# HT46R63/HT46C63
# 8-Bit A/D with LCD Type MCU

## Features

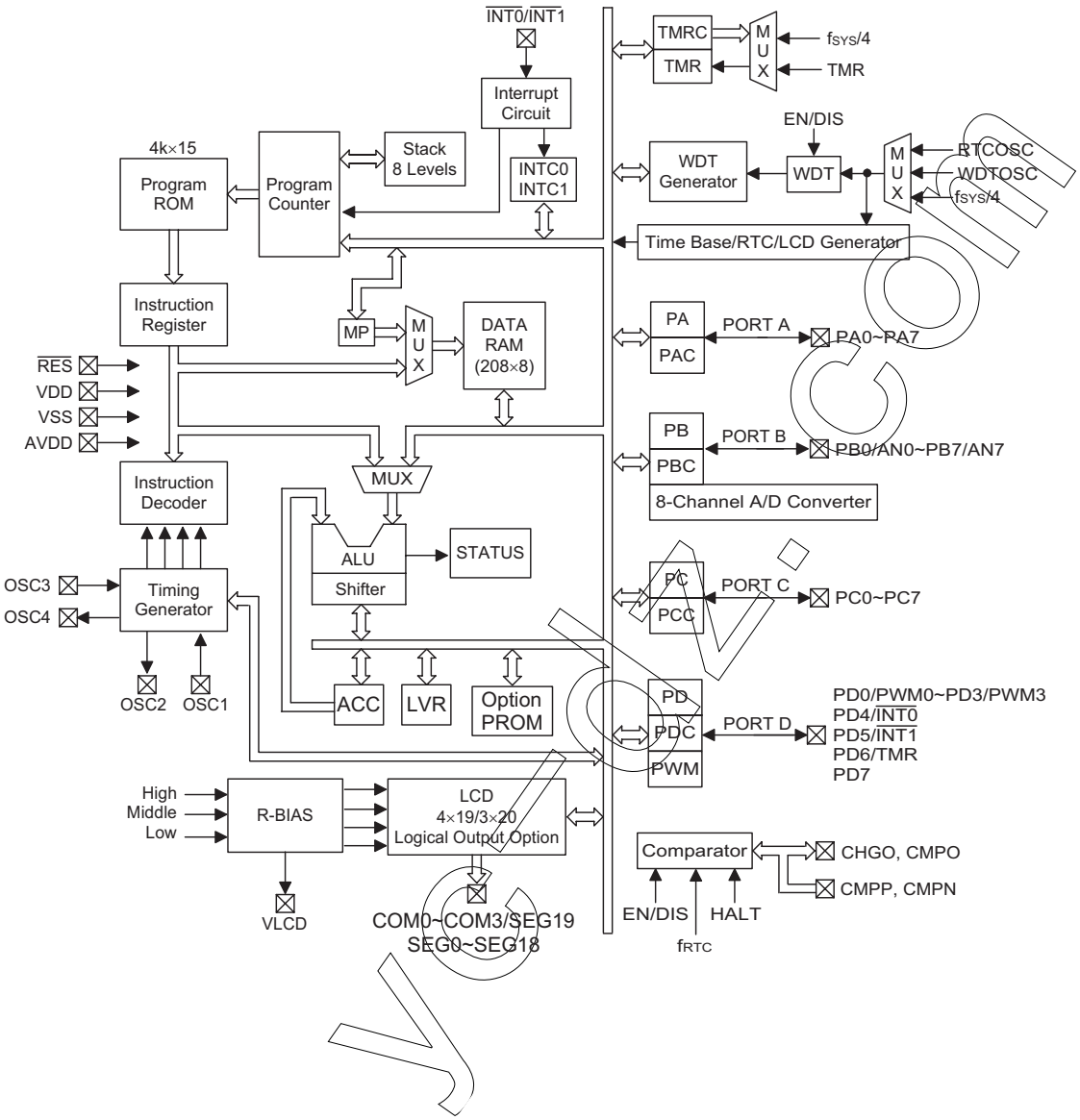
- Operating voltage:
  $f_{SYS}$=4MHz: 2.2V~5.5V
  $f_{SYS}$=8MHz: 3.3V~5.5V
- Operating frequency: External RC or Crystal
- 32.768kHz crystal oscillator used for timing purposes
- Watchdog enable or disable function
- 1x16 bits timer with an overflow interrupt (TMR)
- Time base generator (clock source: 32.768kHz) and RTC interrupts
- 4K×15 program memory
- 208×8 data memory RAM
- Maximum of 32 I/O lines (shared with $\overline{INT0}$, $\overline{INT1}$, TMR, AN0~AN7, PWM0~PWM3)
- 8-level stack
- Up to 0.5μs instruction cycle with 8MHz system clock at $V_{DD}$=5V
- 2 external interrupts (high/low going trigger)
- One comparator
- LCD: 20×3 or 19×4, 1/3 bias with 12 pins logical outputs options. (select by options in unit of 4 pins, ×8 high sink)
- Built-in R type bias generator
- 8 channels 8-bits resolution (7-bit accuracy) A/D converter
- 4 channels PWM outputs
- 56-pin SSOP, 100-pin QFP package

## General Description

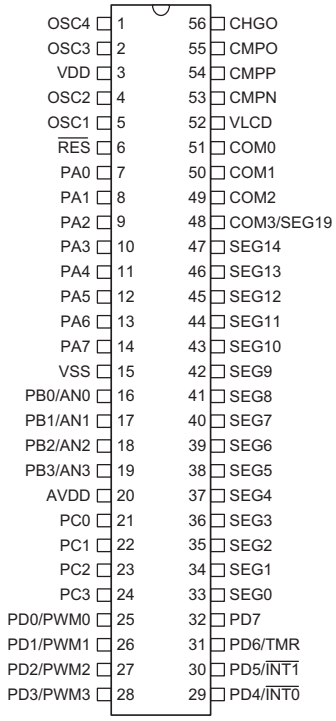
The HT46R63/HT46C63 are 8-bit, high performance, RISC architecture microcontroller devices specifically designed for A/D product applications that interface directly to analog signals and which require LCD Interface. The mask version HT46C63 is fully pin and functionally compatible with the OTP version HT46R63 device.

The advantages of low power consumption, I/O flexibility, timer functions, oscillator options, multi-channel A/D Converter, Pulse Width Modulation function, HALT and wake-up functions, in addition to a flexible and configurable LCD interface enhance the versatility of these devices to control a wide range of applications requiring analog signal processing and LCD interfacing, such as electronic metering, environmental monitoring, handheld measurement tools, motor driving, etc., for both industrial and home appliance application areas.

