RENESAS

APPLICATION NOTE

78K/Ⅱ SERIES

8-BIT SINGLE-CHIP MICROCOMPUTER

BASIC GUIDE

μPD78214 SERIES
μPD78218A SERIES
μPD78224 SERIES
μPD78234 SERIES
μPD78244 SERIES

Document No. IEA-1220E
(O. D. No. IEA-607E)
Date Published December 1993 P
Printed in Japan

APPLICATION NOTE

NEC

78K/Ⅱ SERIES

8-BIT SINGLE-CHIP MICROCOMPUTER

BASIC GUIDE

μPD78214 SERIES
μPD78218A SERIES
μPD78224 SERIES
μPD78234 SERIES
μPD78244 SERIES

M7 92.6

Contents Modified or Added in This Edition

Page	Contents
Throughout	Description regarding the uPD78P224 has been deleted.

INTRODUCTION

Target reader:

This application note is designed to provide information regarding the functions of the 78K/II series to the user engineers who design application programs for a system using the 78K/II series products.

78K/II series products

- uPD78214 series	: uPD78212, uPD78213, uPD78214, uPD78P214, uPD78212(A), uPD78213(A), uPD78214(A), uPD78P214(A)
- uPD78218A series	: uPD78217A, uPD78218A, uPD78P218A, uPD78218A(A)
- uPD78224 series	: uPD78220, uPD78224, uPD78P224
- uPD78234 series	: uPD78233, uPD78234, uPD78237, uPD78238, uPD78P238, uPD78234(A), uPD78238(A)
- uPD78244 series	: uPD78243, uPD78244

Objective:

This application note is designed to provide information regarding the basic functions of the 78K/II series to the users. Application program examples are provided to enhance the user's understanding. However, it must be noted that the programs and hardware configurations shown in this application note are just examples and not intended to be used for commercial productions.

Configuration:

This application note is configured as follows:
- Outline
- Software Part
- Hardware Program examples

The following application notes are also provided separately:
- Application Part (IEA-700)
- Floating Point Operatin Program Part (IEA-686)

How to read:

Unless otherwise specifically noted, the contents of this application note are targeted for all of the 78K/II series products. The index is used to indicate whether or not there is any difference.

o How to read the index:

▬	: Can be used as is.
▭	: Some portion can be used, and some other portion cannot be used.
None	: Cannot be used as is.

Refer to the text for details.

- For examples:
  → Read according to the table of contents.

- To find out how to use instructions:
  → Read according to the table of contents.

- For examples of how to use internal hardware:
  → Read cover to cover for all examples, or refer to the table of contents or Appendix B to see a certain product.

- For the application examples for a certain product:
  → Refer to Appendix B

- For an example for a certain application example:
  → Refer to both the table of contents and Appendix B.

Quality grade:

The uPD78212(A), uPD78213(A), uPD78214(A) and uPD78P214(A) are "special" quality grade versions of the uPD78212, uPD78213, uPD78214 and uPD78P214 respectively. The uPD78218A(A) is "Special" quality grade versions of the uPD78218A. The uPD78234(A) and uPD78238(A) are also "special" quality grade versions of the uPD78234 and uPD78238, respectively.

The examples shown in this application note are designed for "standard" quality versions for general electronics equipment. When using an example shown in this application note for an application requiring "special" quality grade, the actually used components and circuits must be evaluated for the quality grade.

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

Applications:

o Standard grade
  - Printers
  - Cameras
  - Typewriters
  - PPCs
  - Electronic musical instruments
  - Air conditioners, etc.

o Special grade
  - Automotive electronics
  - Fuel control, etc.

Examples:

Significance in data expression

: The left-most side is the most significant digit, and the right-most side is the least significant digit.

Active low indication

: $\overline{\text{xxx}}$ (a line is placed on the signal name)

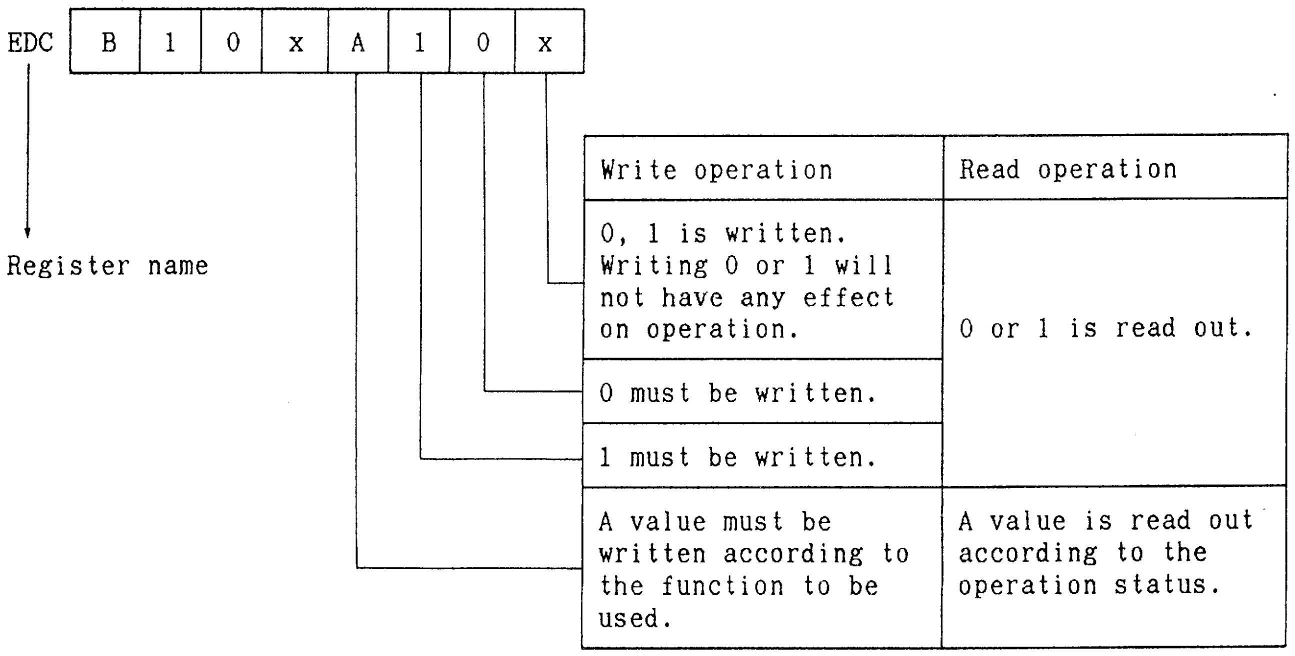
Note: Explanation of contents noted in the text.

Notes: Contents which require attention.

Remarks: Complementary description of the contents.

Numeric expression:	Binary	xxxx or xxxxB
	Decimal	xxxx
	Hexadecimal	xxxxH

Register expression



Write operation	Read operation
0, 1 is written. Writing 0 or 1 will not have any effect on operation.	0 or 1 is read out.
0 must be written.	
1 must be written.	
A value must be written according to the function to be used.	A value is read out according to the operation status.

never write any combination of code noted as "Not allowed" in register expression inthe text.

Confusing characters: 0 (zero), O (oh)
: 1 (one), l (el), I (ai)

Related documents:

o List of common documents

Document name		Document number
Users Manual Instruction Part		IEU-754
SBI User's Manual		IEM-5040
Application Note	Fundamentals	This Application Note
	Application Part	IEA-700
	Floating Point Operation Program Part	IEA-686
Selection Guide		IF-304
Development Tool Selection Guide		EF-231
Instruction Reference Table		IEM-5101
Instruction Set		IEM-5102

o Individual documents

uPD78214 series

Document name \ Product name	uPD78212	uPD78213	uPD78214	uPD78P214
Brochure	IB-5036			
Data Sheet	IC-8149	IC-7649		IC-7732
User's Manual Hardware Part	IEM-5119			
Mode Register Reference Table	IEM-5100			

Document name \ Product name	uPD78212(A)	uPD78213(A)	uPD78214(A)	uPD78P214(A)
Data Sheet	IC-8147	IC-8234		IC-8589
User's Manual Hardware Part	IEM-5119			
Mode Register Reference Table	IEM-5100			

uPD78218A series

Document name \ Product name	uPD78217A	uPD78218A	uPD78P218A	uPD78218A(A)
Brochure	IF-288			
Data Sheet	IC-8132	IC-8131	IC-8133	IC-8685
User's Manual Hardware Part	IEU-755			
Special function Register Reference Table	IEM-5532			

uPD78224 series

Product name Document name	uPD78220	uPD78224	uPD78P224
Brochure	IB-5011		
Data Sheet	IC-5457		IC-7757
User's Manual Hardware Part	IEM-5019		
Special function Register Reference Table	IEM-999		

uPD78234 series

Product name Document name	uPD78233	uPD78234	uPD78237	uPD78238	uPD78P238
Brochure	IF-207				
Data Sheet	IC-7902		IC-8348	IC-8028	IC-8030
User's Manual Hardware Part	IEU-718				
Special function Register Reference Table	IEM-5515				

Product name Document name	uPD78234(A)	uPD78238(A)
Brochure	-	
Data Sheet	IC-8146	IC-8727
User's Manual Hardware Part	IEU-718	
Special function Register Reference Table	-	

uPD78244 series

Product name / Document name	uPD78243	uPD78244
Brochure	IF-214	
Data Sheet	IC-8355	IC-8070
User's Manual Hardware Part	IEU-747	
Special function Register Reference Table	IEM-5528	

Table of Contents

List of Figures

List of Tables

CHAPTER 1 OVERVIEW

1.1 Organization of This Application Note

This Application Note contains the following items of information:

(1) Software

Chapter 2 of this Application Note presents software examples. The registers and memory areas to be used, input and output conditions, processing sequence, the number of steps required, flowchart, and program list for each application example are also presented.

(2) Hardware program examples

Chapters 3 through 10 discuss hardware examples. This chapter presents discussions on the operation, program, mode registers, input/output parameters, and registers to be used for each application example, along with the program example, flowchart, and program list necessary for the example.

1.1.1 Legend for discussion on software part

In Chapter 2, many software application examples are presented. Each example in this chapter includes these items of information:

(1) Registers

Under the heading "Registers", the registers used for the example program are presented. If the contents of the registers already used should not be destroyed, save the register contents to stack by executing the PUSH instruction, etc., before calling the example program.

(2) Memory

How the memory is to be used is indicated under this heading. Note that, when the example program has been executed, the memory work area contents become undefined.

(3) Input conditions

The input conditions necessary for using the example program are shown under this heading.

(4) Output conditions

The data or information, output as a result of executing the example program, are shown under this heading.

(5) Processing sequence

Under this heading, how the example program is executed is discussed in detail. A flowchart and a program list are also presented, along with an example program. Read the discussion under this heading while referring to the flowchart and program list.

(6) Number of steps

Since the example programs are presented as subroutines, they are linked with the main routine by using the LK78K2 linker for the RA78K/2 relocatable assembler. Therefore, each example program ends with the RET instruction. The number of steps presented under this heading includes this RET instruction.

(7) Program list

All the example programs presented under this heading are source programs. The addresses at which each example program is to be located differ, depending on the conditions under which the program has been linked.

1.1.2 Legend for discussion on hardware program examples

Chapter 3 presents application examples for hardware devices. Each example consists of the following items of information:

(1) Operation

Under the heading "Operation", the operation to be executed by the hardware device in question is presented. Occasionally, a blockdiagram of the hardware device is also shown.

(2) Program

Under this heading, the program processing sequence, the setting of the necessary mode registers and memory are indicated.

(3) Mode register setting example

Mode register setting examples are shown under this heading.

(4) Input/Output parameters

These parameters are given or output when the hardware device is to be operated.
Many programs are shown as relocatable programs. Therefore, those programs must be defined by the EQU directive in the user program and then declared to be public.

(5) Registers

Under the heading "Registers", the registers used for the example program are presented. If the contents of registers already used should not be destroyed, save the register contents to stack by executing the PUSH instruction, etc., before calling the example program.

(6) Program example

An example program for operating the hardware device is given.

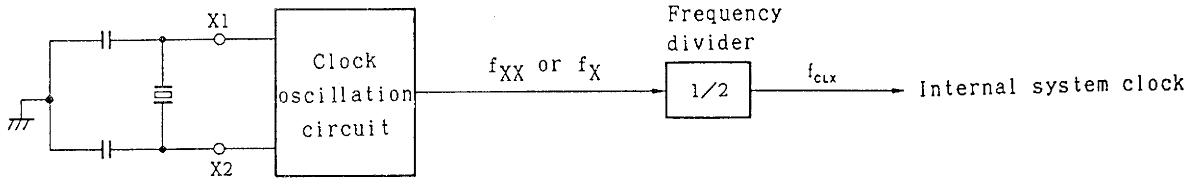
(7) Program list

All the example programs presented under this heading are source programs. The addresses at which each example program is to be located differ, depending on the conditions under which the program has been linked.

1.2 Use of Application Programs

Basically, execute the application programs as described.
Some application programs are used to call other application programs. These programs must be linked by using a linker. Some programs may require a work area. To use these programs, reserve a memory area described and accomplish public declaration.
Internal hardware devices for the microcomputers covered in this Application Note are dependent on the internal system clock for their operations. The 78K/II series products divide the frequency of the externally supplied clock (f_{XX} or f_X) by two to produce the internal system clock (f_{CLK}), as illustrated in Fig. 1-1.

Fig. 1-1 Clock Oscillation Circuit



Remarks:

f_{XX} :	Crystal/ceramic oscillation frequency
f_X :	External clock frequency
f_{CLK}:	Internal system clock frequency (= $1/2f_{XX}$ or $1/2f_X$)

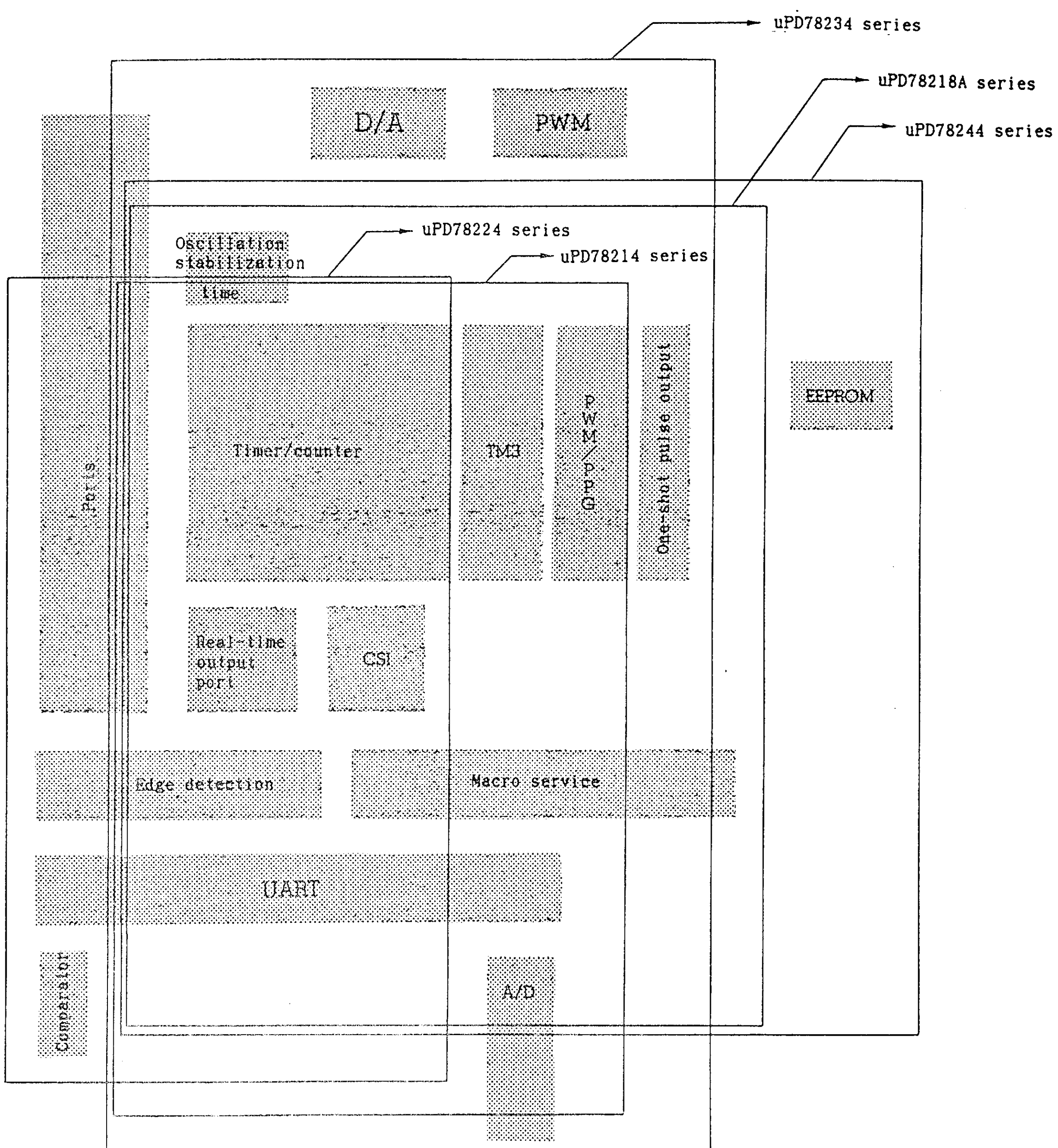
Note that the system clock is represented as f_{CLK} throughout this Application Note.

1.3 Features of 78K/II Series Products

The 78K/II series microcomputers are 78K series 8-bit single-chip microcomputers. Each of the 78K/II series microcomputers contains a high-performance 8-bit CPU, ROM, RAM, and abundant peripheral hardware in the chip.
The instruction set for the 78K/II series microcomputers includes abundant instruction, such as 16-bit operation instruction, multiplication/division instruction, and bit manipulation instructions. The data address space can be extended up to 1M bytes so that a large amount of data can be accommodated.
In addition, for OA application, the 78K/II series microcomputers are provided with peripheral hardware data transfer processing function. The real-time output port, which is effective for stepping motor control, is also provided.

The 78K/II series microcomputers are further divided into five sub-series (uPD78214 series, uPD78218A series, uPD78224 series, uPD78234 series, and uPD78244 series). Therefore, a suitable series can be selected for a desired application. The same CPU is used in these series. Only the hardware is different. Therefore, the instruction set is common to all of these series, so that programs which control individual peripheral hardware units can mostly be commonly used (refer to Fig. 1-2).
Within the sub series, only the internal memory sizes are different.

Fig. 1-2 Peripheral Hardware Correlation Diagram



CHAPTER 2 SOFTWARE PART

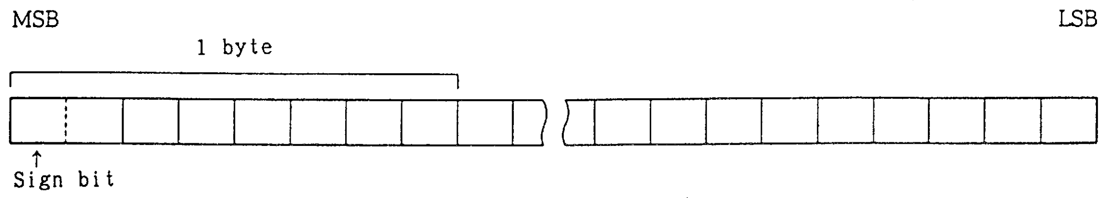
In the Software Part, operation results and numeric data are allocated on the RAM by the program. The program example shown below presents an example of how to assure the area availability.

```
        PUBLIC  BMLCND,BMLIER,BRSLT      ; Binary multiplication
        PUBLIC  DVISOR,DEND,DRMND        ; Binary division
        PUBLIC  DMLCND,DMLIER,DRSLT      ; Decimal multiplication
        PUBLIC  DIVSOR,DIVIND,RMIND      ; Decimal division
        PUBLIC  ERROR                    ; Error processing

        PUBLIC  SFLAG                    ; Sign flag
        PUBLIC  CARRY                    ; Carry data

        BSEG
SFLAG   DBIT                     ; Sign flag

SOFT_D1 DSEG    SADDR
CARRY:  DS      1                ; Carry data
                .
                .
;************************************************
;       Data area
;************************************************
SOFT_D2 DSEG
BMLCND: DS      4                ; Binary multiplication area
BMLIER: DS      4
BRSLT:  DS      8

DVISOR: DS      4                ; Binary division area
DEND:   DS      4
DRMND:  DS      4

DMLCND: DS      4                ; Decimal multiplication area
DMLIER: DS      4
DRSLT:  DS      8

DIVSOR: DS      4                ; Decimal division area
DIVIND: DS      4
RMIND:  DS      4

        CSEG

ERROR:                           ; Write error processing
```

2.1 Binary Operations

The most significant bit is a sign bit with a numeric value represented by the other bits. Negative numbers are represented in the 2s complement system.

Fig. 2-1 Binary Representation

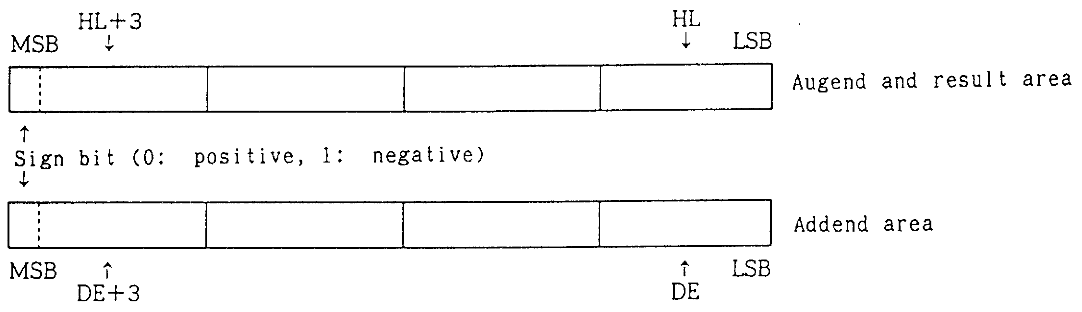


In binary operations, numeric values are located in the memory, both before and after the operations.

2.1.1 Binary addition

$$\boxed{\text{32 bits}} \leftarrow \boxed{\text{32 bits}} + \boxed{\text{32 bits}}$$

(1) Memory



(2) Registers

A, C, DE, HL

(3) Input conditions

As indicated in (1), the contents of HL, DE registers are set as follows.

. The lowest address in the memory area, where the 32-bit augend is stored, should be loaded to register HL.
. The lowest address in the area, where the 32-bit addend is stored, should be loaded to register DE.

Remarks: Addition of data of other than 4 bytes (32 bits) can also be executed by changing the BYTNUM value in the program.

(4) Output conditions

The result of the operation is stored in the memory areas (HL, HL + 1, HL + 2, and HL + 3) indicated in (1).

(5) Processing sequence

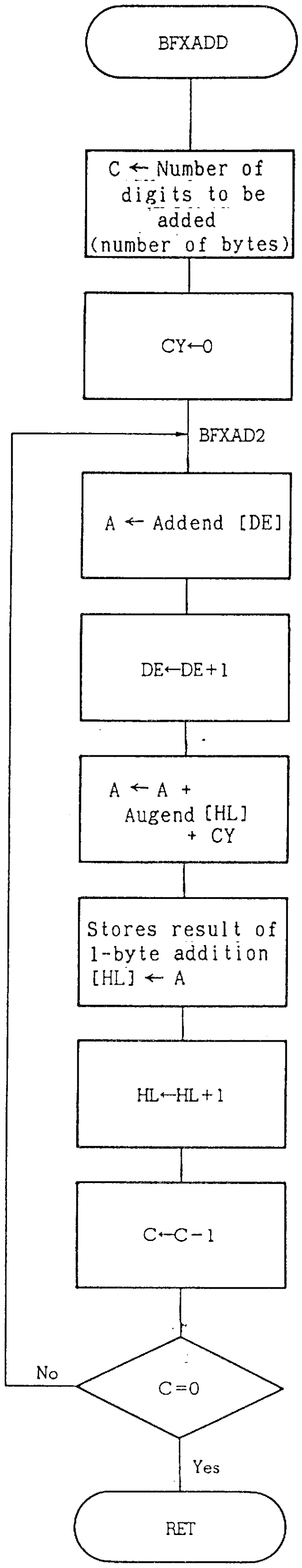
(a) 4 is set in the byte counter (register C).
(b) The carry flag is cleared to 0 in advance.
(c) One byte of data, indicated by the addend register (register DE), is read into register A, and the contents of the addend register are then incremented.
(d) One byte of data indicated by the augend register (register HL) is added with carry to the register A contents.
(e) The register A value is stored in a memory location indicated by the addend register. The addend register contents are then incremented.

(f) The byte counter contents are decremented. Steps (a) to (e) above are repeated until the byte counter contents are decremented to zero.

(6) Number of steps

7

(7) Flowchart


BFXADD
C ← Number of digits to be added (number of bytes)
CY←0
BFXAD2
A ← Addend [DE]
DE←DE+1
A ← A + Augend [HL] + CY
Stores result of 1-byte addition [HL] ← A
HL←HL+1
C←C−1
C=0
No
Yes
RET

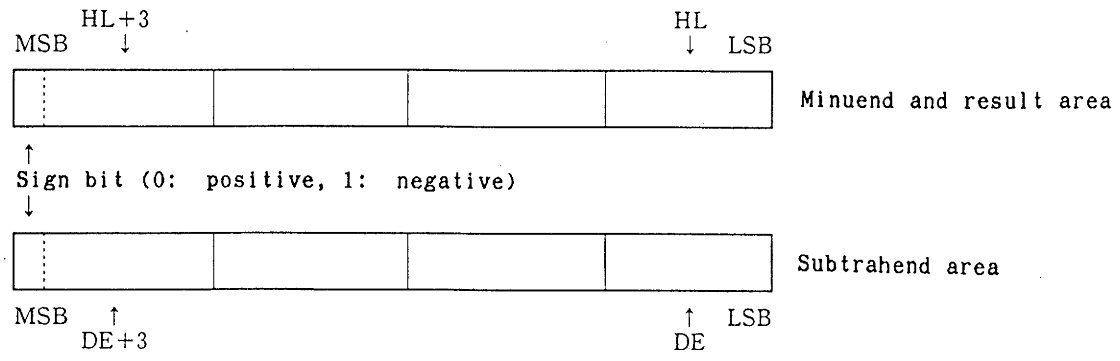
(8) Program list

```
        NAME    BFXADR

;**************************************************
;*      binary addition                           *
;*           32 bit <- 32 bit + 32 bit            *
;*       input condition                          *
;*             HL-register <- augnend top.address *
;*             DE-register <- addend  top.address *
;*       output condition                         *
;*             result <- (HL,HL+1,HL+2,HL+3)      *
;**************************************************

        PUBLIC  BFXADD
;
BYTNUM  EQU     4
;
        CSEG
BFXADD:
        MOV     C,#BYTNUM
BFXAD1:
        CLR1    CY
BFXAD2:
        MOV     A,[DE+]
        ADDC    A,[HL]
        MOV     [HL+],A
        DBNZ    C,$BFXAD2

        RET
;
        END
```

2.1.2 Binary subtraction

$$\boxed{32\text{ bits}} \leftarrow \boxed{32\text{ bits}} - \boxed{32\text{ bits}}$$

(1) Memory



(2) Registers

A, C, DE, HL

(3) Input conditions

As indicated in (1), the contents of registers HL and DE are set as follows.

. The lowest address in the memory area, where the 32-bit minuend is stored, should be loaded to register HL.

. The lowest address in the area, where the 32-bit subtrahend is stored, should be loaded to register DE.

Remarks: Subtraction of data of other than 4 bytes (32 bits) can also be executed by changing the BYTNUM value in the program.

(4) Output conditions

The operation result is stored in the memory areas (HL, HL + 1, HL + 2, and HL + 3) indicated in (1).

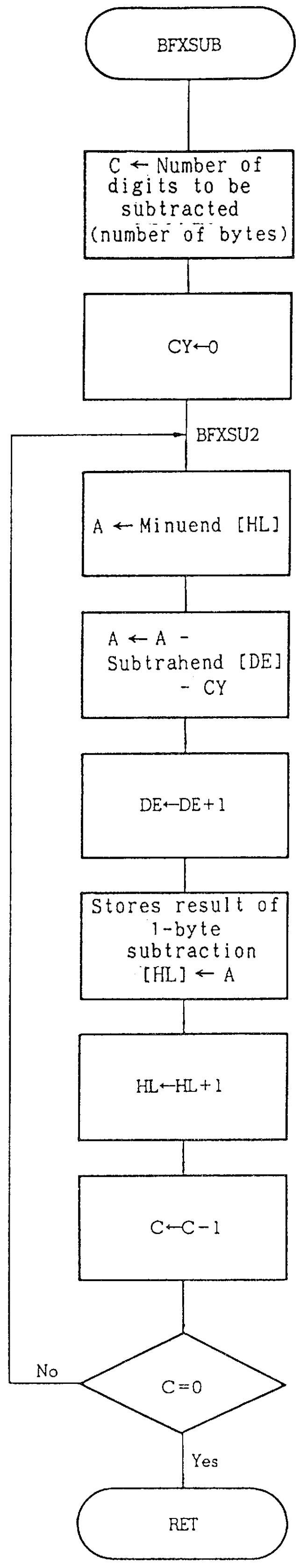
(5) Processing sequence

(a) 4 is set in the byte counter (register C).
(b) The carry flag is cleared to 0 in advance.
(c) One byte of data, indicated by the minuend register (register DE), is read into register A.
(d) One byte of data, indicated by the subtrahend register (register HL), is subtracted with carry from the register A contents. After that, the subtrahend register (register DE) contents are incremented.
(e) The register A value is stored in a memory location indicated by the minuend register. The subtrahend register contents are then incremented.
(f) The byte counter contents are decremented. Steps (a) to (e) above are repeated until the byte counter contents are decremented to zero.

(6) Number of steps

7

(7) Flowchart



BFXSUB
C ← Number of digits to be subtracted (number of bytes)
CY←0
BFXSU2
A ← Minuend [HL]
A ← A - Subtrahend [DE] - CY
DE←DE+1
Stores result of 1-byte subtraction [HL] ← A
HL←HL+1
C←C-1
C=0
No
Yes
RET

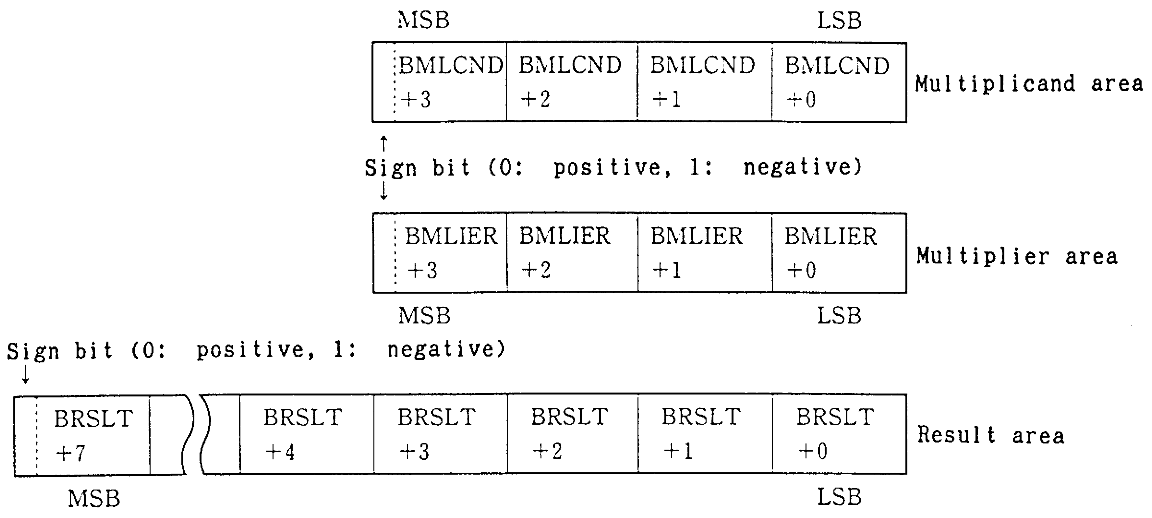
(8) Program list

```
        NAME    BFXSBR

;*****************************************************
;*      binary subtraction                            *
;*           32 bit <- 32 bit - 32 bit                *
;*       input condition                              *
;*             HL-register <- minus value top.address *
;*             DE-register <- subtrahend top.address  *
;*       output condition                             *
;*             result <- (HL,HL+1,HL+2,HL+3)          *
;*****************************************************

        PUBLIC  BFXSUB
;
BYTNUM  EQU     4
;
        CSEG

BFXSUB:
        MOV     C,#BYTNUM
BFXSU1:
        CLR1    CY
BFXSU2:
        MOV     A,[HL]
        SUBC    A,[DE+]
        MOV     [HL+],A
        DBNZ    C,$BFXSU2

        RET
;
        END
```

2.1.3 Binary multiplication

$\boxed{\text{64 bits}} \leftarrow \boxed{\text{32 bits}} \times \boxed{\text{32 bits}}$

(1) Memory



(2) Registers

X, A, C, B, DE, HL

(3) Input conditions

As indicated in (1), the 32-bit multiplicand and multiplier are stored in the following areas:

. Multiplicand: BMLCND, BMLCND + 1, BMLCND + 2, BMLCND + 3
. Multiplier : BMLIER, BMLIER + 1, BMLIER + 2, BMLIER + 3

(4) Output conditions

The operation result is stored in the result areas indicated in (1) (BRSLT, BRSLT + 1, ..., BRSLT + 7).

(5) Processing sequence

Since the multiplication instruction for 78K/II is used, the processing is analogous to hand calculation.

(a) The result area is cleared to 0.

(b) The absolute values of the multiplier and multiplicand are taken. If the sign of the multiplier is different from that of the multiplicand, the sign flag (user flag) is set to 1; if the signs are the same, the sign flag is cleared to 0.

(c) The first address for the result area is set in register HL.

(d) 0 is set as an initial value in the multiplier digit pointer (register DE) and loop counter (register B and C).

(e) One byte of multiplicand, indicated by (BMLCND + register B), is read into register X.

(f) One byte of multiplier, indicated by (BMLIER + register DE), is read into register A. The register A contents are then multiplied by the register X contents, and the result is added to the result area (register HL) contents.

(g) If a carry has been generated as a result of (f), the carry is processed. However, overflow from area BRSLT + 7 is ignored.

(h) The loop counter (register B) contents are incremented.

(i) The BRSLT + B + C value is loaded to the multiplication result pointer (register HL).

(j) The register B value is tested to see whether or not multiplication of four digits of data has been completed. If the register B value is less than 4, steps (e) to (i) above are repeated.

(k) 0 is set in the loop counter (register B) again, and the contents of the multiplicand pointer (register DE) and loop counter (register C) are incremented.

(l) The BRSLT + C value is loaded to the multiplication result pointer.

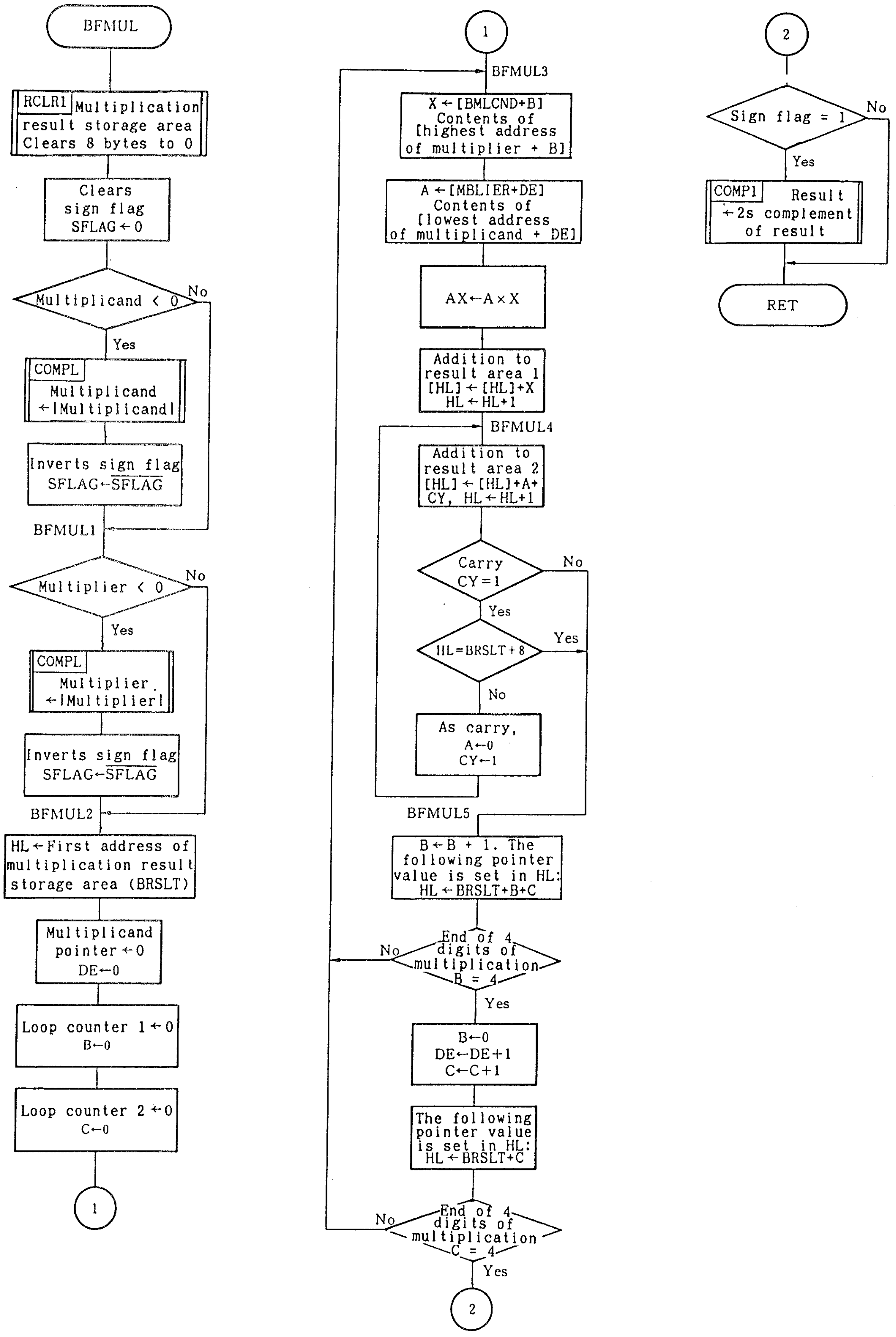
(m) The register C value is tested to see whether or not multiplication of four digits of data has been completed. If the register C value is less than 4, steps (e) to (m) above are repeated.

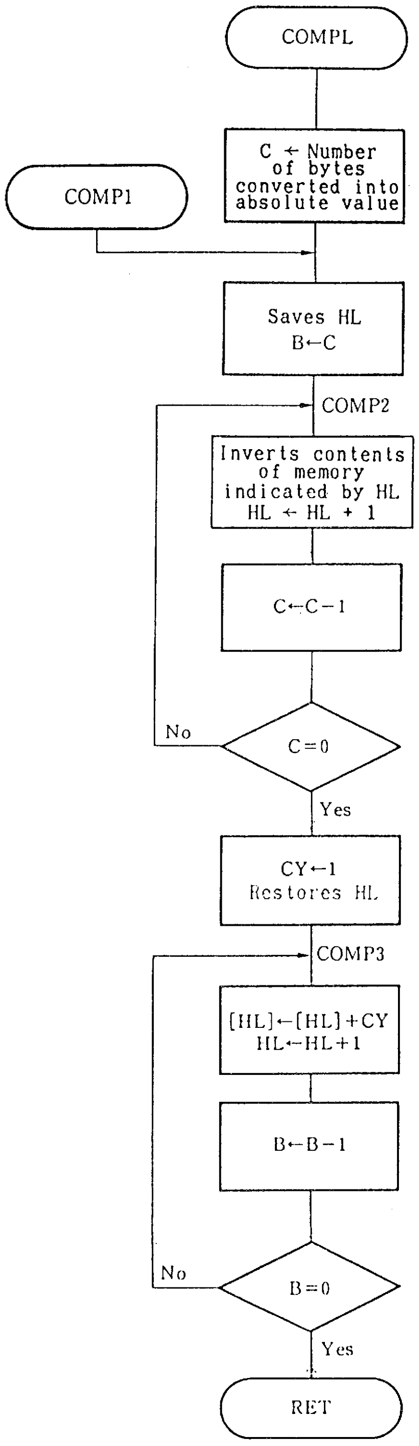
(n) If the sign flag is 1, the 2s complement for the above multiplication result is taken, which is regarded as the real result of the multiplication.

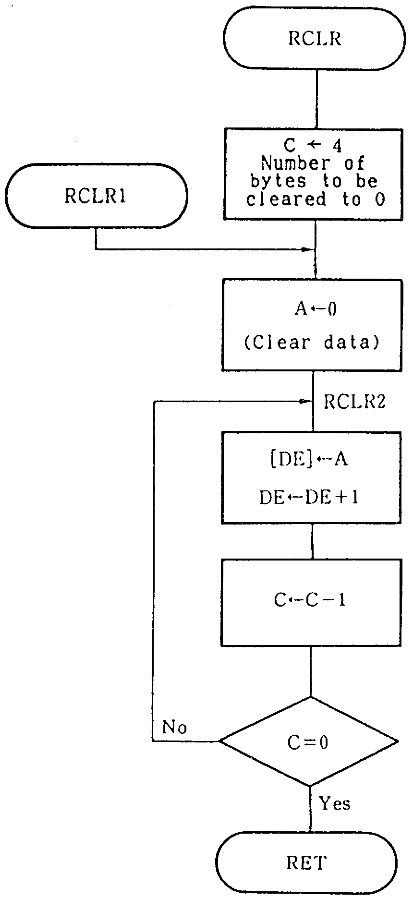
(6) Number of steps

BFMUL:	59
RCLR :	5
COMPL:	14

(7) Flowchart


BFMUL
RCLR1 Multiplication result storage area Clears 8 bytes to 0
Clears sign flag SFLAG←0
Multiplicand < 0
No
Yes
COMPL Multiplicand ←|Multiplicand|
Inverts sign flag SFLAG←$\overline{\text{SFLAG}}$
BFMUL1
Multiplier < 0
No
Yes
COMPL Multiplier ←|Multiplier|
Inverts sign flag SFLAG←$\overline{\text{SFLAG}}$
BFMUL2
HL←First address of multiplication result storage area (BRSLT)
Multiplicand pointer←0 DE←0
Loop counter 1←0 B←0
Loop counter 2←0 C←0
1
1
BFMUL3
X←[BMLCND+B] Contents of [highest address of multiplier + B]
A←[MBLIER+DE] Contents of [lowest address of multiplicand + DE]
AX←A×X
Addition to result area 1 [HL]←[HL]+X HL←HL+1
BFMUL4
Addition to result area 2 [HL]←[HL]+A+CY, HL←HL+1
Carry CY=1
No
Yes
HL=BRSLT+8
Yes
No
As carry, A←0 CY←1
BFMUL5
B←B + 1. The following pointer value is set in HL: HL←BRSLT+B+C
End of 4 digits of multiplication B = 4
No
Yes
B←0 DE←DE+1 C←C+1
The following pointer value is set in HL: HL←BRSLT+C
End of 4 digits of multiplication C = 4
No
Yes
2
2
Sign flag = 1
No
Yes
COMP1 Result ←2s complement of result
RET


COMPL
C ← Number of bytes converted into absolute value
COMP1
Saves HL B←C
COMP2
Inverts contents of memory indicated by HL HL ← HL + 1
C←C−1
C=0
No
Yes
CY←1 Restores HL
COMP3
[HL]←[HL]+CY HL←HL+1
B←B−1
B=0
No
Yes
RET


RCLR
C ← 4 Number of bytes to be cleared to 0
RCLR1
A←0 (Clear data)
RCLR2
[DE]←A DE←DE+1
C←C−1
C=0
No
Yes
RET

(8) Program list

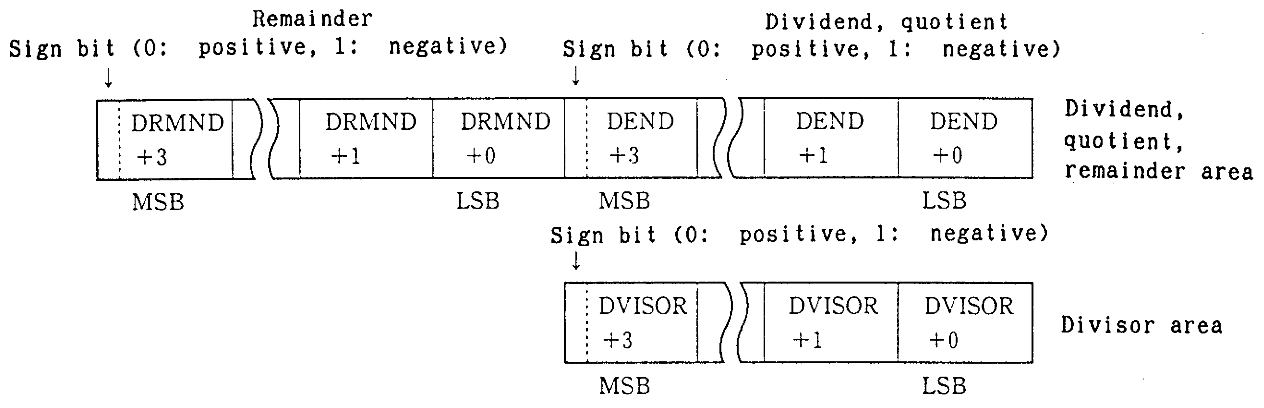
```
        NAME    BFMULR

;***********************************************************
;*      binary multiplication                              *
;*         input condition                                 *
;*              multiplicand <- (BMLCND+3,...,BMLCND)      *
;*              multiplier   <- (BMLIER+3,...,BMLIER)      *
;*         output condition                                *
;*              result <- (BRSLT+7,BRSLT+6...,BRSLT)       *
;***********************************************************

        PUBLIC  BFMUL
        EXTRN   BMLCND,BMLIER,BRSLT
        EXTRN   RCLR,RCLR1,COMPL,COMP1
        EXTBIT  SFLAG               ; sign-flag
;
BYTNUM  EQU     4                   ; value length
;
        CSEG
BFMUL:
;
;       *** result area 0-clear ***
;
        MOV     C,#8                ; set area length
        MOVW    DE,#BRSLT           ; HL-reg. <- BRSLT
        CALL    !RCLR1              ; clear subroutine
;
;       *** compliment convert ***
;
        CLR1    SFLAG               ; sign-flag <- 0
        MOVW    HL,#BMLCND          ; HL-reg. <- BMLCND
        MOV     A,[HL+3]            ; check sign
        BF      A.7,$BFMUL1         ; if data<0 goto BFMUL1
        CALL    !COMPL              ; complement subroutine
        NOT1    SFLAG               ; not sign-flag
BFMUL1:
        MOVW    HL,#BMLIER          ; HL-reg. <- BMLIER
        MOV     A,[HL+3]
        BF      A.7,$BFMUL2
        CALL    !COMPL              ; complement subroutine
        NOT1    SFLAG               ; not sign-flag
;
;       *** set multiplication counter ***
;
BFMUL2:
        MOVW    HL,#BRSLT           ; result top address
        MOVW    DE,#0               ; sub pointer_1
        MOVW    BC,#0               ; loop counter_1,2
;
;       *** binary multiplication process ***
;
BFMUL3:
        MOV     A,BMLCND[B]         ; read base value
        MOV     X,A
        MOV     A,BMLIER[DE]        ; read multiplier
        MULU    X
        XCH     A,X
        ADD     A,[HL]              ; write result
        MOV     [HL+],A
        MOV     A,X
```

```
BFMUL4:
        ADDC    A,[HL]
        MOV     [HL+],A
        BNC     $BFMUL5
        MOVW    AX,HL           ; check end of value
        CMPW    AX,#BRSLT+8
        BZ      $BFMUL5
        MOV     A,#0            ; multiplication carry
        SET1    CY
        BR      BFMUL4
BFMUL5:
        INC     B               ; increment loop counter_1
        MOV     X,B             ; next result pointer
        ADD     X,C
        MOV     A,#0
        ADDW    AX,#BRSLT
        MOVW    HL,AX
        MOV     A,#BYTNUM       ; check loop counter 1
        CMP     B,A
        BNZ     $BFMUL3

        MOV     B,#0
        INCW    DE              ; increment sub pointer
        INC     C               ; increment loop counter_2
        MOV     X,C             ; next result pointer
        MOV     A,#0
        ADDW    AX,#BRSLT
        MOVW    HL,AX
        MOV     A,#BYTNUM       ; check loop counter_2
        CMP     C,A
        BNZ     $BFMUL3

        BT      SFLAG,$BFMUL6   ; if sflag=1 complement convert
        RET
BFMUL6:
        MOV     C,#8
        MOVW    HL,#BRSLT
        CALL    !COMP1
        RET

        END
        NAME    CLR

;****************************************************
;*      0-clear process                            *
;*          input condition                        *
;*              DE-register <- 0-clear start address *
;*                                                 *
;****************************************************

        PUBLIC  RCLR,RCLR1,RCLR2
        PUBLIC  COMPL,COMP1
;
BYTNUM  EQU     4
;
        CSEG
RCLR:
        MOV     C,#BYTNUM       ; C-register <- 4
RCLR1:
        MOV     A,#0            ; Acc <- 0
RCLR2:
        MOV     [DE+],A
        DBNZ    C,$RCLR2
        RET
;
;*******************************************************
;*      complement convert subroutine                  *
;*              input  condition                       *
;*                HL-register <- complement top.address *
;*              output condition                       *
;*                (HL+3,HL+2,...,HL) <- convert data   *
;*                                                     *
;*******************************************************

        CSEG
COMPL:
        MOV     C,#BYTNUM
COMP1:
        PUSH    HL              ; save HL-register
        MOV     B,C
COMP2:
        MOV     A,#0FFH
        XOR     A,[HL]
        MOV     [HL+],A
        DBNZ    C,$COMP2

        SET1    CY
        POP     HL
COMP3:
        MOV     A,#0
        ADDC    A,[HL]
        MOV     [HL+],A
        DBNZ    B,$COMP3

        RET
;
        END
```

2.1.4 Binary division

$$\begin{array}{ll} \text{Quotient} & \boxed{\text{32 bits}} \leftarrow \boxed{\text{32 bits}} \div \boxed{\text{32 bits}} \\ \text{Remainder} & \boxed{\text{32 bits}} \end{array}$$

(1) Memory


Remainder
Sign bit (0: positive, 1: negative)
Dividend, quotient
Sign bit (0: positive, 1: negative)
DRMND +3
DRMND +1
DRMND +0
DEND +3
DEND +1
DEND +0
MSB
LSB
MSB
LSB
Dividend, quotient, remainder area
Sign bit (0: positive, 1: negative)
DVISOR +3
DVISOR +1
DVISOR +0
MSB
LSB
Divisor area

(2) Registers

X, A, C, B, DE, HL

(3) Input conditions

As indicated in (1), the 32-bit dividend and divisor are respectively stored in the following areas:

. Dividend: DEND, DEND + 1, DEND + 2, DEND + 3
. Divisor : DVISOR, DVISOR + 1, DVISOR + 2, DVISOR + 3

(4) Output conditions

The quotient and remainder are stored in the following areas:

- The quotient is stored in the quotient areas indicated in (1):

  DEND, DEND + 1, DEND + 2, DEND + 3
- The remainder is stored in the remainder areas indicated in (1):

  DRMND, DRMND + 1, DRMND + 2, DRMND + 3

(5) Processing sequence

In the division program shown in this section, the dividend (DEND to DEND + 3) and remainder (DRMND to DRMND + 3) are stored in an area consisting of contiguous 8 bytes. One digit (4 bits) of the dividend and remainder is shifted to the left to transfer the highest digit of the dividend to the lowest area of the remainder. Consequently, the quotient, starting from the highest digit, is stored to the lowest area of the dividend on a digit-by-digit basis.
The number of digits of the quotient is the same as the number of times the divisor is subtracted from the remainder until the subtraction result becomes negative.
The processing is indicated in the following sequence:

(a) The divisor is checked to see whether it is 0; if the divisor is 0, execution branches to an error processing routine.

(b) The remainder area is cleared to 0.

(c) The absolute values for the dividend and divisor are computed. Bit 0 of register X serves as a quotient sign flag, while bit 1 of register X is used as a remainder sign flag. If either of the dividend and divisor is negative, the quotient sign flag is set to 1.
If the dividend is negative, the remainder sign flag is set to 1.

(d) As the number of bytes for the dividend, 8 is set in register B.

(e) The quotient and remainder area, which consists of contiguous 8 bytes, is shifted 4 bits to the left.

(f) The divisor is subtracted from the remainder and the result is regarded as the real remainder. If the result is negative, however, execution jumps to (h).

(g) The quotient (DEND) contents are incremented, and execution jumps to (e).

(h) Because the divisor has been subtracted from the remainder too many times, the divisor is added back to the remainder to obtain the real remainder.

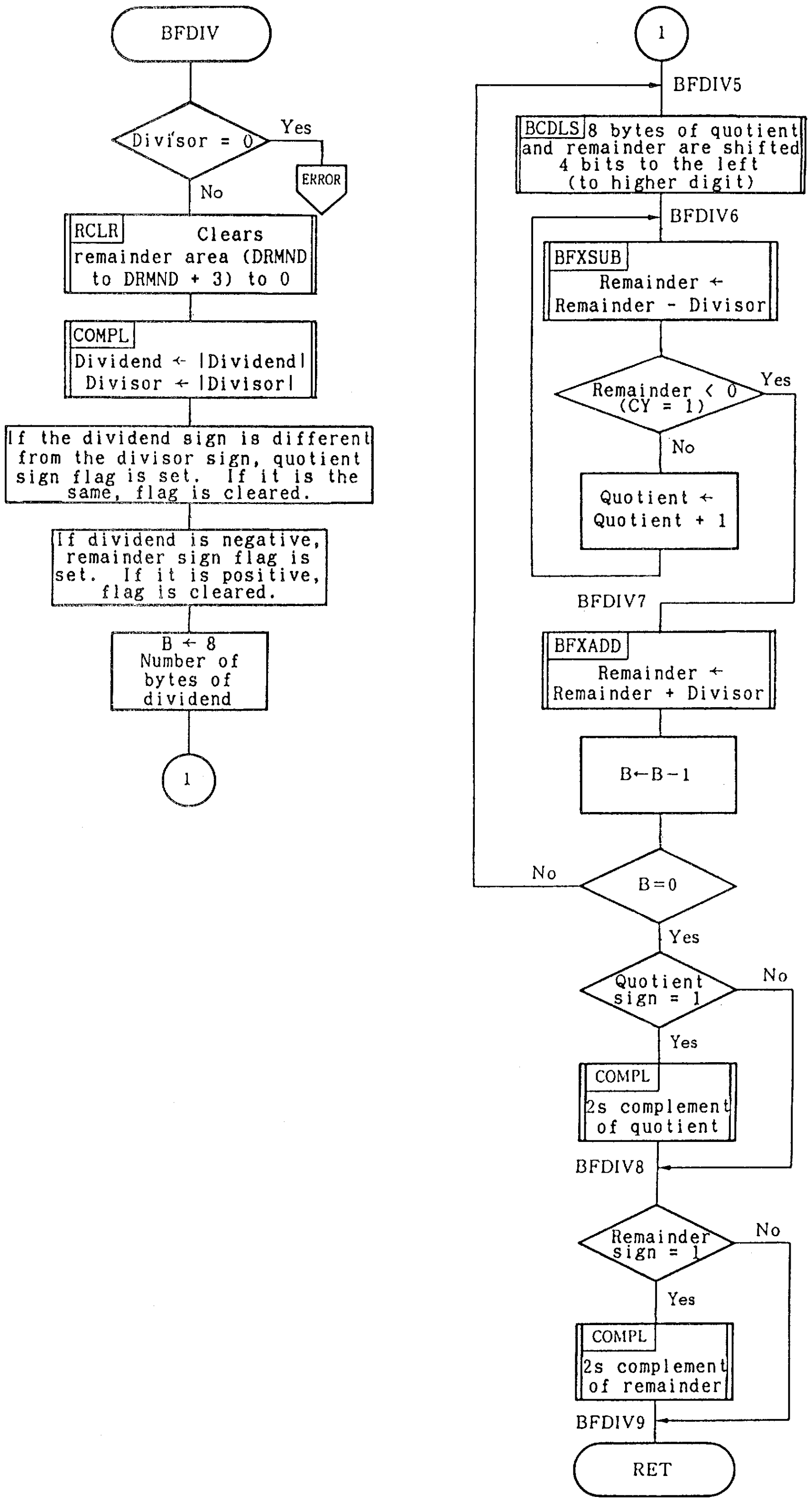
(i) The register B contents are decremented. Steps (e) to (h) above are repeated until the register contents are decremented to 0.

(j) The quotient sign flag is checked. If it is found to be set to 1, the 2s complement for the quotient is calculated.
The remainder sign flag is checked, and if it is found to be set to 1, the 2s complement of the remainder is calculated.

(6) Number of steps

48

(7) Flowchart


BFDIV
Divisor = 0
Yes
ERROR
No
RCLR Clears remainder area (DRMND to DRMND + 3) to 0
COMPL Dividend ← |Dividend| Divisor ← |Divisor|
If the dividend sign is different from the divisor sign, quotient sign flag is set. If it is the same, flag is cleared.
If dividend is negative, remainder sign flag is set. If it is positive, flag is cleared.
B ← 8 Number of bytes of dividend
1
1
BFDIV5
BCDLS 8 bytes of quotient and remainder are shifted 4 bits to the left (to higher digit)
BFDIV6
BFXSUB Remainder ← Remainder - Divisor
Remainder < 0 (CY = 1)
Yes
No
Quotient ← Quotient + 1
BFDIV7
BFXADD Remainder ← Remainder + Divisor
B←B−1
No
B=0
Yes
Quotient sign = 1
No
Yes
COMPL 2s complement of quotient
BFDIV8
Remainder sign = 1
No
Yes
COMPL 2s complement of remainder
BFDIV9
RET

(8) Program list

```
        NAME    BFDIVR

;*************************************************
;*     binary division                           *
;*          32 bit <- 32 bit / 32 bit            *
;*      input condition                          *
;*            dividend <- (DEND+3,...,DEND)      *
;*            divisor  <- (DVISOR+3,...,DVISOR)  *
;*      output condition                         *
;*            quotient  <- (DEND+3,...,DEND)     *
;*            remainder <- (DRMND+3,...,DRMND)   *
;*************************************************
        PUBLIC  BFDIV
        EXTRN   BFXADD,BFXSUB
        EXTRN   RCLR,COMPL,BCDLS,ERROR
        EXTRN   DEND,DVISOR,DRMND
;
SF_REM  EQU     X.0
SF_QUO  EQU     X.1
BYTNUM  EQU     4
;
        CSEG
BFDIV:
;
;       ****  check / divisor = 0 ?  ****
;
        MOV     C,#BYTNUM       ; C-register <- 4
        MOV     A,#0            ; Acc <- 0
        MOVW    HL,#DVISOR      ; HL <- DVISOR
BFDIV1:
        CMP     A,[HL+]
        BNZ     $BFDIV2         ; [HL] = 0 ?
        DBNZ    C,$BFDIV1
;
;       ****  divisor = 0  ****
;
        BR      ERROR           ; OVER FLOW
;
;       ****  quotient 0-clear  ****
;
BFDIV2:
        MOVW    DE,#DRMND       ; DE-register <- DRMND
        CALL    !RCLR
;
;       ****  complement convert  ****
;
        CLR1    SF_REM          ; clear remainder sign-flag
        CLR1    SF_QUO          ; clear quotient sign-flag
        MOVW    HL,#DEND        ; HL-register <- DEND
        MOV     A,[HL+3]
        BF      A.7,$BFDIV3
        CALL    !COMPL          ; complement subroutine
        SET1    SF_REM          ; set remainder sign-flag
        NOT1    SF_QUO          ; not quotient sign-flag
```

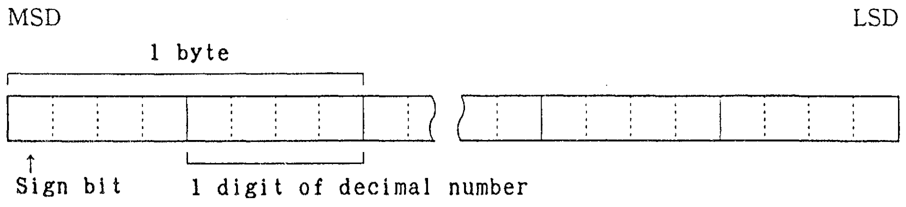
```
BFDIV3:
        MOVW    HL,#DVISOR      ; HL-register <- DVISOR
        MOV     A,[HL+3]
        BF      A.7,$BFDIV4
        CALL    !COMPL          ; complement subroutine
        NOT1    SF_QUO          ; not quotient sign-flag
;
;       ****  byte counter set  ****
;
BFDIV4:
        MOV     B,#8            ; B-register <- 8
;
;       ****  dividend,remainder 1-byte left shift  ****
;
BFDIV5:
        MOVW    HL,#DEND        ; HL <- DEND
        MOV     C,#8            ; C-register <- 8
        CALL    !BCDLS
;
;       ****  subtract divisor from dividend  ****
;
BFDIV6:
        MOVW    DE,#DVISOR      ; DE <- DVISOR
        MOVW    HL,#DRMND       ; HL <- DRMND
        CALL    !BFXSUB

        DECW    HL              ; decrement HL
        MOV     A,[HL]
        BT      A.7,$BFDIV7     ; if borrow

        MOV     A,#1            ; Acc <- 1
        MOVW    HL,#DEND
        ADD     A,[HL]          ; increment DEND
        MOV     [HL],A

        BR      BFDIV6
;
;       ****  if borrow divisor + dividend  ****
;
BFDIV7:
        MOVW    DE,#DVISOR      ; DE <- DVISOR
        MOVW    HL,#DRMND       ; HL <- DRMND
        CALL    !BFXADD
```

```
;
;       ****  check / division end ?  ****
;
        DBNZ    B,$BFDIV5

        BF      SF_REM,$BFDIV8
        MOVW    HL,#DRMND
        CALL    !COMPL
BFDIV8:
        BF      SF_QUO,$BFDIV9
        MOVW    HL,#DEND
        CALL    !COMPL
BFDIV9:
        RET
;
        END
```

2.2 Decimal Operations

The most significant bit is a sign bit with a numeric value represented by the other bits. Decimal numbers are represented as BCD codes.

Fig. 2-2 Decimal Representation

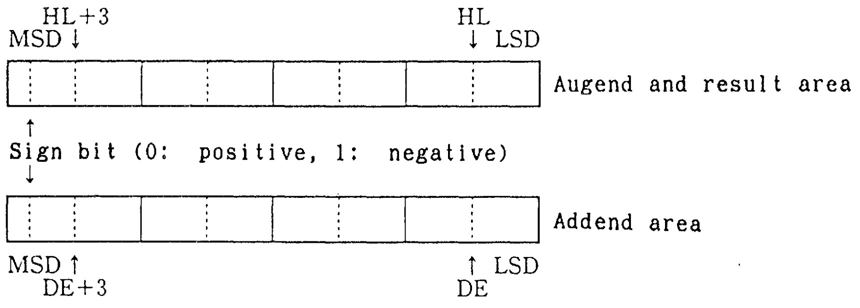


In decimal operations, numeric values are located in the memory, both before and after the operations.

2.2.1 Decimal addition

$\boxed{\text{8 digits}} \leftarrow \boxed{\text{8 digits}} + \boxed{\text{8 digits}}$

(1) Memory



(2) Registers

A, C, B, DE, HL

(3) Input conditions

As indicated in (1), the contents of registers HL and DE are set as follows.

. The lowest address in the memory area, where the 8-digit (4-byte) augend is stored, should be loaded to register HL.
. The lowest address in the area, where 8-digit (4-byte) addend is stored, should be loaded to register DE.

(4) Output conditions

The operation result is stored in the result area indicated in (1). However, the contents of HL register is damaged. If an overflow or underflow occurs, execution branches to an error processing routine.

Note: The numeric values that can be operated must be in a -79999999 to 79999999 range.

(5) Processing sequence

This addition program executes addition, if both the augend and addend have the same sign. If the augend sign differs from the addend sign, the program executes subtraction.
The processing is indicated in the following sequence:

(a) The number of bytes for decimal addition is set in register C.
The contents of this register minus 1 (C - 1) are set in register B, as the number of bytes for decimal addition without sign.
(b) If the signs of the augend and addend differ from each other, execution jumps to (o).
(c) The carry flag and sign flag are cleared to 0.
(d) A 1 byte of augend, indicated by the augend address, is read into register A.

(e) A 1 byte of addend, indicated by the addend address, is added with carry to the register A contents, and the addend address is incremented. The addition result is adjusted for decimal and is stored in a memory location indicated by the augend address. After that, the augend address is incremented.

(f) The register B contents are decremented. Steps (d) to (e) above are repeated until the register contents are decremented to 0.

(g) A 1 byte of augend, indicated by the augend address, is read into register A.

(h) A 1 byte of addend, indicated by the addend address, is added with carry to the register A contents.

(i) If the carry flag is 0, execution jumps to (k).

(j) The sign flag is set to 1 and the carry flag is cleared to 0.

(k) The register A contents are adjusted for decimal.

(l) If an overflow has occurred, the carry flag or bit 7 for register A is set to 1. If either of these bits is 1, execution jumps to an error processing routine.

(m) If the sign flag is 1, bit 7 of register A is set to 1.

(n) The register A contents are stored in a memory location indicated by the augend address. This completes the operation.

(o) The subtrahend is made positive and the sign flag is cleared to 0.

(p) If the minuend is negative, it is made positive and the sign flag is set to 1.

(q) The carry flag is cleared to 0.

(r) A 1 byte of minuend, indicated by the minuend address, is read to register A.

(s) A 1 byte of subtrahend, indicated by the subtrahend address, is subtracted with carry from the register A contents and the subtrahend address is subsequently incremented. The subtraction result is adjusted for decimal, and is stored in a memory location indicated by the minuend address. After that, the minuend address is incremented.

(t) The register C contents are decremented. Steps (r) to (s) above are repeated until the register contents are decremented to 0.

(u) If the carry flag is 0, execution proceeds to (w).

(v) The 10s complement of the result is computed and the sign flag is inverted.

(w) If the result is 0, the operation ends.

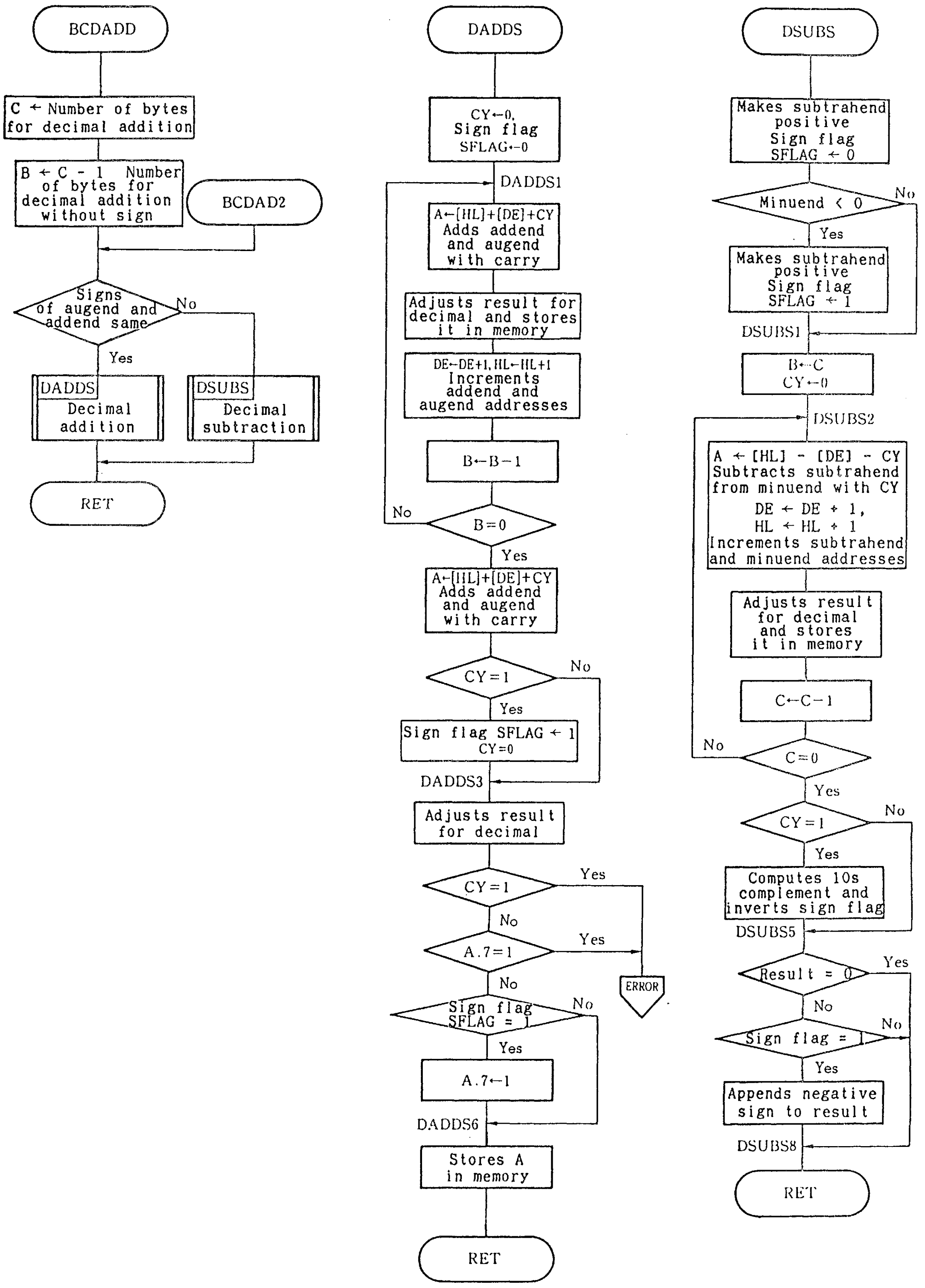
(x) If the sign flag is 1, processing (y) is started. If the flag is 0, the operation ends.

(y) The result sign bit is set to 1, and the operation ends.

(6) Number of steps

83

(7) Flowchart


BCDADD
C ← Number of bytes for decimal addition
B ← C - 1 Number of bytes for decimal addition without sign
BCDAD2
Signs of augend and addend same
No
Yes
DADDS Decimal addition
DSUBS Decimal subtraction
RET
DADDS
CY←0, Sign flag SFLAG←0
DADDS1
A←[HL]+[DE]+CY Adds addend and augend with carry
Adjusts result for decimal and stores it in memory
DE←DE+1, HL←HL+1 Increments addend and augend addresses
B←B−1
B=0
No
Yes
A←[HL]+[DE]+CY Adds addend and augend with carry
CY=1
No
Yes
Sign flag SFLAG ← 1 CY=0
DADDS3
Adjusts result for decimal
CY=1
Yes
No
A.7=1
Yes
No
ERROR
Sign flag SFLAG = 1
No
Yes
A.7←1
DADDS6
Stores A in memory
RET
DSUBS
Makes subtrahend positive Sign flag SFLAG ← 0
Minuend < 0
No
Yes
Makes subtrahend positive Sign flag SFLAG ← 1
DSUBS1
B←C CY←0
DSUBS2
A ← [HL] - [DE] - CY Subtracts subtrahend from minuend with CY
DE ← DE + 1, HL ← HL + 1
Increments subtrahend and minuend addresses
Adjusts result for decimal and stores it in memory
C←C−1
C=0
No
Yes
CY=1
No
Yes
Computes 10s complement and inverts sign flag
DSUBS5
Result = 0
Yes
No
Sign flag = 1
No
Yes
Appends negative sign to result
DSUBS8
RET

(8) Program list

```
        NAME    BCDADR

;*****************************************************
;*      decimal addition                            *
;*           8 digit <- 8 digit + 8 digit           *
;*       input condition                            *
;*            HL-register <- augend area top.address  *
;*            DE-register <- addend  area top.address *
;*       output condition                           *
;*            result <- (HL,HL+1,HL+2,HL+3)         *
;*****************************************************

        PUBLIC  BCDADD,BCDAD1,BCDAD2
        PUBLIC  DADDS
        PUBLIC  DSUBS
        EXTRN   ERROR
        EXTBIT  SFLAG               ; work flag for sign flag
;
BYTNUM  EQU     4
;
        CSEG
BCDADD:
        MOV     C,#BYTNUM           ; C-register <- 4
BCDAD1:
        MOV     B,C                 ; B-register <- C-register - 1
        DEC     B
BCDAD2:
        MOV     A,[HL+BYTNUM-1]
        XOR     A,[DE+BYTNUM-1]

        BT      A.7,$BCDAD3
        CALL    !DADDS
        RET
BCDAD3:
        CALL    !DSUBS
        RET
;================================================
;       ***** decimal addition subroutine *****
;================================================

DADDS:
        CLR1    CY
        CLR1    SFLAG               ; clear sign-flag
DADDS1:
        MOV     A,[HL]
        ADDC    A,[DE+]
        ADJBA                       ; decimal adjust
        MOV     [HL+],A
        DBNZ    B,$DADDS1

        MOV     A,[HL]
        ADDC    A,[DE]
DADDS2:
        BNC     $DADDS3
        SET1    SFLAG               ; set sign-flag
        CLR1    CY
DADDS3:
        ADJBA                       ; decimal adjust
        BNC     $DADDS4
        BR      ERROR
```

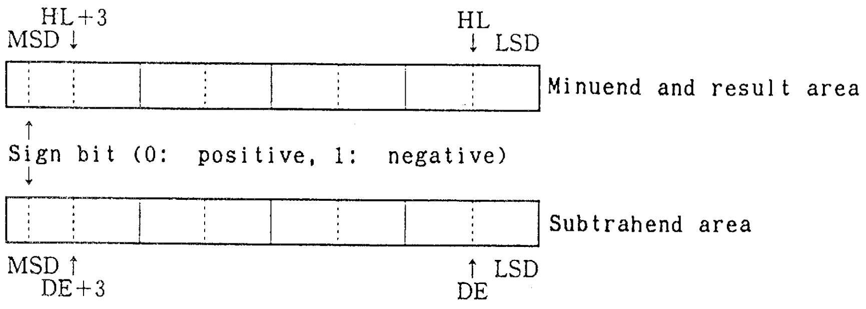
```
DADDS4:
        BF      A.7,$DADDS5
        BR      ERROR
DADDS5:
        BF      SFLAG,$DADDS6
        SET1    A.7
DADDS6:
        MOV     [HL],A
        RET

;=====================================================
;       ***** decimal subtraction subroutine *****
;=====================================================

DSUBS:
        PUSH    HL                  ; save HL-register
        CLR1    SFLAG               ; clear sign-flag
        MOV     A,[DE+BYTNUM-1]
        CLR1    A.7
        MOV     [DE+BYTNUM-1],A
        MOV     A,[HL+BYTNUM-1]
        BF      A.7,$DSUBS1

        CLR1    A.7
        MOV     [HL+BYTNUM-1],A
        SET1    SFLAG               ; set sign-flag
DSUBS1:
        MOV     B,C                 ; save C-register
        CLR1    CY
DSUBS2:
        MOV     A,[HL]
        SUBC    A,[DE+]
        ADJBS                       ; decimal adjust
        MOV     [HL+],A
        DBNZ    C,$DSUBS2

        BNC     $DSUBS5
        POP     HL                  ; load HL-register
        PUSH    HL                  ; save HL-register
        MOV     C,B                 ; load C-register
DSUBS3:
        MOV     A,#99H              ; (HL) <- 9 - (HL)
        SUB     A,[HL]              ;     increment HL-register
        ADJBS                       ; decimal adjust
        MOV     [HL+],A
        DBNZ    C,$DSUBS3

        POP     HL                  ; load HL-register
        PUSH    HL                  ; save HL-register
        SET1    CY

        MOV     C,B                 ; load C-register
DSUBS4:
        MOV     A,#0                ; Acc <- 0
        ADDC    A,[HL]
        ADJBA                       ; decimal adjust
        MOV     [HL+],A
        DBNZ    C,$DSUBS4
        NOT1    SFLAG
```

```
;
;       ****  check / result = 0  ****
;
DSUBS5:
        MOV     C,B             ; load C-register
        POP     HL              ; load HL-register
        PUSH    HL              ; save HL-register
        MOV     A,#0
DSUBS6:
        CMP     A,[HL+]
        BNZ     $DSUBS7
        DBNZ    C,$DSUBS6
        POP     HL              ; load HL-register
        RET
DSUBS7:
        POP     HL              ; load HL-register
        BF      SFLAG,$DSUBS8
        MOV     A,[HL+BYTNUM-1]
        SET1    A.7             ; sign set
        MOV     [HL+BYTNUM-1],A
DSUBS8:
        RET
;
        END
```

2.2.2 Decimal subtraction

$$\boxed{\text{8 digits}} \leftarrow \boxed{\text{8 digits}} - \boxed{\text{8 digits}}$$

(1) Memory



(2) Registers

A, C, B, DE, HL

(3) Input conditions

As indicated in (1), the contents of registers HL and DE are set as follows.

