NEC

Application Note

78K/III Series

16/8 Bit Single-Chip Microcontroller

Software Basic

μPD78322 sub-series
μPD78328 sub-series
μPD78334 sub-series
μPD78352A sub-series
μPD78356 sub-series
μPD78366A sub-series
μPD78372 sub-series

Document No. U12118EJ4V0AN00 (4th edition)
(O.D.No. IEA-1272B)
Date Published June 1997 J

Phase-out/Discontinued

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

Phase-out/Discontinued

Regional Information

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Major Changes

Page	Description
Throughout	The following products have been added. μPD78356(A), μPD78P356(A), μPD78361A, μPD78362A, μPD78P364A, μPD78363A, μPD78365A, μPD78366A, μPD78368A, μPD78P368A, μPD78372(A), μPD78372(A1), μPD78372(A2), μPD78P372(A), μPD78P372(A1), μPD78P372(A2)
	The following products have been deleted. μPD78355A, μPD78356A, μPD78P356A, μPD78362, μPD78P364, μPD78365, μPD78366, μPD78P368, μPD78370, μPD78372, μPD78P372
	The following products have already been developed. μPD78P324, μPD78P324(A), μPD78P324(A1), μPD78P324(A2), μPD78350A, μPD78355, μPD78356, μPD78P356
P. 125	**Appendix B** has been added.

The mark ⋆ shows major revised points.

Phase-out/Discontinued

[MEMO]

PREFACE

The 78K/III Series microcontrollers are 16/8-bit single-chip microcontrollers that belong to the 78K Series. The 78K/III Series has a powerful instruction set especially suitable for control applications, and includes products containing peripheral hardware tailored to a variety of applications for each product.

Among the 78K/III Series products, the μPD78322, μPD78328, μPD78334, μPD78352A, μPD78356, μPD78366A, and μPD78372 sub-series described in this application note have extended functions and an enhanced instruction set based on the μPD78312A.

This application note provides basic sample programs for these seven series, and explains those instructions that are not available with the μPD78312A.

78K/III Series Products

Sub-series name	Product name
μPD78312A	μPD78310A, μPD78312A, μPD78P312A
μPD78322	μPD78320, μPD78322, μPD78P322, μPD78323, μPD78324, μPD78P324, μPD78320(A), μPD78320(A1), μPD78320(A2), μPD78322(A), μPD78322(A1), μPD78322(A2), μPD78323(A), μPD78323(A1), μPD78323(A2), μPD78324(A), μPD78324(A1), μPD78324(A2), μPD78P324(A), μPD78P324(A1), μPD78P324(A2)
μPD78328	μPD78327, μPD78328, μPD78P328, μPD78327(A), μPD78328(A)
μPD78334	μPD78330, μPD78334, μPD78P334, μPD78330(A), μPD78330(A1), μPD78330(A2), μPD78334(A), μPD78334(A1), μPD78334(A2), μPD78P334(A), μPD78P334(A1), μPD78P334(A2)
μPD78352A	μPD78350, μPD78350A, μPD78352A, μPD78P352
μPD78356	μPD78355, μPD78356, μPD78P356, μPD78356(A), μPD78P356(A)
μPD78366A	μPD78361A**Note**, μPD78362A, μPD78P364A, μPD78363A, μPD78365A, μPD78366A, μPD78368A**Note**, μPD78P368A
μPD78372	μPD78372(A), μPD78372(A1), μPD78372(A2), μPD78P372(A), μPD78P372(A1), μPD78P372(A2)

Note Under development

For the instructions available with the μPD78312A as well, refer to **μPD78312A Application Note (I)** (IEM-1133) and **μPD78312A Application Note (II)** (IEA-1243).

In this application note, references to the μPD78320 pertain to all 78K/III Series products.

The description of the μPD78320 also applies to all other products of the 78K/III series: read μPD78320 as the applicable product name.

The programs contained in this application note are samples, and are not intended for actual production use.

The instruction set of the 78K/III Series allows a choice between an absolute name and functional name as a method of coding a register name in the operand field.

An absolute name has a corresponding functional name that belongs to one of two types of functional name sets (RSS = 0, 1). Which of the sets is to be assigned to each function register can be specified using the assembler pseudo instruction RSS.

For detailed information, refer to the user's manual of each product or the user's manual of the assembler.

Correspondence between Absolute Names and Functional Names of Registers

Absolute name	Functional name	
	RSS=0	RSS=1
R0	X	
R1	A	
R2	C	
R3	B	
R4		X
R5		A
R6		C
R7		B
R8	VP_L	VP_L
R9	VP_H	VP_H
R10	UP_L	UP_L
R11	UP_H	UP_H
R12	E	E
R13	D	D
R14	L	L
R15	H	H
RP0	AX	
RP1	BC	
RP2		AX
RP3		BC
RP4	VP	VP
RP5	UP	UP
RP6	DE	DE
RP7	HL	HL

Quality grade

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

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

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

Application

Standard products

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

Special products

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

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

μPD78322 sub-series

Document name	Document number	
	Japanese	English
μPD78322 Product Letter	IF-6293	-
μPD78322(A) Product Letter	IF-6318	-
μPD78320, 78322 Data Sheet	U10455J	U10455E
μPD78P322 Data Sheet	U10435J	U10435E
μPD78323, 78324 Data Sheet	U10456J	U10456E
μPD78P324, 78P324(A) Data Sheet	IC-8315	IC-2857
μPD78P320(A), 78322(A) Data Sheet	IC-8327	IC-2879
μPD78323(A), 78324(A) Data Sheet	IC-8712	IC-3211
μPD78322 User's Manual	IEU-619	IEU-1248
78K/III Series Application Note, Software Basic	U12118J	This manual
78K/III Series Application Note, Floating-Point Operation Program	U12119J	IEA-1291
μPD78322 Special Function Registers	IEM-5501	-
μPD78322 Instruction Set	IEM-601	-
μPD78322 Instruction List	IEM-602	-

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

Phase-out/Discontinued

μPD78328 sub-series

Document name	Document number	
	Japanese	English
μPD78327, 78328 Data Sheet	U10208J	U10208E
μPD78P328 Data Sheet	U10209J	U10209E
μPD78327(A), 78328(A) Data Sheet	IC-8291	IC-2858
μPD78328 User's Manual	IEU-693	IEU-1268
μPD78328 Application Note, Hardware Basic	IEA-716	IEA-1287
78K/III Series Application Note, Software Basic	U12118J	This manual
78K/III Series Application Note, Floating-Point Operation Program	U12119J	IEA-1291
μPD78328 Special Function Registers	IEM-5514	-
μPD78322 Instruction Set	IEM-601	-
μPD78322 Instruction List	IEM-602	-

μPD78334 sub-series

Document name	Document number	
	Japanese	English
μPD78334 Product Letter	IF-6340	-
μPD78334(A) Product Letter	IF-6292	IF-2027
μPD78330, 78334 Data Sheet	U10185J	U10185E
μPD78P334 Data Sheet	IC-8075	IC-2648
μPD78330(A), 78334(A) Data Sheet	IC-8494	IC-3364
μPD78P334(A), (A1), (A2) Data Sheet	IC-8804	IC-3264
μPD78334 User's Manual	IEU-729	IEU-1315
78K/III Series Application Note, Software Basic	U12118J	This manual
78K/III Series Application Note, Floating-Point Operation Program	U12119J	IEA-1291
μPD78334 Special Function Registers	IEM-5518	-
μPD78322 Instruction Set	IEM-601	-
μPD78322 Instruction List	IEM-602	-

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

Phase-out/Discontinued

μPD78352A sub-series

Document name	Document number	
	Japanese	English
μPD78352A Product Letter	IF-6335	IF-2036
μPD78350 Data Sheet	IC-8279	IC-2845
μPD78350A, 78352A Data Sheet	IC-8823	IC-3391
μPD78P352 Data Sheet	IC-8423	IC-2957
μPD78352A User's Manual, Hardware	IEU-781	IEU-1327
μPD78356 User's Manual, Instruction	U12117J	IEU-1395
78K/III Series Application Note, Software Basic	U12118J	This manual
78K/III Series Application Note, Floating-Point Operation Program	U12119J	IEA-1291
μPD78352A Special Function Registers	IEM-5540	IEM-1215
μPD78352A Instruction Set	U11955J	-

μPD78356 sub-series

Document name	Document number	
	Japanese	English
μPD78356 Product Letter	IF-6298	-
μPD78355, 78356 Data Sheet	U10155J	U10155E
μPD78P356 Data Sheet	U10325J	U10325E
μPD78356(A) Data Sheet	U11148J	U11148E
μPD78P356(A) Data Sheet	U11149J	U11149E
μPD78356 User's Manual, Hardware	U10669J	U10669E
μPD78356 User's Manual, Instruction	U12117J	IEU-1395
78K/III Series Application Note, Software Basic	U12118J	This manual
78K/III Series Application Note, Floating-Point Operation Program	U12119J	IEA-1291
μPD78356 Special Function Registers	IEM-5576	IEM-1214
μPD78352A Instruction Set	U11955J	-

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

Phase-out/Discontinued

μPD78366A sub-series

Document name	Document number	
	Japanese	English
μPD78362A Data Sheet	U10098J	U10098E
μPD78P364A Data Sheet	U10106J	U10106E
μPD78363A, 78365A, 78366A Data Sheet	U11109J	U11109E
μPD78P368A Data Sheet	U11373J	U11373E
μPD78362A User's Manual, Hardware	U10745J	U10745E
μPD78366A User's Manual, Hardware	U10205J	U10205E
μPD78356 User's Manual, Instruction	U12117J	IEU-1395
78K/III Series Application Note, Software Basic	U12118J	This manual
78K/III Series Application Note, Floating-Point Operation Program	U12119J	IEA-1291
μPD78362A Special Function Registers	U10210J	-
μPD78366A Special Function Registers	U10107J	-
μPD78352A Instruction Set	U11955J	-

μPD78372 sub-series

Document name	Document number	
	Japanese	English
μPD78372 Product Letter	IF-6351	-
μPD78370(A), 78372(A) Data Sheet	U10789J	U10789E
μPD78P372(A) Data Sheet	U12029J	U12029E
μPD78372 User's Manual, Hardware	U10642J	U10642E
μPD78356 User's Manual, Instruction	U12117J	IEU-1395
78K/III Series Application Note, Software Basic	U12118J	This manual
78K/III Series Application Note, Floating-Point Operation Program	U12119J	IEA-1291
μPD78372 Special Function Registers	U10631J	U10631E
μPD78352A Instruction Set	U11955J	-

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

Phase-out/Discontinued

[MEMO]

CONTENTS

CHAPTER 1 OVERVIEW OF THE 78K/III SERIES 1

CHAPTER 2 COMPARISON WITH THE μPD78312A 9

2.1 PROGRAM STATUS WORD (PSW) 9

2.2 INSTRUCTION DEFFERENCES 10

2.3 NOTES ON TRANSPORTING PROGRAMS CREATED WITH THE μPD78312A TO THE μPD78320 24

CHAPTER 3 SOFTWARE 27

3.1 SIGNED BINARY OPERATIONS 27

3.1.1 Binary Addition 27

3.1.2 Binary Subtraction 30

3.1.3 Binary Multiplication 33

3.1.4 Binary Division 44

3.2 SIGNED DECIMAL OPERATIONS 50

3.2.1 Decimal Addition 50

3.2.2 Decimal Subtraction 58

3.2.3 Decimal Multiplication 61

3.2.4 Decimal Division 67

3.3 DATA TRANSFER 73

3.4 SHIFT PROCESSING 75

3.4.1 Shifting N-Byte Data to Right 75

3.4.2 Shifting N-Byte Data to Left 77

3.4.3 Shifting N-Digit Decimal Data One Digit to Right 79

3.4.4 Shifting N-Digit Decimal Data One Digit to Left 80

3.5 DATA CONVERSION 81

3.5.1 Conversion from Hexadecimal (HEX) to Decimal (BCD) 81

3.5.2 Conversion from Decimal (BCD) to Hexadecimal (HEX) 85

3.5.3 Conversion from ASCII to Hexadecimal (HEX) 89

3.5.4 Conversion from Hexadecimal (HEX) to ASCII 92

3.6 COMPARISON 94

3.6.1 16-Bit (2-Byte) Data Comparison 94

3.6.2 Data Search 96

3.6.3 Memory Bit Test and Set/Reset Operation 100

3.7 TABLE REFERENCE PROCESSING 104

APPENDIX A INSTRUCTION SET 109

* **APPENDIX B REVISION HISTORY 125**

LIST OF FIGURES

Figure No.	Title	Page
2-1.	PSW Difference	9
2-2.	Data Conversion with the CVTBW Instruction	13
2-3.	Data Flow with MACW Instruction	17
2-4.	Data Flow with MOVTBLW Instruction	20
3-1.	Data Transfer	73
3-2.	Data Search	96
3-3.	Data Format	96
3-4.	Function f(x)	104
3-5.	Data Table	105

LIST OF TABLES

Table No.	Title	Page
1-1.	Functional Overview of the 78K/III Series	2
2-1.	Instructions Newly Available with the μPD78320	10
2-2.	Instructions Partly Modified	11
2-3.	Decimal Data Adjustment (ADJBA Instruction)	21
2-4.	Decimal Data Adjustment (ADJBS Instruction)	22
2-5.	Instruction Code Differences	23
A-1.	Operand Notation and Coding Format	110
A-2.	Absolute Names and their Corresponding Function Names of an 8-bit Register	111
A-3.	Absolute Names and their Corresponding Names of a 16-bit Register	111

CHAPTER 1 OVERVIEW OF THE 78K/III SERIES

The 78K/III Series microcontrollers are CMOS 16/8-bit single-chip microcontrollers containing a 16-bit CPU, achieving high speed and high performance.

The major features are indicated below.

- Instruction set suitable for control applications
  - 16-bit arithmetic/logical instructions
  - Multiply/divide instructions (16 bits x 16 bits, 32 bits/16 bits)
  - Bit manipulation instructions
  - String instructions, etc.
- Peripheral hardware suitable for machine control
  - A/D converter
  - Real-time pulse unit useful for pulse signal I/O control
- Built-in high-performance interrupt controller
  - Vector interrupt function
  - Context switching function

    (When an interrupt is generated, register bank switching is performed, thus achieving high-speed interrupt handling.)
  - Macro service function

    (Special processing including data transfer by hardware is performed.)
- General-purpose serial interface

Table 1-1 outlines each product.

In this application note, each program is coded using the μPD78320 instruction set as the base.

Phase-out/Discontinued

Table 1-1. Functional Overview of the 78K/III Series (1/6)

Sub-series name		μPD78312A			μPD78322		
Product name		μPD78310A	μPD78312A	μPD78P312A	μPD78320	μPD78322	μPD78P322
Number of instructions		96			111		
Minimum instruction execution time		500 ns/12 MHz			250 ns/16 MHz		
Internal memory	ROM size	-	8K bytes	8K bytes (PROM)	-	16K bytes	16K bytes (PROM)
	RAM size	256 bytes			640 bytes		
	Number of external sources	4			8		
	Number of internal sources	13			14 (Of these, 2 sources are also used as external sources.)		
Interrupt functions		8-level programmable priority Vectored interrupt function One macro service (high-speed data transfer) function Context switching function			3-level programmable priority Vectored interrupt function Nine macro service (high-speed data transfer/operation) functions Context switching function		
Test sources		-			One internal test source		
I/O lines	Number of input lines	8			16		
	Number of input/output lines	24	40		21	39	
A/D converter		4 channels with a resolution of 8 bits			8 channels with a resolution of 10 bits		
Timer/counter		Two 16-bit up/down counters Two 16-bit interval timers			One 18/16-bit free running timer One 16-bit timer/event counter		
Serial interface		One channel of UART (containing a dedicated baud rate generator) (Clock synchronous)			One channel of UART (containing a dedicated baud rate generator) One channel of clock synchronous serial interface (for SBI)		
Standby function		HALT/STOP mode					
Features		• Contains an up/down counter suitable for DC servo control.			• Real-time pulse unit allowing a wide variety of programmable pulse output functions		
Package		64-pin SDIP 64-pin QUIP 64-pin QFP (14 x 20 mm) 68-pin QFJ		64-pin SDIP 64-pin SDIP with a window 64-pin QUIP with a window	68-pin QFJ 74-pin QFP (20 x 20 mm) 80-pin QFP (14 x 20 mm)		68-pin QFJ 68-pin WQFN 74-pin WQFN 80-pin WQFN

Phase-out/Discontinued

Table 1-1. Functional Overview of the 78K/III Series (2/6)

Sub-series name		μPD78322			μPD78328		
Product name		μPD78323	μPD78324	μPD78P324	μPD78327	μPD78328	μPD78P328
Number of instructions		111					
Minimum instruction execution time		250 ns/16 MHz					
Internal memory	ROM size	-	32K bytes	32K bytes (PROM)	-	16K bytes	16K bytes (PROM)
	RAM size	1024 bytes			512 bytes		
	Number of external sources	8			4		
	Number of internal sources	14			16		
Interrupt functions		3-level programmable priority Vectored interrupt function Nine macro service (high-speed data transfer/operation) functions Context switching function			3-level programmable priority Vectored interrupt function Eight macro service (high-speed data transfer/operation) functions Context switching function		
Test sources		One internal test source					
I/O lines	Number of input lines	16			11		
	Number of input/output lines	21	39		23	41	
A/D converter		8 channels with a resolution of 10 bits					
Timer/counter		One 18/16-bit free running timer One 16-bit timer/event counter			Two 16-bit free running timers One 16-bit timer/event counter		
Serial interface		One channel of UART (containing a dedicated baud rate generator) One channel of clock synchronous serial interface (for SBI)					
Standby function		HALT/STOP mode					
Features		• Real-time pulse unit allowing a wide variety of programmable pulse output functions • The μPD78P324 contains an ECC circuit.			• Real-time pulse unit that can easily produce 6-phase PWM ideal for inverter control		
Package		68-pin QFJ 74-pin QFP (20 x 20 mm)	68-pin QFJ 68-pin WQFN 74-pin WQFN		64-pin SDIP 64-pin QFP (14 x 20 mm)		64-pin SDIP 64-pin SDIP with a window

*

Phase-out/Discontinued

Table 1-1. Functional Overview of the 78K/III Series (3/6)

Sub-series name		μPD78334		
Product name		μPD78330	μPD78334	μPD78P334
Number of instructions		111		
Minimum instruction execution time		250 ns/16 MHz		
Internal memory	ROM size	-	32K bytes	32K bytes (PROM)
	RAM size	1024 bytes		
Interrupt functions	Number of external sources	8		
	Number of internal source	14 (Of these, 2 sources are also used as external sources.)		
		3-level programmable priority Vectored interrupt function Nine macro service (high-speed data transfer/ operation) functions Context switching function		
Test sources		One internal test source		
I/O lines	Number of input lines	24		
	Number of input/ output lines	28	46	
A/D converter		16 channels with a resolution of 10 bits		
Timer/counter		One 18/16-bit free running timer Three 16-bit timer/event counters		
Serial interface		One channel of UART (containing a dedicated baud rate generator) One channel of clock synchronous serial interface (for SBI)		
Standby function		HALT/STOP mode		
Features		• Real-time pulse unit allowing a wide variety of programmable pulse output functions • The μPD78P334 contains an ECC circuit.		
Package		84-pin QFJ 94-pin QFP (20 x 20 mm)		84-pin QFJ 84-pin WQFN 94-pin WQFN

Phase-out/Discontinued

Table 1-1. Functional Overview of the 78K/III Series (4/6)

Sub-series name		μPD78352A				μPD78356		
* Product name		μPD78350	μPD78350A	μPD78352A	μPD78P352	μPD78355	μPD78356	μPD78P356
Number of instructions		113				115		
* Minimum instruction execution time		160 ns/ 25 MHz	125 ns/32 MHz					
Internal memory	ROM size	-		32K bytes	32K bytes (PROM)	-	48K bytes	48K bytes (PROM)
	RAM size	640 bytes				2K bytes		
	Number of external sources	5				6		
	Number of internal sources	4				25 (Of these, 5 sources are also used as external sources.)		
Interrupt functions		4-level programmable priority Vectored interrupt function Five macro service (high-speed data transfer/operation) functions Context switching function						
I/O lines	Number of input lines	6				9		
	Number of input/ output lines	24		44		48	67	
A/D converter		-				8 channels with a resolution of 10 bits		
D/A converter		-				2 channels with a resolution of 8 bits		
Timer/counter		One 16-bit free running timer One 16-bit timer/event counter One 16-bit interval timer				Two 16-bit timer/event counters Two 16-bit interval timers One 16-bit up/down counter One 10-bit interval timer		
Serial interface		-				One channel of UART (containing a dedicated baud rate generator) One channel of clock synchronous serial interface (for SBI) One channel of clock synchronous serial interface (with pin switching function)		
Standby function		HALT/STOP mode						
Features		• High-speed sum-of-products function • A high-speed control system can be configured using this series and ASICs.				• High-speed, high-performance 16-bit CPU • Contains an ultrahigh-speed A/D converter and high-speed D/A converter • The μPD78P356 contains an ECC circuit.		
Package		64-pin QFP (14 x 14 mm)	64-pin QFP (thin type) (14 x 14 mm)		64-pin QFP (thin type) (14 x 14 mm) 64-pin WQFN**Note**	100-pin QFP (14 x 14 mm) 120-pin QFP (28 x 28 mm)		100-pin QFP 120-pin WQFN

Note Under development

Phase-out/Discontinued

* **Table 1-1. Functional Overview of the 78K/III Series (5/6)**

Sub-series name		μPD78366A							
Product name		μPD78361A[Note]	μPD78362A	μPD78P364A	μPD78363A	μPD78365A	μPD78366A	μPD78368A[Note]	μPD78P368A
Number of instructions		115							
Minimum instruction execution time		125 ns/8 MHz							
Internal memory	ROM size	32K bytes	24K bytes	48K bytes (PROM)	24K bytes	-	32K bytes	48K bytes	48K bytes (PROM)
	RAM size	2K bytes	768 bytes	2K bytes	768 bytes	2K bytes			
Interrupt functions	Number of external sources	6							
	Number of internal sources	14 (Of these, 2 sources are also used as external sources.)							
		4-level programmable priority Vectored interrupt function Five macro service (high-speed data transfer/operation) functions Context switching function							
I/O lines	Number of input lines	14							
	Number of input/output lines	38			49	31	49		
A/D converter		8 channels with a resolution of 10 bits							
D/A converter		-							
Timer/counter		Two 16-bit free running timers One 16-bit interval timer/event counter One 16-bit interval timer One 16-bit timer up/down counter							
Serial interface		One channel of UART (containing a dedicated baud rate generator) One channel of clock synchronous serial interface (for SBI)			One channel of UART (containing a dedicated baud rate generator, and with pin switching function) One channel of clock synchronous serial interface (for SBI)				
Standby function		HALT/STOP mode							
Features		• High-speed, high-performance 16-bit CPU • Contains a 6-channel PWM pulse output function ideal for inverter control • Contains a PLL control circuit							
Package		64-pin SDIP			80-pin QFP (14 x 20 mm)				80-pin QFP 80-pin WQFN

Note Under development

Phase-out/Discontinued

* **Table 1-1. Functional Overview of the 78K/III Series (6/6)**

Sub-series name		μPD78372	
Product name		μPD78372(A)	μPD78P372(A)
Number of instructions		115	
Minimum instruction execution time		160 ns/25 MHz	
Internal memory	ROM size	24K bytes	24K bytes (PROM)
	RAM size	768 bytes	
	Number of external sources	11	
	Number of internal sources	18 (Of these, 6 sources are also used as external sources.)	
Interrupt functions		4-level programmable priority Vectored interrupt function Five macro service (high-speed data transfer/operation) functions Context switching function	
I/O lines	Number of input lines	17	
	Number of input/output lines	43	
A/D converter		16 channels with a resolution of 10 bits	
Timer/counter		One 18-/16-bit free running timer One 16-bit timer/event counter	
Serial interface		One channel of UART (containing a dedicated baud rate generator) One channel of clock synchronous serial interface	
Standby function		HALT/STOP mode and standby function disable mode	
Features		• High-speed, high-performance 16-bit CPU • Real-time pulse unit allowing a wide variety of programmable pulse output functions • The μPD78P372(A) contains an ECC circuit.	
Package		80-pin QFP (14 x 20 mm) 80-pin QFP (14 x 14 mm)	

[MEMO]

CHAPTER 2 COMPARISON WITH THE μPD78312A

This chapter explains the instructions of the μPD78320, particularly the differences between the instructions of the μPD78320 and those of the μPD78312A. Be familiar with this chapter especially when converting a source program created with the μPD78312A to a μPD78320 source program.

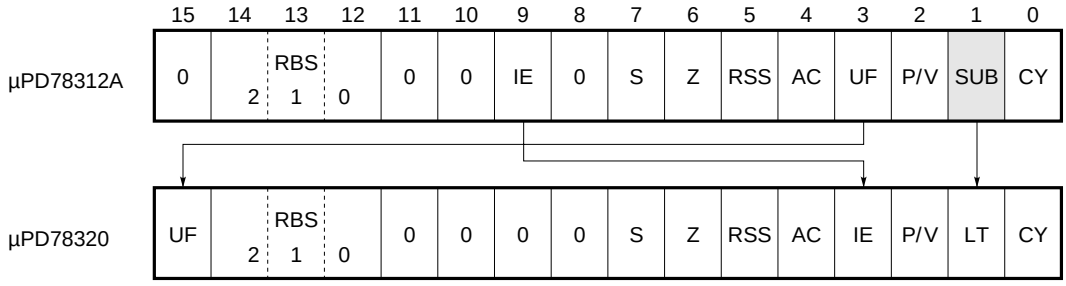
2.1 PROGRAM STATUS WORD (PSW)

Figure 2-1 shows the difference in PSW internal allocation.

With the μPD78320, the user flag (UF) is now moved to bit 15, and the interrupt request enable flag (IE) is now moved to bit 3. Furthermore, the subtraction flag (SUB) has been removed, and an interrupt priority level transition flag (LT) for interrupt control has been newly added.

Figure 2-1. PSW Difference

SUB is not available for μPD78320.



2.2 INSTRUCTION DEFFERENCES

The μPD78320 instruction set is basically upward compatible to the μPD78312A.

The eleven instructions listed in Table 2-1 are not available with the μPD78312A, and are now added to the μPD78320 instruction set. The two instructions indicated in Table 2-2 partly differ from those used with the μPD78312A.

Table 2-1. Instructions Newly Available with the μPD78320 (1/2)

	Instruction added to μPD78320	Mnemonic	Operand	Operation	Flag				
					S	Z	AC	P/V	CY
1	16-bit transfer instruction between memory and register	MOVW	AX, mem	AX <- (mem)					
			mem, AX	(mem) <- AX					
		XCHW	AX, mem	AX <-> (mem)					
2	Sign extension instruction	CVTBW		When A_7 = 0 X <- A, A <- 00H When A_7 = 1 X <- A, A <- FFH					
3	Return instruction from software	RETB interrupt		PC_L <- (SP), PC_H <- (SP+1), PSW_L <- (SP+2), PSW_H <- (SP+3), SP <- SP+4	R	R	R	R	R
4	Return instruction from software context switching	RETCSB	!addr 16	PC_H <- R5, PC_L <- R4, R5 <- $addr16_H$, R4 <- $addr16_L$, PSW_H <- R7, PSW_L <- R6	R	R	R	R	R
5	sfrp push/pop instruction	PUSH	sfrp	(SP - 1) <- sfr_H, (SP - 2) <- sfr_L, SP <- SP - 2					
		POP	sfrp	sfr_L <- (SP), sfr_H <- (SP+1), SP <- SP+2					
6	Port test instruction	CHKL	sfr	(Pin level) ∀ (signal level at pre-stage of output buffer)	x	x		P	
		CHKLA	sfr	A <- {(pin level) ∀ (signal level at pre-stage of output buffer)}	x	x		P	
7	Signed multiply instruction	MULW	rp1	AX (higher 16 bits), rp1 (lower 16 bits) <- AX x rp1					
8	Sum-of-products instruction**Note 1**	MACW**Note 1**	n	AXDE <- (B) x (C) + AXDE, B <- B + 2, C <- C + 2, n <- n - 1, End if n = 0 or P/V = 1	x	x	x	V	x
9	Sum-of-products instruction with indication**Note 2**	MACSW**Note 2** overflow and underflow	n	AXDE <- (B) x (C) + AXDE, B <- B + 2, C <- C + 2, n <- n - 1, if overflow(P/V = 1) then AXDE <- 7FFFFFFFH, if underflow (P/V = 1) then AXDE <- 80000000H, End if n = 0 or P/V = 1	x	x	x	V	x

Notes 1. Only for the μPD78352A, μPD78356, μPD78366A, and μPD78372 sub-series

2. Only for the μPD78356, μPD78366A, and μPD78372 sub-series

Table 2-1. Instructions Newly Available with the μPD78320 (2/2)

Instruction added to μPD78320		Mnemonic	Operand	Operation	Flag				
					S	Z	AC	P/V	CY
10	Correlation instruction**Note 2**	SACW**Note 2**	[DE+], [HL+]	AX <- AX + |(DE) - (HL)|, DE <- DE + 2, HL <- HL + 2, C <- C - 1, End if C = 0 or CY = 1	x	x	x	V	x
11	Table shift instruction**Note 1**	MOVTBLW**Note 1**	!addr16, n	(addr16 + 2) <- (addr16), n <- n = 1, addr16 <- addr16 - 2, End if n = 0					

Notes 1. Only for the μPD78352A, μPD78356, μPD78366A, and μPD78372 sub-series

2. Only for the μPD78356, μPD78366A, and μPD78372 sub-series

Table 2-2. Instructions Partly Modified

Instruction partly modified		Mnemonic	Operand	Operation	Flag				
					S	Z	AC	P/V	CY
1	BCD correction instruction	ADJBA		Decimal	x	x	x	P	x
		ADJBS		Adjust Accumulator					
2	Standby control instruction	MOV	STBC, #byte	STBC <- byte**Note**					
	Watchdog timer control instruction		WDM, #byte	WDM <- byte**Note**					

Note For write access to STBC or WDM, dedicated 4-byte instructions are used. If an object code of these instructions contains an error, the following op-code trap interrupt is generated:

Operation at trap occurrence:

$(SP-1) \leftarrow PSW_H$, $(SP-2) \leftarrow PSW_L$,
$(SP-3) \leftarrow (PC-4)_H$, $(SP-4) \leftarrow (PC-4)_L$,
$PC_L \leftarrow (003CH)$, $PC_H \leftarrow (003DH)$,
$SP \leftarrow SP-4$, $IE \leftarrow 0$

Symbols in the flag field

Symbol	Meaning
	No change occurs.
P	The P/V flag functions as a parity flag.
V	The P/V flag functions as an overflow flag.
R	The previously saved value is restored.
x	The flag is set or cleared according to result.

Phase-out/Discontinued

The functions of the instructions not available with the μPD78312A and instructions partly modified are detailed below.

(1) 16-bit transfer instruction between memory and a register

MOVW AX,mem

Function: AX <- (mem) mem = 0000H-FDFFH (Any address can be specified)
mem = FE00H-FFFFH (Only even addresses can be specified)

Transfers the contents of the memory location addressed in the second operand to register pair AX.

When auto-increment ([DE+], [HL+]) or auto-decrement ([DE-], [HL-]) is specified for mem, the contents of register pair DE or HL are automatically incremented by 2 or decremented by 2 after data transfer.

Flag operation: No change

MOVW mem,AX

Function: (mem) <- AX mem = 0000H-FDFFH (Any address can be specified)
mem = FE00H-FFFFH (Only even addresses can be specified)

Transfers the contents of register pair AX to the memory location addressed in the first operand.

When auto-increment ([DE+], [HL+]) or auto-decrement ([DE-], [HL-]) is specified for mem, the contents of register pair DE or HL are automatically incremented by 2 or decremented by 2 after data transfer.

Flag operation: No change

XCHW AX,mem

Function: AX <-> (mem) mem = 0000H-FDFFH (Any address can be specified)
mem = FE00H-FFFFH (Only even addresses can be specified)

Exchanges the contents of register pair AX with the contents of the 2- byte area addressed in the second operand.

When auto-increment ([DE+], [HL+]) or auto-decrement ([DE-], [HL-]) is specified in mem, the contents of register pair DE or HL are automatically incremented by 2 or decremented by 2 after data exchange.

Flag operation: No change

(2) Sign extension instruction

CVTBW

Function: • When $A_7 = 0$

```
X <- A, A <- 00H
```

• When $A_7 = 1$

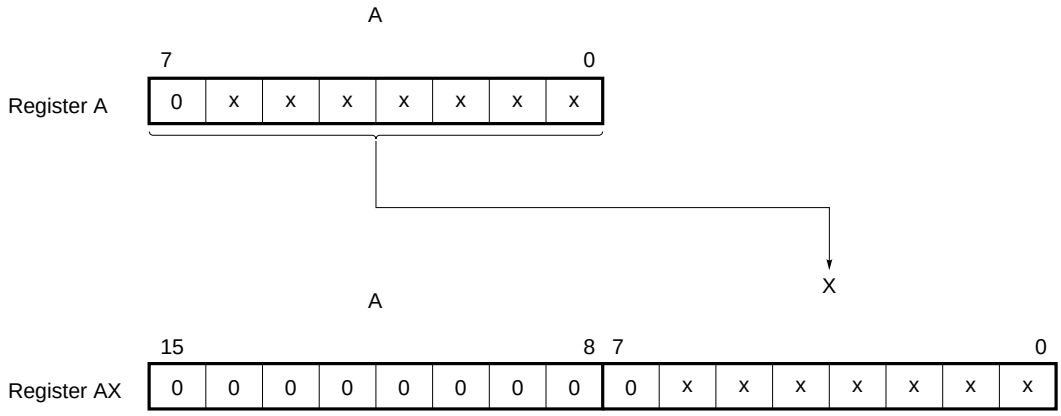
```
X <- A, A <- FFH
```

Converts the signed 8-bit data in register A to signed 16-bit data in register AX. (See **Figure 2-2**.)

