exponent function (base = e)
LEXP10 .SRC	Exponent function (base = 10)
LPOW .SRC	Power function
LSQRT .SRC	Square root
LASIN .SRC	arcsin function
LACOS .SRC	arccos function
LATAN .SRC	arctan function
LHSIN .SRC	sinh function
LHCOS .SRC	cosh function
LHTAN .SRC	tanh function
LABS .SRC	Absolute value
LRCPN .SRC	Reciprocal
POTORA .SRC	Polar coordinate → rectangular coordinate conversion
RATOPO .SRC	Rectangular coordinate → polar coordinate conversion
ATOL .SRC	Character string → floating point format conversion
LTOA .SRC	Floating point format → character string conversion
FTOL .SRC	2-byte integer type → floating point format conversion
LTOF .SRC	Floating point format → 2-byte integer type conversion

(2) Common subroutine files

Source File Name	Function Provided
LFLT2 .SRC LLD .SRC	Polynomial calculation function Inter-floating point register load/exchange function

(3) Common data file

Source File Name	Definition
DFLT .SRC	Floating point register definition

(4) Include files

Source File Name	Definition
EQU .INC REF1 .INT RER2 .INC ASCII .INC	EQU definition Floating point register reference declaration Inter-floating point register load/exchange function use declaration ASCII code definition

The object module files for which linkage should be performed when using a function are given in the description of the individual function.

Remarks : NEC's 78K/0 series assembler package includes a librarian which can be used to create library files. If all the above programs are recorded in a library file, the necessary modules can be linked automatically when linkage is performed simply by specifying that library file. There are no particular restrictions on the order of recording items in a library file.

1.4 PROGRAM CHARACTERISTICS

1.4.1 PROGRAM LOCATION ADDRESS

(1) Work areas

The work areas used by these operation programs are called floating point registers (actually, these are simply global variables, but since their use is limited to floating point operations, they are referred to here as registers).

Reservation of the floating point register area is performed by means of the common data file DFLT.SRC. Floating point registers are reserved in the short direct addressing area. Location addresses in the short direct addressing area are arbitrary.

(2) Code area

This must be located in ROM, but apart from this there are no restrictions.

1.4.2 REENTRANCY

There is no reentrancy.
In a multiple task processing system, resource management is required to ensure that only one task can call a function, etc.

1.4.3 STACK

The maximum required stack size is shown in the individual function descriptions. When using a function, a stack of at least the size shown must be provided.

1.4.4 REGISTER BANKS

A register bank selection instruction is not used in these operation programs. The register bank selected when a function is called is used.

1.4.5 SAVING REGISTERS, FLAGS, ETC.

With the exception of certain type conversion functions, saving and restoration of register contents and flags to/from the stack is not performed. Also, the register used varies from function to function (the register used and saving of its contents are described in the individual function descriptions).

1.5 DATA TRANSFER METHOD

1.5.1 PARAMETERS AND RETURNED VALUES

Transfer is performed by means of the floating point registers described earlier. In these operation programs, the same registers are used throughout to store values operated on and returned values. In view of this, the value operated on is called the destination and the operation value is called the source in this manual.

Parameter and returned value settings are described in the individual function description.

1.5.2 OPERATION RESULT NOTIFICATION

In precise terms, there are five different termination statuses as follows:

(1) Normal termination

(2) Underflow

(3) Overflow

(4) Imaginary number representation

(5) Noncomputable (e.g. log(-1))

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

In the termination notification, error statuses ((3), (4), (5)) are not differentiated: only a normal termination or abnormal termination is reported.

Termination Status	A Register Contents	CY Flag
Normal termination	0	off
Abnormal termination	81H	on

1.6 FLOATING POINT REGISTERS

In this Application Note, work areas used by operation programs are called floating point registers. Here, the role of each register is described briefly.

In the following figures, one box represents one byte and the upper address is on the left.

1.6.1 FLOATING POINT REGISTER 1 (FPR1)

With most functions, this register is used to store the destination.

FPR1 consists of 4 consecutive bytes as shown in the figure below. It is located in the short direct addressing area (on a word boundary).

Exponent + sign	1st mantissa part + exponent LSB	2nd mantissa part	3rd mantissa part	
FPR1_4	FPR1_3	FPR1_2	FPR1_1	FPR1
	FPR1_HP		FPR1_LP	

↑ (on word border)

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

1.6.2 FLOATING POINT REGISTER 2 (FPR2)

This register is used to store the source.
FPR2 consists of 4 consecutive bytes as shown in the figure below. It is located in the short direct addressing area (on a word boundary).

Exponent + sign	1st mantissa part + exponent LSB	2nd mantissa part	3rd mantissa part	
FPR2_4	FPR2_3	FPR2_2	FPR2_1	FPR2
	FPR2_HP		FPR2_LP	

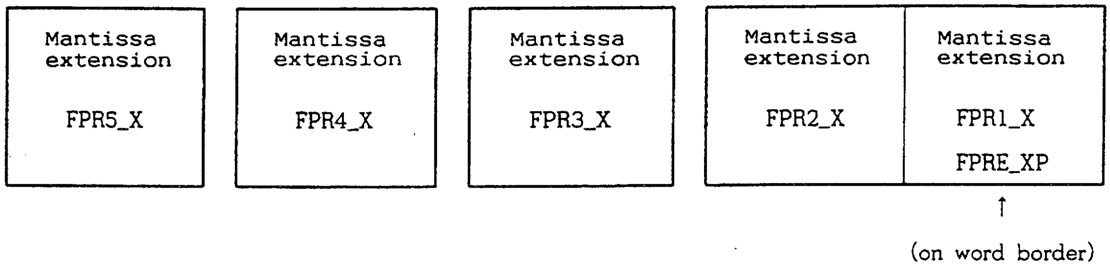
↑ (on word border)

The register configuration is identical to that of FPR1.

1.6.3 FLOATING POINT REGISTERS 3 TO 5 (FPR3 TO FPR5)

With mathematical functions, etc., these registers are used as a temporary work area.
FPR3 to FPR5 consists have the same configuration and global names as FPR1 and FPR2, and are located in the short direct addressing area (on a word boundary).

1.6.4 MANTISSA EXTENSION REGISTERS



These comprise an area for calculation of the 4th mantissa part (binary places 24 to 31) by extending the mantissa internally by one byte.

FPR1_X, FPR2_X, FPR3_X, FPR4_X and FPR5_X are used for FPR1, 2, 3, 4 and 5 respectively.

FPR1_X, FPR2_X, FPR3_X, FPR4_X and FPR5_X are located in the short direct addressing area.

Remarks : These registers are not used by all functions. Up to which register can be used with each function is explained in the individual function descriptions.

1.6.5 INTER-FLOATING POINT REGISTER LOAD/EXCHANGE

With mathematical functions and type conversion functions, register load/store/exchange operations are frequently performed. For this reason, inter-floating point register load/store functions and exchange functions are provided.

The load/store functions and exchange functions are listed below.

Function Name	Operation
LLD21, LLD21X	Loads 1st register contents into 2nd register
LLD31, LLD31X	Loads 1st register contents into 3rd register
LLD41, LLD41X	Loads 1st register contents into 4th register
LLD51, LLD51X	Loads 1st register contents into 5th register
LLD32	Loads 2nd register contents into 3rd register
LLD52	Loads 2nd register contents into 5th register
LLD13	Loads 3rd register contents into 1st register
LLD23, LLD23X	Loads 3rd register contents into 2nd register
LLD34, LLD24X	Loads 4th register contents into 2nd register
LLD15	Loads 5th register contents into 1st register
LLD25, LLD25X	Loads 5th register contents into 2nd register
LLD1C, LLD1CX	Loads constant data into 1st register
LLD2C, LLD2CX	Loads constant data into 2nd register
LXC13, LXC13X	Exchanges contents of 1st register and 3rd register
LXC14, LXC14X	Exchanges contents of 1st register and 4th register
LXC15, LXC15X	Exchanges contents of 1st register and 5th register

NOTE : Function names ending with "X" are load/exchange functions which include an extended mantissa.

CHAPTER 2. CALCULATION ALGORITHMS

This chapter gives a brief description of the algorithms on which the calculations are based.

2.1 FUNCTION EXPANSION METHODS

Three expansion methods are used:

- Square root : Newton-Raphson method
- Inverse trigonometric functions : Best approximation method
- Others : Taylor expansion method

2.2 ROUNDING METHOD

Rounding toward zero is used.

2.3 PREVENTION OF DROPPED DIGITS

When the expansion polynomial in a Taylor expansion is a series of differences, extreme digit dropping may occur depending on the range of values used. To prevent this kind of digit dropping, these operation programs use an expansion expression whereby the value of each term from the 1st term to the n'th term of the expansion expression approaches 0 monotonously and rapidly.

2.4 ERRORS DUE TO POLYNOMIAL ADDITION/MULTIPLICATION

When addition of 8-term polynomials which have been rounded toward 0 is performed in a number system using 24 bits as valid digits, a maximum error of 4 bits is included. Moreover. the same error is also included in the case of multiplication of x^8 seven times.

In order to minimize the cumulative effect of this kind of error, in these operation programs the mantissa is calculated internally as 31 bits (with the addition of 8 mantissa extension bits).

CHAPTER 3. FOUR RULES OPERATIONS

The following four rules operation functions are used.

(1) Floating point addition (LADD)

Performs addition with the value of FPR1 as the augend and the contents of FPR2 as the addend.

(2) Floating point subtraction (LSUB)

Performs subtraction with the value of FPR1 as the minuend and the contents of FPR2 as the subtrahend.

(3) Floating point multiplication (LMLT)

Performs multiplication with the value of FPR1 as the multiplicand and the contents of FPR2 as the multiplier.

(4) Floating point division (LDIV)

Performs division with the value of FPR1 as the dividend and the contents of FPR2 as the divisor.

The operation result is stored in FPR1.

3.1 FLOATING POINT ADDITION OPERATION (LADD)

(1) Processing

With the value of FPR1 designated as x and the value of FPR2 designated as y, returns x + y in FPR1.

(2) Object module files subject to linkage

DFLT, LFLT1

(3) Required stack size

2 (2-byte return address from LADD only)

(4) Registers used

AX, C, DE

(5) Work areas used

FPR1, FPR2, FPR1_X, FPR2_X

(6) Processing time (internal system clock = 8.38 MHz)

Average: 162 us
Maximum: 332 us (0.5 + (-0.50000006))

(7) Processing procedure

(a) If x or y is 0, the operation is ended with the non-zero value as the solution.
(b) If the exponent difference is 32 or more, the operation is ended with the larger value as the solution.
(c) If y exponent > x exponent, the contents of x and y are exchanged.
(d) The exponent of x is stored.

(e) The following method is used to perform mantissa addition/subtraction.
MSB (1) and the mantissa extension (8 bits) are added respectively to the x and y mantissas, and these are regarded as doubleword type variables d and s.

	31	30 – 8	7 – 0
d:	1	x mantissa	Mantissa extension

	31	30 – 8	7 – 0
s:	1	y mantissa	Mantissa extension

In the following procedure, the sum (or difference) of the mantissas is found.

(i) If the x and y exponents are different, s is right-shifted by the number of bits of the exponent difference.

(ii) If the signs of x and y are the same, addition of d and s is performed and the sign is stored.

(iii) If the signs are different, the smaller of d and s is subtracted from the larger, and the sign of the larger value is stored.

(f) Normalization is performed on the stored exponent and the mantissa solution found in (e), and the result is stored in FPR1 together with the sign bit.

(8) Processing diagram

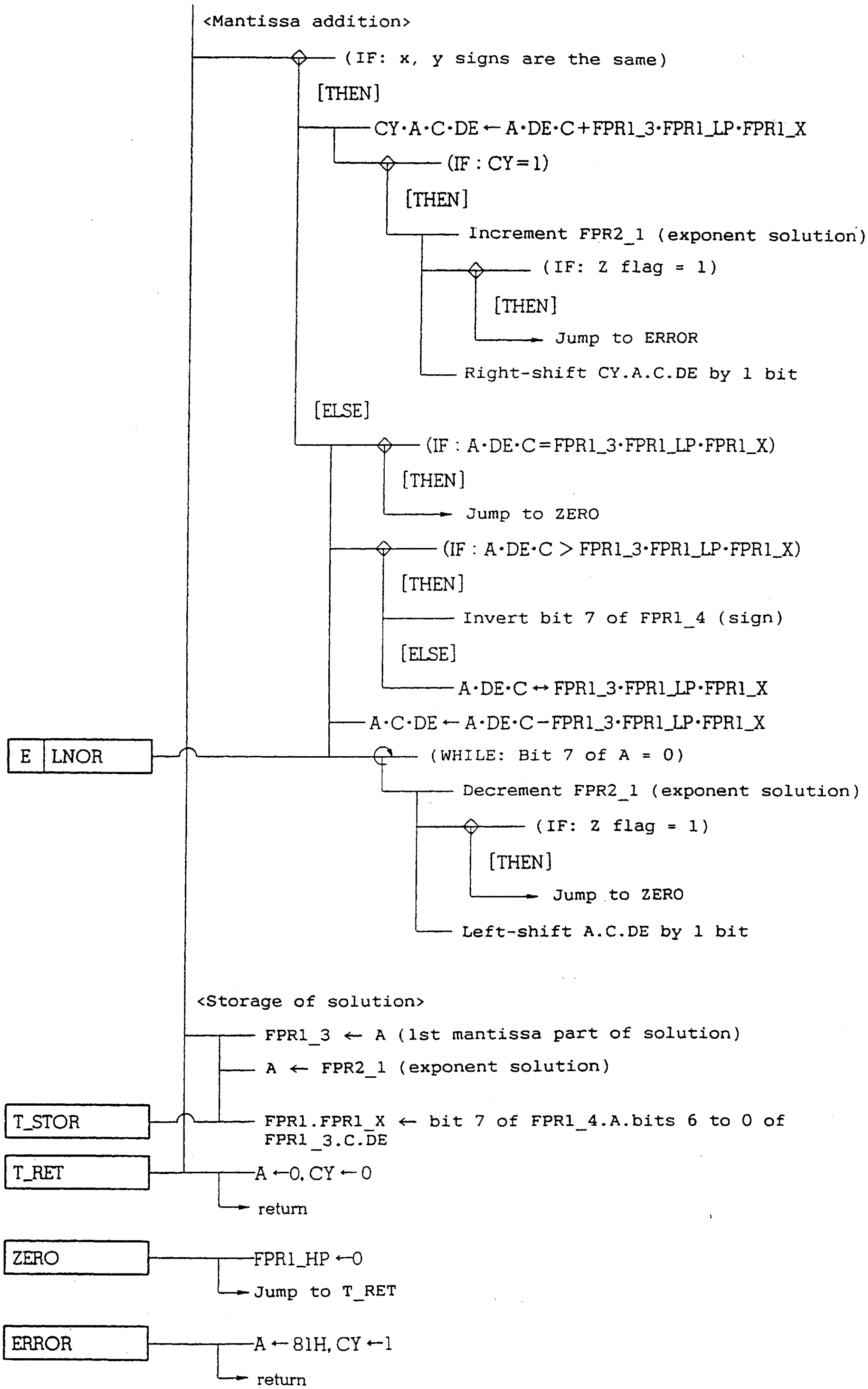
- **E LADD**
  - FPRE_XP (x, y mantissa extension) ← 0
- **E LADDX**
  - A ← y exponent
  - ◇ (IF : A=0)
    - [THEN]
      - → Jump to T_RET
  - D ← A (y exponent)
  - A ← x exponent, C ← A (x exponent)

<Exponent processing>

- A ← A (x exponent) - D (y exponent)
- ◇ (IF: Borrow not generated)
  - [THEN]
    - FPR2_X ← A (exponent difference)
    - FPR2_1 ← C (exponent solution)
    - A.DE.C ← (y mantissa) x 100H + (y mantissa extension)
  - [ELSE]
    - FPR1 · FPR1_X(x) ↔ FPR2 · FPR2_X(y)
    - FPR2_X ← 2's complement of A (exponent difference)
    - FPR2_1 ← D (exponent solution)
    - A.DE.C ← (x mantissa) x 100H + (x mantissa extension)
    - ◇ (IF: x exponent = 0)
      - [THEN]
        - → Jump to T_RET
- ◇ (IF: FPR2_X (exponent difference) ≥ 32)
  - [THEN]
    - → Jump to T_RET
- FPR1_3 ← FPR1_3 or 80H (MSB)
- A ← A or 80H (MSB)

<Mantissa processing>

- ◇ (IF: FPR2_X (mantissa difference) ≠ 0)
  - [THEN]
    - Right-shift A.DE.C by FPR2_X bits


<Mantissa addition>
(IF: x, y signs are the same)
[THEN]
CY·A·C·DE ← A·DE·C+FPR1_3·FPR1_LP·FPR1_X
(IF : CY=1)
[THEN]
Increment FPR2_1 (exponent solution)
(IF: Z flag = 1)
[THEN]
Jump to ERROR
Right-shift CY.A.C.DE by 1 bit
[ELSE]
(IF : A·DE·C=FPR1_3·FPR1_LP·FPR1_X)
[THEN]
Jump to ZERO
(IF : A·DE·C > FPR1_3·FPR1_LP·FPR1_X)
[THEN]
Invert bit 7 of FPR1_4 (sign)
[ELSE]
A·DE·C ↔ FPR1_3·FPR1_LP·FPR1_X
A·C·DE ← A·DE·C−FPR1_3·FPR1_LP·FPR1_X
E LNOR
(WHILE: Bit 7 of A = 0)
Decrement FPR2_1 (exponent solution)
(IF: Z flag = 1)
[THEN]
Jump to ZERO
Left-shift A.C.DE by 1 bit
<Storage of solution>
FPR1_3 ← A (1st mantissa part of solution)
A ← FPR2_1 (exponent solution)
T_STOR
FPR1.FPR1_X ← bit 7 of FPR1_4.A.bits 6 to 0 of FPR1_3.C.DE
T_RET
A ← 0, CY ← 0
return
ZERO
FPR1_HP ← 0
Jump to T_RET
ERROR
A ← 81H, CY ← 1
return

Remarks 1: Label [E |] indicates a global name.

2: Labels [T_STOR], [T_RET], [ZERO] and [ERROR] are also referenced by the LMLT and LDIV functions.

3: Label [E | LADDX] is an internal global name for execution of addition using mantissa extensions by mathematical functions, etc.

4: Label [E | LNOR] is an internal global name for execution of normalization from the digit-drop state by a type conversion function.

5: CY.A.C.DE are represented as 33-bit type variables with MSB = CY.
Other combinations in this processing diagram also have the same meaning.
The representations used in the processing diagram are also used with other functions.

3.2 FLOATING POINT SUBTRACTION OPERATION (LSUB)

(1) Processing

With the value of FPR1 designated as x and the value of FPR2 designated as y, returns x - y in FPR1.

(2) Object module files subject to linkage

DFLT, LFLT1

(3) Required stack size

2 (2-byte return address from LSUB only)

(4) Registers used

AX, B, DE

(5) Work areas used

FPR1, FPR2, FPR1_X, FPR2_X

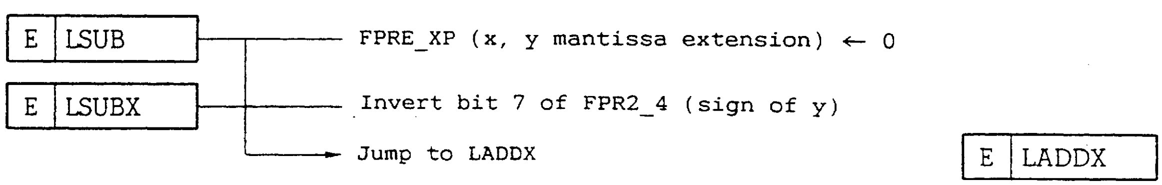
(6) Processing time (internal system clock = 8.38 MHz)

Average: 155 us
Maximum: 335 us (0.5 to -0.50000006)

(7) Processing procedure

The sign of y is inverted, and the processing jumps to LADD.

(8) Processing diagram



Remarks : Label [E | LSUBX] is an internal global name for execution of subtraction using mantissa extensions by mathematical functions, etc.

3.3 FLOATING POINT MULTIPLICATION OPERATION (LMLT)

(1) Processing

With the value of FPR1 designated as x and the value of FPR2 designated as y, returns x × y in FPR1.

(2) Object module files subject to linkage

DFLT, LFLT1

(3) Required stack size

2 (2-byte return address from LMLT only)

(4) Registers used

AX, B, DE

(5) Work areas used

FPR1, FPR2, FPR1_X, FPR2_X

(6) Processing time (internal system clock = 8.38 MHz)

Average: 134 us
Maximum: 138 us (2 x 2)

(7) Processing procedure

(a) If x or y is 0, 0 is returned.
(b) The exponents are added to give the exponent solution.
(c) The signs are XORed, and the result is taken as the sign.
(d) The following method is used to perform mantissa multiplication.
MSB (1) and the mantissa extension (8 bits) respectively are added to the x and y mantissas, and these are regarded as doubleword type variables d and s.

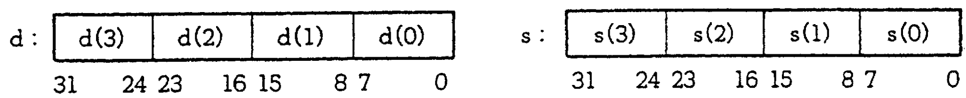
d:

1	x mantissa	FPR1_X
31	30 … 8	7 … 0

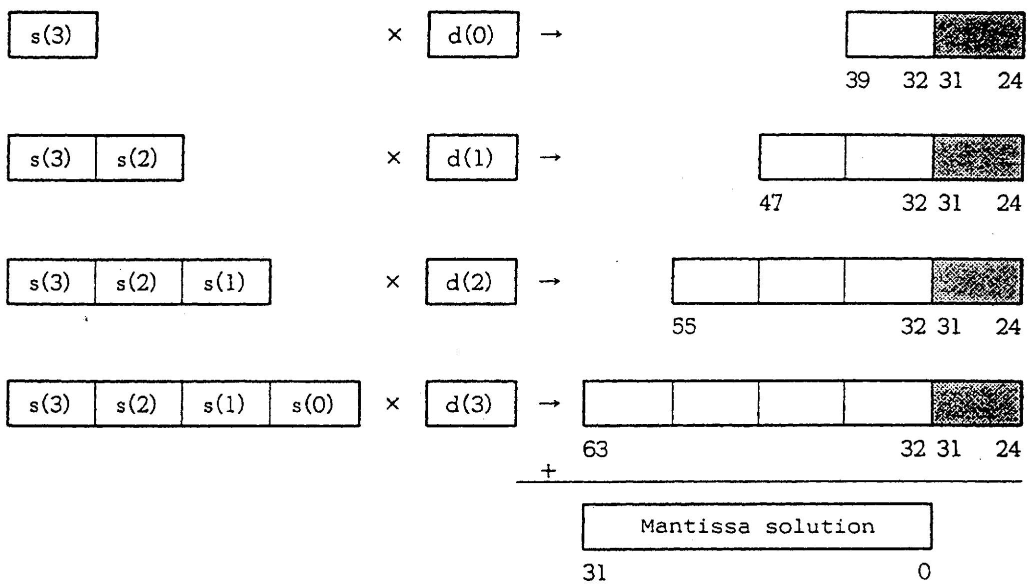
s:

1	y mantissa	FPR2_X
31	30 … 8	7 … 0

Also, as shown in the processing diagram, d and s are regarded as 4-element BYTE type arrays.

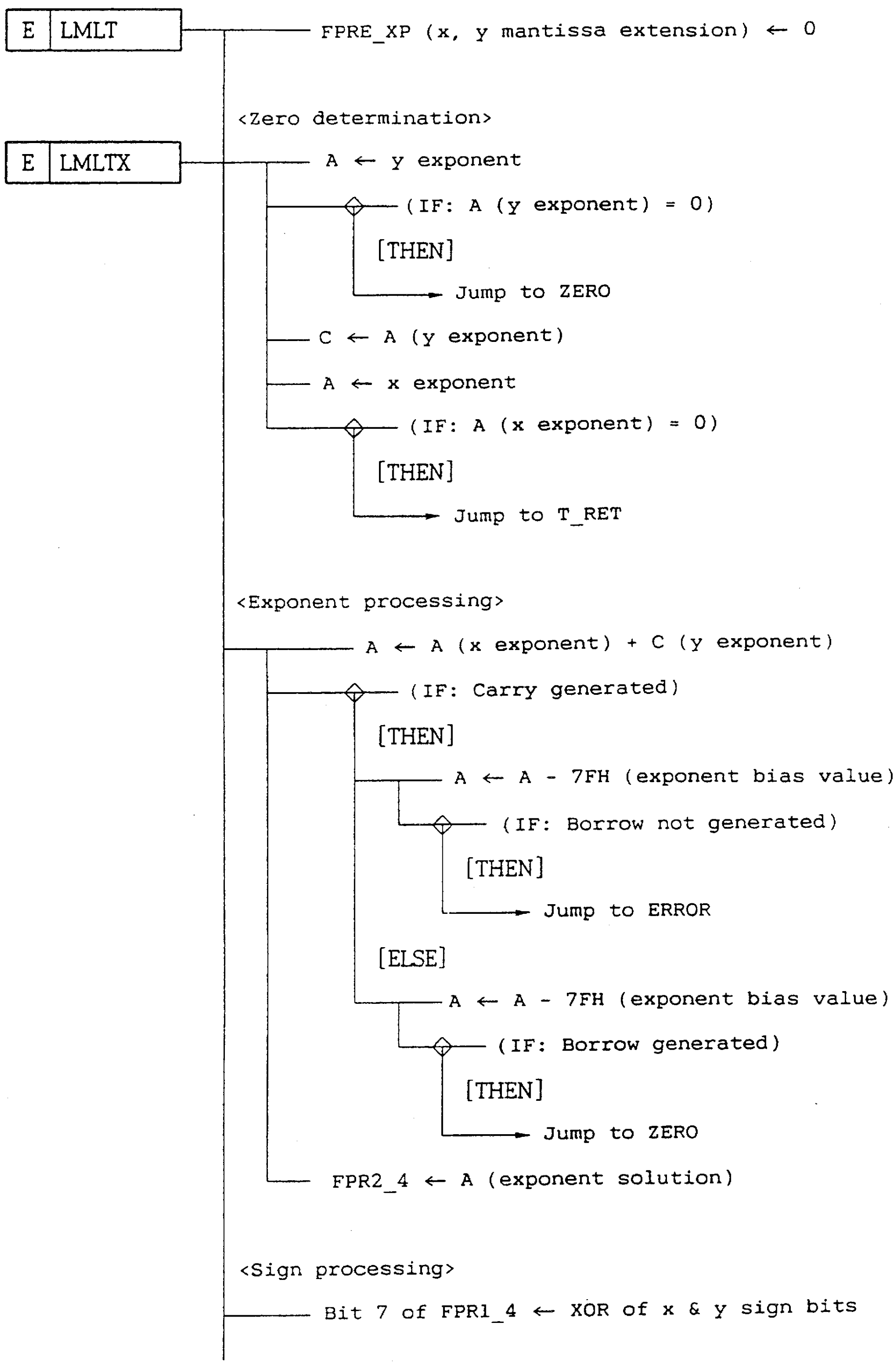


The procedure shown in the figure below is used to calculate the high-order 32 bits of the multiplication result. (The shaded area is truncated.)



(e) Normalization is performed on the exponent solution and the mantissa solution found in (d), and the result is stored in FPR1 together with the sign bit.

(8) Processing diagram



```
<Mantissa multiplication>
 ├── Bit 7 of FPR1_3 ← 1 (MSB)
 │
 ├── Bit 7 of FPR2_3 ← 1 (MSB)
 │
 ├── E ← high(d(0)×s(3))
 ├── DE ← d(1)×s(3)+high(d(1)×s(2))+E
 ├── C・DE ← d(2)×s(3)×100H+d(2)×s(2)+high(d(2)×s(1))+DE
 └── A・C・DE ← d(3)×s(3)×10000H+d(3)×s(2)×100H+d(3)×s(1)
                 +high(d(3)×s(0))+C・DE

<Normalization>
 ◇── (IF: Bit 7 of A = 1)
 [THEN]
   ├── Increment FPR2_4 (exponent solution)
   └─◇── (IF: Z flag = 1)
       [THEN]
       └──→ Jump to ERROR
 [ELSE]
   ├─◇── (IF: FPR2_4 (exponent solution) = 0)
   │   [THEN]
   │   └──→ Jump to ZERO
   └── Left-shift A.C.DE by 1 bit

<Storage of solution>
 ├── FPR1_3 ← A (1st mantissa part of solution)
 ├── A ← FPR2_4 (exponent solution)
 └──→ Jump to T_STOR                                [T_STOR]
```

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

2: Label [E | LMLTX] is an internal global name for execution of multiplication using mantissa extensions by mathematical functions, etc.