. The lowest address in the memory area, where the 8-digit (4-byte) minuend is stored, is loaded to register HL.

. The lowest address in the memory area, where the 8-digit (4-byte) subtrahend is stored, is loaded to register DE.

(4) Output conditions

The operation result is stored in the result area indicated in (1).

If an overflow or underflow occurs, however, execution branches to an error processing routine.

Note: The numeric values that can be operated must be in a -79999999 to 79999999 range.

(5) Processing sequence

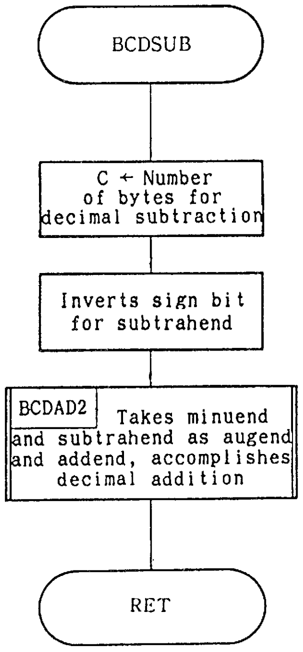
The subtraction program presented in this section converts the processing "minuend - subtrahend" into the processing "minuend + (-subtrahend)".
The processing is executed in the following sequence:

(a) The number of bytes for decimal subtraction is set in register C.
(b) The sign bit for the subtrahend is inverted.
(c) Taking the minuend and subtrahend, respectively, as augend and addend, decimal addition is accomplished.

(6) Number of steps

8

(7) Flowchart


BCDSUB
C ← Number of bytes for decimal subtraction
Inverts sign bit for subtrahend
BCDAD2
Takes minuend and subtrahend as augend and addend, accomplishes decimal addition
RET

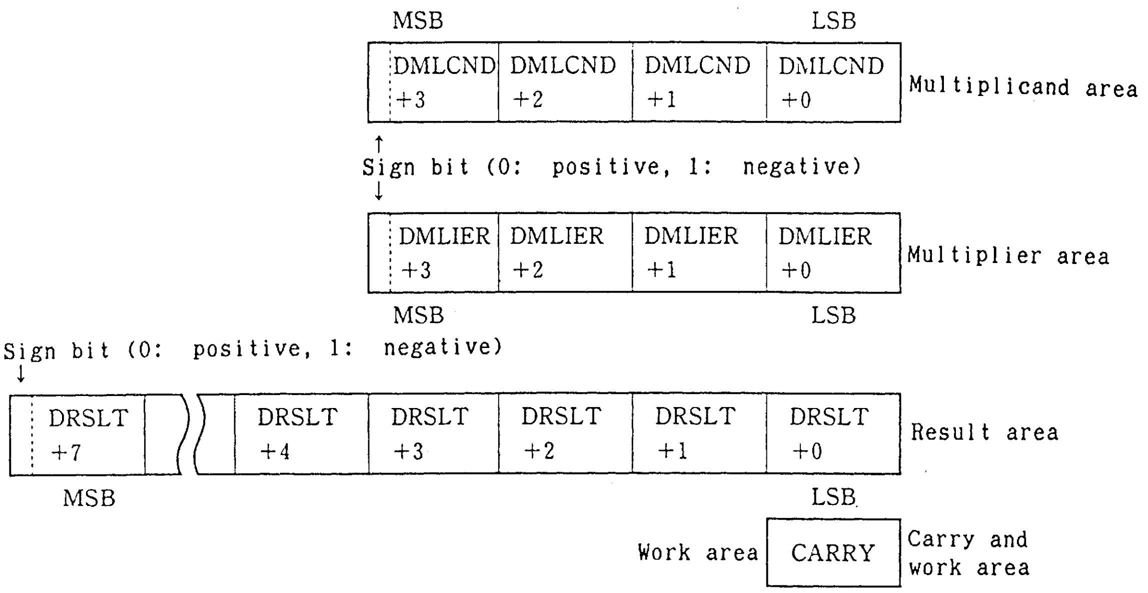
(8) Program list

```
        NAME    BCDSUR

;**********************************************************
;*      decimal subtraction                               *
;*          8 digit <- 8 digit - 8 digit                  *
;*       input condition                                  *
;*            HL-register <- minus value area top.address *
;*            DE-register <- subtrahend area top.address  *
;*       output condition                                 *
;*            result <- (HL,HL+1,HL+2,HL+3)               *
;**********************************************************

        PUBLIC  BYTNUM
        PUBLIC  BCDSUB
        EXTRN   BCDADD,BCDAD2
;
BYTNUM  EQU     4
;
        CSEG
BCDSUB:
        MOV     C,#BYTNUM          ; C-register <- 4
BCDSU1:
        MOV     B,C                ; B-register <- C-register - 1
        DEC     B

        MOV     A,[DE+BYTNUM-1]
        NOT1    A.7
        MOV     [DE+BYTNUM-1],A
        CALL    !BCDAD2
        RET
;
        END
```

2.2.3 Decimal multiplication

$$\boxed{\text{16 digits}} \leftarrow \boxed{\text{8 digits}} \times \boxed{\text{8 digits}}$$

(1) Memory



(2) Registers

X, A, C, B, DE, HL

(3) Input conditions

As indicated in (1), the 8-digit multiplicand and multiplier are respectively stored in the following areas:

- Multiplicand: DMLCND, DMLCND+1, DMLCND+2, DMLCND+3
- Multiplier : DMLIER, DMLIER+1, DMLIER+2, DMLIER+3

(4) Output conditions

The operation result is stored in the result areas (DRSLT, DRSLT+1, ..., DRSLT+7), as indicated in (1).

Note: 1. The multiplicand and multiplier must be in a -79999999 to 79999999 range.
2. The operation range is -6399999840000001 to 6399999840000001.

(5) Processing sequence

This multiplication program shifts the multiplier 1 digit (4 bits) to the right to load the digits to the addition counter, starting from the lowest digit. One digit is loaded to the counter at a time. Therefore, the processing "result result + multiplicand" is repeatedly executed.
When addition by the addition counter has ended, the result area is shifted 1 digit (4 bits) to the right, so that addition of a multiplier 1 digit higher than before can be accomplished.
The processing is carried out in the following sequence:

(a) The result area is cleared to 0.
(b) The absolute values for the multiplier and multiplicand are computed. If the multiplier sign is different from the multiplicand sign, the sign flag is set to 1. If the signs are the same, the sign flag is cleared to 0.
(c) 8 is set in the digit counter (register B).
(d) The multiplier is shifted 1 digit to the right, so that the lowest digit of the multiplier is loaded to the addition counter (register C).
(e) The carry area (CARRY) is cleared.
(f) If the contents of the addition counter are 0, execution branches to (1).

(g) The decimal addition "result (higher 8 digits) result (higher 8 digits) + multiplicand" is performed. If an overflow occurs as a result, the CARRY contents are incremented.

(h) The addition counter is decremented. Step (g) above is repeated until the counter contents are decremented to 0.

(i) The result area is shifted 1 digit (4 bits) to the right with CARRY, and the CARRY is stored in the highest digit of the result area.

(j) The digit counter is decremented. Steps (d) to (i) above are repeated until the counter contents are decremented to 0.

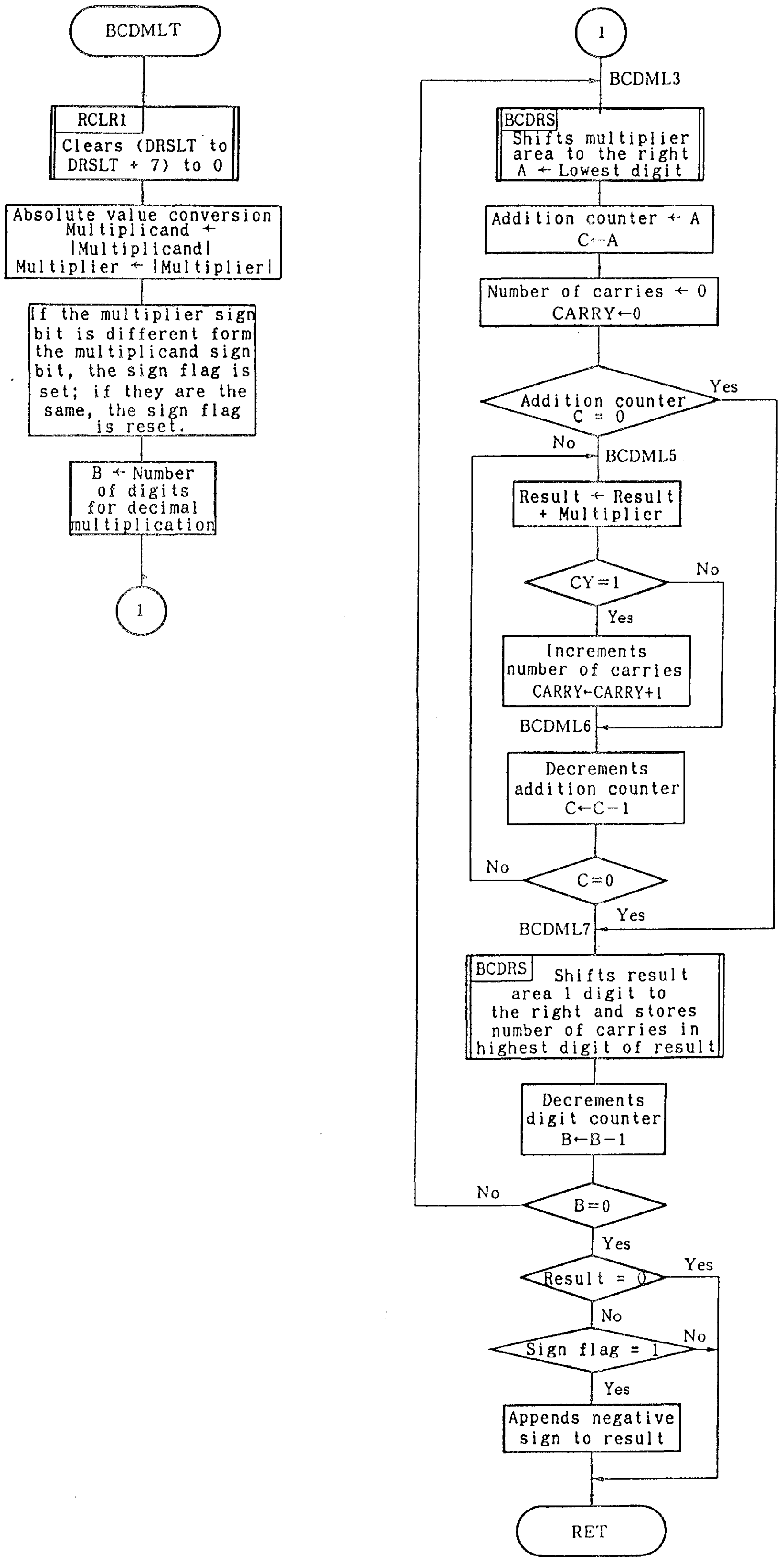
(k) If the result is 0, the operation ends.

(l) If the sign flag is 1, the sign bit for the result area is set to 1.

(6) Number of steps

58

(7) Flowchart



BCDMLT
RCLR1
Clears (DRSLT to DRSLT + 7) to 0
Absolute value conversion Multiplicand ← |Multiplicand| Multiplier ← |Multiplier|
If the multiplier sign bit is different form the multiplicand sign bit, the sign flag is set; if they are the same, the sign flag is reset.
B ← Number of digits for decimal multiplication
1
1
BCDML3
BCDRS
Shifts multiplier area to the right A ← Lowest digit
Addition counter ← A C←A
Number of carries ← 0 CARRY←0
Addition counter C = 0
Yes
No
BCDML5
Result ← Result + Multiplier
CY=1
No
Yes
Increments number of carries CARRY←CARRY+1
BCDML6
Decrements addition counter C←C−1
C=0
No
Yes
BCDML7
BCDRS
Shifts result area 1 digit to the right and stores number of carries in highest digit of result
Decrements digit counter B←B−1
B=0
No
Yes
Result = 0
Yes
No
Sign flag = 1
No
Yes
Appends negative sign to result
RET

(8) Program list

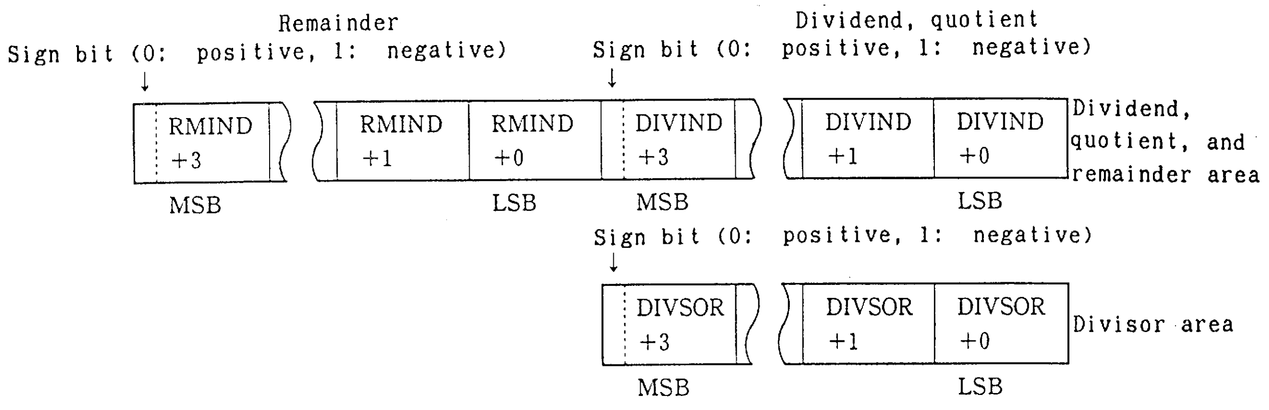
```
        NAME    BCDMLR

;*****************************************************
;*      decimal multiplication                       *
;*          16 digit <- 8 digit * 8 digit            *
;*       input condition                             *
;*           multiplicand <- (DMLCND+3,...,DMLCND)   *
;*           multiplier   <- (DMLIER+3,...,DMLIER)   *
;*       output condition                            *
;*           result <- (DRSLT+7,...,DRSLT)           *
;*****************************************************

        PUBLIC  BCDMLT
        EXTRN   RCLR1,BCDRS,BCDRS1
        EXTRN   DMLCND,DMLIER,DRSLT
        EXTRN   CARRY               ; 1-byte carry area
        EXTBIT  SFLAG               ; work flag for sign flag
;
        CSEG
BCDMLT:
;
;       ****  result area 0-clear  ****
;
        MOV     C,#8                ; C-register <- 8
        MOVW    DE,#DRSLT           ; DE <- DRSLT
        CALL    !RCLR1
;
;       ****  check / sign  ****
;
        MOVW    DE,#DMLCND+3
        MOVW    HL,#DMLIER+3
        CLR1    SFLAG               ; clear sign-flag
        MOV     A,[DE]
        BF      A.7,$BCDML1
        CLR1    A.7
        MOV     [DE],A
        NOT1    SFLAG               ; not sign-flag
BCDML1:
        MOV     A,[HL]
        BF      A.7,$BCDML2
        CLR1    A.7
        MOV     [HL],A
        NOT1    SFLAG               ; not sign-flag
;
;       ****  digit counter set  ****
;
BCDML2:
        MOV     B,#8                ; B-register <- 8
;
;       ****  multiplier right shift  ****
;
BCDML3:
        MOVW    HL,#DMLIER+3
        MOV     C,#4                ; C-register <- 4
        CALL    !BCDRS
        MOV     C,A                 ; C-register <- Acc
        MOV     CARRY,#0            ; carry <- 0
;
;       ****  check / multiplier = 0 ?  ****
;
        ADD     A,#0
        BZ      $BCDML7          ; if Acc = 0 then goto BCDML6
;
;       ****  result <- DMLCND + result  ****
;
BCDML4:
        MOVW    DE,#DMLCND       ; DE <- DMLCND
        MOVW    HL,#DRSLT+4      ; HL <- DRSLT+4
        CLR1    CY               ; clear carry
        PUSH    AX               ; save AX-register
        PUSH    BC               ; save BC-register
        MOV     C,#4             ; C-register <- 4
BCDML5:
        MOV     A,[HL]
        ADDC    A,[DE+]
        ADJBA                    ; decimal adjust
        MOV     [HL+],A
        DBNZ    C,$BCDML5
        POP     BC               ; load BC-register
        POP     AX               ; load AX-register
        BNC     $BCDML6
        INC     CARRY
BCDML6:
        DBNZ    C,$BCDML4
;
;       ****  result right shift with carry  ****
;
BCDML7:
        MOV     A,CARRY
        MOVW    HL,#DRSLT+7      ; HL <- DRSLT+7
        MOV     C,#8
        CALL    !BCDRS1
;
;       ****  check / multiply end ?  ****
;
        DBNZ    B,$BCDML3

;
;       ****  check / multiply = 0  ****
;
        MOVW    HL,#DRSLT
        MOV     C,#8
        MOV     A,#0
BCDML8:
        CMP     A,[HL+]
        BNZ     $BCDML9
        DBNZ    C,$BCDML8
        RET
;
;       ****  check / sign-flag  ****
;
BCDML9:
        BF      SFLAG,$BCDM10
        MOVW    HL,#DRSLT+7
        MOV     A,[HL]
        SET1    A.7
        MOV     [HL],A
BCDM10:
        RET
;
        END
```

2.2.4 Decimal division

Quotient $\boxed{\text{8 digits}} \leftarrow \boxed{\text{8 digits}} \div \boxed{\text{8 digits}}$

Remainder $\boxed{\text{8 digits}}$

(1) Memory



Remainder
Sign bit (0: positive, 1: negative)
RMIND +3
RMIND +1
RMIND +0
MSB
LSB
Dividend, quotient
Sign bit (0: positive, 1: negative)
DIVIND +3
DIVIND +1
DIVIND +0
MSB
LSB
Dividend, quotient, and remainder area
Sign bit (0: positive, 1: negative)
DIVSOR +3
DIVSOR +1
DIVSOR +0
MSB
LSB
Divisor area

(2) Registers

X, A, C,B, DE, HL

(3) Input conditions

As indicated in (1), the 8-digit dividend and divisor are stored in the following areas:

. Dividend: DIVIND, DIVIND+1, DIVIND+2, DIVIND+3
. Divisor : DIVSOR, DIVSOR+1, DIVSOR+2, DIVSOR+3

(4) Output conditions

The quotient and remainder are stored in the following areas:

. The quotient is stored in the quotient area indicated in (1):

DIVIND, DIVIND+1, DIVIND+2, DIVIND+3

. The remainder is stored in the remainder area indicated in (1):

RMIND, RMIND+1, RMIND+2, RMIND+3

(5) Processing sequence

In the division program shown in this section, the dividend (DIVIND to DIVIND+3) and remainder (RMIND to RMIND+3) are stored in an area consisting of contiguous 8 bytes. One digit (4 bits) of the dividend and remainder is shifted to the left to transfer the highest digit of the dividend to the lowest area of the remainder. Consequently, the quotient, starting from the highest digit, is stored to the lowest area of the dividend on a digit-by-digit basis.
The number of digits for the quotient is the same as the number of times that the divisor is subtracted from the remainder, until the subtraction result becomes negative.
The processing is performed in the following sequence:

(a) The divisor is checked to see whether it is 0. If the divisor is 0, execution branches to an error processing routine.

(b) The remainder area is cleared to 0.

(c) The absolute values for the dividend and divisor are computed. Bit 0 for register X serves as a quotient sign flag, while bit 1 for register X is used as a remainder sign flag. If either the dividend or divisor is negative, the quotient sign flag is set to 1.
If the dividend is negative, the remainder sign flag is set to 1.

(d) As the number of bytes for the dividend, 8 is set in register C.

(e) The quotient and remainder area, which consists of 8 contiguous bytes, is shifted 4 bits to the left.

(f) The divisor is subtracted from the remainder and the result is regarded as the real remainder. If the result is negative, however, execution jumps to (h).

(g) The quotient (DIVIND) contents are incremented, and execution jumps to (f).

(h) Because the divisor has been subtracted from the remainder too many times, the divisor is added back to the remainder to obtain the real remainder.

(i) The register C contents are decremented. Steps (e) to (h) above are repeated until the register contents are decremented to 0.

(j) If the quotient is 0, execution proceeds to (l).

(k) If the quotient sign flag is 1, the sign bit for the quotient area is set to 1.

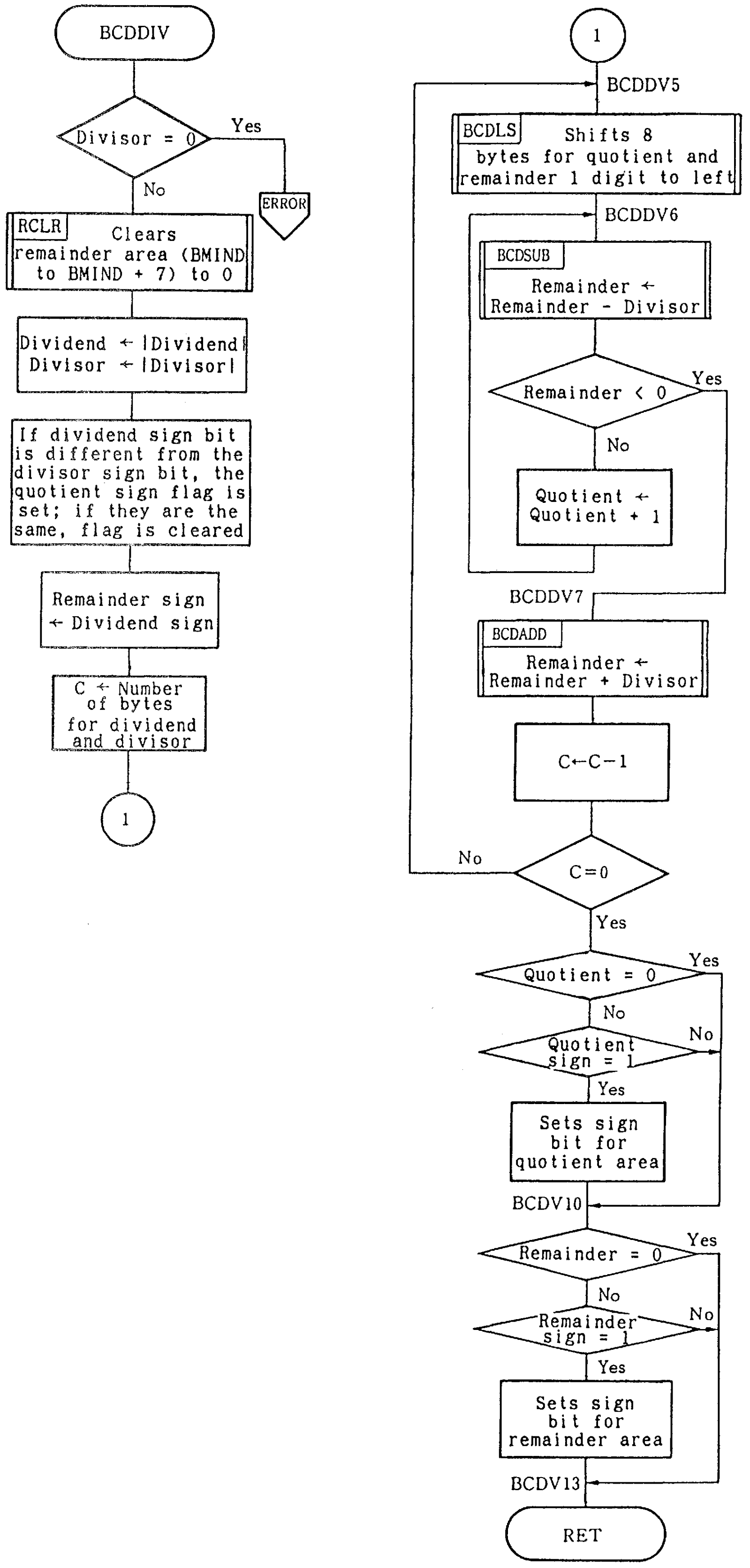
(l) If the remainder is 0, the operation ends.

(m) If the remainder sign flag is 1, the sign bit for the remainder area is set to 1.

(6) Number of steps

70

(7) Flowchart


BCDDIV
Divisor = 0
Yes
ERROR
No
RCLR
Clears remainder area (BMIND to BMIND + 7) to 0
Dividend ← |Dividend|
Divisor ← |Divisor|
If dividend sign bit is different from the divisor sign bit, the quotient sign flag is set; if they are the same, flag is cleared
Remainder sign ← Dividend sign
C ← Number of bytes for dividend and divisor
1
1
BCDDV5
BCDLS
Shifts 8 bytes for quotient and remainder 1 digit to left
BCDDV6
BCDSUB
Remainder ← Remainder - Divisor
Remainder < 0
Yes
No
Quotient ← Quotient + 1
BCDDV7
BCDADD
Remainder ← Remainder + Divisor
C←C−1
No
C=0
Yes
Quotient = 0
Yes
No
Quotient sign = 1
No
Yes
Sets sign bit for quotient area
BCDV10
Remainder = 0
Yes
No
Remainder sign = 1
No
Yes
Sets sign bit for remainder area
BCDV13
RET

(8) Program list

```
        NAME    BCDIVR

;**************************************************
;*      decimal division                          *
;*          8 digit <- 8 digit / 8 digit          *
;*       input condition                          *
;*            dividend <- (DIVIND+3,...,DIVIND)   *
;*            divisor  <- (DIVSOR+3,...,DIVSOR)   *
;*       output condition                         *
;*            quotient  <- (DIVIND+3,...,DIVIND)  *
;*            remainder <- (RMIND+3,...,RMIND)    *
;**************************************************

        PUBLIC  BCDDIV
        EXTRN   ERROR,RCLR,BCDLS,BCDSUB,BCDADD
        EXTRN   DIVIND,DIVSOR,RMIND
;
SF_QUO  EQU     X.0
SF_REM  EQU     X.1
;
        CSEG
BCDDIV:
;
;       ****  check / divisor = 0 ?  ****
;
        MOV     C,#4            ; C-register <- 4
        MOV     A,#0            ; Acc <- 0
        MOVW    HL,#DIVSOR      ; HL <- DIVSOR
BCDDV1:
        CMP     A,[HL+]
        BNZ     $BCDDV2         ; (HL) = 0 ?
        DBNZ    C,$BCDDV1

        BR      ERROR           ; OVER FLOW
;
;       ****  result,remind 0-clear  ****
;
BCDDV2:
        MOVW    DE,#RMIND       ; DE <- RMIND
        CALL    !RCLR
;
;       ****  check / sign  ****
;
        CLR1    SF_QUO          ; clear quotient sign-flag
        CLR1    SF_REM          ; clear remainder sign-flag
        MOVW    HL,#DIVIND+3
        MOV     A,[HL]
        BF      A.7,$BCDDV3
        CLR1    A.7
        MOV     [HL],A
        SET1    SF_REM          ; set remainder sign-flag
        NOT1    SF_QUO          ; not quotient sign-flag
BCDDV3:
        MOVW    HL,#DIVSOR+3
        MOV     A,[HL]
        BF      A.7,$BCDDV4
        CLR1    A.7
        MOV     [HL],A
        NOT1    SF_QUO
```

```
;
;       ****  digit counter set  ****
;
BCDDV4:
        MOV     C,#8
;
;       ****  quotient,remind left shift  ****
;
BCDDV5:
        PUSH    BC
        MOVW    HL,#DIVIND      ; HL <- DIVIND
        MOV     C,#16/2         ; C-register <- 8
        CALL    !BCDLS          ; N-digit data left shift
;
;       ****  subtract divisor from dividend  ****
;
BCDDV6:
        MOVW    DE,#DIVSOR      ; DE <- DIVSOR
        MOVW    HL,#RMIND       ; HL <- RMIND
        CALL    !BCDSUB         ; decimal subtraction

        MOVW    HL,#RMIND+3
        MOV     A,[HL]
        BT      A.7,$BCDDV7     ; if borrow then goto BCDDV7

        MOV     A,#1
        MOVW    HL,#DIVIND
        ADD     A,[HL]          ; increment (DIVIND)
        MOV     [HL],A

        BR      BCDDV6
;
;       ****  if borrow then divisor + dividend  ****
;
BCDDV7:
        MOVW    DE,#DIVSOR      ; DE <- DIVSOR
        MOVW    HL,#RMIND       ; HL <- RMIND
        CALL    !BCDADD         ; decimal addition
;
;       ****  check / division end ?  ****
;
        POP     BC
        DBNZ    C,$BCDDV5
;
;       ****  check / quotient = 0  ****
;
        MOVW    HL,#DIVIND
        MOV     A,#0
        MOV     C,#4
BCDDV8:
        CMP     A,[HL+]
        BNZ     $BCDDV9
        DBNZ    C,$BCDDV8
        BR      BCDV10
;
;       ****  check / quotient sign-flag  ****
;
BCDDV9:
        BF      SF_QUO,$BCDV10
        MOVW    HL,#DIVIND+3
        MOV     A,[HL]
        SET1    A.7
        MOV     [HL],A
```

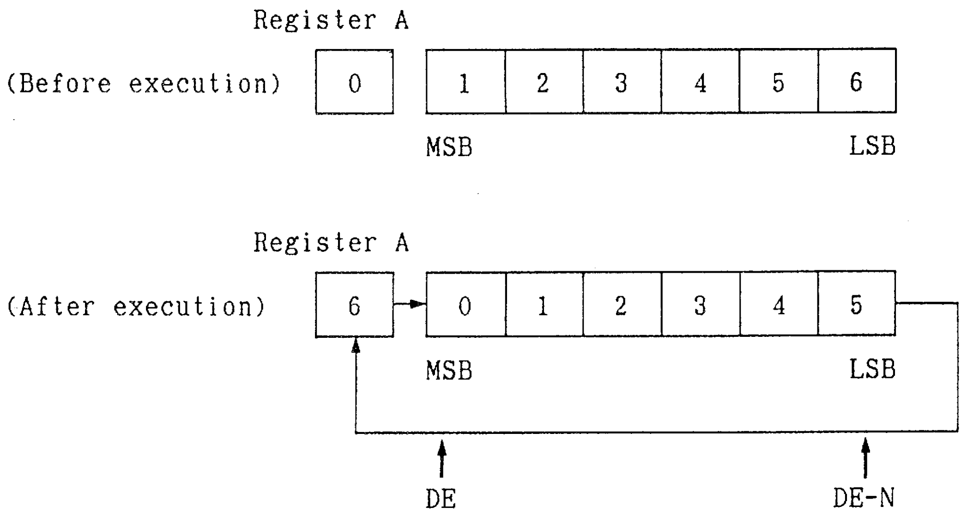
```
;
;       ****  check / remainder = 0  ****
;
BCDV10:
        MOVW    HL,#RMIND
        MOV     A,#0
        MOV     C,#4
BCDV11:
        CMP     A,[HL+]
        BNZ     $BCDV12
        DBNZ    C,$BCDV11
        RET
;
;       ****  check / remainder sign-flag  ****
;
BCDV12:
        BF      SF_REM,$BCDV13
        MOVW    HL,#RMIND+3
        MOV     A,[HL]
        SET1    A.7
        MOV     [HL],A
BCDV13:
        RET
;
        END
```

2.3 Shift Processing

The 78K/II is provided with shift instructions that shift the contents of general-purpose registers (X, A, C, B, E, D, L, and H) and general-purpose register pairs (AX, BC, DE, and HL) in units of 1 and 4 bits (ROR4, ROL4).
The program examples, presented in this section, show two shift operation examples, in units of bytes and 4 bits.

2.3.1 Shifting N-byte data to right

In this example, N-byte of data in the memory is shifted to the right.
After the example program presented in this section has been executed, a value set in advance in register A is stored in the highest byte and the contents of the lowest byte are output to register A.



(1) Registers

A, C, DE

(2) Input conditions

. A ← value to be transferred to highest address
. C ← number of bytes to be shifted (N)
. DE ← highest address of N-byte data

(3) Output conditions

The result of the 1-byte right shift is stored in the result area indicated in (1).
The register A contents are transferred to the highest address.
The contents of byte [DE - N] are transferred to register A.

(4) Number of steps

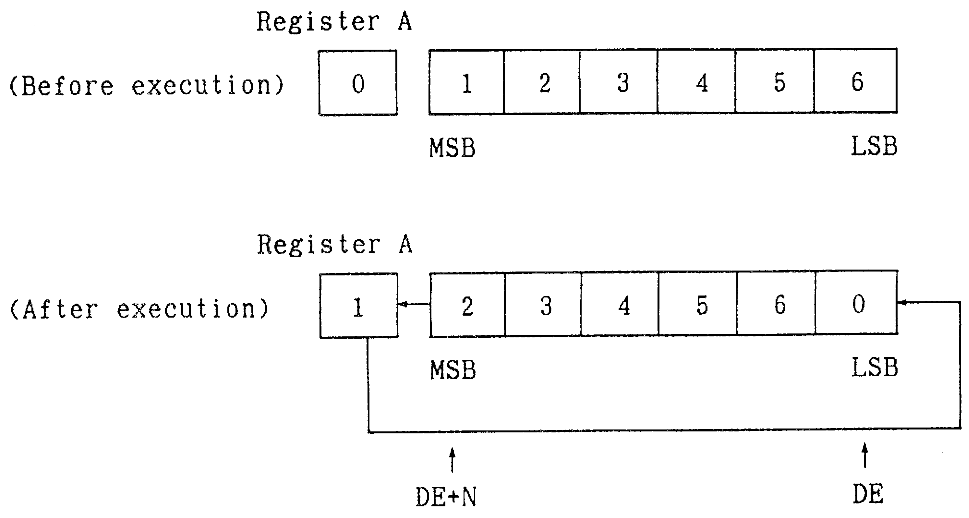
4

(5) Program list

```
        NAME    BYTRSR

;**************************************************
;*      N-byte data right shift                   *
;*          input condition                       *
;*              DE-register <- MSB of N-byte data *
;*              C -register <- byte counter       *
;*          output condition                      *
;*              Acc <- LSB of N-byte data         *
;**************************************************

        PUBLIC  BYTRST,BYTRS1
        PUBLIC  BYTLST,BYTLS1
;
        CSEG
BYTRST:
        MOV     A,#0            ; Acc <- 0
BYTRS1:
        XCH     A,[DE-]
        DBNZ    C,$BYTRS1
        RET
```

2.3.2 Shifting N-byte data to left

In this example, N-bytes of data in the memory are shifted to the left.
After the example program, presented in this section, has been executed, a value set in advance in register A is stored in the lowest byte and the contents of the highest byte are output to register A.



(1) Registers

A, C, DE

(2) Input conditions

. A ← value to be transferred to lowest address
. C ← number of bytes to be shifted (N)
. DE ← lowest address for N-byte data

(3) Output conditions

The 1-byte left shift result is stored in the result area indicated in (1).
The register A contents are transferred to the lowest address.
The contents of byte [DE + N] are transferred to register A.

(4) Number of steps

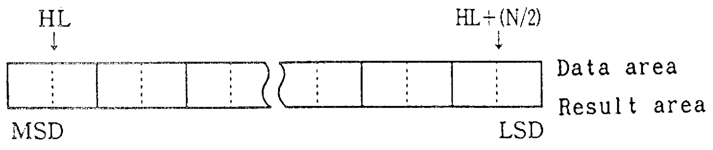
4

(5) Program list

```
;***************************************************
;*      N-byte data left shift                     *
;*          input condition                        *
;*              DE-register <- LSB of N-byte data  *
;*              C -register <- byte counter        *
;*          output condition                       *
;*              Acc <- MSB of N-byte data          *
;***************************************************

BYTLST:
        MOV     A,#0            ; Acc <- 0
BYTLS1:
        XCH     A,[DE+]
        DBNZ    C,$BYTLS1
        RET

        END
```

2.3.3 Shifting N-digit data 1 digit to right (decimal 1/10 processing)

(1) Memory



(2) Registers

A, C, HL

(3) Input conditions

As indicated in (1), contents of registers HL and C are set as follows.

. The highest address of N-digit data is set in register HL.
. The number of bytes (N/2) is set in register C.

(4) Output conditions

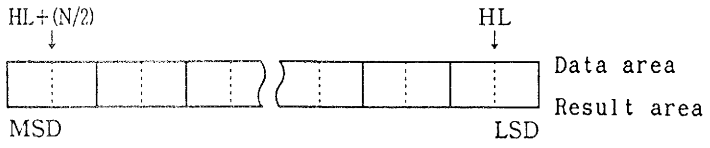
The result of 1-digit right shift is stored in the result area indicated in (1).
The register A contents are transferred to the highest digit.
The lowest digit contents are transferred to register A.

(5) Program list

```
        NAME    BCDRSR

;*****************************************************
;*      N-digit data right shift                    *
;*          input condition                         *
;*              HL-register <- MSD of N-digit data  *
;*              C -register <- digit counter        *
;*          output condition                        *
;*              Acc <- LSD of N-digit data          *
;*****************************************************

        PUBLIC  BCDRS,BCDRS1
        PUBLIC  BCDLS,BCDLS1
;
        CSEG
BCDRS:
        MOV     A,#0            ; Acc <- 0
BCDRS1:
        ROR4    [HL]
        DECW    HL              ; decrement (HL)
        DBNZ    C,$BCDRS1
        RET
```

2.3.4 Shifting N-digit data 1 digit to left (decimal tenfold processing)

(1) Memory



(2) Registers

A, C, HL

(3) Input conditions

As indicated in (1), the contents of registers HL and C are set as follows.

. The lowest address for N-digit data is set in register HL.

. The number of bytes (N/2) is set in register C.

(4) Output conditions

The result of 1-digit left shift is stored in the result area indicated in (1).

The register A contents are transferred to the lowest digit.

The highest digit contents are transferred to register A.

(5) Program list

```
;****************************************************
;*      N-digit data left shift                     *
;*          input condition                         *
;*              HL-register <- LSD of N-digit data  *
;*              C -register <- digit counter        *
;*          output condition                        *
;*              Acc <- MSD of N-digit data          *
;****************************************************

BCDLS:
        MOV     A,#0
BCDLS1:
        ROL4    [HL]
        INCW    HL              ; increment (HL)
        DBNZ    C,$BCDLS1
        RET
;
        END
```

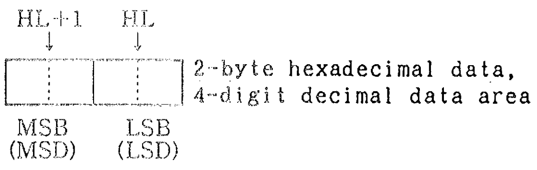
2.4 Data Conversion Processing

The example programs presented in this section converts the data representation format between hexadecimal and decimal, and between hexadecimal and ASCII.

2.4.1 Conversion from hexadecimal (HEX) to decimal (BCD)

Two-byte hexadecimal data is converted into 4-digit decimal data.

(1) Memory



(2) Registers

X, A,. C, B, DE, HL

(3) Input conditions

As indicated in (1), the contents of register HL are set as follows.

. The lowest address of the memory area where the 2-byte hexadecimal data is stored, is set in register HL.

(4) Output conditions

Hexadecimal data greater than 270FH (= 9999) cannot be converted and causes the carry flag (CY) to be set to 1.
When the hexadecimal data has been successfully converted into a 4-digit decimal number, it is stored in areas HL and HL + 1, and the CY remains 0.

(5) Processing sequence

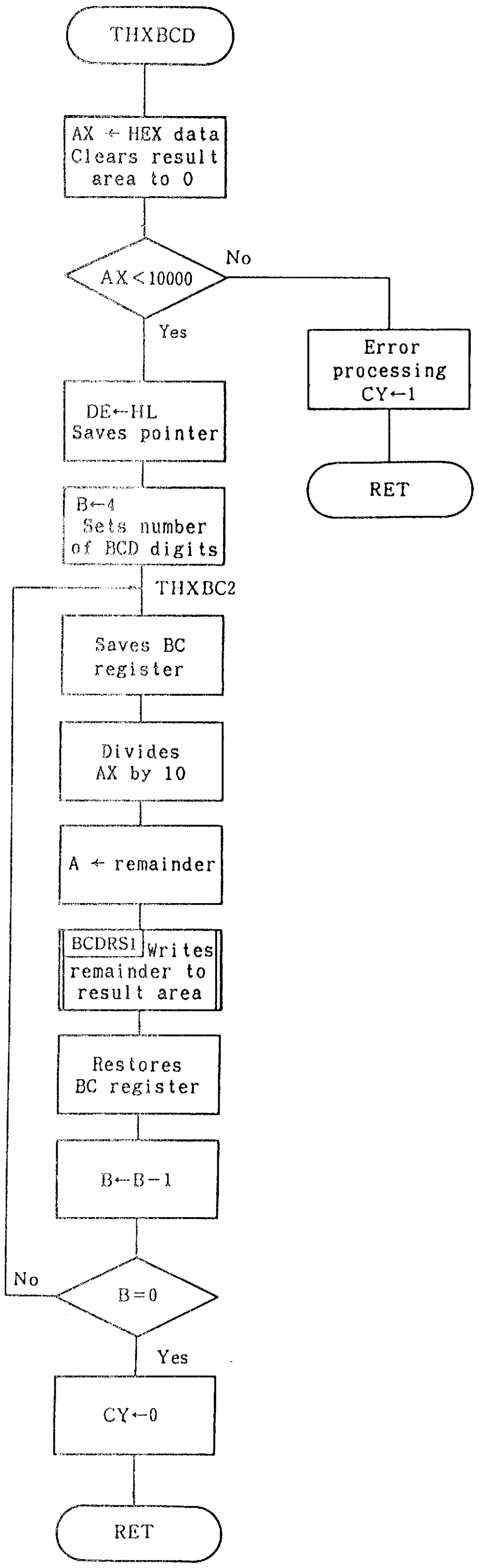
This program converts hexadecimal data into a 4-digit (2-byte) decimal number, starting from the lowest digit and on a digit-by-digit basis.
The hexadecimal data is divided by 10 and the remainder, resulting from the division, is generated as a BCD code.
The processing is accomplished in the following sequence:

(a) The input hexadecimal data is compared to the value 10000.
(b) If the hexadecimal data is greater than 10000, it cannot be converted. Consequently, the carry flag is set to 1 and the processing ends.
(c) The number of digits, 4, is set in register B.
(d) The divisor is set to 10.
(e) The input hexadecimal data is divided by the set divisor, 10.
(f) The remainder resulting from the division is set in register A, and the result area is shifted 1 digit to the right.
(g) The register B contents are decremented. Steps (e) to (f) above are repeated, as long as the register contents are not 0.
(h) When conversion has ended, the carry flag is cleared to 0.

(6) Number of steps

24

(7) Flowchart


THXBCD
AX ← HEX data
Clears result area to 0
AX < 10000
No
Yes
Error processing
CY←1
RET
DE←HL
Saves pointer
B←4
Sets number of BCD digits
THXBC2
Saves BC register
Divides AX by 10
A ← remainder
BCDRS1
Writes remainder to result area
Restores BC register
B←B−1
B=0
No
Yes
CY←0
RET

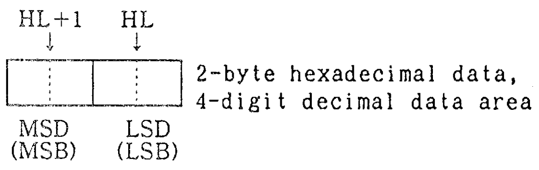
(8) Program list

```
        NAME    TRBCDR

;***************************************************
;*      transform BCD <- HEX                       *
;*         input condition                         *
;*              HL-register <- HEX-2byte data      *
;*                                   LSB address   *
;*         output condition                        *
;*              normal ... cy = 0                  *
;*                  decimal 4-digit -> (HL,HL+1)   *
;*              overflow ... cy = 1                *
;*                  HEX data > 9999                *
;***************************************************
;
        PUBLIC  THXBCD
        EXTRN   BCDRS1
;
        CSEG
THXBCD:
        MOVW    AX,#0           ; AX <-> [HL](hex data)
        XCH     A,[HL+]
        XCH     A,X
        XCH     A,[HL-]

        CMPW    AX,#10000       ; hex data >= 10000 then ret.
        BC      $THXBC1
        SET1    CY              ; 'CY' <- 1
        RET
;
THXBC1:
        MOVW    DE,HL           ; save HL-register
        MOV     B,#4            ; loop counter
THXBC2:
        PUSH    BC              ; save loop counter
        MOV     B,#10           ; set divisor
        DIVUW   B               ; AX / C
        PUSH    AX              ; save AX-register
        MOV     A,B
        MOVW    HL,DE           ; load HL-register

        MOV     C,#4            ; set length
        INCW    HL              ; set pointer
        CALL    !BCDRS1         ; 1-digit left shift
        POP     AX              ; load AX-register
        POP     BC              ; restore loop counter
        DBNZ    B,$THXBC2

        CLR1    CY              ; 'CY' <- 0
        RET
;
        END
```

2.4.2 Conversion from decimal (BCD) to hexadecimal (HEX)

Four-digit decimal data is converted into two-byte hexadecimal data.

(1) Memory



(2) Registers

X, A, C, B, DE, HL

(3) Input conditions

As indicated in (1), the contents of register HL are set as follows.

. The lowest address for the memory area, where the 4-digit decimal data is stored, is set in register HL.

(4) Output conditions

If the input data is not a decimal number, the data cannot be converted and the carry flag (CY) is set to 1.
When the decimal data has been successfully converted into 2-byte hexadecimal data, it is stored in areas HL and HL + 1, and the CY remains 0.

(5) Processing sequence

This program converts decimal data into 2-byte hexadecimal data, starting from the highest digit. Each of the four digits is transferred to register A one after another by shifting data 1 digit to left. The following operation is repeated four times to complete the conversion:

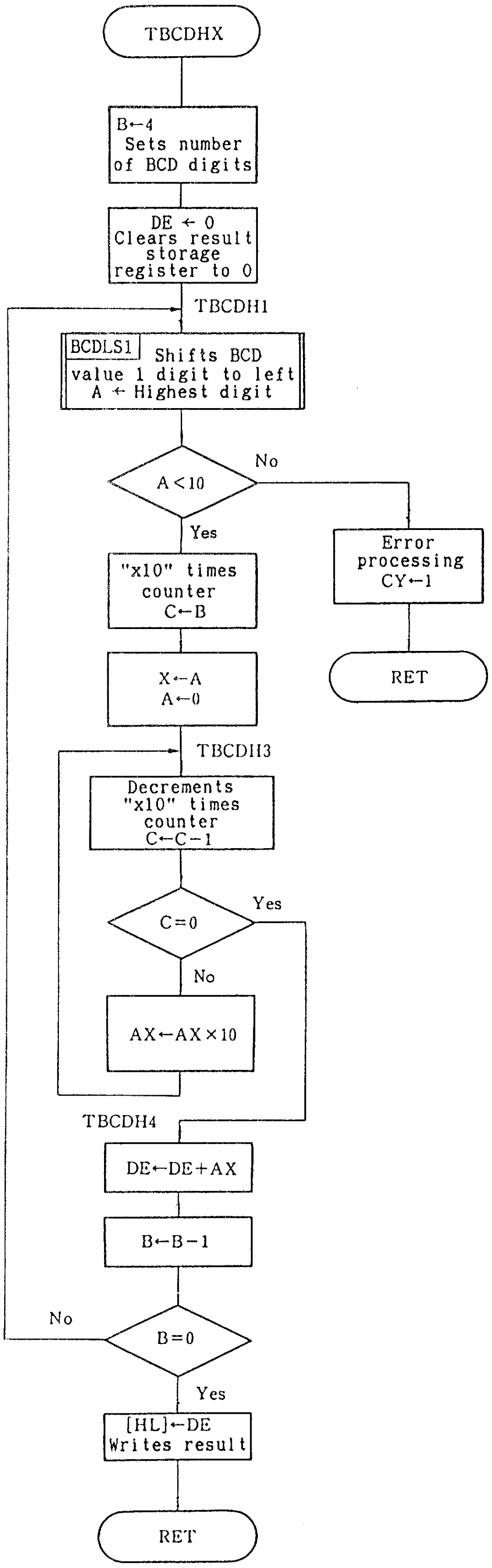
(Storage area) $\longleftarrow$ (Storage area) x 10 + Register A

(a) The number of BCD code digits, 4, is set in register B.
(b) The conversion result storage register (register DE) is cleared to 0.
(c) The input decimal data is shifted 1 digit to the left and the lowest digit is read into register A.
(d) The highest digit is checked, to see if it is decimal data (0 to 9). If it is not, conversion cannot be accomplished and the carry flag is set to 1.
(e) The operation "conversion result storage register conversion value storage register x 10 + register A" is executed.
(f) The register B contents are decremented. As long as the register contents are not 90, steps (c) to (e) above are repeated.
(g) The contents of the conversion result storage register are stored in the storage area.

(6) Number of steps

32

(7) Flowchart



TBCDHX
B←4 Sets number of BCD digits
DE ← 0 Clears result storage register to 0
TBCDH1
BCDLS1 Shifts BCD value 1 digit to left A ← Highest digit
A < 10
No
Yes
Error processing CY←1
RET
"x10" times counter C←B
X←A A←0
TBCDH3
Decrements "x10" times counter C←C−1
C=0
Yes
No
AX←AX×10
TBCDH4
DE←DE+AX
B←B−1
No
B=0
Yes
[HL]←DE Writes result
RET

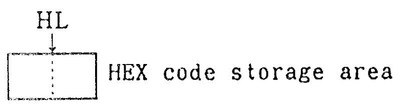
(8) Program list

```
        NAME    TRHEXR

;*******************************************************
;*      transform HEX <- BCD                           *
;*          input condition                            *
;*              HL-register <- decimal 4 digit data    *
;*                                      LSD address    *
;*          output condition                           *
;*              normal ... cy = 0                      *
;*                    HEX 2 byte -> (HL,HL+1)          *
;*              error  ... cy = 1                      *
;*******************************************************

        PUBLIC  TBCDHX
        EXTRN   BCDLS1
;
        CSEG
TBCDHX:
        MOV     B,#4            ; BCD length
        MOVW    DE,#0           ; result work
TBCDH1:
        PUSH    HL              ; save pointer
        MOV     C,#2            ; shift counter
        MOV     A,#0
        CALL    !BCDLS1         ; BCD left shift
        POP     HL              ; restore pointer
        CMP     A,#10           ; error check
        NOT1    CY
        BNC     $TBCDH2
        RET                     ; error return
TBCDH2:
        MOV     C,B
        MOV     X,#0
        XCH     A,X
TBCDH3:
        DEC     C
        BZ      $TBCDH4

        PUSH    BC              ; AX <- AX * 10
        MOVW    BC,AX
        SHLW    AX,2
        ADDW    AX,BC
        SHLW    AX,1
        POP     BC
        BR      TBCDH3
TBCDH4:
        ADDW    AX,DE           ; result addition
        MOVW    DE,AX
        DBNZ    B,$TBCDH1       ; check length

        MOVW    AX,DE           ; write result to memory
        XCH     A,X
        MOV     [HL+],A
        MOV     A,X
        MOV     [HL],A
        RET
;
        END
```

2.4.3 Converting ASCII into hexadecimal (HEX)

Two ASCII codes (30H to 39H and 41H to 46H) are converted into two hexadecimal codes (0 to FFH).

(1) Memory



(2) Registers

A, C, B, HL

(3) Input conditions

As indicated in (1), the contents of registers BC and HL are set as follows.

. Two ASCII codes are set in register BC.
. The two hexadecimal codes, obtained as a result of converting the ASCII codes, are stored in an address indicated by register HL.

(4) Output conditions

If the input data is not an ASCII code, conversion cannot be accomplished and the carry flag (CY) is set to 1. When the input ASCII codes have been converted into two hexadecimal codes and stored in an area indicated in (1), the carry flag is cleared to 0.

(5) Processing sequence

(a) The higher ASCII code (register B) is read into register A.

(b) Whether the register A contents are within a range of 30H to 39H or 41H to 46H is checked. If the register contents are not within either of the ranges, the conversion cannot be accomplished and the carry flag is set to 1.

(c) If the register A contents are 30H to 39H, 30H is subtracted from the register contents.
If the register A contents are 41H to 46H, 37H is subtracted from the register contents.

(d) The contents of the address indicated by register HL are shifted 4 bits and the register A contents are stored in the lower 4 bits of the address.

(e) The lower ASCII code (register C) is read into register A, and steps (b) to (d) are performed.

(6) Number of steps

19

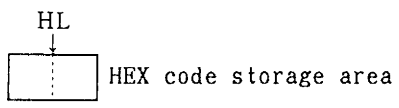
(7) Program list

```
        NAME    GHEXR

;***********************************************
;*      transform   HEX <-  ASCII             *
;*                (2code)  (2code)            *
;*                                            *
;*              input  condition              *
;*                BC-register <- ASCII        *
;*                                            *
;*              output condition              *
;*                (HL)  <-  hex               *
;***********************************************

        PUBLIC  GETHEX
        PUBLIC  SHEX
;
        CSEG
GETHEX:
        MOV     A,B             ; ASCII upper-code load
        CALL    !SHEX           ; get  hex 1th code
        BC      $GTHEX1

        ROL4    [HL]
        MOV     A,C             ; ASCII lower-code load
        CALL    !SHEX           ; get  hex 2th code
        BC      $GTHEX1
        ROL4    [HL]
GTHEX1:
        RET

;*******************************************
;*     subroutine  /  get hex 1-code(Acc)  *
;*******************************************

SHEX:
        CMP     A,#'9'          ; check / ASCII > 39H
        BNC     $SHEX1
        SUB     A,#30H
        RET
SHEX1:
        CMP     A,#'F'          ; check / ASCII < 46H
        BNC     $SHEX2
        SUB     A,#37H
        RET
SHEX2:
        SET1    CY              ; error
        RET
;
        END
```

2.4.4 Converting hexadecimal (HEX) into ASCII

Two hexadecimal codes (0 to FFH) are converted into two ASCII codes (30H to 39H and 41H to 46H).

(1) Memory



(2) Registers

A, C, B, HL

(3) Input conditions

As indicated in (1), the contents of register HL are set as follows.

. The address where two hexadecimal codes to be converted into ASCII codes are set in register HL.

(4) Output conditions

The two ASCII codes, obtained as a result of the conversion, are output to register BC.

(5) Processing sequence

(a) The higher 4 bits of the address indicated by register HL are transferred to register A.
(b) Whether the register A contents are 10 or greater is checked. If the register contents are less than 10, execution branches to (d).
(c) The value 7 is added to the register A contents.
(d) The value 30H is added to the register A contents.
(e) The register A contents are transferred to register B.

(f) The lower 4 bits of register HL are transferred to register A.

(g) Steps (b) to (c) above are carried out again, and the register A contents are transferred to register C.

(6) Number of steps

14

(7) Program list

```
        NAME    ASCII

;************************************************
;*      transform    ASCII  <-  HEX            *
;*                   (2code)    (2code)         *
;*                                              *
;*              input  condition               *
;*                 (HL) <- hex 2-code           *
;*              output condition               *
;*                 BC-register <- ASCII 2-code  *
;************************************************

        PUBLIC  GETASC
        PUBLIC  SASC
;
        CSEG
GETASC:
        MOV     A,#0
        ROL4    [HL]            ; hex upper code load
        CALL    !SASC
        MOV     B,A             ; store result

        MOV     A,#0
        ROL4    [HL]            ; hex lower code load
        CALL    !SASC
        MOV     C,A             ; store result
        RET

;****************************************************
;*      subroutine  /  get ASCII 1-code(BC-register) *
;****************************************************

SASC:
        CMP     A,#0AH          ; check / hex > 9
        BC      $SASC1
        ADD     A,#07H          ; bias (+7)
SASC1:
        ADD     A,#30H          ; bias (+30H)
        RET
;
        END
```

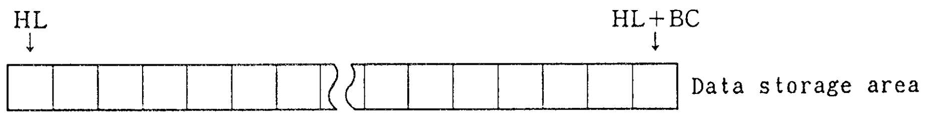
2.5 Data Processing

This section shows two data processing program examples, one of which sorts data, while the other searches for data.

2.5.1 Sorting data

The program example presented here is used to sort data file contents in an ascending order, using the bubble sort technique. Each set of data in this file is 8 bits long.

(1) Memory



(2) Registers

X, A, C, B, HL (The HL register contents are retained.)

(3) Input conditions

As indicated in (1), the contents of registers HL and BC are set as follows.

. The first address for the data string to be sorted is set in register HL.
. The quantity of data (i.e., the number of bytes) is set in register BC.

(4) Output conditions

The data string in the area indicated in (1) is sorted in ascending order. The contents of register HL, which indicates the first address for the data string, are retained.

(5) Processing sequence

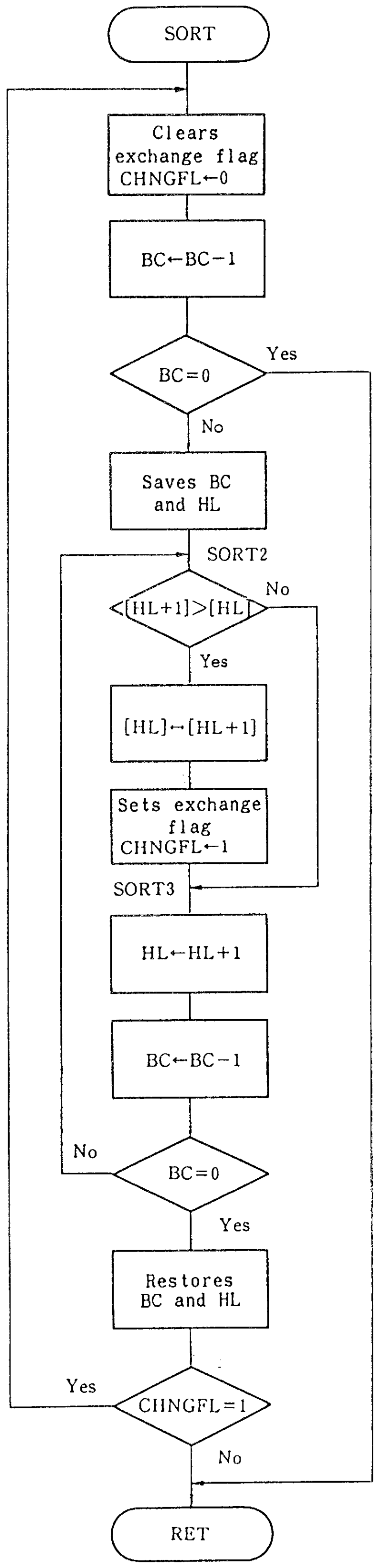
This sorting program uses the bubble sort technique to sort data. Processing is carried out in the following sequence:

(a) CNGGFL, a flag that indicates the data has been exchanged, is cleared to 0.
(b) The register BC value, which indicates the number of bytes in the data, is decremented. If the value of this register is 0, it means that the sorting has been completed.
(c) The contents of registers HL and BC are saved.
(d) The value of (HL) and the next address (HL + 1) are compared. If the next address is equal to (HL) or if the value of (HL) is greater than the address value, execution branches to (f).
(e) The contents of the address indicated by register HL are exchanged with the contents of the address indicated by the HL register contents plus 1, and the CHNGFL flag is set to 1.
(f) The contents of register HL are incremented, and those of register BC are decremented.
(g) If the register BC value is not 0, steps (d) to (f) above are repeated.
(h) The contents of registers HL and BC are restored.
(i) If the exchange flag, CHNGFL, is set to 1, steps (a) to (h) are repeated. If the flag is not set to 1, the sorting is completed.

(6) Number of steps

24

(7) Flowchart


SORT
Clears exchange flag CHNGFL←0
BC←BC−1
BC=0
Yes
No
Saves BC and HL
SORT2
[HL+1]>[HL]
No
Yes
[HL]↔[HL+1]
Sets exchange flag CHNGFL←1
SORT3
HL←HL+1
BC←BC−1
No
BC=0
Yes
Restores BC and HL
Yes
CHNGFL=1
No
RET

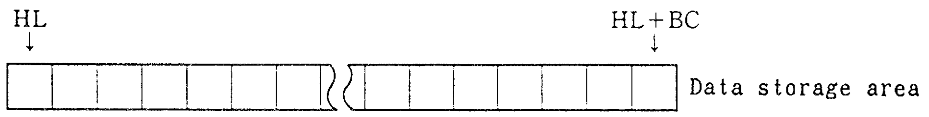
(8) Program list

```
        NAME    SORTR

;*************************************************
;*      bubble sort                              *
;*       input condition                         *
;*             BC-register <- number of data     *
;*             HL-register <- data top.address   *
;*        output condition                       *
;*             HL-register <- data top.address   *
;*************************************************

        PUBLIC  SORT

        BSEG
CHNGFL  DBIT                    ; change-flag

SORT_D  DSEG    SADDR
CNTSTK: DS      1               ; counter save area (saddr area)
;
        CSEG
SORT:
        CLR1    CHNGFL          ; change-flag <- 0
        DECW    BC
        MOV     A,B
        OR      A,C
        BNZ     $SORT1
        RET
SORT1:
        PUSH    BC              ; save pointer/counter
        PUSH    HL
SORT2:
        MOV     A,[HL]          ; change process
        CMP     A,[HL+1]
        BC      $SORT3          ; A <= [HL+1] goto SORT3
        BZ      $SORT3
        XCH     A,[HL+1]
        MOV     [HL],A
        SET1    CHNGFL          ; change-flag <- 1
SORT3:
        INCW    HL              ; increment pointer
        DECW    BC
        MOV     A,B
        OR      A,C
        BNZ     $SORT2
        POP     HL              ; restore pointer/counter
        POP     BC
        BT      CHNGFL,$SORT
        RET
;
        END
```

2.5.2 Searching data

The program introduced in this section is used to search for specified data, implementing the binary search technique. When the data has been found, the address at which the data is stored is returned.

(1) Memory



(2) Registers

X, A, C, B, DE, HL (The register AX contents are retained.)

(3) Input conditions

As indicated in (1), the contentsl of registers A, HL and BC are set as follows.

. The data to be searched for is set in register A.
. Register HL holds the first address for the data string to be searched.
. The quantity of data (number of bytes) to be searched for is set in register BC.

(4) Output conditions

When the carry flag (CY) is cleared to 0, the address at which the searched for data is stored is output to register HL. The carry flag is set to 1, if the specified data is not found. In this case, the register HL contents become undefined.

(5) Processing sequence

This program uses the binary search technique to search for the specified data.

The processing is accomplished in the following sequence:

(a) The data to be searched for is saved.

(b) The first address for the data is set in register DE. The last address for the data is set in register BC.

(c) The data to be searched for is restored.

(d) The intermediate address for the search range is set in register HL.

(e) The register DE value is compared with the register BC value. If DE register value is less than the BC register value, the data to be searched for is restored. After that, the carry flag is set to 1 and the processing ends.

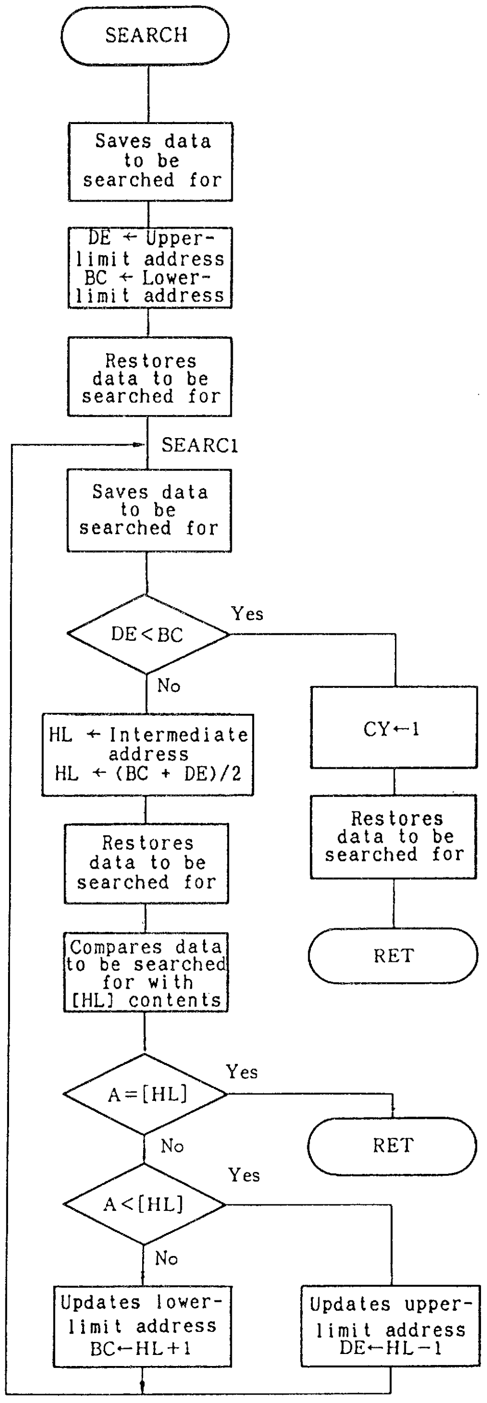
(f) The data to be searched for and the contents indicated by register HL are compared. If the data coincides with the register contents, which means that the data has been found, the searching process is completed.

(g) If the carry flag is set to 1, the contents of HL register minus 1 are transferred to register BC to set a new last address. If the carry flag is cleared to 0, the contents of register HL plus 1 are transferred to register DE to set a new first address. Execution then jumps to (d).

(6) Number of steps

27

(7) Flowchart


SEARCH
Saves data to be searched for
DE ← Upper-limit address
BC ← Lower-limit address
Restores data to be searched for
SEARC1
Saves data to be searched for
DE < BC
Yes
No
CY ← 1
Restores data to be searched for
RET
HL ← Intermediate address
HL ← (BC + DE)/2
Restores data to be searched for
Compares data to be searched for with [HL] contents
A = [HL]
Yes
RET
No
A < [HL]
Yes
No
Updates lower-limit address BC ← HL + 1
Updates upper-limit address DE ← HL − 1

(8) Program list

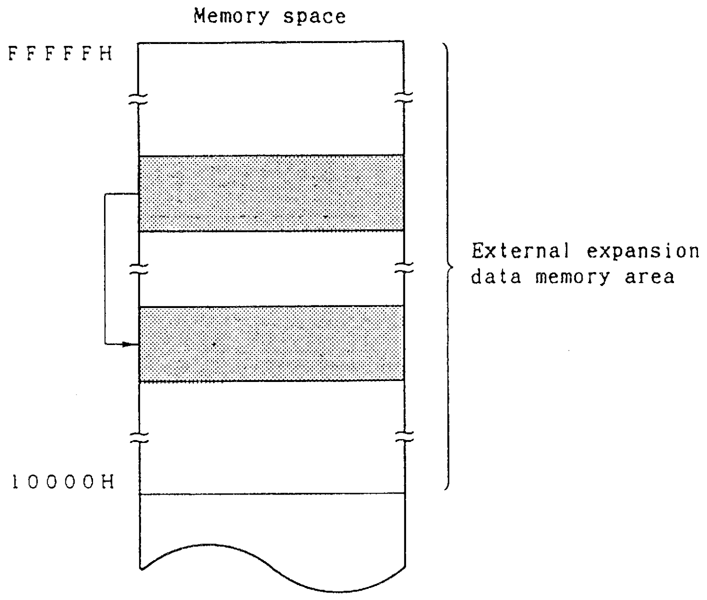
```
        NAME    SEARCR

;*************************************************
;*      bubble sort                              *
;*       input condition                         *
;*              A-register  <- search data       *
;*              BC-register <- number of data    *
;*              HL-register <- data top.address  *
;*        output condition                       *
;*              HL-register <- found data address *
;*************************************************

        PUBLIC  SEARCH
;
        CSEG
SEARCH:
        PUSH    AX              ; save search data
        MOVW    AX,HL
        ADDW    AX,BC
        MOVW    DE,AX           ; DE-register <- upper.address
        MOVW    BC,HL           ; BC-register <- lower.address
        POP     AX              ; restore search data
SEARC1:
        PUSH    AX              ; save search data
        MOVW    AX,DE           ; HL-register <- center.address
        SUBW    AX,BC
        BC      $SEARC4         ; search end check
        SHRW    AX,1
        ADDW    AX,BC
        MOVW    HL,AX

        POP     AX
        CMP     A,[HL]
        BNZ     $SEARC2
        RET                     ; found data
SEARC2:
        BC      $SEARC3
        INCW    HL              ; 'CY' = 0
        MOVW    BC,HL
        BR      SEARC1
SEARC3:
        DECW    HL              ; 'CY' = 1
        MOVW    DE,HL
        BR      SEARC1
SEARC4:
        POP     AX
        SET1    CY
        RET
;
        END
```

2.6 Data Transfer in External Expansion Data Memory Space

1-byte data is continuously transferred within the external expansion data memory (10000H to FFFFFH).

(1) Memory used

Fig. 2-3 Memory Block Diagram for Data Transfer in External Expansion Data Memory Space



(2) Registers used
A, B, DE, HL (register bank 0)

(3) Input condition
Parameters indicated in Table 2-1 are defined.

Table 2-1 Input Parameters for Data Transfer in External Expansion Data Memory Space

Parameter	Fixed/variable	Contents
TRSADR	Fixed value	Lower 16 bits for transfer source address
TRDADR	Fixed value	Lower 16 bits for transfer destination address
MBANKS	Fixed value	Upper 4 bits for transfer source address (memory bank)
MBANKD*	Fixed value	Upper 4 bits for transfer destination address (memory bank)
TRCOUNT	Fixed value	Transfer data amount

*: Set MBANKD to 0 in the uPD78224 series.

(4) Output condition

None

(5) Processing procedure

Set the transfer source memory bank (0H to FH) in the P6 register, and the transfer destination memory bank (0H to FH) in the PM6 register.

The lower 4 bits of PM6 are fixed to 0 in the uPD78224 series.

(a) Sets the 1M-byte expansion mode, (refer to Mode Register Setting Example in the figure below).

Memory expansion mode register

	7	6	5	4	3	2	1	0	
MM	IFCH	MM6	PW21	PW20	0	MM2	MM1	MM0	(20H when reset)
Setting example	x	1	x	x	0	x	x	x	x: Not manipulated

MM2	MM1	MM0	Mode	P50-P57	P40-P47	P65	P64
0	0	0	Single-chip mode	Port mode	Input port	Port mode	
0	0	1			Output port		
1	1	1	External memory expan-sion mode	A8-A15	AD0-AD7	$\overline{WR}$	$\overline{RD}$

PW21	PW20	Number of waits (specification range: 00000H to 0FFFFH)
0	0	0 wait
0	1	1 wait
1	0	2 waits
1	1	Number of waits equivalent to low level period for $\overline{WAIT}$ pin input

MM6	1M-byte expansion mode specification
0	P60-P63 are general purpose output ports.
1	The output latches (PL60-PL63) for P60-P63 store the upper address (A16-A19) in the external memory expansion mode, and P60-P63 serve as the output pins for A16-A19.

IFCH	Internal fetch cycle control
0	Instruction execution cycle becomes a cycle similar to the external ROM fetch cycle.
1	Carries out high speed internal ROM fetch operation. (Execution cycle becomes faster than external ROM fetching.)

(b) Set number of transfer data, transfer source and transfer destination addresses.

(c) Advancing the pointer until the data counter number (register B) becomes 0, carry out write operation.

