To our customers,

Old Company Name in Catalogs and Other Documents

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

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

April 1st, 2010
Renesas Electronics Corporation

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

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

Notice

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

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

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

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

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

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

RENESAS

APPLICATION NOTE

μPD750008 SUBSERIES

4-BIT SINGLE-CHIP MICROCONTROLLER

μPD750004
μPD750006
μPD750008
μPD75P0016

Document No. U10452EJ1V0AN00 (1st edition)
Date Published February 1996 P
Printed in Japan

The export of this product from Japan is regulated by the Japanese government. To export this product may be prohibited without governmental license, the need for which must be judged by the customer. The export or re-export of this product from a country other than Japan may also be prohibited without a license from that country. Please call an NEC sales representative.

The application circuits and their parameters are for references only and are not intended for use in actual design-in's.

INTRODUCTION

Intended Readers This application note is intended for engineers who understand the functions of μPD750008 subseries and who design an application system using any of these microcontrollers.
The μPD750008 subseries is a generic name that stands for the μPD750004, 750006, 750008, and 75P0016.

Purpose The main purpose of this manual is to help you understand the μPD750008 subseries hardware functions.

How to Read This application note is prepared on the assumption that its readers have general knowledge regarding electricity, logic circuitry, and microcontroller.
The examples in this application note refer only to the μPD750008, if there are no differences from a functional standpoint with the other microcontrollers, μPD750008 can be read as μPD750004, μPD750006 or μPD75P0016 where appropriate.

Legend

Data significance	:	Left-hand digits are higher and right-hand digits are lower
Active low	:	$\overline{xxx}$ (a bar over pin or signal name)
Note	:	A point to be noted
Caution	:	Information requiring your attention
Remark	:	Supplementary information
Number representation	:	Binary ..... xxxx or xxxxB
		Decimal xxxx
		Hexadecimal ..... xxxxH

Related Documents

Documents related to devices

Document / Product	Brochure	Data Sheet	User's Manual	Instruction Application Table	Application Note
μPD750004	—	IC-3647	IEU-1421	—	U10452E (this manual)
μPD750006					
μPD750008					
μPD75P0016		U10328E			

[MEMO]

CONTENTS

LIST OF FIGURES (1/2)

LIST OF FIGURES (2/2)

LIST OF TABLES

[MEMO]

CHAPTER 1 GENERAL DESCRIPTION

The μPD750008 subseries comprises 75XL series 4-bit single-chip microcontrollers, successors of the 75X series products, boasting a comprehensive product line-up.

The differences among the μPD750008 subseries products are shown in Table 1-1.

Table 1-1. Differences between μPD750008 Subseries Products

Product Name	Program Memory (ROM)	ROM Configuration	Program Counter
μPD750004	4096 bytes	Mask ROM	12 bits
μPD750006	6144 bytes		13 bits
μPD750008	8192 bytes		
μPD75P0016	16384 bytes	One-time PROM	14 bits

1.1 Differences between μPD75008 and μPD750008

The μPD750008 inherits the functions and instructions of the previous μPD75008 (75X series), which facilitates replacement between the old and new products.

Table 1-2 shows the differences between the μPD75008 and μPD750008.

Table 1-2. Differences between μPD75008 and μPD750008 (1/2)

Item		μPD75008	μPD750008
Program memory		0000H - 1F7FH 8064 × 8 bits	0000H - 1FFFH 8192 × 8 bits
Data memory		000H to 1FFH (512 × 4 bits)	
CPU		75X Standard	75XL
Oscillation stabilization wait time		Fixed at 31.3 ms	$2^{15}/f_X$, $2^{17}/f_X$[Note] (selectable by mask option)
Instruction execution time	When main system clock is selected	0.95, 1.91, 15.3 μs (4.19 MHz operation only)	• 0.95, 1.91, 3.81, 15.3 μs (in 4.19 MHz operation) • 0.67, 1.33, 2.67, 10.7 μs (in 6.0 MHz operation)
	When subsystem clock is selected	122 μs (in 32.768 kHz operation)	
Stack	SBS register	No	Yes • SBS.3 = 1: Mk I mode selected • SBS.3 = 0: Mk II mode selected
	Stack area	000H - 0FFH	n00H - nFFH (n = 0, 1)
	Stack operation of subroutine call instruction	2-byte stack	In Mk I mode : 2-byte stack In Mk II mode: 3-byte stack
Instruction	BRA !addr1 operation CALLA !addr1 operation	Unusable	In Mk I mode : Unusable In Mk II mode: Usable

	MOVT XA, @BCDE
	MOVT XA, @BCXA
	BR BCDE
	BR BCXA
			Usable

	CALL !addr	3 machine cycles	In Mk I mode : 3 machine cycles In Mk II mode: 4 machine cycles
	CALLF !faddr	2 machine cycles	In Mk I mode : 2 machine cycles In Mk II mode: 3 machine cycles
Timer		3 channels • Basic interval timer • Timer/event counter • Watch timer	4 channels • Basic interval timer/watchdog timer • Timer/event counter • Timer counter • Watch timer
Clock output (PCL)		• Φ, 524, 262, 65.5 kHz (in 4.19 MHz operation)	• Φ, 524, 262, 65.5 kHz (in 4.19 MHz operation) • Φ, 750, 375, 93.7 kHz (in 6.0 MHz operation)

Note $2^{15}/f_X$: 5.46 ms at 6.0 MHz, 7.81 ms at 4.19 MHz
$2^{17}/f_X$: 21.8 ms at 6.0 MHz, 31.3 ms at 4.19 MHz

Table 1-2. Differences between μPD75008 and μPD750008 (2/2)

Item		μPD75008	μPD750008
Buzzer output (BUZ)		2 kHz	• 2.4, 32 kHz (in 4.19 MHz operation) • 2.86, 5.72, 45.8 kHz (in 6.0 MHz operation)
Serial interface		Compatible with 3 types of mode • 3-wire serial I/O mode ··· MSB first/LSB first can be switched. • 2-wire serial I/O mode • SBI mode	
SOS register	Feedback resistor cut flag (SOS.0)	Feedback resistor can be incorporated using mask option.	Incorporated
	Sub-oscillator current cut flag (SOS.1)	No	
Register bank selection register (RBS)		No	Yes
Standby release by INT0		Not possible	Possible
Vectored interrupt		External: 3 Internal: 3	External: 3 Internal: 4
Processor clock control register (PCC)		PCC = 0, 2, 3 can be used.	PCC = 0 to 3 can be used.
Supply voltage		V_{DD} = 2.7 to 6.0 V	V_{DD} = 2.2 to 5.5 V
Package		• 42-pin plastic shrink DIP (600 mil) • 44-pin plastic QFP (10 × 10 mm)	

1.2 Switching between Mk I Mode and Mk II Mode

1.2.1 Using Mk I mode and Mk II mode

The CPU of the μPD750008 subseries has two modes, Mk I mode and Mk II mode, and can select either of the two. The mode can be switched by bit 3 of the stack bank selection register (SBS).

Table 1-3 shows the differences between Mk I mode and Mk II mode, and Figure 1-1 shows the format of the stack bank selection register.

- Mk I mode: Has upward compatibility with the μPD75008 subseries.
  Can be used with a 75XL CPU which has a ROM capacity of up to 16K bytes.
- Mk II mode: Does not have upward compatibility with the μPD75008 subseries.
  Can be used with all 75XL CPUs including products which have ROM capacity of 16K bytes or more.

Table 1-3. Differences between Mk I Mode and Mk II Mode

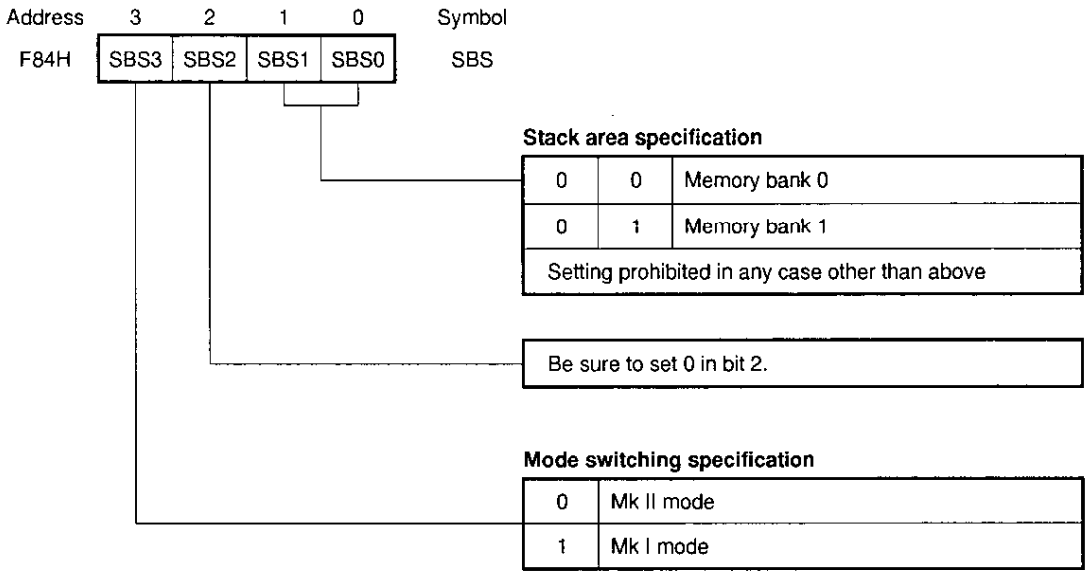
	Mk I Mode	Mk II Mode
Number of stack bytes of subroutine instruction	2 bytes	3 bytes
BRA !addr1 instruction CALLA !addr1 instruction	Undefined operation	Normal operation
CALL !addr instruction	3 machine cycles	4 machine cycles
CALLF !faddr instruction	2 machine cycles	3 machine cycles

Remark The Mk II mode is used to maintain software compatibility with those 75X series or 75XL series which have a program memory of 24K bytes or more. Therefore, when ROM efficiency or speed needs to be given priority, use the Mk I mode.

The stack bank selection register is set by a 4-bit memory manipulation instruction.

When the Mk I mode is used, be sure to initialize the stack bank selection register to 10××B[Note]. When the Mk II mode is used, be sure to initialize it to 00××B[Note].

Figure 1-1. Format of Stack Bank Selection Register



Note Set a desired value in "××."

Caution SBS.3 becomes "1" after the generation of a $\overline{\text{RESET}}$ signal, and therefore the CPU operates in Mk I mode. When using an instruction in Mk II mode, set SBS.3 to "0" and set Mk II mode before using the instruction.

1.2.2 Using register bank

The μPD750008 subseries incorporates 4 register banks, with each bank consisting of 8 general registers X, A, B, C, D, E, H and L. This general register area is mapped at addresses 00H to 1FH of memory bank 0 of the data memory (refer to **Figure 1-3**). A register bank enable flag (RBE) and register bank selection register (RBS) are incorporated in order to specify these general register banks. The RBS register is used to select the register banks and the RBE flag is used to determine whether the register bank selected by RBS should be enabled or disabled. The register banks (RB) enabled in execution of an instruction are as follows.

RB = RBE·RBS

Table 1-4. RBE and RBS, and Register Banks Selected

RBE	RBS				Register Bank
	3	2	1	0	
0	0	0	×	×	Fixed at bank 0
1	0	0	0	0	Bank 0 selected
			0	1	Bank 1 selected
			1	0	Bank 2 selected
			1	1	Bank 3 selected

Fixed at 0

Remark ×: don't care

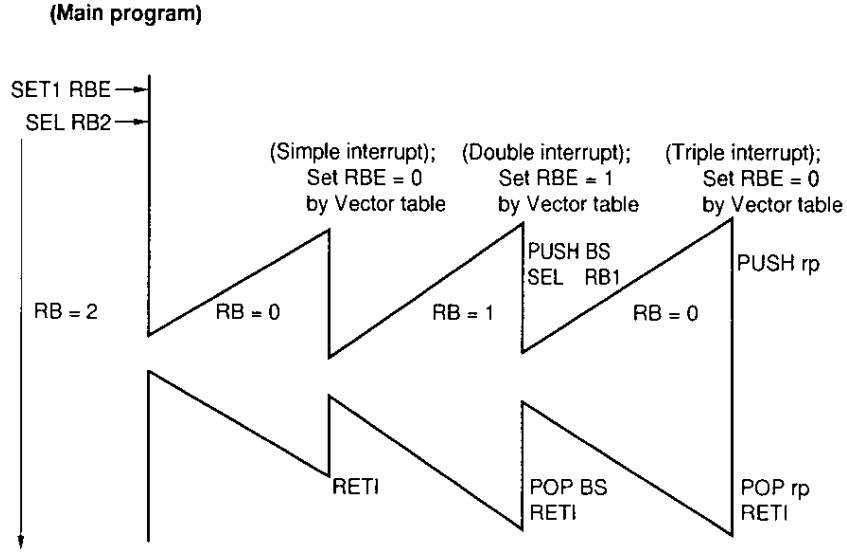
RBE is automatically saved/restored during subroutine processing and so it can freely be set during subroutine processing. Moreover, when an interrupt is serviced, RBE can automatically be saved/restored, and RBE can be set during interrupt servicing by setting an interrupt vector table concurrently with the start of interrupt servicing.

Therefore, as shown in Table 1-5, it is possible to speed up interrupt servicing by using different register banks for normal processing and interrupt servicing so that no save/restore of a general register is required in a simple interrupt, and only RBS save/restore is required in double interrupts.

Table 1-5. Example of Use of Different Register Banks for Normal Routines and Interrupt Routines

Normal Processing	Set RBE = 1 and use register banks 2 and 3.
Simple interrupt servicing	Set RBE = 0 and use register bank 0.
Double interrupt servicing	Set RBE = 1 and use register bank 1. (In this case, RBS save/restore is required.)
Triple or superior interrupt servicing	Save register by PUSH and POP.

Figure 1-2. Example of Use of Different Register Banks



When RBS is changed by subroutine processing or interrupt servicing, it should be saved/restored by a PUSH/POP instruction.

RBE is set by the SET1/CLR1 instruction. RBS is set by the SEL instruction.

```
Example  SET1  RBE  ; RBE ← 1
         CLR1  RBE  ; RBE ← 0
         SEL   RB0  ; RBS ← 0
         SEL   RB3  ; RBS ← 3
```

The general register area incorporated in the μPD750008 can be used not only as a 4-bit register but also as an 8-bit register by a register pair, enabling programming centered on general registers to be performed by transfer, operation, compare, increment/decrement instructions comparable to those of an 8-bit microcontroller.

(1) Using the general register area as a 4-bit register

When using the general register area as a 4-bit register, a total of eight general registers, X, A, B, C, D, E, H and L of the register banks specified by RB = RBE·RBS can be used as shown in Figure 1-3. Of these, the A register plays a central role as a 4-bit accumulator in transfer, operation and comparison of 4-bit data. The other general registers can perform accumulator transfer, comparison and increment/decrement.

(2) Using the general register area as an 8-bit register

When using the general register area as an 8-bit register, a total of eight 8-bit registers can be used as shown in Figure 1-4 by designating register pairs of the register banks specified by RB = RBE·RBS as XA, BC, DE and HL, and register pairs of the register banks whose bank (RB) bit 0 is inverted as XA', BC', DE' and HL'. Of these, the XA register pair plays a central role in transfer, operation, comparison, etc. of 8-bit data as an 8-bit accumulator. The other register pairs can perform accumulator transfer, operation, comparison and increment/decrement. Furthermore, the HL register pair functions mainly as a data pointer. The register pairs DE and DL also function as auxiliary data pointers.

Examples 1.
```
INCS    HL          ; HL←HL + 1, skips with HL = 00H
ADDS    XA, BC      ; XA←XA + BC, skips with a carry
SUBC    DE', XA     ; DE'←DE' – XA – CY
MOV     XA, XA'     ; XA←XA'
MOVT    XA, @PCDE   ; XA←(PC12-8 + DE) ROM, table reference
SKE     XA, BC      ; Skips if XA = BC.
```

2. Perform a test to determine whether the value of the count register (T0) of timer/event counter 0 is greater than the value of the BC' register pair and wait until it becomes greater.

```
      CLR1    MBE      ;
NO:   MOV     XA, T0   ; Reading of count register
      SUBS    XA, BC'  ; XA ≥ BC?
      BR      YES      ; YES
      BR      NO       ; NO
```

Figure 1-3. Configuration of General Registers (In Case of 4-Bit Processing)

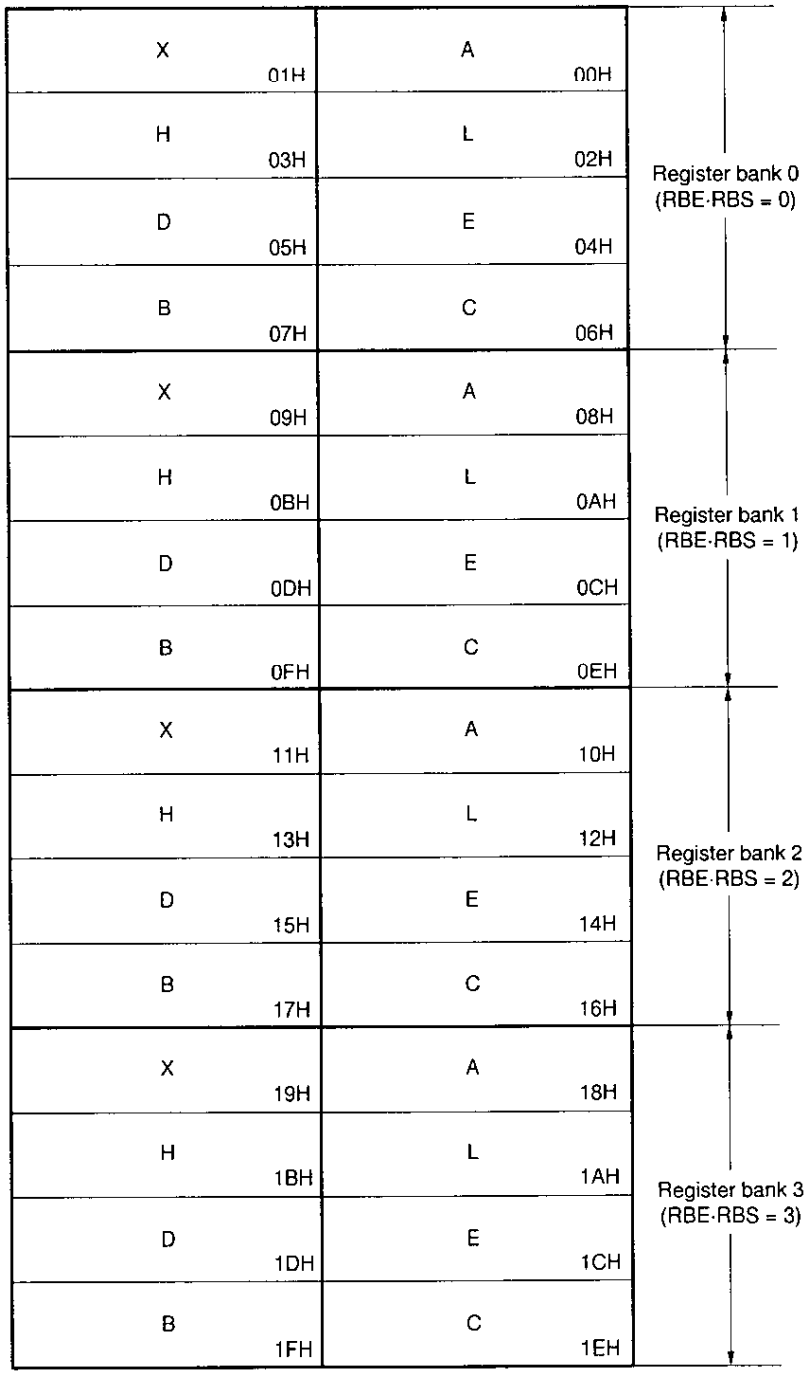
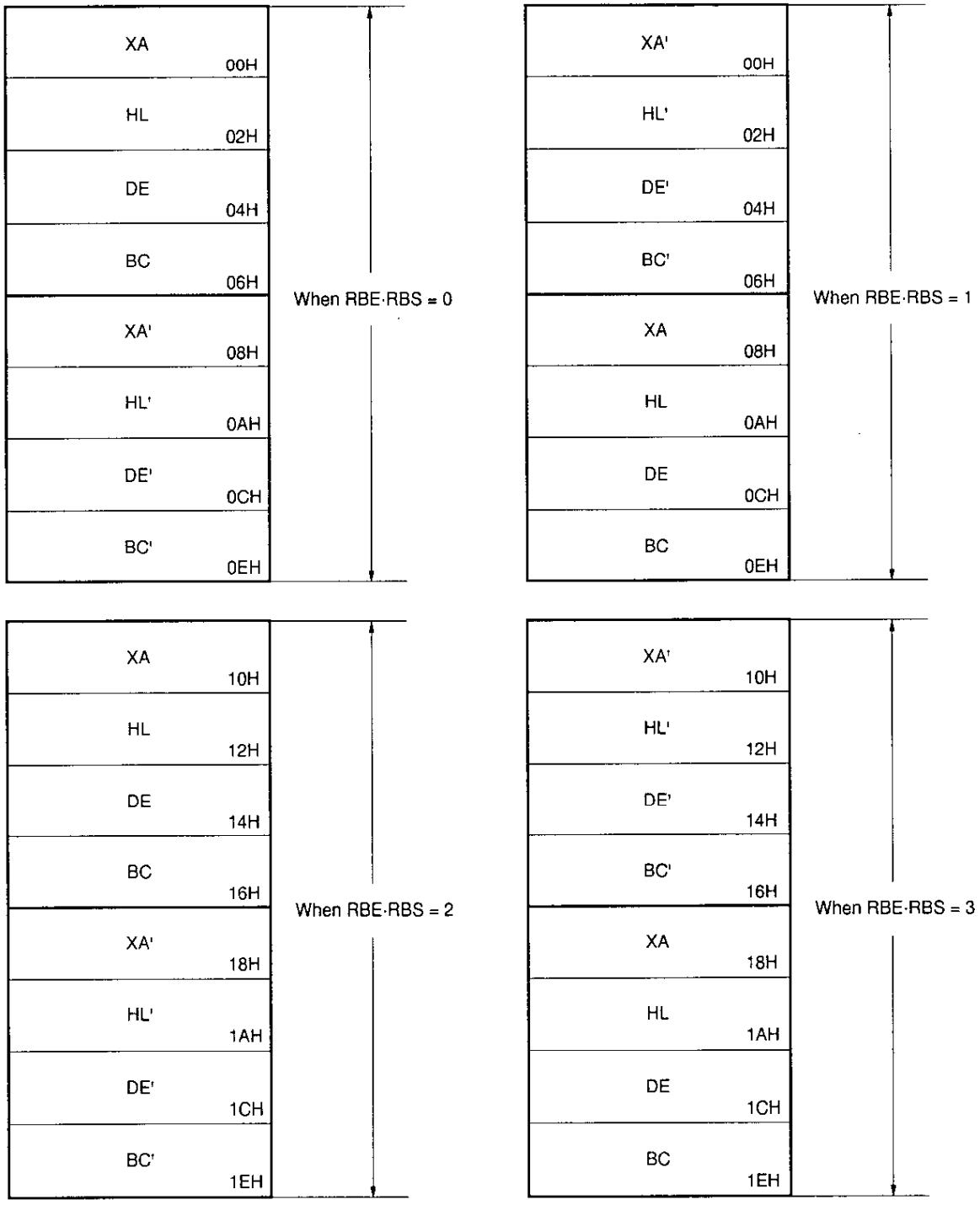


Figure 1-4. Configuration of General Registers (In Case of 8-Bit Processing)



1.3 Explanation of Application Programs

In the chapters of this Application Note, application programs are described on packaged for each function. Therefore, if an application program is combined with a user program (main program) using a linker, it can also function as part of a system program.

Furthermore, in **CHAPTER 11 APPLICATION EXAMPLES OF THIS APPLICATION PROGRAM** some of the application programs explained in CHAPTER 10 are used to create a system program.

When using each package, follow the explanation of the program corresponding to the package. A brief description of the items used in the program explanations is given below.

<Public declaration symbol>	:	Indicates the subroutine used in the package. If this symbol is externally referenced/declared, referencing is possible within the package.
<Externally referenced/declared symbol>	:	Indicates the area of data memory used in the package. If this symbol is defined and publicly declared, it can be used within the package.
<Register used>	:	Indicates the register used in the package.
<RAM used>	:	Indicates the area of data memory used in the package.
<Nesting>	:	Indicates the nesting level enabled in the package. Figures in parentheses denote the maximum value of the stack used.
<Hardware used>	:	Indicates the hardware used in the package.
<Interrupt>	:	Indicates the interrupt used in the package.
<Initialization>	:	Indicates initialization required to operate the package. However, SCC = 0 and PCC = 3 are set in each package unless specified otherwise.
<Start-up method>	:	Indicates the procedure necessary to operate the package.

Each package operates in Mk II mode. The register banks can be switched by the following interrupts.

RBE = 0, RBS = 0: Main routine
RBE = 1, RBS = 1: INT0 interrupt (reception of remote control)
RBE = 1, RBS = 2: Basic interval timer
RBE = 1, RBS = 3: SBI

An example of program description is shown below.

```
        VENT0   MBE = 0,RBE = 0,INIT        ; RESET
        VENT1   MBE = 0,RBE = 1,INTBT       ; INTBT/INT4
        VENT2   MBE = 0,RBE = 1,INT0        ; INT0
        VENT4   MBE = 0,RBE = 1,SBI         ; SBI
;**********************************
;     Initial setting
;**********************************
INIT:
        DI                                  ; Disables all interrupts.
        CLR1    MBE                         ; MBE ← 0
        MOV     XA, #00H                    ;
        MOV     SP,XA                       ; Stack pointer ← 00H
        MOV     SBS,A                       ; Memory bank ← 0, Mk II mode selected
;++++++++++++++++++++++++++++++++++
;     Remote control processing
;++++++++++++++++++++++++++++++++++
INT0:
        DI                                  ;
        PUSH    BS                          ;
        SEL     RB1                         ; Register bank ← 1
;++++++++++++++++++++++++++++++++++
;     Timer
;++++++++++++++++++++++++++++++++++
INTBT:
        SEL     RB2                         ; Register bank ← 2
;++++++++++++++++++++++++++++++++++
;     SBI processing
;++++++++++++++++++++++++++++++++++
SBI;
        SEL     RB3                         ; Register bank ← 3
```

CHAPTER 2 SYSTEM CLOCK SELECTION FUNCTION APPLICATIONS

For the μPD750008, the CPU clock and system clock can be switched by modifying the processor clock control register (PCC) and system clock control register (SCC). Figures 2-1 and 2-2 show the formats of PCC and SCC, respectively.

Figure 2-1. Processor Clock Control Register Format

Address	3	2	1	0	Symbol
FB3H	PCC3	PCC2	PCC1	PCC0	PCC

CPU clock selection bit
(where fx = 6.0 MHz)

		SCC3, SCC0 = 00 ( ) indicates fx = 6.0 MHz		SCC3, SCC0 = 01 or 11 ( ) indicates fxT = 32.768 kHz	
		CPU clock frequency	One machine cycle	CPU clock frequency	One machine cycle
0	0	Φ = fx/64 (93.7 kHz)	10.7 μs	Φ = fxT/4 (8.192 kHz)	122 μs
0	1	Φ = fx/16 (375 kHz)	2.67 μs		
1	0	Φ = fx/8 (750 kHz)	1.33 μs		
1	1	Φ = fx/4 (1.5 MHz)	0.67 μs		

(where fx = 4.19 MHz)

		SCC3, SCC0 = 00 ( ) indicates fx = 4.19 MHz		SCC3, SCC0 = 01 or 11 ( ) indicates fxT = 32.768 kHz	
		CPU clock frequency	One machine cycle	CPU clock frequency	One machine cycle
0	0	Φ = fx/64 (65.5 kHz)	15.3 μs	Φ = fxT/4 (8.192 kHz)	122 μs
0	1	Φ = fx/16 (261.8 kHz)	3.82 μs		
1	0	Φ = fx/8 (524 kHz)	1.91 μs		
1	1	Φ = fx/4 (1.05 MHz)	0.95 μs		

Remarks 1. fx : Main system clock oscillator output frequency
2. fxT: Subsystem clock oscillator output frequency

CPU operation mode control bit

0	0	Normal operation mode
0	1	HALT mode
1	0	STOP mode
1	1	Setting prohibited

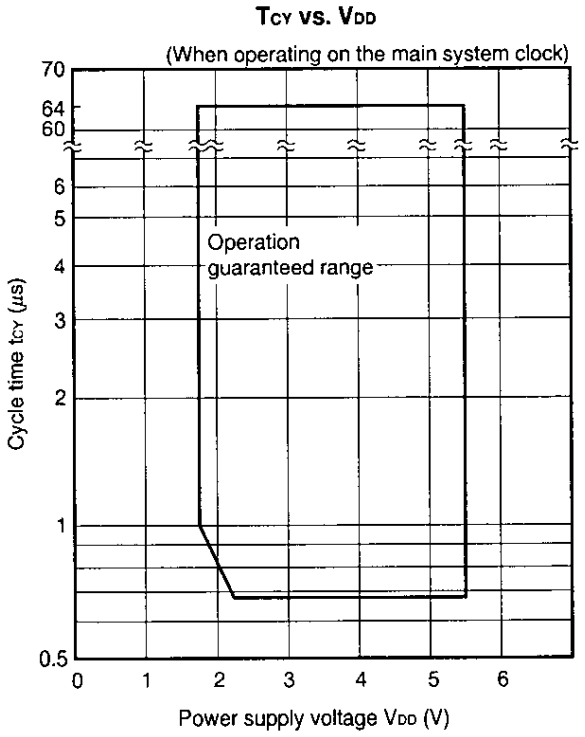
Figure 2-2. System Clock Control Register Format

Address	3	2	1	0	Symbol
FB7H	SCC3	—	—	SCC0	SCC

SCC3	SCC0	CPU clock selection	Main system clock oscillation
0	0	Main system clock	Oscillation possible
0	1	Subsystem clock	
1	0	Setting prohibited	
1	1	Subsystem clock	Oscillation stopped

Figure 2-3 shows the minimum instruction execution time (t_{CY}) v.s. power supply voltage (V_{DD}).

Figure 2-3. Power Supply Voltage (V_{DD}) v.s. Minimum Instruction Execution Time (t_{CY})



2.1 PCC Selection after $\overline{\text{RESET}}$

When a $\overline{\text{RESET}}$ signal is generated, the lowest speed mode of the main system clock is selected as the CPU clock. Therefore, for high-speed operation, the CPU clock must be set to the highest speed mode by modifying the PCC. However, the V_{DD} pin voltage must reach a voltage at which the CPU can operate at the highest speed before the PCC is modified (refer to **Figure 2-3**). For a system which uses a subsystem clock, the oscillation of the subsystem clock must be confirmed in advance.

In the following program examples, the system waits until the V_{DD} pin voltage rises and checks the subsystem clock oscillation using the clock timer; the CPU clock is then set to the highest speed mode (f_X = 4.19 MHz, f_{XT} = 32.768 kHz). Figure 2-4 shows the timing of this operation.

- **Program example**

(1) Wait until V_{DD} rises

```
        EXTRN   BIT(SSCOKF)         ; Subsystem clock oscillation check flag
                                      (Initial value = 0)
        VENT0   MBE=0,RBE=0,START
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
START   CSEG    INBLOCK             ; Specifies allocation in 4K-byte blocks of program
                                      memory
        MOV     XA,#00000110B       ; Clock timer advance mode
        MOV     WM,XA               ; (operates on the main system clock)
INILOP1:
        SKTCLR  IRQW                ; 3.9 ms waitNote
        BR      INILOP1
        MOV     A, #0010B           ; Drive current small
        MOV     SOS,A
        MOV     XA,#00000110B       ; Operates on the clock timer subsystem clock
        MOV     WM,XA
        MOV     A,#0011B            ; Sets the CPU clock to the highest speed mode
        MOV     PCC,A               ; Initialization processing follows
```

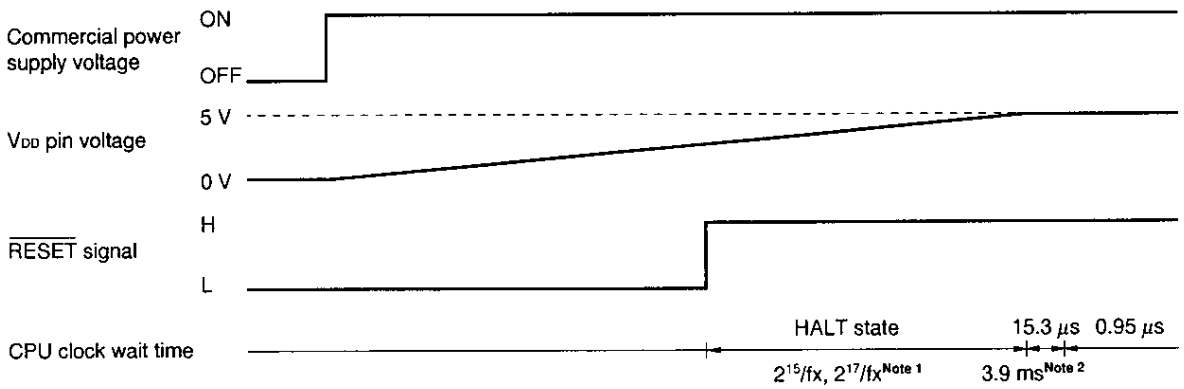
Note This time depends on the hardware. Set the time relevant to each system.

(2) Subsystem clock oscillation check subroutine (called by user processing)

```
SSCCHK  CSEG    INBLOCK
        SKF     SSCOKF          ; If the subsystem clock oscillation has already been
                                  checked, nothing is performed.
        RET
        SKTCLR  IRQW            ; Subsystem clock oscillation
        RET
        MOV     XA,#00000100B   ; Clock timer normal operation mode
        MOV     WM,XA
        SET1    SSCOKF          ; Subsystem clock oscillation check flag
        RET
                END
```

It takes approximately 1 second to stabilize the oscillation of the subsystem clock. Therefore, no wait (until the clock oscillation stabilizes) is executed in the initialization processing, and processing is performed in the main routine to check the subsystem clock oscillation. Then a flag is set to inform the other systems that subsystem clock oscillation has been initiated.

Figure 2-4. CPU Clock Selection after $\overline{\text{RESET}}$



Notes 1. Can be selected by mask option. (2^{15}/fx = 7.81 ms, 2^{17}/fx = 31.3 ms at 4.19 MHz operation).

2. This time depends on the hardware. Set the time relevant to each system.

2.2 System Clock Selection when Commercial Power Line Failure is Detected

When the subsystem clock (32.768 kHz) is selected by the SCC, the μPD750008 can operate at very low power dissipation. Therefore, if a back-up power system such as a NiCd battery or super capacitor is provided on the system, clock count operation, etc, can be maintained at very low power dissipation in the event of power failure.

In the following program example, an interruption is detected in the commercial power line by external interrupt INT4, and the system clock is switched so as to operate at very low power.

The following describes the system clock selection procedure using Figures 2-5 and 2-6.

Figure 2-5. System Clock Selection when Commercial Power Supply is Turned ON/OFF

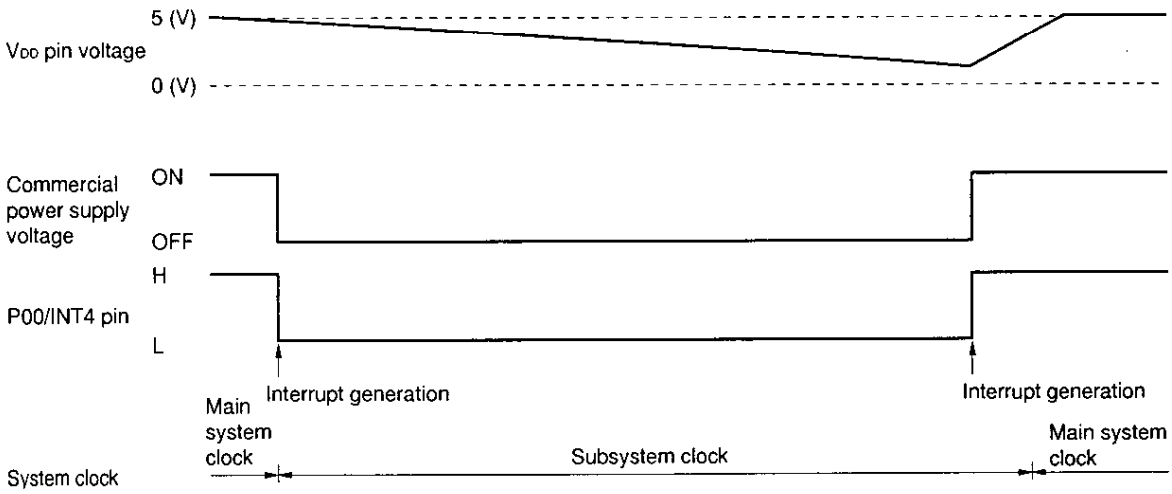
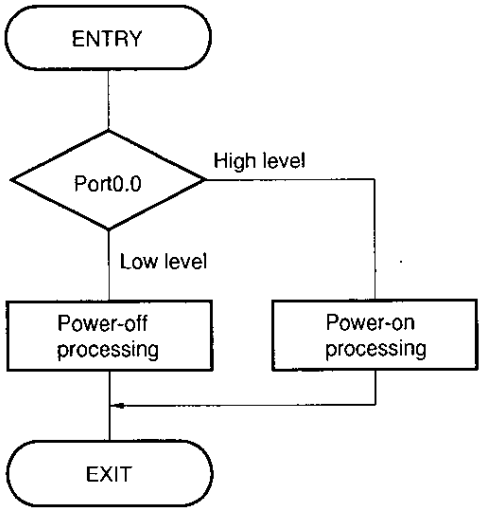


Figure 2-6. Algorithm of INT4 Interrupt Servicing



2.2.1 Power-off processing

The Port0.0 is checked in INT4 interrupt servicing, and if the level is LOW, it is determined that the commercial power supply is off. Power-off processing is then initiated.

The following describes the power-off processing procedure.

<1> Set 0011B to PCC.

<2> Set bit 0 of SCC to select the subsystem clock as the system clock.
All peripheral hardware except the clock timer and basic interval timer (which cannot be stopped) must be stopped before changing the system clock.

<3> Set the input/output pins in a manner so that the power dissipation is minimized (See note).

<4> More than 32 machine cycles after bit 0 of SCC is set, set bit 3 of SCC to stop the main system clock oscillation.

Caution Handling pins during power-off

When commercial power supply is turned off, normally, the power of the peripheral circuits is also turned off. Therefore, the following procedure is required using the connected peripheral circuits.