3.4 FLOATING POINT DIVISION OPERATION (LDIV)

(1) Processing

With the value of FPR1 designated as x and the value of FPR2 designated as y, returns x ÷ y in FPR1.

(2) Object module files subject to linkage

DFLT, LFLT1

(3) Required stack size

2 (2-byte return address from LDIV only)

(4) Registers used

AX, BC, DE, HL

(5) Work areas used

FPR1, FPR2, FPR1_X, FPR2_X

(6) Processing time (internal system clock = 8.38 MHz)

Average: 516 us
Maximum: 620 us (1.9999999/1)

(7) Processing procedure

(a) If y is 0, the operation terminates abnormally; if x is 0, 0 is returned.
(b) The x exponent is subtracted from the y exponent to give the exponent solution.
(c) The signs are XORed, and the result is taken as the sign.
(d) The following method is used to perform mantissa division.

MSB (1) is added to the x and y mantissas, and these are regarded as 25 and 24-bit variables d and s.

d:

0	1	x mantissa
24	23	22 0

s:

1	y mantissa
23	22 0

25-bit calculation of the quotient is performed using the usual manual calculation procedure.

(i) d and s are compared, and a quotient of 1 is obtained if $d \geqq s$, and a quotient of 0 if $d < s$.

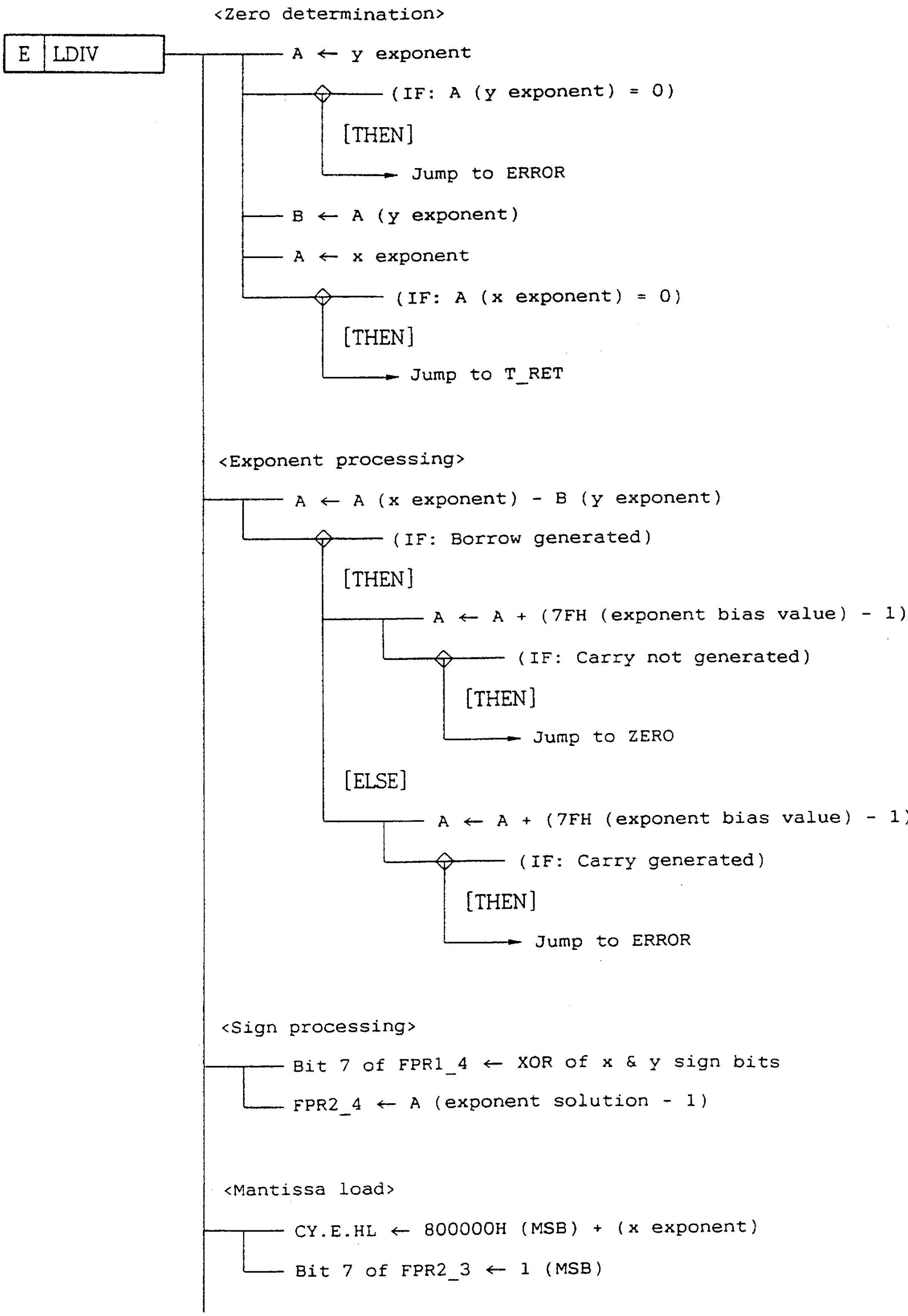
(ii) s x (quotient) is subtracted from d

(iii) d is left-shifted by 1 bit.

(iv) Steps (i) to (iii) are repeated 25 times (however, (iii) is not executed the 25th time).

(e) Normalization is performed on the exponent solution and the mantissa solution found in (d), and the result is stored in FPR1 together with the sign bit.

(8) Processing diagram



T_DIV1

<Mantissa division>

- Jump to T_DIV1
  - (LOOP: 25 times)
    - Left-shift CY.E.HL(d) by 1 bit
    - (IF : CY=0)
      [THEN]
      - (IF : E·HL(d)<FPR2_3·FPR2_LP(s))
        [THEN]
        - CY ← 0
        [ELSE]
        - CY ← 1
    - (IF : CY=1)
      [THEN]
      - E·HL ← E·HL(d) − FPR2_3·FPR2_LP(s)
      - CY ← 1
    - Left-rotate X.C.D.CY by 1 bit

<Normalization>

- (IF : CY=1)
  [THEN]
  - Increment FPR2_4 (exponent solution - 1)
  - (IF: Z flag = 1)
    [THEN]
    - → Jump to ERROR
  - Right-shift CY.X.C.D by 1 bit

<Storage of solution>

- FPR1_3 ← X (1st mantissa part of solution)
- E ← 0 (mantissa extension)
- A ← FPR2_4 (exponent solution)
- → Jump to T_STOR

T_STOR

CHAPTER 4. MATHEMATICAL FUNCTIONS

The following mathematical functions are used.

(1) sin function (LSIN)

Finds the sine of the value of FPR1.

(2) cos function (LCOS)

Finds the cosine of the value of FPR1.

(3) tan function (LTAN)

Finds the tangent of the value of FPR1.

(4) Natural logarithm function (LLOG)

Finds the natural logarithm of the value of FPR1.

(5) Common logarithm function (LLOG10).

Finds the common logarithm of the value of FPR1.

(6) Exponent function (base = e) (LEXP)

Finds the exponent solution where the value of FPR1 is the exponent value and the base is e.

(7) Exponent function (base = 10) (LEXP10)

Finds the exponent solution where the value of FPR1 is the exponent value and the base is 10.

(8) Power function (LPOW)

Finds the power relation between the value of FPR1 and the value of FPR2.

(9) Square root function (LSQRT)

Finds the square root of the value of FPR1.

(10)arcsin function (LASIN)

Finds the arcsine of the value of FPR1.

(11)arccos function (LACOS)

Finds the arccosine of the value of FPR1.

(12)arctan function (LATAN)

Finds the arctangent of the value of FPR1.

(13)sinh function (LHSIN)

Finds the hyperbolic sine function solution for the value of FPR1.

(14)cosh function (LHCOS)

Finds the hyperbolic cosine function solution for the value of FPR1.

(15)tanh function (LHTAN)

Finds the hyperbolic tangent function solution for the value of FPR1.

(16)Absolute value function (LABS)

Gives the absolute value of FPR1.

(17) Reciprocal function (LRCPN)

Finds the reciprocal of the value of FPR1.

The operation result is stored in FPR1.

4.1 COMMON SUBROUTINES (LPLY, LPLY2)

As stated earlier, all the mathematical functions except for the square root function use either a Taylor approximation expression or a best approximation expression for their calculations. These approximation expressions are given as high-order polynomials, and have a common pattern in terms of Taylor expansion and best approximation characteristics.

Consequently, two types of polynomial calculation functions (LPLY and LPLY2) are provided for use as common subroutines.

(1) Processing

The polynomial calculation result, including the mantissa extension, is returned in FPR1.FPR1_X.

(2) Object module files subject to linkage

DFLT, LFLT1, LFLT2, LLD

(3) Required stack size

4 (including 2-byte return address from LPLY and LPLY2)

(4) Registers used

AX, BC, DE, HL

(5) Work areas used

FPR1, FPR2, FPR3, FPR4, FPR1_X, FPR2_X, FPR3_X, FPR4_X (the contents of FPR4 and FPR4_X are retained)

(6) Algorithms

The following two types of polynomial calculations can be used.

① $x+k_1xy+k_1k_2xy^2+k_1k_2k_3xy^3+\cdots+k_1k_2\cdots k_nxy^n$

② $z+k_1xy+k_1k_2xy^2+k_1k_2k_3xy^3+\cdots+k_1k_2\cdots k_nxy^n$

Remarks : x : 1st term of polynomial

y : Multiplication constant corresponding to order transformation of each term (x^2 etc.)

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

Polynomial	Conditions of Use	Calculation Function
①	When each term has common aliquot x	LPLY
②	When 1st term is constant(z)	LPLY2

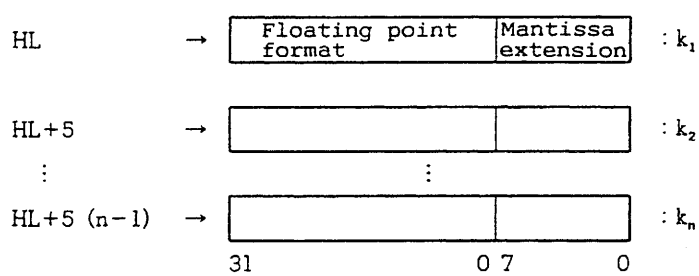
To minimize the degree of error, floating point numbers with a mantissa extension are used for x, y, z, k_1, k_2, $\ldots k_n$.

NOTE : Due to the characteristics of the Taylor expansion, the following condition must be satisfied: |1st term|>|2nd term|>...>|n'th term|

(7) Input conditions

Polynomial	X	Y	Z	n	Start address of coefficient series ($k_1, k_2, k_3, \ldots, k_n$)
①	FPR1. FPR1_X	FPR4. FPR4_X	-	B	HL
②	FPR3. FPR3_X	FPR4. FPR4_X	FPR1. FPR1_X	B	HL

The storage format for the coefficient series is shown below.



(8) Processing procedure

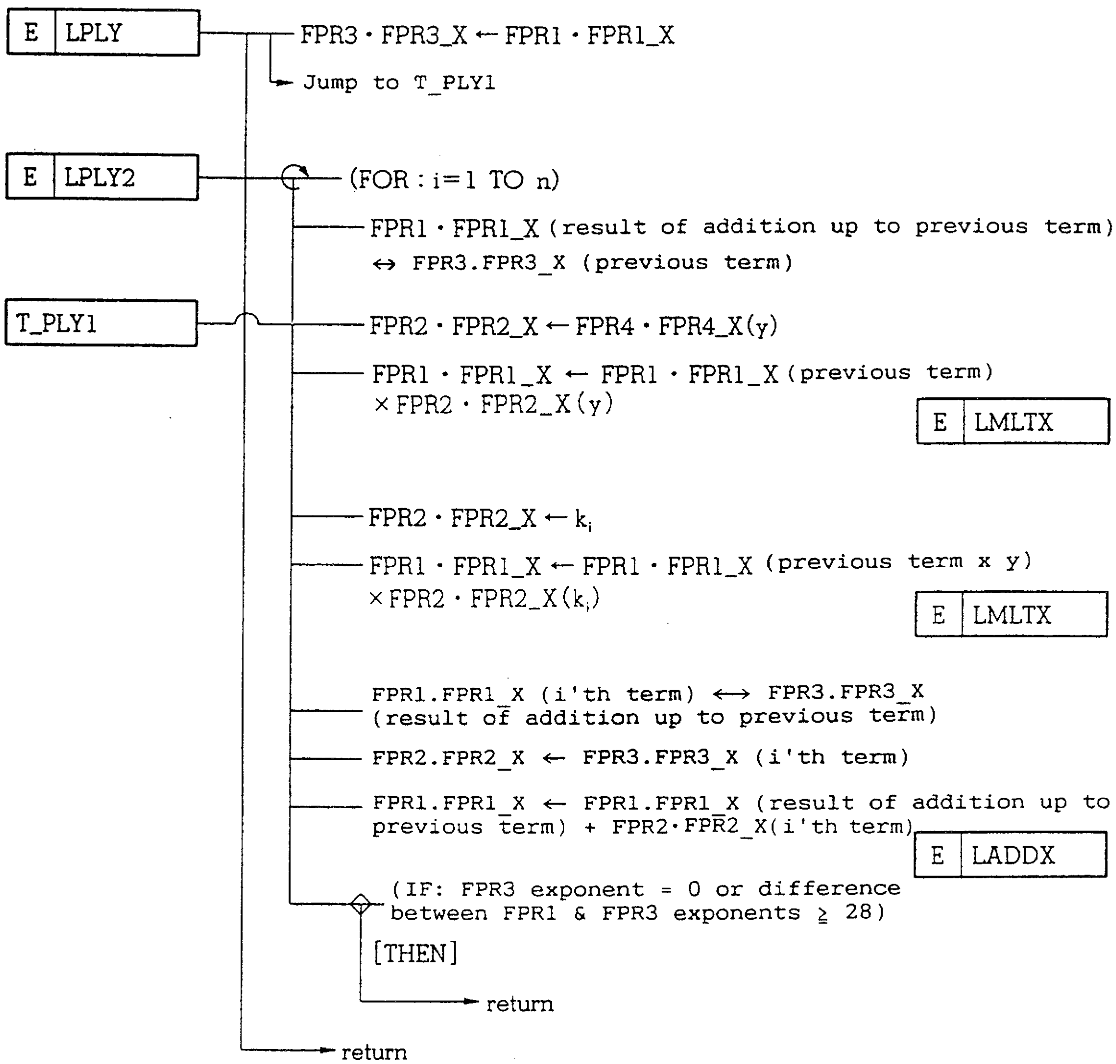
(a) The i'th term is found by multiplication of term (i-1) by y and k_i.

(b) The value of the i'th term is added to the sum of terms up to term (i-1).

(c) If the i'th term is significantly smaller than the sum of terms up to the i'th term, the calculation is ended.

(d) Steps (a) to (c) are repeated up to the n'th term.

(9) Processing diagram



4.2 sin FUNCTION (LSIN)

(1) Processing

With the value of FPR1 designated as x, returns sin(x) in FPR1.

- Unit: Radians

(2) Object module files subject to linkage

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

(3) Required stack size

6 (including 2-byte return address from LSIN)

(4) Registers used

AX, BC, DE, HL

(5) Work areas used

FPR1, FPR2, FPR3, FPR4, FPR1_X, FPR2_X, FPR3_X, FPR4_X

(6) Processing time (internal system clock = 8.38 MHz)

Average: 2343 us
Maximum: 8005 us (sin(6.8056469e + 38))

(7) Algorithm

The following Taylor approximation expression is used to find sin(x).

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

(8) Processing procedure

(a) The sign of x is stored, and the absolute value of x is taken.

(b) x is scaled to the range $0 \leqq x < \pi/2$.
If the value after scaling is designated as x', the following expressions apply:

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

Thus, if the quotient obtained by dividing x by $\pi/2$, is designated as n, and the remainder as x', then the following substitutions can be made for sin(x).

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 stored sign is assigned to x".

(d) sin(x") is found by the Taylor approximation expression, to give the solution.

(9) Floating point constant data

(a) Constant data $2/\pi$ and $\pi/2$ with a mantissa extension are used.

(b) Constant data -1/3!, -3!/5!, -5!/7!, -7!/9! and -9!/11! with a mantissa extension are used for the approximation polynomial coefficient series as a 5-element array, K.

(10) Processing diagram

E	LSIN

<Conversion of x to (sign)(x' + nπ/2)>

- FPR1 · FPR1_X ← x'
- FPR4_3 ← n, bit 7 of FPR4_4 ← sign [E | LMOD90]

<Adjustment of x by n>

- (IF: Bit 0 of FPR4_3 (n) = 1)
  - [THEN]
    - FPR1 · FPR1_X ← − x'
    - FPR1 · FPR1_X(x") ← FPR1 · FPR1_X(−x') + π/2 [E | LADDX]
- FPR1 (x") sign bit ← FPR4_3 (n) bit 1 XOR FPR4_4 bit 7 (sign)

<Approximation expression calculation>

- FPR4 · FPR4_X ← FPR1 · FPR1_X(x")
- FPR2 · FPR2_X ← FPR1 · FPR1_X(x")
- FPR1 · FPR1_X ← FPR1 · FPR1_X(x") × FPR2 · FPR2_X(x") [E | LMLTX]
- FPR1 · FPR1_X(x"²) ↔ FPR4 · FPR4_X(x")
- HL ← Start address of array K
- B ← Number of elements of array K, 5
- FPR1.FPR1_X ← Approximation polynomial solution (sin(x")) [E· | LPLY]

- A ← 0, CY ← 0
- → return

(Subroutine to find quotient and remainder from division by $\pi/2$)

E | LMOD90

<Initialization>

- FPR1_X (x mantissa extension) ← 0
- FPR4_3(n) ← 0
- FPR4_4 bit 7 ← x sign bit
- FPR1 · FPR1_X ← |x|

<Division loop>

- (LOOP)
  - (IF : FPR1 · FPR1_X(|x|) < π/2)
    - [THEN]
      - → return
  - FPR3 · FPR3_X ← FPR1 · FPR1_X(|x|)
  - FPR1 · FPR1_X ← FPR1 · FPR1_X(|x|) × 2/π — **E | LMLTX**
  - Number of valid digits of FPR1.FPR1_X (|x| x 2/π) reduced to 14 digits

  <Conversion of quotient to integer>

  - DE ← FPR1.FPR1_X (|x| x 2/π) converted to integer type — **E | LTOF**
  - (IF: CY (conversion overflow) = 0)
    - [THEN]
      - (IF: Z (|x| x 2/π is integer)=0)
        - [THEN]
          - FPR1.FPR1_X ← DE (int(|x| x 2/π)) converted to floating point number — **E | FTOL**
      - FPR4_3 (n) ← FPR4_3 + E(int(|x| x 2/π) lower)

  <Remainder calculation>

  - FPR1 · FPR1_X ← FPR1 · FPR1_X(int(|x| × 2/π)) × π/2 — **E | LMLTX**
  - FPR1.FPR1_X sign inversion
  - FPR2 · FPR2_X ← FPR3 · FPR3_X(|x|)
  - FPR1 · FPR1_X ← FPR1 · FPR1_X(−int(|x| × 2/π) × π/2) + FPR2 · FPR2_X(|x|) — **E | LADDX**

Remarks: int(x) is the integral part of x.

4.3 cos FUNCTION (LCOS)

(1) Processing

With the value of FPR1 designated as x, returns cos(x) in FPR1.

- Unit: Radians

(2) Object module files subject to linkage

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

(3) Required stack size

6 (including 2-byte return address from LCOS)

(4) Registers used

AX, BC, DE, HL

(5) Work areas used

FPR1, FPR2, FPR3, FPR4, FPR1_X, FPR2_X, FPR3_X, FPR4_X

(6) Processing time (internal system clock = 8.38 MHz)

Average: 2842 us
Maximum: 7804 us (cos(6.8056469e + 38))

(7) Algorithm

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

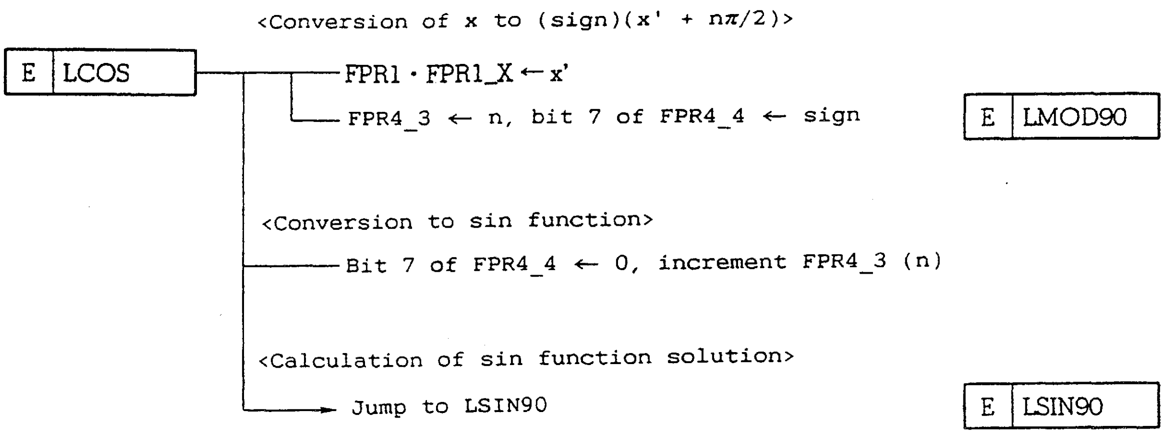
$$\boxed{\cos(x) = \sin(|x| + \pi/2)}$$

(8) Processing procedure

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

(b) $\sin(x'+(n+1)\pi/2)$ is found using the LSIN90 subroutine, to give the solution.

(9) Processing diagram



4.4 tan FUNCTION (LTAN)

(1) Processing

With the value of FPR1 designated as x, returns tan(x) in FPR1.

- Unit: Radians

(2) Object module files subject to linkage

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

(3) Required stack size

10 (including 2-byte return address from LTAN)

(4) Registers used

AX, BC, DE, HL

(5) Work areas used

FPR1, FPR2, FPR3, FPR4, FPR5, FPR1_X,FPR2_X, FPR3_X, FPR4_X, FPR5_X

(6) Processing time (internal system clock = 8.38 MHz)

Average: 5428 us
Maximum: 11040 us (tan(6.8056469e + 38))

(7) Algorithm

The following expression is used to find tan(x).

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

(8) Processing procedure

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

(b) (s)sin(x'+nπ/2) is found using the LSIN90 subroutine.

(c) cos(x'+nπ/2) is found using the LCOS90 subroutine.

(d) (s)sin(x'+nπ/2) ÷ cos(x'+nπ/2) is found, to give the solution.

(9) Processing diagram

E | LTAN

<Conversion of x to (sign)(x' + nπ/2)>

- FPR1・FPR1_X ← x'
- FPR4_3 ← n, bit 7 of FPR4_4 ← sign (E | LMOD90)
- Save FPR4_3 (n) and FPR4_4 bit 7 (sign) to stack
- FPR5・FPR5_X ← FPR1・FPR1_X(x')

<Calculation of cos function solution>

- FPR1 ← cos(FPR1・FPR1_X(x') + FPR4_3(n) π/2) (E | LCOS90)

<Calculation of sin function solution>

- FPR1(cos(|x|)) ↔ FPR5(x')
- FPR1_X ← FPR5_X (x' mantissa extension)
- FPR4_4 bit 7 ← sign restored from stack, FPR4_3 ← n restored from stack
- FPR1 ← FPR4_4 bit 7 (sign)・sin(FPR1・FPR1_X(x') + FPR4_3(n) π/2) (E | LSIN90)

<Calculation of tan function solution>

- FPR2 ← FPR5(cos(|x|))
- Jump to LDIV (E | LDIV)

4.5 NATURAL LOGARITHM FUNCTION (LLOG)

(1) Processing

With the value of FPR1 designated as x, returns log(x) in FPR1.

(2) Object module files subject to linkage

DFLT, LFLT1, LFLT2, LLD, LLOG, FTOL

(3) Required stack size

6 (including 2-byte return address from LLOG)

(4) Registers used

AX, BC, DE, HL

(5) Work areas used

FPR1, FPR2, FPR3, FPR4, FPR1_X, FPR2_X, FPR3_X, FPR4_X

(6) Processing time (internal system clock = 8.38 MHz)

Average: 2857 us
Maximum: 3659 us (log(1.4397301e - 25))

(7) Algorithm

log(x) can be found by performing a Taylor expansion in the following expression.

With $x' = \frac{x-1}{x+1}$

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

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

This expression is theoretically satisfied for $0 < x < \infty$, but except in the vicinity of x = 1 the convergence is to slow to be of practical use.

For this reason, the range of x' used in the approximation expression is made approximately -0.17 to 0.17 by means of the following method.

- Let the exponent of x be xe, and the mantissa be xf.
- If xf < $\sqrt{2}$, xe' = xe and xf' = xf
- If xf >= $\sqrt{2}$, xe' = xe + 1 and xf' = xf/2
- Function conversion is performed in the following expression
  log(x) = xe' x log2 + log(xf')

(8) Processing procedure

(a) If x $\leq$ 0, processing terminates abnormally.
(b) If xf < $\sqrt{2}$, xe' = xe and xf' = xf.
(c) If xf $\geq$ $\sqrt{2}$, xe' = xe + 1 and xf' = xf/2.
(d) xe' x log2 is found.
(e) x' is found from (xf'-1)/(xf'+1).
(f) The 1st term (2x') of the approximation polynomial is replaced with xe' x log2 + 2x', and the approximation polynomial is calculated.

(9) Floating point constant data

(a) Constant data log2 and 1 with a mantissa extension are used.
(b) Constant data 1/3, 3/5, 5/7, and 7/9 with a mantissa extension are used for the approximation polynomial coefficient series as a 4-element array, K.

(10) Processing diagram

E | LLOG

- CY.A ← Bits 31 to 23 of FPR1

<Exception processing>

- ◇ (IF: A (exponent) = 0 or CY (sign bit) = 1)
  - [THEN]
    - A ← 81H, CY ← 1
    - → return

<Exponent logarithm calculation>

- X ← A (exponent), A ← 0
- DE ← AX - 7FH (exponent bias value)
- FPR3 ← FPR1(x)
- A ← FPR1_3 (1st mantissa part) or 80H (MSB)
- ◇ (IF: A ≥ 0B5H (1st mantissa part of $\sqrt{2}$)
  - [THEN]
    - FPR3 exponent ← -1 + 7FH
    - Increment DE (xe)
  - [ELSE]
    - FPR3 exponent ← 0 + 7FH
- FPR1.FPR1_X ← DE (xe') converted to floating point format — E | FTOL
- FPR1 · FPR1_X ← FPR1 · FPR1_X(xe') × log2 — E | LMLTX
- FPR4 · FPR4_X ← FPR1 · FPR1_X(xe'× log2)

<x'=(xf'-1)/(xf'+1) calculation>

- FPR1 ← FPR3(xf')
- FPR1 ← FPR1(xf') + 1 — E | LADD
- FPR1(xf'+ 1) ↔ FPR3(xf')
- FPR1 ← FPR1(xf') − 1 — E | LSUB
- FPR2 ← FPR3(xf'+ 1)
- FPR1 · FPR1_X ← FPR1(xf'− 1)/FPR2(xf'+ 1) — E | LDIV

<Approximation expression calculation>

- FPR3 · FPR3_X ← FPR1 · FPR1_X(x')
  - (IF : FPR1(x') ≠ 0)
    - [THEN]
      - Increment FPR3 exponent
  - FPR2 · FPR2_X ← FPR1 · FPR1_X(x')
  - FPR1 · FPR1_X ← FPR1 · FPR1_X(x') × FPR2 · FPR2_X(x')
    - E | LMLTX
  - FPR1 · FPR1_X(x'^2) ↔ FPR4 · FPR4_X(xe' × log2)
  - FPR2 · FPR2_X ← FPR3 · FPR3_X(2x')
  - FPR1 · FPR1_X ← FPR1 · FPR1_X(xe' × log2) + FPR2 · FPR2_X(2x')
    - E | LADDX
  - HL ← Start address of array K
  - B ← Number of elements of array K, 4
  - FPR1.FPR1_X ← Approximation polynomial solution (xe' x log2 + log(xf'))
    - E | LPLY2
- A ← 0, CY ← 0
  - return

4.6 COMMON LOGARITHM FUNCTION (LLOG10)

(1) Processing

With the value of FPR1 designated as x, returns $\log_{10}(x)$ in FPR1.