(6) Number of steps

10 steps

(7) Stack used

2 bytes: 1 level

(8) Programming example

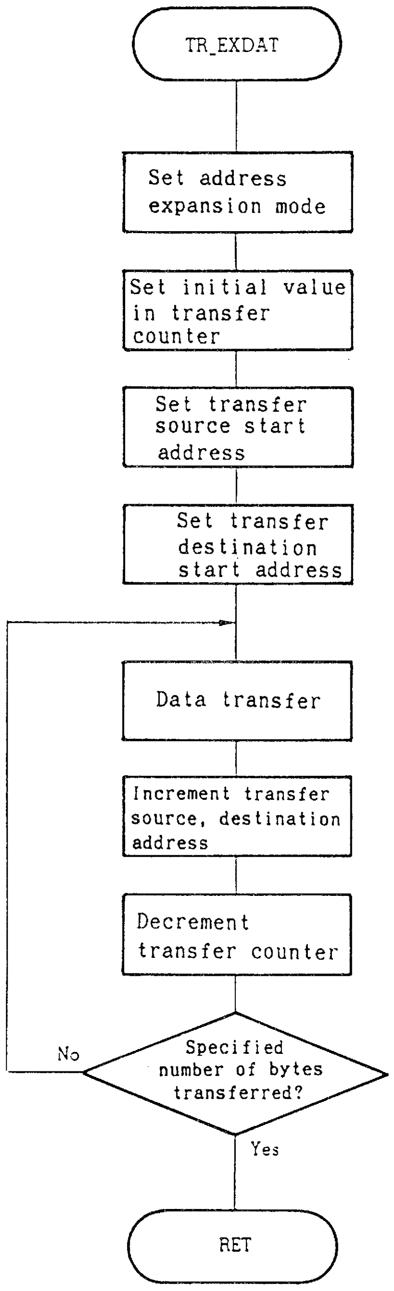
The program shown below is an example of transferring 32 bytes of data from address 60000H to addresses, starting from address 40000H.

Define the parameters in the user program, as shown below, to call the TR_EXDAT subroutine.

```
          PUBLIC  TRSADR,TRDADR,TRCOUNT    ; data
          PUBLIC  MBANKS,MBANKD            ; data
          EXTRN   TR_EXDAT                 ; package

 ;        --- define data ---

 TRSADR  EQU     0000H                     ; source address of transmission
 TRDADR  EQU     0000H                     ; destination address of transmission
 TRCOUNT EQU     32                        ; data count of transmission
 MBANKS  EQU     6                         ; memory bank NO. (source)
 MBANKD  EQU     4                         ;                 (destination)

          CSEG
            :
            :
          CALL    !TR_EXDAT                ; data transmission package

            :
            :
```

Remarks: When relocatably defining symbols in the external expansion data memory space, use a directive file when linking. Refer to Appendix B.3, "How to Use 1M-Byte Expansion Data Memory Space" for details.

(9) Flowchart



TR_EXDAT
Set address expansion mode
Set initial value in transfer counter
Set transfer source start address
Set transfer destination start address
Data transfer
Increment transfer source, destination address
Decrement transfer counter
Specified number of bytes transferred?
No
Yes
RET

(10) Program list

```
        NAME    TREXDAT
;**********************************************************
;*      DATA TRANSMISSION IN                             *
;*              EXTERNALLY EXTENDED MEMORY AREA          *
;**********************************************************

        PUBLIC  TR_EXDAT                ; package
        EXTRN   TRSADR,TRDADR,TRCOUNT   ; data
        EXTRN   MBANKS,MBANKD           ; data

EXDATCS CSEG
TR_EXDAT:
        OR      MM,#01000000B           ; extended address mode
        MOV     B,#TRCOUNT              ; count of transmission
        MOV     P6,#MBANKS              ; memory bank NO. (source)
        MOV     PM6,#MBANKD             ;                 (destination)
        MOVW    HL,#TRSADR              ; source address of transmission
        MOVW    DE,#TRDADR              ; destination address of transmission

TR_LOOP:
        MOV     A,&[HL+]                ; transmit data
        MOV     [DE+],A
        DBNZ    B,$TR_LOOP              ; transmit until counter is 0

        RET
        END
```

CHAPTER 3 TIMER/COUNTER PROGRAM EXAMPLES

The 78K/II series timer/counter configuration and functions differ, depending on the product. Tables 3-1, 3-2, and 3-3 show the configuration and functions for each of the 78K/II series products.

Table 3-1 uPD78214 Series Timer/Counter Functions and Types

Type and function \ Unit			16-bit timer/counter	8-bit timer/counter 1	8-bit timer/counter 2	8-bit timer/counter 3
Type	Interval timer		2ch	2ch	2ch	1ch
	External event counter		-	-	o	-
	One-shot timer		-	-	o	-
Function	Timer output		2ch	-	2ch	-
		Toggle output	o	-	o	-
		PWM/PPG output	o	-	o	-
		One-shot pulse output	-	-	-	-
	Real-time output		-	o	-	-
	Pulse width measurement		o	o	o	-
	Number of interrupt request sources		2	2	2	1
	Serial interface clock source		-	-	-	o

Table 3-2 Timer/Counter Functions and Types in uPD78218A Series, uPD78234 Series, uPD78244 Series

Type and function \ Unit		16-bit timer/ counter	8-bit timer/ counter 1	8-bit timer/ counter 2	8-bit timer/ counter 3
Type	Interval timer	2ch	2ch	2ch	1ch
	External event counter	-	-	o	-
	One-shot timer	-	-	o	-
Function	Timer output	2ch	-	2ch	-
	Toggle output	o	-	o	-
	PWM/PPG output	o	-	o	-
	One-shot pulse output	o	-	-	-
	Real-time output	-	o	-	-
	Pulse width measurement	o	o	o	-
	Number of interrupt request sources	2	2	2	1
	Serial interface clock source	-	-	-	o

Table 3-3 uPD78224 Series Timer/Counter Functions and Types

Type and function \ Unit		16-bit timer/counter	8-bit timer/ counter 1	8-bit timer/ counter 2
Type	Interval timer	2ch	2ch	1ch
	External event counter	-	-	o
	One-shot timer	-	-	o
Function	Timer output	2ch	-	2ch
	Toggle output	o	-	o
	PWM/PPG output	-	-	-
	One-shot pulse output	-	-	-
	Real-time output	-	o	-
	Number of interrupt request sources	2	2	1
	Pulse width measurement	o	o	o
	Serial interface clock source	-	-	-

This section introduces program examples for the following timer/counter functions:

(i) Internal interval timer (3.1)
(ii) Toggle output (3.2)
(iii) Free-running interval timer (3.3)
(iv) PWM/PPG output (3.4)
(v) Pulse interval measurement (3.5)

3.1 Internal Interval Timer

The interval timer generates an interrupt signal to the CPU at fixed intervals.

(1) Program example with 16-bit timer/counter

An interval timer function can be programmed by using a 16-bit timer/counter. In this case, an interval of 1.3 microsecond to 87.4 milliseconds (where f_{CLK} = 6 MHz) can be programmed with a 1.3 microsecond resolution. This section introduces a program example that uses a 16-bit timer/counter to generate an interrupt request (INTC01) at fixed intervals.

(a) Operation

Fig. 3-1 is a blockdiagram that illustrates the interval timer functions that generate INTC01 interrupt request.

Fig. 3-1 Interval Timer Generating INTC01 Interrupt Request

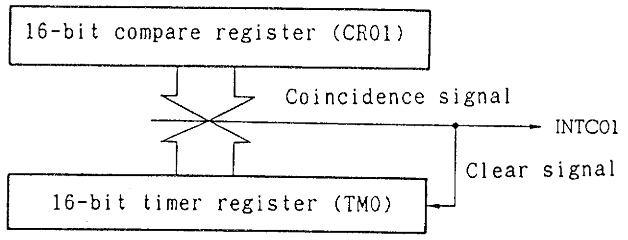
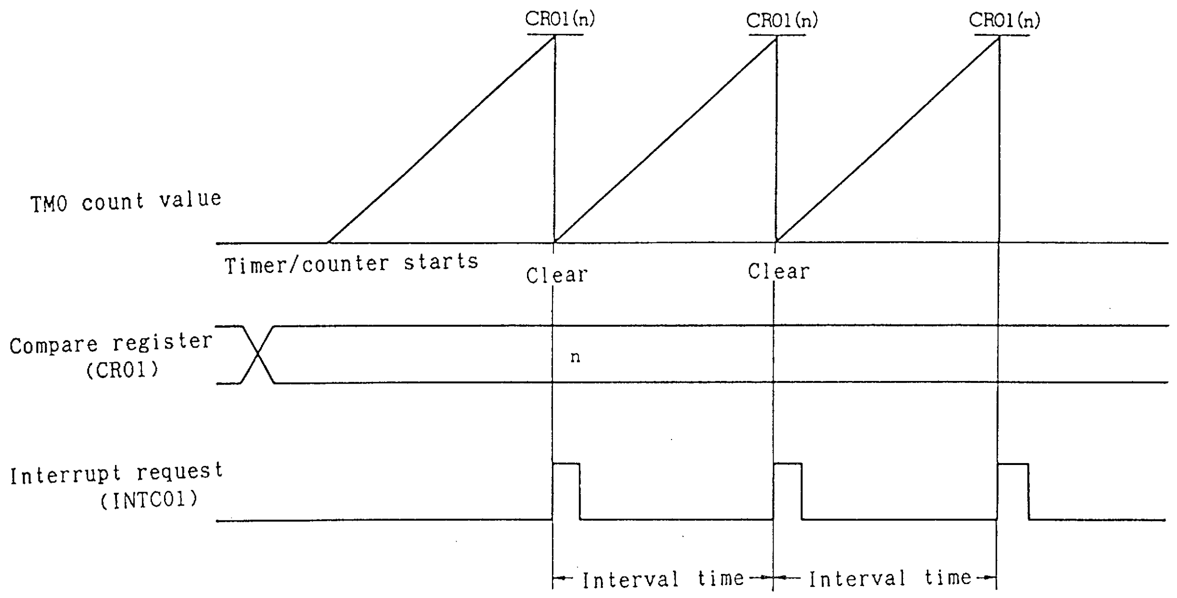


Fig. 3-2 Timing Chart



$$\text{Interval time} = (n + 1) \times 8/f_{CLK}$$

where,

$\{n: 0 \leqq n \leqq \text{FFFFH}\}$

The interrupt request INTC01 is generated, when the 16-bit timer register TM0 contents coincide with those for 16-bit compare register CR01. To generate the interrupt request at fixed intervals, therefore, TM0 must be cleared when coincidence has taken place.
Note that, although the 16-bit timer/counter has two 16-bit compare registers, CR00 and CR01, only CR01 has the timer clear function. Therefore, set an interval time in CR01.

(b) Program [label: IITVL1] ... Refer to (h) Program list

(i) When the 16-bit timer register (TM0) contents coincide with those for the 16-bit compare register (CR01), clearing TM0 is enabled.

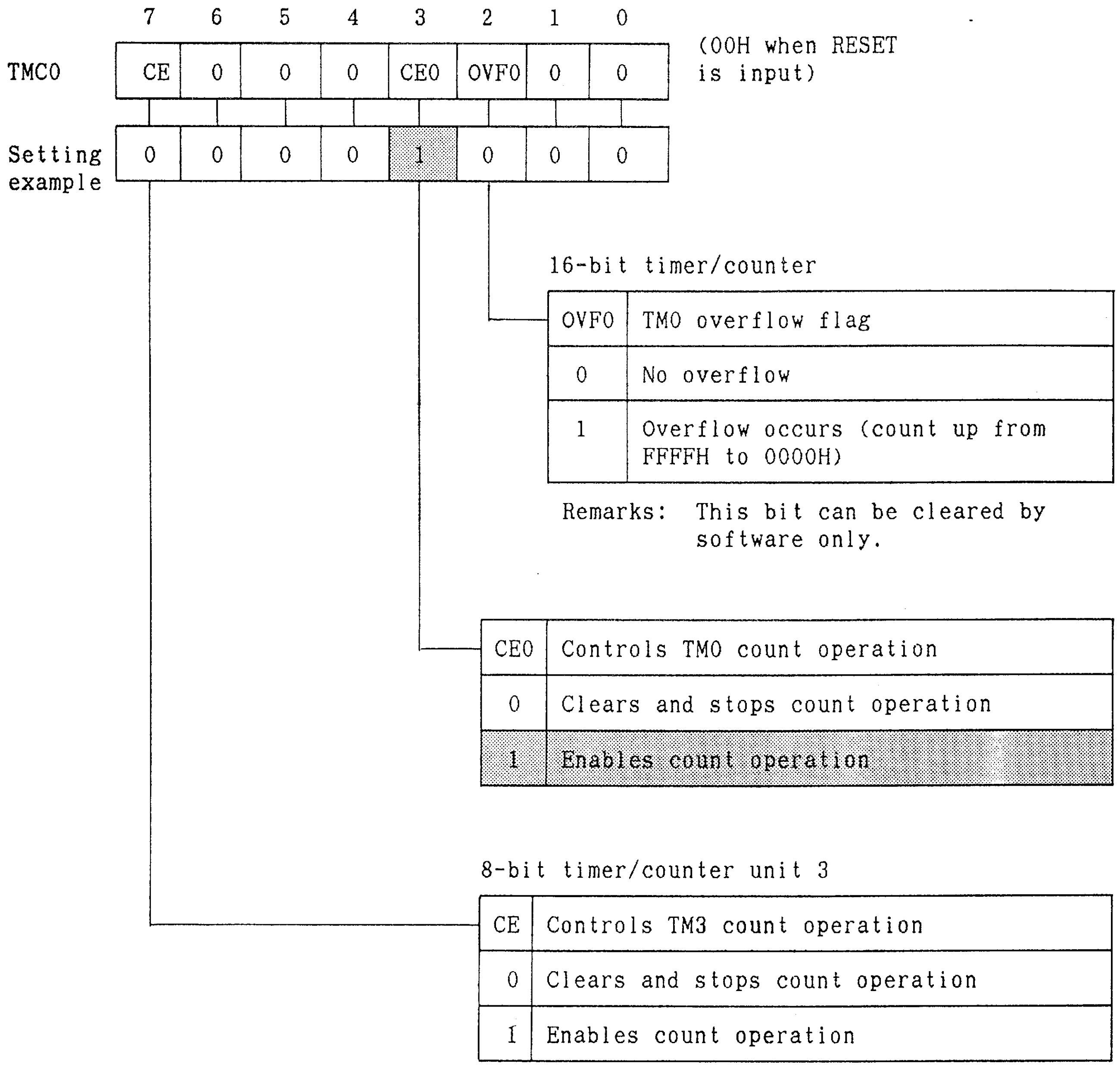
(ii) An interval time is set in CR01.

(iii) Interrupt request INTC01 is unmasked.

(iv) The TM0 counting operation is enabled.

(c) Mode register setting

Timer control register



	7	6	5	4	3	2	1	0
TMC0	CE	0	0	0	CE0	OVF0	0	0
Setting example	0	0	0	0	1	0	0	0

(00H when RESET is input)

16-bit timer/counter

OVF0	TM0 overflow flag
0	No overflow
1	Overflow occurs (count up from FFFFH to 0000H)

Remarks: This bit can be cleared by software only.

CE0	Controls TM0 count operation
0	Clears and stops count operation
1	Enables count operation

8-bit timer/counter unit 3

CE	Controls TM3 count operation
0	Clears and stops count operation
1	Enables count operation

Capture/compare control register 0

	7	6	5	4	3	2	1	0	
CRC0	MOD1	MOD0	0	1	CLR01	0	0	0	(10H when RESET is input)
Setting example	0	0	0	1	1	0	0	0	

MOD1	MOD0	CLR01	Timer output mode specification		TM0 clear operation when TM0=CR01
			TO0	TO1	
0	0	0	Toggle output	Toggle output	Disabled
0	0	1	Toggle output	Toggle output	Enabled
0	1	0	PWM output	Toggle output	Disabled
0	1	1	Not allowed		
1	0	0	PWM output	PWM output	Disabled
1	0	1	Not allowed		
1	1	0	Not allowed		
1	1	1	PPG output	Toggle output	Enabled

Interrupt mask register L

	7	6	5	4	3	2	1	0	
MK0L	CMK11	CMK10	CMK01	CMK00	PMK3	PMK2	PMK1	PMK0	(FFH when RESET is input)
Setting example	x	x	0	x	x	x	x	x	x: not used

MK	Interrupt mask flag
0	Enables interrupt processing
1	Disables interrupt processing

(d) Input/output parameter

INTVL1: Sets a parameter that determines the interval time.

Since the count clock for the 16-bit timer/counter is fixed to $f_{CLK}/8$, the interval time that is determined by the value set as parameter INTVL1 can be expressed as follows:

$$\text{Interval time} = \text{INTVL1} \times 8/f_{CLK}$$

where,

$\{\text{INTVL1}: 0 \leqq \text{INTVL1} \leqq \text{FFFFH}\}$

(e) Registers

No register is used.

(f) Program example

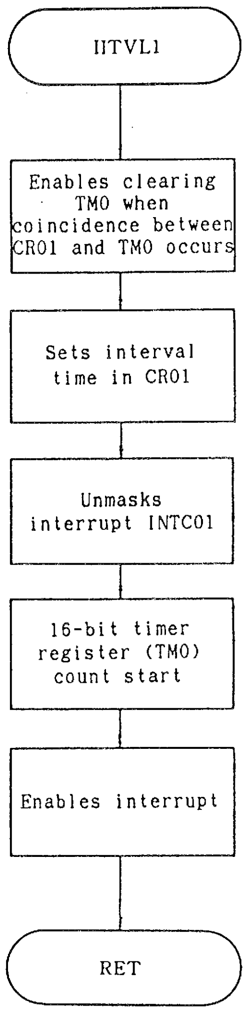
An example program that generates interrupt request INTC01 at 10 ms intervals, where f_{CLK} = 6 MHz, is shown on the next page.
To generate the interrupt request every 10 ms, the input parameter INTVL1 value should be as follows:

$$\boxed{\begin{aligned} \text{INTVL1} &= \frac{10 \times 10^{-3}}{8/(6 \times 10^{6})} \\ &= 7500 \end{aligned}}$$

Therefore, program an interval timer by defining an external definition and external reference directives, as follows:

```
        PUBLIC  INTVL1          ; PARAMETER
        EXTRN   IITVL1          ; PACKAGE
                .
                .
INTVL1  EQU     7500            ; PARAMETER FOR INTERVAL
                .
                .
        CALL    !IITVL1
                .
                .
```

(g) Flowchart


IITVL1
Enables clearing TM0 when coincidence between CR01 and TM0 occurs
Sets interval time in CR01
Unmasks interrupt INTC01
16-bit timer register (TM0) count start
Enables interrupt
RET

(h) Program list

```
        NAME    IITV1M
;
;*******************************************************
;* 16bit-Timer / Counter Unit                          *
;*      internal interval timer                        *
;*******************************************************
;
        PUBLIC  IITVL1
        EXTRN   INTVL1          ; Compare data for interval timer

CMK01   EQU     MK0L.5          ; INTC01 mask flag
;
        CSEG
IITVL1:
        MOV     CRC0,#00011000B ; clear enable TM0 by CR01
        MOVW    CR01,#INTVL1-1  ; set interval time
        CLR1    CMK01           ; open INTC01 mask
        MOV     TMC0,#00001000B ; timer start
        EI                      ; interrupt enable

        RET
;
        END
```

(2) Program example with 8-bit timer/counter 2

This section presents an example program that generates time interval by using interrupt request INTC21.

(a) Operation

Fig. 3-3 shows the interval timer that generates interrupt request INTC21 at fixed intervals.

Fig. 3-3 Interval Timer Generating Interrupt Request INTC21

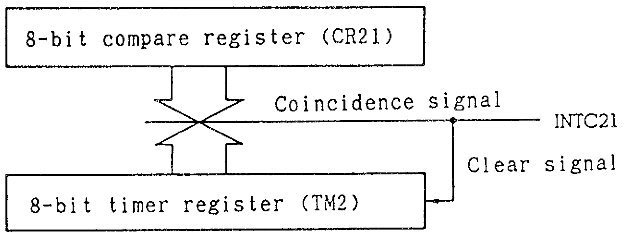
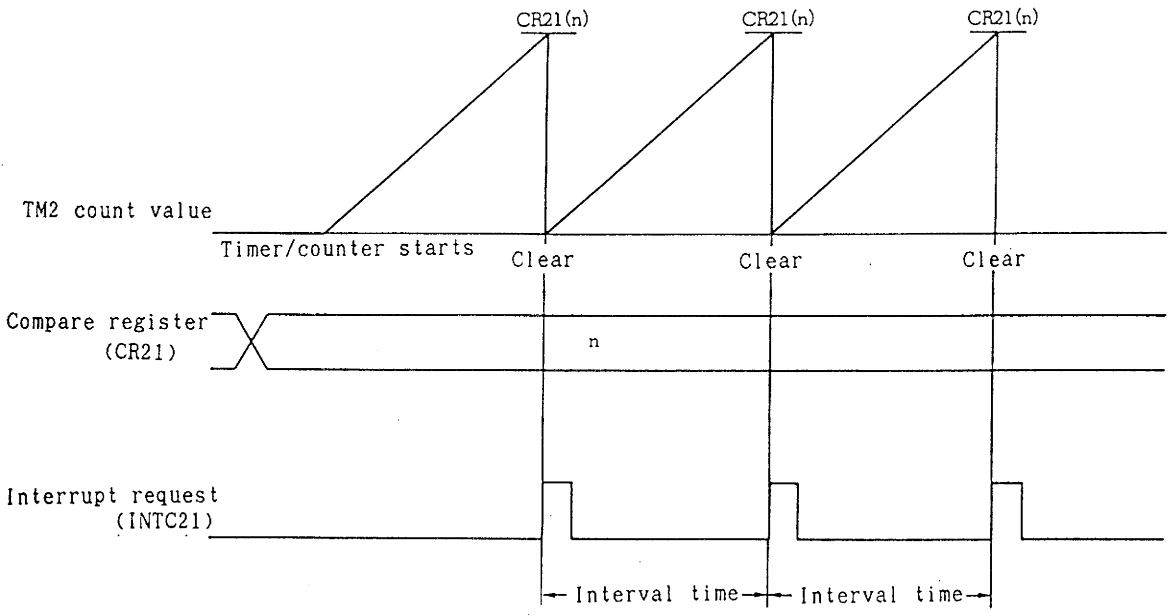


Fig. 3-4 Timing Chart



$$\text{Interval time} = (n + 1) \times X/f_{CLK}$$

where,

$\{N: 0 \leqq n \leqq FFH\}$

$X = 16, 32, 64, 128, 256, 512$

Interrupt request INTC21 is generated when the 8-bit timer register TM2 contents coincide with those for 8-bit compare register CR21. To generate the interrupt request at fixed intervals, TM2 must be cleared, when coincidence has taken place.

(b) Program [label: IITVL2] ... Refer to (h) Program list

(i) The count clock for 8-bit timer/counter 2 is set to $f_{CLK}/128$.

(ii) Clearing TM2 is enabled, when the TM2 contents coincide with those for CR21.

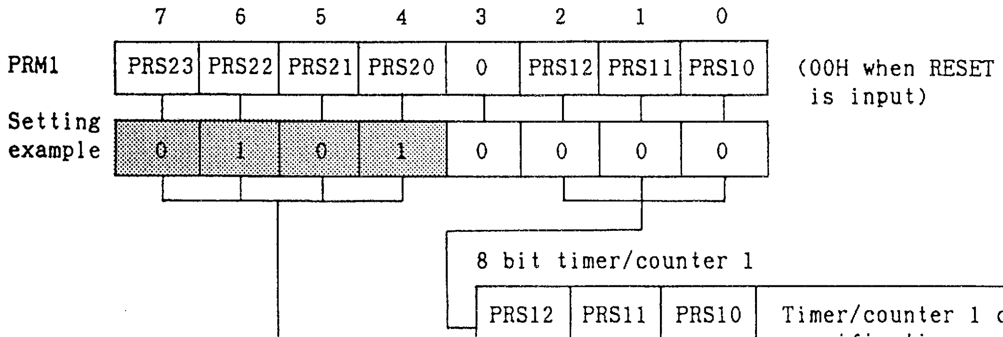
(iii) An interval time is set in CR21.

(iv) Interrupt request INTC21 is unmasked.

(v) The count operation for 8-bit timer register 2 (TM2) is enabled.

(c) Mode register setting

Prescaler mode register 1


7 6 5 4 3 2 1 0
PRM1
PRS23 PRS22 PRS21 PRS20 0 PRS12 PRS11 PRS10
(00H when RESET is input)
Setting example
0 1 0 1 0 0 0 0

8 bit timer/counter 1

PRS12	PRS11	PRS10	Timer/counter 1 count clock frequency specifications
0	0	0	$f_{CLK}/16$ (Note)
0	0	1	
0	1	0	
0	1	1	$f_{CLK}/32$
1	0	0	$f_{CLK}/64$
1	0	1	$f_{CLK}/128$
1	1	0	$f_{CLK}/256$
1	1	1	$f_{CLK}/512$

8 bit timer/counter 2

PRS23	PRS22	PRS21	PRS20	Timer/counter 3 count clock frequency specifications
0	0	0	0	$f_{CLK}/16$
0	0	0	1	
0	0	1	0	
0	0	1	1	$f_{CLK}/32$
0	1	0	0	$f_{CLK}/64$
0	1	0	1	$f_{CLK}/128$
0	1	1	0	$f_{CLK}/256$
0	1	1	1	$f_{CLK}/512$
1	1	1	1	External clock (CI)

Note: f_{CLK}: Internal system clock frequency ($f_{XX}/2$)

Capture/compare register control register 2

	7	6	5	4	3	2	1	0	
CRC2	MOD1	MOD0	CLR22	1	CLR21	0	0	0	(00H when RESET is input)
Setting example	0	0	0	1	1	0	0	0	

MOD1	MOD0	CLR22	CLR21	Timer output mode		TM2 clear operation
				TO2	TO3	
0	0	0	0	Toggle output	Toggle output	Not cleared
0	0	0	1	Toggle output	Toggle output	Cleared when TM2 and CR21 registers coincide
0	0	1	0	Toggle output	Toggle output	Cleared after capturing TM2 contents into CR22 register
0	0	1	1	Toggle output	Toggle output	Cleared when TM2 and CR21 registers coincide or are cleared after capturing TM2 contents into CR22 register
0	1	0	0	PWM output	Toggle output	Not cleared
1	0	0	0	PWM output	PWM output	Not cleared
1	1	0	1	PPG output	Toggle output	Cleared when TM2 and CR21 registers coincide

Note: No combination other than above is allowed.

Interrupt mask register H

	7	6	5	4	3	2	1	0	
MK0H	CSIMK	STMK	SRMK	SERMK	CMK20	PMK5	PMK4	CMK21	(FFH when RESET is input)
Setting example	x	x	x	x	x	x	x	0	

MK	Interrupt mask flag
0	Enables interrupt processing
1	Disables interrupt processing

Timer control register 1

	7	6	5	4	3	2	1	0	
TMC1	CE2	OVF2	CMD2	0	CE1	OVF1	0	0	(00H when RESET is input)
Setting example	1	0	0	0	0	0	0	0	

8-bit timer/counter 1

OVF1	TM1 overflow flag
0	No overflow
1	Overflow occurs (count up from FFH to 00H)

Remarks: This bit can be cleared by software only.

CE1	Controls TM1 count operation
0	Clears and stops count operation
1	Enables count operation

8-bit timer/counter 2

CMD2	Specifies TM2 operation mode
0	Ordinary mode
1	One-shot mode

OVF2	TM2 overflow flag
0	No overflow
1	Overflow occurs (count up from FFH to 00H)

Remarks: This bit can be cleared by software only.

CE2	Controls TM2 count operation
0	Clears and stops count operation
1	Enables count operation

(d) Input/output parameter

INTVL2: Sets a parameter that determines the interval time.

The count clock for the 8-bit timer/counter can be set to $f_{CLK}/16$, $f_{CLK}/32$, $f_{CLK}/64$, $f_{CLK}/128$, $f_{CLK}/256$, $f_{CLK}/512$, or the frequency of an external clock. In the example program presented in this section, it is fixed to $f_{CLK}/128$.

The interval time that is determined by the parameter INTVL2 value can be calculated by the following expression:

$$\text{Interval time} = \text{INTVL2} \times x/f_{CLK}$$

where,
(INTVL2: $0 \leqq \text{INTVL2} \leqq \text{FFH}$)
$x = 16, 32, 64, 128, 256, 512$

(e) Registers

No register is used.

(f) Program

An example program that generates interrupt request INTC21 at 3.2 ms intervals, where $f_{CLK} = 6$ MHz, is shown below.

Note that the count clock for 8-bit timer/counter 2 is set to $f_{CLK}/128$ in this program.

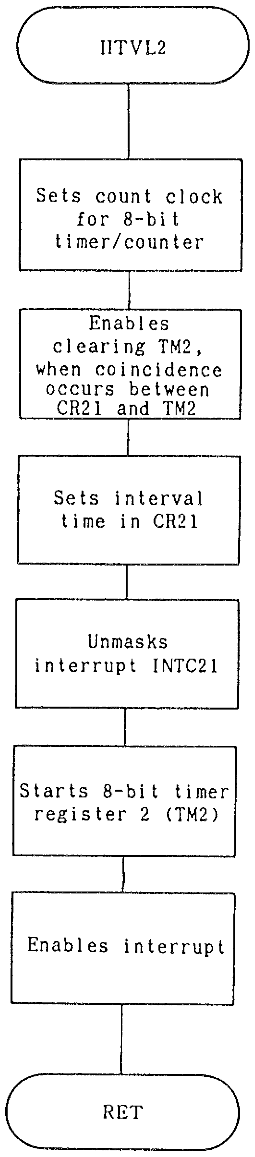
To generate the interrupt request every 3.2 ms, the input parameter INTVL2 value should be as follows:

$$\text{INTVL2} = \frac{3.2 \times 10^{-3}}{128/(6 \times 10^{6})} = 150$$

Therefore, program an interval timer by defining an external definition and external reference directives, as follows:

```
        PUBLIC  INTVL2          ; PARAMETER
        EXTRN   IITVL2          ; PACKAGE
                .
                .
INTVL2  EQU     150             ; PARAMETER FOR INTERVAL
                .
                .
        CALL    !IITVL2
                .
                .
```

(g) Flowchart


IITVL2
Sets count clock for 8-bit timer/counter
Enables clearing TM2, when coincidence occurs between CR21 and TM2
Sets interval time in CR21
Unmasks interrupt INTC21
Starts 8-bit timer register 2 (TM2)
Enables interrupt
RET

(h) Program list

```
        NAME    IITV2M
;
;*******************************************************
;* 8bit- Timer / Counter Unit-2                        *
;*      internal interval timer                        *
;*******************************************************
;
        PUBLIC  IITVL2
        EXTRN   INTVL2          ; Compare data for interval timer

CMK21   EQU     MK0H.0          ; INTC21 mask flag
;
        CSEG
IITVL2:
        MOV     PRM1,#01010000B ; select fclk/128 (TM2)
        MOV     CRC2,#00011000B ; clear enable TM2 by CR21
        MOV     CR21,#LOW(INTVL2-1)     ; set interval time
        CLR1    CMK21           ; open INTC21 mask
        MOV     TMC1,#10000000B ; timer start
        EI                      ; interrupt enable

        RET
;
        END
```

3.2 Programmable Rectangular Pulse Output

To output programmable rectangular pulses, basically the function of the interval timer is used. The signal output to an external device is inverted by an interrupt request signal. Therefore, the duty factor for the output pulse is 50%.
This section presents an example program that outputs rectangular pulses through the TO1 pin by using a 16-bit timer/counter.
When a 16-bit timer/counter unit is used, rectangular pulses, having a 2.6 microsecond resolution, can be output at a 2.6 microseconds to 174.8 milliseconds cycle, where f_{CLK} = 6 MHz.

(1) Operation

Figure 3-5 is a blockdiagram that illustrates how the rectangular pulse is output from the TO1 pin.

Fig. 3-5 Rectangular Pulse Output from TO1 Pin

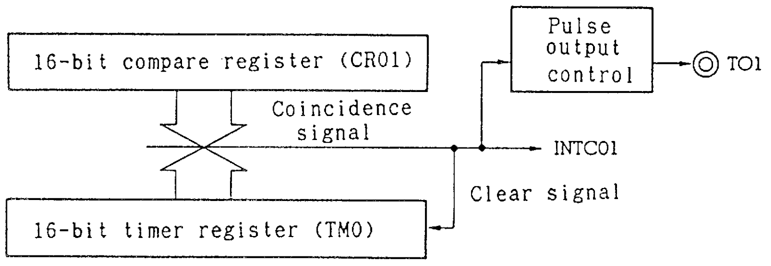
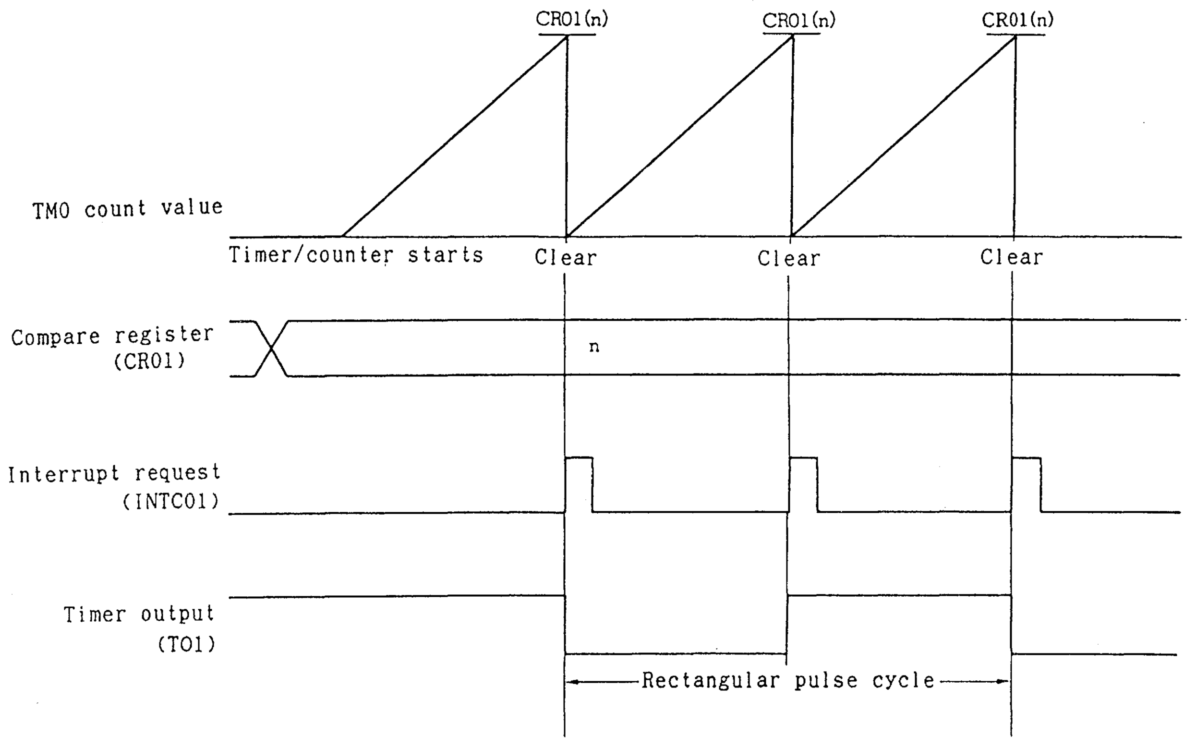


Fig. 3-6 Timing Chart



Rectangular pulse cycle = (n + 1) x 2 x 8/f_{CLK}

where,

{n: 0 ≦ n ≦ FFFFH}

Interrupt request INTC01 is generated, when the 16-bit timer register TM0 contents coincide with those for 16-bit compare register CR01. To generate the interrupt request at fixed intervals, therefore, TM0 must be cleared, when coincidence has taken place.

Since the timer output is inverted by the interrupt request, the rectangular pulse cycle must be two times that output by the interval timer discussed in (1) in 3.1.1.

(2) Program [label: TOUT] ... Refer to (8) Program list

(a) The TO1 pin active level is set to low, and the timer output is enabled.

(b) Since pin 35 is used as the TO1 pin, it is specified as a control port.

(c) Clearing 16-bit timer register (TM0) is enabled, when the TM0 contents coincide with those for 16-bit compare register CR01.

(d) An interval cycle is set to CR01.

(e) The TM0 count operation is enabled.

(3) Mode register setting

Timer output control register

	7	6	5	4	3	2	1	0
TOC	ENTO3	ALV3	ENTO2	ALV2	ENTO1	ALV1	ENTO0	ALV0
Setting example	0	0	0	0	1	0	0	0

(00H when RESET is input)

ALV0	Specifies TO0 pin active level
0	Low level
1	High level

ENTO0	Specifies TO0 pin operation
0	Outputs $\overline{\text{ALV0}}$
1	Enables pulse output

ALV1	Specifies TO1 pin active level
0	Low level
1	High level

ENTO1	Specifies TO1 pin operation
0	Outputs $\overline{\text{ALV1}}$
1	Enables pulse output

ALV2	Specifies TO2 pin active level
0	Low level
1	High level

(Cont'd)

ENTO2	Specifies TO2 pin operation
0	Outputs $\overline{\text{ALV2}}$
1	Enables pulse output

ALV3	Specifies TO3 pin active level
0	Low level
1	High level

ENTO3	Specifies TO3 pin operation
0	Output $\overline{\text{ALV3}}$
1	Enables pulse output

Port 3 mode control register

	7	6	5	4	3	2	1	0
PMC3	PMC37	PMC36	PMC35	PMC34	PMC33	PMC32	PMC31	PMC30
Setting example	0	0	1	0	0	0	0	0

(00H when RESET is input)

PMC30	Specifies P30 pin control mode
0	I/O port mode
1	RxD input mode

PMC31	Specifies P31 pin control mode
0	I/O port mode
1	TxD output mode

PMC32	Specifies P32 pin control mode
0	I/O port mode
1	$\overline{\text{SCK}}$ output mode

PMC33	Specifies P33 pin control mode
0	I/O port mode
1	SB0 output mode/SO output mode

PMC3n	Specifies P3n pin control mode (n = 4 to 7)
0	I/O port mode
1	TOm output mode (m = 0 to 3)

Capture/compare control register 0

	7	6	5	4	3	2	1	0	
CRC0	MOD1	MOD0	0	1	CLR01	0	0	0	(10H when RESET is input)
Setting example	0	0	0	1	1	0	0	0	

MOD1	MOD0	CLR01	Timer output mode specification		TM0 clear operation when TM0=CR01
			TO0	TO1	
0	0	0	Toggle output	Toggle output	Disabled
0	0	1	Toggle output	Toggle output	Enabled
0	1	0	PWM output	Toggle output	Disabled
0	1	1	Not allowed		
1	0	0	PWM output	PWM output	Disabled
1	0	1	Not allowed		
1	1	0	Not allowed		
1	1	1	PPG output	Toggle output	Enabled

Timer control register 0

	7	6	5	4	3	2	1	0	
TMC0	CE	0	0	0	CE0	OVF0	0	0	(00H when RESET is input)
Setting example	0	0	0	0	1	0	0	0	

16-bit timer/counter

OVF0	TM0 overflow flag
0	No overflow
1	Overflow occurs (count up from FFFFH to 0000H)

Remarks: This bit is cleared by software only.

CE0	Controls TM0 count operation
0	Clears and stops count operation
1	Enables count operation

8-bit timer/counter 3

CE	Controls TM3 count operation
0	Clears and stops count operation
1	Enables count operation

(4) Input/output parameter

INTVL3: Sets a parameter that determines the output rectangular pulse cycle.

Since the count clock for the 16-bit timer/counter is fixed to $f_{CLK}/8$, the cycle for the rectangular pulse, that is determined by the value set as parameter INTVL3, can be calculated by the following expression:

Rectangular pulse cycle: INTVL3 x 2 x $8/f_{CLK}$
where,
{INTVL3: $0 \leqq \text{INTVL3} \leqq \text{FFFFH}$}

(5) Registers

No register is used.

(6) Program

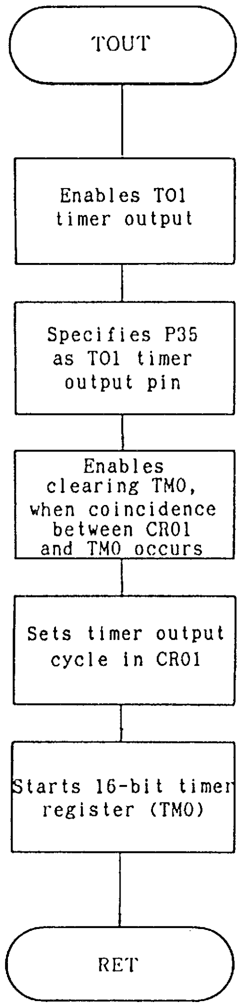
The example program shown below outputs rectangular pulses at a 1 kHz frequency.
This program generates interrupt request INTC01 at 500 microsecond (2 kHz) intervals. Therefore, where $f_{CLK} = 6$ MHz, the input parameter INTVL3 value is as follows:

$$\text{INTVL3} = \frac{500 \times 10^{-6}}{8/(6 \times 10^{6})} = 375$$

Program example is :

```
        PUBLIC  INTVL3          ; PARAMETER
        EXTRN   TOUT            ; PACKAGE
                .
                .
INTVL3  EQU     375             ; PARAMETER FOR INTERVAL
                .
                .
        CALL    !TOUT
```

(7) Flowchart



TOUT
Enables TO1 timer output
Specifies P35 as TO1 timer output pin
Enables clearing TM0, when coincidence between CR01 and TM0 occurs
Sets timer output cycle in CR01
Starts 16-bit timer register (TM0)
RET

(8) Program list

```
        NAME    TOUTM
;
;*******************************************************
;* programable pulse output                            *
;*******************************************************
;
        PUBLIC  TOUT
        EXTRN   INTVL3          ; timer output frequency
;
        CSEG
TOUT:
        MOV     TOC,#00001000B  ; enable T01 timer output
        MOV     PMC3,#00100000B ; P35=control port
        MOV     CRC0,#00011000B ; clear enable TM0 by CR01
        MOVW    CR01,#INTVL3-1  ; set interval time
        MOV     TMC0,#00001000B ; timer start

        RET
;
        END
```

3.3 Free-running Interval Timer

The free-running interval timer generates an interval by free-running the timer and adding a certain value to the compare register during an interrupt processing.
In the following example, two compare registers are used to generate two different intervals for one timer. Two different timer outputs are made by these two intervals (timer output duty is 50%).

(1) Program example using the 16-bit timer/counter

In this program example, The INTC00 and INTC01 interrupt sources are used and two different timer outputs are made from the TO0 and TO1 pins.

(a) Operational outline

Figure 3-7 shows a blockdiagram for making timer outputs from the TO0 and TO1 pins.

Fig. 3-7 Timer Outputs from TO0/TO1 Pins

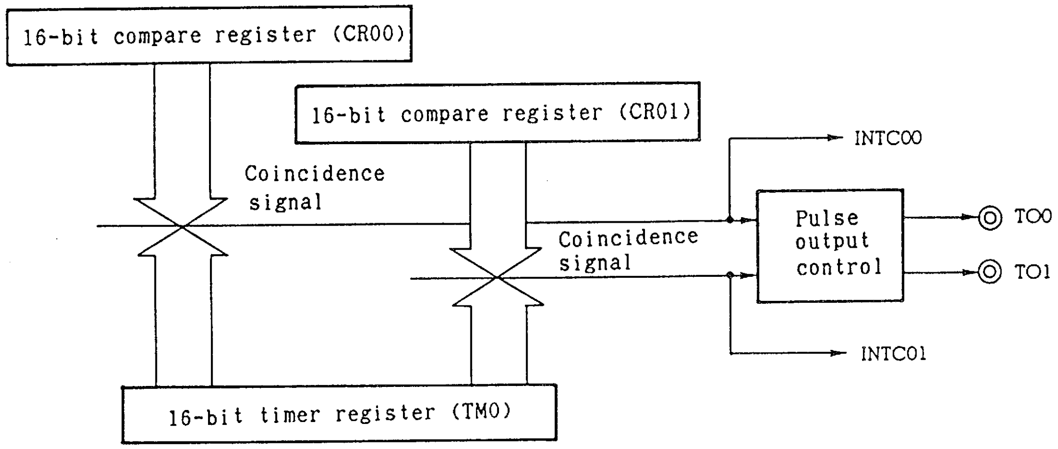
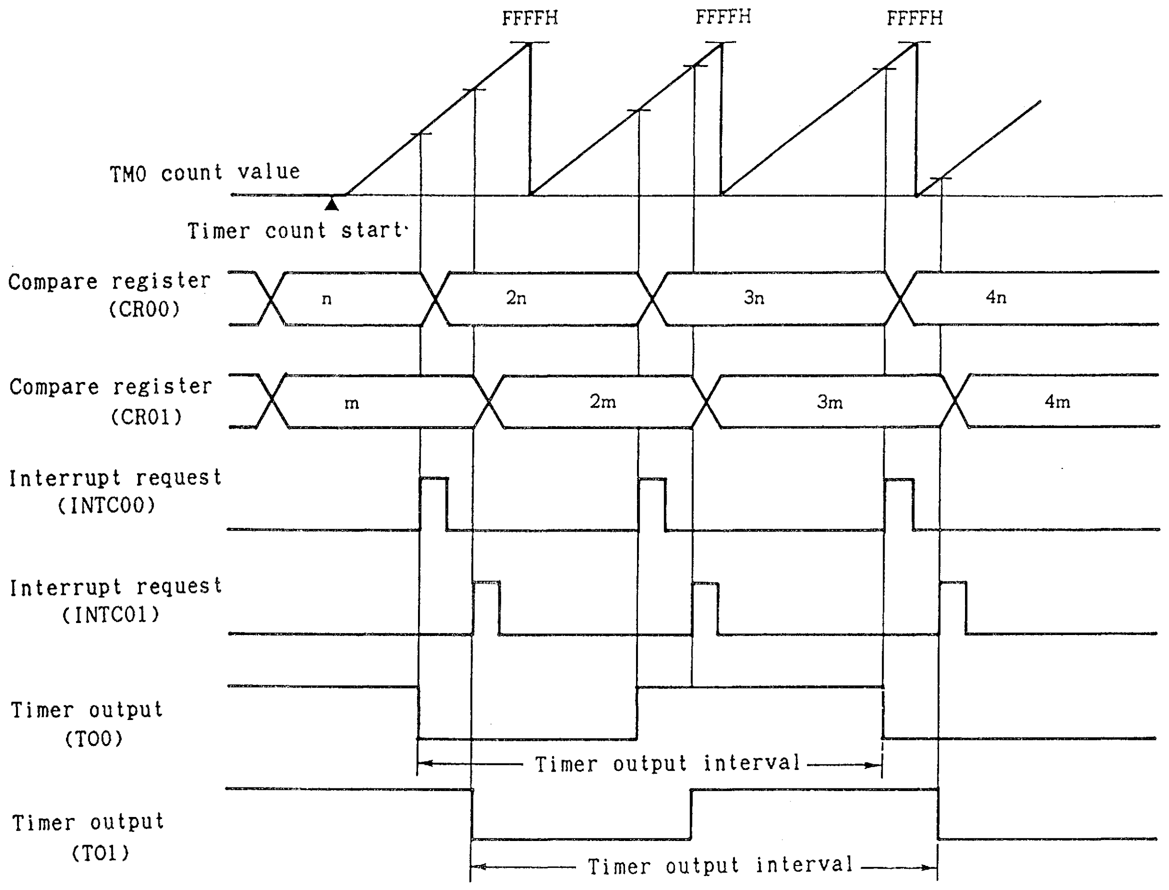

16-bit compare register (CR00)
16-bit compare register (CR01)
Coincidence signal
Coincidence signal
INTC00
Pulse output control
TO0
TO1
INTC01
16-bit timer register (TM0)

Fig. 3-8 Timing Chart for Outputting Timer from TO0/TO1 Pins



Timer output interval

$= n\ (m) \times 2 \times 8/f_{CLK}$ $\{n\ (m): 1 \leqq n\ (m) \leqq \text{FFFFH}\}$

When generating two intervals of different periods, the timer must be free run first. Afterwards, two timer outputs, having a certain interval, can be generated by adding a value corresponding to each interval period to the current compare register (CR00, CR01) value, when the INTC00/INTC01 interrupt is generated.

(b) Program description

Refer to (h), "Program List".

(i) Initialization processing [label name: FRUN0]

① Sets the active level for TO0 and TO1 timer outputs to low level, and enables timer output.
② Specifies P34 and P35 as control ports, so that they can be used as the TO0 and TO1 output pins.
③ Disables clearing the 16-bit timer register (TM0) by coincidence with the 16-bit compare register (CR00, CR01) (free-running mode).
④ Sets each interval to the 16-bit compare register (CR00, CR01).
⑤ Releases masking for INTC00 and INTC01 interrupt requests.
⑥ Enables count operation for 16-bit timer/counter.

(ii) INTC00 interrupt processing [label name: INTC00]

① Selects register bank 1.
② Adds the value corresponding to the interval period to the current compare register (CR00) value. Ignores overflow caused by the addition.
③ Sets the sum to the compare register (CR00).

(iii) INTC01 interrupt processing [label name: INTC01]

① Selects register bank 1.

② Adds the value corresponding to the interval period to the current compare register (CR01) value. Ignores overflow caused by the addition.

③ Sets the sum to the compare register (CR01).

(c) Mode register setting examples

Timer output control register

	7	6	5	4	3	2	1	0	
TOC	ENT03	ALV3	ENT02	ALV2	ENT01	ALV1	ENT00	ALV0	(00H when RESET is input)
Setting example	0	0	0	0	1	0	1	0	

ALV0	TO0 pin active level specification
0	Low level
1	High level

ENT00	TO0 pin operation specification
0	$\overline{\text{ALV0}}$ output
1	Enables pulse output

ALV1	TO1 pin active level specification
0	Low level
1	High level

(Cont'd)

ENT01	TO1 pin operation specification
0	$\overline{\text{ALV1}}$ output
1	Enables pulse output

ALV2	TO2 pin active level specification
0	Low level
1	High level

ENT02	TO2 pin operation specification
0	$\overline{\text{ALV2}}$ output
1	Enables pulse output

ALV3	TO3 pin active level specification
0	Low level
1	High level

ENT03	TO3 pin operation specification
0	$\overline{\text{ALV3}}$ output
1	Enables pulse output

Port 3 mode control register

	7	6	5	4	3	2	1	0	
PMC3	PMC37	PMC36	PMC35	PMC34	PMC33	PMC32	PMC31	PMC30	(00H when RESET is input)
Setting example	0	0	1	1	0	0	0	0	

PMC30	P30 pin control mode specification
0	Input/output port mode
1	RxD input mode

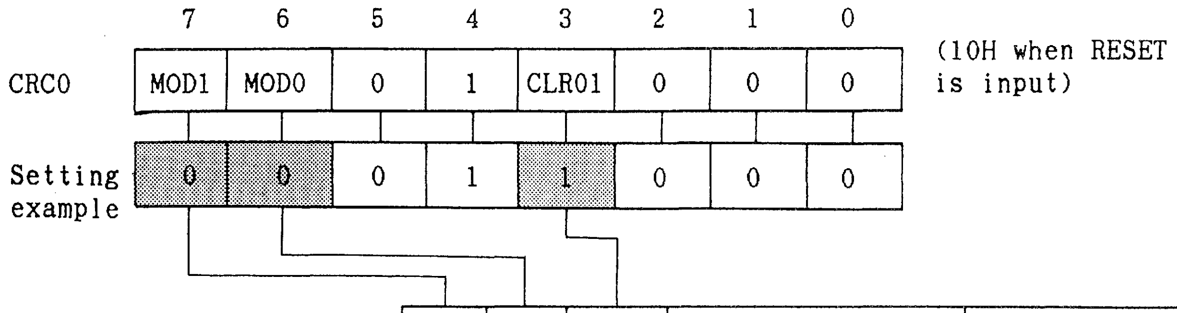
PMC31	P31 pin control mode specification
0	Input/output port mode
1	TxD output mode

PMC32	P32 pin control mode specification
0	Input/output port mode
1	SCK input/output mode

PMC33	P33 pin control mode specification
0	Input/output port mode
1	SO output/SB0 input/output mode

PMC3n	PM3n pin control mode specification (n = 4 to 7)
0	Input/output port mode
1	TOm output mode (m = 0 to 3)

Capture/compare control register 0



MOD1	MOD0	CLR01	Timer output mode specification		TM0 clear operation when TM0=CR01
			TO0	TO1	
0	0	0	Toggle output	Toggle output	Disabled
0	0	1	Toggle output	Toggle output	Enabled
0	1	0	PWM output	Toggle output	Disabled
0	1	1	Not allowed		
1	0	0	PWM output	PWM output	Disabled
1	0	1	Not allowed		
1	1	0	Not allowed		
1	1	1	PPG output	Toggle output	Enabled

Interrupt mask register L

	7	6	5	4	3	2	1	0	
MK0L	CMK11	CMK10	CMK01	CMK00	PMK3	PMK2	PMK1	PMK0	(FFH when RESET is input)
Setting example	x	x	0	0	x	x	x	x	x: Not manipulated

MK	Interrupt mask flag
0	Enable interrupt processing
1	Retains interrupt processing

Timer control register 0

	7	6	5	4	3	2	1	0	
TMC0	CE3	0	0	0	CE0	OVF0	0	0	(00H when RESET is input)
Setting example	0	0	0	0	1	0	0	0	

16-bit timer/counter

OVF0	Timer/counter overflow flag
0	No overflow
1	Overflow (count up from FFFFH to 0000H)

Remarks: This bit is reset only by the software.

CE0	Timer/counter 0 count operation control
0	Stops count operation with timer/counter cleared
1	Enables count operation

8-bit timer/counter 3

CE3	Timer/counter 3 count operation control
0	Stops count operation with timer/counter cleared
1	Enables count operation

(d) Input parameters

INTVL4: This value determines the interval of the timer output from the TO0 pin. This value is added to the current compare register (CR00) value in the interrupt processing, each time an interrupt is generated.

INTVL5: This value determines the interval for the timer output from the TO1 pin. This value is added to the current compare register (CR01) value in the interrupt processing, each time an interrupt is generated.

The count clock of the 16-bit timer/counter is fixed to $f_{CLK}/8$. Therefore, the timer output interval for the values set to INTVL4 and INTVL5 can be obtained by the following expression:

Timer output interval
$= \text{INTVL4 (INTVL5)} \times 2 \times 8/f_{CLK}$
$\{\text{INTVL4 (INTVL5)} : 0 \leqq \text{INTVL4 (INTVL5)} \leqq \text{FFFFH}\}$

(e) Registers used

AX (register bank 1)

(f) Program example

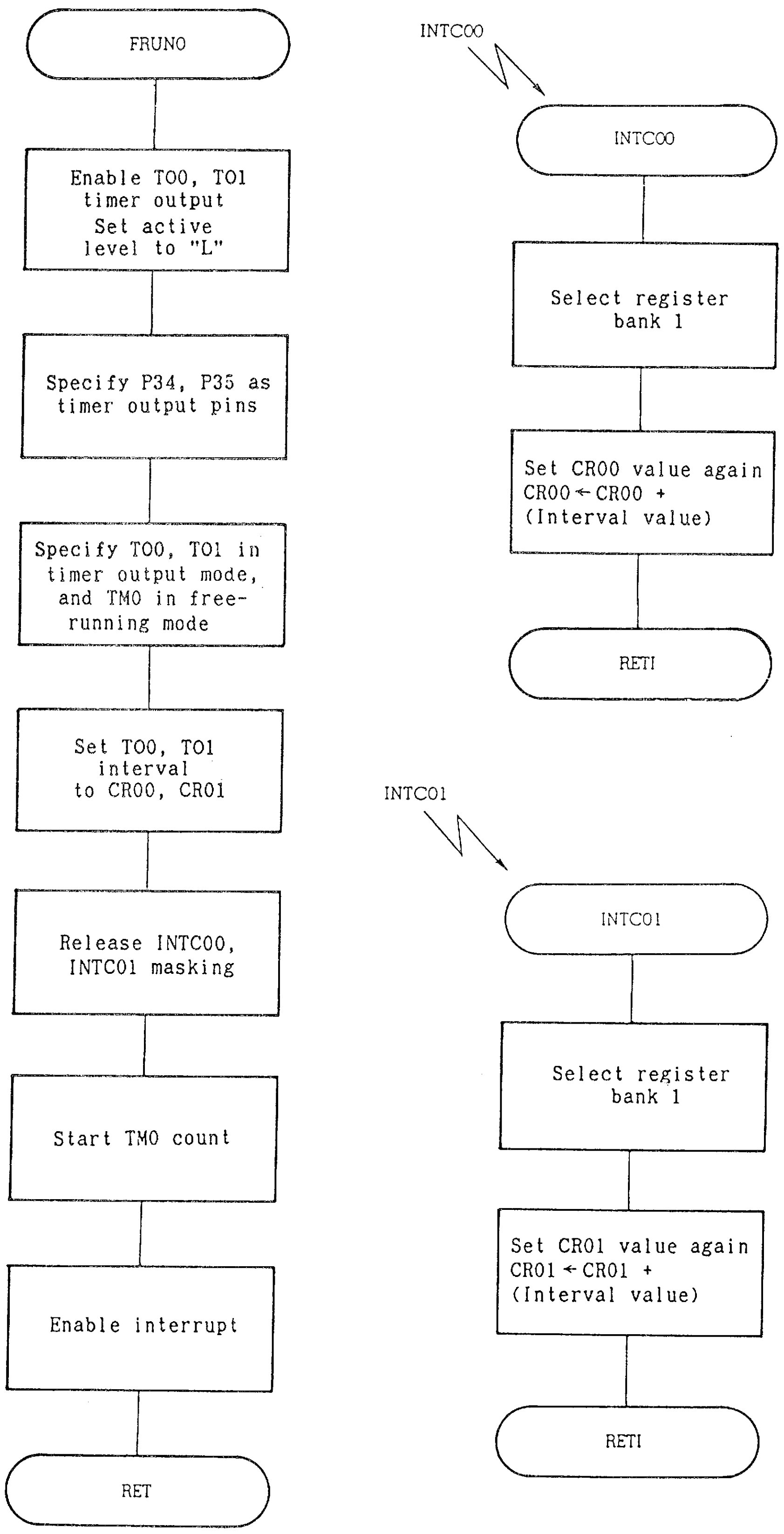
The following program example is to output 500 Hz timer from the TO0 pin, and 1kHz timer from the TO1 pin. In this case, INTC00 is generated every 1ms, and INTC01 is generated every 500 us.
If f_{CLK} is 6 MHz, the values for input parameters INTVL4 and INTVL5 will be as follows:

$$\text{INTVL4} = \frac{1 \times 10^{-3}}{8/(6 \times 10^{6})}$$
$$= 750$$
$$\text{INTVL5} = \frac{500 \times 10^{-6}}{8/(6 \times 10^{6})}$$
$$= 375$$

The following shows program example.

```
        PUBLIC  INTVL4,INTVL5   ; PARAMETER
        EXTRN   FRUNO           ; PACKAGE
                .
                .
INTVL4  EQU     750             ; free running timer INTC00 compare data
INTVL5  EQU     375             ; free running timer INTC01 compare data
                .
                .
        CALL    !FRUNO
                .
                .
```

(g) Flowchart


FRUNO
Enable T00, T01 timer output Set active level to "L"
Specify P34, P35 as timer output pins
Specify T00, T01 in timer output mode, and TM0 in free-running mode
Set T00, T01 interval to CR00, CR01
Release INTC00, INTC01 masking
Start TM0 count
Enable interrupt
RET
INTC00
INTC00
Select register bank 1
Set CR00 value again CR00 ← CR00 + (Interval value)
RETI
INTC01
INTC01
Select register bank 1
Set CR01 value again CR01 ← CR01 + (Interval value)
RETI

(h) Program list

```
        NAME    F_RUNO
;
;*********************************************************
;* 16bit-Timer / Counter                                 *
;*      free running interval timer                      *
;*********************************************************
;
        PUBLIC  FRUNO
        EXTRN   INTVL4,INTVL5

CMK00   EQU     MK0L.4          ; INTC00 mask flag
CMK01   EQU     MK0L.5          ; INTC01 mask flag
;
INTC00VT CSEG   AT 00014H
        DW      INTC00          ; INTC00
INTC01VT CSEG   AT 00016H
        DW      INTC01          ; INTC01
;
        CSEG
FRUNO:
        MOV     TOC,#00001010B  ; timer output,active level low
        MOV     PMC3,#00110000B ; P3 control port
        MOV     CRC0,#00010000B ; set timer free running mode
        MOVW    CR00,#INTVL4    ; set interval time
        MOVW    CR01,#INTVL5    ; set interval time
        CLR1    CMK00           ; open INTC00 mask
        CLR1    CMK01           ; open INTC01 mask
        MOV     TMC0,#00001000B ; start timer
        EI                      ; interrupt enable

        RET
;
;*********************************************************
;*              INTC00 interrupt routine                 *
;*********************************************************
INTC00:
        SEL     RB1             ; register bank change
        MOVW    AX,CR00
        ADDW    AX,#INTVL4      ; count INTC00 compare data
        MOVW    CR00,AX
        RETI
;
;*********************************************************
;*              INTC01 interrupt routine                 *
;*********************************************************
INTC01:
        SEL     RB1             ; register bank change
        MOVW    AX,CR01
        ADDW    AX,#INTVL5      ; count INTC01 compare data
        MOVW    CR01,AX
        RETI
;
        END
```

(2) Program example using 8-bit timer/counter 2

In this program example, INTC20 and INTC21 interrupt sources are used and two different timer outputs are made from the TO2 and TO3 pins.

(a) Operational outline

Figure 3-9 shows a blockdiagram for making timer outputs from the TO2 and TO3 pins.

Fig. 3-9 Timer Outputs from TO2/TO3 Pins

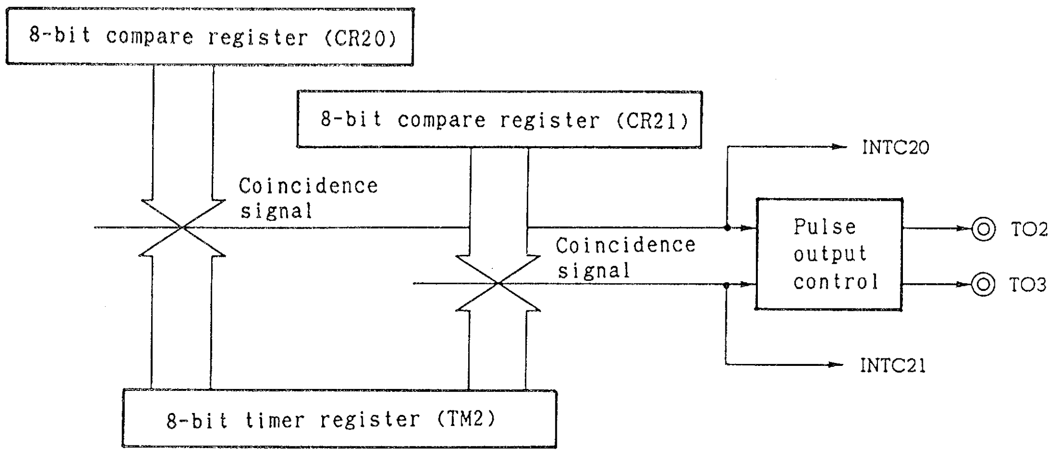
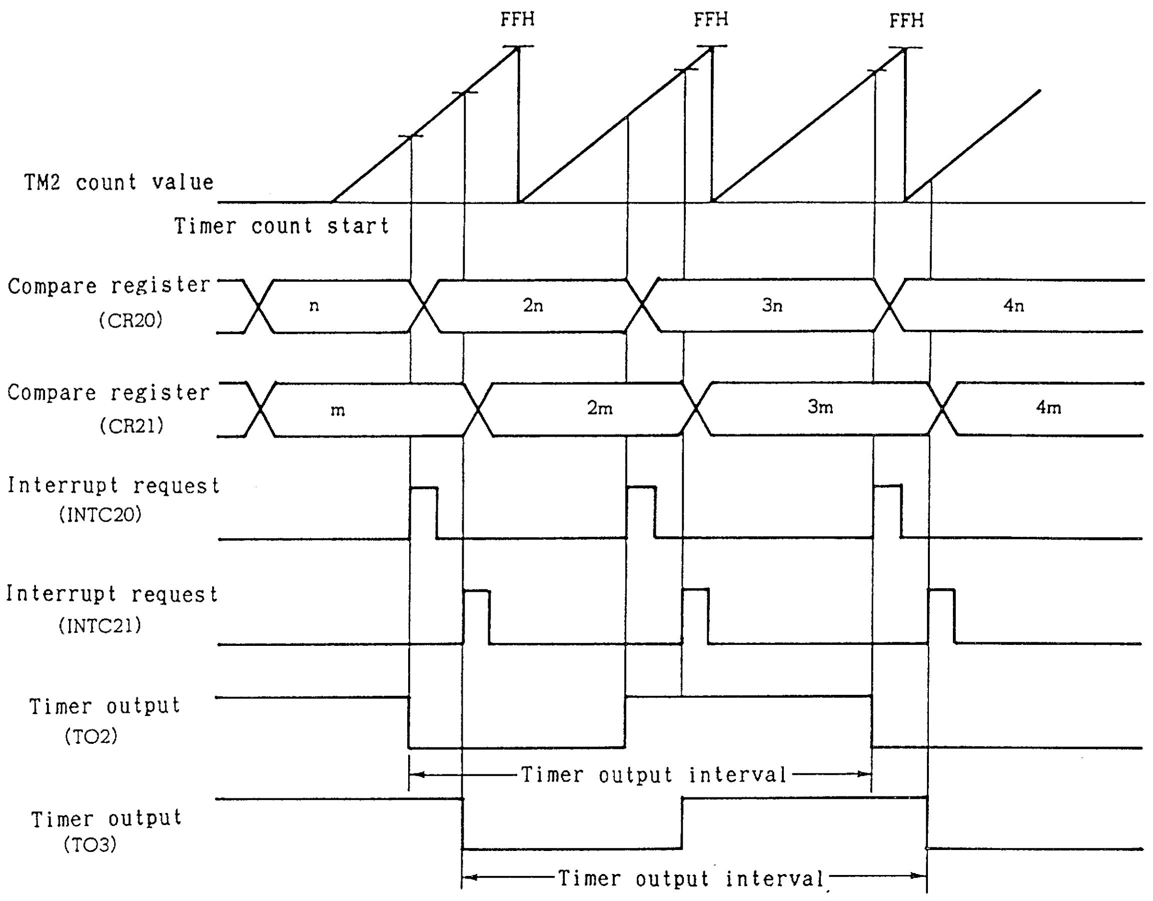

8-bit compare register (CR20)
8-bit compare register (CR21)
Coincidence signal
Coincidence signal
INTC20
Pulse output control
TO2
TO3
INTC21
8-bit timer register (TM2)

Fig. 3-10 Timing Chart for Outputting Timer from TO2/TO3 Pins



Timer output interval

$$= n\ (m) \times 2 \times X/f_{CLK} \quad \{n\ (m) : 1 \leqq n\ (m) \leqq FFH\}$$

$$X = 16,\ 32,\ 64,\ 128,\ 256,\ 512$$

When generating two intervals with different periods, the timer must be free run first. Afterwards, two timer outputs having a certain interval can be generated by adding a value, corresponding to each interval period, to the current compare register (CR20, CR21) value, when the INTC20/INTC21 interrupt is generated.

(b) Program description

Refer to (h), "Program List".

(i) Initialization processing [label name: FRUN2]

① Sets the active level for TO2 and TO3 timer outputs to low level, and enables timer output.
② Specifies P36 and P37 as control ports, so that they can be used as the TO2 and TO3 output pins.
③ Disables clearing the 8-bit timer register (TM2) by coincidence with the 8-bit compare register (CR20, CR21) (free-running mode).
④ Sets the count clock for 8-bit timer/counter 2 to $f_{CLK}/16$.
⑤ Sets each interval to the 8-bit compare register (CR20, CR21).
⑥ Releases masking of INTC20 and INTC21 interrupt requests.
⑦ Enables count operation for 8-bit timer/counter 2.

(ii) INTC20 interrupt processing [label name: INTC20]

① Adds the value corresponding to the interval period to the compare register (CR20). Ignores overflow caused by the addition.

(iii) INTC21 interrupt processing [label name: INTC21]

① Adds the value corresponding to the interval period to the compare register (CR21). Ignores overflow caused by the addition.

(c) Mode register setting examples

Timer output control register

	7	6	5	4	3	2	1	0	
TOC	ENT03	ALV3	ENT02	ALV2	ENT01	ALV1	ENT00	ALV0	(00H when RESET is input)
Setting example	1	0	1	0	0	0	0	0	

ALV0	TO0 pin active level specification
0	Low level
1	High level

ENT00	TO0 pin operation specification
0	$\overline{\text{ALV0}}$ output
1	Enables pulse output

ALV1	TO1 pin active level specification
0	Low level
1	High level

ENT01	TO1 pin operation specification
0	$\overline{\text{ALV1}}$ output
1	Enables pulse output

ALV2	TO2 pin active level specification
0	Low level
1	High level

(Cont'd)

ENT02	TO2 pin operation specification
0	$\overline{ALV2}$ output
1	Enables pulse output

ALV3	TO3 pin active level specification
0	Low level
1	High level

ENT03	TO3 pin operation specification
0	$\overline{ALV3}$ output
1	Enables pulse output

Port 3 mode control register

	7	6	5	4	3	2	1	0	
PMC3	PMC37	PMC36	PMC35	PMC34	PMC33	PMC32	PMC31	PMC30	(00H when reset)
Setting example	1	1	0	0	0	0	0	0	

PMC30	P30 pin control mode specification
0	Input/output port mode
1	RxD input mode

PMC31	P31 pin control mode specification
0	Input/output port mode
1	TxD output mode

PMC32	P32 pin control mode specification
0	Input/output port mode
1	$\overline{\text{SCK}}$ input/output mode

PMC33	P33 pin control mode specification
0	Input/output port mode
1	SO output/SB0 input/output mode

PMC3n	PM3n pin control mode specification (n = 4 to 7)
0	Input/output port mode
1	TOm output mode (m = 0 to 3)

Capture/compare control register 2

	7	6	5	4	3	2	1	0	
CRC2	MOD1	MOD0	CLR22	1	CLR21	0	0	0	(00H when RESET is input)
Setting example	0	0	0	1	0	0	0	0	

MOD1	MOD0	CLR22	CLR21	Timer output mode		TM2 clear operation
				TO2	TO3	
0	0	0	0	Toggle output	Toggle output	Not cleared
0	0	0	1	Toggle output	Toggle output	Cleared when TM2 and CR21 registers coincide
0	0	1	0	Toggle output	Toggle output	Cleared after capturing TM2 contents into CR22 register
0	0	1	1	Toggle output	Toggle output	Cleared when TM2 and CR21 registers coincide or are cleared after capturing TM2 contents into CR22 register
0	1	0	0	PWM output	Toggle output	Not cleared
1	0	0	0	PWM output	PWM output	Not cleared
1	1	0	1	PPG output	Toggle output	Cleared when TM2 and CR21 registers coincide

Note: No combination other than above is allowed.

Prescaler mode register 1

	7	6	5	4	3	2	1	0	
PRM1	PRS23	PRS22	PRS21	PRS20	0	PRS12	PRS11	PRS10	(00H when reset)
Setting example	▨ 0	▨ 0	▨ 0	0	0	0	0	0	

8-bit timer/counter 1

PRS12	PRS11	PRS10	Timer/counter 1 count clock frequency specification
0	0	0	$f_{CLK}/16$ (Note)
0	0	1	
0	1	0	
0	1	1	$f_{CLK}/32$
1	0	0	$f_{CLK}/64$
1	0	1	$f_{CLK}/128$
1	1	0	$f_{CLK}/256$
1	1	1	$f_{CLK}/512$

(Cont'd)

8-bit timer/counter 2

PRS23	PRS22	PRS21	PRS20	Timer/counter 3 count clock frequency specification
0	0	0	0	$f_{CLK}/16$
0	0	0	1	
0	0	1	0	
0	0	1	1	$f_{CLK}/32$
0	1	0	0	$f_{CLK}/64$
0	1	0	1	$f_{CLK}/128$
0	1	1	0	$f_{CLK}/256$
0	1	1	1	$f_{CLK}/512$
1	1	1	1	External clock (CI)

Note: f_{CLK}: Internal system clock frequency ($f_{XX}/2$)

Interrupt mask register

	7	6	5	4	3	2	1	0	
MK0H	CSIMK	STMK	SRMK	SERMK	CMK20	PMK5	PMK4	CMK21	(FFH when reset)
Setting example	x	x	x	x	0	x	x	0	x: Not manipulated

MK	Interrupt mask flag
0	Enables interrupt processing
1	Retains interrupt processing

Timer control register 1

	7	6	5	4	3	2	1	0	
TMC1	CE2	OVF2	CMD2	0	CE1	OVF1	0	0	(00H when reset)
Setting example	1	0	0	0	0	0	0	0	

8-bit timer/counter 1

OVF1	Timer/counter 1 overflow flag
0	No overflow
1	Overflow (count up from FFH to 00H)

Remarks: This bit is reset only by the software.

CE1	Timer/counter 1 count operation control
0	Stops count operation with timer/counter cleared
1	Enables count operation

8-bit timer/counter 2

CMD2	Timer/counter 2 operation mode specification
0	Normal mode
1	One-shot mode

OVF2	Timer/counter 2 overflow flag
0	No overflow
1	Overflow (count up from FFH to 00H)

Remarks: This bit is reset only by the software.

(Cont'd)

CE2	Timer/counter 2 count operation control
0	Stops count operation with timer/counter cleared
1	Enables count operation

(d) Input parameters

(i) INTVL6: This value determines the interval for the timer output from the TO2 pin. This value is added to the current compare register (CR20) value in the interrupt processing, each time an interrupt is generated.

(ii) INTVL7: This value determines the interval for the timer output from the TO3 pin. This value is added to the current compare register (CR21) value in the interrupt processing, each time an interrupt is generated.

The count clock of the 8-bit timer/counter 2 can be selected from $f_{CLK}/16$, $f_{CLK}/32$, $f_{CLK}/64$, $f_{CLK}/128$, $f_{CLK}/256$, $f_{CLK}/512$, and the external clock. In this program example, $f_{CLK}/16$ is selected. The timer output interval for the values set to INTVL6 and INTVL7 can be obtained by the following expression:

```
Timer output interval
   = INTVL6 (INTVL7) x 2 x 16/fCLK
     {INTVL6 (INTVL7) : 0 ≦ INTVL6 (INTVL7) ≦ FFH}
```

(e) Registers used

None

(f) Program example

The following program example is to output 1.5 kHz timer from the TO2 pin, and 2.5 kHz timer from the TO3 pin. In this case, INTC20 is generated every 333 us, and INTC21 every 200 us.