1. When a peripheral circuit connected to a pin becomes high impedance

- **For the pin for which input/output can be selected, set to the output mode and output a low level.**
- **For the input pin, pull down or pull up in advance.**

2. When the peripheral circuit connected to a pin does not become high impedance

- **Output a level which creates no current.**
- **If the input pin is stable at high or low level, no special procedure is required.**

Perform the necessary procedures depending on the condition of each peripheral circuit.

2.2.2 Power-on processing

The Port0.0 is checked in INT4 interrupt servicing, and if the level is HIGH, it is determined that the commercial power supply is on. Power-on processing is then initiated. The following describes the power-on processing procedure.

<1> Wait until the V_{DD} pin voltage rises at the level that can be operated at highest speed.

<2> Clear bit 3 of SCC to initiate the main system clock oscillation.

<3> Wait for a time period necessary for the main system clock to stabilize oscillation, then clear bit 0 of SCC to switch to the system clock.

2.2.3 Power-on/off processing application

An example of a program using power-on/off processing is shown below.

This program is not used in **CHAPTER 11 APPLICATION EXAMPLES OF THIS APPLICATION PROGRAM.**

(1) Program description

<External reference declaration symbol>

SSCOKF : Subsystem clock oscillation check flag
INTD0 to 5: Interrupt-related initialize data
INDBTM : BTM initialization data
INTM0 : TM0 initialization data
INCSIM : CSIM initialization data
INWM : WM initialization data
WATCH : Clock count processing entry label

<Registers used>

XA, HL

<Nesting>

4 levels (20 words)

<Initial setting>

INT4 interrupt enable

<Start-up procedure>

(a) Power on/off

When started up by the INT4 interrupt, one of the following processes is performed depending on the status of Port0.0.

(i) When Port0.0 is high (power-on processing):

The main system clock is selected as the system clock and each peripheral device is started up.

(ii) When Port0.0 is low (power-off processing):

- If the subsystem clock is oscillating (SSCOKF:1), the peripheral devices (except for the clock timer and basic interval timer) are stopped. The subsystem clock is then selected as the system clock and the main system clock is stopped.
- If the subsystem clock is stopped (SSCOKF:0), the peripheral devices (except for the clock times and basic interval timer) are stopped.

(b) Clock count check subroutine

When TIMCNT is called, the clock count processing routine (a user program) is called if the subsystem clock is oscillating. If the subsystem clock is not oscillating, the clock count processing routine is not called and processing returns to the point from where TIMCNT was called.

If the power is off (Port0.0 is low), the following processes are repeated until the power is turned on (Port0.0 is high):

(i) HALT mode setting
(ii) Recovery by IRQW
(iii) Clock count processing routine call

(2) Program example

<INT4 interrupt servicing>

```
          VENTI    MBE=0,RBE=0,INT4
          EXTRN    NUMBER(INTD0,INTD1,INTD2,INTD3,INTD4,INTD5)
          EXTRN    NUMBER(INDBTM,INTM0,INCSIM,INWM)
          EXTRN    BIT(SSCOKF)
          EXTRN    CODE(WATCH)
AN12DT1   DSEG     0        AT 0
ROA:      DS       1H                ; A register
AN12DT    DSEG     0        AT 0D0H
PTSVA:    DS       4
;
;
INT4      CSEG     INBLOCK
          PUSH     BS
          PUSH     XA
          PUSH     HL
          SEL      MB15
          SKT      PORT0.0           ; Port0.0 check
          BR       POWOFF
          CLR1     SCC.3             ; Power-ON processing
          MOV      XA,#56
ONWAIT:
          DECS     A                 ; Wait until system clock oscillation stabilizes
          BR       ONWAIT
          DECS     X
          BR       ONWAIT
          CLR1     SCC.0             ; System clock selection
          CALLF    !HRDSET           ; Hardware restart subroutine
          BR       REINT4
POWOFF:
          SKT      SSCOKF            ; Subsystem clock oscillation check
          BR       $HSTPRO
          MOV      A,#0011B
          MOV      PCC,A
          SET1     SCC.0             ; System clock selection
HSTPRO:
          MOV      XA,#0             ; Hardware stop processing
          MOV      TM0,XA
          MOV      CSIM,XA
          MOV      A,#0101B
          SKF      SSCOKF
          MOV      WM,XA
          MOV      A,#8              ; Disables interrupts other than INT4
          MOV      0B8H,A
          SET1     MBE
          MOV      HL,#0BCH
          MOV      A,#0
DILOP:
          MOV      @HL,A
          INCS     L
          BR       DILOP
          CLR1     MBE
          EI       IEW
```

```
          ; <Subroutine input/output port processing>
            SKF       SSCOKF            ;
            SET1      SCC.3             ; Main system clock stop
REINT4:
            POP       HL
            POP       XA
            POP       BS
            RETI

          ; <Subroutine hardware restarts>
HRDSET      CSEG      SENT
            MOV       A,#INDBTM         ; Hardware restart
            MOV       BTM,A
            MOV       XA,#INTM0
            MOV       TM0,XA
            MOV       XA,#INCSIM
            MOV       CSIM,XA
            MOV       XA,#INWM
            MOV       WM,XA
;
;
;
;           PORT RECOVER PROCESS
;
;                                       ; Interrupt enable
;
            MOV        A,#INTD0
            MOV        0B8H,A
            MOV        A,#INTD1
            SKT        ROA.1
            DI         IEW
            MOV        A,#INTD2
            MOV        0BCH,A
            MOV        A,#INTD3
            MOV        0BDH,A
            MOV        A,#INTD4
            MOV        0BEH,A
            MOV        A,#INTD5
            MOV        0BFH,A
            RET

          ; <Subroutine clock count check>
TIMCNT      CSEG       INBLOCK
            SKF        SSCOKF           ; Clock count is not performed during subsystem clock
                                          check
            CALL       !WATCH
            SKF        PORT0.0          ; Power down check
            RET
            HALT                        ; Sets standby mode
            NOP
            BR         TIMCNT
```

CHAPTER 3 BASIC INTERVAL TIMER APPLICATIONS

3.1 Reference Time Generation

The μPD750008 has an 8-bit basic interval timer (BT). Four different intervals can be selected. The basic interval timer interrupt request flag (IRQBT) is set with this interval time.

The basic interval timer is controlled by the basic interval timer mode register (BTM). Figure. 3-1 shows the format of this register.

Figure 3-1. Basic Interval Timer Mode Register Format

Address	3	2	1	0	Symbol
F85H	BTM3	BTM2	BTM1	BTM0	BTM

(fx = 6.00 MHz)

			Input clock specification	Interrupt interval time (Wait time when releasing the standby mode)
0	0	0	$fx/2^{12}$ (1.46 kHz)	$2^{20}/fx$ (175 ms)
0	1	1	$fx/2^{9}$ (11.7 kHz)	$2^{17}/fx$ (21.8 ms)
1	0	1	$fx/2^{7}$ (46.9 kHz)	$2^{15}/fx$ (5.46 ms)
1	1	1	$fx/2^{5}$ (188 kHz)	$2^{13}/fx$ (1.37 ms)
Other than above			Setting prohibited	—

(fx = 4.19 MHz)

			Input clock specification	Interrupt interval time (Wait time when releasing the standby mode)
0	0	0	$fx/2^{12}$ (1.02 kHz)	$2^{20}/fx$ (250 ms)
0	1	1	$fx/2^{9}$ (8.18 kHz)	$2^{17}/fx$ (31.3 ms)
1	0	1	$fx/2^{7}$ (32.768 kHz)	$2^{15}/fx$ (7.81 ms)
1	1	1	$fx/2^{5}$ (131 kHz)	$2^{13}/fx$ (1.95 ms)
Other than above			Setting prohibited	—

Basic interval timer start control bit

Basic interval timer operation is started (the counter and interrupt request flag are cleared) when "1" is written to this bit. This bit is automatically reset to 0 when operation is started.

The following shows four examples of interval time settings (at fx = 4.19 MHz operation).

(1) To set the interval time to 250 ms (IRQBT is set every 250 ms):

```
SEL     MB15

MOV     A,#1000B
MOV     BTM,A
```

(2) To set the interval time to 31.3 ms (IRQBT is set every 31.3 ms):

```
SEL     MB15

MOV     A,#1011B
MOV     BTM,A
```

(3) To set the interval time to 7.81 ms (IRQBT is set every 7.81 ms):

```
SEL     MB15

MOV     A,#1101B
MOV     BTM,A
```

(4) To set the interval time to 1.95 ms (IRQBT is set every 1.95 ms):

```
SEL     MB15

MOV     A,#1111B
MOV     BTM,A
```

3.2 Watchdog Timer Application

When "1" is set in the watchdog timer enable flag (WDTM), the basic interval timer/watchdog timer operates as a watchdog timer that generates an internal reset signal by overflow of the basic interval timer (BT) (however, once WDTM is set to "1", no operation but a reset can clear it). BT is always incremented by a clock from the clock generator, and cannot stop count operation (refer to **Figure 3-2**).

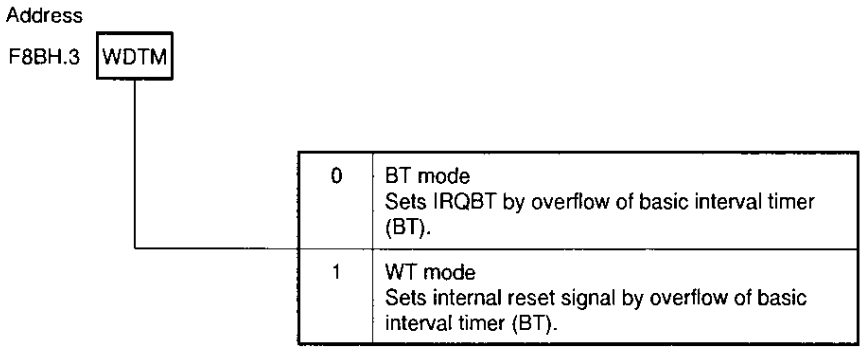
In watchdog timer mode, an inadvertent program loop is detected by using the interval time during which BT overflows. Four types of time are available for this interval time by setting BTM bits 2 to 0 (refer to **Figure 3-1**). Determine the time required for detection of any inadvertent program loop among these types according to the user system. After setting the interval time, divide the program into units by which the entire program can be executed within the time so that the instruction clearing BT at the end of each unit is executed. Thus, if the program does not reach the instruction clearing this BT within the set time (an inadvertent program loop will result if execution of the program does not proceed normally), BT will overflow, generating an internal reset signal which will forcibly end the program. As a result, the fact that the internal reset is executed means that an inadvertent program loop has occurred and that it has been successfully detected.

Set the watchdog timer according to the following procedure. (Settings in <1> and <2> can be performed simultaneously.)

<1> Set the interval time in BTM.
<2> Set "1" in BTM bit 3.
<3> Set "1" in WDTM.
} Initial setting

<4> After setting <1> to <3>, set "1" in BTM bit 3 within the interval time.

Figure 3-2. Format of Watchdog Timer Enable Flag



Example Used as a 7.81 ms watchdog timer (in 4.19 MHz operation)

The program is divided into several modules whose processing ends within the set time of BTM (7.81 ms) and BT is cleared at the end of each module. In the event of an inadvertent program loop, BT overflows because it has not been cleared within the set time, generating an internal reset signal.

Initial setting:

```
SET1    MBE
SEL     MB15
MOV     A.#1101B
MOV     BTM.A       ; Time setting and start
SET1    WDTM        ; Enables watchdog timer.
  ⋮
```

(Subsequently, "1" is set in BTM bit 3 every 7.81 ms.)

Module 1:

```
  ⋮
SET1    MBE
SEL     MB15
SET1    BTM.3
```

Processing completed within 7.81 ms

Module 2:

```
  ⋮
SET     MBE
SEL     MB15
SET1    BTM.3
  ⋮
```

Processing completed within 7.81 ms

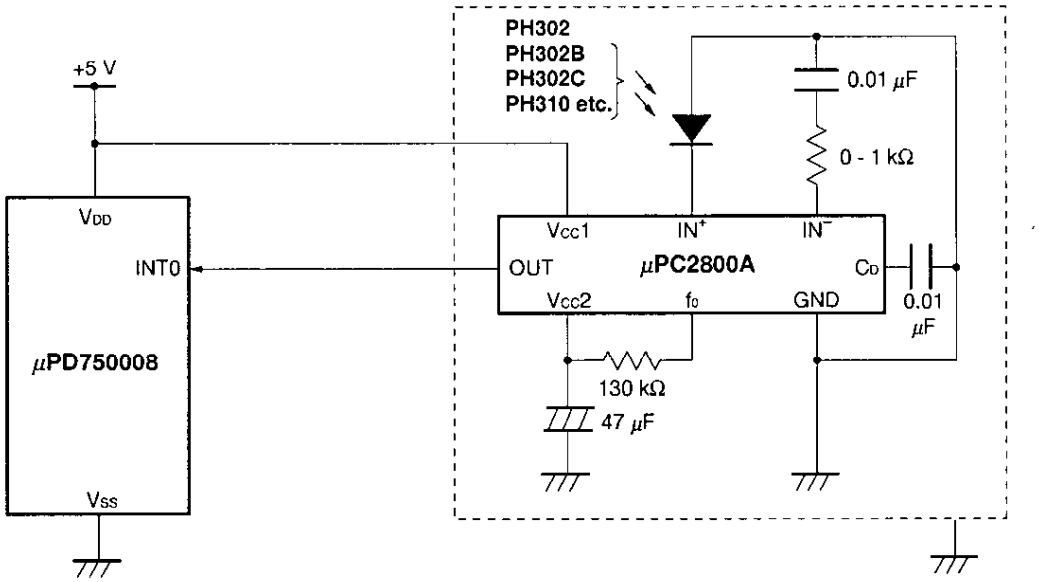
3.3 Remote Control Reception Application

This section introduces an example program which receives transmission data from the μPD6122 transmitter for the general-purpose infrared remote controller using the basic interval timer.

The remote control signal is received with the PIN photo diode, passed through the μPC2800A remote control preamplifier and input to the μPD750008 through the P10/INT0 pin. (Refer to **Figure 3-3**).

For this program, the edge-to-edge length of the remote control signal is counted using INTBT and coded.

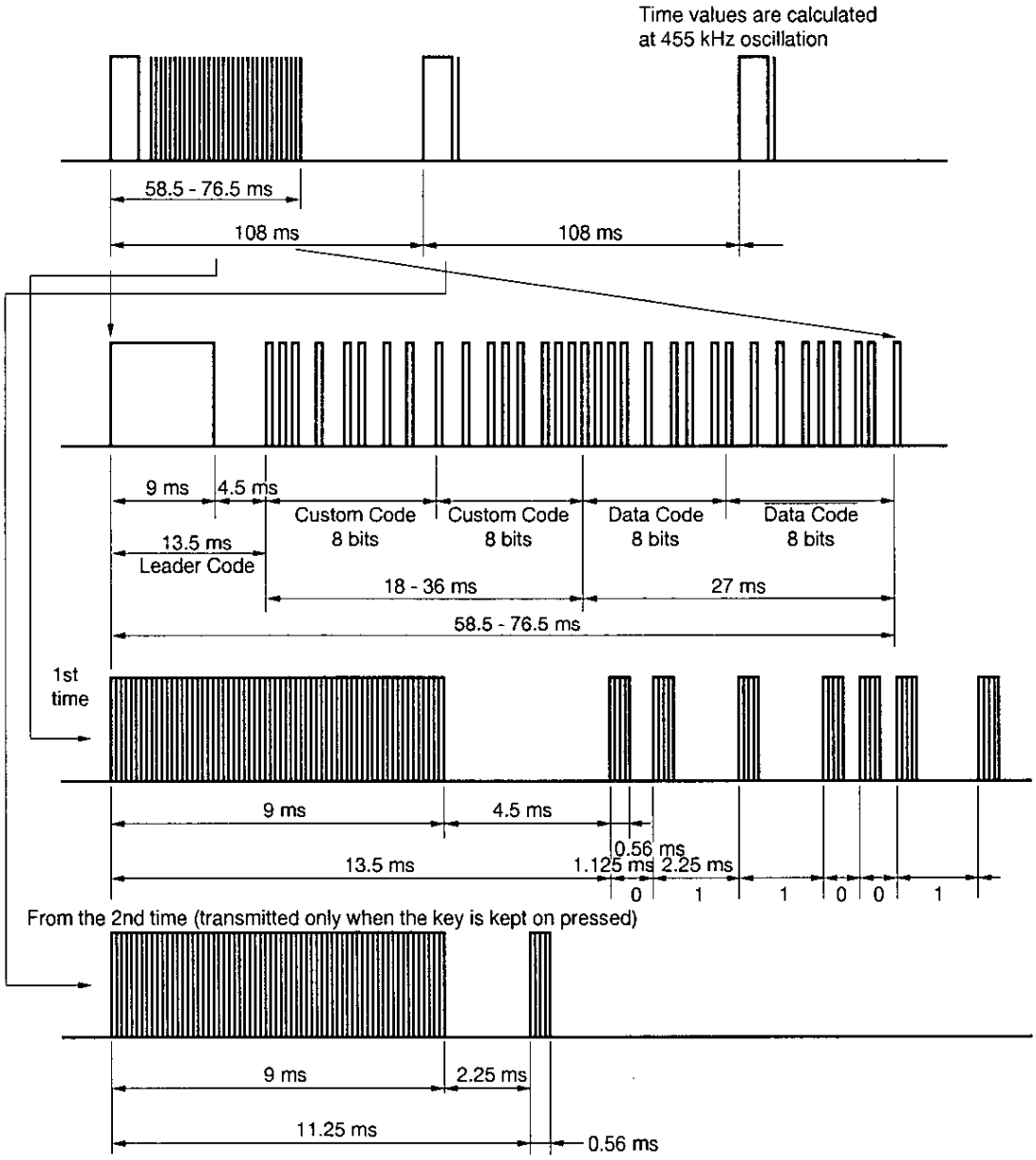
Figure 3-3. Remote Control Receiver Circuit Example



The remote control signal consists of the leader code, custom code, data code, and repeat code.

The remote control signal shown here conforms to that of NEC remote control signal. Figure 3-4 shows the format of the remote control signal.

Figure 3-4. Remote Control Transmitter IC Output Signal

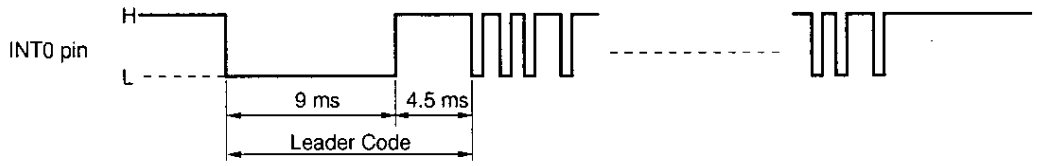


The out put from the μPC2800A remote control preamplifier is low, as shown in Figure 3-5 (a).

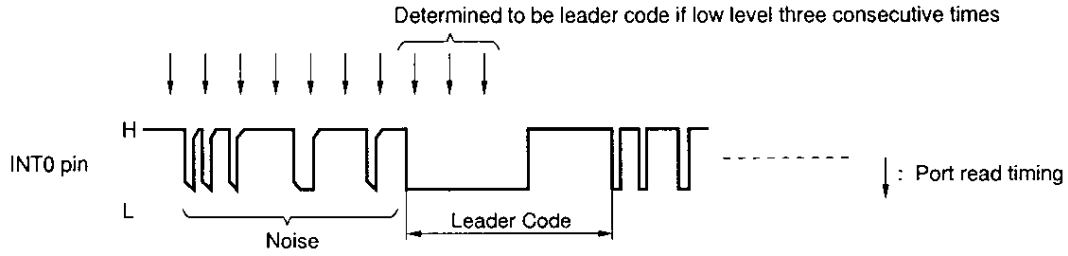
In actual operation, however, electrical noise sometimes appears before the leader code due to external light sources such as fluorescent lights. Therefore, this program confirms the falling edge of the leader code by reading the port level during the basic interval timer interrupt servicing. (Refer to **Figure 3-5 (b)**).

Figure 3-5. Receiver Preamplifier Output Waveform

(a) Ideal Wavefrom



(b) Actual Waveform



In the following **(1) Explanation of program** and **(2) Flowchart**, data recognized as normal by a remote control signal is expressed as "valid."

(1) Explanation of program

The program explained here is the one used in **CHAPTER 11 APPLICATION EXAMPLES OF THIS APPLICATION PROGRAM** with some modifications to its input/output interface.

<Registers used>

XA, B, HL

<RAM used>

RAM can be placed from address 20H of memory bank 0.

```
WORK:    2 words   ; Work area to store received data
RMDATA:  2 words   ; Area to store valid data code
LDCODE:  1 word    ; Counts reader code low level time.
MODEP:   1 word    ; Parameter to indicate which edge is being detected
RPTIM:   1 word    ; Counts time detected from valid data input (up to 200 ms)
RPCODE:  1 word    ; Counts valid repeat code.
RMFLG:   1 word    ; Stores flag.
REP_F:   1 bit     ; If 1, starts RPTIM count.
```

<Nesting>

2 levels (16 words)

<Hardware used>

- Port: Port1.0 (used as INT0)
- Timer: Basic interval timer

<Interrupts used>

INT0 and INTBT

<Initial setting>

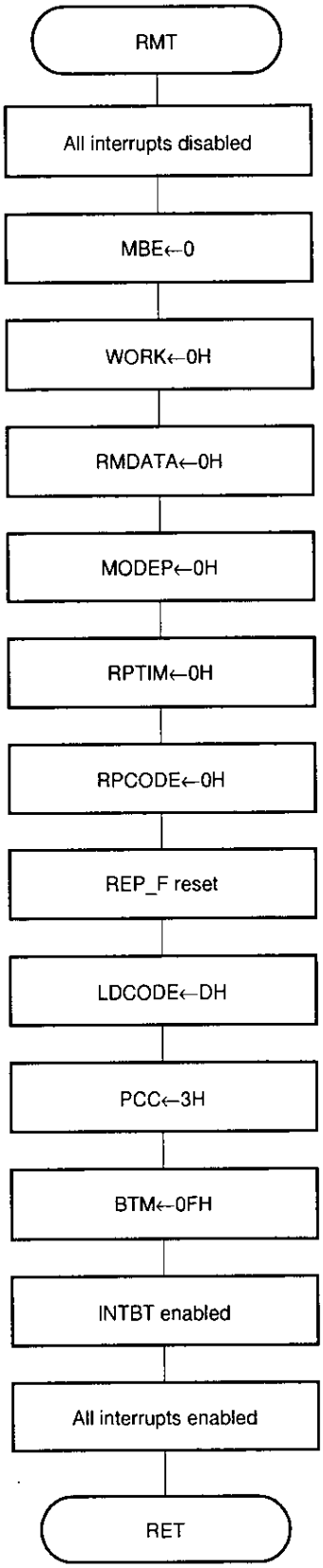
- RAM initial setting
- Hardware initial setting
- INTBT interrupt enabled

<Start-up procedure>

- Started by executing initial setting processing (RMT:).
- If a valid remote control code is input, REP_F = 1 is set, and the code is stored in RMDATA. Then, if no repeat code is input in 200 ms (it is judged that the remote control key has been released) or if a repeat code is input four times or more (it is judged that a remote control signal for 2 or more machines has been received) REP_F = 0 is set.
- While the system waits for reception of a remote control code, MODEP = 0 is set.

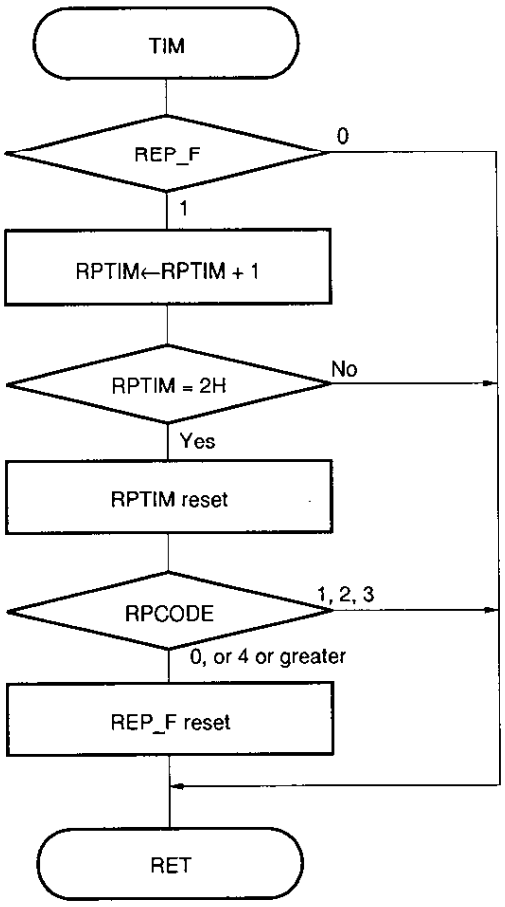
(2) Flowchart

Initial setting



1 machine cycle ← 0.95 μs (minimum)

Interval time ← 1.95 ms

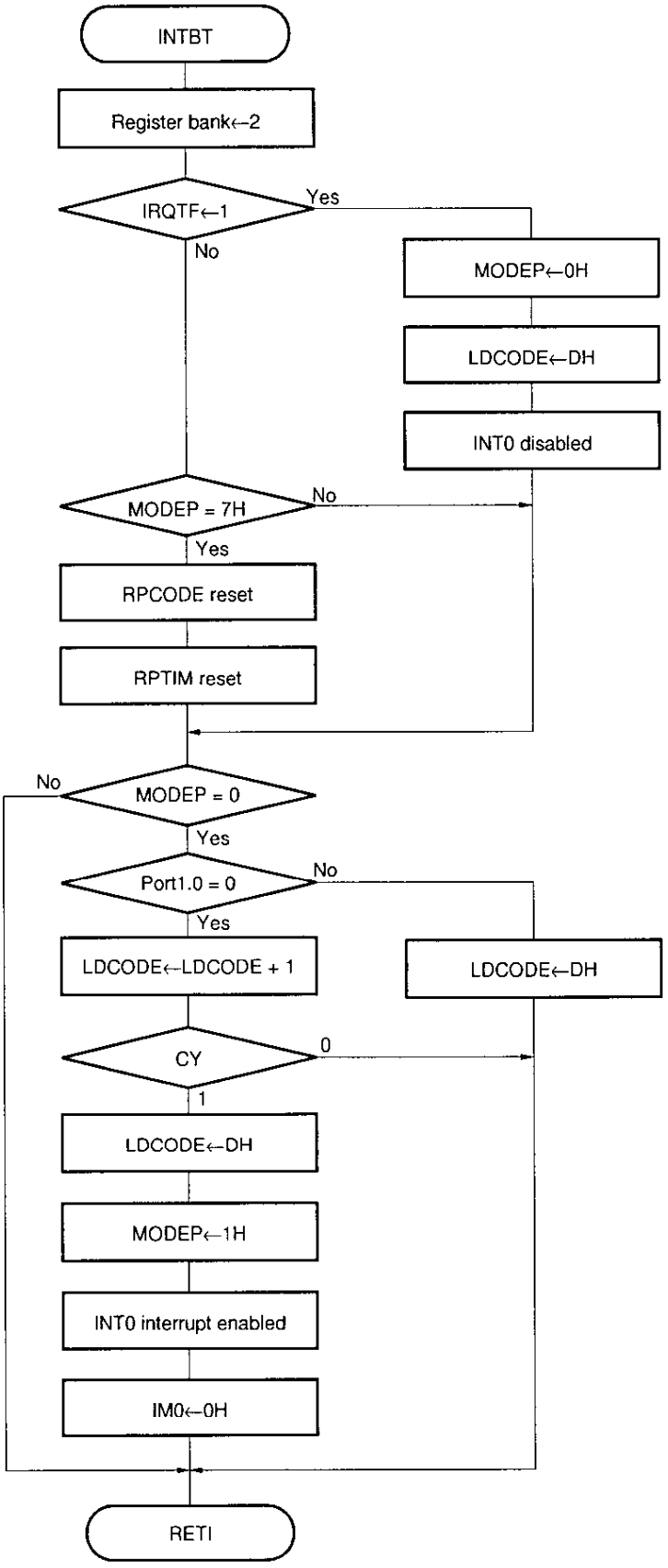


Processes every 100 ms.

Checks if 200 ms have elapsed.

Checks number of times of repeat code.

INTBT processing



Processes every 2 ms.

If IRQTF = 1, counting of modulo register ends.

Resets repeat code counter.

Resets 200 ms counter.

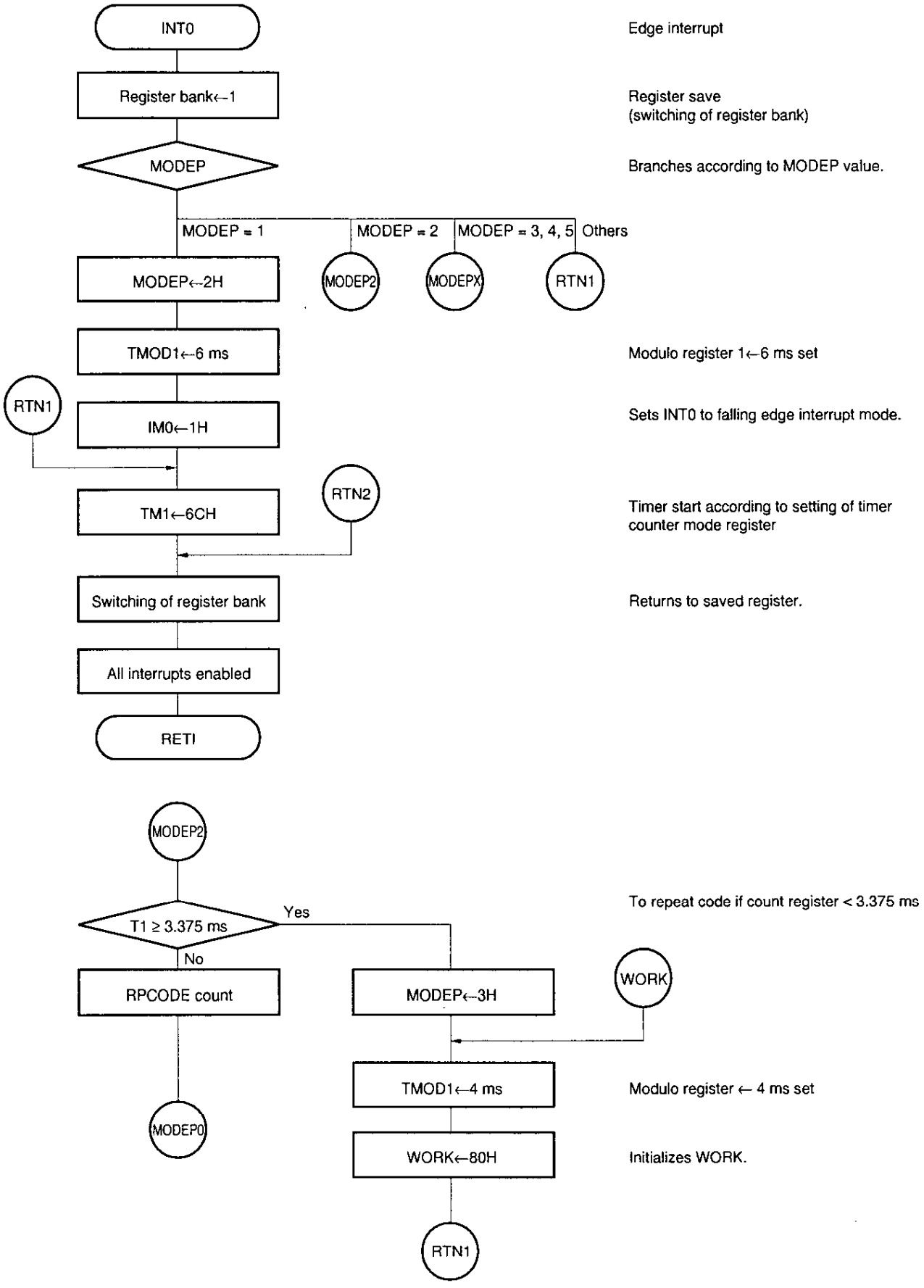
Checks low level time of reader code when MODEP = 0.

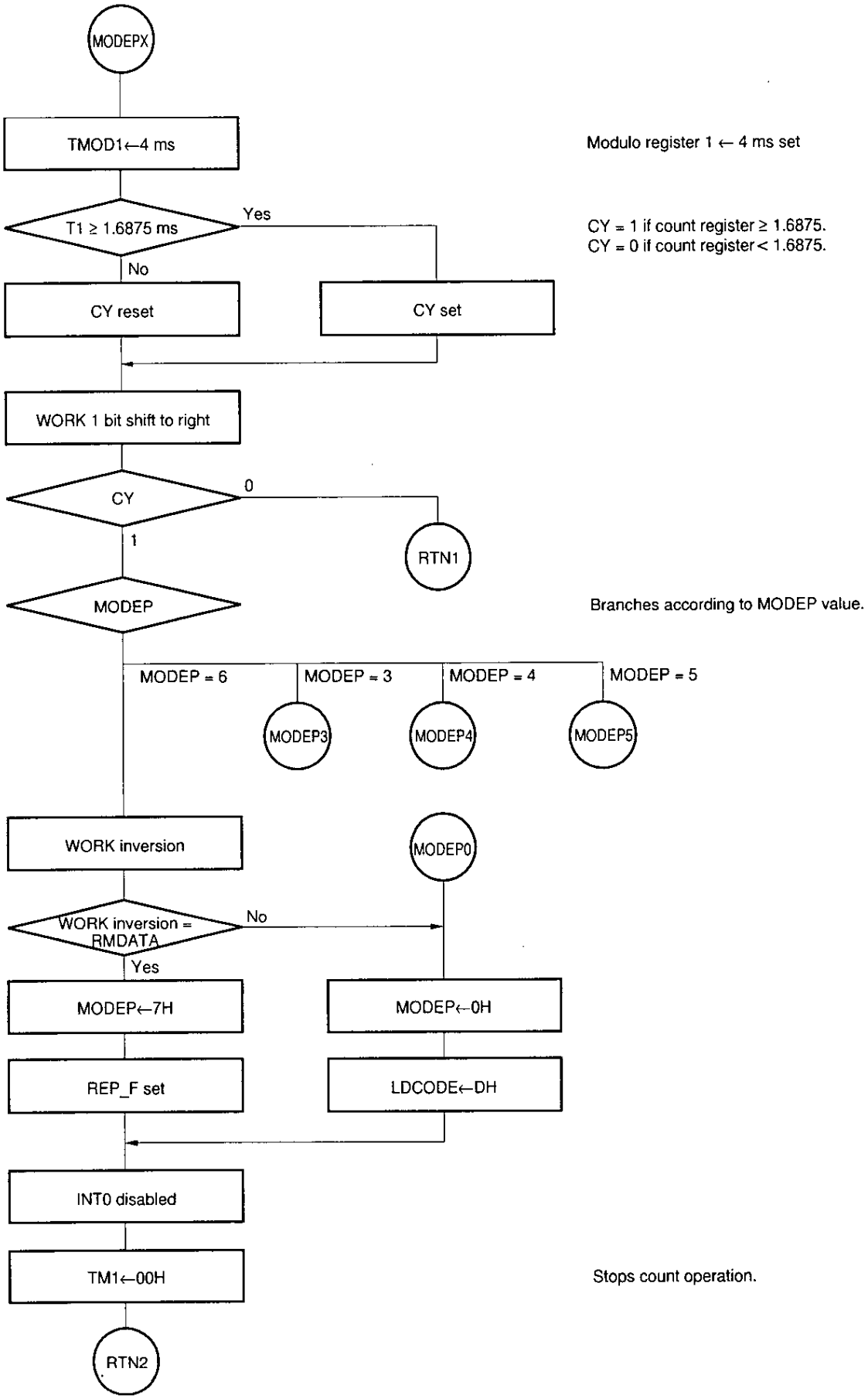
Counts LDCODE if INT0 is low.
Initializes LDCODE when INT0 is high.

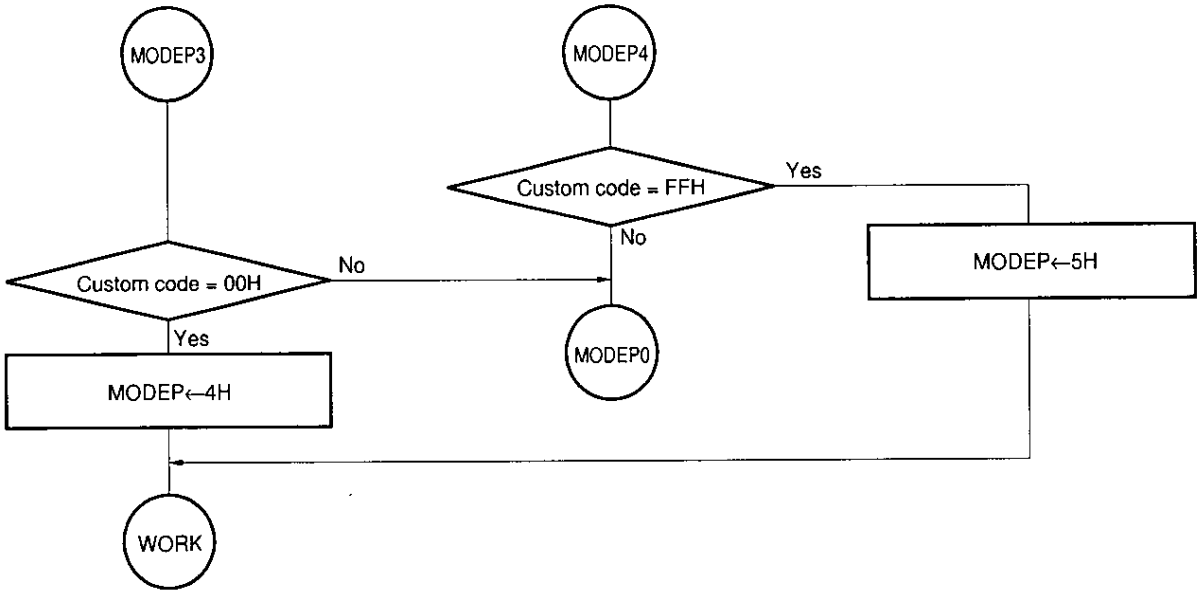
To checking of high level time of reader code

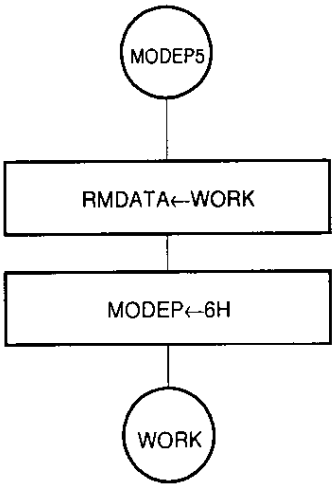
Sets INT0 to rising edge interrupt mode.