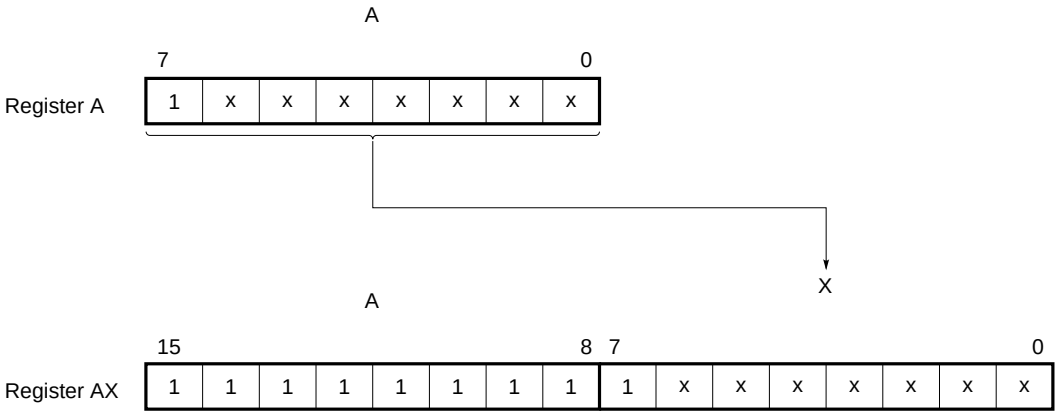
Flag operation: No change

Figure 2-2. Data Conversion with the CVTBW Instruction

(a) When $A_7 = 0$



(b) When $A_7 = 1$



(3) Return instruction from a software interrupt

RETB

Function: PC_L <- (SP), PC_H <- (SP+1),

PSW_L <- (SP+2), PSW_H <- (SP+3), SP <- SP+4

Restores the contents of the instruction location (stack) addressed by the stack pointer (SP) to the program counter and program status word (PSW), then increments the contents of SP.

This instruction is used to return from a BRK instruction or op-code trap.

Flag operation:

S	Z	AC	P/V	CY
R	R	R	R	R

Caution Be sure to use the RETB instruction to return from a BRK instruction or interrupt service routine for an op-code trap. When the RETI instruction is used, the interrupt control circuit does not operate normally.

(4) Return instruction from software context switching

RETCSB !addr16

Function: PC_H <- R5, PC_L <- R4, R5 <- $addr16_H$, R4 <- $addr16_L$, PSW_H <- R7, PSW_L <- R6

addr16 = 0000H-FDFFH

Transfers the contents of the 8-bit registers (R7, R6, R5, R4) in the register bank specified at the time of execution of this instruction to the program status word (PSW) and program counter (PC), then returns to the address set in R5 and R4.

This instruction is used to return from a BRKCS instruction.

Flag operation:

S	Z	AC	P/V	CY
R	R	R	R	R

Caution Be sure to use the RETCSB instruction to return from an interrupt service routine based on the BRKCS instruction. When the RETCS instruction is used, the interrupt control circuit does not operate normally.

(5) sfrp push/pop instruction

PUSH sfrp

Function: (SP-1) <- sfr_H, (SP-2) <- sfr_L, SP <- SP-2

Saves the contents of the special function register (register allowing 16-bit manipulation) to the instruction location (stack) addressed by the stack pointer (SP), then decrements SP.

Flag operation: No change

POP sfrp

Function: sfr_L <- SP, sfr_H <- (SP+1), SP <- SP+2

Restores the contents of the instruction location (stack) addressed by the stack pointer (SP) to the special function register (register allowing 16-bit manipulation), then increments SP.

Flag operation: No change

(6) Port test instruction

CHKL sfr

Function: (Pin level) ∀ (signal level at pre-stage of output buffer)

Exclusive ORs the pin level with the signal level at the pre-stage of the output buffer, then sets the result in the flags (S, Z).

Flag operation:

S	Z	AC	P/V	CY
x	x		P	

CHKLA sfr

Function: A <- {(Pin level) ∀ (signal level at pre-stage of output buffer)}

Exclusive ORs the pin level with the signal level at the pre-stage of the output buffer, then sets the result in register A.

Flag operation:

S	Z	AC	P/V	CY
x	x		P	

(7) Signed multiply instruction

MULW rp1

Function: AX, rp1 <- AX x rp1 (signed)

Multiplies the contents of register pair AX with the contents of the 16-bit register pair specified in the operand, then sets the higher 16 bits of the result in register pair AX, and the lower 16 bits in the 16-bit register pair specified in the operand.

Flag operation: No change

(8) Sum-of-products instruction

MACW n

Function: AXDE <- (B)x(C)+AXDE, B <- B+2, C <- C+2, n <- n-1 end if n=0 or P/V=1

Multiplies the contents of the 2-byte area addressed in register B by the contents of the 2-byte area addressed in register C with the sign, then adds the result to the contents of register pair AXDE in binary. The result of the addition is set in register pair AXDE.

Then, the contents of register B and register C are incremented by 2.

The operation above is repeated as many times as the 8-bit immediate data specified in the operand.

If an overflow occurs in the add operation, the overflow flag is set; the value of register AXDE is unpredictable. Register B and register C hold the values present before an overflow occurs.

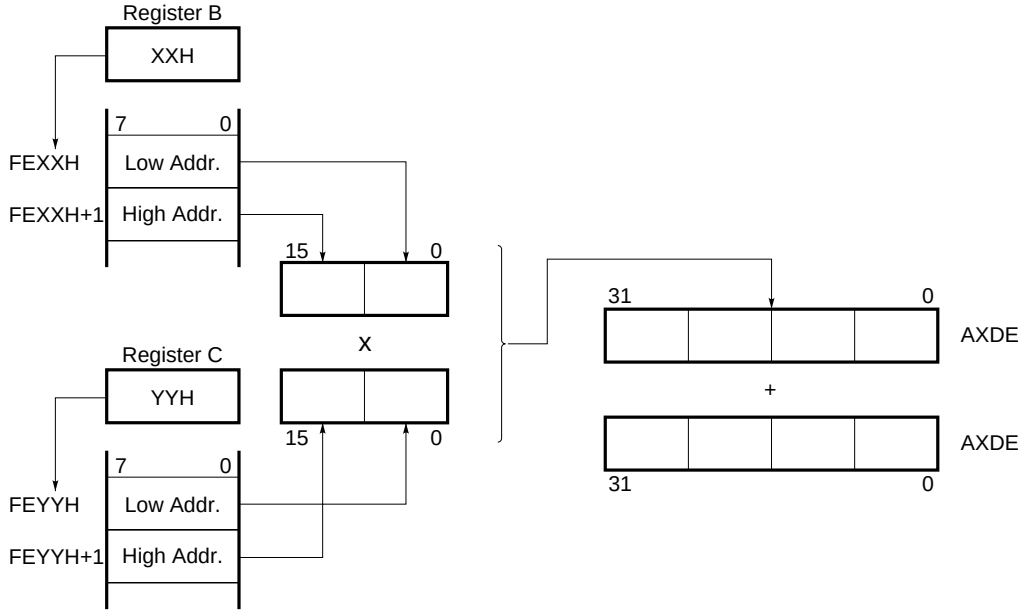
Flag operation:

S	Z	AC	P/V	CY
x	x	x	V	x

Cautions 1. The MACW instruction is only added to the μPD78352A, μPD78356, μPD78366A, and μPD78372 sub-series.

2. Only an area from addresses FE00H to FEFFH is addressed with the MACW instruction. The low-order one byte of an address is to be specified with register B and register C. Addresses FE80H-FEFFH are used also for general registers.
Interrupts and macro services are not accepted during MACW instruction execution.
The value of register pair AXDE is not cleared when MACW instruction execution is started. So clear the value, if necessary, by programming.

Figure 2-3. Data Flow with MACW Instruction



[Signed integer and overflow when the sum-of-products instruction is executed]

(a) Signed integer

Only 16-bit signed integers can be handled with the sum-of-products instruction. The 16-bit data range, examples of data coding, and the range of resultant data set in register AXDE after operation are indicated below.

16-bit data range: -32768 ≤ X ≤ 32767 (decimal)
8000H ≤ X ≤ 7FFFH

Examples of 16-bit data coding: 15 (decimal) =000FH
-1 (decimal) =FFFFH

AXDE register range: -2147483648 ≤ X ≤ 2147483647 (decimal)
80000000H ≤ X ≤ 7FFFFFFFH

(b) Overflow

An overflow occurs if the result of operation exceeds the positive or negative maximum allowable value.

(9) Sum-of-products instruction with overflow and underflow indication

MACSW n

Function: AXDE <- (B)x(C)+AXDE, B <- B+2, C <- C+2, n <- n-1

if n=0 then end

if P/V=1 then AXDE <- 7FFFFFFFH (overflow), end
or AXDE <- 80000000H (underflow), end

overflow : The sign of the addition result changes from + to -.

underflow : The sign of the addition result changes from - to +.

Multiplies the contents of the 2-byte area addressed in register B by the contents of the 2-byte area addressed in register C with the sign, then adds the result to the contents of register pair AXDE in binary. The result of the addition is set in register pair AXDE.

Then, the contents of register B and register C are incremented by 2.

The operation above is repeated as many times as the 8-bit immediate data specified in the operand.

If an overflow or underflow occurs during addition operation, the overflow flag is set, and the operation terminates.

If an overflow occurs, the maximum positive value (7FFFFFFFH) is set to register pair AXDE.

If an underflow occurs, the maximum negative value (80000000H) is set to register pair AXDE.

Register B and register C hold the values present before an overflow occurs.

Flag operation:

S	Z	AC	P/V	CY
x	x	x	V	x

Cautions 1. The MACSW instruction is only added to the μPD78356, μPD78366A, and μPD78372 sub-series.

2. Only an area from addresses FE00H to FEFFH is addressed with the MACSW instruction. The low-order one byte of an address is to be specified with register B and register C. Addresses FE80H-FEFFH are used also for general registers.
Interrupts and macro services are not accepted during MACSW instruction execution.
The value of register pair AXDE is not cleared when MACSW instruction execution is started. So clear the value, if necessary, by programming.

(10) Correlation instruction

SACW[DE+],[HL+]

Function: `AX <- AX+|(DE)-(HL)|, DE <- DE+2, HL <- HL+2, C <- C-1`

`if C=0 or CY=1 then end`

Subtracts the memory contents addressed by register pair HL from the memory contents addressed by register pair DE. The absolute value of the subtraction result is added to the contents of register pair AX.

Then, 2 is added to the contents of both register pairs DE and HL, and the C register contents are decremented.

The above operation is repeated until the C register contents become 0.

Flag operation:

S	Z	AC	P/V	CY
x	x	x	V	x

Caution The SACW instruction is only added to the μPD78356, μPD78366A, and μPD78372 subseries.

(11) Table shift instruction

MOVTBLW !addr16, n

Function: `(addr16+2) <- (addr16), addr16 <- addr16-2, n <- n-1`

`end if n=0`

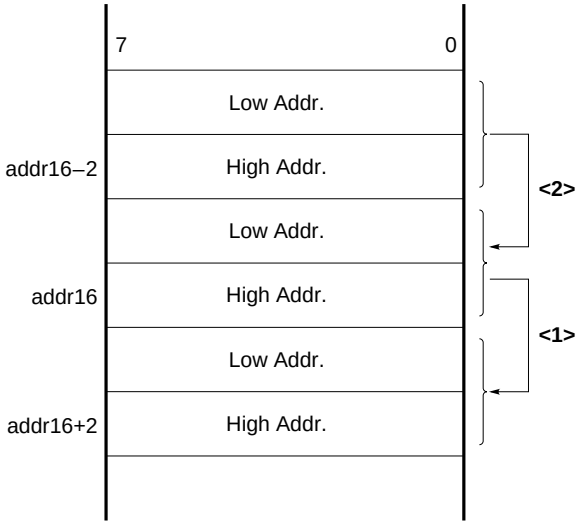
Transfers the contents of the memory addressed by the 16-bit immediate data specified in the first operand to the address incremented by 2. Then, addr16 is decremented by 2.

This operation is repeated as many times as the 8-bit immediate data specified in the second operand, and the MACW instruction data table is shifted.

Enter the lowest address of the data table in !addr16 of the first operand directly with a label or numeric value.

Flag operation: No change

Figure 2-4. Data Flow with MOVTBLW Instruction



Cautions 1. The MOVTBLW instruction is only added to the μPD78352A, μPD78356, μPD78366A, and μPD78372 sub-series.

2. Only an area from addresses FE00H to FEFFH is addressed with the MOVTBLW instruction. The low-order one byte of an address is to be specified in !addr16 of the first operand.

Interrupts and macro services are not accepted during MOVTBLW instruction execution.

(12) BCD correction instruction

With the μPD78320, the BCD correction instruction ADJ4 and the SUB flag of the PSW are not provided, but a BCD correction instruction is available separately for addition and subtraction: ADJBA for addition, and ADJBS for subtraction.

ADJBA

Function: Tests the contents of register A, the carry flag (CY), and auxiliary carry flag (AC), and makes decimal adjustments as indicated in Table 2-3. This instruction does not function until decimal (BCD) data is added to decimal (BCD) data.

Table 2-3. Decimal Data Adjustment (ADJBA Instruction)

Condition		Operation	After adjustment	
			CY	AC
$A_{3-0} \leq 9$ AC = 0	$A_{7-4} \leq 9$ and CY = 0	A <- A	0	0
	$A_{7-4} \geq 10$ or CY = 1	A <- A+60H	1	0
$A_{3-0} \geq 10$ AC = 0	$A_{7-4} < 9$ and CY = 0	A <- A+06H	0	1
	$A_{7-4} \geq 9$ or CY = 1	A <- A+66H	1	1
AC = 1	$A_{7-4} \leq 9$ and CY = 0	A <- A+06H	0	1
	$A_{7-4} \geq 10$ or CY = 1	A <- A+66H	1	1

Flag operation:

S	Z	AC	P/V	CY
x	x	x	P	x

Phase-out/Discontinued

ADJBS

Function: Tests the contents of register A, the carry flag (CY), and auxiliary carry flag (AC), and makes decimal adjustments as indicated in Table 2-4. This instruction does not function until decimal (BCD) data is subtracted from decimal (BCD) data.

Table 2-4. Decimal Data Adjustment (ADJBS Instruction)

Condition		Operation	After adjustment	
			CY	AC
$A_{3-0} \leq 9$ AC = 0	$A_{7-4} \leq 9$ and CY = 0	A <- A	0	0
	$A_{7-4} \geq 10$ or CY = 1	A <- A-60H	1	0
$A_{3-0} \geq 10$ AC = 0	$A_{7-4} < 9$ and CY = 0	A <- A-06H	0	1
	$A_{7-4} \geq 9$ or CY = 1	A <- A-66H	1	1
AC = 1	$A_{7-4} \leq 9$ and CY = 0	A <- A-06H	0	1
	$A_{7-4} \geq 10$ or CY = 1	A <- A-66H	1	1

Flag operation:

S	Z	AC	P/V	CY
x	x	x	P	x

(13) Standby control instruction, watchdog timer control instruction

With the μPD78320, the instruction code in the second byte position differs as indicated in Table 2-5.

Table 2-5. Instruction Code Differences

Shaded portions are different

Product name	Mnemonic	Operand	Instruction code			
			B1	B2	B3	B4
μPD78320	MOV	STBC, #byte	0 0 0 0 1 0 0 1	1 1 0 0 0 0 0 0	$\overline{\text{data}}$	data
		WDM, #byte	0 0 0 0 1 0 0 1	1 1 0 0 0 0 1 0	$\overline{\text{data}}$	data
μPD78312A	MOV	STBC, #byte	0 0 0 0 1 0 0 1	0 1 0 0 0 1 0 0	$\overline{\text{data}}$	data
		WDM, #byte	0 0 0 0 1 0 0 1	0 1 0 0 0 0 1 0	$\overline{\text{data}}$	data

If the third byte of an instruction code is not the complement of the fourth byte or vice versa, the μPD78320 does not write to STBC or WDM, but generates an op-code trap interrupt.

In this case, the return address saved to a stack area is the address of an instruction that caused the trap. So by using the RETB instruction, program execution can be restarted at the address of the instruction that caused the trap.

However, the RETB instruction results in an endless loop if the cause of an op-code trap remains as in the case of a hardware error.

2.3 NOTES ON TRANSPORTING PROGRAMS CREATED WITH THE μPD78312A TO THE μPD78320

(1) BCD correction

The μPD78312A, which contains a SUB flag, can automatically select and execute an ADD or SUB operation according to the level of the SUB flag when the ADJ4 instruction is executed. With the μPD78320, however, an instruction for BCD correction after ADD operation is different from an instruction for BCD correction after SUB operation; a source program modification is required.

Example 1

μPD78320			μPD78312A	
MOV	A, saddr		MOV	A, saddr
ADD	A, B	<-	ADD	A, B
ADJBA			ADJ4	

Example 2

μPD78320			μPD78312A	
MOV	A, saddr		MOV	A, saddr
SUBC	A, B	<-	SUBC	A, B
ADJBS			ADJ4	

(2) Return from the BRK instruction and BRKCS instruction

With the μPD78312A, after CCW.0 (EOS flag) is set, a return instruction is executed to ensure normal interrupt control circuit operation. On the other hand, dedicated instructions are used with the μPD78320, so that a source program modification is required.

Example 1

μPD78320			μPD78312A		
SET1	CY		SET1	CY	
RETB		<-	SET1	CCW.0	;EOS<-1
			RETI		

Replaces 2 instructions with 1 instruction.

Example 2

μPD78320			μPD78312A		
CLR1	CY		CLR1	CY	
RETCSB	!addr16	<-	SET1	CCW.0	;EOS<-1
			RETCS	!addr16	

Replaces 2 instructions with 1 instruction.

(3) Program using an external SFR

The addresses of the external SFR (FFB0H-FFBFH) of the μPD78312A differs from those of the external SFR (FFD0H-FFDFH) of the μPD78320, so a source program modification is required.

(4) PSW manipulation (mainly the UF flag)

The μPD78320 has the format of the PSW modified. So when a PSW bit manipulation instruction is used, a bit position modification is required.

When the EI or DI instruction is used, a source program modification is not required for the IE flag. For the UF flag, however, a source program modification is always required.

Example 1

μPD78320			μPD78312A		
MOV	MK0, #5AH		MOV	MK0, #5AH	
SET1	PSWL.3	<-	SET1	PSWH.1	;IE<-1
⋮			⋮		

Example 2

μPD78320			μPD78312A		
MOV	A,Rn		MOV	A,Rn	
SET1	PSWH.7	<-	SET1	PSWL.3	;UF<-1
⋮			⋮		

[MEMO]

CHAPTER 3 SOFTWARE

3.1 SIGNED BINARY OPERATIONS

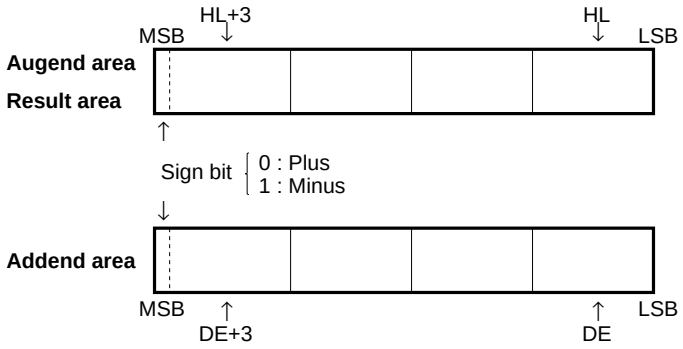
This section explains signed binary add, subtract, multiply, and divide operations.

The most significant bit is used as a sign bit, and the remaining bits represent a numeric value. A negative number is represented as a two's complement.

3.1.1 Binary Addition

32 bits <- 32 bits + 32 bits

(1) Memory areas used



(2) Registers used

A, C, D, E, H, L

(3) Input conditions

As shown in (1) above, the memory addresses of an addend and augend are specified using the following register pairs:

Register pair HL <- Low-order address of the area where a 32-bit augend is stored

Register pair DE <- Low-order address of the area where a 32-bit addend is stored

(4) Output conditions

The result of an operation is stored in the result area (HL, HL+1, HL+2, HL+3) shown in (1) above.

However, when P/V (parity/overflow flag) = 1, the operation has resulted in an overflow or underflow.

(5) Processing procedure

<1> The byte counter (register C) is set to 4.

<2> The carry flag is cleared in advance.

<3> The one byte of the addend specified by an addend address is loaded into the accumulator, then the addend address is incremented by 1.

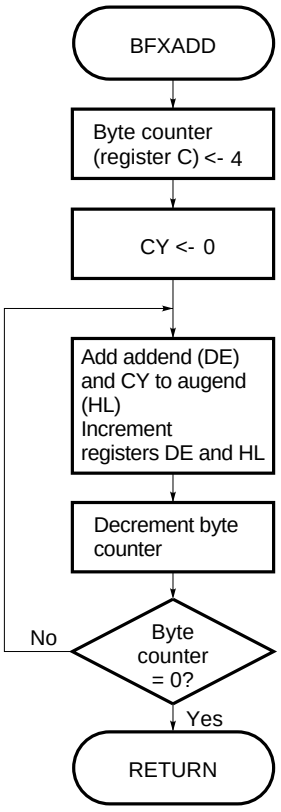
<4> The one byte of the augend specified by an augend address is added together with the carry flag to the accumulator, and the operation result is stored in the memory location specified by the augend address, then the augend address is incremented by 1.

<5> The byte counter is decremented by 1, and the processing of **<3>** and **<4>** is repeated until the byte counter reaches 0.

(6) Number of steps

9 bytes

Flowchart



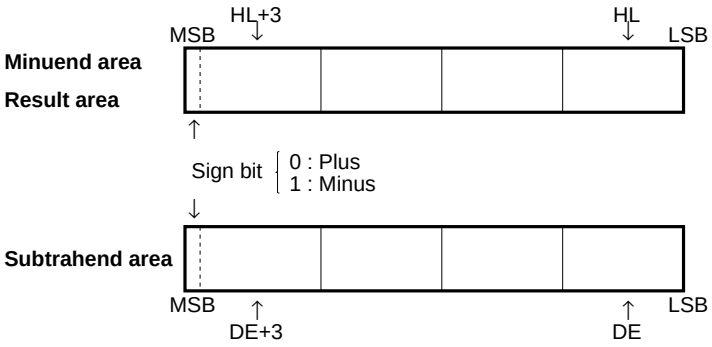
Phase-out/Discontinued

```
      Assemble list

 ALN0   STN0  ADRS   OBJECT    M  I   SOURCE STATEMENT

    1      1                          $       TITLE   ('binary addition')
    2      2                                  NAME    BFXADR
    3      3                          ;*************************************************
    4      4                          ;*      binary addition                          *
    5      5                          ;*          32 bit <- 32 bit + 32 bit            *
    6      6                          ;*       input condition                         *
    7      7                          ;*              HL-register <- augnend top.address *
    8      8                          ;*              DE-register <- addend  top.address *
    9      9                          ;*       output condition                        *
   10     10                          ;*              result <- (HL,HL+1,HL+2,HL+3)      *
   11     11                          ;*                                                 *
   12     12                          ;*                RSS <- 0                         *
   13     13                          ;*************************************************
   14     14
   15     15                                  PUBLIC  BFXADD
   16     16                          ;
   17     17          (0004)          BYTNUM  EQU     4
   18     18                          ;
   19     19  ----                            CSEG
   20     20                                  RSS     0
   21     21                          ;
   22     22  0000                    BFXADD:
   23     23  0000    BA04                    MOV     C.#BYTNUM
   24     24  0002                    BFXAD1:
   25     25  0002    40                      CLR1    CY
   26     26  0003                    BFXAD2:
   27     27  0003    58                      MOV     A,[DE+]
   28     28  0004    1699                    ADDC    [HL+],A
   29     29  0006    32FB                    DBNZ    C,$BFXAD2
   30     30  0008    56                      RET
   31     31                          ;
   32     32                                  ENDS
   33     33                                  END

Segment informations:

ADRS  LEN    NAME

0000  0009H  ?CSEG

         Cross-Reference List

NAME      VALUE   R  ATTR   RTYP   SEGNAME    XREFS

?CSEG               CSEG          ?CSEG          19#
BFXAD1      2H    R  ADDR          ?CSEG          24#
BFXAD2      3H    R  ADDR          ?CSEG          26#    29
BFXADD      0H    R  ADDR   PUB    ?CSEG          15@    22#
BFXADR              MOD                            2#
BYTNUM      4H       NUM                          17#    23

Target chip:uPD78320
Assembly complete,      0 error(s) and       0 warning(s) found. (   0)
```

3.1.2 Binary Subtraction

32 bits <- 32 bits - 32 bits

(1) Memory areas used



(2) Registers used

A, C, D, E, H, L

(3) Input conditions

As shown in (1) above, the memory addresses of a subtrahend and minuend are specified using the following register pairs:

Register pair HL <- Low-order address of the area where a 32-bit minuend is stored

Register pair DE <- Low-order address of the area where a 32-bit subtrahend is stored

(4) Output conditions

The result of an operation is stored in the result area (HL, HL+1, HL+2, HL+3) shown in (1) above. However, when P/V flag = 1, the operation has resulted in an overflow or underflow.

(5) Processing procedure

<1> The byte counter (register C) is set to 4.

<2> The carry flag is cleared in advance.

<3> The one byte of the subtrahend specified by a subtrahend address is loaded into the accumulator, then the subtrahend address is incremented by 1.

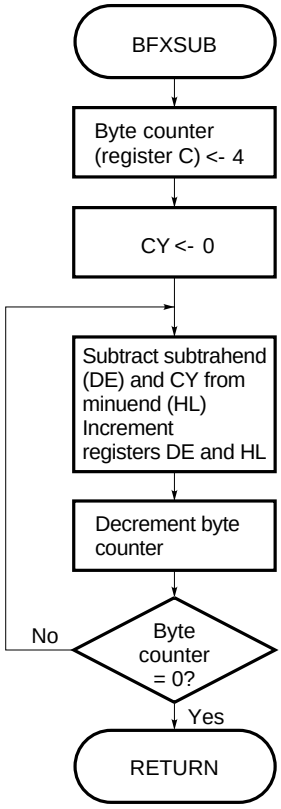
<4> The value of the accumulator is subtracted together with the carry flag from the one byte of the minuend specified by a minuend address, and the operation result is stored in the memory location specified by the minuend address, then the minuend address is incremented by 1.

<5> The byte counter is decremented by 1, and the processing of **<3>** and **<4>** is repeated until the byte counter reaches 0.

(6) Number of steps

9 bytes

Flowchart

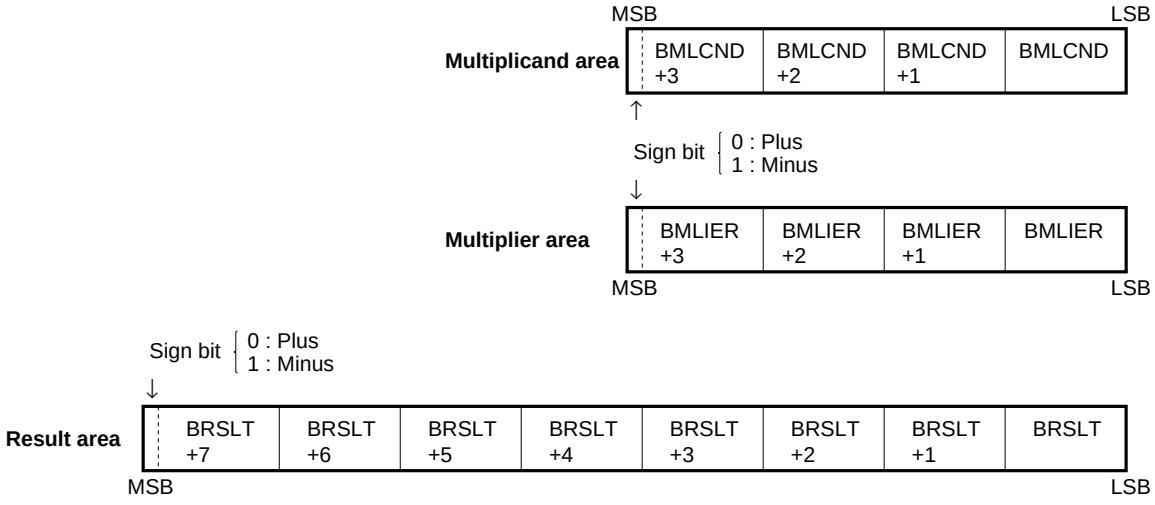


```
      Assemble list

ALN0   STN0  ADRS   OBJECT    M  I   SOURCE STATEMENT

   1      1                          $       TITLE ('binary subtraction')
   2      2                          NAME    BFXSBR
   3      3                          ;*************************************************
   4      4                          ;*      binary subtraction                       *
   5      5                          ;*          32 bit <- 32 bit - 32 bit            *
   6      6                          ;*       input condition                         *
   7      7                          ;*              HL-register <- minuend   top.address     *
   8      8                          ;*              DE-register <- subtrahend top.address     *
   9      9                          ;*       output condition                        *
  10     10                          ;*              result <- (HL,HL+1,HL+2,HL+3)    *
  11     11                          ;*                                               *
  12     12                          ;*                RSS <- 0                       *
  13     13                          ;*************************************************
  14     14
  15     15                                  PUBLIC  BFXSUB
  16     16                          ;
  17     17         (0004)           BYTNUM  EQU     4
  18     18                          ;
  19     19  ----                            CSEG
  20     20                                  RSS     0
  21     21                          ;
  22     22  0000                    BFXSUB:
  23     23  0000   BA04                      MOV     C,#BYTNUM
  24     24  0002                    BFXSU1:
  25     25  0002   40                        CLR1    CY
  26     26  0003                    BFXSU2:
  27     27  0003   58                        MOV     A,[DE+]
  28     28  0004   169B                      SUBC    [HL+],A
  29     29  0006   32FB                      DBNZ    C,$BFXSU2
  30     30  0008   56                        RET
  31     31                          ;
  32     32                                  ENDS
  33     33                                  END

Segment informations:

ADRS  LEN    NAME

0000  0009H  ?CSEG

         Cross-Reference List

NAME      VALUE   R  ATTR   RTYP   SEGNAME    XREFS

?CSEG               CSEG          ?CSEG          19#
BFXSBR              MOD                           2#
BFXSU1      2H    R  ADDR          ?CSEG          24#
BFXSU2      3H    R  ADDR          ?CSEG          26#    29
BFXSUB      0H    R  ADDR   PUB    ?CSEG          15@    22#
BYTNUM      4H       NUM                         17#    23

Target chip:uPD78320
Assembly complete,      0 error(s) and     0 warning(s) found. (    0)
```

3.1.3 Binary Multiplication

64 bits <- 32 bits x 32 bits

(1) Memory areas used



(2) Registers used

X, A, C, B, R4, R5, R8, R9, R10, R11, D, E, H, L

(3) Input conditions

As shown in (1) above, a 32-bit multiplicand and 32-bit multiplier are stored in the following memory areas:
Multiplicand (BMLCND, BMLCND+1, BMLCND+2, BMLCND+3)
Multiplier (BMLIER, BMLIER+1, BMLIER+2, BMLIER+3)

(4) Output conditions

The result of an operation is stored in the result area (BRSLT, BRSLT+1, ..., BRSLT+7).

(5) Processing procedure

The operation program uses a multiply instruction specific to the μPD78320; a 32-bit multiplier is divided into higher 16 bits and lower 16 bits, and a multiplication of a 32-bit multiplicand by a 16-bit multiplier is performed twice.

<1> The result area is cleared to 0 beforehand.

<2> The absolute values of the multiplier and multiplicand are found. If the sign of the multiplier is different from that of the multiplicand, the sign flag (user flag) is set. If the two signs match, the sign flag is reset.

<3> The low-order address of the result area is set in register HL, and is saved to a stack so that the result of the operation performed in **<5>** can be stored in the result area.

<4> (BMLIER, BMLIER+1) is set as the two bytes of the multiplier.

<5> The result of (4-byte multiplicand x 2-byte of multiplier) is added to the result area (HL, HL+1, ..., HL+5), then the operation result is stored in the result area (HL, HL+1, ..., HL+5).

<6> The multiplier occupies the higher digits, so the result of **<4>** is stored. The low-order address of the result area is restored from the stack to register HL, and is incremented by 2.