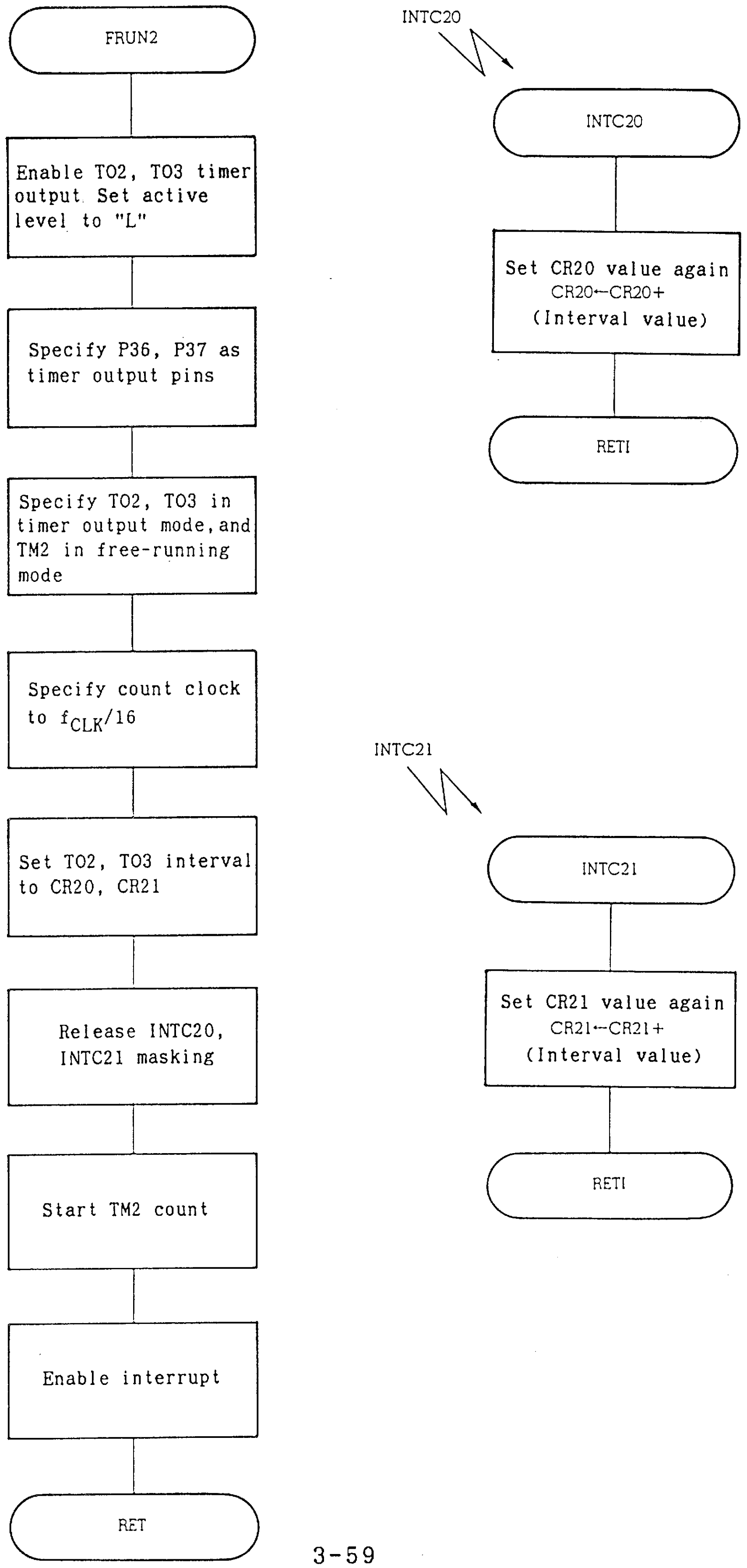
If f_{CLK} is 6 MHz, and the count clock is $f_{CLK}/16$, the values of input parameters INTVL6 and INTVL7 will be as follows:

$$\boxed{\begin{aligned} \text{INTVL6} &= \frac{333 \times 10^{-6}}{16/(6 \times 10^{6})} \\ &= 125 \\ \text{INTVL7} &= \frac{200 \times 10^{-6}}{16/(6 \times 10^{6})} \\ &= 75 \end{aligned}}$$

The following shows a program example.

```
        PUBLIC  INTVL6,INTVL7    ; PARAMETER
        EXTRN   FRUN2            ; PACKAGE
                .
                .
INTVL6  EQU     125              ; free running timer INTC20 compare data
INTVL7  EQU     75               ; free running timer INTC21 compare data
                .
                .
        CALL    !FRUN2
                .
                .
```

(g) Flow chart


FRUN2
Enable TO2, TO3 timer output. Set active level to "L"
Specify P36, P37 as timer output pins
Specify TO2, TO3 in timer output mode, and TM2 in free-running mode
Specify count clock to $f_{CLK}/16$
Set TO2, TO3 interval to CR20, CR21
Release INTC20, INTC21 masking
Start TM2 count
Enable interrupt
RET
INTC20
INTC20
Set CR20 value again
CR20←CR20+
(Interval value)
RETI
INTC21
INTC21
Set CR21 value again
CR21←CR21+
(Interval value)
RETI

(h) Program list

```
        NAME    F_RUN2
;
;**********************************************************
;* 8bit-Timer / Counter_2                                 *
;*      free running interval timer                       *
;**********************************************************
;
        PUBLIC  FRUN2
        EXTRN   INTVL6,INTVL7

CMK20   EQU     MK0H.3           ; INTC20 mask flag
CMK21   EQU     MK0H.0           ; INTC21 mask flag
;
INTC20VT CSEG   AT 00012H
        DW      INTC20           ; INTC20
INTC21VT CSEG   AT 0001CH
        DW      INTC21           ; INTC21
;
        CSEG
FRUN2:
        MOV     TOC,#10100000B  ; timer output,active level low
        MOV     PMC3,#11000000B ; P3 control port
        MOV     CRC2,#00010000B ; timer free running mode
        MOV     PRM1,#00000000B ; set prescaler fclk/i6
        MOV     CR20,#LOW(INTVL6)       ; set interval time
        MOV     CR21,#LOW(INTVL7)       ; set interval time
        CLR1    CMK20           ; open INTC20 mask
        CLR1    CMK21           ; open INTC21 mask
        MOV     TMC1,#10000000B ; start timer
        EI                      ; interrupt enable
        RET
;
;**********************************************************
;*              INTC20 interrupt routine                  *
;**********************************************************
INTC20:
        ADD     CR20,#LOW(INTVL6)  ; count INTC20 compare data
        RETI
;
;**********************************************************
;*              INTC21 interrupt routine                  *
;**********************************************************
INTC21:
        ADD     CR21,#LOW(INTVL7)  ; count INTC21 compare data
        RETI
;
        END
```

3.4 PWM/PPG Output

The uPD78214 series, 78218A series, 78234 series and 78244 series are provided with a PWM/PPG output function, which outputs variable duty square-wave using a timer interrupt request.
In PWM output function, one interval is a period, during which the timer full-counts. In PPG output function, the width of one interval is determined by one of two coincidence signals generated by one timer.

(1) PWM output program example, using 16-bit timer/counter

The following program example is to output PWM wave (from TO0 pin) whose pulse width is determined by the INTC00 interrupt request.
When changing the duty, set the value to determine the duty in the work area in the RAM and call the subroutine to change the duty.

(a) Operational outline

Figure 3-11 shows a blockdiagram for generating PWM output from the TO0 pin.

Fig. 3-11 PWM Output from TO0 Pin

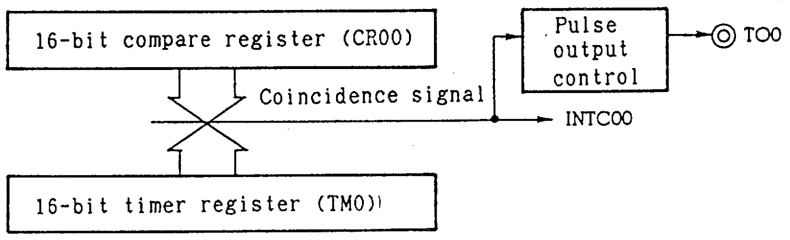
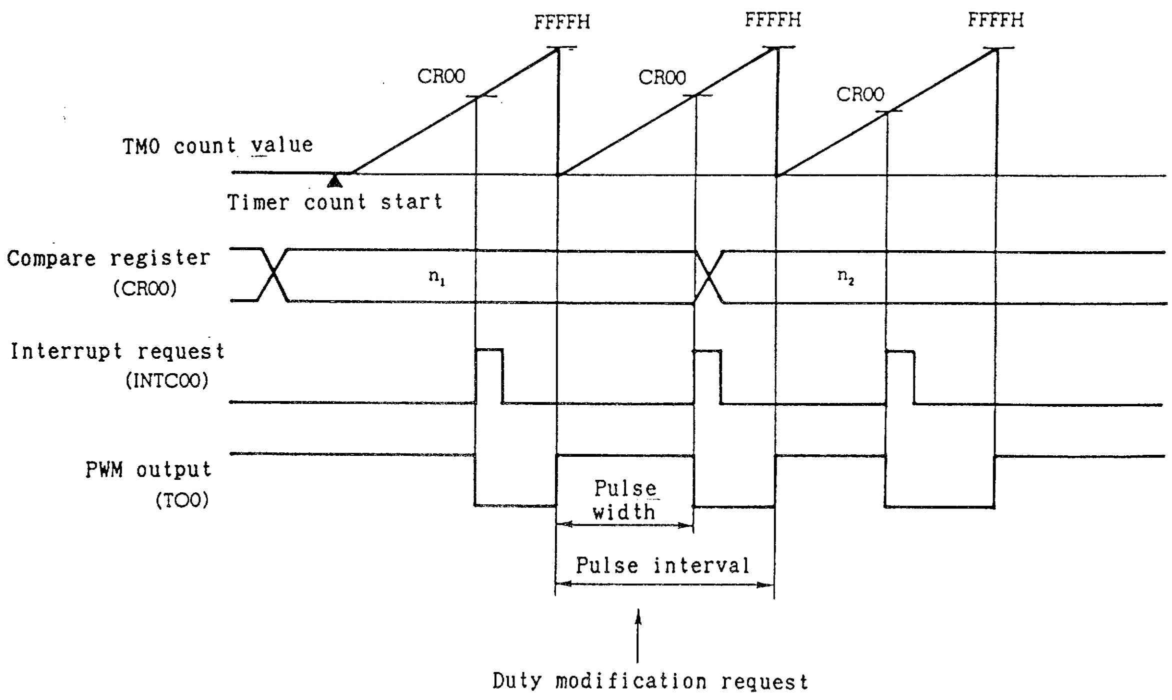


Fig. 3-12 Timing Chart for Outputting PWM from TO0 Pin



Remarks: ALV0 = 0

Pulse interval = 65536 x $8/f_{CLK}$

Pulse width = n x $8/f_{CLK}$ {n : $1 \leqq n \leqq$ FFFFH}

Since one output pulse interval is a period during which the 16-bit timer register (TM0) full-counts (FFFFH), the timer must be free-run. When f_{CLK} = 6 MHz, one interval is approximately 87.4 ms.

When a duty modification request is generated, the value of the compare register (CR00), which determines the pulse width, is modified in the interrupt processing for the INTC00 interrupt request generated by the first coincidence of the timer register (TM0) and the compare register (CR00) after the duty modification request.

Remarks: To simultaneously feed out two different PWM outputs from the TO0 and TO1 pins, set two different values, corresponding to two different pulse widths, to the 16-bit compare registers (CR00, CR01). Two different PWMs can be simultaneously output by two interrupt requests (INTC00, INTC01) generated by coincidence of the 16-bit compare registers (CR00, CR01) and the 16-bit timer register (TM0).

In this program example, a 2-byte work area is used to store a value to determine the duty. When a duty modification request is generated, a value which determines the next duty will be stored in this area. The work area must be allocated in an area, where short direct addressing is possible.

Table 3-4 Work Area Used by PWM Output Program by TM0

Work area name	Function
DUTY1	Stores a value which determines the duty.

(b) Program description

(i) Initialization processing [label name: PWM00]

① Sets the active level for the TO0 timer output to high level, and enables timer output.

Note: For PWM/PPG output, the active level becomes high level when the ALV0 bit for the timer control register (TOC) is set to "0".

② Specifies P34 as control port, so that it can be used as the TO0 output pin.

③ Disables clearing the 16-bit timer register (TM0) by coinciding with the 16-bit compare register (CR00) (free-running mode), and sets the TO0 pin to the PWM output mode.

④ Sets a value which determines the pulse width for the PWM output from the TO0 pin in the 16-bit compare register (CR00).

⑤ Enables count operation for the 16-bit timer/counter.

(ii) Duty modification request processing [label name: C_DTY0]

① Clears the INTC00 interrupt request flag.

② Releases the masking for the INTC00 interrupt request.

(iii) INTC00 interrupt processing [label name: INTC00]

① Selects register bank 1.

② Modifies the value for the 16-bit compare register (CR00).

③ Masks the INTC00 interrupt request.

(c) Mode register setting examples

Timer output control register

	7	6	5	4	3	2	1	0	
TOC	ENT03	ALV3	ENT02	ALV2	ENT01	ALV1	ENT00	ALV0	(00H when reset)
Setting example	0	0	0	0	0	0	1	0	

ALV0	TO0 pin active level specification
0	Low level
1	High level

ENT00	TO0 pin operation specification
0	$\overline{\text{ALV0}}$ output
1	Enables pulse output

ALV1	TO1 pin active level specification
0	Low level
1	High level

ENT01	TO1 pin operation specification
0	$\overline{\text{ALV1}}$ output
1	Enables pulse output

ALV2	TO2 pin active level specification
0	Low level
1	High level

(Cont'd)

ENT02	TO2 pin operation specification
0	$\overline{\text{ALV2}}$ output
1	Enables pulse output

ALV3	TO3 pin active level specification
0	Low level
1	High level

ENT03	TO3 pin operation specification
0	$\overline{\text{ALV3}}$ output
1	Enables pulse output

Port 3 mode control register

	7	6	5	4	3	2	1	0	
PMC3	PMC37	PMC36	PMC35	PMC34	PMC33	PMC32	PMC31	PMC30	(00H when reset)
Setting example	0	0	0	1	0	0	0	0	

PMC30	P30 pin control mode specification
0	Input/output port mode
1	RxD input mode

PMC31	P31 pin control mode specification
0	Input/output port mode
1	TxD output mode

PMC32	F32 pin control mode specification
0	Input/output port mode
1	$\overline{\text{SCK}}$ input/output mode

PMC33	P32 pin control mode specification
0	Input/output port mode
1	SO output, SB0 input/output mode

PMC3n	PM3n pin control mode specification (n = 4 to 7)
0	Input/output port mode
1	TOm output mode (m = 0 to 3)

Capture/compare control register 0

	7	6	5	4	3	2	1	0	
CRC0	MOD1	MOD0	0	1	CLR01	0	0	0	(10H when RESET is input)
Setting example	0	1	0	1	0	0	0	0	

MOD1	MOD0	CLR01	Timer output mode specification		TM0 clear operation when TM0=CR01
			TO0	TO1	
0	0	0	Toggle output	Toggle output	Disabled
0	0	1	Toggle output	Toggle output	Enabled
0	1	0	PWM output	Toggle output	Disabled
0	1	1	Not allowed		
1	0	0	PWM output	PWM output	Disabled
1	0	1	Not allowed		
1	1	0	Not allowed		
1	1	1	PPG output	Toggle output	Enabled

Interrupt mask register L

	7	6	5	4	3	2	1	0	
MK0L	CMK11	CMK10	CMK01	CMK00	PMK3	PMK2	PMK1	PMK0	(FFH when RESET is input)
Setting example	x	x	x	0	x	x	x	x	x: Not manipulated

MK	Interrupt mask flag
0	Enables interrupt processing
1	Retains interrupt processing

Timer control register 0

	7	6	5	4	3	2	1	0	
TMC0	CE3	0	0	0	CE0	OVF0	0	0	(00H when RESET is input)
Setting example	0	0	0	0	1	0	0	0	

16-bit timer/counter

OVF0	Timer/counter overflow flag
0	No overflow
1	Overflow (count up from FFFFH to 0000H)

Remarks: This bit is reset only by the software.

CE0	Timer/counter 0 count operation control
0	Stops count operation with timer/counter cleared
1	Enables count operation

(Cont'd)

8-bit timer/counter 3

CE3	Timer/counter 3 count operation control
0	Stops count operation with timer/counter cleared
1	Enables count operation

(d) Input parameter

DUTY1: Sets a value to determine the duty.

(e) Registers used

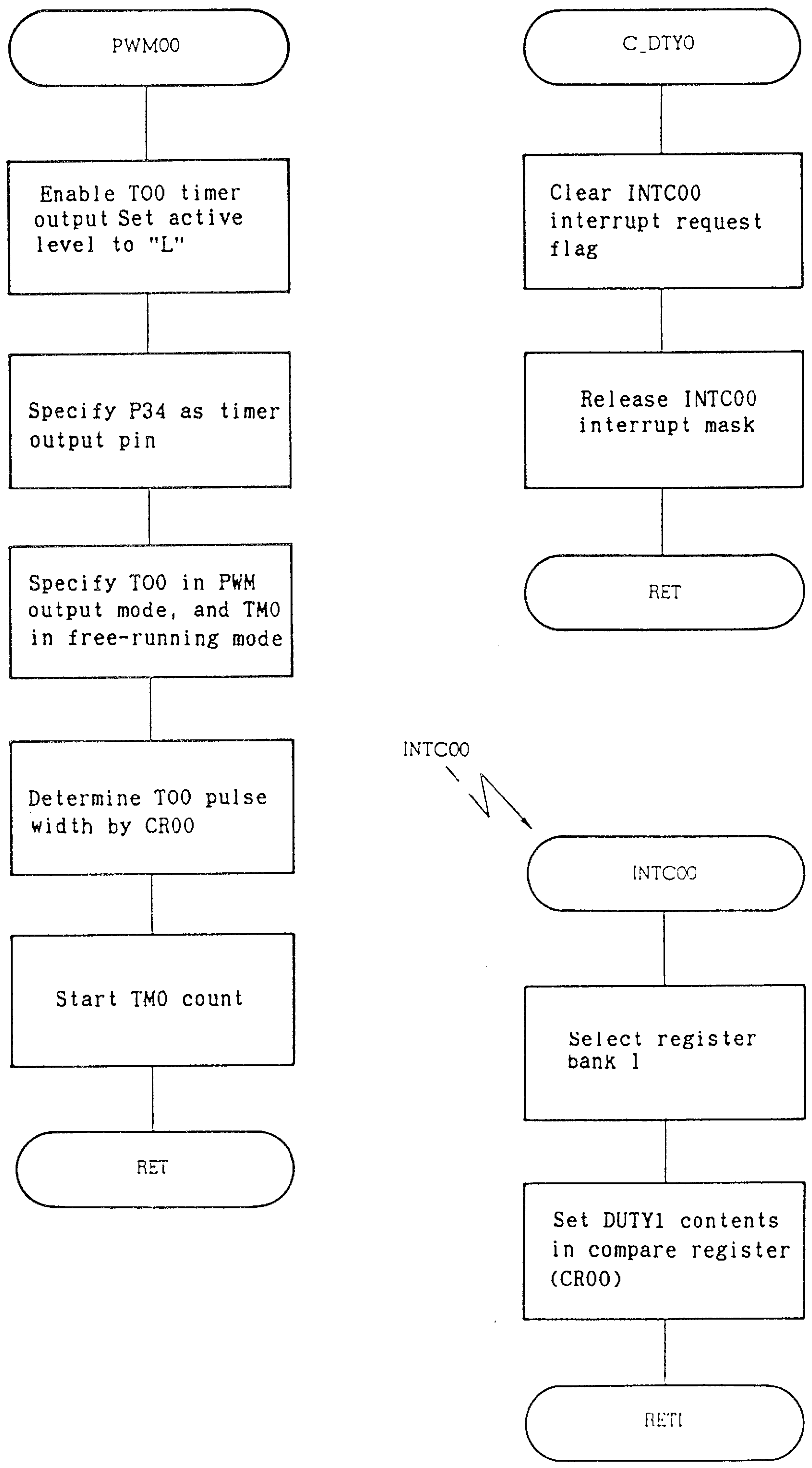
PWM00 processing	:	AX
C_DTY0 processing	:	None
Interrupt processing:		AX (register bank 1)

(f) Program example

The following program example is to output PWM from the TO0 pin and process the duty modification request.
In the duty modification request processing example, it is assumed that a value to determine the duty for the next PWM output has been set in the AX register by some means.

```
        PUBLIC  DUTY1
        EXTRN   C_DTY0,PWM00
                .
                .
DUTY1_D DSEG    SADDR
DUTY1:  DS      2               ; work area for duty
                .
                .
        CSEG
OUT00:
                .
                .
        MOVW    DUTY1,#7FFFH    ; set first duty
        CALL    !PWM00          ; PWM initialize routine
        EI
                .
                .
;       <<Duty modification request>>
C_DTY:
                .
                .
        MOVW    DUTY1,AX        ; set next duty
        CALL    !C_DTY0         ; change duty routine
                .
                .
```

(g) Flow chart


PWM00
Enable T00 timer output Set active level to "L"
Specify P34 as timer output pin
Specify T00 in PWM output mode, and TM0 in free-running mode
Determine T00 pulse width by CR00
Start TM0 count
RET
C_DTY0
Clear INTC00 interrupt request flag
Release INTC00 interrupt mask
RET
INTC00
INTC00
Select register bank 1
Set DUTY1 contents in compare register (CR00)
RETI

(h) Program list

```
        NAME    PWM_0
;
;**********************************************************
;* 16bit-Timer / Counter                                  *
;*      PWM output                                        *
;**********************************************************
;
        PUBLIC  PWM00,C_DTY0
        EXTRN   DUTY1
;
CMK00   EQU     MK0L.4          ; INTC00 mask flag
CIF00   EQU     IF0L.4          ; INTC00 request flag
;
INTC00VT CSEG   AT 00014H
        DW      INTC00          ; INTC00 vector
;
        CSEG
PWM00:
        MOV     TOC,#00000010B  ; timer output,active level high
        MOV     PMC3,#00010000B ; P3 control port
        MOV     CRC0,#01010000B ; TM0 free funning mode
        MOVW    AX,DUTY1        ; set first data of duty
        MOVW    CR00,AX
        MOV     TMC0,#00001000B ; start timer
        RET
;
;       ***** request of change duty *****

C_DTY0:
        CLR1    CIF00           ; clear request flag of INTC00
        CLR1    CMK00           ; open mask of INTC00
        RET
;
;**********************************************************
;*              INTC00 interrupt routine                  *
;**********************************************************
INTC00:
        SEL     RB1
        MOVW    AX,DUTY1        ; set next data of duty
        MOVW    CR00,AX
        SET1    CMK00           ; mask INTC00
        RETI
;
        END
```

(2) PWM output program example using 8-bit timer/counter 2

The following program example is to output PWM wave (from the TO2 pin) whose pulse width is determined by the INTC20 interrupt request.
When changing the duty, set the value to determine the duty in the work area in the RAM and call the subroutine to change the duty.

(a) Operational outline

Figure 3-13 shows a blockdiagram for generating PWM output from the TO2 pin.

Fig. 3-13 PWM Output from TO2 Pin

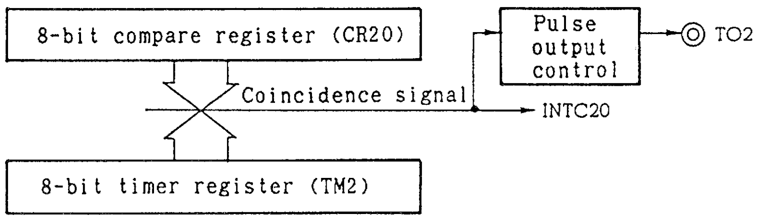
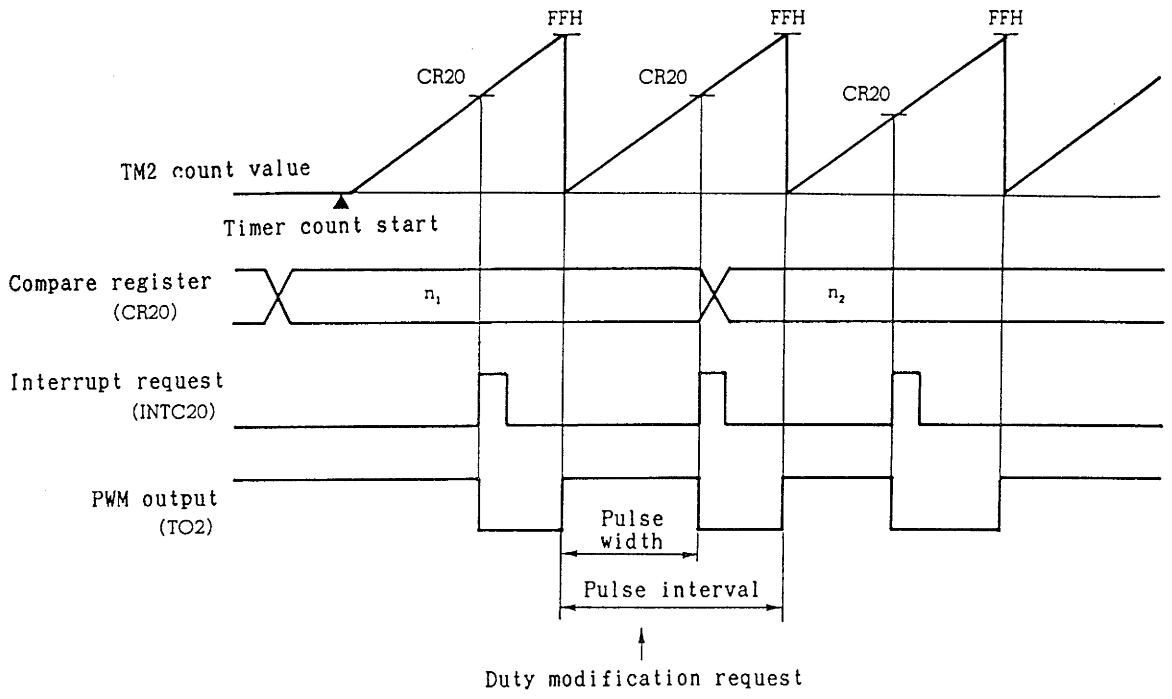


Fig. 3-14 Timing Chart for Outputting PWM from TO2 Pin



Remarks: ALV2 = 0

Pulse interval = 256 x X/f_{CLK}
Pulse width = n x X/f_{CLK} {n : 1 $\leqq$ n $\leqq$ FFH}
X = 16, 32, 64, 128, 256, 512

Since one output pulse interval is a period during which the 8-bit timer register (TM2) full-counts (FFH), the timer must be free-run. One interval is determined by the timer count clock.
When a duty modification request is generated, the value for the compare register (CR20) which determines the pulse width, is modified in the interrupt processing for the INTC20 interrupt request generated by the first coincidence of the timer register (TM2) and the compare register (CR20) after the duty modification request.

Remarks: To simultaneously output two different PWM outputs from the TO2 and TO3 pins, set two different values corresponding to two different pulse widths to the 8-bit compare registers (CR20, CR21). Two different PWMs can be simultaneously output by two interrupt requests (INTC20, INTC21) generated by coincidence of the 8-bit compare registers (CR20, CR21) and the 8-bit timer register (TM2).

In this program example, a 1-byte work area is used to store a value to determine the duty. When a duty modification request is generated, a value which determines the next duty will be stored in this area. The work area must be allocated in the area where a short direct addressing is possible.

Table 3-5 Work Area Used for PWM Output Program by TM2

Work area name	Function
DUTY3	Stores a value which determines the duty.

(b) Program description

(i) Initialization processing [label name: PWM20]

① Sets the active level for TO2 timer output to high level, and enables timer output.

Note: For PWM/PPG output, the active level becomes high level, when the ALV2 bit for the timer control register (TOC) is set to "0"

② Specifies P36 as control port, so that it can be used as the TO2 output pin.

③ Disables clearing the 8-bit timer register (TM2) by coinciding with the 8-bit compare register (CR20) (free-running mode), and sets the TO2 pin to the PWM output mode.

④ Sets count clock for the 8-bit timer/counter 2 to $f_{CLK}/16$.

⑤ Sets a value which determines the pulse width for PWM output from the TO2 pin in the 8-bit compare register (CR20).

⑥ Enables count operation for the 8-bit timer/counter 2.

(ii) Duty modification request processing [label name: C_DTY2]

① Clears the INTC20 interrupt request flag.

② Releases the masking for the INTC20 interrupt request.

(iii) INTC20 interrupt processing [label name: INTC20]

① Modifies the value for the 8-bit compare register (CR20).

② Masks the INTC20 interrupt request.

(c) Mode register setting examples

Timer output control register

	7	6	5	4	3	2	1	0	
TOC	ENT03	ALV3	ENT02	ALV2	ENT01	ALV1	ENT00	ALV0	(00H when RESET is input)
Setting example	0	0	1	0	0	0	0	0	

ALV0	TO0 pin active level specification
0	Low level
1	High level

ENT00	TO0 pin operation specification
0	$\overline{\text{ALV0}}$ output
1	Enables pulse output

ALV1	TO1 pin active level specification
0	Low level
1	High level

ENT01	TO1 pin operation specification
0	$\overline{\text{ALV1}}$ output
1	Enables pulse output

(Cont'd)

ALV2	TO2 pin active level specification
0	Low level
1	High level

ENT02	TO2 pin operation specification
0	$\overline{\text{ALV2}}$ output
1	Enables pulse output

ALV3	TO3 pin active level specification
0	Low level
1	High level

ENT03	TO3 pin operation specification
0	$\overline{\text{ALV3}}$ output
1	Enables pulse output

Port 3 mode control register

	7	6	5	4	3	2	1	0	
PMC3	PMC37	PMC36	PMC35	PMC34	PMC33	PMC32	PMC31	PMC30	(00H when reset)
Setting example	0	1	0	0	0	0	0	0	

PMC30	P30 pin control mode specification
0	Input/output port mode
1	RxD input mode

PMC31	P31 pin control mode specification
0	Input/output port mode
1	TxD output mode

PMC32	P32 pin control mode specification
0	Input/output port mode
1	$\overline{\text{SCK}}$ input/output mode

PMC33	P33 pin control mode specification
0	Input/output port mode
1	SO output, SB0 input/output mode

PMC3n	PM3n pin control mode specification (n = 4 to 7)
0	Input/output port mode
1	TOm output mode (m = 0 to 3)

Capture/compare control register 2

	7	6	5	4	3	2	1	0	
CRC2	MOD1	MOD0	CLR22	1	CLR21	0	0	0	(00H when RESET is input)
Setting example	0	1	0	1	0	0	0	0	

MOD1	MOD0	CLR22	CLR21	Timer output mode		TM2 clear operation
				T02	T03	
0	0	0	0	Toggle output	Toggle output	Not cleared
0	0	0	1	Toggle output	Toggle output	Cleared when TM2 and CR21 registers coincide
0	0	1	0	Toggle output	Toggle output	Cleared after capturing TM2 contents into CR22 register
0	0	1	1	Toggle output	Toggle output	Cleared when TM2 and CR21 registers coincide or are cleared after capturing TM2 contents into CR22 register
0	1	0	0	PWM output	Toggle output	Not cleared
1	0	0	0	PWM output	PWM output	Not cleared
1	1	0	1	PPG output	Toggle output	Cleared when TM2 and CR21 registers coincide

Note: No combination other than above is allowed.

Prescaler mode register 1

	7	6	5	4	3	2	1	0	
PRM1	PRS23	PRS22	PRS21	PRS20	0	PRS12	PRS11	PRS10	(00H when reset)
Setting example	0	0	1	1	0	0	0	0	

8-bit timer/counter 1

PRS12	PRS11	PRS10	Timer/counter 1 count clock frequency specification
0	0	0	$f_{CLK}/16$ (Note)
0	0	1	
0	1	0	
0	1	1	$f_{CLK}/32$
1	0	0	$f_{CLK}/64$
1	0	1	$f_{CLK}/128$
1	1	0	$f_{CLK}/256$
1	1	1	$f_{CLK}/512$

(Cont'd)

8-bit timer/counter 2

PRS23	PRS22	PRS21	PRS20	Timer/counter 3 count clock frequency specification
0	0	0	0	$f_{CLK}/16$
0	0	0	1	
0	0	1	0	
0	0	1	1	$f_{CLK}/32$
0	1	0	0	$f_{CLK}/64$
0	1	0	1	$f_{CLK}/128$
0	1	1	0	$f_{CLK}/256$
0	1	1	1	$f_{CLK}/512$
1	1	1	1	External clock (CI)

Note: f_{CLK}: Internal system clock frequency ($f_{XX}/2$)

Interrupt mask register H

	7	6	5	4	3	2	1	0	
MK0H	CSIMK	STMK	SRMK	SERMK	CMK20	PMK5	PMK4	CMK21	(FFH when reset)
Setting example	x	x	x	x	0	x	x	x	x: Not manipulated

MK	Interrupt mask flag
0	Enables interrupt processing
1	Retains interrupt processing

Timer control register 1

	7	6	5	4	3	2	1	0	
TMC1	CE2	OVF2	CMD2	0	CE1	OVF1	0	0	(00H when reset)
Setting example	1	0	0	0	0	0	0	0	

8-bit timer/counter 1

OVF1	Timer/counter 1 overflow flag
0	No overflow
1	Overflow (count up from FFH to 00H)

Remarks: This bit is reset only by the software.

CE1	Timer/counter 1 count operation control
0	Stops count operation with timer/counter cleared
1	Enables count operation

8-bit timer/counter 2

CMD2	Timer/counter 2 operation mode specification
0	Normal mode
1	One-shot mode

OVF2	Timer/counter 2 overflow flag
0	No overflow
1	Overflow (count up from FFH to 00H)

Remarks: This bit is reset only by the software.

(Cont'd)

CE2	Timer/counter 2 count operation control
0	Stops count operation with timer/counter cleared
1	Enables count operation

(d) Input parameter

DUTY3: Sets a value to determine the duty.

(e) Registers used

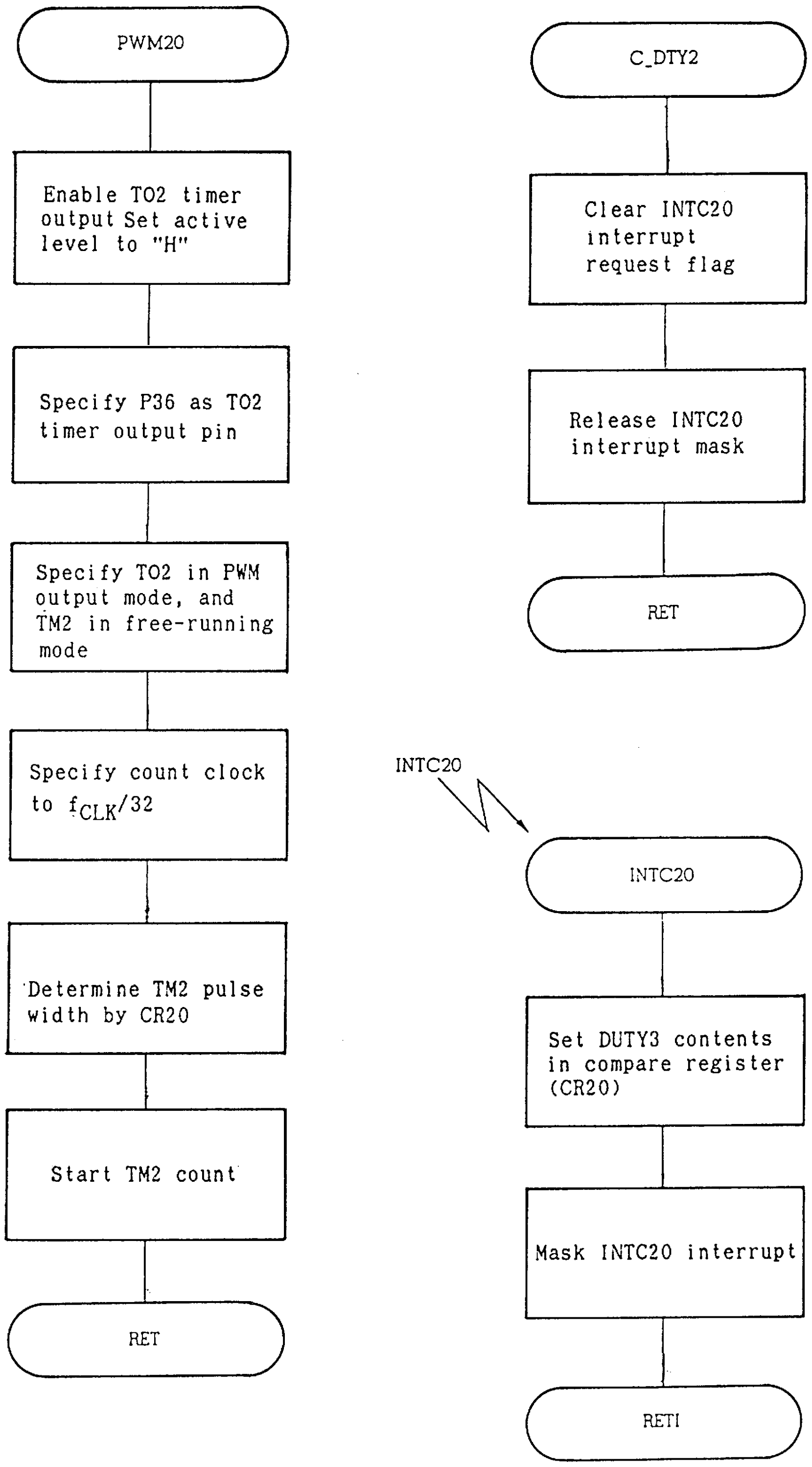
None

(f) Program example

The following program example is to output PWM from the TO2 pin and process the duty modification request.
In the duty modification request processing example, it is assumed that a value to determine the duty for the next PWM output has been set in the A register by some means.

```
        PUBLIC  DUTY3
        EXTRN   C_DTY2,PWM20
                .
                .
DUTY3_D DSEG    SADDR
DUTY3:  DS      1               ; work area for duty
                .
                .
        CSEG
OUT20:
                .
                .
        MOV     DUTY3,#7FH      ; set first duty
        CALL    !PWM20          ; PWM initialize routine
        EI
                .
                .
;       <<Duty modification request>>

C_DTY:
                .
                .
        MOV     DUTY3,A         ; set next duty
        CALL    !C_DTY2
                .
                .
```

(g) Flow chart


PWM20
Enable TO2 timer output Set active level to "H"
Specify P36 as TO2 timer output pin
Specify TO2 in PWM output mode, and TM2 in free-running mode
Specify count clock to fCLK/32
Determine TM2 pulse width by CR20
Start TM2 count
RET
C_DTY2
Clear INTC20 interrupt request flag
Release INTC20 interrupt mask
RET
INTC20
INTC20
Set DUTY3 contents in compare register (CR20)
Mask INTC20 interrupt
RETI

(h) Program list

```
        NAME    PWM_2
;
;*********************************************************
;* 8bit-Timer / Counter-2                               *
;*      PWM output                                      *
;*********************************************************
;
        PUBLIC  PWM20,C_DTY2
        EXTRN   CYCL2,DUTY3
;
CMK20   EQU     MK0H.3           ; INTC20 mask flag
CIF20   EQU     IF0H.3           ; INTC20 request flag
;
INTC20VT CSEG   AT 00012H
        DW      INTC20           ; INTC20 vector
;
        CSEG
PWM20:
        MOV     TOC,#00100000B  ; timer output,active level high
        MOV     PMC3,#01000000B ; P3 control port
        MOV     CRC2,#01010000B ; TM2 free funning mode
        MOV     PRM1,#00110000B ; TM2 prescaler fclk/32
        MOV     CR20,DUTY3      ; set first duty
        MOV     TMC1,#10000000B ; start timer
        RET
;
;       ***** request of change duty *****

C_DTY2:
        CLR1    CIF20           ; clear request flag of INTC20
        CLR1    CMK20           ; open mask of INTC20
        RET
;
;*********************************************************
;*              INTC20 interrupt routine                *
;*********************************************************

INTC20:
        MOV     CR20,DUTY3      ; set next duty
        SET1    CMK20           ; mask INTC20
        RETI
;
        END
```

(3) PPG output program example using 16-bit timer/counter

The following program example is to output PPG wave (from TO0 pin) whose interval is determined by the INTC01 interrupt request and the pulse width is determined by the INTC00 interrupt request.
When changing the duty, set the value to determine the duty in the work area in the RAM and call the subroutine to change the duty.

(a) Operational outline

Figure 3-15 shows a blockdiagram for generating PPG output from the TO0 pin.

Fig. 3-15 PPG Output from TO0 Pin

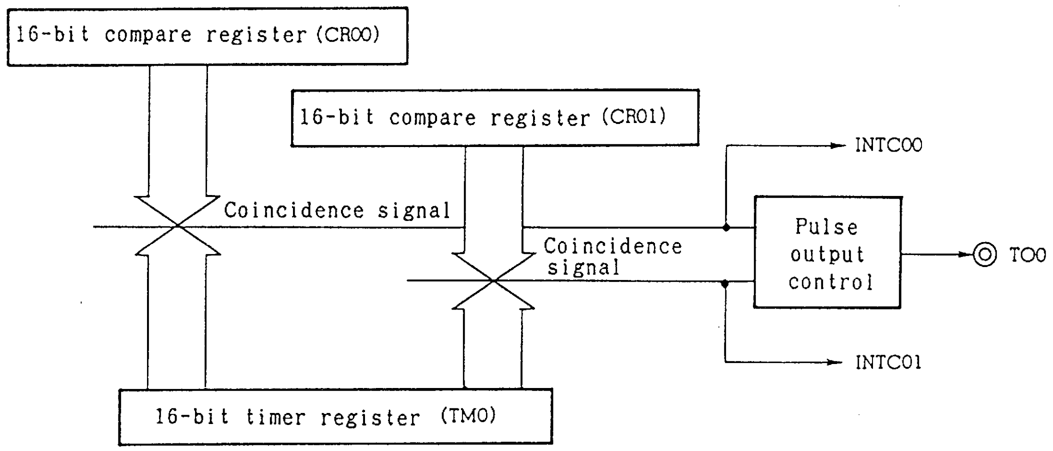
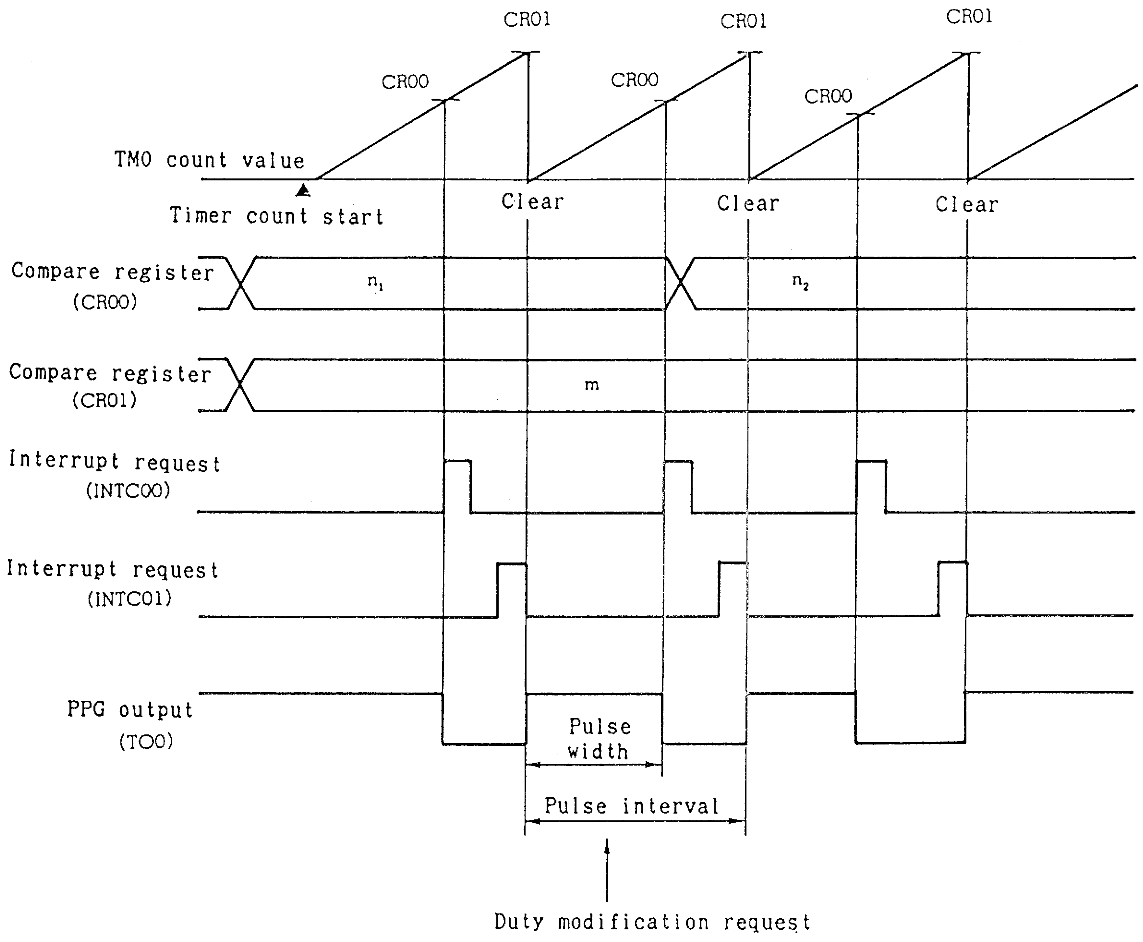

16-bit compare register (CR00)
16-bit compare register (CR01)
INTC00
Coincidence signal
Coincidence signal
Pulse output control
TO0
INTC01
16-bit timer register (TM0)

Fig. 3-16 Timing Chart for Outputting PPG from TO0 Pin



Remarks: ALV0 = 0

Pulse interval = (m + 1) x 8/f_{CLK} {m : $2 \leq m \leq$ FFFFH}

Pulse width = n x 8/f_{CLK} {n : $1 \leq n \leq$ FFFFH}

However, pulse width (CR00) $\leq$ pulse width (CR01)

Since one output pulse interval is a period from a coincidence occurrence for the 16-bit compare register (CR01) value and the 16-bit timer register (TM0) value to the next coincidence, the timer is enabled to be cleared by the TM0 and CR01 coincidence. The pulse width is a period from when the timer is cleared to when the 16-bit compare register (CR00) value and the 16-bit timer register (TM0) value are coincided.
When a duty modification request is generated, the value for the compare register (CR00) which determines the pulse width, is modified in the interrupt processing for the INTC00 interrupt request generated by the first coincidence of the timer register (TM0) and the compare register (CR00) after the duty modification request.
In this program example, a 2-byte work area is used to store a value to determine the duty. When a duty modification request is generated, a value which determines the next duty will be stored in this area. The work area must be allocated in the area where a short direct addressing is possible.

Table 3-6 Work Area Used by PPG Output Program by TM0

Work area name	Function
DUTY2	Stores a value which determines the duty.

(b) Program description

Refer to (h), "Program list".

(i) Initialization processing [label name: PPG00]

① Sets the active level for TO0 timer output to high level, and enables timer output.

Note: For PWM/PPG output, the active level becomes high level, when the ALV0 bit for the timer control register (TOC) is set to "0"

② Specifies P34 as control port, so that it can be used as the TO0 output pin.

③ Enables clearing the 16-bit timer register by coincidence of the 16-bit timer register (TM0) and the 16-bit compare register (CR01), and sets the TO0 pin to the PPG output mode.

④ Sets a value which determines the interval for PPG output from the TO0 pin in the 16-bit compare register (CR01), and a value which determines the pulse width for PPG output from the TO0 pin in the 16-bit compare register (CR00).

⑤ Enables count operation for the 16-bit timer/counter.

(ii) Duty modification request processing [label name: C_DTY0]

① Clears the INTC00 interrupt request flag.

② Releases masking for the INTC00 interrupt request.

(iii) INTC00 interrupt processing [label name: INTC00]

① Selects register bank 1.

② Modifies the value of the 16-bit compare register (CR00).

③ Masks the INTC00 interrupt request.

(c) Mode register setting examples

Timer output control register

	7	6	5	4	3	2	1	0	
TOC	ENT03	ALV3	ENT02	ALV2	ENT01	ALV1	ENT00	ALV0	(00H when reset)
Setting example	0	0	0	0	0	0	1	0	

ALV0	TO0 pin active level specification
0	Low level
1	High level

ENT00	TO0 pin operation specification
0	$\overline{\text{ALV0}}$ output
1	Enables pulse output

ALV1	TO1 pin active level specification
0	Low level
1	High level

ENT01	TO1 pin operation specification
0	$\overline{\text{ALV1}}$ output
1	Enables pulse output

ALV2	TO2 pin active level specification
0	Low level
1	High level

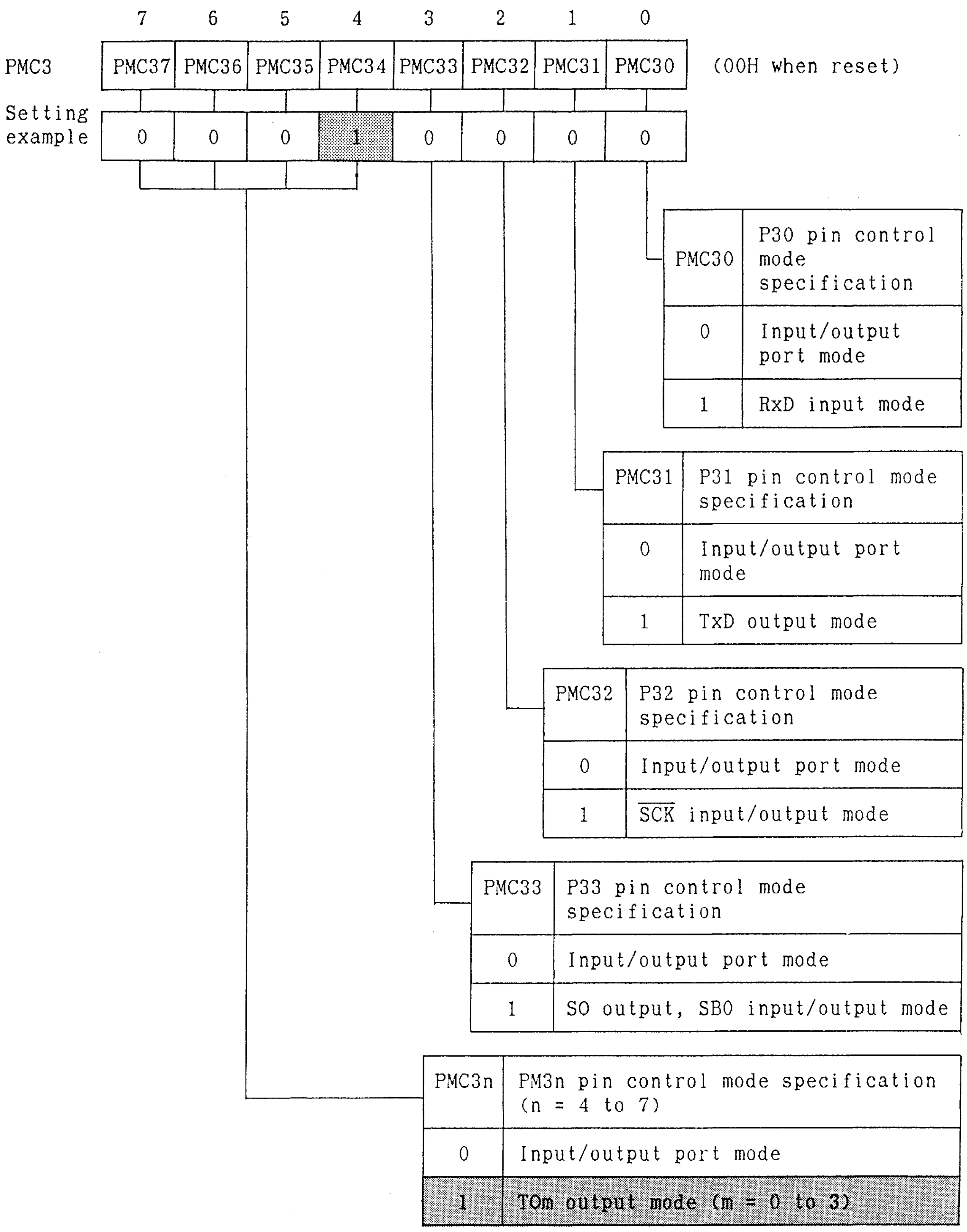
(Cont'd)

ENT02	TO2 pin operation specification
0	$\overline{\text{ALV2}}$ output
1	Enables pulse output

ALV3	TO3 pin active level specification
0	Low level
1	High level

ENT03	TO3 pin operation specification
0	$\overline{\text{ALV3}}$ output
1	Enables pulse output

Port 3 mode control register



	7	6	5	4	3	2	1	0	
PMC3	PMC37	PMC36	PMC35	PMC34	PMC33	PMC32	PMC31	PMC30	(00H when reset)
Setting example	0	0	0	1	0	0	0	0	

PMC30	P30 pin control mode specification
0	Input/output port mode
1	RxD input mode

PMC31	P31 pin control mode specification
0	Input/output port mode
1	TxD output mode

PMC32	P32 pin control mode specification
0	Input/output port mode
1	$\overline{\text{SCK}}$ input/output mode

PMC33	P33 pin control mode specification
0	Input/output port mode
1	SO output, SB0 input/output mode

PMC3n	PM3n pin control mode specification (n = 4 to 7)
0	Input/output port mode
1	TOm output mode (m = 0 to 3)

Capture/compare control register 0

	7	6	5	4	3	2	1	0
CRC0	MOD1	MOD0	0	1	CLR01	0	0	0
Setting example	1	1	0	1	1	0	0	0

(10H when RESET is input)

MOD1	MOD0	CLR01	Timer output mode specification		TM0 clear operation when TM0=CR01
			TO0	TO1	
0	0	0	Toggle output	Toggle output	Disabled
0	0	1	Toggle output	Toggle output	Enabled
0	1	0	PWM output	Toggle output	Disabled
0	1	1	Not allowed		
1	0	0	PWM output	PWM output	Disabled
1	0	1	Not allowed		
1	1	0	Not allowed		
1	1	1	PPG output	Toggle output	Enabled

Interrupt mask register L

	7	6	5	4	3	2	1	0	
MK0L	CMK11	CMK10	CMK01	CMK00	PMK3	PMK2	PMK1	PMK0	(FFH when RESET is input)
Setting example	x	x	x	0	x	x	x	x	x: Not manipulated

MK	Interrupt mask flag
0	Enables interrupt processing
1	Retains interrupt processing

Timer control register 0

	7	6	5	4	3	2	1	0	
TMC0	CE3	0	0	0	CE0	OVF0	0	0	(00H when reset)
Setting example	0	0	0	0	1	0	0	0	

16-bit timer/counter

OVF0	Timer/counter 0 overflow flag
0	No overflow
1	Overflow (count up from FFFFH to 0000H)

Remarks: This bit is reset only by the software.

CE0	Timer/counter 0 count operation control
0	Stops count operation with timer/counter cleared
1	Enables count operation

8-bit timer/counter 3

CE3	Timer/counter 3 count operation control
0	Stops count operation with timer/counter cleared
1	Enables count operation

(d) Input parameters

CYCL0: Sets a value which determines the PPG output interval from the TO0 pin.

DUTY2: Sets a value to determine the duty.

(e) Registers used

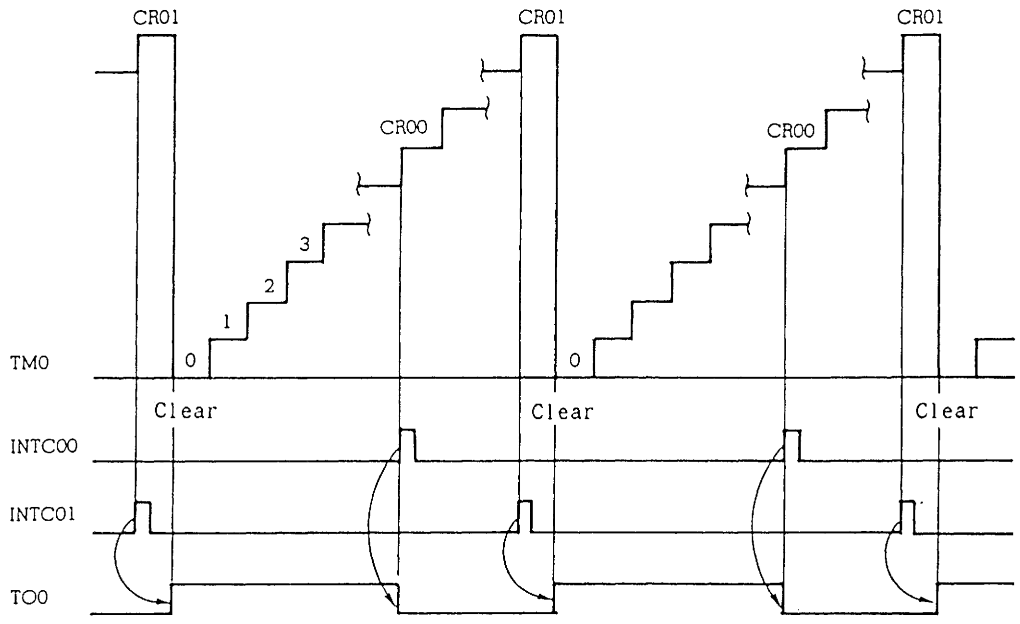
PPG00 processing	:	AX
C_DTY0 processing	:	None
Interrupt processing	:	AX (register bank 1)

(f) Program example

The following program example is to output PPG from the TO0 pin and process the duty modification request. The interval is fixed to 12.5 Hz.

Remarks: When outputting PPG, INTC00 and INTC01 generation timings and PPG output timing are as shown in Fig. 3-17.

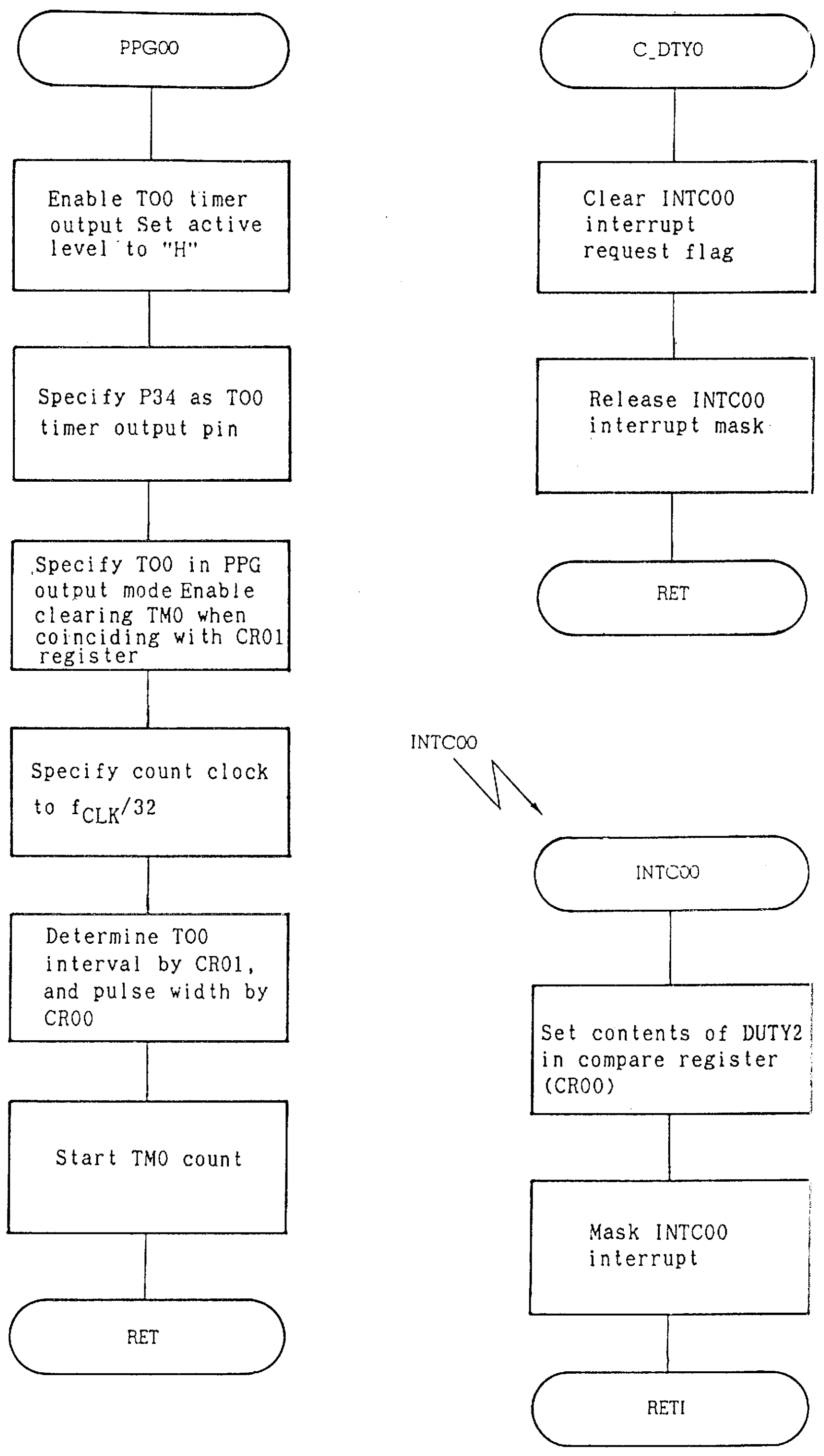
Fig. 3-17 PPG Output Timing (TM0)



The timer output inverts after one timer count of the INTC01 generation, which determines the interval. Therefore, the value set to the compare register (CR01) must be a value calculated for the interval minus 1.
The following shows a program example. In the duty modification request processing example, it is assumed that a value to determine the next duty has been set in the AX register by some means.

```
        PUBLIC  DUTY2,CYCL0
        EXTRN   C_DTY0,PPG00
                .
                .
CYCL0   EQU     59999           ; PPG output cycle

DUTY2_D DSEG    SADDR
DUTY2:  DS      2               ; work area for duty
                .
                .
        CSEG
OUT00:
                .
                .
        MOVW    DUTY2,#7FFFH    ; set first duty
        CALL    !PPG00          ; PPG initialize routine
        EI
                .
                .
;       <<Duty modification request>>

C_DTY:
                .
                .
        MOVW    DUTY2,AX        ; set next duty
        CALL    !C_DTY0
                .
                .
```

(g) Flow chart



PPG00
Enable T00 timer output Set active level to "H"
Specify P34 as T00 timer output pin
Specify T00 in PPG output mode Enable clearing TM0 when coinciding with CR01 register
Specify count clock to $f_{CLK}/32$
Determine T00 interval by CR01, and pulse width by CR00
Start TM0 count
RET
C_DTY0
Clear INTC00 interrupt request flag
Release INTC00 interrupt mask
RET
INTC00
INTC00
Set contents of DUTY2 in compare register (CR00)
Mask INTC00 interrupt
RETI

(h) Program list

```
        NAME    PPG_0
;
;***********************************************************
;* 16bit-Timer / Counter                                   *
;*      PPG output                                         *
;***********************************************************
;
        PUBLIC  PPG00,C_DTY0
        EXTRN   CYCL0,DUTY2
;
CMK00   EQU     MK0L.4              ; INTC00 mask flag
CIF00   EQU     IF0L.4              ; INTC00 request flag
;
INTC00VT CSEG   AT 00014H
        DW      INTC00              ; INTC00 vector
;
        CSEG
PPG00:
        MOV     TOC,#00000010B  ; timer output,active level high
        MOV     PMC3,#00010000B ; P3 control port
        MOV     CRC0,#11011000B ; TM0 PPG output mode
        MOVW    CR01,#CYCL0     ; set output cycle
        MOVW    AX,DUTY2        ; set first output duty
        MOVW    CR00,AX
        MOV     TMC0,#00001000B ; start timer

        RET
;
;       ***** request of change duty *****

C_DTY0:
        CLR1    CIF00           ; clear request flag of INTC00
        CLR1    CMK00           ; open mask of INTC00
        RET
;
;***********************************************************
;*              INTC00 interrupt routine                   *
;***********************************************************

INTC00:
        SEL     RB1
        MOVW    AX,DUTY2        ; set data of duty
        MOVW    CR00,AX
        SET1    CMK00           ; mask INTC00
        RETI
;
        END
```

(4) PPG output program example using 8-bit timer/counter 2

The following program example is to output PPG wave (from TO2 pin) whose interval is determined by the INTC21 interrupt request and pulse width is determined by the INTC20 interrupt request.
When changing the duty, set the value to determine the duty in the work area in the RAM and call the subroutine to change the duty.

(a) Operational outline

Figure 3-18 shows a blockdiagram for generating PPG output from the TO2 pin.

Fig. 3-18 PPG Output from TO2 Pin

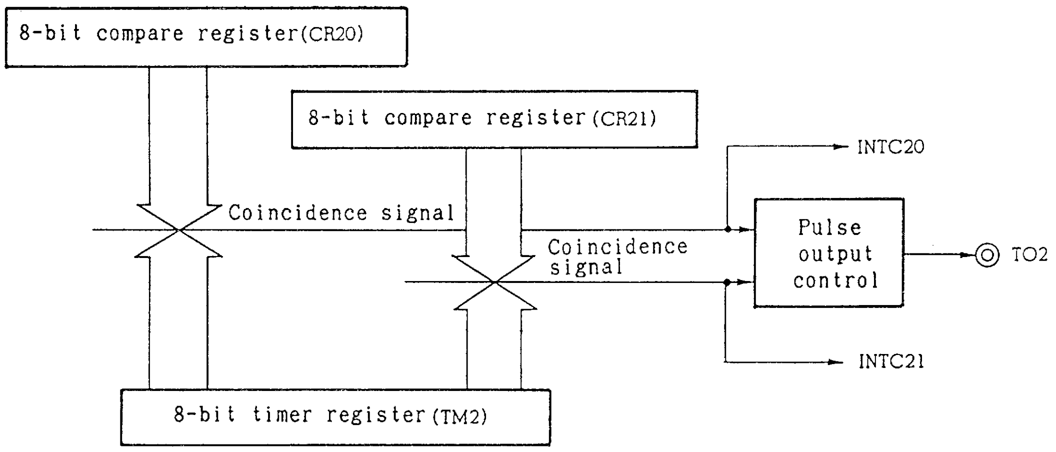
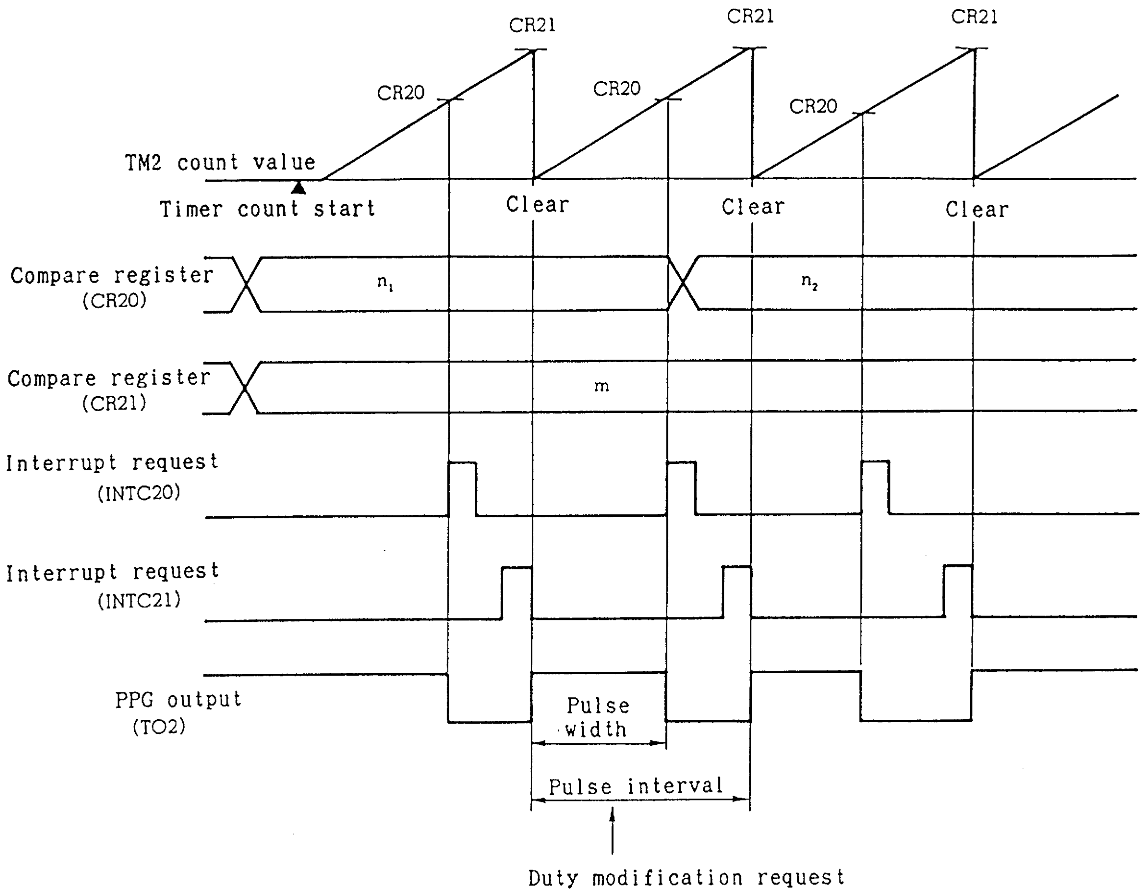

8-bit compare register(CR20)
8-bit compare register (CR21)
Coincidence signal
Coincidence signal
INTC20
Pulse output control
TO2
INTC21
8-bit timer register (TM2)

Fig. 3-19 Timing Chart for Outputting PPG from TO2 Pin



Remarks: ALV2 = 0

Pulse interval = $(m + 1) \times X/f_{CLK}$ {m : $2 \leqq m \leqq$ FFH}
X = 16, 32, 64, 128, 256, 512

Pulse width = $n \times X/f_{CLK}$ {n : $1 \leqq n \leqq$ FFH}

However, pulse width (CR20) $\leqq$ pulse interval (CR21)

Since one output pulse interval is a period from a occurrence of coincidence of the 8-bit compare register (CR21) value and the 8-bit timer register (TM2) value to the next coincidence, the timer is enabled to be cleared by the coincidence of TM2 and CR21. The pulse width is a period from when the timer is cleared to when the 8-bit compare register (CR20) value and the 8-bit timer register 2 (TM2) value are coincided.

When a duty modification request is generated, the value of the compare register (CR20), which determines the pulse width, is modified in the interrupt processing for the INTC20 interrupt request generated by the first coincidence of the timer register (TM2) and the compare register (CR20) after the duty modification request.

In this program example, a 1-byte work area is used to store a value to determine the duty. When a duty modification request is generated, a value which determines the next duty will be stored in this area. The work area must be allocated in an area where short direct addressing is possible.

Table 3-7 Work Area Used by PPG Output Program by TM2

Work area name	Function
DUTY4	Stores a value which determines the duty.

(b) Program description

Refer to (h), "Program list".

(i) Initialization processing [label name: PPG20]

① Sets the active level for TO2 timer output to high level, and enables timer output.

Note: For PWM/PPG output, the active level becomes high level when the ALV2 bit for the timer control register (TOC) is set to "0".

② Specifies P36 as control port, so that it can be used as the TO2 output pin.

③ Enables clearing the 8-bit timer register (TM2) by coincidence of the 8-bit timer register (TM2) and the 8-bit compare register (CR21), and sets the TO2 pin to the PPG output mode.

④ Specifies the 8-bit timer/counter 2 count clock to $f_{CLK}/64$.

⑤ Sets a value which determines the interval for PPG output from the TO2 pin in the 8-bit compare register (CR21), and a value which determines the pulse width for PPG output from the TO2 pin in the 8-bit compare register (CR20).

⑥ Enables count operation of the 8-bit timer/counter 2.

(ii) Duty modification request processing
[label name: C_DTY2]

① Clears the INTC20 interrupt request flag.
② Releases the masking of the INTC20 interrupt request.

(iii) INTC20 interrupt processing [label name: INTC20]

① Modifies the value for the 8-bit compare register (CR20).
② Masks the INTC20 interrupt request.