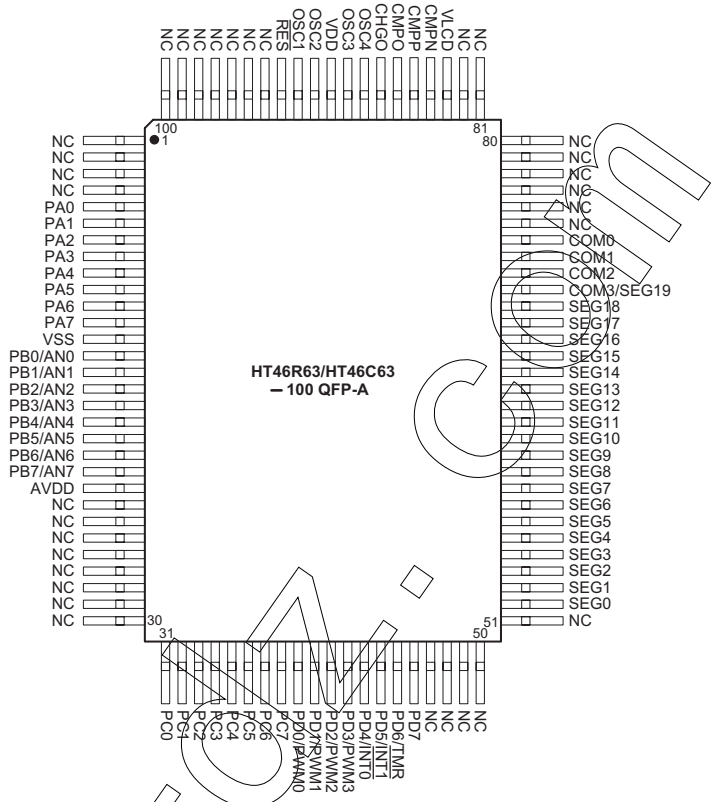
## Block Diagram



## Pin Assignment

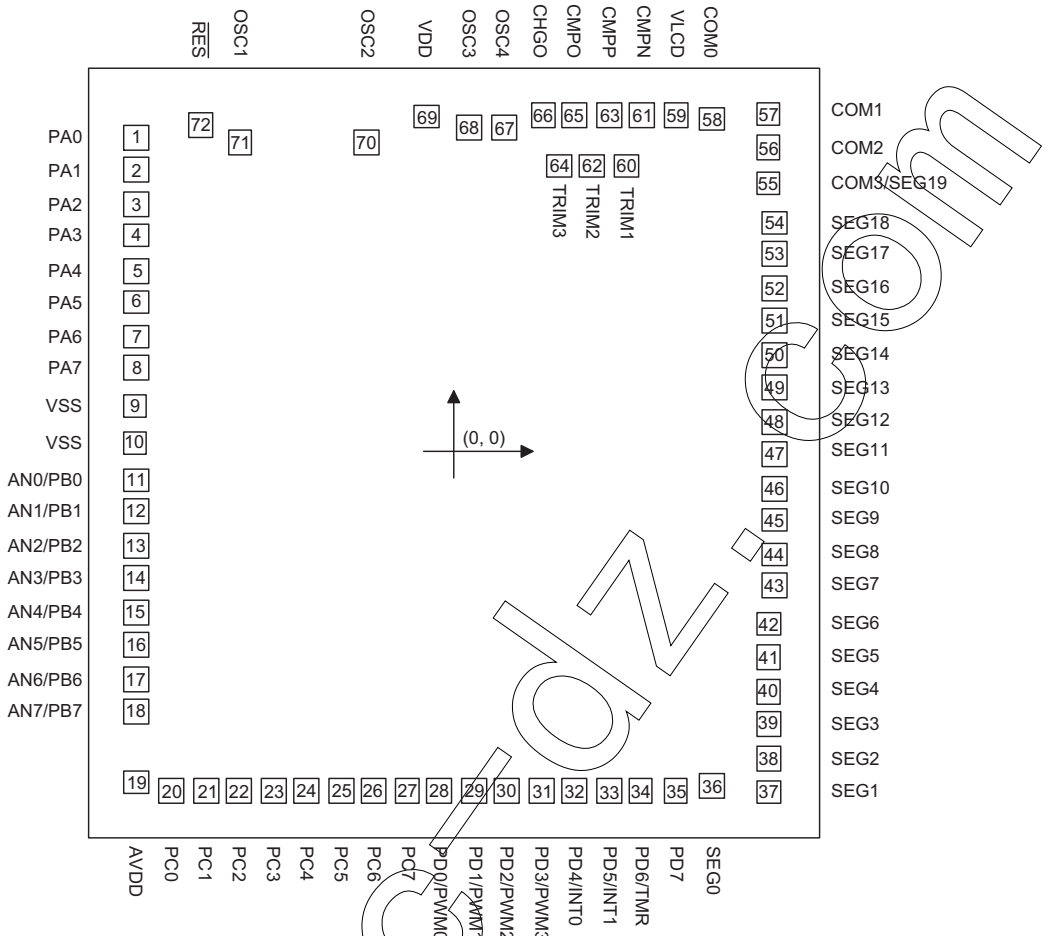



**HT46R63/HT46C63**
**— 56 SSOP-A**



## Pad Assignment

**HT46C63**




* The IC substrate should be connected to VSS in the PCB layout artwork.

## Pin Description

| Pin Name | I/O | Option | Description |
|---|---|---|---|
| PA0~PA7 | I/O | Pull-high<br>Wake-up | I/O lines with pull-high resistors (bit option). I/O modes of each line are controlled by related control register bit (PAC). Each line of PA can be optioned as a wake-up input (bit option). I/O configurations: Schmitt trigger/CMOS |
| PB0/AN0~<br>PB7/AN7 | I/O | Pull-High | I/O lines with pull-high resistors (bit option). I/O modes of each line are controlled by related control register bit (PBC). I/O configurations: Schmitt trigger/CMOS. Each PB line is pin shared with an A/D converter input. |
| PC0~PC6,<br>PC7 | I/O | Pull-High | I/O lines with pull-high resistors (bit option). I/O modes of each line are controlled by related control register bit (PCC). I/O configurations: Schmitt trigger/CMOS. |
| PD0/PWM0~<br>PD3/PWM3,<br>PD4/INT0,<br>PD5/INT1,<br>PD6/TMR,<br>PD7 | I/O | Pull-High PWM<br>Interrupt Falling<br>and/or Rising | I/O lines with pull-high resistors (bit option). I/O modes of each line are controlled by related control register bit (PDC). I/O configurations: Schmitt trigger/CMOS. The PD0~PD3 can be selected as PWM outputs. $\overline{\text{INT0}}$/$\overline{\text{INT1}}$ are falling/rising edge selectable triggers. |

| Pin Name | I/O | Option | Description |
|---|---|---|---|
| OSC1<br>OSC2 | I<br>O | RC or crystal | A resistor across OSC1 and VDD or a crystal across OSC1 and OSC2 will generate a system clock. |
| OSC3<br>OSC4 | I<br>O | — | 32768Hz crystal across OSC3 and OSC4 will generate RTC clock signal which only provides system timing. |
| CMPN | I | — | Negative input for comparator |
| CMPP | I | — | Positive input for comparator |
| CMPO | O | — | Comparator output |
| CHGO | O | — | Comparator output with 32768Hz carrier |
| VDD | — | — | Positive power supply |
| AVDD | — | — | A/D converter Positive power supply, AVDD should be externally connected to VDD |
| VSS | — | — | Negative power supply, ground |
| $\overline{RES}$ | I | — | Schmitt trigger reset input |
| VLCD | I/O | — | LCD highest voltage; should be connected to VDD with external resistor. |
| SEG0~SEG18 | O | SEG7~SEG18 logical CMOS | LCD segment signal driving outputs SEG7~SEG10 can be optioned as output lines. SEG11~SEG14, SEG15~SEG18 can be optioned as a high sinking output lines. |
| COM0~COM2<br>COM3/SEG19 | O | COM3 or SEG19 | LCD common signal driving outputs |

## Absolute Maximum Ratings

Supply Voltage ..... $V_{SS}$–0.3V to $V_{SS}$+6.0V
Input Voltage ..... $V_{SS}$–0.3V to $V_{DD}$+0.3V
Storage Temperature ..... –50°C to 125°C
Operating Temperature ..... –40°C to 85°C

Note: These are stress ratings only. Stresses exceeding the range specified under "Absolute Maximum Ratings" may cause substantial damage to the device. Functional operation of this device at other conditions beyond those listed in the specification is not implied and prolonged exposure to extreme conditions may affect device reliability.

## D.C. Characteristics

Ta=25°C

| Symbol | Parameter | Test Conditions | | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|---|---|
| | | $V_{DD}$ | Conditions | | | | |
| $V_{DD}$ | Operating Voltage | — | $f_{SYS}$=4MHz | 2.2 | — | 5.5 | V |
| | | | $f_{SYS}$=8MHz | 3.3 | — | 5.5 | V |
| $I_{DD1}$ | Operating Current (RC OSC) (Analog Circuit Disabled) | 3V | No load, $f_{SYS}$=4MHz | — | 1 | 2 | mA |
| | | 5V | | — | 3 | 5 | |
| $I_{DD2}$ | Operating Current (RC OSC) | 3V | No load, $f_{SYS}$=4MHz | — | 1 | 2 | mA |
| | | 5V | | — | 3 | 5 | |
| $I_{DD3}$ | Operating Current | 5V | No load, $f_{SYS}$=8MHz | — | 3 | 5 | mA |
| $I_{STB1}$ | Standby Current (WDT OSC On, RTC Off, LCD Off) | 3V | No load, System HALT | — | — | 5 | μA |
| | | 5V | | — | — | 15 | |
| $I_{STB2}$ | Standby Current (WDT OSC Off, RTC Off, LCD Off) | 3V | System HALT | — | — | 1 | μA |
| | | 5V | | — | — | 1 | |

| Symbol | Parameter | Test Conditions | | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|---|---|
| | | $V_{DD}$ | Conditions | | | | |
| $I_{STB3}$ | Standby Current (WDT OSC Off, RTC On, LCD Off) | 3V | System HALT | — | — | 5 | μA |
| | | 5V | | — | — | 15 | |
| $I_{STB4}$ | Standby Current (WDT OSC Off, RTC On, LCD On with Low Current Internal R Type Bias Option) | 3V | System HALT $V_{LCD}=V_{DD}$ | 10 | 12 | 16 | μA |
| | | 5V | | 20 | 24 | 32 | |
| $I_{STB5}$ | Standby Current (WDT OSC Off, RTC On, LCD On with Middle Current Internal R Type Bias Option) | 3V | System HALT $V_{LCD}=V_{DD}$ | 16 | 20 | 26 | μA |
| | | 5V | | 32 | 40 | 52 | |
| $I_{STB6}$ | Standby Current (WDT OSC Off, RTC On, LCD On with High Current Internal R Type Bias Option) | 3V | System HALT $V_{LCD}=V_{DD}$ | 38 | 52 | 68 | μA |
| | | 5V | | 76 | 104 | 136 | |
| $V_{IL1}$ | Input Low Voltage for I/O Ports | — | — | 0 | — | $0.3V_{DD}$ | V |
| $V_{IH1}$ | Input High Voltage for I/O Ports | — | — | $0.7V_{DD}$ | — | 3 | V |
| $V_{IL2}$ | Input Low Voltage ($\overline{RES}$) | — | — | 0 | — | $0.4V_{DD}$ | V |
| $V_{IH2}$ | Input High Voltage ($\overline{RES}$) | — | — | $0.9V_{DD}$ | — | $V_{DD}$ | V |
| $V_{LCD}$ | LCD Highest Voltage | — | — | 0 | — | $V_{DD}$ | V |
| $I_{OH1}$ | I/O Port Source Current | 3V | $V_{OH}=0.9V_{DD}$ | –2 | –4 | — | mA |
| | | 5V | | –5 | –8 | — | |
| $I_{OL1}$ | I/O Port Sink Current | 3V | $V_{OL}=0.1V_{DD}$ | 6 | 12 | — | mA |
| | | 5V | | 10 | 25 | — | |
| $I_{OH2}$ | SEG7~18 Logical Source Current | 3V | $V_{OH}=0.9V_{DD}$ | –2 | –4 | — | mA |
| | | 5V | | –4 | –8 | — | |
| $I_{OL2}$ | SEG7~10 Logical Sink Current | 3V | $V_{OL}=0.1V_{DD}$ | 8 | — | — | mA |
| | | 5V | | 16 | — | — | |
| $I_{OL3}$ | SEG11~18 Logical Sink Current | 3V | $V_{OL}=0.1V_{DD}$ | 16 | — | — | mA |
| | | 5V | | 32 | — | — | |
| $I_{OHTOTAL}$ | I/O Port Total Source Current | — | — | — | — | –100 | mA |
| $I_{OLTOTAL}$ | I/O Port Total Sink Current | — | — | — | — | 100 | mA |
| $R_{PH}$ | Pull-High Resistance (I/O) | 3V | — | 40 | 60 | 80 | kΩ |
| | | 5V | | 10 | 30 | 50 | |
| $V_{OS}$ | Comparator Input Offset Voltage | — | — | –10 | — | 10 | mV |
| $V_I$ | Comparator Input Voltage Range | — | — | 0.2 | — | $V_{DD}$–0.8 | V |
| $V_{AD}$ | A/D Input Voltage | — | — | 0 | — | $V_{DD}$ | V |
| $E_{AD}$ | A/D Conversion Integral Nonlinearity Error | — | — | — | ±0.5 | ±1 | LSB |
| $I_{ADC}$ | Additional Power Consumption if A/D Converter is Used | 3V | — | — | 0.5 | 1 | mA |
| | | 5V | | — | 1.5 | 3 | mA |