INT0 processing




MODEPX
TMOD1←4 ms
Modulo register 1 ← 4 ms set
T1 ≥ 1.6875 ms
Yes
No
CY = 1 if count register ≥ 1.6875.
CY = 0 if count register < 1.6875.
CY reset
CY set
WORK 1 bit shift to right
CY
0
1
RTN1
MODEP
Branches according to MODEP value.
MODEP = 6
MODEP = 3
MODEP = 4
MODEP = 5
MODEP3
MODEP4
MODEP5
WORK inversion
MODEP0
WORK inversion = RMDATA
No
Yes
MODEP←7H
MODEP←0H
REP_F set
LDCODE←DH
INT0 disabled
TM1←00H
Stops count operation.
RTN2


MODEP3
Custom code = 00H
No
Yes
MODEP←4H
WORK
MODEP4
Custom code = FFH
Yes
No
MODEP←5H
MODEP0


MODEP5
RMDATA←WORK
MODEP←6H
WORK

(3) Program example

```
          VENT0     MBE=0,RBE=0,INIT        ; RESET
          VENT1     MBE=0,RBE=1,INTBT       ; INTBT/INT4
          VENT2     MBE=0,RBE=1,INT0        ; INT0

DSEG0     DSEG      0         AT      20H   ; Stores from address 20H of memory bank 0.

WORK:               DS        2             ; Work area to store received data
RMDATA:             DA        2             ; Stores valid data code.
LDCODE:             DS        1             ; Counts low level time of reader code.
MODEP:              DS        1             ; Parameter that indicates which edge is being
                                              detected
RPTIM:              DS        1             ; Counts time elapsed from valid data input
                                              (up to 200 ms).
RPCODE:             DS        1             ; Counts valid repeat code.
RMFLG:              DS        1             ; Flag

REP_F               EQU       RMFLG.0       ; If 1, RPTIM count start

        ; <Subroutine initial setting>
INIT:
          DI                                ; Disables all interrupts.
          CLR1      MBE                     ; MBE ← 0

          MOV       XA,#00H                 ; Sets RAM.
          MOV       WORK,XA
          MOV       RMDATA,XA
          MOV       MODEP,A
          MOV       RPTIM,A
          MOV       RPCODE,A
          MOV       RMFLG,A
          MOV       A,#0DH
          MOV       LDCODE,A                ; Initializes LDCODE.

          MOV       A,#0011B
          MOV       PCC,A                   ; 1 machine cycle ← 0.95 μs, CPU is set to
                                              normal operating mode.

          MOV       A,#1111B
          MOV       BTM,A                   ; Interval time ← 1.95 ms

          EI        IEBT                    ; Enables use of basic interval timer (INTBT).
          EI                                ; Enables all interrupts.
```

```
        ; <Subroutine  100 ms timer processing>
            SKT       REP_F
            RET
            INCS      RPTIM                 ; Remote control repeat timer + 1
            NOP
            MOV       A,RPTIM
            SKE       A,#2H                 ; Remote control repeat timer = 200 ms?
            RET
            MOV       A,#0H
            MOV       RPTIM,A               ; Clears RPTIM.
            MOV       A,RPCODE
            ADDS      A,#0FH                ; Remote control repeat code counter = 0?
            BR        TIM_52
            ADDS      A,#0FH                ; Remote control repeat code counter = 1?
            RET
            ADDS      A,#0FH                ; Remote control repeat code counter = 2?
            RET
            ADDS      A,#0FH                ; Remote control repeat code counter = 3?
            RET

TIM_52:
            CLR1      REP_F                 ; REP_F ← 0
            RET
```

; **<Subroutine INTBT processing>**

```
        SEL     RB2             ; Register bank ← 2
        SKTCLR  IRQT1           ; If 1, modulo register = count register
        BR      MODEP7
        MOV     A,#0H
        MOV     MODEP,A         ; MODEP ← OH
        MOV     A,#0DH
        MOV     LDCODE,A        ; Initializes LDCODE.
        DI      IE0             ; Disables INT0 interrupt.
        BR      RMCNT
MODEP7:
        MOV     A,MODEP
        SKE     A,#7H           ; MODEP = 7?
        BR      RMCNT
        MOV     A,#0H
        MOV     MODEP,A         ; MODEP ← 0H
        MOV     RPCODE,A        ; Repeat code count reset
        MOV     RPTIM,A         ; 200 ms count reset
RMCNT:
        MOV     A,MODEP
        SKE     A,#0H           ; MODEP = 0?
        RETI
        SKF     PORT1.0
        BR      RMCNT_1
        INCS    LDCODE          ; Reader code scan counter + 1
        RETI
        MOV     A,#0DH          ; Initializes LDCODE.
        MOV     LDCODE,A
        MOV     A,#1H
        MOV     MODEP,A         ; MODEP ← 1
        MOV     A,#0H
        MOV     IM0,A           ; Specifies rising edge.
        EI      IE0             ; Enables INT0 interrupt.
        RETI

RMCNT_1:
        MOV     A,#0DH
        MOV     LDCODE,A        ; Initializes LDCODE.
        RETI
```

```
;<Subroutine INT0 processing>
        DI
        PUSH    BS
        SEL     RB1                     ; Register bank ← 1
        MOV     A,MODEP
        ADDS    A,#0FH                  ; MODEP = 0?
        BR      INT0_E
        ADDS    A,#0FH                  ; MODEP = 1?
        BR      INT0_P1
        ADDS    A,#0FH                  ; MODEP = 2?
        BR      INT0_P2
        ADDS    A,#09H                  ; MODEP = 3 - 6?
        BR      INT0_PX
        BR      INT0_E

INT0_P1:
        MOV     A,#2H
        MOV     MODEP,A                 ; MODEP ← 2
        MOV     XA,#62H
        MOV     TM0D1,XA                ; Modulo register ← 6 ms
        MOV     A,#1H
        MOV     IM0,A                   ; Specifies falling edge.
        BR      INT0_E

INT0_P2:
        MOV     XA,T1
        ADDS    XA,#0C9H                ; Count register = 3.375 ms?
        BR      INT0_RP1
        MOV     A,#3H
        MOV     MODEP,A                 ; MODEP ← 3
        BR      INT0_W1

INT0_RP1:
        INCS    RPCODE                  ; Remote control repeat code counter + 1
        NOP
        BR      INT0_P0

INT0_PX:
        MOV     XA,T1
        ADDS    XA,#0E4H                ; Count register = 1.6875 ms?
        BR      INT0_CY0
        SET1    CY                      ; CY ← 1
        BR      INT0_L1
INT0_CY0:
        CLR1    CY                      ; CY ← 0
INT0_L1:
        MOV     XA,WORK
        MOV     B,A
        MOV     A,X
        RORC    A
        MOV     X,A
        MOV     A,B
        RORC    A
```

```
        MOV     WORK,XA             ; Shifts 1 bit of receive data (WORK) to right.
        SKT     CY
        BR      INT0_E
        MOV     A,MODEP
        ADDS    A,#0CH              ; MODEP = 3?
        BR      INT0_P3
        ADDS    A,#0FH              ; MODEP = 4?
        BR      INT0_P4
        ADDS    A,#0FH              ; MODEP = 5?
        BR      INT0_P5
        MOV     XA,WORK             ; MODEP = 6
        MOV     HL,#0FFH
        XOR     HL,XA
        MOV     XA,RMDATA
        SKE     XA,HL               ; Inverts valid remote control data = receive data?
        BR      INT0_P0
        MOV     A,#7H
        MOV     MODEP,A             ; MODEP ← 7
        SET1    REP_F
        DI      IE0                 ; Disables INT0 interrupt.
        MOV     XA,#0H
        MOV     TM1,XA              ; Stops timer count.
        BR      INT0_E1

INT0_P3:
        MOV     XA,WORK
        MOV     HL,#00H
        SKE     XA,HL               ; Custom code = 00H?
        BR      INT0_P0
        MOV     A,#4H
        MOV     MODEP,A             ; MODEP ← 4
        BR      INT0_W1

INT0_P4:
        MOV     XA,WORK
        MOV     HL,#0FFH
        SKE     XA,HL               ; Custom code = FFH?
        BR      INT0_P0
        MOV     A,#5H
        MOV     MODEP,A             ; MODEP ← 5
        BR      INT0_W1

INT0_P5:
        MOV     XA,WORK
        MOV     RMDATA,XA           ; Inputs receive data to valid remote control data.
        MOV     A,#6H
        MOV     MODEP,A             ; MODEP ← 6
        BR      INT0_W1

INT0_P0:
        MOV     A,#0H
        MOV     MODEP,A             ; MODEP ← 0
        MOV     A,#0DH
        MOV     LDCODE,A            ; Initializes LDCODE.
        DI      IE0                 ; Disables INT0 interrupt.
        MOV     XA,#0H
```

```
          MOV       TM1,XA                ; Stops timer count.
          BR        INT0_E1

INT0_W1:
          MOV       XA,#0FFH
          MOV       TMOD1,XA              ; Modulo register ← 4 ms
          MOV       XA,#80H
          MOV       WORK,XA               ; Initializes WORK.

INT0_E:
          MOV       A,MODEP
          ADDS      A,#0DH
          MOV       XA,#6CH
          MOV       XA,#7CH
          MOV       TM1,XA                ; Timer count start
INT0_E1:
          POP       BS
          EI                              ; Enables all interrupts.
          RETI
```

CHAPTER 4 TIMER/EVENT COUNTER APPLICATIONS

4.1 Interval Timer Setting

The timer setting is determined by the count pulse frequency selected by the timer mode register and the value of the modulo register.

The timer setting time T(sec) can be obtained by the following formula.

$$T = \frac{N + 1}{f_{CP}} = (N + 1) \cdot (\text{Resolution}) \quad \ldots\ldots <1>$$

f_{CP}(Hz) : Count pulse frequency
N : Modulo register value ($N \neq 0$)

Table 4-1 indicates the resolution and the setting time range of each count pulse of the timer/event counter.

(1) Determining the count pulse frequency

Determine the count pulse frequency in a manner such that the timer setting time falls within the setting range indicated in Table 4-1 and the highest resolution is obtained.

(2) Determining the value of the modulo register From formula

<1> N can be determined as follows:

$$N = \frac{T}{(\text{Resolution})} - 1 \quad \ldots\ldots <2>$$

The value N, which is set to the modulo register, is determined by substituting the mode resolution determined by <1> to (Resolution).

Table 4-1. Resolution and Maximum Timer Value (fx = 4.194304 MHz)

Mode Register			Resolution	Maximum Time
TM06	TM05	TM04		
1	0	0	244 μs	62.5 ms
1	0	1	61 μs	15.6 ms
1	1	0	15.3 μs	3.9 ms
1	1	1	3.8 μs	977 μs

The following shows examples of setting the interval time using the timer mode register and the modulo register. In example 1 and 2, the timer/event counter interrupt request flag (IRQT0) is set every interval time.

Example 1. To set the interval time to 10 ms. (fx = 4.194304 MHz)

In this case, use the mode which yields a maximum setting time of 10 ms or longer and the lowest resolution, that is, the mode in which the 15.6 ms maximum setting time is used.

From formula <2>, the value set to the modulo register will be:

$$N = \frac{T}{(\text{Resolution})} - 1$$

$$= \frac{10 \times 10^{-3}}{61 \times 10^{-6}} - 1$$

$$= 162.9 = 0A3H$$

```
TIMER0:
        SEL     MB15
        MOV     XA,#0A3H
        MOV     TMOD0,XA            ; Modulo register setting
        MOV     XA,#01011100B
        MOV     TM0,XA              ; Timer mode register setting
```

Example 2. To set the interval time to 120 μs. (fx = 4.194304 MHz)

In this case, use the mode which yields a maximum setting time of 120 μs or longer and the lowest resolution, that is, the mode in which the 977 μs maximum setting time is used.

From formula <2>, the value set to the modulo register will be:

$$N = \frac{T}{(\text{Resolution})} - 1$$

$$= \frac{120 \times 10^{-6}}{3.8 \times 10^{-6}} - 1$$

$$= 30.5 \fallingdotseq 1FH$$

```
TIMER1:
        SEL     MB15
        MOV     XA,#1FH
        MOV     TMOD0,XA            ; Modulo register setting
        MOV     XA,#01111100B
        MOV     TM0,XA              ; Timer mode register setting
```

4.2 Example of Output to PTO Pin

For the μPD750008, the content of the TOUT F/F of the timer/event counter or timer counter can be output to the PTO0 or PTO1 pin.

TOUT F/F is inverted each time the values of the modulo register and count register coincide. Therefore, square waves can be output to the PTO0 or PTO1 pin. In this case, the output frequency is the count pulse frequency divided by the modulo register.

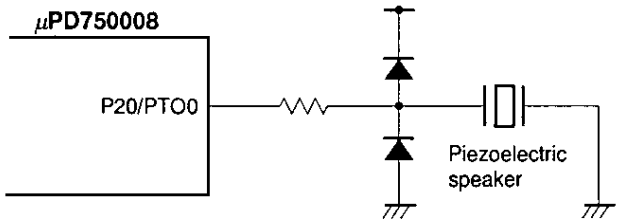
The following describes the procedures for outputting the contents of TOF0 to the PTO pin.

- Set the TO enable flag (TOE0 ← 1)
- Reset the output latch of Port2.0 (Port2.0 ← 0)
- Set Port2 to the output port mode (PM2 ← 1)
- Do not specify on-chip pull-up resistor connection of Port2 (POGA.2 ← 0)

4.2.1 Melody output

The following describes a program which externally outputs musical melodies using the PTO0 pin which is used as the timer event counter output.

Figure 4-1. PTO Pin Using Example



The frequency output from pin P20/PTO0 is determined according to the value set in the modulo registers and the count pulse frequency (CP), which is determined according to the timer/event counter mode register.

Table 4-2 indicates the value which should be set in the modulo register to obtain each musical note and the error in the frequency of each note derived from the value set in the modulo register when $fx/2^4$ has been selected as the count pulse (f_{CP} = 262 kHz when fx = 4.194304 MHz).

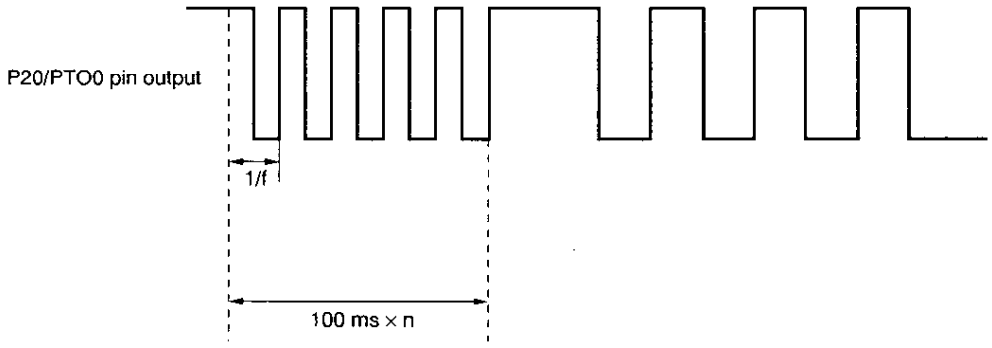
Table 4-2. Values to be Set in Modulo Register for Each Musical Scale, and Error between Frequency Produced and Each Musical Scale Frequency

Average Musical Scale			PTO0 Pin Output		Error (%)
Musical Scale		Frequency (Hz)	TMOD1	Frequency (Hz)	
Do	C5	523.25	F9H	523.75	+0.10
Do	C5#	554.37	EBH	554.82	+0.08
Re	D5	587.33	DEH	587.16	−0.03
Re	D5#	622.25	D1H	623.51	+0.20
Mi	E5	659.26	C6H	657.98	−0.19
Fa	F5	698.46	BAH	696.48	−0.28
Fa	F5#	739.99	B0H	739.76	−0.03
So	G5	783.99	A6H	784.06	+0.01
So	G5#	830.61	9CH	828.72	−0.23
La	A6	880.00	94H	878.78	−0.14
La	A6#	932.33	8BH	928.63	−0.40
Si	B6	987.77	83H	984.49	−0.33
Do	C6	1046.50	7CH	1047.50	+0.10
Do	C6#	1108.73	75H	1109.64	+0.08

Melodies are output with the frequency and time length of each musical scale changed as shown in Table 4-2. The time length is changed according to the number of count operations to be performed during a reference time of 100 ms created by using the basic interval timer.

Figure 4-2 is a conceptual chart of an output signal from pin P20/PTO0.

Figure 4-2. Conceptual Chart of Output Signal from Pin P20/PTO0



In this program example, a data table that conforms to **Figure 4-3 Data Format** is stored in RAM that starts from the performance data address (MUS) and melodies are output according to this data table. This data table consists of the data that determines musical scales and the data that determines the tone length, and a variety of melodies can be output by changing this combination.

Figure 4-3. Data Format

Data	Contents
0XH - FXH	Musical scale data (refer to **Table 4-4**)
X0H - XEH	Musical scale data (refer to **Table 4-5**)
nFH	After silent tone of n × 100 ms is output, repeats performance from the beginning.

Musical scales can be output by extracting data from the performance memory area.

Musical scale output

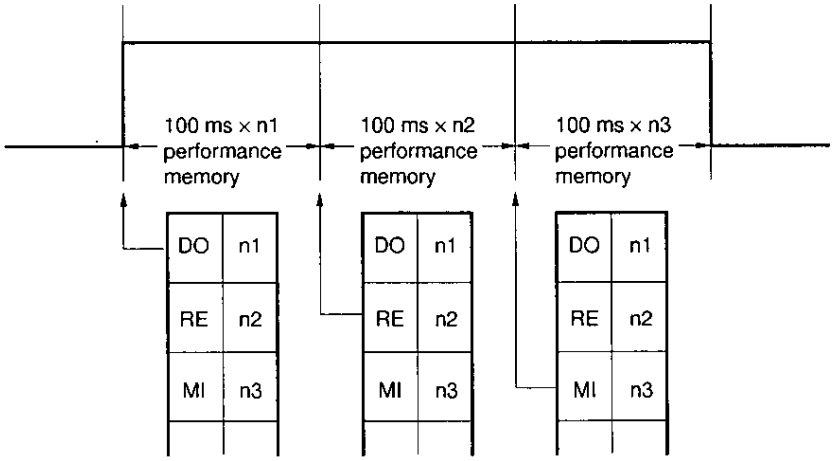


Table 4-3 shows the relationship between musical scale data and musical scale, display and PTO0 pin output, and Table 4-4 shows the relationship between tone length data and musical scale output.

Table 4-3. Relationship between Musical Scale Data and Musical Scale, Display and PTO0 Pin Output

Musical Scale Data	Average Musical Scale		PTO0 Pin Output	
	Musical Scale	Frequency (Hz)	TMOD1	Frequency (Hz)
00H	Do C5	523.25	F9H	523.75
01H	Do C5#	554.37	EBH	554.82
02H	Re D5	587.33	DEH	587.16
03H	Re D5#	622.25	D1H	623.51
04H	Mi E5	659.26	C6H	657.98
05H	Fa F5	698.46	BAH	696.48
06H	Fa F5#	739.99	B0H	739.76
07H	So G5	783.99	A6H	784.06
08H	So G5#	830.61	9CH	828.72
09H	La A6	880.00	94H	878.78
0AH	La A6#	932.33	8BH	928.63
0BH	Si B6	987.77	83H	984.49
0CH	Do C6	1046.50	7CH	1047.50
0DH	Do C6#	1108.73	75H	1109.64
0EH	Silent tone	0.0	FFH	0.0
0FH	Silent tone	0.0	FFH	0.0

Table 4-4. Relationship between Tone Length Data and Musical Scale Output

Tone Length Data	Musical Scale Output Time T = 100 ms
00H	T × 1
01H	T × 2
02H	T × 3
03H	T × 4
04H	T × 5
05H	T × 6
06H	T × 7
07H	T × 8
08H	T × 9
09H	T × 10
0AH	T × 11
0BH	T × 12
0CH	T × 13
0DH	T × 14
0EH	T × 15
0FH	Repeat

(1) Explanation of program

The program explained here is the one used in **CHAPTER 11 APPLICATION EXAMPLES OF THIS APPLICATION PROGRAM** with some modifications to its input/output interface.

<Registers used>

XA, BC, HL

<RAM used>

RAM can be placed from address 20H of memory bank 0. The performance melody data is allocated at address 00H to address FFH of memory bank 1.

```
MUS:      2 words     ; Performance data address
TONE:     1 word      ; Musical scale data
TONETIM:  1 word      ; Tone length data
TONEFLG:  1 word      ; Flag area
TONE_F:   1-bit EQU   ; If 1, enables musical scale output.
TM0_F:    1-bit EQU   ; If 1, starts timer count.
```

<Nesting>

2 levels (12 words)

<Hardware used>

- Port : Port2.0 (PTO0 output dual-function pin)
- Timer: Basic interval timer, timer/event counter

<Initial setting>

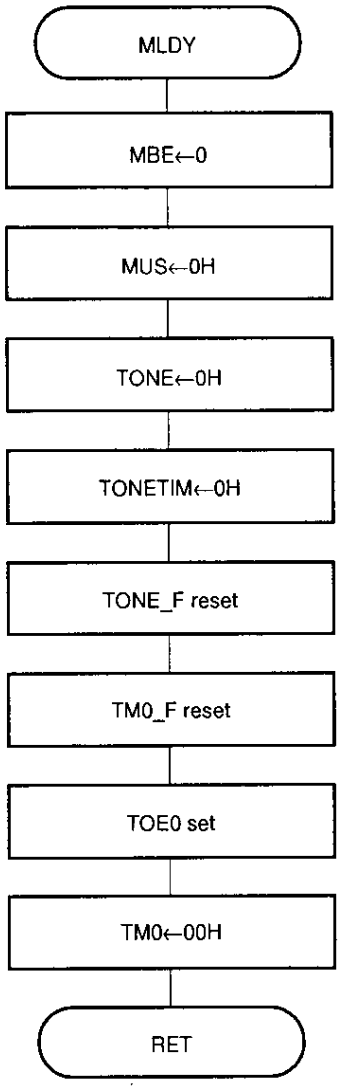
- Clears performance address.
- Enables PTO0 output.
- Stops timer/event counter.

<Start-up procedure>

Melody performance is started when MLDY is initialized and TONE_F = 1 is set, provided that TIM processing is executed every 100 ms and M_OUT processing is always executed.

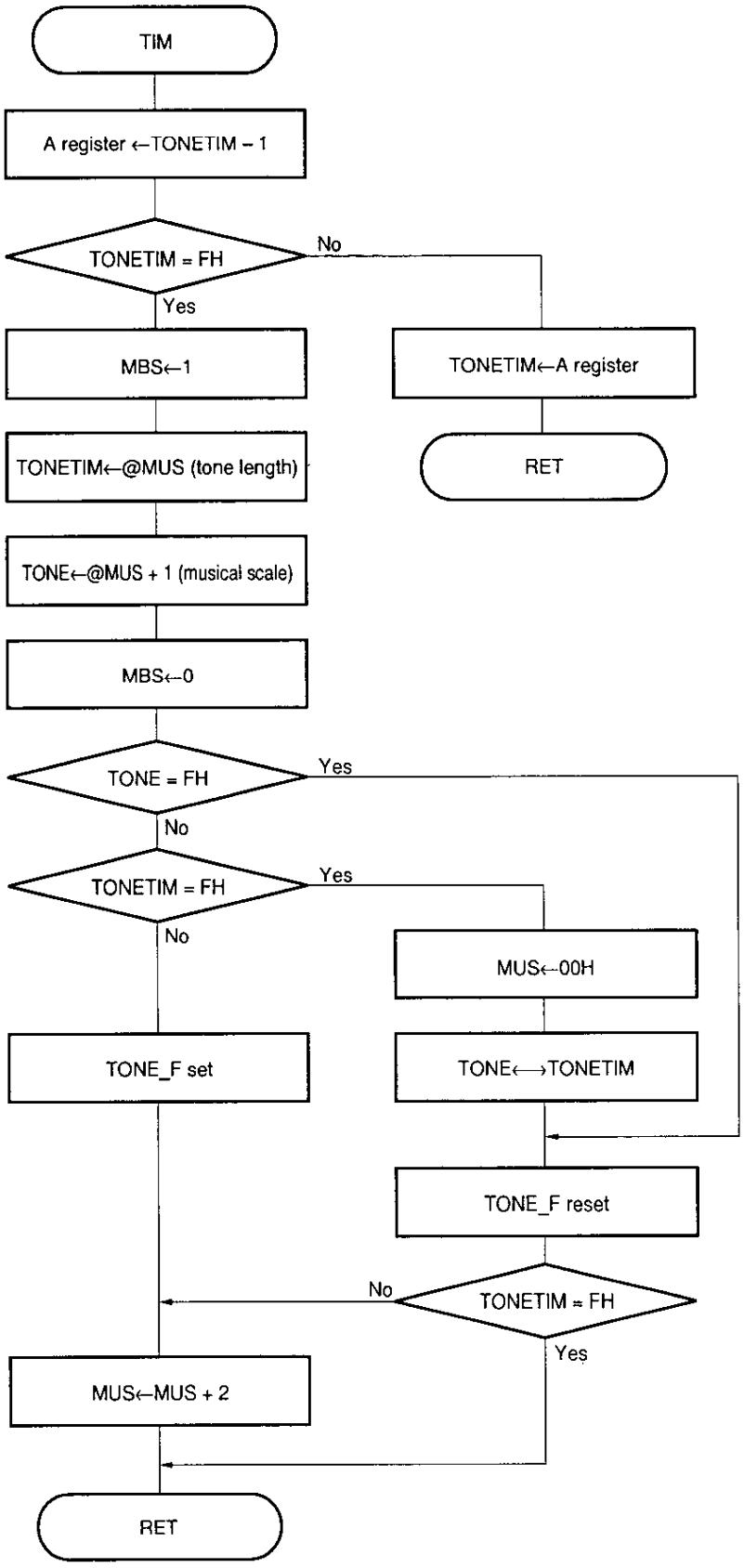
(2) Flowchart

<Initial setting>



Sets output enable flag.

Stops count operation.



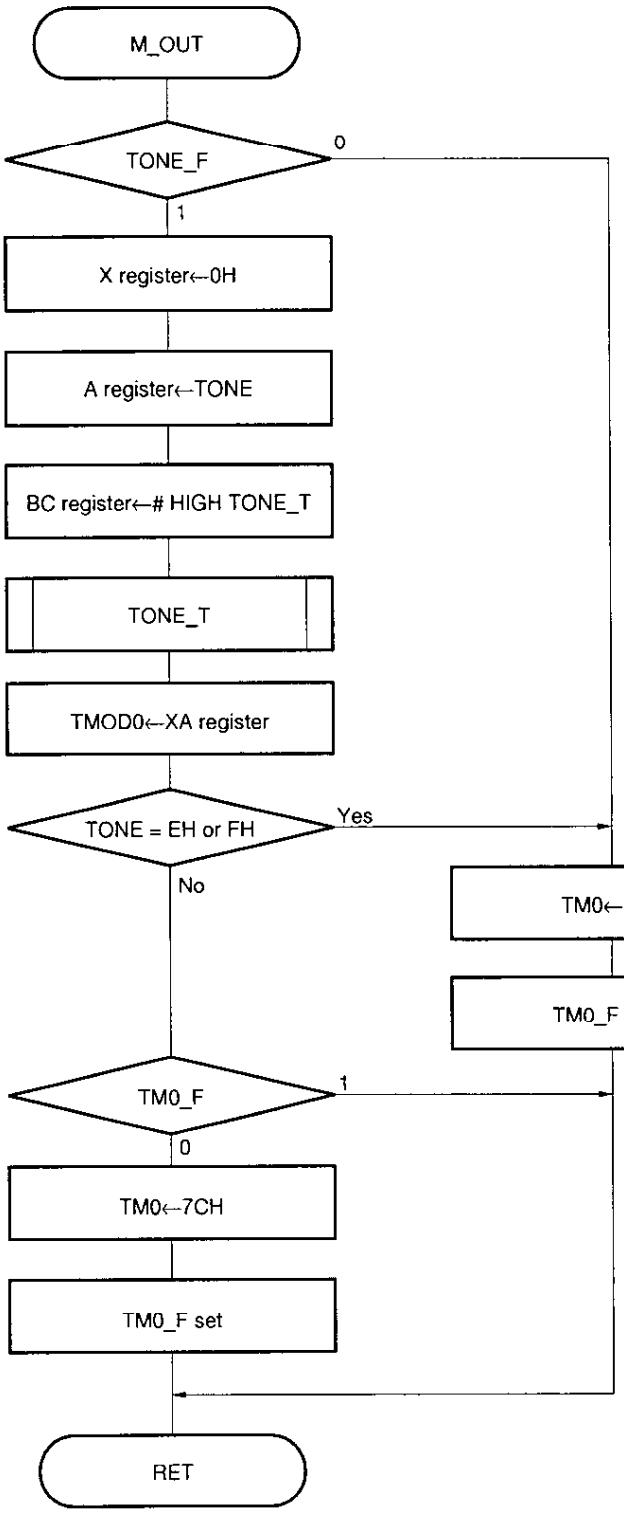
Processes every 100 ms.

Reads tone length data.

Reads musical scale data.

Resets performance address.

Increments (+2) performance address.



If 1, enables musical scale output.

Sets musical scale data in modulo register 0.

Judges with musical scale data whether it is musical scale output or silent tone output.

Starts count operation.

(3) Program example

```
          VENT0   MBE=0,RBE=0,INIT          ; RESET

DSEG0     DSEG    0       AT      20H       ; Stores from address 20H of memory bank 0.

MUS:              DS      2                 ; Performance data address
TONE:             DS      1                 ; Musical scale data
TONETIM:          DS      1                 ; Tone length data
TONEFLG:          DS      1                 ; Flag area

TONE_F            EQU     TONEFLG.0         ; If 1, enables musical scale output.
TM0_F             EQU     TONEFLG.1         ; If 1, starts timer count.
```

;<Subroutine initial setting>

```
          CLR1    MBE                       ; MBE ← 0

          MOV     XA,#00H                   ; Sets RAM.
          MOV     MUS,XA
          MOV     TONE,A
          MOV     TONETIM,A
          MOV     TONEFLG,A
          SET1    TOE0                      ; Sets output enable flag.
          MOV     XA,#00H
          MOV     TM0,XA                    ; Stops oscillation of timer/event counter.
```

```
;<Subroutine timer processing>
        MOV     A,TONETIM
        DECS    A
        BR      TIM_116
        MOV     XA,MUS

        SET1    MBE                     ; MBE ← 1
        SEL     MB1                     ; Memory bank ← 1
        MOV     HL,XA
        MOV     A,@HL+
        NOP
        MOV     B,A
        MOV     A,@HL
        MOV     C,A
        SEL     MB0                     ; Memory bank ← 0
        CLR1    MBE                     ; MBE ← 0

        MOV     A,B
        MOV     TONETIM,A               ; Tone length data ← @performance data address
        MOV     A,C
        MOV     TONE,A                  ; Musical scale data ← @performance data
                                          address + 1
        MOV     A,#0H
        MOV     T_TIMC,A
        SKE     C,#0FH
        BR      TIM_115
        BR      TIM_114

TIM_115:
        SKE     B,#0FH
        BR      TIM_111
        BR      TIM_112

TIM_112:
        MOV     XA,#0H
        MOV     MUS,XA
        MOV     A,TONE
        MOV     TONETIM,A               ; TONETIM ← repeat time
        MOV     A,#0FH
        MOV     TONE,A                  ; TONE ← FH
TIM_114:
        CLR1    TONE_F
        SKE     B,#0FH
        BR      TIM_113
        RET

TIM_111:
        SET1    TONE_F
TIM_113:
        MOV     XA,MUS
        ADDS    XA,#2H
        NOP
        MOV     MUS,XA                  ; Performance data address + 2
        RET
```

```
TIM_116:
        MOV     TONETIM,A               ; Tone length data – 1
        RET

    ; <Subroutine melody output processing>

TONEOUT:
        SKT     TONE_F                  ; If 1, enables musical scale output.
        BR      TONE0
        MOV     X,#0H
        MOV     A,TONE
        MOV     BC,#HIGH TONE_T
        CALL    !TABLE
        MOV     TMOD0,XA                ; Modulo register ← musical scale data
        MOV     A,TONE
        ADDS    A,#2H
        BR      TONE1
TONE0:
        MOV     XA,#0H                  ; Stops timer count.
        CLR1    TM0_F                   ; If 0, starts timer count.
        BR      TONE2
TONE1:
        SKF     TM0_F
        RET
        MOV     XA,#7CH                 ; Timer count start
        SET1    TM0_F                   ; If 1, stops timer count.
TONE2:
        MOV     TM0,XA
        RET
```

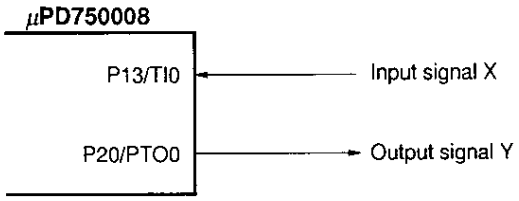
```
TABLE   CSEG    PAGE

        DB      0F9H    ; DO
        DB      0EBH    ; DO#
        DB      0DEH    ; RE
        DB      0D1H    ; RE#
        DB      0C6H    ; MI
        DB      0BAH    ; FA
        DB      0B0H    ; FA#
        DB      0A6H    ; SO
        DB      09CH    ; SO#
        DB      094H    ; LA+
        DB      08BH    ; LA#+
        DB      083H    ; TI+
        DB      07CH    ; DO+
        DB      075H    ; DO#+
        DB      0FFH    ; Silent tone
        DB      0FFH    ; Silent tone

TABLE:
        MOVT    XA,@BCXA
        RET
```

4.2.2 Divided event pulse output

The pulse input from the TI0 pin is selected as the count pulse of the timer/event counter, and divided by N, and output to the P20/PTO0 pin. However, if the value set to the modulo register is M, N can be obtained by the following formula:

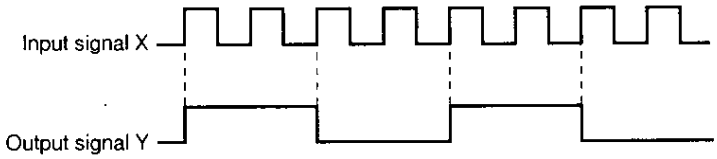
$N = 2 (M + 1)$

Figure 4-4. Divided Event Pulse Output

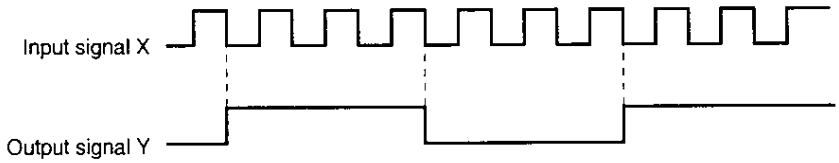


The relationship between input signal X and output signal Y are as described below.

(a) When dividing by 4 (when M = 1, TM0 ← 0CH)



(b) When dividing by 6 (when M = 2, TM0 ← 1CH)



The following program counts the pulse input from the TI0 pin at the rising edge to divide the frequency by 6 and outputs the pulse to the P20/PTO0 pin.

This program is not used in **CHAPTER 11 APPLICATION EXAMPLES OF THIS APPLICATION PROGRAM.**

(1) Program description

<External reference declaration symbol>

M

<Registers used>

XA

<Nesting>

1 level (4 words)

<Hardware used>

- Port : Port2.0 (shared with PTO0 output)
- Timer: Timer/event counter

<Initial setting>

- Port mode: Port2 ← Output mode

<Start-up procedure>

Set the timer/event counter to the event counter mode, and enable TOUT, output then Port2.0 ← 0.

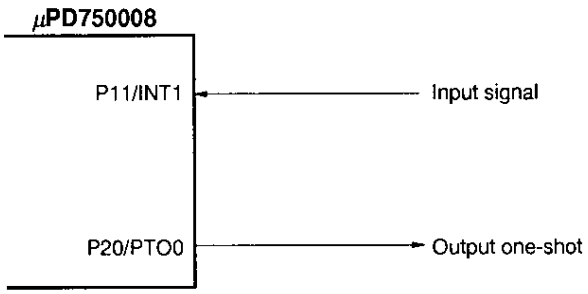
(2) Program example

```
PULSE    CSEG    INBLOCK

         CLR1    MBE
         MOV     XA,#M
         MOV     TMOD0,XA              ; Sets modulo mode register
         MOV     XA,#00001100B
         MOV     TM0,XA                ; Sets timer mode register
         SET1    TOE0                  ; Enables TOUT
         CLR1    PORT2.0               ; Port2.0 ← 0
         RET
```

4.3 One-shot Pulse Output

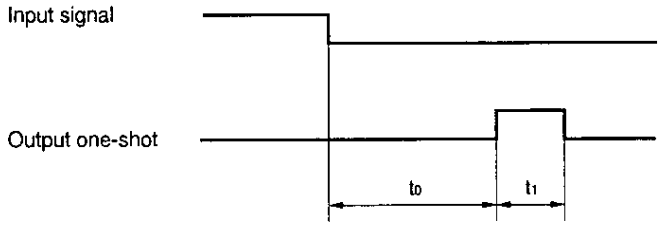
For this application, the PTO pin of the timer/event counter is used to output a one-shot output. Output timing is made by the edge detection of the P11/INT1 pin.

Figure 4-5. One-shot Pulse Output



For this program, a low level is output to the PTO0 pin at the falling edge of the INT1 input, and a high level is output after $(1/16.4 \times 10^3) \times (N_0 + 1)$ seconds. $(1/16.4 \times 10^3) \times (N_1 + 1)$ seconds after the output of the PTO0 pin is changed to a high level, a low level is again output (N_0 = 1 to 255, N_1 = 1 to 255).

Figure 4-6. One-shot Pulse Output Timing (1)

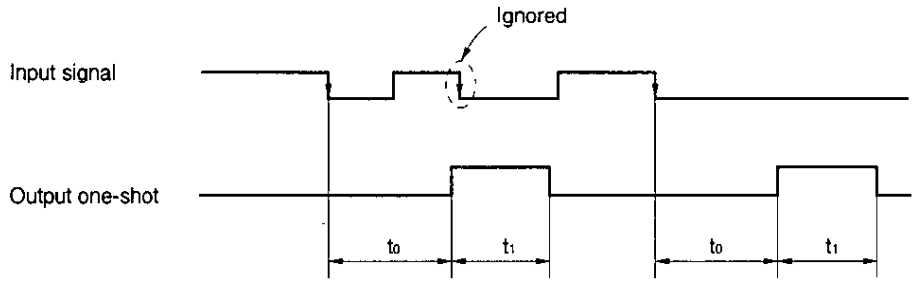


$$t_0 = \frac{1}{16.4} \times (N_0 + 1) \quad \text{(ms)}$$

$$t_1 = \frac{1}{16.4} \times (N_1 + 1) \quad \text{(ms)}$$

However, even if a falling edge is input again before the one-shot output from the input signal falling edge detection is terminated, this edge is ignored.