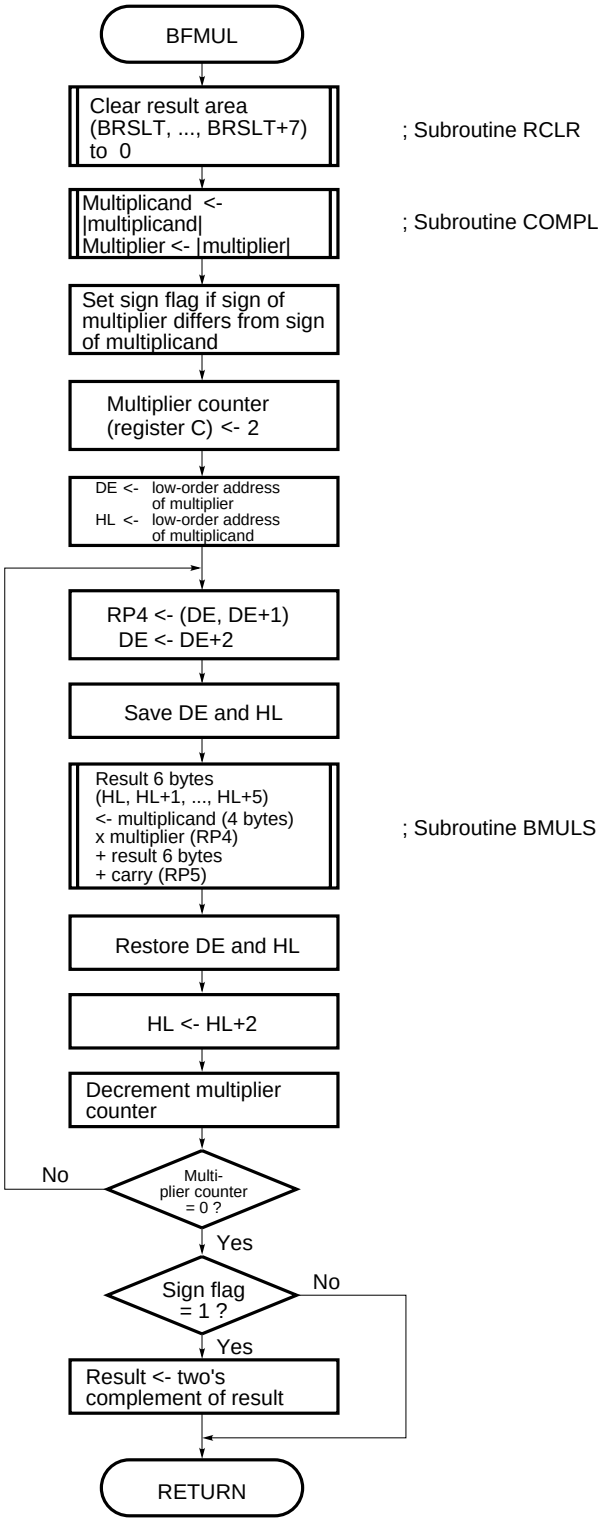
<7> (BMLIER+2, BMLIER+3) is set as the two bytes of the multiplier, then the processing of **<5>** and **<6>** is performed.

<8> The sign flag is checked. If the flag is set, the two's complement of the result is taken.

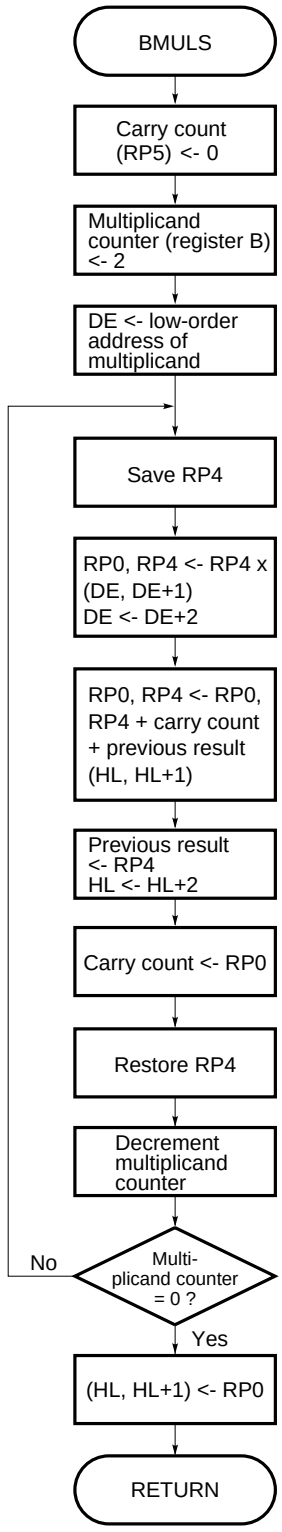
(6) Number of steps

151 bytes (BFMULR: 122 bytes, CLR: 29 bytes)

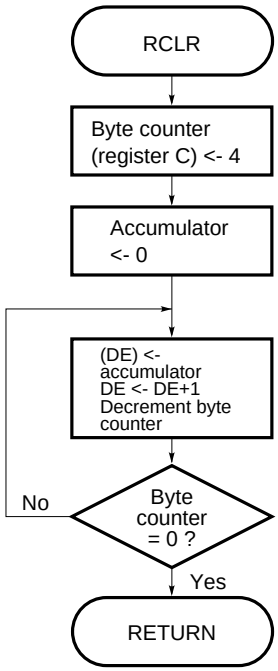
Flowchart



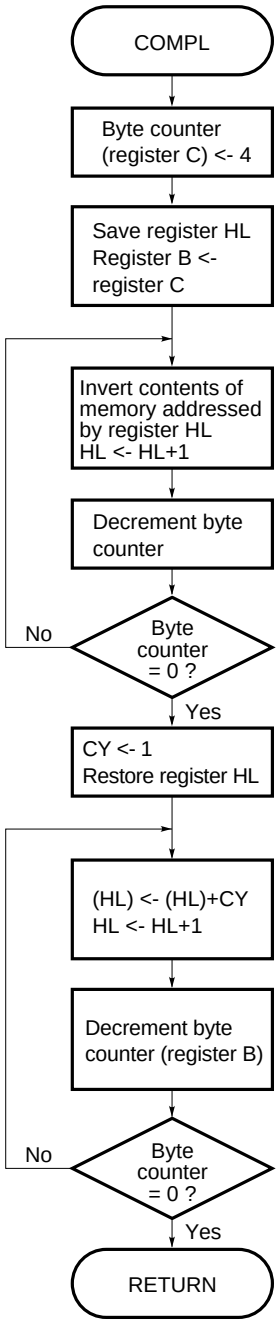
Flowchart



Flowchart



Flowchart



Assemble list

```
ALN0   STN0  ADRS   OBJECT    M  I   SOURCE STATEMENT

   1      1                          $       TITLE   ('binary multiplication')
   2      2                                  NAME    BFMULR
   3      3                          ;*******************************************************
   4      4                          ;*      binary multiplication                          *
   5      5                          ;*           input condition                           *
   6      6                          ;*              multiplicand <- (BMLCND+3,...,BMLCND)  *
   7      7                          ;*              multiplier  <- (BMLIER+3,...,BMLIER)   *
   8      8                          ;*           output condition                          *
   9      9                          ;*              result <- (BRSLT+7,BRSLT+6...,BRSLT)   *
  10     10                          ;*                                                     *
  11     11                          ;*              RSS <- 0                               *
  12     12                          ;*******************************************************
  13     13
  14     14                                  PUBLIC  BFMUL
  15     15                                  PUBLIC  BMULS
  16     16                                  EXTRN   BMLCND,BMLIER,BRSLT
  17     17                                  EXTRN   RCLR,RCLR1,COMPL,COMP1
  18     18                          ;
  19     19          (01FF.7)        SFLAG   EQU     PSWH.7
  20     20                          ;
  21     21  ----                            CSEG
  22     22                                  RSS     0
  23     23  0000                    BFMUL:
  24     24                          ;
  25     25                          ;       **** result area 0-clear ****
  26     26                          ;
  27     27  0000   BA08                     MOV     C,#8            ;
  28     28  0002  R650000                   MOVW    DE,#BRSLT       ; DE-register <- BRSLT
  29     29  0005  R280000                   CALL    !RCLR1          ; clear subroutine
  30     30                          ;
  31     31                          ;       **** complement convert ****
  32     32                          ;
  33     33  0008   029F                     CLR1    SFLAG           ; sign-flag <-0
  34     34  000A  R670000                   MOVW    HL,#BMLCND      ; HL-register <- BMLCND
  35     35  000D   062003                   MOV     A,[HL+3]
  36     36  0010   03AF05                   BF      A.7,$BFMUL1
  37     37  0013  R280000                   CALL    !COMPL          ; complement subroutine
  38     38  0016   027F                     NOT1    SFLAG           ; not sign-flag
  39     39  0018                    BFMUL1:
  40     40  0018  R670000                   MOVW    HL,#BMLIER      ; HL-register <- BMLIER
  41     41  001B   062003                   MOV     A,[HL+3]
  42     42  001E   03AF05                   BF      A.7,$BFMUL2
  43     43  0021  R280000                   CALL    !COMPL          ; complement subroutine
  44     44  0024   027F                     NOT1    SFLAG           ; not sign-flag
  45     45                          ;
  46     46                          ;       **** multiplicand counter set ****
  47     47                          ;
  48     48  0026                    BFMUL2:
  49     49  0026   BA02                     MOV     C,#2            ; C-register <-2
  50     50  0028  R650000                   MOVW    DE,#BMLIER      ; multiplier top.address
  51     51  002B  R670000                   MOVW    HL,#BRSLT       ; result top.address
  52     52                          ;
  53     53  002E                    BFMUL3:
  54     54  002E   1601                     MOVW    AX,[DE+]        ; multiplier set
  55     55  0030   2488                     MOVW    VP,AX
  56     56                          ;
  57     57  0032   35C0                     PUSH    DE,HL           ; SEVE DE,HL
  58     58                          ;
  59     59                          ;       ****  48 bit <- 32 bit * 16 bit  ****
  60     60                          ;
  61     61  0034  R284900                   CALL    !BMULS
  62     62                          ;
  63     63  0037   34C0                     POP     DE,HL           ; LOAD HL,DE
```

```
 64    64  0039   47                      INCW    HL              ; HL <- HL+2
 65    65  003A   47                      INCW    HL
 66    66                         ;
 67    67                         ;       ****  check / multiply end ?  ****
 68    68                         ;
 69    69  003B   32F1                    DBNZ    C,$BFMUL3
 70    70                         ;
 71    71  003D   02AF08                  BF      SFLAG,$BFMUL4   ; sign-flag = 1 ?
 72    72  0040  R670000                  MOVW    HL,#BRSLT
 73    73  0043   BA08                    MOV     C,#8
 74    74  0045  R280000                  CALL    !COMP1          ; complement subroutine
 75    75  0048                       BFMUL4:
 76    76  0048   56                      RET
 77    77                             ENDS
 78    78
 79    79                         $       EJECT
 80    80
 81    81                         ;
 82    82                         ;*********************************************
 83    83                         ;*      binary multiply subroutine             *
 84    84                         ;*          --- 48 bit <- 32 bit * 16 bit ---  *
 85    85                         ;*********************************************
 86    86                         ;
 87    87  ----                           CSEG
 88    88                             RSS     0
 89    89  0049                       BMULS:
 90    90  0049   630000                  MOVW    UP,#0           ;  carry <- 0
 91    91                         ;
 92    92                         ;       **** multiplier counter set ****
 93    93                         ;
 94    94  004C   BB02                    MOV     B,#2
 95    95                         ;
 96    96                         ;       **** multiplicand set ****
 97    97                         ;
 98    98  004E  R650000                  MOVW    DE,#BMLCND
 99    99                         ;
100   100                         ;       **** 16 bit * 16 bit + carry + result ****
101   101                         ;
102   102  0051                       BMULS1:
103   103  0051   3510                    PUSH    VP
104   104  0053   1601                    MOVW    AX,[DE+]
105   105                         ;
106   106  0055   0529                    MULUW   VP
107   107                         ;
108   108                             RSS     1
109   109  0057   43                      SWRS
110   110                         ;
111   111  0058   1651                    MOVW    AX,[HL]
112   112                         ;
113   113                             RSS     0
114   114  005A   43                      SWRS
115   115                         ;
116   116  005B   8888                    ADDW    VP,AX
117   117  005D   8203                    BNC     $BMULS2
118   118  005F   2D0100                  ADDW    AX,#1           ; increment AX
119   119                         ;
120   120  0062                       BMULS2:
121   121  0062   888B                    ADDW    VP,UP
122   122                         ;
123   123  0064   8203                    BNC     $BMULS3
124   124  0066   2D0100                  ADDW    AX,#1           ; increment AX
125   125                         ;
126   126  0069                       BMULS3:
127   127                             RSS     1
128   128  0069   43                      SWRS
129   129                         ;
130   130  006A   2549                    XCHW    AX,VP           ; AX <-> VP
```

```
 131    131  006C   1691                    MOVW    [HL+],AX
 132    132                          ;
 133    133                                  RSS     0
 134    134  006E   43                      SWRS                    ; RSS <- 0
 135    135                          ;
 136    136  006F   24A8                    MOVW    UP,AX
 137    137  0071   3410                    POP     VP
 138    138                          ;
 139    139  0073   33DC                    DBNZ    B,$BMULS1
 140    140                          ;
 141    141  0075   D8                      XCH     A,X
 142    142  0076   51                      MOV     [HL+],A
 143    143  0077   D8                      XCH     A,X
 144    144  0078   55                      MOV     [HL],A
 145    145  0079   56                      RET
 146    146
 147    147                                  ENDS
 148    148                                  END

Segment informations:

ADRS  LEN     NAME

0000  007AH   ?CSEG

         Cross-Reference List

NAME      VALUE   R  ATTR   RTYP   SEGNAME    XREFS

?CSEG               CSEG          ?CSEG          21#   87#
BFMUL       0H    R  ADDR   PUB    ?CSEG          14@   23#
BFMUL1     18H    R  ADDR          ?CSEG          36    39#
BFMUL2     26H    R  ADDR          ?CSEG          42    48#
BFMUL3     2EH    R  ADDR          ?CSEG          53#   69
BFMUL4     48H    R  ADDR          ?CSEG          71    75#
BFMULR              MOD                            2#
BMLCND    ----H    E         EXT                  16@   34     98
BMLIER    ----H    E         EXT                  16@   40     50
BMULS      49H    R  ADDR   PUB    ?CSEG          15@   61     89#
BMULS1     51H    R  ADDR          ?CSEG         102#  139
BMULS2     62H    R  ADDR          ?CSEG         117   120#
BMULS3     69H    R  ADDR          ?CSEG         123   126#
BRSLT     ----H    E         EXT                  16@   28     51     72
COMP1     ----H    E         EXT                  17@   74
COMPL     ----H    E         EXT                  17@   37     43
RCLR      ----H    E         EXT                  17@
RCLR1     ----H    E         EXT                  17@   29
SFLAG    1FFH.7    RBIT                           19#   33     38     44          71

Target chip:uPD78320
Assembly complete,   0 error(s) and          0 warning(s) found. (   0)
```

Assemble list

```
 ALN0   STN0  ADRS   OBJECT    M  I   SOURCE STATEMENT

    1      1                          $       TITLE   ('0-clear routine')
    2      2                                  NAME    CLR
    3      3                          ;*************************************************
    4      4                          ;*      0-clear process                          *
    5      5                          ;*         input condition                       *
    6      6                          ;*              DE-register <- 0-clear start adress      *
    7      7                          ;*                                               *
    8      8                          ;*              RSS <- 0                         *
    9      9                          ;*************************************************
   10     10
   11     11                                  PUBLIC  RCLR,RCLR1,RCLR2
   12     12                                  PUBLIC  COMPL,COMP1
   13     13                          ;
   14     14         (0004)           BYTNUM  EQU     4
   15     15                          ;
   16     16  ----                           CSEG
   17     17                                 RSS     0
   18     18  0000                    RCLR:
   19     19  0000   BA04                    MOV     C,#4            ; C-register <- 4
   20     20  0002                    RCLR1:
   21     21  0002   B900                    MOV     A,#0            ; Acc <- 0
   22     22  0004                    RCLR2:
   23     23  0004   1500                    MOVM    [DE+],A
   24     24                          ;
   25     25  0006   56                      RET
   26     26                                 ENDS
   27     27
   28     28                          ;*******************************************************
   29     29                          ;*      complement convert subroutine                  *
   30     30                          ;*              input  condition                       *
   31     31                          ;*                HL-register <- complement top.address  *
   32     32                          ;*              output condition                       *
   33     33                          ;*                (HL+3,HL+2,...,HL) <- convert data   *
   34     34                          ;*                                                     *
   35     35                          ;*                RSS <- 0                             *
   36     36                          ;*******************************************************
   37     37
   38     38  ----                           CSEG
   39     39                                 RSS     0
   40     40  0007                    COMPL:
   41     41  0007   BA04                    MOV     C,#BYTNUM
   42     42  0009                    COMP1:
   43     43  0009   3580                    PUSH    HL              ; save HL-register
   44     44  000B   2432                    MOV     B,C             ; B-register <- C-register
   45     45  000D   B9FF                    MOV     A,#0FFH         ; Acc <- FFH
   46     46  000F                    COMP2:
   47     47  000F   169D                    XOR     [HL+],A
   48     48  0011   32FC                    DBNZ    C,$COMP2
   49     49                          ;
   50     50  0013   B900                    MOV     A,#0            ; Acc <- 0
   51     51  0015   41                      SET1    CY
   52     52  0016   3480                    POP     HL              ; load HL-register
   53     53                          ;
   54     54  0018                    COMP3:
   55     55  0018   1699                    ADDC    [HL+],A
   56     56  001A   33FC                    DBNZ    B,$COMP3
   57     57                          ;
   58     58  001C   56                      RET
   59     59                                 ENDS
   60     60                                 END
   61     61
```

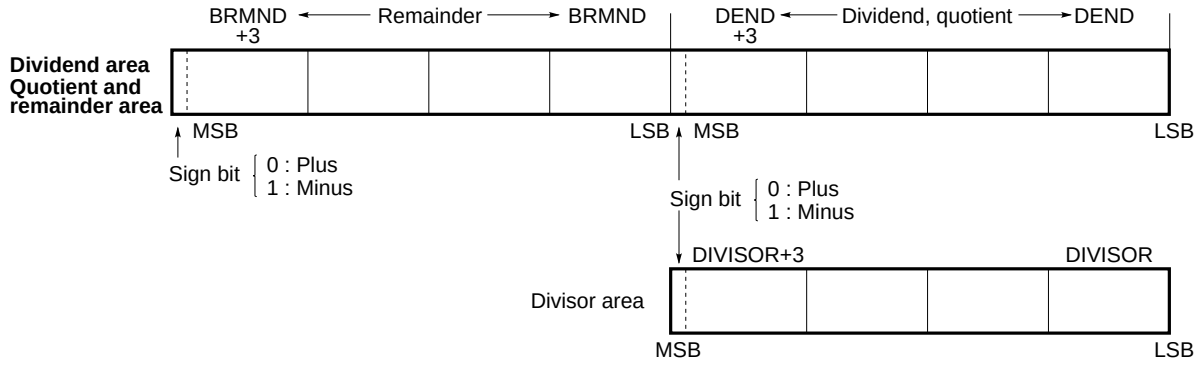
Phase-out/Discontinued

```
Segment informations:

ADRS    LEN      NAME

0000    001DH    ?CSEG

          Cross-Reference List

NAME      VALUE   R  ATTR   RTYP   SEGNAME       XREFS

?CSEG               CSEG          ?CSEG           16#   38#
BYTNUM      4H      NUM                           14#   41
CLR                 MOD                            2#
COMP1       9H    R ADDR   PUB    ?CSEG           12@   42#
COMP2       FH    R ADDR          ?CSEG           46#   48
COMP3      18H    R ADDR          ?CSEG           54#   56
COMPL       7H    R ADDR   PUB    ?CSEG           12@   40#
RCLR        0H    R ADDR   PUB    ?CSEG           11@   18#
RCLR1       2H    R ADDR   PUB    ?CSEG           11@   20#
RCLR2       4H    R ADDR   PUB    ?CSEG           11@   22#

Target chip:uPD78320
Assembly complete,      0 error(s) and        0 warning(s) found. (    0)
```

3.1.4 Binary Division

Quotient : 32 bits <- 32 bits/32 bits
Remainder : 32 bits

(1) Memory areas used



(2) Registers used

X, A, C, B, D, E, H, L

(3) Input conditions

A 32-bit dividend and 32-bit divisor are stored in the following memory areas shown in (1) above:
Dividend (DEND, DEND+1, DEND+2, DEND+3)
Divisor (DIVISOR, DIVISOR+1, DIVISOR+2, DIVISOR+3)

(4) Output conditions

A quotient and remainder are stored in the following areas shown in (1) above:
Quotient : Quotient area (DEND, DEND+1, DEND+2, DEND+3)
Remainder : Remainder area (BRMND, BRMND+1, BRMND+2, BRMND+3)
When a divisor of 0 is specified, a branch to error processing occurs.

(5) Processing procedure

The operation program provided here uses an 8-byte contiguous area for a dividend (DEND, DEND+1, DEND+2, DEND+3) and remainder (BRMND, BRMND+1, BRMND+2, BRMND+3). By shifting the dividend and remainder area one digit (4 bits) to the left, the most significant digit of the dividend is moved to the least significant digit position of the remainder, and the quotient, starting with the most significant digit, enters the least significant digit position of the dividend digit by digit.

The digits of a quotient represent the number of repetitions occurring until the result of (remainder - dividend) becomes negative (until a borrow occurs).

<1> A check is made to see if the divisor is 0. If the divisor is 0, a branch to error processing occurs.

<2> The remainder area is cleared to 0.

<3> The absolute values of the dividend and divisor are found. The quotient sign flag (bit 0 of X) and the remainder sign flag (bit 1 of X) are used. If only the dividend or divisor is negative, the quotient sign flag is set. If the dividend is negative, the remainder sign flag is set.

<4> The quotient byte counter (register B) is set to 8.

<5> The contiguous 8-byte quotient and remainder area is shifted four bits to the left.

<6> (Remainder - divisor) is loaded into the remainder area. When (remainder - divisor) assumes a negative value, a jump to **<8>** occurs.

<7> The quotient (DEND) is incremented. A jump to **<5>** occurs.

<8> An excessive subtraction was performed. So (remainder + divisor) is loaded into the remainder area.

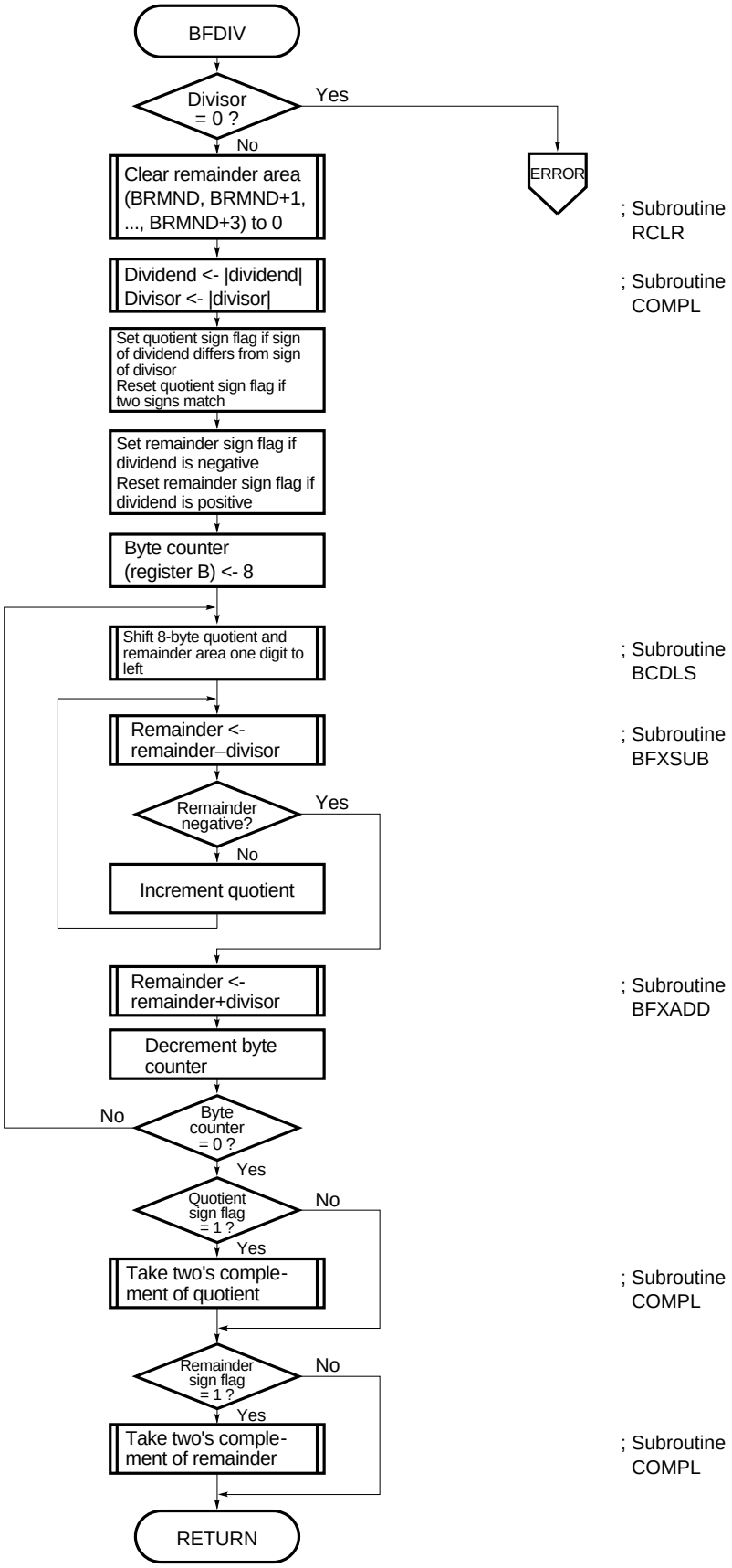
<9> The quotient byte counter is decremented, and the processing of **<5>** to **<8>** is repeated until the quotient byte counter reaches 0.

<10> The quotient sign flag is checked. If the quotient sign flag is set, the two's complement of the quotient is taken. The remainder sign flag is checked. If the remainder sign flag is set, the two's complement of the remainder is taken.

(6) Number of steps

119 bytes

Flowchart



Phase-out/Discontinued

Assemble list

```
ALN0   STN0  ADRS   OBJECT    M  I   SOURCE     STATEMENT

   1      1                          $       TITLE   ('binary division')
   2      2                                  NAME    BFDIVR
   3      3                          ;***************************************************
   4      4                          ;*      binary division                            *
   5      5                          ;*             32 bit <- 32 bit / 32 bit           *
   6      6                          ;*      input condition                            *
   7      7                          ;*             dividend <- (DEND+3,...,DEND)       *
   8      8                          ;*             divisor <- (DVISOR+3,...,DVISOR)    *
   9      9                          ;*      output condition                           *
  10     10                          ;*             quotient <- (DEND+3,...,DEND)       *
  11     11                          ;*             remainder <- (DRMND+3,...,DRMND)    *
  12     12                          ;*                                                 *
  13     13                          ;*               RSS <- 0                          *
  14     14                          ;***************************************************
  15     15
  16     16                                  PUBLIC  BFDIV
  17     17                                  EXTRN   BFXADD,BFXSUB
  18     18                                  EXTRN   RCLR,COMPL,BCDLS,ERROR
  19     19                                  EXTRN   DEND,DVISOR,DRMND
  20     20                          ;
  21     21          (0000.0)        SF_REM  EQU     X.0
  22     22          (0000.1)        SF_QUO  EQU     X.1
  23     23          (0004)          BYTNUM  EQU     4
  24     24                          ;
  25     25  ----                            CSEG
  26     26                                  RSS     0
  27     27  0000                    BFDIV:
  28     28                          ;
  29     29                          ;       ****  check / divisor = 0 ?  ***
  30     30                          ;
  31     31  0000    BA04                    MOV     C,#BYTNUM       ; C-register <- 4
  32     32  0002    B900                    MOV     A,#0            ; Acc <- 0
  33     33  0004  R670000                    MOVW    HL,#DVISOR      ; HL <- DVISOR
  34     34                          ;
  35     35  0007                    BFDIV1:
  36     36  0007    169F                    CMP     [HL+],A
  37     37  0009    8005                    BNZ     $BFDIV2         ; [HL] = 0 ?
  38     38  000B    32FA                    DBNZ    C,$BFDIV1
  39     39                          ;
  40     40                          ;       ****  divisor = 0  ****
  41     41                          ;
  42     42  000D  R2C0000                   BR      !ERROR          ; OVER FLOW
  43     43                          ;
  44     44                          ;       ****  quotient 0-clear  ****
  45     45                          ;
  46     46  0010                    BFDIV2:
  47     47  0010  R650000                   MOVW    DE,#DRMND       ; DE-register <- DRMND
  48     48  0013  R280000                   CALL    !RCLR
  49     49                          ;
  50     50                          ;       ****  complement convert  ****
  51     51                          ;
  52     52  0016    0390                    CLR1    SF_REM          ; clear remainder sign-flag
  53     53  0018    0391                    CLR1    SF_QUO          ; clear quotient sign-flag
  54     54  001A  R670000                   MOVW    HL,#DEND        ; HL-register <- DEND
  55     55  001D    062003                  MOV     A,[HL+3]
  56     56  0020    03AF07                  BF      A.7,$BFDIV3
  57     57  0023  R280000                   CALL    !COMPL          ; complement subroutine
  58     58  0026    0380                    SET1    SF_REM          ; set remainder sign-flag
  59     59  0028    0371                    NOT1    SF_QUO          ; not quotient sign-flag
  60     60  002A                    BFDIV3:
  61     61  002A  R670000                   MOVW    HL,#DVISOR      ; HL-register <- DVISOR
  62     62  002D    062003                  MOV     A,[HL+3]
  63     63  0030    03AF05                  BF      A.7,$BFDIV4
```

```
  64     64  0033  R280000                BFDIV3:   CALL    !COMPL          ; complement subroutine
  65     65  0036   0371                            NOT1    SF_QUO          ; not quotient sign-flag
  66     66                          ;
  67     67                          ;       **** byte counter set  ****
  68     68                          ;
  69     69  0038                   BFDIV4:
  70     70  0038   BB08                            MOV     B,#8            ; B-register <- 8
  71     71                          ;
  72     72                          ;       **** dividend,remainder 1-bye left shift ****
  73     73                          ;
  74     74  003A                   BFDIV5:
  75     75  003A  R670000                        MOVW    HL,#DEND        ; HL <- DEND
  76     76  003D   BA08                            MOV     C,#8            ; C-register <- 8
  77     77  003F  R280000                        CALL    !BCDLS
  78     78                          ;
  79     79                          ;       **** subtract divisor from dividend ****
  80     80                          ;
  81     81  0042                   DFDIV6:
  82     82  0042  R650000                        MOVW    DE,#DVISOR      ; DE <- DVISOR
  83     83  0045  R670000                        MOVW    HL,#DRMND       ; HL <- DRMND
  84     84  0048  R280000                        CALL    !BFXSUB
  85     85                          ;
  86     86  004B   4F                            DECW    HL              ; decrement HL
  87     87  004C   5D                            MOV     A,[HL]
  88     88  004D   03BF09                        BT      A.7,$BFDIV7     ; if borrow
  89     89                          ;
  90     90  0050   B901                          MOV     A,#1            ; ACC <- 1
  91     91  0052  R670000                        MOVW    HL;#DEND
  92     92  0055   16D8                          ADD     [HL],A          ; increment DEND
  93     93                          ;
  94     94  0057   14E9                          BR      $BFDIV6
  95     95                          ;
  96     96                          ;       **** if borrow divisor + dividend  ****
  97     97                          ;
  98     98  0059                   BFDIV7:
  99     99  0059  R650000                        MOVW    DE,#DVISOR      ; DE <- DVISOR
 100    100  005C  R670000                        MOVW    HL,#DRMND       ; HL <- DRMND
 101    101  005F  R280000                        CALL    !BFXADD
 102    102                          ;
 103    103                          ;       **** check / division end ?  ****
 104    104                          ;
 105    105  0062   33D6                          DBNZ    B,$BFDIV5
 106    106                          ;
 107    107  0064   03A006                        BF      SF_REM,$BFDIV8
 108    108  0067  R670000                        MOVW    HL,#DRMND
 109    109  006A  R280000                        CALL    !COMPL
 110    110                          ;
 111    111  006D                   BFDIV8:
 112    112  006D   03A106                        BF      SF_QUO,#BFDIV9
 113    113  0070  R670000                        MOVW    HL,#DEND
 114    114  0073  R280000                        CALL    !COMPL
 115    115                          ;
 116    116  0076                   BFDIV9:
 117    117  0076   56                            RET
 118    118
 119    119                                     ENDS
 120    120                                     END

Segment informations:

ADRS   LEN    NAME

0000   0077H ?CSEG
```


```
         Cross-Reference List

NAME      VALUE   R  ATTR   RTYP  SEGNAME    XREFS

?CSEG               CSEG         ?CSEG          25#
BCDLS     ----H   E            EXT                 18@   77
BFDIV       0H    R  ADDR   PUB   ?CSEG          16@   27#
BFDIV1      7H    R  ADDR         ?CSEG          35#   38
BFDIV2     10H    R  ADDR         ?CSEG          37    46#
BFDIV3     2AH    R  ADDR         ?CSEG          56    60#
BFDIV4     38H    R  ADDR         ?CSEG          63    69#
BFDIV5     3AH    R  ADDR         ?CSEG          74#  105
BFDIV6     42H    R  ADDR         ?CSEG          81#   94
BFDIV7     59H    R  ADDR         ?CSEG          88    98#
BFDIV8     6DH    R  ADDR         ?CSEG         107   111#
BFDIV9     76H    R  ADDR         ?CSEG         112   116#
BFDIVR              MOD                          2#
BFXADD    ----H   E            EXT                 17@  101
BFXSUB    ----H   E            EXT                 17@   84
BYTNUM      4H        NUM                          23#   31
COMPL     ----H   E            EXT                 18@   57     64    109    114
DEND      ----H   E            EXT                 19@   54     75     91    113
DRMND     ----H   E            EXT                 19@   47     83    100    108
DVISOR    ----H   E            EXT                 19@   33     61     82     99
ERROR     ----H   E            EXT                 18@   42
RCLR      ----H   E            EXT                 18@   48
SF_QUO      0H.1  RBIT                             22#   53     59     65    112
SF_REM      0H.0  RBIT                             21#   52     58    107

Target chip:uPD78320
Assembly complete,      0 error(s) and        0 warning(s) found. (   0)
```

3.2 SIGNED DECIMAL OPERATIONS

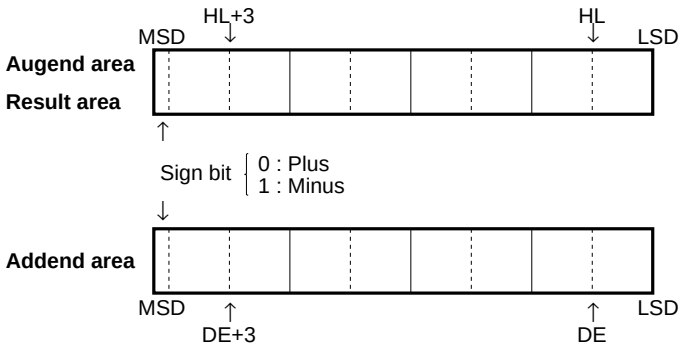
This section explains signed decimal add, subtract, multiply, and divide operations.