## A.C. Characteristics

Ta=25°C

| Symbol | Parameter | Test Conditions | | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|---|---|
| | | $V_{DD}$ | Conditions | | | | |
| $f_{SYS1}$ | System Clock (Crystal) | — | 2.2V~5.5V | 400 | — | 4000 | kHz |
| | | — | 3.3V~5.5V | 400 | — | 8000 | |
| $f_{SYS2}$ | System Clock (32768Hz Crystal OSC) | — | 2.2V~5.5V | — | 32768 | — | Hz |
| $f_{TIMER}$ | Timer Input Frequency | — | 2.2V~5.5V | 0 | — | 4000 | kHz |
| | | — | 3.3V~5.5V | 0 | — | 8000 | |
| $t_{WDTOSC}$ | Watchdog Oscillator Period | 3V | — | 45 | 90 | 180 | μs |
| | | 5V | | 32 | 65 | 130 | |
| $t_{WDT}$ | Watchdog Time-out Period | — | Note: $t_{SYS}=4/f_{SYS}$ | | $65536 \times t_{SYS}$ or $65536 \times t_{WDTOSC}$ or $65536 \times t_{RTCOSC}$ | | |
| $t_{RES}$ | External Reset Low Pulse Width | — | — | 1 | — | — | μs |
| $t_{SST}$ | System Start-up Timer Period | — | Power-up or wake-up from HALT | — | 1024 | — | $t_{SYS}$ |
| $t_{INT}$ | Interrupt Pulse Width | — | — | 1 | — | — | μs |
| $t_{AD}$ | A/D Clock Period | — | — | 1 | — | — | μs |
| $t_{ADC}$ | A/D Conversion Time | — | — | 64 | — | — | $t_{AD}$ |
| $t_{ADCS}$ | A/D Sampling Time | — | — | — | 32 | — | $t_{AD}$ |
| $t_{COMP}$ | Response Time of Comparator | — | — | — | — | 3 | μs |

Note: $t_{SYS}=1/f_{SYS}$

## Functional Description

### Execution Flow

The system clock for the microcontroller is derived from an external RC or crystal oscillator. The system clock is internally divided into four non-overlapping clocks. One instruction cycle consists of 4 system clock cycles.

Instruction fetching and execution are pipelined in such a way that a fetch and decoding takes an instruction cycle while execution take the next instruction cycle. However, the pipelining scheme causes each instruction to effectively execute in a cycle. If an instruction changes the program counter, two cycles are required to complete the instruction.

### Program Counter – PC

The program counter controls the sequence in which the instructions stored in the program memory are executed and its contents specify full range of program memory. After accessing a program memory word to fetch an instruction code, the contents of the program counter are incremented by one. The program counter then points to the memory word containing the next instruction code.

When executing a jump instruction, conditional skip execution, loading PCL (program counter lower-order byte register), subroutine call, initial reset, interrupts or return from subroutine or interrupts, the program counter manipulates the program transfer by loading the address corresponding to each instruction.

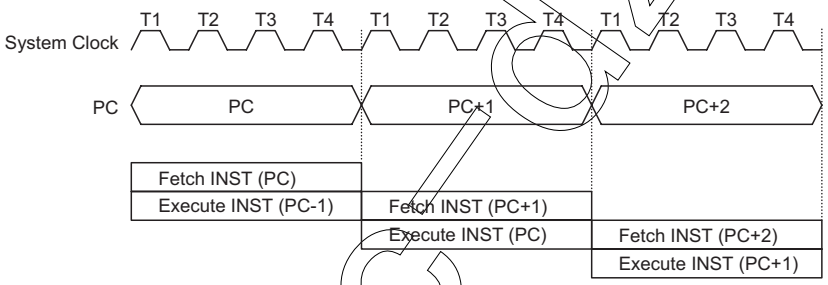
The conditional skip is activated by instructions. Once the condition is met, the next instruction, fetched during the current instruction execution, is discarded and a dummy cycle replaces it to get the proper instruction. Otherwise proceed with the next instruction.

The lower-order byte of the program counter (PCL) can be accessed by using software instructions. Moving data into the PCL performs a short jump. The destination will be within the current program ROM page.

Once the control transfer takes place, the execution suffers from having an additional dummy cycle.

### Program Memory – PROM

The program memory is used to store the program instructions which are to be executed. It also contains data, table, and interrupt entries, and is organized into




**Execution Flow**

| Mode | Program Counter | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | *11~*8 | *7 | *6 | *5 | *4 | *3 | *2 | *1 | *0 |
| Initial Reset | 0000 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| External Interrupt 0 | 0000 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| External Interrupt 1 | 0000 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| Timer/Event Counter Overflow | 0000 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| Time Base Time-out | 0000 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| A/D Interrupt | 0000 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| RTC Interrupt | 0000 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| Skip | PC+2 | | | | | | | | |
| Loading PCL | @11~@8 | @7 | @6 | @5 | @4 | @3 | @2 | @1 | @0 |
| Jump, Call Branch | #11~#8 | #7 | #6 | #5 | #4 | #3 | #2 | #1 | #0 |
| Return (RET, RETI) | S11~S8 | S7 | S6 | S5 | S4 | S3 | S2 | S1 | S0 |

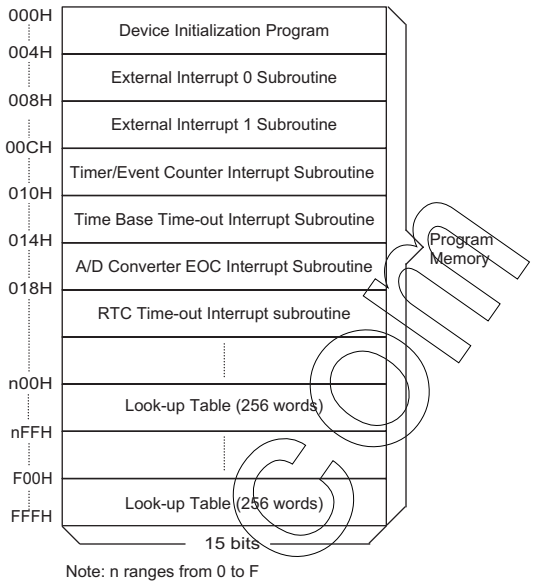
**Program Counter**

Note: *11~*0: Program counter bits
#11~#0: Instruction code bits
S11~S0: Stack register bits
@7~@0: PCL bits

4096×15 bits, addressed by the program counter and table pointer.

Certain locations in the program memory are reserved for special usage:

- Location 000H
  This area is reserved for program initialization. After chip reset, the program always begins execution at location 000H.
- Location 004H
  This area is reserved for the external interrupt 0 service program. If the $\overline{INT0}$ input pin is activated, the interrupt is enabled and the stack is not full, the program begins execution at this location.
- Location 008H
  This area is reserved for the external interrupt 1 service program. If the $\overline{INT1}$ input pin is activated, the interrupt is enabled and the stack is not full, the program begins execution at this location.
- Location 00CH
  This area is reserved for the timer/event counter interrupt service program. If a timer interrupt results from a timer/event counter overflow, and the interrupt is enabled and the stack is not full, the program begins execution at location 00CH.
- Location 010H
  This area is reserved for the time base interrupt service program. If the a time base time-out occurs, the interrupt is enabled and the stack is not full, the program begins execution at this location.
- Location 014H
  This area is reserved for the A/D converter interrupt service program. If the interrupt is activated (when the A/D conversion is completed), the interrupt is enabled and the stack is not full, the program begins execution at this location.
- Location 018H
  This area is reserved for the RTC interrupt service program. When the RTC time-out occurs, the interrupt is enabled and the stack is not full, the program begins execution at this location.
- Table location
  Any location in the program memory can be used as look-up tables. The instructions "TABRDC [m]" (the current page, 1 page=256 words) and "TABRDL [m]" (the last page) transfer the contents of the lower-order byte to the specified data memory, and the higher-order byte to lower portion of TBLH(08H) and the remaining bits (1 bits) of TBLH are read as "0". The table pointer (TBLP) is read/write register (07H), which indicates the table location. Before accessing the table, the location has to be placed in TBLP. The TBLH is read only and cannot be restored. If the main routine and the ISR(interrupt service routine) both employ the table read instruction, the contents of TBLH in the main routine are likely to be changed by the table read instruction used in the ISR. Errors are thus brought about. Given this, using the table read instruction in the main routine and the ISR simultaneously should be avoided. However, if the table read instruction has to be applied in both main routine and the ISR, the interrupt is supposed to be disabled prior to the table read instruction. It will not be enabled until the TBLH in the main routine has been backup. All table related instructions require two cycles to complete the operation. These areas may function as normal program memory depending upon the requirements.




**Program Memory**

**Stack Register – STACK**

This is a special part of memory, which is used to save the contents of the program counter only. The stack is organized into 8 levels and is neither part of the data not programmable space, and is not accessible. The activated level is indexed by the stack pointer and is not ac-

| Instruction | Table Location | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | *11 | *10 | *9 | *8 | *7 | *6 | *5 | *4 | *3 | *2 | *1 | *0 |
| TABRDC [m] | P11 | P10 | P9 | P8 | @7 | @6 | @5 | @4 | @3 | @2 | @1 | @0 |
| TABRDL [m] | 1 | 1 | 1 | 1 | @7 | @6 | @5 | @4 | @3 | @2 | @1 | @0 |

**Table Location**

Note: *11~*0: Table location bits
P11~P8: Current program counter bits
@7~@0: Table pointer bits

cessible. At a subroutine call or interrupt acknowledgment, the contents of the program counter are pushed onto the stack. At the end of a subroutine or an interrupt routine, signaled by a return instruction (RET or RETI), the program counter is restored to its previous value from the stack. After a chip reset, the stack pointer will point to the top of the stack.