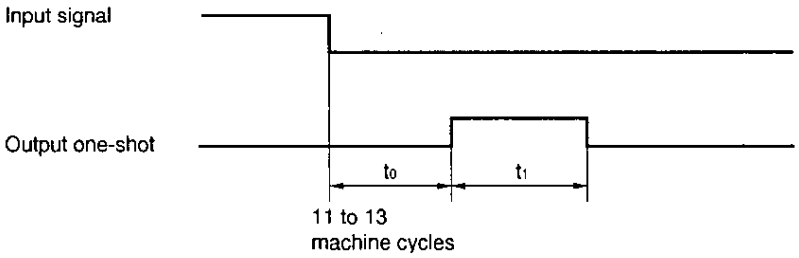
Figure 4-7. One-shot Pulse Output Timing (2)



(1) Output time performance

The time period actually elapsed from the falling edge detection of the input signal to outputting a one-shot is between 11 to 13 machine cycles longer than the time determined by the value set to the modulo register (refer to **Figure 4-8**). The resolution will be $1/16.4 \times 10^3$ sec. $= 6.1 \times 10^{-5}$ sec.

Figure 4-8. One-shot Pulse Output Timing (3)



(2) Program description

This program is not used in **CHAPTER 11 APPLICATION EXAMPLES OF THIS APPLICATION PROGRAM.**

<External reference declaration symbol>

N0, N1

<Registers used>

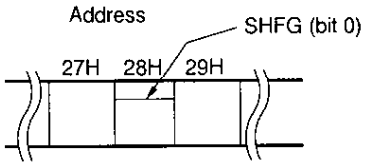
XA

<RAM area used>

1 bit RAM is used for a flag, and the RAM area can be allocated to addresses 08H to 7FH in memory bank 0.

SHFG: 1 bit This flag is set when the delay time has elapsed and a high level is output to the PTO0 pin.

Figure 4-9. RAM Area Layout Used in This Program



<Hardware used>

- Port : Port2.0 (pin shared with PTO0 output), Port1.1 (pin shared with INT1)
- Timer: Timer/event counter

<Interrupt>

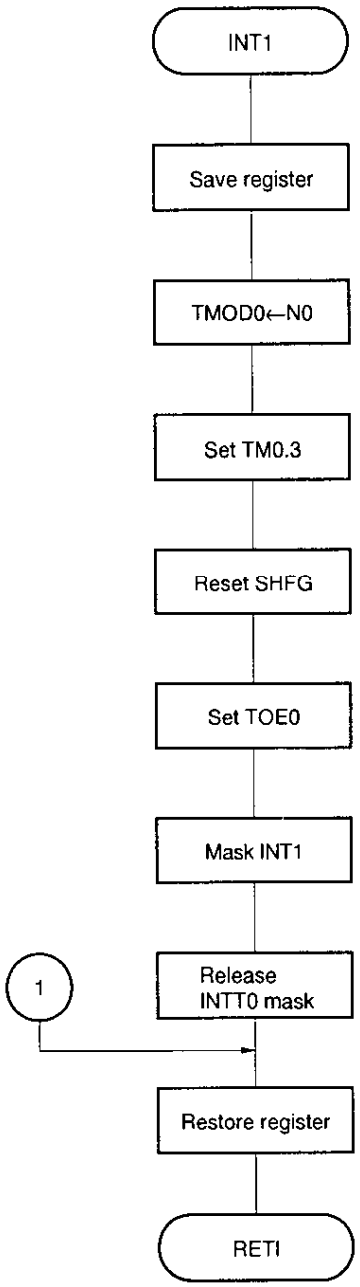
INT1, INTT0

<Initial setting>

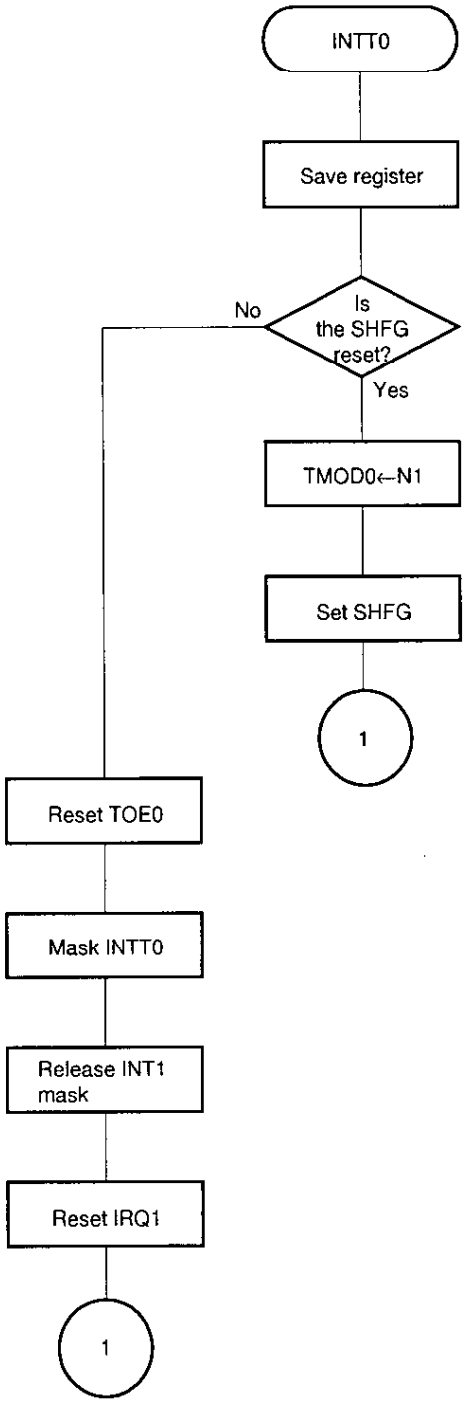
- RAM : SHFG ← 0
- Port output : Port2.0 ← 0
- Port mode : Port2 ← Output mode
- Timer : TM0 ← 5CH
- Interrupt : Enables INT1 interrupt
- INT1 : Falling edge detection mode

(3) Flow chart

INT1 processing



INTT0 processing



Set the output time

Disable TOUT output
(P20/PTO0 ← Low level)

Cancels the falling edge detection performed by the P11/INT1 pin during one-shot output.

(4) Program example

```
          VENT3     MBE=0,RBE=0,INT1

          VENT4     MBE=0,RBE=0,INTT0

DSHOT     DSEG      AT        28H             ; Defines data memory

FLAG:     DS        1H

SHFG      EQU       FLAG.0

          PUBLIC    SHFG

          EXTRN     N0,N1

        ;<Subroutine INT1 processing (MBE = 0)>
INT1      CSEG       INBLOCK

          PUSH      XA
          MOV       XA,#N0
          MOV       TMOD0,XA                ; Sets delay time
          SET1      TM0.3
          CLR1      SHFG
          SET1      TOE0                    ; Enables TOUT output
          DI        IE1                     ; Disables INT1
          EI        IET0                    ; Enables INTT0
RETURN:
          POP       XA
          RETI

        ; <Subroutine INTT0 processing (MBE = 0)>
INTT0:
          PUSH      XA
          SKF       SHFG
          BR        OVER

          MOV       XA,#N1                  ; Sets output time
          MOV       TMOD0,XA
          SET1      SHFG
          BR        RETURN

OVER:                                       ; Output stop processing
          CLR1      TOE0                    ; Disables TOUT output
          DI        IET0                    ; Disables INTT0
          EI        IE1                     ; Enables INT1
          CLR1      IRQ1
          BR        RETURN
```

CHAPTER 5 CLOCK TIMER APPLICATIONS

The μPD750008 has a built-in clock timer (1 channel) that has the following functions:

<1> The test flag (IRQW) can be set every 0.5 s. The standby mode can be released by IRQW.
<2> An interval of 0.5 s can be generated from either the main system clock or the subsystem clock.
<3> The time interval can be reduced to 1/128 (3.91 ms) by using the advance mode. This is convenient for program debugging or checking.
<4> A fixed frequency of 2.048 kHz can be output to the P23/BUZ pin. This signal can be used for generating a buzzer sound or for trimming the system clock oscillation frequency.
<5> The frequency divider circuit can be cleared by this timer so that the clock can be started from 0 seconds.

5.1 Clock Program

This section introduces a clock count subroutine program which uses the clock timer.

In this program, the clock timer is set to the normal clock mode, and the seconds counter is incremented each time the clock timer interrupt request flag (IRQW) is set. One_minute is timed by incrementing the seconds counter 120 times.

A skip is performed each time the minute counter is counted full after a return.

(1) Program description

This program is not used in **CHAPTER 11 APPLICATION EXAMPLES OF THIS APPLICATION PROGRAM.**

<Public declaration symbol>

CNT10 : Decimal count subroutine entry label
CNT6 : Sexagesimal count subroutine entry label
CNT7 : Base 7count subroutine entry label

<External reference declaration symbol>

SECD : Seconds count address
MIND : Minutes count address
HOURD : Hours count address
DAYD : Days data address

<Registers used>

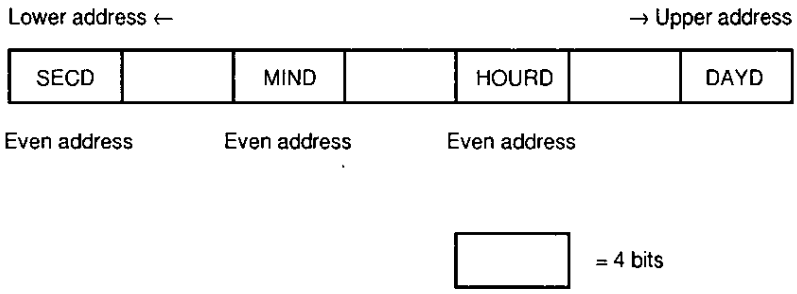
XA, HL

<RAM area used>

Address [H]	Name	Application	Initial Value
External reference	SECD (2 × 4 bits)	Seconds counter (counts from 88H to FFH in binary)	88H
External reference	MIND (2 × 4 bits)	Minutes counter (counts from 0 to 59 in decimal)	0
External reference	HOURD (2 × 4 bits)	Hours counter (counts from 0 to 23 in decimal)	0
External reference	DAYD (1 × 4 bits)	Su M T W T F Sa Days data 0 1 2 3 4 5 6	0

Conditions for RAM address declaration

- The row address of each symbol must be set to the same value. The column address must be set within 0 to 0EH.
- Each symbol must be allocated as follows:



<Nesting>

2 levels (8 words)

<Hardware used>

- Clock timer (Initial setting: Normal clock mode)

<Initial setting>

- Initial value set of the RAM used
- Clock timer mode set, clear, and start

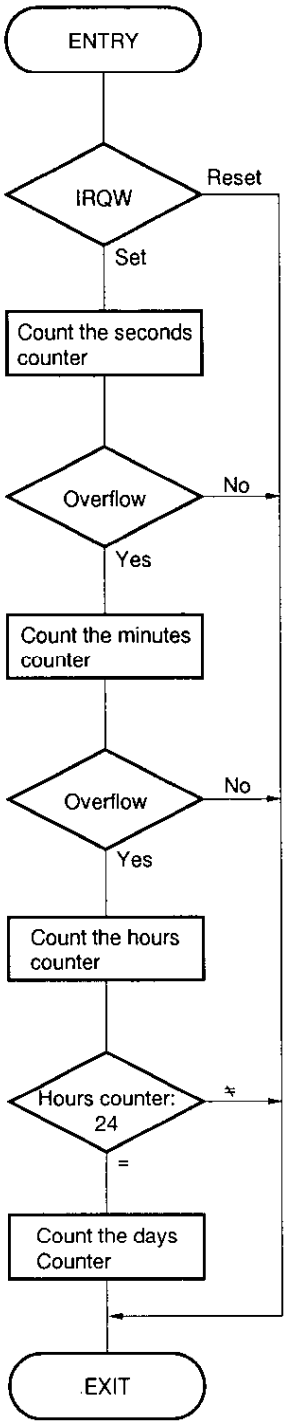
<Start-up procedure>

(i) Call WATCH at least once within 0.5 seconds.

(ii) When WATCH is called, IRQW is checked. One of the following two processes is then performed.

- If IRQW = 0:
  Processing returns to the user program by the RET instruction.
- If IRQW = 1:
  After resetting IRQW to zero, time is counted and the appropriate count is stored in each respective area (seconds, minutes, hours, day of the week). Processing returns to the user program by the RET instruction
  However, if the minutes digit has changed, processing returns to the user program by the RETS instruction.

(2) Flow chart


ENTRY
IRQW
Reset
Set
Count the seconds counter
Overflow
No
Yes
Count the minutes counter
Overflow
No
Yes
Count the hours counter
Hours counter: 24
≠
=
Count the days Counter
EXIT

(3) Program example

```
EXTRN     DATA(SECD,MIND,HOURD,DAYD)
PUBLIC    CNT10,CNT6,CNT7

        ;<Subroutine clock count>

WATCH     CSEG      SENT
          CLR1      MBE
          SKTCLR    IRQW              ; 0.5 s check
          RET
          INCS      SECD              ; Counts seconds
          RET
          INCS      SECD + 1
          RET

          MOV       XA,#100H - 120
          MOV       SECD,XA           ; Counts minutes
          MOV       HL,#MIND
          CALLF     !CNT10
          RETS
          CALLF     !CNT6
          RETS
          CALLF     !CNT10            ; Counts hours
          BR        WATCH2
          INCS      @HL
WATCH2:
          INCS      L
          MOV       XA,HOURD
          SKE       A,#4H
          RETS
          SKE       X,#2H
          RETS
          MOV       XA,#00H
          MOV       HOURD,XA
          CALLF     !CNT7             ; Updates days data
          NOP
          RETS
```

; **<Subroutine base-N count>**

```
CNTN    CSEG    SENT
CNT10:
        MOV     A,#(10H-0AH)        ; Decimal
CNT6:
        MOV     A,#(10H-6)          ; Sexadecimal

CNT7:
        MOV     A,#(10H-7)          ; Base 7
;
        INCS    @HL
        NOP
        ADDS    A,@HL
        BR      NOMRET
        XCH     A,@HL
        INCS    L
        RETS
NOMRET:
        INCS    L
        RET
```

CHAPTER 6 CLOCK OUTPUT CIRCUIT APPLICATION

6.1 PCL Clock Output

The μPD750008 has a built-in clock output circuit. Therefore, a clock can be supplied to peripheral LSIs and slave microcontrollers such as the μPD7500 series microcontroller from the P22/PCL pin. The clock output frequency and output enable/disable are determined by the clock output mode register (CLOM). Figure 6-1 shows the format of the clock output mode register.

Figure 6-1. Clock Output Mode Register Format

Address	3	2	1	0	Symbol
FD0H	CLOM3	0	CLOM1	CLOM0	CLOM

Clock output frequency selection bit
(fx = 6.00 MHz)

0	0	Φ Output[Note] (1.5 MHz, 750 kHz, 375 kHz, 93.8 kHz)
0	1	$fx/2^3$ Output (750 kHz)
1	0	$fx/2^4$ Output (375 kHz)
1	1	$fx/2^6$ Output (93.8 kHz)

(fx = 4.19 MHz)

0	0	Φ Output[Note] (1.05 MHz, 524 kHz, 262 kHz, 65.5 kHz)
0	1	$fx/2^3$ Output (524 kHz)
1	0	$fx/2^4$ Output (262 kHz)
1	1	$fx/2^6$ Output (65.5 kHz)

Note Φ is the CPU clock selected by PCC

Clock output enable/disable bit

0	Disables output
1	Enables output

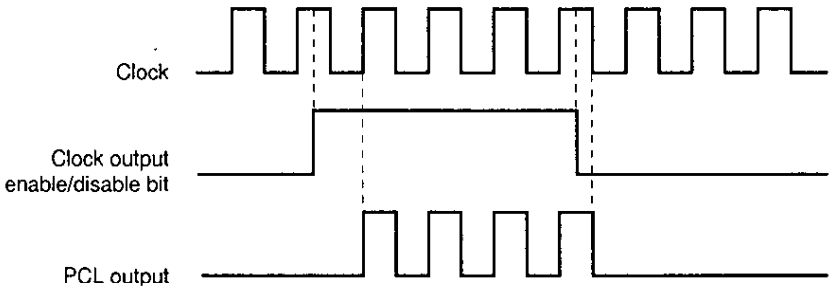
Caution Be sure to write 0 to bit 2 of CLOM.

The following describes the procedure for outputting a clock to the P22/PCL pin.

- Set the clock output frequency using the clock output mode register. The output must be disabled.
- Reset the output latch of Port2.2.
- Set Port2 to the output port mode.
- Enables the clock output.

For the μPD750008, measures are taken to prevent an output glitch when enabling/disabling the clock output as shown in Figure 6-2.

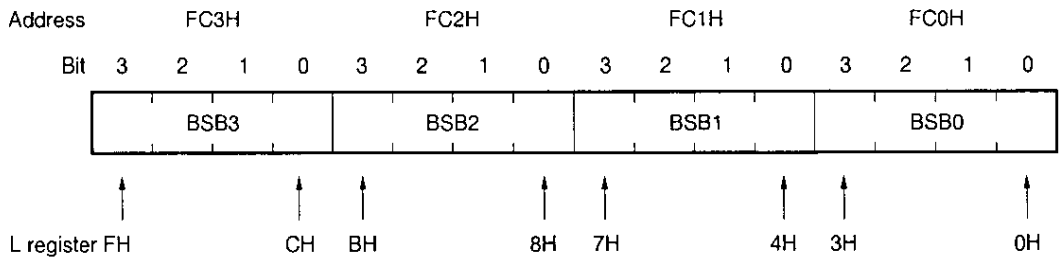
Figure 6-2. Wave Form of Clock Output



CHAPTER 7 BIT SEQUENTIAL BUFFER APPLICATIONS

The bit sequential buffer (BSB) is a special data memory used for bit manipulation. Bit specification can be performed indirectly by using the L register. Therefore, the specification bit in the BSB can be changed simply by incrementing or decrementing the L register.

Figure 7-1. Bit Sequential Buffer Indirect Addressing

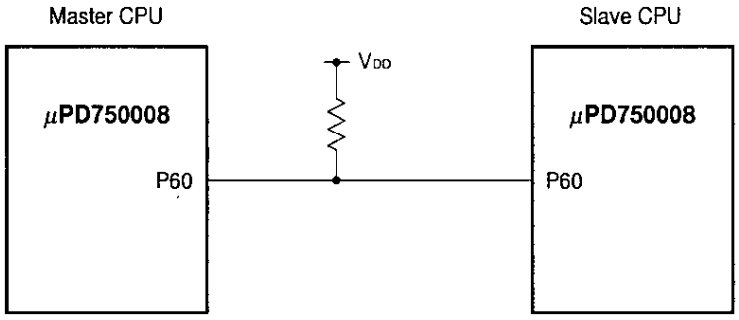


The following shows an example of a BSB.

7.1 High-Speed Serial Data Transfer

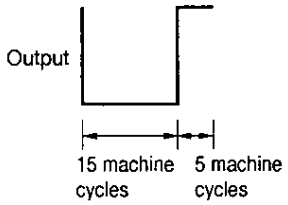
When a multi-processor system is configured using μPD750008s as shown in Figure 7-2, serial data transfer can be performed by using the bit sequential buffer. One signal line is used. In addition to serving as the transmit and receive signal line, this signal line also serves as the $\overline{\text{BUSY}}$ signal output from the slave CPU.

Figure 7-2. Configuration of Multi-Processor System (High-Speed Serial Data Transfer)

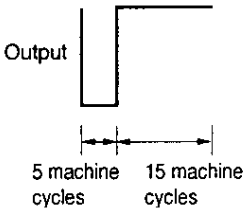


The length of serial data is N (1 to 16 bits). The following shows the format of the serial data.

- When data is 0
  A low level is output for 15 machine cycles,
  then a high level is output for 5 machine cycles.

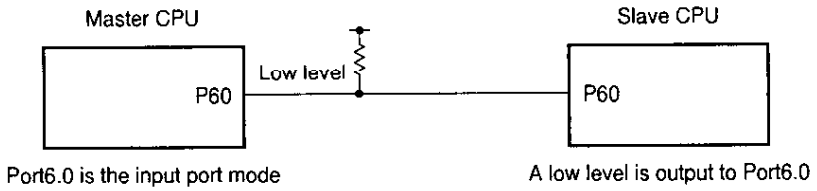


- When data is 1
  A low level is output for 5 machine cycles,
  then a high level is output for 15 machine cycles.



When the $\overline{\text{BUSY}}$ signal from the slave CPU is ON (high level), N bits of data are sent from the master CPU, then N bits of data are sent from the slave CPU. The transfer data clock begins operating at the falling edge of the first data. The following shows the flow of the data transfer operation.

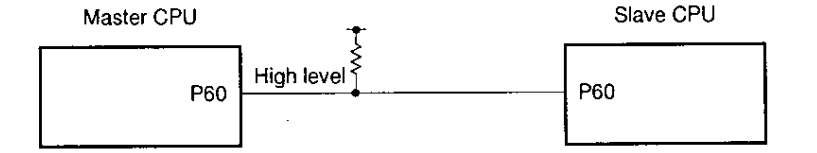
(a) The $\overline{\text{BUSY}}$ signal from the slave CPU is ON



Port6.0 is the input port mode

A low level is output to Port6.0

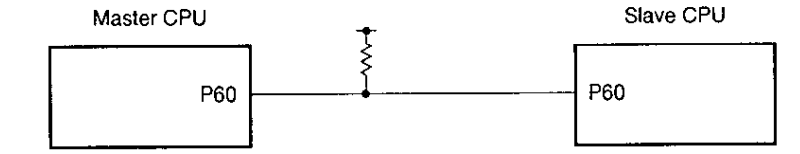
(b) The $\overline{\text{BUSY}}$ signal from the slave CPU is OFF



Port6.0 is the input port mode

Port6.0 is set to input port mode

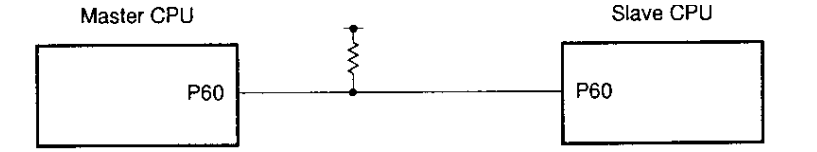
(c) N bits of data are sent from the master CPU



Port6.0 is set to the output port mode, and data is output

Data is input from Port6.0

(d) N bits of data are output from the slave CPU



Port6.0 is set to the input port mode, and data is input.

Port6.0 is set to the output port mode, and data is output.

(1) Program description (Common to master CPU and slave CPU)

This program is not used in **CHAPTER 11 APPLICATION EXAMPLES OF THIS APPLICATION PROGRAM.**

<Public declaration symbol>

TDATA, RDATA

<Registers used>

XA, L

<RAM area used>

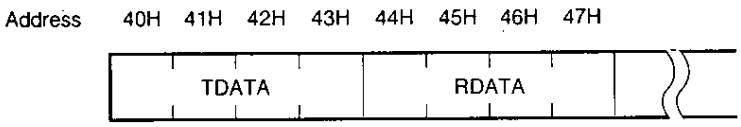
A transmit data area and receive data area are provided. The transfer data is stored to the N bits from the lowest address of these areas.

In RAM, the transmit data area and receive data area can be allocated to addresses 20H to 7FH in memory bank 0. However, the address must be an even address.

TDATA : 4 words : This is the transmit data area. N bits of data are sent from the lowest address.

RDATA : 4 words : This is the receive data area. Receive data are stored to the lowest N bits of this area.

Figure 7-3. RAM layout this program



<Nesting>

1 level (4 words)

<Hardware used>

- Port: Port6.0
- Bit sequential buffers 0 to 3

<Initial setting>

- Master PCC ← 3H
  Port6.0 ← Input mode
- Slave PCC ← 3H
  Port6.0 ← Output mode
  Port6.0 ← 0

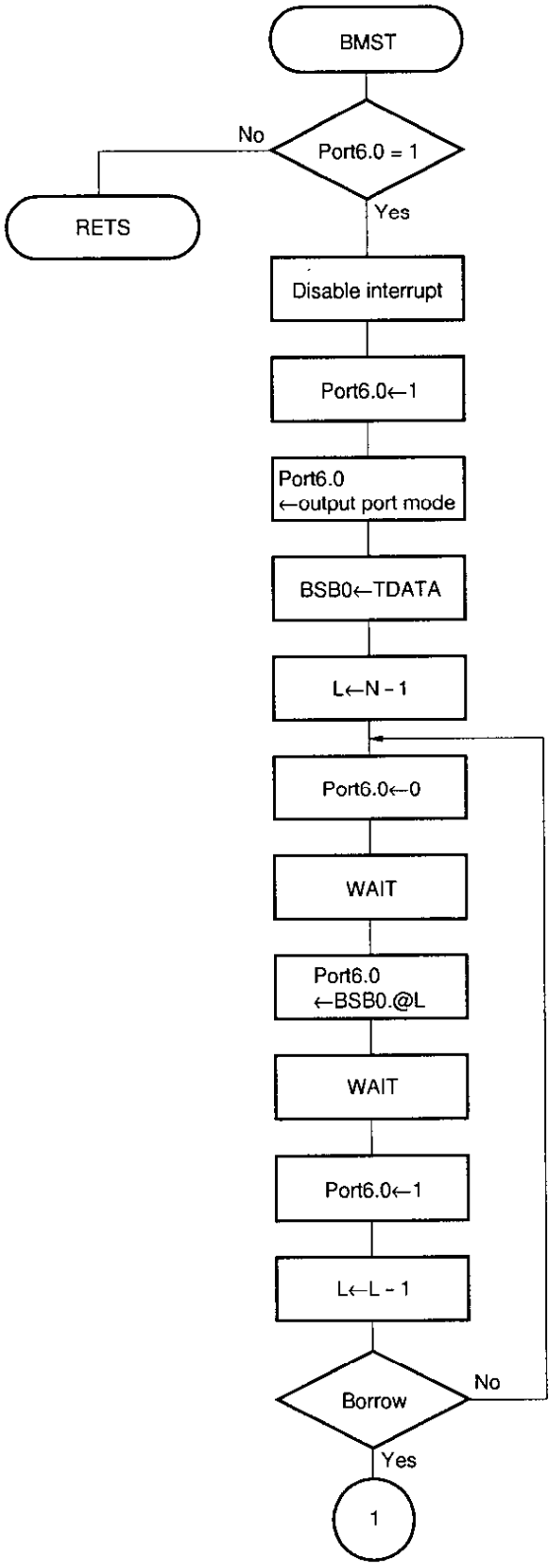
<Main processing>

- Master CALL !BMST
- Slave CALL !BSLV

(2) Master CPU

(a) Flow chart

Transfer processing

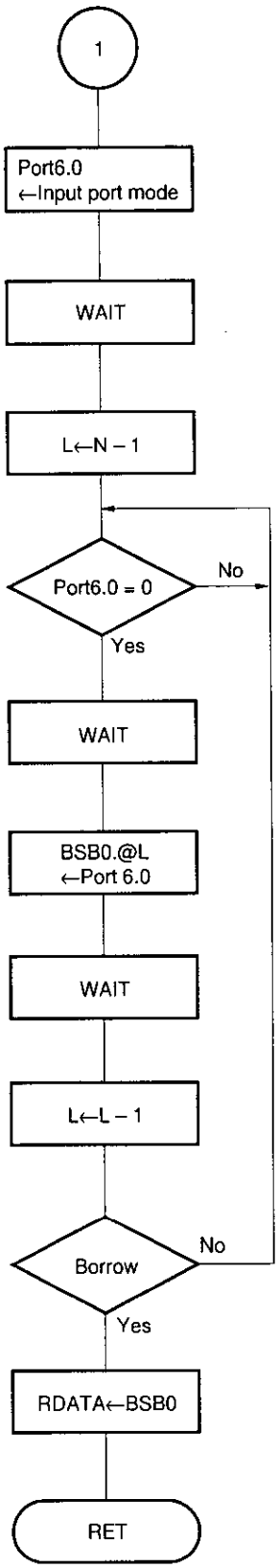


Transfer operation is enabled when Port6.0 is set to high level.

Sets transfer data to the bit sequential buffer.

Outputs transmit data 5 μs after a low level is output to port6.0.

Outputs a high level 10 machine cycles after the data is output.



Clocks in data approximately 10 μs after a low level is input from Port6.0.

Stores input data to RDATA.

(b) Program example

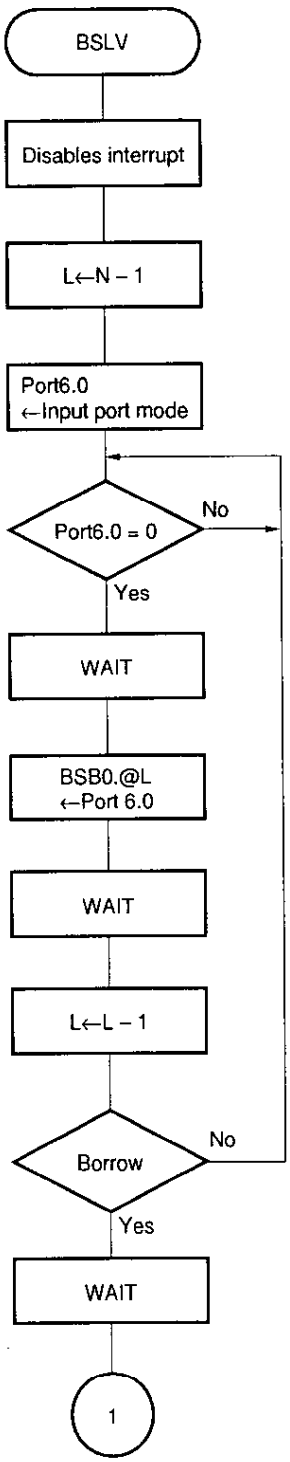
```
        PUBLIC    TDATA,RDATA

DBSIO   DSEG      0         AT      40H

TDATA:  DS        4H                          ; Transmit data area
RDATA:  DS        4H                          ; Receive data area

N       EQU       8
```

```
BMST    CSEG    INBLOCK

        CLR1    MBE
        SKT     PORT6.0
        RETS

        DI
        SET1    PORT6.0             ; Port6.0 ← High level
        MOV     XA,#00010000B       ; Sets Port6.0 to the output mode
        MOV     PMGA,XA
        MOV     XA,TDATA            ; Sets transmit data to BSB
        MOV     BSB0,XA
        MOV     XA,TDATA+2
        MOV     BSB2,XA
        MOV     L,#N-1
LOOP:
        CLR1    PORT6.0             ; Port6.0 ← Low level
        NOP
        SKF     BSB0.@L             ; Port6.0 ← BSB0.@L
        SET1    PORT6.0
        MOV     A,#0EH
WAIT:
        INCS    A
        BR      WAIT
        NOP
        SKT     PORT6.0
        SET1    PORT6.0             ; Port6.0 ← High level
        DECS    L
        BR      LOOP

        MOV     XA,#00000000B       ; Sets Port6.0 to the input mode
        MOV     PMGA,XA

        NOP
        NOP
        NOP

        MOV     L,#N-1
CHECK:
        SKF     PORT6.0             ; Checks the falling edge of receive data
        BR      CHECK
        NOP
        SET1    BSB0.@L
        SKT     PORT6.0             ; Clocks in data
        CLR1    BSB0.@L
        NOP
        NOP
        NOP
        DECS    L
        BR      CHECK

        MOV     XA,BSB0             ; Stores input data to RDATA
        MOV     RDATA,XA
        MOV     XA,BSB2
        MOV     RDATA+2,XA
        RET
```

(3) Slave CPU

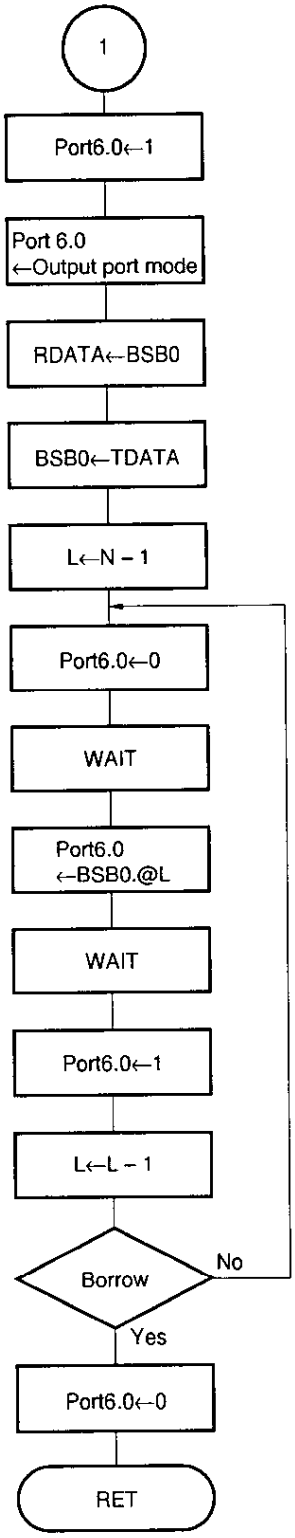
(a) Flow chart

Transfer processing



Checks the falling edge of the signal sent from the master

Clocks in data approximately 1.0 μs after a low level is input from Port6.0.



Saves the input data to RDATA.

Sets transfer data to the bit sequential buffer.

Outputs transmit data 5 μs after a low level is output to Port6.0.

Sets the $\overline{\text{BUSY}}$ signal to ON

(b) Program example

```
        PUBLIC    TDATA,RDATA

DBSIOS  DSEG      0         AT      40H

TDATA:  DS        4H                          ; Transmit data area
RDATA:  DS        4H                          ; Receive data area

N       EQU       8
```

```
BSLV    CSEG    INBLOCK

        DI
        CLR1    MBE
        MOV     L,#N-1
        MOV     XA,#00000000B       ; Sets Port6.0 to the input mode
        MOV     PMGA,XA
CHECK:
        SKF     PORT6.0             ; Checks the rising edge of the receive signal
        BR      CHECK
        NOP
        SET1    BSB0.@L             ; Clocks in data from Port6.0
        SKT     PORT6.0
        CLR1    BSB0.@L
        NOP
        NOP
        NOP
        DECS    L
        BR      CHECK

        NOP
        NOP

        SET1    PORT6.0             ; Port6.0 ← High level
        MOV     XA,#10H             ; Sets Port6.0 to the output mode
        MOV     PMGA,XA

        MOV     XA,BSB0             ; Stores receive data to RDATA
        MOV     RDATA,XA
        MOV     XA,BSB2
        MOV     RDATA+2,XA
        MOV     XA,TDATA
        MOV     BSB0,XA
        MOV     XA,TDATA+2
        MOV     BSB2,XA
        MOV     L,#N-1

LOOP:
        CLR1    PORT6.0             ; Port6.0 ← Low level
        NOP
        SKF     BSB0,@L             ; Outputs data to Port6.0
        SET1    PORT6.0
        MOV     A,#0EH
WAIT:
        INCS    A
        BR      WAIT
        NOP
        SKT     PORT6.0
        SET1    PORT6.0             ; Port6.0 ← High level
        DECS    L
        BR      LOOP

        CLR1    PORT6.0
        RET
```

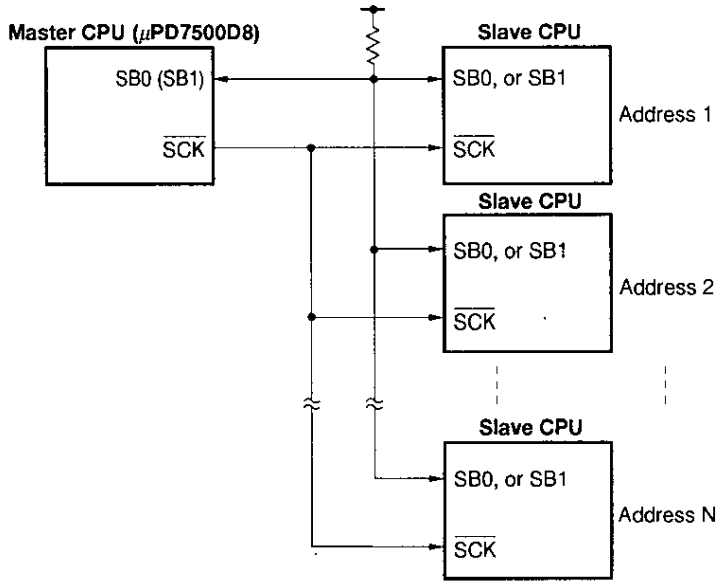
[MEMO]

CHAPTER 8 SERIAL INTERFACE APPLICATIONS

8.1 Application Example in SBI Mode

The following introduces an example in which the serial interface of the μPD750008 is operated in the SBI mode. Figure 8-1 shows a typical serial bus configuration in the SBI mode.

Figure 8-1. Typical Configuration of Serial Bus Interface



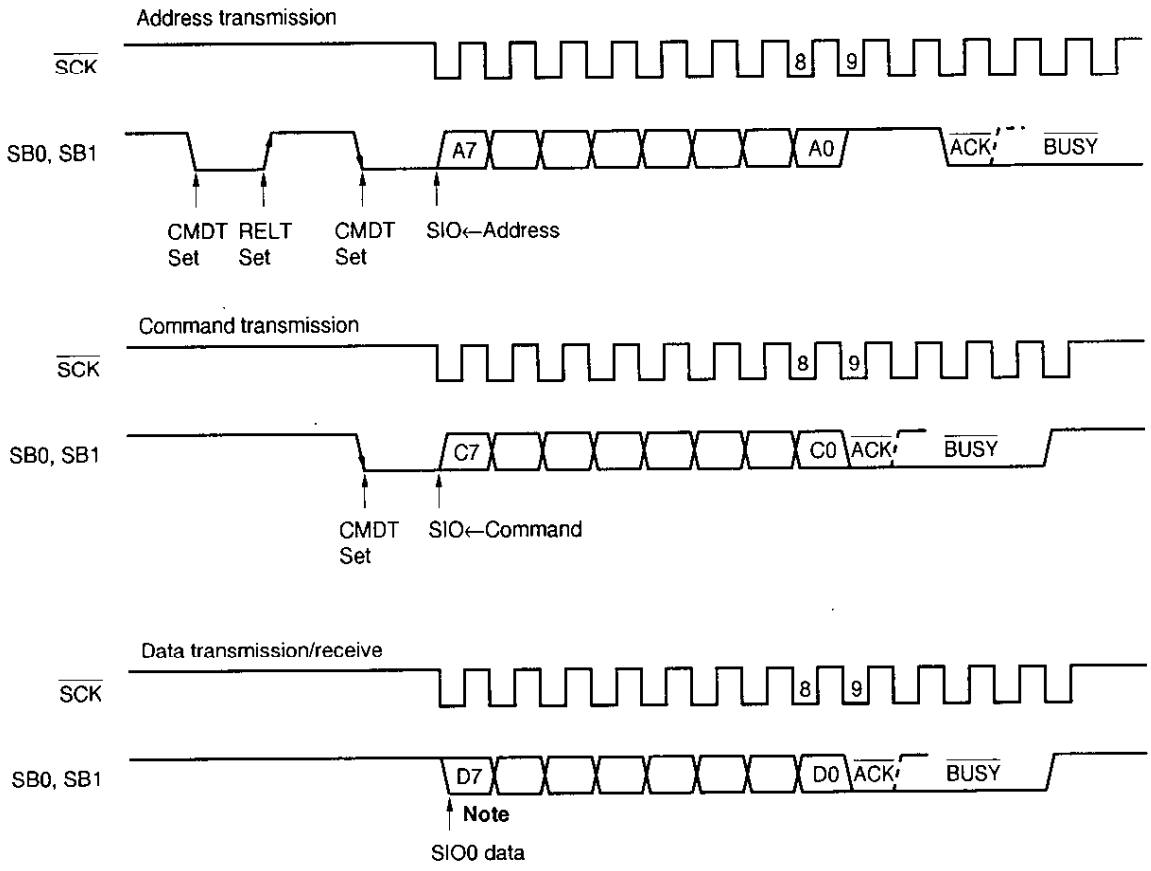
(1) Application as a master CPU

The master CPU performs the following processing.