The most significant bit of the most significant digit is used as a sign bit, and the remaining bits represent a numeric value, which is an absolute value.

3.2.1 Decimal Addition

8 digits <- 8 digits + 8 digits

(1) Memory areas used



(2) Registers used

A, B, C, D, E, H, L

(3) Input conditions

As shown in (1) above, the memory addresses of an addend and augend are specified using the following register pairs:

Register pair HL <- Low-order address of the area where an 8-digit (4-byte) augend is stored

Register pair DE <- Low-order address of the area where an 8-digit (4-byte) addend is stored

(4) Output conditions

The result of an operation is stored in the result area shown in (1) above.

However, if an overflow or underflow occurs, a branch to error processing occurs.

Caution The range of operation is -79999999 to 79999999.

(5) Processing procedure

If an addend has the same sign as an augend, the operation program provided here performs an add operation. If an addend does not have the same sign as the augend, the operation program performs a subtract operation.

<1> Digit counter 1 (register C) is set to 4.

Digit counter 2 (register B) is set to (value of digit counter 1 - 1).

<2> If the addend does not have the same sign as the augend, processing proceeds to **<15>**.

<3> The carry flag and sign flag (user flag) are cleared.

<4> The one byte of the augend specified by an augend address is loaded into the accumulator.

<5> The one byte of the addend specified by an addend address is added together with the carry flag to the accumulator, and the addend address is incremented, A decimal adjustment is applied to the operation result, which is then stored in the memory location specified by the augend address. Then the augend address is incremented.

<6> Digit counter 2 is decremented, and the processing of **<4>** and **<5>** is repeated until digit counter 2 reaches 0.

<7> The one byte of the augend specified by the augend address is loaded into the accumulator.

<8> The one byte of the addend specified by the addend address is added together with the carry flag to the accumulator.

<9> If the carry flag is set to 0, processing proceeds to **<11>**.

<10> The sign flag is set, and the carry flag is cleared.

<11> A decimal adjustment is applied to the accumulator.

<12> If the carry flag is set to 1 or bit 7 of the accumulator is set to 1, an overflow has occurs; processing proceeds to error processing.

<13> If the sign flag is set to 1, bit 7 of the accumulator is set.

<14> The contents of the accumulator are stored in the memory location specified by the augend address, then the operation is terminated.

<15> The subtrahend is made positive, and the sign flag is cleared.

<16> If the minuend is negative, the minuend is made positive, and the sign flag is set.

<17> The carry flag is cleared.

<18> The one byte of the minuend specified by the minuend address is loaded into the accumulator.

<19> The one byte of the subtrahend specified by the subtrahend address is subtracted together with the carry flag from the accumulator, and the subtrahend address is incremented by 1. A decimal adjustment is applied to the operation result, which is then stored in the memory location specified by the minuend address. Then the minuend address is incremented.

<20> Digit counter 1 is decremented by 1, and the processing of **<18>** and **<19>** is repeated until digit counter reaches 0.

<21> If the carry flag is set to 0, processing proceeds to **<23>**.

<22> The ten's complement of the result is taken, and the sign flag is inverted.

<23> If the result is 0, the operation is terminated.

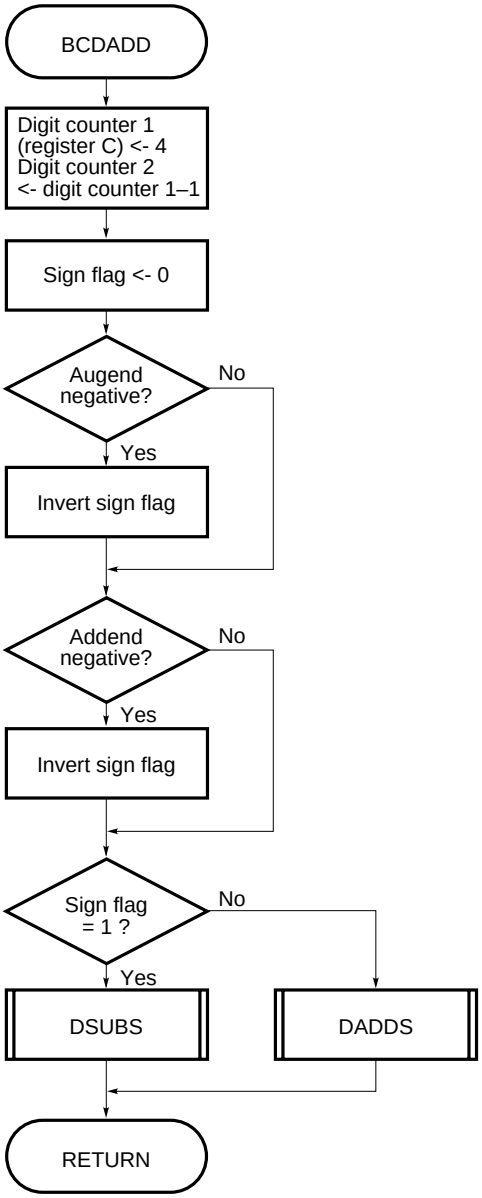
<24> If the sign flag is set to 1, processing proceeds to **<25>**. If the sign flag is set to 0, the operation is terminated.

<25> The sign bit of the result is set, then the operation is terminated.

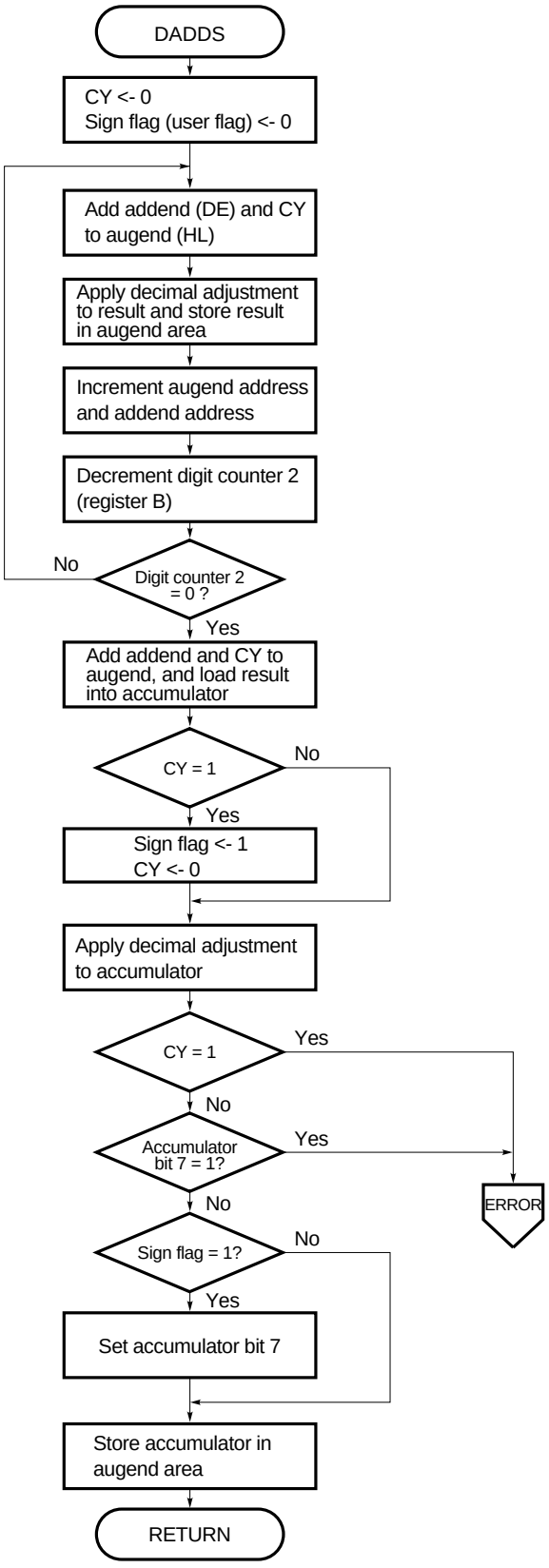
(6) Number of steps

147 bytes

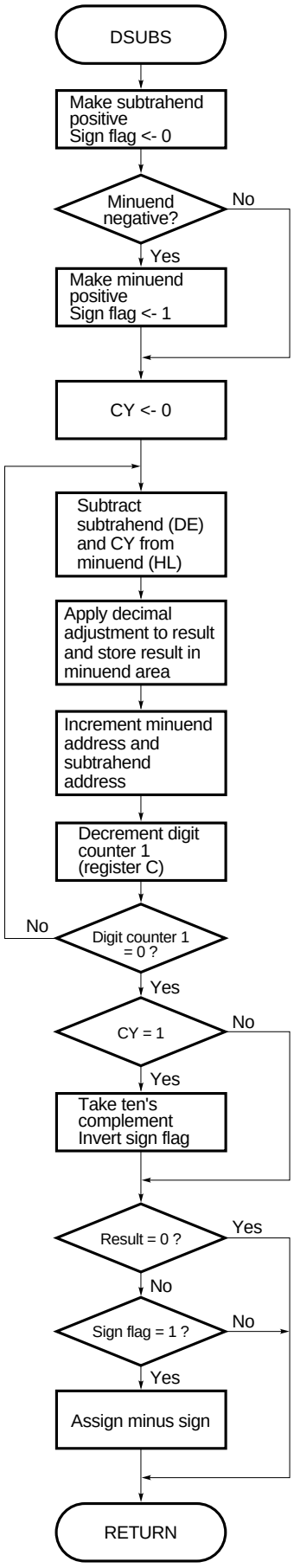
Flowchart



Flowchart



Flowchart



Assemble list

```
ALN0   STN0  ADRS   OBJECT    M  I   SOURCE      STATEMENT

   1      1                          $       TITLE   ('decimal addition')
   2      2                                  NAME    BCDADR
   3      3                          ;*******************************************************
   4      4                          ;*      decimal addition                               *
   5      5                          ;*          8 digit <- 8 digit + 8 digit               *
   6      6                          ;*       input condition                               *
   7      7                          ;*            HL-register <- augend area top.address   *
   8      8                          ;*            DE-register <- addend  area top.address  *
   9      9                          ;*       output condition                              *
  10     10                          ;*            result <- (HL+3,HL+2,HL+1,HL)            *
  11     11                          ;*                                                     *
  12     12                          ;*               RSS <- 0                              *
  13     13                          ;*******************************************************
  14     14                                  PUBLIC  BCDADD,BCDAD1,BCDAD2
  15     15                                  PUBLIC  DADDS
  16     16                                  PUBLIC  DSUBS
  17     17                                  EXTRN   ERROR
  18     18          (0004)          BYTNUM  EQU     4
  19     19          (01FF.7)        SFLAG   EQU     PSWH.7
  20     20                          ;
  21     21  ----                            CSEG
  22     22                                  RSS     0
  23     23  0000                    BCDADD:
  24     24  0000    BA04                    MOV     C,#BYTNUM       ; C-register <- 4
  25     25  0002                    BCDAD1:
  26     26  0002    2432                    MOV     B,C             ; B-register <- C-register - 1
  27     27  0004    CB                      DEC     B
  28     28  0005                    BCDAD2:
  29     29  0005    1730                    MOV     A,[HL+B]
  30     30  0007    172D                    XOR     A,[DE+B]
  31     31                          ;
  32     32  0009      03BF04                BT      A.7,$BCDAD3
  33     33  000C    R281400                 CALL    !DADDS
  34     34  000F    56                      RET
  35     35                          ;
  36     36  0010                    BCDAD3:
  37     37  0010    R283B00                 CALL    !DSUBS
  38     38  0013    56                      RET
  39     39
  40     40                                  ENDS
  41     41
  42     42                          ;=================================================
  43     43                          ;       ***** decimal addition subroutine *****
  44     44                          ;=================================================
  45     45  ----                            CSEG
  46     46                                  RSS     0
  47     47  0014                    DADDS:
  48     48  0014    40                      CLR1    CY
  49     49  0015    029F                    CLR1    SFLAG
  50     50  0017                    DADDS1:
  51     51  0017    5D                      MOV     A,[HL]
  52     52  0018    1609                    ADDC    A,[DE+]
  53     53  001A    05FE                    ADJBA                   ; decimal adjust
  54     54  001C    51                      MOV     [HL+],A
  55     55  001D    33F8                    DBNZ    B,$DADDS1
  56     56                          ;
  57     57  001F    5D                      MOV     A,[HL]
  58     58  0020    1649                    ADDC    A,[DE]
  59     59                          ;
  60     60  0022                    DADDS2:
  61     61  0022    8203                    BNC     $DADDS3
  62     62  0024    028F                    SET1    SFLAG           ; set sign-flag
  63     63  0026    40                      CLR1    CY
```

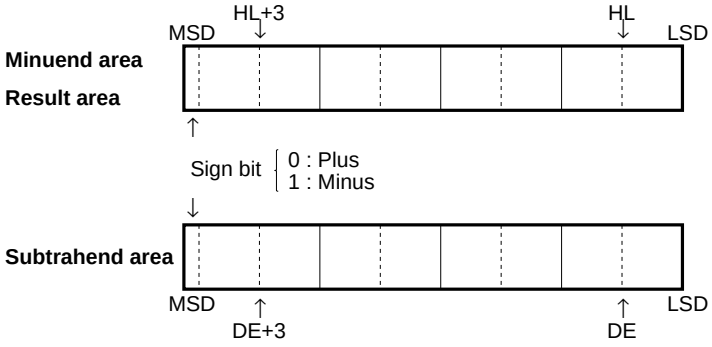
```
 64    64                          ;
 65    65  0027                    DADDS3:
 66    66  0027   05FE                      ADJBA                   ; decimal adjust
 67    67  0029   8203                      BNC     $DADDS4
 68    68  002B  R2C0000                    BR      !ERROR
 69    69  002E                    DADDS4:
 70    70  002E   03AF03                    BF      A.7,$DADDS5
 71    71  0031  R2C0000                    BR      !ERROR
 72    72                          ;
 73    73  0034                    DADDS5:
 74    74  0034   02AF02                    BF      SFLAG,$DADDS6
 75    75  0037   038F                      SET1    A.7
 76    76  0039                    DADDS6:
 77    77  0039   55                        MOV     [HL],A
 78    78  003A   56                        RET
 79    79
 80    80                                  ENDS
 81    81
 82    82                          $       EJECT
 83    83
 84    84                          ;=================================================
 85    85                          ;       ***** decimal subtraction subroutine *****
 86    86                          ;=================================================
 87    87                          ;
 88    88  ----                            CSEG
 89    89                                  RSS     0
 90    90  003B                    DSUBS:
 91    91  003B   244F                      MOVW    RP2,HL          ; save HL-register
 92    92  003D   029F                      CLR1    SFLAG           ; clear sign-flag
 93    93  003F   1720                      MOV     A,[DE+B]
 94    94  0041   039F                      CLR1    A.7
 95    95  0043   17A0                      MOV     [DE+B],A
 96    96  0045   1730                      MOV     A,[HL+B]
 97    97  0047   03AF06                    BF      A.7,$DSUBS1
 98    98                          ;
 99    99  004A   039F                      CLR1    A.7
100   100  004C   17B0                      MOV     [HL+B],A
101   101  004E   028F                      SET1    SFLAG           ; set sign-flag
102   102                          ;
103   103  0050                    DSUBS1:
104   104  0050   2462                      MOV     R6,C            ; R6 register <- C register
105   105  0052   40                        CLR1    CY
106   106  0053                    DSUBS2:
107   107  0053   5D                        MOV     A,[HL]
108   108  0054   160B                      SUBC    A,[DE+]
109   109  0056   05FF                      ADJBS
110   110  0058   51                        MOV     [HL+],A
111   111  0059   32F8                      DBNZ    C,$DSUBS2
112   112                          ;
113   113  005B   821D                      BNC     $DSUBS5
114   114  005D   24EC                      MOVW    HL,RP2          ; load HL-register
115   115  005F   2426                      MOV     C,R6            ; load C-register
116   116  0061                    DSUBS3:
117   117  0061   B999                      MOV     A,#99H          ; (HL) <- 9 - (HL)
118   118  0063   165A                      SUB     A,[HL]
119   119  0065   05FF                      ADJBS
120   120  0067   51                        MOV     [HL+],A
121   121  0068   32F7                      DBNZ    C,$DSUBS3
122   122                          ;
123   123  006A   24EC                      MOVW    HL,RP2          ; load HL-register
124   124  006C   41                        SET1    CY
125   125                          ;
126   126  006D   2426                      MOV     C,R6            ; load C-register
127   127  006F                    DSUBS4:
128   128  006F   B900                      MOV     A,#0            ; Acc <- 0
129   129  0071   1659                      ADDC    A,[HL]
130   130  0073   05FE                      ADJBA
```

```
 131    131  0075   51                      MOV     [HL+],A
 132    132  0076   32F7                    DBNZ    C,$DSUBS4
 133    133  0078   027F                    NOT1    SFLAG
 134    134                         ;
 135    135                         ;       ****  check / result = 0  ****
 136    136                         ;
 137    137  007A                   DSUBS5:
 138    138  007A   2426                    MOV     C,R6              ; load C-register
 139    139  007C   24EC                    MOVW    HL,RP2
 140    140  007E   B900                    MOV     A,#0
 141    141  0080                   DSUBS6:
 142    142  0080   169F                    CMP     [HL+],A
 143    143  0082   8003                    BNZ     $DSUBS7
 144    144  0084   32FA                    DBNZ    C,$DSUBS6
 145    145  0086   56                      RET
 146    146  0087                   DSUBS7:
 147    147  0087   02AF08                  BF      SFLAG,$DSUBS8
 148    148  008A   24EC                    MOVW    HL,RP2
 149    149  008C   1730                    MOV     A,[HL+B]
 150    150  008E   038F                    SET1    A.7
 151    151  0090   17B0                    MOV     [HL+B],A
 152    152  0092                   DSUBS8:
 153    153  0092   56                      RET
 154    154                                 ENDS
 155    155                                 END
 156    156

Segment informations:

ADRS   LEN    NAME

0000   0093H ?CSEG

          Cross-Reference List

NAME      VALUE   R  ATTR   RTYP   SEGNAME    XREFS

?CSEG               CSEG          ?CSEG          21#   45#   88#
BCDAD1      2H    R  ADDR   PUB    ?CSEG          14@   25#
BCDAD2      5H    R  ADDR   PUB    ?CSEG          14@   28#
BCDAD3     10H    R  ADDR          ?CSEG          32    36#
BCDADD      0H    R  ADDR   PUB    ?CSEG          14@   23#
BCDADR              MOD                           2#
BYTNUM      4H       NUM                          18#   24
DADDS      14H    R  ADDR   PUB    ?CSEG          15@   33    47#
DADDS1     17H    R  ADDR          ?CSEG          50#   55
DADDS2     22H    R  ADDR          ?CSEG          60#
DADDS3     27H    R  ADDR          ?CSEG          61    65#
DADDS4     2EH    R  ADDR          ?CSEG          67    69#
DADDS5     34H    R  ADDR          ?CSEG          70    73#
DADDS6     39H    R  ADDR          ?CSEG          74    76#
DSUBS      3BH    R  ADDR   PUB    ?CSEG          16@   37    90#
DSUBS1     50H    R  ADDR          ?CSEG          97   103#
DSUBS2     53H    R  ADDR          ?CSEG         106#  111
DSUBS3     61H    R  ADDR          ?CSEG         116#  121
DSUBS4     6FH    R  ADDR          ?CSEG         127#  132
DSUBS5     7AH    R  ADDR          ?CSEG         113   137#
DSUBS6     80H    R  ADDR          ?CSEG         141#  144
DSUBS7     87H    R  ADDR          ?CSEG         143   146#
DSUBS8     92H    R  ADDR          ?CSEG         147   152#
ERROR     ----H   E         EXT                  17@   68    71
SFLAG    1FFH.7      RBIT                        19#   49    62    74      92      101     133     147

Target chip:uPD78320
Assembly complete,  0 error(s) and           0 warning(s) found. (   0)
```

3.2.2 Decimal Subtraction

8 digits <- 8 digits - 8 digits

(1) Memory areas used



(2) Registers used

A, B, C, D, E, H, L

(3) Input conditions

As shown in (1) above, the memory addresses of a subtrahend and minuend are specified using the following register pairs:

Register pair HL <- Low-order address of the area where an 8-digit (4-byte) minuend is stored

Register pair DE <- Low-order address of the area where an 8-digit (4-byte) subtrahend is stored

(4) Output conditions

The result of an operation is stored in the result area shown in (1) above.

However, if an overflow or underflow occurs, a branch to error processing occurs.

Caution The range of operation is -79999999 to 79999999.

(5) Processing procedure

The operation program provided here performs a subtract operation by converting (minuend - subtrahend) to [minuend + (- subtrahend)].

The processing procedure is described below.

<1> The digit counter (register C) is set to 4.

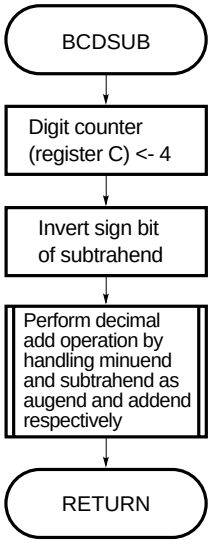
<2> The sign bit of the subtrahend is inverted.

<3> By handling the minuend and subtrahend as an augend and addend, a decimal add operation is performed.

(6) Number of steps

15 bytes

Flowchart

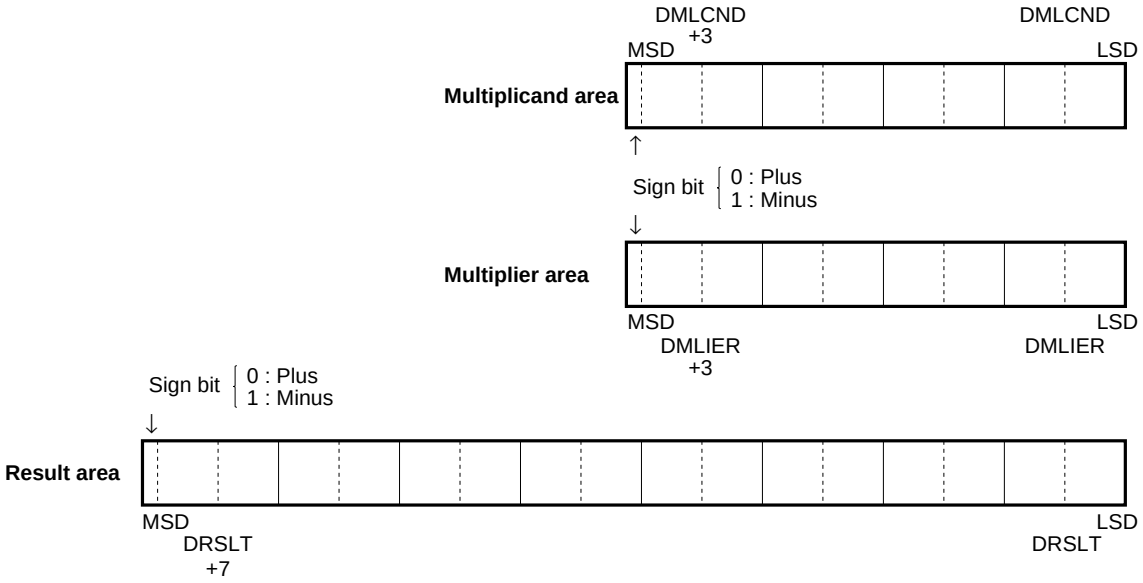


```
      Assemble list

 ALN0   STN0  ADRS    OBJECT    M  I   SOURCE STATEMENT

    1      1                          $       TITLE (‘decimal subtraction’)
    2      2                                  NAME    BCDSUR
    3      3                          ;*************************************************************
    4      4                          ;*      decimal subtraction                                  *
    5      5                          ;*          8 digit <- 8 digit - 8 digit                     *
    6      6                          ;*       input condition                                     *
    7      7                          ;*            HL-register <- minuend   area top.address     *
    8      8                          ;*           DE-register <- subtrahend area top.address     *
    9      9                          ;*       output condition                                    *
   10     10                          ;*            result <- (HL+3,HL+2,HL+1,HL)                  *
   11     11                          ;*                                                           *
   12     12                          ;*              RSS <- 0                                     *
   13     13                          ;*************************************************************
   14     14                          ;
   15     15                                  PUBLIC  BCDSUB
   16     16                                  EXTRN   BCDADD,BCDAD2
   17     17          (0004)          BYTNUM  EQU     4
   18     18                          ;
   19     19  ----                            CSEG
   20     20                                  RSS     0
   21     21  0000                    BCDSUB:
   22     22  0000    BA04                    MOV     C,#BYTNUM       ; C-register <- 4
   23     23  0002                    BCDSU1:
   24     24  0002    2432                    MOV     B,C             ; B-register <- C-register - 1
   25     25  0004    CB                      DEC     B
   26     26                          ;
   27     27  0005    1720                    MOV     A,[DE+B]
   28     28  0007    037F                    NOT1    A.7
   29     29  0009    17A0                    MOV     [DE+B],A
   30     30                          ;
   31     31  000B    R280000                 CALL    !BCDAD2
   32     32                          ;
   33     33  000E    56                      RET
   34     34                                  ENDS
   35     35                                  END

Segment informations:

ADRS   LEN    NAME

0000   000FH  ?CSEG


         Cross-Reference List

NAME      VALUE   R  ATTR   RTYP   SEGNAME    XREFS

?CSEG               CSEG          ?CSEG           19#
BCDAD2    ----H   E          EXT                  16@   31
BCDADD    ----H   E          EXT                  16@
BCDSU1       2H   R  ADDR          ?CSEG           23#
BCDSUB       0H   R  ADDR   PUB    ?CSEG           15@   21#
BCDSUR              MOD                           2#
BYTNUM       4H     NUM                           17#   22

Target chip:uPD78320
Assembly complete,      0 error(s) and        0 warning(s) found. (   0)
```

3.2.3 Decimal Multiplication

16 digits <- 8 digits x 8 digits

(1) Memory areas used



(2) Registers used

A, C, B, R4, D, E, H, L

(3) Input conditions

As shown in (1) above, an 8-digit multiplicand and 8- digit multiplier are stored in the following memory areas:

Multiplicand (DMLCND, DMLCND+1, DMLCND+2, DMLCND+3)

Multiplier (DMLIER, DMLIER+1, DMLIER+2, DMLIER+3)

(4) Output conditions

The result of an operation is stored in the result area (DRSLT, DRSLT+1, ..., DRSLT+7).

Cautions 1. The effective range of a multiplicand and multiplier is -79999999 to 79999999.

2. The range of operation results is -6399999840000001 to 6399999840000001.

(5) Processing procedure

The operation program provided here loads a multiplier digit by digit starting with the least significant digit into the addition counter by shifting the multiplicand one digit (4 bits) to the right at a time. By using the addition counter as a counter, the program repeats (result <- result + multiplicand).

When an add operation using the addition counter is completed, an add operation is performed with the multiplier digit which is one digit higher. So the result area is shifted one digit (4 bits) to the right beforehand.

The processing procedure is described below.

<1> The result area is cleared to 0 beforehand.

<2> The absolute values of the multiplier and multiplicand are found. If the sign of the multiplier is different from that of the multiplicand, the sign flag (user flag) is set. If the two signs match, the sign flag is reset.

<3> The digit counter (register B) is set to 8.

<4> By shifting the multiplier one digit to the right, the least significant digit is loaded into the addition counter (register C).

<5> The digit count (R4) is cleared.

<6> If the addition counter is set to 0, processing proceeds to **<9>**.

<7> A decimal add operation [(result (higher 8 digits) <- result (higher 8 digits) + multiplicand] is performed. If an overflow occurs, the carry count is incremented by 1.

<8> The addition counter is decremented by 1, and the processing of **<7>** is repeated until the addition counter reaches 0.

<9> The result area including the carry count is shifted one digit (4 bits) to the right. The carry count is stored in the most significant digit of the result area.

<10> The digit counter is decremented by 1, and the processing of **<4>** to **<9>** is repeated until the digit counter reaches 0.

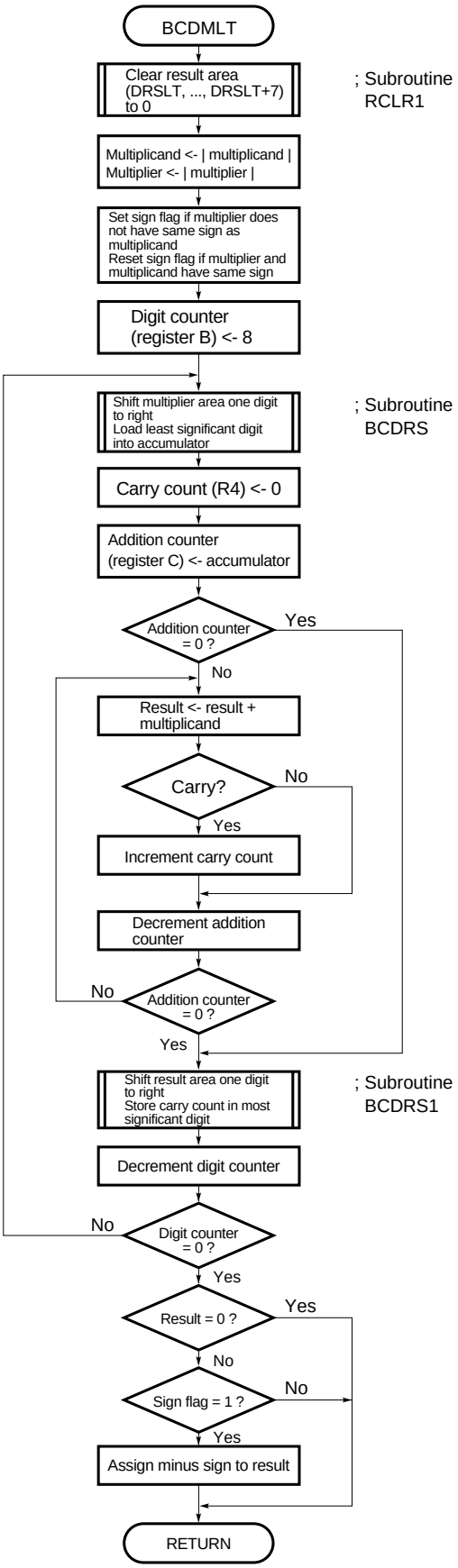
<11> If the result is 0, the operation is terminated.

<12> If the sign flag is set to 1, the sign bit of the result area is set.