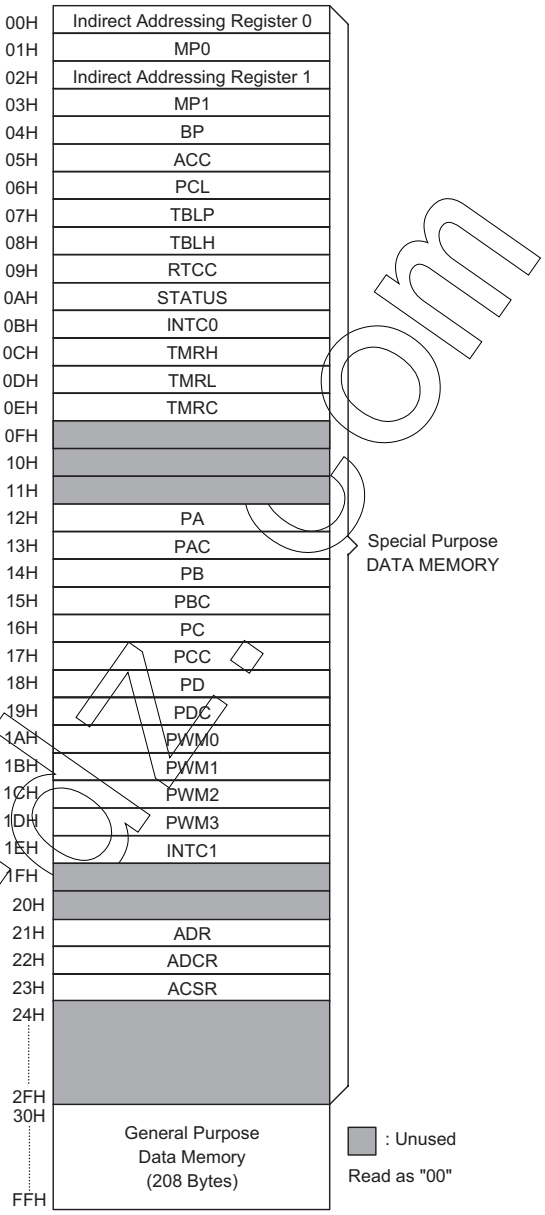
If the stack is full and a non-masked interrupt takes place, the interrupt request flag will be recorded but the acknowledgment will be inhibited. When the stack pointer is decreased (by RET or RETI), the interrupt will be serviced. This feature prevents stack overflow allowing the programmer to use the structure more easily. In similar case, if the stack is full and a "call" is subsequently executed, stack overflow occurs and the first entry will be lost (only the most recent 8 return addresses are stored).

**Data Memory – RAM**

The data memory is designed with 239×8 bits. The data memory is divided into two functional groups: special function registers and general purpose data memory (208×8). Most are read/write, but some are read only.

The special function registers include the indirect addressing register 0 and 1 (R0;00H, R1;02H), memory pointer 0 and 1 (MP0;01H, MP1;03H), bank pointer (BP:04H), accumulator (ACC;05H), program counter lower-order byte register (PCL;06H), table pointer (TBLP;07H), table higher-order byte register (TBLH;08H), real time clock control register (RTCC;09H), status register (STATUS;0AH), interrupt control register (INTC0;0BH), timer higher-order byte register (TMRH;0CH), timer lower-order byte register (TMRL;0DH), timer control register (TMRC;0EH), I/O port data registers (PA;12H, PB;14H, PC;16H, PD;18H), I/O port control registers (PAC;13H, PBC;15H, PCC;17H, PDC;19H), PWM0 (1AH), PWM1 (1BH), PWM2 (1CH), PWM3 (1DH), INTC1 (1EH), the A/D result register (ADR;21H), the A/D control register (ADCR;22H) and the A/D clock setting register (ACSR;23H). The remaining space before the 60H is reserved for future expansion and reading these locations will return the result "00H". The general-purpose data memory, addressed from 30H to FFH, is used for data and control information under instruction commands.

All of the data memory areas can handle arithmetic, logic, increment, decrement and rotate operations directly. Except for some dedicated bits, each bit in the data memory can be set and cleared by "SET [m].i" and "CLR [m].i", respectively. They are also indirectly accessible through memory pointers (MP0 and MP1).




**RAM Mapping**

**Indirect Addressing Register**

Location 00H (02H) is indirect addressing registers that are not physically implemented. Any read/write operation of [00H] ([02H]) will access data memory pointed to by MP0 (MP1). Reading location 00H (02H) itself indirectly will return the result "00H". Writing indirectly results in no operation.

The memory pointers are 8-bit registers. Only the MP1/R1 can be used to access the LCD RAM (BP=1).

| Labels | Bits | Function |
|---|---|---|
| C | 0 | C is set if an operation results in a carry during an addition operation or if a borrow does not take place during a subtraction operation; otherwise C is cleared. C is also affected by a rotate through carry instruction. |
| AC | 1 | AC is set if an operation results in a carry out of the low nibbles in addition or no borrow from the high nibble into the low nibble in subtraction; otherwise AC is cleared. |
| Z | 2 | Z is set if the result of an arithmetic or logical operation is zero; otherwise Z is cleared. |
| OV | 3 | OV is set if the operation results in a carry into the highest-order bit but not a carry out of the highest-order bit, or vice versa; otherwise OV is cleared. |
| PDF | 4 | PDF is cleared by a system power-up or executing the ″CLR WDT″ instruction. PDF is set by executing the ″HALT″ instruction. |
| TO | 5 | TO is cleared by system power-up or executing the ″CLR WDT″ or ″HALT″ instruction. TO is set by a WDT time-out. |
| — | 6, 7 | Unused bit, read as ″0″ |

**Status Register**

### Bank Pointer

The bank pointer is used to assign the accessed RAM bank. When the users want to access the RAM bank ″0″ a 0 should be loaded onto BP. When the BP is equal to ″1″, the LCD RAM will be accessed (use MP1/R1 indirect addressing only). RAM locations before 40H in any bank are overlapped.

### Accumulator

The accumulator is closely related to ALU operations. It is also mapped to location 05H of the data memory and can carry out immediate data operations. The data movement between two data memory locations must pass through the accumulator.

### Arithmetic and Logic Unit – ALU

This circuit performs 8-bit arithmetic and logic operations. The ALU provides the following functions:

- Arithmetic operations (ADD, ADC, SUB, SBC, DAA)
- Logic operations (AND, OR, XOR, CPL)
- Rotation (RL, RR, RLC, RRC)
- Increment and Decrement (INC, DEC)
- Branch decision (SZ, SNZ, SIZ, SDZ ....)

The ALU not only saves the results of a data operation but also changes the status register.

### Status Register – STATUS

This 8-bit register (0AH) contains the zero flag (Z), carry flag (C), auxiliary carry flag (AC), overflow flag (OV), power down flag (PDF), and watchdog time-out flag (TO). It also records the status information and controls the operation sequence.

With the exception of the TO and PDF flags, bits in the status register can be altered by instructions like most other registers. Any data written into the status register will not change the TO or PDF flag. In addition operations related to the status register may give different results from those intended. The TO flag can be affected only by system power-up, a WDT time-out or executing the ″CLR WDT″ or ″HALT″ instruction. The PDF flag can be affected only by executing the ″HALT″ or ″CLR WDT″ instruction or a system power-up.

The Z, OV, AC and C flags generally reflect the status of the latest operations.

In addition, on entering the interrupt sequence or executing the subroutine call, the status register will not be pushed onto the stack automatically. If the contents of the status are important and if the subroutine can corrupt the status register, precautions must be taken to save it properly.

### Interrupt

The microcontroller provides two external interrupts, an internal timer/event counter overflow interrupt, a time base time-out interrupt, an A/D converter end-of-conversion interrupt and a real time clock time-out interrupt. The interrupt control registers (INTC0: 0BH and INTC1: 1EH) contains the interrupt control bits to set the enable or disable and the interrupt request flags.

Once an interrupt subroutine is serviced, all the other interrupts will be blocked (by clearing EMI bit). This scheme may prevent any further interrupt nesting. Other interrupt requests may happen during this interval but only the interrupt request flags are recorded. If a certain interrupt requires servicing within the service routine, the programmer may set the EMI and the corresponding bit of INTC0/INTC1 to allow interrupt nesting. If the stack is full, the interrupt request will not be acknowledged, even if the related interrupt is enabled, until the SP is decreased. If immediate service is desired, the stack has to be prevented from becoming full.

All these kinds of interrupts have the wake-up capability. As an interrupt is serviced, a control transfer occurs by pushing the program counter onto the stack and then branching to subroutines at specified location(s) in the program memory. Only the program counter is pushed onto the stack. If the contents of the register or status register are altered by the interrupt service program, which corrupts the desired control sequence, the programmer should save these contents first.

External interrupts are triggered by a high to low and/or low to high transition of $\overline{INT0}$/$\overline{INT1}$ and the related interrupt request flag (bit 4/5 of INTC0 ) will be set. When the interrupt is enabled, the stack is not full and the external interrupt is active, a subroutine call to location 004H/008H will occur. The external interrupt request flag and EMI bits will cleared to disable other interrupts.

The internal timer/event counter interrupt is initialized by setting the timer/event counter interrupt request flag (bit 6 of INTC0), caused by a timer overflow. When the interrupt is enabled, the stack is not full and the timer/event counter interrupt request flag is set, a subroutine call to location 00CH will occur. The related interrupt request flag will be reset and the EMI bit cleared to disable further interrupts.

The time base time-out interrupt is initialized by setting the time base time-out interrupt request flag (bit 4 of INTC1), caused by a time base time-out. When the interrupt is enabled, the stack is not full and the time base time-out interrupt request flag is set, a subroutine call to location 010H will occur. The related interrupt request flag will be reset and the EMI bit cleared to disable further interrupts.

The A/D converter end-of-conversion interrupt is initialized by setting the A/D end-of-conversion interrupt request flag (bit 5 of INTC1), caused by an end of A/D conversion. When the interrupt is enabled, the stack is not full and the end of A/D conversion interrupt request flag is set, a subroutine call to location 014H will occur. The related interrupt request flag will be reset and the EMI bit cleared to disable further interrupts.

The real time clock time-out interrupt is initialized by setting the real time clock interrupt request flag (bit 6 of INTC1), caused by a RTC time-out. When the interrupt is enabled, the stack is not full and the RTC time-out interrupt request flag is set, a subroutine call to location 018H will occur. The related interrupt request flag will be reset and the EMI bit cleared to disable further interrupts.