(2) Object module files subject to linkage

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

(3) Required stack size

8 (including 2-byte return address from LLOG10)

(4) Registers used

AX, BC, DE, HL

(5) Work areas used

FPR1, FPR2, FPR3, FPR4, FPR1_X, FPR2_X, FPR3_X, FPR4_X

(6) Processing time (internal system clock = 8.38 MHz)

Average: 3014 us
Maximum: 3801 us ($\log_{10}$(2.4001264e - 18))

(7) Algorithm

$\log_{10}(x)$ is found by means of the following expression.

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

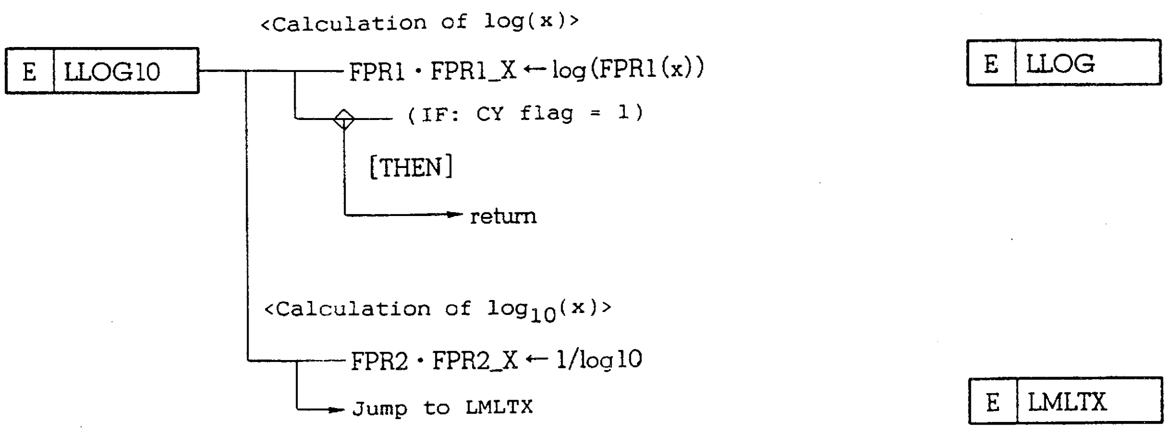
(8) Processing procedure

(a) log(x) is found by means of the LLOG function.
(b) If the LLOG function terminates abnormally, processing terminates abnormally at this point.
(c) log(x)/log10 is found, giving the solution.

(9) Floating point constant data

Constant data 1/log10 with a mantissa extension is used.

(10) Processing diagram



4.7 EXPONENT FUNCTION (BASE = e) (LEXP)

(1) Processing

With the value of FPR1 designated as x, returns e^x in FPR1.

(2) Object module files subject to linkage

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

(3) Required stack size

6 (including 2-byte return address from LEXP)

(4) Registers used

AX, BC, DE, HL

(5) Work areas used

FPR1, FPR2, FPR3, FPR4, FPR1_X, FPR2_X, FPR3_X, FPR4_X, FPR5_X

(6) Processing time (internal system clock = 8.38 MHz)

Average: 3591 us
Maximum: 4084 us ($e^{-0.82129019}$)

(7) Algorithm

The method of first finding the exponent part of the solution by converting the exponent base to 2 is used.

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

Remarks 1: floor(x) indicates the low-order integer of x (floor(x) $\leqq$ x < floor(x) + 1).

2: dec(x) indicates the decimal part of x (dec(x) = x - floor(x); 0 $\leqq$ dec(x) < 1).

Since 1 $\leqq$ $2^{dec(x/log2)}$<2, floor(x/log2) is the exponent part of the solution.
The mantissa is found by using the following Taylor approximation expression.

If dec(x/log2) < 1/2, $x' = dec(x/\log 2)$

If dec(x/log2) $\geqq$ 1/2, $x' = dec(x/\log 2) - 1$

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

Remarks 1: Since the mantissa obtained is the same even if $2^{x'}$ is multiplied by 1/2, the most useful range of -1/2 $\leqq$ x' < 1/2 is used in the approximation expression.

2: When dec(x/log2) $\geqq$ 1/2, mathematically x' < 0, but x' = 0 may be obtained due to the calculation error. In this case, $2^{x'}$ = 1, and a totally different mantissa is obtained from that expected.
To overcome this problem, the following expression is used in the calculation of x'.

If dec(x/log2) $\geqq$1/2, $x' = dec(x/\log 2) - (1 + 2^{-30})$

(8) Processing procedure

(a) x/log2 is found.
(b) In case of overflow,
. If x < 0, the operation is ended with 0 as the solution.

. If x > 0, the operation terminates abnormally.

(c) floor(x/log2) is found.

(d) If floor(x/log2) < -126, the calculation is ended with 0 as the solution.
If floor(x/log2) > 128, the operation terminates abnormally.

(e) dec(x/log2) is found and taken as x'.

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

(g) Calculation of the $2^{x'}$ Taylor approximation expression is performed.

(h) The exponent value floor(x/log2) is incorporated in the result of the approximation expression calculation to give the solution.

(9) Floating point constant data

(a) Constant data 1/log2 and 1 with a mantissa extension are used.

(b) Constant data log2, log2/2, log2/3 ...,log 2/7 with a mantissa extension are used for the approximation polynomial coefficient series as a 7-element array, K.

(10) Processing diagram

E LEXP / E LEXPX

- FPR1_X (mantissa extension) ← 0
- FPR3_X bit 7 ← sign of x

<Exponent base conversion (e → 2)>

- FPR1 · FPR1_X ← FPR1 · FPR1_X(x) × 1/log2 — E LMLTX

<Exponent calculation>

- (IF: CY (overflow) ≠ 1)
  - [THEN]
    - DE ← trunk(FPR1(x/log2)) — E LTOF
- (IF: CY (overflow) = 1)
  - [THEN]
    - → Jump to T_FLOW
- (IF: Z (FPR1: integer) ≠ 1 or FPR1_X (mantissa extension) ≠ 0)
  - [THEN]
    - (IF: FPR1 sign bit = 1 and FPR1 exponent ≠ 0)
      - [THEN]
        - Decrement DE
- AX ← DE (floor(x/log2)) + 7FH (exponent bias)
- (IF: AX (exponent of solution) ≧ 100H)
  - [THEN]
    - → Jump to T_FLOW
- FPR5_X ← X (exponent of solution)

<Mantissa calculation>

- FPR2 · FPR2_X ← FPR1 · FPR1_X(x/log2)
- FPR1.FPR1_X ← DE (floor(/log2)) in floating point form — E FTOL
- Invert FPR1 sign bit
- FPR1 · FPR1_X ← FPR1 · FPR1_X(−floor(x/log2)) + FPR2 · FPR2_X(x/log2) — E LADDX
- (IF : FPR1(dec(x/log2)) ≧ 1/2)
  - [THEN]
    - FPR1 · FPR1_X ← FPR1 · FPR1_X(dec(x/log2)) − $(1+2^{-30})$ — E LSUBX

- FPR4 · FPR4_X ← FPR1 · FPR1_X(x')
- FPR1 · FPR1_X ← FPR1 · FPR1_X(x') × log2 — [E | LMLTX]
- HL ← Start address of 2nd element of array K
- B ← Number of elements of array K - 1
- FPR1.FPR_X ← approximation polynomial solution($2^{x'}$ - 1) — [E | LPLY]
- FPR1 · FPR1_X ← FPR1 · FPR1_X($2^{x'}$ − 1) + 1 — [E | LADDX]

<Solution combination>

- FPR1 exponent ← FPR5_X (exponent of solution)
- FPR1 sign bit ← 0

[T_EXP9]

- A ← 0, CY ← 0
- return

[T_FLOW]

- (IF: FPR3_X bit 7 (sign of x) = 1)
  - [THEN]
    - FPR1_HP ← 0
    - Jump to T_EXP9
- A ← 81H, CY ← 1
- return

Remarks 1: trunk (x) indicates rounding of the decimal part of x toward zero.

If x ≧ 0, trunk(x) ≦ x < trunk(x) + 1

If x < 0, trunk(x) - 1 < x ≦ trunk(x)

2: Label E LEXPX is an internal global name for execution of exponent calculation using a mantissa extension by other mathematical functions, etc.

4.8 EXPONENT FUNCTION (BASE = 10) (LEXP10)

(1) Processing

With the value of FPR1 designated as x, returns 10^x in FPR1.

(2) Object module files subject to linkage

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

(3) Required stack size

6 (including 2-byte return address from LEXP10)

(4) Registers used

AX, BC, DE, HL

(5) Work areas used

FPR1, FPR2, FPR3, FPR4, FPR1_X, FPR2_X, FPR3_X, FPR4_X, FPR5_X

(6) Processing time (internal system clock = 8.38 MHz)

Average: 3807 us
Maximum: 4241 us ($10^{-0.35975304}$)

(7) Algorithm

The following expression is used.

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

(8) Processing procedure

(a) x X log10 is found.

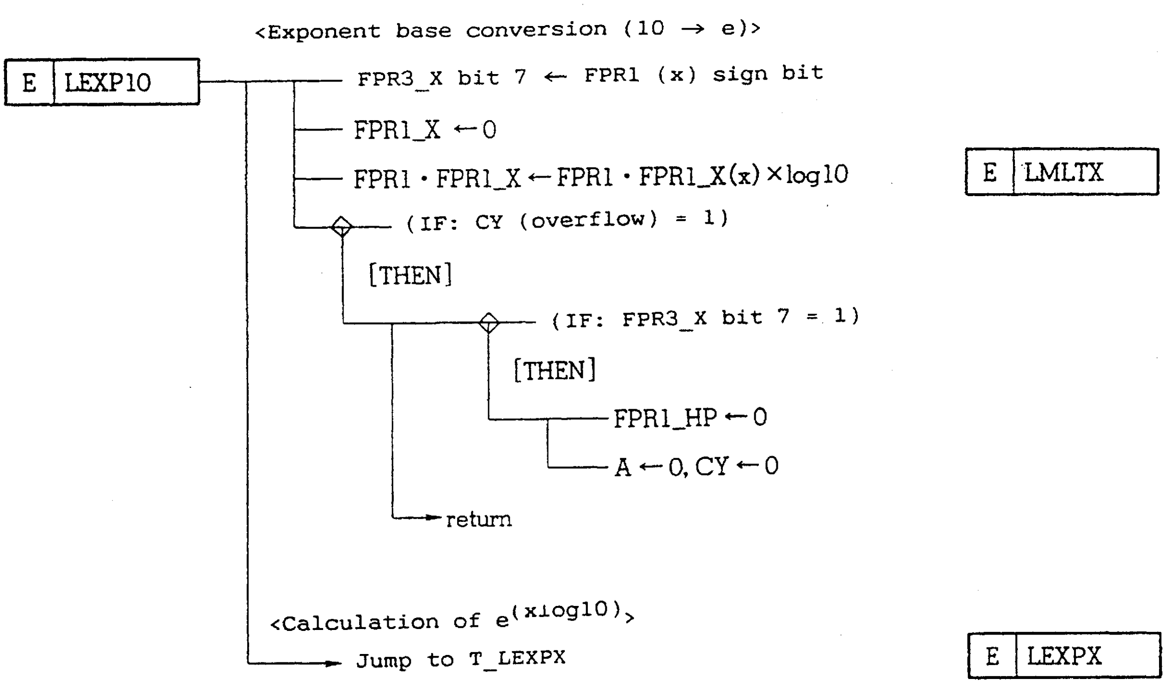
(b) If the multiplication results in overflow,
If x < 0, the operation is ended with 0 as the solution.
If x > 0, the operation terminates abnormally.

(c) The procedure jumps to the LEXP function.

(9) Floating point constant data

Constant data log10 with a mantissa extension is used.

(10) Processing diagram



4.9 POWER FUNCTION (LPOW)

(1) Processing

With the value of FPR1 designated as a and the value of FPR2 as b, returns a^b in FPR1.

(2) Object module files subject to linkage

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

(3) Required stack size

8 (including 2-byte return address from LPOW)

(4) Registers used

AX, BC, DE, HL

(5) Work areas used

FPR1, FPR2, FPR3, FPR4, FPR5, FPR1_X, FPR2_X, FPR3_X, FPR4_X, FPR5_X

(6) Processing time (internal system clock = 8.38 MHz)

Average: 6672 us
Maximum: 7835 us ($(1.50487e + 12)^{(-0.20180109)}$)

(7) Algorithm

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

a, b	a^b
$a = 0,\ b \leq 0$	Error
$a = 0,\ b > 0$	0
$a > 0$	$e^{b\log(a)}$
$a < 0,\ b = 0$	1
$a < 0$, b is a non-zero integer, b is even:	$e^{b\log(\lvert a\rvert)}$
b is odd :	$-e^{b\log(\lvert a\rvert)}$
$a < 0$, b is not an integer	Error

(8) Processing procedure

(a) If $a = 0$ and $b \leq 0$, the operation terminates abnormally.

(b) If $a = 0$ and $b > 0$, 0 is returned as the operation result.

(c) If $a < 0$ and $b = 0$, 1 is returned as the operation result.

(d) If $a < 0$ and $b \neq 0$, b integer determination is performed, and if b is an integer, even/odd number determination is performed.
If b is not an integer, the operation terminates abnormally.

(e) $e^{b\log(\lvert a\rvert)}$ is found using the LLOG and LEXP functions.

(f) If $a < 0$ and b is an odd integer, the sign of the solution found in (e) is inverted.

(9) Floating point constant data

Constant data 1 is used.

(10) Processing diagram

- E LPOW
  - C ← FPR2 (b) exponent
  - A ← FPR1 (a) exponent
  - A ↔ C
  - <a = 0 exception processing>
  - ◇ (IF: C (exponent of a) = 0)
    - [THEN]
      - ◇ (IF: A (exponent of b) = 0 or FPR2 (b) sign bit = 1)
        - [THEN]
          - → Jump to ERROR
      - → Jump to T_POW9
  - <b integer determination when a < 0>
  - FPR5_X bit 0 ← 0
  - ◇ (IF: FPR1 (a) sign bit = 1)
    - [THEN]
      - ◇ (IF: A (exponent of b) = 0)
        - [THEN]
          - FPR1 ← 1
          - → Jump to T_POW9
      - ◇ (IF: A (exponent of b) < 7FH (exponent bias))
        - [THEN]
          - → Jump to ERROR
      - C ← A
      - AX.B ← 800000H (MSB) + FPR2 (b) mantissa
      - ◇ (IF : C<(7FH+23))
        - [THEN]
          - ○ (LOOP: (7FH + 23) -C times)
            - Right-shift AX.B.CY by 1 bit
            - ◇ (IF : CY=1)
              - [THEN]
                - → Jump to ERROR
      - ◇ (IF : C≦(7FH+23))
        - [THEN]
          - FPR5_X ← B

<Calculation of $e^{b \times \log |a|}$>

- FPR5 ← FPR2(b)
- FPR1_4 bit 7 (sign bit of a) ← 0
- FPR1 · FPR1_X ← log(FPR1(|a|)) — [E | LLOG]
- FPR2 ← FPR5(b), FPR2_X ← 0
- FPR1 · FPR1_X ← FPR1 · FPR1_X(log |a|) × FPR2 · FPR2_X(b) — [E | LMLTX]
- (IF: overflow)
  - [THEN]
    - (IF: FPR5_4 bit 7 (sign bit of b) = 1)
      - [THEN]
        - FPR1_HP ← 0
        - → Jump to T_POW 9
    - → Jump to ERROR
- FPR5_1 ← FPR5_X
- FPR1 · FPR1_X ← $e^{FPR1 \cdot FPR1\_X(b \times \log |a|)}$ — [E | LEXPX]
- (IF: overflow)
  - [THEN]
    - → return

<Sign setting>

- FPR1_4 bit 7 ← FPR5_1 bit 0 (sign of solution)

[T_POW9]
- A ← 0, CY ← 0
- → return

[ERROR]
- A ← 81H, CY ← 1
- → return

4.10 SQUARE ROOT FUNCTION (LSQRT)

(1) Processing

With the value of FPR1 designated as a, returns $\sqrt{a}$ in FPR1.

(2) Object module files subject to linkage

DFLT, LFLT1, LLD, LSQRT

(3) Required stack size

4 (including 2-byte return address from LSQRT)

(4) Registers used

AX, BC, DE, HL

(5) Work areas used

FPR1, FPR2, FPR3, FPR4, FPR1_X, FPR2_X, FPR3_X, FPR4_X

(6) Processing time (internal system clock = 8.38 MHz)

Average: 1993 us
Maximum: 2076 us ($\sqrt{(5.1001101e - 35)}$)

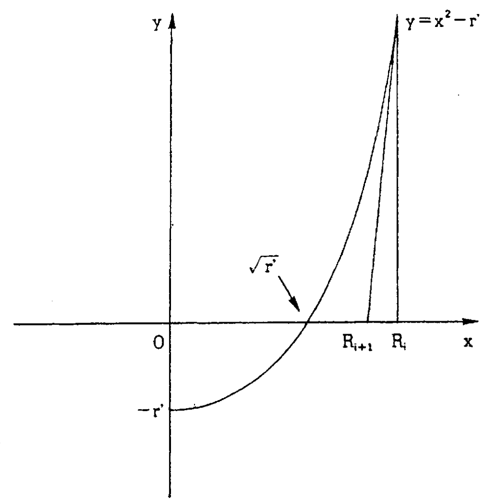
(7) Algorithm

The exponent n of $\sqrt{a}$ is found from the following expression.

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

The Newton-Raphson method is used to calculate $\sqrt{r}$. As shown in the figure on the next page, $\sqrt{r'}$ is the x coordinate of the intersection point of the quadratic function $y = x^2 - r'$ and the x axis.

Taking R_i as the approximate value of $\sqrt{r'}$, the straight line $y = x^2 - r'$ is drawn from the point (R_i, $R_1{}^2 - r'$). If the x coordinate of the point of intersection of this line with the x axis is designated R_{i+1}, R_{i+1} is a more closely approximate value of $\sqrt{r'}$ than R_i.



R_{i+1} is found from R_i using the following expression.

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

With this function, the 5th approximate value R5 is found with $r' = r$ if $1 \leqq r < 2$ and $r' = r/4$ if $2 \leqq r < 4$, and initial approximate value R1 = 1, and the mantissa of $\sqrt{a}$ is obtained.

(8) Processing procedure

(a) If a = 0, 0 is returned as the operation result.
(b) If a < 0, the operation terminates abnormally.
(c) n'=n + 7FH and r'/2 are obtained for ae, the exponent of a, (including a 7FH bias) and the mantissa, af, using the following expressions.

If ae is odd : n' = (ae - 7FH)/2 + 7FH = (ae - 1)/2 + 40H, r'/2 = af/2

If ae is even: n' = (ae - 7FH - 1)/2 + 7FH = ae/2 + 3FH, r'/2 = ((af x 2)/4)/2

(d) The 2nd order approximate value of $\sqrt{r'}$, R2 = 1/2 + r'/2, is calculated.
(e) The 3rd, 4th and 5th approximate values are calculated from the approximation expression.
(f) The exponent of the 5th approximate value is substituted for n', giving the solution.

(9) Floating point constant data
Constant data 1/2 is used.

(10) Processing diagram

[E | LSQRT]

- A ← FPR1 exponent

<Exponent processing>

- (IF: A (exponent of a) = 0)
  - [THEN]
    - Jump to T_QRT9
- (IF: FPR1_4 bit 7 (sign bit of a) = 1)
  - [THEN]
    - A ← 81H, CY ← 1
    - return

<Exponent calculation>

- Right-shift A.CY by 1 bit
- (IF: CY (ae = odd number) = 1)
  - [THEN]
    - FPR1 exponent ← 7FH (exponent bias) - 1
  - [ELSE]
    - FPR1 exponent ← 7FH - 2
- FPR4_X ← A+3FH+CY

<Mantissa calculation>

- FPR3 ← FPR1(r'/2)
- FPR1 ← FPR1(r'/2) + 1/2 — [E | LADD]
- (FOR : i=2 to 4)
  - FPR4 ← FPR1(R_i)
  - FPR2 ← FPR1(R_i)
  - FPR1 ← FPR3(r'/2)
  - FPR1 ← FPR1(r'/2) ÷ FPR2(R_i) — [E | LDIV]
  - FPR2 ← FPR4(R_i)
  - FPR2_HP ← FPR2_HP − 80H
  - FPR1 ← FPR1(r'/2R_i) + FPR2(R_i/2) — [E | LADD]

<Solution combination>

- FPR1 exponent ← FPR4_X (n')

[T_QRT9]

- A ← 0, CY ← 0
- return

4.11 arcsin FUNCTION (LASIN)

(1) Processing

With the value of FPR1 designated as x, returns arcsin(x) in FPR1.

- Valid range of input value x: -1 to 1
- Returned value range : $-\pi/2$ to $\pi/2$
- Unit : Radians

(2) Object module files subject to linkage

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

(3) Required stack size

6 (including 2-byte return address from LASIN)

(4) Registers used

AX, BC, DE, HL

(5) Work areas used

FPR1, FPR2, FPR3, FPR4, FPR5, FPR1_X, FPR2_X, FPR3_X, FPR4_X, FPR5_X

(6) Processing time (internal system clock = 8.38 MHz)

Average: 5265 us
Maximum: 6675 us (arcsin(0.98437494))

(7) Algorithm

The solution is found by conversion to the arctangent function using the following expression.

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

(8) Processing procedure

(a) If x = 1 or x = -1, the operation is ended with $\pi/2$ or $-\pi/2$ as the solution, respectively.
(b) If $|x| > 1$, the operation terminates abnormally.
(c) $x/\sqrt{(1-x^2)}$ is found, and the procedure jumps to the LATAN function.

(9) Floating point constant data

Constant data 1 and $\pi/2$ with mantissa extensions are used.

(10) Processing diagram

E	LASIN

<Exception processing>

- FPR5 ← FPR1(x)
- FPR1_4 bit 7 (sign bit) ← 0
- ◇ (IF : FPR1(|x|) = 1)
  - [THEN]
    - FPR1・FPR1_X ← $\pi/2$
    - FPR1_4 bit 7 ← FPR5_4 bit 7 (sign bit of x)
    - A ← 0, CY ← 0
    - → return
- ◇ (IF : FPR1(|x|) > 1)
  - [THEN]
    - A ← 81H, CY ← 1
    - → return

<Conversion to arctan function>

- FPR2 ← FPR1(|x|)
- FPR1・FPR1_X ← FPR1(|x|) × FPR2(|x|) — E LMLT
- FPR1_4 bit 7 ← 1
- FPR1 ← FPR1・FPR1_X($-x^2$) + 1 — E LADDX
- FPR1 ← $\sqrt{(FPR1(1-x^2))}$ — E LSQRT
- FPR2 ← FPR1($\sqrt{(1-x^2)}$)
- FPR1 ← FPR5(x)
- FPR1 ← FPR1(x) ÷ FPR2($\sqrt{(1-x^2)}$) — E LDIV

→ Jump to LATAN function — E LATAN

4.12 arccos FUNCTION (LACOS)

(1) Processing

With the value of FPR1 designated as x, returns arccos(x) in FPR1.

- Valid range of input value x: -1 to +1
- Returned value range : 0 to π
- Unit : Radians

(2) Object module files subject to linkage

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

(3) Required stack size

8 (including 2-byte return address from LACOS)

(4) Registers used

AX, BC, DE, HL

(5) Work areas used

FPR1, FPR2, FPR3, FPR4, FPR5, FPR1_X, FPR2_X, FPR3_X, FPR4_X, FPR5_X

(6) Processing time (internal system clock = 8.38 MHz)

Average: 5230 us
Maximum: 6794 us (arccos(0.98437494))

(7) Algorithm
arccos(x) is found from the following expression.

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

(8) Processing procedure

(a) arcsin(x) is found using the LASIN function.

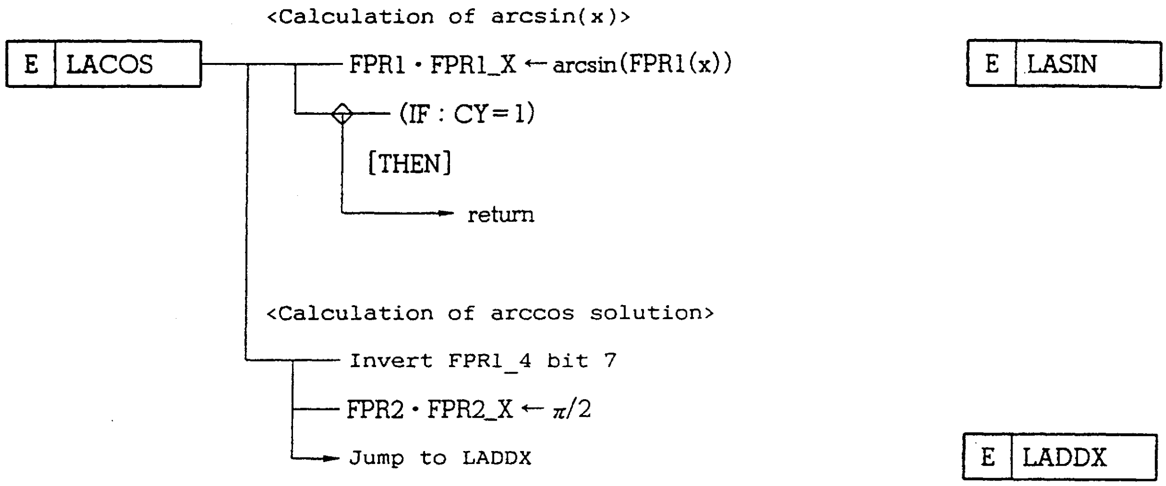
(b) If the LASIN function terminates abnormally, the operation terminates abnormally at that point.

(c) $\pi/2$ - arcsin(x) gives the solution.

(9) Floating point constant data

Constant data $\pi/2$ with a mantissa extension is used.

(10) Processing diagram



4.13 arctan FUNCTION (LATAN)

(1) Processing

With the value of FPR1 designated as x, returns arctan(x) in FPR1.

- Returned value range : $-\pi/2$ to $+\pi/2$
- Unit : Radians

(2) Object module files subject to linkage

DFLT, LFLT1, LFLT2, LLD, LATAN, LRCPN

(3) Required stack size

6 (including 2-byte return address from LATAN)

(4) Registers used

AX, BC, DE, HL

(5) Work areas used

FPR1, FPR2, FPR3, FPR4, FPR1_X, FPR2_X, FPR3_X, FPR4_X, FPR5_X

(6) Processing time (internal system clock = 8.38 MHz)

Average: 2630 us
Maximum: 3839 us (arctan(1.0000001))

(7) Algorithm

The best approximation expression devised by Mr. Goichi Shimauchi (Rikkyo University) is used.

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

In this function, n is taken as 3, and the coefficients a_0, a_1, a_2 and a_3 are fixed as shown below.

a_0 = 0.24999 99999 43
a_1 = -0.00520 83303 18
a_2 = 0.00019 52689 40
a_3 = -0.00000 84855 00

NOTE : Mr. Goichi Shimauchi's best approximation expression can only be used in the range $|x| \leqq 1/8$. Therefore, if $|x| \geqq 1/8$, the following method is used to find the arctangent.

- If $|x| \geqq 1$, let $x' = 1/|x|$

  $\arctan(|x|) = \pi/2 - \arctan(1/|x|)$
- If $x' \geqq 1/8$, V and W are determined as follows:

  $V = \frac{x' - W}{1 + x' \times W}$, W is the most approximate value to x' from among 1/8, 3/8, 5/8, 7/8

  $\arctan(x') = \arctan(W) + \arctan(V)$

Remarks : A TAN function addition theorem is used.

(8) Processing procedure

(a) The sign bit of x is saved, and the absolute value of x is taken.

(b) If $|x| \geqq 1$, $1/|x|$ is found.

(c) Then W is found from the following table, and V is calculated.

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

(d) The 1st term ($4a_0V$) of the arctan(V) approximation polynomial is substituted for arctan(W) $+4a_0V$, and the approximation polynomial calculation is performed.

(e) If $|x| \geq 1$, $\pi/2$ - (approximation expression solution) is found, giving the solution of arctan ($|x|$).

(f) the original sign of x is incorporated in the arctan($|x|$) solution.

(9) Floating point constant data

(a) Constant data 1 and $\pi/2$ with mantissa extensions are used.