(6) Number of steps

121 bytes

Flowchart



Assemble list

```
ALN0   STN0  ADRS   OBJECT    M  I   SOURCE     STATEMENT

   1      1                          $       TITLE ('decimal multiplication')
   2      2                                  NAME    BCDMLR
   3      3                          ;*******************************************************
   4      4                          ;*      decimal multiplication                         *
   5      5                          ;*          16 digit <- 8 digit * 8 digit              *
   6      6                          ;*       input condition                               *
   7      7                          ;*           multiplicand <- (DMLCND+3,...,DMLCND)     *
   8      8                          ;*           multiplier  <- (DMLIER+3,...,DMLIER)      *
   9      9                          ;*       output condition                              *
  10     10                          ;*            result <- (DRSLT+7,...,DRSLT)            *
  11     11                          ;*                                                     *
  12     12                          ;*              RSS <- 0                               *
  13     13                          ;*******************************************************
  14     14                          ;
  15     15                                  PUBLIC  BCDMLT
  16     16                                  EXTRN   RCLR1,BCDRS,BCDRS1
  17     17                                  EXTRN   DMLCND,DMLIER,DRSLT
  18     18          (01FF.7)        SFLAG   EQU     PSWH.7
  19     19                          ;
  20     20  ----                            CSEG
  21     21                                  RSS     0
  22     22  0000                    BCDMLT:
  23     23                          ;
  24     24                          ;       **** result area 0-clear  ****
  25     25                          ;
  26     26  0000    BA08                    MOV     C,#8            ; C-register <- 8
  27     27  0002   R650000                  MOVW    DE,#DRSLT       ; DE <- DRSLT
  28     28  0005   R280000                  CALL    !RCLR1
  29     29                          ;
  30     30                          ;       ****  check / sign  ****
  31     31                          ;
  32     32  0008    029F                    CLR1    SFLAG           ; clear sign-flag
  33     33  000A   R09F00300                MOV     A,!DMLCND+3
  34     34  000E    03AF08                  BF      A.7,$BCDML1
  35     35  0011    039F                    CLR1    A.7
  36     36  0013   R09F10300                MOV     !DMLCND+3,A
  37     37  0017    027F                    NOT1    SFLAG           ; not sign-flag
  38     38                          ;
  39     39  0019                    BCDML1:
  40     40  0019   R09F00300                MOV     A,!DMLIER+3
  41     41  001D    03AF08                  BF      A.7,$BCDML2
  42     42  0020    039F                    CLR1    A.7
  43     43  0022   R09F10300                MOV     !DMLIER+3,A
  44     44  0026    027F                    NOT1    SFLAG           ; not sign-flag
  45     45                          ;
  46     46                          ;       ****  digit counter set  ****
  47     47                          ;
  48     48  0028                    BCDML2:
  49     49  0028    BB08                    MOV     B,#8            ; B-register <- 8
  50     50                          ;
  51     51                          ;       ****  multiplier right shift  ****
  52     52                          ;
  53     53  002A                    BCDML3:
  54     54  002A   R670300                  MOVW    HL,#DMLIER+3
  55     55  002D    BA04                    MOV     C,#4            ; C-register <- 4
  56     56  002F   R280000                  CALL    !BCDRS
  57     57  0032    2421                    MOV     C,A             ; C-register <- Acc
  58     58                          ;
  59     59  0034    BC00                    MOV     R4,#0           ; carry <- 0
  60     60                          ;
  61     61                          ;       ****  check / multiplier = 0 ?  ****
  62     62                          ;
  63     63  0036    A800                    ADD     A,#0
  64     64  0038    8118                    BZ      $BCDML7
```

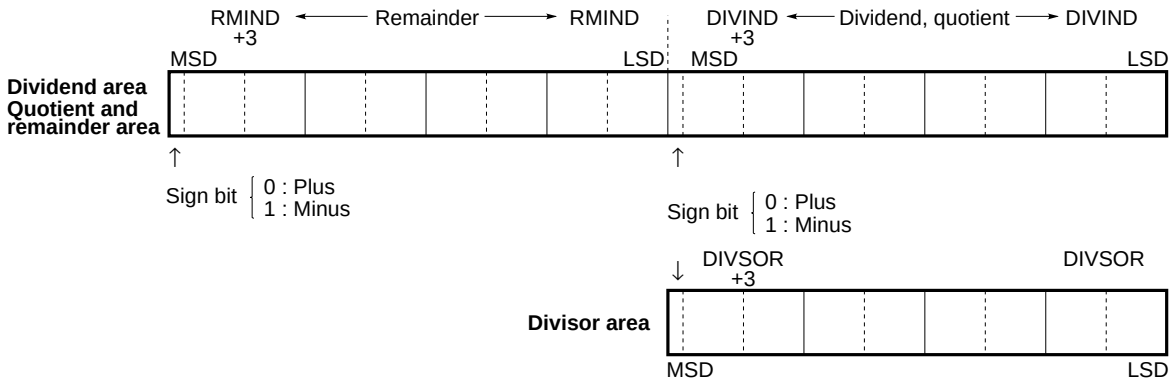
```
  65     65                          ;
  66     66                          ;       ****  result <- DMLCND + result  ****
  67     67                          ;
  68     68  003A                    BCDML4:
  69     69  003A  R650000                    MOVW    DE,#DMLCND      ; DE <- DMLCND
  70     70  003D  R670400                    MOVW    HL,#DRSLT+4     ; HL <- DRSLT+4
  71     71  0040   40                        CLR1    CY              ; clear carry
  72     72                                   RSS     1
  73     73  0041   43                        SWRS
  74     74  0042   BE04                      MOV     C,#4            ; C-register <- 4
  75     75                          ;
  76     76  0044                    BCDML5:
  77     77  0044   5D                        MOV     A,[HL]
  78     78  0045   1609                      ADDC    A,[DE+]
  79     79  0047   05FE                      ADJBA                   ; decimal adjust
  80     80  0049   51                        MOV     [HL+],A
  81     81  004A   32F8                      DBNZ    C,$BCDML5
  82     82                                   RSS     0
  83     83  004C   43                        SWRS
  84     84  004D   8201                      BNC     $BCDML6
  85     85  004F   C4                        INC     R4
  86     86  0050                    BCDML6:
  87     87  0050   32E8                      DBNZ    C,$BCDML4
  88     88                          ;
  89     89                          ;       ****  result right shift with carry  ****
  90     90                          ;
  91     91  0052                    BCDML7:
  92     92  0052   D4                        MOV     A,R4
  93     93  0053  R670700                    MOVW    HL,#DRSLT+7     ; HL <- DRSLT+7
  94     94  0056   BA08                      MOV     C,#8
  95     95  0058  R280000                    CALL    !BCDRS1
  96     96                          ;
  97     97                          ;       ****  check / multiply end ?  ****
  98     98                          ;
  99     99  005B   33CD                      DBNZ    B,$BCDML3
 100    100                          ;
 101    101                          ;       ****  check / multiply = 0  ****
 102    102                          ;
 103    103  005D  R670000                    MOVW    HL,#DRSLT
 104    104  0060   BA08                      MOV     C,#8
 105    105  0062   B900                      MOV     A,#0
 106    106  0064                    BCDML8:
 107    107  0064   169F                      CMP     [HL+],A
 108    108  0066   8003                      BNZ     $BCDML9
 109    109  0068   32FA                      DBNZ    C,$BCDML8
 110    110  006A   56                        RET
 111    111                          ;
 112    112                          ;       ****  check / sign-flag  ****
 113    113                          ;
 114    114  006B                    BCDML9:
 115    115  006B   02AF0A                    BF      SFLAG,$BCDM10
 116    116  006E  R09F00700                  MOV     A,!DRSLT+7
 117    117  0072   038F                      SET1    A.7
 118    118  0074  R09F10700                  MOV     !DRSLT+7,A
 119    119                          ;
 120    120  0078                    BCDM10:
 121    121  0078   56                        RET
 122    122                                   ENDS
 123    123                                   END

Segment informations:

ADRS   LEN    NAME

0000   0079H  ?CSEG
```

```
         Cross-Reference List

NAME      VALUE   R  ATTR   RTYP   SEGNAME    XREFS

?CSEG               CSEG          ?CSEG         20#
BCDM10      78H   R  ADDR          ?CSEG        115    120#
BCDML1      19H   R  ADDR          ?CSEG         34     39#
BCDML2      28H   R  ADDR          ?CSEG         41     48#
BCDML3      2AH   R  ADDR          ?CSEG         53#    99
BCDML4      3AH   R  ADDR          ?CSEG         68#    87
BCDML5      44H   R  ADDR          ?CSEG         76#    81
BCDML6      50H   R  ADDR          ?CSEG         84     86#
BCDML7      52H   R  ADDR          ?CSEG         64     91#
BCDML8      64H   R  ADDR          ?CSEG        106#   109
BCDML9      6BH   R  ADDR          ?CSEG        108    114#
BCDMLR              MOD                          2#
BCDMLT       0H   R  ADDR   PUB    ?CSEG         15@    22#
BCDRS     ----H   E         EXT                 16@    56
BCDRS1    ----H   E         EXT                 16@    95
DMLCND    ----H   E         EXT                 17@    33    36      69
DMLIER    ----H   E         EXT                 17@    40    43      54
DRSLT     ----H   E         EXT                 17@    27    70      93    103    116    118
RCLR1     ----H   E         EXT                 16@    28
SFLAG    1FFH.7      RBIT                       18#    32    37      44    115

Target chip:uPD78320
Assembly complete,        0 error(s) and    0 warning(s) found. (    0)
```

3.2.4 Decimal Division

Quotient : 8 digits <- 8 digits/8 digits
Remainder : 8 digits

(1) Memory areas used



(2) Registers used

X, A, B, C, D, E, H, L

(3) Input conditions

An 8-bit dividend and 8-bit divisor are stored in the following memory areas:
Dividend (DIVIND, DIVIND+1, DIVIND+2, DIVIND+3)
Divisor (DIVSOR, DIVSOR+1, DIVSOR+2, DIVSOR+3)

(4) Output conditions

The result of an operation is stored in the quotient and remainder areas shown in (1) above.
When a divisor of 0 is specified, a branch to error processing occurs.

Caution The range of operation is -79999999 to 79999999.

(5) Processing procedure

The operation program provided here uses an 8-byte contiguous area for a dividend (DIVIND, DIVIND+1, DIVIND+2, DIVIND+3) and remainder (RMIND, RMIND+1, RMIND+2, RMIND+3). By shifting the dividend and remainder area one digit (4 bits) to the left, the most significant digit of the dividend is moved to the least significant digit position of the remainder, and the quotient, starting with the most significant digit, enters the least significant digit position of the dividend digit by digit.
The digits of of a quotient represent the number of repetitions occurring until the result of (remainder - dividend) becomes negative (until a borrow occurs).
The processing procedure is described below.

<1> A check is made to see if the divisor is 0. If the divisor is 0, a branch to error processing occurs.

<2> The remainder area (RMIND, RMIND+1, RMIND+2, RMIND+3) is cleared to 0.

<3> The absolute values of the dividend and divisor are found. If the divisor does not have the same sign as the dividend, the quotient sign flag (bit 0 of X) is set. If the divisor and dividend have the same sign, the quotient sign flag is reset.

If the dividend is negative, the remainder sign flag (bit 1 of X) is set. If the dividend is positive, the remainder sign flag is reset.

<4> The quotient digit counter (register B) is set to 8.

<5> The contiguous 8-byte quotient and remainder area is shifted one digit (four bits) to the left.

<6> A decimal subtract operation (remainder <- remainder - divisor) is performed. If the result is negative, processing proceeds to **<8>**.

<7> The quotient (DIVIND) is incremented by 1. Processing proceeds to **<6>**.

<8> An excessive subtraction was performed. So a decimal add operation (remainder <- remainder + divisor) is performed.

<9> The quotient digit counter is decremented by 1, and the processing of **<5>** to **<8>** is repeated until the quotient digit counter reaches 0.

<10> If the quotient is 0, processing proceeds to **<12>**.

<11> If the quotient sign flag is set to 1, the sign bit of the quotient area is set.

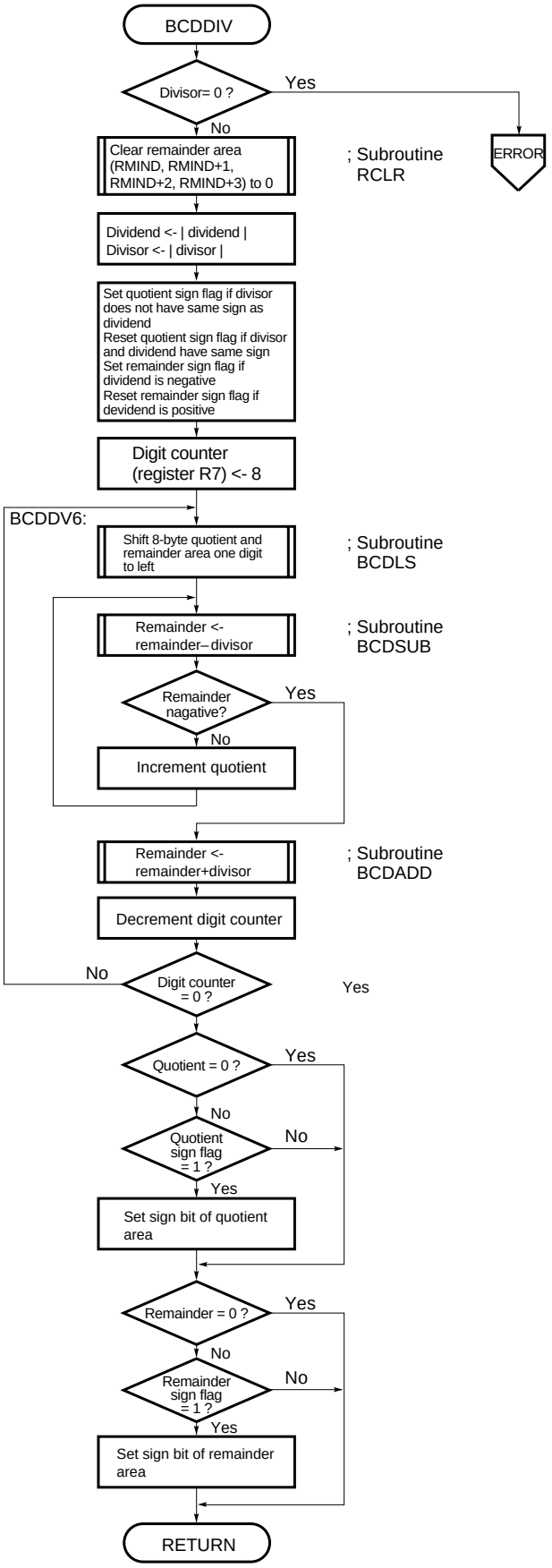
<12> If the remainder is 0, the operation is terminated.

<13> If the remainder sign flag is set to 1, the sign bit of the remainder area is set.

(6) Number of steps

164 bytes

Flowchart



Assemble list

```
 ALN0   STN0  ADRS    OBJECT  M  I      SOURCE STATEMENT

    1      1                          $      TITLE ('decimal division')
    2      2                                 NAME    BCDIVR
    3      3                          ;*************************************************
    4      4                          ;*      decimal division                         *
    5      5                          ;*          8 digit <- 8 digit / 8 digit         *
    6      6                          ;*      input condition                          *
    7      7                          ;*              dividend <- (DIVIND+3,...,DIVIND)    *
    8      8                          ;*              divisor <- (DIVSOR+3,...,DIVSOR)    *
    9      9                          ;*      output condition                         *
   10     10                          ;*              quotient <- (DIVIND+3,...,DIVIND)    *
   11     11                          ;*              remainder <- (RMIND+3,...,RMIND)     *
   12     12                          ;*                                               *
   13     13                          ;*              RSS <- 0                         *
   14     14                          ;*************************************************
   15     15                                 PUBLIC  BCDDIV
   16     16                                 EXTRN   ERROR,RCLR,BCDLS,BCDSUB,BCDADD
   17     17                                 EXTRN   DIVIND,DIVSOR,RMIND
   18     18          (0000.0)         SF_QUO  EQU     X.0
   19     19          (0000.1)         SF_REM  EQU     X.1
   20     20                           ;
   21     21  ----                           CSEG
   22     22                                 RSS     0
   23     23  0000                     BCDDIV:
   24     24                           ;
   25     25                           ;       ****  check / divisor = 0 ?  ****
   26     26                           ;
   27     27  0000    BA04                   MOV     C,#4            ; C-register <- 4
   28     28  0002    B900                   MOV     A,#0            ; Acc <- 0
   29     29  0004  R670000                 MOVW    HL,#DIVSOR      ; HL <- DIVSOR
   30     30  0007                     BCDDV1:
   31     31  0007    169F                   CMP     [HL+],A
   32     32  0009    8005                   BNZ     $BCDDV2         ; (HL) = 0 ?
   33     33  000B    32FA                   DBNZ    C,$BCDDV1
   34     34                           ;
   35     35  000D  R2C0000                 BR      !ERROR          ; OVER FLOW
   36     36                           ;
   37     37                           ;       **** result,remind 0-clear ****
   38     38                           ;
   39     39  0010                     BCDDV2:
   40     40  0010  R650000                 MOVW    DE,#RMIND       ; DE <- RMIND
   41     41  0013  R280000                 CALL    !RCLR
   42     42                           ;
   43     43                           ;       ****  check / sign  ****
   44     44                           ;
   45     45  0016  R670300                 MOVW    RP7,#DIVSOR+3
   46     46  0019    0390                   CLR1    SF_QUO          ; clear quotient sign-flag
   47     47  001B    0391                   CLR1    SF_REM          ; clear remainder sign-flag
   48     48  001D  R09F00300                MOV     A,!DIVIND+3
   49     49  0021    03AF0A                 BF      A.7,$BCDDV3
   50     50  0024    039F                   CLR1    A.7
   51     51  0026  R09F10300                MOV     !DIVIND+3,A
   52     52  002A    0381                   SET1    SF_REM          ; set remainder sign-flag
   53     53  002C    0370                   NOT1    SF_QUO          ; not quotient sign-flag
   54     54  002E                     BCDDV3:
   55     55  002E  R09F00300                MOV     A,!DIVSOR+3
   56     56  0032    03AF08                 BF      A.7,$BCDDV4
   57     57  0035    039F                   CLR1    A.7
   58     58  0037  R09F10300                MOV     !DIVSOR+3,A
   59     59  003B    0370                   NOT1    SF_QUO
   60     60                           ;
   61     61                           ;       ****  digit counter set  ****
   62     62                           ;
```

```
 63    63  003D                    BCDDV4:
 64    64  003D   BF08                     MOV     R7,#8
 65    65                         ;
 66    66                         ;       **** quotient,remind left sift ****
 67    67                         ;
 68    68  003F                    BCDDV5:
 69    69  003F  R670000                   MOVW    HL,#DIVIND      ; HL <- DIVIND
 70    70  0042   BA08                     MOV     C,#16/2         ; C-register <- 8
 71    71  0044  R280000                   CALL    !BCDLS          ; N-digit data left shift
 72    72                         ;
 73    73                         ;       **** subtract divisor from dividend ****
 74    74                         ;
 75    75  0047                    BCDDV6:
 76    76  0047  R650000                   MOVW    DE,#DIVSOR      ; DE <- DIVSOR
 77    77  004A  R670000                   MOVW    HL,#RMIND       ; HL <- RMIND
 78    78  004D  R280000                   CALL    !BCDSUB         ; decimal subtraction
 79    79                         ;
 80    80  0050  R09F00300                MOV     A,!RMIND+3
 81    81  0054   03BF09                   BT      A.7,$BCDDV7     ; if borrow then goto BCDDV7
 82    82                         ;
 83    83  0057   B901                     MOV     A,#1
 84    84  0059  R670000                   MOVW    HL,#DIVIND
 85    85  005C   16D8                     ADD     [HL],A          ; increment (DIVIND)
 86    86                         ;
 87    87  005E   14E7                     BR      $BCDDV6
 88    88                         ;
 89    89                         ;       **** if borrow then divisor + dividend ****
 90    90                         ;
 91    91  0060                    BCDDV7:
 92    92  0060  R650000                   MOVW    DE,#DIVSOR      ; DE <- DIVSOR
 93    93  0063  R670000                   MOVW    HL,#RMIND       ; HL <- RMIND
 94    94  0066  R280000                   CALL    !BCDADD         ; decimal addition
 95    95                         ;
 96    96                         ;       **** check / division end ?  ****
 97    97                         ;
 98    98  0069   CF                       DEC     R7
 99    99  006A   80D3                     BNZ     $BCDDV5
100   100                         ;
101   101                         ;       **** check / quotient = 0  ****
102   102                         ;
103   103  006C  R670000                   MOVW    HL,#DIVIND
104   104  006F   B900                     MOV     A,#0
105   105  0071   BA04                     MOV     C,#4
106   106  0073                    BCDDV8:
107   107  0073   169F                     CMP     [HL+],A
108   108  0075   8004                     BNZ     $BCDDV9
109   109  0077   32FA                     DBNZ    C,$BCDDV8
110   110  0079   140D                     BR      $BCDV10
111   111                         ;
112   112                         ;       **** check / quotient sign-flag  ****
113   113                         ;
114   114  007B                    BCDDV9:
115   115  007B   03A00A                   BF      SF_QUO,$BCDV10
116   116  007E  R09F00300                MOV     A,!DIVIND+3
117   117  0082   038F                     SET1    A.7
118   118  0084  R09F10300                MOV     !DIVIND+3,A
119   119                         ;
120   120                         ;       **** check / remainder = 0  ****
121   121                         ;
122   122  0088                    BCDV10:
123   123  0088  R670000                   MOVW    HL,#RMIND
124   124  008B   B900                     MOV     A,#0
125   125  008D   BA04                     MOV     C,#4
126   126  008F                    BCDV11:
127   127  008F   169F                     CMP     [HL+],A
128   128  0091   8003                     BNZ     $BCDV12
129   129  0093   32FA                     DBNZ    C,$BCDV11
```

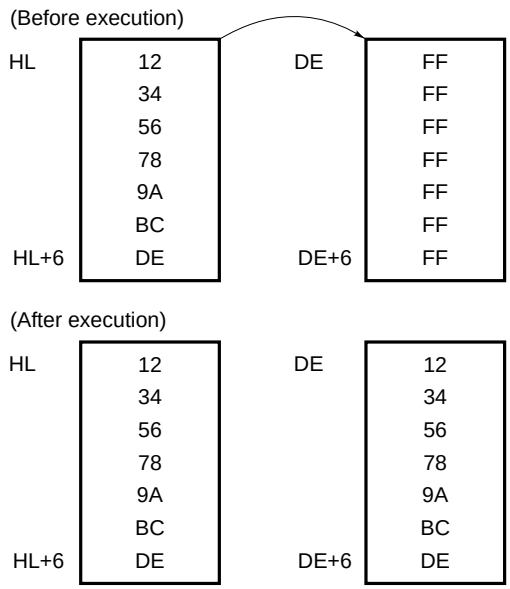
```
 130    130  0095    56                        RET
 131    131                          ;
 132    132                          ;       **** check / remainder sign-flag ****
 133    133                          ;
 134    134  0096                    BCDV12:
 135    135  0096    03A10A                    BF      SF_REM,$BCDV13
 136    136  0099   R09F00300                  MOV     A,!RMIND+3
 137    137  009D    038F                      SET1    A.7
 138    138  009F   R09F10300                  MOV     !RMIND+3,A
 139    139  00A3                    BCDV13:
 140    140  00A3    56                        RET
 141    141                                    ENDS
 142    142                                    END

Segment informations:

ADRS   LEN    NAME:

0000   00A4H  ?CSEG

          Cross-Reference List

NAME      VALUE   R  ATTR   RTYP   SEGNAME   XREFS

?CSEG               CSEG          ?CSEG        21#
BCDADD     ----H  E           EXT               16@     94
BCDDIV    0H       R  ADDR   PUB    ?CSEG        15@    23#
BCDDV1    7H       R  ADDR          ?CSEG        30#     33
BCDDV2    10H      R  ADDR          ?CSEG        32     39#
BCDDV3    2EH      R  ADDR          ?CSEG        49     54#
BCDDV4    3DH      R  ADDR          ?CSEG        56     63#
BCDDV5    3FH      R  ADDR          ?CSEG        68#     99
BCDDV6    47H      R  ADDR          ?CSEG        75#     87
BCDDV7    60H      R  ADDR          ?CSEG        81     91#
BCDDV8    73H      R  ADDR          ?CSEG        106#   109
BCDDV9    7BH      R  ADDR          ?CSEG        108   114#
BCDIVR              MOD                          2#
BCDLS     ----H    E           EXT               16@     71
BCDSUB    ----H    E           EXT               16@     78
BCDV10    88H      R  ADDR          ?CSEG        110    115     122#
BCDV11    8FH      R  ADDR          ?CSEG        126#   129
BCDV12    96H      R  ADDR          ?CSEG        128   134#
BCDV13    A3H      R  ADDR          ?CSEG        135   139#
DIVIND    ----H    E           EXT               17@     48     51    69       84       103      116      11
8
DIVSOR    ----H    E           EXT               17@     29     45    55       58       76       92
ERROR     ----H    E           EXT               16@     35
RCLR      ----H    E           EXT               16@     41
RMIND     ----H    E           EXT               17@     40     77    80       93       123      136      13
8
SF_QU0     0H.0     RBIT                         18#     46     53    59       115
SF_REM    0H.1      RBIT                         19#     47     52    135

Target chip:uPD78320
Assembly complete,  0 error(s) and          0 warning(s) found. (   0)
```

3.3 DATA TRANSFER

This section provides an example of transferring data from an arbitrary area to another arbitrary area in memory.

Example As shown in Figure 3-1, 7-byte memory data specified by HL to HL+6 in memory to an area specified by DE to DE+6.

Figure 3-1. Data Transfer



With the μPD78320, N-byte transfer can be performed using the string instruction (MOVBK).

Before execution of transfer, the high-order address or low-order address of a transfer source and the high-order address or low-order address of a transfer destination must be set in registers HL and DE, respectively. In addition, the number of transfer bytes must be set in register C beforehand.

However, no more than 256 bytes are transferred.

In the sample program provided here, 1000H is loaded into register HL, 1100H is loaded into register DE, and 8 is loaded into register C.

Number of steps: 12 bytes

```
      Assemble list

 ALN0   STN0  ADRS    OBJECT    M   I   SOURCE      STATEMENT

    1      1                          $       TITLE   (‘data transmission’)
    2      2                                  NAME    TRANSR
    3      3                          ;*************************************************
    4      4                          ;*      data transmission                        *
    5      5                          ;*************************************************
    6      6
    7      7                          ;===============================================
    8      8                          ;       data table area
    9      9                          ;===============================================
   10     10
   11     11          (1000)          ADDRS1  EQU     1000H
   12     12          (2000)          ADDRS2  EQU     2000H
   13     13
   14     14                          ;===============================================
   15     15                          ;               data transmission routine
   16     16                          ;===============================================
   17     17  ----                            CSEG
   18     18                                  RSS     0
   19     19                          ;
   20     20  0000    670010                  MOVW    HL,#ADDRS1
   21     21  0003    650020                  MOVW    DE,#ADDRS2
   22     22                          ;
   23     23  0006    BA07                    MOV     C,#7          ; counter set
   24     24
   25     25  0008    1520                    MOVBK   [DE+],[HL+]
   26     26
   27     27  000A                      DTL1:
   28     28  000A    14FE                    BR      $DTL1
   29     29                                  ENDS
   30     30                                  END
   31     31

Segment informations:

ADRS   LEN    NAME

0000   000CH ?CSEG

          Cross-Reference List

NAME       VALUE R  ATTR   RTYP   SEGNAME    XREFS

?CSEG               CSEG          ?CSEG          17#
ADDRS1    1000H     NUM                          11#     20
ADDRS2    2000H     NUM                          12#     21
DTL1         AH   R  ADDR          ?CSEG          27#     28
TRANSR              MOD                           2#

Target chip:uPD78320
Assembly complete,      0 error(s) and     0 warning(s) found. (    0)
```

3.4 SHIFT PROCESSING

With the μPD78320, two types of shift instructions are available for the registers (R0, R1, ..., R15) and the register pairs (RP0, RP1, ..., RP7). One type includes 1-bit shift instructions (ROR, ROL, etc.), and the other type includes 4-bit shift instructions (ROR4, ROL4).

The sample programs provided here introduce two types of shift operations:

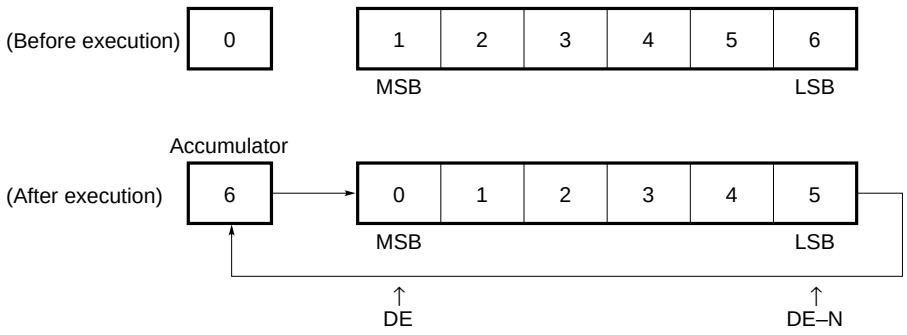
- Shift operations byte by byte
- Shift operations in units of four bits

3.4.1 Shifting N-Byte Data to Right

An example is provided which shifts N-byte data in memory to the right.

After the program is executed, a value loaded beforehand into the accumulator is stored in the most significant byte position, and the value of the least significant byte position is output to the accumulator as shown below.

(1) Memory areas used



(2) Registers used

A, C, D, E

(3) Input conditions

Data required for this processing is loaded into the following registers:

Accumulator <- Value to be transferred to the most significant byte position in memory

Register C <- Number of bytes (N)

Register DE <- High-order address of N-byte data

(4) Output conditions

The result of a 1-byte right shift is stored in the result area.

The contents of the accumulator are transferred to the most significant byte position in memory.

Accumulator <- Contents of LSB

(5) Number of steps

5 bytes

Assemble list

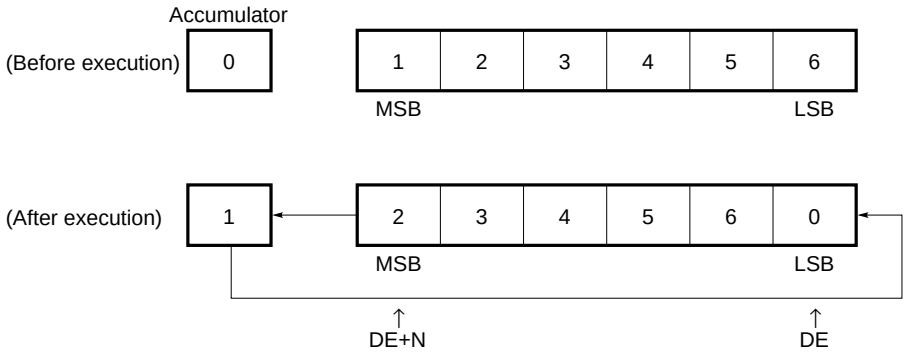
```
ALN0  STN0 ADRS   OBJECT   M  I  SOURCE      STATEMENT

   1     1                       $       TITLE   ('N-byte data left shift')
   2     2                               NAME    BYTRSR
   3     3                       ;*************************************************
   4     4                       ;*      N-byte data right shift                  *
   5     5                       ;*         input condition                       *
   6     6                       ;*              HL-register <- MSD of N-digit data *
   7     7                       ;*              C-register  <- digit counter       *
   8     8                       ;*         output condition                      *
   9     9                       ;*              Acc <- LSD of N-digit data        *
  10    10                       ;*                                                *
  11    11                       ;*              RSS <- 0                          *
  12    12                       ;*************************************************
  13    13                               PUBLIC  BYTRST,BYTRS1
  14    14                               PUBLIC  BYTLST,BYTLS1
  15    15                       ;
  16    16  ----                         CSEG
  17    17                               RSS     0
  18    18  0000                 BYTRST:
  19    19  0000   B900                  MOV     A,#0          ; Acc <- 0
  20    20  0002                 BYTRS1:
  21    21  0002   1511                  XCHM    [DE-],A
  22    22                       ;
  23    23  0004   56                    RET
  24    24                               ENDS
```

3.4.2 Shifting N-Byte Data to Left

An example is provided which shifts N-byte data in memory to the right.

After the program is executed, a value loaded beforehand into the accumulator is stored in the least significant byte position, and the value of the most significant byte is output to the accumulator as shown below.

(1) Memory areas used



(2) Registers used

A, C, D, E

(3) Input conditions

Data required for this processing is loaded into the following registers:

Accumulator <- Value to be transferred to the least significant byte position in memory

Register C <- Number of bytes (N)

Register DE <- Low-order address of N-byte data

(4) Output conditions

The result of a 1-byte left shift is stored in the result area.

The contents of the accumulator are transferred to the least significant byte position in memory.

Accumulator <- Contents of MSB

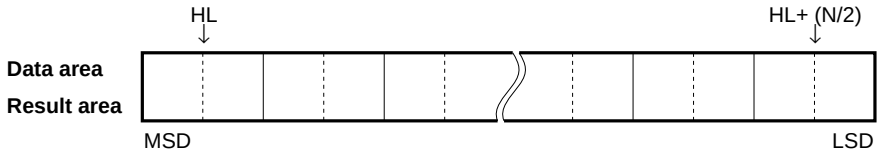
(5) Number of steps

5 bytes

```
 25     25
 26     26                      ;*************************************************
 27     27                      ;*      N-byte data left shift                   *
 28     28                      ;*         input condition                       *
 29     29                      ;*              DE-register <- LSB of N-byte data*
 30     30                      ;*              C-register <- byte counter       *
 31     31                      ;*           output condition                    *
 32     32                      ;*              Acc <- MSB of N-byte data        *
 33     33                      ;*                                               *
 34     34                      ;*              RSS <- 0                         *
 35     35                      ;*************************************************
 36     36  ----                        CSEG
 37     37                              RSS     0
 38     38  0005                BYTLST:
 39     39  0005    B900                MOV     A,#0          ; Acc <- 0
 40     40  0007                BYTLS1:
 41     41  0007    1501                XCHM    [DE+],A
 42     42                      ;
 43     43  0009    56                  RET
 44     44                              ENDS
 45     45                              END
```

3.4.3 Shifting N-Digit Decimal Data One Digit to Right

(1) Memory areas used



(2) Registers used

A, C, H, L

(3) Input conditions

As shown in (1) above, data required for this processing is loaded into the following registers:
Register pair HL <- High-order address of N-digit data
Register C <- Number of bytes (N/2)

(4) Output conditions

The result of a 1-digit right shift is stored in the result area shown in (1) above.
Accumulator <- Contents of LSD
Zero is loaded into the MSD.