During the execution of an interrupt subroutine, other interrupt acknowledgments are held until the RETI instruction is executed or the EMI bit and the related interrupt control bit are set to ″1″ (of course, if the stack is not full). To return from the interrupt subroutine, ″RET″ or ″RETI″ may be invoked. RETI will set the EMI bit to enable an interrupt service, but RET will not.

Interrupts, occurring in the interval between rising edge of two consecutive T2 pulses, will be serviced on the later of the two T2 pulses, if the corresponding interrupts are enabled. In the case of simultaneous requests the priorities in the follow table apply. These can be masked by clearing the EMI bit.

| Interrupt Source | Priority | Vector |
|---|---|---|
| External Interrupt 0 | 1 | 004H |
| External Interrupt 1 | 2 | 008H |
| Timer/Event Counter Overflow Interrupt | 3 | 00CH |
| Time Base Time-out Interrupt | 4 | 010H |
| End of A/D Conversion Interrupt | 5 | 014H |
| RTC Time-out Interrupt | 6 | 018H |

The external interrupt 0/1 request flags (EI0F/EI1F), timer/event counter interrupt request flag (TF), time base interrupt request flag (TBF), A/D converter interrupt request flag (ADF), RTC interrupt request flag (RTF), enable external interrupt 0/1 (EE0I/EE1I), enable timer/event counter interrupt bit (ETI), enable time base interrupt (ETBI), enable A/D converter interrupt (EADI), enable RTC interrupt (ERTI) and enable master interrupt bit(EMI) constitute interrupt control registers (INTC0/INTC1) which is located at 0BH/1EH in the data memory. EMI, EE0I, EE1I, ETI, EADI and ERTI are used to control the enabling/disabling of interrupts. These bits prevent the requested interrupts from being serviced. Once the interrupt request flags (EI0F, EI1F, TF, TBF, ADF, RTF) are set, they will remain in the INTC0/INTC1 until the interrupts are serviced or cleared by software instructions.

It is suggested that a program does not use the ″call″ within a interrupt subroutine. It because interrupts often occur in an unpredictable manner or need to be serviced immediately in some applications. If only one stack is left and enabling the interrupt is not well controlled, the original control sequence will be damaged once the ″CALL″ operates in the interrupt subroutine. The definitions of INTC0 and INTC1 registers are as shown.

| Bit No. | Label | Function |
|---|---|---|
| **INTC0 Register** | | |
| 0 | EMI | Controls the master (global) interrupt (1= enabled; 0= disabled) |
| 1 | EEI0 | Controls the external interrupt 0 (1= enabled; 0= disabled) |
| 2 | EEI1 | Controls the external interrupt 1 (1= enabled; 0= disabled) |
| 3 | ETI | Controls the timer/event counter overflow interrupt (1= enabled; 0= disabled) |
| 4 | EIF0 | External interrupt 0 request flag (1= active; 0= inactive) |

| Bit No. | Label | Function |
|---|---|---|
| 5 | EIF1 | External interrupt 1 request flag (1= active; 0= inactive) |
| 6 | TF | Timer/Event Counter overflow request flag (1= active; 0= inactive) |
| 7 | — | Unused bit, read as "0" |
| **INTC1 Register** | | |
| 0 | ETBI | Controls the time base interrupt (1= enabled; 0= disabled) |
| 1 | EADI | Controls the A/D converter interrupt (1= enabled; 0= disabled) |
| 2 | ERTI | Controls the real time clock interrupt (1= enabled; 0= disabled) |
| 3 | — | Unused bit, read as "0" |
| 4 | TBF | Time base time-out interrupt 0 request flag (1= active; 0= inactive) |
| 5 | ADF | End of A/D conversion interrupt request flag (1= active; 0= inactive) |
| 6 | RTF | RTC time-out interrupt request flag (1= active; 0= inactive) |
| 7 | — | Unused bit, read as "0" |

**Oscillator Configuration**

There are four oscillator circuits implemented in the micro-controller.

Two of them are designed for system clocks, namely the external RC oscillator and the crystal oscillator, which are determined by options. The HALT mode stops the system oscillator and resists the external signal to conserve power. Another one is a 32768Hz crystal oscillator, which only provides use for real time clock. The other one is a built-in 12KHz RC oscillator, which is used for WDTOSC.

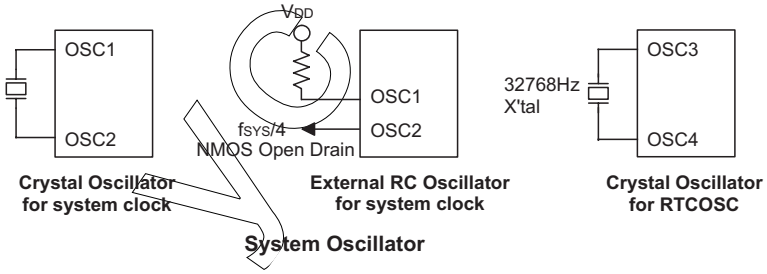
If the system clock uses the external RC oscillator, an external resistor between OSC1 and VDD is required and the resistance should range from 24kΩ to 1MΩ. The system clock, divided by 4, is available on OSC2, which can be used to synchronize external logic.

If the system clock uses the crystal oscillator, a crystal across OSC1 and OSC2 is needed to provide the feedback and phase shift required for the oscillator, and no other external components are demanded. Instead of a crystal, the resonator can also be connected between OSC1 and OSC2 to get a frequency reference, but two external capacitors in OSC1 and OSC2 are required.

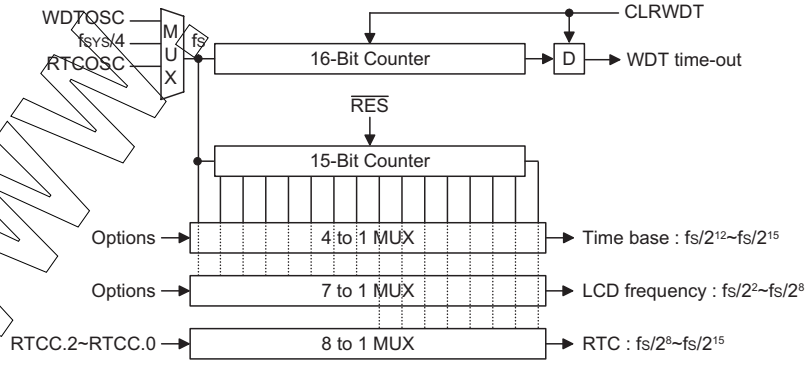
If the RTCOSC is used, a crystal across OSC3 and OSC4 is needed to provide the feedback and phase shift required for the oscillator, and no other external components are demanded.

**Watchdog Timer – WDT**

The clock source of WDT (and LCD, RTC, Time Base ) is implemented by a dedicated crystal oscillator (32.768kHz: RTCOSC) or instruction clock (system frequency divided by 4: $f_{SYS}/4$) or a dedicated RC oscillator (12KHz:WDTOSC) decided by options. This timer is designed to prevent a software malfunction or sequence from jumping to an unknown location with unpredictable




**System Oscillator**




**Watchdog Timer**

results. The watchdog timer can be disabled by options. If the watchdog timer is disabled, all the executions related to the WDT result in no operation. The WDT time-out period is fixed as $2^{16}/f_S$. The $f_S$ means the clock frequency of WDT, time base, RTC and LCD. If WDTOSC is selected as the WDT clock, the time-out period may vary with temperatures, VDD and process variations. The WDTOSC and RTCOSC can be still running (decided by option) at the halt mode if they are selected as the WDT clock source. Once the 32.768kHz oscillator (with a period of 31.25μs normally) is selected to be the clock source of WDT (and LCD, RTC, Time Base), it is directly divided by $2^{16}$ to get the nominal time-out period of 2 seconds. If the WDT clock comes from the instruction clock, the WDT will stop counting and lose its protecting purpose in halt mode. In this situation the logic can only be restarted by external logic. If the device operates in a noisy environment, using the RTCOSC or WDTOSC is strongly recommended, since the HALT will stop the system clock.

The overflow of WDT under normal operation will initialize ″chip reset″ and set the status bit ″TO″. But in the HALT mode, the overflow will initialize a ″warm reset″, and only the PC and SP are reset to zero. To clear the contents of WDT , 3 methods are adopted; external reset (a low level to $\overline{RES}$), software instruction(s) and a HALT instruction. The software instruction(s) include ″CLR WDT″ and the other set – ″CLR WDT1″ and ″CLR WDT2″ Of these two types of instruction, only one can be active depending on the options – ″CLR WDT times selection option″. If the ″CLR WDT″ is selected (i.e. CLRWDT times equal one), any execution of the ″CLR WDT″ instruction will clear the WDT. In the case that ″CLR WDT1″ and ″CLR WDT2″ are chosen (i.e. CLR WDT times equal two), these two instructions must be executed to clear the WDT; otherwise, the WDT may reset the chip as a result of time-out. The RTC oscillator should be designed as an auto-speed-up oscillator. After the RTC oscillator is oscillating, the auto-speed-up should be turned off.

### Time Base Generator

There is a time base generator implemented in the micro-controller. The time base generator provides time-out periods selection whose range from $f_S/2^{12}$ to $f_S/2^{15}$. When the time base time-out occurs and the stack is not full and the time base interrupt is enabled, an interrupt subroutine call to ROM location 010H will activate.

### RTC Generator

There is an RTC generator implemented in the micro-controller. The RTC generator provides software configurable real time clock periods whose range from $f_S/2^8$ to $f_S/2^{15}$. When the RTC time-out occurs and the stack is not full and the RTC interrupt is enabled, an interrupt subroutine call to ROM location 018H will activate. The RTCC is the real time clock control register used to select the division ratio of RTC clock sources. RTCC.7~RTCC.3 cannot be used.

| RTCC.2 | RTCC.1 | RTCC.0 | RTC clock divided factor |
|---|---|---|---|
| 0 | 0 | 0 | $2^8$ |
| 0 | 0 | 1 | $2^9$ |
| 0 | 1 | 0 | $2^{10}$ |
| 0 | 1 | 1 | $2^{11}$ |
| 1 | 0 | 0 | $2^{12}$ |
| 1 | 0 | 1 | $2^{13}$ |
| 1 | 1 | 0 | $2^{14}$ |
| 1 | 1 | 1 | $2^{15}$ |

### Power Down Operation – HALT

The HALT mode is initialized by the ″HALT″ instruction and results in the following...