(c) Mode register setting examples

Timer output control register

	7	6	5	4	3	2	1	0	
TOC	ENT03	ALV3	ENT02	ALV2	ENT01	ALV1	ENT00	ALV0	(00H when reset)
Setting example	0	0	1	0	0	0	0	0	

ALV0	TO0 pin active level specification
0	Low level
1	High level

ENT00	TO0 pin operation specification
0	$\overline{\text{ALV0}}$ output
1	Enables pulse output

ALV1	TO1 pin active level specification
0	Low level
1	High level

ENT01	TO1 pin operation specification
0	$\overline{\text{ALV1}}$ output
1	Enables pulse output

ALV2	TO2 pin active level specification
0	Low level
1	High level

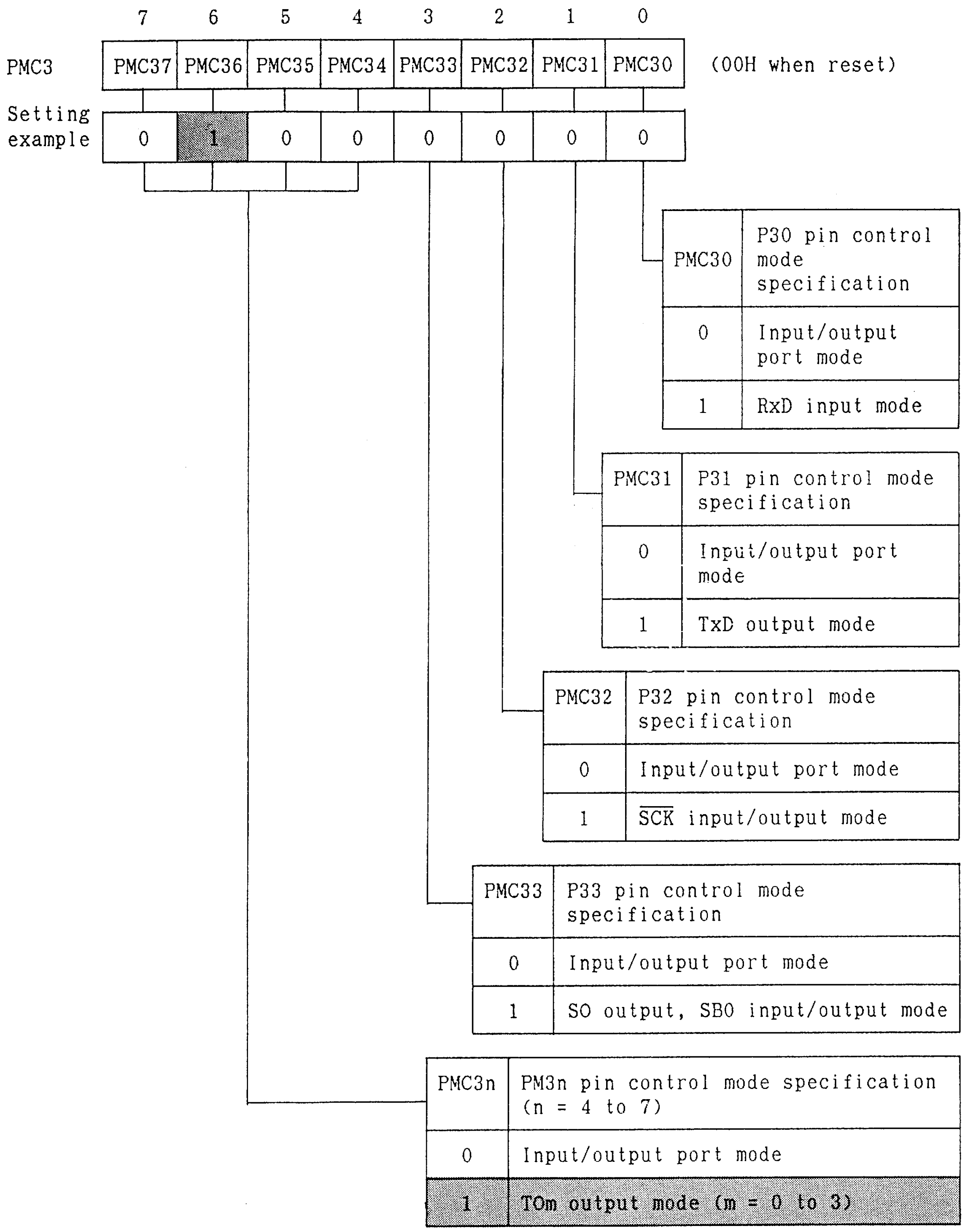
(Cont'd)

ENT02	TO2 pin operation specification
0	$\overline{\text{ALV2}}$ output
1	Enables pulse output

ALV3	TO3 pin active level specification
0	Low level
1	High level

ENT03	TO3 pin operation specification
0	$\overline{\text{ALV3}}$ output
1	Enables pulse output

Port 3 mode control register



PMC30	P30 pin control mode specification
0	Input/output port mode
1	RxD input mode

PMC31	P31 pin control mode specification
0	Input/output port mode
1	TxD output mode

PMC32	P32 pin control mode specification
0	Input/output port mode
1	$\overline{\text{SCK}}$ input/output mode

PMC33	P33 pin control mode specification
0	Input/output port mode
1	SO output, SB0 input/output mode

PMC3n	PM3n pin control mode specification (n = 4 to 7)
0	Input/output port mode
1	TOm output mode (m = 0 to 3)

Capture/compare control register 2

	7	6	5	4	3	2	1	0	
CRC2	MOD1	MOD0	CLR22	1	CLR21	0	0	0	(00H when RESET is input)
Setting example	1	1	0	1	1	0	0	0	

MOD1	MOD0	CLR22	CLR21	Timer output mode		TM2 clear operation
				TO2	TO3	
0	0	0	0	Toggle output	Toggle output	Not cleared
0	0	0	1	Toggle output	Toggle output	Cleared when TM2 and CR21 registers coincide
0	0	1	0	Toggle output	Toggle output	Cleared after capturing TM2 contents into CR22 register
0	0	1	1	Toggle output	Toggle output	Cleared when TM2 and CR21 registers coincide or are cleared after capturing TM2 contents into CR22 register
0	1	0	0	PWM output	Toggle output	Not cleared
1	0	0	0	PWM output	PWM output	Not cleared
1	1	0	1	PPG output	Toggle output	Cleared when TM2 and CR21 registers coincide

Note: No combination other than above is allowed.

Prescaler mode register 1

	7	6	5	4	3	2	1	0	
PRM1	PRS23	PRS22	PRS21	PRS20	0	PRS12	PRS11	PRS10	(00H when reset)
Setting example	0	1	0	0	0	0	0	0	

8-bit timer/counter 1

PRS12	PRS11	PRS10	Timer/counter 1 count clock frequency specification
0	0	0	$f_{CLK}/16$ (Note)
0	0	1	
0	1	0	
0	1	1	$f_{CLK}/32$
1	0	0	$f_{CLK}/64$
1	0	1	$f_{CLK}/128$
1	1	0	$f_{CLK}/256$
1	1	1	$f_{CLK}/512$

(Cont'd)

8-bit timer/counter 2

PRS23	PRS22	PRS21	PRS20	Timer/counter 3 count clock frequency specification
0	0	0	0	$f_{CLK}/16$
0	0	0	1	
0	0	1	0	
0	0	1	1	$f_{CLK}/32$
0	1	0	0	$f_{CLK}/64$
0	1	0	1	$f_{CLK}/128$
0	1	1	0	$f_{CLK}/256$
0	1	1	1	$f_{CLK}/512$
1	1	1	1	External clock (CI)

Note: f_{CLK}: Internal system clock frequency ($f_{XX}/2$)

Interrupt mask register

	7	6	5	4	3	2	1	0	
MK0H	CSIMK	STMK	SRMK	SERMK	CMK20	PMK5	PMK4	CMK1	(FFH when RESET is input)
Setting example	x	x	x	x	0	x	x	x	x: Not manipulated

MK	Interrupt mask flag
0	Enables interrupt processing
1	Retains interrupt processing

Timer control register 1

	7	6	5	4	3	2	1	0	
TMC1	CE2	OVF2	CMD2	0	CE1	OVF1	0	0	(00H when reset)
Setting example	1	0	0	0	0	0	0	0	

8-bit timer/counter 1

OVF1	Timer/counter 1 overflow flag
0	No overflow
1	Overflow (count up from FFH to 00H)

Remarks: This bit is reset only by the software.

CE1	Timer/counter 1 count operation control
0	Stops count operation with timer/counter cleared
1	Enables count operation

8-bit timer/counter 2

CMD2	Timer/counter 2 operation mode specification
0	Normal mode
1	One-shot mode

OVF2	Timer/counter 2 overflow flag
0	No overflow
1	Overflow (count up from FFH to 00H)

Remarks: This bit is reset only by the software.

(Cont'd)

CE2	Timer/counter 2 count operation control
0	Stops count operation with timer/counter cleared
1	Enables count operation

(d) Input parameters

CYCL2: Sets a value which determines the PPG output interval from the TO2 pin.

DUTY4: Sets a value to determine the duty.

(e) Registers used

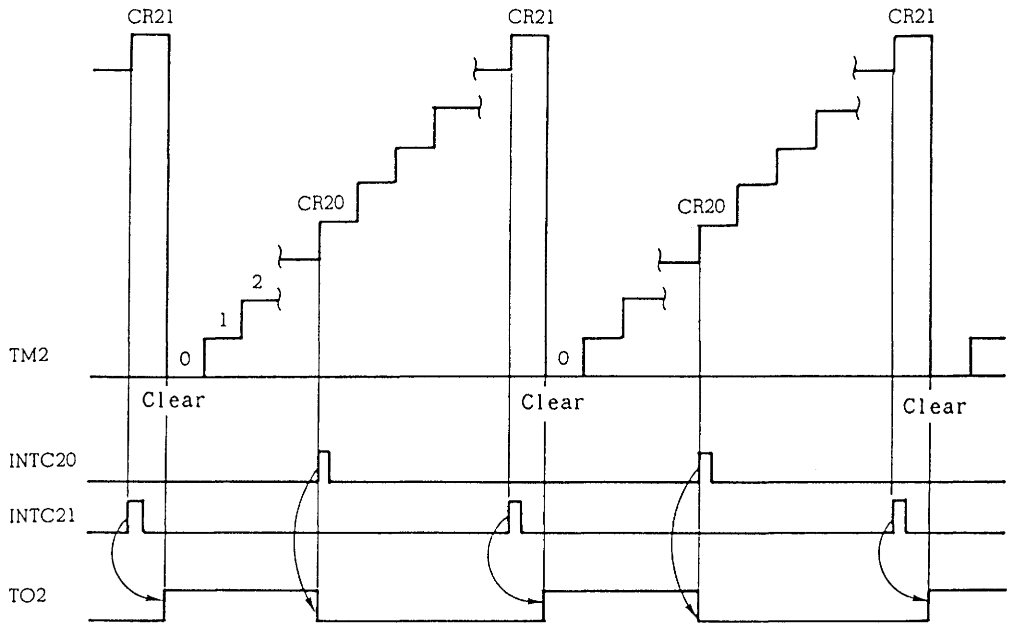
None

(f) Program example

The following program example is to output PPG from the TO2 pin and process the duty modification request. The interval is fixed to 750 Hz.

Remarks: When outputting PPG, INTC20 and INTC21 generation timings and PPG output timing are as shown in Fig. 3-20.

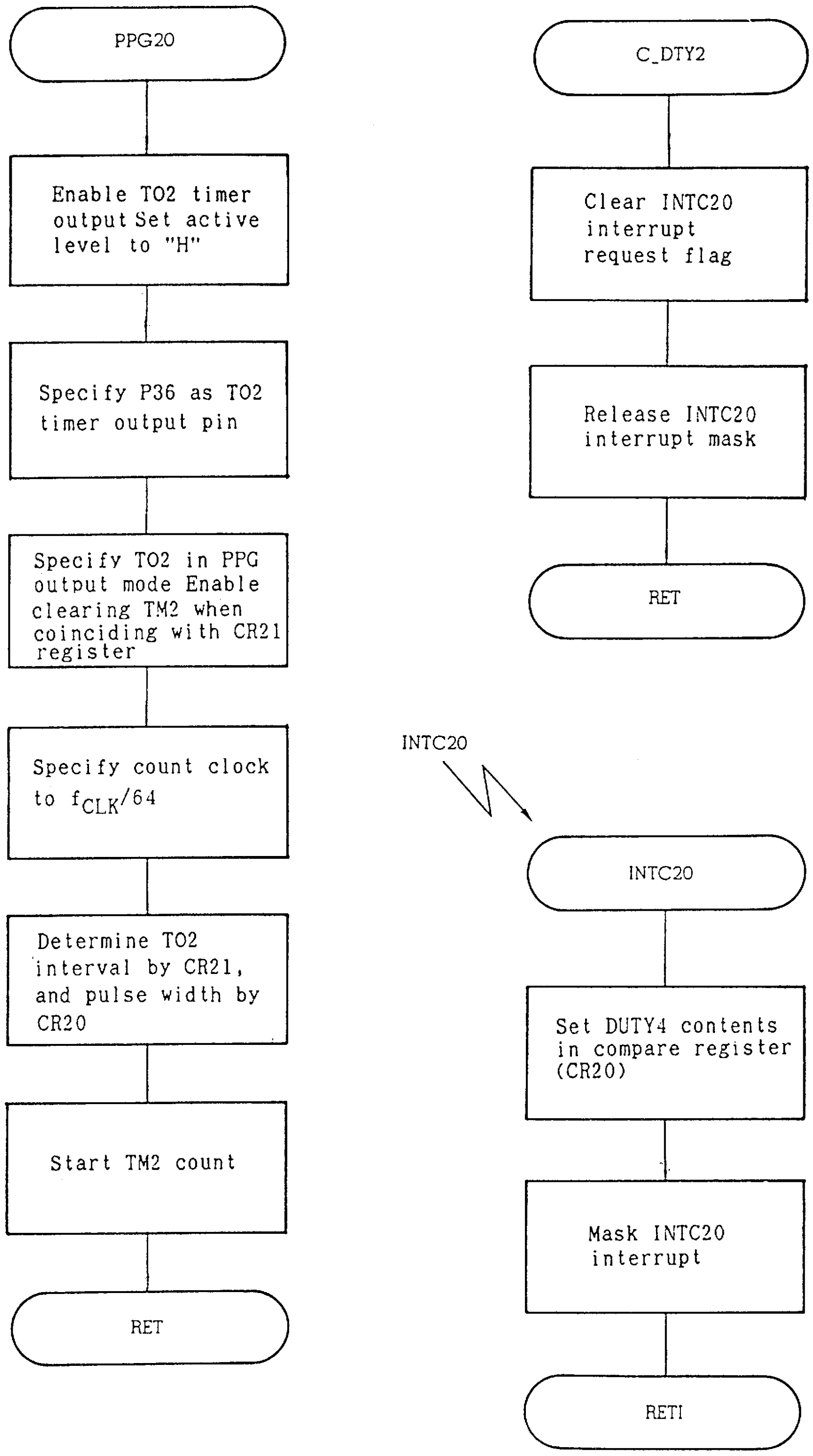
Fig. 3-20 PPG Output Timing (TM2)



The timer output inverts after one timer count of the INTC21 generation, which determines the interval. Therefore, the value set to the compare register (CR21) must be a value calculated for the interval minus 1.
The following shows a program example. In the duty modification request processing example, it is assumed that a value to determine the next duty has been set in the A register by some means.

```
        PUBLIC  DUTY4,CYCL2
        EXTRN   C_DTY2,PPG20
                .
                .
CYCL2   EQU     124             ; PPG output cycle
                .
                .
DUTY4_D DSEG    SADDR
DUTY4:  DS      1               ; work area for duty
                .
                .
        CSEG
OUT20:
                .
                .
        MOV     DUTY4,#40H      ; set first duty
        CALL    !PPG20          ; PPG initialize routine
        EI
                .
                .
;       <<Duty modification request>>
C_DTY:
                .
                .
        MOV     DUTY4,A         ; set next duty
        CALL    !C_DTY2         ; change duty routine
                .
                .
```

(g) Flow chart


PPG20
Enable TO2 timer output Set active level to "H"
Specify P36 as TO2 timer output pin
Specify TO2 in PPG output mode Enable clearing TM2 when coinciding with CR21 register
Specify count clock to fCLK/64
Determine TO2 interval by CR21, and pulse width by CR20
Start TM2 count
RET
C_DTY2
Clear INTC20 interrupt request flag
Release INTC20 interrupt mask
RET
INTC20
INTC20
Set DUTY4 contents in compare register (CR20)
Mask INTC20 interrupt
RETI

(h) Program list

```
        NAME    PPG_2
;
;*********************************************************
;* 8bit-Timer / Counter-2                                *
;*      PPG output                                       *
;*********************************************************
;
        PUBLIC  PPG20,C_DTY2
        EXTRN   CYCL2,DUTY4
;
CMK20   EQU     MK0H.3            ; INTC20 mask flag
CIF20   EQU     IF0H.3            ; INTC20 request flag
;
INTC20VT CSEG   AT 00012H
        DW      INTC20            ; INTC20 vector
;
        CSEG
PPG20:
        MOV     TOC,#00100000B  ; timer output,active level high
        MOV     PMC3,#01000000B ; P3 control port
        MOV     CRC2,#11011000B ; TM2 PPG mode
        MOV     PRM1,#01000000B ; TM2 prescaler fclk/64
        MOV     CR21,#LOW(CYCL2)  ; set cycle
        MOV     CR20,DUTY4      ; set first duty
        MOV     TMC1,#10000000B ; start timer
        RET
;
;       ***** request of change duty *****

C_DTY2:
        CLR1    CIF20             ; clear request flag of INTC20
        CLR1    CMK20             ; open mask of INTC20
        RET
;
;*********************************************************
;*              INTC20 interrupt routine                 *
;*********************************************************

INTC20:
        MOV     CR20,DUTY4        ; set next duty
        SET1    CMK20             ; mask INTC20
        RETI
;
        END
```

3.5 Software Triggered One-shot Pulse Output

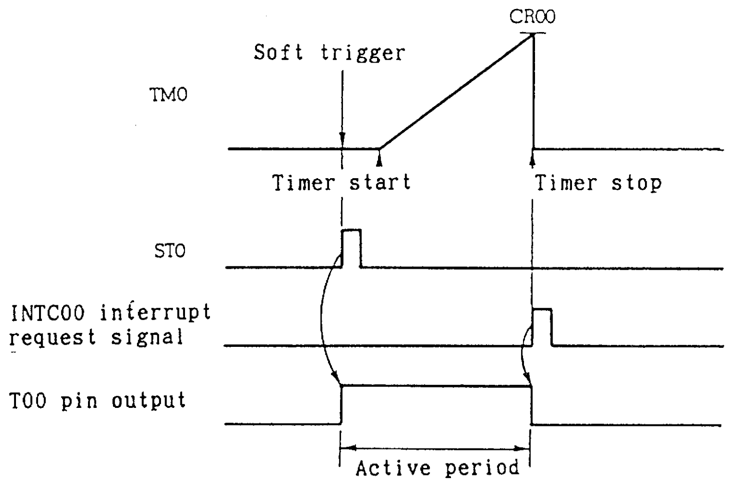
The 16-bit timer/counter (TM0) is provided with the software triggered one-shot pulse output function. A trigger is set by the software for outputting a one-shot pulse from the TOn (n=0, 1) pin. The software triggered one-shot pulse output is provided in these products:

. uPD78218A series, uPD78234 series, uPD78244 series

(1) Operation outline

The timing chart in Fig. 3-21 shows an example of outputting a one-shot pulse from the TO0 pin.

Fig. 3-21 Typical Example of Outputting One-Shot Pulse from TO0 Pin



Remarks: When ALV0=1 (Active High)

A software trigger (setting the ST0 bit for the OSPC register) causes outputting active level from the TO0 pin. After the timer (TM0) count operation is started, the active level is maintained until the TM0 count value coincides with the value set to the CR00.
When TM0 and CR00 coincide, TO0 inverts (inactive level output).

When INTC00 interrupt is generated, the timer (TM0) count operation stops. TO0 continues to output inactive level, until the software trigger is set again.

(2) Program explanation

(a) Processing outline (refer to (7), "Program example")

(i) Initialization processing
Initializes the one-shot pulse output, and enables interrupt.

(ii) Software trigger setting processing

Enables one-shot pulse output, and starts timer (TM0) count operation.

(iii) INTC00 interrupt processing

Stops timer (TM0) count operation.

(b) RAM used

None.

(3) Input/output parameter

None.

(4) Registers

No register is used.

(5) Stacks

No stack is used.

(6) Mode register setting example

Port 3 mode control register

	7	6	5	4	3	2	1	0	
PMC3	PMC37	PMC36	PMC35	PMC34	PMC33	PMC32	PMC31	PMC30	(00H when RESET is input)
Setting example	0	0	0	1	0	0	0	0	

PMC30	P30 pin control mode specification
0	Input/output port mode
1	RxD input mode

PMC31	P31 pin control mode specification
0	Input/output port mode
1	TxD output mode

PMC32	P32 pin control mode specification
0	Input/output port mode
1	$\overline{\text{SCK}}$ input/output mode

PMC33	P33 pin control mode specification
0	Input/output port mode
1	SO output mode SB0 input/output mode

PMC3n	P3n pin control mode specification (n=4 to 7)
0	Input/output port mode
1	TOn output mode (n=0 to 3)

Capture/compare control register 0

	7	6	5	4	3	2	1	0	
CRC0	MOD1	MOD0	0	1	CLR01	0	0	0	(10H when RESET is input)
Setting example	0	0	0	1	0	0	0	0	

MOD1	MOD0	CLR01	Timer output mode specification		TM0 clear operation when TM0=CR01
			TO0	TO1	
0	0	0	Toggle output	Toggle output	Disabled
0	0	1	Toggle output	Toggle output	Enabled
0	1	0	PWM output	Toggle output	Disabled
0	1	1	Not allowed		
1	0	0	PWM output	PWM output	Disabled
1	0	1	Not allowed		
1	1	0	Not allowed		
1	1	1	PPG output	Toggle output	Enabled

One-shot pulse output control register

	7	6	5	4	3	2	1	0
OSPC	ST1	RT1	0	OS1	ST0	RT0	0	OS0
Setting example	0	0	0	0	0	0	0	1

(00H when RESET is input)

OS0	Pulse output type selection.
0	Selects toggle output, PWM output, PPG output.
1	Selects software triggered one-shot pulse

ST0	RT0	T00 output control
0	0	Dose not change output
0	1	Outputs inactive level to T00
1	0	Outputs active level to T00
1	1	Not allowed

OS1	Pulse output type selection
0	Selects toggle output, PWM output, PPG output
1	Selects software triggered one-shot pulse

(Cont'd)

ST1	RT1	TO1 output control
0	0	Does not change output
0	1	Outputs inactive level to TO1
1	0	Outputs active level to TO1
1	1	Not allowed

Remarks 1: RTO, STO, RT1, and ST1 bits are write-only, and 0 is read out if read operation.

2: Disabling/enabling pulse output from pins and active level specifications are implemented using the timer output control register (TOC).

Timer output control register

	7	6	5	4	3	2	1	0
TOC	ENTO3	ALV3	ENTO2	ALV2	ENTO1	ALV1	ENTO0	ALV0
Setting example	0	0	0	0	0	0	1	1

(00H when RESET is input)

ALV0	TO0 pin active level	
	When toggle output is specified or when one-shot pulse output is specified	When PWM/PPG output is specified
0	Low level	High level
1	High level	Low level

ENTO0	TO0 pin operation specification
0	Outputs $\overline{\text{ALV0}}$
1	Enables pulse output

ALV1	TO1 pin active level	
	When toggle output is specified or when one-shot pulse output is specified	When PWM/PPG output is specified
0	Low level	High level
1	High level	Low level

(Cont'd)

ENTO1	TO1 pin operation specification
0	Outputs $\overline{\text{ALV1}}$
1	Enables pulse output

Controls timer output (TO2, TO3) by 8-bit timer/counter 2.

Interrupt mask register L

	7	6	5	4	3	2	1	0	
MK0L	CMK11	CMK10	CMK01	CMK00	PMK3	PMK2	PMK1	PMK0	(FFH when RESET is input)
Setting example	x	x	x	0	x	x	x	x	

MK	Interrupt mask flag
0	Enables interrupt processing
1	Holds interrupt processing

Timer control register 0

	7	6	5	4	3	2	1	0	
TMC0	CE	0	0	0	CE0	OVF0	0	0	(00H when RESET is input)
Setting example	0	0	0	0	Note	0	0	0	Note: Set to 0 or 1 by the software.

16-bit timer/counter

OVF0	TM0 overflow flag
0	No overflow
1	Overflow (count up from FFFFH to 0000H)

Remarks: This bit is reset only by the software.

CE0	TM0 count operation control
0	Stops count operation in cleared condition
1	Enables count operation

8-bit timer/counter unit3

CE	TM3 count operation control
0	Stops count operation in cleared condition
1	Enables count operation

(7) Program example

The following program example shows an example of outputting 500us one-shot pulse from the TO0 pin. In this case, the setting value to the compare register (CR00) is calculated as follows:

$$\boxed{\frac{500 \times 10^{-6}}{8/(6 \times 10^{6})} = \underset{\sim\sim}{375}}$$

When setting a software trigger in an interrupt processing, one-shot pulse can be output by the program as shown below. However, initialization of interrupt (such as releasing masking) is not included, so that it must be added.

```
                 :
                 :
OSPWID  EQU     375                     ; one-shot pulse width

;       *** INITIALIZATION ***
                 :
                 :
        MOV     PMC3,#00010000B         ; set TO0 output mode
        MOV     CRC0,#00010000B         ; set TO0 timer out, disable clear TM0
        MOV     OSPC,#00000001B         ; set TO0 one-shot pulse output mode
        MOVW    CR00,#OSPWID            ; set one-shot pulse width
        MOV     TOC,#00000011B          ; set TO0 high active, enable output
        CLR1    CMK00                   ; open INTC00 mask
        EI                              ; enable interrupt
                 :
                 :

;       *** SOFTWARE TRIGGER SETTING ***
INT:
        SET1    ST0                     ; output active level from TO0
        MOV     TMC0,#00001000B         ; start TM0
                 :
                 :
        RETI
```

```
;        *** INTCOO INTERRUPT PROCESS ***
INTCOO:
         MOV       TMCO,#00000000B           ; stop TMO
                     :
                     :
         RETI
                     :
                     :
```

3.6 Pulse Cycle Measurement

As example program, that measures the cycle for a pulse by using a 16-bit timer/counter, is given in this section.
The cycle for an external pulse, that is input to the external interrupt request input pin INTP3, is detected and measured. The pulse input width to the INTP3 pin must be at least 12 system clocks (2 microseconds, where f_{CLK} = 6 MHz), regardless of whether the level is high or low. If the pulse width is less than 12 system clocks, the specified edge of the pulse cannot be detected and, thus, the pulse is not captured.
The example program introduced in this section can measure a pulse cycle of 2 microseconds to 87.4 milliseconds (where f_{CLK} = 6 MHz) with a 1.3 microsecond resolution.

(1) Operation

As illustrated in Fig. 3-22, the 16-bit timer register (TM0) value is captured to capture register (CR02) in synchronization with the specified edge (in this example, the rising edge) of the signal input to the INTP3 pin. The captured edge is retained in the capture register. The pulse cycle is obtained as follows: First, the difference is calculated between the count value for TM0 (D_n), which has been captured and retained in capture register (CR02) at the nth edge, and the count value at the n-1th edge (D_{n-1}). Then, this difference is multiplied by the count clock ($8/f_{CLK}$). The product is the pulse cycle.

Fig. 3-22 Pulse Cycle Measurement

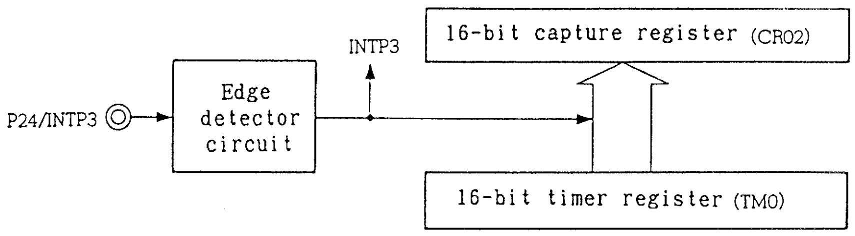
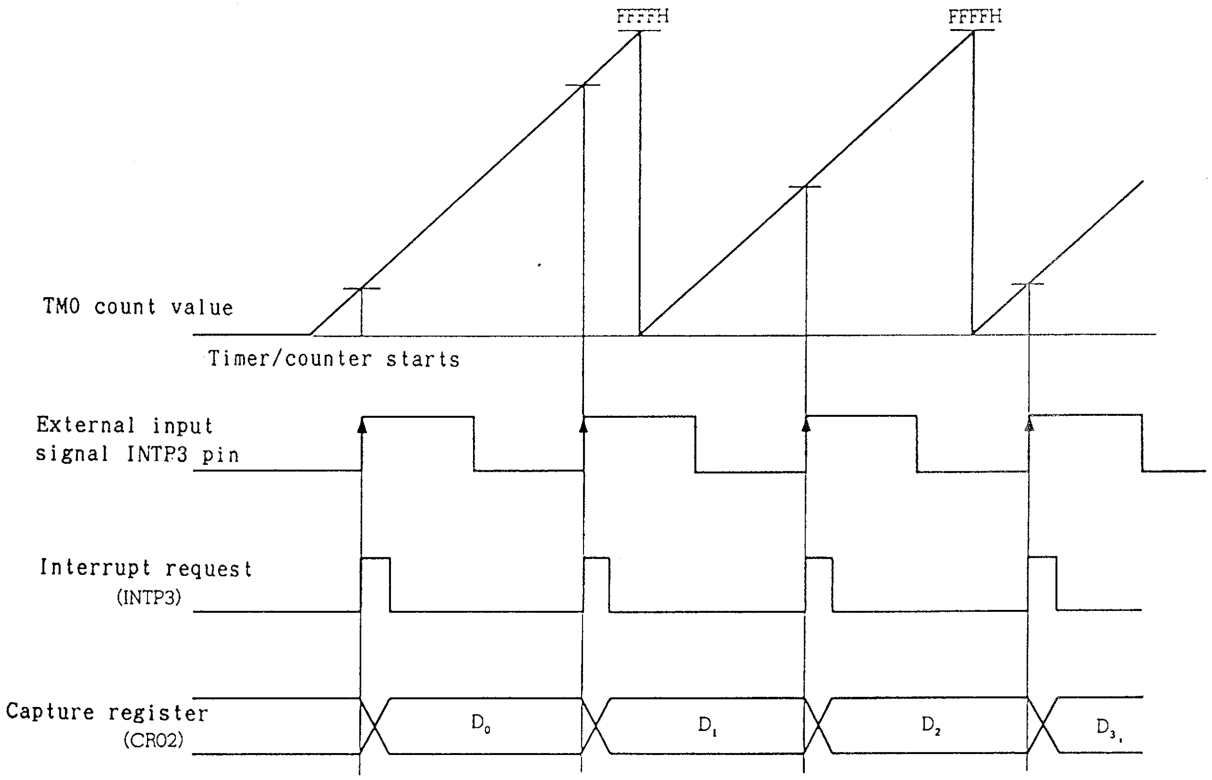


Fig. 3-23 Timing Chart



Pulse interval = $(D_{n+1} - D_n) \times 8/f_{CLK}$

In the example program in this section, the rising edge of the signal input to the INTP3 pin is detected and the signal cycle is measured. The measurement range is from 2 microseconds to 87.4 milliseconds, because overflow is not taken into consideration.

(2) Program ... Refer to (7) Program list

(a) Foreground processing [label: PULSE]

(i) The rising edge for the INTP3 input is regarded as the valid edge.

(ii) Clearing 16-bit timer register (TM0), when the TM0 contents coincide with those for 16-bit compare register (CR01), is inhibited.

(iii) Work area CAPWK, in which the previously captured value is stored, and work area WIDTH, in which the result of pulse cycle calculation is stored, are cleared.

(iv) The count operation for 16-bit timer register (TM0) is enabled.

(v) Interrupt request INTP3 is unmasked.

(b) Background processing [label: PLSANA]

This is vector interrupt processing for interrupt request INTP3.

(i) The value previously captured to register BC is read.

(ii) The captured value is read from 16-bit capture register (CR02) to register AX. The AX register value is stored in work area CAPWK, where the captured value is to be stored.

(iii) The difference between the AX register value and the BC register value is calculated. The result of this calculation is stored in work area WIDTH as the pulse cycle.

(3) Mode register setting

External interrupt mode register 1

	7	6	5	4	3	2	1	0	
INTM1	0	0	ES51	ES50	ES41	ES40	ES31	ES30	(00H when RESET is input)
Setting example	0	0	0	0	0	0	0	1	

ES31	ES30	Specifies input edge to be detected on INTP3 pin
0	0	Falling edge
0	1	Rising edge
1	0	Inhibited
1	1	Both falling and rising edges

ES41	ES40	Specifies input edge to be detected on INTP4 pin
0	0	Falling edge
0	1	Rising edge
1	0	Selects INTC30
1	1	Inhibited

ES51	ES50	Specifies input edge to be detected on INTP5 pin
0	0	Falling edge
0	1	Rising edge
1	0	Inhibited
1	1	Inhibited

Capture/compare control register 0

	7	6	5	4	3	2	1	0	
CRC0	MOD1	MOD0	0	1	CLR01	0	0	0	(10H when RESET is input)
Setting example	0	0	0	1	0	0	0	0	

MOD1	MOD0	CLR01	Timer output mode specification		TM0 clear operation when TM0=CR01
			TO0	TO1	
0	0	0	Toggle output	Toggle output	Disabled
0	0	1	Toggle output	Toggle output	Enabled
0	1	0	PWM output	Toggle output	Disabled
0	1	1	Not allowed		
1	0	0	PWM output	PWM output	Disabled
1	0	1	Not allowed		
1	1	0	Not allowed		
1	1	1	PPG output	Toggle output	Enabled

Timer control register 0

	7	6	5	4	3	2	1	0	
TMC0	CE	0	0	0	CE0	OVF0	0	0	(00H when RESET is input)
Setting example	0	0	0	0	1	0	0	0	

16-bit timer/counter

OVF0	TM0 overflow flag
0	No overflow
1	Overflow occurs (count up from FFFFH to 0000H)

Remarks: This bit is cleared by software only.

CE0	Controls TM0 count operation
0	Clears and stops count operation
1	Enables count operation

8-bit timer/counter unit 3

CE	Controls TM3 count operation
0	Clears and stops count operation
1	Enables count operation

Interrupt mask register L

	7	6	5	4	3	2	1	0	
MK0L	CMK11	CMK10	CMK01	CMK00	PMK3	PMK2	PMK1	PMK0	(FFH when RESET is input)
Setting example	x	x	x	x	0	x	x	x	x: Not used

MK	Interrupt mask flag
0	Enables interrupt processing
1	Keeps interrupt processing pending

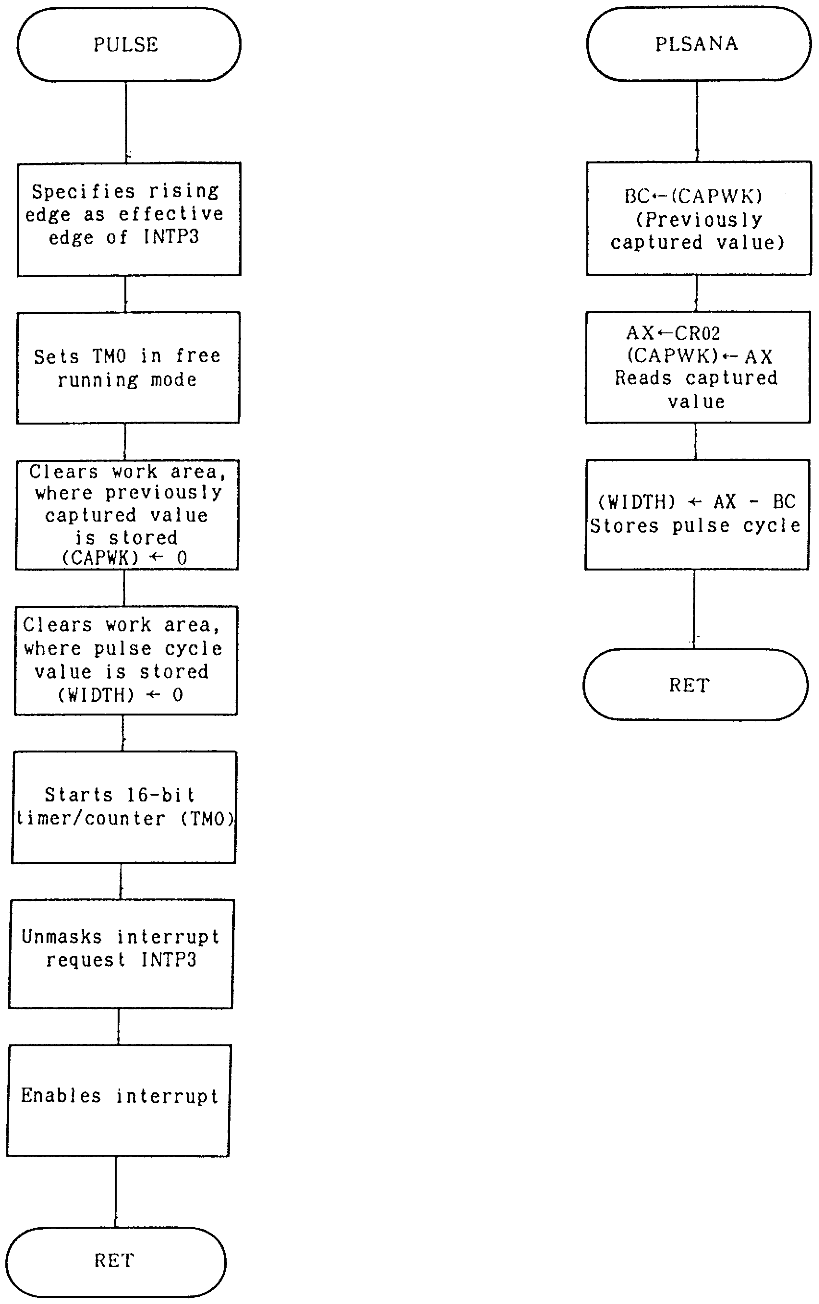
(4) Input/output parameter

WIDTH: Generates a hexadecimal value for the pulse cycle. Because work areas CAPWK and WIDTH are accessed by an instruction in the short direct accessing mode, they must be located in an address range (FE20H to FEF7H) to which the short direct addressing is applicable.

(5) Registers

No register is used.

(6) Flowchart


PULSE
Specifies rising edge as effective edge of INTP3
Sets TM0 in free running mode
Clears work area, where previously captured value is stored (CAPWK) ← 0
Clears work area, where pulse cycle value is stored (WIDTH) ← 0
Starts 16-bit timer/counter (TM0)
Unmasks interrupt request INTP3
Enables interrupt
RET
PLSANA
BC←(CAPWK) (Previously captured value)
AX←CR02 (CAPWK)←AX Reads captured value
(WIDTH) ← AX - BC Stores pulse cycle
RET

(7) Program list

```
        NAME    PULSEM
;
;***********************************************************
;* 16bit-Timer / Counter Unit                              *
;*      measure pulse cycle                                *
;***********************************************************
;
        PUBLIC  PULSE,PLSANA
        EXTRN   CAPWK,WIDTH       ; work area

PMK3    EQU     MKOL.3            ; INTP3 mask flag
;
        CSEG
PULSE:
        MOV     INTM1,#00000001B
                                  ; INTP3's enable edge is rise
        MOV     CRCO,#00010000B ; clear disable TMO by CR01
        MOVW    CAPWK,#0          ; clear result
        MOVW    WIDTH,#0


        MOV     TMCO,#00001000B ; timer start
        CLR1    PMK3              ; open INTP3 mask
        EI                        ; interrupt enable

        RET
;
PLSANA:
        PUSH    BC                ; save register
        PUSH    AX
        MOVW    AX,CAPWK          ; read last caputure data
        MOVW    BC,AX
        MOVW    AX,CR02           ; caputure read
        MOVW    CAPWK,AX          ; save caputure data
        SUBW    AX,BC
        MOVW    WIDTH,AX
        POP     AX                ; restore data
        POP     BC

        RET
;
        END
```

CHAPTER 4 PWM OUTPUT UNIT PROGRAM EXAMPLE (uPD78234)

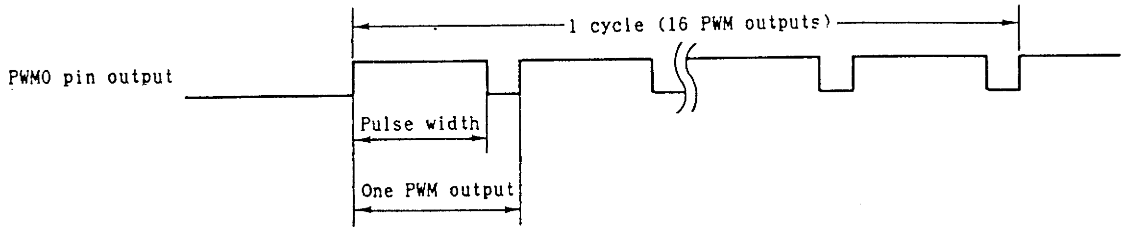
The uPD78234 series is provided with the PWM output unit. Two channels of 12-bit resolution PWM output can be obtained from PWM0 and PWM1 pins.

Generally, the PWM output signal is shaped through a low-pass filter and is used for voltage control.

(1) Operation outline

The timing chart in Fig. 4-1 shows an example of outputting PWM from the PWM0 pin.

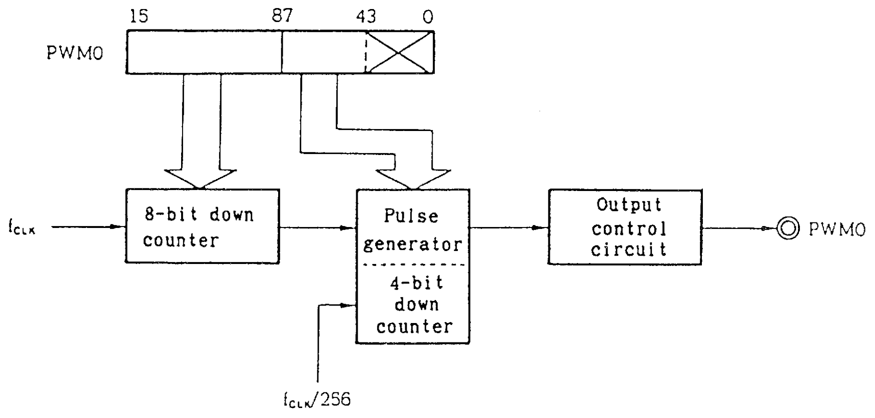
Fig. 4-1 Timing Chart for Outputting PWM from PWM0 Pin



Remarks: When ALV0=1 1(active high)

The PWM output pulse width is determined by a value (16-bit) set to the PWM modulo register (PWM0).
As shown in Fig. 4-2, an 8-bit PWM signal (counted by the 8-bit down counter) can be output 16 times (counted by 4-bit down counter) to obtain 12-bit resolution. The PWM output is output 16 times in one PWM output cycle.

Fig. 4-2 PWM Output Function Block Diagram



One PWM output cycle is as indicated below. Therefore, PWM output pulse width is changed in this cycle.

$$\boxed{1/f_{CLK} \times 2^{12}\ (4096) \doteqdot 682.7\ \text{us}\ (f_{CLK} = 6\ \text{MHz})}$$

(2) Program explanation

(a) Processing outline (refer to (7) "Program example")

(i) Sets initial pulse width for PWM output, and enables PWM output.

(ii) Changes PWM pulse width. Then continue to output this pulse until the value set to the modulo register (PWM0) is modified.

(b) RAM used

None.

(3) Input/output parameter

None.

(4) Register used

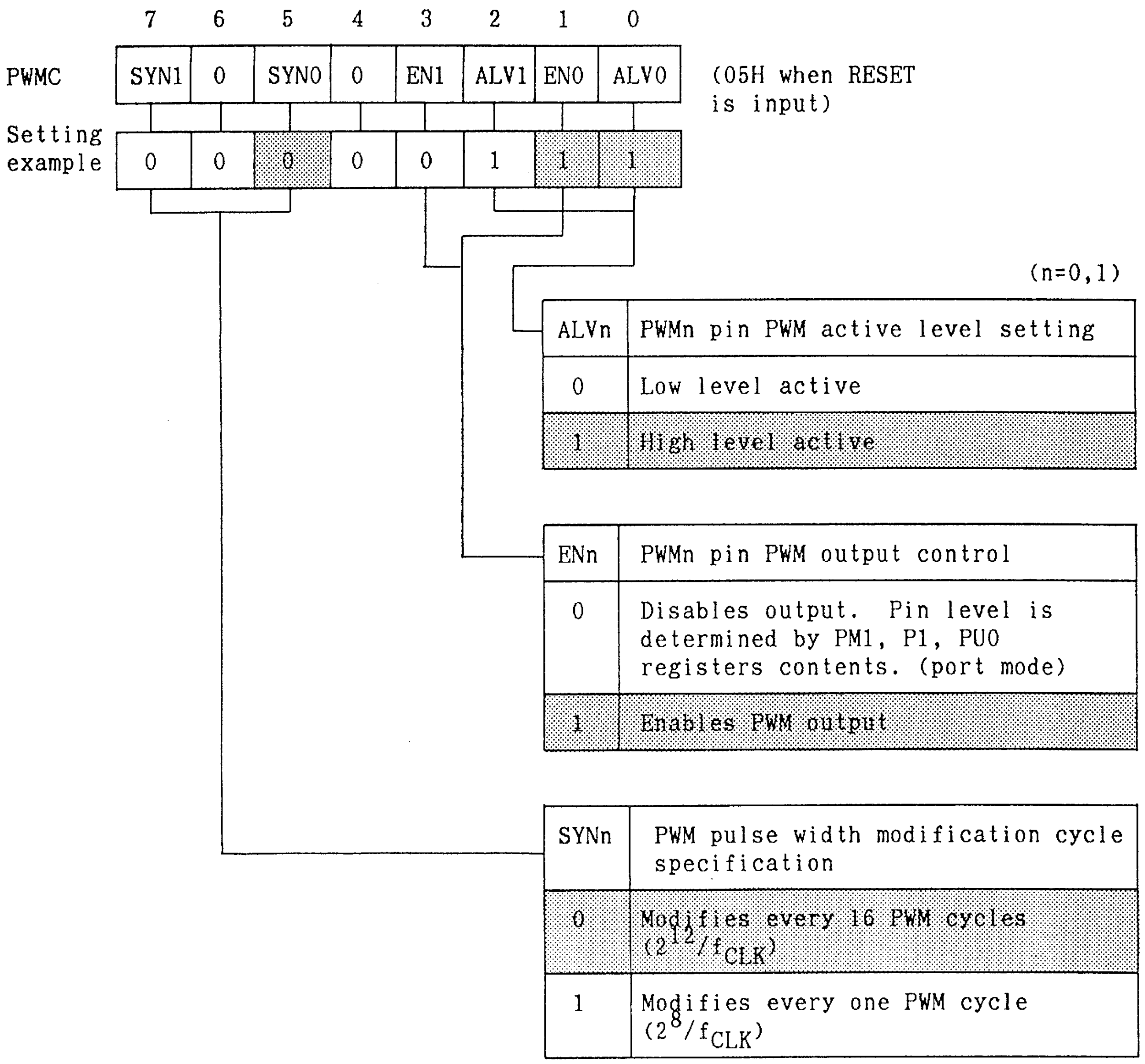
None.

(5) Stack used

None.

(6) Mode register setting example

PWM control register



	7	6	5	4	3	2	1	0	
PWMC	SYN1	0	SYN0	0	EN1	ALV1	EN0	ALV0	(05H when RESET is input)
Setting example	0	0	0	0	0	1	1	1	

(n=0,1)

ALVn	PWMn pin PWM active level setting
0	Low level active
[✓ 1]	[✓ High level active]

ENn	PWMn pin PWM output control
0	Disables output. Pin level is determined by PM1, P1, PU0 registers contents. (port mode)
[✓ 1]	[✓ Enables PWM output]

SYNn	PWM pulse width modification cycle specification
[✓ 0]	[✓ Modifies every 16 PWM cycles ($2^{12}/f_{CLK}$)]
1	Modifies every one PWM cycle ($2^{8}/f_{CLK}$)

(7) Program example

The following program is to output 80% duty PWM signal from the PWM0 pin. In this case, the value to be set to the PWM modulo register (PWM0) is calculated as follows:

$$\boxed{\begin{aligned}(4096 \times 0.8) - 1 &= 3275.8\\ &\doteqdot \underset{\sim\sim\sim}{3276}\ (\mathrm{CCCH})\end{aligned}}$$

The following program can output an 80% duty PWM pulse.

```
                :
                :
MOVW    PWM0,#0000H             ; set PWM0 duty 0%
MOV     PWMC,#00000111B         ; high active, 16 cycle, enable output
MOVW    PWM0,#0CCC0H            ; set PWM0 duty 80%
                :
                :
```

Notes: When setting a value to the modulo registers (PWM0, PWM1), the value must be set in 16-bit format (the lower 4 bits will be ignored).

CHAPTER 5 ASYNCHRONOUS SERIAL INTERFACE PROGRAM EXAMPLES

The 78K/II series asynchronous serial interface (UART) baud rate can be set in three different ways, depending on the device category. Table 5-1 shows the baud rate setting procedure for each device category.

Table 5-1 78K/II UART Baud Rate Setting Method

Setting procedure		Device category	
		uPD78214 series uPD78218A series uPD78234 series uPD78244 series	uPD78224 series
8-bit timer/counter 3 (internal baud rate generator*)		o	o
Baud rate generator	Baud rate generator input clock	o	x
Baud rate generator	Baud rate generator input clock + Frequency divider counter	o	x

*: uPD78224

The following introduces a UART program example, using the above three different baud rate setting procedures.
Description is made for each item in the following manner; (a) uPD78214 series, 78218A series, 78234 series, 78244 series, (b) uPD78224 series. For the uPD78214 series, 78218A series, 78234 series, 78244 series, both (a) and (b) programs can be used.
The program examples are for connection with the character display terminal (MD-910TM).
These programs receive characters from the MD-910TM keyboard and sends them to the MD-910TM to display on the screen.

5.1 Operation Outline

Fig. 5-1 shows how to connect the 78K/II series to the MD-910TM. The software operates in the loop back mode. The data lines (TXD, RXD) and handshake lines ($\overline{\text{CTS}}$, $\overline{\text{RTS}}$) are cross connected.

Fig. 5-1 Connection of 78K/II Series and MD-910TM

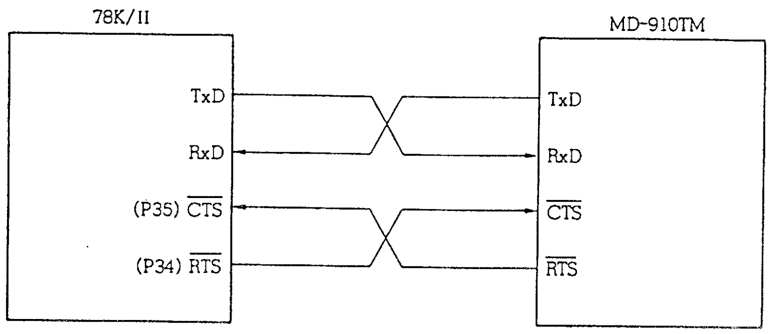


Table 5-2 indicates specifications, except for the baud rate.

Table 5-2 UART Program Example Specifications

Item	Specifications
Stop bits	2 bits
Character bits	8 bits
Parity bits	None
$\overline{\text{CTS}}$ pin	P35
$\overline{\text{RTS}}$ pin	P34

The following three work areas are used in this program. The work areas must be located in an area where short direct addressing is possible.

Table 5-3 Work Area Used by UART Program Example

Work area name	Application
RCV_DT	Receive data work area
TRN_DT	Transmit data work area

Table 5-4 Flags Used by UART Program Example

Flag name	Application
RCVFLG	Receive completion flag

5.2 Program Description

(1) Initialization processing
[label name: ASY214, ASY220]

(i) Uses P30 and P31 as control ports (RXD, TXD).

(ii) Sets $\overline{\text{RTS}}$ to "1" to release transmit request. In this case, no transmission is made from the MD-910TM.

(iii) Sets P35 (used as $\overline{\text{CTS}}$ pin) as input port, and sets P34 (used as $\overline{\text{RTS}}$ pin) as output port.

(iv) Initializes the asynchronous serial interface mode register.

(v) Sets the serial clock (baud rate). In these program examples, the settings are made in different ways, depending on the device category (uPD78214, uPD78220).

(vi) In the transmission processing, completion of the previous transmission is checked by the STIF (UART transmission complete) interrupt request flag. Therefore, the STIF interrupt flag must be set to "1" in advance.
For the UART of the 78K/II series, the STIF flag will not be set to "1" by the shift register (TXS) being empty. The STIF flag is set to "1" only when the data written to the shift register (TXS) is completely sent.

(vii) Releases the masking of the INTSR (UART receive complete) interrupt request.

(viii) Enables interrupt.

(ix) Sets $\overline{\text{RTS}}$ to "0" to make a transmit request.

(2) Receive processing [label name: RECIV]

(i) Sets $\overline{\text{RTS}}$ to "1" to release transmit request.

(ii) Reads receive data into the A register from the receive buffer (RXB).

(iii) Stores the receive data into the memory.

(iv) Sets the receive complete flag (RCVFLG) to "1".

Remarks: If a receive error occurs in the UART for the 78K/II, both the INTSR (UART receive complete) flag and the INTSER (UART receive error) flag will be set to "1" at a time. In this case, if both of these flags are to be processed by vector interrupt, INTSER will be accepted first. This is because INTSER has a higher priority.

(3) Transmit request processing [label name: CLRRTS]

(i) Sets $\overline{\text{RTS}}$ to "0" to make a transmit request to the MD-910TM.

(4) Transmit processing [label name: TRANS]

(i) Checks the completion of previous transmission by the STIF flag. STIF being set to "1" is awaited.
When STIF is set to 1, checks the next $\overline{\text{CTS}}$ after clearing.

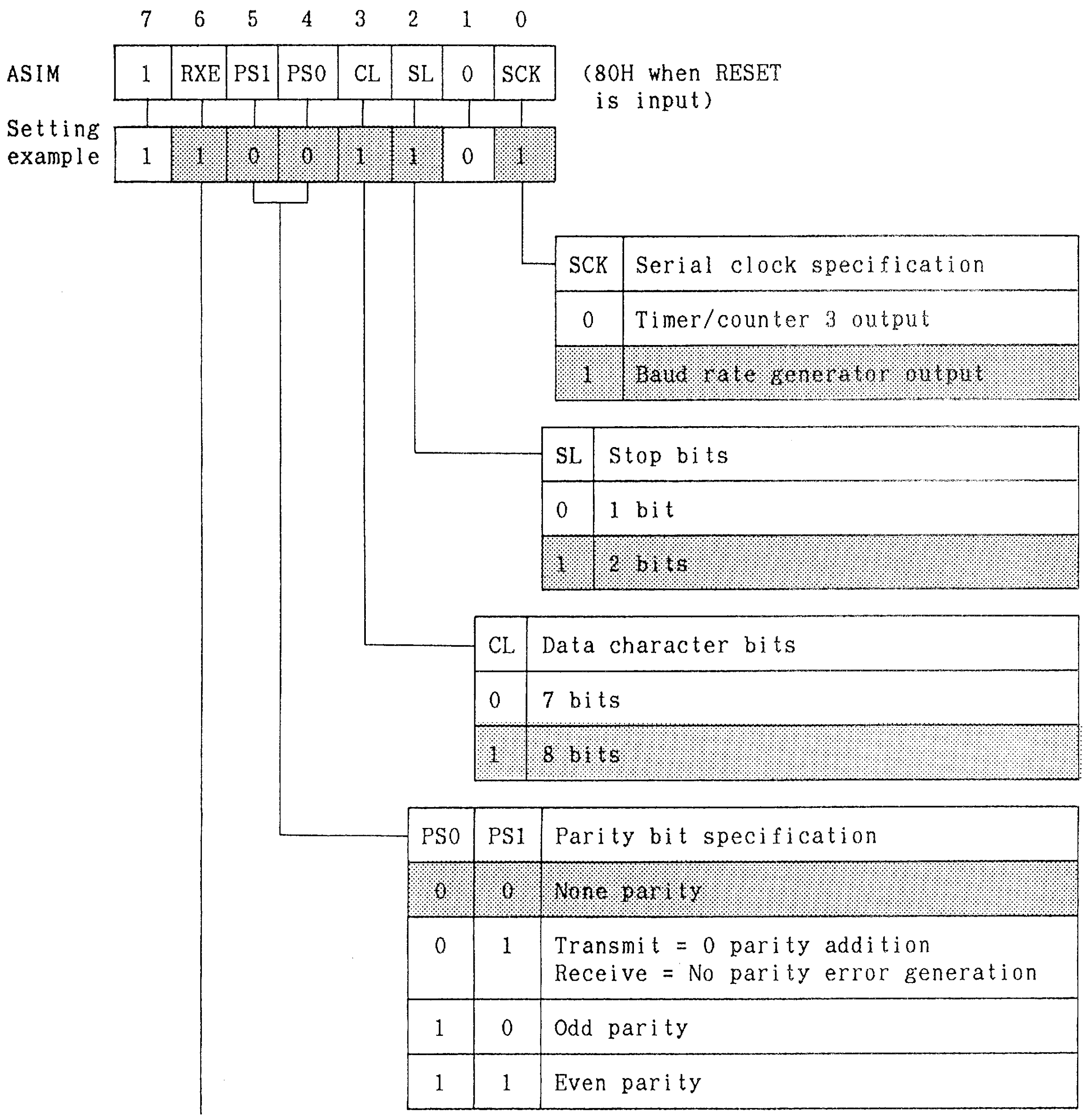
(ii) Checks that the MD-910TM is ready to receive. $\overline{\text{CTS}}$ becoming "0" is awaited.

(iii) Writes transmit data to the shift register (TXS).

5.3 Mode Register Setting Example

(a) Program for uPD78214

Asynchronous serial interface mode register



SCK	Serial clock specification
0	Timer/counter 3 output
1	Baud rate generator output

SL	Stop bits
0	1 bit
1	2 bits

CL	Data character bits
0	7 bits
1	8 bits

PS0	PS1	Parity bit specification
0	0	None parity
0	1	Transmit = 0 parity addition Receive = No parity error generation
1	0	Odd parity
1	1	Even parity

(Cont'd)

RXE	Receive enable control
0	Disable receive
1	Enable receive

Baud rate generator control register

	7	6	5	4	3	2	1	0	
BRGC	CE	TPS2	TPS1	TPS0	MDL3	MDL2	MDL1	MDL0	(00H when RESET is input)
Setting example	1	0	1	0	0	1	0	0	

4-bit modulo register setting value m (1 to 15)

TPS2	TPS1	TPS0	Frequency divider tap n
0	0	0	1/2
0	0	1	1/4
0	1	0	1/8
0	1	1	1/16
1	0	0	1/32
1	0	1	1/64
1	1	0	1/128
1	1	1	1/256

TPS2	4-bit counter and frequency divider operation
1	Stop
0	Count operation

Baud rate = $f_{XX}/2 \times 1/(m + 1) \times 1/n \times 1/16$

f_{XX}: System clock frequency

Port 3 mode control register

	7	6	5	4	3	2	1	0	
PMC3	PMC37	PMC36	PMC35	PMC34	PMC33	PMC32	PMC31	PMC30	(00H when reset)
Setting example	0	0	0	0	0	0	1	1	

PMC30	P30 pin control mode specification
0	Input/output port mode
1	RxD input mode

PMC31	P31 pin control mode specification
0	Input/output port mode
1	TxD output mode

PMC32	P32 pin control mode specification
0	Input/output port mode
1	$\overline{\text{SCK}}$ input/output mode

PMC33	P33 pin control mode specification
0	Input/output port mode
1	SO output/SB0 input/output mode

PMC3n	PM3n pin control mode specification (n = 4 to 7)
0	Input/output port mode
1	TOm output mode (m = 0 to 3)

Port 3 mode register

	7	6	5	4	3	2	1	0	
PM3	PM37	PM36	PM35	PM34	PM33	PM32	PM31	PM30	(FFH when RESET is input)
Setting example	0	0	1	0	0	0	0	0	

PM3n	P3n pin input/output mode specification (n = 0 to 7)
0	Output mode
1	Input mode

Interrupt request flag register H

	7	6	5	4	3	2	1	0	
IF0H	CSIIF	STIF	SRIF	SERIF	CIF20	PIF5	PIF4	CIF21	(00H when RESET is input)
Setting example	x	1	x	x	x	x	x	x	x: Not manipulated

IF	Interrupt request flag
0	Interrupt request not generated
1	Interrupt request generated

Interrupt mask register H

	7	6	5	4	3	2	1	0	
MK0H	CSIMK	STMK	SRMK	SERMK	CMK20	PMK5	PMK4	CMK21	(FFH when RESET is input
Setting example	x	x	0	x	x	x	x	x	x: Not manipulated

MK	Interrupt mask flag
0	Enables interrupt processing
1	Retains interrupt processing

(b) Program for uPD78224

Asynchronous serial interface mode register

	7	6	5	4	3	2	1	0
ASIM	1	RXE	PS1	PS0	CL	SL	0	SCK
Setting example	1	1	0	0	1	1	0	0

(80H when RESET is input)

SCK	Serial clock specification
0	Internal baud rate generator output
1	Internal clock $f_{CLK}/8$ (Note)

SL	Stop bits
0	1 bit
1	2 bits

CL	Data character bits
0	7 bits
1	8 bits

PS0	PS1	Parity bit specification
0	0	None parity
0	1	Transmit = 0 parity addition Receive = No parity error generation
1	0	Odd parity
1	1	Even parity

(Cont'd)

RXE	Receive enable control
0	Disable receive
1	Enable receive

Note: f_{CLK}: Internal system clock frequency ($f_{XX}/2$)

Prescaler mode register 0

	7	6	5	4	3	2	1	0
PRM0	PRS3	PRS2	PRS1	PRS0	0	0	0	0
Setting example	0	0	1	0	0	0	0	0

8-bit timer/counter 3^{Note}

PRS3	-	PRS1	PRS0	-	uPD78224
PRS33	PRS32	PRS31	PRS30	Other than above	-
0	0	0	0	$f_{CLK}/8$	$f_{CLK}/8$
0	0	0	1		
0	0	1	0	$f_{CLK}/16$	$f_{CLK}/16$
0	0	1	1	$f_{CLK}/32$	$f_{CLK}/32$
0	1	0	0	$f_{CLK}/64$	Not allowed
0	1	0	1	$f_{CLK}/128$	
0	1	1	0	$f_{CLK}/256$	
0	1	1	1	$f_{CLK}/512$	
1	0	0	0	Not allowed	High speed transfer mode
1	0	0	1		
1	0	1	0		
1	0	1	1		
1	1	x	x	Not allowed	

Note: Internal baud rate generator in case of uPD78224.

Remarks: f_{CLK}: Internal system cock

Timer control register 0

	7	6	5	4	3	2	1	0	
TMC0	Note 1 CE3	0	0	0	CE0	OVF0	0	0	(00H when reset)
Setting example	1	0	0	0	0	0	0	0	

16-bit timer/counter

OVF0	Timer/counter 0 overflow flag
0	No overflow
1	Overflow (count up from FFFFH to 0000H)

Remarks: This bit is reset only by the software.

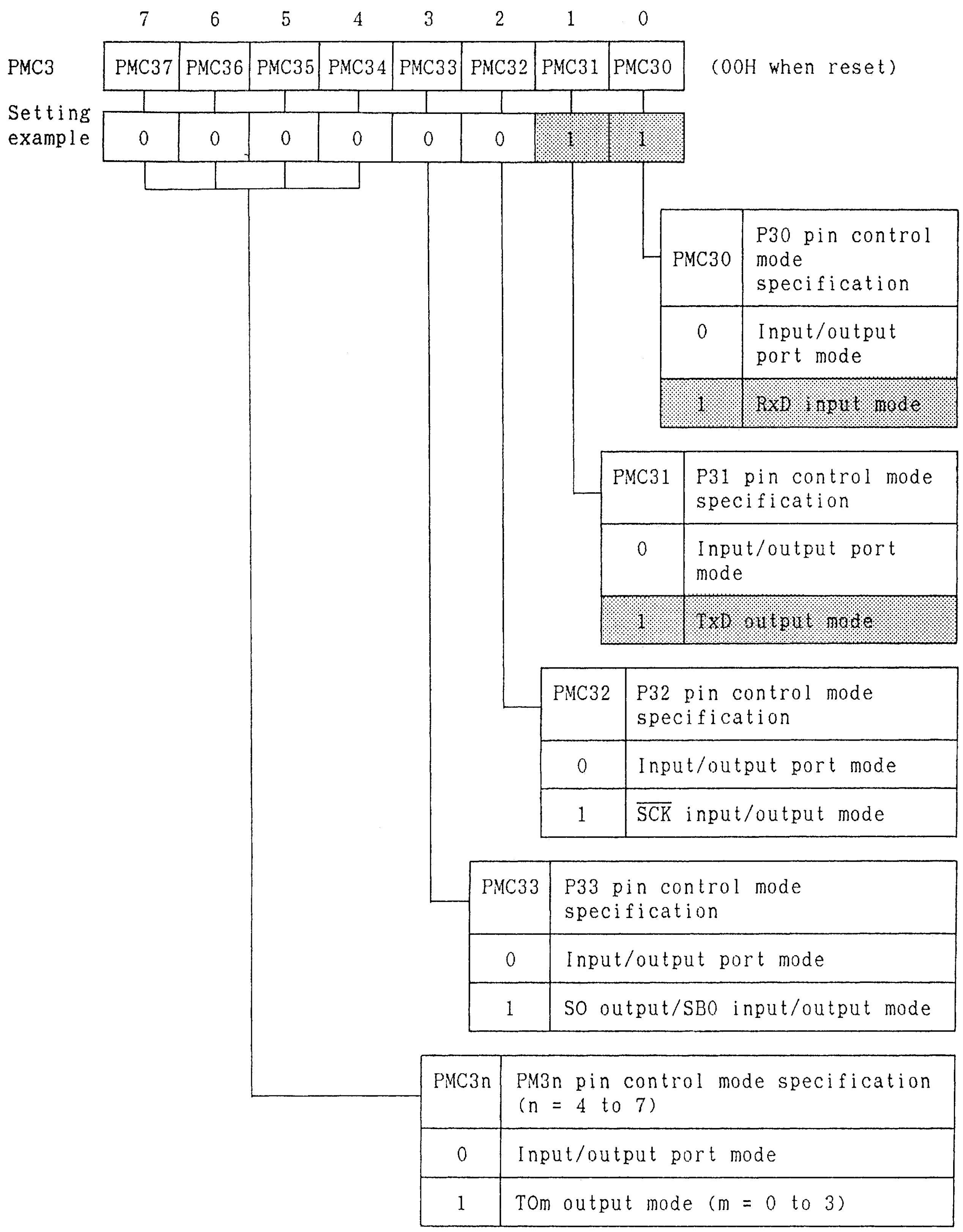
CE0	Timer/counter 0 count operation control
0	Stops count operation with timer/counter cleared
1	Enables count operation

8-bit timer/counter 3 (Note 2)

CE3	Timer/counter 3 count operation control
0	Stops count operation with timer/counter cleared
1	Enables count operation

Note 1: CE for the uPD78224.
2: Internal baud rate generator for the uPD78224.

Port 3 mode control register



	7	6	5	4	3	2	1	0	
PMC3	PMC37	PMC36	PMC35	PMC34	PMC33	PMC32	PMC31	PMC30	(00H when reset)
Setting example	0	0	0	0	0	0	1	1	

PMC30	P30 pin control mode specification
0	Input/output port mode
1	RxD input mode

PMC31	P31 pin control mode specification
0	Input/output port mode
1	TxD output mode

PMC32	P32 pin control mode specification
0	Input/output port mode
1	$\overline{\text{SCK}}$ input/output mode

PMC33	P33 pin control mode specification
0	Input/output port mode
1	SO output/SB0 input/output mode

PMC3n	PM3n pin control mode specification (n = 4 to 7)
0	Input/output port mode
1	TOm output mode (m = 0 to 3)

Port 3 mode register

	7	6	5	4	3	2	1	0	
PM3	PM37	PM36	PM35	PM34	PM33	PM32	PM31	PM30	(FFH when RESET is input)
Setting example	0	0	1	0	0	0	0	0	

PM3n	P3n pin input/output mode specification (n = 0 to 7)
0	Output mode
1	Input mode

Interrupt request flag register H

	7	6	5	4	3	2	1	0	
IF0H	CSIIF	STIF	SRIF	SERIF	CIF20	PIF5	PIF4	CIF21	(00H when RESET is input)
Setting example	x	1	x	x	x	x	x	x	x: Not manipulated

IF	Interrupt request flag
0	Interrupt request not generated
1	Interrupt request generated

Interrupt mask register H

	7	6	5	4	3	2	1	0	
MK0H	CSIMK	STMK	SRMK	SERMK	CMK20	PMK5	PMK4	CMK21	(FFH when RESET is input)
Setting example	x	x	0	x	x	x	x	x	x: Not manipulated

MK	Interrupt mask flag
0	Enables interrupt processing
1	Retains interrupt processing

(1) Parameters common to the both device categories

Input

TRN_DT: Transmit data. Set the transmit data in this memory and call [TRANS].

Output

RCV_DT: Receive data is stored in this memory.

RCVFLG: Receive complete flag. This flag is set to "1" each time a byte is received by the INTSR interrupt processing.

(2) Input parameters for each device

(a) Program for uPD78214

BRGCD: Value set to the baud rate generator control register (BRGCD). The baud rate generator input clock and its dividing ratio are set. The baud rate is calculated in the following manner:

$$\boxed{\text{Baud rate} = \frac{\text{Baud rate generator count clock}}{k + 1} \times \frac{1}{n} \times \frac{1}{16}}$$

k : Setting value of MDL3 to MDL0 bits for the BRGC register

1/n: Frequency divider tap

Therefore, the BRGCD value is determined by the k and n values.

(b) Program for uPD78224

PRMOD: Value set to the prescaler mode register (PRM0). This value determines the count clock for the internal baud rate generator.

CR30D: Value set to the 8-bit compare register (CR30).

The baud rate is calculated in the following manner:

$$\boxed{\text{Baud rate} = \frac{\text{Internal baud rate generator count clock}}{\text{CR30} + 1} \times \frac{1}{2} \times \frac{1}{16}}$$

Therefore, the CR30 value is determined in the following manner.

$$\boxed{\text{CR30} = \frac{\text{Count clock}}{\text{Baud rate}} \times \frac{1}{2} \times \frac{1}{16}}$$

(3) Registers used

(i)	ASY214, ASY210,	:	None
(ii)	RECIV	:	Register A
(iii)	CLRRTS	:	None
(iv)	TRANS	:	Register A

(4) Program examples

The previous program examples covered the UART initialization processing and transmit/receive processing. The following program examples are to set the baud rate and call the loop back processing.

(a) Program for uPD78214

This program example is to performed communication with the MD-910TM at 9600bps (f_{CLK} = 6 MHz).

$$\boxed{9600 = \frac{6 \times 10^6}{k + 1} \times \frac{1}{n} \times \frac{1}{16}}$$

From this expression, k is 9 and n is 4. Therefore, the baud rate generator count clock frequency is $f_{CLK}/10$, and the frequency divider tap is 1/4. Therefore, the value set to input parameter BRGCD becomes "10100100B".
In this case, the actual baud rate generated will be as calculated below. Therefore, the error will be -2.3% for 9600bps.

$$\boxed{\text{Baud rate} = \frac{6 \times 10^6}{9 + 1} \times \frac{1}{4} \times \frac{1}{16} = 9375 \text{ (bps)}}$$

The following is a program example.

```
        PUBLIC  TRN_DT,RCV_DT                           ; DATA WORK
        PUBLIC  RCVFLG                                  ; FLAG
        PUBLIC  BRGCD

        EXTRN   ASY214                                  ; PACKAGE
        EXTRN   TRANS,RECIV,CLRRTS                      ; PACKAGE
                .
                .
:       BAUD RATE :             9600bps
;       STOP BIT :              2bit
;       CHARACTER LENGTH :      8bit
;       PARITY :                NO PARITY

BRGCD   EQU     10100100B       ; BRGC DATA (9600bps,fclk/5)

        BSEG
RCVFLG  DBIT                    ; RECEIVE FLAG

ASY14_D DSEG    SADDR
TRN_DT: DS      1               ; TRANSMIT DATA
RCV_DT: DS      1               ; RECEIVE DATA

INTSRVT CSEG    AT 00022H
        DW      INTSR           ; INTSR
                .
                .
        CALL    !ASY214         ; <<< ASY214 >>>
SER_L1:
        BTCLR   RCVFLG,$SER_J1  ; CHECK RECEIVE FLAG
        BR      $SER_L1
SER_J1:
        MOV     TRN_DT,RCV_DT   ; TRANS DATA
        CALL    !TRANS          ; <<< TRANS >>>
        CALL    !CLRRTS         ; <<< CLRRTS >>>
        BR      SER_L1

INTSR:
        CALL    !RECIV          ; <<< RECIV >>>
        RETI
```

(b) Program for uPD78224

This program example is to performed communication with the MD-910TM at 4800bps (f_{CLK} = 5 MHz). If the internal baud rate generator count clock is specified to $f_{CLK}/16$, the value set to input parameter CR30D is calculated in the following manner. This method can also be used in the uPD78214.

$$\boxed{\text{CR30D} = \frac{5 \times 10^6/16}{4800} \times \frac{1}{16} \times \frac{1}{2} - 1 \doteqdot 1}$$

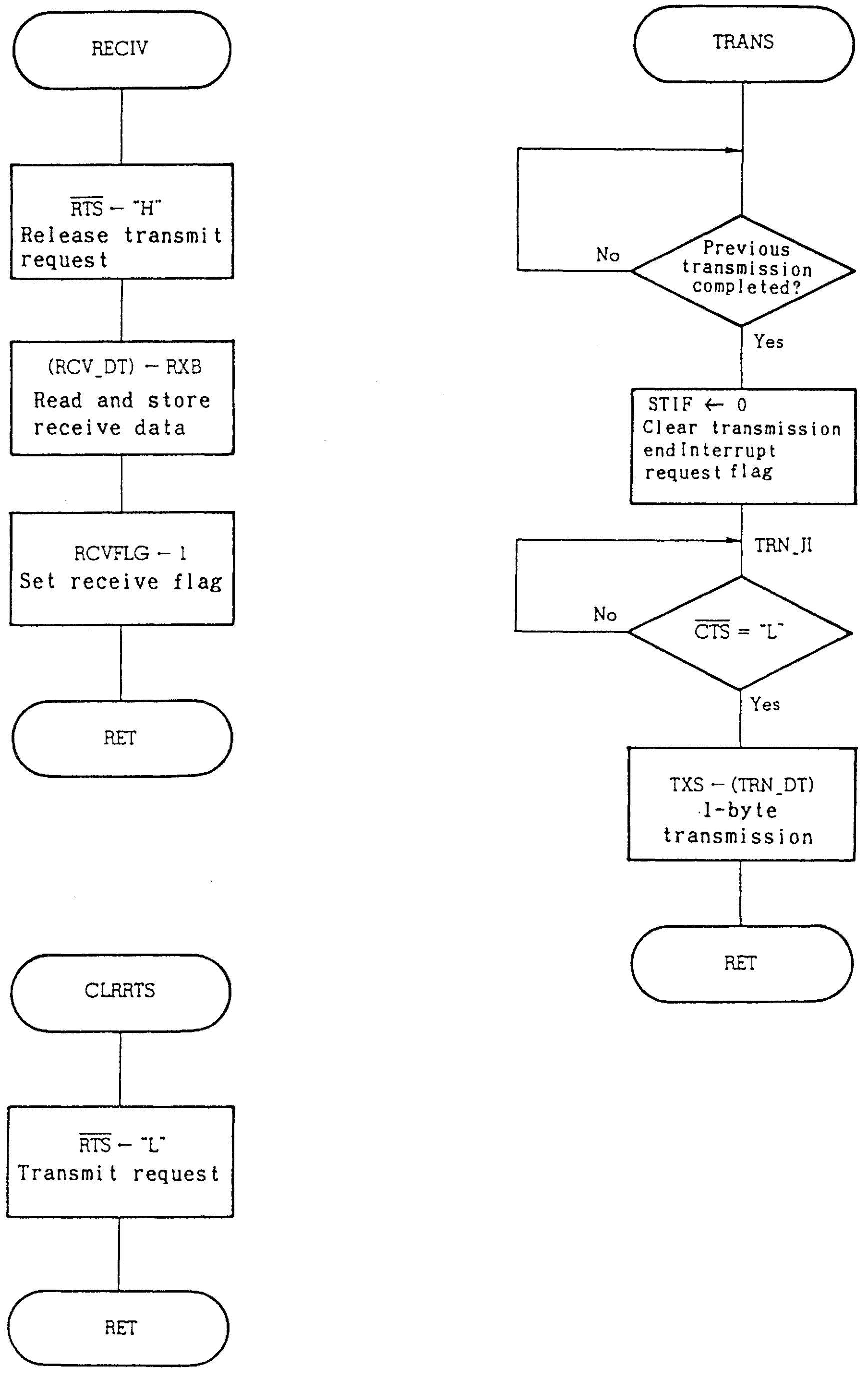
If "1" is set to CR30D, the actual baud rate generated will be as calculated below. Therefore, the error will be -1.7% for 4800bps.

$$\boxed{\text{Baud rate} = \frac{5 \times 10^6/16}{1 + 1} \times \frac{1}{16} \times \frac{1}{2} \doteqdot 4883 \text{ (bps)}}$$

The following is a program example.