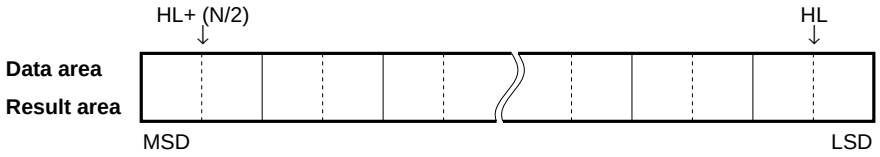
(5) Number of steps

8 bytes

```
      Assemble list

ALN0   STN0  ADRS   OBJECT    M  I  SOURCE      STATEMENT

   1      1                        $        TITLE   ('N-digit shift')
   2      2                                 NAME    BCDRSR
   3      3                        ;*************************************************
   4      4                        ;*       N-digit data right shift                *
   5      5                        ;*          input condition                      *
   6      6                        ;*               HL-register <- MSD of N-digit data *
   7      7                        ;*               C -register <- digit counter    *
   8      8                        ;*           output condition                    *
   9      9                        ;*               Acc <- LSD of N-digit data      *
  10     10                        ;*                                               *
  11     11                        ;*               RSS <- 0                        *
  12     12                        ;*************************************************
  13     13                                PUBLIC  BCDRS,BCDRS1
  14     14                                PUBLIC  BCDLS,BCDLS1
  15     15                        ;
  16     16  ----                          CSEG
  17     17                                RSS     0
  18     18  0000                  BCDRS:
  19     19  0000   B900                   MOV     A,#0          ; Acc <- 0
  20     20  0002                  BCDRS1:
  21     21  0002   058F                   ROR4    [HL]
  22     22  0004   4F                     DECW    HL            ; decrement (HL)
  23     23  0005   32FB                   DBNZ    C,$BCDRS1
  24     24                        ;
  25     25  0007   56                     RET
  26     26                                ENDS
```

3.4.4 Shifting N-Digit Decimal Data One Digit to Left

(1) Memory areas used



(2) Registers used

A, C, H, L

(3) Input conditions

As shown in (1) above, data required for this processing is loaded into the following registers:
Register pair HL <- Low-order address of N-digit data
Register C <- Number of bytes (N/2)

(4) Output conditions

The result of a 1-digit left shift is stored in the result area shown in (1) above.
Accumulator <- Contents of MSD
Zero is loaded into the LSD.

(5) Number of steps

8 bytes

```
27     27
28     28                        ;*************************************************
29     29                        ;*       N-digit data left shift                 *
30     30                        ;*          input condition                      *
31     31                        ;*               HL-register <- LSD of N-digit data   *
32     32                        ;*               C -register <- digit counter         *
33     33                        ;*            output condition                     *
34     34                        ;*               Acc <- MSD of N-digit data        *
35     35                        ;*                                                 *
36     36                        ;*               RSS <- 0                          *
37     37                        ;*************************************************
38     38  ----                          CSEG
39     39                                RSS     0
40     40  0008                  BCDLS:
41     41  0008   B900                    MOV     A,#0
42     42  000A                  BCDLS1:
43     43  000A   059F                    ROL4    [HL]
44     44  000C   47                      INCW    HL              ; increment (HL)
45     45  000D   32FB                    DBNZ    C,$BCDLS1
46     46                        ;
47     47  000F   56                      RET
48     48                                ENDS
49     49                                END
```

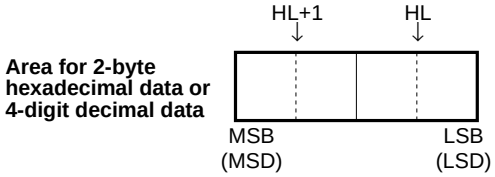
3.5 DATA CONVERSION

This section explains sample programs that convert the format of numeric data representation from hexadecimal to decimal and vice versa, and from hexadecimal to ASCII and vice versa.

3.5.1 Conversion from Hexadecimal (HEX) to Decimal (BCD)

The sample program converts 2-byte hexadecimal data to 4-digit decimal data.

(1) Memory areas used



(2) Registers used

X, A, C, B, R4, R5, R6, D, E, H, L

(3) Input conditions

As shown in (1) above, the following setting is performed:

Register pair HL <- Address of the LSD of the area where 2-byte hexadecimal input data is stored

(4) Output conditions

CY = 1: Conversion is impossible because hexadecimal data is greater than 270FH (= 9999).

CY = 0: Four-digit decimal data produced by conversion is stored in (HL, HL+1).

(5) Processing procedure

This conversion subroutine finds a converted 4-digit (2-byte) decimal value digit by digit, starting with the most significant digit.

A divisor is set for each digit: 1000 for the fourth digit, 100 for the third digit, 10 for the second digit, and 1 for the first digit. Then a converted value is found as a quotient obtained by dividing hexadecimal input data by each divisor; a converted value is found digit by digit, starting with the most significant digit.

<1> The divisor is set to 10000D.

<2> A comparison is made between hexadecimal input data and the divisor. If the hexadecimal input data is 10000 or greater, it cannot be represented by two bytes. This disables conversion and terminates processing.

<3> The digit counter (register B) is set to 4.

<4> The divisor is set to 10.

<5> The hexadecimal input data is divided by the divisor.

<6> The conversion value storage area is shifted one digit to the left; the quotient is stored in the least significant digit position.

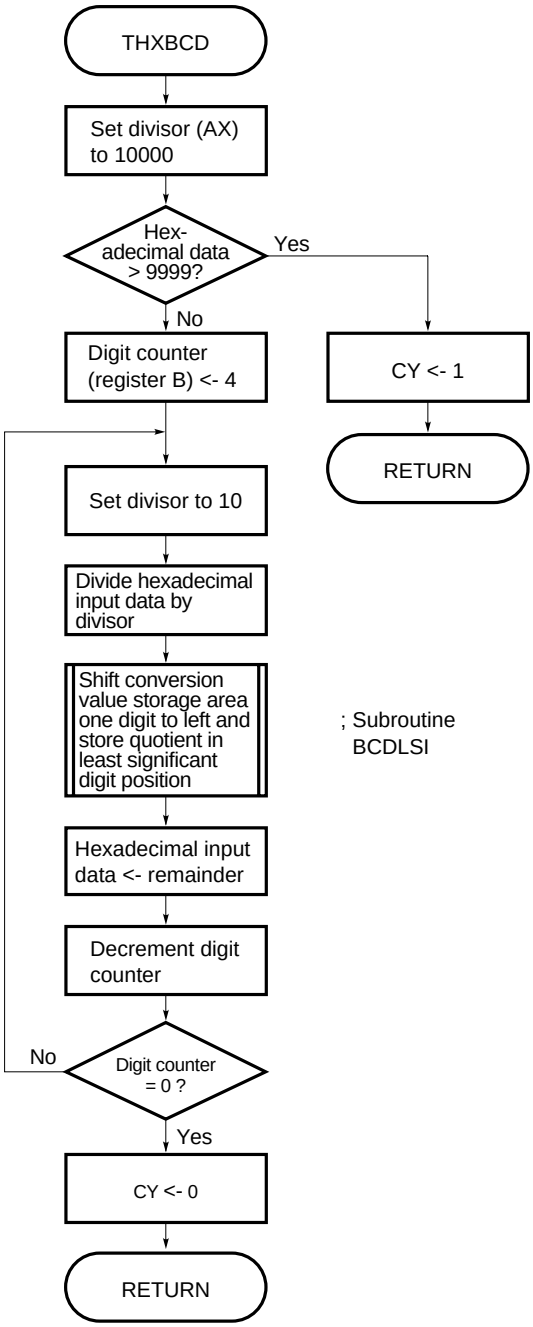
<7> The remainder is used as hexadecimal input data.

<8> The digit counter is decremented, and the processing of **<4>** to **<7>** is repeated until the digit counter reaches 0.

(6) Number of steps

47 bytes

Flowchart



Assemble list

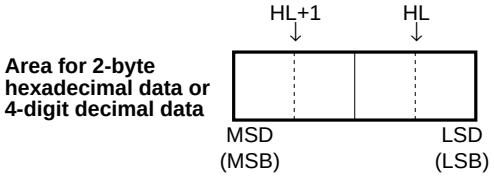
```
ALN0   STN0  ADRS   OBJECT    M  I  SOURCE      STATEMENT

  1      1                          $        TITLE   ('transform BCD <- HEX')
  2      2                                   NAME    TRBCDR
  3      3                          ;*************************************************
  4      4                          ;*      transform BCD <- HEX                     *
  5      5                          ;*         input condition                       *
  6      6                          ;*              HL-register <- HEX-2byte data    *
  7      7                          ;*                                   LSB address *
  8      8                          ;*           output condition                    *
  9      9                          ;*             normal ... cy = 0                 *
 10     10                          ;*                decimal 4-digit -> (HL,HL+1)   *
 11     11                          ;*             overflow ... cy = 1               *
 12     12                          ;*                 HEX data > 9999               *
 13     13                          ;*                                               *
 14     14                          ;*             RSS <- 0                          *
 15     15                          ;*************************************************
 16     16                                   PUBLIC  THXBCD
 17     17                                   EXTRN   BCDLS1
 18     18                          ;
 19     19  ----                             CSEG
 20     20                                   RSS     0
 21     21  0000                    THXBCD:
 22     22  0000   1651                     MOVW    AX,[HL]       ; DE <- hex data
 23     23  0002   24C8                     MOVW    DE,AX
 24     24  0004   601027                   MOVW    AX,#10000
 25     25                          ;
 26     26  0007   8FC8                     CMPW    DE,AX
 27     27  0009   8302                     BC      $THXBD1
 28     28  000B   41                       SET1    CY
 29     29  000C   56                       RET
 30     30                          ;
 31     31  000D                    THXBD1:
 32     32  000D   248F                     MOVW    VP,HL         ; save HL
 33     33                          ;
 34     34                          ;       **** digit counter set ****
 35     35                          ;
 36     36  000F   BB04                     MOV     B,#4
 37     37                          ;
 38     38  0011                    THXBD2:
 39     39  0011   BE0A                     MOV     R6,#10
 40     40  0013   051E                     DIVUW   R6            ; AX <- AX / 10
 41     41  0015   24A8                     MOVW    UP,AX         ; save AX
 42     42                          ;
 43     43                                   RSS     1
 44     44  0017   43                       SWRS
 45     45                          ;
 46     46  0018   640000                   MOVW    AX,#0         ; AX <- 0
 47     47  001B   05E8                     DIVUX   RP0           ; AXDE <- AXDE / RP0
 48     48                                                       ; RP0 <- remainder
 49     49  001D   25C5                     XCH     E,A           ; A <- E
 50     50  001F   24E9                     MOVW    HL,VP
 51     51  0021   BE02                     MOV     C,#2          ; C <- 2
 52     52  0023   R280000                  CALL    !BCDLS1       ; digit left shift
 53     53                          ;
 54     54                                   RSS     0
 55     55  0026   43                       SWRS
 56     56                          ;
 57     57  0027   24C8                     MOVW    DE,AX         ; DE <- remainder
 58     58  0029   240B                     MOVW    AX,UP         ; load AX
 59     59  002B   33E4                     DBNZ    B,$THXBD2
 60     60                          ;
 61     61  002D   40                       CLR1    CY
 62     62  002E   56                       RET
 63     63
```

```
   64      64                                      ENDS
   65      65                                      END

Segment informations:

ADRS   LEN    NAME

0000   002FH  ?CSEG


          Cross-Reference List

NAME      VALUE   R  ATTR   RTYP   SEGNAME    XREFS


?CSEG               CSEG          ?CSEG          19#
BCDLS1    ----H   E           EXT                17@     52
THXBCD      0H    R  ADDR    PUB    ?CSEG          16@     21#
THXBD1      DH    R  ADDR           ?CSEG          27      31#
THXBD2     11H    R  ADDR           ?CSEG          38#     59
TRBCDR              MOD                             2#


Target chip:uPD78320
Assembly complete,      0 error(s) and        0 warning(s) found. (   0)
```

3.5.2 Conversion from Decimal (BCD) to Hexadecimal (HEX)

The sample program converts 4-digit decimal data to 2-byte hexadecimal data.

(1) Memory areas used



(2) Registers used

X, A, C, B, R5, R6, D, E, H, L

(3) Input conditions

As shown in (1) above, the following setting is performed:
Register pair HL <- Address of the LSD of the area where 4-digit decimal input data is stored

(4) Output conditions

CY = 1: Conversion is impossible because input data is not decimal data.
CY = 0: Two-byte hexadecimal data produced by conversion is stored in (HL, HL+1).

(5) Processing procedure

The conversion value storage area is cleared to 0. Then decimal input data is shifted left one digit at a time starting with the most significant digit so that the decimal input data is loaded into the accumulator digit by digit. Then the following operation is performed four times to complete conversion:
Storage area <- Storage area x 10 + accumulator

<1> The conversion value storage register (register DE) is cleared to 0.

<2> The digit counter (register B) is set to 4.

<3> The decimal input data is shifted one digit to the left to load the most significant digit into the accumulator.

<4> A check is made to see if the most significant digit is decimal data (0 to 9). If the digit is not decimal data, conversion is impossible, and CY is set.

<5> Conversion value storage register <- Conversion value storage register x 10 + accumulator

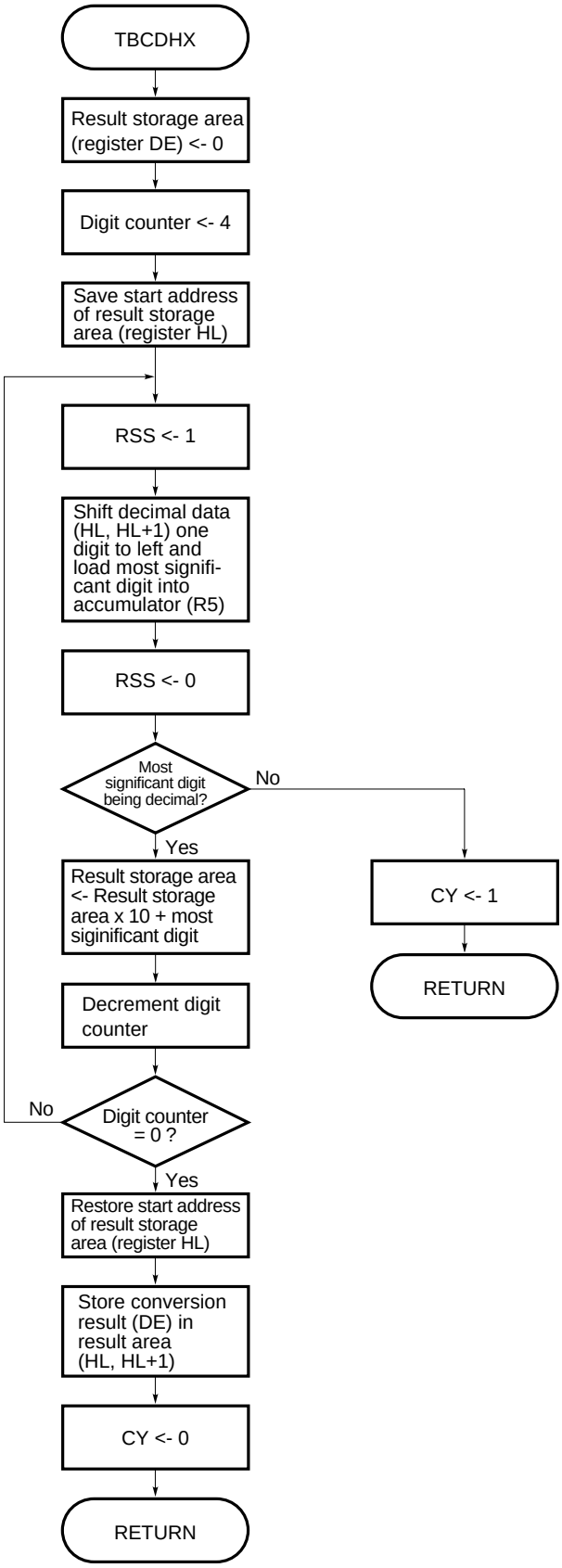
<6> The digit counter is decremented, and the processing of **<3>** to **<5>** is repeated until the digit counter reaches 0.

<7> The contents of the conversion value storage register are stored in the conversion result area, then CY is reset.

(6) Number of steps

45 bytes

Flowchart



Phase-out/Discontinued

Assemble list

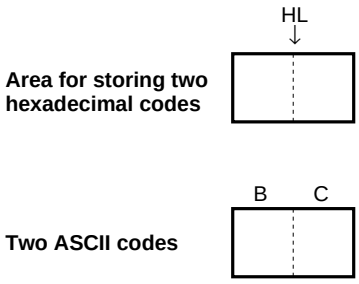
```
ALN0   STN0  ADRS   OBJECT    M  I  SOURCE      STATEMENT

   1      1                        $        TITLE   ('transform HEX <- BCD')
   2      2                                 NAME    TRHEXR
   3      3                        ;*******************************************************
   4      4                        ;*       transform HEX <- BCD                          *
   5      5                        ;*           input condition                           *
   6      6                        ;*              HL-register <- decimal 4 digit data    *
   7      7                        ;*                                    LSD address      *
   8      8                        ;*           output condition                          *
   9      9                        ;*              normal ... cy = 0                      *
  10     10                        ;*                   HEX 2 byte -> (HL,HL+1)           *
  11     11                        ;*              error  ... cy = 1                      *
  12     12                        ;*                                                     *
  13     13                        ;*              RSS <- 0                               *
  14     14                        ;*******************************************************
  15     15                                 PUBLIC  TBCDHX
  16     16                                 EXTRN   BCDLS1
  17     17                        ;
  18     18  ----                           CSEG
  19     19                                 RSS     0
  20     20  0000                 TBCDHX:
  21     21  0000   650000                 MOVW    DE,#0         ; DE <- 0
  22     22                        ;
  23     23                        ;        ****  digit counter set  ****
  24     24                        ;
  25     25  0003   BB04                   MOV     B,#4
  26     26                        ;
  27     27  0005   248F                   MOVW    VP,HL         ; save HL
  28     28                        ;
  29     29  0007                 TBDHX1:
  30     30                                 RSS     1
  31     31  0007   43                     SWRS
  32     32                        ;
  33     33                        ;        ****  output 1 digit from MSD ****
  34     34                        ;
  35     35  0008   BE02                   MOV     C,#2          ; C-register <- 2
  36     36  000A   BD00                   MOV     A,#0          ; Acc <- 0
  37     37  000C   24E9                   MOVW    HL,VP
  38     38  000E   R280000                CALL    !BCDLS1       ; N-digit data left shift
  39     39                        ;
  40     40                        ;        ****  check / BCD display ?  ****
  41     41                        ;
  42     42  0011   AF0A                   CMP     A,#10
  43     43                                 RSS     0
  44     44  0013   43                     SWRS
  45     45                        ;
  46     46  0014   8302                   BC      $TBDHX2
  47     47                        ;
  48     48  0016   41                     SET1    CY            ; not BCD
  49     49  0017   56                     RET
  50     50                        ;
  51     51                        ;        ****  subroutine / HEX <- BCD  ****
  52     52                        ;
  53     53  0018                 TBDHX2:
  54     54  0018   BC00                   MOV     R4,#0
  55     55  001A   2545                   XCH     R4,R5
  56     56                        ;
  57     57                        ;        ****  result * 10 + 1 digit  ****
  58     58                        ;
  59     59  001C   600A00                 MOVW    AX,#10
  60     60  001F   052D                   MULUW   DE
  61     61  0021   88CC                   ADDW    DE,RP2
  62     62                        ;
```

```
   63     63  0023   33E2                    DBNZ    B,$TBDHX1
   64     64  0025   24E9                    MOVW    HL,VP
   65     65                         ;
   66     66                         ;       **** store result ****
   67     67                         ;
   68     68  0027   25C8                    XCHW    DE,AX
   69     69  0029   16D1                    MOVW    [HL],AX
   70     70  002B   40                      CLR1    CY
   71     71  002C   56                      RET
   72     72
   73     73                                 ENDS
   74     74                                 END

Segment informations:

ADRS   LEN    NAME

0000   002DH ?CSEG


         Cross-Reference List


NAME      VALUE   R  ATTR   RTYP   SEGNAME    XREFS

?CSEG               CSEG          ?CSEG         18#
BCDLS1    ----H   E          EXT                16@     38
TBCDHX      0H    R  ADDR   PUB    ?CSEG         15@     20#
TBDHX1      7H    R  ADDR          ?CSEG         29#     63
TBDHX2     18H    R  ADDR          ?CSEG         46      53#
TRHEXR              MOD                           2#

Target chip:uPD78320
Assembly complete,      0 error(s) and        0 warning(s) found. (   0)
```

3.5.3 Conversion from ASCII to Hexadecimal (HEX)

The sample program provided here converts two ASCII codes (30H-39H, 41H-46H) to two hexadecimal codes (00H-FFH).

(1) Memory areas used



(2) Registers used

A, B, C, H, L

(3) Input conditions

As shown in (1) above, the following setting is performed:

Register BC <- Two ASCII codes

Register HL <- Address of the area where two hexadecimal codes are stored

(4) Output conditions

CY = 1: Conversion is impossible because input data is not ASCII data.

CY = 0: Two hexadecimal codes produced by conversion are stored in the storage area shown in (1) above.

(5) Processing procedure

<1> The higher one ASCII code (register B) is loaded into the accumulator.

<2> A check is made to see if the contents of the accumulator are within the range 30H-39H or 41H-46H. If the contents of the accumulator are not within these ranges, conversion is impossible, and CY is set.

<3> If the contents of the accumulator are within 30H-39H, 30H is subtracted.

If the contents of the accumulator are within 41H-46H, 37H is subtracted.

<4> The contents of the address specified by register HL are shifted four bits; the contents of the accumulator are stored in the lower bits.

<5> The lower one ASCII code (register C) is loaded into the accumulator, then the processing of **<2>** to **<4>** is performed.

(6) Number of steps

41 bytes

Assemble list

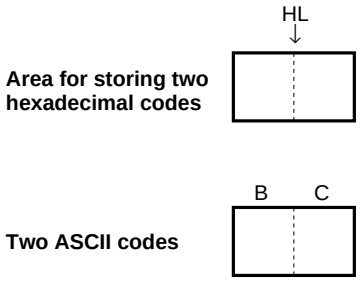
```
 ALN0   STN0  ADRS    OBJECT    M  I  SOURCE      STATEMENT

    1      1                          $       TITLE   ('HEX <- ASCII')
    2      2                                  NAME    GHEXR
    3      3                          ;*********************************************
    4      4                          ;*      transform    HEX <- ASCII            *
    5      5                          ;*                 (2code)  (2code)           *
    6      6                          ;*                                           *
    7      7                          ;*              input condition              *
    8      8                          ;*                 BC-register <- ASCII      *
    9      9                          ;*                                           *
   10     10                          ;*              output condition             *
   11     11                          ;*                 (HL)  <-  hex             *
   12     12                          ;*********************************************
   13     13                                  PUBLIC  GETHEX
   14     14                                  PUBLIC  SHEX
   15     15                          ;
   16     16  ----                            CSEG
   17     17                                  RSS     0
   18     18  0000                    GETHEX:
   19     19  0000    D3                      MOV     A,B           ; ASCII upper-code load
   20     20  0001   R281100                  CALL    !SHEX         ; get  hex 1th code
   21     21  0004    830A                    BC      $GTHEX1
   22     22                          ;
   23     23  0006    059F                    ROL4    [HL]
   24     24  0008    D2                      MOV     A,C           ; ASCII lower-code load
   25     25  0009   R281100                  CALL    !SHEX         ; get  hex 2th code
   26     26  000C    8302                    BC      $GTHEX1
   27     27  000E    059F                    ROL4    [HL]
   28     28                          ;
   29     29  0010                    GTHEX1:
   30     30  0010    56                      RET
   31     31
   32     32                                  ENDS
   33     33
   34     34                          ;*********************************************
   35     35                          ;*      subroutine  /  get hex 1-code(Acc)    *
   36     36                          ;*********************************************
   37     37  ----                            CSEG
   38     38                                  RSS     0
   39     39  0011                    SHEX:
   40     40  0011    AF30                    CMP     A,#'0'        ; check / ASCII > 30H
   41     41  0013    8312                    BC      $SHEX2
   42     42                          ;
   43     43  0015    AF39                    CMP     A,#'9'        ; check / ASCII > 39H
   44     44  0017    8203                    BNC     $SHEX1
   45     45  0019    AA30                    SUB     A,#30H
   46     45  001B    56                      RET
   47     47                          ;
   48     48  001C                    SHEX1:
   49     49  001C    AF41                    CMP     A,#'A'        ; check / ASCII > 41H
   50     50  001E    8307                    BC      $SHEX2
   51     51                          ;
   52     52  0020    AF46                    CMP     A,#'F'        ; check / ASCII < 46H
   53     53  0022    8203                    BNC     $SHEX2
   54     54  0024    AA37                    SUB     A,#37H
   55     55  0026    56                      RET
   56     56                          ;
   57     57  0027                    SHEX2:
   58     58  0027    41                      SET1    CY            ; error
   59     59  0028    56                      RET
   60     60
   61     61                                  ENDS
   62     62                                  END
```

```
Segment informations:

ADRS   LEN    NAME

0000   0029H  ?CSEG

          Cross-Reference List

NAME      VALUE   R  ATTR   RTYP   SEGNAME    XREFS

?CSEG               CSEG          ?CSEG          16#     37#
GETHEX      0H    R  ADDR   PUB    ?CSEG          13@     18#
GHEXR               MOD                            2#
GTHEX1     10H    R  ADDR         ?CSEG          21      26      29#
SHEX       11H    R  ADDR   PUB    ?CSEG          14@     20      25      39#
SHEX1      1CH    R  ADDR         ?CSEG          44      48#
SHEX2      27H    R  ADDR         ?CSEG          41      50      53      57#

Target chip:uPD78320
Assembly complete,      0 error(s) and         0 warning(s) found. (    0)
```

3.5.4 Conversion from Hexadecimal (HEX) to ASCII

The sample program provided here converts two hexadecimal codes (00H-FFH) to two ASCII codes (30H-39H, 41H-46H).

(1) Memory areas used



(2) Registers used

A, B, C, H, L

(3) Input conditions

As shown in (1) above, the following setting is performed:
Register HL <- Address of the area where two hexadecimal codes are stored

(4) Output conditions

Register BC <- Two ASCII codes produced by conversion

(5) Processing procedure

<1> The higher four bits of the address specified by register HL are loaded into the accumulator.

<2> A check is made to see if the accumulator contains a value not greater than 9. If the accumulator contains a value not greater than 9, 37H is added to the accumulator. If the accumulator contains a value greater than 9, 30H is added to the accumulator.

<3> The contents of the accumulator are loaded into register B.

<4> The lower four bits of the address specified by register HL are loaded into the accumulator.

<5> The processing of **<2>** is performed, then the contents of the accumulator are loaded into register C.

(6) Number of steps

28 bytes

Phase-out/Discontinued

```
     Assemble list

 ALN0   STN0  ADRS   OBJECT    M  I  SOURCE      STATEMENT

    1      1                        $       TITLE   ('ASCII <- HEX')
    2      2                                NAME    ASCII
    3      3                        ;*********************************************
    4      4                        ;*      transform   ASCII  <-  HEX             *
    5      5                        ;*                  (2code)   (2code)          *
    6      6                        ;*                                             *
    7      7                        ;*              input condition                *
    8      8                        ;*                (HL) <- hex 2-code           *
    9      9                        ;*              output condition               *
   10     10                        ;*                BC-register <- ASCII 2-code  *
   11     11                        ;*********************************************
   12     12                                PUBLIC  GETASC
   13     13                                PUBLIC  SASC
   14     14                        ;
   15     15  ----                          CSEG
   16     16                                RSS     0
   17     17  0000                  GETASC:
   18     18  0000   B900                    MOV     A,#0
   19     19  0002   059F                    ROL4    [HL]          ; hex upper code load
   20     20  0004  R281300                 CALL    !SASC
   21     21  0007   2431                    MOV     B,A           ; store result
   22     22                        ;
   23     23  0009   B900                    MOV     A,#0
   24     24  000B   059F                    ROL4    [HL]          ; hex lower code load
   25     25  000D  R281300                 CALL    !SASC
   26     26  0010   2421                    MOV     C,A           ; store result
   27     27  0012   56                      RET
   28     28
   29     29                                ENDS
   30     30
   31     31                        ;*****************************************************
   32     32                        ;*      subroutine  /  get ASCII 1-code(BC-register)   *
   33     33                        ;*****************************************************
   34     34  ----                          CSEG
   35     35                                RSS     0
   36     36  0013                  SASC:
   37     37  0013   AF0A                    CMP     A,#0AH        ; check / hex > 9
   38     38  0015   8302                    BC      $SASC1
   39     39  0017   A807                    ADD     A,#07H        ; bias (+7)
   40     40  0019                  SASC1:
   41     41  0019   A830                    ADD     A,#30H        ; bias (+30H)
   42     42  001B   56                      RET
   43     43
   44     44                                ENDS
   45     45                                END

Segment informations:

ADRS   LEN    NAME

0000   001CH  ?CSEG

          Cross-Reference List

NAME      VALUE   R  ATTR   RTYP   SEGNAME    XREFS

?CSEG               CSEG          ?CSEG         15#     34#
ASCII               MOD                          2#
GETASC       0H   R  ADDR   PUB    ?CSEG         12@     17#
SASC        13H   R  ADDR   PUB    ?CSEG         13@     20      25      36#
SASC1       19H   R  ADDR          ?CSEG         38      40#

Target chip:uPD78320
Assembly complete,      0 error(s) and        0 warning(s) found. (   0)
```

3.6 COMPARISON

This section explains three types of processing:

- Comparison between two numeric data items to see if which is greater, and branch operation according to the result
- Data search
- Set/reset processing and test processing on a bit-by-bit basis for software flags provided in memory

3.6.1 16-Bit (2-Byte) Data Comparison

The sample program provided here performs three types of comparisons:

=; Equal

<; Greater than

>; Less than

For these three types of processing, the µPD78320 provides a 16-bit instruction (CMPW rp,rp1), which allows a register pair to be compared with another register pair.

The result of a comparison is indicated by the Z flag and CY flag. These flags can be tested with a conditional branch instruction to cause a branch to an arbitrary address.

Number of steps: 30 bytes

```
    Assemble list

ALN0   STN0  ADRS   OBJECT    M  I  SOURCE      STATEMENT

  1      1                          $       TITLE   ('data comparison')
  2      2                                  NAME    DCOMPR
  3      3                          ;*******************************************************
  4      4                          ;*      16bit(2byte) data comparison                   *
  5      5                          ;*******************************************************
  6      6  ----                            CSEG
  7      7                                  RSS     0
  8      8
  9      9  0000                    DCOMP:
 10     10  0000   650200                    MOVW    DE,#2
 11     11  0003   670300                    MOVW    HL,#3
 12     12
 13     13  0006   8FCF                      CMPW    DE,HL
 14     14  0008   8108                      BZ      $ADRS1
 15     15  000A   830C                      BC      $ADRS2
 16     16
 17     17                          ;===============================================
 18     18                          ;               RP > RP1
 19     19                          ;===============================================
 20     20
 21     21  000C                    DCL1:
 22     22  000C   00                        NOP
 23     23  000D   00                        NOP
 24     24  000E   00                        NOP
 25     25  000F   00                        NOP
 26     26  0010   14FA                      BR      $DCL1
 27     27
 28     28                          ;===============================================
 29     29                          ;               RP = RP1
 30     30                          ;===============================================
 31     31
 32     32  0012                    ADRS1:
 33     33  0012   00                        NOP
 34     34  0013   00                        NOP
```


```
  35     35  0014   00                      NOP
  36     36  0015   00                      NOP
  37     37  0016   14FA                    BR      $ADRS1
  38     38
  39     39                          ;================================================
  40     40                          ;               RP < RP1
  41     41                          ;================================================
  42     42
  43     43  0018                    ADRS2:
  44     44  0018   00                      NOP
  45     45  0019   00                      NOP
  46     46  001A   00                      NOP
  47     47  001B   00                      NOP
  48     48  001C   14FA                    BR      $ADRS2
  49     49
  50     50                                ENDS
  51     51                                END

Segment informations:

ADRS   LEN     NAME

0000   001EH   ?CSEG

          Cross-Reference List

NAME       VALUE   R  ATTR   RTYP   SEGNAME    XREFS

?CSEG                 CSEG          ?CSEG          6#
ADRS1        12H   R  ADDR          ?CSEG         14      32#                37
ADRS2        18H   R  ADDR          ?CSEG         15      43#                48
DCL1          CH   R  ADDR          ?CSEG         21#     26
DCOMP         0H   R  ADDR          ?CSEG          9#
DCOMPR                MOD                         2#

Target chip:uPD78320
Assembly complete,      0 error(s) and     0 warning(s) found. (    0)
```

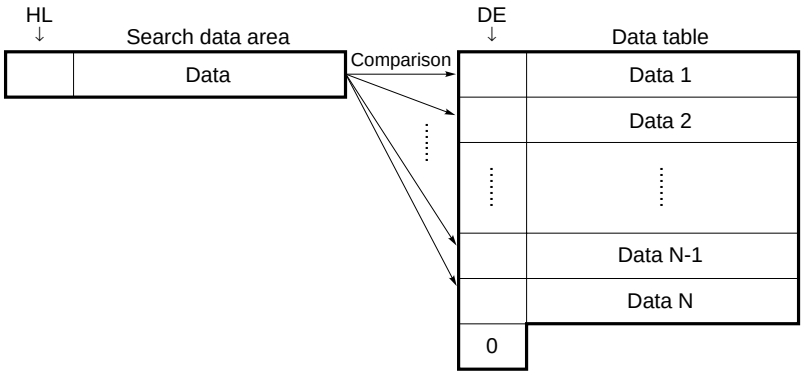
Remarks 1. rp/rp1: RPn (n = 0 to 7)

2. Those register pairs that are subject to coding in the operands rp and rp1 are identical. In an object program actually generated, however, rp contains a different register pair from a register pair contained in rp1.

3.6.2 Data Search

This section explains how to search for memory data from a data table set beforehand, as shown in Figure 3-2.