(b) Constant data arctan(0), arctan(1/8), arctan(3/8), arctan(5/8) and arctan(7/8) with a mantissa extension are used for as a 5-element array, R_W.

(c) Constant data $4a_0$, $16a_1/a_0$, $16a_2/a_1$ and $16a_3/a_2$ with a mantissa extension are used for the approximation polynomial coefficient series as a 4-element array, K.

(10) Processing diagram

[E | LATAN]

- FPR5_X bit 7 ← FPR1_4 bit 7 (sign bit)

<Primary function conversion>

- FPR1_4 bit 7 ← 0
  - (IF : FPR1(|x|) ≧ 1)
    - [THEN]
      - FPR1 ← 1/FPR1(|x|) — [E | LRCPN]
      - FPR5_X bit 6 ← 0
    - [ELSE]
      - FPR5_X bit 6 ← 1

<Secondary function conversion>

- FPR3_X ← 0
  - (IF : FPR1(x') ≧ 1/8)
    - [THEN]
      - FPR4 ← FPR1(x')
        - (CASE)
          - [FPR1 ≧ 6/8]
            - FPR2 ← 7/8
            - FPR3_X ← 20 (arctan(7/8) array R_w internal relative address)
          - [4/8 ≦ FPR1 < 6/8]
            - FPR2 ← 5/8
            - FPR3_X ← 15 (arctan(5/8) array R_w internal relative address)
          - [2/8 ≦ FPR1 < 4/8]
            - FPR2 ← 3/8
            - FPR3_X ← 10 (arctan(3/8) array R_w internal relative address)
          - [FPR1 < 2/8]
            - FPR2 ← 1/8
            - FPR3_X ← 5 (arctan(1/8) array R_w internal relative address)
      - FPR3 ← FPR2(W)
      - FPR1 ← FPR1(x') × FPR2(W) — [E | LMLT]
      - FPR1 ← FPR1(x' × W) + 1 — [E | LADD]
      - FPR1(x' × W + 1) ↔ FPR4(x')
      - FPR2 ← FPR3(W)
      - FPR1 ← FPR1(x') − FPR2(W) — [E | LSUB]
      - FPR2 ← FPR4(x' × W + 1)
      - FPR1 ← FPR1(x' − W) ÷ FPR2(x' × W + 1) — [E | LDIV]

<Calculation of approximation solution>

- FPR4 ← FPR1(V)
- FPR2 ← FPR1(V)
- FPR1 · FPR1_X ← FPR1 × FPR2 — [E | LMLT]
- FPR1(V^2) ↔ FPR4(V)
- FPR4_X ← FPR1_X (V^2 mantissa extension)
- FPR1_X ← 0
- FPR1 · FPR1_X ← FPR1 · FPR1_X(V) × $4a_0$ — [E | LMLTX]
- FPR2.FPR2_X ← load arctan(W) constant indicated by FPR3_X
- FPR3 · FPR3_X ← FPR1 · FPR1_X($4a_0$V)
- FPR1 · FPR1_X ← FPR1 · FPR1_X($4a_0$V) + FPR2 · FPR2_X(arctan(W)) — [E | LADDX]
- HL ← address of 2nd element of array K
- B ← Number of elements of array K - 1
- FPR1 · FPR1_X ← approximation solution (arctan(W) + arctan(V)) — [E | LPLY2]
- (IF: FPR5_X bit 6 (|x| < 1) = 0)
  - [THEN]
    - Invert FPR1_4 bit 7 (sign bit)
    - FPR1 · FPR1_X ← FPR1 · FPR1_X(−arctan(x')) + $\pi/2$ — [E | LADDX]

<Sign processing>

- FPR1_4 bit 7 ← FPR5_X bit 7 (sign bit of x)
- A ← 0, CY ← 0
- → return

4.14 sinh FUNCTION (LHSIN)

(1) Processing

With the value of FPR1 designated as x, returns sinh(x) in FPR1.

(2) Object module files subject to linkage

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

(3) Required stack size

8 (including 2-byte return address from LHSIN)

(4) Registers used

AX, BC, DE, HL

(5) Work areas used

FPR1, FPR2, FPR3, FPR4, FPR5_1, FPR5_X, FPR2_X, FPR3_X, FPR4_X, FPR5_X

(6) Processing time (internal system clock = 8.38 MHz)

Average: 3310 us
Maximum: 4784 us (sinh(3.351413))

(7) Algorithm

- If $|x| \geqq 0.5$, the following expression is used.

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

- If $|x| < 0.5$, the Taylor approximation expression is used.

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

(8) Processing procedure

- When $|x| \geq 0.5$

(a) The sign of x is stored, and the absolute value of x is taken.
(b) $e^{|x|}$ is found using the LEXP function.
(c) If $e^{|x|}$ overflows, the operation terminates abnormally.
(d) $(e^{|x|} - e^{-|x|})/2$ is found and the original sign of x is incorporated, to give the solution.

- When $|x| < 0.5$

sinh(x) is found by means of the Taylor approximation expression.

(9) Floating point constant data

Constant data 1/3!, 3!/5! and 5!/7! with a mantissa extension are used for the Taylor approximation expression coefficient series as a 3-element array, K.

(10) Processing diagram

- E | LHSIN
  - FPR5_1 bit 7 ← FPR1_4 bit 7 (sign bit)
  - A ← FPR1_4 & 7FH
  - <Case where |x| ≧ 1/2>
    (IF : A≧3FH)
    [THEN]
    - FPR1_4 bit 7 ← 0
    - FPR1 ← $e^{FPR1(|x|)}$ — E | LEXP
    - (IF: overflow)
      [THEN]
      - → return
    - FPR3 ← FPR1($e^{|x|}$)
    - FPR1 ← 1÷FPR1($e^{|x|}$) — E | LRCPN
    - FPR2 ← FPR3($e^{|x|}$)
    - FPR1 ← FPR1($e^{-|x|}$) − FPR2($e^{|x|}$) — E | LSUB
    - FPR1_HP ← FPR1_HP − 80H
    - FPR1_4 bit 7 ← FPR5_1 bit 7 (sign bit)

    <Case where |x| < 1/2>
    [ELSE]
    - FPR4 ← FPR1(x)
    - FPR2 ← FPR1(x)
    - FPR1 · FPR1_X ← FPR1(x) × FPR2(x) — E | LMLT
    - FPR1(x^2) ↔ FPR4(x)
    - FPR4_X ← FPR1_X (mantissa extension of x^2)
    - FPR1_X ← 0
    - HL ← start address of array K
    - B ← Number of elements of array K, 3
    - FPR1.FPR1_X ← direct approximation solution — E | LPLY
  - A ← 0, CY ← 0
  - → return

4.15 cosh FUNCTION (LHCOS)

(1) Processing

With the value of FPR1 designated as x, returns cosh(x) in FPR1.

(2) Object module files subject to linkage

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

(3) Required stack size

8 (including 2-byte return address from LHCOS)

(4) Registers used

AX, BC, DE, HL

(5) Work areas used

FPR1, FPR2, FPR3, FPR4, FPR1_X, FPR2_X, FPR3_X, FPR4_X, FPR5_X

(6) Processing time (internal system clock = 8.38 MHz)

Average: 4139 us
Maximum: 4768 us (cosh(4.0319099))

(7) Algorithm

cosh(x) is found by means of the following expression.

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

(8) Processing procedure

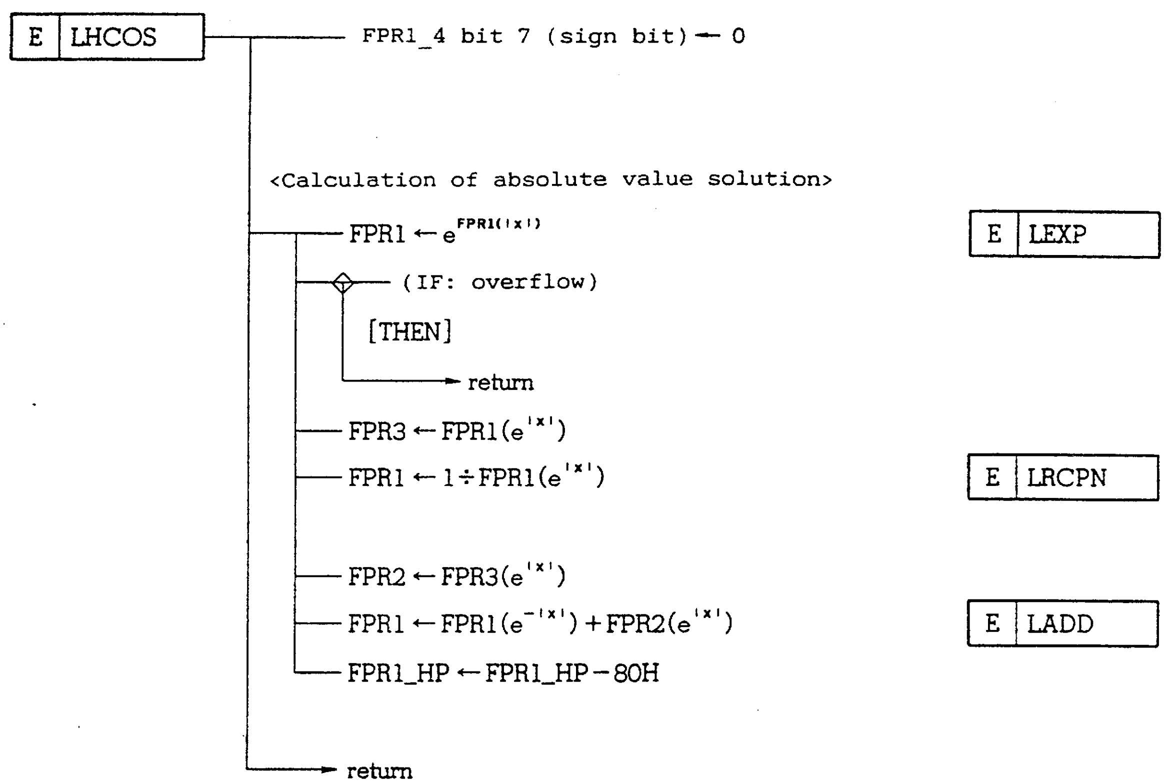
(a) The absolute value of x is taken.

(b) $e^{|x|}$ is found using the LEXP function.

(c) If $e^{|x|}$ overflows, the operation terminates abnormally.

(d) $(e^{|x|} + 1/e^{|x|})/2$ is found, giving the solution.

(9) Processing diagram



4.16 tanh FUNCTION (LHTAN)

(1) Processing

With the value of FPR1 designated as x, returns tanh(x) in FPR1.

(2) Object module files subject to linkage

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

(3) Required stack size

12 (including 2-byte return address from LHTAN)

(4) Registers used

AX, BC, DE, HL

(5) Work areas used

FPR1, FPR2, FPR3, FPR4, FPR5, FPR1_X, FPR2_X, FPR3_X, FPR4_X, FPR5_X

(6) Processing time (internal system clock = 8.38 MHz)

Average: 7792 us
Maximum: 10021 us (tanh(9.4484739))

(7) Algorithm

tanh(x) is found by means of the following expression.

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

(8) Processing procedure

(a) cosh(x) is found using the LHCOS function.

(b) In the case of overflow:

. If x > 0, the operation is ended with 1 as the solution.

. If x < 0, the operation is ended with -1 as the solution.

(c) sinh(x) is found using the LHSIN function.

(d) sinh(x)/cosh(x) is found, giving the solution.

(9) Floating point constant data

Constant data 1 is used.

(10) Processing diagram

- [E | LHTAN]
  - FPR5 ← FPR1(x)
  - <Calculation of cosh solution>
  - FPR1 ← cosh(FPR1(x)) [E | LHCOS]
    - (IF: overflow)
      - [THEN]
        - FPR1 ← 1
        - FPR1_4 bit 7 ← FPR5_4 bit 7 (sign bit)
        - A ← 0, CY ← 0
        - → return
    - FPR1_HP · FPR1_2(cosh(x)) ↔ FPR5_HP · FPR5_2(x)
    - Save FPR1_1 (cosh(x) 3rd mantissa part) to stack
    - FPR1_1 ← FPR5_1 (x 3rd mantissa part)
  - <Calculation of sinh solution>
  - FPR1 ← sinh(FPR1(x)) [E | LHSIN]
  - <Calculation of tanh solution>
  - FPR2_HP · FPR2_2 ← FPR5_HP · FPR5_2(cosh(x))
    - FPR2_1 ← cosh(x) 3rd mantissa part restored from stack
    - → Jump to LDIV [E | LDIV]

4.17 ABSOLUTE VALUE FUNCTION (LABS)

(1) Processing

Takes the absolute value of FPR1, and returns this value in FPR1.

(2) Object module files subject to linkage

DFLT, LABS

(3) Required stack size

2 (2-byte return address from LABS only)

(4) Registers used

A

(5) Work areas used

FPR1

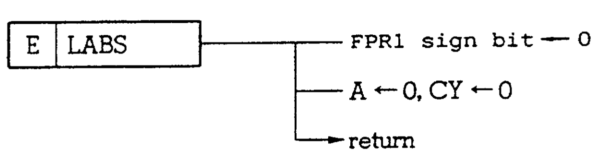
(6) Processing time (internal system clock = 8.38 MHz)

6.4 us

(7) Processing procedure

The sign bit of FPR1 is zeroized.

(8) Processing diagram



4.18 RECIPROCAL FUNCTION (LRCPN)

(1) Processing

Takes the reciprocal of the value of FPR1, and returns this value in FPR1.

(2) Object module files subject to linkage

DFLT, LFLT1, LLD, LRCPN

(3) Required stack size

4 (including 2-byte return address from LRCPN)

(4) Registers used

AX, BC, DE, HL

(5) Work areas used

FPR1, FPR2, FPR1_X, FPR2_X

(6) Processing time (internal system clock = 8.38 MHz)

Average: 539 us
Maximum: 637 us (1/(8.5070602e + 37))

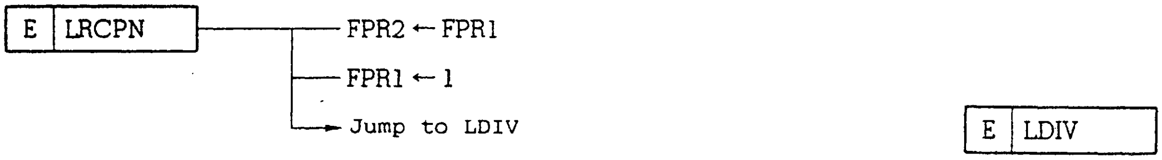
(7) Processing procedure

(a) The value of FPR1 is transferred to FPR2, and the constant 1 is set in FPR1.
(b) The procedure jumps to the LDIV function.

(8) Floating point constant data

Constant data 1 is used.

(9) Processing diagram



CHAPTER 5. COODINATE CONVERSION FUNCTIONS

The following coordinate conversion functions are provided.

(1) Polar coodinate → rectangular coordinate conversion function (POTORA)

Converts polar coordinate values (r. θ) to rectangular coordinate values (x, y).
FPR1 is used for transfer of the r and x values, and FPR2 for transfer of the θ and y values.

(2) Rectangular coordinate → polar coordinate conversion function (RATOPO)

Converts rectangular coordinate values (x, y) to polar coordinate values (r, θ).
FPR1 is used for transfer of the x and r values, and FPR2 for transfer of the y and θ values.

5.1 POLAR COORDINATE → RECTANGULAR COORDINATE CONVERSION FUNCTION (POTORA)

(1) Processing

With the value of FPR1 designated as r and the value of FPR2 as Θ, converts polar coordinates (r, Θ) to rectangular coordinates (x, y), and returns x in FPR1 and y in FPR2.

- Unit of Θ: Radians

(2) Object module files subject to linkage

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

(3) Required stack size

14 (including 2-byte return address from POTORA)

(4) Registers used

AX, BC, DE, HL

(5) Work areas used

FPR1, FPR2, FPR3, FPR4, FPR5, FPR1_X, FPR2_X, FPR3_X, FPR4_X, FPR5_X

(6) Processing time (internal system clock = 8.38 MHz)

Average: 5336 us
Maximum: 10845 us (r = 0.5, θ = 6.8056469e + 38)

(7) Algorithm

Conversion is performed by means of the following expressions.

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

(8) Processing procedure

(a) If r = 0, coordinates (0, 0) are returned.
(b) If r < 0, the operation terminates abnormally.
(c) θ is found using the LMOD90 function, and converted to $\theta = s(\theta' + n\pi/2)$, where s is the θ sign, n is an integer and $0 \leq \theta' < \pi/2$.
(d) $\sin(s(\theta' + n\pi/2))$ is found using the LSIN90 function, and $\cos(\theta' + n\pi/2)$ using the LCOS90 function.
(e) $r \times \cos(\theta' + n\pi/2)$ is taken as x, and $r \times \sin(s(\theta' + n\pi/2))$ as y.

(9) Processing diagram

<Exception processing>

E POTORA

- (IF : FPR1(r) = 0)
  - [THEN]
    - FPR2_HP ← 0
    - → Jump to T_ORA9
- (IF : FPR1(r) < 0)
  - [THEN]
    - A ← 81H, CY ← 1
    - → return

<Coordinate conversion>

- Save FPR1 (r) to stack
- FPR1 ← FPR2(θ)
- FPR1.FPR1_X ← θ', FPR4_3 ← n, FPR4_4 bit 7 ← s — E LMOD90
- Save FPR4_3(n), FPR4_4 bit 7(sign bit) to stack
- FPR5 · FPR5_X ← FPR1 · FPR1_X(θ')
- FPR1 · FPR1_X ← cos(FPR1 · FPR1_X(θ') + FPR4_3(n) × π/2) — E LCOS90
- FPR1 · FPR1_X(cos |θ|) ↔ FPR5 · FPR5_X(θ')
- FPR4_3 ← n restored from stack, FPR4_4 bit 7 ← s restored from stack
- FPR1.FPR1_X ← sin(FPR4_4 bit 7 (s)(FPR1 · FPR1_X(θ') + FPR4_3(n) × π/2)) — E LSIN90
- FPR3 ← r restored from stack
- FPR3_X ← 0
- FPR2 · FPR2_X ← FPR3 · FPR3_X(r)
- FPR1 · FPR1_X ← FPR1 · FPR1 _X(sinθ) × FPR2 · FPR2_X(r) — E LMLTX
- FPR1 · FPR1 _X(rsinθ) ↔ FPR5 · FPR5_X(cos |θ|)
- FPR2 · FPR2_X ← FPR3 · FPR3_X(r)
- FPR1 · FPR1_X ← FPR1 · FPR1 _X(cos |θ|) × FPR2 · FPR2_X(r) — E LMLTX
- FPR2 · FPR2_X ← FPR5 · FPR5 _X(rsinθ)

T_ORA9

- A ← 0, CY ← 0
- → return

5.2 RECTANGULAR COORDINATE → POLAR COORDINATE CONVERSION FUNCTION (RATOPO)

(1) Processing

With the value of FPR1 designated as x and the value of FPR2 as y, converts rectangular coordinates (x, y) to polar coordinates (r, θ), and returns r in FPR1 and θ in FPR2.

- Range of returned value θ: $-\pi$ to $+\pi$
- Unit : Radians

(2) Object module files subject to linkage

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

(3) Required stack size

10 (including 2-byte return address from RATOPO)

(4) Registers used

AX, BC, DE, HL

(5) Work areas used

FPR1, FPR2, FPR3, FPR4, FPR5, FPR1_X, FPR2_X, FPR3_X, FPR4_X, FPR5_X

(6) Processing time (internal system clock = 8.38 MHz)

Average: 5631 us
Maximum:6979 us (x = -12.220954, y = 69.662003)

(7) Algorithm

The following two expressions are used.

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

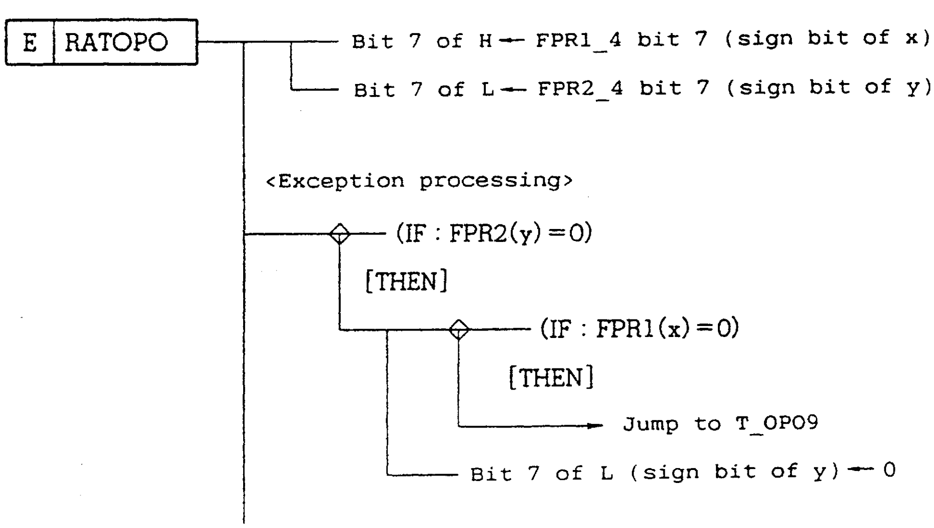
(8) Processing procedure

(a) If $x = y = 0$, these values are returned directly.
(b) $x^2 + y^2$ is found.
(c) If $x^2 + y^2$ overflows, the operation terminates abnormally.
(d) y/x is found.
(e) If y/x overflows, the following procedure is used:
. If $y > 0$, $\theta = \pi/2$
. If $y < 0$, $\theta = -\pi/2$
(f) If y/x terminates normally, arctan(y/x) is found using the LATAN function, and this is taken as θ.
(g) If $x < 0$ in (f), the following procedure is used:
. If $y \geqq 0$, π is added to θ
. If $y < 0$, π is subtracted from θ
(h) $\sqrt{(x^2 + y^2)}$ is found, giving r.

(9) Floating point constant data

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

(10) Processing diagram



<Calculation of x^2 + y^2>

- FPR4 ← FPR1(x)
- FPR5 ← FPR2(y)
- FPR2 ← FPR1(x)
- FPR1 · FPR1_X ← FPR1(x) × FPR2(x) — **E | LMLT**
- (IF: overflow)
  - [THEN]
    - → return
- FPR3 · FPR3_X ← FPR1 · FPR1_X(x^2)
- FPR1 ← FPR5(y)
- FPR2 ← FPR1(y)
- FPR1 · FPR1_X ← FPR1(y) × FPR2(y) — **E | LMLT**
- (IF: overflow)
  - [THEN]
    - → return
- FPR2 · FPR2_X ← FPR3 · FPR3_X(X^2)
- FPR1 ← FPR1 · FPR1_X(y^2) + FPR2 · FPR2_X(x^2) — **E | LADDX**
- (IF: overflow)
  - [THEN]
    - → return
- Save bit 7 of H (sign bit of x) and bit 7 of L (sign bit of y) to stack

<Calculation of y/x>

- FPR1(x^2+y^2) ↔ FPR5(y)
- FPR2 ← FPR4(x)
- FPR1 ← FPR1(y) ÷ FPR2(x) — **E | LDIV**
- (IF: overflow)
  - [THEN]
    - FPR1 ← $\pi/2$
    - Bit 7 of X ← sign bit of y restored from stack
    - Bit 7 of A ← sign bit of x restored from stack
    - Bit 7 of FPR1_4 ← bit 7 of X (sign bit of y)
    - → Jump to T_OP08

<Calculation of θ>

- FPR1 · FPR1_X ← arctan(FPR1(y/x)) — E | LATAN
- Bit 7 of X ← sign bit of y restored from stack
- Bit 7 of A ← sign bit of x restored from stack
- (IF: Bit 7 of A (sign bit) = 1)
  - [THEN]
    - FPR2 · FPR2_X ← π
    - Bit 7 of FPR2_4 ← bit 7 of X (sign bit of y)
    - FPR1 · FPR1_X ← FPR1 · FPR1_X(arctan(y/x)) + FPR2 · FPR2_X($\pm\pi$) — E | LADDX

<Calculation of r>

T_OPO8

- FPR1(θ) ↔ FPR5(x^2+y^2)
- FPR1 ← $\sqrt{(FPR1(x^2+y^2))}$ — E | LSQRT
- FPR2 ← FPR5(θ)

T_OPO9

- A ← 0, CY ← 0
- → return

CHAPTER 6. TYPE CONVERSION FUNCTIONS

The following type conversion functions are provided.

(1) Character string → floating point format conversion function (ATOL)

Converts the character string for which the start address is indicated by the HL register to floating point format, and stores the result in FPR1.

(2) Floating point format → character string conversion function (LTOA)

Converts the value of FPR1 to a character string, and stores the result starting in the address indicated by the HL register.

(3) 2-byte integer type → floating point format conversion function (FTOL)

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

(4) Floating point format → 2-byte integer type conversion function (LTOF)

Converts the value of FPR1 to a 2-byte integer type, and stores the result in the DE register.

6.1 CHARACTER STRING → FLOATING POINT FORMAT CONVERSION FUNCTION (ATOL)

(1) Processing

Converts the character string for which the start address is indicated by the HL register to floating point format, and returns the result in FPR1.

(2) Object module files subject to linkage

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

(3) Required stack size

14 (including 2-byte return address from ATOL)

(4) Registers used

AX, BC, DE, HL (HL register contents are retained)

(5) Work areas used

FPR1, FPR2, FPR3, FPR4, FPR5, FPR1_X, FPR2_X, FPR3_X, FPR4_X, FPR5_X

(6) Processing time (internal system clock = 8.38 MHz)

Average: 5049 us
Maximum: 6388 us ("0.000000000000000000117549428")

(7) Character string components

The character string is composed of the following 17 kinds of characters.

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

(8) Character string format

The character string format is shown below.

```
[(+/-)] 99・・・・・・9 [(E/e) [(+/-)] 99]
             |                      |
          Mantissa              Exponent
                                (base 10)
```

Remarks : [ ]: Can be omitted
9: 0 to 9
(/): One or other to be selected

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

(9) Character string rules

A character string which does not conform to the following rules will result in an error if used in this function.

(a) The end of the string is determined by △ (space) or NUL.

(b) The string must not contain characters other than those shown in the string components table.

(c) The maximum length of the mantissa string is 27 characters, and a maximum of one '.' may be included.
One or more numerals must be included.

(d) The exponent string must be 1 or 2 characters in length.

(e) An error will result if the value is 2^{129} or more, or -2^{129} or less.

Remarks : If the mantissa value is 0, it is processed as an exception. In this case, the exponent string is ignored and 0 is returned as the solution.

(10) Processing procedure

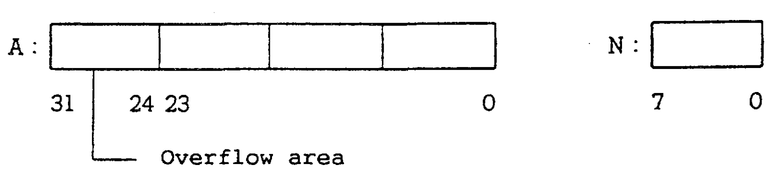
(a) The sign is stored as negative if the first character is '-', and otherwise as positive,

(b) The number of digits in the decimal part is found from the mantissa string, and is designated as F.

(c) The mantissa value A is found as a 4-byte integer type from the mantissa string with the decimal part removed, "A_1, A_2, ..., A_n".

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

The following calculation method is used.



(i) Initial values are set as A = 0, N = 0.

(ii) A_1 is added to A.
If A_1 does not exist, the operation terminates abnormally.

(iii) If the overflow area = 0

A is multiplied by 10, and A_k is added to the result.

(iv) If the overflow area ≠ 0

1 is added to N, and A_k is ignored.

(v) Steps (iii) and (iv) are repeated from k = 2 until k = n.

(v) Nomal termination

if n > 27

(d) The mantissa value A found in (c) is normalized to floating point format, and the stored sign bit is incorporated, giving A'.

(e) The exponent value B is found from the exponent string "(+/-) B_1B_2".

(f) The actual exponent B' is found by adding the number of digits in the decimal part and the number of digits in the ignored mantissa to the exponent value B (B' = B - F + N).

(g) The solution is calculated from A' and B' by means of the following expression.

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

Remarks : dec(x) indicates the decimal part of x, and int(x) indicates the integral part of x.

(11) Floating point constant data

Constant data $\log_2 10$ and log2 with a mantissa extension are used.