(a) Address transmission ··· Selects the slave to communicate with.
(b) Command transmission
(c) Data transmission/receive } ··· Performs data communication with the slave selected in (a) using commands and data

Figure 8-2 shows the address, command, and data formats.

Figure 8-2. Address, Command, and Data Format

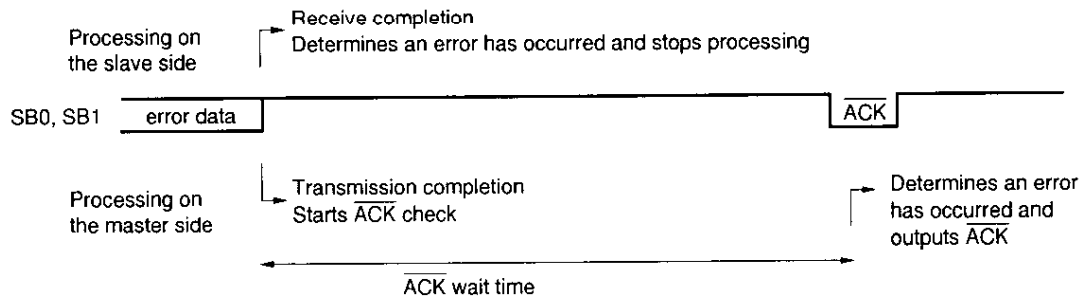


Note Write 0FFH to receive data

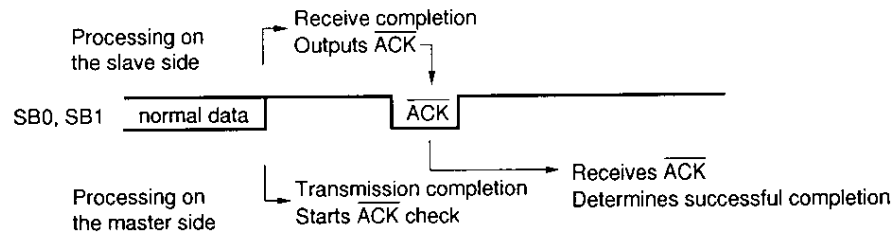
An error generated on the slave CPU side is checked by the acknowledge signal ($\overline{ACK}$) returned from the slave CPU.

Figure 8-3. Error Check using the Acknowledge Signal

- **When an error is generated**



- **When normal**



After 1-byte (8-bit) data transfer has been completed (INTCSI has been generated), the master CPU checks wheter the $\overline{ACK}$ signal has been received from the slave CPU.

If the $\overline{ACK}$ signal is not received from the slave CPU within a certain time after the completion of the transfer, the master CPU determines that an error has occurred on the slave CPU side.

When sending an address, command, and data, the data i written to the SIO, and at the same time, the same data is written to the slave address register to check if the transmit data has changed on the bus line or not.

(2) Explanation of program

This program is an example of transferring data indicating musical scales and tone length to the display driver. The program explained here is the one used in **CHAPTER 11 APPLICATION EXAMPLES OF THIS APPLICATION PROGRAM** with some modifications to its input/output interface.

<Registers used>

Register bank 0: XA
Register bank 3: XA, BC

<RAM used>

RAM is placed from address 20H of memory bank 0.

MODESBI:	4 words	; Indicates which data is being transferred.
LCD:	16 words	; Transfer data area
REFRES_F:	1 bit	; If 1, transfer start or error

<Nesting>

1 level (6 words)

<Hardware used>

Port: Port0.1 ($\overline{SCK}$ dual-function pin), Port0.2 (SB0 dual-function pin)
Serial interface

<Interrupt>

INTCSI

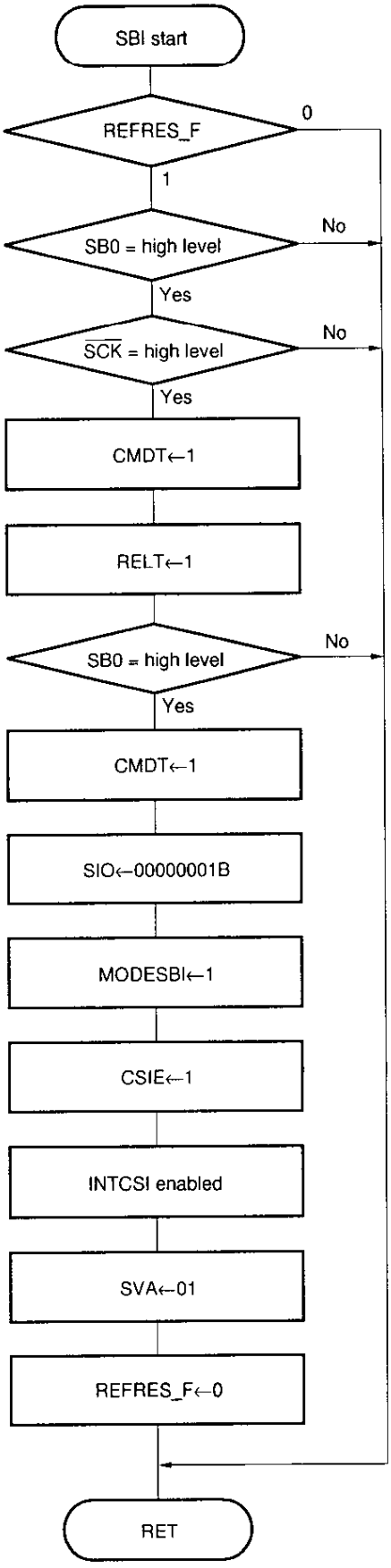
<Initial setting>

- Specifies connection of on-chip pull-up resistor of port used.
- Serial operating mode

<Start-up procedure>

After REFRES_F is set (1), if SBI is started, data of 16 words from the start of LCD is transferred to address 1H of the slave CPU.
When transfer is completed normally, REFRES_F is reset (0).
If an error occurs, data is transferred again with REFRES_F fixed at 1.

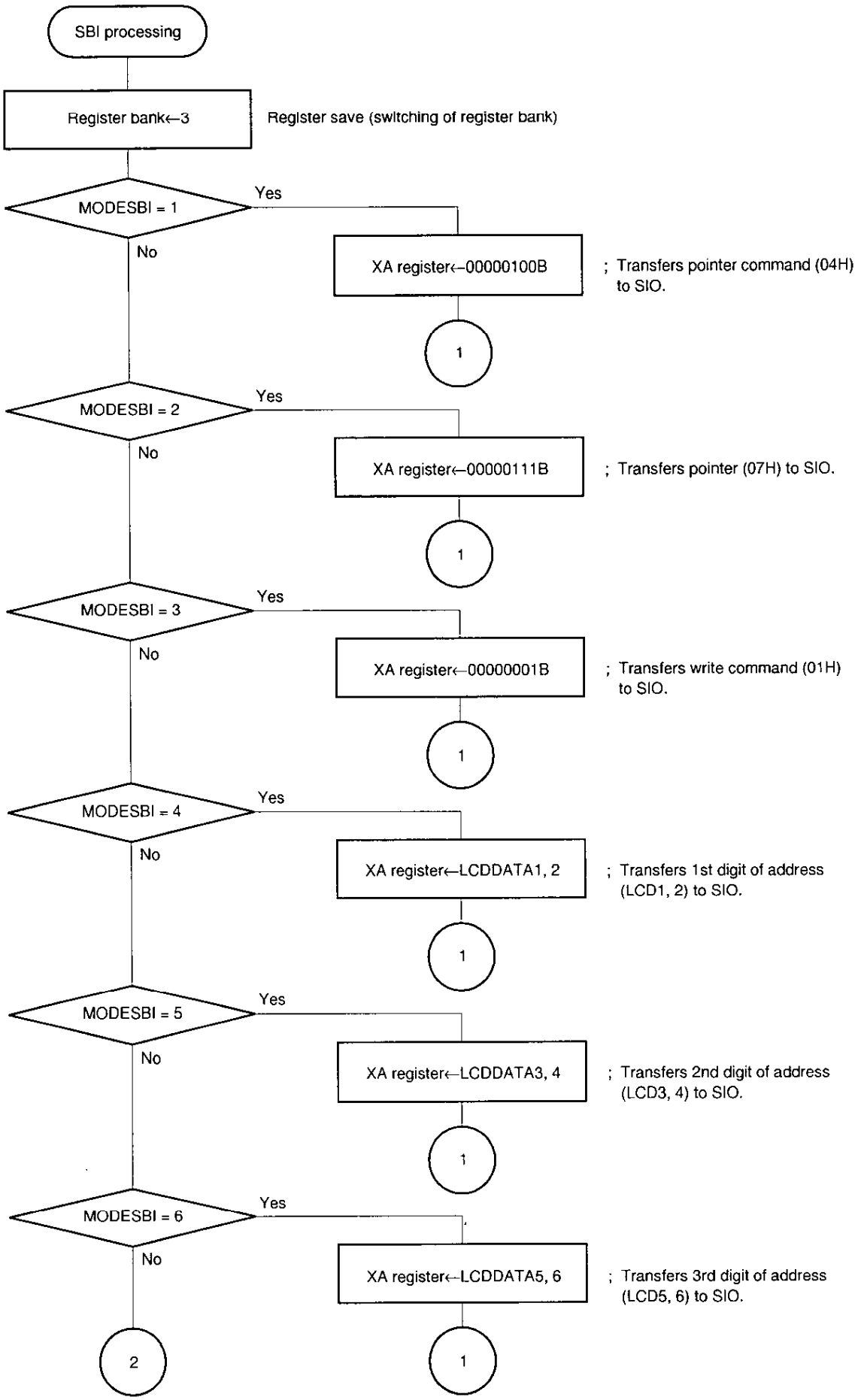
(3) Flowchart

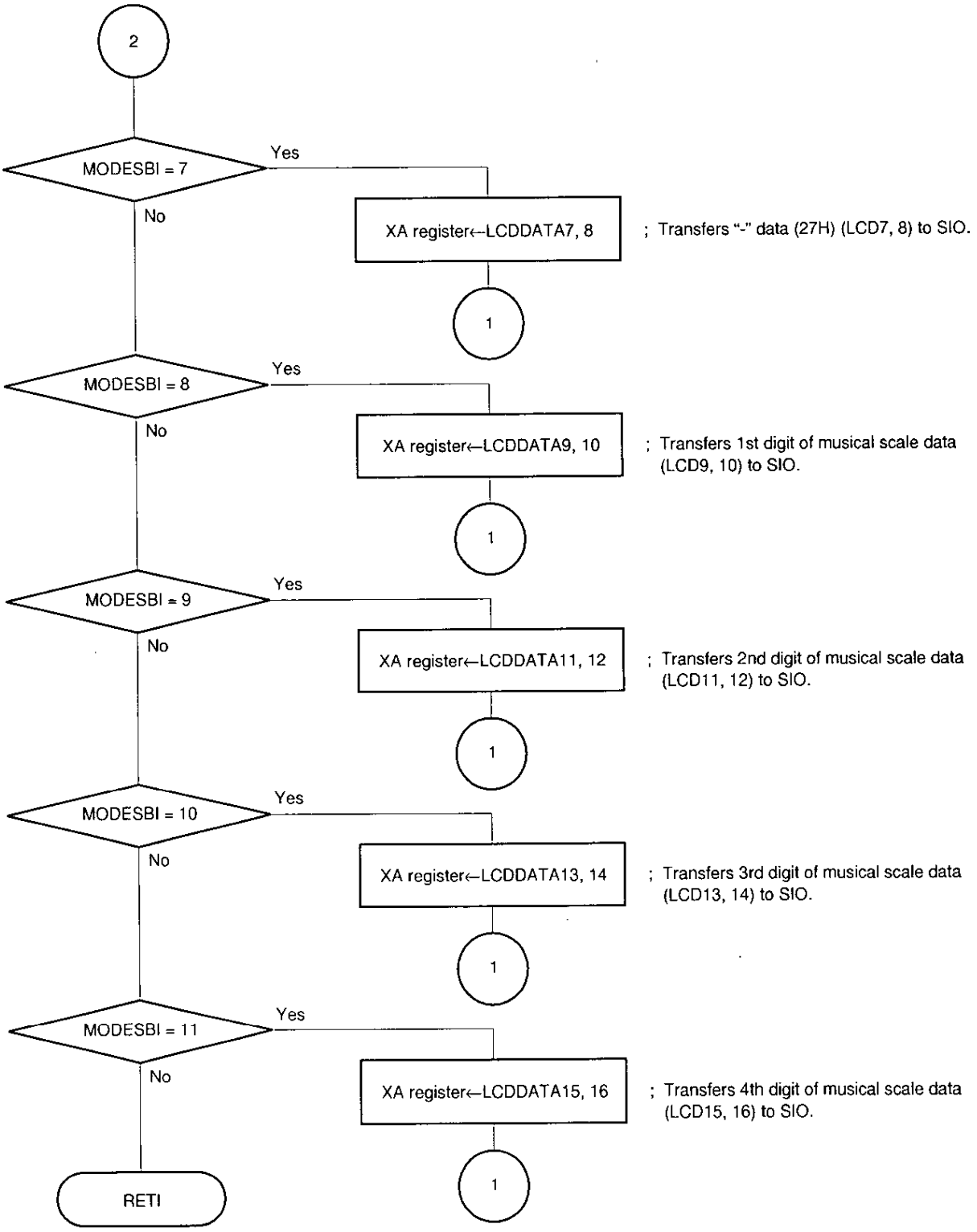


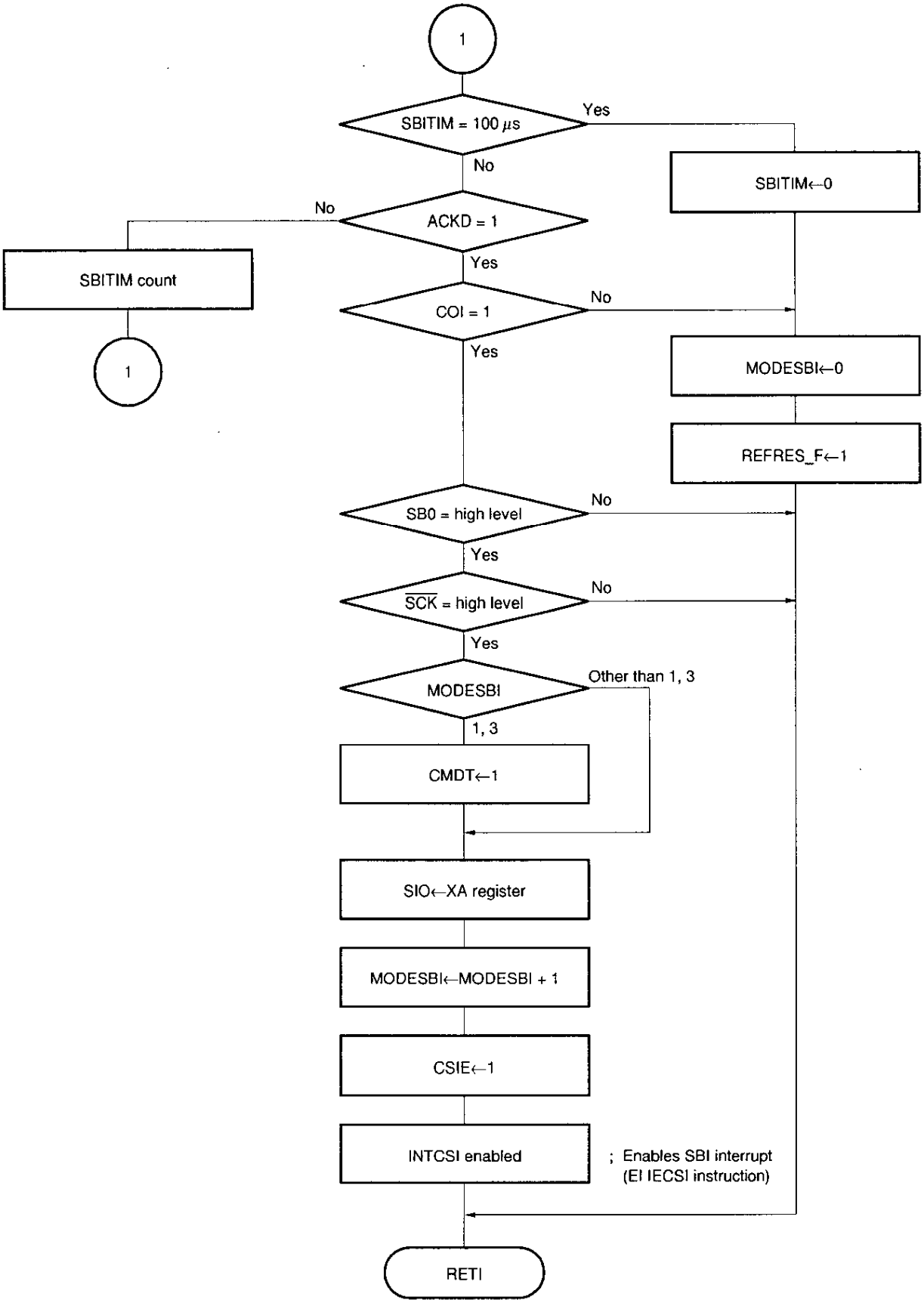
Transfers slave address (01H) to SIO.

Enables SBI interrupt (EI IECSI instruction).

If 1, refreshes LCD.


SBI processing
Register bank←3
Register save (switching of register bank)
MODESBI = 1
Yes
No
XA register←00000100B
; Transfers pointer command (04H) to SIO.
1
MODESBI = 2
Yes
No
XA register←00000111B
; Transfers pointer (07H) to SIO.
1
MODESBI = 3
Yes
No
XA register←00000001B
; Transfers write command (01H) to SIO.
1
MODESBI = 4
Yes
No
XA register←LCDDATA1, 2
; Transfers 1st digit of address (LCD1, 2) to SIO.
1
MODESBI = 5
Yes
No
XA register←LCDDATA3, 4
; Transfers 2nd digit of address (LCD3, 4) to SIO.
1
MODESBI = 6
Yes
No
XA register←LCDDATA5, 6
; Transfers 3rd digit of address (LCD5, 6) to SIO.
2
1


2
MODESBI = 7
Yes
No
XA register←LCDDATA7, 8
; Transfers "-" data (27H) (LCD7, 8) to SIO.
1
MODESBI = 8
Yes
No
XA register←LCDDATA9, 10
; Transfers 1st digit of musical scale data (LCD9, 10) to SIO.
1
MODESBI = 9
Yes
No
XA register←LCDDATA11, 12
; Transfers 2nd digit of musical scale data (LCD11, 12) to SIO.
1
MODESBI = 10
Yes
No
XA register←LCDDATA13, 14
; Transfers 3rd digit of musical scale data (LCD13, 14) to SIO.
1
MODESBI = 11
Yes
No
XA register←LCDDATA15, 16
; Transfers 4th digit of musical scale data (LCD15, 16) to SIO.
1
RETI


1
SBITIM = 100 μs
Yes
No
SBITIM←0
No
ACKD = 1
SBITIM count
Yes
COI = 1
No
1
Yes
MODESBI←0
REFRES_F←1
SB0 = high level
No
Yes
SCK = high level
No
Yes
MODESBI
Other than 1, 3
1, 3
CMDT←1
SIO←XA register
MODESBI←MODESBI + 1
CSIE←1
INTCSI enabled
; Enables SBI interrupt
(EI IECSI instruction)
RETI

(4) Program example

```
          VENT4   MBE=0,RBE=1,SBI          ; SBI

DSEG0       DSEG  0          AT      20H   ; Stores from address 20H of memory bank 0.

MODESBI:          DS         1             ; Indicates which data is being transferred.
LCD:              DS         16            ; LCD display area

        ; <Subroutine initial setting>
            MOV     XA,#0C1H               ; Does not specify connection of on-chip pull-up resistor
                                           for ports1, 2 and 3.
            MOV     POGA,XA                ; Specifies connection of on-chip pull-up resistor for
                                           ports0, 6 and 7.
            MOV     XA,#8AH
            MOV     CSIM,XA                ; Sets serial operating mode.
            SET1    RELT
            EI                             ; Enables all interrupts.

        ; <Subroutine SBI start>
            SKT     REFRES_F               ; If 1, refreshes LCD.
            RET
            SKT     PORT0.2                ; SB0 = high level?
            RET
            SKT     PORT0.1                ; SCK = high level?
            RET
            SET1    CMDT                   ; CMDT set
            NOP
            NOP
            SET1    RELT                   ; RELT set
            NOP
            NOP
            SKT     PORT0.2                ; SB0 = high level?
            RET
            SET1    CMDT                   ; CMDT set
            MOV     XA,#01H
            MOV     SIO,XA                 ; Shift register ← slave address (01H)
            MOV     A,#1H
            MOV     MODESBI,A              ; MODESBI ← 1H
            EI      IECSI                  ; Enables SBI interrupt.
            MOV     XA,#1H
            MOV     SVA,XA                 ; Slave address register ← 1H
            CLR1    REFRES_F               ; REFRES_F ← 0
            RET
            ;
```

```
        ; <Subroutine SBI processing>
SBI:
        SEL     RB3             ; Register bank ← 3
        DI      IEBT            ; Disables use of basic interval timer (INTBT).
        EI                      ; Enables all interrupts.
        MOV     A,MODESBI
        ADDS    A,#0FH
        BR      SBI_E
        ADDS    A,#0FH
        BR      SBI1
        ADDS    A,#0FH
        BR      SBI2
        ADDS    A,#0FH
        BR      SBI3
        ADDS    A,#0FH
        BR      SBI4
        ADDS    A,#0FH
        BR      SBI5
        ADDS    A,#0FH
        BR      SBI6
        ADDS    A,#0FH
        BR      SBI7
        ADDS    A,#0FH
        BR      SBI8
        ADDS    A,#0FH
        BR      SBI9
        ADDS    A,#0FH
        BR      SBI10
        ADDS    A,#0FH
        BR      SBI11
        BR      SBI_E

SBI1:
        MOV     BC,#04H         ; Shift register ← pointer command (04H)
        BR      SBITIM

SBI2:
        MOV     BC,#07H         ; Shift register ← pointer (07H)
        BR      SBITIM

SBI3:
        MOV     BC,#01H         ; Shift register ← write command (01H)
        BR      SBITIM

SBI4:
        MOV     XA,LCD          ; Shift register ← address 1st digit
        MOV     BC,XA
        BR      SBITIM

SBI5:
        MOV     XA,LCD+2        ; Shift register ← address 2nd digit
        MOV     BC,XA
        BR      SBITIM
```

```
SBI6:
        MOV     XA,LCD+4            ; Shift register ← address 3rd digit
        MOV     BC,XA
        BR      SBITIM

SBI7:
        MOV     XA,LCD+6            ; Shift register "_" data
        MOV     BC,XA
        BR      SBITIM

SBI8:
        MOV     XA,LCD+8            ; Shift register ← musical scale data 1st digit
        MOV     BC,XA
        BR      SBITIM

SBI9:
        MOV     XA,LCD+10           ; Shift register ← musical scale data 2nd digit
        MOV     BC,XA
        BR      SBITIM

SBI10:
        MOV     XA,LCD+12           ; Shift register ← musical scale data 3rd digit
        MOV     BC,XA
        BR      SBITIM

SBI11:
        MOV     XA,LCD+14           ; Shift register ← musical scale data 4th digit
        MOV     BC,XA
SBITIM:
        MOV     L,#1110B            ; Initialization of 100 μs counter
SBICNT:
        DECS    L
        BR      SBI_ACKD

    ; <Subroutine error handling>
SBI0:
        MOV     A,#0H
        MOV     MODESBI,A           ; MODESBI ← 0
        SET1    REFRES_F            ; If 1, refreshes LCD display.
        BR      SBI_E

SBI_ACKD:
        SKT     ACKD                ; Acknowledge detection flag = 1?
        BR      SBICNT
        SKT     COI                 ; Address comparator signal = 1?
        BR      SBI0
        SKT     PORT0.2             ; SB0 = high level?
        BR      SBI_E
        SKT     PORT0.1             ; SCK = high level?
        BR      SBI_E
        MOV     A.MODESBI
        ADDS    A,#0FH              ; MODESBI = 0?
        BR      SBI_SIO
        ADDS    A,#0FH              ; MODESBI = 1?
        BR      SBI_CMDT
```

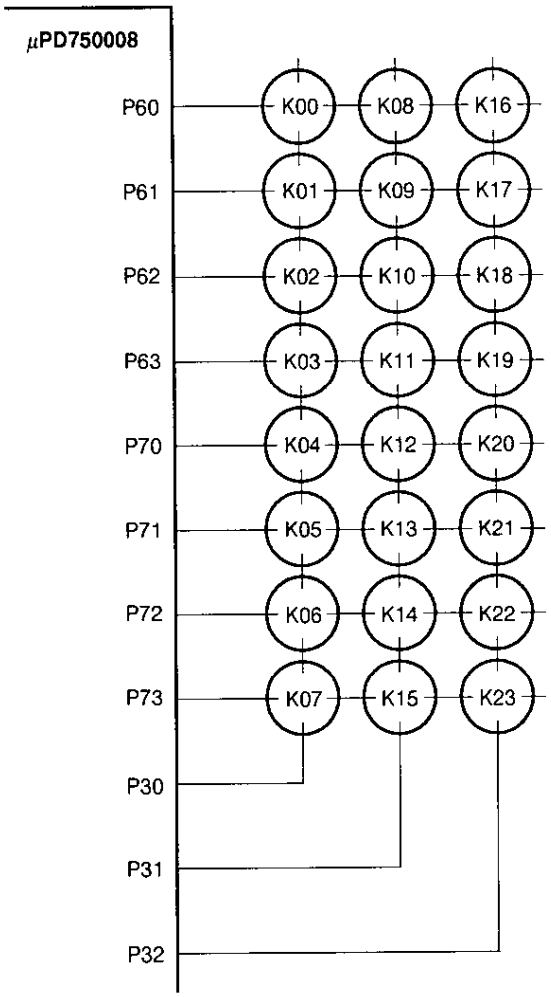
```
          ADDS    A,#0FH              ; MODESBI = 2?
          BR      SBI_SIO
          ADDS    A,#0FH              ; MODESBI = 3?
SBI_CMDT:
          SET1    CMDT                ; CMDT set
SBI_SIO:
          MOV     XA,BC
          MOV     SIO,XA              ; Shift register ← BC register
          INCS    MODESBI             ; MODESBI + 1
          NOP
          SET1    CSIE                ; Serial interface operation, disables use of shift
                                        register.
          EI      IECSI               ; Enables SBI interrupt
SBI_E:
          RETI
          ;
```

CHAPTER 9 KEY INPUT SUBROUTINE

Key input is performed from the momentary keys (3 × 8).

Here, port3 is used for key scan signal output, and ports6 and 7 for return signal fetching to make up a key matrix as shown in Figure 9-1.

Figure 9-1. Key Matrix Configuration



In this program, a key scan is performed every 8 ms, and if the key data matches three consecutive times, the key is confirmed (KEYCHKF = 1).

The key data input is stored in 4-bit key source data (KEYS) and 8-bit key return data (KEYR). The relationship between each key and key data is shown below.

Relationship between each key and key data

Key	KEYS	KEYR
K00	: 1110B	11111110B
K01	: 1110B	11111101B
K02	: 1110B	11111011B
K03	: 1110B	11110111B
K04	: 1110B	11101111B
K05	: 1110B	11011111B
K06	: 1110B	10111111B
K07	: 1110B	01111111B
K08	: 1101B	11111110B
K09	: 1101B	11111101B
K10	: 1101B	11111011B
K11	: 1101B	11110111B
K12	: 1101B	11101111B
K13	: 1101B	11011111B
K14	: 1101B	10111111B
K15	: 1101B	01111111B
K16	: 1011B	11111110B
K17	: 1011B	11111101B
K18	: 1011B	11111011B
K19	: 1011B	11110111B
K20	: 1011B	11101111B
K21	: 1011B	11011111B
K22	: 1011B	10111111B
K23	: 1011B	01111111B

(1) Explanation of program

The program explained here is the one used in **CHAPTER 11 APPLICATION EXAMPLES OF THIS APPLICATION PROGRAM** with some modifications to its input/output interface.

<Registers used>

XA, BC, HL

<RAM used>

RAM is placed from address 20H of memory bank 0.

```
LEDDIG:      1 word     ; LED display digit number. Changes every 2 ms as follows.
                        ; 0111B → 1011B → 1101B → 1110B
KEYCODE:     2 words    ; Performs coding of key source and key return.
  KEYR:      2 words    ; Key return
  KEYRB:     2 words    ; Previous key return
  KEYTIM:    2 words    ; Timer for counting key on time
KEYS:        1 word     ; Key source
  KEYSB:     1 word     ; Previous key source
  KEYCATT:   1 word     ; Key chattering counter
KEYF:        1 word     ; Flag
  KEYONF:    1 bit      ; If 1, key on
  KEYON2SF:  1 bit      ; If key is held down 2 seconds or more, becomes 1.
  HTKEYONF:  1 bit      ; If 1, main unit keyed on.
  KEYCHKF:   1 bit      ; If 1, ends key scan.
```

<Nesting>

1 level (6 words)

<Hardware used>

Port: Ports3, 6, 7

<Initial setting>

- Sets port3 to output mode.
- Sets ports6 and 7 to input mode and specifies connection of on-chip pull-up resistor.

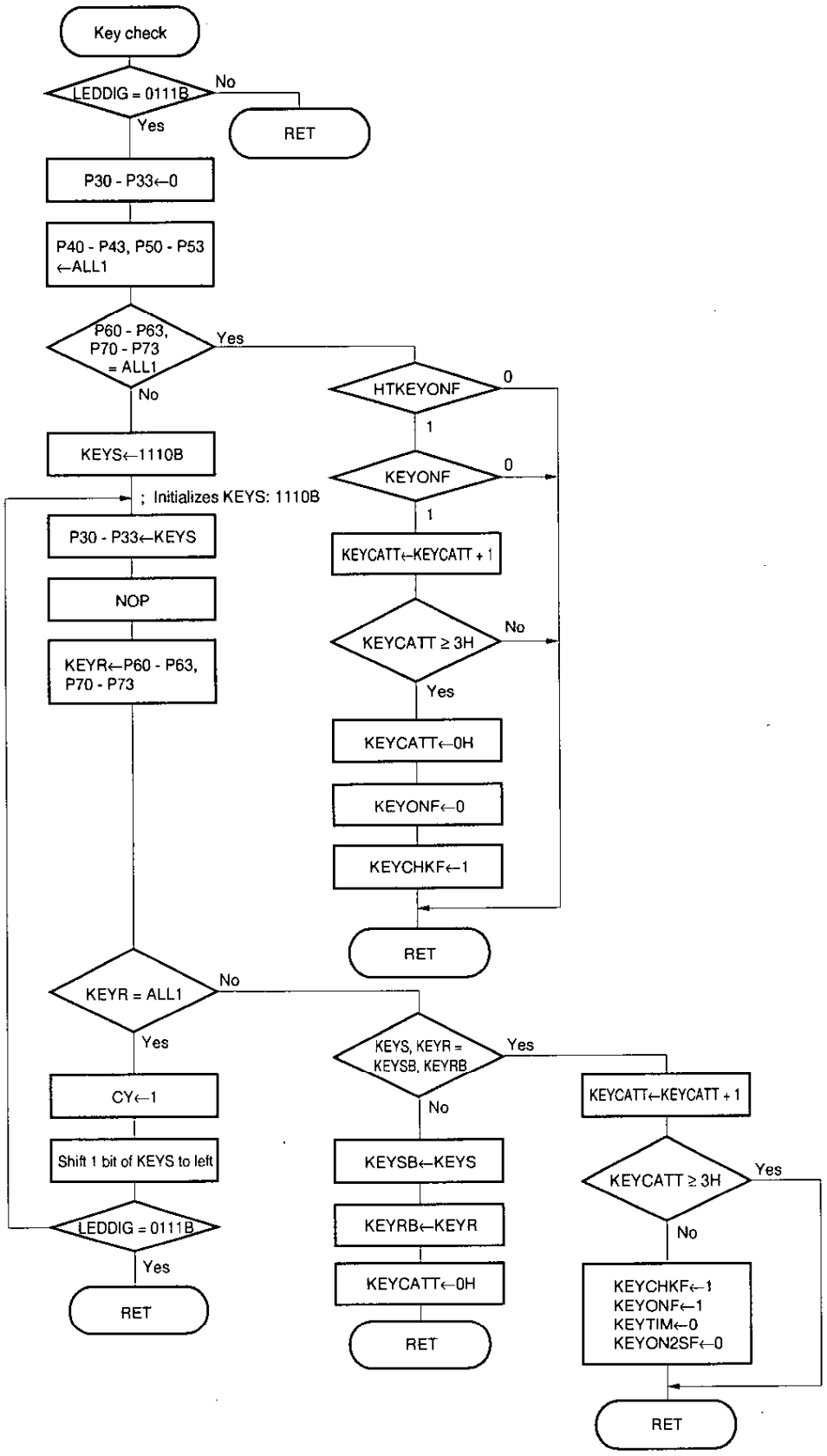
<Start-up procedure>

Set basic interval timer interrupts are to be generated every 2 ms.

Use the LED display digit counter (LEDDIG) for the 8 ms timer for key check.

Use key decoding of the main routine to judge key scan end and process each key.

(2) Flowchart


Key check
LEDDIG = 0111B
No
RET
Yes
P30 - P33←0
P40 - P43, P50 - P53 ←ALL1
P60 - P63, P70 - P73 = ALL1
Yes
No
KEYS←1110B
; Initializes KEYS: 1110B
P30 - P33←KEYS
NOP
KEYR←P60 - P63, P70 - P73
KEYR = ALL1
No
Yes
CY←1
Shift 1 bit of KEYS to left
LEDDIG = 0111B
Yes
RET
HTKEYONF
0
1
KEYONF
0
1
KEYCATT←KEYCATT + 1
KEYCATT ≥ 3H
No
Yes
KEYCATT←0H
KEYONF←0
KEYCHKF←1
RET
KEYS, KEYR = KEYSB, KEYRB
Yes
No
KEYSB←KEYS
KEYRB←KEYR
KEYCATT←0H
RET
KEYCATT←KEYCATT + 1
KEYCATT ≥ 3H
Yes
No
KEYCHKF←1
KEYONF←1
KEYTIM←0
KEYON2SF←0
RET

(3) Program example

```
DSEG0       DSEG  0      AT      20H          ; Stores from address 20H of memory bank 0.

LEDDIG:           DS      1                   ; LED display digit number

KEYCODE:          DS      2                   ; Performs coding of key source and key return.
KEYR:             DS      2                   ; Key return
KEYRB:            DS      2                   ; Previous key return
KEYTIM:           DS      2                   ; Timer for counting key on time
KEYS:             DS      1                   ; Key source
KEYSB:            DS      1                   ; Previous key source
KEYCATT:          DS      1                   ; Key chattering counter

KEYFLG:           DS      1                   ; Flag
 KEYONF           EQU     KEYFLG.3            ; If 1, key on
 KEYON2SF         EQU     KEYFLG.2            ; If 1, key is held down 2 seconds or more.
 HTKEYONF         EQU     KEYFLG.1            ; If 1, main unit key on.
 KEYCHKF          EQU     KEYFLG.0            ; If 1, ends key scan

;***** KEYR data *****
KEYR_1            EQU     11111110B
KEYR_2            EQU     11111101B
KEYR_3            EQU     11111011B
KEYR_4            EQU     11110111B
KEYR_5            EQU     11101111B
KEYR_6            EQU     11011111B
KEYR_7            EQU     10111111B
KEYR_8            EQU     01111111B
```

; **<Subroutine initialization>**

```
   MOV      A,#0FH
   OUT      PORT3,A
   MOV      XA,#0FH
   MOV      PMGA,XA          ; Sets port3 to output mode, and port6 to input
                               mode.
   MOV      XA,#34H
   MOV      PMGB,XA          ; Sets ports2, 4 and 5 to output mode, and port7
                               to input mode.
   MOV      XA,#0C1H         ; Connects on-chip pull-up resistor to ports1, 2
                               and 3.
   MOV      PMGA,XA          ; Connects on-chip pull-up resistor to ports0, 6
                               and 7.
```

```
      ;Main routine
;++++++++++++++++++++++++++++++++
;            Key decode
;++++++++++++++++++++++++++++++++
KEYDEC:
         SKT      KEYCHKF                  ; If 1, ends key scan.
         BR       KEYEND

         CLR1     KEYCHKF
         SKT      KEYONF                   ; If 1, key on
         BR       KEYOFF
; ------------------------------
;     Key on processing
; ------------------------------
         BR       KEYEND
         ;
KEYOFF:
; ------------------------------
;     Key off processing
; ------------------------------

KEYEND:

      ; <Subroutine 100 ms timer>
; ------------------------------
;     Key on time count
; ------------------------------
         MOV      XA KEYTIM
         MOV      HL,XA
         INCS     HL
         NOP
         MOV      XA,HL
         MOV      KEYTIM,XA                ; Key on time count timer + 1
```

; **<Subroutine 2 ms basic interval timer interrupt>**

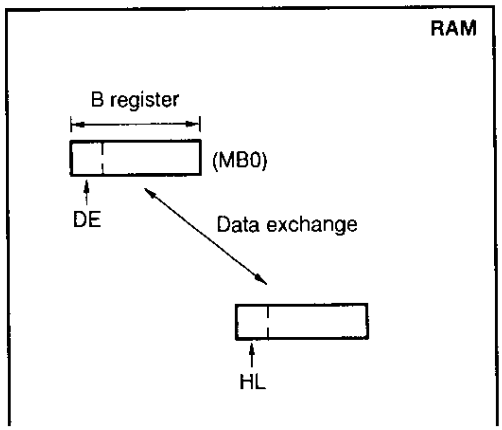
```
;++++++++++++++++++++++++++++++++
;            Key check
;++++++++++++++++++++++++++++++++
KEYCHK:
        MOV     A,LEDDIG
        SKE     A,#0111B                ; 8 ms?

        MOV     A,#0H
        OUT     PORT3,A                 ; Port3 ← 0
        MOV     XA,#0FFH
        OUT     PORT4,XA                ; Ports4, 5 ← ALL1
        IN      XA,PORT6
        MOV     BC,#0FFH
        SKE     XA,BC                   ; Ports6, 7 = ALL1?
        BR      KEYCHK1
        SKT     KEYONF                  ; If 1, key on
        RET
        INCS    KEYCATT                 ; Chattering counter + 1
        NOP
        MOV     A,KEYCATT
        ADDS    A,#0DH                  ; Key matches 3 times?
        RET
        MOV     A,#0H
        MOV     KEYCATT,A               ; Chattering counter ← 0H
        CLR1    KEYONF                  ; KEYONF ← 0
        SET1    KEYCHKF                 ; If 1, ends key scan.
        RET

KEYCHK1:
        MOV     A,#1110B
        MOV     KEYS,A                  ; Key source ← 1110B
KEYCHK2:
        OUT     PORT3,A                 ; Port3 ← 1110B
        NOP                             ; Waits for key source stabilization time.
        NOP
        NOP
        NOP
        NOP
        IN      XA,PORT6
        MOV     KEYR,XA                 ; Key return ← ports6, 7
        MOV     BC,#0FFH
        SKE     XA,BC                   ; Ports6, 7 = ALL1?
        BR      KEYCHK3
        SET1    CY                      ; CY ← 1
        MOV     A,KEYS
        ADDC    XA,XA
        MOV     KEYS,A                  ; Shifts 1 bit of key source to left.
        SKE     A,#0111B
        BR      KEYCHK2
        RET
```

```
KEYCHK3:
        MOV     A,KEYS
        MOV     HL,#KEYSB
        SKE     A,@HL                   ; Key source = KEYSB?
        BR      KEYCHK23
        MOV     A,KEYR
        MOV     HL,#KEYRB
        SKE     A,@HL                   ; Key return = KEYRB?
        BR      KEYCHK23
        MOV     A,KEYR+1
        MOV     HL,#KEYRB+1
        SKE     A,@HL                   ; Key return + 1 = KEYRB + 1?
        BR      KEYCHK23
        INCS    KEYCATT                 ; Chattering counter + 1
        NOP
        MOV     A,KEYCATT
        ADDS    A,#0DH                  ; Key matches 3 times?
        RET
        ;

      ; <Subroutine end of chattering elimination>
        SET1    KEYONF
        CLR1    KEYON2SF
        MOV     XA,#0H
        MOV     KEYTIM,XA               ; Starts 2s timer.
        SET1    KEYCHKF                 ; If 1, ends key scan.
        RET
        ;

      ; <Subroutine key contents different from previous time>
KEYCHK23:
        MOV     A,KEYS
        MOV     KEYSB,A                 ; KEYSB ← key source
        MOV     XA,KEYR
        MOV     KEYRB,XA                ; KEYRB ← key return
        MOV     A,#0H
        MOV     KEYCATT,A               ; Chattering counter ← 0
        RET
```

CHAPTER 10 SUBROUTINES

10.1 Data Transfer

Up to 16 words of data is exchanged between memory bank 0 and the memory bank selected by MBE·MBS. The start address of each area is specified by the DE register (memory bank = 0) and HL register (memory bank = MBE·MBS). The data length (number of words) is specified by the B register. In this case, the address specified by the HL register and DE register must not span more than 2 lines.