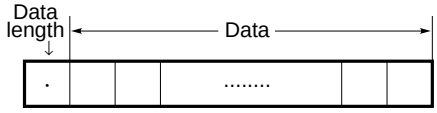
Figure 3-2. Data Search



A data length may or may not be defined among search data or data tables. In this explanation, however, the length of data is undefined. So a data format is set as shown in Figure 3-3 so that data can be checked in order of data length or data contents in search operation.

As a data table termination indication, the start address (data length) of the data format (Figure 3-3) contains 0.

Figure 3-3. Data Format

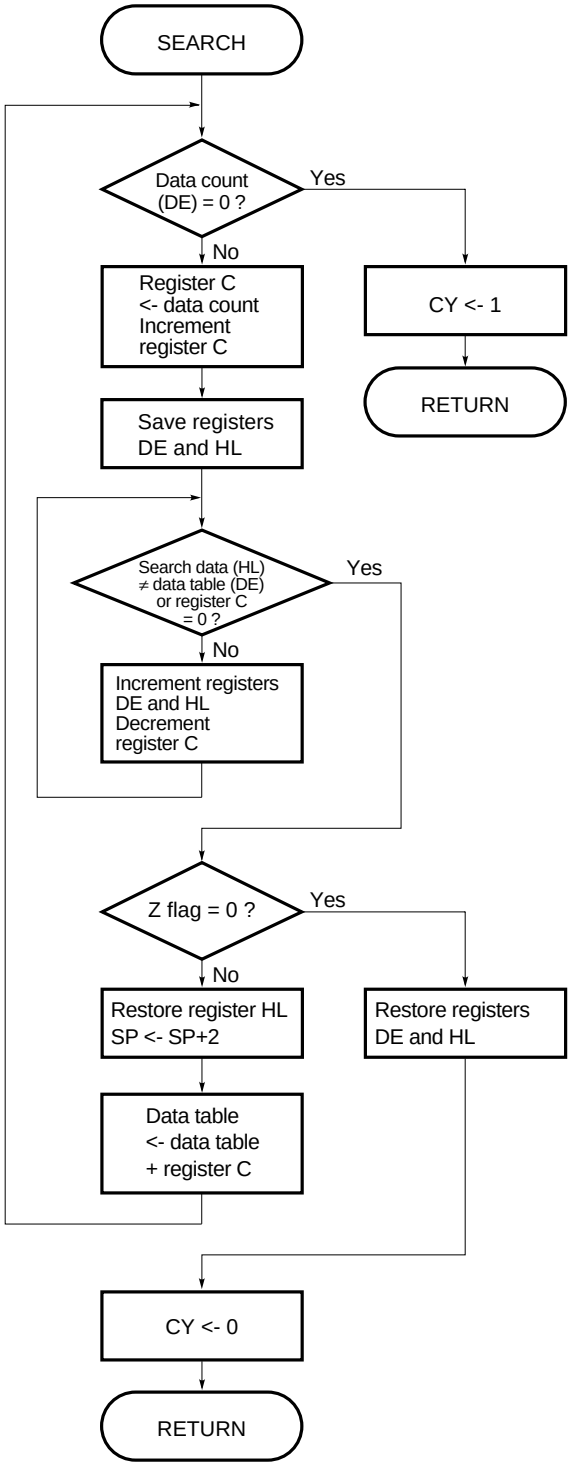


In data search processing, the low-order address of a search data area and the low-order address of a data table must be set in registers HL and DE, respectively.

When data is found from a data table, the low-order address of the data table is output with register DE.

Number of steps: 93 bytes

Flowchart



Assemble list

```
ALN0   STN0  ADRS   OBJECT    M  I  SOURCE      STATEMENT

  1      1                        $        TITLE    (‘data searching’)
  2      2                                 NAME     SEARC
  3      3                        ;*******************************************************
  4      4                        ;*       data searching                                *
  5      5                        ;*        input condition                              *
  6      6                        ;*                HL-register <- search data address   *
  7      7                        ;*                DE-register <- table top address     *
  8      8                        ;*        output condition                             *
  9      9                        ;*                cy = 1                               *
 10     10                        ;*                   not search data                   *
 11     11                        ;*                cy = 0                               *
 12     12                        ;*                  DE-register <- search table address*
 13     13                        ;*                                                     *
 14     14                        ;*                        RSS <- 0                     *
 15     15                        ;*******************************************************
 16     16                                 PUBLIC SEARCH
 17     17                        ;
 18     18  ----                           CSEG
 19     19                                 RSS      0
 20     20  0000                  SEARCH:
 21     21  0000   620000                  MOVW     BC,#0
 22     22  0003   5C                      MOV      A,[DE]       ; load data-length
 23     23  0004   AF01                    CMP      A,#1         ; check / length = 0
 24     24  0006   8317                    BC       $SERCH2
 25     25                        ;
 26     26  0008   2421                    MOV      C,A          ; store length
 27     27  000A   C2                      INC      C
 28     28  000B   35C0                    PUSH     DE,HL
 29     29                        ;
 30     30  000D   1524                    CMPBKE   [DE+],[HL+]
 31     31  000F   8004                    BNZ      $SERCH1
 32     32  0011   34C0                    POP      DE,HL
 33     33  0013   40                      CLR1     CY           ; search success
 34     34  0014   56                      RET
 35     35                        ;
 36     36  0015                  SERCH1:
 37     37  0015   05C8                    INCW     SP
 38     38  0017   05C8                    INCW     SP
 39     39  0019   3480                    POP      HL
 40     40  001B   88CA                    ADDW     DE,BC        ; DE <- next-table data address
 41     41  001D   14E1                    BR       $SEARCH
 42     42                        ;
 43     43  001F                  SERCH2:
 44     44  001F   41                      SET1     CY
 45     45  0020   56                      RET
 46     46
 47     47                                 ENDS
 48     48
 49     49                        $        EJECT
 50     50
 51     51                        ;*******************************************************
 52     52                        ;*       data searching initilize                      *
 53     53                        ;*******************************************************
 54     54
 55     55                        ;===============================================
 56     56                        ;                 data table
 57     57                        ;===============================================
 58     58   ----                          CSEG
 59     59                                 RSS      0
 60     60  0021                  STABLE:
 61     61  0021   03123478                DB       03,12H,34H,78H
 62     62  0025   04556677                DB       04,55H,66H,77H,88H
             0029   88
```

```
  63     63  002A   05123456                DB      05,12H,34H,56H,78H,10H
             002E   7810
  64     64  0030   03123456                DB      03,12H,34H,56H
  65     65  0034   0412340A                DB      04,12H,34H,0AH,78H
             0038   78
  66     66  0039   04123456                DB      04,12H,34H,56H,70H
             003D   70
  67     67  003E   04123456                DB      04,12H,34H,56H,78H
             0042   78
  68     68  0043   01AB                    DB      01,0ABH
  69     69  0045   023478                  DB      02,34H,78H
  70     70  0048   00                      DB      00
  71     71
  72     72                         ;=================================================
  73     73                         ;               searching data
  74     74                         ;=================================================
  75     75
  76     76  0049                   SDATA:
  77     77  0049   04123456                DB      04,12H,34H,56H,78H
             004D   78
  78     78
  79     79                         ;=================================================
  80     80                         ;               main routine
  81     81                         ;=================================================
  82     82
  83     83  004E                   SMAIN:
  84     84  004E   R652100                 MOVW    DE,#STABLE
  85     85  0051   R674900                 MOVW    HL,#SDATA
  86     86
  87     87  0054   R280000                 CALL    !SEARCH
  88     88  0057    8302                   BC      $SERR
  89     89  0059                   SL1:
  90     90  0059   14FE                    BR      $SL1
  91     91
  92     92  005B                   SERR:
  93     93  005B   14FE                    BR      $SERR
  94     94
  95     95                                 ENDS
  96     96                                 END

Segment informations:

ADRS   LEN     NAME

0000   005DH  ?CSEG

         Cross-Reference List

NAME       VALUE   R  ATTR   RTYP   SEGNAME    XREFS

?CSEG                 CSEG          ?CSEG          18#     58#
SDATA        49H   R  ADDR          ?CSEG          76#     85
SEARC                 MOD                           2#
SEARCH        0H   R  ADDR   PUB    ?CSEG          16@     20#                41       87
SERCH1       15H   R  ADDR          ?CSEG          31      36#
SERCH2       1FH   R  ADDR          ?CSEG          24      43#
SERR         5BH   R  ADDR          ?CSEG          88      92#                93
SL1          59H   R  ADDR          ?CSEG          89#     90
SMAIN        4EH   R  ADDR          ?CSEG          83#
STABLE       21H   R  ADDR          ?CSEG          60#     84

Target chip:uPD78320
Assembly complete,      0 error(s) and     0 warning(s) found. (    0)
```

3.6.3 Memory Bit Test and Set/Reset Operation

(1) When several bits are tested

The contents of memory are compared (AND) with the contents of the accumulator, and a bit test is performed using the Z flag.

Example 1.
```
MOV R1,#00000011B  ; R1 = A
AND A,saddr        ; Test A and Memory
BNZ $LABEL1
```

In example 1, a branch to address LABEL1 occurs when memory bit 0 and/or bit 1 is set to 1 (result of AND operation ≠ 0).

(2) When one bit is tested

When just one bit is tested, a bit test instruction (BT) can be used.

Example 2.
```
BT saddr.1,$LABEL2
```

In example 2, a branch to address LABEL2 occurs when memory bit 1 is set to 1.

Example 3.
```
MOV A,mem
BT  A.2,$LABEL3
```

In example 3, a branch to address LABEL3 occurs when memory bit 2 is set to 1.

(3) When several bits are set or reset

The contents of short direct memory are ORed or ANDed with immediate data, then a bit-by-bit set or reset operation is performed.

Example 4.
```
OR saddr,#00000011B
```
5.
```
AND saddr,#00001111B
```

In example 4, saddr is ORed with 00000011, then bits 0 and 1 are set.
In example 5, saddr is ANDed with 00001111, then bits 4 to 7 are reset.
Then mem is ORed or ANDed with the accumulator, then a bit-by-bit set or reset operation is performed.

Example 6.
```
MOV R1,#00000011B
OR mem,A (or AND mem,A)
```

The contents of memory are ORed (or ANDed) with 00000011, then bits 0 and 1 are reset.

(4) When just one bit is set or reset

When just one bit is set or reset, a bit manipulation instruction can be used.

Example 7. `SET1 saddr.1`
8. `CLR1 saddr.1`

In example 7, bit 1 of saddr is set.
In example 8, bit 1 of saddr is reset.

Number of steps: 70 bytes

```
   Assemble list

ALN0   STN0  ADRS   OBJECT    M  I  SOURCE      STATEMENT

  1      1                          $        TITLE   ('memory test')
  2      2                                   NAME    BTESTR
  3      3                          ;*********************************************
  4      4                          ;*      memory bit test                      *
  5      5                          ;*********************************************
  6      6
  7      7                          ;=============================================
  8      8                          ;               test data
  9      9                          ;=============================================
 10     10                                   EXTRN   SADD1,SADD2,SADD3
 11     11                                   EXTRN   SADD4,SADD5,SADD6
 12     12                          ;
 13     13  ----                             CSEG
 14     14                                   RSS     0
 15     15  0000                   BTEST1:
 16     16  0000   03                        DB      00000011B
 17     17  0001                   BTEST2:
 18     18  0001   02                        DB      00000010B
 19     19
 20     20                          ;=============================================
 21     21                          ;               plural bit test
 22     22                          ;=============================================
 23     23                          ;
 24     24                          ;       **** mem ****
 25     25                          ;
 26     26  0002  R670000                    MOVW    HL,#BTEST1
 27     27                          ;
 28     28  0005   B903                      MOV     A,#00000011B
 29     29  0007   165C                      AND     A,[HL]
 30     30  0009   800A                      BNZ     $LABEL1
 31     31  000B                   BTL1:
 32     32  000B   14FE                      BR      $BTL1
 33     33                          ;
 34     34                          ;       **** saddr ****
 35     35                          ;
 36     36  000D   B903                      MOV     A,#00000011B
 37     37  000F  R9C00                     AND     A,SADD1
 38     38  0011   8002                      BNZ     $LABEL1
 39     39  0013                   BTL2:
 40     40  0013   14FE                      BR      $BTL2
 41     41
 42     42  0015                   LABEL1:
 43     43  0015   14FE                      BR      $LABEL1
 44     44
 45     45                          $        EJECT
```

```
 46    46
 47    47                        ;=============================================
 48    48                        ;               singular number
 49    49                        ;=============================================
 50    50
 51    51  0017  R71000B                 BT      SADD2.1,$LABEL2
 52    52
 53    53  001A                  SNL1:
 54    54  001A   14FE                   BR      $SNL1
 55    55
 56    56  001C  R670100                 MOVW    HL,#BTEST2
 57    57
 58    58  001F   5D                     MOV     A,[HL]
 59    59  0020   03BA02                 BT      A.2,$LABEL2
 60    60
 61    61  0023                  SNL2:
 62    62  0023   14FE                   BR      $SNL2
 63    63
 64    64  0025                  LABEL2:
 65    65  0025   14FE                   BR      $LABEL2
 66    66
 67    67                        $       EJECT
 68    68
 69    69                        ;************************************************
 70    70                        ;*      bit set/reset                           *
 71    71                        ;************************************************
 72    72
 73    73                        ;=============================================
 74    74                        ;               bit set
 75    75                        ;=============================================
 76    76
 77    77  0027  R6E0003                 OR      SADD3,#00000011B
 78    78
 79    79                        ;=============================================
 80    80                        ;               bit reset
 81    81                        ;=============================================
 82    82
 83    83  002A  R6C000F                 AND     SADD4,#00001111B
 84    84
 85    85                        ;=============================================
 86    86                        ;                set/reset data
 87    87                        ;=============================================
 88    88
 89    89  002D                  SRDATA:
 90    90  002D  02                   DB     00000010B
 91    91
 92    92                        ;=============================================
 93    93                        ;               bit set
 94    94                        ;=============================================
 95    95
 96    96  002E  R672D00                 MOVW    HL,#SRDATA
 97    97
 98    98  0031   B903                   MOV     A,#00000011B
 99    99
100   100  0033   16DE                   OR      [HL],A
101   101
102   102  0035                  SRL1:
103   103  0035   14FE                   BR      $SRL1
104   104
105   105                        ;=============================================
106   106                        ;               bit reset
107   107                        ;=============================================
108   108
109   109  0037  R672D00                 MOVW    HL,#SRDATA
110   110
111   111  003A   B903                   MOV     A,#00000011B
112   112  003C   16DC                   AND     [HL],A
```

```
 113    113
 114    114  003E                    SRL2:
 115    115  003E   14FE                      BR      $SRL2
 116    116
 117    117                          ;=========================================
 118    118                          ;               singular number set/reset
 119    119                          ;=========================================
 120    120
 121    121  0040  RB100                     SET1    SADD5.1
 122    122
 123    123  0042  RA100                     CLR1    SADD6.1
 124    124
 125    125  0044                    SRL3:
 126    126  0044   14FE                      BR      $SRL3
 127    127
 128    128                                  ENDS
 129    129                                  END

Segment informations:

ADRS   LEN    NAME

0000   0046H ?CSEG

         Cross-Reference List

NAME      VALUE   R  ATTR   RTYP   SEGNAME    XREFS

?CSEG               CSEG          ?CSEG         13#
BTEST1      0H   R  ADDR          ?CSEG         15#     26
BTEST2      1H   R  ADDR          ?CSEG         17#     56
BTESTR              MOD                          2#
BTL1        BH   R  ADDR          ?CSEG         31#     32
BTL2       13H   R  ADDR          ?CSEG         39#     40
LABEL1     15H   R  ADDR          ?CSEG         30      38      42#     43
LABEL2     25H   R  ADDR          ?CSEG         51      59      64#     65
SADD1     ----H   E         EXT                10@     37
SADD2     ----H   E         EXT                10@     51
SADD3     ----H   E         EXT                10@     77
SADD4     ----H   E         EXT                11@     83
SADD5     ----H   E         EXT                11@    121
SADD6     ----H   E         EXT                11@    123
SNL1       1AH   R  ADDR          ?CSEG         53#     54
SNL2       23H   R  ADDR          ?CSEG         61#     62
SRDATA     2DH   R  ADDR          ?CSEG         89#     96     109
SRL1       35H   R  ADDR          ?CSEG        102#    103
SRL2       3EH   R  ADDR          ?CSEG        114#    115
SRL3       44H   R  ADDR          ?CSEG        125#    126

Target chip:uPD78320
Assembly complete,      0 error(s) and     0 warning(s) found. (    0)
```

3.7 TABLE REFERENCE PROCESSING

This section explains an interpolation computation method as an example of table reference application.

As shown in Figure 3-4, when function f(x) is given, and for 26 points (X)

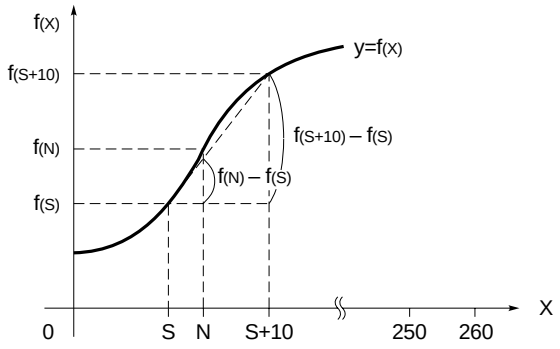
(X = 0, 10, 20, ..., 260)

equally spaced in a segment [0,260], the values of f(x) are given as

($f_{(0)}$, $f_{(10)}$, $f_{(20)}$, ..., $f_{(260)}$),

the value of $f_{(N)}$ for an arbitrary point (N) in the section [0,255] is found using the interpolation computation method.

Figure 3-4. Function f(x)



First, the following expression is used to identify which one of the subsections produced by equally spaced 26 points contains point N:

$$S = INT\ (N/10)*10 \qquad \cdots\cdots (1)$$

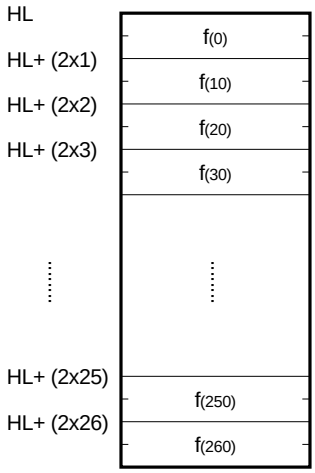
From the similarity shown in Figure 3-4, the following expression is given:

$$f_{(N)} - f_{(S)} : f_{(S+10)} - f_{(S)} = N - S : S + 10 - S$$

$$\therefore f_{(N)} = f_{(S)} + \frac{(f_{(S+10)} - f_{(S)})(N - S)}{10} \qquad \cdots\cdots (2)$$

So, for an arbitrary point (N), the value of $f_{(N)}$ can be found.

Figure 3-5. Data Table



An explanation of a program for a sample interpolation computation follows.

As shown in Figure 3-5, the start address of a data table is set in register HL.

Point N is given using the accumulator. Then, by using an interpolation computation, $f_{(N)}$ is found from the values (2-byte data) of $f_{(0)}$, $f_{(10)}$, $f_{(20)}$, ..., $f_{(260)}$ in the data table in memory.

(1) Processing procedure

<1> S is found from Expression (1).

<2> The found value of S is multiplied by 2 because each data item ($f_{(0)}$, $f_{(10)}$, $f_{(20)}$, ..., $f_{(260)}$) in the data table is given on a two-byte basis.

<3> By adding the value of S obtained in **<2>** to the start address of the data table, the address where $f_{(S)}$ is stored is found.

<4> A comparison is made between $f_{(S)}$ and $f_{(S+10)}$. If $f_{(S)}$ is greater, the contents of $f_{(S)}$ are exchanged with the contents of $f_{(S+10)}$, and

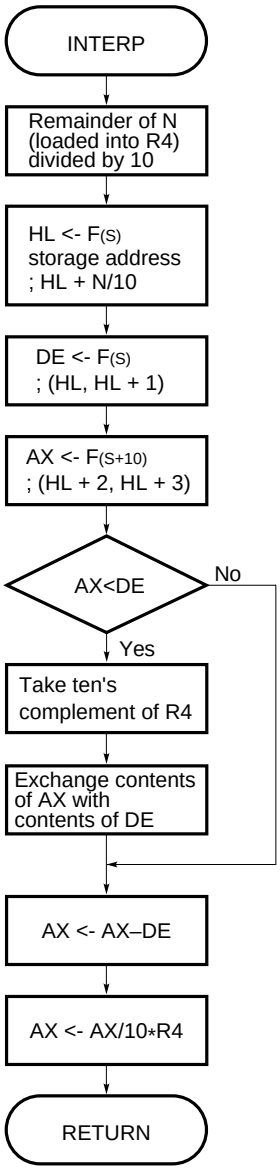
10 - (N-S) is used for (N-S).

<5> $f_{(N)}$ is found from Expression (2).

(2) Number of steps

55 bytes

Fowchart



Assemble list

```
ALN0   STN0  ADRS   OBJECT    M  I  SOURCE      STATEMENT

   1      1                        $       TITLE   ('interpolation polynomial')
   2      2                                NAME    INTER
   3      3                        ;*********************************************
   4      4                        ;*      interpolation polynomial               *
   5      5                        ;*              input condition                *
   6      6                        ;*                Acc <- N                     *
   7      7                        ;*                HL  <- data table top address*
   8      8                        ;*              output condition               *
   9      9                        ;*                AX  <- result                *
  10     10                        ;*  F(N) = F(S) + ( F(S+10)-F(S) ) * (N-S) /10 *
  11     11                        ;*                                             *
  12     12                        ;*              RSS <- 0                       *
  13     13                        ;*********************************************
  14     14                                PUBLIC  INTERP
  15     15                        ;
  16     16  ----                          CSEG
  17     17                                RSS     0
  18     18  0000                 INTERP:
  19     19  0000    B800                  MOV     X,#0
  20     20  0002    BD00                  MOV     R5,#0
  21     21  0004    D8                    XCH     A,X             ; AX <- N
  22     22  0005    BC0A                  MOV     R4,#10
  23     23  0007    051C                  DIVUW   R4              ; AX <- AX / 10
  24     24  0009    31C8                  SHLW    AX,1            ; AX <- AX * 2
  25     25  000B    88E8                  ADDW    HL,AX           ; HL <- HL + AX
  26     26                        ;
  27     27  000D    1611                  MOVW    AX,[HL+]        ; DE <- (HL,HL+1)
  28     28  000F    24C8                  MOVW    DE,AX
  29     29                        ;
  30     30  0011    1651                  MOVW    AX,[HL]         ; AX <- (HL+2,HL+3)
  31     31  0013    4F                    DECW    HL
  32     32                        ;
  33     33  0014    8F0D                  CMPW    AX,DE
  34     34  0016    8209                  BNC     $INTER1
  35     35  0018    660A00                MOVW    RP3,#10         ; complement of RP2
  36     36  001B    8A6C                  SUBW    RP3,RP2
  37     37  001D    254E                  XCHW    RP2,RP3
  38     38                        ;
  39     39  001F    250D                  XCHW    AX,DE
  40     40  0021                 INTER1:
  41     41  0021    8A0D                  SUBW    AX,DE           ; AX <- AX - DE
  42     42  0023    BE0A                  MOV     R6,#10
  43     43  0025    051E                  DIVUW   R6              ; AX,R6 <- AX / 10
  44     44                        ;
  45     45  0027    BF05                  MOV     R7,#5           ; getting the nearest
  46     46  0029    8F67                  CMP     R6,R7           ;      integral number
  47     47  002B    8303                  BC      $INTER2
  48     48  002D    2D0100                ADDW    AX,#1           ; increment AX
  49     49  0030                 INTER2:
  50     50  0030    052C                  MULUW   RP2             ; AX,RP2 <- AX * RP2
  51     51  0032    884D                  ADDW    RP2,DE
  52     52  0034    240C                  MOVW    AX,RP2
  53     53                        ;
  54     54  0036    56                    RET
  55     55
  56     56                                ENDS
  57     57                                END
```

```
Segment informations:

ADRS   LEN    NAME

0000   0037H  ?CSEG


          Cross-Reference List

NAME      VALUE   R  ATTR   RTYP   SEGNAME    XREFS

?CSEG               CSEG          ?CSEG          16#
INTER               MOD                           2#
INTER1      21H   R  ADDR          ?CSEG          34      40#
INTER2      30H   R  ADDR          ?CSEG          47      49#
INTERP       0H   R  ADDR   PUB    ?CSEG          14@     18#

Target chip:uPD78320
Assembly complete,      0 error(s) and         0 warning(s) found. (    0)
```

APPENDIX A INSTRUCTION SET

This appendix explains only the operations of the instructions.

See the user's manual of the product used for the instruction codes and the number of clocks for executing each instruction.

(1) Operand notation and coding format

Operands are coded in the operand field of each instruction as listed in the coding column of Table 9-1. For details of the operand format, refer to the relevant assembler specifications. When several coding forms are presented, any one of them is selected. Uppercase letters and the symbols, +, -, #, $, !, and [ ], are keywords and must be written as they are.

For immediate data, an appropriate numeric or label must be written. The symbols #, $, !, and [ ] must not be omitted when describing labels.

Table A-1. Operand Notation and Coding Format

Notation	Coding
r r1 r2	R0, R1, R2, R3, R4, R5, R6, R7, R8, R9, R10, R11, R12, R13, R14, R15 R0, R1, R2, R3, R4, R5, R6, R7 C, B
rp rp1 rp2	RP0, RP1, RP2, RP3, RP4, RP5, RP6, RP7 RP0, RP1, RP2, RP3, RP4, RP5, RP6, RP7 DE, HL, VP, UP
sfr sfrp	Special function register abbreviation Special function register abbreviation (16-bit manipulation register)
post	RP0, RP1, RP2, RP3, RP4, RP5/PSW, RP6, RP7 (Can be coded more than once. However, RP5 can only be used in a PUSH or POP instruction and PSW can only be used in a PUSHU or POPU instruction.)
mem	[DE], [HL], [DE+], [HL+], [DE-], [HL-], [VP], [UP]: Register indirect mode [DE+A], [HL+A], [DE+B], [HL+B], [VP+DE], [VP+HL]: Based indexed mode [DE+byte], [HL+byte], [VP+byte], [UP+byte], [SP+byte]: Based mode word[A], word[B], word[DE], word[HL]: Indexed mode
saddr saddrp	FE20H-FF1FH Immediate data or label FE20H-FF1EH Immediate data (bit 0 = 0, however) or label (for 16-bit manipulation)
$addr16 !addr16 addr11 addr5	0000H-FDFFH Immediate data or label: Relative addressing 0000H-FDFFH Immediate data or label: Immediate addressing (Data up to FFFFH can be coded in an MOV instruction.) 800H-FFFH Immediate data or label 40H-7EH Immediate data (bit 0 = 0, however)**Note** or label
word byte bit n	16-bit immediate data or label 8-bit immediate data or label 3-bit immediate data or label 3-bit immediate data (0 to 7)

Note Do not attempt to access word data at an odd-numbered address (bit 0 = 1).

Remarks 1. The same register name can be specified in rp and rp1, but different codes are generated.

2. Immediate addressing is effective for entire address spaces. Relative addressing is effective for the locations within a displacement range of -128 to +127 from the starting address of the next instruction.

Absolute names (R0 to R15, RP0 to RP7) and function names enable operand notations of an 8-bit register, r and r1, those of a 16-bit register, rp, rp1, and post to be described. Table A-2 lists the absolute names and corresponding function names of an 8-bit register. Table A-3 lists those of a 16-bit register.

Table A-2. Absolute Names and their Corresponding Function Names of an 8-bit Register

Absolute name	Function name		Absolute name	Function name	
	RSS=0	RSS=1		RSS=0	RSS=1
R0	X		R8	VP_L	VP_L
R1	A		R9	VP_H	VP_H
R2	C		R10	UP_L	UP_L
R3	B		R11	UP_H	UP_H
R4		X	R12	E	E
R5		A	R13	D	D
R6		C	R14	L	L
R7		B	R15	H	H

Table A-3. Absolute Names and their Corresponding Names of a 16-bit Register

Absolute name	Function name	
	RSS=0	RSS=1
RP0	AX	
RP1	BC	
RP2		AX
RP3		BC
RP4	VP	VP
RP5	UP	UP
RP6	DE	DE
RP7	HL	HL

RSS stands for the register set selection flag (bit 5 of PSW). Setting or resetting RSS switches function names for correspondence with an absolute name.

(2) Legend

A	:	A register; 8-bit accumulator
X	:	X register
B	:	B register
C	:	C register
D	:	D register
E	:	E register
H	:	H register
L	:	L register
R0-R15	:	Register 0 to register 15 (absolute name)
AX	:	Register pair (AX); 16 bit accumulator
BC	:	Register pair (BC)
DE	:	Register pair (DE)
HL	:	Register pair (HL)
RP0-RP7	:	Register pair 0 to register pair 7 (absolute name)
PC	:	Program counter
SP	:	Stack pointer
UP	:	User stack pointer
PSW	:	Program status word
CY	:	Carry flag
AC	:	Auxiliary carry flag
Z	:	Zero flag
P/V	:	Parity/overflow flag
S	:	Sign flag
TPF	:	Table position flag
RBS	:	Register bank select flag
RSS	:	Register set select flag
IE	:	Interrupt enable flag
STBC	:	Standby control register
WDM	:	Watchdog timer mode register
jdisp8	:	8-bit signed data (displacement value: -128 to +127)
( )	:	Contents of memory at an address enclosed in parentheses or at the address specified by a register enclosed in parentheses. (+) and (-) indicate that an address or the contents of a register enclosed in parentheses are incremented and decremented by one after execution of the instruction, respectively.
(( ))	:	Contents of memory at the address specified by the contents of memory at an address enclosed in parentheses (( )).
xxH	:	Hexadecimal number
x_H, x_L	:	High-order 8 bits and low-order 8 bits of 16-bit register

Phase-out/Discontinued

Instructions	Mnemonic	Operand	Byte	Operation	Flag				
					S	Z	AC	P/V	CY
8-bit data transfer	**MOV**	r1, #byte	2	r1 <- byte					
		saddr, #byte	3	(saddr) <- byte					
		sfr[Note], #byte	3	sfr <- byte					
		r, r1	2	r <- r1					
		A, r1	1	A <- r1					
		A, saddr	2	A <- (saddr)					
		saddr, A	2	(saddr) <- A					
		saddr, saddr	3	(saddr) <- (saddr)					
		A, sfr	2	A <- sfr					
		sfr, A	2	sfr <- A					
		A, mem	1-4	A <- (mem)					
		mem, A	1-4	(mem) <- A					
		A, [saddrp]	2	A <- ((saddrp))					
		[saddrp], A	2	((saddrp)) <- A					
		A, !addr16	4	A <- (addr16)					
		!addr16, A	4	(addr16) <- A					
		PSWL, #byte	3	PSW_L <- byte	x	x	x	x	x
		PSWH, #byte	3	PSW_H <- byte					
		PSWL, A	2	PSW_L <- A	x	x	x	x	x
		PSWH, A	2	PSW_H <- A					
		A, PSWL	2	A <- PSW_L					
		A, PSWH	2	A <- PSW_H					
	XCH	A, r1	1	A <-> r1					
		r, r1	2	r <-> r1					
		A, mem	2-4	A <-> (mem)					
		A, saddr	2	A <-> (saddr)					
		A, sfr	3	A <-> sfr					
		A, [saddrp]	2	A <-> ((saddrp))					
		saddr, saddr	3	(saddr) <-> (saddr)					

Note If STBC or WDM is coded in sfr, a different instruction having the different number of bytes is generated.

Remark The table below shows the symbols used in the flag column and their meaning.

Symbol	Explanation
(Blank)	No change
0	Cleared to zero.
1	Set to 1.
x	Set or reset according to the result.
P	P/V flag operates as a parity flag.
V	P/V flag operates as an overflow flag.
R	Saved values are restored.