(12) Processing diagram

- **E | ATOL**
  - Save HL (string start address) to stack
  - <Sign processing>
  - FPR1_4 bit 7 (sign storage bit) ← 0
    - A ← 1st character no. — GETC
    - (CASE)
      - [A = '+' No.]
        - A ← Next character no. — GETC
      - [A = '-' No.]
        - FPR1_4 bit 7 ← 1
        - A ← Next character No. — GETC
  - <Conversion of mantissa string to numeric value>
  - FPR2_1(F) ← 0, FPR2_2(N) ← 0
    - FPR2_3 (digit counter) ← -1 (initial value)
    - FPR2_4 bit 0 (fraction digit flag) ← 0, FPR2_4 bit 1 (ignored digit flag) ← 0
    - (LOOP)
      - (CASE)
        - [A≦9]
          - (IF: FPR2_3 (digit counter) = initial value)
            - [THEN]
              - BC.DE (A) ← A (1st digit No.)
              - FPR2_3 ← 27
            - [ELSE]
              - Decrement FPR2_3 (digit counter)
              - (IF : FPR2_3=0)
                - [THEN]
                  - Jump to ERROR
              - (IF: FPR2_4 bit 1 (ignored digit flag) = 0)
                - [THEN]
                  - BC·DE ← C·DE×10+A (current digit No.)
                    - (IF: B (A overflow area) ≠ 0)
                      - [THEN]
                        - FPR2_4 bit 1 ← 1
                - [ELSE]
                  - Increment FPR2_2 (N)

```
            (IF: FPR2_4 bit 0 (fraction digit
             flag) = 1)
            [THEN]
              Increment FPR2_1 (F)

        [A = '.' No.]
            (IF: FPR2_4 bit 0 (fraction digit
             flag) = 0)
            [THEN]
              FPR2_4 bit 0 ← 1
            [ELSE]
              → Jump to ERROR

        [OTHERS]
          → Break loop

      Save BC.DE (A) to stack
      A ← Next character No.                    [GETC]
      BC. DE ← A restored from stack
```

<Exception processing>

```
  FPR2_X ← A (character No.)
    (IF: FPR2_3 (digit counter) = initial value)
    [THEN]
      → Jump to ERROR
    (IF : BC・DE(A) =0)
    [THEN]
      FPR1_HP ← 0
      → Jump to T_TOL9
    (IF: FPR2_X (next character No.) ≧ '-' No.)
    [THEN]
      → Jump to ERROR
  FPR2_2 ← FPR2_1(F) − FPR2_2(N)
```

<Normalization of exponent A>

```
  A・C・DE ← BC・DE(A)
  FPR2_1 ← 31 + 7FH (exponent bias)
  FPR1.FPR1_X ← FPR1_4 bit 7 (sign), FPR2_1 (exponent),
  A.C.DE (exponent) normalization              [E | LNOR]
  FPR5・FPR5_X ← FPR1・FPR1_X(A')
  A ← FPR2_X (next character No.)
```

```
<Conversion of exponent string to numeric value>
├── X (B) ← 0, FPR1_1 bit 7 ( sign of B) ← 0
├─◇── (IF: A = 'E' No. or A = 'e' No.)
│  [THEN]
│  └── A ← Next character No.                      [GETC]
│      ├─◇── (CASE)
│      │  [A = '+' No.]
│      │  ├── A ← Next character N.o.                [GETC]
│      │  [A = '-' N.o.]
│      │     FPR1_1 bit 7 (sign of B) ← 1
│      │  └── A ← Next character No.                 [GETC]
│      ├─◇── (IF : A>9)
│      │  [THEN]
│      │  └──→ Jump to ERROR
│      ├── X ← A(B1)
│      ├── A ← Next character No.                    [GETC]
│      └─◇── (IF : A≦9)
│         [THEN]
│         └── B ← A(B2)
│             ├── X ← X(B1) × 10
│             ├── X ← X(B1 × 10) + B(B2)
│             └── A ← Next character No.             [GETC]
├─◇── (IF: A ≠ 'Δ' No. and A ≠ NUL N.o.)
│  [THEN]
│  └──→ Jump to ERROR
└─◇── (IF: FPR1_1 bit 7 (sign of B) = 1)
   [THEN]
   ├── A ← two's complement of X
   [ELSE]
   └── A ← X

<Combination of exponent value and mantissa value>
├── A ← A(B) - FPR2_2(F - N)
├─◇── (IF : A(B') < 0)
│  [THEN]
│  ├── E ← A(B')
│  │   └── D ← FFH
│  [ELSE]
│  └── E ← A(B')
│      └── D ← 0
```

FPR1.FPR1_X ← DE (B') converted to floating point number — E FTOL

FPR1 · FPR1_X ← FPR1 · FPR1_X(B') × $\log_2 10$ — E LMLTX

FPR2 · FPR2_X ← FPR1 · FPR1_X($\log_2 10$ × B')

DE ← FPR1 ($\log_2 10$ x B') exponent — E LTOF

FPR1.FPR1_X ← DE (int($\log_2 10$ x B')) converted to floating point number — E FTOL

Save DE (int($\log_2 10$ x B')) to stack

Invert FPR1_4 bit 7 (sign bit)

FPR1 · FPR1_X ← FPR1 · FPR1_X(−int($\log_2 10$ × B'))
+ FPR2 · FPR2_X($\log_2 10$ × B') — E LADDX

FPR1 · FPR1_X ← FPR1 · FPR1_X(dec($\log_2 10$ × B')) × log2 — E LMLTX

Save FPR5_X (A' mantissa extension) to stack

FPR1 · FPR1_X ← $e^{(FPR1 \cdot FPR1\_X(dec(\log_2 10 \times B') \times \log 2))}$ — E LEXPX

FPR2_X ← A' mantissa extension restored from stack

FPR2 ← FPR5(A')

FPR1 · FPR1_X ← FPR1 · FPR1_X($e^{(dec(\log_2 10 \times B') \times \log 2)}$)
× FPR2 · FPR2_X(A') — E LMLTX

A ← FPR1 exponent

DE ← int($\log_2 10$ x B') restored from stack

A · E ← DE(int($\log_2 10$ × B')) + A

(IF : A ≠ 0)

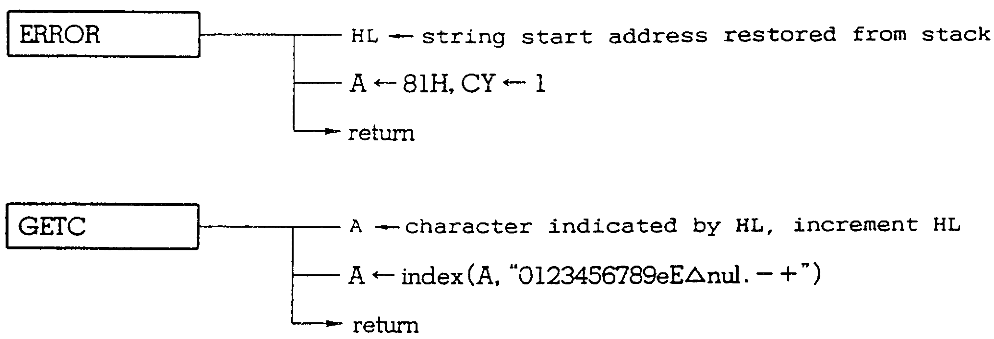
[THEN]

- (IF: A bit 7 = 1)
  - [THEN]
    - E (solution exponent) ← 0
  - [ELSE]
    - Jump to ERROR

FPR1 exponent ← E (solution exponent)

T_TOL9

- HL ← string start address restored from stack
- A ← 0, CY ← 0
- return



Remarks : index(character, string) returns the position (0 to 16) of the character in the string. If the character is not included in the string, 0FFH is returned.

6.2 FLOATING POINT FORMAT CHARACTER STRING CONVERSION FUNCTION (LTOA)

(1) Processing

Converts the value of FPR1 to a character string and stores the string starting at the address indicated by the HL register.

(2) Object module files subject to linkage

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

(3) Required stack size

12 (including 2-byte return address from LTOA)

(4) Registers used

AX, BC, DE, HL (HL register contents are retained)

(5) Work areas used

FPR1, FPR2, FPR3, FPR4, FPR5, FPR1_X, FPR2_X, FPR3_X, FPR4_X, FPR5_X

(6) Processing time (internal system clock = 8.38 MHz)

Average: 9405 us
Maximum: 10757 us (-1.2677555e + 13)

(7) Output string format

```
Format 1:  [-] 9.999999e [-] 99
               (Mantissa)    (Exponent)

Format 2: 0
```

Rules :

(a) Format 2 is output when the value is 0; format 1 is used in all other cases.

(b) A "NUL" code is added to the end of the string.

(c) The mantissa and exponent strings in format 1 have fixed lengths of 8 and 2 characters respectively.

(d) The sign of the mantissa and exponent in format 1 is added only when negative.

(e) The maximum length of the string, including the "NUL" code, is 14 characters.

(8) Processing procedure

(a) If the floating point value $x = 0$, the string "0NUL" is output and the operation ends.

(b) x is converted to $a \times 10^b$ using the following expression.

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

Here, floor(x) is the nearest integer in the negative direction from x.
10^{-b} is not calculated directly if $b = 38$ because 10^{-38} will result in underflow in the 78K/0 floating point system.

(c) If $a < 0$, '-' is output, and a $|a|$ is performed.

(d) Mathematically $1 \leqq a < 10$, but in actuality $a < 1$ or $a \geqq 10$ may be obtained due to calculation error. In this case, correction is performed as shown blow.

$$\text{If } a \geqq 10, \; a \leftarrow a/10, \; b \leftarrow b+1$$
$$\text{If } a < 1, \; a \leftarrow a \times 10, \; b \leftarrow b-1$$

(e) Since $1 \leqq a < 10$, the decimal point position is fixed.
From the result of the following calculation, the string "A_1, A_2, ..., A_7" is output.

$$
\begin{array}{l}
\qquad\qquad a_1 = a \\
A_1 = \mathrm{int}(a_1),\ a_2 = \mathrm{dec}(a_1) \times 10 \\
A_2 = \mathrm{int}(a_2),\ a_3 = \mathrm{dec}(a_2) \times 10 \\
\qquad \cdots\cdots\cdots\cdots \\
A_7 = \mathrm{int}(a_7)
\end{array}
$$

Here, int(a) is the integral part of a and dec(a) is the decimal part of a.

(f) 'e' is output.

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

(h) The b string "B_1B_2" ($B_1 = b/10$, $B_2 = b - B_1 \times 10$) is output.

(i) A "NUL" code is output.

(9) Floating point constant data

Constant data $10^{-38} \times 4$, 10 and 1/10 with a mantissa extension are used.

(10) Processing diagram

E | LTOA

- Save HL (output start address) to stack
- <0 format output>
- ◇ (IF : FPR1(x) =0)
  - [THEN]
    - Output "0" to address indicated by HL, and increment HL
    - → Jump to T_TOA9
- <Exponent base conversion (2 → 10)>
- FPR5 ← FPR1(x)
  - FPR1_4 bit 7 (sign bit) ← 0
  - FPR1 · FPR1_X ← $\log_{10}$(FPR1(|x|)) — E | LLOG10
- DE ← trunk(FPR1($\log_{10}$ |x|)) — E | LTOF
  - ◇ (IF: Z (FPR1 = integer) ≠ 1 or FPR1_X (mantissa extension) ≠ 0) and (FPR1 sign bit = 1 and FPR1 exponent ≠ 0))
    - [THEN]
      - Decrement DE
  - A ← E
  - Save A (b) to stack
  - ◇ (IF : DE(b) =38)
    - [THEN]
      - FPR1 ← FPR5(x)
      - Subtract 2 from FPR1 exponent
      - FPR1_X ← 0
      - FPR1 · FPR1_X ← FPR1 · FPR1_X(x/4) × (10^{-38} × 4) — E | LMLTX
    - [ELSE]
      - FPR1 ← DE(b) converted to floating point number — E | FTOL
      - Invert FPR_4 bit 7 (sign bit)
      - FPR1 · FPR1_X ← $10^{FPR1(-b)}$ — E | LEXP10
      - FPR2 ← FPR5(x)
      - FPR2_X ← 0
      - FPR1 · FPR1_X ← FPR1 · FPR1_X(10^{-b}) × FPR2 · FPR2_X(x) — E | LMLTX
  - A ← b restored from stack
  - FPR3_1 ← A(b)

<Mantissa string output>

- HL ← Output start address restored from stack
- Save HL to stack
- (IF: FPR1_4 bit 7 (sign of a) = 1)
  - [THEN]
    - Output '-' to address indicated by HL, increment HL
    - FPR1_4 bit 7 ← 0
- Save HL to stack
- (IF : FPR1 · FPR1_X(|a|) ≧ 10)
  - [THEN]
    - FPR1 · FPR1_X ← FPR1 · FPR1_X(|a|) × 1/10 | E | LMLTX |
    - Increment FPR3_1 (b)
- (IF : FPR1 · FPR1_X(|a|) < 1)
  - [THEN]
    - FPR1 · FPR1_X ← FPR1 · FPR1_X(|a|) × 10 | E | LMLTX |
    - Decrement FPR3_1
- DE ← int(FPR1(a_1)) | E | LTOF |
- HL ← Output address restored from stack
- Output E (A_1) + 30H to address indicated by HL, increment HL
- Output '.' to address indicated by HL, increment HL
- (FOR : i=1 TO 6)
  - Save HL to stack
  - FPR2 · FPR2_X ← FPR1 · FPR1_X(a_i)
  - FPR1.FPR1_X ← DE (int(a_i)) converted to floating point number | E | FTOL |
  - FPR1_4 bit 7 (sign bit) ← 1
  - FPR1 · FPR1_X ← FPR1 · FPR1_X(−int(a_i)) + FPR2 · FPR2_X(a_i) | E | LADDX |
  - FPR1 · FPR1_X ← FPR1 · FPR1_X(dec(a_i)) × 10 | E | LMLTX |
  - DE ← int(FPR1(a_{i+1})) | E | LTOF |
  - HL ← Output address restored from stack
  - Output E (A_{i+1}) + 30H to address indicated by HL, increment HL

<Exponent string output>

- Output 'e' to address indicated by HL, increment HL
- A ← FPR3_1(b)
- (IF : A<0)
  - [THEN]
    - Output '-' to address indicated by HL, increment HL
    - Find two's complement of A
- AX ← A
- AX ← quotient of AX (|b|) ÷ 10, C ← remainder of AX (|b|) ÷ 10
- A ← C (remainder of (|b|)/10)
- Output X (B_1) + 30H to address indicated by HL, increment HL
- Output A (B_2) + 30H to address indicated by HL, increment HL

T_TOA9

- Output NUL code to address indicated by HL
- HL ← Output start address restored from stack
- A ← 0, CY ← 0
- → return

Remarks : trunk (x) indicates the integer obtained by rounding the decimal part of x toward zero.

$$\left[\begin{array}{l} \text{If } x \geq 0,\ \text{trunk}(x) \leq x < \text{trunk}(x) + 1 \\ \text{If } x < 0,\ \text{trunk}(x) - 1 < x \leq \text{trunk}(x) \end{array}\right]$$

6.3 2-BYTE INTEGER TYPE → FLOATING POINT FORMAT CONVERSION FUNCTION (FTOL)

(1) Processing

Converts the contents of the DE register to floating point format as a signed 2-byte integer type, and returns the result in FPR1.

(2) Object module files subject to linkage

DFLT, FTOL

(3) Required stack size

2 (2-byte return address from FTOL only)

(4) Registers used

AX, C, DE (DE register contents are retained)

(5) Work areas used

FPR1, FPR1_X

(6) Processing time (internal system clock = 8.38 MHz)

Average: 40.1 us
Maximum: 72.3 us (-1)

(7) 2-byte integer type format

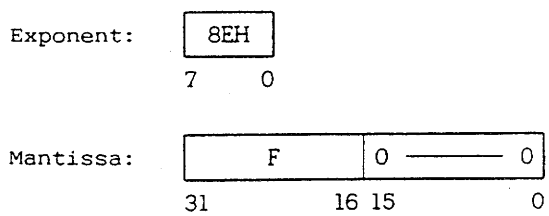
The MSB is the sign bit, and a negative number is expressed as the two's complement.
The 2-byte integer type F represents an integer value in the range -8000H to +7FFFH.

(8) Processing procedure

(a) If the 2-byte integer F = 0, 0 is returned.

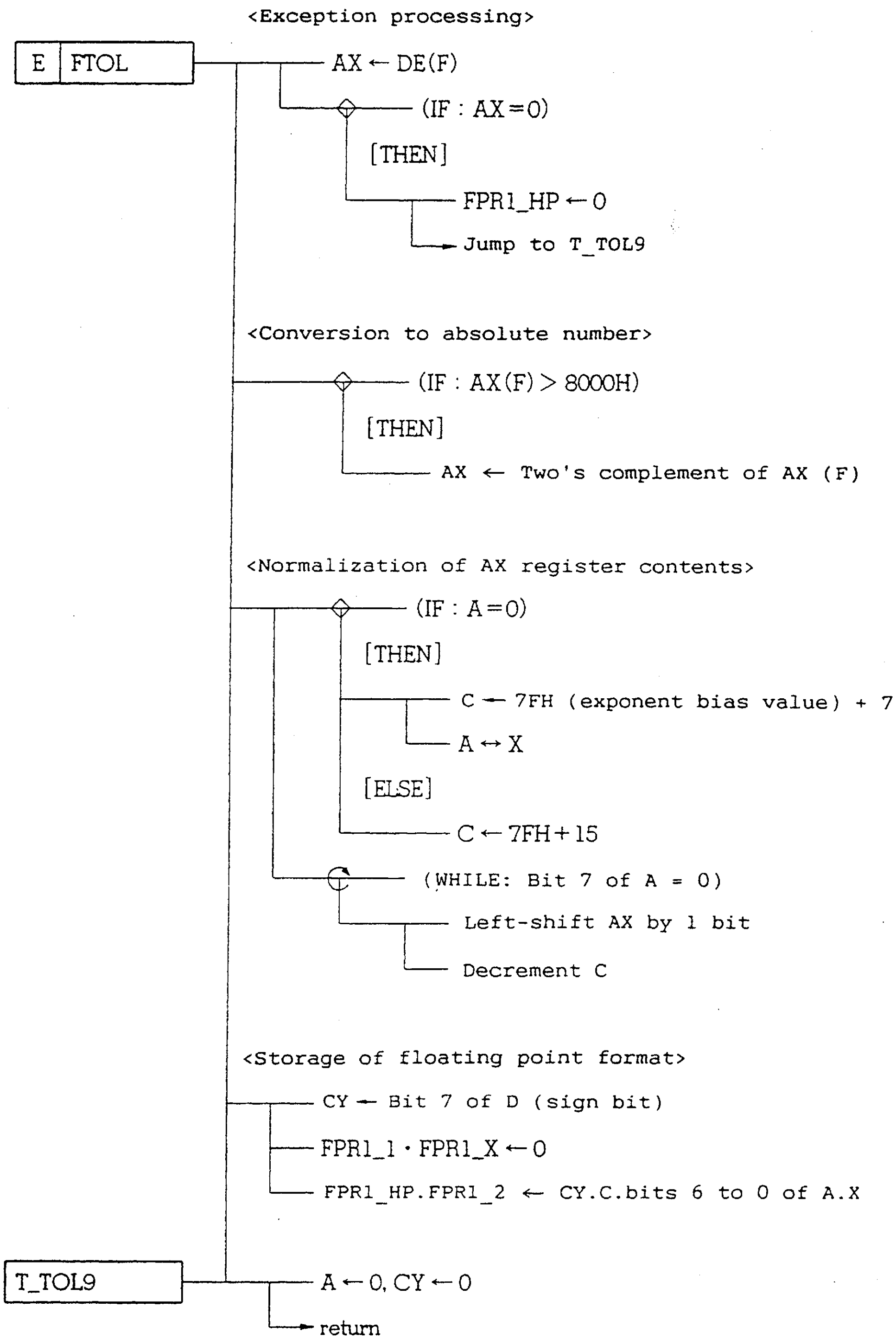
(b) If -7FFFH ≦ F <0, the two's complement of F is taken.

(c) Considering the state of the exponent and mantissa to be the initial state shown below, conversion to floating point format is performed by performing normalization.



(d) The exponent and mantissa obtained in (c) and the sign bit of F are stored in FPR1.

(9) Processing diagram



6.4 FLOATING POINT FORMAT → 2-BYTE INTEGER TYPE CONVERSION FUNCTION (LTOF)

(1) Processing

Converts the value of FPR1 to a signed 2-byte integer type, and returns the result in the DE register. Also, Z flag = 1 is returned if FPR1 does not contain a decimal part, and Z flag = 0 is returned if rounding (truncation of the decimal part) is performed in the conversion process.

(2) Object module files subject to linkage

DFLT, LTOF

(3) Required stack size

2 (2-byte return address from LTOF only)

(4) Registers used

AX, C, DE

(5) Work areas used

FPR1, FPR1_X (FPR1 and FPR1_X contents are retained)

(6) Processing time (internal system clock = 8.38 MHz)

Average: 62.3 us
Maximum: 88.5 us (-1)

(7) Processing procedure

(a) If the exponent = 0, integer value 0 and Z flag = 1 are returned.
If the exponent < 7FH, integer value 0 and Z flag

= 0 are returned.

If the exponent ≥8FH, an error is returned.

(b) The exponent is extracted in unsigned integer format.

The MSB is set.

If the exponent < 87H, the 1st mantissa part is right-shifted by (86H - exponent) bits, and an integer value is obtained.

If the exponent ≥ 87H, the 1st and 2nd mantissa parts are right-shifted by (8EH - exponent) bits, and an integer value is obtained.

(c) Unsigned integer → integer conversion is performed. Conversion is performed on the integer value obtained in (b) and the sign of FPR1 in accordance with the following conditions.

Negative and greater than 8000H → error

Negative and 8000H or less → two's complement is taken

Positive and 8000H or greater → error

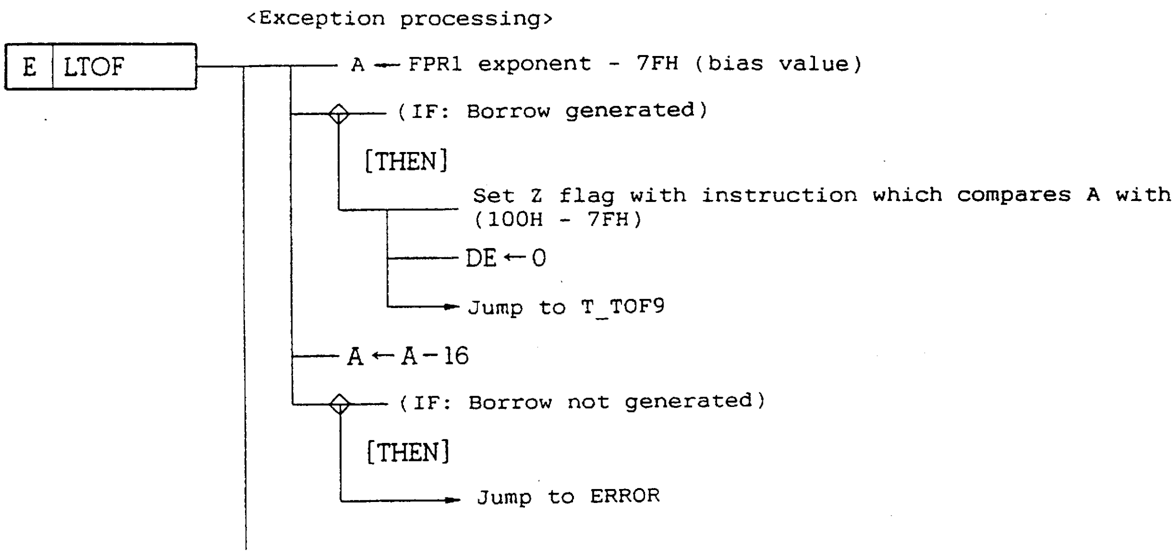
Positive and less than 8000H → unchanged

(d) In (b),

- When Z flag = 1 is returned
  - . In the case where the exponent < 87H, when all bits of the 2nd and 3rd mantissa parts are 0, and no carry is generated by the right-shift of the 1st mantissa part
  - . In the case where the exponent ≥ 87H, when all bits of the 3rd mantissa parts are 0, and no carry is generated by the right-shift of the 1st and 2nd mantissa parts
- When Z flag = 0 is returned

  Other than the above

(8) Processing diagram



<Conversion to unsigned integer type>

- (IF : A≧(100H−8))
  - [THEN]
    - C ← A
    - A ← FPR1_3 (1st mantissa part) or 80H (MSB)
    - X ← FPR1_2 (2nd mantissa part)
    - E ← FPR1_1 (3rd mantissa part)
  - [ELSE]
    - C ← A+8
    - A ← 0
    - X ← FPR1_3 (1st mantissa part) or 80H (MSB)
    - E ← FPR1_2 (2nd mantissa part) or FPR1_1 (3rd mantissa part)
- D ← 0
- Right-shift AX.D ((100H - C) - 1) times

<Conversion to signed integer type>

- (IF: FPR1_4 bit 7 (sign bit) = 1)
  - [THEN]
    - AX ← Two's complement of AX (unsigned integer value)
      - (IF : AX<8000H)
        - [THEN]
          - → Jump to ERROR
  - [ELSE]
    - (IF : A≧80H)
      - [THEN]
        - → Jump to ERROR
- AX ↔ DE

<Determination of decimal part>

- Set Z flag by comparison of AX with 0

T_TOF9

- A ← 0, CY ← 0
- → return

ERROR

- A ← 81H, CY ← 1
- → return

CHAPTER 7. EXECUTION RESULTS

This chapter shows the operation results and processing time for each function.

The processing times given are values measured under the following conditions.

CPU	: uPD78P014 (IE-78014-R-EM)
Operating clock	: Main system clock (internal system clock = 8.38 MHz)
Code area	: External alternate memory
Work area	: Internal RAM
Stack area	: 0FE00H to 0FE1FH
Programmable wait	: Total area no wait

Timer 0 of the uPD78P014 is used to measure the processing time, and the value includes the time taken to read the timer value (approx. 10 clock cycles).

The operation results include a rounding error due to conversion between the internal format (floating point format) and decimal notation.
Operation results for a PC-9801 (MSC ver. 5.1) are also given for reference.