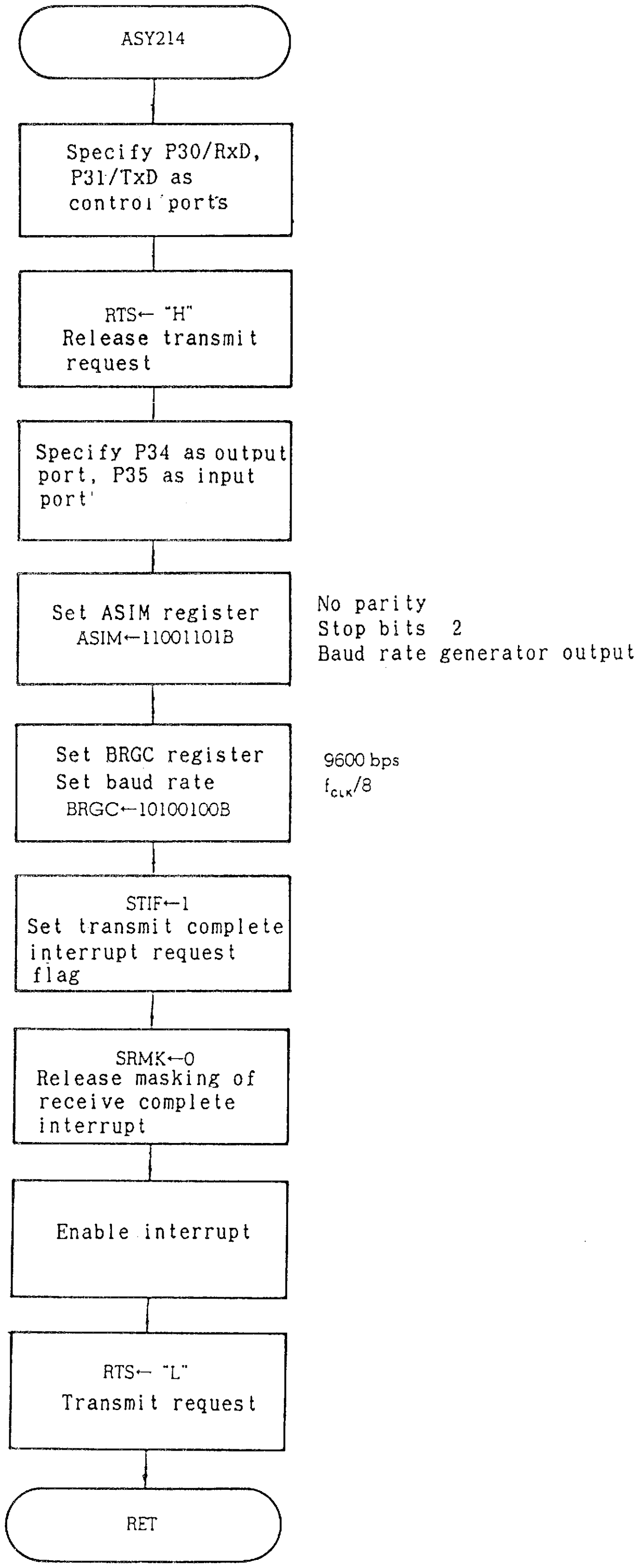
```
        PUBLIC  TRN_DT,RCV_DT                           ; DATA WORK
        PUBLIC  RCVFLG                                  ; FLAG
        PUBLIC  PRMOD,CR30D

        EXTRN   ASY220                                  ; PACKAGE
        EXTRN   TRANS,RECIV,CLRRTS                      ; PACKAGE
                .
                .
;       BAUD RATE :             4800bps
;       STOP BIT :              2bit
;       CHARACTER LENGTH :      8bit*
;       PARITY :                NO PARITY

PRMOD   EQU     00100000B       ; PRMO DATA (fclk/16)
CR30D   EQU     1               ; CR30 DATA

        BSEG
RCVFLG  DBIT                    ; RECEIVE FLAG

ASY20_D DSEG    SADDR
TRN_DT: DS      1               ; TRANSMIT DATA
RCV_DT: DS      1               ; RECEIVE DATA
                .
                .
INTSRVT CSEG    AT 00022H
        DW      INTSR           ; INTSR

        CSEG
                .
                .
        CALL    !ASY220         ; <<< ASY220 >>>
SER_L1:
        BTCLR   RCVFLG,$SER_J1  ; CHECK RECEIVE FLAG
        BR      SER_L1
SER_J1:
        MOV     TRN_DT,RCV_DT   ; TRANS DATA
        CALL    !TRANS          ; <<< TRANS >>>
        CALL    !CLRRTS         ; <<< CLRRTS >>>
        BR      SER_L1

INTSR:
        CALL    !RECIV          ; <<< RECIV >>>
        RETI
```

(5) Flow chart common to both different device categories

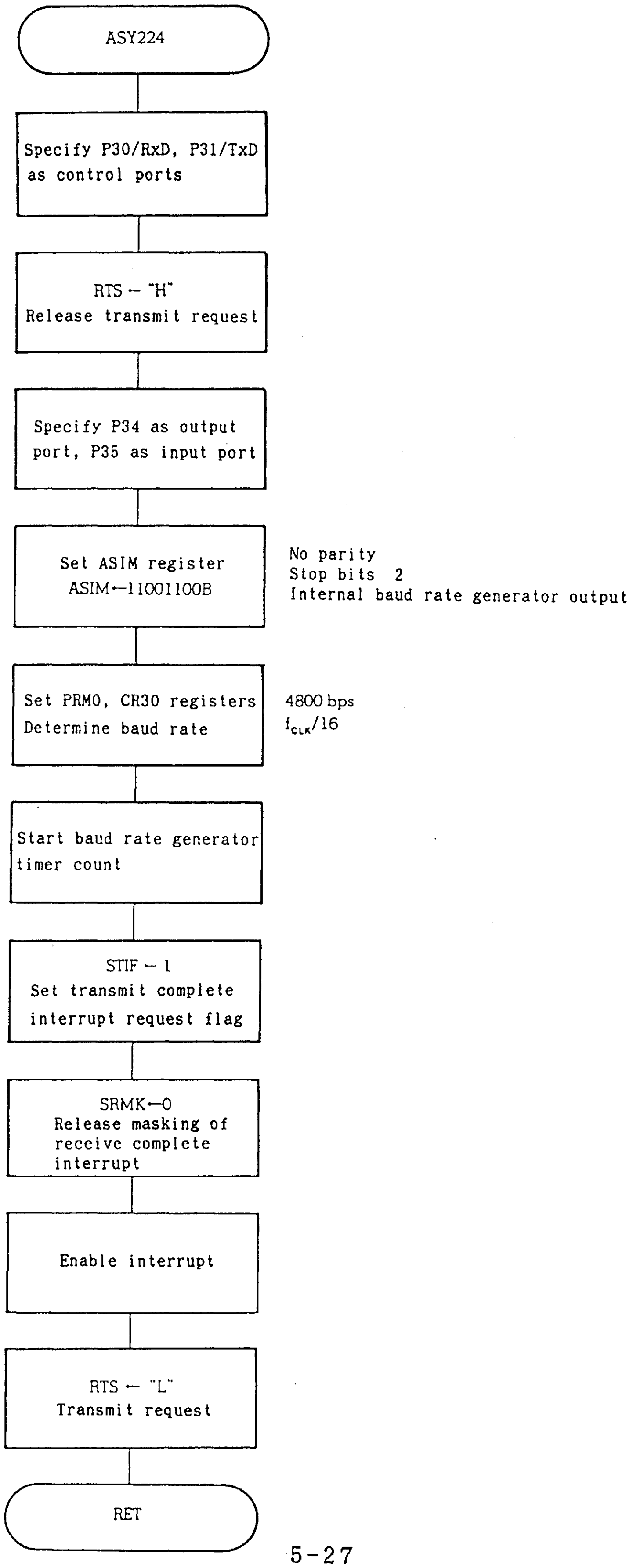


(6) Flow chart for each device

(a) Program for uPD78214


ASY214
Specify P30/RxD, P31/TxD as control ports
RTS← "H" Release transmit request
Specify P34 as output port, P35 as input port
Set ASIM register ASIM←11001101B
No parity
Stop bits 2
Baud rate generator output
Set BRGC register Set baud rate BRGC←10100100B
9600 bps
fCLK/8
STIF←1 Set transmit complete interrupt request flag
SRMK←0 Release masking of receive complete interrupt
Enable interrupt
RTS← "L" Transmit request
RET

(b) Program for uPD78224


ASY224
Specify P30/RxD, P31/TxD as control ports
RTS ← "H" Release transmit request
Specify P34 as output port, P35 as input port
Set ASIM register ASIM←11001100B
No parity
Stop bits 2
Internal baud rate generator output
Set PRM0, CR30 registers Determine baud rate
4800 bps
fCLK/16
Start baud rate generator timer count
STIF ← 1 Set transmit complete interrupt request flag
SRMK←0 Release masking of receive complete interrupt
Enable interrupt
RTS ← "L" Transmit request
RET

(7) Program list

(a) Program for uPD78214

```
        NAME    AS214

;**********************************************************
;       asynchronous serial interface for 78214
;                       at 12MHz
;**********************************************************

        PUBLIC  ASY214,RECIV,TRANS,CLRRTS
        EXTRN   TRN_DT,RCV_DT
        EXTRN   BRGCD
        EXTBIT  RCVFLG            ; receive flag
STIF    EQU     IF0H.6            ; INTST flag
SRMK    EQU     MK0H.5            ; mask of INTSR
CTS     EQU     P3.5              ; CTS for RS-232-C
RTS     EQU     P3.4              ; RTS for RS-232-C

        CSEG
ASY214:
        OR      PMC3,#00000011B ; P3.0,P3.1 = Control port
        SET1    RTS               ; RTS <- 1
        MOV     PM3,#00100000B  ; P3.5=IN , P3.4=OUT
        MOV     ASIM,#11001101B ; ASIM initialize
        MOV     BRGC,#LOW(BRGCD)  ; Baud rate = 9600bps
        SET1    STIF              ; dummy set INTST
        CLR1    SRMK              ; open mask of INTSR
        EI                        ; interrupt enable
        CLR1    RTS               ; receive enable
        RET

;**********************************************************
;       receive process
;**********************************************************

RECIV:
        SET1    RTS               ; RTS <- 1
        MOV     A,RXB             ; read receive data
        MOV     RCV_DT,A
        SET1    RCVFLG            ; set receive flag
        RET
CLRRTS:                           ; clear RTS
        CLR1    RTS
        RET
```

```
;*************************************************************
;       transmit process
;*************************************************************

TRANS:
        BTCLR   STIF,$TRN_J1     ; check complete of transmit
        BR      TRANS
TRN_J1:
        BT      CTS,$TRN_J1      ; check CTS
        MOV     A,TRN_DT
        MOV     TXS,A            ; transmit
        RET

        END
```

(b) Program for uPD78224

```
        NAME    AS220

;***********************************************************
;       asynchronous serial interface for 78220
;                       at 10MHz
;***********************************************************

        PUBLIC  ASY220,RECIV,TRANS,CLRRTS
        EXTRN   TRN_DT,RCV_DT
        EXTRN   PRMOD,CR30D
        EXTBIT  RCVFLG          ; receive flag
;
STIF    EQU     IF0H.6          ; INTST flag
SRMK    EQU     MK0H.5          ; mask of INTSR
CTS     EQU     P3.5            ; CTS for RS-232-C
RTS     EQU     P3.4            ; RTS for RS-232-C
;
        CSEG
ASY220:
        OR      PMC3,#00000011B ; P3.0,P3.1 = Control port
        SET1    RTS             ; RTS <- 1
        MOV     PM3,#00100000B  ; P3.5=IN , P3.4=OUT
        MOV     ASIM,#11001100B ; ASIM initialize
        MOV     PRM0,#LOW(PRMOD)  ; Baud rate = 4800bps
        MOV     CR30,#LOW(CR30D)
        MOV     TMC0,#10000000B
        SET1    STIF            ; dummy set INTST
        CLR1    SRMK            ; open mask of INTSR
        EI                      ; interrupt enable
        CLR1    RTS             ; receive enable
        RET

;***********************************************************
;       receive process
;***********************************************************

RECIV:
        SET1    RTS             ; RTS <- 1
        MOV     A,RXB           ; read receive data
        MOV     RCV_DT,A
        SET1    RCVFLG          ; set receive flag
        RET
CLRRTS:                         ; clear RTS
        CLR1    RTS
        RET
```

```
;***********************************************************
;       transmit process
;***********************************************************

TRANS:
        BTCLR   STIF,$TRN_J1     ; check complete of transmit
        BR      TRANS
TRN_J1:
        BT      CTS,$TRN_J1      ; check CTS
        MOV     A,TRN_DT
        MOV     TXS,A            ; transmit
        RET

        END
```

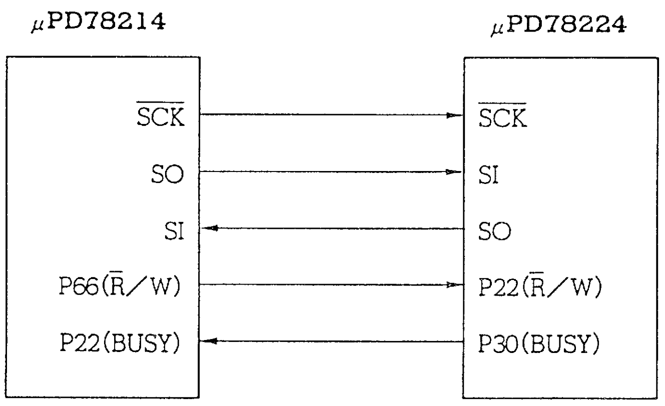
CHAPTER 6 THREE-LINE SERIAL INTERFACE

As a program example for the three-line serial interface, an example of data communication between devices in the 78K/II series is discussed in this section.

(1) Operation

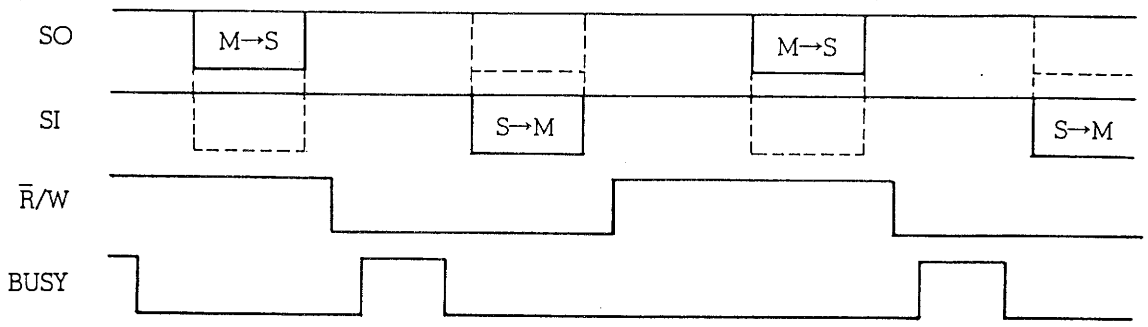
Devices, uPD78214 and uPD78224, are respectively used as master and slave. Fig. 6-1 shows the connection diagram.

Fig. 6-1 Connecting uPD78214 to uPD78224



The specifications for the communication protocol as follows:
The baud rate is set to 312.5 kbps at f_{CLK} = 5 MHz. As interconnections lines, the $\overline{R}$/W and BUSY pins are used. Data transfer is alternately performed between the master (uPD78214) and slave (uPD78224).
Fig. 6-2 shows the timing chart viewed from master.

Fig. 6-2 Timing Chart (Viewed from master)



Remarks: M → S: Data transfer from master to slave
S → M: Data transfer from slave to master

In the program example to be given in this section, the master and slave respectively use the following three work areas. These areas, however, must be located in an address range (FE20H to FEF7H) to which short direct addressing is applicable.

Table 6-1 Work Areas for Three-Line Serial Interface Program (Master)

Work area	Application
RCV_DT	Receive data work area
TRN_DT	Transfer data work area

Table 6-2 Flags for Three-Line Serial Interface Program (Master)

Flag	Application
TRNEND	Transfer end flag
RCVEND	Receive end flag

Table 6-3 Work Areas for Three-Line Serial Interface Program (Slave)

Work area	Application
RCV_DT	Receive data work area
TRN_DT	Transfer data work area

Table 6-4 Flags for Three-Line Serial Interface Program (Slave)

Flag	Application
RCVEND	Receive end flag

(2) Program ... Refer to (7) Program list

(a) Master initialization processing [label: CLKMIN]

(i) Pins P32 and P33 are used in the control port mode (and thus serve as $\overline{\text{SCK}}$ and SO pins, respectively).
(ii) PWFLAG is set to 1, so that data is transferred from the master to slave.
(iii) Pin P66 set in the output port mode.
(iv) The three-line serial interface is initialized, and transfer is enabled.
(v) The serial clock (baud rate) is set.
(vi) The falling edge of the BUSY (INTP1) pin is specified to be the valid edge.
(vii) The INTP1 interrupt request flag is cleared.
(viii) Interrupt request INTCSI (clock-synchronized serial interface transfer end) is unmasked.

(b) Master transfer processing [label: TRANS]

- (i) Transfer is postponed until the BUSY signal for the slave is cleared.
- (ii) Transfer is enabled.
- (iii) The data to be transferred is written to the shift register (SIO).

(c) Master reception processing [label: RECIVE]

Reception is enabled. When reception is enabled by the master in a communication system coupled by the clock-synchronized serial interface for 78K/II, and when data is read from the shift register (SIO), the serial clock is output by the master.

(d) Master transfer end interrupt processing
[label: ENDTR]

- (i) The RWFLAG flag status is inverted.
- (ii) If the RWFLAG flag is set to 1 as a result (indicating that the previous data has been transferred from the slave to master), execution branches to (v).
- (iii) The PIF1 flag is polled to check whether or not the slave has received data.
- (iv) Transfer is disabled and the transfer end flag TRNEND is set to 1. Execution is then returned to the main routine.
- (v) Reception is enabled, and receive data is written into the memory.
- (vi) The receive end flag RCVEND is set to 1 and execution is returned to the main routine.

(e) Slave initialization processing [label: CLKSIN]

- (i) Pins P32 and P33 are set in the control port mode (and thus serve as the $\overline{SCK}$ and SO pins, respectively).
- (ii) Since pin P30 is used as the BUSY pin, it is set in the BUSY status and output port mode. The BUSY signal must be cleared after the initialization processing has been completed.
- (iii) The three-line serial interface is initialized. Transfer is enabled.
- (iv) Both the rising and falling edges on the $\overline{R}$/W pin (INTP1) are specified to be valid edges.
- (v) The INTP1 interrupt request flag is cleared.
- (vi) Interrupt request INTCSI (clock-synchronized serial interface transfer end) is unmasked.

(f) Slave transfer processing [label: TRANS]

- (i) The data to be transferred is written to the shift register (SIO).
- (ii) The BUSY signal is cleared.

(g) Slave transfer end interrupt processing [label: ENDTR]

- (i) The processing is postponed, until the $\overline{R}$/W signal status changed.
- (ii) If the $\overline{R}$/W signal is logical 0, execution branches to (iv).
- (iii) The BUSY signal is cleared and execution is returned to the main routine.
- (iv) The BUSY signal is set.
- (v) The receive data is written into the memory.
- (vi) Receive end flag RCVEND is set to 1 and execution is returned.

(3) Mode register setting

(a) Setting mode registers for master

Port 3 mode control register

	7	6	5	4	3	2	1	0
PMC3	PMC37	PMC36	PMC35	PMC34	PMC33	PMC32	PMC31	PMC30
Setting example	0	0	0	0	1	1	0	0

(00H when RESET is input)

PMC30	Specifies control mode for pin P30
0	I/O port mode
1	RxD input mode

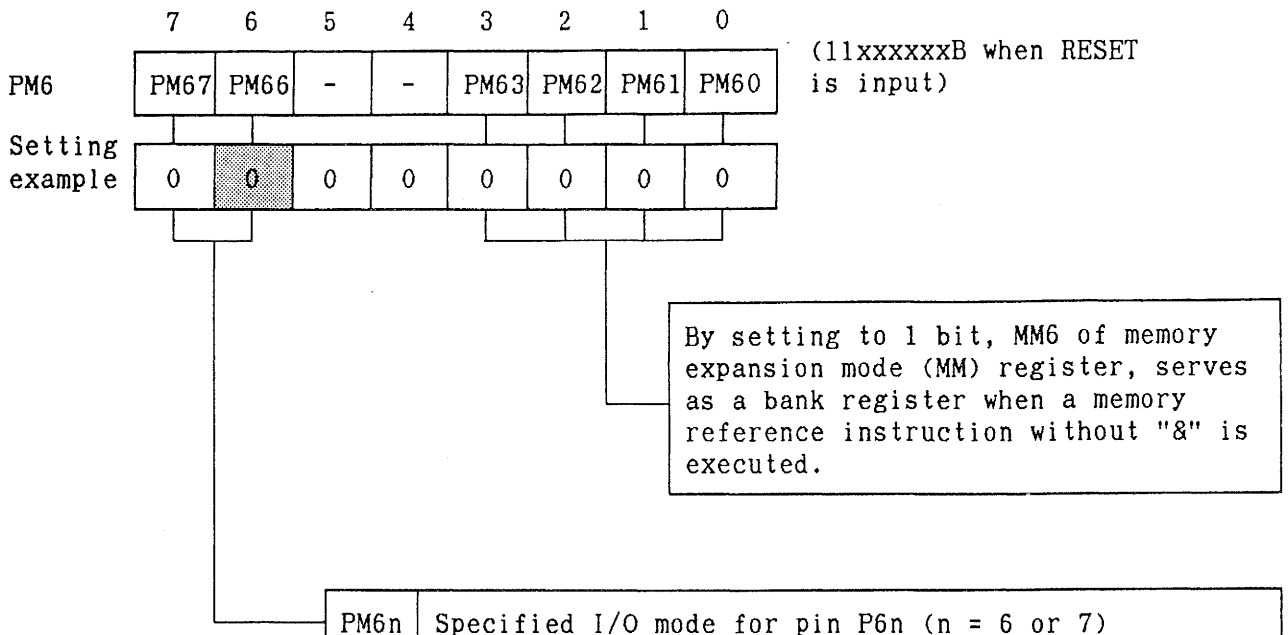
PMC31	Specifies control mode for pin P31
0	I/O port mode
1	TxD output mode

PMC32	Specifies control mode for pin P32
0	I/O port mode
1	$\overline{\text{SCK}}$ input/output mode

PMC33	Specifies control mode for pin P33
0	I/O port mode
1	SB0 input/output mode/SO output mode

PMC3n	Specifies control mode for pin P3n (n = 4 to 7)
0	I/O port mode
1	TOm output mode (m = 0 to 3)

Port 6 mode register


7 6 5 4 3 2 1 0
PM6
PM67 PM66 - - PM63 PM62 PM61 PM60
(11xxxxxxB when RESET is input)
Setting example
0 0 0 0 0 0 0 0
By setting to 1 bit, MM6 of memory expansion mode (MM) register, serves as a bank register when a memory reference instruction without "&" is executed.

PM6n	Specified I/O mode for pin P6n (n = 6 or 7)
0	Output mode (output buffer ON)
1	Input mode (output buffer OFF)

Note: Bits marked "-" can be 1 or 0.

Clock-synchronized serial interface mode register

	7	6	5	4	3	2	1	0	
CSIM	CRXE	CRXE	WUP	0	MOD1	0	CLS1	CLS0	(00H when RESET is input)
Setting example	1	0	0	0	0	0	0	1	

CLS1	CLS0	Select serial clock		$\overline{\text{SCK}}$ pin
0	0	External clock		Input
0	1	Inter-nal clock	8-bit timer/counter 3	Output
1	0		Note $f_{CLK}/32$	
1	1		Note $f_{CLK}/8$	

Note: f_{CLK}: internal system clock

MOD1	Selects operation mode for clock-synchronized serial interface
0	Three-line serial I/O mode
0	SBI mode

WUP	Specifies wake-up function
0	Generates interrupt request in all modes, each time serial transfer has been performed
1	Generates interrupt request only when an address has been received in SBI mode

CRXE	Enables/disables reception
0	Disables
1	Enables

(Cont'd)

CTXE	Enables/disables transfer
0	Disables
1	Enables

Remarks: f_{CLK} is the internal system clock.

Prescaler mode register 0

	7	6	5	4	3	2	1	0
PRM0	PRS3	PRS2	PRS1	PRS0	0	0	0	0
Setting example	0	0	0	0	0	0	0	0

8-bit timer/counter 3[Note]

PRS3	-	PRS1	PRS0	-	uPD78224
PRS33	PRS32	PRS31	PRS30	Other than above	-
0	0	0	0	$f_{CLK}/8$	$f_{CLK}/8$
0	0	0	1		
0	0	1	0	$f_{CLK}/16$	$f_{CLK}/16$
0	0	1	1	$f_{CLK}/32$	$f_{CLK}/32$
0	1	0	0	$f_{CLK}/64$	Not allowed
0	1	0	1	$f_{CLK}/128$	
0	1	1	0	$f_{CLK}/256$	
0	1	1	1	$f_{CLK}/512$	
1	0	0	0	Not allowed	High speed transfer mode
1	0	0	1		
1	0	1	0		
1	0	1	1		
1	1	x	x	Not allowed	

Note: Internal baud rate generator in case of uPD78224.

Remarks: f_{CLK}: Internal system clock

Timer control register 0

	7	6	5	4	3	2	1	0	
TMC0	CE	0	0	0	CE0	OVF0	0	0	(00H when RESET is input)
Setting example	▨ 1	0	0	0	0	0	0	0	

16-bit timer/counter

OVF0	TM0 overflow flag
0	No overflow
1	Overflow occurs (count up from FFFFH to 0000H)

Remarks: This bit is cleared by software only.

CE0	Controls TM0 count operation
0	Clears and stops count operation
1	Enables count operation

8-bit timer/count unit 3

CE	Controls TM3 count operation
0	Clears and stops count operation
▨ 1	▨ Enables count operation

Interrupt mask register H

	7	6	5	4	3	2	1	0	
MK0H	CSIMK	STMK	SRMK	SERMK	CMK20	PMK5	PMK4	CMK21	(FFH when RESET is input)
Setting example	0	x	x	x	x	x	x	x	x: Not used

MK	Interrupt mask flag
0	Enables interrupt processing
1	Keeps interrupt pending

External interrupt mode register 0

	7	6	5	4	3	2	1	0	
INTM0	ES21	ES20	ES11	ES10	ES01	ES00	0	ESNMI	(00H when RESET is input)
Setting example	0	0	0	0	0	0	0	0	

ESNMI	Specifies input edge to be detected on NMI pin
0	Falling edge
1	Rising edge

ES01	ES00	Specifies input edge to be detected on INTP0 pin
0	0	Falling edge
0	1	Rising edge
1	0	Inhibited
1	1	Both falling and rising edges

ES11	ES10	Specifies input edge to be detected on INTP1 pin
0	0	Falling edge
0	1	Rising edge
1	0	Inhibited
1	1	Both falling and rising edges

ES21	ES20	Specifies input edge to be detected on INTP2 pin
0	0	Falling edge
0	1	Rising edge
1	0	Inhibited
1	1	Both falling and rising edges

(b) Slave setting mode registers example

Port 3 mode control register

	7	6	5	4	3	2	1	0
PMC3	PMC37	PMC36	PMC35	PMC34	PMC33	PMC32	PMC31	PMC30
Setting example	0	0	0	0	1	1	0	0

(00H when RESET is input)

PMC30	Specifies control mode for pin P30
0	I/O port mode
1	RxD input mode

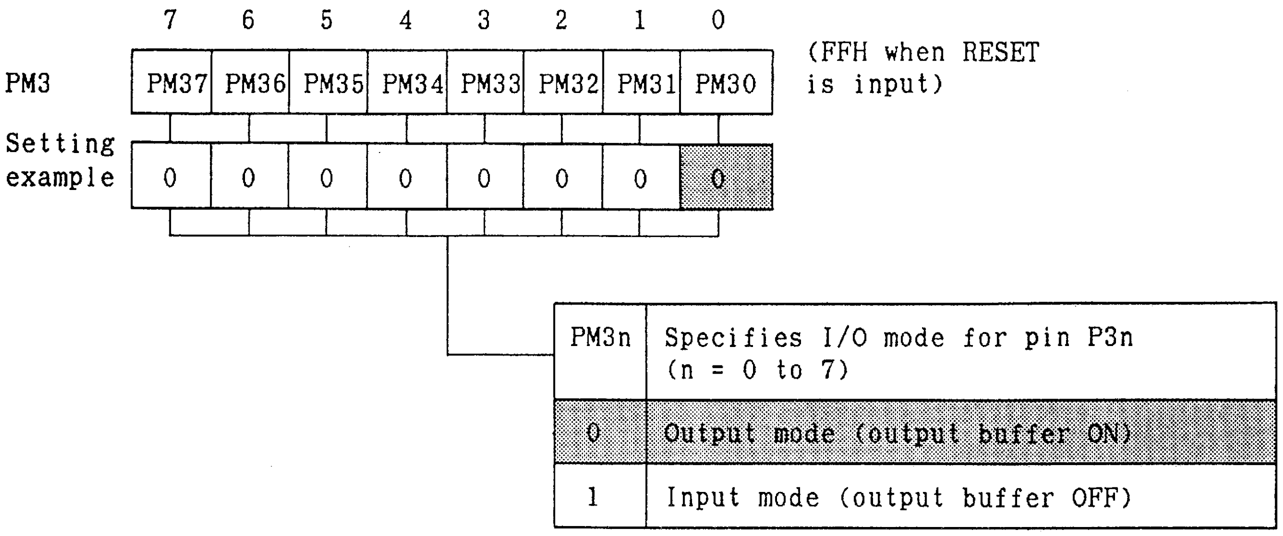
PMC31	Specifies control mode for pin P31
0	I/O port mode
1	TxD output mode

PMC32	Specifies control mode for pin P32
0	I/O port mode
1	$\overline{SCK}$ input/output mode

PMC33	Specifies control mode for pin P33
0	I/O port mode
1	SB0 input/output mode/SO output mode

PMC3n	Specifies control mode for pin P3n (n = 4 to 7)
0	I/O port mode
1	TOm output mode (m = 0 to 3)

Port 3 mode register


7 6 5 4 3 2 1 0
PM3
PM37 PM36 PM35 PM34 PM33 PM32 PM31 PM30
(FFH when RESET is input)
Setting example
0 0 0 0 0 0 0 0

PM3n	Specifies I/O mode for pin P3n (n = 0 to 7)
0	Output mode (output buffer ON)
1	Input mode (output buffer OFF)

Clock-synchronized serial interface mode register

	7	6	5	4	3	2	1	0	
CSIM	CRXE	CRXE	WUP	0	MOD1	0	CLS1	CLS0	(00H when RESET is input)
Setting example	1	1	0	0	0	0	0	0	

CLS1	CLS0	Select serial clock		$\overline{\text{SCK}}$ pin
0	0	External clock		Input
0	1	Inter-nal clock	8-bit timer/ counter 3	Output
1	0		Note $f_{CLK}/32$	
1	1		Note $f_{CLK}/8$	

Note: f_{CLK}: internal system clock

MOD1	Selects operation mode for clock-synchronized serial interface
0	Three-line serial I/O mode
0	SBI mode

WUP	Specifies wake-up function
0	Generates interrupt request in all modes, each time serial transfer has been performed
1	Generates interrupt request only when an address has been received in SBI mode

CRXE	Enables/disables reception
0	Disables
1	Enables

(Cont'd)

CTXE	Enables/disables transfer
0	Disables
1	Enables

Remarks: f_{CLK} is the internal system clock.

Interrupt mask register H

	7	6	5	4	3	2	1	0	
MK0H	CSIMK	STMK	SRMK	SERMK	CMK20	PMK5	PMK4	PMK21	(FFH when RESET is input)
Setting example	0	x	x	x	x	x	x	x	x: Not used

MK	Interrupt mask flag
0	Enables interrupt processing
1	Keeps interrupt pending

External interrupt mode register 0

	7	6	5	4	3	2	1	0	
INTM0	ES21	ES20	ES11	ES10	ES01	ES00	0	ESNMI	(00H when RESET is input)
Setting example	0	0	1	1	0	0	0	0	

ESNMI	Specifies input edge to be detected on NMI pin
0	Falling edge
1	Rising edge

ES01	ES00	Specifies input edge to be detected on INTP0 pin
0	0	Falling edge
0	1	Rising edge
1	0	Inhibited
1	1	Both falling and rising edges

ES11	ES10	Specifies input edge to be detected on INTP1 pin
0	0	Falling edge
0	1	Rising edge
1	0	Inhibited
1	1	Both falling and rising edges

ES21	ES20	Specifies input edge to be detected on INTP2 pin
0	0	Falling edge
0	1	Rising edge
1	0	Inhibited
1	1	Both falling and rising edges

(4) Input/output parameter

(a) PRMOD: A value to be set in the prescaler mode register (PRM0)

CR30D: A value to be set in the 8-bit compare register (CR30)

PRMOD and CR30D are parameters for setting the serial clock (baud rate), which can be calculated as follows:

$$\boxed{\begin{aligned}\text{Baud rate} &= \frac{\text{TM3 count clock}}{\text{CR30} + 1} \times \frac{1}{2} \\ &= \frac{5 \times 10^6/8}{\text{CR30} + 1} \times \frac{1}{2}\end{aligned}}$$

Therefore, to set the baud rate to 312.5 kbps, PRM0 = 00000000B and CR30 = 0.

(b) TRN_DT: Data to be transferred. Set the data to be transferred in this memory area and call [TRANS].

(c) RCV_DT: Data to be received is stored in this memory area.

(d) TRNEND: Transfer end flag. This flag is set to 1, each time 1 byte of data has been transferred.

(e) RCVEND: Receive end flag. This flag is set to 1, each time 1 byte of data has been received by performing INTCSI interrupt processing.

(5) Registers

Master

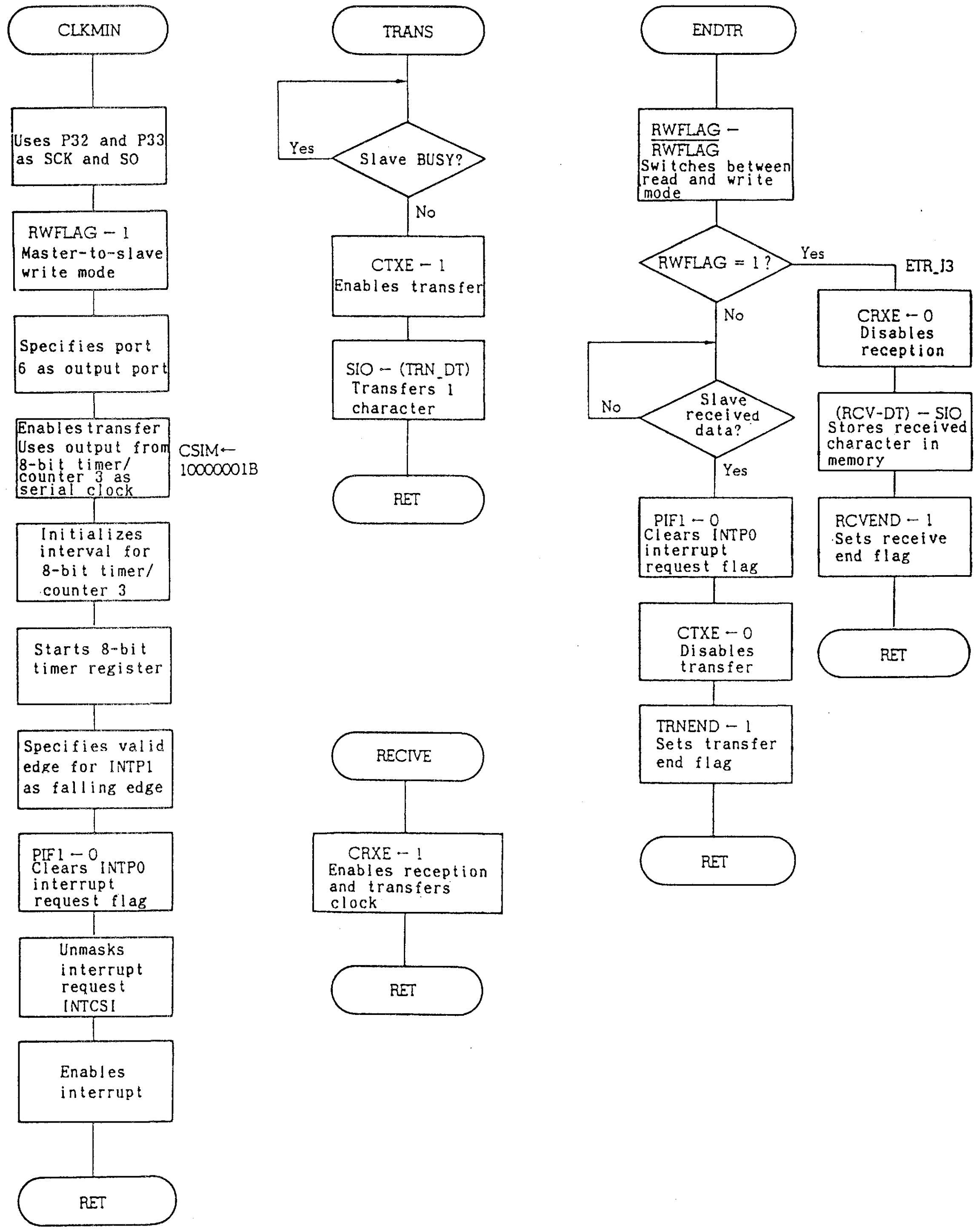
(a)	CLKMIN:	None
(b)	TRANS :	Register A
(c)	RECIVE:	None
(d)	ENDTR :	Register A

Slave

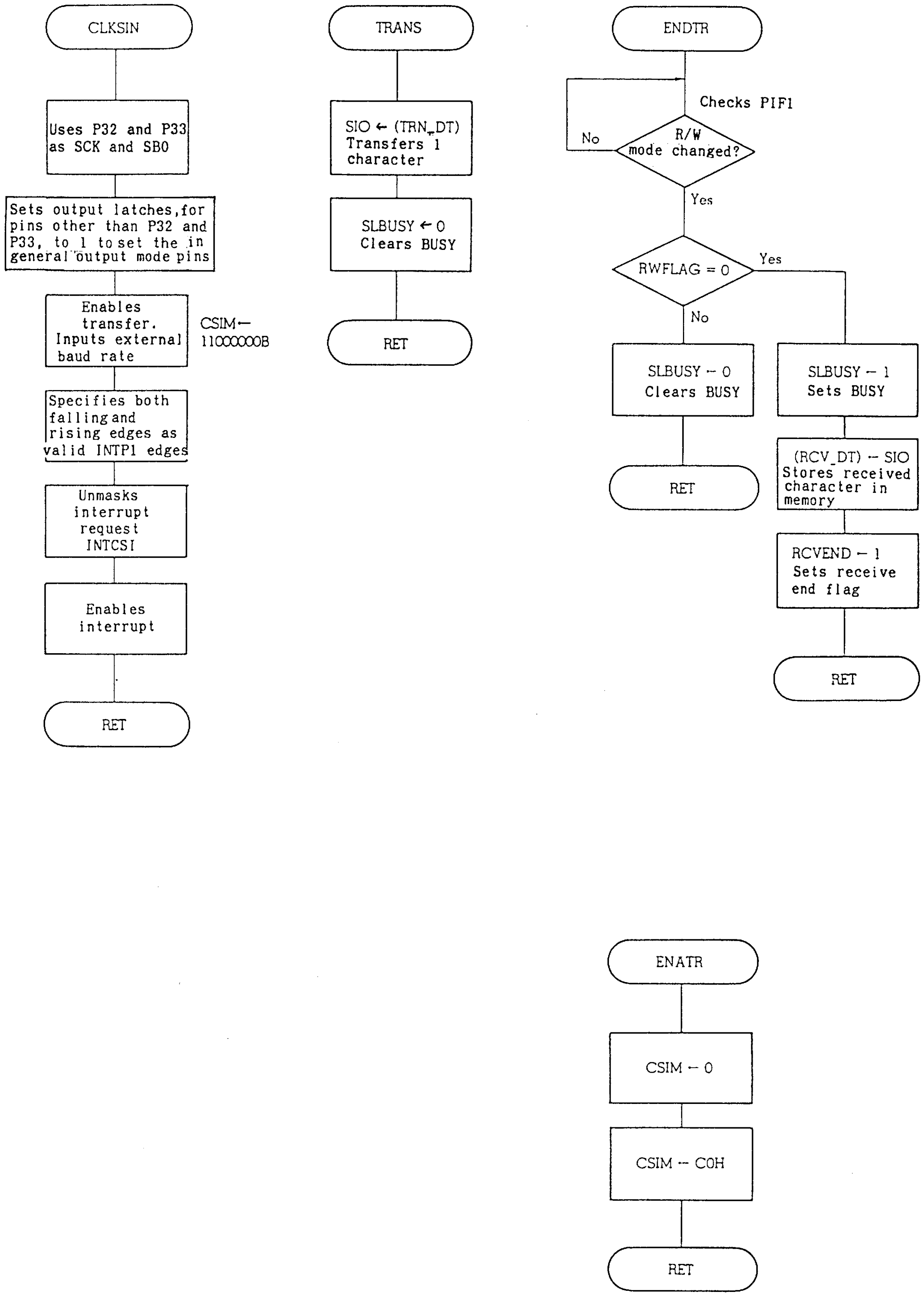
(a)	CLKSIN:	Register A
(b)	TRANS :	Register A
(c)	ENDTR :	Register A

(6) Flowchart

(a) Master


CLKMIN
Uses P32 and P33 as SCK and SO
RWFLAG ← 1 Master-to-slave write mode
Specifies port 6 as output port
Enables transfer Uses output from 8-bit timer/counter 3 as serial clock
CSIM← 10000001B
Initializes interval for 8-bit timer/counter 3
Starts 8-bit timer register
Specifies valid edge for INTP1 as falling edge
PIF1 ← 0 Clears INTP0 interrupt request flag
Unmasks interrupt request INTCSI
Enables interrupt
RET
TRANS
Slave BUSY?
Yes
No
CTXE ← 1 Enables transfer
SIO ← (TRN_DT) Transfers 1 character
RET
RECIVE
CRXE ← 1 Enables reception and transfers clock
RET
ENDTR
RWFLAG ← $\overline{\text{RWFLAG}}$ Switches between read and write mode
RWFLAG = 1 ?
Yes
ETR_J3
No
Slave received data?
No
Yes
PIF1 ← 0 Clears INTP0 interrupt request flag
CTXE ← 0 Disables transfer
TRNEND ← 1 Sets transfer end flag
RET
CRXE ← 0 Disables reception
(RCV-DT) ← SIO Stores received character in memory
RCVEND ← 1 Sets receive end flag
RET

(b) Slave


CLKSIN
Uses P32 and P33 as SCK and SB0
Sets output latches, for pins other than P32 and P33, to 1 to set the in general output mode pins
Enables transfer. Inputs external baud rate
CSIM← 11000000B
Specifies both falling and rising edges as valid INTP1 edges
Unmasks interrupt request INTCSI
Enables interrupt
RET
TRANS
SIO ← (TRN_DT) Transfers 1 character
SLBUSY ← 0 Clears BUSY
RET
ENDTR
Checks PIF1
R/W mode changed?
No
Yes
RWFLAG = 0
Yes
No
SLBUSY ← 0 Clears BUSY
RET
SLBUSY ← 1 Sets BUSY
(RCV_DT) ← SIO Stores received character in memory
RCVEND ← 1 Sets receive end flag
RET
ENATR
CSIM ← 0
CSIM ← C0H
RET

(7) Program list

(a) Master

```
        NAME    CLKDMR
;
;**********************************************************
;       clocked serial I/O interface
;**********************************************************
;
        PUBLIC  CLKMIN,TRANS,RECIVE,ENDTR
        EXTRN   TRN_DT,RCV_DT
        EXTRN   PRMOD,CR30D
        EXTBIT  TRNEND,RCVEND

CTXE    EQU     CSIM.7           ; transmit enable
CRXE    EQU     CSIM.6           ; receive enable
CSIMK   EQU     MKOH.7           ; mask of INTCSI
PIF1    EQU     IFOL.1           ; slave busy flag
SLBUSY  EQU     P2.2             ; SLAVE BUSY FLAG
RWFLAG  EQU     P6.6             ; R/W FLAG 0:READ 1:WRITE
;
;       *** initialize of clocked serial I/O ***
;
        CSEG
CLKMIN:
        MOV     PMC3,#00001100B ; P32 -> SCK  ,  P33 -> SO
        SET1    RWFLAG           ; default=write mode
        MOV     PM6,#00H         ; P66 -> R/W
        MOV     CSIM,#10000001B ; CSIM initialize ( RCV:bad , TRN:ok )
        MOV     PRMO,#LOW(PRMOD)  ; initialize PRMO
        MOV     CR30,#LOW(CR30D)  ; initialize CR30
        MOV     TMCO,#10000000B ; internal serial clock start
        MOV     INTMO,#00000000B
                                 ; INTPO rise edge enable
        CLR1    PIF1             ; clear INTP1 flag
        CLR1    CSIMK            ; INTCSI enable
        EI                       ; interrupt enable

        RET
;
;       *** trans process ***
;
TRANS:
        BT      SLBUSY,$TRANS    ; check busy
        SET1    CTXE             ; transmit enable
        MOV     A,TRN_DT
        MOV     SIO,A            ; transmit data
        RET
;
;       *** receive data ***
;
RECIVE:
        SET1    CRXE             ; receive enable
        RET
```

```
;
;       *** transmit/receive complete ***
;
;
ENDTR:
        NOT1    RWFLAG          ; change read/write mode
        BT      RWFLAG,$ETR_J3  ; check R/W mode
ETR_J1:
        BTCLR   PIF1,$ETR_J2    ; slave receive check
        BR      ETR_J1
ETR_J2:
        CLR1    CTXE            ; transmit disable
        SET1    TRNEND          ; set transmit complete flag
        RET
ETR_J3:
        CLR1    CRXE            ; receive disable
        MOV     A,SIO           ; read data
        MOV     RCV_DT,A
        SET1    RCVEND          ; set receive end flag
        RET
;
        END
```

(b) Slave

```
        NAME    CLKDSR
;
;************************************************************
;       clocked serial I/O interface
;************************************************************
;
        PUBLIC  CLKSIN,TRANS,ENDTR
        EXTRN   TRN_DT,RCV_DT
        EXTBIT  RCVEND

PIF1    EQU     IF0L.1          ; INTP0 flag
CSIMK   EQU     MK0H.7          ; mask of INTCSI
SLBUSY  EQU     P3.0            ; SLAVE BUSY FLAG
RWFLAG  EQU     P2.2            ; R/W FLAG 0:READ 1:WRITE
;
;       *** initialize of clocked serial I/O ***
;
        CSEG
CLKSIN:
        MOV     PMC3,#00001100B ; P32 -> SCK  ,  P33 -> SO
        MOV     P3,#0FFH        ; initialize P3
        MOV     PM3,#0
        MOV     CSIM,#11000000B ; CSIM initialize ( RCV:ok , TRN:ok )
        MOV     INTM0,#00110000B
                                ; INTP0 all edge enable
        CLR1    PIF1            ; clear INTP1 flag
        CLR1    CSIMK           ; INTCSI enable
        EI                      ; interrupt enable

        RET
;
;       *** transmit ***
;
TRANS:
        MOV     A,TRN_DT
        MOV     SIO,A           ; transmit data
        CLR1    SLBUSY          ; busy clear
        RET
;
;       *** transmit/receive complete ***
;
ENDTR:
        BTCLR   PIF1,$EDT_J1    ; check R/W edge
        BR      ENDTR
EDT_J1:
        BF      RWFLAG,$EDT_J2  ; check R/W mode
        CLR1    SLBUSY          ; clear busy
        RET
EDT_J2:
        SET1    SLBUSY          ; SET BUSY
        MOV     A,SIO           ; receive data
        MOV     RCV_DT,A
        SET1    RCVEND
        RET
;
        END
```

CHAPTER 7 INTERRUPT PROCESSING PROGRAM EXAMPLES

78K/II series microcomputers can select either of the two processing modes listed in Table 7-1 as interrupt processing.

Table 7-1 Interrupt Processing for 78K/II Series

Processing mode	Processed by:	Processing	PC and PSW contents
Vector interrupt	Software	Branches to processing routine	Saved and restored
Macro service	Firmware	Executes data transfer processing between memory and I/O	Retained

To compare the above two interrupt processing modes, this section presents program examples, each of which performs following processing:

(i) Asynchronous serial interface (UART) reception processing

(ii) Parallel data input, in synchronization with external interrupt request

(iii) Open loop control for stepping motor (macro service only)

In the macro service mode, (i) is processed by macro service A, (ii) is processed by B, and(iii) is processed by C.

7.1 UART Reception Processing

An example program that continuously inputs data through the internal UART for a 78K/II series microcomputer is discussed in this section. Internal system clock f_{CLK} is set to 6 MHz in this example.

(1) Operation

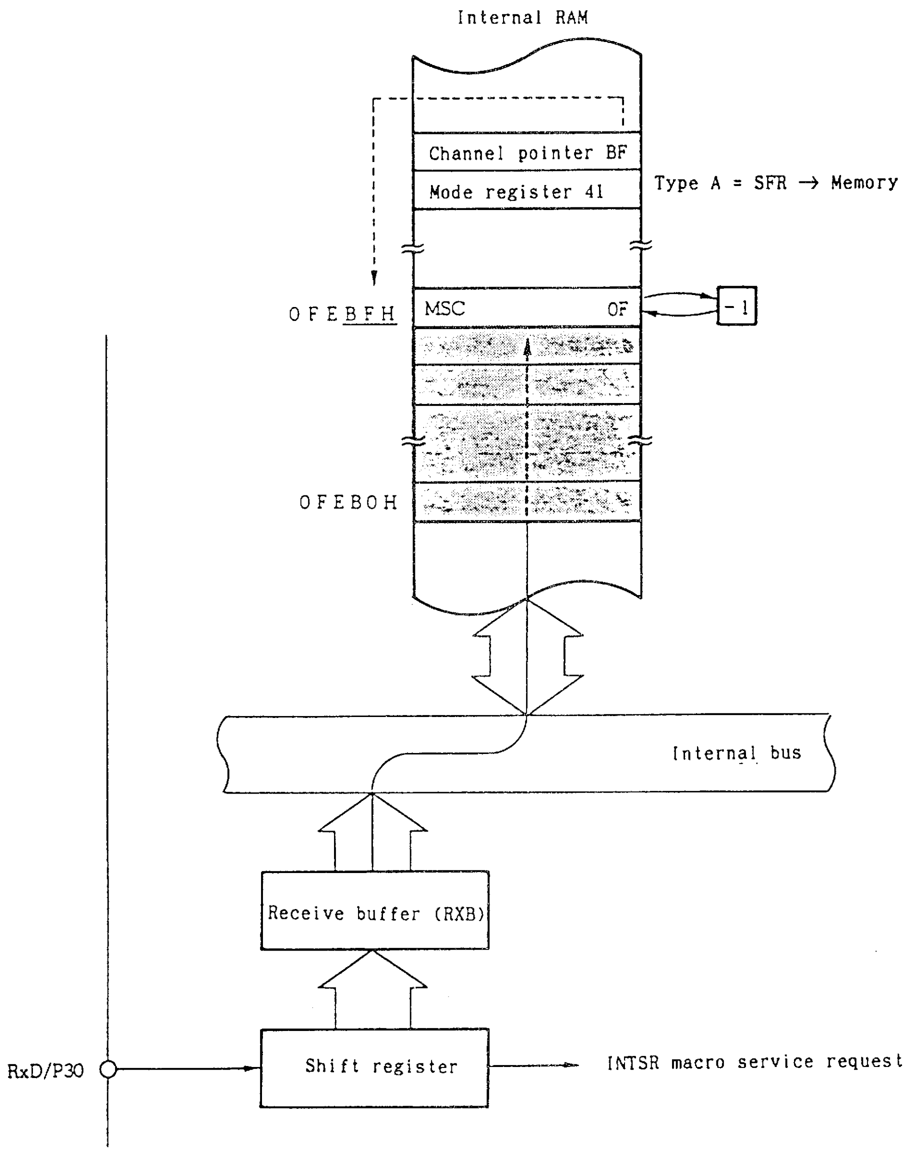
(a) Macro service

Fifteen-byte data received through the UART is transferred to the buffer area of the internal RAM by using macro service type A. The macro service counter (MSC) is stored in address FEBFH. Initially, a 15 (FH) value is set in the MSC. When macro service end interrupt processing is performed, the initial values for MSC and channel pointer (CHP) are set again.

(b) Vector interrupt

By using the same work area as that used for (a) macro service, the same operation as above is simulated by means of vector interrupt.

Fig. 7-1 Memory Mapping when Macro Service is Used (UART Reception Processing)



(2) Program ... Refer to (8) Program list

Macro service

(a) Macro service initialization processing
[label: TY_AMI]

(i) The macro service mode register is set.
Macro service type A is set and transfer direction is set from SFR to memory.

(ii) The channel pointer is set.

(iii) The number of times transfer is to be performed is set in the macro service counter.

(iv) Processing, performed in response to interrupt request INTSR, is defined as macro service.

(v) The interrupt is enabled.

(b) Macro service end interrupt processing [label: TY_AMR]

This processing is performed to restart the macro service.

(i) The macro service counter value is set again.

(ii) When the macro service end interrupt occurs, INTSR is in the vector interrupt mode. Therefore, the macro service mode is specified again.

(c) Vector interrupt: UART reception processing [label: TY_AIR]

This processing is equivalent to transfer processing for macro service.

(i) Received data is stored in (STRDTA + (SPINTA)).

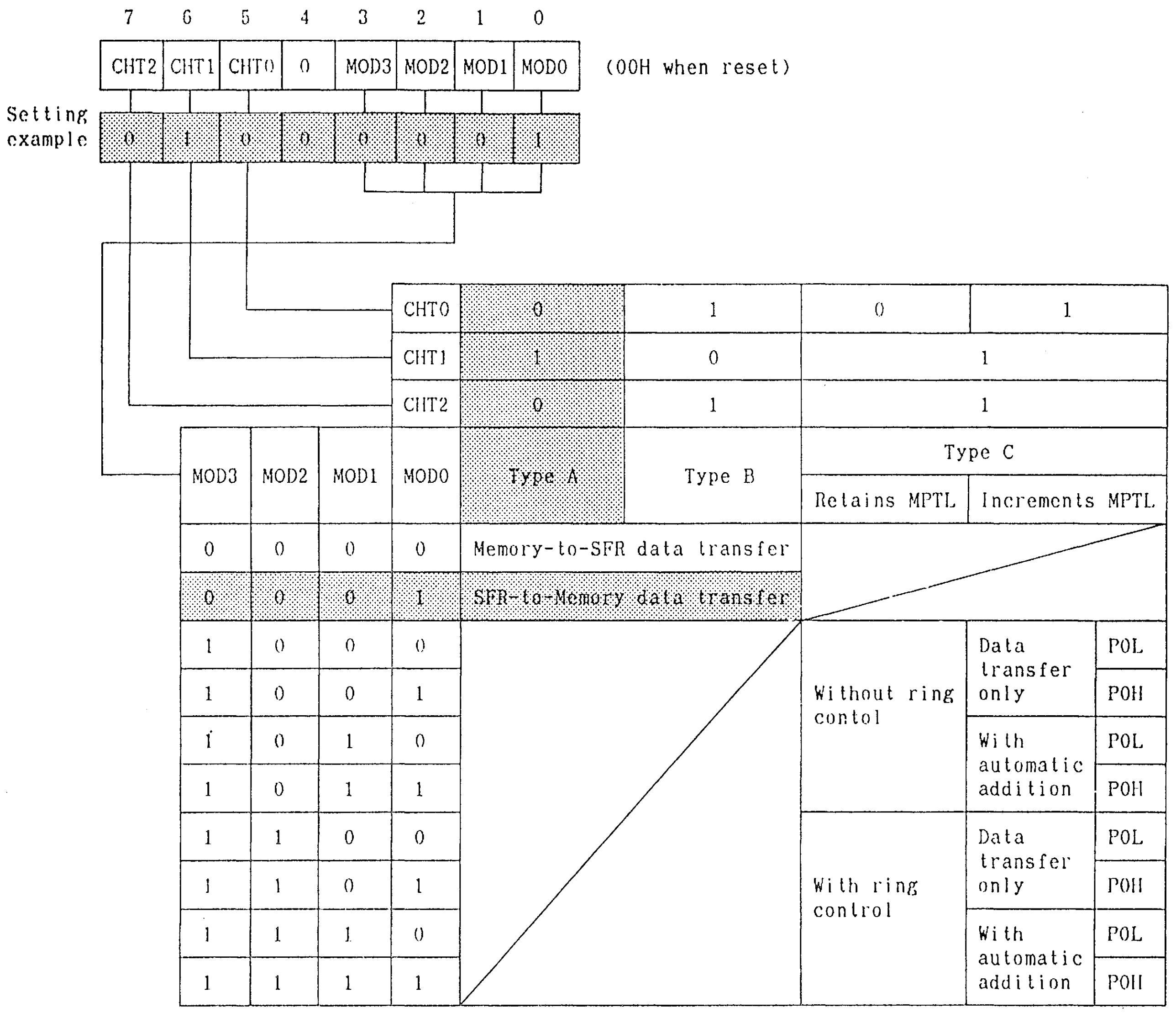
(ii) The (SPINTA) contents are incremented.

(iii) If the (SPINTA) contents coincide with those for (CNTNMA), (SPINTA) is cleared to 0.

(iv) Reception end flag ENDRCV is set to 1.

(3) Mode register setting

Macro service mode register



	7	6	5	4	3	2	1	0	
	CHT2	CHT1	CHT0	0	MOD3	MOD2	MOD1	MOD0	(00H when reset)
Setting example	0	1	0	0	0	0	0	1	

			CHT0	0	1	0	1	
			CHT1	1	0	1		
			CHT2	0	1	1		
MOD3	MOD2	MOD1	MOD0	Type A	Type B	Type C		
						Retains MPTL	Increments MPTL	
0	0	0	0	Memory-to-SFR data transfer				
0	0	0	1	SFR-to-Memory data transfer				
1	0	0	0			Without ring contol	Data transfer only	POL
1	0	0	1					POH
1	0	1	0				With automatic addition	POL
1	0	1	1					POH
1	1	0	0			With ring control	Data transfer only	POL
1	1	0	1					POH
1	1	1	0				With automatic addition	POL
1	1	1	1					POH

Interrupt service mode register H

	7	6	5	4	3	2	1	0	
ISM0H	CSIISM	STISM	SRISM	0	0	PISM5	PISM4	0	(00H when RESET is input)
Setting example	x	x	1	x	x	x	x	x	x: Not used

ISM	Interrupt service mode flag
0	Vector interrupt processing
1	Macro service processing

(4) Input/output parameters

- Macro service

(a) CNTNMA:

A value set to the macro service counter. After data transfer has been performed the number of times specified by this value, the macro service end interrupt occurs. This interrupt is started by INTSR. Therefore, the vector address (0022H) of INTSR in the vector table is referenced.

(b) CHASR:

An address to be set in the channel pointer. Since macro service type A is performed, the location address of the macro service counter (see below) is CHASR.

(c) MSCSR:

The location address of the macro service counter.

- Vector interrupt

(a) SPINTA:

The address storing the pointer pointing to the destination to which the receive data is to be written. The write destination address is represented in the form STRDTA + (SPINTA).

(b) STRDTA:

The transfer destination base address for the received data

(c) CNTNMA:

The number of times a cluster of data has been received. When data reception has been performed the specified number of times, receive end flag ENDRCV is set to 1.

(d) ENDRCV:

This flag is set to 1, when a cluster of data has been received the specified number of times.

- UART initialization

(a) BRGC:

Value set to the baud rate generator control register (BRGC)

The baud rate is calculated in the following manner:

$$\text{Baud rate} = \frac{\text{Baud rate generator count clock}}{k + 1} \times \frac{1}{n} \times \frac{1}{16}$$

k: Setting value of MDL3 to MDL0 bits of BRGC register

1/n: Frequency divider tap

(5) Registers

- Macro service

  No register is used.

- Vector interrupt

  A and B

- UART initialize

  Not initialized

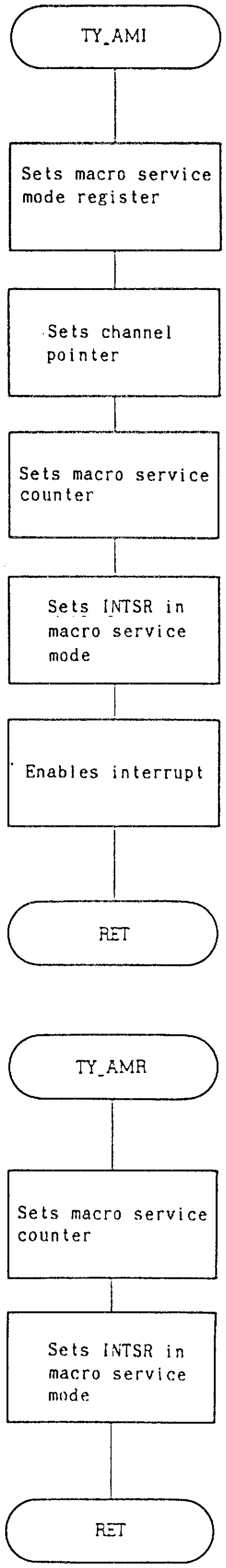
(6) Program example

Either when vector interrupt is used or when macro service is used, UART initialization processing [ASY214] is called. Therefore, module name [AS214] must be linked.
The following example is for when macro service is used.

```
;       --- FOR TYPE A ---
;
        PUBLIC  BRGCD                           ; PARAMETER
        PUBLIC  CNTNMA,MSCSR,CHASR              ; PARAMETER
        PUBLIC  TRN_DT,RCV_DT,RCVFLG            ; DUMMY PUBLIC FOR /ASY214/

        EXTRN   ASY214,TY_AMI,TY_AMR            ; PACKAGE
                .
                .
;       BAUD RATE :             9600bps
;       STOP BIT :              2bit
;       CHARACTER LENGTH :      8bit*
;       PARITY :                NO PARITY
;
BRGCD   EQU     10100100B       ; BRGC DATA (9600bps,fclk/5)

SRMK    EQU     MK0H.5          ; MASK OF INTSR
CTS     EQU     P3.5            ; CTS FOR RS-232-C
RTS     EQU     P3.4            ; RTS FOR RS-232-C

CNTNMA  EQU     0FH             ; STORE POINTER DATA
MCHSR   DSEG    AT 0FEB0H
BASR:   DS      CNTNMA
MSCSR:  DS      1               ; MACRO SERVICE COUNTER
CHASR   EQU     $-1

TRN_DT  EQU     0FEFFH          ; DUMMY FOR ASYNCHRONOUS MODULE EXTRN
RCV_DT  EQU     TRN_DT          ; DUMMY FOR ASYNCHRONOUS MODULE EXTRN
RCVFLG  EQU     TRN_DT.0        ; DUMMY FOR ASYNCHRONOUS MODULE EXTRN

INTSRVT CSEG    AT 00022H
        DW      INTSR           ; INTSR  vector
                .
                .
TY_A_M:
                .
                .
        CALL    !TY_AMI         ; <<< TY_AMI >>>
        CALL    !ASY214         ; SERIAL INTERFACE INITIALIZE
TYA_L1:                         ; DUMMY LOOP
        BR      TYA_L1
;
INTSR:                          ; FOR END OF RECEIVE
        SEL     RB1             ; CHANGE REG BANK
        SET1    RTS             ; RTS -> 1
                .
                .
        CALL    !TY_AMR         ; <<< TY_AMR >>>
        CLR1    RTS             ; RTS <- 0
        RETI
```

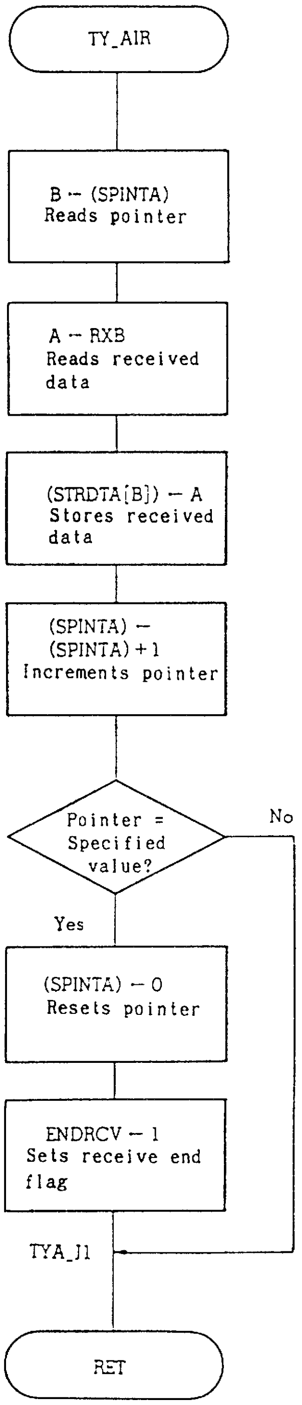
(7) Flowchart

(a) Macro Service



TY_AMI
Sets macro service mode register
Sets channel pointer
Sets macro service counter
Sets INTSR in macro service mode
Enables interrupt
RET
TY_AMR
Sets macro service counter
Sets INTSR in macro service mode
RET

(b) Vector interrupt



(8) Program list

(a) Macro service

```
        NAME    TYA_MR

;**********************************************************
;* macro service type-A application at macro service      *
;*      asynchronous serial receive                        *
;**********************************************************

        PUBLIC  TY_AMI,TY_AMR

        EXTRN   CNTNMA,MSCSR,CHASR

MCWSR   DSEG    AT OFEDEH
MSMSR:  DS      1               ; INTSR macro service mode register
CHPSR:  DS      1               ; INTSR macro service channel pointer

MKSR    EQU     MKOH.5          ; INTSR mask
SRISM   EQU     ISMOH.5         ; INTSR macro service mode
;
;       *** initialize ***
;
        CSEG
TY_AMI:
        MOV     MSMSR,#01000001B
                                ; set macro service mode register
        MOV     CHPSR,#LOW(CHASR)
                                ; set macro service channel pointer
        MOV     MSCSR,#LOW(CNTNMA)
                                ; set macro service counter
        SET1    SRISM           ; initialize interrupt
        EI
        RET
;
;       *** interrupt of complete macro-service ***
;
TY_AMR:
        MOV     MSCSR,#LOW(CNTNMA)
                                ; restore macro service counter
        SET1    SRISM           ; set interrupt service mode
        RET
;
        END
```

(b) Vector interrupt

```
        NAME    TYA_IR
;
;**********************************************************
;* macro service type-A application at interrupt           *
;*      asynchronous serial receive                        *
;**********************************************************
;
        PUBLIC  TY_AIR
        EXTRN   STRDTA,SPINTA,CNTNMA
        EXTBIT  ENDRCV
;
;       *** receive data ***
;
        CSEG
TY_AIR:
        MOV     A,SPINTA          ; read store pointer
        MOV     B,A               ; save A-register
        MOV     A,RXB             ; read receive data
        MOV     STRDTA[B],A       ; store receive data
        INC     SPINTA            ; increment store pointer
        CMP     SPINTA,#LOW(CNTNMA)
        BNZ     $TYA_J1
        MOV     SPINTA,#0
        SET1    ENDRCV            ; set end flag
TYA_J1:
        RET
;
        END
```

7.2 Parallel Data Input in Synchronization with External Interrupt Request

Data input to a port in synchronization with an external interrupt request is continuously transferred to memory.

(1) Operation

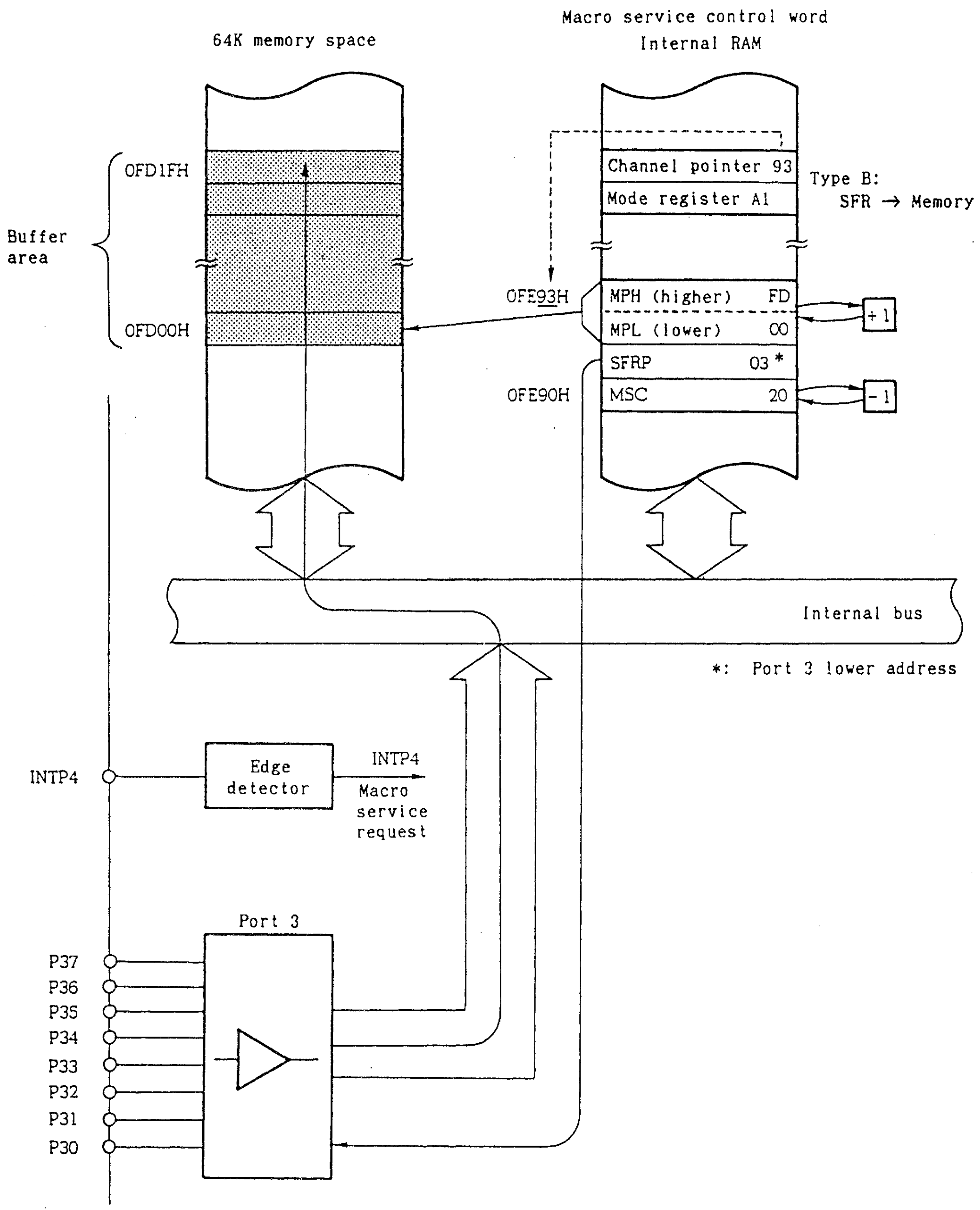
(a) Macro service

Parallel data are input from port 3 in synchronization with external interrupt request INTP4 by using macro service type B. The value of the channel pointer (CHP) is 93H, while a value of 32 (20H) is set in the macro service counter (MSC). The first address of the buffer area is FD00H.
Macro service end interrupt processing is used to reset the initial values of the MSC, CHP, and macro service pointers (MPH and MPL).

(b) Vector interrupt

By using the same work areas as those for macro service processing in (a) above, the same operation is simulated.

Fig. 7-2 Memory Mapping when Macro Service is Used (Parallel Data Input)



(2) Program ... Refer to (8) Program list

- Macro service

(a) Macro service initialization processing
[label: TY_BMI]

(i) Port 3 is set in the input port mode.

(ii) The rising edge is specified to be detected on the INTP4 pin.

(iii) The macro service mode register is set so that macro service type B is selected to transfer data from SFR to memory.

(iv) The channel pointer is set.

(v) The number of times data transfer is to be performed is set in the macro service counter.

(vi) The lower 8 bits for port 3 location address are set in the SFR pointer.

(vii) The first transfer destination address is set in the macro service pointer.

(viii) A macro service is specified as the processing for interrupt request INTP4.

(ix) Interrupt request INTP4 is unmasked.

(x) Interrupt is enabled.

(b) Macro service end interrupt processing [label: TY_BMR]

This processing is used to restart the macro service.

(i) The macro service counter value is set again.

(ii) The macro service pointer is returned to the originally specified value.

(iii) When macro service end interrupt occurs, the INTP4 signal is in the vector interrupt mode. Therefore, the macro service mode is set again.

- Vector interrupt

(a) Interrupt and word area initialization processing [label: TY_BII]

(i) Port 3 is set in the input port mode.

(ii) The rising edge is specified to be detected on the INTP4 pin.

(iii) Interrupt request INTP4 is unmasked.

(iv) The INTP4 pin is set in the vector interrupt processing mode.

(v) The first destination address is set in (SPINTB).

(vi) The number of times transfer is to be performed is set in (SCUNTB).

(vii) The interrupt is enabled.

(b) Parallel data input processing [label: TY_BIR]

This processing is equivalent to transfer processing for the macro service.

(i) Data input through port 3 is stored in (SPINTB).

(ii) The (SPINTB) contents are incremented.

(iii) The (SCUNTB) contents are decremented. If the contents become 0 as a result, the processing ends.

(iv) Sampling end flag ENDSMP is set to 1.