Program example

This program is not used in **CHAPTER 11 APPLICATION EXAMPLES OF THIS APPLICATION PROGRAM.**

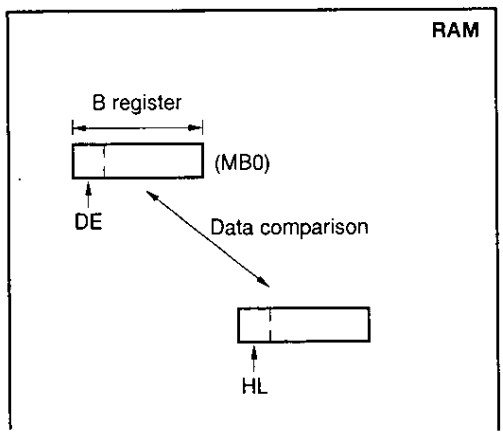
- Number of steps: 9 steps (9 bytes)
- Registers used : A, E, DE, HL

```
EXCH    CSEG    INBLOCK

EXCH:
        XCH     A,@DE
        XCH     A,@HL+
        XCH     A,@DE
        INCS    E
        NOP
        DECS    B
        BR      EXCH

        RET
```

10.2 Data Comparison

Up to 16 words of data of memory bank 0 and the memory bank selected by MBE·MBS are compared.

The start address of the data to be compared is specified by the DE register (memory bank = 0) and HL register (memory bank = MBE·MBS). The data length (number of words) is specified by the B register. In this case, the address specified by the HL register and DE register must not span more than 2 lines. If two data are equal as a result of comparison, a skip is performed after a return.



Program example

This program is not used in **CHAPTER 11 APPLICATION EXAMPLES OF THIS APPLICATION PROGRAM.**

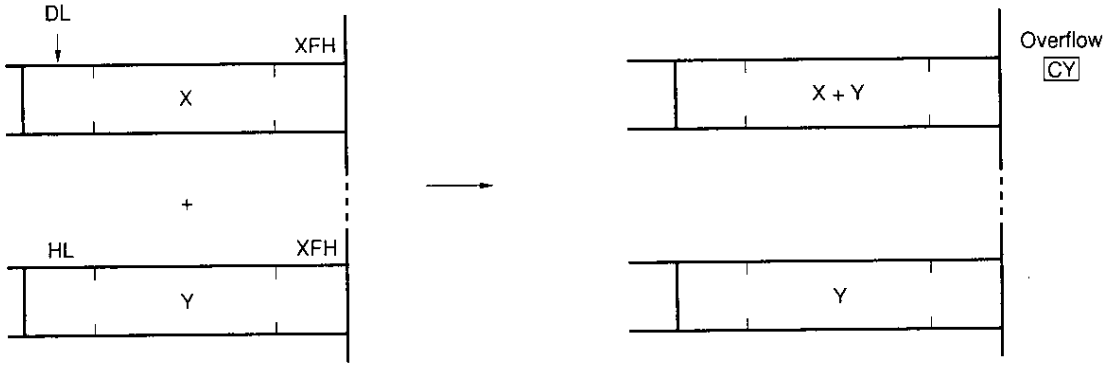
- Number of steps: 10 steps (10 bytes)
- Registers used : A, B, DE, HL

```
COMP    CSEG    INBLOCK

        MOV     A,@DE
        SKE     A,@HL
        RET
        INCS    L
        NOP
        INCS    E
        NOP
        DECS    B
        BR      COMP

        RETS
```

10.3 Decimal Addition

Up to 16 digits (16 words) of decimal addition can be performed between memory bank 0 and the memory bank selected by MBE·MBS. The address of the lowest digit of the data for decimal addition is specified by the DL register and HL register. The number of digits will be 10H minus the value of the L register. In this case, the data memory address indicated by @DL is 000H to 0FFH (memory bank 0) regardless of the value of MBE·MBS, and the data memory address indicated by @HL is the memory bank range indicated by the value of MBE·MBS.

The result of this decimal addition is stored to the data area addressed by the DL register. The overflow generated will be left to the carry flag (CY).



Program example

This program is not used in **CHAPTER 11 APPLICATION EXAMPLES OF THIS APPLICATION PROGRAM.**

- Number of steps: 9 steps (9 bytes)
- Registers used : A, D, HL

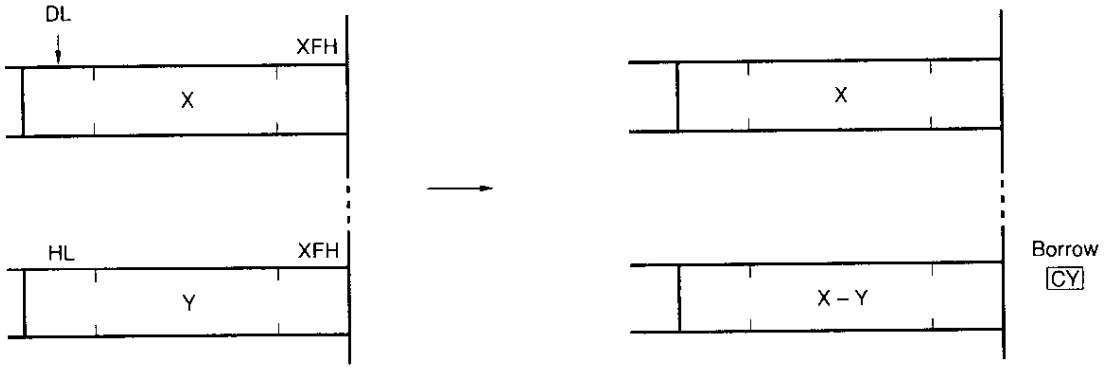
```
DECADD   CSEG        INBLOCK

         CLR1        CY
LOOP:
         MOV         A,@DL
         ADDS        A,#6
         ADDC        A,@HL
         ADDS        A,#0AH
         XCH         A,@DL
         INCS        L
         BR          LOOP

         RET
```

10.4 Decimal Subtraction

Up to 16 digits (16 words) of decimal subtraction can be performed between memory bank 0 and the memory bank selected by MBE·MBS.

The lowest address of the data area to which the minuend for decimal subtraction is stored is specified by the DL register. The lowest address of the data area to which the subtrahend is stored is specified by the HL register. The number of digits will be 10H minus the value of the L register. In this case, the data memory address indicated by @DL is 000H to 0FFH (memory bank 0) regardless of the value of MBE·MBS, and the data memory address indicated by @HL is the memory bank range indicated by the value of MBE·MBS.

The result of this decimal subtraction is stored to the data area indicated by the HL register. Any borrow generated will be left on the carry flag (CY).



Program example

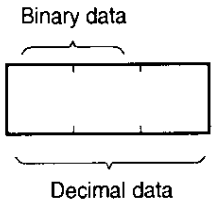
This program is not used in **CHAPTER 11 APPLICATION EXAMPLES OF THIS APPLICATION PROGRAM.**

- Number of steps: 19 steps (19 bytes)
- Registers used : A, D, HL
- Stacks used : 2 words

```
DECSUB    CSEG        INBLOCK

          PUSH        HL
          CLR1        CY
LOOP0:
          MOV         A,@DL
          SUBC        A,@HL
          ADDS        A,#0AH
          XCH         A,@HL
          INCS        L
          BR          LOOP0

          POP         HL
          SKT         CY
          RET

          CLR1        CY
LOOP1:
          MOV         A,#0H
          SUBC        A,@HL
          ADDS        A,#0AH
          XCH         A,@HL
          INCS        L
          BR          LOOP1

          RET
```

10.5 Binary/Decimal Conversion

Binary 8-bit data in the data memory is converted to BCD and the result is stored both in the data memory where the data existed before the conversion and in the address 1 word higher.



If the lower 4 bits of the binary 8-bit data is X, and the higher 4 bits is Y, this operation can be expressed as follows:

X (binary) + Y (binary) (10 + 6)

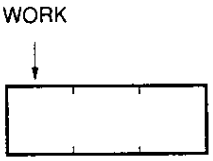
This can be expressed in the following way.

(X (binary) + Y (binary) × 10) + Y (binary) × 6

Here, X and Y are binary numbers. Therefore, X and Y of (X + Y × 10) are first converted to decimal numbers, and 6 is added Y times. In this case, decimal adjustment is performed for each addition.

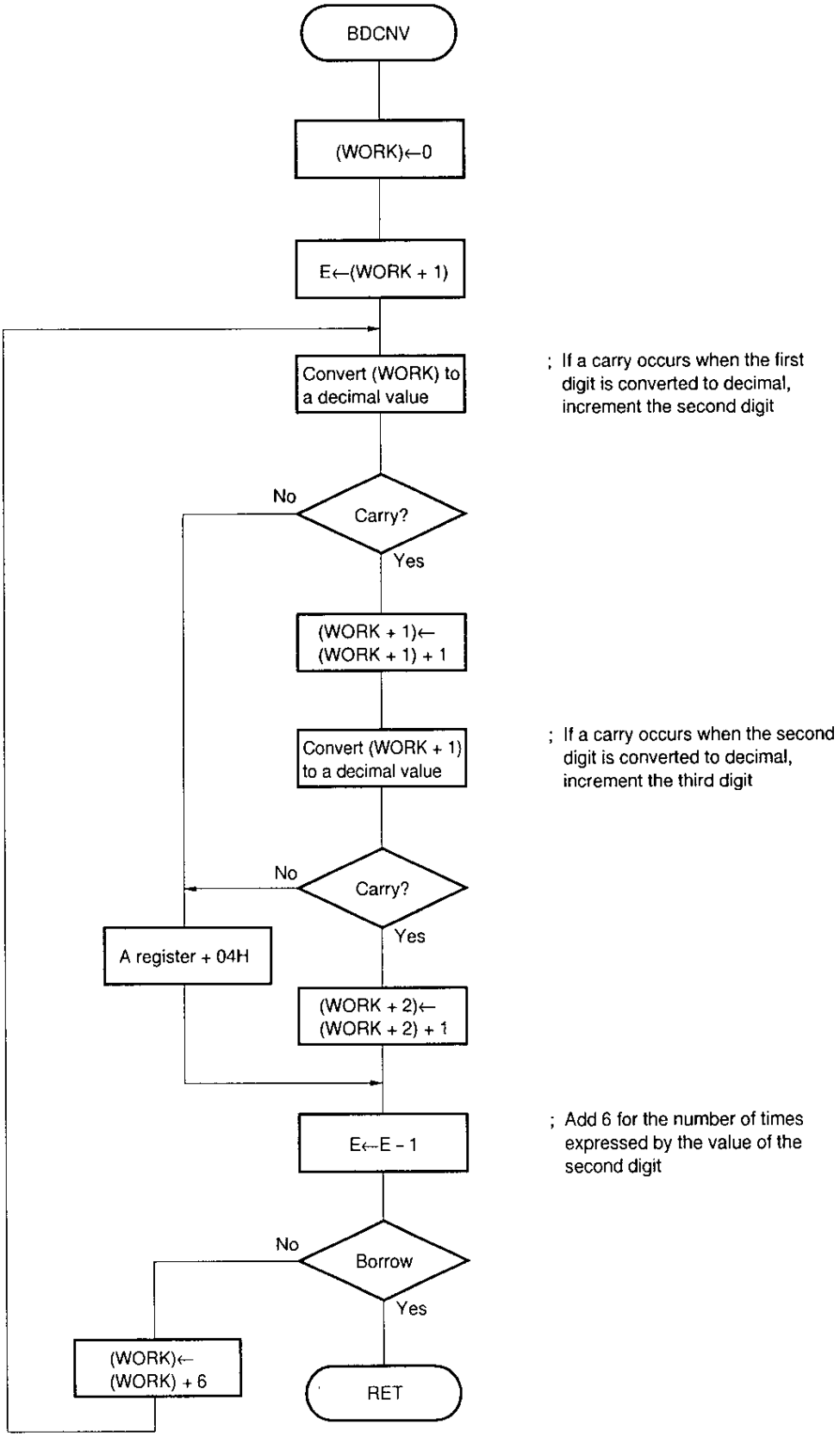
In this manner, a decimal value converted from a binary value can be obtained.

(1) RAM



12 bits: Converts the data of the lower 8 bits to BCD and stores it.

(2) Flow chart


BDCNV
(WORK)←0
E←(WORK + 1)
Convert (WORK) to a decimal value
; If a carry occurs when the first digit is converted to decimal, increment the second digit
No
Carry?
Yes
(WORK + 1)← (WORK + 1) + 1
Convert (WORK + 1) to a decimal value
; If a carry occurs when the second digit is converted to decimal, increment the third digit
No
Carry?
Yes
A register + 04H
(WORK + 2)← (WORK + 2) + 1
E←E - 1
; Add 6 for the number of times expressed by the value of the second digit
No
Borrow
Yes
(WORK)← (WORK) + 6
RET

(3) Program example

This program is not used in **CHAPTER 11 APPLICATION EXAMPLES OF THIS APPLICATION PROGRAM.**

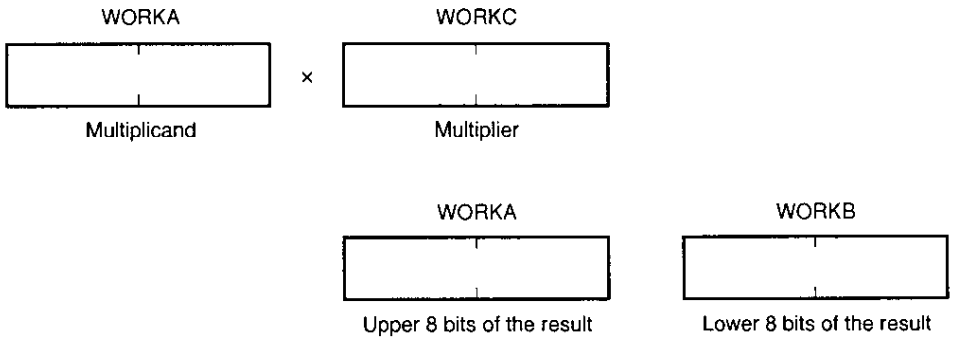
- Number of steps: 24 steps (32 bytes)
- Registers used : A, E, HL

```
        PUBLIC  WORK

BDDATA  DSEG    AT      10H

WORK:   DS      3H
BDCNV   CSEG    INBLOCK

        MOV     A,#0H
        MOV     WORK + 2,A
        MOV     A,WORK + 1
        XCH     A,E

        MOV     A,WORK

BD:
        MOV     HL,#WORK        ; Decimal adjustment for the first digit
        ADDS    A,#6H
        BR      ADD10

        MOV     @HL,A           ; Increments the second digit
        INCS    L
        MOV     A,#7H
        ADDS    A,@HL           ; Decimal adjustment
        BR      ADD10

        INCS    WORK + 2        ; Increments the third digit
CTLOOP:
        MOV     @HL,A           ; Add 6 for number of times expressed by the value of
                                  the second digit
        DECS    E
        BR      BDX2
        RET
BDX2:
        MOV     A,WORK
        ADDS    A,#6H
        BR      BD
ADD10:
        ADDS    A,#0AH
        NOP
        BR      CTLOOP
```

10.6 8-bit Multiplication

8-bit binary multiplication can be performed by using work area WORKA for the multiplicand and work area WORKC for the multiplier, in the data memory.

The upper 8 bits of the result of this multiplication is stored to WORKA, and the lower 8 bits of the result to WORKB. However, WORKA and WORKB are continuous from the lower address, and the memory bank of these work areas is selected by MBE·MBS. The memory bank for WORKC is memory bank 0.



Program example

This program is not used in **CHAPTER 11 APPLICATION EXAMPLES OF THIS APPLICATION PROGRAM.**

- Number of steps: 38 steps (50 bytes)
- Registers used : XA, BC, DE, HL
- Stacks used : 2 words

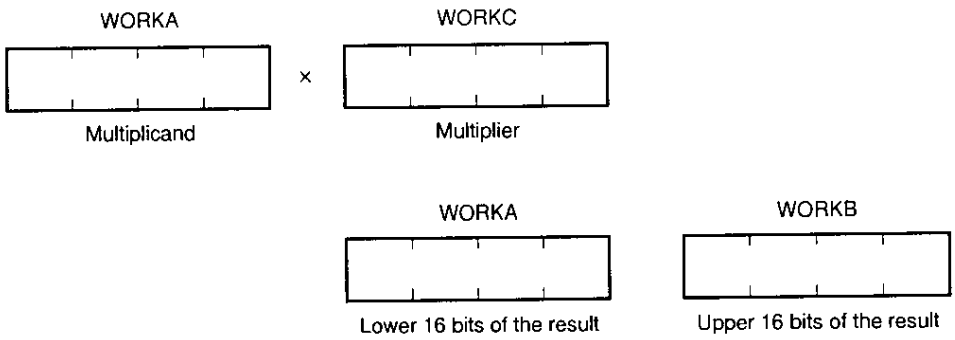
```
        PUBLIC  WORKA,WORKB,WORKC

DMLT8   DSEG    AT      20H

WORKC:  DS      2H
WORKB:  DS      2H
WORKA:  DS      2H

MULT8   CSEG    INBLOCK

        MOV     XA,#00H
        MOV     WORKB,XA
        MOV     C,#1H
LOOP:
        MOV     B,#3H
        MOV     HL,#WORKB
        MOV     A,#0H
```

```
SHLOOP:
        XCH     A,@HL+
        NOP
        DECS    B
        BR      SHLOOP

        XCH     A,B
FIGURE:
        DECS    B
        BR      ADD

        DECS    C
        BR      LOOP

        RET
ADD:
        PUSH    BC
        CLR1    CY
        MOV     B,#1H
        MOV     DE,#WORKC
        MOV     HL,#WORKB
ADLOOP:
        MOV     A,@DE
        ADDC    A,@HL
        XCH     A,@HL+
        INCS    L
        NOP
        INCS    E
        NOP
        DECS    B
        BR      ADLOOP

        POP     BC
        MOV     HL,#WORKA
        NOT1    CY
        SKT     CY
INCA:
        INCS    @HL
        BR      FIGURE

        INCS    L
        NOP
        BR      INCA
```

10.7 16-bit Multiplication

16-bit binary multiplication can be performed by using the 16-bit work area WORKA for the multiplicand and the 16-bit work area WORKC for the multiplier, in the data memory.

The upper 16 bits of the result of this multiplication is stored to WORKB, and the lower 16 bits of the result to WORKA. However, WORKA and WORKB are continuous from the lower address, and their memory bank is determined by MBE·MBS. The memory bank for WORKC is memory bank 0.



Program example

This program is not used in **CHAPTER 11 APPLICATION EXAMPLES OF THIS APPLICATION PROGRAM.**

- Number of steps: 39 steps (51 bytes)
- Registers used : XA, BC, DE, HL
- Stacks used : 2 words

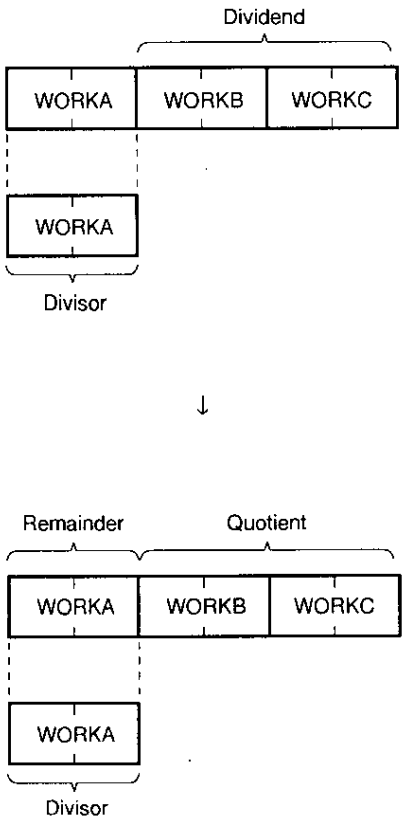
```
        PUBLIC  WORKA,WORKB,WORKC

DMLT16  DSEG    AT      20H

WORKC:  DS      4H
WORKB:  DS      4H
WORKA:  DS      4H

MULT16  CSEG    INBLOCK

        MOV     XA,#00H
        MOV     WORKB,XA
        MOV     WORKB+2,XA
        MOV     C,#3H
LOOP:
        MOV     B,#7H
        MOV     HL,#WORKB
        MOV     A,#0H
SHLOOP:
        XCH     A,@HL+
        NOP
        DECS    B
        BR      SHLOOP

        XCH     A,B
FIGURE:
        DECS    B
        BR      ADD
        DECS    C
        BR      LOOP
        RET
ADD:
        PUSH    BC
        CLR1    CY
        MOV     B,#3H
        MOV     DE,#WORKC
        MOV     HL,#WORKB
ADLOOP:
        MOV     A,@DE
        ADDC    A,@HL
        XCH     A,@HL+
        NOP
        INCS    E
        NOP
        DECS    B
        BR      ADLOOP

        POP     BC
        MOV     HL,#WORKA
        NOT1    CY
        SKT     CY
INCA:
        INCS    @HL
        BR      FIGURE

        INCS    L
        NOP
        BR      INCA
```

10.8 Binary Division (16-bit ÷ 8-bit)

Binary division can be performed by using the 16-bit work areas WORKB and WORKC for the dividend and the 8-bit work area WORKD for the divisor, in the data memory.

The higher 8-bits of the quotient is stored to WORKC and the lower 8-bits of the quotient is stored to WORKB, and the remainder is stored to WORKA.

However, WORKA, WORKB, and WORKC have the same row address and are continuous from the lower address. In addition, the column address of WORKD must be the same as that of WORKA. All of these work areas must lie in the same memory bank.



Program example

This program is not used in **CHAPTER 11 APPLICATION EXAMPLES OF THIS APPLICATION PROGRAM.**

- Number of steps: 42 steps (59 bytes)
- Registers used : XA, B, D, HL

```
          PUBLIC  WORKA,WORKB,WORKC,WORKD

DIV8D0    DSEG    AT      20H

WORKA:    DS      2H
WORKB:    DS      2H
WORKC:    DS      2H

DIV8D1    DSEG    AT      30H

WORKD:    DS      2H

DIV8      CSEG    INBLOCK

          CLR1    MBE
          MOV     B,#0H
          MOV     XA,#00H
          MOV     WORKA,XA
LOOP:
          MOV     HL,#WORKA
          SET1    CY
          SKT     (WORKA + 5).3
          CLR1    CY
ROTAT:
          MOV     A,@HL
          ADDC    A,@HL
          MOV     @HL,A
          INCS    L
          NOP
          SKE     L,#(WORKA + 6) AND 0FH
          BR      ROTAT

          CLR1    CY

          MOV     HL,#WORKD
          MOV     D,#WORKA SHR 4
```

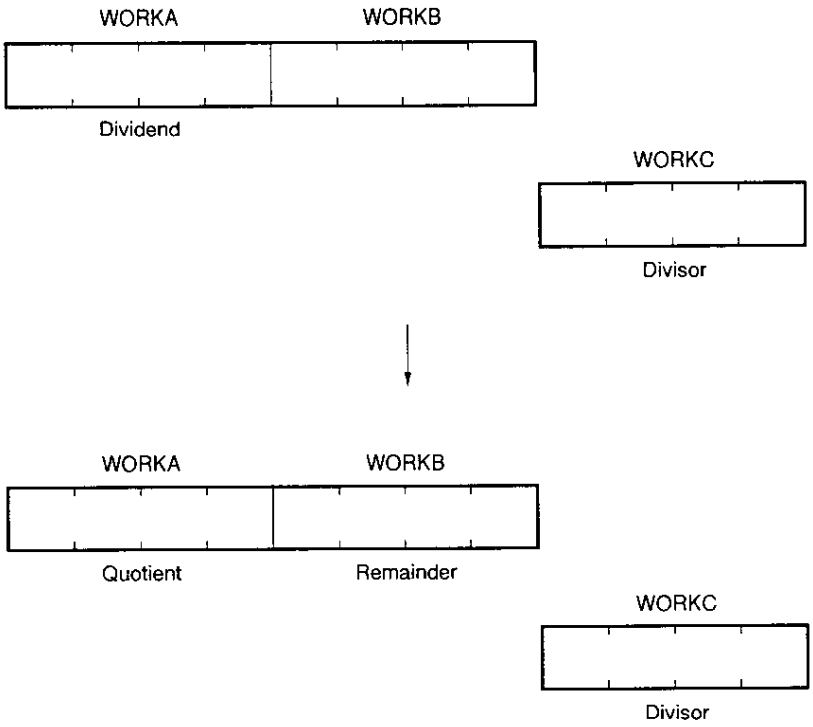
```
SUB:
        MOV     A,@DL
        SUBC    A,@HL
        XCH     A,@DL
        INCS    L
        SKE     L,#(WORKA + 2) AND 0FH
        BR      SUB

        SKT     CY
        BR      SETB0

        SKT     WORKB.0
        BR      ADD

FIGCNT:
        INCS    B
        BR      LOOP
        RET

ADD:
        CLR1    CY
        MOV     L,#WORKA AND 0FH
ADDLP:
        MOV     A,@DL
        ADDC    A,@HL
        XCH     A,@DL
        INCS    L
        SKE     L,#(WORKA + 2) AND 0FH
        BR      ADDLP
        BR      FIGCNT

SETB0:
        SET1    WORKB.0
        BR      FIGCNT
```

10.9 Binary Division (16-bit ÷ 16-bit)

Binary division can be performed by using the 16-bit work area WORKA for the dividend and the 16-bit work area WORKC for the divisor, in the data memory.

The quotient is stored to WORKA, and the remainder to 16-bit work areas WORKB.

However, WORKA and WORKB are continuous from the lower address, and their memory bank is 0. The memory bank for WORKC is selected by MBE·MBS.



Program example

This program is not used in **CHAPTER 11 APPLICATION EXAMPLES OF THIS APPLICATION PROGRAM.**

- Number of steps: 49 steps (63 bytes)
- Registers used : A, B, DE, HL

```
        PUBLIC  WORKA,WORKB,WORKC

DDIV16  DSEG    AT      20H

WORKA:  DS      4H
WORKB:  DS      4H
WORKC:  DS      4H

DIV16   CSEG    INBLOCK

        MOV     DE,#WORKB
CLRB:
        MOV     A,#0H
        XCH     A,@DE
        INCS    E
        NOP

        SKE     E,#(WORKB + 4) AND 0FH
        BR      CLRB

        MOV     B,#0CH
LOOP:
        MOV     DE,#WORKA
        MOV     A,#0H
SHFT:
        XCH     A,@DE
        INCS    E
        NOP
        SKE     E,#(WORKA + 8) AND 0FH
        BR      SHFT
SUB:
        MOV     DE,#WORKB
        MOV     HL,#WORKC
        CLR1    CY
SLOOP:
        MOV     A,@DE
        SUBC    A,@HL
        XCH     A,@DE
        INCS    L
        NOP
        INCS    E
        NOP
        SKE     L,#(WORKA + 4) AND 0FH
        BR      SLOOP
```

```
        NOT1    CY
        SKT     CY
        BR      D0

        MOV     DE,#WORKA
        XCH     A,@DE
        ADDS    A,#1H
        XCH     A,@DE
        BR      SUB
D0:
        MOV     DE,#WORKB
        MOV     HL,#WORKC
ADD:
        MOV     A,@DE
        ADDC    A,@HL
        XCH     A,@DE
        INCS    L
        NOP
        INCS    E
        NOP
        SKE     E,#(WORKB + 4) AND 0FH
        BR      ADD
        INCS    B
        BR      LOOP

        RET
```

CHAPTER 11 APPLICATION EXAMPLES OF THIS APPLICATION PROGRAM

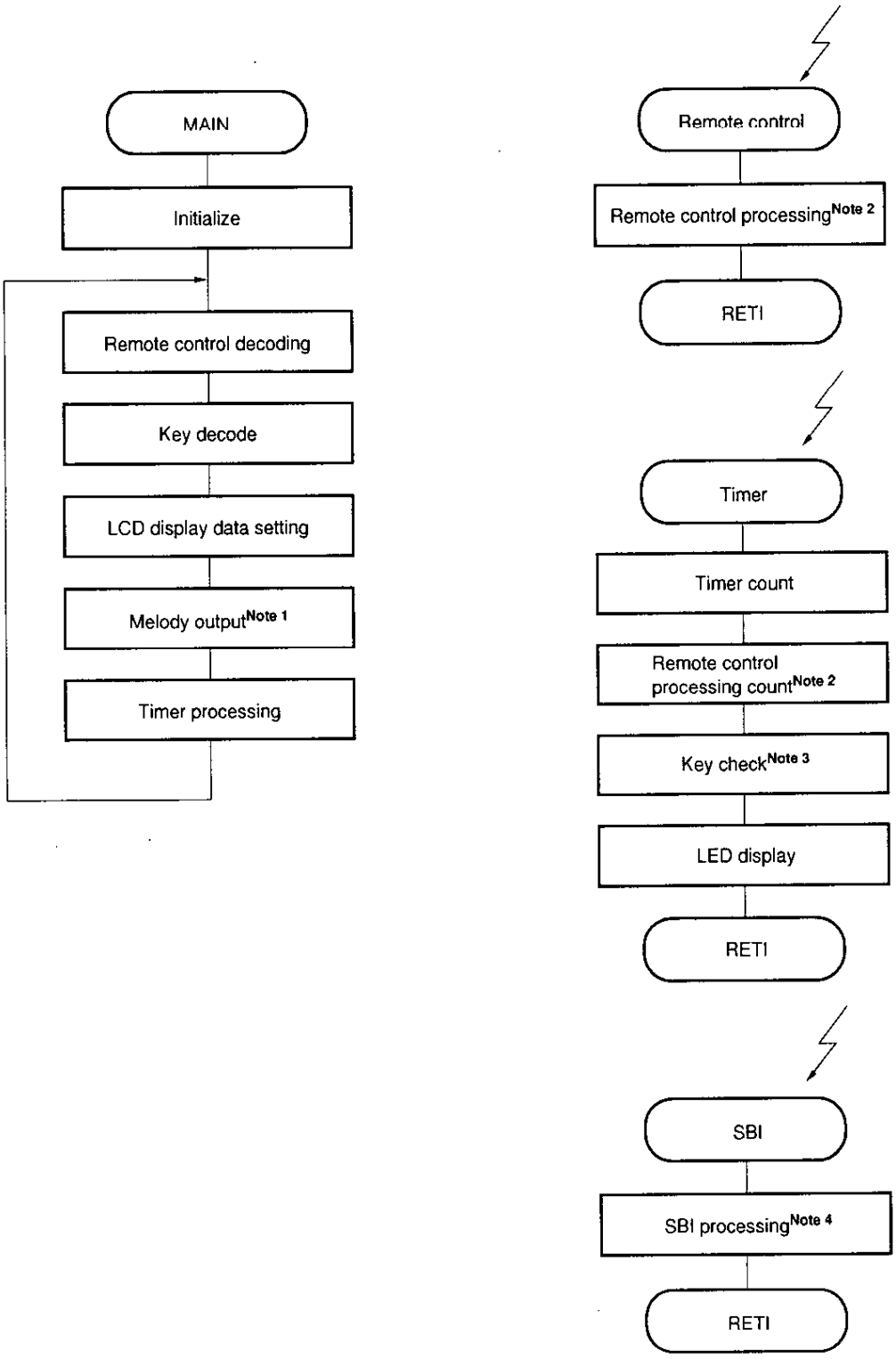
This chapter presents system program examples created using the application programs described in chapters until the previous chapter. The application programs used of those described in chapters until the previous chapter are shown below.

3.3 Remote Control Signal Reception Application
4.2.1 Melody output
8.1 Application Example in SBI Mode
CHAPTER 9 KEY INPUT SUBROUTINE

11.1 Program Outline

This program automatically presents a musical performance by inputting tone length data according to the musical scale data and keying duration by the momentary keys or remote control keys likened to a keyboard.

The entire flowchart is shown below.



Notes 1. Use of application software of **4.2.1 Melody output**

2. Use of application software of **3.3 Remote Control Signal Reception Application**

3. Use of application software of **CHAPTER 9 KEY INPUT SUBROUTINE**

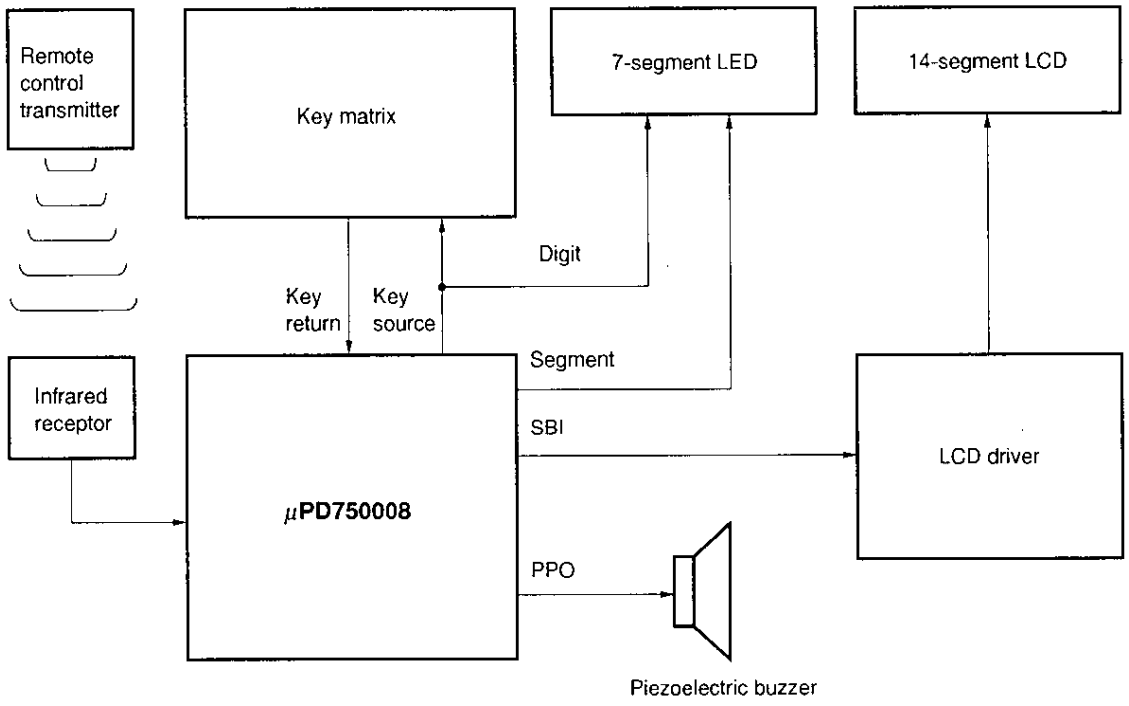
4. Use of application software of **8.1 Application Example in SBI Mode**

11.2 System Configuration

This system uses the μPD750008 as its device and controls output to the 14-segment LCD driver, 7-segment LED driver and piezoelectric buzzer by key matrix or remote control input.

Figure 11-1 shows the system configuration.