7.1 FLOATING POINT ADDITION (LADD)

0 + 0 = 0 (13.6 us)
1 + 0 = 1 (13.6 us)
0 + 1 = 1 (36.5 us)
1234 + 98765 = 99999 (112 us)
4.5567831e + 09 + 2.1447790e + 06 = 4.5589279e + 09 (141 us)
1.3e + 20 + 4e - 30 = 1.3e + 20 (29.4 us)
223 + 0.111111 = 223.11111 (141 us)
2.1474836e + 09 + 0.5 = 2.1474836e + 09 (29.4 us)
3.4028237e + 38 + 3.4028237e + 38 = Abnormal termination (48.4 us)
0.5 + (-0.5) = 0 (53.7 us)

1.7632415e - 38 + (-1.1754944e - 38) = 0 (63.7 us)

1.0737418e + 09 + 0.5 = 1.0737418e + 09 (293 us)

0.5 + (-0.50000006) = -5.9604645e - 08 (332 us)

7.2 FLOATING POINT SUBTRACTION (LSUB)

0 - 0 = 0 (16.5 us)

1 - 0 = 1 (16.5 us)

0 - 1 = -1 (39.4 us)

1.7014110e + 38 - 1e + 32 = 1.70141e + 38 (223 us)

2.3352e - 05 - 9.99999e - 21 = 2.3352e - 05 (32.2 us)

3.7634668e - 24 - 1.2 = -1.2 (42.2 us)

9.8999999e - 38 - 2.2000001e - 38 = 7.6999998e - 38 (104 us)

12356565 - 45876 = 12310689 (131 us)

1.2345679e + 08 - 1.2345679e + 08 = 0 (56.6us)

125654 - 988656 = -863002 (103 us)

0.5 - 0.50000006 = -5.9604645e - 08 (335 us)

7.3 FLOATING POINT MULTIPLICATION (LMLT)

0 x 0 = 0 (16.9 us)

1 x 0 = 0 (16.9 us)

0 x 1 = 0 (22.2 us)

1.701411e + 38 x 0.1 = 1.701411e + 37 (132 us)

6.5436653e + 08 x 12345 = 8.0781548e + 12 (138 us)

1.2345679e + 08 x 1.2345679e + 08 = 1.5241579e + 16 (132 us)

1e + 30 x 0 = 0 (16.9 us)

1e + 30 x 1e - 10 = 1e + 20 (132 us)

1.234e + 20 x 2.34e + 02 = 2.8875600e + 22 (132 us)

5.1042355e + 38 x 1.5 = Abnormal termination (121 us)

2.5521178e + 38 x 1.5 = 3.8281766e + 38 (132 us)

1 x 1.1754944e - 38 = 1.1754944e - 38 (136 us)

2 x 2 = 4 (138 us)

7.4 FLOATING POINT DIVISION (LDIV)

0 ÷ 1 = 0 (16.9 us)

1.701411e + 38 ÷ 2 = 8.5070551e + 37 (602 us)

1 ÷ 0 = Abnormal termination (11.7 us)

9.9999997e + 37 ÷ 1e + 08 = 9.9999994e + 29 (514 us)

12 ÷ 21 = 0.57142854 (524 us)

1.1754944e - 38 ÷ 2 = 0 (25.1 us)

3.4028237e + 38 ÷ 0.25 = Abnormal termination (21.7 us)

(-2.3509887e - 38) ÷ 2 = -1.1754944e - 38 (479 us)

1   8.5070592e + 37 = 1.1754944e - 38 (479 us)

1.9999999 ÷ 1 = 1.9999999 (620 us)

7.5 sin FUNCTION (LSIN)

	μPD78P014		PC-9801
sin(3.1415927)	−8.7544322e−08	(1.34 ms)	−8.7422780e−08
sin(1.5707964)	0.99999995	(3.70 ms)	1
sin(100)	−0.50636563	(3.34 ms)	−0.50636564
sin(9999999)	0.98960368	(4.01 ms)	0.99066465
sin(−100)	0.50636563	(3.34 ms)	0.50636564
sin(0)	0	(204 μs)	0
sin(0.2)	0.19866933	(2.18 ms)	0.19866933
sin(−32.967228)	−0.99980993	(3.19 ms)	−0.99980998
sin(33.333332)	0.94053001	(3.37 ms)	0.94053001
sin(6.2831593)	−2.6050024e−05	(1.41 ms)	−2.6051198e−05
sin(9.424778)	−2.6077032e−08	(1.38 ms)	−2.3849761e−08
sin(6.8056469e+38)	0.93973852	(8.01 ms)	Error due to dropped digits

7.6 cos FUNCTION (LCOS)

	μPD78P014		PC-9801
cos(0)	0.99999995	(2.69 ms)	1
cos(3.1415927)	−0.99999995	(3.68 ms)	−1
cos(1.5707964)	−4.3772161e−08	(1.35 ms)	−4.3711390e−08
cos(−10000)	−0.95215377	(3.40 ms)	−0.95215537
cos(8.3775806)	−0.5000002	(3.30 ms)	−0.5000002
cos(−9424.7783)	0.99999988	(3.57 ms)	0.99999994
cos(162.31561)	0.49999341	(3.25 ms)	0.49999338
cos(6.8056469e+38)	0.34189401	(7.80 ms)	Error due to dropped digits
cos(4.712389)	1.3038516e−08	(1.39 ms)	1.1924880e−08

7.7 tan FUNCTION (LTAN)

	μPD78P014		PC-9801
tan(0)	0	(2.94 ms)	0
tan(3.1415927)	8.7544322e−08	(4.76 ms)	8.7422780e−08
tan(1.5707964)	−22845568	(4.81 ms)	−22877332
tan(−1.0471976)	−1.7320508	(6.08 ms)	−1.7320509
tan(2.0999999)	−1.7098470	(6.71 ms)	−1.7098469
tan(1000)	1.4703251	(6.71 ms)	1.4703242
tan(500)	0.52924407	(6.69 ms)	0.52924386
tan(157.07964)	2.9802322e−06	(4.69 ms)	2.9406275e−06
tan(6.8056469e+38)	2.7486253	(11.04 ms)	Error due to dropped digits
tan(4.712389)	−7.6695840e+07	(4.87 ms)	−8.3858283e+07

7.8 NATURAL LOGARITHM FUNCTION (LLOG)

	μPD78P014		PC-9801
log(2.7182817)	0.99999996	(3.41 ms)	0.99999997
log(9.9999996e+35)	82.893063	(3.07 ms)	82.893063
log(1)	0	(495 μs)	0
log(0)	Abnormal termination	(11.7 μs)	Illegal argument
log(−0.1)	Abnormal termination	(10.3 μs)	Illegal argument
log(12345.679)	9.4210614	(3.66 ms)	9.4210614
log(59874.141)	11	(2.87 ms)	11
log(20.085537)	3	(3.29 ms)	3
log(4.5399931e−05)	−9.9999999	(3.57 ms)	−10
log(6.8056469e+38)	89.415986	(2.06 ms)	89.415986
log(1.1754944e−38)	−87.336545	(632 μs)	−87.336545
log(1.4397301e−25)	−57.200172	(3.66 ms)	−57.200172

7.9 COMMON LOGARITHM FUNCTION (LLOG10)

	μPD78P014		PC-9801
$\log_{10}(0)$	Abnormal termination	(16.2 μs)	Illegal argument
$\log_{10}(-1)$	Abnormal termination	(14.8 μs)	Illegal argument
$\log_{10}(1)$	0	(538 μs)	0
$\log_{10}(10)$	0.99999999	(3.43 ms)	1
$\log_{10}(1.2345679e+08)$	8.091515	(2.67 ms)	8.091515
$\log_{10}(9.8765434e+08)$	8.994605	(2.67 ms)	8.994605
$\log_{10}(0.44400001)$	-0.35261702	(3.15 ms)	-0.35261702
$\log_{10}(100000)$	5	(3.53 ms)	5
$\log_{10}(6.8056469e+38)$	38.832869	(2.21 ms)	38.832869
$\log_{10}(1.1754944e-38)$	-37.929779	(784 μs)	-37.929779
$\log_{10}(2.4001264e-18)$	-17.619766	(3.80 ms)	-17.619766

7.10 EXPONENT FUNCTION (BASE = e) (LEXP)

	μPD78P014		PC-9801
$e^{(0)}$	1	(342 μs)	1
$e^{(1)}$	2.7182818	(3.67 ms)	2.7182818
$e^{(-1)}$	0.36787944	(3.87 ms)	0.36787944
$e^{(0.98765433)}$	2.6849291	(3.67 ms)	2.6849291
$e^{(20)}$	4.8516519e+08	(3.85 ms)	4.851652e+08
$e^{(11)}$	59874.142	(3.88 ms)	59874.142
$e^{(89.415993)}$	Abnormal termination	(214 μs)	Overflow
$e^{(89.415985)}$	6.8056386+38	(2.11 ms)	6.8056393+38
$e^{(-87.336548)}$	0	(2.09 ms)	1.1754907e-38
$e^{(-87.33654)}$	1.1754998e-38	(1.92 ms)	1.1754997e-38
$e^{(-0.82129019)}$	0.43986378	(4.08 ms)	0.43986378

7.11 EXPONENT FUNCTION (BASE = 10) (LEXP10)

	μPD78P014		PC-9801
$10^{(38.832863)}$	6.8055425e+38	(2.16 ms)	6.8055441e+38
$10^{(-37.929775)}$	1.1755061e−38	(2.06 ms)	1.1755058e−38
$10^{(89.415993)}$	Abnormal termination	(415 μs)	Overflow
$10^{(-87.336548)}$	0	(424 μs)	4.60736e−88
$10^{(0)}$	1	(387 μs)	1
$10^{(0.56666666)}$	3.686945	(3.68 ms)	3.686945
$10^{(0.96666664)}$	9.2611866	(4.01 ms)	9.2611867
$10^{(1.7333332)}$	54.116939	(4.13 ms)	54.11694
$10^{(0.34659675)}$	2.2212464	(3.84 ms)	2.2212465
$10^{(-0.35975304)}$	0.43676412	(4.24 ms)	0.43676412
$10^{(38.83287)}$	Abnormal termination	(374 μs)	Overflow

7.12 POWER FUNCTION (LPOW)

	μPD78P014		PC-9801
$(0)^{(0)}$	Abnormal termination	(17.9 μs)	Overflow
$(0)^{(1)}$	0	(20.3 μs)	0
$(1)^{(0)}$	1	(892 μs)	1
$(1)^{(1)}$	1	(898 μs)	1
$(1)^{(-1)}$	1	(898 μs)	1
$(2)^{(-2)}$	0.25	(2.70 ms)	0.25
$(50)^{(2)}$	2500	(7.55 ms)	2500
$(2050)^{(2)}$	4202499.9	(4.24 ms)	4202500
$(-1)^{(2.5669999)}$	Abnormal termination	(41.8 μs)	Illegal argument
$(0)^{(-9.8765001)}$	Abnormal termination	(20.3 μs)	Overflow
$(1.3038405e+19)^{(2)}$	1.6999996e+38	(6.21 ms)	1.7e+38
$(9.876543)^{(1.2345679)}$	16.900803	(6.91 ms)	16.900803
$(9)^{(1.2345001)}$	15.066501	(6.48 ms)	15.066502
$(2.1900001)^{(-9.1199999)}$	7.8550622e−04	(6.90 ms)	7.8550618e−04
$(4)^{(6.8056469e+38)}$	Abnormal termination	(819 μs)	Overflow
$(1.50487e+12)^{(-0.20180109)}$	3.4879273e−03	(7.84 ms)	3.4879272e−03
$(2.7182817)^{(89.415985)}$	6.8056125e+38	(5.63 ms)	6.8056208e+38

7.13 SQUARE ROOT FUNCTION (LSQRT)

	μPD78P014		PC-9801
$\sqrt{(0)}$	0	(11.7 μs)	0
$\sqrt{(1)}$	1	(1.86 ms)	1
$\sqrt{(2)}$	1.4142135	(1.96 ms)	1.4142136
$\sqrt{(121)}$	11	(1.99 ms)	11
$\sqrt{(2500)}$	50	(1.98 ms)	50
$\sqrt{(1e-06)}$	9.9999993e−04	(2.03 ms)	1e−03
$\sqrt{(-9.9999998e-03)}$	Abnormal termination	(11.2 μs)	Illegal argument
$\sqrt{(30.863079)}$	5.5554547	(1.99 ms)	5.5554549
$\sqrt{(11.111111)}$	3.3333333	(1.94 ms)	3.3333333
$\sqrt{(5.1001101e-35)}$	7.1415052e−18	(2.08 ms)	7.1415055e−18

7.14 arcsin FUNCTION (LASIN)

	μPD78P014		PC-9801
arcsin(0)	0	(2.28 ms)	0
arcsin(1)	1.5707963	(45.8 μs)	1.5707963
arcsin(−1)	−1.5707963	(46.8 μs)	−1.5707963
arcsin(−0.5)	−0.52359877	(5.70 ms)	−0.52359878
arcsin(3.1415927)	Abnormal termination	(21.0 μs)	Illegal argument
arcsin(0.78539819)	0.90333916	(6.35 ms)	0.90333915
arcsin(−0.86602539)	−1.0471976	(6.31 ms)	−1.0471975
arcsin(0.98437494)	1.3937883	(6.67 ms)	1.3937883
arcsin(0.99999994)	1.5704504	(5.16 ms)	1.5704511

7.15 arccos FUNCTION (LACOS)

	μPD78P014		PC-9801
arccos(0)	1.5707963	(2.34 ms)	1.5707963
arccos(1)	0	(121 μs)	0
arccos(−1)	3.1415927	(132 μs)	3.1415927
arccos(0.52359879)	1.0197267	(5.25 ms)	1.0197267
arccos(−0.5)	2.0943951	(5.81 ms)	2.0943951
arccos(−0.86602539)	2.6179939	(6.40 ms)	2.6179938
arccos(0.1)	1.4706289	(4.92 ms)	1.4706289
arccos(−0.1)	1.6709638	(4.91 ms)	1.6709637
arccos(0.98437494)	0.17700801	(6.79 ms)	0.17700802
arccos(0.99999994)	3.4594024e−04	(5.38 ms)	3.4526698e−04

7.16 arctan FUNCTION (LATAN)

	μPD78P014		PC-9801
arctan(0)	0	(289 μs)	0
arctan(1)	0.78539817	(3.70 ms)	0.78539816
arctan(−1)	−0.78539817	(3.70 ms)	−0.78539816
arctan(3.1415927)	1.2626273	(3.80 ms)	1.2626273
arctan(1.5707964)	1.0038848	(3.04 ms)	1.0038848
arctan(1.7014110e+38)	1.5707963	(838 μs)	1.5707963
arctan(10000000)	1.5707962	(1.60 ms)	1.5707962
arctan(0.001)	9.9999971e−04	(1.36 ms)	9.9999971e−04
arctan(10)	1.4711277	(2.65 ms)	1.4711277
arctan(−10)	−1.4711277	(2.65 ms)	−1.4711277
arctan(1.0000001)	0.78539823	(3.84 ms)	0.78539822

7.17 sinh FUNCTION (LHSIN)

	μPD78P014		PC-9801
sinh(89.415993)	Abnormal termination	(225 μs)	3.4028456e+38
sinh(−89.415993)	Abnormal termination	(225 μs)	−3.4028456e+38
sinh(89.415985)	3.4028192e+38	(2.23 ms)	3.4028196e+38
sinh(−89.415985)	−3.4028192e+38	(2.23 ms)	−3.4028196e+38
sinh(0)	0	(187 μs)	0
sinh(0.4998779)	0.52095762	(1.65 ms)	0.52095763
sinh(0.125)	0.12532578	(1.86 ms)	0.12532578
sinh(1.0842022e−19)	1.0842022e−19	(317 μs)	1.0842022e−19
sinh(0.76666665)	0.84400989	(4.10 ms)	0.84400998
sinh(1.0666666)	1.2807617	(4.50 ms)	1.2807619
sinh(1.8666666)	3.1560328	(4.62 ms)	3.1560329
sinh(3.351413)	14.254	(4.78 ms)	14.254001

7.18 cosh FUNCTION (LHCOS)

	μPD78P014		PC-9801
cosh(−89.415993)	Abnormal termination	(220 μs)	3.4028456e+38
cosh(−89.415985)	3.4028192e+38	(2.22 ms)	3.4028196e+38
cosh(0)	1	(935 μs)	1
cosh(1.0842022e−19)	1	(1.31 ms)	1
cosh(0.76666665)	1.3085689	(4.07 ms)	1.308569
cosh(1.0666666)	1.6249156	(4.48 ms)	1.6249157
cosh(1.8666666)	3.3106711	(4.59 ms)	3.3106712
cosh(4.0319099)	28.193103	(4.77 ms)	28.193105

7.19 tanh FUNCTION (LHTAN)

	μPD78P014		PC-9801
tanh(89.415993)	1.0000001	(254 μs)	1
tanh(−89.415993)	−1.0000001	(255 μs)	−1
tanh(0)	0	(1.18 ms)	0
tanh(0.4998779)	0.46202111	(6.63 ms)	0.46202113
tanh(0.125)	0.124353	(6.84 ms)	0.124353
tanh(1.0842022e−19)	1.0842022e−19	(2.14 ms)	1.0842022e−19
tanh(0.76666665)	0.64498699	(8.74 ms)	0.64498699
tanh(1.0666666)	0.78820205	(9.53 ms)	0.78820205
tanh(1.8666666)	0.953291	(9.74 ms)	0.95329096
tanh(9.4484739)	0.99999988	(10.02 ms)	0.99999999
tanh(7.8125e−03)	7.8123417e−03	(4.92 ms)	7.812341le−03

7.20 ABSOLUTE VALUE FUNCTION (LABS)

| 0 | = 0 (6.4 us)
| -2.1290744e - 19 | = 2.1290744e - 19 (6.4 us)
| 1.6415355e + 27 | = 1.6415355e + 27 (6.4 us)

7.21 RECIPROCAL FUNCTION (LRCPN)

1/0 = Abnormal termination (38.9 us)
1/ (-1.1754944e - 38) = -8.5070592e + 37 (505 us)
1/0.99999994 = 1 (490 us)
1/ (-1.0000001) = -0.99999988 (636 us)
1/ (8.5070602e + 37) = 0 (637 us)
1/ (8.5070592e + 37) = 1.1754944e - 38 (506 us)
1/ (2.8545976e - 18) = 3.5031207e + 17 (541 us)
1/ (-2.9092885e + 21) = -3.4372664e - 22 (549 us)

7.22 POLAR COORDINATE → RECTANGULAR COORDINATE CONVERSION FUNCTION (POTORA)

(r, θ)	(x, y)
(1, 1.5707964)	(−4.3772161e−08, 0.99999995) (4.63 ms) (−4.3711390e−08, 1)
(1, 0.52359879)	(0.8660254, 0.50000001) (5.91 ms) (0.8660254, 0.50000001)
(7, 2.6179938)	(−6.0621777, 3.5000003) (6.48 ms) (−6.0621777, 3.5000003)
(99, −2.0943952)	(−49.500005, −85.736512) (6.51 ms) (−49.500005, −85.736512)
(8.8888798, 2.3561945)	(−6.2853872, 6.2853871) (6.74 ms) (−6.2853872, 6.2853871)
(4.4443998, 3.1415927)	(−4.4443996, −3.8908197e−07) (4.61 ms) (−4.4443998, −3.8854179e−07)
(0.5, 6.8056469e+38)	(0.17094701, 0.46986926) (10.84 ms) Error due to dropped digits
(0.5, 4.712389)	(6.5192580e−09, −0.49999997) (4.68 ms) (5.9624402e−09, −0.5)
(0.5, 6.283185)	(0.49999997, −1.5040860e−07) (4.48 ms) (0.5, −1.5099580e−07)

Remarks : The upper figures are the operation results for the uPD78P014, and the lower are those for the PC-9801.

7.23 RECTANGULAR COORDINATE → POLAR COORDINATE CONVERSION FUNCTION (RATOPO)

(x, y)	(r, θ)	
(0, 0)	(0, 0) (0, 0)	(17.9 μs)
(1, 1)	(1.4142135, 0.78539813) (1.4142136, 0.78539816)	(6.60 ms)
(0, 1)	(1, 1.5707963) (1, 1.5707963)	(2.19 ms)
(1, −1)	(1.4142135, −0.78539813) (1.4142136, −0.78539816)	(6.60 ms)
(−1, 1)	(1.4142135, 2.3561943) (1.4142136, 2.3561945)	(6.71 ms)
(−1, −1)	(1.4142135, −2.3561943) (1.4142136, −2.3561945)	(6.71 ms)
(0, −1)	(1, −1.5707963) (1, −1.5707963)	(2.19 ms)
(1, 0)	(1, 0) (1, 0)	(2.49 ms)
(−1, 0)	(1, 3.1415925) (1, 3.1415927)	(2.54 ms)
(11111, 11111)	(15713.326, 0.78539813) (15713.327, 0.78539816)	(6.58 ms)
(−12.220954, 69.662003)	(70.725845, 1.7444612) (70.725853, 1.7444613)	(6.98 ms)
(0.92719781, 0.23236816)	(0.95587164, 0.24555582) (0.95587172, 0.24555586)	(5.90 ms)

Remarks : The upper figures are the operation results for the uPD78P014, and the lower are those for the PC-9801.

7.24 CHARACTER STRING → FLOATING POINT FORMAT CONVERSION FUNCTION (ATOL)

"1234567.8901234567890123456 78"	= Abnormal termination	(1.11ms)
"0Q"	= 0	(126 us)
"E12"	= Abnormal termination	(78.8 us)
"1e"	= Abnormal termination	(562 us)
"1E + 123"	= Abnormal termination	(581 us)
"1.17549427E - 38"	= 0	(5.39 ms)
"1.17549428E - 38"	= 1.1754944e - 38	(5.39 ms)
"6.8056476E + 38"	= Abnormal termination	(4.43 ms)
"6.8056475E + 38"	= 6.8056473e + 38	(4.44 ms)
"655361"	= 655361	(1.20 ms)
"1e - 20"	= 1e - 20	(5.18 ms)
"12345678901234567890123456 7E - 32"	= 1.2345679e - 06	(5.62 ms)
"1.0000000000000000000000000000E - 9	= 1e - 09	(5.88 ms)
"+1.2030646E + 22"	= 1.2030646e + 22	(5.42 ms)
"-4.6231684E - 18"	= -4.6231685e - 18	(4.95 ms)
"0.00000000000000000117549428E - 20	= 1.1754944e - 38	(6.39 ms)

7.25 FLOATING POINT FORMAT → CHARACTER STRING CONVERSION FUNCTION (LTOA)

0	=	"0"	(18.4 us)
1.0000002e - 37	=	"1.000000e - 37"	(8.32 ms)
9.9999999e - 38	=	"9.999998e - 38"	(9.04 ms)
1.0000001e - 05	=	"1.000000e - 05"	(9.73 ms)
1	=	"1.000000e00"	(1.91 ms)
100000	=	"1.000000e05"	(9.61 ms)
9.9999993e + 19	=	"1.000000e20"	(9.50 ms)
9.9999987e + 37	=	"9.999999e37"	(9.28 ms)
1.0000001e + 38	=	"1.000000e38"	(8.85 ms)
6.8056469e + 38	=	"6.805646e38"	(4.81 ms)
-6.8056469e + 38	=	"-6.805646e38"	(4.81 ms)
-1.2677555e + 13	=	"-1.267755e13"	(10.76 ms)
-1.1907760e - 29	=	"-1.190775e- 29"	(9.02 ms)

7.26 2-BYTE INTEGER TYPE → FLOATING POINT FORMAT CONVERSION FUNCTION (FTOL)

0	(12.2 us)
-1	(72.3 us)
-32768	(26.5 us)
1	(68.5 us)
511	(66.6 us)
-511	(70.4 us)
255	(28.4 us)
32767	(32.2 us)

7.27 FLOATING POINT FORMAT → 2-BYTE INTEGER TYPE CONVERSION FUNCTION (LTOF)

-0.99999994	=	0 (Z flag = 0)	(14.6 us)
0	=	0 (Z flag = 1)	(14.6 us)
65536	=	Abnormal termination	(14.6 us)
-32769	=	Abnormal termination	(35.1 us)
-32768	=	-32768 (Z flag = 1)	(37.5 us)
32767.5	=	32767 (Z flag = 0)	(38.9 us)
1	=	1 (Z flag = 1)	(83.3 us)
1.5	=	1 (Z flag = 0)	(83.3 us)
-1	=	-1 (Z flag = 1)	(88.5 us)

CHAPTER 8. PROGRAM LISTINGS

(1) EQU.INC

```
$       NOLIST
;****************************************************
;*
;*  78K0 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
;*****************************************************
;*
;*  78K0 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   FPRE_XP
        EXTRN   FPR1_X,FPR2_X
        EXTRN   FPR3_X,FPR4_X,FPR5_X
$       LIST
```

(3) REF2.INC

```
$       NOLIST
;*************************************************************
;*
;*  78K0 FLOATING POINT REGISTER LOAD FUNCTION REFFERENCE
;*
;*
;*
;*************************************************************
        EXTRN   LLD21,LLD21X
        EXTRN   LLD31,LLD31X
        EXTRN   LLD41,LLD41X
        EXTRN   LLD51,LLD51X
        EXTRN   LLD32
        EXTRN   LLD52
        EXTRN   LLD13
        EXTRN   LLD23,LLD23X
        EXTRN   LLD24,LLD24X
        EXTRN   LLD15
        EXTRN   LLD25,LLD25X
        EXTRN   LLD1C,LLD1CX
        EXTRN   LLD2C,LLD2CX
        EXTRN   LXC13,LXC13X
        EXTRN   LXC14,LXC14X
        EXTRN   LXC15,LXC15X
$       LIST
```

(4) ASCII.INC

```
$       NOLIST
;*****************************************************************
;
; 78K0 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
;****************************************************
;*
;*  78K0 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  FPRE_XP
        PUBLIC  FPR1_X,FPR2_X

        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

;******** 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 **
FPRE_XP:
FPR1_X:
        DS      1
FPR2_X:
        DS      1

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
        PUBLIC  LNOR

        CSEG
;***************************************************
;*
;*  78K0 FLOATING POINT ADDITION FUNCTION
;*
;*    DESTINATION REGISTER: FPR1
;*    SOURCE REGISTER     : FPR2
;*
;*    RESULT : FPR1 += FPR2
;*             ERROR then set CY
;*
;***************************************************

LADD:
        FPRE_XP = #0                ;clear 4th mantissa
LADDX:
        CY  = FPR2_3.7
        A = FPR2_4
        ADDC A,A
        if_bit (Z)
            goto T_RET
        endif
        D = A                       ;FPR2 exponent

        CY  = FPR1_3.7
        A = FPR1_4
        ROLC A,1
        C = A                       ;FPR1 exponent

;******* CHECK EXPONENT & LOAD **
                                    ;d : FPR1·FPR1_X <- one of higher exp.
                                    ;s : A·DE·C <- mantissa (lower exp. one)
                                    ;    FPR2_X <- difference of exp.
                                    ;    FPR2_4.7 <- sign (lower exp. one)
```

```
A -= D                          ;difference of exp.

if_bit (!CY)
  A <-> FPR2_X
  A <-> C
  A <-> FPR2_1
  E = A
  D = FPR2_2 (A)
  A = FPR2_3

else
  A ^= #0FFH
  A++                           ;|difference of exp.|
  A <-> FPR2_X
  A <-> FPR1_X
  A <-> C
  A <-> D
  A <-> FPR2_1
  A <-> FPR1_1
  E = A
  A = FPR2_4
  A <-> FPR1_4
  FPR2_4 = A
  A = FPR2_2
  A <-> FPR1_2
  A <-> D
  A <-> FPR2_3
  A <-> FPR1_3

  if (FPR2_3 == #0)             ;exp. of destination == 0?
    goto T_RET
  endif

endif

if (FPR2_X >= #SHORT*BYTE)
  goto T_RET                    ;neglect lower value
endif

FPR1_3 |= #80H                  ;(set mantissa MSB)
A |= #80H                       ;(set mantissa MSB)
```

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

        if (FPR2_X != #0)       ;exp. agree?
          repeat
            CLR1 CY
            RORC A,1
            A <-> D
            RORC A,1
            A <-> E
            RORC A,1
            A <-> C
            RORC A,1
            A <-> C
            A <-> E
            A <-> D             ;s':A·DE·C <- mantissa be agreed potential
            FPR2_X--
          until_bit (Z)
        endif

;******* CALC. MANTISSA(set result to A·C·DE) **

        CY = FPR2_4.7
        CY ^=  FPR1_4.7

        if_bit (!CY)            ;sign agree?
          A <-> C               ;s' += d
          ADD A,FPR1_X
          A <-> E
          ADDC A,FPR1_1
          A <-> D
          ADDC A,FPR1_2
          A <-> C
          ADDC A,FPR1_3

          if_bit (CY)
            FPR2_1++            ;normalize mantissa overflow
            if_bit (Z)
              goto ERROR
            endif
            RORC A,1
            A <-> C
            RORC A,1
            A <-> D
            RORC A,1
            A <-> E
            RORC A,1
            A <-> E
            A <-> D
            A <-> C
          endif
```

```
else
  X = A
  if (A == FPR1_3)
    if (D == FPR1_2) (A)
      if (E == FPR1_1) (A)
        if (C == FPR1_X) (A)    ;if(s' == d)
          goto ZERO
        endif
      endif
    endif
  endif

  if_bit (!CY)          ;if(s' > d)
    FPR1_4 ^= #80H      ;turn sign bit
    A = X
  else
    A = C
    A <-> FPR1_X
    C = A
    A = E
    A <-> FPR1_1
    E = A
    A = D
    A <-> FPR1_2
    D = A
    A = X
    A <-> FPR1_3
  endif

  A <-> C               ;|s' -= d|
  SUB A,FPR1_X
  A <-> E
  SUBC A,FPR1_1
  A <-> D
  SUBC A,FPR1_2
  A <-> C
  SUBC A,FPR1_3
```

```
LNOR:
          while_bit (!A.7)      ;normalize catastrophic cancellation
            FPR2_1--
            if_bit (Z)
              goto ZERO
            endif
            A <-> E
            ROLC A,1
            A <-> D
            ROLC A,1
            A <-> C
            ROLC A,1
            A <-> E
            ROLC A,1
            A <-> E
            A <-> C
            A <-> D
            A <-> E
          endw
        endif

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

        FPR1_3 = A              ;1st mantissa
        A = FPR2_1
T_STOR:
        CY = FPR1_4.7
        RORC A,1
        FPR1_4 = A              ;sign,exponent
        FPR1_3.7 = CY           ;exponent LSB
        FPR1_2 = C (A)          ;2nd mantissa
        FPR1_1 = D (A)          ;3rd mantissa
        FPR1_X = E (A)          ;4th mantissa

T_RET:
        A = #R_OK
        CLR1 CY
        RET
ZERO:
        FPR1_HP = #0
        goto T_RET
ERROR:
        A = #R_ERR
        SET1 CY
        RET
```

```
;************************************************
;*
;*  78K0 FLOATING POINT SUBTRACTION FUNCTION
;*
;*    DESTINATION REGISTER: FPR1
;*    SOURCE REGISTER     : FPR2
;*
;*    RESULT : FPR1 -= FPR2
;*             ERROR then set CY
;*
;************************************************

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


;************************************************
;*
;*  78K0 FLOATING POINT MULTIPLICATION FUNCTION
;*
;*    DESTINATION REGISTER: FPR1
;*    SOURCE REGISTER     : FPR2
;*
;*    RESULT : FPR1 *= FPR2
;*             ERROR then set CY
;*
;************************************************

LMLT:
        FPRE_XP = #0            ;clear 4th mantissa

;******* ZERO EXCEPTION **
LMLTX:
        CY = FPR2_3.7
        A = FPR2_4
        ADDC A,A
        if_bit (Z)
          goto ZERO
        endif
        C = A                   ;FPR2 exp.