(3) Mode register setting

Port 3 mode control register

	7	6	5	4	3	2	1	0
PMC3	PMC37	PMC36	PMC35	PMC34	PMC33	PMC32	PMC31	PMC30
Setting example	0	0	0	0	0	0	0	0

(00H when RESET is input)

PMC30	Specifies control mode for pin P30
0	I/O port mode
1	RxD input mode

PMC31	Specifies control mode for pin P31
0	I/O port mode
1	TxD output mode

PMC32	Specifies control mode for pin P32
0	I/O port mode
1	$\overline{\text{SCK}}$ input/output mode

PMC33	Specifies control mode for pin P33
0	I/O port mode
1	SB0 input/output mode/SO output mode

PMC3n	Specifies control mode for pin P3n (n = 4 to 7)
0	I/O port mode
1	TOm output mode (m = 0 to 3)

Port 3 mode register

	7	6	5	4	3	2	1	0	
PM3	PM37	PM36	PM35	PM34	PM33	PM32	PM31	PM30	(FFH when RESET is input)
Setting example	1	1	1	1	1	1	1	1	

PM3n	Specifies I/O mode for pin P3n (n = 0 to 7)
0	Output mode (output buffer ON)
1	Input mode (input buffer OFF)

External interrupt mode register 1

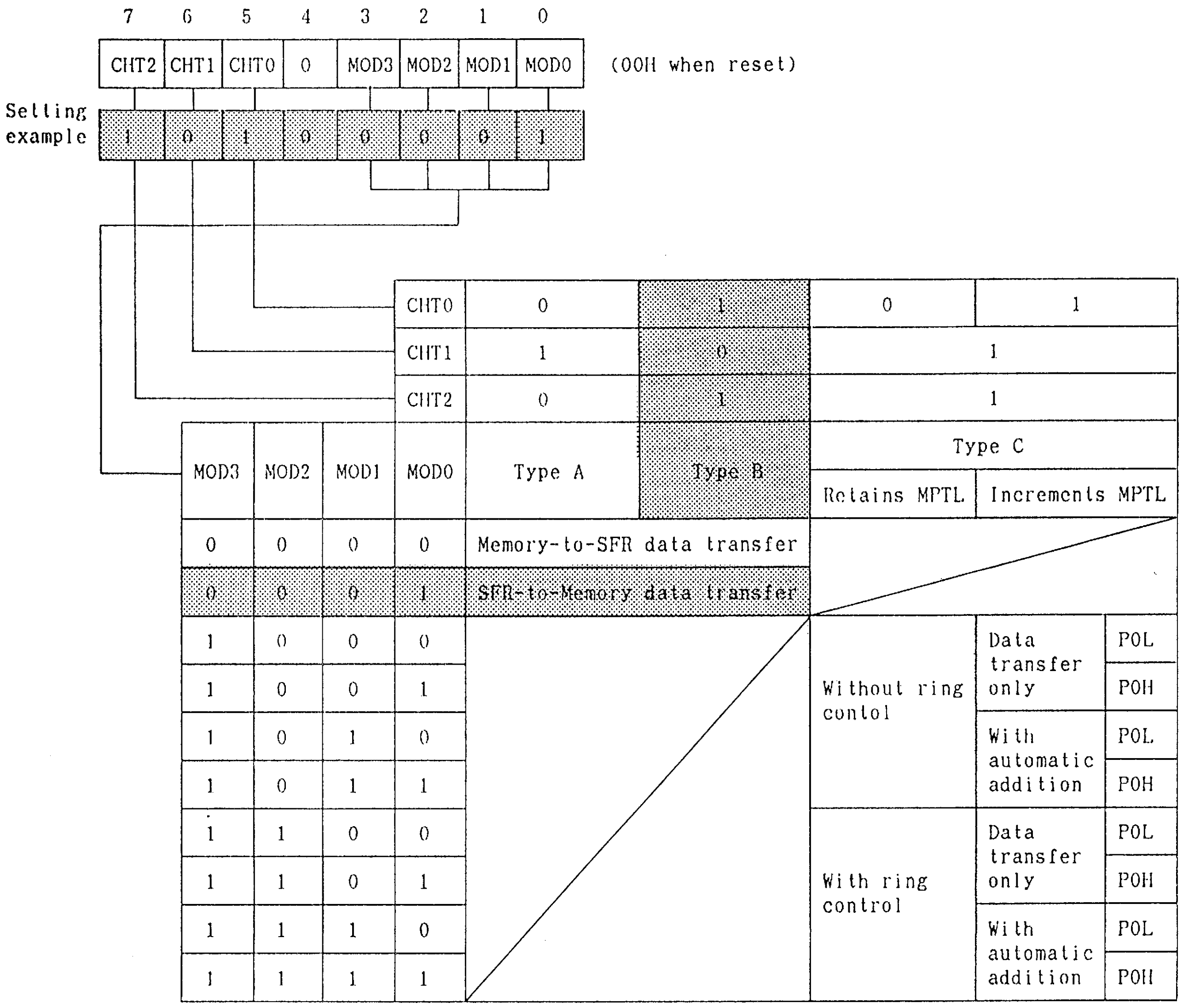
	7	6	5	4	3	2	1	0	
INTM1	0	0	ES51	ES50	ES41	ES40	ES31	ES30	(00H when RESET is input)
Setting example	x	x	x	x	x	1	x	x	x: Not used

ES31	ES30	Specifies input edge to be detected on INTP3 pin
0	0	Falling edge
0	1	Rising edge
1	0	Inhibited
1	1	Both falling and rising edges

ES41	ES40	Specifies input edge to be detected on INTP4 pin
0	0	Falling edge
0	1	Rising edge
1	0	Selects INTC30
1	1	Inhibited

ES51	ES50	Specifies input edge to be detected on INTP5 pin
0	0	Falling edge
0	1	Rising edge
1	0	Inhibited
1	1	

Macro service mode register



	7	6	5	4	3	2	1	0	
	CHT2	CHT1	CHT0	0	MOD3	MOD2	MOD1	MOD0	(00H when reset)
Setting example	1	0	1	0	0	0	0	1	

			CHT0	0	1	0	1	
			CHT1	1	0	1		
			CHT2	0	1	1		
MOD3	MOD2	MOD1	MOD0	Type A	Type B	Type C		
						Retains MPTL	Increments MPTL	
0	0	0	0	Memory-to-SFR data transfer				
0	0	0	1	SFR-to-Memory data transfer				
1	0	0	0			Without ring contol	Data transfer only	POL
1	0	0	1					POH
1	0	1	0				With automatic addition	POL
1	0	1	1					POH
1	1	0	0			With ring control	Data transfer only	POL
1	1	0	1					POH
1	1	1	0				With automatic addition	POL
1	1	1	1					POH

Interrupt service mode register H

	7	6	5	4	3	2	1	0	
ISM0H	CSIISM	STISM	SRISM	0	0	PISM5	PISM4	0	(00H when RESET is input)
Setting example	x	x	1	x	x	x	1	x	x: Not used

ISM	Interrupt service mode flag
0	Vector interrupt processing
1	Macro service processing

Interrupt mask register H

	7	6	5	4	3	2	1	0	
MK0H	CSIMK	STMK	SRMK	SERMK	CMK20	PMK5	PMK4	CMK21	(00H when RESET is input)
Setting example	x	x	x	x	x	x	0	x	x: Not used

MK	Interrupt mask flag
0	Enables interrupt processing
1	Keeps interrupt processing pending

(4) Input/output parameters

- Macro service

(a) STRDTB:

A value set to the macro service pointer. The first address of the transfer destination is set.

(b) CNTNMB:

A value set to the macro service counter. After data transfer has been performed the number of times specified by this value, the macro service end interrupt occurs. This interrupt is started by INTP4. Therefore, the vector address (000EH) of INTP4 in the vector table is referenced.

- Vector interrupt

(a) STRDTB:

The first address of the destination to which the parallel input data is to be written.

(b) CNTNMB:

The number of times a cluster of data has been received. When data reception has been performed the specified number of times, sampling end flag ENDSMP is set to 1.

(c) SPINTB:

The memory address storing the transfer destination address [STRDTB].

(d) SCUNTB:

The memory address storing the number of times [CNTNMB] the parallel data is to be input.

(e) ENDSMP:

This flag is set to 1, when a cluster of data has been input the specified number of times.

(5) Registers

- Macro service

  None

- Vector interrupt

  X, A, HL

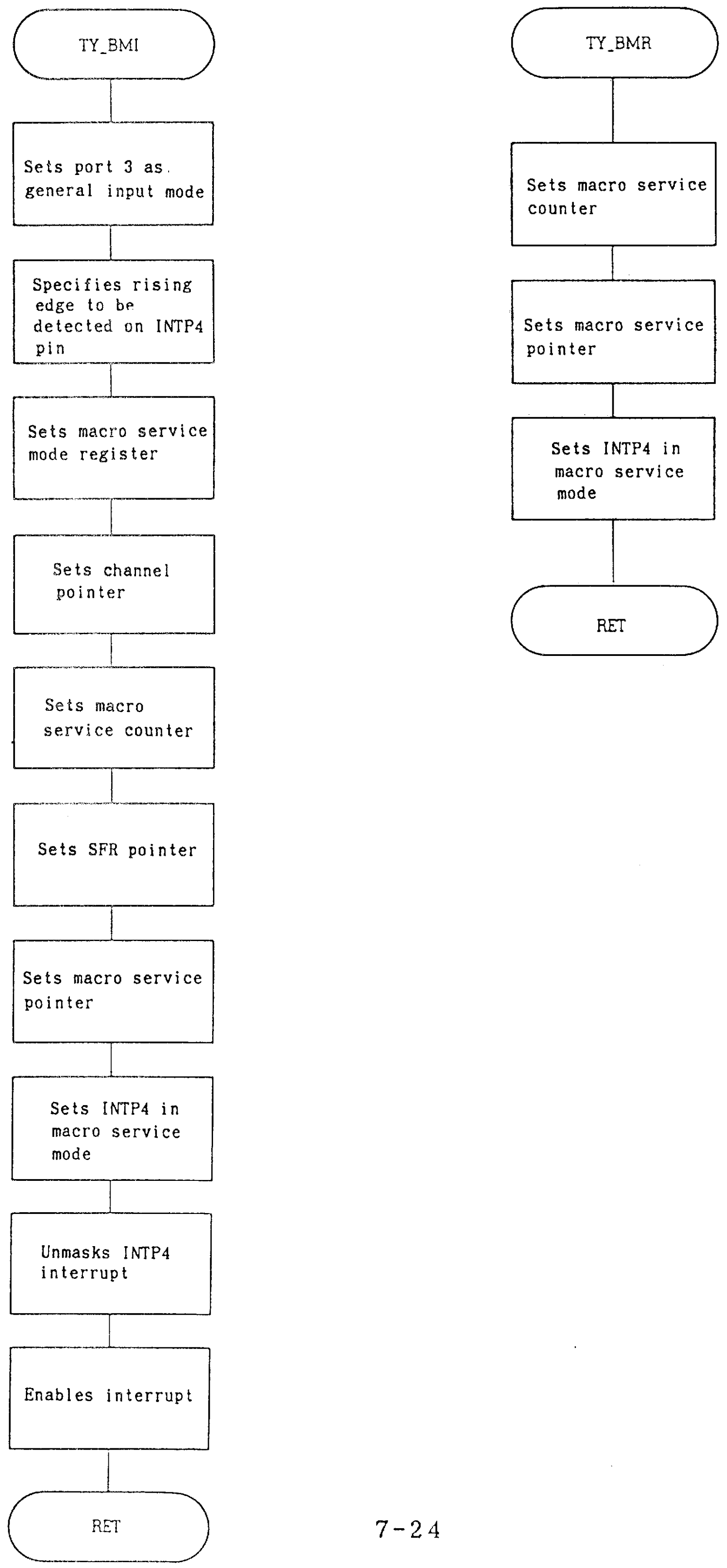
(6) Program example

The following program example is for the macro service.

```
;       --- FOR TYPE B ---
        PUBLIC  STRDTB,CNTNMB                     ; PARAMETER
        EXTRN   TY_BMI,TY_BMR                     ; PACKAGE
                .
                .
STRDTB  EQU     0A000H          ; sampling data store area
CNTNMB  EQU     20H             ; number of data
STRAREA DSEG    AT STRDTB
        DS      CNTNMB

INTP4VT CSEG    AT 0000EH
        DW      INTP4           ; INTP4 vector
                .
                .
TY_B_M:
                .
                .
        CALL    !TY_BMI         ; <<< TY_B_II >>>
TYB_L1:                         ; DUMMY LOOP
        BR      TYB_L1
;
INTP4:                          ; FOR END OF SAMPLING
        SEL     RB2             ; CHANGE REG BANK
                .
                .
        CALL    !TY_BMR         ; <<< TY_BMR >>>
        RETI
```

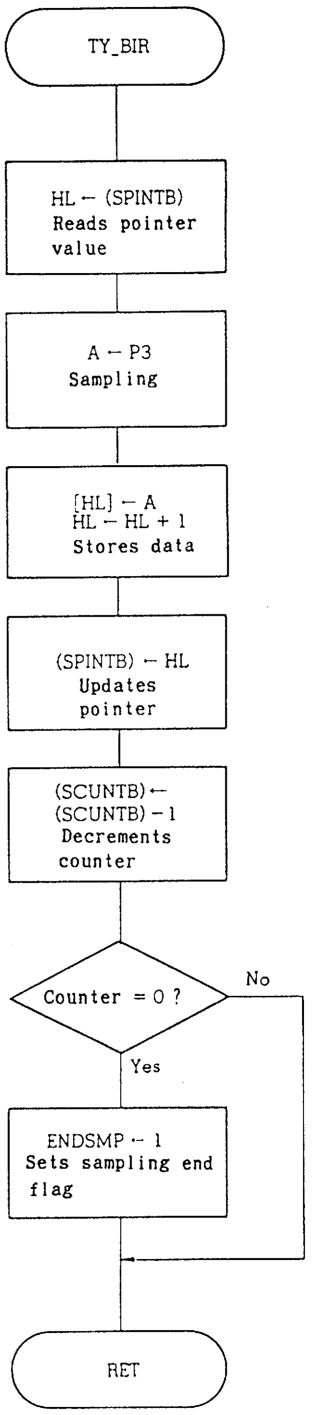
(7) Flowchart

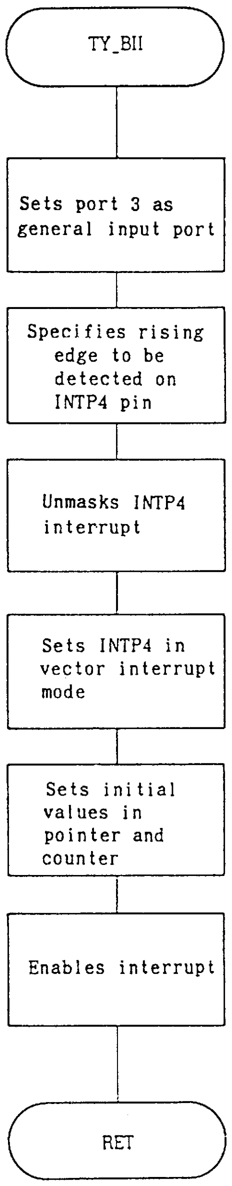
(a) Macro service



TY_BMI
Sets port 3 as general input mode
Specifies rising edge to be detected on INTP4 pin
Sets macro service mode register
Sets channel pointer
Sets macro service counter
Sets SFR pointer
Sets macro service pointer
Sets INTP4 in macro service mode
Unmasks INTP4 interrupt
Enables interrupt
RET
TY_BMR
Sets macro service counter
Sets macro service pointer
Sets INTP4 in macro service mode
RET

(b) Vector interrupt





TY_BII
Sets port 3 as general input port
Specifies rising edge to be detected on INTP4 pin
Unmasks INTP4 interrupt
Sets INTP4 in vector interrupt mode
Sets initial values in pointer and counter
Enables interrupt
RET

(8) Program list

(a) Macro service

```
        NAME    TYB_MR
;
;*********************************************************
;* macro service type-B application at macro service     *
;*      synchronous parallel data input                  *
;*********************************************************
;
        PUBLIC  TY_BMI,TY_BMR
        EXTRN   STRDTB,CNTNMB
;
MCHP4   DSEG    AT OFE90H
MSCP4:  DS      1               ; macro service counter
SFRP4:  DS      1               ; SFR pointer
MPP4P:  DS      2               ; macro service pointer
CHAP4   EQU     $-1

MCWP4   DSEG    AT OFED4H
MSMP4:  DS      1               ; INTP4 macro service mode register
CHPP4:  DS      1               ; INTP4 macro service channel pointer
;
PISM4   EQU     ISMOH.1         ; INTP4 interrupt service mode
PMK4    EQU     MKOH.1          ; INTP4 mask
;
;       *** initialize ***
;
        CSEG
TY_BMI:
        MOV     PMC3,#0         ; initialize P3
        MOV     PM3,#OFFH
        MOV     INTM1,#00000100B  ; INTP4 active edge is rise
        MOV     MSMP4,#10100001B
                                ; set macro service mode register
        MOV     CHPP4,#LOW(CHAP4)
                                ; set macro service channel pointer
        MOV     MSCP4,#LOW(CNTNMB)
                                ; set macro service counter
        MOV     SFRP4,#LOW(P3)  ; set SFR pointer
        MOVW    MPP4P,#STRDTB   ; set macro service memory pointer

        SET1    PISM4           ; initialize interrupt
        CLR1    PMK4
        EI
        RET
;
;       *** interrupt of complete macro-service ***
;
TY_BMR:
        MOV     MSCP4,#LOW(CNTNMB)
                                ; restore macro service counter
        MOVW    MPP4P,#STRDTB   ; restore macro service memory pointer
        SET1    PISM4           ; set interrupt service mode
        RET
;
        END
```

(b) Vector interrupt

```
        NAME    TYB_IR
;
;*******************************************************
;* macro service type-B application at interrupt        *
;*      synchronous parallel data input                 *
;*******************************************************
;
        PUBLIC  TY_BII,TY_BIR
        EXTRN   STRDTB,CNTNMB,SPINTB,SCUNTB
        EXTBIT  ENDSMP

ES40    EQU     INTM1.2         ; INTP4 active edge
;
        CSEG
TY_BII:
        MOV     PMC3,#0         ; initialize port-3
        MOV     PM3,#0FFH

        SET1    ES40            ; INTP4 active edge is rise
        MOV     MK0H,#11111101B ; open INTP4 mask
        MOV     ISM0H,#00

        MOVW    SPINTB,#STRDTB  ; set store pointer
        MOV     SCUNTB,#LOW(CNTNMB)
                                ; set store counter
        EI
        RET
;
;       *** sampling data ***
;
TY_BIR:
        MOVW    AX,SPINTB       ; read store pointer
        MOVW    HL,AX
        MOV     A,P3            ; sampling
        MOV     [HL+],A         ; store data
        MOVW    AX,HL
        MOVW    SPINTB,AX       ; write new store pointer

        DEC     SCUNTB          ; increment store counter
        BNZ     $TYB_J1
        SET1    ENDSMP          ; set end of sampling
TYB_J1:
        RET
;
        END
```

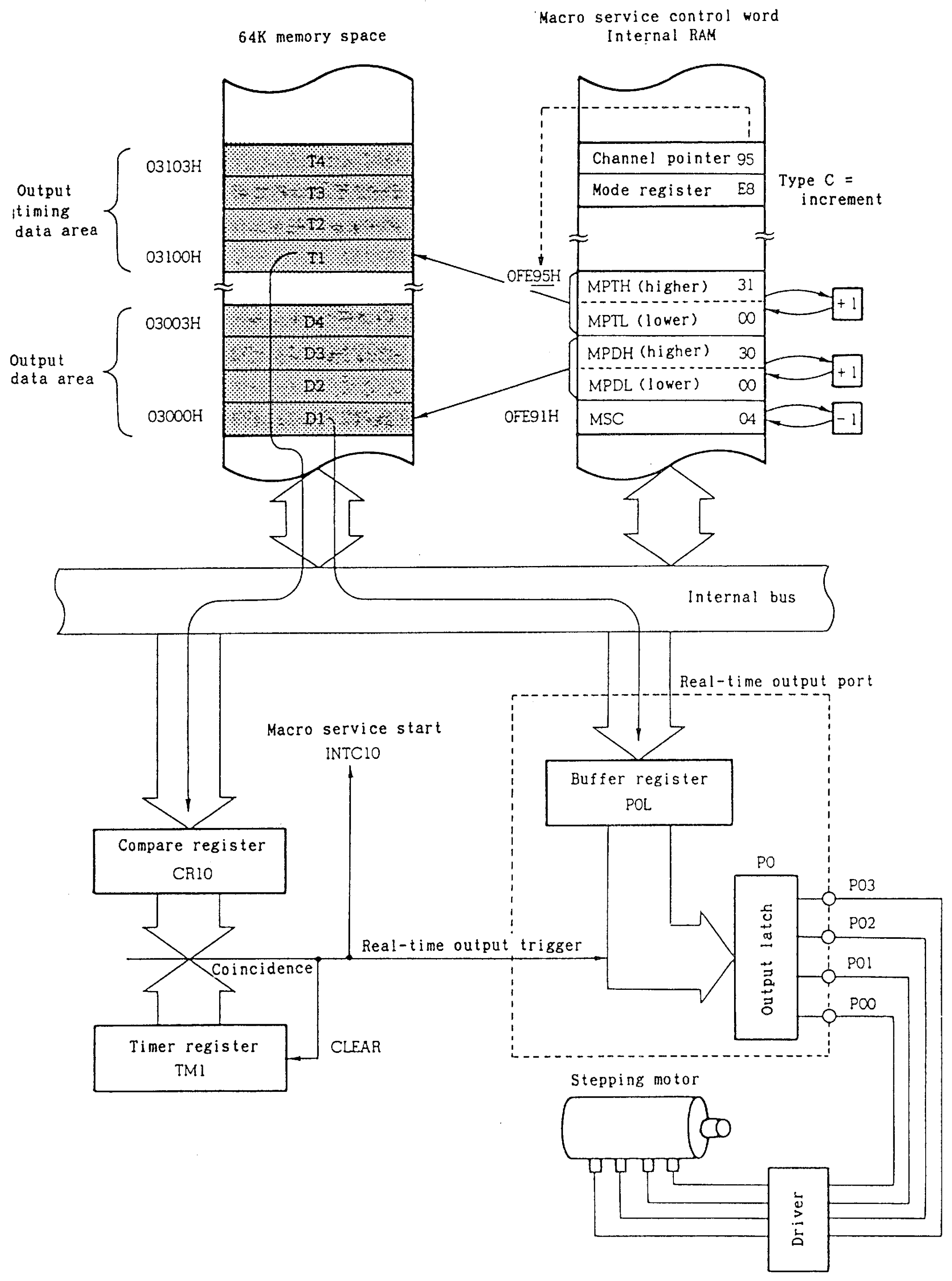
7.3 Open Loop Control for Stepping Motor (1)

An example program that accomplishes open loop control for a 4-phase stepping motor connector to real-time output port P0L by using macro service C.
In this example, internal system clock f_{CLK} is set to 6 MHz.

(1) Operation

The stepping motor is revolved by using the real-time output port. The motor is connected to the lower 4 bits (P0L) of the real-time output port and is revolved at a constant speed of 200 pps. A single phase or two phases are excited. Therefore, there are 4 patterns.
The initial value for the channel pointer (CHP) is 95H and that for the macro service counter (MSC) is 4. The values for timer and data macro service pointers (MPT and MPD) are 3100H and 3000H, respectively.
The macro service end interrupt processing is used to set the initial values for the counter and pointers again to continuously revolve the motor.
The minimum step angle for the stepping motor, used in this example, is 1.8 degree.

Fig. 7-3 Memory Mapping for Open Loop Control of Stepping Motor



(2) Program ... Refer to (8) Program list

- Macro service initialization processing [label: TY_CMI]

(a) Initial excitation data is set in port 0.
(b) Port 0 is set in the output port mode.
(c) The lower 4 bits for port 0 are set in the real-time output port mode.
(d) 8-bit timer/counter 1 (TM1) is reset.
(e) A mode is set in which TM1 is cleared, when the TM1 contents coincide with those for CR10.
(f) The TM1 count clock is set.
(g) The macro service mode register is set, so that macro service C is specified.
(h) The channel pointer is set.
(i) The data macro service pointer is set.
(j) The timer macro service pointer is set.
(k) The number of excitation patterns is set in the macro service counter.
(l) 8-bit timer/counter 1 is started.
(m) INTC10 is set in the macro service mode.
(n) INTC10 us unmasked.
(o) The interrupt is enabled.

- Macro service end interrupt processing [label: INTC10]

(a) The data macro service pointer is set again.
(b) The timer macro service pointer is set again.
(c) The number of excitation patterns is set in the macro service counter again.
(d) INTC10 is set again in the macro service mode.

(3) Mode register setting

Real-time output port control register

	7	6	5	4	3	2	1	0	
RTPC	BYTE	Note 0	Note 0	POMH	EXTR	Note 0	Note 0	POML	(00H when RESET is input)
Setting example	0	0	0	0	0	0	0	1	

POML	Specifies P0L function
0	Port mode
1	Real-time output port mode

EXTR	Enables data transfer from buffer register to output latch by INTP0
0	Disables

1	Enables
	BYTE = 0: Only P0L is transferred
	Byte = 1: P0L/H is transferred

POMH	Specifies P0H function
0	Port mode
1	Real-time output port mode

BYTE	Specifies operation mode for real-time output port
0	4-bit separate real-time output port
1	8-bit real-time output port

Note: Be sure to write 0 to bits 1, 2, 5, and 6.

Port 0 mode register

	7	6	5	4	3	2	1	0	
PM0	PM07	PM06	PM05	PM04	PM03	PM02	PM01	PM00	(00H when RESET is input)
Setting example	0	0	0	0	0	0	0	0	

PM0n	Specifies P0n pin input/output mode (n = 0 to 7)
00H	Output mode (output buffer ON)
FFH	High-impedance state (output buffer OFF)
Other than above	Inhibited

Timer control register 1

	7	6	5	4	3	2	1	0	
TMC1	CE2	OVF2	CMD2	0	CE1	OVF1	0	0	(00H when RESET is input)
Setting example	x	x	x	0	1	0	0	0	x: Not used

8-bit timer/counter 1

OVF1	TM1 overflow flag
0	No overflow
1	Overflow occurs (count up from FFH to 00H)

Remarks: This bit is reset by software only.

CE1	Controls count operation for TM1
0	Clears and stops count operation
1	Enables count operation

8-bit timer/counter 2

CMD2	Specifies count operation mode for TM2
0	Ordinary mode
1	One-shot mode

OVF2	TM2 overflow flag
0	No overflow
1	Overflow occurs (count up from FFH to 00H)

Remarks: This bit is reset by software only.

CE2	Controls count operation for TM2
0	Clears and stops count operation
1	Enables count operation

Capture/compare control register 1

	7	6	5	4	3	2	1	0	
CRC1	0	0	0	0	CLR11	CM	CLR10	0	(00H when RESET is input)
Setting example	0	0	0	0	0	0	1	0	

CLR10	Clearing TM1 when coincidence occurs between TM1 and CR10
0	Disabled
1	Enabled

CLR11	CM	Specifies CR11 register operation	TM1 clearing operation
0	0	Compare register	Inhibited
1	0	Compare register	Enabled (when TM1 contents coincide with those for CR11 register)
0	1	Capture register	Inhibited
1	1	Capture register	Enabled (when TM1 contents are captured into CR11 register)

Prescaler mode register 1

	7	6	5	4	3	2	1	0	
PRM1	PRS23	PRS22	PRS21	PRS20	0	PRS12	PRS11	PRS10	(00H when RESET is input)
Setting example	0	0	0	0	0	1	1	1	

8-bit timer/counter 1

PRS12	PRS11	PRS10	Timer/counter 1 count clock specification
0	0	0	$f_{CLK}/16$Note
0	0	1	
0	1	0	
0	1	1	$f_{CLK}/32$
1	0	0	$f_{CLK}/64$
1	0	1	$f_{CLK}/128$
1	1	0	$f_{CLK}/256$
1	1	1	$f_{CLK}/512$

(Cont'd)

8-bit timer/counter 2

PRS23	PRS22	PRS21	PRS20	Timer/counter 3 count clock specification
0	0	0	0	$f_{CLK}/16$
0	0	0	1	
0	0	1	0	
0	0	1	1	$f_{CLK}/32$
0	1	0	0	$f_{CLK}/64$
0	1	0	1	$f_{CLK}/128$
0	1	1	0	$f_{CLK}/256$
0	1	1	1	$f_{CLK}/512$
1	1	1	1	External clock (CI)

Note: f_{CLK}: Internal system clock frequency ($f_{XX}/2$)

Macro service mode register

	7	6	5	4	3	2	1	0	
	CHT2	CHT1	CHT0	0	MOD3	MOD2	MOD1	MOD0	(00H when reset)
Setting example	1	1	1	0	1	0	0	0	

			CHT0	0	1	0	1	
			CHT1	1	0	1		
			CHT2	0	1	1		
MOD3	MOD2	MOD1	MOD0	Type A	Type B	Type C		
						Retains MPTL	Increments MPTL	
0	0	0	0	Memory-to-SFR data transfer				
0	0	0	1	SFR-to-Memory data transfer				
1	0	0	0			Without ring contol	Data transfer only	POL
1	0	0	1					POH
1	0	1	0				With automatic addition	POL
1	0	1	1					POH
1	1	0	0			With ring control	Data transfer only	POL
1	1	0	1					POH
1	1	1	0				With automatic addition	POL
1	1	1	1					POH

Interrupt service mode register L

	7	6	5	4	3	2	1	0	
ISM0L	CISM11	CISM10	0	0	0	0	0	0	(00H when RESET is input)
Setting example	x	1	x	x	x	x	x	x	x: Not used

ISM	Interrupt service mode flag
0	Vector interrupt processing
1	Macro service processing

Interrupt mask register L

	7	6	5	4	3	2	1	0	
MK0L	CMK11	CMK10	CMK01	CMK00	PMK3	PMK2	PMK1	PMK0	(FFH when RESET is input)
Setting example	x	0	x	x	x	x	x	x	x: Not used

MK	Interrupt mask flag
0	Enables interrupt processing
1	Keeps interrupt processing pending

(4) Input/output parameters

In this example, there is no input/output parameters because the parameters necessary for the macro service are described in (8) Program list. The step rate is calculated in the following mannar.
The "step rate" is the number of pulses applied to the stepping motor in a unit time. For example, if the step rate is 200 pps, the stepping motor revolves 200 steps in 1 second. Since the minimum step angle for the stepping motor used in this example is 1.8 degree, the 200 pps step rate is equal to 1 rps (60 rpm).
Therefore, a value must be calculated as follows and set in the compare register CR10 for 8-bit timer/counter 1 so that interrupt request INTC10 is generated every 5 ms, because the one pulse is input to the stepping motor at 5 ms intervals.

$$\boxed{\begin{aligned} \mathrm{CR10} &= (5\ (\mathrm{ms}) \times \text{Count clock of TM1}) - 1 \\ &= (5\ (\mathrm{ms}) \times 6\ (\mathrm{MHz})/512) - 1 \\ &= (5 \times -10^{-3} \times 6 \times 10^{6}/512) - 1 \\ &\fallingdotseq 58.6 - 1 \\ &= 57.6 \rightarrow 58 \end{aligned}}$$

(5) Registers

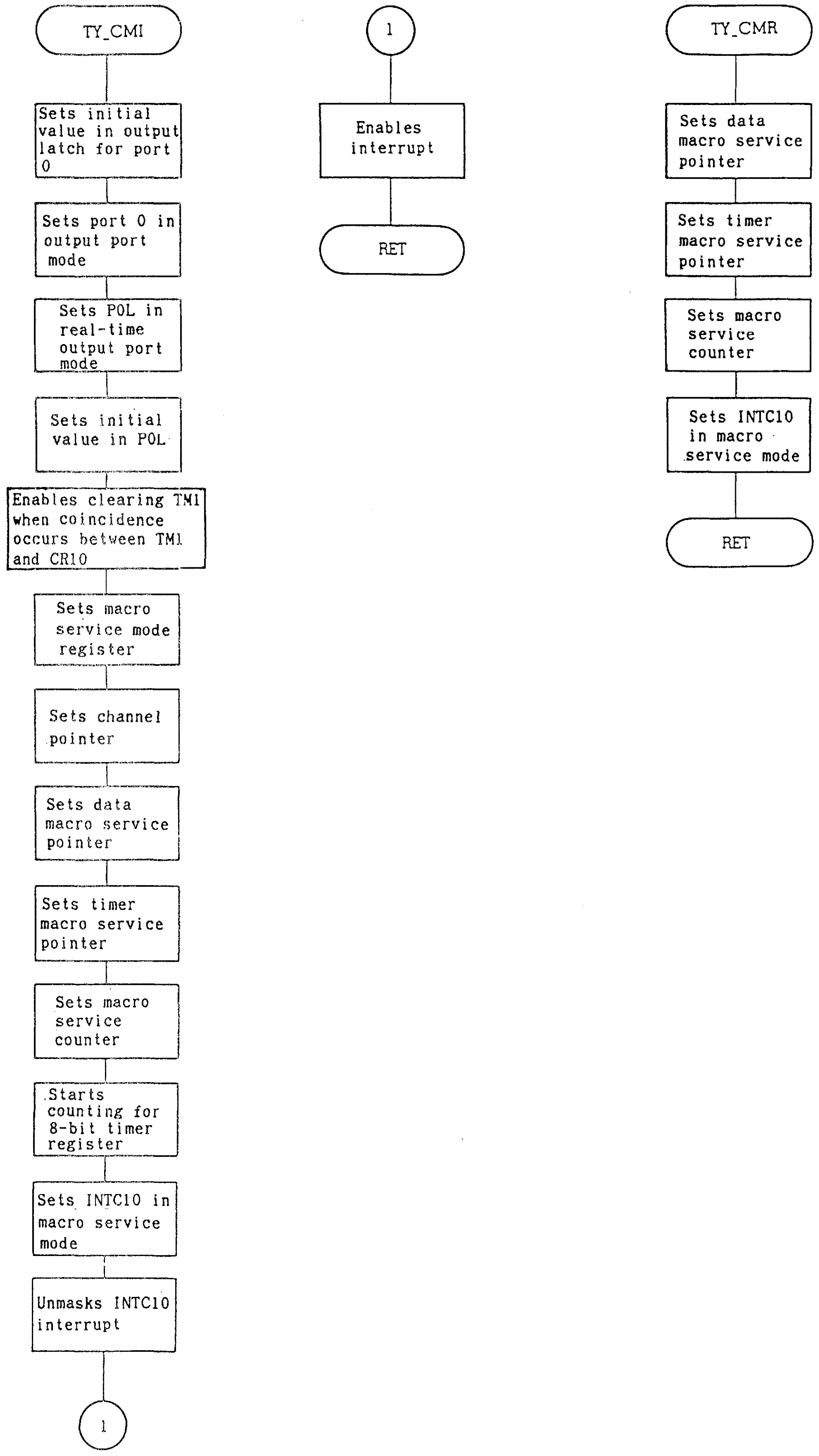
No register is used.

(6) Program example

The program examples are shown below.

```
;       --- FOR TYPE C ---
;
        EXTRN   TY_CMI          ; PACKAGE

TY_C_M:
        CALL    !TY_CMI         ; <<< TY_CMI >>>
TYC_L1:
        BR      TYC_L1          ; DUMMY LOOP
```

(7) Flowchart


TY_CMI
Sets initial value in output latch for port 0
Sets port 0 in output port mode
Sets POL in real-time output port mode
Sets initial value in POL
Enables clearing TM1 when coincidence occurs between TM1 and CR10
Sets macro service mode register
Sets channel pointer
Sets data macro service pointer
Sets timer macro service pointer
Sets macro service counter
Starts counting for 8-bit timer register
Sets INTC10 in macro service mode
Unmasks INTC10 interrupt
1
1
Enables interrupt
RET
TY_CMR
Sets data macro service pointer
Sets timer macro service pointer
Sets macro service counter
Sets INTC10 in macro service mode
RET

(8) Program list

```
        NAME    TYC_MR
;
;**********************************************************
;* macro service type-C application at macro service      *
;*      for stepping motor control                        *
;**********************************************************
;
        PUBLIC  TY_CMI
;
MCHC10  DSEG    AT 0FE91H
MSCC10: DS      1               ; macro service counter
MPDC10P:DS      2               ; macro service pointer for data
MPTC10P:DS      2               ; macro service pointer for timing
CHAC10  EQU     $-1             ; macro service channel address

MCWC10  DSEG    AT 0FED8H
MSMC10: DS      1               ; INTC10 macro service mode register
CHPC10: DS      1               ; INTC10 channel pointer

CISM10  EQU     ISM0L.6
CMK10   EQU     MK0L.6
;
INTC10VT CSEG   AT 00018H
        DW      INTC10          ; INTC10 vector
;
;       *** initialize ***
;
        CSEG
TY_CMI:
        MOV     P0,#LOW(D4)     ; P0 output latch <- 1st data
        MOV     PM0,#0          ; P0 <- output port
        MOV     RTPC,#00000001B ; initialize real time output port
        MOV     P0L,#LOW(D4)    ; 1st data

        MOV     CRC1,#00000010B ; enable clear TM1
        MOV     PRM1,#00000111B ; set TM1 prescaler fclk/512

        MOV     MSMC10,#11101000B
                                ; set macro service mode register
        MOV     CHPC10,#LOW(CHAC10)
                                ; set macro service channel pointer
        MOVW    MPDC10P,#PFDT   ; set memory pointer for data
        MOVW    MPTC10P,#PFTM   ; set memory pointer for timing
        MOV     MSCC10,#LOW(CYCNUM)
                                ; set macro service counter
        OR      TMC1,#00001000B ; timer start
        SET1    CISM10          ; initialize interrupt
        CLR1    CMK10
        EI
        RET
;
;       *** complete of macro service ***
;
INTC10:
        MOVW    MPDC10P,#PFDT   ; restore memory pointer for data
        MOVW    MPTC10P,#PFTM   ; restore memory pointer for timing
        MOV     MSCC10,#LOW(CYCNUM)
                                ; restore macro service counter
        SET1    CISM10          ; set interrupt service mode
        RETI
```

```
;
;       *** data profile ***
;
D1      EQU     00000011B       ; define output data
D2      EQU     00000110B
D3      EQU     00001100B
D4      EQU     00001001B
TN      EQU     58              ; define timing data
CYCNUM  EQU     4               ; number of step cycle

DTFLIE  CSEG    AT 03000H
PFDT:   DB      D1,D2,D3,D4     ; output data
TMFILE  CSEG    AT 03100H
PFTM:   DB      TN,TN,TN,TN     ; timing data
;
        END
```

7.4 Open Loop Control for Stepping Motor (2)

Compared to the uPD78214 series and uPD78224 series, the macro service function is enhanced in the uPD78218A series, uPD78234 series, and the uPD78244 series. Therefore, 16-bit data can be set in the macro service counter.
The next program example is for macro service type C for these series.

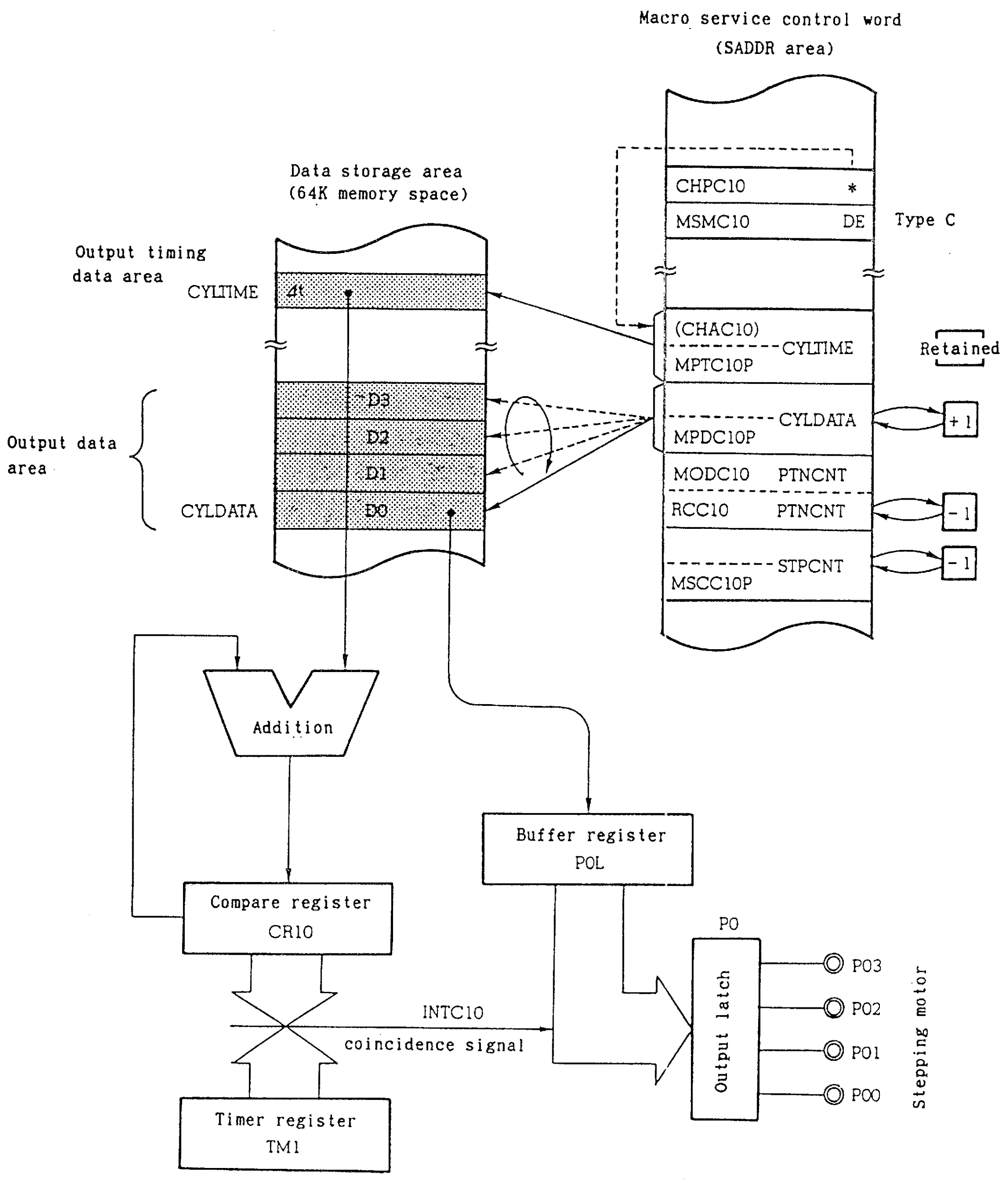
(1) Operation outline

The same operation as that indicated in 7.3, "Stepping motor open loop control (1)" can be obtained in the folloiwng mode:

- Macro service counter (MSC) = 16 bits
- Pointer for timing data is retained
- Ring control
- Automatic addition

f_{CLK} = 6MHz is used for this application example. $f_{CLK}/512$ is selected for the count clock for the 8-bit timer/counter 1. Fig. 7-4 shows the memory map for this macro service.

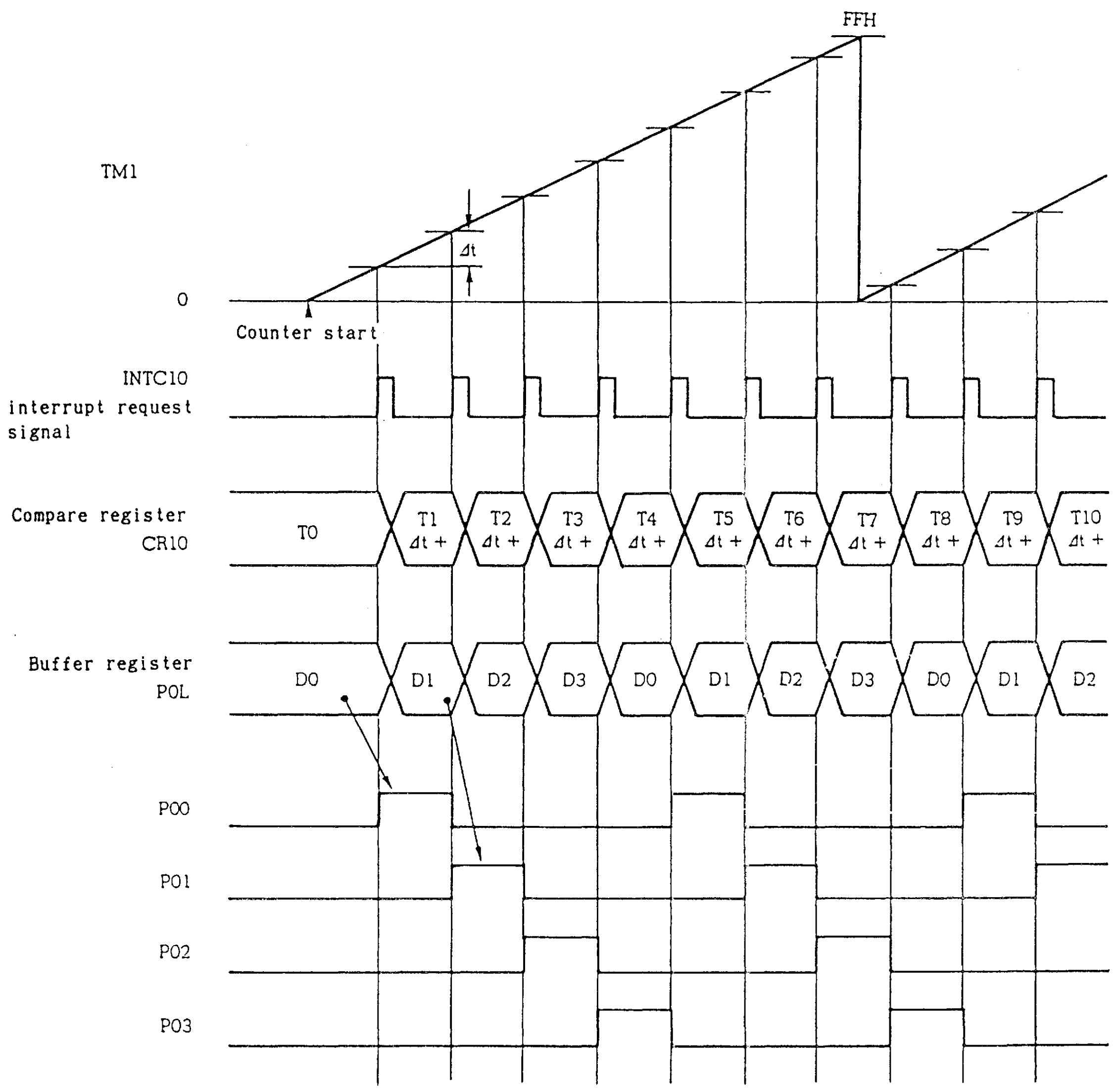
Fig. 7-4 Memory Map for Macro Service (when MSC = 16 bits)



*: Lower byte of CHAC10 (=MPTC10P+1)

When rotating the steppin motor at a constant speed, only resetting the macro service is accomplished in the macro service completion interrupt processing. Therefore, moving as many steps as possible by one macro service start up (CISM10 $\leftarrow$ 1) reduces the number of completion interrupts. This increases the CPU service time.
Setting 0 to the macro service counter (MSC) gives the maximum number of transfer operations, which is 65536. In this case, if the number of energization patterns (4 for single or 2 phase energization, 8 for 1-2 phase energization) is set to the ring counter (RC) and modulo register (MOD), the energization data pointer will be automatically returned to the original position, even if it exceeds the end.
In addition, with the automatic addition mode, the oepration as shown by the timing chart in Fig. 7-5 can be realized.

Fig. 7-5 Automatic Addition Mode Timing Chart (Single Phase Energization)



When the timing data pointer is set to "retain", value Δt is added to the previous compare register (CR10) value and output as the timing data. In this case, the timer (TM1) is used in the free running mode. Therefore, another stepping motor can be easily operated using the INTC11 macro service.

(2) Program example

(a) Processing outline (Refer to (8) Flow chart)

(i) Macro service initialization processing
[label name: TYC16_M]
Initializes port 0, real-time output port, 8-bit timer/counter 1, INTC10 macro service.

(ii) INTC10 macro service completion interrupt processing [label name: INTC10]
Sets INTC10 service mode again to macro service.
In this application exmple, 0 is set to the macro service counter as the initial value. Therefore, the macro service counter does not need to be set again, when the completion interrupt is generated (becase MSC = 0)
In addition, ring control is used, and this automatically sets the macro service pointer. Therefore, the macro service counter does not need to be set again either.

(b) RAM used

In this application example, 8 bytes of RAM is used for macro service channel. Table 7-2 lists the RAM areas used for the macro service channel.

Table 7-2 RAM Areas Used for Macro Service (MSC = 16 bits)

RAM name	Allocation area	Purpose	Number of bytes	Initial value
MPTC10P	SADDR*	Macro service pointer for INTC10 timing	2	CYLTIME
MPDC10P		Macro service pointer for INTC10 data	2	CYLDATA
MODC10		INTC10 macro service modulo register	1	PTNCNT
RCC10		INTC10 macro service ring counter	1	PTNCNT
MSCC10P		INTC10 macro service counter	2	STPCNT

*: SADDR: Short direct addressing application range (0FE20H to 0FEFFH)

(3) Input/Output parameter

None.

(4) Register used

None.

(5) Stack used

The stack size used for the subroutine and interrupt processing for this application is 3 bytes (Nesting level: 1).

(6) Mode register setting

Port 0 mode register

	7	6	5	4	3	2	1	0	
PM0	PM07	PM06	PM05	PM04	PM03	PM02	PM01	PM00	(00H when RESET is input)
Setting example	0	0	0	0	0	0	0	0	

PM0n	P0n pin input/output mode specificatin (n=0 to 7)
00H	Output mode (output buffer ON)
FFH	High impedance state (output buffer OFF)
Other than above	Not allowed.

Real-time output port control register

	7	6	5	4	3	2	1	0	
RTPC	BYTE	Note 0	Note 0	POMH	EXTR	Note 0	Note 0	POML	(00H when RESET is input)
Setting example	0	0	0	0	0	0	0	1	

POML	POL function specification
0	Port mode
1	Real-time output port mode

EXTR	Enables data transfer from buffer register to output latch by INTP0
0	Disable
1	Eanbles BYTE = 0 : Transfers POL only BYTE = 1 : Transfers POL/H

POMH	POH function specification
0	Port mode
1	Real-time output port mode

BYTE	Real-time output port mode operation mode
0	4-bit separate real-time output port
1	8-bit real-time output port

Note: 0 must always be written to bits 1, 2, 5, and 6.

Capture/compare control register 1

	7	6	5	4	3	2	1	0	
CRC1	0	0	0	0	CLR11	CM	CLR10	0	(00H when RESET is input)
Setting example	0	0	0	0	0	0	0	0	

CLR10	Clear operation by coincidence of TM1 and CR10 register contents
0	Disables
1	Enables

CLR11	CM	CR11 register operation specification	TM1 clear operation
0	0	Compare register	Disables
1	0		Enables (when TM1 and CR11 register contents coincide)
0	1	Capture register	Disables
1	1		Enables (when TM1 contents are captured into CR11 register)

Prescaler mode register 1

	7	6	5	4	3	2	1	0	
PRM1	PRS23	PRS22	PRS21	PRS20	0	PRS12	PRS11	PRS10	(00H when RESET is input)
Setting example	0	0	0	0	0	1	1	1	

8-bit timer/counter 1

PRS12	PRS11	PRS10	Timer/counter 1 count clock frequency specification
0	0	0	$f_{CLK}/16$ Note
0	0	1	
0	1	0	
0	1	1	$f_{CLK}/32$
1	0	0	$f_{CLK}/64$
1	0	1	$f_{CLK}/128$
1	1	0	$f_{CLK}/256$
1	1	1	$f_{CLK}/512$

(Cont'd)

8-bit timer/counter 2

PRS23	PRS22	PRS21	PRS20	Timer/counter 3 count clock frequency specification
0	0	0	0	$f_{CLK}/16$
0	0	0	1	
0	0	1	0	
0	0	1	1	$f_{CLK}/32$
0	1	0	0	$f_{CLK}/64$
0	1	0	1	$f_{CLK}/128$
0	1	1	0	$f_{CLK}/256$
0	1	1	1	$f_{CLK}/512$
1	1	1	1	External clock (CI)

Note: f_{CLK}: Internal system clock frequency ($f_{CLK}/2$)

Macro service mode register

	7	6	5	4	3	2	1	0	
	CHT2	CHT1	CHT0	0	MOD3	MOD2	MOD1	MOD0	(00H when RESET is input)
Setting example	1	1	0	1	1	1	1	0	

			CHT0	0	1	0	1	
			CHT1	1	0	1		
			CHT2	0	1	1		
						Type C		
MOD3	MOD2	MOD1	MOD0	Type A	Type B	Retains MPTL	Increments MPTL	
0	0	0	0	Memory-to-SFR data transfer				
0	0	0	1	SFR-to-Memory data transfer				
1	0	0	0			Without ring contol	Data transfer only	P0L
1	0	0	1					P0H
1	0	1	0				With automatic addition	P0L
1	0	1	1					P0H
1	1	0	0			With ring control	Data transfer only	P0L
1	1	0	1					P0H
1	1	1	0				With automatic addition	P0L
1	1	1	1					P0H

Interrupt service mode register L

	7	6	5	4	3	2	1	0	
ISM0L	CISM11	CIMS10	CISM01	CISM00	PISM3	PISM2	PISM1	PISM0	(00H when RESET is input)
Setting example	x	1	x	x	x	x	x	x	x: Not manipulated

ISM	Interrupt service flag
0	Processes in vector interrupt
1	Processes in macro service

Interrupt mask register L

	7	6	5	4	3	2	1	0	
MK0L	CMK11	CMK10	CMK01	CMK00	PMK3	PMK2	PMK1	PMK0	(00H when RESET is input)
Setting example	x	0	x	x	x	x	x	x	x: Not manipulated

MK	Interrupt service flag
0	Interrupt not generated
1	Interrupt generated

Timer control register 1

	7	6	5	4	3	2	1	0	
TMC1	CE2	OVF2	CMD2	0	CE1	OVF1	0	0	(00H when RESET is input)
Setting example	0	0	0	0	1	0	0	0	

8-bit timer/counter 1

OVF1	TM1 overflow flag
0	No overflow
1	Overflow (count up from FFH to 00H)

Remarks: This bit is reset only by the software.

CE1	TM1 count operation control
0	Stops count operation in cleared condition
1	Enables count operation

8-bit timer/counter 2

CMD2	TM2 count operation mode specification
0	Normal mode
1	one-shot mode

OVF2	TM2 overflow flag
0	No overflow
1	Overflow (count up from FFH to 00H)

Remarks: This bit is reset only by the software.

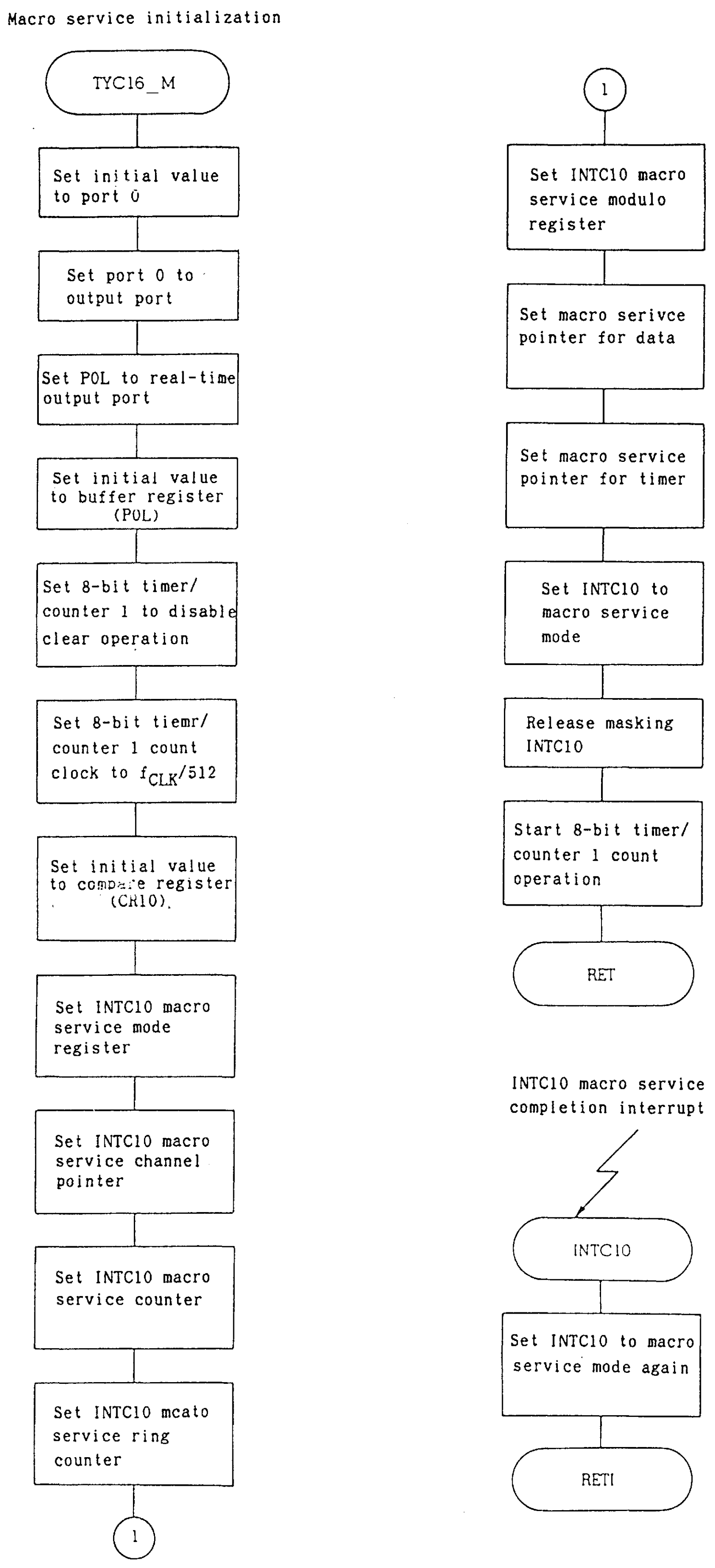
CE2	TM2 count operation control
0	Stops count operation in cleared condition
1	Enables count operation

(7) Program example

An example of ths program is as shown below.

```
        EXTRN   TYC16_M                 ; package
                  :
                  :
        CSEG
  MAIN:
                  :
                  :
        CALL    !TYC16_M                ; call TYC16_M routine
        EI                              ; enable interrupt
                  :
                  :
```

(8) Flow chart



Macro service initialization
TYC16_M
Set initial value to port 0
Set port 0 to output port
Set POL to real-time output port
Set initial value to buffer register (P0L)
Set 8-bit timer/ counter 1 to disable clear operation
Set 8-bit tiemr/ counter 1 count clock to fCLK/512
Set initial value to compare register (CR10).
Set INTC10 macro service mode register
Set INTC10 macro service channel pointer
Set INTC10 macro service counter
Set INTC10 mcato service ring counter
1
1
Set INTC10 macro service modulo register
Set macro serivce pointer for data
Set macro service pointer for timer
Set INTC10 to macro service mode
Release masking INTC10
Start 8-bit timer/ counter 1 count operation
RET
INTC10 macro service completion interrupt
INTC10
Set INTC10 to macro service mode again
RETI

(9) Program list

In this program example, vector table address and macro service area definition file (INTMS.DEF) and SFR bit name definition file (SFRBIT.DEF) are included in the beginning of the source program.

```
$       IC(INTMS.DEF)
        NAME    TYC16M
;******************************************************
;*      MACRO SERVICE TYPE C MSC=16bit                *
;*                              uPD78234,uPD78244     *
;*              for stepping motor control            *
;******************************************************

$       IC(SFRBIT.DEF)
        PUBLIC  TYC16_M                 ; package

STPCNT  EQU     0                       ; step count (65536)
PTNCNT  EQU     4                       ; pattern count

;       *** define INTC10 macro service area ***

        DSEGCR16 C10,'SADDR'            ; macro service channel(MSC=16bit)
        CMCW10                          ; macro service control word

;       *** define vector table ***

        CVENT10                         ; INTC10 vector table

TYC16MS CSEG
TYC16_M:
        MOV     P0,#00001000B           ; P0 output latch <- first data
        MOV     PM0,#00000000B          ; set P0 output port
        MOV     RTPC,#00000001B         ; initialize realtime output port
        MOV     P0L,#00001000B          ; set P0 buffer register first data

        MOV     CRC1,#00000000B         ; disable clear TM1
        MOV     PRM1,#00000111B         ; set TM1 prescaler fCLK/512
        MOV     CR10,#0                 ; set first timing data
        MOV     MSMC10,#11011110B        ; set mode register (TYPE C)
        MOV     CHPC10,#LOW CHAC10       ; set channel pointer
        MOVW    MSCC10P,#STPCNT          ; set macro service counter
        MOV     RCC10,#PTNCNT            ; set ring counter
        MOV     MODC10,#PTNCNT           ; set modulo register
        MOVW    MPDC10P,#CYLDATA         ; set pointer for data
        MOVW    MPTC10P,#CYLTIME         ; set pointer for timing

        SET1    CISM10                   ; set INTC10 macro service mode
        CLR1    CMK10                    ; open INTC10 mask

        MOV     TMC1,#00001000B          ; start TM1
        RET

;****************************************
;*      end of INTC10 macro service     *
;****************************************

INTC10:
        SET1    CISM10                   ; set INTC10 macro service mode again
        RETI

;****************************************
;*      define data                     *
;****************************************

;       *** for data ***

CYLDATA:
        DB      00000001B
        DB      00000010B
        DB      00000100B
        DB      00001000B

;       *** for timing ***

CYLTIME:
        DB      58

        END
```

CHAPTER 8 A/D CONVERTER PROGRAM EXAMPLES (uPD78214)

uPD78214 is provided with a digital-to-analog (A/D) converter having six multiplexed analog inputs (AN0 to AN7).
This A/D converter is a successive approximation category and has an 8-bit A/D conversion result register (ADCR) that holds the result of the conversion. The A/D conversion is performed in the following two modes:

o Scan mode:
  Several analog inputs are selected, one after another, to input data to be converted from each pin.

o Select mode:
  Only one analog input pin is used for continuous conversion.

In the program example presented in this section, the A/D converter is used in the scan mode, in which input pins AN0 to AN3 are scanned. Each channel is sampled 16 times and the average value is obtained.

(1) Operation

A memory area from address C000H to C03FH is used as a sampling data storage area, while an area consisting of addresses C040H to C043H stores the average value.
The A/D converter conversion time uses 30 microseconds (where f_{CLK} = 6 MHz). Therefore, macro service B is used to sample data, in order to reduce the CPU overhead. The average value is calculated by macro service end interrupt processing. While this processing is performed, conversion is stopped.

Fig. 8-1 Memory Mapping for A/D Converter Program Example

(a) Conversion result storage area

C000	C001	C002	C003	C004	C005	C006	
AN0	AN1	AN2	AN3	AN0	AN1	AN2	------

	C03A	C03B	C03C	C03D	C03E	C03F
------	AN2	AN3	AN0	AN1	AN2	AN3

(b) Average value storage area

C040	C041	C042	C043
AN0	AN1	AN2	AN3

(2) Program ... Refer to (6) Program list

- System initialization processing [label: ADMAIN]

(a) Mode registers related to memory are set.
(b) Port 6 mode register is cleared to 0 to select from 0000H to FFFFH bank.
(c) The stack pointer is set to FC00H.
(d) INTAD macro service is initialized.

- INTAD macro service initialization processing [label: MS_AD]

(a) The macro service mode register is set so that macro service mode B is selected and that data is transferred from SFR to memory.
(b) The channel pointer is set.
(c) The macro service counter is set to 40H.
(d) The lower 8 bits for the ADCR address, 6AH, are set in the SFR pointer.

(e) The first address for the conversion result area, C000H, is set in the macro service pointer.

(f) INTAD is set in macro service mode.

(g) Interrupt request INTAD is unmasked.

(h) Interrupt is enabled.

(i) Analog input pins AN0 to AN3 are set in the scan mode, and the A/D converter is started.

- INTAD macro service end interrupt processing
  [label: END_AD]

(a) The A/D converter is stopped.

(b) The A/D conversion end interrupt request flag (ADIF) is cleared to 0.

(c) The registers are saved.

(d) The total conversion results for individual channels are calculated.

(e) The average value is calculated by shifting the total value four times to the left.

(f) The average value is stored in addresses C040H to C043H.

(g) The macro service counter and macro service pointer are set again.

(h) The registers are restored.

(i) INTAD is set in the macro service mode.

(j) The A/D converter is restarted.

(3) Mode register setting

Memory expansion mode register

	7	6	5	4	3	2	1	0	
MM	0	MM6	PW21	PW20	0	0	0	0	(20H when RESET is input)
Setting example	0	1	1	0	0	0	0	0	

PW21	PW20	Specifies the number of wait state (range: 00000H to 0FE7FH)
0	0	0
0	1	1
1	0	2
1	1	The number of wait states specified by external wait pin

MM6	Controls 1M-byte expansion
0	P60 to P63 function as general output port.
1	P60 to P63 function as higher address output pins (A16 to A19) in external memory expansion mode

Programmable wait control register

	7	6	5	4	3	2	1	0	
PW	PW31	PW30	0	0	0	0	0	0	(80H when RESET is input)
Setting example	1	0	0	0	0	0	0	0	

PW31	PW30	Specifies the number of wait state in 10000H to FFFFFH
0	0	0
0	1	1
1	0	2
1	1	The number of wait states specified by external wait pin

Refresh mode register

	7	6	5	4	3	2	1	0
RFM	RFLV	0	0	RFEN	0	0	RFT1	RFT0
Setting example	0	0	0	0	0	0	0	0

(00H when RESET is input)

f_{CLK} = 6 MHz

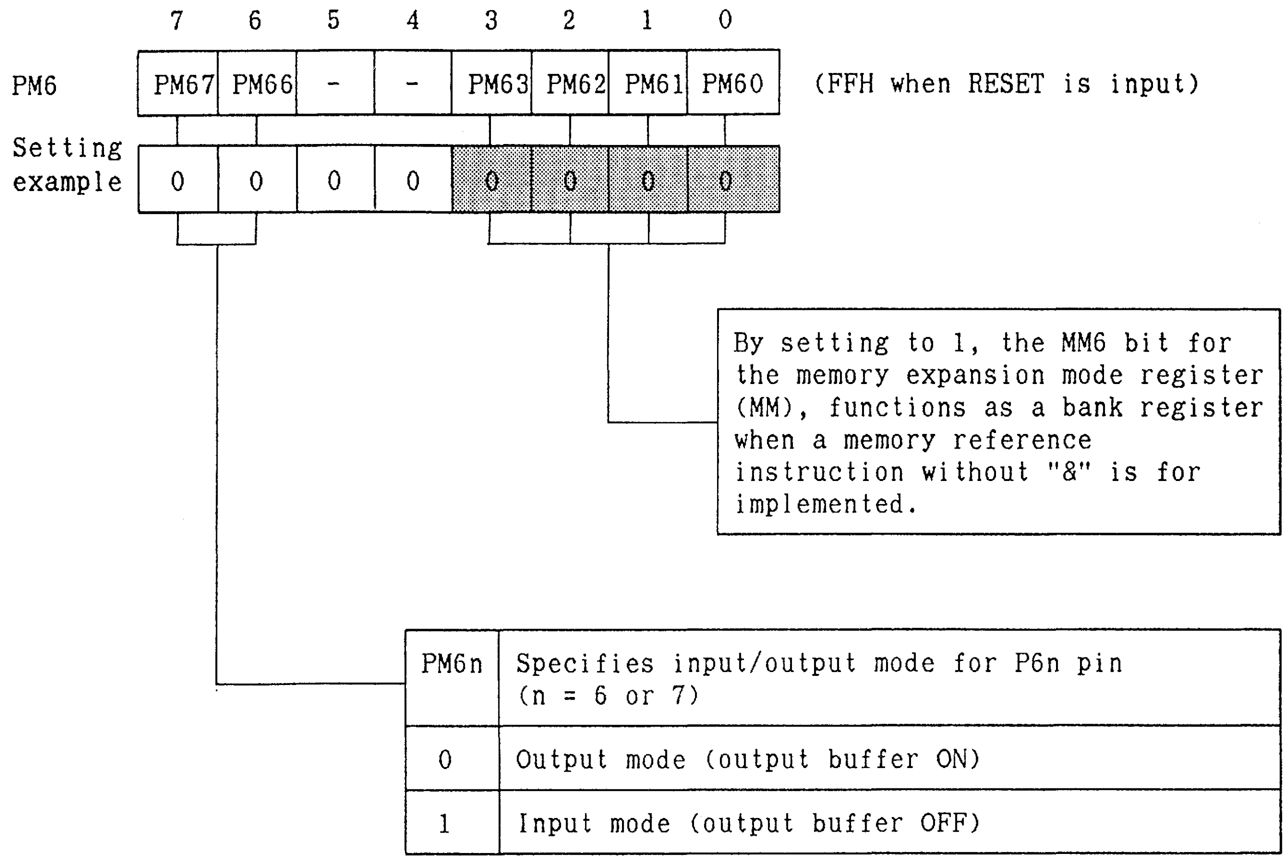
RFT1	RFT0	Specifies refresh pulse output cycle
0	0	$16/f_{CLK}$ (2.6 us)
0	1	$32/f_{CLK}$ (5.3 us)
1	0	$64/f_{CLK}$ (10.7 us)
1	1	$128/f_{CLK}$ (21.3 us)

Remarks: f_{CLK} is the internal system clock.

RFLV	RFEN	Controls RFREQ pin output
x	0	Port mode
0	1	Self-refresh operation (REFRQ: low-level output)
1	1	Enables refresh pulse output

Remarks: x indicates 1 or 0.

Port 6 mode register

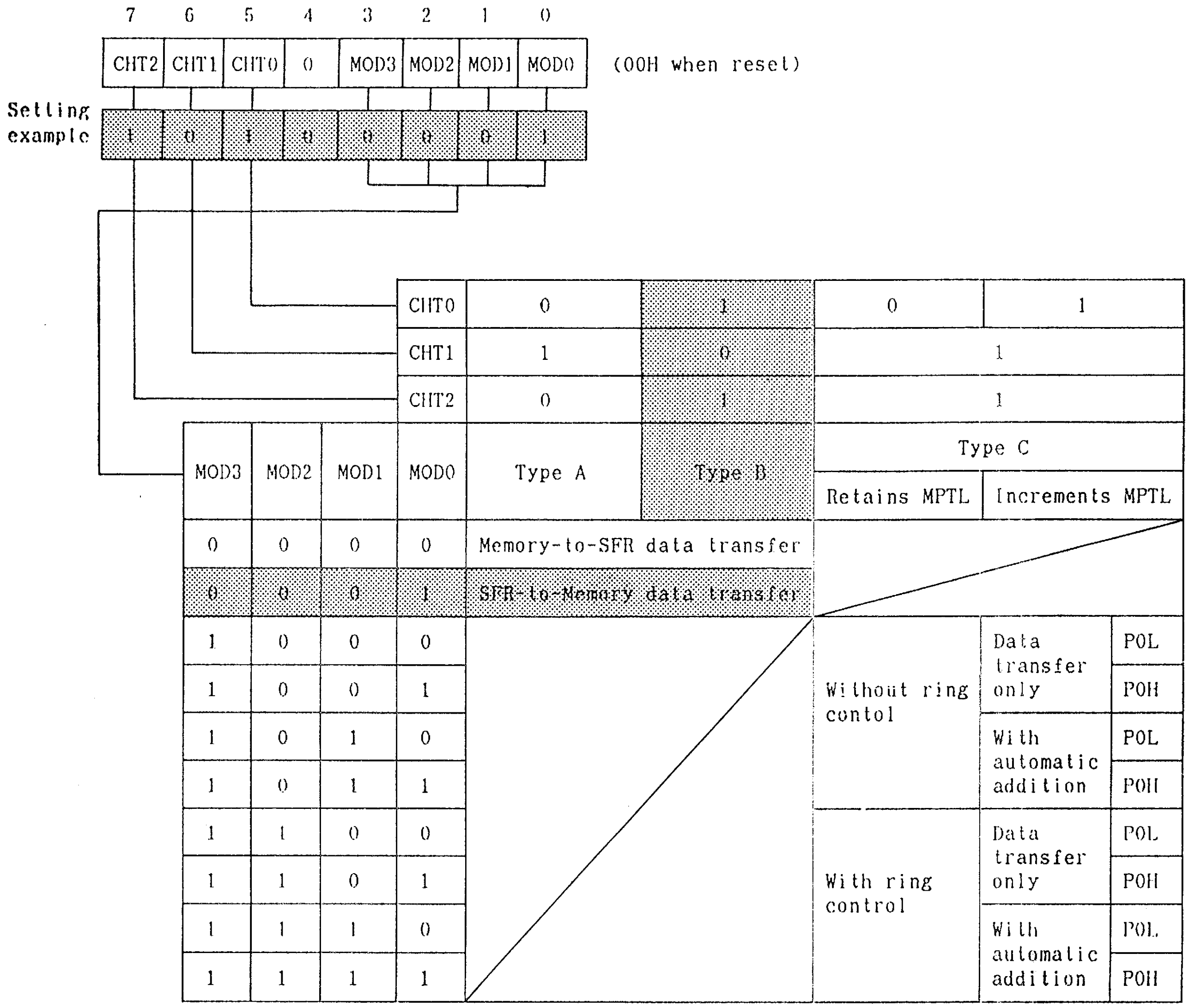


By setting to 1, the MM6 bit for the memory expansion mode register (MM), functions as a bank register when a memory reference instruction without "&" is for implemented.

PM6n	Specifies input/output mode for P6n pin (n = 6 or 7)
0	Output mode (output buffer ON)
1	Input mode (output buffer OFF)

Note: Bits marked "-" can be 1 or 0.