Figure 11-1. System Configuration



Key matrix	:	Consists of MODE, UP, DOWN, END, WAIT and musical scale keys.
Infrared receptor	:	Receives NEC format infrared code. MODE, UP, DOWN, END, WAIT and musical scale keys are coded.
7-segment LED	:	The 4-digit 7-segment LED displays "PROG", "PLAY," "PAUS" modes and tone length of 100 ms to 900 ms. Digit signals are also used as key source signals of the key matrix.
LCD driver	:	Displays performance data or musical scale being output on 8-digit 14-segment LCD with SBI format serial data.
Piezoelectric buzzer	:	Outputs data of the key pressed or perfor-mance data.

11.3 Port Assignment

Table 11-1. μPD750008 Port Assignment (1/2)

Pin Number	Pin Name	I/O	Dual-Function Pin	Function	After Reset	Active Value	Initial Set Value
15	P00	Input	INT4	Unused (connected to GND)	Input	–	In
14	P01	I/O	$\overline{SCK}$	Serial shift clock		L	H
13	P02	I/O	SO/SB0	Serial bus line with LCD driver		–	H
12	P03	I/O	SI/SB1	Unused (connected to GND)		–	In
19	P10	Input	INT0	Remote control input	Input	L	In
18	P11		INT1	Unused (connected to GND)		–	In
17	P12		INT2	Unused (connected to GND)		–	In
16	P13		TI0	Unused (connected to GND)		–	In
25	P20	I/O	PTO0	Piezoelectric buzzer	Input	L	H
24	P21		PTO1	Unused (connected to GND)		–	In
23	P22		PCL	Unused (connected to GND)		–	In
22	P23		BUZ	Unused (connected to GND)		–	In
9	P30	I/O	–	Digit output (LED7)/key strobe signal	Input	L	H
8	P31		–	Digit output (LED6)/key strobe signal		L	H
7	P32		–	Digit output (LED5)/key strobe signal		L	H
6	P33		–	Digit output (LED4)/key strobe signal		L	H
41	P40	I/O	–	Segment output (D.P.)	High level (with on-chip pull-up resistor) or high impedance	L	H
40	P41			Segment output (g)		L	H
39	P42			Segment output (f)		L	H
38	P43			Segment output (e)		L	H
37	P50	I/O	–	Segment output (d)	High level (with on-chip pull-up resistor or high impedance	L	H
36	P51			Segment output (c)		L	H
35	P52			Segment output (b)		L	H
34	P53			Segment output (a)		L	H
33	P60	I/O	KR0	Key input pin	Input	L	H
32	P61		KR1	Key input pin		L	H
31	P62		KR2	Key input pin		L	H
30	P63		KR3	Key input pin		L	H
29	P70	I/O	KR4	Key input pin	Input	L	H
28	P71		KR5	Key input pin		L	H
27	P72		KR6	Key input pin		L	H
26	P73		KR7	Key input pin		L	H
11	P80	I/O	–	Unused (connected to GND)	Input	–	In
10	P81		–	Unused (connected to GND)		–	In

Table 11-1. μPD750008 Port Assignment (2/2)

Pin Number	Pin Name	I/O	Dual-Function Pin	Function	After Reset	Active Value	Initial Set Value
4	X1, X2	Input	–	Main system clock oscillation crystal/ceramic connection pin	Input	–	–
5						–	–
1	XT1	Input	–	Unused (connected to GND)	Input	–	In
2	XT2	–		Unused (connected to GND)	–	–	In
3	$\overline{\text{RESET}}$	Input	–	System reset input pin	–	L	–
20	IC	–	–	Unused (connected to GND)	–	–	In
21	V_{DD}	–	–	Unused (connected to GND)	–	–	In
42	V_{SS}	–	–	Unused (connected to GND)	–	–	In

Table 11-2. State Transition Table

Mode \ Key Input		Mode Key		Musical Scale Key		END Key		UP Key	DOWN Key
		Key On 2 Secs. or More	Key On Less than 2 Secs.	Key On	Key Off	Key On	Key Off		
PROG	Function	Switches to PAUS mode.	Switches whether data input is performed from the start data or last data.	Judges the musical scale keys and outputs musical scales. Also counts tone length.	Output silent tone and stops tone length count. Stores musical scale and tone length data at performance address and moves to the next address.	Outputs silent tone and counts tone length counter.	Stops tone length count and stores repeat code and tone length data (repeat time) at performance address. Addresses are not changed.	Increments performance address by +1, and outputs performance data musical scale during key on. After key off, outputs silent tone.	Decrements performance address by −1, and outputs performance data musical scale during key on. After key off, outputs silent tone.
	LED display	PAUS	Prog	Prog	Prog	Prog	Prog	Prog	Prog
	LCD display	__END_	EX. 008-END__	EX. 000-DO__	EX. 001-RE__	EX. 127-WAIT	EX. 127-END_	EX. 004-SO__	EX. 006-SI_+
PLAY	Function	Switches to PROG mode. Inputs data from rest of performance address.	Aborts automatic performance and switches to PAUS mode.	Does not accept.		Returns performance address to address 000 and restarts automatic performance.	Does not accept.	Increments reference time by +1. LED displays reference time for 1 second and then returns to mode display.	Decrements reference time by −1. LED displays reference time for 1 second and then returns to mode display.
	LED display	Prog	PASU	PLAY		PLAY		EX. 300 → PLAY	EX. 100 → PLAY
	LCD display	EX. 007-DO_+	__END_	EX. 006-SI__		EX. 000-DO__		EX. 000-DO__	
PAUS	Function	Switches to PROG mode. Inputs data from rest of performance address.	Aborts automatic performance and switches to PLAY mode.	Judges the musical scale keys and outputs musical scales. Does not count tone length.	Outputs silent tone. Does not store data. Addresses are not changed either.	Returns performance address to address 000. LCD displays address 1 second.	Does not accept.	Increments reference time by +1. LED displays reference time for 1 second and then returns to mode display.	Decrements reference time by −1. LED displays reference time for 1 second and then returns to mode display.
	LED display	Prog	PLAY	PAUS	PASU	PAUS		EX. 300 → PAUS	EX. 100 → PAUS
	LCD display	EX. 003-FA__	EX. 004-SO__	EX. __LA#1	__END_	EX. 000-END → (after 1 sec.) → __END_		__END_	__END_

Remark PROG mode is set at the time of reset start.

11.5 Explanation of Function by Key

A key is valid when it is pressed while no other keys are pressed. If a key is pressed, other keys are not valid even if they are pressed (multiple pressing) until the key is released.

When no key on the key matrix is pressed, remote control input is accepted.

For the key matrix configuration, refer to **11.7 Hardware Configuration**.

Each key is explained below.

11.5.1 MODE key

Pressing any key for 2 seconds or more switches between PAUS mode and PLAY mode.

In PROG mode, performance data can be input by the musical scale keys. Furthermore, pressing the MODE key for 2 seconds or shorter can switch whether data input is performed from the start (start of performance data memory) data or the end (address FnH) data.

In PLAY mode or PAUS mode, automatic performance is possible by the performance data input in PROG mode. Furthermore, double pressing of the MODE key can switch whether performance should be started or paused.

The operation and display of the MODE key are shown below.

Mode	MODE Key		7-Segment LED
	Pressing 2 Seconds or More	Pressing 2 Seconds or Shorter	(LED4, LED5, LED6, LED7) Display
PROG mode	Switches to PAUS mode.	Switches whether data should be input from the start or end.	
PLAY mode	Switches to PROG mode (data is input from the end).	Aborts automatic performance and sets PAUS mode.	
PAUS mode (pause)		Restarts automatic performance and sets PLAY mode.	

PROG mode is set at the time of reset start.

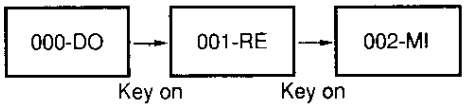
11.5.2 UP/DOWN key

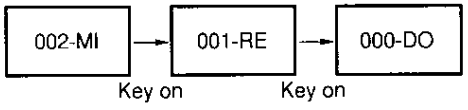
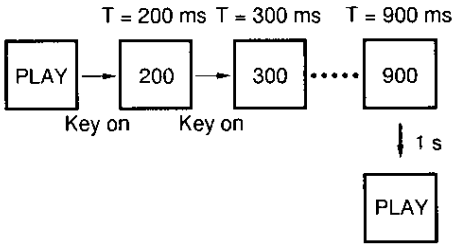
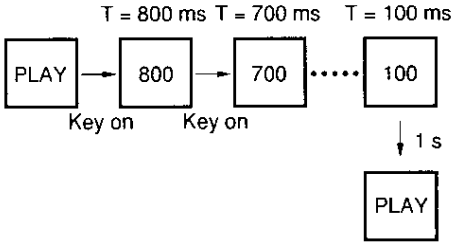
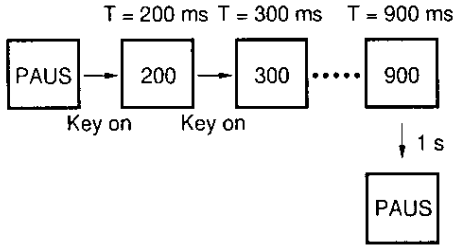
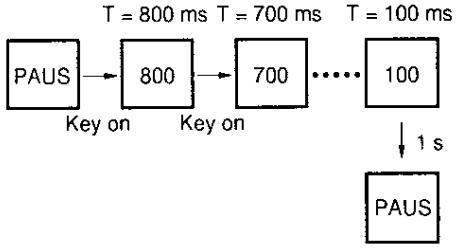
In PROG mode, each press of the UP/DOWN key increments/decrements the performance data address, and the musical scale of the data is displayed on an LCD while the key is held down.

In PLAY mode or PAUS mode, each press of the UP/DOWN key increments/decrements reference time T and displays it on an LCD. If the display reaches the maximum/minimum, the display returns to the mode display approximately 1 second after.

The operation and display of the UP/DOWN key are shown below.

<table>
<tr><th>Mode</th><th>UP Key</th><th>DOWN Key</th></tr>
<tr><td>PROG mode
Used to edit performance data</td><td>Data contents (DO RE MI)

14-segment LCD
 
</td><td>Data contents (DO RE MI)

14-segment LCD
 
</td></tr>
<tr><td>PLAY mode
Used to set reference time</td><td>7-segment LCD (Max.)
 
</td><td>7-segment LCD (Min.)
 
</td></tr>
<tr><td>PAUS mode
Used to set reference time</td><td>7-segment LCD (Max.)
 
</td><td>7-segment LCD (Max.)
 
</td></tr>
</table>

T = 200 ms at the time of reset start

11.5.3 Musical scale keys

The following 15 keys on the key matrix are called musical scale keys.

DO, RE, MI, FA, SO, LA+, SI+, DO+, DO#, RE#, FA#, SO#, LA+#, DO+#, WAIT (silent tone)

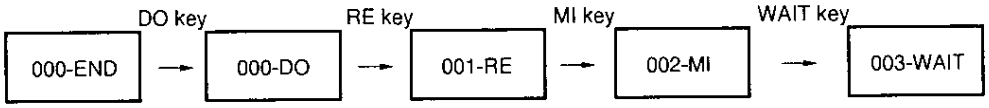
These keys are valid only in PROG mode or PAUS mode.

In PROG mode, the musical scale corresponding to each key or silent tone is output while the key is pressed, and the output musical scale is displayed. When the key is released, the musical scale and tone length data output to the performance memory area is written and the musical scale data to be output at the next address is displayed.

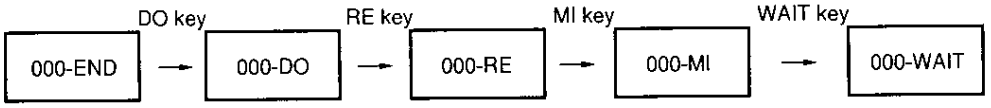
In PAUS mode, the musical scale corresponding to each key or silent tone is output while the key is pressed, and the output musical scale is displayed.

An operation example is shown below.

In PROG mode (14-segment LCD)



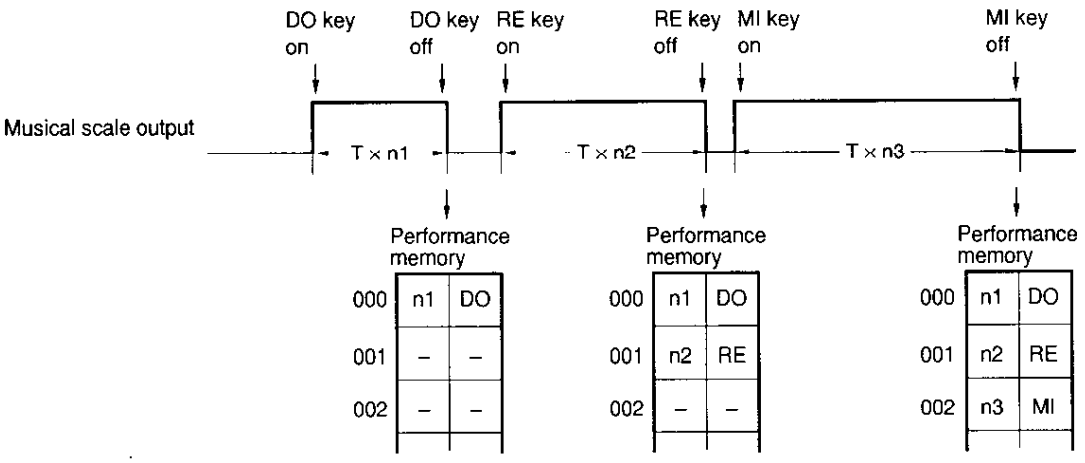
In PAUS mode (14-segment LCD)



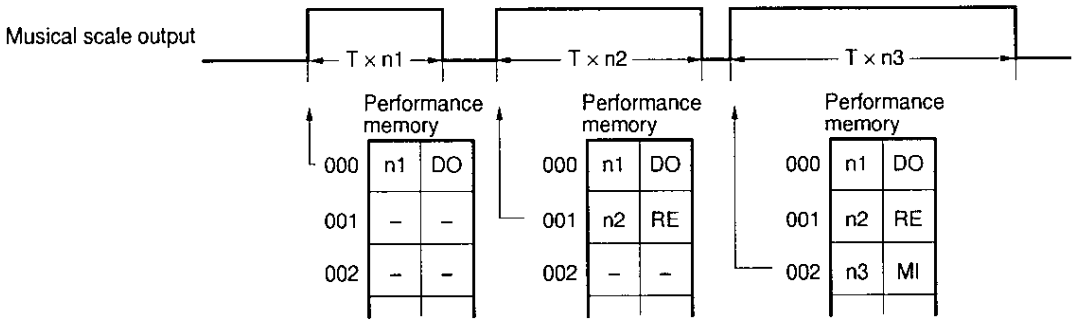
Pressing a musical scale key generates 8-bit data and reads it to the performance memory area sequentially. The lower 4 bits correspond to the musical scale data and the higher 4 bits correspond to the tone length data. The tone length data is a multiple of reference time (T).

When the tone length data is FXH, a repeat operation that repeats performance from the beginning is performed.

A maximum of 128 data items can be stored in the performance data memory.



Contrarily, data can also be extracted from the performance memory area to repeat performance.



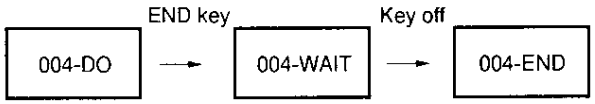
11.5.4 END key

In PROG mode, each press of the END key writes a repeat code (FnH). n is determined by the duration of key pressing.

In PLAY mode or PAUS mode, the END key returns the performance data address to the start.

An example of operation in PROG mode is shown below.

14-segment LCD



11.5.5 Other keys

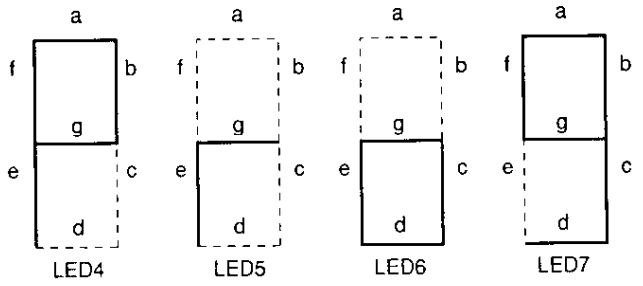
Pressing these keys performs no operation.

11.6 Explanation of Display

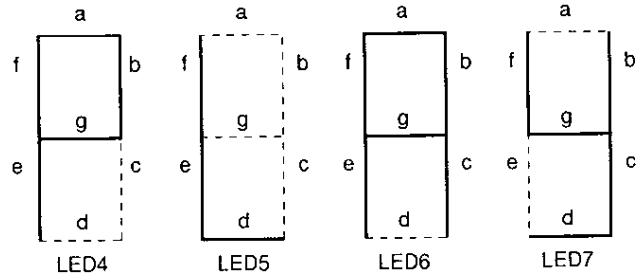
11.6.1 LED display

LED display is used for time display when mode display and reference time T are set.
Example of display patterns are shown below.

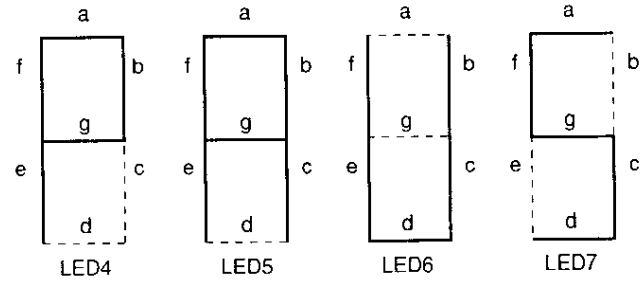
PROG mode display



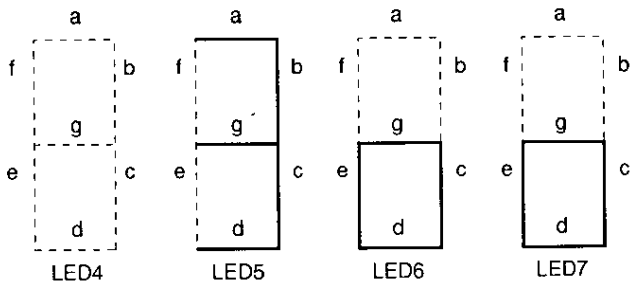
PLAY mode display



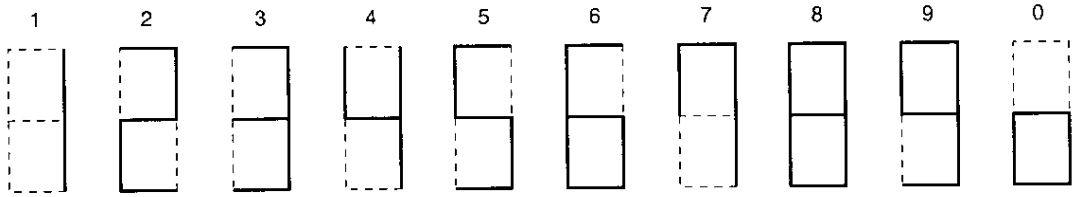
PAUS mode display



Reference time (T) display
(300 ms setting)



Number display pattern



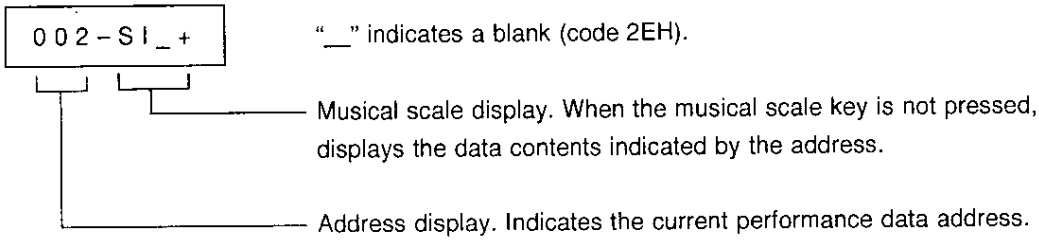
11.6.2 LCD display

This is used to display performance data and addresses.

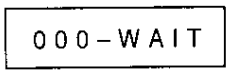
LCD display is performed by transferring character codes to the LCD controller/driver in SBI mode.

Display examples are shown below.