        CY  = FPR1_3.7
        A = FPR1_4
        ADDC A,A                ;FPR1 exp.
        if_bit (Z)
          goto ZERO
        endif
```

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

        A += C
        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

        A <-> FPR2_4            ;FPR2_4 <- exp.
        A ^= FPR1_4
        FPR1_4 = A              ;FPR1_4.7 <- sign

;******* CALC.MANTISSA (set result to A·C·DE) **
                                ;d: FPR1 mantissa
                                ;s: FPR2 mantissa

        SET1 FPR1_3.7           ;(set mantissa MSB)
        SET1 FPR2_3.7           ;(set mantissa MSB)

        X = FPR1_X (A)
        A = FPR2_3
        MULU X                  ;d(0) * s(3)

        A <-> FPR1_1
        X = A
        E = A
        A = FPR2_2
        MULU X                  ;d(1) * s(2)
        A += FPR1_1             ;->CY

        A <-> E
        X = A
        A = FPR2_3
        MULU X                  ;d(1) * s(3)
        ADDC A,#0               ;<-CY
        A <-> X
        E += A                  ;->CY
        A = X
        ADDC A,#0               ;<-CY
```

```
A <-> FPR1_2
X = A
D = A
A = FPR2_1
MULU X                  ;d(2) * s(1)
E += A                  ;->CY

X = D (A)
A = FPR2_2
MULU X                  ;d(2) * s(2)
ADDC A,#0               ;<-CY
A <-> X
E += A                  ;->CY
A = X
ADDC A,FPR1_2           ;<-CY, ->CY

A <-> D
X = A
A = FPR2_3
MULU X                  ;d(2) * s(3)
ADDC A,#0               ;<-CY
A <-> X
D += A                  ;->CY
A = X
ADDC A,#0               ;<-CY

A <-> FPR1_3
X = A
C = A
A = FPR2_X
MULU X                  ;d(3) * s(0)
E += A                  ;->CY

X = C (A)
A = FPR2_1
MULU X                  ;d(3) * s(1)
ADDC A,#0               ;<-CY
A <-> X
E += A                  ;->CY
A = X
ADDC D,A                ;<-CY,->CY

X = C (A)
A = FPR2_2
MULU X                  ;d(3) * s(2)
ADDC A,#0               ;<-CY
A <-> X
D += A                  ;->CY
A = X
ADDC A,FPR1_3           ;<-CY,->CY
```

```
        A <-> C
        X = A
        A = FPR2_3
        MULU X                  ;d(3) * s(3)
        ADDC A,#0               ;<-CY
        A <-> X
        C += A                  ;->CY
        A = X
        ADDC A,#0               ;<-CY

;******* NORMALIZE **

        if_bit (A.7)
          FPR2_4++              ;2 <= mantissa < 4
          if_bit (Z)
            goto ERROR          ;exp. = 100H
          endif
        else
          if (FPR2_4 == #0)     ;1 <= mantissa < 2
            goto ZERO
          endif
          CLR1 CY
          A <-> E
          ROLC A,1
          A <-> D
          ROLC A,1
          A <-> C
          ROLC A,1
          A <-> E
          ROLC A,1
          A <-> D
          A <-> E
          A <-> C
          A <-> D
        endif

        FPR1_3 = A              ;1st mantissa
        A = FPR2_4

        goto T_STOR
```

```
;***************************************************
;*
;*  78K0 FLOATING POINT DIVISION FUNCTION
;*
;*    DESTINATION REGISTER: FPR1
;*    SOURCE REGISTER     : FPR2
;*
;*    RESULT : FPR1 /= FPR2
;*             ERROR then set CY
;*
;***************************************************

LDIV:

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

        CY = FPR2_3.7
        A = FPR2_4
        ADDC A,A
        if_bit (Z)
          goto ERROR
        endif
        B = A                   ;FPR2 exp.

        CY  = FPR1_3.7
        A = FPR1_4
        ADDC A,A                ;FPR1 exp.
        if_bit (Z)
          goto T_RET
        endif

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

        A -= B

        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
```

```
        A <-> FPR2_4            ;STORE:FPR2_4 <- (exp.-1)
        A ^=  FPR1_4
        FPR1_4 = A              ;FPR1_4.7 <- sign

;******* LOAD MANTISSA **
                                ;d: CY·E·HL <- FPR1 mantissa
                                ;s: FPR2_3·FPR2_LP
        B = #(SHORT-1)*BYTE+1   ;loop counter
        HL = FPR1_LP (AX)
        A = FPR1_3
        A |= #80H               ;(set mantissa MSB)
        E = A
        CLR1 CY

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

;******* DIVIDE MANTISSA (set quotient to CY·X·C·D) **

        goto T_DIV1

        repeat
          A = L                 ;d * 2
          ADD L,A
          A = H
          ADDC H,A
          A = E
          ADDC E,A
T_DIV1:
          if_bit (!CY)
            if (E == FPR2_3) (A)
              if (H == FPR2_2) (A)
                A = L
                CMP A,FPR2_1
              endif
            endif
            NOT1 CY
          endif

          if_bit (CY)           ;if(d >= s)
            A = L               ;d -= s
            SUB A,FPR2_1
            L = A
            A = H
            SUBC A,FPR2_2
            H = A
            A = E
            SUBC A,FPR2_3
            E = A
            SET1 CY             ;quotient digit
          endif
```

```
    A = D                   ;shift in quotient digit
    ADDC D,A
    A = C
    ADDC C,A
    A = X
    ADDC X,A
    B--
  until_bit(Z)

;******* NORMALIZE **

  if_bit (CY)               ;1 <= mantissa < 2
    FPR2_4++
    if_bit (Z)
      goto ERROR
    endif
    A = X
    RORC A,1
    X = A
    A = C
    RORC A,1
    C = A
    A = D
    RORC A,1
    D = A
  endif

  FPR1_3 = X (A)            ;1st mantissa
  E = #0                    ;4th mantissa
  A = FPR2_4

  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
;*****************************************************************
;*
;*  78K0 FLOATING POINT FUNCTION THAT
;*
;*     CALC. A POLYNOMIAL EXPRESSION by EXTENDED FORMAT
;*
;*                                    2                  n
;*  polynomial.1: x + k1xy + k1k2xy  + ... + k1k2..knxy
;*
;*     input  conditions:
;*        FPR1・FPR1_X <- x , FPR4・FPR4_X <- y
;*        HL <- head address of coefficient array (k1,,kn)
;*        B  <- n
;*
;*                                    2                  n
;*  polynomial.2: z + k1xy + k1k2xy  + ... + k1k2..knxy
;*
;*     input  conditions:
;*        FPR3・FPR3_X <- x , FPR4・FPR4_X <- y, FPR1・FPR1_X <- z
;*        HL <- head address of coefficient array (k1,,kn)
;*        B  <- n
;*
;*     output conditions(common to both):
;*        FPR1・FPR1_X <- result of polynomial expression
;*        FPR4・FPR4_X : 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

          CY = FPR3_3.7
          A = FPR3_4
          ADDC A,A
          if_bit (Z)
            RET
          endif
          C = A

          CY = FPR1_3.7
          A = FPR1_4
          ROLC A,1

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

          B--
        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  LLD31,LLD31X
        PUBLIC  LLD41,LLD41X
        PUBLIC  LLD51,LLD51X
        PUBLIC  LLD32
        PUBLIC  LLD52
        PUBLIC  LLD13
        PUBLIC  LLD23,LLD23X
        PUBLIC  LLD24,LLD24X
        PUBLIC  LLD15
        PUBLIC  LLD25,LLD25X
        PUBLIC  LLD1C,LLD1CX
        PUBLIC  LLD2C,LLD2CX
        PUBLIC  LXC13,LXC13X
        PUBLIC  LXC14,LXC14X
        PUBLIC  LXC15,LXC15X

        CSEG
;*****************************************************
;*
;*  78K0 FLOATING POINT REGISTER LOAD FUNCTIONS
;*
;*
;*
;*****************************************************

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

LLD21X:
        FPR2_X  = FPR1_X (A)
LLD21:
        FPR2_LP = FPR1_LP (AX)
        FPR2_HP = FPR1_HP (AX)
        RET
```

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

LLD31X:
        FPR3_X  = FPR1_X (A)
LLD31:
        FPR3_LP = FPR1_LP (AX)
        FPR3_HP = FPR1_HP (AX)
        RET

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

LLD41X:
        FPR4_X  = FPR1_X (A)
LLD41:
        FPR4_LP = FPR1_LP (AX)
        FPR4_HP = FPR1_HP (AX)
        RET

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

LLD51X:
        FPR5_X  = FPR1_X (A)
LLD51:
        FPR5_LP = FPR1_LP (AX)
        FPR5_HP = FPR1_HP (AX)
        RET

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

LLD32:
        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

LLD13:
        FPR1_LP = FPR3_LP (AX)
        FPR1_HP = FPR3_HP (AX)
        RET
```

```
;******* LOAD FPR2.FPR3

LLD23X:
        FPR2_X  = FPR3_X (A)
LLD23:
        FPR2_LP = FPR3_LP (AX)
        FPR2_HP = FPR3_HP (AX)
        RET

;******* LOAD FPR2.FPR4

LLD24X:
        FPR2_X  = FPR4_X (A)
LLD24:
        FPR2_LP = FPR4_LP (AX)
        FPR2_HP = FPR4_HP (AX)
        RET

;******* LOAD FPR1.FPR5

LLD15:
        FPR1_LP = FPR5_LP (AX)
        FPR1_HP = FPR5_HP (AX)
        RET

;******* LOAD FPR2.FPR5

LLD25X:
        FPR2_X  = FPR5_X (A)
LLD25:
        FPR2_LP = FPR5_LP (AX)
        FPR2_HP = FPR5_HP (AX)
        RET

;******* LOAD FPR1.constant

LLD1CX:
        FPR1_X = [HL] (A)
        HL++
LLD1C:
        FPR1_1 = [HL] (A)
        HL++
        FPR1_2 = [HL] (A)
        HL++
        FPR1_3 = [HL] (A)
        HL++
        FPR1_4 = [HL] (A)
        HL++
        RET
```

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

LLD2CX:
        FPR2_X = [HL] (A)
        HL++
LLD2C:
        FPR2_1 = [HL] (A)
        HL++
        FPR2_2 = [HL] (A)
        HL++
        FPR2_3 = [HL] (A)
        HL++
        FPR2_4 = [HL] (A)
        HL++
        RET

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

LXC13X:
        A        =  FPR3_X
        A       <-> FPR1_X
        FPR3_X   =  A
LXC13:
        AX       =  FPR3_LP
        A       <-> FPR1_2
        A       <-> X
        A       <-> FPR1_1
        A       <-> X
        FPR3_LP  =  AX
        AX       =  FPR3_HP
        A       <-> FPR1_4
        A       <-> X
        A       <-> FPR1_3
        A       <-> X
        FPR3_HP  =  AX
        RET

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

LXC14X:
        A        =  FPR4_X
        A       <-> FPR1_X
        FPR4_X   =  A
```

```
LXC14:
        AX       =  FPR4_LP
        A       <-> FPR1_2
        A       <-> X
        A       <-> FPR1_1
        A       <-> X
        FPR4_LP =  AX
        AX       =  FPR4_HP
        A       <-> FPR1_4
        A       <-> X
        A       <-> FPR1_3
        A       <-> X
        FPR4_HP =  AX
        RET

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

LXC15X:
        A        =  FPR5_X
        A       <-> FPR1_X
        FPR5_X   =  A
LXC15:
        AX       =  FPR5_LP
        A       <-> FPR1_2
        A       <-> X
        A       <-> FPR1_1
        A       <-> X
        FPR5_LP =  AX
        AX       =  FPR5_HP
        A       <-> FPR1_4
        A       <-> X
        A       <-> FPR1_3
        A       <-> X
        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
;*********************************************************
;*
;*  78K0 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 **
                                               ; n = 0,1,2 or 3
        CALL !LMOD90
```

```
;******* TRANS. to (sign')sin(x'') by n : 0<=x''<π/2 **
LSIN90:
        if_bit (FPR4_3.0)       ;(n == odd)?
          SET1 FPR1_4.7
          HL = #C1
          CALL !LLD2CX
          CALL !LADDX   ;π/2 -x'
        endif

        CY = FPR4_3.1           ;(n/2 = odd)?
        CY ^= FPR4_4.7          ;  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·FPR1_X
        HL = #CK
        B = #S_PLY
        CALL !LPLY

        A = #R_OK
        CLR1 CY
        RET

;************************************************
;*      GET MOD by π/2
;*
;* input  conditions: FPR1 <- x
;* output conditions: FPR1·FPR1_X = x % π/2
;*                    FPR4_3.(0,1bit) <- quotient
;*                    FPR4_4.7   <- sign of x
;************************************************
LMOD90:
        FPR1_X = #0             ;clear 4th mantissa
        FPR4_3 = #0
        FPR4_4 = FPR1_4 (A)     ;set sign bit
        CLR1 FPR1_4.7           ;|x|
```

```
        while (forever)
          if (FPR1_HP == #C1_4*100H+C1_3) (AX)
            if (FPR1_LP == #C1_2*100H+C1_1) (AX)
               A = FPR1_X
               CMP A,#C1_X
            endif
          endif

          if_bit (CY)
            RET
          endif

          CALL !LLD31X

          HL = #C2
          CALL !LLD2CX
          CALL !LMLTX            ;x / (π/2) : quotient
          FPR1_X  = #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
            A = E
            A += FPR4_3
            FPR4_3 = A           ;add last 2bit of quotient
          endif

          HL = #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
;*********************************************************
;*
;*  78K0 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 **
                                        ; n = 0,1,2 or 3
        CALL !LMOD90

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

LCOS90:
        CLR1 FPR4_4.7           ;clear sign bit
        FPR4_3++                ;n++

        goto LSIN90

        END
```

(11) LTAN.SRC

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

#include "EQU.INC"
#include "REF1.INC"
#include "REF2.INC"

        EXTRN   LDIV
        EXTRN   LMOD90,LSIN90,LCOS90

        PUBLIC  LTAN

        CSEG
;*******************************************************
;*
;*  78K0 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) : 0<=x'<π/2 **
;               cos(x) to       cos(x' +nπ/2) : n = 0,1,2 or 3

        CALL !LMOD90

        AX = FPR4_HP
        PUSH AX                 ;esc. n & sign
        CALL !LLD51X            ;esc. x'

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

        CALL !LCOS90
        CALL !LXC15X

        POP AX
        FPR4_HP = AX
        CALL !LSIN90

        CALL !LLD25
        goto LDIV

        END
```

(12) LLOG.SRC

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

#include "EQU.INC"
#include "REF1.INC"
#include "REF2.INC"

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

        PUBLIC  LLOG

CO_3    EQU     0B5H    ;1st mantissa of √2

S_PLY   EQU     4

        CSEG
;*******************************************************
;*
;*  78K0 FLOATING POINT LOGARITHMIC FUNCTION
;*
;*     input  condition : FPR1 <- x
;*
;*     output conditions: FPR1 <- log(x)
;*                        ERROR then set CY
;*
;*******************************************************
LLOG:
        CY = FPR1_3.7
        A = FPR1_4
        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_LP = FPR1_LP (AX)  ;set mantissa value (:xf) to FPR3
        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'

        HL = #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 !LLD13
        HL = #C2
        CALL !LLD2C
        CALL !LADD                  ;xf' +1

        CALL !LXC13
        HL = #C2
        CALL !LLD2C
        CALL !LSUB                  ;xf' -1

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

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

        CALL !LLD31X

        A = FPR3_4
        ADD A,A
        if_bit (!Z)             ;if(x' != 0) set 2x' to FPR3·FPR3_X
          ADD FPR3_3,#80H       ;else        set 0
          ADDC FPR3_4,#0
        endif

        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·FPR1_X

        HL = #CK
        B  = #S_PLY
        CALL !LPLY2

        A = #R_OK
        CLR1 CY
        RET
C1:
        DB 0F7H,017H,072H,031H,03FH ; const log2
C2:
        DB      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
;*******************************************************
;*
;*  78K0 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

        HL = #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
;*******************************************************
;*
;*  78K0 FLOATING POINT EXPONENTIAL FUNCTION
;*
;*     input  condition : FPR1 <- x
;*
;*     output conditions: FPR1 <- e^x
;*                        ERROR then set CY
;*
;*******************************************************
LEXP:
        FPR1_X = #0
LEXPX:
        FPR3_X = FPR1_4 (A)      ;esc sign bit

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

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

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

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

        if_bit (Z)
          CMP FPR1_X,#0
        endif
        if_bit (!Z)                       ;if ((dec(x/log2) != 0) &&
          if (FPR1_HP >= #8080H) (AX)     ;     (negative))
            DE--                          ;   floor(x/log2) = trunk(x/log2)-1
          endif
        endif

        AX = DE
        AX += #ZEROEX

        if (A != #0)
          goto T_FLOW
        endif

        FPR5_X = X (A)            ;esc exp.
;******* CALC. MANTISSA **

        CALL !LLD21X
        CALL !FTOL                ;floor(x/log2)
        FPR1_4 ^= #80H
        CALL !LADDX               ;x' : x/log2 -floor(x/log2)

        if (FPR1_4 == #3FH)       ;(1/2<=x'<1)
          HL = #C2+1
          CALL !LLD2C
          FPR2_X = #02H           ;(power adjust to (x'<1) for boundary)
          CALL !LSUBX             ;x' :  decimal(x/log2)-1
        endif

        CALL !LLD41X              ;set x'

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

```
        B = #S_PLY
        CALL !LPLY

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

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

        A = FPR5_X
        RORC A,1
        FPR1_3.7 = CY
        FPR1_4 = A
T_EXP9:
        A = #R_OK
        CLR1 CY
        RET
T_FLOW:
        if_bit (FPR3_X.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
;*******************************************************
;*
;*  78K0 FLOATING POINT EXPONENTIAL FUNCTION 2
;*
;*     input  condition : FPR1 <- x
;*
;*     output conditions: FPR1 <- 10^x
;*                        ERROR then set CY
;*
;*******************************************************
LEXP10:
        FPR1_X = #0             ;clear 4th mantissa

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

        FPR3_X = FPR1_4 (A)

        HL = #C1
        CALL !LLD2CX
        CALL !LMLTX
        if_bit (CY)             ;overflow
          if_bit (FPR3_X.7)     ;x<0
            FPR1_HP = #0
            A = #R_OK
            CLR1 CY
          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
;********************************************************
;*
;*  78K0 FLOATING POINT POWER FUNCTION
;*
;*     input  condition : FPR1 <- a , FPR2 <- b
;*
;*     output conditions: FPR1 <- a^b
;*                        ERROR then set CY
;*
;********************************************************
LPOW:
        CY = FPR2_3.7
        A = FPR2_4
        ROLC A,1                ;exp. of b
        C = A

        CY = FPR1_3.7
        A  = FPR1_4
        ADDC A,A                ;exp. of a
        A <-> C

;******* 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 **

        CLR1 FPR5_X.0           ;FPR5_X.0 :sign of result
        if_bit (CY)
          if (A == #0)
            HL = #C1
            CALL !LLD1C
            goto T_POW9         ;x^0=1
          endif

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

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

          A -= #ZEROEX+BYTE*(SHORT-1)-1
          C = A
          B = FPR2_1 (A)
          X = FPR2_2 (A)
          A = FPR2_3
          SET1 A.7

          if_bit (CY)
            repeat
              RORC A,1
              A <-> X
              RORC A,1
              A <-> B
              RORC A,1
              A <-> B
              A <-> X
              if_bit (CY)
                goto ERROR      ;include decimal digit
              endif
              C++
            until_bit (Z)
          endif
          if_bit (Z)            ;if (exp.of b <= 23)
            FPR5_X = B (A)      ;FPR5_X.0 = UNIT1
          endif
        endif
```

```
;***** 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
        FPR2_X = #0
        CALL !LMLTX             ;b * log|a|

        if_bit (CY)             ;overflow
          if_bit (FPR5_4.7)
            FPR1_HP = #0
            goto T_POW9
          endif
          goto ERROR
        endif

        FPR5_1 = FPR5_X (A)
        CALL !LEXPX

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

        if_bit (CY)
          RET
        endif

        if_bit (FPR5_1.0)       ;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   LADD,LDIV

        PUBLIC  LSQRT

C_LIM   EQU     5       ;limiter of approximate

        CSEG
;*******************************************************
;*
;*  78K0 FLOATING POINT SQUARE ROOT FUNCTION
;*
;*   input  condition : FPR1 <- x
;*
;*   output conditions: FPR1 <- √(x)
;*                      ERROR then set CY
;*
;*******************************************************
LSQRT:
        CY = FPR1_3.7
        A = FPR1_4
        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) **

        RORC 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
        FPR1_3 ^= #80H
        FPR4_X = A                 ;escape exp.part root (n)

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

        CALL !LLD31      ;esc. r'/2 to FPR3

        HL = #C1
        CALL !LLD2C
        CALL !LADD       ; r'/2 + .5  : 2ndary approximate (R2)

        FPR3_X = #C_LIM-2
        repeat
          CALL !LLD41    ; esc.previous approximate(Ri) to FPR4
          CALL !LLD21
          CALL !LLD13
          CALL !LDIV     ; (r'/2) / Ri

          CALL !LLD24
          SUB FPR2_3,#80H        ; Ri/2
          SUBC FPR2_4,#0

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

          FPR3_X--
        until_bit (Z)

        A = FPR4_X
        RORC A,1
        FPR1_4 = A                 ;ret. exp.part root (n)
        FPR1_3.7 = CY
        FPR1_X = #0
T_QRT9:
        A = #R_OK
        CLR1 CY
        RET
C1:
        DB 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,LDIV
        EXTRN   LADDX
        EXTRN   LSQRT,LATAN

        PUBLIC  LASIN

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

        CSEG
;*********************************************************
;*
;*  78K0 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|

        if (FPR1_HP == #C1_4*100H+C1_3) (AX)
          if (FPR1_LP == #C1_2*100H+C1_1) (AX)
            HL = #C2            ;|x|=1 exception
            CALL !LLD1CX
```

```
            if_bit (FPR5_4.7)
              SET1 FPR1_4.7        ;± π/2
            endif
            A = #R_OK
            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

        HL = #C1
        CALL !LLD2CX
        CALL !LADDX                ;1-x*x
        CALL !LSQRT                ;√(1-x*x)

        CALL !LLD21
        CALL !LLD15
        CALL !LDIV                 ;x/√(1-x*x)

        goto LATAN

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
;********************************************************
;*
;*  78K0 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

        FPR1_4 ^= #80H

        HL = #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   LADD,LSUB,LMLT,LDIV
        EXTRN   LADDX,LMLTX
        EXTRN   LPLY2
        EXTRN   LRCPN

        PUBLIC  LATAN

S_PLY   EQU     3

        CSEG
;*******************************************************
;*
;*  78K0 FLOATING POINT ARCTANGENT FUNCTION
;*
;*     input  condition : FPR1 <- x
;*
;*     output conditions: FPR1 <- arctan(x)
;*
;*******************************************************
LATAN:
        AX = FPR1_HP
        FPR5_X = A        ;esc. 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|

        CLR1 A.7
        CMPW AX,#ZEROEX SHL 7

        FPR5_X.6 = CY     ;(|x|<1)

        if_bit (!CY)
          CALL !LRCPN     ;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)

    FPR3_X = #0
    if (FPR1_4 >= #3EH)      ;1/8

      CALL !LLD41
      AX = FPR1_HP
      if (AX >= #3F40H)                                  ;6/8
            FPR2_HP = #060H+(SHORT+1)*400H               ;W=7/8
      elseif (A >= #3FH)                                 ;4/8
            FPR2_HP = #020H+(SHORT+1)*300H               ;W=5/8
      elseif (FPR1_3 >= #80H)                            ;2/8
            FPR2_HP = #0C0H+(SHORT+1)*200H               ;W=3/8
      else
            FPR2_HP = #000H+(SHORT+1)*100H               ;W=1/8
      endif

      A <-> FPR2_4
      FPR3_X = A      ;store data pointer of arctan(W)
      FPR2_LP = #0

      CALL !LLD32

      CALL !LMLT      ;x'*W

      HL = #C1
      CALL !LLD2C
      CALL !LADD      ;x'*W +1

      CALL !LXC14
      CALL !LLD23
      CALL !LSUB      ;x' -W

      CALL !LLD24
      CALL !LDIV      ;(x'-W)/(1+x'*W)
    endif
```

```
;******* CALC. APPROXIMATE POLINOMIAL FUNCTION **

        CALL !LLD41

        CALL !LLD21
        CALL !LMLT      ;V*V

        CALL !LXC14X    ;set V*V to FPR4·FPR4_X
        FPR1_X = #0

        HL = #CK0
        CALL !LLD2CX
        CALL !LMLTX     ;4a0*V

        X = FPR3_X (A)
        A = #0
        AX += #CW
        HL = AX
        CALL !LLD2CX    ;arctan(W)

        CALL !LLD31X    ;set 4a0*V to FPR3·FPR3_X

        CALL !LADDX     ;4a0*V +arctan(W)

        HL = #CK1
        B  = #S_PLY
        CALL !LPLY2     ;arctan(x')

        if_bit (!FPR5_X.6)      ;|x|>=1?
          FPR1_4 ^= #80H
          HL = #C2
          CALL !LLD2CX
          CALL !LADDX   ;π/2 -arctan(x')
        endif

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

        FPR1_4.7 = FPR5_X.7 (CY)

        A = #R_OK
        CLR1 CY
        RET
C1:
        DB      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,0BFH ;     a3/a2 *16