Macro service mode register



7	6	5	4	3	2	1	0	
CHT2	CHT1	CHT0	0	MOD3	MOD2	MOD1	MOD0	(00H when reset)
1	0	1	0	0	0	0	1	Setting example

			CHT0	0	1	0	1	
			CHT1	1	0	1		
			CHT2	0	1	1		
MOD3	MOD2	MOD1	MOD0	Type A	Type B	Type C		
						Retains MPTL	Increments MPTL	
0	0	0	0	Memory-to-SFR data transfer				
0	0	0	1	SFR-to-Memory data transfer				
1	0	0	0			Without ring contol	Data transfer only	POL
1	0	0	1					POH
1	0	1	0				With automatic addition	POL
1	0	1	1					POH
1	1	0	0			With ring control	Data transfer only	POL
1	1	0	1					POH
1	1	1	0				With automatic addition	POL
1	1	1	1					POH

A/D converter mode register

	7	6	5	4	3	2	1	0	
ADM	CS	TRG	0	FR	ANI2	ANI1	ANI0	MS	(20H when reset)
Setting example	1	0	0	0	0	1	1	0	

ANI2	ANI1	ANI0	MS	A/D conversion operation mode specification	
0	0	0	0	Scan mode	Scans AN0 input
0	0	1	0		Scans AN0, AN1 inputs
0	1	0	0		Scans AN0 to AN2 inputs
0	1	1	0		Scans AN0 to AN3 inputs
1	0	0	0		Scans AN0 to AN4 inputs
1	0	1	0		Scans AN0 to AN5 inputs
1	1	0	0		Scans AN0 to AN6 inputs
1	1	1	0		Scans AN0 to AN7 inputs
0	0	0	1	Select mode	Scans AN0 input
0	0	1	1		Scans AN1 input
0	1	0	1		Scans AN2 input
0	1	1	1		Scans AN3 input
1	0	0	1		Scans AN4 input
1	0	1	1		Scans AN5 input
1	1	0	1		Scans AN6 input
1	1	1	1		Scans AN7 input

FR	Conversion time control	
0	$180/f_{CLK}$ (Note)	$f_{CLK} > 4$ MHz
1	$120/f_{CLK}$ (Note)	$f_{CLK} \leqq 4$ MHz

TRG	External pin trigger control
0	Disables external trigger
1	Enables external trigger

CS	A/D conversion operation control
0	Stops A/D conversion operation
1	Starts A/D conversion operation

Note: f_{CLK}: System clock oscillation

Interrupt service mode register H

	7	6	5	4	3	2	1	0	
ISM0H	CSIISM	STISM	SRISM	0	0	PISM5	PISM4	0	(00H when RESET is input)
Setting example	x	x	x	x	x	1	x	x	x: Not used

ISM	Interrupt service mode flag
0	Vector interrupt processing
1	Macro service processing

Interrupt mask register H

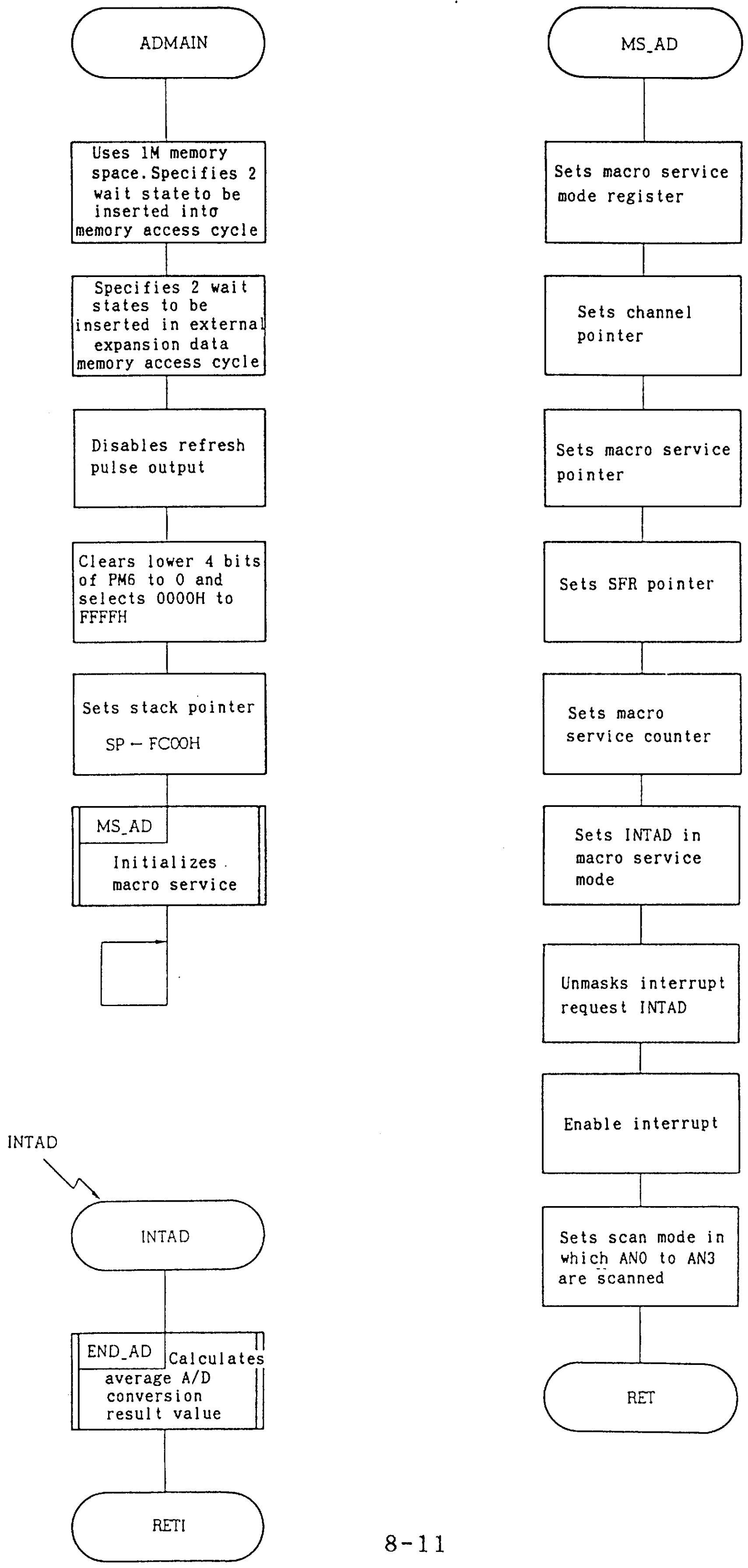
	7	6	5	4	3	2	1	0	
MK0H	CSIMK	STMK	SRMK	SERMK	CMK20	PMK5	PMK4	CMK21	(FFH when RESET is input)
Setting example	x	x	x	x	x	0	x	x	x: Not used

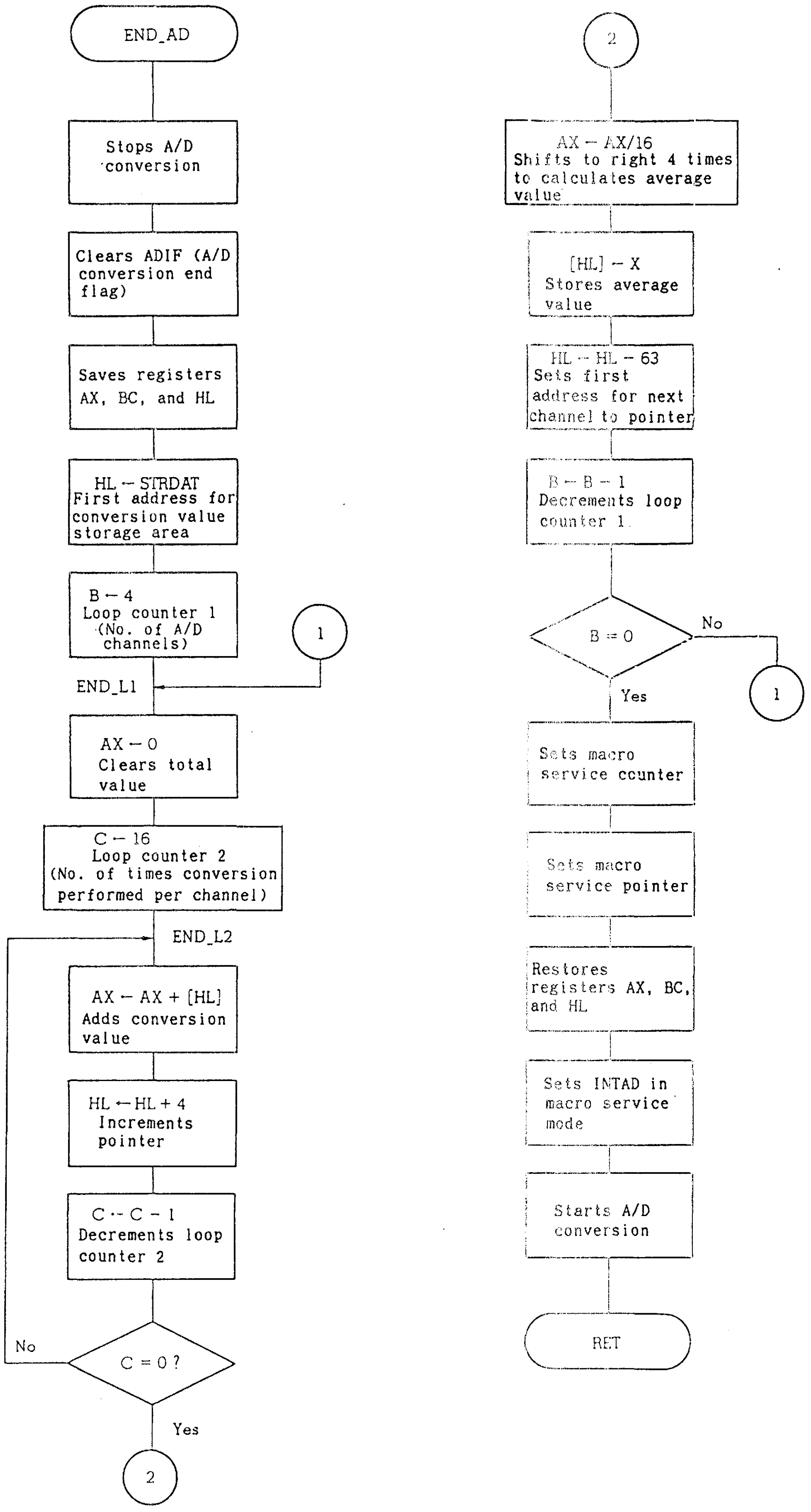
MK	Interrupt service flag
0	Enables interrupt processing
1	Keeps interrupt processing pending

(4) Input/output parameters and registers

Omitted because this is an absolute program.

(5) Flowchart


ADMAIN
Uses 1M memory space. Specifies 2 wait state to be inserted into memory access cycle
Specifies 2 wait states to be inserted in external expansion data memory access cycle
Disables refresh pulse output
Clears lower 4 bits of PM6 to 0 and selects 0000H to FFFFH
Sets stack pointer SP ← FC00H
MS_AD
Initializes macro service
INTAD
INTAD
END_AD
Calculates average A/D conversion result value
RETI
MS_AD
Sets macro service mode register
Sets channel pointer
Sets macro service pointer
Sets SFR pointer
Sets macro service counter
Sets INTAD in macro service mode
Unmasks interrupt request INTAD
Enable interrupt
Sets scan mode in which AN0 to AN3 are scanned
RET



END_AD
Stops A/D conversion
Clears ADIF (A/D conversion end flag)
Saves registers AX, BC, and HL
HL ← STRDAT First address for conversion value storage area
B ← 4 Loop counter 1 (No. of A/D channels)
1
END_L1
AX ← 0 Clears total value
C ← 16 Loop counter 2 (No. of times conversion performed per channel)
END_L2
AX ← AX + [HL] Adds conversion value
HL ← HL + 4 Increments pointer
C ← C − 1 Decrements loop counter 2
C = 0 ?
No
Yes
2
2
AX ← AX/16 Shifts to right 4 times to calculate average value
[HL] ← X Stores average value
HL ← HL − 63 Sets first address for next channel to pointer
B ← B − 1 Decrements loop counter 1
B = 0
No
Yes
1
Sets macro service counter
Sets macro service pointer
Restores registers AX, BC, and HL
Sets INTAD in macro service mode
Starts A/D conversion
RET

(6) Program list

```
        NAME    MS_ADR

;**********************************************************
;       A/D Converter application by macro service
;**********************************************************

STACK   EQU     0FC00H
M214    EQU     01000111B       ; uPD78214 MM data

PWDAT   EQU     10000000B       ; programable wait
RFDAT   EQU     10010000B       ; refresh mode
PM6DAT  EQU     0F0H            ; PM6 data

MSMAD   EQU     0FED2H          ; INTAD macro service mode register
CHPAD   EQU     0FED3H          ; INTAD macro service channel pointer
MSCAD   EQU     0FECEH          ; INTAD macro service counter
SFRAD   EQU     0FECFH          ; INTAD SFR pointer
MPADP   EQU     0FED0H          ; INTAD memory pointer

ADISM   EQU     ISM0H.2         ; ISMAD flag
CSAD    EQU     ADM.7           ; A/D convert start bit
ADMK    EQU     MK0H.2          ; INTAD mask flag
ADIF    EQU     IF0H.2          ; INTAD flag
;
;       --- for data area ---
;
CNTNUM  EQU     40H
LP2     EQU     16
LP1     EQU     CNTNUM/LP2
AD_D    DSEG    AT      0C000H  ; result store area
STRDAT: DS      CNTNUM
;
;       --- vector table ---
;
RSTVT   CSEG    AT 00000H
        DW      RST             ; reset
INTADVT CSEG    AT 00010H
        DW      INTAD           ; INTAD
;
;       --- main ---
;
AD_C    CSEG
        ORG     0080H
RST:
        MOV     MM,#M214        ; set memory mapping register
        MOV     PW,#PWDAT       ; set programable wait register
        MOV     RFM,#RFDAT      ; set refresh mode register
        MOV     PM6,#PM6DAT
        MOVW    SP,#STACK       ; set stack pointer

        CALL    !MS_AD          ; macro service initialize for INTAD
DMLP:                           ; dummy loop
        BR      DMLP
;
INTAD:                          ; complete of INTAD macro service
        CALL    !END_AD
        RETI
```

```
;
;       *** initialize ***
;
MS_AD:
        MOV     MSMAD,#10100001B
                                ; set macro service mode register
        MOV     CHPAD,#0D1H     ; set macro service channel pointer
        MOV     MSCAD,#CNTNUM   ; set macro service counter
        MOV     SFRAD,#LOW(ADCR)  ; set SFR pointer
        MOVW    MPADP,#STRDAT   ; set macro service memory pointer
;
        SET1    ADISM           ; set interrupt mode
        CLR1    ADMK            ; open mask of INTAD
        EI                      ; interrupt enable
        MOV     ADM,#10000110B  ; initialize A/D converter
        RET
;
;       *** interrupt of complete macro-service ***
;
END_AD:
        CLR1    CSAD            ; A/D converter stop
        CLR1    ADIF            ; INTAD flag clear
        PUSH    AX              ; save register
        PUSH    BC
        PUSH    HL

        MOVW    HL,#STRDAT      ; pointer set
        MOV     B,#LP1          ; loop counter_1 set (4)
EAD_L1:
        MOVW    AX,#0           ; sum clear
        MOV     C,#LP2          ; loop counter_2 set (16)
EAD_L2:
        XCH     A,X             ; addition result
        ADD     A,[HL]
        XCH     A,X
        ADDC    A,#0            ; carry addition

        INCW    HL              ; pointer <- pointer + 4
        INCW    HL
        INCW    HL
        INCW    HL
        DBNZ    C,$EAD_L2       ; check loop counter-2

        SHRW    AX,4            ; average of result
        XCH     A,X
        MOV     [HL],A
        MOVW    AX,HL           ; next pointer set
        SUBW    AX,#63          ; 16*4-1
        MOVW    HL,AX
        DBNZ    B,$EAD_L1       ; check loop counter-1

        MOV     MSCAD,#CNTNUM   ; set macro service counter
        MOVW    MPADP,#STRDAT   ; set macro service memory pointer

        POP     HL              ; restore register
        POP     BC
        POP     AX
        SET1    ADISM
        SET1    CSAD            ; A/D convert start

        RET
```

```
        END
```

CHAPTER 9 COMPARATOR PROGRAM EXAMPLE (uPD78224)

uPD78224 and 78220 have eight input pins whose threshold voltages are variable.
This section introduces the program which measures voltage by using the comparator.

(1) Operation

Voltage input to the PT4 pin is successively converted into digital data at a 4 bits resolution and by means of binary search.
The conversion result is stored in the accumulator. Fig. 9-1 illustrates successive conversions. In this figure, the voltage applied to the PT4 pin is represented as V_{PT4} and the comparison reference voltage, as V_{DD}.

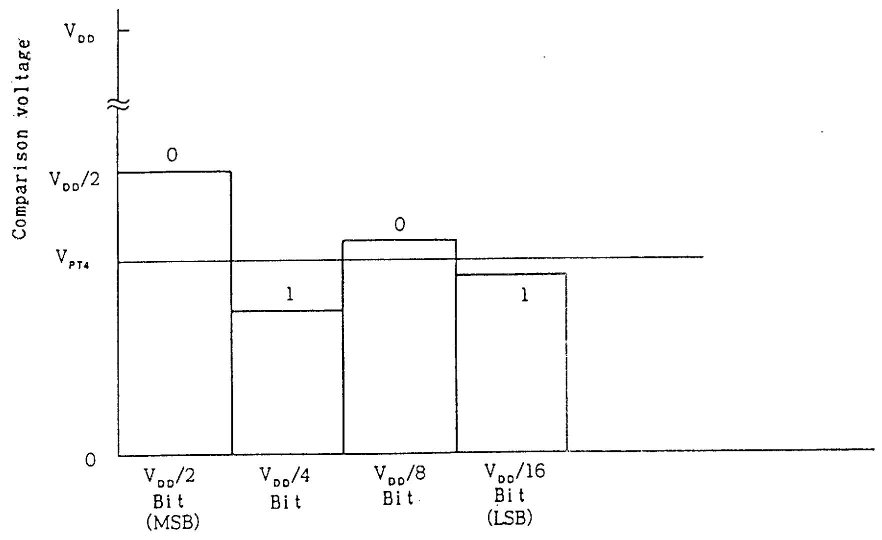
(i) Where comparison voltage $V_{DA} = V_{DD}/2$, $V_{DA} > V_{PT4}$. Therefore, the MSB (bit $V_{DD}/2 = 0$) is regarded as 0.

(ii) Where $V_{DA} = V_{DD}/4$, $V_{DA} < V_{PT4}$. Therefore, bit $V_{DD}/4$ is regarded as 1.

(iii) Where $V_{DA} = V_{DD} \times 3/8$ ($V_{DD}/4 + V_{DD}/8$), $V_{DA} > V_{PT4}$. Therefore, bit $V_{DD}/8$ is regarded as 0.

(iv) Where $V_{DA} = V_{DD} \times 5/16$ ($V_{DD}/4 + V_{DD}/16$), $V_{DA} < V_{PT4}$. Therefore, the LSB (bit $V_{DD}/16 = 1$) is regarded as 1.

Fig. 9-1 A/D Conversion by Port T



(2) Program ... Refer to (7) Program list

(a) The initial comparison voltage is set to $V_{DD}/2$.

(b) As a value that gives weight to the comparison voltages, 08H is set in register X.

(c) The program waits until the conversion time, 15.8 microseconds (at f_{CLK} = 5 MHz), elapses.

(d) The comparison voltage is read into register A.

(e) The weight-giving value in register X is halved.

(f) If a carry is generated, execution branches to (k).

(g) The voltage placed on the PT4 pin as a result of the conversion is checked. Implementation branches to (i), if $V_{PT4} < V_{REF}$.

(h) The previous comparison voltage (register A contents) is added to the register X contents. Implementation branches to (j).

(i) The register X contents are subtracted from the previous comparison voltage (the contents of register A).

(j) PMT register comparison voltage is set in register A. Implementation branches to (c).

(k) The comparison result for the least significant bit is read, which is regarded as the least significant bit for register A.

(l) The higher 4 bits for register A are cleared to 0.

(3) Mode register setting

Port T mode register

	7	6	5	4	3	2	1	0	
PMT	PUPH	PUPL	0	0	MT3	MT2	MT1	MT0	(00H when RESET is input)
Setting example	0	0	0	0	-	-	-	-	-: This setting value will be changed depending on each case.

MT3	MT2	MT1	MT0	Specifies reference voltage (V_{REF})
0	0	0	0	Stops supply of V_{REF} (Note)
0	0	0	1	1/16 x V_{DD}
0	0	1	0	2/16 x V_{DD}
0	0	1	1	3/16 x V_{DD}
0	1	0	0	4/16 x V_{DD}
0	1	0	1	5/16 x V_{DD}
0	1	1	0	6/16 x V_{DD}
0	1	1	1	7/16 x V_{DD}
1	0	0	0	8/16 x V_{DD}
1	0	0	1	9/16 x V_{DD}
1	0	1	0	10/16 x V_{DD}
1	0	1	1	11/16 x V_{DD}
1	1	0	0	12/16 x V_{DD}
1	1	0	1	13/16 x V_{DD}
1	1	1	0	14/16 x V_{DD}
1	1	1	1	15/16 x V_{DD}

(Cont'd)

PUPL/PUPH	Specifies connection of pull-up resistor
0	Disconnects pull-up resistor
1	Connects pull-up resistor

Note: Set this mode when the standby mode is set.

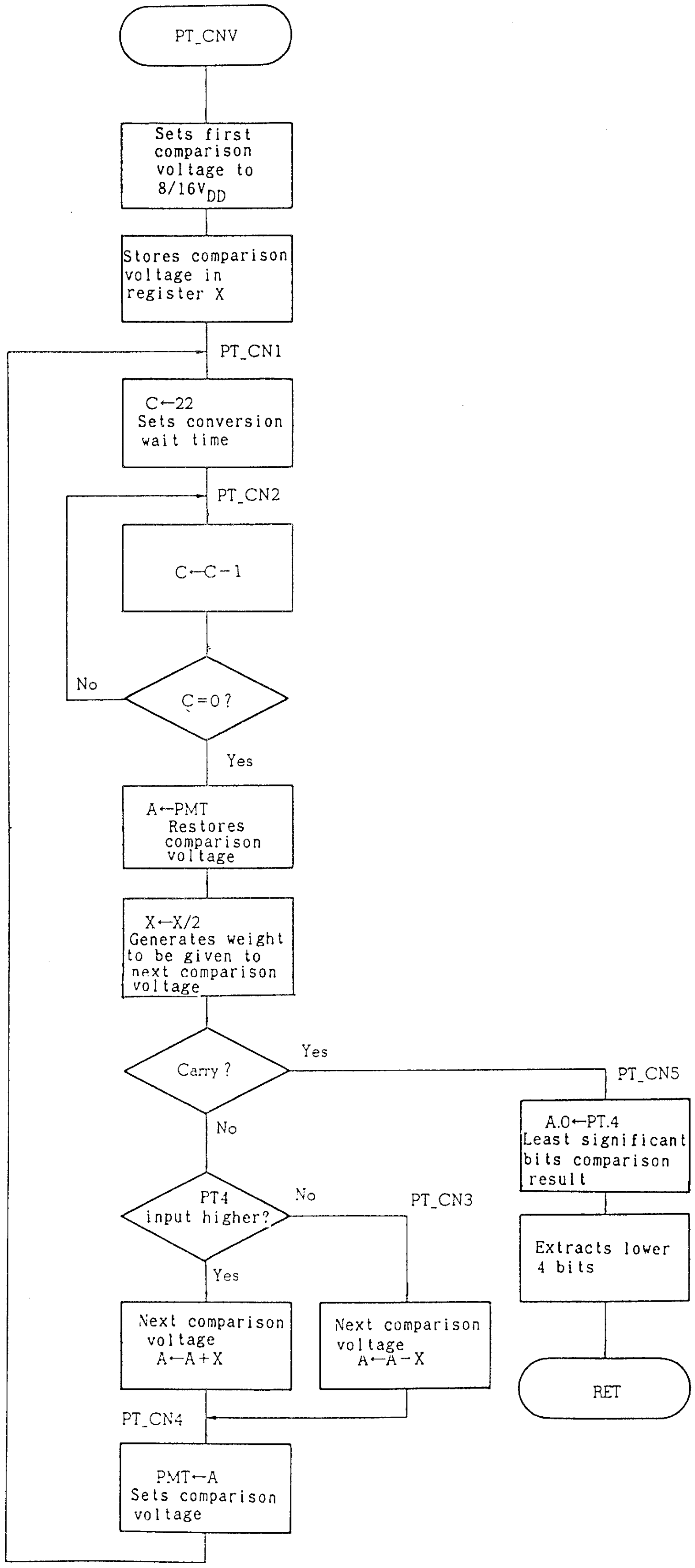
(4) Input/output parameters

The voltage input to the PT4 pin is generated and set in register A as a value from 0 to 15.

(5) Registers

X, A, C

(6) Flowchart



PT_CNV
Sets first comparison voltage to 8/16V_{DD}
Stores comparison voltage in register X
PT_CN1
C←22 Sets conversion wait time
PT_CN2
C←C−1
C=0?
No
Yes
A←PMT Restores comparison voltage
X←X/2 Generates weight to be given to next comparison voltage
Carry ?
Yes
No
PT_CN5
A.0←PT.4 Least significant bits comparison result
Extracts lower 4 bits
RET
PT4 input higher?
No
PT_CN3
Yes
Next comparison voltage A←A+X
Next comparison voltage A←A−X
PT_CN4
PMT←A Sets comparison voltage

(7) Program list

```
        NAME    PTCNVR
;
;**************************************************************
;* analogue-digital convert on                                *
;*              programable threshold port                    *
;*                      for only uPD78224,78220               *
;*      sampling port                                         *
;*              PT4                                           *
;*      result                                                *
;*              A                                             *
;**************************************************************
;
        PUBLIC  PT_CNV

PTWAIT  EQU     22              ; read result wait time
SCANPT  EQU     PT.4            ; scan port
;
        CSEG
PT_CNV:
        MOV     PMT,#08H        ; select 1/2 VDD
        MOV     X,#08H          ; compare data
PT_CN1:
        MOV     C,#PTWAIT       ; time delay
PT_CN2:
        DBNZ    C,$PT_CN2
        MOV     A,PMT
        SHR     X,1             ; add/sub parameter -> 1/2
        BC      $PT_CN5
        BF.     SCANPT,$PT_CN3  ; result check

        ADD     A,X
        BR      PT_CN4
PT_CN3:
        SUB     A,X
PT_CN4:
        MOV     PMT,A           ; next compare
        BR      PT_CN1
PT_CN5:
        MOV1    CY,SCANPT
        MOV1    A.0,CY
        AND     A,#0FH
        RET

        END
```

CHAPTER 10 EEPROM PROGRAM EXAMPLE (uPD78224)

The EEPROM is provided in the uPD78244 series as hardware, and is mapped in the 512-byte area of addresses 0FB00H to 0FCFFH in the data memory space.
The EEPROM purpose is to write a parameter (characteristics vary in the same type of products) typical to a product (such as a camera). In this case, the parameter is not changed after factory shipment.
Another EEPROM application is to write parameters which need to be saved, even after the power is turned off.
The following example is for these two types of application.
The source program example for this application is written in the 78K/II series Structured Assembler (ST78K2).

10.1 Operation Outline

(1) Outline

Continuously writes the internal RAM data in the continuous area in the EEPROM, and sets write protect on the area (hereinafter, this operation is expressed as continuous write mode).
When the program is reset, an immediate data is written once into another 1 byte area, when a write request is generated (hereinafter, this operation is expressed as 1-byte write mode). If an write error occurs, error processing will be performed.

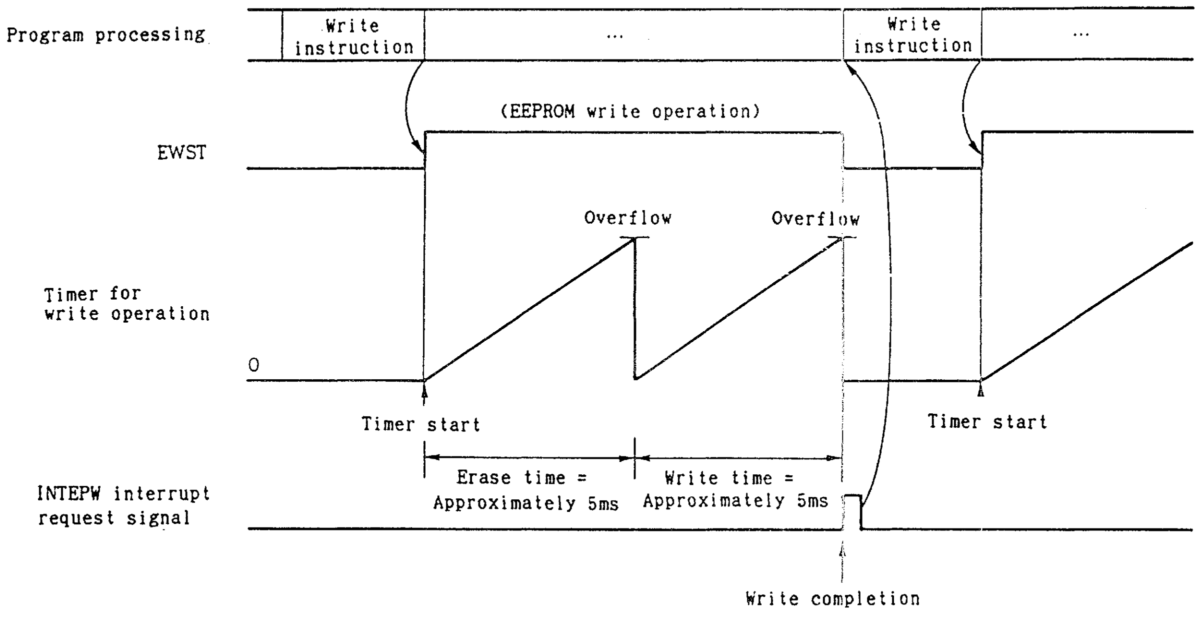
(2) Method to write in program

Approximately 10 seconds (write processing time) are required to write to the EEPROM. This time period is assured by the hardware using the internal timer for writing.

When continuously writing to the EEPROM, the write processing time must be taken before the next write operation. If this time period is not taken, an overwrite error will occur.

Fig. 10-1 shows a timing chart for writing to the EEPROM.

Fig. 10-1 Timing Chart for Writing to EEPROM



Since other program processing is possible during write completion wait time, the program can be made very efficient, if write processing is accomplishes in interrupt processing or in flag check processing in the main program.
Using the functions provided for EEPROM as follows may be convenient:

① Accomplish EEPROM writing in write completion interrupt (INTEPW) processing.

② Accomplish EEPROM writing when EWST = 0 (not in writing), upon checking the write status flag (EWST = EWC.7).

In this program example, method ① is used in the continuous write mode, and method ② is used in the 1-byte write mode.

10.2 Program Example

(1) Processing outline (refer to (8) SPD chart)

(a) Continuous write mode [module name: WRC]

(i) Continuous mode initialization processing [label name: WRC_INI]

① Initializes write control register (EWC). Operation frequency is set to (f_{XX}) = 12MHz. Therefore, the timer clock for writing is set to 65536/f_{XX}.

② Initializes the work area (WRPNTSP, WRPNTDP, CNTAREA) necessary for write processing.

③ Sets (to 1) the INTEPW interrupt request flag (EPWIF) by dummy in order to write the first 1 byte, and releases masking, to allow the first interrupt generation.

(ii) INTEPW interrupt processing [label name: INTEPW] When the specified number of bytes has not been written, 1 byte is written and increments the write pointer.

(b) 1-byte write mode [module name: WR1]

(i) 1-byte mode initialization processing [label name: WR1_INI]
This processing is carried out when the program is reset, after a continuous writing is completed.
Write protect is set on the area to which a write was accomplished in the continuous write mdoe, and at the same time 1-byte writing is enabled.
In this program example, write protect is set on a certain area, even when no continuous write processing is carried out.

(ii) 1-byte write processing [label name: WR_1BYTE]
In this processing, the write status flag (EWST) is checked, and completion of the previous write operation is awaited (until EWST = 0). Afterwards, 1 byte is written.

(c) Write error processing [module name: WRERR]

(i) INTEER interrupt processing [label name: INTEER]
In this program example, the error processing is simply an infinite loop. However, for actual application, appropriate processing must be implemented, according to the target system.

(2) RAM used

In this program example, 5 bytes of RAM are used as work area in the continuous write mode (module name: WRC).
In addition to this, an area for the number of bytes to be written is necessary for storing write data.
Table 10-1 lists the RAM areas to be used.

Table 10-1 RAM Areas Used for EEPROM Program Example

Module	RAM name	Allocation area	Purpose	Number of bytes	Initial value
WRC	WRPNTSP	SADDR*1	Stores write source pointer	2	WRCDATA
	WRPNTDP		Stores write destination pointer	2	WRCADRS
	CNTAREA		Counts number of bytes to be written	1	WRCNT+1
	Arbi-trary*3	RAM*2	Stores write source data	Arbi-trary	Arbi-trary

*1: SADDR: Short direct addressing application area (0FE20H to 0FEFFH)
*2: RAM: Arbitrary RAM area within memory bank 0.
*3: Arbitrary means setting by the user.

(3) Input/output parameter

(a) Input parameter

Table 10-2 Input Parameters for EEPROM Program Example

Module	Parameter	Fixed/variable	Contents
WRC	WRCDATA	Fixed value	Write source address (internal RAM)
	WRCADRS	Fixed value	Write destination address (EEPROM)
	WRCNT	Fixed value	Number of write bytes
WR1	A register	Variable value	Sets write data
	HL register	Variable value	Sets write address
WRERR	None	-	-

(b) Output parameter

None.

(4) Register used

Table 10-3 Registers Used for EEPROM Program

Module	Routine	Label	Register	Bank
WRC	Initialization of continuous write mode	WRC_INI	None	-
	INTEPW interrupt processing	INTEPW	AX, DE, HL	1
WR1	Initialization of 1-byte write mode	WR1_INI	None	-
	1-byte write processing	WR_1BYTE	A, HL	0
WRERR	Write error processing	INTEER	None	-

(5) Stack used

The size of the stack used for the subroutine and interrupt processing in this program example is 3 bytes (nesting level: 1).

(6) Mode register setting example

(a) Continuous write mode

EEPROM write control register

	7	6	5	4	3	2	1	0
EWC	EWST	EOVW	WPA1	WPA0	EWE	EWP	EWTC1	EWTC0
Setting example	0	0	1	1	1	0	0	0

(34H when RESET is input)

EEPROM write timer clock selection bit

EWTC1	EWTC0	Count clock selection
0	0	$65536/f_{XX}$ ($f/_{XX}$ = 12MHz)
0	1	$49152/f_{XX}$ (8MHz $\leqq f_{XX} <$ 12MHz)
1	0	$32768/f_{XX}$ (6MHz $\leqq f_{XX} <$ 8MHz)
1	1	$24576/f_{XX}$ (4MHz $\leqq f_{XX} <$ 6MHz)

Write protect specification enable/disable bit

EWP	Write protect (write disable) specification
0	Disable (Release write protect in entire EEPROM area)
1	Enable (protects the portion specified as write protected area)

EEPROM write enable/disable control bit

EWE	EEPROM write operation
0	Disables
1	Enables

(Cont'd)

EEPROM write protect area specification bit

WPA1	WPA0	Write protect area specification
0	0	Area 3 (FC80H-FCFFH)
0	1	Area 3-2 (FC00H-FCFFH)
1	0	Area 3-1 (FB80H-FCFFH)
1	1	Area 3-0 (FB00H-FCFFH)

EEPROM overwrite error detection flag

EOVW	Overwrite error generation generated/not generated
0	Overwrite error not generated
1	Overwrite error generated

EEPROM write status flag

EWST	Write status
0	EEPROM is not being written
1	EEPROM is being written

Remarks: f_{XX} indicats the crystal/ceramic oscillator frequency.
When using the external clock frequency, read f_{XX} as f_X.

(b) 1-byte write mode

EEPROM write control register

	7	6	5	4	3	2	1	0	
EWC	EWST	EOVW	WPA1	WPA0	EWE	EWP	EWTC1	EWTC0	(34H when RESET is input)
Setting example	0	0	0	0	1	1	0	0	

EEPROM write timer clock selection bit

EWTC1	EWTC0	Count clock selection
0	0	$65536/f_{XX}$ ($f/_{XX}$ = 12MHz)
0	1	$49152/f_{XX}$ ($8MHz \leqq f_{XX} < 12MHz$)
1	0	$32768/f_{XX}$ ($6MHz \leqq f_{XX} < 8MHz$)
1	1	$24576/f_{XX}$ ($4MHz \leqq f_{XX} < 6MHz$)

Write protect specification enable/disable bit

EWP	Write protect (write disable) specification
0	Disable (Release write protect in entire EEPROM area)
1	Enable (protects the portion specified as write protected area)

EEPROM write enable/disable control bit

EWE	EEPROM write operation
0	Disables
1	Enables

(Cont'd)

EEPROM write protect area specification bit

WPA1	WPA0	Write protect area specification
0	0	Area 3 (FC80H-FCFFH)
0	1	Area 3-2 (FC00H-FCFFH)
1	0	Area 3-1 (FB80H-FCFFH)
1	1	Area 3-0 (FB00H-FCFFH)

EEPROM overwrite error detection flag

EOVW	Overwrite error generation generated/not generated
0	Overwrite error not generated
1	Overwrite error generated

EEPROM write status flag

EWST	Write status
0	EEPROM is not being written
1	EEPROM is being written

Remarks: f_{XX} indicats the crystal/ceramic oscillator frequency.
When using the external clock frequency, read f_{XX} as f_X.

(7) Program example

The following program example is to implement both continuous writing and 1-byte writing. Whether the continuous write mode or 1-byte write mode will be accomplished is determined by the port level immediately after reset. In this example, P26 is used. The continuous mode is used when the port level is low (WRMODE=0). The 1-byte mode is used when the port level is high (WRMODE=1).

```
        PUBLIC  WRCDATA,WRCADRS,WRCNT    ; data
        EXTRN   WRC_INI,WR1_INI,WR_1BYTE; package

EERMK   EQU     MK1L.0                   ; INTEER mask flag
WRMODE  EQU     P2.6                     ; judge writing mode port

;       --- define area for writing 1byte ---

WR1DATA EQU     0FFH                     ; write data

EPRMS1  DSEG
WR1ADRS:DS      1                        ; write area (EEPROM)

;       --- define area for writing continuously ---

WRCNT   EQU     32                       ; write count

WRDS    DSEG
WRCDATA:DS      WRCNT                    ; source area of writing

EPRMS2  DSEG
WRCADRS:DS      WRCNT                    ; destination area of writing(EEPROM)
            :
            :
        CSEG
RST:
            :
            :
        CLR1    EERMK                    ; open INTEER mask
        if_bit(!WRMODE)                  ; IF writing continuously mode
                CALL    !WRC_INI         ; initialize of writing continuously
                EI                       ; enable interrupt
                while(forever)           ; LOOP forever
                endw
        endif                            ; IF writing 1 byte mode
        CALL    !WR1_INI                 ; initialize of writing 1 byte
        EI                               ; enable interrupt
            :
            :
        HL=#WR1ADRS                      ; set write address
        A=#WR1DATA                       ; set write data
        CALL    !WR_1BYTE                ; writing 1 byte
            :
            :
```

Note: It is recommended to define the address for the EEPROM area (data segment EPRMS1 and EPRMS2 in this example) in the link directive file in advance, then link this directive file when linking the program.
The following example shows directive file contents.

```
MEMORY  EEPROM0 : (0FB00H,128) / REGULAR
MEMORY  EEPROM1 : (0FB80H,128) / REGULAR
MEMORY  EEPROM2 : (0FC00H,128) / REGULAR
MEMORY  EEPROM3 : (0FC80H,128) / REGULAR
;
MERGE   EPRMS1  : = EEPROM0
MERGE   EPRMS2  : = EEPROM3
```

(8) SPD chart

- Continuous write initialization process

```
WRC_INI ─┬─── Releases write protect
         │    Sets the timer clock for write operation to
         │    65536/fXX
         │    Enables write operation
         ├─── Sets write source start address to memory
         │    (WRPNTSP)
         ├─── Sets write destination start address to memory
         │    (WRPNTDP)
         ├─── Sets number of write bytes + 1  to memory
         │    (CNTAREA)
         ├─── Sets INTEPW interrupt request flag
         ├─── Releases INTEPW masking
         └─── Return
```

- Continuous write processing (interrupt)

```
INTEPW ──┬───
         ├─── Selects register bank 1
         ├◇── Decrements write counter (CNTAREA value)
         │ ├─ (IF: Not all the specified number of bytes has
         │ │  been written)
         │ │(THEN)
         │ ├─── Reads write source pointer from memory
         │ │    (WRPNTSP)
         │ ├─── Reads write destination pointer from memory
         │ │    (WRPNTDP)
         │ ├─── Writes 1 byte
         │ │    Increments write pointer
         │ ├─── Stores write source pointer into memory
         │ │    (WRPNTSP)
         │ └─── Stores write destination pointer into memory
         │      (WRPNTDP)
         └─── Return from interrupt processing
```

- 1-byte write initialization processing

```
WR1_INI ─┬── Sets write protect area to area 3 (FC80H to
         │   FCFFH)
         │   Enables write protect
         │   Sets timer clock for write to 65536/fXX
         │   Enables write
         └── Return
```

- 1-byte write initialization processing

```
WR_1BYTE ─┬── (WHILE:  Previous writing has not been completed
          │            (EWST="1"))
          ├── 1-byte write
          └── Return
```

- Write error processing

```
INTEER ──── LOOP
```

(9) Program list

(a) Continuous write mode

In this program example, the definition file (INTMS.DEF) for the vector table address and macro service area is included in the beginning of the source program.

```
$       IC(INTMS.DEF)
        NAME    WRC
;*********************************************************
;*      WRITING TO EEPROM                                *
;*                  uPD78244                             *
;*              writing continuously mode                *
;*********************************************************

        PUBLIC  WRC_INI         ; package
        EXTRN   WRCDATA,WRCADRS ; data
        EXTRN   WRCNT           ; data

EPWIF   EQU     IF1L.1          ; INTEPW request flag
EPWMK   EQU     MK1L.1          ; INTEPW mask flag

;       *** work area for writing to EEPROM ***

WRWOKD  DSEG    SADDR
WRPNTSP:DS      2               ; sorce pointer store area
WRPNTDP:DS      2               ; destination pointer store area
CNTAREA:DS      1               ; writing count store area

;       *** vector table ***

        EPWVENT                 ; INTEPW vector table

;****************************************
;*      initialize of                   *
;*              writing continuously    *
;****************************************

WRCS    CSEG
WRC_INI:
        EWC=#00111000B          ; open write protect,
                                ; timer clock=65536/fxx, enable writing
        WRPNTSP=#WRCDATA        ; set sorce address of first writing
        WRPNTDP=#WRCADRS        ; set destination address of first writing
        CNTAREA=#WRCNT+1        ; set writing count
        SET1    EPWIF           ; set INTEPW request flag
        CLR1    EPWMK           ; open INTEPW mask
        RET

;****************************************
;*      writing continuously routine    *
;*              by INTEPW               *
;****************************************

INTEPW:
        SEL     RB1             ; set register bank 1
        CNTAREA--               ;       decrement writing counter
        if(CNTAREA>#0)          ; not end writing of specified byte ?
                                ; THEN
                HL=WRPNTSP (AX) ;       read write pointer
                DE=WRPNTDP (AX)
                [DE+]=[HL+] (A) ;       write 1 byte & increment write pointer
                WRPNTSP=HL (AX) ;       store sorce pointer
                WRPNTDP=DE (AX) ;       store destination pointer
        endif
        RETI

        END
```

(b) 1-byte write mode

In this program example, SFR bit name definition file (SFRBIT.DEF) is included.

```
        NAME    WR1
;***********************************************************
;*      WRITING TO EEPROM                                  *
;*                      uPD78244                           *
;*              writing 1byte mode                         *
;*                                                         *
;*      input condition                                    *
;*              A  register <-- write data                 *
;*              HL register <-- write address              *
;***********************************************************

$       IC(SFRBIT.DEF)
        PUBLIC  WR1_INI,WR_1BYTE        ; package

;****************************************
;*      initialize of                   *
;*              writing 1 byte          *
;****************************************

WR1S    CSEG
WR1_INI:
        EWC=#00001100B          ; enable write protect about AREA 3
                                ; timer clock=65536/fxx
                                ; enable writing, protect area=FC80H-FCFFH
        RET

;****************************************
;*      writing 1 byte routine          *
;****************************************

WR_1BYTE:
        while_bit(EWST)         ; wait writing data
        endw
        [HL]=A                  ; write 1 byte
        RET

        END
```

(c) Write error processing

In this program example, the definition file (INTMS.DEF) for the vector table address and macro service area is included in the beginning of the source program.

```
$       IC(INTMS.DEF)
        NAME    WRERR
;************************************************************
;*      WRITING TO EEPROM                                  *
;*                     uPD78244                            *
;*              ERROR of writing routine                   *
;************************************************************

;       *** define vector table ***

        EERVENT                 ; INTEER vector table

WRERRS  CSEG
INTEER:
        while(forever)          ; LOOP forever
        endw

        END
```

APPENDIX A NOTES ON 78K/II SERIES PROGRAMMING

A.1 Notes on Vector Interrupt Processing

78K/II series has the following information in its program status word (PSW) shown in Fig. A-1:

(a) Interrupt request enable flag (IE)
(b) Interrupt priority status flag (ISP)
(c) Register bank selector flags (RBS0, RBS1)

Fig. A-1 Program Status Word

	7	6	5	4	3	2	1	0	
PSW	IE	Z	RBS1	AC	RBS0	0	ISP	CY	(02H when RESET is input)

CY	Carry flag

ISP	Interrupt priority status flag
0	Disables vector interrupt with low priority
1	Enables vector interrupt, regardless of priority

RBS1	RBS0	Register bank selector flags
0	0	Selects register bank 0
0	1	Selects register bank 1
1	0	Selects register bank 2
1	1	Selects register bank 3

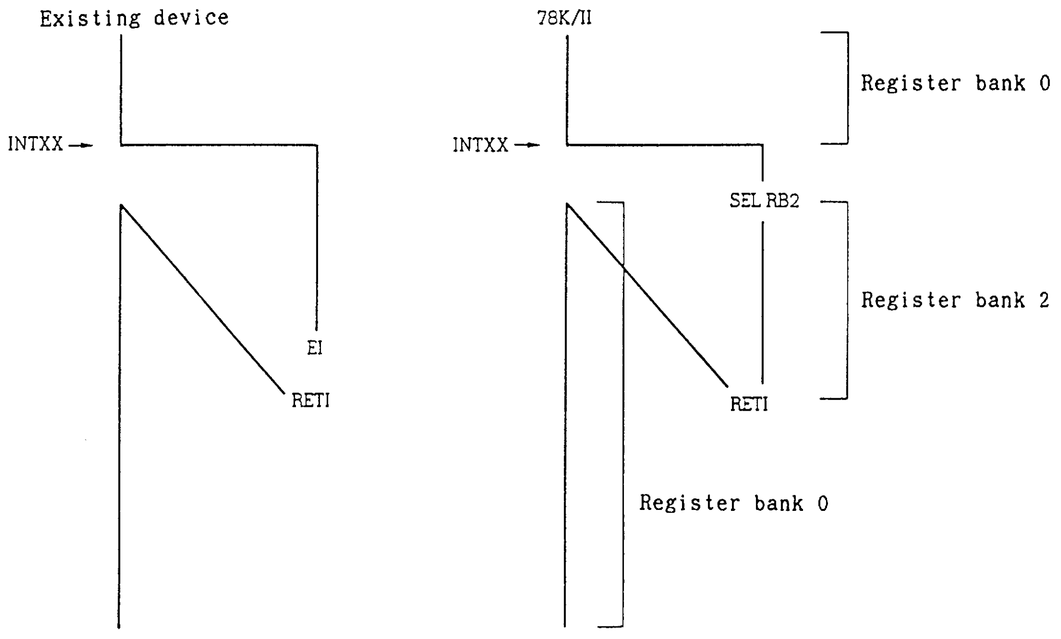
AC	Auxiliary carry flag

Z	Zero flag

IE	Interrupt request enable flag
0	Disables vector interrupt (DI state)
1	Enables vector interrupt (EI state)

When a vector interrupt is accepted, the program counter (PC) and PSW contents are saved to stack. At this time, the interrupt enable (IE) flag is cleared to 0, disabling the other vector interrupts. When a vector interrupt service program has been executed, the RETI instruction is executed to return execution to the main routine.
With the existing devices, the EI instruction is carried out before the RETI instruction, in order to enable the next vector interrupt.
With 78K/II series, however, the interrupts are enabled automatically, because the PC and PSW contents are restored from stack by the RETI instruction.
Even if a register bank has been switched to another bank, when a vector interrupt has been accepted, the original register bank is selected, when the RETI instruction is executed.

Fig. A-2 Relation between Vector Interrupt and PSW



The interrupt priority status (ISP) flag is cleared to 0 after the PSW contents are saved to stack, when a vector interrupt with a high priority (specified by PROL/H) occurs, disabling the interrupts with the lower priority. Normally, the vector interrupts are enabled, regardless of their priorities, when execution is returned to the original routine. However, even the vector interrupts with low priorities can also be enabled by setting the ISP flag to 1, even while a vector interrupt with a higher priority is being processed (the IE flag is 1 when the EI instruction is executed).

A.2 Notes on Accessing External Expansion Data Memory for uPD7821x and uPD7822x

In the uPD7822x, the value set in advance to the output latch for P6 (port 6) is output to the higher 4 bits of the address bus, when "&" is appended to the operand that indicates the memory address to be accessed.
In addition, the uPD7821x outputs the value set in the lower 4 bits for PM6 (port 6 mode register) by the addressing modes (direct addressing, register indirect addressing, indexed addressing, and base addressing) common to all the memory spaces, to the higher 4 bits for the address bus.

Note: When the $\overline{\text{RESET}}$ signal is cleared, PM6 is initialized to 0FxH and the lower 4 bits become undefined. It is therefore necessary to set bit 6 (MM6) of the MM register to 1 to execute initialization, such as clearing the lower 4 bits of the PM register to 0, before enabling access to the external expansion data memory.

A.3 Notes on Accessing Port 0 and Real-Time Output Port

When port 0 is used as a real-time output port, the contents of its output latch cannot be changed by directly accessing port 0. The output latch contents can be changed only by transferring data from the buffer register by hardware.

APPENDIX B RA78K/II

B.1 File Provided with RA78K/II

The file provided with the RA78/II is used in this application note. However, assemblers of version 3.00 or higher are provided with a text file that can be included during assembling, in addition to the assembler package program.
Files provided are a file (SFRBIT.DEF)* in which bit names for the special function registers (SFR) are probably used, when using the RA78K/II assembler package, and a file (INTMS.DEF), in which macros for assuring the interrupt vector table and the area are used for macro service.
These files are provided in the directory for each device named ¥DEVICE.

*: From version 4.00 or higher, this file is built in the assembler main unit, and it is not necessary to include it. Therefore, file SFRBIT.DEF is not provided either.

```
¥ ── DEVICE ─┬─ 78212    ─┬─ SFRBIT.DEF
             │            └─ INTMS.DEF
             ├─ 78213    ─┬─ SFRBIT.DEF
             │            └─ INTMS.DEF
             ├─ 78214    ─┬─ SFRBIT.DEF
             │            └─ INTMS.DEF
             ├─ 78217A* ──── INTMS.DEF
             │
             ├─ 78218A* ──── INTMS.DEF
             │
             ├─ 78220    ─┬─ SFRBIT.DEF
             │            └─ INTMS.DEF
             ├─ 78224    ─┬─ SFRBIT.DEF
             │            └─ INTMS.DEF
             ├─ 78233    ─┬─ SFRBIT.DEF
             │            └─ INTMS.DEF
             ├─ 78234    ─┬─ SFRBIT.DEF
             │            └─ INTMS.DEF
             ├─ 78237    ─┬─ SFRBIT.DEF
             │            └─ INTMS.DEF
             ├─ 78238    ─┬─ SFRBIT.DEF
             │            └─ INTMS.DEF
             ├─ 78243    ─┬─ SFRBIT.DEF
             │            └─ INTMS.DEF
             └─ 78244    ─┬─ SFRBIT.DEF
                          └─ INTMS.DEF
```

*: Version 4.00 or higher only

Refer to. the document (How to use the file, provided with RA78K/II assembler package (SUD-M-0319)) provided with the assembler package for details on how to use each file.

B.1.1 SFRBIT.DEF

In the SFRBIT.DEF file, bits of the special function registers probably manipulated in bit units are defined. Bit names are the same as those used in the users manual for each device.
This file is provided only for version 3.00. From version 4.00 or higher, bit names are treated as reserved words of the assembler main unit. When using version 4.00 or higher, SFRBIT.DEF does not need to be included. In addition, when using a source file created for version 3.00, delete control instructions which include SFRBIT.DEF.

B.1.2 INTMS.DEF

In the INTMS.DEF file, macros used to assure the memory area used for interrupt and macro service are defined. Areas are assured by referencing macros. By using this, interrupt and macro service can be used without knowing the absolute address.
Macros are segregated into three categories, depending on the contents to be defined.

(1) Vector table definition macro

Used to define vector table. Macro is defined for each table.

(2) Macro service control word area assuring macro availability.

Used to assure the macro service control word area availability and to define the label. Macro is defined for each interrupt (only interrupts that can be used by macro service).

(3) Macro service channel area assuring macro availability.

Used to assure macro service channel area availability and define the label. Macro is defined for each macro service type.

B.2 Register Bank Area Assuring Method

In the 78K/II series, general registers are mapped in the internal RAM. Generally for the assembly language, the DS pseudo-instruction is used to assure the data memory area availability. However, in the RA78K/II, if DS pseudo-instruction is used to assure the area availability, the availability of the area shared with the register bank may also be assured as the data memory area.
The following method is to assure the availability of the register bank area currently used, using the assembler package function without overlapping on the data memory area.

B.2.1 Preparation

Create files for lists 1 to 5. Of these files, SELRB.DEF is a source file which will be included. Files SELRB0.ASM, SELRB1.ASM, SELRB2.ASM, and SELRB3.ASM will be assembled and used as one library file by the librarian when linking.

- List 1 SELRB.DEF

```
$        NOLIST
         EXTRN    DSRB0
SELDRB1  SET      0FFFFH
SELDRB2  SET      0FFFFH
SELDRB3  SET      0FFFFH
;
SELRB0   MACRO
$        NOGEN
         SEL      RB0
$        GEN
         ENDM
;
SELRB1   MACRO
$        NOGEN
$        _IF      SELDRB1 EQ 0FFFFH
         EXTRN    DSRB1
SELDRB1  SET      0
$        ENDIF
         SEL      RB1
$        GEN
         ENDM
;
SELRB2   MACRO
$        NOGEN
$        _IF      SELDRB2
         EXTRN    DSRB2
SELDRB2  SET      0
$        ENDIF
         SEL      RB2
$        GEN
         ENDM
;
SELRB3   MACRO
$        NOGEN
$        _IF      SELDRB3
         EXTRN    DSRB3
SELDRB3  SET      0
$        ENDIF
         SEL      RB3
$        GEN
         ENDM
$        LIST
```

- List 2 SELRB0.ASM

```
        NAME    DSRB
        PUBLIC  DSRB0
;
SEGRB0  DSEG    AT 0FEF8H
DSRB0:  DS      8
;
        END
```

- List 3 SELRB1.ASM

```
        NAME    DSRB
        PUBLIC  DSRB1
;
SEGRB1  DSEG    AT 0FEF0H
DSRB1:  DS      8
;
        END
```

- List 4 SELRB2.ASM

```
        NAME    DSRB
        PUBLIC  DSRB2
;
SEGRB2  DSEG    AT 0FEE8H
DSRB2:  DS      8
;
        END
```

- List 5 SELRB3.ASM

```
        NAME    DSRB
        PUBLIC  DSRB3
;
SEGRB3  DSEG    AT 0FEF0H
DSRB3:  DS      8
;
        END
```

The following lists are batch files. With these batch files, the library file (SELRB.LIB) can be created by copying SELRB.DEF into the same directory introduced in B.1.

- Batch file main unit MKLIBALL.BAT

```
FOR %%D IN(212 213 214 217A 218A 220 224 233 234 237 238 243 244)
DO COMMAND/C MKLIB %%D %1
```

- Batch file called by batch file MKLIB.BAT

```
  FOR %%F IN (0 1 2 3) DO RA78K2 DSRB%%F.ASM -C%1 -NP
  LB78K2 < MKLIB.CMD
  COPY DSRB.LIB %2¥DEVICE¥78%1
  COPY DSRB.DEF %2¥DEVICE¥78%1
  DEL DSRB?.REL
  DEL DSRB.LIB
```

- Command file input to librarian MKLIB.CMD

```
  C DSRB.LIB
  A DSRB.LIB DSRB0.REL
  A DSRB.LIB DSRB1.REL
  A DSRB.LIB DSRB2.REL
  A DSRB.LIB DSRB3.REL
  E
```

Actual work is initiated by making an input on the command line, as follows.

MKLIBALL   Directory under which assembler package is located

B.2.2 How to use

(1) Source file description method

Specify to include SELRB.DEF in the beginning of the source program module body.
In addition, write macro name SELRBn, instead of writing the SEL RBn instruction, which selects a register bank.

Example:

```
$       TITLE ("TEST")
$       IC (INTMS.DEF)
$       IC (SELRB.DEF)
;
        NAME TEST
;
$       IC (SFRBIT.DEF)

                 .
                 .
                 .

SELRB2  ; Selects register bank 2
```

In the above example, the path to specify reading the include file needs to be specified on the command line, when assembling, or the path to read the include file needs to be specified, using the environment variable for MS-DOS.

Example 1: Specifying on command line

```
RA78K2 TEST -IA:¥DEVICE¥78214 -C214
```

2: Environment variable setting method

```
SET INC78K2 = A:¥DEVICE¥78214
```

(2) Specifying when linking

Specify specification to use the library file (SELRB.LIB) when linking. Library file specification should be specified on the command line or by the parameter file.

Example: `LK78K2 TEST -BA:¥DEVICE¥78214¥SELRB.LIB`

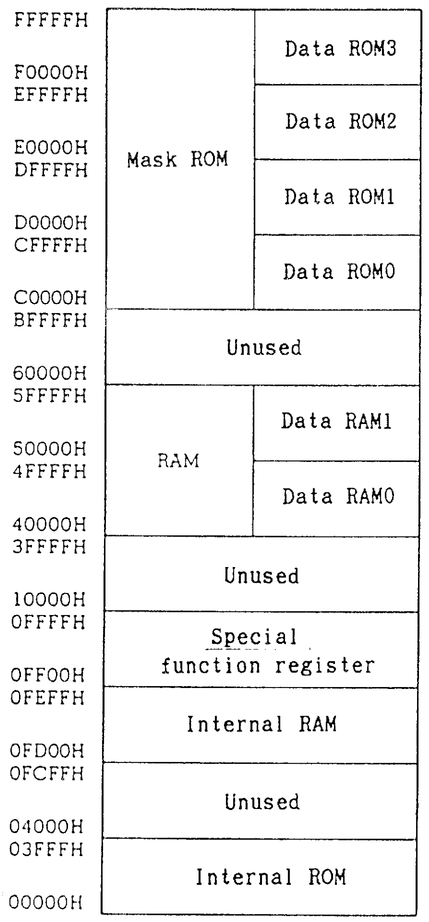
B.3 How to Use 1M-Byte Extension Data Memory Space

In the 78K/II series the data memory space can be extended to 1M bytes. The following example shows how to use the extended data memory space.

(1) Memory space use example

The memory space is used as shown in Fig. B-1. In the 78K/II series, 64K bytes of the memory space are used as one bank. Therefore, if the memory is larger than 64K bytes, the memory is divided into two or more banks in the 64K-byte unit.

Fig. B-1 Memory Space Use Example



(2) How to write on source program

On the source program, the availability of an area for data allocated in the extended data memory space must always be assured in a segment having a name.
In this example, the segment names are determined as indicated in Table B-1.

Table B-1 Segment Name and Allocation Location

Segment name	Allocation location
LINEBUFF	Data RAM0
SUBBUFF	
MAINBUFF	Data RAM1
SYSPARA	Data ROM0
KDATA	Data ROM1
RDATA	
FONT1	Data ROM2
FONT2	Data ROM3

The actual description is as shown below.

```
        LINEBUFF  DSEG

        LL0:      DS 1024
        LM0:      DS 1024

                      .
                      .
                      .

        SUBBUF    DSEG
        SB:       DS  512

                      .
                      .
                      .

        SYSPARA   CSEG
        PARA1:    DB  50H, 43H, 7FH

                      .
                      .
                      .
```

When referencing, specify the memory bank using the PM6 or P6 register in advance.

Example:

```
        MOVW      HL, #LM0
        MOV       P6, #4
        MOV       &[HL], A

                  .
                  .
                  .

        MOV       P6, #12    ; 0CH
        MOV       A, &!PARA1
```

(3) How to specify when linking

The linker resolves the allocation address by the directive file. The directive file defines the memory area, and specifies segment allocation to the memory area.
The memory area is defined by assigning names to a part or all of the memory banks*. Individual memory banks can contain two or more areas. Segments will be allocated in the memory area defined here.

*: In the manual for the RA78K series, this is expressed as "memory space", due to the relationship with other functions.

The memory areas are defined as indicated in Table B-2.

Table B-2 Memory Area Definition

Memory area name	Allocation location
ERAM0	Data RAM0
ERAM1	Data RAM1
EROM0	Data ROM0
EROM1_0	0000H-7FFFH for Data ROM1*
EROM1_1	8000H-FFFFH for Data ROM1*
EROM2	Data ROM2
EROM3	Data ROM3

*: These are addresses in the bank. In an absolute address, these addresses are D0000H to D7FFFH, and D8000H to DFFFFH.

Segments are allocated as indicated in Table B-3.

Table B-3 Segment Allocation

Segment name	Memory area to be allocated
LINEBUFF	ERAM0
SUBBUFF	ERAM0
MAINBUFF	ERAM1
SYSPARA	EROM0
KDATA	EROM1_0
RDATA	EROM1_1
FONT1	EROM2
FONT2	EROM3

In this case, the directive file becomes as follows:

```
MEMORY    ERAM0         : (0,0FFFFH)/EX4
MEMORY    ERAM1         : (0,0FFFFH)/EX5
MEMORY    EROM0         : (0,0FFFFH)/EX12
MEMORY    EROM1_0       : (0,80000H)/EX13
MEMORY    EROM1_1       : (8000H,8000H)/EX13
MEMORY    EROM2         : (0,0FFFFH)/EX14
MEMORY    EROM3         : (0,0FFFFH)/EX15
;
MERGE     LINEBUFF      : =ERAM0/EX4
MERGE     SUBBUFF       : =ERAM0/EX4
MERGE     MAINBUFF      : =ERAM1/EX5
MERGE     SYSPARA       : =EROM0/EX12
MERGE     KDATA         : =EROM1_0/EX13
MERGE     PDATA         : =EROM1_1/EX13
MERGE     FONT1         : =EROM2/EX14
MERGE     FONT2         : =EROM3/EX14
```

Specify the directory file using the -D option on the command line or in the parameter file, when executing linking.

APPENDIX C LIST OF 78K/II SERIES PRODUCTS

Appendix C lists the products in the 78K/II series.

Table C-1 List of 78K/II Series Products

(1/5)

Series name		uPD78214 series			uPD78218A series		uPD78224 series		uPD78234 series				uPD78244 series	
Item / Product name		78212	78213	78214 (78P214)	78217A	78218A (78P218A)	78220	78224 (78P224)	78233	78234	78237	78238 (78P238)	78243	78244
Number of basic instructions		65 (all 78K/II series)												
Minimum instruction execution time (ns)		333	500	333	500	333	500	333	500	333	500	333	500	333
PUSH PSW instruction execution time (Number of clocks)		When the stack area is in the internal dual port RAM: 5 or 7 Other than the above: 7 or 9			When the stack area is in the internal dual port RAM : 6 Other than the above: 8		When the stack area is in the internal dual port RAM : 5 or 7 Other than the above: 7 or 9		When the stack area is in the internal dual port RAM: 6 Other than the above : 8					
Operating temperature range and power requirements		Other than uPD78P218A: -40 to +85°C, V_{DD} = +5V $\pm$10% uPD78P218A: -40 to +85°C, V_{DD} = +5V $\pm$0.3V					-40 to +85°C, V_{DD} = +5 V$\pm$5% -10 to +70°C, V_{DD} = +5 V$\pm$10%		-40 to +85°C, V_{DD} = +5 V$\pm$10%				-10 to +70°C, V_{DD} = +5 V$\pm$10%	
General-purpose register		8 bits x 8 x 4 banks												
Bank register		P6 and PM6					P6 only		P6 and PM6					
Internal memory (byte)	ROM	8K	None	16K	None	32K	None	16K	None	16K	None	32K (32K/ 16K*)	None	16K
	EEPROM	None											512	
	RAM	384	512		1024		640				1024	1024 (1024/ 640*)	512	
Memory space		Program memory space: 64K bytes, data memory space: 1M bytes												

*: Set through software

(2/5)

Series name		uPD78214 series			uPD78218A series		uPD78224 series		uPD78234 series				uPD78244 series	
Item	Product name	78212	78213	78214 (78P214)	78217A	78218A (78P218A)	78220	78224 (78P224)	78233	78234	78237	78238 (78P238)	78243	78244
I/O pins	Input	14					8		16				14	
	Output	12					12	20	12					
	I/O	28	10	28	10	28	25	35	18	36	18	36	10	28
	Total	54	36	54	36	54	45	63	46	64	46	64	36	54
* Pins with auxiliary functions	w/pull-up resistor	34	16	34	16	34	None		24	42	24	42	16	34
	LED direct drive output	16	0	16	0	16	8			24	8	24	0	16
	Transistor direct drive output	8					None		8					
I/O pins	P0	8-bit output port												
	P1	-					8-bit I/O port						-	
	P2	8-bit input port												
	P3	8-bit I/O port												
	P4	8-bit I/O port	-	8-bit I/O port	-	8-bit I/O port	-	8-bit I/O port	-	8-bit I/O port	-	8-bit I/O port	-	8-bit I/O port
	P5	8-bit I/O port	-	8-bit I/O port	-	8-bit I/O port	-	8-bit Output port	-	8-bit I/O port	-	8-bit I/O port	-	8-bit I/O port
	P6	4-bit output port or 4-bit I/O port	4-bit output port or 2-bit I/O port	4-bit output port or 4-bit I/O port	4-bit output port or 2-bit I/O port	4-bit output port or 4-bit I/O port	4-bit output port or 2-bit I/O port	4-bit output port or 4-bit I/O port	4-bit output port or 2-bit I/O port	4-bit output port or 4-bit I/O port	4-bit output port or 2-bit I/O port	4-bit output port or 4-bit I/O port	4-bit output port or 2-bit I/O port	4-bit output port or 4-bit I/O port
	P7	6-bit input port					7-bit I/O port		8-bit input port				6-bit input port	

*: The pins with auxiliary functions are included in the I/O pins.

Series name		uPD78214 series			uPD78218A series		uPD78224 series		uPD78234 series				uPD78244 series	
Item \ Product name		78212	78213	78214 (78P214)	78217A	78218A (78P218A)	78220	78224 (78P224)	78233	78234	78237	78238 (78P238)	78243	78244
PWM output		None							12 bits x 2				None	
Comparator		None					4 bits x 8		None					
A/D converter		8 bits x 8					None		8 bits x 8					
	Selection of conversion time	Selected according to operating frequency					-		Any time can be selected				Selected according to operating frequency	
	AV_{REF} input voltage range	3.4 V to V_{DD}			3.6 V to V_{DD}		-		3.4 V to V_{DD}				3.6 V to V_{DD}	
	Limitation of input voltage	Normally 0 V to AV_{REF} pin voltage for only pins selected by ANI0-ANI2 bits of ADM register			0 V to AV_{REF} for pins used for A/D conversion during conversion		-		0 V to AV_{REF} for pins used for A/D conversion during conversion					
D/A converter		None							8 bits x 2				None	
Timer counter	16-bit timer/counter	1												
	8-bit timer/counter	3					2		3					
	Timer output	4												
	PWM/PPG output	Provided					None		Provided					
	One-shot pulse	None			Provided		None		Provided					
	Interrupt source	7					5		7					
External SFR area		16 bytes of 0FFD0H-0FFDFH					None		16 bytes of 0FFD0H-0FFDFH					

Series name		uPD78214 series			uPD78218A series		uPD78224 series		uPD78234 series				uPD78244 series	
Item \ Product name		78212	78213	78214 (78P214)	78217A	78218A (78P218A)	78220	78224 (78P224)	78233	78234	78237	78238 (78P238)	78243	78244
Serial interface	UART	1 channel												
	CSI	1 channel (for SBI)												
	BRG timer	Provided (shared with timer/counter 3)					Provided		Provided (shared with timer/counter 3)					
	Dedicated baud rate generator	Provided					None		Provided					
	External baud rate clock input	Provided					None		Provided					
Real-time output port		4 bits x 2 or 8 bits x 1												
Interrupt		2 levels (programmable), vector/macro service												
	External	7					8		7					
	Internal	12					9		12				14	
	Number of macro services that can be used	15					6		15					
	Number of bits for macro service counter	8 bits only			8/16 bits selectable (except type A)		8 bits only		8/16 bits selectable (except type A)					
	Incrementing MPD, MPT of macro service type C	Only lower 8 bits are incremented (higher bits unchanged)			Incremented in 16-bit units		Only lower 8 bits are incremented (higher bits unchanged)		Incremented in 16-bit units					
	Macro service execution time	Macro service execution times for uPD78214 series and uPD78224 are the same. Macro service execution times for uPD78218A series, uPD78234 series, and uPD78244 series are the same. Execution times differ, depending on modes. Compare their execution times, referring to the user's manuals.												

(5/5)

Series name	uPD78214 series			uPD78218A series		uPD78224 series		uPD78234 series				uPD78244 series	
Item / Product name	78212	78213	78214 (78P214)	78217A	78218A (78P218A)	78220	78224 (78P224)	78233	78234	78237	78238 (78P238)	78243	78244
Limitation of data transfer from memory to SFR by macro service type A	Occurs when transfer data is DOH-DFH			Occurs when address of transfer source buffer(memory) is OFEDOH-OFEDFH		Occurs when transfer data is DOH-DFH						Occurs when address of transfer source buffer is OFEDOH-OFEDFH	
Standby function	HALT/STOP mode												
Oscillation stabilization time on releasing STOP mode	Fixed			Two types of time selectable		Fixed		Two types of time selectable					
Pseudo SRAM refresh function	Provided (refresh pulse width: $1/f_{CLK}$)					Provided (refresh pulse width: $1.5/f_{CLK}$)		Provided (refresh pulse width: $1/f_{CLK}$)					
Limitation of memory access	None					FC80H-FDFFH cannot be accessed when refresh		None					
Setting of ROM-less mode	$\overline{EA}$ = low	-	$\overline{EA}$ = low	-	$\overline{EA}$ = low	-	$\overline{EA}$ = low	-	MODE = high	-	MODE = high (inhibited)	-	$\overline{EA}$ = low
Package	o 64-pin plastic shrink DIP (750 mil) o 68-pin plastic QFJ: except uPD78212 o 64-pin plastic QFP (chip: 14 x 14 mm) o 74-pin plastic QFP (chip: 20 x 20 mm) o 64-pin plastic QUIP: except uPD78212 o 64-pin ceramic shrink DIP w/window: uPD78P214 only			o 64-pin plastic shrink DIP (750 mil) o 64-pin plastic QFP (chip: 14 x 14 mm) o 64-pin ceramic shrink DIP w/window: uPD78P218A only		o 84-pin plastic QFJ o 94-pin plastic QFP (chip: 20 x 20 mm)		o 84-pin plastic QFJ o 80-pin plastic QFP (chip: 14 x 14 mm) o 94-pin plastic QFP (chip: 20 x 20 mm) o 94-pin ceramic WQFN uPD78P238 only				o 64-pin plastic shrink DIP (750 mil) o 64-pin plastic QFP (chip: 14 x 14 mm)	

APPENDIX D　CORRESPONDENCE FOR PROGRAMS, DEVICES AND INTERNAL PERIPHERAL HARDWARE

Item	Target device					Internal peripheral hardware															
						Timer/counter				Asynchronous serial interface	3-line serial interface	SBI	Real-time output port	A/D converter	Port T	PWM	EEPROM	Interrupt			
																		Macro service			Vector interrupt
	uPD78214	uPD78218A	uPD78224	uPD78234	uPD78244	16-bit timer/ counter	8-bit timer/ counter 1	8-bit timer/ counter 2	8-bit timer/ counter 3									Type A	Type B	Type C	
CHAPTER 3 TIMER/COUNTER PROGRAM EXAMPLE 3.1 Internal interval timer (1) Program examplwe with 16-bit timer/counter	○	○	○	○	○	○															○
(2) program example with 8-bit timer/counter 2	○	○	○	○	○		△	○	△												○
3.2 Programmable rectangular pulse output	○	○	○	○	○	○															○
3.3 Free-running interval timer (1) Program example using the 16-bit timer/ counter	○	○	○	○	○	○															○
(2) Program example using 8-bit timer/counter 2	○	○	○	○	○		△	○													○
3.4 PWM/PPG output (uPD78214) (1) PWM output program example using 16-bit timer/counter	○	○		○	○	○															○
(2) PWM output program example using 8-bit timer/counter 2	○	○		○	○																○
(3) PPG output programexample using 16-bit timer/counter	○	○		○	○																○
(4) PPG output program example using 8-bit timer/counter 2	○	○		○	○			○													○
3.5 Software triggered one-shot pulse output		○		○	○																
3.6 Pulse cycle measurement	○	○	○	○	○	○	△	△													○
CHAPTER 4 PWM OUTPUT UNIT PROGRAM EXAMPLE (uPD78234)				○												○					
CHAPTER 5 ASYNCHRONOUS SERIAL INTERFACE PROGRAM EXAMPLES (a) Program for uPD78214	○	○		○	○					○											
(b) Program for uPD78224	○	○	○	○	○					○											
CHAPTER 6 THREE-LINE SERIAL INTERFACE PROGRAM EXAMPLE	○	○	○	○	○						○										
CHAPTER 7 INTERRUPT PROCESSING PROGRAM EXAMPLES 7.1 UART reception processing	○	○	○	○	○					○								○	△		○
7.2 Parallel data input in synchronization with external interrupt request	○	○	○	○	○														○		○
7.3 Open-loop control for stepping motor (1)	○	○	○	○	○		○													○	
7.4 Open-loop control for stepping motor (2)		○		○	○		○						○					△	○	○	
CHAPTER 8 A/D CONVERTER PROGRAM EXAMPLES	○	○		○	○								○	○							
CHAPTER 9 COMPARATOR PROGRAM EXAMPLE			○												○						
CHAPTER 10 EEPROM PROGRAM EXAMPLE (uPD78244)					○												○				