- The system oscillator will be turned off but the WDTOSC or RTCOSC will stop or keep running decided by option (If the WDTOSC or RTCOSC is selected)
- The contents of the on-chip RAM and registers remain unchanged.
- WDT will be cleared and recounted again (if the WDT clock is from the WDTOSC or RTCOSC).
- All of the I/O ports maintain their original status.
- The PDF flag is set and the TO flag is cleared.

The system can leave the HALT mode by means of an external reset, an interrupt, an external falling edge signal on port A or a WDT overflow. An external reset causes a device initialization and the WDT overflow performs a ″warm reset″. After the TO and PDF flags are examined, the reason for chip reset can be determined. The PDF flag is cleared by system power-up or executing the ″CLR WDT″ instruction and is set when executing the ″HALT″ instruction. The TO flag is set if the WDT time-out occurs, and causes a wake-up that only resets the PC and SP; the others keep their original status.

The port A wake-up and interrupt methods can be considered as a continuation of normal execution. Each bit in port A can be independently selected to wake up the device by the option. Awakening from an I/O port stimulus, the program will resume execution of the next instruction. If it is awakening from an interrupt, two sequences may happen. If the related interrupt is disabled or the interrupt is enabled but the stack is full, the program will resume execution at the next instruction. If the interrupt is enabled and the stack is not full, the regular interrupt response takes place. If an interrupt request flag is set to ″1″ before entering the HALT mode, the wake-up function of the related interrupt will be disabled.

Once a wake-up event occurs, it takes 1024 $t_{SYS}$ (system clock period) to resume normal operation. In other words, a dummy period will be inserted after wake-up. If the wake-up results from an interrupt acknowledgment, the actual interrupt subroutine execution will be delayed by one or more cycles. If the wake-up results in the next instruction execution, this will be executed immediately after the dummy period is finished.

To minimize power consumption, all the I/O pins should be carefully managed before entering the HALT status.

The 32.768kHz crystal oscillator still run or stop in the halt mode. (decided by option)

**Reset**

There are three ways in which a reset can occur:

- $\overline{RES}$ reset during normal operation
- $\overline{RES}$ reset during HALT
- WDT time-out reset during normal operation

The WDT time-out during HALT is different from other chip reset conditions, since it can perform a "warm reset" that resets only the PC and SP, leaving the other circuits in their original state. Some registers remain unchanged during other reset conditions. Most registers are reset to the "initial condition" when the reset conditions are met. By examining the PDF and TO flags, the program can distinguish between different "chip resets".

| TO | PDF | Reset Conditions |
|---|---|---|
| 0 | 0 | $\overline{RES}$ reset during power-up |
| u | u | $\overline{RES}$ reset during normal operation |
| 0 | 1 | $\overline{RES}$ wake-up HALT |
| 1 | u | WDT time-out during normal operation |
| 1 | 1 | WDT wake-up HALT |

Note: "u" means unchanged

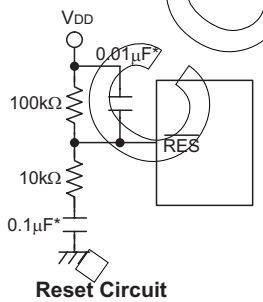
To guarantee that the system oscillator is started and stabilized, the SST (system start-up timer) provides an extra-delay to delay 1024 system clock pulses when system power-up or the system awakes from the HALT state.

When the system power-up occurs, the SST delay is added during the reset period. But when the reset comes from the RES pin, the SST delay is disabled. Any wake-up from HALT will enable the SST delay.

An extra option load time delay is added during system reset (power-up, WDT time-out at normal mode or $\overline{RES}$ reset).

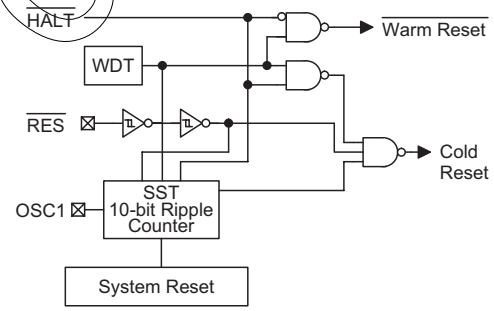
The chip reset statuses of the functional units are as shown.

| | |
|---|---|
| PC | 000H |
| Interrupt | Disable |
| WDT | Clear. After master reset, WDT begins counting |
| Timer/Event Counter | Off |
| Input/Output Ports | Input mode |
| SP | Points to the top of the stack |

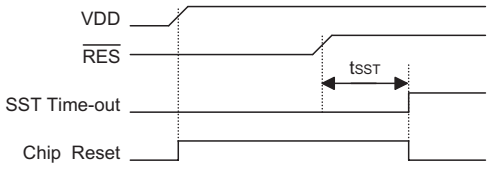



**Reset Circuit**

Note: "*" Make the length of the wiring, which is connected to the $\overline{RES}$ pin as short as possible, to avoid noise interference.




**Reset Configuration**




**Reset Timing Chart**

The registers states are summarized in the following table.

| Register | Reset (Power On) | WDT Time-out (Normal Operation) | $\overline{RES}$ Reset (Normal Operation) | $\overline{RES}$ Rese (HALT) | WDT Time-out (HALT)* |
|---|---|---|---|---|---|
| MP0 | xxxx xxxx | uuuu uuuu | uuuu uuuu | uuuu uuuu | uuuu uuuu |
| MP1 | xxxx xxxx | uuuu uuuu | uuuu uuuu | uuuu uuuu | uuuu uuuu |
| BP | 0000 0000 | 0000 0000 | 0000 0000 | 0000 0000 | uuuu uuuu |
| ACC | xxxx xxxx | uuuu uuuu | uuuu uuuu | uuuu uuuu | uuuu uuuu |
| PCH.PCL | 000H | 000H | 000H | 000H | 000H |
| TBLP | xxxx xxxx | uuuu uuuu | uuuu uuuu | uuuu uuuu | uuuu uuuu |
| TBLH | -xxx xxxx | -uuu uuuu | -uuu uuuu | -uuu uuuu | -uuu uuuu |
| RTCC | --xx x111 | --xx x111 | --xx x111 | --xx x111 | --uu uuuu |
| STATUS | --00 xxxx | --1u uuuu | --uu uuuu | --01 uuuu | --11 uuuu |
| INTC0 | -000 0000 | -000 0000 | -000 0000 | -000 0000 | -uuu uuuu |
| INTC1 | -000 -000 | -000 -000 | -000 -000 | -000 -000 | -uuu -uuu |
| TMRL | xxxx xxxx | xxxx xxxx | xxxx xxxx | xxxx xxxx | uuuu uuuu |
| TMRH | xxxx xxxx | xxxx xxxx | xxxx xxxx | xxxx xxxx | uuuu uuuu |
| TMRC | 00-0 1--- | 00-0 1--- | 00-0 1--- | 00-0 1--- | uu-u u--- |
| PA | 1111 1111 | 1111 1111 | 1111 1111 | 1111 1111 | uuuu uuuu |
| PAC | 1111 1111 | 1111 1111 | 1111 1111 | 1111 1111 | uuuu uuuu |
| PB | 1111 1111 | 1111 1111 | 1111 1111 | 1111 1111 | uuuu uuuu |
| PBC | 1111 1111 | 1111 1111 | 1111 1111 | 1111 1111 | uuuu uuuu |
| PC | 1111 1111 | 1111 1111 | 1111 1111 | 1111 1111 | uuuu uuuu |
| PCC | 1111 1111 | 1111 1111 | 1111 1111 | 1111 1111 | uuuu uuuu |
| PD | 1111 1111 | 1111 1111 | 1111 1111 | 1111 1111 | uuuu uuuu |
| PDC | 1111 1111 | 1111 1111 | 1111 1111 | 1111 1111 | uuuu uuuu |
| PWM0 | xxxx xxxx | xxxx xxxx | xxxx xxxx | xxxx xxxx | uuuu uuuu |
| PWM1 | xxxx xxxx | xxxx xxxx | xxxx xxxx | xxxx xxxx | uuuu uuuu |
| PWM2 | xxxx xxxx | xxxx xxxx | xxxx xxxx | xxxx xxxx | uuuu uuuu |
| PWM3 | xxxx xxxx | xxxx xxxx | xxxx xxxx | xxxx xxxx | uuuu uuuu |
| ADR | xxxx xxxx | xxxx xxxx | xxxx xxxx | xxxx xxxx | uuuu uuuu |
| ADCR | 0100 0000 | 0100 0000 | 0100 0000 | 0100 0000 | uuuu uuuu |
| ACSR | 0--- -100 | 0--- -100 | 0--- -100 | 0--- -100 | u--- -uuu |

Note: "*" stands for warm reset
"u" stands for unchanged
"x" stands for unknown

**Timer/Event Counter**

A timer/event counter is implemented in the device. The timer/event counter contains a 16-bit programmable count-up counter and the clock may come from an external source or the internal clock source.

The internal clock source is the system clock divided by 4: $f_{SYS}/4$. The external clock input allows the user to count external events, measure time intervals or pulse widths, or to generate an accurate time base.

There are 3 registers related to timer/event counter; TMRH(0CH), TMRL(0DH), TMRC(0EH). Writing TMRL only stores the data into a low byte buffer, and writing TMRH will put the written data and the low contents of low byte buffer to preload register (16 bits) simultaneously. The timer/event counter preload register is changed by writing TMRH operations and writing TMRL will keep the timer/event counter preload register unchanged.

Reading TMRH will also latch the TMRL into the low byte buffer to avoid the false timing problem. Reading TMRL returns the contents of the low byte buffer. In other words, the low byte of timer/event counter cannot be read directly. It has to read the TMRH first to make the low byte contents of timer/event counter latched into the buffer. The TMRC is the timer/event counter control register, which defines the operating mode, counting enable or disable and active edge.