        END
```

(21) LHSIN.SRC

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

#include "EQU.INC"
#include "REF1.INC"
#include "REF2.INC"

        EXTRN   LMLT,LSUB
        EXTRN   LPLY
        EXTRN   LEXP
        EXTRN   LRCPN

        PUBLIC  LHSIN

S_PLY   EQU     3

        CSEG
;*******************************************************
;*
;*  78K0 FLOATING POINT HYPERBOLICSINE FUNCTION
;*
;*     input  condition : FPR1 <- x
;*
;*     output conditions: FPR1 <- sinh(x)
;*                        ERROR then set CY
;*
;*******************************************************
LHSIN:

        FPR5_1 = FPR1_4 (A)

;******* CALC. (e^|x|-e^(-|x|))/2  case if |x| >= 0.5 **

        A &= #7FH
        if (A >= #ZEROEX/2)     ;|x| >= 0.5

          CLR1 FPR1_4.7

          CALL !LEXP            ;e^|x|
          if_bit (CY)
            RET                 ;overflow
          endif
```

```
          CALL !LLD31
          CALL !LRCPN              ;e^(-|x|)
          CALL !LLD23

          CALL !LSUB               ;e^(-|x|)-e^|x|
          SUB FPR1_3,#80H
          SUBC FPR1_4,#0           ;(e^(-|x|)-e^|x|)/2

          FPR1_4.7 = FPR5_1.7 (CY)      ;sign bit

;******** CALC. DIRECT APPROXIMATE  case if |x| <  0.5 **

        else                       ;|x| < 0.5

          CALL !LLD41

          CALL !LLD21
          CALL !LMLT               ;x*x

          CALL !LXC14X
          FPR1_X = #0

          HL = #CK
          B  = #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   LADD
        EXTRN   LRCPN
        EXTRN   LEXP

        PUBLIC  LHCOS

        CSEG
;*******************************************************
;*
;*  78K0 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 !LLD31
        CALL !LRCPN             ;e^(-|x|)
        CALL !LLD23

        CALL !LADD              ;e^|x| +e^(-|x|)
        SUB FPR1_3,#80H
        SUBC FPR1_4,#0          ;(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   LDIV
        EXTRN   LHSIN,LHCOS

        PUBLIC  LHTAN

        CSEG
;*******************************************************
;*
;*  78K0 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)
          HL = #C1
          CALL !LLD1C           ;tanh(positive infinity)=1
          if_bit (FPR5_4.7)
            SET1 FPR1_4.7       ;tanh(negative infinity)=-1
          endif
          A = #R_OK
          CLR1 CY
          RET
        endif

        CALL !LXC15             ;esc. cosh(x)
        A = FPR5_1
        PUSH AX
```

```
;******* CALC. LHSIN **

        CALL !LHSIN             ;sinh(x)

;******* CALC. LHTAN **

        POP AX
        FPR5_1 = A
        CALL !LLD25             ;ret. cosh(x)
        goto  LDIV              ;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
;*********************************************************
;*
;*  78K0 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   LDIV

        PUBLIC  LRCPN

        CSEG
;**************************************************
;*
;*  78K0 FLOATING POINT FUNCTION
;*                      GET RECIPROCAL NUMBER
;*
;*    input condition   : FPR1 <- x
;*
;*    output conditions: FPR1 <- 1/x
;*              ERROR then set CY
;*
;**************************************************
LRCPN:
        CALL !LLD21

        HL = #C1
        CALL !LLD1C

        goto LDIV


C1:
        DB 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
;********************************************************
;*
;*  78K0 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:
        A = FPR1_4
        CY = FPR1_3.7
        ADDC A,A

;******* EXCEPTION **

        if_bit (Z)              ;r == 0
          FPR2_HP = #0
          goto T_ORA9
        endif

        if_bit (CY)             ;r < 0
          A = #R_ERR
          RET
        endif
```

```
;******* TRANSLATE  **

        AX = FPR1_LP
        PUSH AX
        AX = FPR1_HP
        PUSH AX                 ;esc. r

        FPR1_HP = FPR2_HP (AX)
        FPR1_LP = FPR2_LP (AX)

        CALL !LMOD90            ;θ -> (sign)(θ'+ nπ/2) (0≦θ'<π/2)

        AX = FPR4_HP
        PUSH AX                 ;esc. n & sign
        CALL !LLD51X            ;esc. θ'

        CALL !LCOS90            ;cos(θ'+nπ/2)
        CALL !LXC15X

        POP AX
        FPR4_HP = AX            ;ret. n & sign
        CALL !LSIN90            ;sin((sign)(θ'+nπ/2))

        POP AX
        FPR3_HP = AX
        POP AX
        FPR3_LP = AX
        FPR3_X = #0             ;ret. r

        CALL !LLD23X
        CALL !LMLTX             ;rsinθ

        CALL !LXC15X

        CALL !LLD23X
        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   LSQRT,LATAN

        PUBLIC  RATOPO

        CSEG
;**********************************************************
;*
;*  78K0 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:
        A = FPR2_4
        L = A                   ;L.7 <- sign of y
        CY = FPR2_3.7
        ADDC A,A

        A = FPR1_4
        H = A                   ;H.7 <- sign of x
        CY = FPR1_3.7
        ROLC A,1

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

        if_bit (Z)
          if (A == #0)          ;if (x==0 && y==0)
            goto T_OP09         ;  {(r,θ) = (0,0)}
          endif
          L = #0                ;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

        PUSH HL                 ;sign bit of x,y

;******** CALC. y/x  **

        CALL !LXC15
        CALL !LLD24
        CALL !LDIV
        if_bit (CY)
          HL = #C1
          CALL !LLD1C           ;arctan(infinity)=π/2 or -π/2
          POP AX
          A = X
          ROLC A,1
          FPR1_4.7 = CY         ;sign of θ <- sign of y
          goto T_OP08
        endif
```

```
;******** CALC. θ **

        CALL !LATAN
        POP AX
        if_bit (A.7)
          HL = #C2
          CALL !LLD2CX          ;x<0, y≧0 : θ=arctan(y/x)+π
          A = X
          ROLC A,1
          FPR2_4.7 = CY         ;x<0, y<0 : θ=arctan(y/x)-π
          CALL !LADDX
        endif

;******** CALC. r  **

T_OP08:
        CALL !LXC15
        CALL !LSQRT             ;√(x*x+y*y)

        CALL !LLD25             ;θ

T_OP09:
        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   LMLT
        EXTRN   LADDX,LMLTX
        EXTRN   LNOR
        EXTRN   LEXPX

        PUBLIC  ATOL

S_VIRT  EQU     27      ;maximum length of mantissa

        CSEG
;*******************************************************
;*
;*  78K0 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 **

        CLR1 FPR1_4.7           ;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)
              E = A              ;initial digit
              D = #0
              BC = #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) ?
                A <-> E
                X = #10
                MULU X

                A <-> X
                E += A
                A = X
                A <-> D
                X = #10
                MULU X

                A <-> X
                ADDC D,A
                A = X
                A <-> C
                X = #10
                MULU X

                A <-> X
                ADDC C,A
                A = X
                ADDC A,#0
                B = A            ;BC·DE <- mantissa val.
```

```
          if_bit (!Z)      ;if (work area fill)
            SET1 FPR2_4.1 ;  {neglect forword 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

    PUSH BC
    PUSH DE
    CALL !GETC
    POP DE
    POP BC
  endw

;******* EXCEPTION **

  FPR2_X = A

  if_bit (FPR2_3.7)       ;no mantissa digit
    goto ERROR
  endif

  if (BC == #0) (AX)
    if (DE == #0) (AX)
      FPR1_HP = #0        ;ZERO EXCEPTION
      goto T_TOL9
    endif
  endif

  if (FPR2_X >= #N_MN)
    goto ERROR
  endif
```

```
        A = FPR2_1          ;decimal digit - neglect digit (F-N)
        A -= FPR2_2
        FPR2_2 = A

;******* NORMALIZE MANTISSA val. **

        A = B

        FPR2_1 = #ZEROEX+SHORT*BYTE-1
        CALL !LNOR                  ;normalize with sign bit
        CALL !LLD51X                ;esc. mantissa val(A')

        A = FPR2_X

;******* 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)
            B = A
            A = #10
            MULU X
            A = B
            X += A
            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_2                ; B - (F-N) (:B')
        E = A
        if_bit (A.7)
          D = #0FFH
        else
          D = #0
        endif

        CALL !FTOL                 ; B' → real

        HL = #C1
        CALL !LLD2CX
        CALL !LMLTX                ;log2(10) * B'
        CALL !LLD21X

        CALL !LTOF
        CALL !FTOL
        PUSH DE                    ;int(log2(10)*B')

        FPR1_4 ^= #80H
        CALL !LADDX                ;dec(log2(10)*B')

        HL = #C2
        CALL !LLD2CX
        CALL !LMLTX                ;dec(log2(10)*B')*log2

        A = FPR5_X
        PUSH AX                    ;esc. A' 4th mantissa
        CALL !LEXPX

        CALL !LLD25                ;ret. A'
        POP AX
        FPR2_X = A                 ;ret. A' 4th mantissa
        CALL !LMLTX                ;A' * e^(dec(log2(10)*B')*log2)
```

```
        A = FPR1_4
        CY = FPR1_3.7
        ROLC A,1

        POP DE
        ADD E,A                  ;exp.part RESULT
        A = D
        ADDC A,#0
        if_bit (A.7)
          E = #0                 ;underflow
        elseif_bit (!Z)
          goto ERROR             ;overflow
        endif

        CY = FPR1_4.7
        A = E
        RORC A,1
        FPR1_4 = 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]
        HL++
        if (A >= #A_0 && A < #A_9+1)
          A -= #A_0
          RET
        endif
        C = A

        DE = #INDEX
        B = #S_INDX
```

```
        repeat
          A = [DE]
          DE++
          if (A == C)
            A = B
            A += #N_9
            RET
          endif
          B--
        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
        EXTRN   LLOG10,LEXP10
        EXTRN   LTOF,FTOL

        PUBLIC  LTOA

S_VIRT  EQU     7       ;string length of mantissa

C2_4    EQU     41H
C2_3    EQU     20H

        CSEG
;*******************************************************
;*
;*  78K0 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:
        CY = FPR1_3.7
        A = FPR1_4
        ADDC A,A

        PUSH HL

;******* ZERO FORMAT **

        if_bit (Z)
          [HL]   = #A_0 (A)
          HL++
          goto T_TOA9
        endif
```

```
;******* TRANS. to a * 10^b (1<= a <10) **

        CALL !LLD51             ;esc. x to FPR5
        CLR1 FPR1_4.7

        CALL !LLOG10            ;log10(|x|)

        CALL !LTOF              ;trunc integer(log10(|x|))
        if_bit (Z)
          CMP FPR1_X,#0
        endif
        if_bit (!Z)                     ;include decimal digit &&
          if (FPR1_HP >= #8080H) (AX)   ;negative ?
            DE--                        ;floor integer(log10(|x|))  : b
          endif
        endif

        A = E
        PUSH AX
        if (A == #38)
          CALL !LLD15
          FPR1_4--
          FPR1_X = #0
          HL = #C1
          CALL !LLD2CX
          CALL !LMLTX           ;x/4 * ((10^-38)*4)
        else
          CALL !FTOL
          FPR1_4 ^= #80H
          CALL !LEXP10          ;10^(-b)
          CALL !LLD25
          FPR2_X = #0
          CALL !LMLTX           ;x * (10^(-b))
        endif
        POP AX
        FPR3_1 = A              ;esc. b

;******* OUTPUT MANTISSA **

        POP HL
        PUSH HL
        if_bit (FPR1_4.7)       ; a<0 ?
          [HL] = #A_MN (A)
          HL++
          CLR1 FPR1_4.7         ; a <- |a|
        endif
        PUSH HL
```

```
if (FPR1_HP >= #C2_4*100H+C2_3)  (AX)   ;if (limit |a|<10 over)
  HL = #C3                              ;  {normalize}
  CALL !LLD2CX
  CALL !LMLTX
  FPR3_1++
endif
if (FPR1_HP < #3F80H) (AX)      ;if (limit |a|>=1 over)
  HL = #C2                      ;  {normalize}
  CALL !LLD2CX
  CALL !LMLTX
  FPR3_1--
endif

CALL !LTOF      ;integer(a)
A = E
A += #A_0
POP HL
[HL] = A
HL++
[HL] = #A_PD (A)
HL++

B = #S_VIRT-1
repeat
  PUSH HL
  CALL !LLD21X
  CALL !FTOL
  SET1 FPR1_4.7
  CALL !LADDX           ;a - integer(a)
  HL = #C2
  CALL !LLD2CX
  CALL !LMLTX           ;(a - integer(a))*10

  CALL !LTOF
  POP HL
  A = E
  A += #A_0
  [HL] = A
  HL++

  B--
until_bit (Z)
```

```
;******* OUTPUT EXP.PART **

        [HL] = #A_E2 (A)
        HL++

        A = FPR3_1
        if_bit (A.7)
          [HL] = #A_MN (A)
          HL++
          A = #0
          A -= FPR3_1
        endif
        X = A
        A = #0

        C = #10
        DIVUW C
        A = C

        AX += #A_0*100H+A_0
        A <-> X
        [HL] = A
        HL++
        A = X
        [HL] = A
        HL++

T_TOA9:
        [HL] = #A_NL (A)
        POP HL
        A = #R_OK
        CLR1 CY
        RET

C1:
        DB 0EDH,0DCH,0C7H,059H,001H ;const (10^-38)*4
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
;*******************************************************
;*
;*  78K0 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 **

        if (AX >= #8000H+1)
          A ^= #0FFH
          A <-> X
          A ^= #0FFH
          A <-> X
          AX++
        endif
```

```
;******* TRANSLATE **

        if (A == #0)
          C = #ZEROEX+BYTE-1
          A <-> X
        else
          C = #ZEROEX+BYTE*INTEGR-1
        endif

        while_bit (!A.7)
          A <-> X
          ROLC A,1
          A <-> X
          ROLC A,1
          C--
        endw

;******* STORE **

        FPR1_X = #0     ;4th mantissa
        FPR1_1 = #0     ;3rd mantissa
        A <-> X
        FPR1_2 = A      ;2nd mantissa

        A = D
        ROL A,1         ;CY <- sign
        A = C
        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
;*******************************************************
;*
;*  78K0 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:
        CY = FPR1_3.7
        A = FPR1_4
        ROLC A,1
        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 **

        C  = A
        DE = FPR1_LP (AX)
        A  = FPR1_3
        A |= #80H                   ;(set mantissa MSB)
        A <-> C

        if_bit (!A.3)
          SET1 A.3
          A <-> D
          E |=  A
          A  = #0
          A <-> C
          A <-> D
        endif

        A <-> C
        A <-> D
        X  =  A
        A  =  D
        D  = #0

        C++
        CLR1 CY
        while_bit (!Z)
          RORC A,1
          A <-> X
          RORC A,1
          A <-> D
          RORC A,1
          A <-> D
          A <-> X
          C++
        endw

;******* UNSIGNED -> SIGNED INTEGER **

        if_bit (FPR1_4.7)
          A ^= #0FFH
          A <-> X
          A ^= #0FFH
          A <-> X
          AX++
          if_bit (!A.7)
            goto ERROR
          endif
```

```
        else
          if_bit (A.7)
            goto ERROR
          endif
        endif

;******* DECIMAL PART JUDGE **

        XCHW AX,DE

        CMPW AX,#0
T_TOF9:
        A = #R_OK
        CLR1 CY
        RET
ERROR:
        A = #R_ERR
        SET1 CY
        RET

        END
```

APPENDIX. EXPLANATION OF SPD CHARTS

SPD is the abbreviation of "Structured Programming Diagrams".

"Structured" here refers to the logical processing structure of a program, involving logical design and assembly performed using basic logical structures.

All programs can be written using only a combination of basic logical structures (sequence, selection, repetition) (this is called a structuring theorem), and the use of structuring clarifies the flow of a program and improves its reliability. There are various methods of representing the structure of a program, but NEC uses the graphic technique known as SPD.

The following table explains the SPD symbols used with this technique, and also shows the equivalent flowchart symbols.

Table A-1 Comparison of SPD and Flowchart Symbols

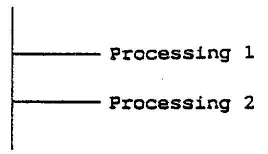
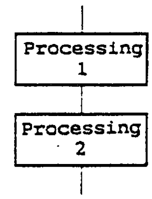
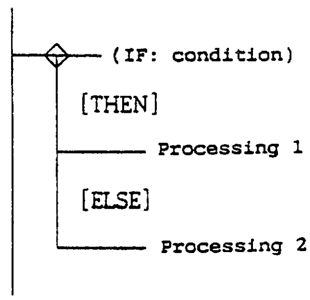
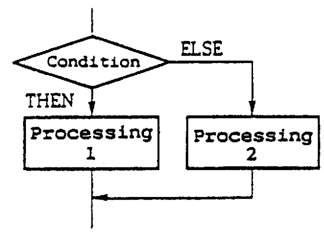
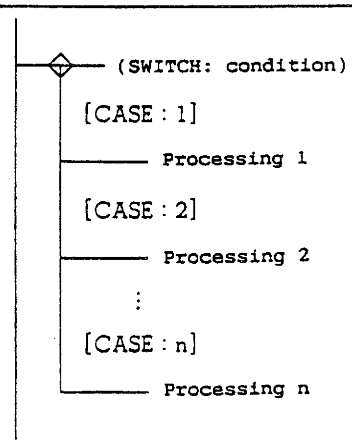
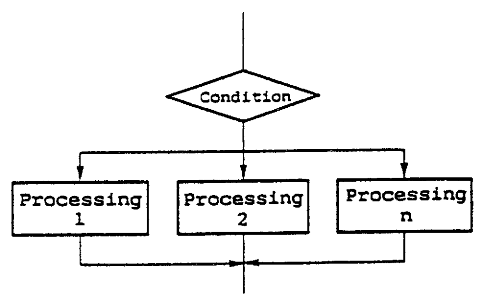
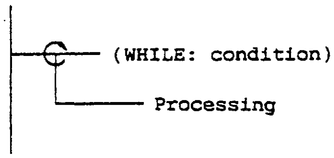
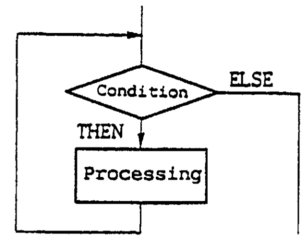
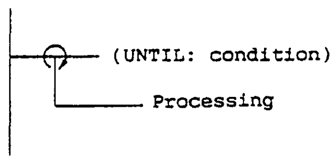
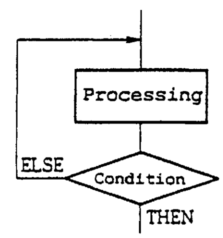
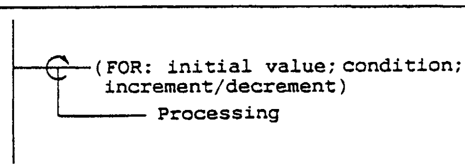
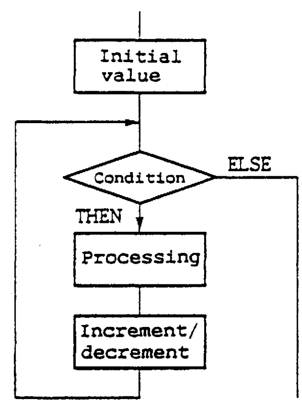
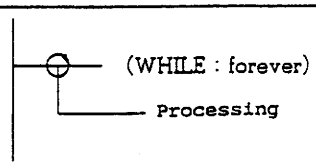
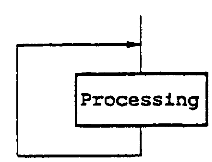
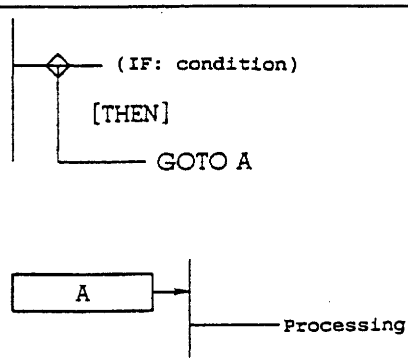
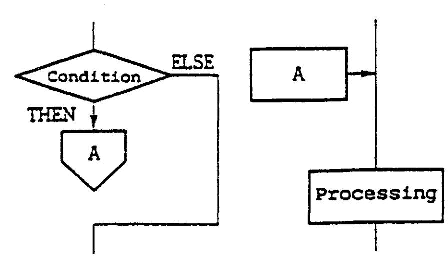
Processing Name	SPD Symbols	Flowchart Symbols
Sequential processing		
Conditional branch (IF)		
Conditional branch (SWITCH)		

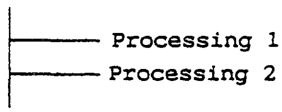
Table A-1 Comparison of SPD and Flowchart Symbols (cont'd)

Processing Name	SPD Symbols	Flowchart Symbols
Conditional loop (WHILE)		
Conditional loop (UNTIL)		
Conditional loop (FOR)		
Endless loop		
Connector		

1. SEQUENTIAL PROCESSING

In sequential processing, the processing is executed in order of appearance from top to bottom.

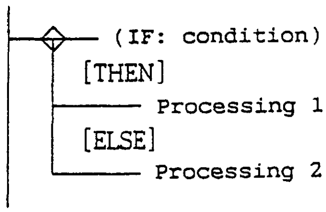
- SPD chart



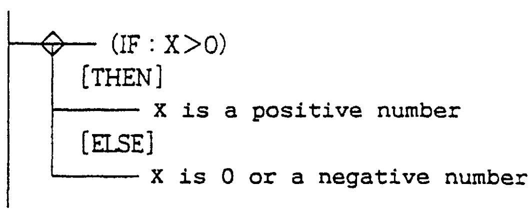
2. CONDITIONAL BRANCH: 2-WAY BRANCH (IF)

The processing to be performed is selected according to whether the condition shown by IF is true or false (THEN/ELSE).

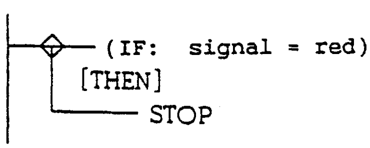
- SPD chart



Example 1: To decide if X is positive or negative



Example 2: To stop if the signal is red

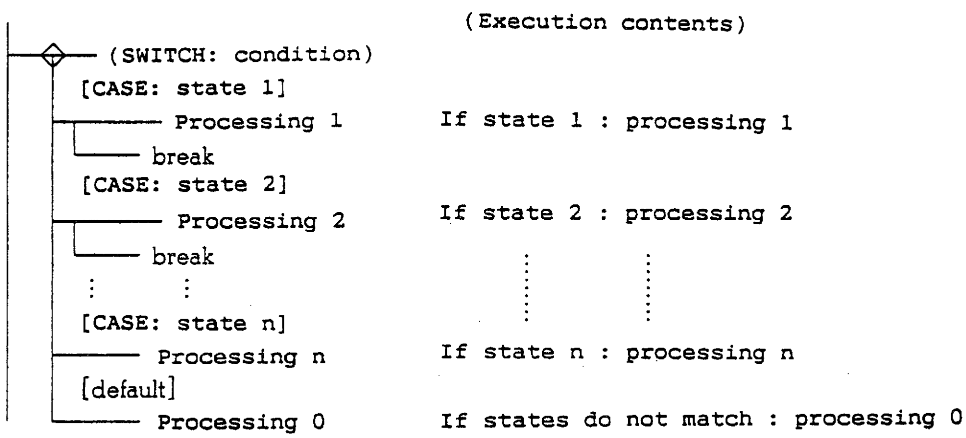


3. CONDITIONAL BRANCH: MULTIPLE BRANCH (SWITCH)

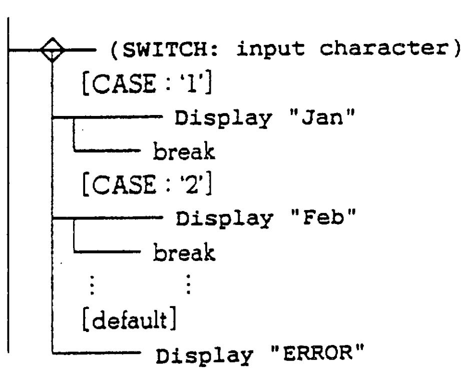
The processing to be performed is selected by comparing the condition shown by SWITCH with the states shown by CASE. There are two cases with SWITCH statement processing: when only the processing for the matching state is executed, and when processing continues from the matching state (when processing doe not continue downward, "break" is written). If there is no matching state, "default" processing is executed (the "default" description can be omitted).

(1) Matching state only

- SPD chart



Example: To display a month according to the input character



(2) When processing continues from matching state

● SPD chart

```
                                         (Execution contents)
    (SWITCH: condition)
    [CASE: state 1]                      If state 1 : processing 1 to processing 2 to
         Processing 1                    processing n
    [CASE: state 2]
         Processing 2                    If state 2 : processing 2 to processing n
    :        :                                     :        :
    [CASE: state n]
         Processing n                    If state n : processing n
    [default]
         processing 0                    If states do not match : processing 0
```

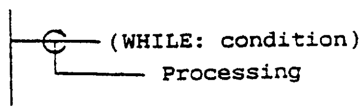
Example: To display a month according to the input character

```
                                         (Execution contents)
    (SWITCH: transfer
             mode)
    [CASE : 1]                           If state 1 : address transmission
         Address                         to data transmission
         transmission
    [CASE : 2]
         Data
         transmission
         break                           If state 2 : data transmission
    [CASE : 3]
         Data reception                  If state 3 : data reception
```

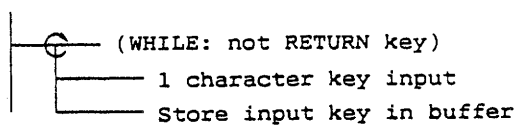
4. CONDITIONAL LOOP (WHILE)

The condition shown by WHILE is judged, and the processing is executed repeatedly while the condition is satisfied (if the condition is satisfied from the start, the processing is not executed).

- SPD chart



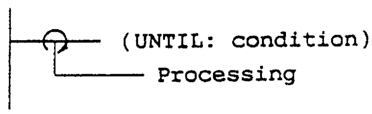
Example: To perform key buffering until RETURN key input



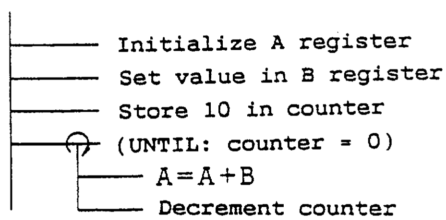
5. CONDITIONAL LOOP (UNTIL)

After processing is performed, the condition shown by UNTIL is judged, and the processing is executed repeatedly until the condition is satisfied (if the condition is not satisfied from the start, the processing is executed once).

- SPD chart



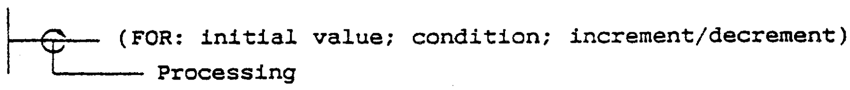
Example: To multiply the value of the B register by 10 and store the result in the A register



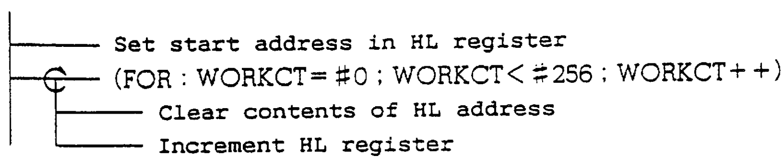
6. CONDITIONAL LOOP (FOR)

The processing is executed repeatedly while the condition of the parameters shown by FOR is satisfied.

- SPD chart



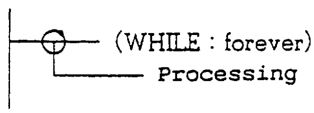
Example: Clear 256 bytes from HL address.



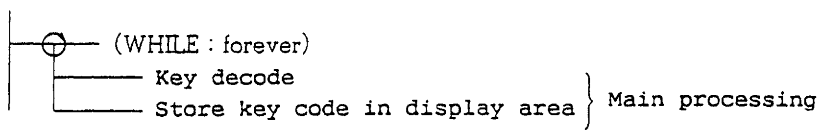
7. ENDLESS LOOP

If "forever" is set as the WHILE condition, execution of the processing is repeated endlessly.

- SPD chart



Example: Repeat main processing.

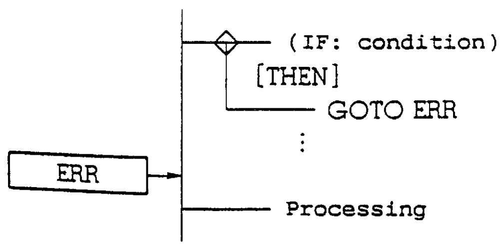


8. CONNECTOR (GOTO)

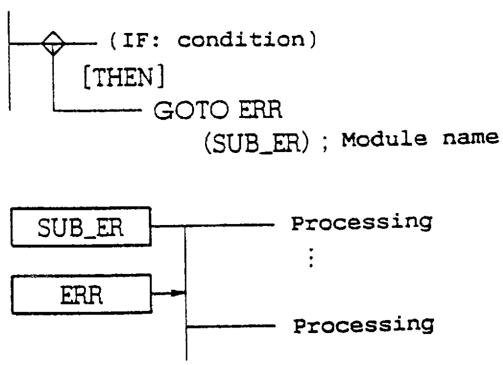
A branch is made to the specified address unconditionally.

● SPD chart

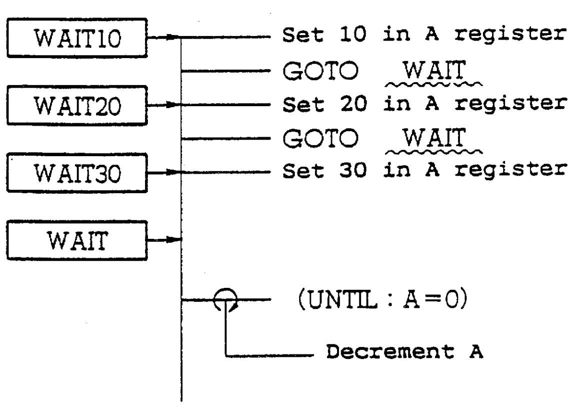
(1) Branch to same module



(2) Branch to other module



Example: To select a parameter and set a wait at a subroutine start address



9. CONNECTOR (CONTINUATION)

Used to indicate the processing flow when the SPD for one module runs over a number of pages.

- SPD chart