Instructions	Mnemonic	Operand	Byte	Operation	Flag				
					S	Z	AC	P/V	CY
16-bit data transfer	**MOVW**	rp1, #word	3	rp1 <- word					
		saddrp, #word	4	(saddrp) <- word					
		sfrp, #word	4	sfrp <- word					
		rp, rp1	2	rp <- rp1					
		AX, saddrp	2	AX <- (saddrp)					
		saddrp, AX	2	(saddrp) <- AX					
		saddrp, saddrp	3	(saddrp) <- (saddrp)					
		AX, sfrp	2	AX <- sfrp					
		sfrp, AX	2	sfrp <- AX					
		rp1, !addr16	4	rp1 <- (addr16)					
		!addr16, rp1	4	(addr16) <- rp1					
		AX, mem	2-4	AX <- mem					
		mem, AX	2-4	mem <- AX					
	XCHW	AX, saddrp	2	AX <-> (saddrp)					
		AX, sfrp	3	AX <-> sfrp					
		saddrp, saddrp	3	(saddrp) <-> (saddrp)					
		rp, rp1	2	rp <-> rp1					
		AX, mem	2-4	AX <-> mem					
8-bit arithmetic/logical	**ADD**	A, #byte	2	A, CY <- A+byte	x	x	x	V	x
		saddr, #byte	3	(saddr), CY <- (saddr)+byte	x	x	x	V	x
		sfr, #byte	4	sfr, CY <- sfr+byte	x	x	x	V	x
		r, r1	2	r, CY <- r+r1	x	x	x	V	x
		A, saddr	2	A, CY <- A+(saddr)	x	x	x	V	x
		A, sfr	3	A, CY <- A+sfr	x	x	x	V	x
		saddr, saddr	3	(saddr), CY <- (saddr)+(saddr)	x	x	x	V	x
		A, mem	2-4	A, CY <- A+(mem)	x	x	x	V	x
		mem, A	2-4	(mem), CY <- (mem)+A	x	x	x	V	x
	ADDC	A, #byte	2	A, CY <- A+byte+CY	x	x	x	V	x
		saddr, #byte	3	(saddr), CY <- (saddr)+byte+CY	x	x	x	V	x
		sfr, #byte	4	sfr, CY <- sfr+byte+CY	x	x	x	V	x
		r, r1	2	r, CY <- r+r1+CY	x	x	x	V	x
		A, saddr	2	A, CY <- A+(saddr)+CY	x	x	x	V	x
		A, sfr	3	A, CY <- A+sfr+CY	x	x	x	V	x
		saddr, saddr	3	(saddr), CY <- (saddr)+(saddr)+CY	x	x	x	V	x
		A, mem	2-4	A, CY <- A+(mem)+CY	x	x	x	V	x
		mem, A	2-4	(mem), CY <- (mem)+A+CY	x	x	x	V	x

Phase-out/Discontinued

Instructions	Mnemonic	Operand	Byte	Operation	Flag				
					S	Z	AC	P/V	CY
8-bit arithmetic/logical	**SUB**	A, #byte	2	A, CY <- A-byte	x	x	x	V	x
		saddr, #byte	3	(saddr), CY <- (saddr)-byte	x	x	x	V	x
		sfr, #byte	4	sfr, CY <- sfr-byte	x	x	x	V	x
		r, r1	2	r, CY <- r-r1	x	x	x	V	x
		A, saddr	2	A, CY <- A-(saddr)	x	x	x	V	x
		A, sfr	3	A, CY <- A-sfr	x	x	x	V	x
		saddr, saddr	3	(saddr), CY <- (saddr) - (saddr)	x	x	x	V	x
		A, mem	2-4	A, CY <- A-(mem)	x	x	x	V	x
		mem, A	2-4	(mem), CY <- (mem)-A	x	x	x	V	x
	SUBC	A, #byte	2	A, CY <- A-byte-CY	x	x	x	V	x
		saddr, #byte	3	(saddr), CY <- (saddr)-byte-CY	x	x	x	V	x
		sfr, #byte	4	sfr, CY <- sfr-byte-CY	x	x	x	V	x
		r, r1	2	r, CY <- r-r1-CY	x	x	x	V	x
		A, saddr	2	A, CY <- A-(saddr)-CY	x	x	x	V	x
		A, sfr	3	A, CY <- A-sfr-CY	x	x	x	V	x
		saddr, saddr	3	(saddr), CY <- (saddr)-(saddr)-CY	x	x	x	V	x
		saddr, saddr	3	(saddr), CY <- (saddr)-(saddr)	x	x	x	V	x
		A, mem	2-4	A, CY <- A-(mem)-CY	x	x	x	V	x
		mem, A	2-4	(mem), CY <- (mem)-A-CY	x	x	x	V	x
	AND	A, #byte	2	A <- A∧byte	x	x		P	
		saddr, #byte	3	(saddr) <- (saddr)∧byte	x	x		P	
		sfr, #byte	4	sfr <- sfr∧byte	x	x		P	
		r, r1	2	r <- r∧r1	x	x		P	
		A, saddr	2	A <- A∧ (saddr)	x	x		P	
		A, sfr	3	A <- A∧sfr	x	x		P	
		saddr, saddr	3	(saddr) <- (saddr)∧(saddr)	x	x		P	
		A, mem	2-4	A <- A∧ (mem)	x	x		P	
		mem, A	2-4	(mem) <- (mem) ∧A	x	x		P	
	OR	A, #byte	2	A <- A∨byte	x	x		P	
		saddr, #byte	3	(saddr) <- (saddr)∨byte	x	x		P	
		sfr, #byte	4	sfr <- sfr∨byte	x	x		P	
		r, r1	2	r <- r∨r1	x	x		P	
		A, saddr	2	A <- A∨(saddr)	x	x		P	
		A, sfr	3	A <- A∨sfr	x	x		P	
		saddr, saddr	3	(saddr) <- (saddr)∨(saddr)	x	x		P	
		A, mem	2-4	A <- A∨(mem)	x	x		P	
		mem, A	2-4	(mem) <- (mem)∨A	x	x		P	
	XOR	A, #byte	2	A <- A∀byte	x	x		P	
		saddr, #byte	3	(saddr) <- (saddr)∀byte	x	x		P	
		sfr, #byte	4	sfr <- sfr∀byte	x	x		P	
		r, r1	2	r <- r∀r1	x	x		P	
		A, saddr	2	A <- A∀(saddr)	x	x		P	
		A, sfr	3	A <- A∀sfr	x	x		P	
		saddr, saddr	3	(saddr) <- (saddr)∀(saddr)	x	x		P	
		A, mem	2-4	A <- A∀(mem)	x	x		P	
		mem, A	2-4	(mem) <- (mem)∀A	x	x		P	

Phase-out/Discontinued

Instructions	Mnemonic	Operand	Byte	Operation	Flag				
					S	Z	AC	P/V	CY
8-bit arithmetic/logical	**CMP**	A, #byte	2	A-byte	x	x	x	V	x
		saddr, #byte	3	(saddr)-byte	x	x	x	V	x
		sfr, #byte	4	sfr-byte	x	x	x	V	x
		r, r1	2	r-r1	x	x	x	V	x
		A, saddr	2	A-(saddr)	x	x	x	V	x
		A, sfr	3	A-sfr	x	x	x	V	x
		saddr, saddr	3	(saddr)-(saddr)	x	x	x	V	x
		A, mem	2-4	A-(mem)	x	x	x	V	x
		mem, A	2-4	(mem)-A	x	x	x	V	x
16-bit arithmetic/logical	**ADDW**	AX, #word	3	AX, CY <- AX+word	x	x	x	V	x
		saddrp, #word	4	(saddrp), CY <- (saddrp)+word	x	x	x	V	x
		sfrp, #word	5	sfrp, CY <- sfrp+word	x	x	x	V	x
		rp, rp1	2	rp, CY <- rp+rp1	x	x	x	V	x
		AX, saddrp	2	AX, CY <- AX+(saddrp)	x	x	x	V	x
		AX, sfrp	3	AX, CY <- AX+sfrp	x	x	x	V	x
		saddrp, saddrp	3	(saddrp), CY <- (saddrp)+(saddrp)	x	x	x	V	x
	SUBW	AX, #word	3	AX, CY <- AX-word	x	x	x	V	x
		saddrp, #word	4	(saddrp), CY <- (saddrp)-word	x	x	x	V	x
		sfrp, #word	5	sfrp, CY <- sfrp-word	x	x	x	V	x
		rp, rp1	2	rp, CY <- rp-rp1	x	x	x	V	x
		AX, saddrp	2	AX, CY <- AX-(saddrp)	x	x	x	V	x
		AX, sfrp	3	AX, CY <- AX-sfrp	x	x	x	V	x
		saddrp, saddrp	3	(saddrp), CY <- (saddrp)-(saddrp)	x	x	x	V	x
	CMPW	AX, #word	3	AX-word	x	x	x	V	x
		saddrp, #word	4	(saddrp)-word	x	x	x	V	x
		sfrp, #word	5	sfrp-word	x	x	x	V	x
		rp, rp1	2	rp-rp1	x	x	x	V	x
		AX, saddrp	2	AX-(saddrp)	x	x	x	V	x
		AX, sfrp	3	AX-sfrp	x	x	x	V	x
		saddrp, saddrp	3	(saddrp)-(saddrp)	x	x	x	V	x
Multiply/divide	**MULU**	r1	2	AX <- A x r1					
	DIVUW	r1	2	AX (quotient), r1 (remainder) <- AX/r1					
	MULUW	rp1	2	AX (high-order 16 bits), rp1 (low-order 16 bits) <- AX x rp1					
	DIVUX	rp1	2	AXDE (quotient), rp1 (remainder) <- AXDE/rp1					
Signed multiply	**MULW**	rp1	2	AX (high-order 16 bits), rp1 (low-order 16 bits) <- AX x rp1					
Sum-of-products	**MACW**[Note]	n	3	AXDE <- (B)x(C)+AXDE B <- B+2, C <- C+2, n <- n-1 End if n=0 or P/V=1	x	x	x	V	x

Note Only for the μPD78352A, μPD78356, μPD78366A, and μPD78372 sub-series

Phase-out/Discontinued

Instructions	Mnemonic	Operand	Byte	Operation	Flag S	Z	AC	P/V	CY

Sum-of-products with overflow and underflow indication	**MACSW**Note 1	n	3	AXDE <- (B)x(C)+AXDE
				B <- B+2, C <- C+2, n <- n-1
				if overflow (P/V=1) then
				AXDE <- 7FFFFFFFH
				if underflow (P/V=1) then
				AXDE <- 80000000H
				End if n=0 or P/V=1
					x	x	x	V	x

Correla-tion	**SACW**Note 1	[DE+],[HL+]	4	AX <- AX+|(DE)-(HL)| DE <- DE+2, HL <- HL+2, C <- C-1 End if C=0 or CY=1	x	x	x	V	x
Table shift	**MOVTBLW**Note 2	!addr16, n	4	(addr16+2) <- (addr16), n <- n-1 addr16 <- addr16-2, End if n=0					
Increment/decrement	**INC**	r1	1	r1 <- r1+1	x	x	x	V	
		saddr	2	(saddr) <- (saddr)+1	x	x	x	V	
	DEC	r1	1	r1 <- r1-1	x	x	x	V	
		saddr	2	(saddr) <- (saddr)-1	x	x	x	V	
	INCW	rp2	1	rp2 <- rp2+1					
		saddrp	3	(saddrp) <- (saddrp)+1					
	DECW	rp2	1	rp2 <- rp2-1					
		saddrp	3	(saddrp) <- (saddrp)-1					
Sihft/rotate	**ROR**	r1, n	2	(CY, $r1_7$ <- $r1_0$,$r1_{m-1}$ <- $r1_m$) x n times				P	x
	ROL	r1, n	2	(CY, $r1_0$ <- $r1_7$,$r1_{m+1}$ <- $r1_m$) x n times				P	x
	RORC	r1, n	2	(CY <- $r1_0$, $r1_7$ <- CY,$r1_{m-1}$ <- $r1_m$) x n times				P	x
	ROLC	r1, n	2	(CY <- $r1_7$, $r1_0$ <- CY,$r1_{m+1}$ <- $r1_m$) x n times				P	x
	SHR	r1, n	2	(CY <- $r1_0$, $r1_7$ <- 0,$r1_{m-1}$ <- $r1_m$) x n times	x	x	0	P	x
	SHL	r1, n	2	(CY <- $r1_7$, $r1_0$ <- 0,$r1_{m+1}$ <- $r1_m$) x n times	x	x	0	P	x
	SHRW	rp1, n	2	(CY <- $rp1_0$, $rp1_{15}$ <- 0,$rp1_{m-1}$ <- $rp1_m$) x n times	x	x	0	P	x
	SHLW	rp1, n	2	(CY <- $rp1_{15}$, $rp1_0$ <- 0,$rp1_{m+1}$ <- $rp1_m$) x n times	x	x	0	P	x
	ROR4	[rp1]	2	A_{3-0} <- $(rp1)_{3-0}$, $(rp1)_{7-4}$ <- A_{3-0}, $(rp1)_{3-0}$ <- $(rp1)_{7-4}$					
	ROL4	[rp1]	2	A_{3-0} <- $(rp1)_{7-4}$, $(rp1)_{3-0}$ <- A_{3-0}, $(rp1)_{7-4}$ <- $(rp1)_{3-0}$					
BCD adjust	**ADJBA**		2	Decimal Adjust Accumulator	x	x	x	P	x
	ADJBS		2	Decimal Adjust Accumulator	x	x	x	P	x
Data conversion	**CVTBW**		1	When A_7 = 0, X <- A, A <- 00H When A_7 = 1, X <- A, A <- FFH					

Notes 1. Only for the μPD78356, μPD78366A, and μPD78372 sub-series

2. Only for the μPD78352A, μPD78356, μPD78366A, and μPD78372 sub-series

Remarks 1. The address range for the table shift instruction is FE00H to FEFFH.

2. In the shift/rotate instructions, n indicates the number of shifts or rotations.

Instructions	Mnemonic	Operand	Byte	Operation	Flag				
					S	Z	AC	P/V	CY
Bit manipulation	**XOR1**	CY, saddr.bit	3	CY <- CY∀(saddr.bit)					x
		CY, sfr.bit	3	CY <- CY∀sfr.bit					x
		CY, A.bit	2	CY <- CY∀A.bit					x
		CY, X.bit	2	CY <- CY∀X.bit					x
		CY, PSWH.bit	2	CY <- CY∀PSW_H.bit					x
		CY, PSWL.bit	2	CY <- CY∀PSW_L.bit					x
	SET1	saddr.bit	2	(saddr.bit) <- 1					
		sfr.bit	3	sfr.bit <- 1					
		A.bit	2	A.bit <- 1					
		X.bit	2	X.bit <- 1					
		PSWH.bit	2	PSW_H.bit <- 1					
		PSWL.bit	2	PSW_L.bit <- 1	x	x	x	x	x
	CLR1	saddr.bit	2	(saddr.bit) <- 0					
		sfr.bit	3	sfr.bit <- 0					
		A.bit	2	A.bit <- 0					
		X.bit	2	X.bit <- 0					
		PSWH.bit	2	PSW_H.bit <- 0					
		PSWL.bit	2	PSW_L.bit <- 0	x	x	x	x	x
	NOT1	saddr.bit	3	(saddr.bit) <- $\overline{(saddr.bit)}$					
		sfr.bit	3	sfr.bit <- $\overline{sfr.bit}$					
		A.bit	2	A.bit <- $\overline{A.bit}$					
		X.bit	2	X.bit <- $\overline{X.bit}$					
		PSWH.bit	2	PSW_H.bit <- $\overline{PSW_H.bit}$					
		PSWL.bit	2	PSW_L.bit <- $\overline{PSW_L.bit}$	x	x	x	x	x
	SET1	CY	1	CY <- 1					1
	CLR1	CY	1	CY <- 0					0
	NOT1	CY	1	CY <- $\overline{CY}$					x
	MOV1	CY, saddr.bit	3	CY <- (saddr.bit)					x
		CY, sfr.bit	3	CY <- sfr.bit					x
		CY, A.bit	2	CY <- A.bit					x
		CY, X.bit	2	CY <- X.bit					x
		CY, PSWH.bit	2	CY <- PSW_H.bit					x
		CY, PSWL.bit	2	CY <- PSW_L.bit					x
		saddr.bit, CY	3	(saddr.bit) <- CY					
		sfr.bit, CY	3	sfr.bit <- CY					
		A.bit, CY	2	A.bit <- CY					
		X.bit, CY	2	X.bit <- CY					
		PSWH.bit, CY	2	PSW_H.bit <- CY					
		PSWL.bit, CY	2	PSW_L.bit <- CY					

Phase-out/Discontinued

Instructions	Mnemonic	Operand	Byte	Operation	Flag S	Z	AC	P/V	CY
Bit manipulation	**AND1**	CY, saddr.bit	3	CY <- CY∧(saddr.bit)					x
		CY, /saddr.bit	3	CY <- CY∧($\overline{\text{saddr.bit}}$)					x
		CY, sfr.bit	3	CY <- CY∧sfr.bit					x
		CY, /sfr.bit	3	CY <- CY∧$\overline{\text{sfr.bit}}$					x
		CY, A.bit	2	CY <- CY∧A.bit					x
		CY, /A.bit	2	CY <- CY∧$\overline{\text{A.bit}}$					x
		CY, X.bit	2	CY <- CY∧X.bit					x
		CY, /X.bit	2	CY <- CY∧$\overline{\text{X.bit}}$					x
		CY, PSWH.bit	2	CY <- CY∧PSW_H.bit					x
		CY, /PSWH.bit	2	CY <- CY∧$\overline{PSW_H.bit}$					x
		CY, PSWL.bit	2	CY <- CY∧PSW_L.bit					x
		CY, /PSWL.bit	2	CY <- CY∧$\overline{PSW_L.bit}$					x
	OR1	CY, saddr.bit	3	CY <- CY∨(saddr.bit)					x
		CY, /saddr.bit	3	CY <- CY∨($\overline{\text{saddr.bit}}$)					x
		CY, sfr.bit	3	CY <- CY∨sfr.bit					x
		CY, /sfr.bit	3	CY <- CY∨$\overline{\text{sfr.bit}}$					x
		CY, A.bit	2	CY <- CY∨A.bit					x
		CY, /A.bit	2	CY <- CY∨$\overline{\text{A.bit}}$					x
		CY, X.bit	2	CY <- CY∨X.bit					x
		CY, /X.bit	2	CY <- CY∨$\overline{\text{X.bit}}$					x
		CY, PSWH.bit	2	CY <- CY∨PSW_H.bit					x
		CY, /PSWH.bit	2	CY <- CY∨$\overline{PSW_H.bit}$					x
		CY, PSWL.bit	2	CY <- CY∨PSW_L.bit					x
		CY, /PSWL.bit	2	CY <- CY∨$\overline{PSW_L.bit}$					x
Call/return	**CALL**	!addr16	3	(SP-1) <- $(PC+3)_H$, (SP-2) <- $(PC+3)_L$, PC <- addr16, SP <- SP-2					
	CALLF	!addr11	2	(SP-1) <- $(PC+2)_H$, (SP-2) <- $(PC+2)_L$, PC_{15-11} <- 00001, PC_{10-0} <- addr11, SP <- SP-2					
	CALLT	[addr5]	1	(SP-1) <- $(PC+1)_H$, (SP-2) <- $(PC+1)_L$, PC_H <- (TPF, 000000001, addr5+1), PC_L <- (TPF, 000000001, addr5), SP <- SP-2					
	CALL	rp1	2	(SP-1) <- $(PC+2)_H$, (SP-2) <- $(PC+2)_L$, PC_H <- $rp1_H$, PC_L <- $rp1_L$, SP <- SP-2					
		[rp1]	2	(SP-1) <- $(PC+2)_H$, (SP-2) <- $(PC+2)_L$, PC_H <- (rp1+1), PC_L <- (rp1), SP <- SP-2					
	BRK		1	(SP-1) <- PSW_H, (SP-2) <- PSW_L (SP-3) <- $(PC+1)_H$, (SP-4) <- $(PC+1)_L$, PC_L <- (003EH), PC_H <- (003FH), SP <- SP-4, IE <- 0					
	RET		1	PC_L <- (SP), PC_H <- (SP+1), SP <- SP+2					
	RETB		1	PC_L <- (SP), PC_H <- (SP+1), PSW_L <- (SP+2), PSW_H <- (SP+3), SP <- SP+4	R	R	R	R	R
	RETI		1	PC_L <- (SP), PC_H <- (SP+1), PSW_L <- (SP+2), PSW_H <- (SP+3), SP <- SP+4, EOS <- 0	R	R	R	R	R

Phase-out/Discontinued

Instructions	Mnemonic	Operand	Byte	Operation	Flag S	Z	AC	P/V	CY
Stack manipulation	**PUSH**	sfrp	3	(SP-1) <- sfr_H (SP-2) <- sfr_L SP <- SP-2					
		post	2	{(SP-1) <- $post_H$, (SP-2) <- $post_L$, SP <- SP-2} x n times**Note**					
		PSW	1	(SP-1) <- PSW_H, (SP-2) <- PSW_L, SP <- SP-2					
	PUSHU	post	2	{(UP-1) <- $post_H$, (UP-2) <- $post_L$, UP <- UP-2} x n times**Note**					
	POP	sfrp	3	sfr_L <- (SP) sfr_H <- (SP+1) SP <- SP+2					
		post	2	{$post_L$ <- (SP), $post_H$ <- (SP+1), SP <- SP+2} x n times**Note**					
		PSW	1	PSW_L <- (SP), PSW_H <- (SP+1), SP <- SP+2	R	R	R	R	R
	POPU	post	2	{$post_L$ <- (UP), $post_H$ <- (UP+1), UP <- UP+2} x n times**Note**					
	MOVW	SP, #word	4	SP <- word					
		SP, AX	2	SP <- AX					
		AX, SP	2	AX <- SP					
	INCW	SP	2	SP <- SP+1					
	DECW	SP	2	SP <- SP-1					
Special	**CHKL**	sfr	3	(Pin level)$\forall$(Signal level before output buffer)	x	x		P	
	CHKLA	sfr	3	A <- {(Pin level)$\vee$(Signal level before output buffer)}	x	x		P	
Unconditional branch	**BR**	!addr16	3	PC <- addr16					
		rp1	2	PC_H <- $rp1_H$, PC_L <- $rp1_L$					
		[rp1]	2	PC_H <- (rp1+1), PC_L <- (rp1)					
		$addr16	2	PC <- PC+2+jdisp8					

Note n indicates the number of registers specified in post.

Phase-out/Discontinued

Instructions	Mnemonic	Operand	Byte	Operation	Flag S Z AC P/V CY
Conditional branch	**BC**	$addr16	2	PC <- PC+2+jdisp8 if CY=1	
	BL	$addr16	2	PC <- PC+2+jdisp8 if CY=1	
	BNC	$addr16	2	PC <- PC+2+jdisp8 if CY=0	
	BNL	$addr16	2	PC <- PC+2+jdisp8 if CY=0	
	BZ	$addr16	2	PC <- PC+2+jdisp8 if Z=1	
	BE	$addr16	2	PC <- PC+2+jdisp8 if Z=1	
	BNZ	$addr16	2	PC <- PC+2+jdisp8 if Z=0	
	BNE	$addr16	2	PC <- PC+2+jdisp8 if Z=0	
	BV	$addr16	2	PC <- PC+2+jdisp8 if P/V=1	
	BPE	$addr16	2	PC <- PC+2+jdisp8 if P/V=1	
	BNV	$addr16	2	PC <- PC+2+jdisp8 if P/V=0	
	BPO	$addr16	2	PC <- PC+2+jdisp8 if P/V=0	
	BN	$addr16	2	PC <- PC+2+jdisp8 if S=1	
	BP	$addr16	2	PC <- PC+2+jdisp8 if S=0	
	BGT	$addr16	3	PC <- PC+3+jdisp8 if (P/V⊽S)∨Z=0	
	BGE	$addr16	3	PC <- PC+3+jdisp8 if P/V⊽S=0	
	BLT	$addr16	3	PC <- PC+3+jdisp8 if P/V⊽S=1	
	BLE	$addr16	3	PC <- PC+3+jdisp8 if (P/V⊽S)∨Z=1	
	BH	$addr16	3	PC <- PC+3+jdisp8 if Z∨CY=0	
	BNH	$addr16	3	PC <- PC+3+jdisp8 if Z∨CY=1	
	BT	saddr.bit, $addr16	3	PC <- PC+3+jdisp8 if (saddr.bit)=1	
		sfr.bit, $addr16	4	PC <- PC+4+jdisp8 if sfr.bit=1	
		A.bit, $addr16	3	PC <- PC+3+jdisp8 if A.bit=1	
		X.bit, $addr16	3	PC <- PC+3+jdisp8 if X.bit=1	
		PSWH.bit, $addr16	3	PC <- PC+3+jdisp8 if PSW_H.bit=1	
		PSWL.bit, $addr16	3	PC <- PC+3+jdisp8 if PSW_L.bit=1	
	BF	saddr.bit, $addr16	4	PC <- PC+4+jdisp8 if (saddr.bit)=0	
		sfr.bit, $addr16	4	PC <- PC+4+jdisp8 if sfr.bit=0	
		A.bit, $addr16	3	PC <- PC+3+jdisp8 if A.bit=0	
		X.bit, $addr16	3	PC <- PC+3+jdisp8 if X.bit=0	
		PSWH.bit, $addr16	3	PC <- PC+3+jdisp8 if PSW_H.bit=0	
		PSWL.bit, $addr16	3	PC <- PC+3+jdisp8 if PSW_L.bit=0	
	BTCLR	saddr.bit, $addr16	4	PC <- PC+4+jdisp8 if (saddr.bit)=1 then reset (saddr.bit)	
		sfr.bit, $addr16	4	PC <- PC+4+jdisp8 if sfr.bit=1 then reset sfr.bit	
		A.bit, $addr16	3	PC <- PC+3+jdisp8 if A.bit=1 then reset A.bit	
		X.bit, $addr16	3	PC <- PC+3+jdisp8 if X.bit=1 then reset X.bit	
		PSWH.bit, $addr16	3	PC <- PC+3+jdisp8 if PSW_H.bit=1 then reset PSW_H.bit	
		PSWL.bit, $addr16	3	PC <- PC+3+jdisp8 if PSW_L.bit=1 then reset PSW_L.bit	x x x x x

Phase-out/Discontinued

Instructions	Mnemonic	Operand	Byte	Operation	Flag S	Z	AC	P/V	CY
Conditional branch	**BFSET**	saddr.bit, $addr16	4	PC <- PC+4+jdisp8 if (saddr.bit)=0 then set (saddr.bit)					
		sfr.bit, $addr16	4	PC <- PC+4+jdisp8 if sfr.bit=0 then set sfr.bit					
		A.bit, $addr16	3	PC <- PC+3+jdisp8 if A.bit=0 then set A.bit					
		X.bit, $addr16	3	PC <- PC+3+jdisp8 if X.bit=0 then set X.bit					
		PSWH.bit, $addr16	3	PC <- PC+3+jdisp8 if PSW_H.bit=0 then set PSW_H.bit					
		PSWL.bit, $addr16	3	PC <- PC+3+jdisp8 if PSW_L.bit=0 then set PSW_L.bit	x	x	x	x	x
	DBNZ	r2, $addr16	2	r2 <- r2-1, then PC <- PC+2+jdisp8 if r2 ≠ 0					
		saddr, $addr16	3	(saddr) <- (saddr)-1, then PC <- PC+3+jdisp8 if (saddr) ≠ 0					
Context switching	**BRKCS**	RBn	2	PC_H <-> R5, PC_L <-> R4, R7 <- PSW_H, R6 <- PSW_L, RBS2-0 <- n, RSS <- 0, IE <- 0					
	RETCS	!addr16	3	PC_H <- R5, PC_L <- R4, R5, R4 <- addr16, PSW_H <- R7, PSW_L <- R6	R	R	R	R	R
	RETCSB	!addr16	4	PC_H <- R5, PC_L <- R4, R5 <- $addr16_H$, R4 <- $addr16_L$ PSW_H <- R7, PSW_L <- R6	R	R	R	R	R
String	**MOVM**	[DE+], A	2	(DE+) <- A, C <- C-1 End if C=0					
		[DE-], A	2	(DE-) <- A, C <- C-1 End if C=0					
	MOVBK	[DE+], [HL+]	2	(DE+) <- (HL+), C <- C-1 End if C=0					
		[DE-], [HL-]	2	(DE-) <- (HL-), C <- C-1 End if C=0					
	XCHM	[DE+], A	2	(DE+) <-> A, C <- C-1 End if C=0					
		[DE-], A	2	(DE-) <-> A, C <- C-1 End if C=0					
	XCHBK	[DE+], [HL+]	2	(DE+) <-> (HL+), C <- C-1 End if C=0					
		[DE-], [HL-]	2	(DE-) <-> (HL-), C <- C-1 End if C=0					
	CMPME	[DE+], A	2	(DE+)-A, C <- C-1 End if C=0 or Z=0	x	x	x	V	x
		[DE-], A	2	(DE-)-A, C <- C-1 End if C=0 or Z=0	x	x	x	V	x
	CMPBKE	[DE+], [HL+]	2	(DE+)-(HL+), C <- C-1 End if C=0 or Z=0	x	x	x	V	x
		[DE-], [HL-]	2	(DE-)-(HL-), C <- C-1 End if C=0 or Z=0	x	x	x	V	x
	CMPMNE	[DE+], A	2	(DE+)-A, C <- C-1 End if C=0 or Z=1	x	x	x	V	x
		[DE-], A	2	(DE-)-A, C <- C-1 End if C=0 or Z=1	x	x	x	V	x

Phase-out/Discontinued

Instructions	Mnemonic	Operand	Byte	Operation	Flag				
					S	Z	AC	P/V	CY
String	**CMPBKNE**	[DE+], [HL+]	2	(DE+)-(HL+), C <- C-1 End if C=0 or Z=1	x	x	x	V	x
		[DE-], [HL-]	2	(DE-)-(HL-), C <- C-1 End if C=0 or Z=1	x	x	x	V	x
	CMPMC	[DE+], A	2	(DE+)-A, C <- C-1 End if C=0 or CY=0	x	x	x	V	x
		[DE-], A	2	(DE-)-A, C <- C-1 End if C=0 or CY=0	x	x	x	V	x
	CMPBKC	[DE+], [HL+]	2	(DE+)-(HL+), C <- C-1 End if C=0 or CY=0	x	x	x	V	x
		[DE-], [HL-]	2	(DE-)-(HL-), C <- C-1 End if C=0 or CY=0	x	x	x	V	x
	CMPMNC	[DE+], A	2	(DE+)-A, C <- C-1 End if C=0 or CY=1	x	x	x	V	x
		[DE-], A	2	(DE-)-A, C <- C-1 End if C=0 or CY=1	x	x	x	V	x
	CMPBKNC	[DE+], [HL+]	2	(DE+)-(HL+), C <- C-1 End if C=0 or CY=1	x	x	x	V	x
		[DE-], [HL-]	2	(DE-)-(HL-), C <- C-1 End if C=0 or CY=1	x	x	x	V	x
CPU control	**MOV**	STBC, #byte	4	STBC <- byte**Note**					
		WDM, #byte	4	WDM <- byte**Note**					
	SWRS		1	RSS <- $\overline{\text{RSS}}$					
	SEL	RBn	2	RBS2-0 <- n, RSS <- 0					
		RBn, ALT	2	RBS2-0 <- n, RSS <- 1					
	NOP		1	No Operation					
	EI		1	IE <- 1 (Enable Interrupt)					
	DI		1	IE <- 0 (Disable Interrupt)					

Note Operation when an op-code trap interrupt occurs:
$(SP-1) \leftarrow PSW_H$, $(SP-2) \leftarrow PSW_L$,
$(SP-3) \leftarrow (PC-4)_H$, $(SP-4) \leftarrow (PC-4)_L$,
$PC_L \leftarrow (003CH)$, $PC_H \leftarrow (003DH)$,
SP <- SP-4, IE <- 0

[MEMO]

*

APPENDIX B REVISION HISTORY

The revision history is shown below. The chapters described in the revised-chapter column indicate those for the corresponding edition.

Edition	Major changes	Revised chapter
Third	The following products have been added. μPD78320(A1), μPD78320(A2), μPD78322(A1), μPD78322(A2), μPD78323(A), μPD78323(A1), μPD78323(A2), μPD78324(A), μPD78324(A1), μPD78324(A2), μPD78P324(A), μPD78P324(A1), μPD78P324(A2), μPD78330(A), μPD78330(A1), μPD78330(A2), μPD78334(A), μPD78334(A1), μPD78334(A2), μPD78P334(A), μPD78P334(A1), μPD78P334(A2), μPD78350, μPD78350A, μPD78352A, μPD78P352, μPD78355, μPD78356, μPD78P356, μPD78355A, μPD78356A, μPD78P356A, μPD78362, μPD78P364, μPD78365, μPD78366, μPD78P368, μPD78370, μPD78372, μPD78P372	Throughout
	The following products have already been developed. μPD78323, μPD78324, μPD78P334	
Fourth	The following products have been added. μPD78356(A), μPD78P356(A), μPD78361A), μPD78362A, μPD78P364A, μPD78363A, μPD78365A, μPD78366A, μPD78368A, μPD78P368A, μPD78372(A), μPD78372(A1), μPD78372(A2), μPD78P372(A), μPD78P372(A1), μPD78P372(A2)	Throughout
	The following products have been deleted. μPD78355A, μPD78356A, μPD78P356A, μPD78362, μPD78P364, μPD78365, μPD78366, μPD78P368, μPD78370, μPD78372, μPD78P372	
	The following products have already been developed. μPD78P324, μPD78P324(A), μPD78P324(A1), μPD78P324(A2), μPD78350A, μPD78355, μPD78356, μPD78P356	

[MEMO]

Phase-out/Discontinued

Facsimile Message

From:

Name

Company

Tel. FAX

Address

Although NEC has taken all possible steps to ensure that the documentation supplied to our customers is complete, bug free and up-to-date, we readily accept that errors may occur. Despite all the care and precautions we've taken, you may encounter problems in the documentation. Please complete this form whenever you'd like to report errors or suggest improvements to us.

Thank you for your kind support.

North America
NEC Electronics Inc.
Corporate Communications Dept.
Fax: 1-800-729-9288
1-408-588-6130

Europe
NEC Electronics (Europe) GmbH
Technical Documentation Dept.
Fax: +49-211-6503-274

South America
NEC do Brasil S.A.
Fax: +55-11-889-1689

Hong Kong, Philippines, Oceania
NEC Electronics Hong Kong Ltd.
Fax: +852-2886-9022/9044

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

Taiwan
NEC Electronics Taiwan Ltd.
Fax: 02-719-5951

Asian Nations except Philippines
NEC Electronics Singapore Pte. Ltd.
Fax: +65-250-3583

Japan
NEC Corporation
Semiconductor Solution Engineering Division
Technical Information Support Dept.
Fax: 044-548-7900

I would like to report the following error/make the following suggestion:

Document title: _____

Document number: _____ Page number: _____

If possible, please fax the referenced page or drawing.

Document Rating	Excellent	Good	Acceptable	Poor
Clarity	❑	❑	❑	❑
Technical Accuracy	❑	❑	❑	❑
Organization	❑	❑	❑	❑

Phase-out/Discontinued