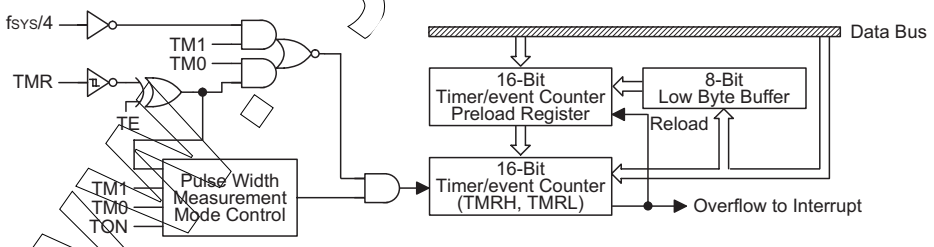
The TM0, TM1 bits define the operating mode. The event count mode is used to count external events, which means the clock source comes from an external (TMR) pin. The timer mode functions as a normal timer with the clock source coming from $f_{SYS}/4$. The pulse width measurement mode can be used to count the high or low level duration of the external signal (TMR). The counting is based on $f_{SYS}/4$.

In the event count or timer mode, once the timer/event counter starts counting, it will count from the current contents in the timer/event counter to FFFFH. Once overflow occurs, the counter is reloaded from the timer/event counter preload register and generates the corresponding interrupt request flag (TF; bit 6 of INTC0) at the same time.

In pulse width measurement mode with the TON and TE bits are equal to one, once the TMR has received a transition from low to high (or high to low if the TE bit is 0) it will start counting until the TMR returns to the original level and reset the TON. The measured result will remain in the timer/event counter even if the activated transition occurs again. In other words, only one cycle measurement can be done. Until setting the TON, the cycle measurement will function again as long as it receives further transition pulse. Note that, in this operating mode, the timer/event counter starts counting not according to the logic level but according to the transition edges. In the case of counter overflows, the counter is reloaded from the timer/event counter preload register and issues the interrupt request just like the other two modes.

To enable the counting operation, the timer ON bit (TON; bit 4 of TMRC) should be set to 1. In the pulse width measurement mode, the TON will be cleared automatically after the measurement cycle is complete. But in the other two modes the TON can only be reset by instructions. The overflow of the timer/event counter is one of the wake-up sources. No matter what the operation mode is, writing a 0 to ETI can disabled the corresponding interrupt service.

In the case of timer/event counter OFF condition, writing data to the timer/event counter preload register will also load the data to timer/event counter. But if the timer/event counter is turned on, data written to the timer/event counter will only be kept in the timer/event counter preload register. The timer/event counter will still operate until the overflow occurs (a timer/event counter reloading will occur at the same time).




**Timer/Event Counter**

When the timer/event counter (reading TMRH) is read, the clock will be blocked to avoid errors. As this may results in a counting error, this must be taken into consideration by the programmer.

| Label (TMRC) | Bits | Function |
|---|---|---|
| — | 0~2 | Unused bits, read as "0" |
| TE | 3 | To define the active edge of TMR pin input signal<br>(0=active on low to high;<br>1=active on high to low) |
| TON | 4 | To enable or disable timer counting<br>(0=disabled; 1=enabled) |
| — | 5 | Unused bit, read as "0" |
| TM0<br>TM1 | 6<br>7 | To define the operating mode<br>01=Event count mode (external clock)<br>10=Timer mode (internal clock)<br>11=Pulse width measurement mode<br>00=Unused |

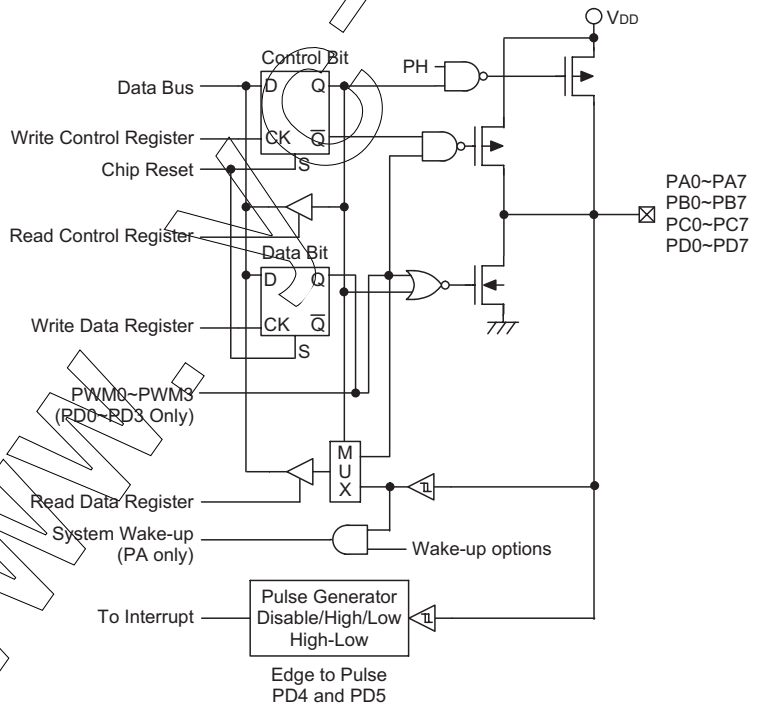
**TMRC Register**

**Input/Output Ports**

There are 32 bi-directional input/output lines in the micro-controller, labeled from PA to PD, which are mapped to the data memory of [12H], [14H], [16H] and [18H], respectively. All of these I/O ports can be used as input and output operations. For input operation, these ports are non-latching, that is, the inputs must be ready at the T2 rising edge of instruction "MOV A,[m]" (m=12H, 14H, 16H or 18H). For output operation, all the data is latched and remains unchanged until the output latch is rewritten.

Each I/O line has its own control register (PAC, PBC, PCC, PDC) to control the input/output configuration. With this control register, CMOS output or schmitt trigger input with or without (depends on options) pull-high resistor structures can be reconfigured dynamically (i.e., on-the fly) under software control. To function as an input, the corresponding latch of the control register has to be set as "1". The pull-high resistor (if the pull-high resistor is enabled) will be exhibited automatically. The input sources are also dependent on the control register. If the control register bit is "1", the input will read the pad state ("mov" and read-modify-write instructions). If the control register bit is "0", the contents of the latches will move to internal data bus ("mov" and read-modify-write instructions). The input paths (pad state or latches) of read-modify-write instructions are dependent on the control register bits. For output function, CMOS is the only configuration. These control registers are mapped to locations 13H, 15H, 17H and 19H.

After a chip reset, these input/output lines stay at a high level (pull-high options) or floating state (non-pull-high options). Each bit of these input/output latches can be set or cleared by "SET [m].i" (m=12H, 14H, 16H or 18H) instructions. Some instructions first input data and then follow the output operations. For example, "SET [m].i" "CLR [m].i", "CPLA [m]" read the entire port states into the CPU, execute the defined operations (bit-operation),




**Input/Output Ports**

and then write the results back to the latches or the accumulator.

Each line of port A has the capability of waking-up the device. The pull-high resistor of each I/O line is decided by options.

**Comparator**

There is a comparator implemented in this microcontroller. This comparator can be enabled/disabled by options. Its inputs are CMPP(+) and CMPN(-) and outputs are CMPO and CHGO. When the CMPN input level is less than the level of CMPP, the CMPO output is $V_{DD}$. When the CMPN input level is higher than the level of CMPP, the CMPO output is $V_{SS}$.

The CHGO signal is combined with CMPO and 32768Hz carrier if 32768Hz RTC oscillator is applied.

This comparator also can be disabled by options. When the system enters halt mode, the comparator is disabled to reduce power consumption. Once the comparator is disabled, the CHGO and CMPO will stay at VSS level.

**LCD Display Memory**

The micro-controller provides an area of embedded data memory for LCD driver. This area is located from 40H to 53H of he RAM Bank 1. Bank pointer (BP; located at 04H of the RAM) is the switch between the general purpose RAM and the LCD display memory. When the BP is set to ″1″, any data written into 40H~53H (indirect accessing by using the MP1and R1) will effect the LCD display. When the BP is cleared to ″0″, any data written into 40H to 53H will access the general purpose data memory. The LCD display memory can be read and written to only by indirect addressing mode using MP1. When data is written into the display data area it is automatically read by the LCD driver which then generates the corresponding LCD driving signals. To turn the display on or off, an ″1″ or a ″0″ is written to the corresponding bit of the display memory, respectively. The figure illustrates the mapping between the display memory and LCD pattern for the micro-controller.

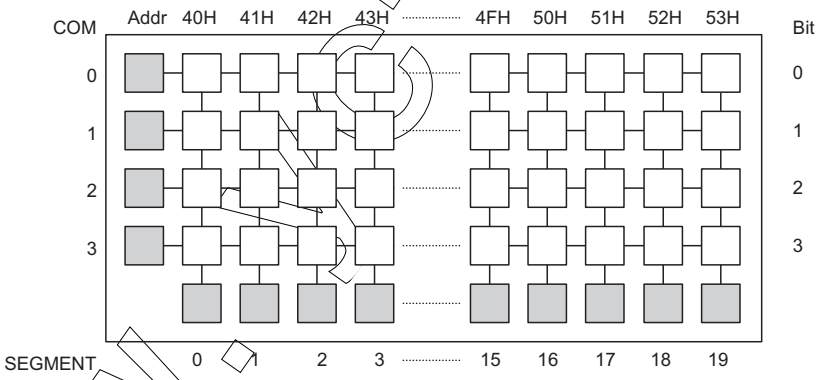
**LCD Driver Output and Bias Circuit**

The output number of the micro-controller LCD driver can be 20×3 or 19×4 by options (ie., 1/3 duty or 1/4 duty). The bias type of LCD driver is ″R″ type, no external capacitor is required. The LCD can be optioned as ″LCD on at HALT″ or ″LCD off at HALT″ which are dependent on options.