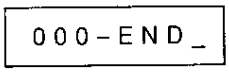
SI + key on



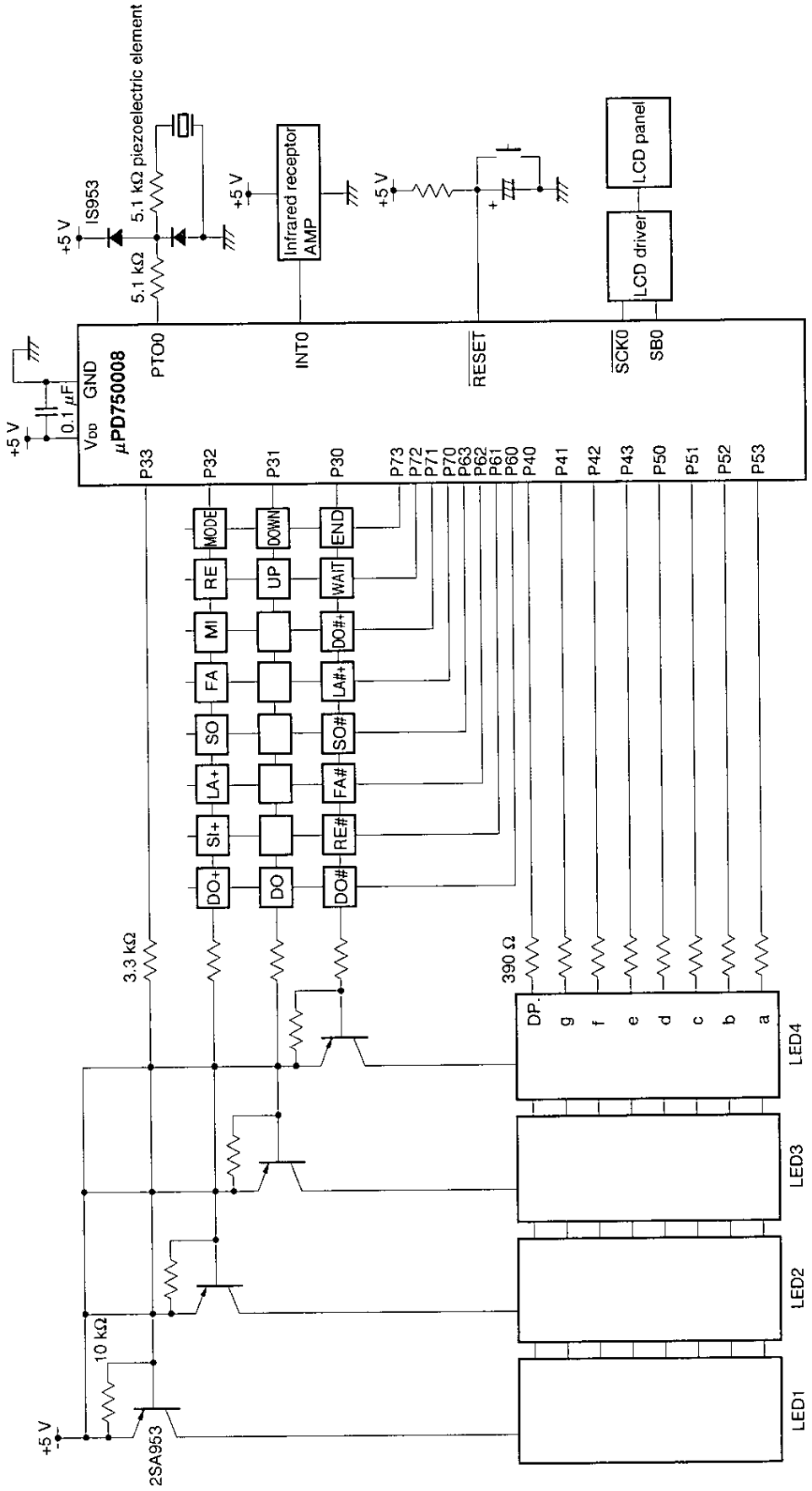
WAIT key on



When performance data is a repeat code



11.7 Hardware Configuration



11.8 Application Program

```
;************************************************************
;***                                                      ***
;***                  Application program                 ***
;***                                                      ***
;************************************************************
        NAME    MAIN                    ; Module name
        STKLN   40                      ; Stack size specification

    VENT0   MBE=0,RBE=0,INIT            ; RSET
    VENT1   MBE=0,RBE=1,INTBT           ; INTBT/INT4
    VENT2   MBE=0,RBE=1,INT0            ; INT0
    VENT4   MBE=0,RBE=1,SBI             ; SBI

DSEG0   DSEG    0       AT  20H         ; Stores from address 20H of memory bank 0.

KEYCODE:        DS      2               ; Performs coding of key source and key return.
KEYR:           DS      2               ; Key return
KEYRB:          DS      2
MUS:            DS      2               : Performance data address
KEYTIM:         DS      2               ; Key on time count timer
LCD:            DS      16              ; LCD display area
WORK:           DS      2               ; Stores data received with remote control.
RMDATA:         DS      2               ; Stores valid remote control data code.
KEYS:           DS      1               ; Key source
KEYSB:          DS      1
KEYCATT:        DS      1               ; Key chattering counter
KEYFLG:         DS      1               ; Flag
TONE:           DS      1               ; Musical scale data
TONEFLG:        DS      1               ; Flag
B_TIM:          DS      1               ; Base timer (1.95 ms count up)
B_TIMC:         DS      1               ; 50 ms, 100 ms count timer (9.75 ms count up)
DISPTIM:        DS      1               ; LED display timer
TONETIM:        DS      1               ; Tone length data
T_TIM:          DS      1               ; Reference time
T_TIMC:         DS      1               ; Reference time counter
RPTIM:          DS      1               ; Remote control repeat timer
RPCODE:         DS      1               ; Remote control repeat code counter
LEDDIG:         DS      1               ; LED display digit number
MODE:           DS      1               ; Mode setting 0:PROG, 1:PLAY, 2:PAUS
DEC:            DS      3               ; HEX → DEC
LCDTIM:         DS      1               ; 1s counter
LCDFLG:         DS      1               ; Flag
MODESBI:        DS      1               ; Indicates which data is being transferred.
LDCODE:         DS      1               ; Reader code scan counter
```

```
MODEP:          DS      1           ; Indicates which edge is being detected.
RMFLG:          DS      1           ; Flag
KEYONF          EQU     KEYFLG.3    ; If 1, key on
KEYON2SF        EQU     KEYFLG.2    ; If 1, key is held down 2 seconds or more.
HTKEYONF        EQU     KEYFLG.1    ; If 1, main unit key on
KEYCHKF         EQU     KEYFLG.0    ; If 1, ends key scan.

TONE_F          EQU     TONEFLG.0   ; If 1, enables musical scale output.

TM0_F           EQU     TONEFLG.1   ; If 1, timer count start

REFRES_F        EQU     LCDFLG.0    ; If 1, refreshes LCD.

REP_F           EQU     RMFLG.0     ; If 1, starts remote control repeat timer count.

;*****   LED character data   *****

LED_P           EQU     00110001B   ; "P"
LED_R           EQU     11110101B   ; "R"
LED_O           EQU     11000101B   ; "O"
LED_G           EQU     00001001B   ; "G"
LED_L           EQU     11100011B   ; "L"
LED_A           EQU     00010001B   ; "A"
LED_Y           EQU     10001001B   ; "Y"
LED_U           EQU     10000011B   ; "U"
LED_S           EQU     01001001B   ; "S"

LED_O           EQU     11000101B   ; "O"
LED__           EQU     11111111B   ; "_"

;*****    KEYR data      *****

KEYR_1          EQU     11111110B
KEYR_2          EQU     11111101B
KEYR_3          EQU     11111011B
KEYR_4          EQU     11110111B
KEYR_5          EQU     11101111B
KEYR_6          EQU     11011111B
KEYR_7          EQU     10111111B
KEYR_8          EQU     01111111B

;*****       MAIN ROUTINE     *****
MAIN_C  CSEG    INBLOCK

INIT:
;*********************************
;         Initial setting
;*********************************
        DI                          ; Disables all interrupts
        CLR1    MBE                 ; MBE ← 0
```

```
;++++++++++++++++++++++++++++++++
;       System clock setting
;++++++++++++++++++++++++++++++++
        MOV     A,#0011B
        MOV     PCC,A                   ; 0.95 μs, normal operating mode

;++++++++++++++++++++++++++++++++
;       Stack pointer setting
;++++++++++++++++++++++++++++++++
        MOV     XA,#00H
        MOV     SP,XA                   ; Stack pointer ← 00H
        MOV     SBS,A                   ; Memory bank ← 0, Mk II mode

;++++++++++++++++++++++++++++++++
;     Input/output port setting
;++++++++++++++++++++++++++++++++
        MOV     A,#0H
        OUT     PORT2,A
        OUT     PORT8,A

        MOV     A,#0FH
        OUT     PORT3,A
        OUT     PORT4,A
        OUT     PORT5,A
        OUT     PORT6,A
        OUT     PORT7,A

        MOV     XA,#0FH
        MOV     PMGS,XA                 ; Sets port 3 to output mode, and port 6 to input mode.

        MOV     XA,#34H
        MOV     PMGB,XA                 ; Sets ports 2, 4, 5 to output mode, and port 7 to input
                                          mode.
        MOV     XA,#00H
        MOV     PMGC,XA                 ; Sets port 8 to input mode.

        MOV     XA,#0C1H                ; Does not specify connection of on-chip pull-up resistor
                                          for ports 1, 2, 3.
        MOV     POGA,XA                 ; Specifies connection of on-chip pull-up resistor for ports
                                          0, 6, 7.
        MOV     XA,#0H
        MOV     POGB,XA                 ; Does not specify connection of on-chip pull-up resistor
                                          for port 8.

;++++++++++++++++++++++++++++++++
;        Hardware setting
;++++++++++++++++++++++++++++++++
        MOV     XA,#00H
        MOV     WM,XA                   ; Disables watch mode.
        MOV     TM0,XA                  ; Stops oscillation of timer/event counter.
        MOV     TM1,XA                  ; Stops oscillation of timer counter.
```

```
        SET1    TOE0                    ; Output enable flag ← 1
        MOV     A,#0000B
        MOV     IPS,A                   ; Sets high level interrupt.
        MOV     XA,#8AH
        MOV     CSIM,XA                 ; Sets serial operating mode.
        SET1    RELT

;++++++++++++++++++++++++++++++++
;         RAM clear setting
;++++++++++++++++++++++++++++++++
        SET1    MBE                     ; MBE ← 1
        SEL     MB0                     ; Memory bank ← 0
        MOV     A,#0H
        MOV     HL,#04H
LOOP1:
        MOV     @HL,A
        INCS    L
        BR      LOOP1

        INCS    H
        BR      LOOP1

        SEL     MB1                     ; Memory bank ← 1
        MOV     A,#0FH
LOOP2:
        MOV     @HL,A
        INCS    L
        BR      LOOP2

        INCS    H
        BR      LOOP2

        SEL     MB0                     ; Memory bank ← 0
        CLR1    MBE                     ; MBE ← 0

;++++++++++++++++++++++++++++++++
;           RAM setting
;++++++++++++++++++++++++++++++++
        MOV     A,#0DH
        MOV     LDCODE,A                ; Initializes LDCODE.
        MOV     A,#0111B
        MOV     LEDDIG,A                ; Initializes LEDDIG.
        MOV     A,#2H
        MOV     T_TIM,A                 ; Initializes T_TIM.

;++++++++++++++++++++++++++++++++
;  Basic interval timer setting
;++++++++++++++++++++++++++++++++
        MOV     A,#1111B
        MOV     BTM,A                   ; Timer start
```

```
        EI      IEBT                    ; Enables basic interval timer interrupt (INTBT).
        EI                              ; Enables all interrupts.

;++++++++++++++++++++++++++++++++
;           Main routine
;++++++++++++++++++++++++++++++++
MAIN:
        CALL    !RMDEC                  ; Remote control decoding
        CALL    !KEYDEC                 ; Key decode
        CALL    !LCDDATA                ; LCD display data setting
        CALL    !TONEOUT                ; Melody output
        CALL    !TIM
        BR      MAIN                    ; Timer processing

;*****    TABLE  LOOK  UP    *****
$IC=TABLE.INC

;*****      SUBROUTINESE     *****
$IC=SUB.INC

;++++++++++++++++++++++++++++++++
;     Remote control decoding
;++++++++++++++++++++++++++++++++
RMDEC:
        SKF     HTKEYONF                ; If 1, main unit key on
        RET

        SKTCLR  IRQT1                   ; If 1, modulo register = count register
        BR      MODEP7
        MOV     A,#0H
        MOV     MODEP,A                 ; MODEP ← 0H
        MOV     A,#0DH
        MOV     LDCODE,A                ; Initializes LDCODE.
        DI      IE0                     ; Disables INT0 interrupt.
        RET

MODEP7:
        MOV     A,MODEP
        SKE     A,#7H                   ; MODEP = 7?
        RET
        MOV     A,#0H
        MOV     MODEP,A                 ; MODEP ← OH
        MOV     RPCODE,A                ; Clears number of times of repeat code.
        MOV     PRTIM,A                 ; Clears 200 ms counter.
        MOV     XA,RMDATA
        MOV     HL,XA
        ADDS    XA,#0EDH
        BR      MODEP7_1
        RET
```

```
MODEP7_1:
        MOV     XA,HL
        MOV     BC,#HIGH  RM_T
        CALL    !TABLE
        MOV     BC,XA                   ; BC register ← RMDATA is coded.
        SKF     KEYONF
        RET
        SET1    KEYONF
        MOV     XA,#10H
        SKE     XA,BC
        BR      MODEP7_2
        CLR1    KEYON2SF
        MOV     XA,#0H
        MOV     KEYTIM,XA
MODEP7_2:
        MOV     XA,BC
        MOV     KEYCODE,XA
        SET1    KEYCHKF
        RET

;++++++++++++++++++++++++++++++++
;           Key decode
;++++++++++++++++++++++++++++++++
KEYDEC:
        SKT     KEYCHKF                 ; If 1, ends key scan.
        RET
        CLR1    KEYCHKF
        SKT     KEYONF                  ; If 1, key on
        BR      KEYOFF
        MOV     XA,KEYCODE
        ADDS    XA,#0F0H                ;  Musical scale key on?
        BR      ONKAI1
        ADDS    A,#0FH
        BR      MODE1                   ; If 0, MODE key on
        ADDS    A,#0FH
        BR      UPKEY1                  ; If 1, UP key on
        ADDS    A,#0FH
        BR      DOWN1                   ; If 2, DOWN key on
        BR      ENDKEY1                 ; If 3, END key on

;--------------------------
;   Musical scale key on
;--------------------------
ONKAI1:
        MOV     A,MODE                  ; PROG mode?
        SKE     A,#0H
        BR      ONKAI1_1
        MOV     A,#0H                   ; Tone length data ← 0H
```

```
        MOV     T_TIMC,A
        BR      ONKAI1_2

ONKAI1_1:
        SKE     A,#2H                   ; PAUS mode?
        RET
        MOV     A,#0FH                  ; Tone length data ← FH
ONKAI1_2:
        MOV     TONETIM,A
        MOV     A,KEYCODE               ; TONE ← KEYCODE
        MOV     TONE,A
        SET1    TONE_F
        RET

;--------------------------
;       MODE key (≥ 2s)
;--------------------------
MODE1:
        SKT     KEYON2SF
        RET
        MOV     A,MODE
        SKE     A,#0H                   ; PROG mode?
        BR      MODE1_1
        MOV     A,#2H
        MOV     MODE,A                  ; MODE ← PAUS mode
        RET
MODE1_1:
        MOV     A,#0H
        MOV     MODE,A                  ; MODE ← PROG mode
        CLR1    TONE_F
        RET

;--------------------------
;          UP key on
;--------------------------
UPKEY1:
        MOV     A,MODE
        SKE     A,#0H                   ; PROG mode?
        BR      UPKEY1_1
        MOV     XA,MUS
        ADDS    XA,#2H
        NOP
        MOV     MUS,XA                  ; Performance data address + 2H
        BR      DOWN1_1
UPKEY1_1:
        MOV     A,T_TIM
        SKE     A,#9H
        INCS    T_TIM                   ; Reference time + 1H
```

```
        NOP
        BR      DOWN1_3

;-------------------------
;       DOWN key on
;-------------------------
DOWN1:
        MOV     A,MODE
        SKE     A,#0H               ; PROG mode?
        BR      DOWN1_2
        MOV     XA,MUS
        DECS    XA
        NOP
        DES     XA
        NOP
        MOV     MUS,XA              ; Performance data address - 2
DOWN1_1:
        MOV     XA,MUS
        MOV     HL,XA
        INCS    HL
        NOP

        SET1    MBE                 ; MBE ← 1
        SEL     MB1                 ; Memory bank ← 1
        MOV     A,@HL
        NOP
        SEL     MB0                 ; Memory bank ← 0
        CLR1    MBE                 ; MBE ← 0

        MOV     TONE,A              ; Musical scale data ← @performance data address + 1
        SET1    TONE_F              ; Outputs performance data during key on
        RET

DOWN1_2:
        MOV     A,T_TIM
        SKE     A,#1H
        DECS    A
        NOP
        MOV     T_TIM,A             ; Reference time - 1

DOWN1_3:
        MOV     A,#0AH
        MOV     DISPTIM,A           ; Reference time display count start
        RET
```

```
;--------------------------
;       END key on
;--------------------------
ENDKEY1:
        MOV     A,MODE
        SKE     A,#0H                   ; PROG mode?
        BR      END2
        MOV     A,#0H
        MOV     TONETIM,A               ; Tone length count start
        MOV     T_TIMC,A
        BR      ONKAI2_1

END2:
        CLR1    TONE_F
        MOV     A,MODE
        SKE     A,#1H                   ; PLAY mode?
        BR      END2_2
        MOV     A,#0H
        MOV     TONETIM,A
        MOV     T_TIMC,A
END2_1:
        MOV     XA,#0H
        MOV     MUS,XA                  ; Performance data address ← 0H
        RET

END2_2:
        MOV     A,#0AH
        MOV     LCDTIM,A                ; LCD display timer (1 s) count start
        BR      END2_1

;--------------------------
;          key off
;--------------------------
KEYOFF:
        MOV     XA,KEYCODE
        ADDS    XA,#0F0H                ; Musical scale key off?
        BR      ONKAI2
;--------------------------
;       END key off
;--------------------------
        SKE     A,#3H                   ; END key off?
        BR      MODE2
        MOV     A,MODE
        SKE     A,#0H                   ; PROG mode?
        RET
        MOV     XA,MUS
        MOV     HL,XA
        MOV     A,TONETIM
        MOV     B,A
```

```
        SET1    MBE                     ; MBE ← 1
        SEL     MB1                     ; Memory bank ← 1
        MOV     A,#0FH
        MOV     @HL,A                   ; @Performance data address ← FH
        INCS    HL                      ; Performance data address + 1H
        NOP
        MOV     A,B
        MOV     #HL,A                   ; @Performance data address + 1 ← tone length data
        SEL     MB0                     ; Memory bank ← 0
        CLR1    MBE                     ; MBE ← 0
        RET

;--------------------------
;  Musical scale key off
;--------------------------
ONKAI2:
        MOV     A,MODE
        SKE     A,#0H                   ; PROG mode?
        BR      ONKAI2_2
        MOV     A,TONETIM               ; A register ← TONETIM
        CALL    !TONE_S
        MOV     XA,MUS
        ADDS    XA,#2H
        NOP
        MOV     MUS,XA                  ; Performance data address + 2H
        INCS    HL                      ; Performance data address + 1H
        NOP
        MOV     A,TONE                  ; A register ← TONE

        SET1    MBE                     ; MBE ← 1
        SEL     MB1                     ; Memory bank ← 1
        MOV     @HL,A                   ; @Performance data address + 1H ← TONE
        SEL     MB0                     ; Memory bank ← 0
        CLR1    MBE                     ; MBE ← 0

ONKAI2_1:
        CLR1    TONE_F                  ; Outputs silent tone.
        RET

ONKAI2_2:
        SKE     A,#2H                   ; PAUS mode?
        RET
        BR      ONKAI2_1
```

```
;-------------------------
;       Mode key (< 2s)
;-------------------------
MODE2:
        SKE     A,#0H                   ; MODE key off?
        BR      ONKAI2_1
        SKF     KEYON2SF
        RET
        MOV     A,MODE
        SKE     A,#0H                   ; PROG mode?
        BR      MODE2_4
        MOV     XA,MUS
        ADDS    XA,#0FFH                ; Performance data address = 00H?
        BR      MODE2_2
        BR      END2_1

MODE2_1:
        MOV     MUS,XA
MODE2_2:
        MOV     XA,MUS
        MOV     HL,XA

        SET1    MBE
        SEL     MB1
        MOV     A,@HL
        SEL     MB0
        CLR1    MBE

        SKE     A,#0FH                  ; @Performance data address = FH?
        BR      MODE2_3
        RET

MODE2_3:
        MOV     XA,MUS
        ADDS    XA,#2H                  ; Performance data address + 2H > 256?
        BR      MODE2_1
        BR      END2_1

MODE2_4:
        SKE     A,#1H                   ; PLAY mode?
        BR      MODE2_5
        MOV     A,#2H
        MOV     MODE,A                  ; MODE ← PAUS mode
        BR      ONKAI2_1

MODE2_5:
        MOV     A,#1H
        MOV     MODE,A                  ; MODE ← PLAY mode
        MOV     A,#0H
```

```
        MOV     TONETIM,A
        MOV     T_TIMC,A

;++++++++++++++++++++++++++++++++
;    LCD display data setting
;++++++++++++++++++++++++++++++++
LCDDATA:
        SKT     REFRES_F            ; If 1, refreshes LCD.
        RET
        CLR1    REFRES_F            ; REFRES_F ← 0
;------------------------------
;Address HEX → DEC conversion
;------------------------------
        MOV     A,#0H
        MOV     DEC,A               ; DEC ← 0H
        MOV     A,MUS
        XCH     A,E                 ; E register ← MUS

        MOV     A,MUS+1
LCDHEX1:
        MOV     HL,#DEC_1           ; Decimal correction (1st digit)
        ADDS    A,#6H
        BR      LCDHEX4

        MOV     @HL,A               ; Increments 2nd digit.
        INCS    L
        MOV     A,#7H
        ADDS    A,@HL               ; Decimal correction
        BR      LCDHEX4

        INCS    DEC                 ; Increments 3rd digit.
LCDHEX2:
        MOV     @HL,A               ; Adds 6 to number of times of 2nd digit value.
        DECS    E
        BR      LCDHEX3
        BR      LCD1

LCDHEX3:
        MOV     A,DEC+2
        ADDS    A,#6H
        BR      LCDHEX1

LCDHEX4:
        ADDS    A,#0AH
        NOP
        BR      LCDHEX2

LCD1:
        MOV     A,MODE
        SKE     A,#2H               ; PAUS mode?
```

```
        BR      LCD2
        MOV     A,LCDTIM
        SKE     A,#0H                   ; LCD display timer (1 s) = 0?
        BR      LCD2
        MOV     XA,#2EH
        MOV     LCD,XA                  ; Sets address 1st digit.
        MOV     LCD+2,XA                ; Sets address 2nd digit.
        MOV     LCD+4,XA                ; Sets address 3rd digit.
        BR      LCD3

LCD2:
        MOV     X,#0H
        MOV     A,DEC
        MOV     LCD,XA                  ; Sets address 1st digit.
        MOV     A,DEC+1
        MOV     LCD+2,XA                ; Sets address 2nd digit.
        MOV     A,DEC+2
        MOV     LCD+4,XA                ; Sets address 3rd digit.
LCD3:
        MOV     XA,#27H
        MOV     LCD+6,XA                ; "-" data setting
        MOV     XA,MUS
        MOV     HL,XA
        MOV     XA,@HL                  ; XA register ← @performance data address
        ADDS    XA,XA
        ADDS    XA,XA                   ; XA register × 4
        MOV     DE,XA
        MOV     BC,#HIGH  LCD_T
        CALL    !TABLE
        MOV     LCD+8,XA                ; Sets musical scale data 1st digit.
        CALL    !TABLEINC
        MOV     LCD+10,XA               ; Sets musical scale data 2nd digit.
        CALL    !TABLEINC
        MOV     LCD+12,XA               ; Sets musical scale data 3rd digit.
        CALL    !TABLEINC
        MOV     LCD+14,XA               ; Sets musical scale data 4th digit.
        SKT     PORT0.2                 ; SB0 = high level?
        RET
        SKT     PORT0.1                 ; SCK = high level?
        RET
        SET1    CMDT                    ; CMDT set
        NOP
        NOP
        SET1    RELT                    ; RELT set
        NOP
        NOP
        SKT     PORT0.2                 ; SB0 = high level?
        RET
        SET1    CMDT                    ; CMDT set
        MOV     XA,#01H
```

```
        MOV     SIO,XA              ; Shift register ← slave address (01H)
        MOV     A,#1H
        MOV     MODESBI,A           ; MODESBI ← 1H
        EI      IECSI               ; Enables SBI interrupt.
        MOV     XA,#1H
        MOV     SVA,XA              ; Slave address register ← 1H
        CLR1    REFRES_F            ; REFRES_F ← 0
        RET

;++++++++++++++++++++++++++++++++
;          Melody output
;++++++++++++++++++++++++++++++++
TONEOUT:
        SKT     TONE_F              ; If 1, enables musical scale output.
        BR      TONE0
        MOV     X,#0H
        MOV     A,TONE
        MOV     BC,#HIGH  TONE_T
        CALL    !TABLE
        MOV     TMOD0,XA            ; Modulo register ← musical scale data
        MOV     A,TONE
        ADDS    A,#2H
        BR      TONE1
TONE0:
        MOV     XA,#0H              ; Stops timer counter.
        CLR1    TM0_F               ; If 0, timer count start
        BR      TONE2

TONE1:
        SKF     TM0_F
        RET
        MOV     XA,#7CH             ; Timer count start
        SET1    TM0_F               ; If 1, stops timer counter.
TONE2:
        MOV     TM0,XA
        RET

;++++++++++++++++++++++++++++++++
;          Timer processing
;++++++++++++++++++++++++++++++++
TIM:
        MOV     A,B_TIMC
        SKE     A,#0AH              ; B_TIMC ≥ 100 ms?
        BR      TIM_50
;=========================
;     Elapse of 100 ms
;=========================
        MOV     A,#0H
        MOV     B_TIMC,A
        SKT     REP_F.
```

```
        BR      TIM_51
        INCS    RPTIM                   ; Remote control repeat timer + 1
        NOP
        MOV     A,RPTIM
        SKE     A,#2H                   ; Remote control repeat timer = 200 ms?
        BR      TIM_51
        MOV     A,#0H
        MOV     RPTIM,A                 ; Clears PRTIM.
        MOV     A,RPCODE
        ADDS    A,#0FH                  ; Remote control repeat code counter = 0
        BR      TIM_52
        ADDS    A,#0FH                  ; Remote control repeat code counter = 1
        BR      TIM_54
        ADDS    A,#0FH                  ; Remote control repeat code counter = 2
        BR      TIM_54
        ADDS    A,#0FH                  ; Remote control repeat code counter = 3
        BR      TIM_54

TIM_52:
        SKF     HTKEYONF
        BR      TIM_51
        SET1    KEYCHKF                 ; Remote control repeat code counter = 0 or ≥ 4
        CLR1    KEYONF                  ; KEYONF ← 0
        CLR1    REP_F                   ; REP_F ← 0
TIM_54:
        MOV     A,#0H
        MOV     RPCODE,A                ; Clears RPCODE.
TIM_51:
        MOV     XA,KEYCODE
        MOV     HL,#10H
        SKE     XA,HL
        BR      TIM_101
        SKF     KEYON2SF
        BR      TIM_101
        MOV     XA,KEYTIM
        MOV     HL,#14H
        SKE     XA,HL                   ; Key is held down 2 seconds or more?
        BR      TIM_105
        SET1    KEYON2SF                ; If 1, key is held down 2 seconds or more.
        SET1    KEYCHKF
TIM_101:
        MOV     A,DISPTIM
        SKE     A,#0H                   ; LED display timer (1 s) is being counted?
        BR      TIM_107
TIM_102:
        MOV     A,LCDTIM
        SKE     A,#0H                   ; LED display timer (1 s) is being counted?
        BR      TIM_108
TIM_103:
        MOV     A,MODE
```

```
        SKE     A,#0H                   ; PROG mode?
        BR      TIM_109
        INCS    T_TIMC                  ; Reference time counter + 1
        NOP
        MOV     A,T_TIM
        MOV     L,A
        MOV     A,T_TIMC
        SKE     A,L                     ; Reference time = reference time counter?
        BR      TIM_53
        MOV     A,#0H
        MOV     T_TIMC,A
        MOV     A,TONETIM
        SKE     A,#0EH                  ; Tone length data = EH?
        BR      TIM_104
        BR      TIM_53

TIM_104:
        INCS    TONETIM                 ; Tone length data + 1
        NOP
        BR      TIM_53

;--------------------------
;    Key on time count
;--------------------------
TIM_105:
        MOV     XA,KEYTIM
        MOV     HL,XA
        INCS    HL
        NOP
        MOV     XA,HL
        MOV     KEYTIM,XA               ; Key on time count timer + 1
        BR      TIM_101

;----------------------------
;LED display timer (1s) count
;----------------------------
TIM_107:
        MOV     A,DISPTIM
        DECS    A
        NOP
        MOV     DISPTIM,A               ; LED display (1 s) timer - 1
        BR      TIM_102
;----------------------------
;LCD display timer (1s) count
;----------------------------
TIM_108:
        MOV     A,LCDTIM
        DECS    A
        NOP
```

```
        MOV     LCDTIM,A            ; LCD display (1 s) timer – 1
        BR      TIM_103

TIM_109:
        SKE     A,#1H               ; PLAY mode?
        BR      TIM_53
        INCS    T_TIMC              ; Reference time counter + 1
        NOP
TIM_110:
        MOV     A,T_TIM
        MOV     L,A
        MOV     A,T_TIMC
        SKE     A,L                 ; Reference time = reference time counter?
        BR      TIM_53
        MOV     A,#0H
        MOV     T_TIMC,A
        MOV     A,TONETIM
        DECS    A
        BR      TIM_116
        MOV     XA,MUS

        SET1    MBE                 ; MBE ← 1
        SEL     MB1                 ; Memory bank ← 1
        MOV     HL,XA
        MOV     A,@HL+
        NOP
        MOV     B,A
        MOV     A,@HL
        MOV     C,A
        SEL     MB0                 ; Memory bank ← 0
        CLR1    MBE                 ; MBE ← 0

        MOV     A,B
        MOV     TONETIM,A           ; Tone length data ← @performance data address
        MOV     A,C
        MOV     TONE,A              ; Tone length data ← @performance data address + 1
        MOV     A,#0H
        MOV     T_TIMC,A
        SKE     C,#0FH
        BR      TIM_115
        BR      TIM_114

TIM_115:
        SKE     B,#0FH
        BR      TIM_111
        BR      TIM_112

TIM_112:
        MOV     XA,#0H
        MOV     MUS,XA
```

```
        MOV     A,TONE
        MOV     TONETIM,A           ; TONETIM ← repeat time
        MOV     A,#0FH
        MOV     TONE,A              ; TONE ← FH
TIM_114:
        CLR1    TONE_F
        SKE     B,#0FH
        BR      TIM_113
        BR      TIM_53

TIM_111:
        SET1    TONE_F
TIM_113:
        MOV     XA,MUS
        ADDS    XA,#2H
        NOP
        MOV     MUS,XA              ; Performance data address + 2
        BR      TIM_53

TIM_116:
        MOV     TONETIM,A           ; Tone length data – 1
        BR      TIM_53
;=========================
;      Elapse of 50 ms
;=========================
TIM_50:
        SKE     A,#5H               ; B_TIMC ≥ 50ms ?
        RET
;---------------------------------
;Remote control repeat timer counter
;---------------------------------
TIM_53:
        SET1    REFRES_F            ; If 1, refreshes LCD display.
        RET
;++++++++++++++++++++++++++++++++
;   Remote control processing
;++++++++++++++++++++++++++++++++
INT0:
        DI
        PUSH    BS
        SEL     RB1                 ; Register bank ← 1
        MOV     A,MODEP
        ADDS    A,#0FH              ; MODEP = 0?
        BR      INT0_E
        ADDS    A,#0FH              ; MODEP = 1?
        BR      INT0_P1
        ADDS    A,#0FH              ; MODEP = 2?
        BR      INT0_P2
```

```
        ADDS    A,#09H          ; MODEP = 3 – 6?
        BR      INT0_PX
        BR      INT0_E

INT0_P1:
        MOV     A,#2H
        MOV     MODEP,A         ; MODEP ← 2
        MOV     XA,#62H
        MOV     TMOD1,XA        ; Modulo register ← 6 ms
        MOV     A,#1H
        MOV     IMO,A           ; Specifies falling edge.
        BR      INT0_E

INT0_P2:
        MOV     XA,T1
        ADDS    XA,#0C9H        ; Count register = 3.375 ms?
        BR      INT0_RP1
        MOV     A,#3H
        MOV     MODEP,A         ; MODEP ← 3
        BR      INT0_W1

INT0_RP1:
        INCS    RPCODE          ; Remote control repeat code counter + 1
        NOP
        BR      INT0_P0
INT0_PX:
        MOV     XA,T1
        ADDS    XA,#0E4H        ; Count register = 1.6875 ms?
        BR      INT0_CY0
        SET1    CY              ; CY ← 1
        BR      INT0_L1

INT0_CY0:
        CLR1    CY              ; CY ← 0
INT0_L1:
        MOV     XA,WORK
        MOV     B,A
        MOV     A,X
        RORC    A
        MOV     X,A
        MOV     A,B
        RORC    A
        MOV     WORK,XA         ; Shifts 1 bit of receive data (WORK) to right.
        SKT     CY
        BR      INT0_E
        MOV     A,MODEP
        ADDS    A,#0CH          ; MODEP = 3?
        BR      INT0_P3
        ADDS    A,#0FH          ; MODEP = 4 ?
```

```
        BR      INT0_P4
        ADDS    A,#0FH                  ; MODE = 5?
        BR      INT0_P5
        MOV     XA,WORK                 ; MODEP = 6
        MOV     HL,#0FFH
        XOR     HL,XA
        MOV     XA,RMDATA
        SKE     XA,HL                   ; Valid remote control data = receive data inverted?
        BR      INT0_P0
        MOV     A,#7H
        MOV     MODEP,A                 ; MODEP ← 7
        SET1    REP_F
        DI      IE0                     ; Disables INT0 interrupt.
        MOV     XA,#0H
        MOV     TM1,XA                  ; Stops timer counter.
        BR      INT0_E1

INT0_P3:
        MOV     XA,WORK
        MOV     HL,#00H
        SKE     XA,HL                   ; Custom code = 00H?
        BR      INT0_P0
        MOV     A,#4H
        MOV     MODEP,A                 ; MODEP ← 4
        BR      INT0_W1

INT0_P4:
        MOV     XA,WORK
        MOV     HL,#0FFH
        SKE     XA,HL                   ; Custom code = FFH?
        BR      INT0_P0
        MOV     A,#5H
        MOV     MODEP,A                 ; MODEP ← 5
        BR      INT0_W1
INT0_P5:
        MOV     XA,WORK
        MOV     RMDATA,XA               ; Valid remote control data ← receive data
        MOV     A,#6H
        MOV     MODEP,A                 ; MODEP ← 6
        BR      INT0_W1

INT0_P0:
        MOV     A,#0H
        MOV     MODEP,A                 ; MODEP ← 0
        MOV     A,#0DH
        MOV     LDCODE,A                ; Initializes LDCODE.
        DI      IE0                     ; Disables INT0 interrupt.
        MOV     XA,#0H
        MOV     TM1,XA                  ; Stops timer counter.
        BR      INT0_E1
```

```
INT0_W1:
        MOV     XA,#42H
        MOV     TMOD1,XA            ; Modulo register ← 4 ms
        MOV     XA,#80H
        MOV     WORK,XA             ; Initializes WORK.
INT0_E:
        MOV     XA,#6CH
        MOV     TM1,XA              ; Timer count start
INT0_E1:
        POP     BS
        EI                          ; Enables all interrupts.
        RET1

;++++++++++++++++++++++++++++++++
;             Timer
;++++++++++++++++++++++++++++++++
INTBT:
        SEL     RB2                 ; Register bank ← 2
        DI      IECSI               ; Disables SBI interrupt.
        EI                          ; Enables all interrupts.
        CALL    !TIMCNT             ; Timer count
        CALL    !RMCNT              ; Remote control processing count
        CALL    !KEYCHK             ; Key check
        CALL    !LED1               ; LED display
        EI      IECSI               ; Enables SBI interrupt.
        RET1

;++++++++++++++++++++++++++++++++
;          SBI processing
;++++++++++++++++++++++++++++++++
SBI:
        SEL     RB3                 ; Register bank ← 3
        DI      IEBT                ; Disables basic interval timer interrupt (INTBT).
        EI                          ; Enables all interrupts.
        MOV     A,MODESBI
        ADDS    A,#0FH
        BR      SBI_E
        ADDS    A,#0FH
        BR      SBI1
        ADDS    A,#0FH
        BR      SBI2
        ADDS    A,#0FH
        BR      SBI3
        ADDS    A,#0FH
        BR      SBI4
        ADDS    A,#0FH
        BR      SBI5
        ADDS    A,#0FH
        BR      SBI6
```

```
        ADDS    A,#0FH
        BR      SBI7
        ADDS    A,#0FH
        BR      SBI8
        ADDS    A,#0FH
        BR      SBI9
        ADDS    A,#0FH
        BR      SBI10
        ADDS    A,#0FH
        BR      SBI11
        BR      SBI_E

SBI1:
        MOV     BC,#04H         ; Shift register ← pointer command (04H)
        BR      SBITIM

SBI2:
        MOV     BC,#07H         ; Shift register ← pointer (07H)
        BR      SBITIM

SBI3:
        MOV     BC,#01H         ; Shift register ← write command (01H)
        BR      SBITIM

SBI4:
        MOV     XA,LCD          ; Shift register ← address 1st digit
        MOV     BC,XA
        BR      SBITIM

SBI5:
        MOV     XA,LCD+2        ; Shift register ← address 2nd digit
        MOV     BC,XA
        BR      SBITIM

SBI6:
        MOV     XA,LCD+4        ; Shift register ← address 3rd digit
        MOV     BC,XA
        BR      SBITIM

SBI7:
        MOV     XA,LCD+6        ; Shift register ← "_" data
        MOV     BC,XA
        BR      SBITIM

SBI8:
        MOV     XA,LCD+8        ; Shift register ← musical scale data 1st digit
        MOV     BC,XA
        BR      SBITIM
```

```
SBI9:
        MOV     XA,LCD+10           ; Shift register ← musical scale data 2nd digit
        MOV     BC,XA
        BR      SBITIM

SBI10:
        MOV     XA,LCD+12           ; Shift register ← musical scale data 3rd digit
        MOV     BC,XA
        BR      SBITIM
SBI11:
        MOV     XA,LCD+14           ; Shift register ← musical scale data 4th digit
        MOV     BC,XA
SBITIM:
        MOV     L,#1110B            ; Initializes 100 μs counter.
SBICNT:
        DECS    L
        BR      SBI_ACKD

SBI0:
        MOV     A,#0H
        MOV     MODESBI,A           ; MODESBI ← 0
        SET1    REFRES_F            ; If 1, refreshes LCD display.
        BR      SBI_E

SBI_ACKD:
        SKT     ACKD                ; Acknowledge detection flag = 1?
        BR      SBICNT
        SKT     COI                 ; Address comparator signal = 1?
        BR      SBIO
        SKT     PORT0.2             ; SBO = high level?
        BR      SBI_E
        SKT     PORT0.1             ; SCK = high level?
        BR      SBI_E
        MOV     A,MODESBI
        ADDS    A,#0FH              ; MODESBI = 0?
        BR      SBI_SIO
        ADDS    A,#0FH              ; MODESBI = 1?
        BR      SBI_CMDT
        ADDS    A,#0FH              ; MODESBI = 2?
        BR      SBI_SIO
        ADDS    A,#0FH              ; MODESBI = 3?
SBI_CMDT:
        SET1    CMDT                ; CMDT set
SBI_SIO:
        MOV     XA,BC
        MOV     SIO,XA              ; Shift register ← BC register
        INCS    MODESBI             ; MODESBI + 1
        NOP
        SET1    CSIE                ; Serial interface operation, disables shift register.
        EI      IECSI               ; Enables SBI interrupt.
```

```
SBI_E:
        EI      IEBT                    ; Enables basic interval timer interrupt (INTBT).
        RETI
END
```

```
$NOLIST
;************************************************
;***                  Subroutine                ***
;************************************************
$LIST
$TT='SUBROUTINE'
TONE_S:
;++++++++++++++++++++++++++
;+++  Tone length data  +++
;++++++++++++++++++++++++++
        MOV     B,A
        MOV     XA,MUS
        MOV     HL,XA
        SET1    MBE                     ; MBE ← 1
        SEL     MB1                     ; Memory bank ← 1
        MOV     A,B
        MOV     @HL,A
        CLR1    MBE                     ; MBE ← 0
        RET
;++++++++++++++++++++++++++++++++
;           Timer count
;++++++++++++++++++++++++++++++++
TIMCNT:
        INCS    B_TIM                   ; Base timer + 1
        NOP
        MOV     A,B_TIM
        ADDS    A,#0BH                  ; Base timer > 10 ms?
        RET
        INCS    B_TIMC                  ; 100 ms count timer + 1
        NOP
        MOV     A,#0H
        MOV     B_TIM,A                 ; Base timer ← 0H
        RET

;++++++++++++++++++++++++++++++++
; Remote control processing count
;++++++++++++++++++++++++++++++++
RMCNT:

        SKF     HTKEYONF
        BR      RMCNT_1
        MOV     A,MODEP
        SKE     A,#0H                   ; MODEP = 0?
        RET
        SKF     PORT1.0
        BR      RMCNT_1
        INCS    LDCODE                  ; Reader code scan counter + 1
        RET
        MOV     A,#0DH                  ; Initializes LDCODE.
```

```
        MOV     LDCODE,A
        MOV     A,#1H
        MOV     MODEP,A             ; MODEP ← 1
        MOV     A,#0H
        MOV     IMO,A               ; Specifies rising edge.
        MOV     XA,#62H             ; Modulo register ← 6 ms
        MOV     TMOD1,XA
        MOV     XA,#6CH
        MOV     TM1,XA              ; Timer count start
        CLR1    IRQ0
        EI      IE0                 ; Enables INT0 interrupt.
        RET

RMCNT_1:
        MOV     A,#0DH
        MOV     LDCODE,A            ; Initializes LDCODE.
        RET

;++++++++++++++++++++++++++++++++
;           Key check
;++++++++++++++++++++++++++++++++
KEYCHK:
        MOV     A,LEDDIG
        SKE     A,#0111B            ; LEDDIG = 0111B?
        RET
        MOV     A,#0H
        OUT     PORT3,A             ; Port 3 ← 0
        MOV     XA,#0FFH
        OUT     PORT4,XA            ; Ports 4, 5 ← ALL1
        IN      XA,PORT6
        MOV     BC,#0FFH
        SKE     XA,BC               ; Ports 6, 7 = ALL1?
        BR      KEYCHK1
        SKT     HTKEYONF            ; If 1, main unit key on
        RET
        SKT     KEYONF              ; If 1, key on
        RET
        INCS    KEYCATT             ; Chattering counter + 1
        NOP
        MOV     A,KEYCATT
        ADDS    A,#0DH              ; Key matches 3 times?
        RET
        MOV     A,#0H
        MOV     KEYCATT,A           ; Chattering counter ← 0H
        CLR1    KEYONF              ; KEYONF ← 0
        SET1    KEYCHKF             ; If 1, ends key scan.
        CLR1    HTKEYONF
        RET
```

```
KEYCHK1:
        MOV     A,#1110B
        MOV     KEYS,A              ; Key source ← 1110B
KEYCHK2:
        OUT     PORT3,A             ; Port 3 ← 1110B
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        IN      XA,PORET6
        MOV     KEYR,XA             ; Key return ← ports 6, 7
        MOV     BC,#0FFH
        SKE     XA,BC               ; Ports 6, 7 = ALL1?
        BR      KEYCHK3
        SET1    CY                  ; CY ← 1
        MOV     A,KEYS
        ADDC    XA,XA
        MOV     KEYS,A              ; Shifts 1 bit of key source to left.
        SKE     A,#0111B
        BR      KEYCHK2
        RET

KEYCHK3:
        MOV     A,KEYS
        MOV     HL,#KEYSB
        SKE     A,@HL               ; Key source = KEYSB?
        BR      KEYCHK23
        MOV     A,KEYR
        MOV     HL,#KEYRB
        SKE     A,@HL               ; Key return = KEYRB?
        BR      KEYCHK23
        MOV     A,KEYR+1
        MOV     HL,#KEYRB+1
        SKE     A,@HL               ; Key return + 1 = KEYRB + 1?
        BR      KEYCHK23
        NCS     KEYCATT             ; Chattering counter + 1
        NOP
        MOV     A,KEYCATT
        ADDS    A,#0DH              ; Key matches 3 times?
        RET
        MOV     A,KEYS
        SKE     A,#1110B            ; Key source = 1110B?
        BR      KEYCHK11
        MOV     XA,KEYR
```

```
        MOV     HL,#KEYR_1
        SKE     XA,HL
        BR      KEYCHK4
        MOV     BC,#01H
        BR      KEYCHK22

KEYCHK4:
        MOV     HL,#KEYR_2
        SKE     XA,HL
        BR      KEYCHK5
        MOV     BC,#03H             ; KEYCODE ← 03H
        BR      KEYCHK22

KEYCHK5:
        MOV     HL,#KEYR_3
        SKE     XA,HL
        BR      KEYCHK6
        MOV     BC,#06H             ; KEYCODE ← 06H
        BR      KEYCHK22


KEYCHK6:
        MOV     HL,#KEYR_4
        SKE     XA,HL
        BR      KEYCHK7
        MOV     BC,#08H             ; KEYCODE ← 08H
        BR      KEYCHK22

KEYCHK7:
        MOV     HL,#KEYR_5
        SKE     XA,HL
        BR      KEYCHK8
        MOV     BC,#0AH             ; KEYCODE ← 0AH
        BR      KEYCHK22

KEYCHK8:
        MOV     HL,#KEYR_6
        SKE     XA,HL
        BR      KEYCHK9
        MOV     BC,#0DH             ; KEYCODE ← 0DH
        BR      KEYCHK22

KEYCHK9:
        MOV     HL,#KEYR_7
        SKE     XA,HL
        BR      KEYCHK10
        MOV     BC,#0FH             ; KEYCODE ← 0FH
        BR      KEYCHK22

KEYCHK10:
        MOV     HL,#KEYR_8
```

```
        SKE     XA,HL
        RET
        MOV     BC,#13H             ; KEYCODE ← 13H
        BR      KEYCHK22

KEYCHK11:
        SKE     A,#1101B            ; Key source = 1101B?
        BR      KEYCHK14
        MOV     XA,KEYR
        MOV     HL,#KEYR_1
        SKE     XA,HL
        BR      KEYCHK12
        MOV     BC,#00H             ; KEYCODE ← 00H
        BR      KEYCHK22

KEYCHK12:
        MOV     HL,#KEYR_7
        SKE     XA,HL
        BR      KEYCHK13
        MOV     BC,#11H             ; KEYCODE ← 11H
        BR      KEYCHK22

KEYCHK13:
        MOV     HL,#KEYR_8
        SKE     XA,HL
        RET
        MOV     BC,#12H             ; KEYCODE ← 12H
        BR      KEYCHK22

KEYCHK14:
        SKE     A,#1011B            ; Key source = 1011B?
        RET
        MOV     XA,KEYR
        MOV     HL,#KEYR_1
        SKE     XA,HL
        BR      KEYCHK15
        MOV     BC,#0CH             ; KEYCODE ← 0CH
        BR      KEYCHK22

KEYCHK15:
        MOV     HL,#KEYR_2
        SKE     XA,HL
        BR      KEYCHK16
        MOV     BC,#0BH             ; KEYCODE ← 0BH
        BR      KEYCHK22

KEYCHK16:
        MOV     HL,#KEYR_3
        SKE     XA,HL
        BR      KEYCHK17
```

```
        MOV     BC,#09H             ; KEYCODE ← 09H
        BR      KEYCHK22

KEYCHK17:
        MOV     HL,#KEYR_4
        SKE     XA,HL
        BR      KEYCHK18
        MOV     BC,#07H             ; KEYCODE ← 07H
        BR      KEYCHK22

KEYCHK18:
        MOV     HL,#KEYR_5
        SKE     XA,HL
        BR      KEYCHK19
        MOV     BC,#05H             ; KEYCODE ← 05H
        BR      KEYCHK22

KEYCHK19:
        MOV     HL,#KEYR_6
        SKE     XA,HL
        BR      KEYCHK20
        MOV     BC,#04H             ; KEYCODE ← 04H
        BR      KEYCHK22

KEYCHK20:
        MOV     HL,#KEYR_7
        SKE     XA,HL
        BR      KEYCHK21
        MOV     BC,#02H             ; KEYCODE ← 02H
        BR      KEYCHK22

KEYCHK21:
        MOV     HL,#KEYR_8
        SKE     XA,HL
        RET
        MOV     BC,#10H             ; KEYCODE ← 10H
KEYCHK22:
        SKF     HTKEYONF
        RET
        SET1    HTKEYONF            ; If 1, main unit key on
        SET1    KEYONF
        MOV     XA,#10H
        SKE     XA,BC
        BR      KEYCHK24
        CLR1    KEYON2SF
        MOV     XA,#0H
        MOV     KEYTIM,XA
KEYCHK24:
        MOV     XA,BC
        MOV     KEYCODE,XA
```

```
        SET1    KEYCHKF             ; If 1, ends key scan.
        RET

KEYCHK23:
        MOV     A,KEYS
        MOV     KEYSB,A             ; KEYSB ← key source
        MOV     XA,KEYR
        MOV     KEYRB,XA            ; KEYRB ← key return
        MOV     A,#0H
        MOV     KEYCATT,A           ; Chattering counter ← 0
        RET

;++++++++++++++++++++++++++++++++
;           LED display
;++++++++++++++++++++++++++++++++
LED1:
        SET1    CY
        MOV     A,LEDDIG
        RORC    A                   ; Shifts 1 bit of LEDDIG to right.
        MOV     LEDDIG,A
        SKT     CY
        BR      LED2

LEDMODE0:
        MOV     A,MODE
        SKE     A,#0H
        BR      LEDMODE1
        SKT     LEDDIG.3            ; Displays "PROG" mode
        BR      LED4P
        SKT     LEDDIG.2
        BR      LED5R
        SKT     LEDDIG.1
        BR      LED60
        MOV     HL,#LED_G           ; LED7 ← displays "G."
LED4P
        MOV     HL,#LED_P           ; LED4 ← displays "P."
LED5R:
        MOV     HL,#LED_R           ; LED5 ← displays "R."
LED60:
        MOV     HL,#LED_O           ; LED6 ← displays "O."
        BR      LED_OUT

LED2:
        MOV     A,#0111B
        MOV     LEDDIG,A            ; Initializes LEDDIG.
        BR      LEDMODE0

LEDMODE1:
        SKE     A,#1H
        BR      LEDMODE2
```

```
        SKT     LEDDIG.3            ; Displays "PLAY" mode.
        BR      LED4P1
        SKT     LEDDIG.2
        BR      LED5L
        SKT     LEDDIG.1
        BRE     LED6A
        MOV     HL,#LED_Y           ; LED7 ← displays "Y."
LED4P1:
        MOV     HL,#LED_P           ; LED4 ← displays "P."
LED5L:
        MOV     HL,#LED_L           ; LED5 ← displays "L."
LED6A:
        MOV     HL,#LED_A           ; LED6 ← displays "A."
        BR      LED_CNT

LEDMODE2:
        SKT     LEDDIG.3            ; Displays "PAUS" mode.
        BR      LED4P1
        SKT     LEDDIG.2
        BR      LED5A
        SKT     LEDDIG.1
        BR      LED6U
        MOV     HL,#LED_S           ; LED7 ← displays "S."
LED5A:
        MOV     HL,#LED_A           ; LED5 ← displays "A."
LED6U:
        MOV     HL,#LED_U           ; LED6 ← displays "U."
LED_CNT:
        MOV     A,DISPTIM
        ADDS    A,#0FH
        BR      LED_OUT
        SKT     LEDDIG.3            ; Displays reference time.
        BR      LED40
        SKT     LEDDIG.2
        BR      LED5TIM
        MOV     HL,#LED_0           ; LED6, 7 ← displays "O."
LED40:
        MOV     HL,#LED__           ; LED4 ← displays "_"
        BR      LED_OUT

LED5TIM:
        MOV     X,#0H               ; LED5 ← displays 3rd digit of time.
        MOV     A,T_TIM
        MOV     BC,#HIGH LED_T
        CALL    !TABLE
        MOV     HL,XA
LED_OUT:
        MOV     A,#1111B
        OUT     PORT3,A             ; Turns off digit output.
        MOV     XA,HL
```

```
OUT     PORT4,XA            ; Outputs segment.
MOV     A,LEDDIG
OUT     PORT3,A             ; Digit output
RET
```

```
$TT='TABLE'
;*********************************
;***     Table reference area    ***
;*********************************

RM              CSEG     PAGE

RM_T:
;++++++++++++++++++++++++++++++++++
;+++      RMDATA → KEYCODE       +++
;++++++++++++++++++++++++++++++++++
DB          10H                  ; MODE
DB          11H                  ; UP
DB          12H                  ; DOWN
DB          13H                  ; END
DB          0FH                  ; WAIT
DB          0DH                  ; DO# +
DB          0AH                  ; LA# +
DB          08H                  ; SO#
DB          06H                  ; FA#
DB          03H                  ; RE#
DB          01H                  ; DO#
DB          0CH                  ; DO +
DB          0BH                  ; SI +
DB          09H                  ; LA +
DB          07H                  ; SO
DB          05H                  ; FA
DB          04H                  ; MI
DB          02H                  ; RE
DB          00H                  ; DO

TONE_C          CSEG     PAGE

TONE_T:
;++++++++++++++++++++++++++
;+++ Musical scale data +++
;++++++++++++++++++++++++++
DB          0F9H                 ; DO
DB          0EBH                 ; DO#
DB          0DEH                 ; RE
DB          0D1H                 ; RE#
DB          0C6H                 ; MI
DB          0BAH                 ; FA
DB          0B0H                 ; FA#
DB          0A6H                 ; SO
DB          09CH                 ; SO#
DB          094H                 ; LA +
DB          08BH                 ; LA# +
DB          083H                 ; SI +
```

```
DB          07CH                ; DO +
DB          075H                ; DO# +
DB          0FFH                ; Silent tone
DB          0FFH                ; Silent tone

LCD_C         CSEG     PAGE

LCD_T:
;++++++++++++++++++++++++++++++++++
;+++     LCD display pattern      +++
;++++++++++++++++++++++++++++++++++
DB          0EH                 ; "DO "
DB          19H
DB          2EH
DB          2EH

DB          0EH                 ; "DO# "
DB          19H
DB          25H
DB          2EH

DB          1CH                 ; "RE "
DB          0FH
DB          2EH
DB          2EH

DB          1CH                 ; "RE# "
DB          0FH
DB          25H
DB          2EH

DB          17H                 ; "MI "
DB          13H
DB          2EH
DB          2EH

DB          10H                 ; "FA "
DB          0BH
DB          2EH
DB          2EH

DB          10H                 ; "FA# "
DB          0BH
DB          25H
DB          2EH

DB          1DH                 ; "SO "
DB          19H
DB          2EH
DB          2EH
```

```
DB          1DH                 ; "SO# "
DB          19H
DB          25H
DB          2EH

DB          1CH                 ; "LA +"
DB          0BH
DB          2EH
DB          26H

DB          1CH                 ; "LA# +"
DB          0BH
DB          25H
DB          26H

DB          1DH                 ; "SI +"
DB          13H
DB          2EH
DB          26H

DB          0EH                 ; "DO +"
DB          19H
DB          2EH
DB          26H

DB          0EH                 ; "DO# +"
DB          19H
DB          25H
DB          26H

DB          21H                 ; "WAIT "
DB          0BH
DB          13H
DB          1EH

DB          0FH                 ; "END "
DB          18H
DB          0EH
DB          2EH

LED_C       CSEG    PAGE

LED_T:
```

```
;++++++++++++++++++++++++++
;+++    LED segment     +++
;++++++++++++++++++++++++++
;Address     abcdefg.        Display
DB          11000101B       ;"0"
DB          10011111B       ;"1"
DB          00100101B       ;"2"
DB          00001101B       ;"3"
DB          10011001B       ;"4"
DB          01001001B       ;"5"
DB          01000001B       ;"6"
DB          00011011B       ;"7"
DB          00000001B       ;"8"
DB          00001001B       ;"9"

;++++++++++++++++++++++++++++++++
;+++ Table reference subroutine +++
;++++++++++++++++++++++++++++++++
TABLEINC:
        INCS    DE              ; Table Address + 1
        MOV     XA,DE
TABLE:
        MOVT    XA,@BCXA
        RET
```

[MEMO]

NEC

Facsimile Message

From:

Name

Company

Tel. FAX

Address

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

Thank you for your kind support.

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

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

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

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

Korea
NEC Electronics Hong Kong Ltd.
Seoul Branch
Fax: 02-551-0451

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

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

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

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

Document title: _____

Document number: _____ Page number: _____

If possible, please fax the referenced page or drawing.

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

CS 95.12