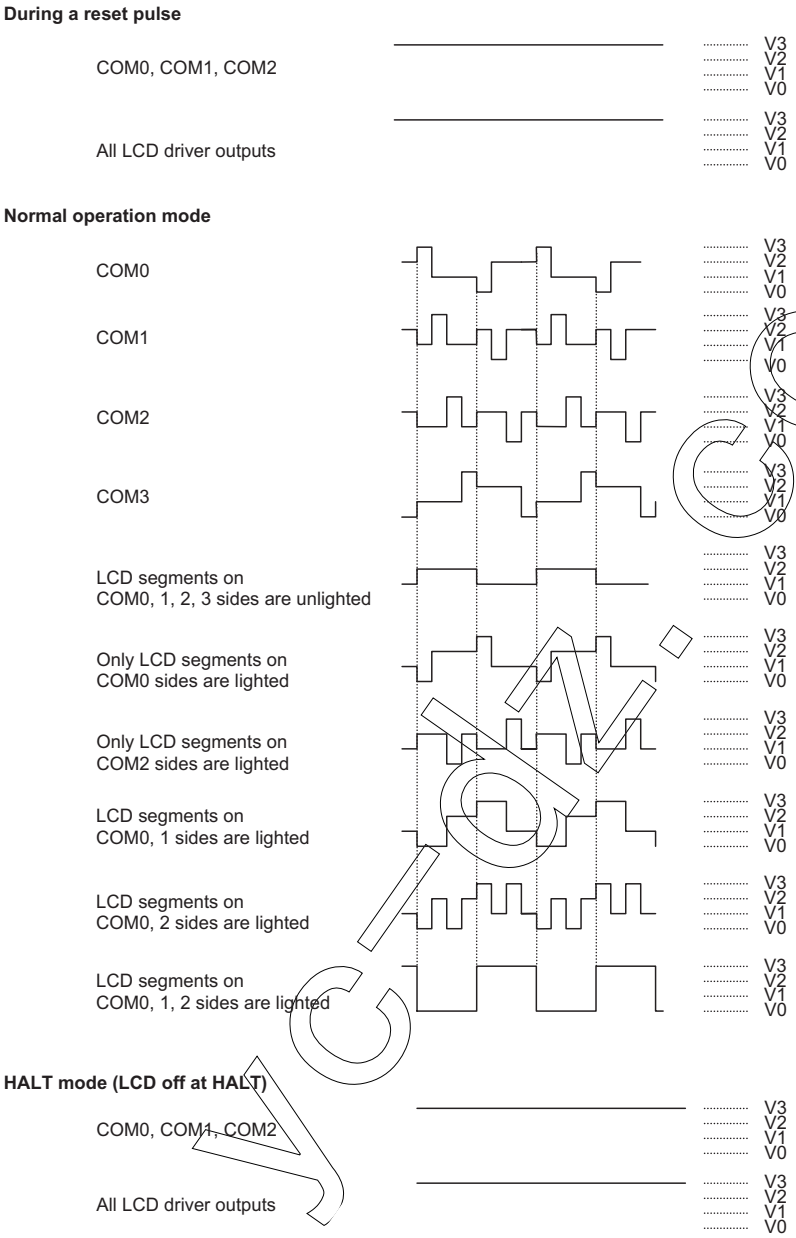
The SEG7~SEG18 also can be optioned as logical outputs. Each group of SEG7~SEG10, SEG11~SEG14 and SEG15~SEG18 can be optioned individually. Once an LCD segment is optioned as a logical output, the contents of bit 0 of the related segment address in LCD RAM will appear on the segment.

| Memory | Segment Output |
|---|---|
| Bit 0=0 | VSS |
| Bit 0=1 | VDD |

**Logical Output Function**

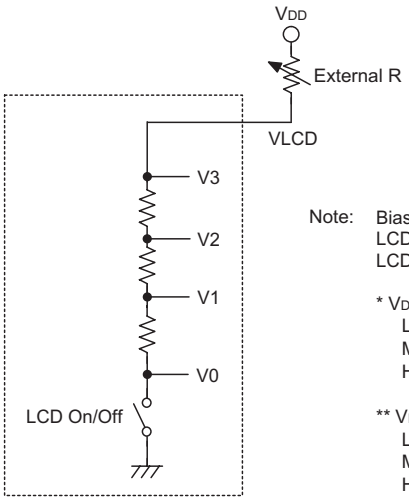



**Display Memory**




**LCD Driver Outputs (1/4 Duty, 1/3 Bias)**

Note: If LCD is turned on at HALT mode, the LCD outputs are dependent on LCD display memory.
If LCD is turned off at HALT mode, the power will be V3=V2=V1=V0=VDD




Note: Bias current (low, middle or high) is selectable by ROM code option.
LCD Off: V0=V1=V2=V3=$V_{LCD}$ (=$V_{DD}$, if connect $V_{LCD}$ to $V_{DD}$ with external resistor)
LCD driver and bias On: V0=$V_{SS}$,V1=$V_{LCD}$/3,V2=$V_{LCD}$*2/3,V3=$V_{LCD}$

* $V_{DD}$=5V, Bias for $V_{LCD}$=3V
Low: 8μA, external R=240kΩ
Middle: 16μA, external R=120kΩ
High: 48μA, external R=40kΩ

** $V_{DD}$=3V, Bias for $V_{LCD}$=3V
Low: 8μA, external R=0kΩ
Middle: 16μA, external R=0kΩ
High: 48μA, external R=0kΩ

**LCD Bias Block Diagram and Application Circuit**

### A/D Converter

The 8 channels and 8-bit resolution (7-bit accuracy) A/D converter are implemented in this microcontroller. The reference voltage is AVDD. The AVDD pin must be connected to VDD externally. Conversion accuracy may therefore be degraded by voltage drops and noise in the event of heavily loaded or badly coupled power supply lines. The A/D converter contains 3 special registers which are; ADR (21H), ADCR (22H) and ACSR (23H). The ADR is A/D result register. After the A/D conversion is completed, the ADR should be read to get the conversion result data. The ADCR is an A/D converter control register, which defines the A/D channel number, analog channel select, start A/D conversion control bit and the end of A/D conversion flag. If the users want to start an A/D conversion, after select the converted analog channel, and then give START bit a positive pulse (0→1→0). At the end of A/D conversion, the EOCB bit is cleared and an A/D converter interrupt occurs(if the A/D converter interrupt is enabled). The ACSR is an A/D clock setting register, which is used to select the A/D clock source.

The A/D converter control register is used to control the A/D converter. The bit2~bit0 of the ADCR are used to select an analog input channel. There are a total of 8 channels to select. The bit5~bit3 of the ADCR are used to set PB configurations. PB can be an analog input or as digital I/O line decided by these 3 bits. Once a PB line is selected as an analog input, the I/O functions and pull-high resistor of this I/O line are disabled. The EOCB bit (bit 6 of the ADCR) is end of A/D conversion flag. Check this bit to know when A/D conversion is completed. The START bit of the ADCR is used to begin the conversion of A/D converter. Give START bit a falling edge that means the A/D conversion has started. The A/D converter remains in reset state while the START stays at ″1″. In order to ensure the A/D conversion is completed, the START should stay at ″0″ until the EOCB is cleared to ″0″ (end of A/D conversion).

The bit 7 of the ACSR is used for testing purpose only. It can not be used for the users. The bit1 and bit0 of the ACSR are used to select A/D clock sources.

When the A/D conversion is completed, the A/D interrupt request flag is set. The bit is set to ″1″ when the START bit is set to ″1″.

| Register | Bit7 | Bit6 | Bit5 | Bit4 | Bit3 | Bit2 | Bit1 | Bit0 |
|---|---|---|---|---|---|---|---|---|
| ADR | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |

| Label (ADCR) | Bits | Functions |
|---|---|---|
| ACS0<br>ACS1<br>ACS2 | 0<br>1<br>2 | ACS2, ACS1, ACS0: A/D channel selection<br>0,0,0: AN0<br>0,0,1: AN1<br>0,1,0: AN2<br>0,1,1: AN3<br>1,0,0: AN4<br>1,0,1: AN5<br>1,1,0: AN6<br>1,1,1: AN7 |
| PCR0<br>PCR1<br>PCR2 | 3<br>4<br>5 | PCR2, PCR1, PCR0: PB7~PB0 pad functions<br>0,0,0: PB7, PB6, PB5, PB4, PB3, PB2, PB1, PB0<br>0,0,1: PB7, PB6, PB5, PB4, PB3, PB2, PB1, AN0<br>0,1,0: PB7, PB6, PB5, PB4, PB3, PB2, AN1, AN0<br>0,1,1: PB7, PB6, PB5, PB4, PB3, AN2, AN1, AN0<br>1,0,0: PB7, PB6, PB5, PB4, AN3, AN2, AN1, AN0<br>1,0,1: PB7, PB6, PB5, AN4, AN3, AN2, AN1, AN0<br>1,1,0: PB7, PB6, AN5, AN4, AN3, AN2, AN1, AN0<br>1,1,1: AN7, AN6, AN5, AN4, AN3, AN2, AN1, AN0 |
| EOCB | 6 | End of A/D conversion flag<br>(0: end of A/D conversion) |
| START | 7 | A/D conversion sequence (START=010)<br>0: Initial value after chip RESET<br>0→1: Initial next A/D conversion.<br>1: reset A/D converter and set EOCB to ″1″<br>1→0: Starts the A/D conversion.<br>0: Normal state for A/D |

Note: It is recommended that START is ″0″ and PCR2~PCR0 is ″000″ before MCU entering HALT mode. HALT will not standby the A/D converter automatically.

**ACSR Register**

| Label (ACSR) | Bits | Functions |
|---|---|---|
| ADCS0<br>ADCS1 | 0<br>1 | ADCS1, ADCS0: Selects the A/D converter clock source<br>0,0: $f_{SYS}/2$<br>0,1: $f_{SYS}/8$<br>1,0: $f_{SYS}/32$<br>1,1: Cannot be used |
| CMPC | 2 | Comparator control (*)<br>0: Disable<br>1: Enable |
| — | 3~6 | Unused bit, read as ″0″ |
| TEST | 7 | For test mode used only<br>0: Normal mode<br>1: TEST only, cannot be used |

Note: ″*″ This bit is 0 during reset.

The following two programming examples illustrate how to setup and implement an A/D conversion. In the first example, the method of polling the EOCB bit in the ADCR register is used to detect when the conversion cycle is complete, whereas in the second example, the A/D interrupt is used to determine when the conversion is complete.

Example: using EOCB Polling Method to detect end of conversion

```
    clr INTC0.3                 ; disable A/D interrupt in interrupt control register
    mov a,00100000B
    mov ADCR,a                  ; setup ADCR register to configure Port PB0~PB3 as A/D inputs and select
                                ; AN0 to be connected to the A/D converter
    mov a,00000001B
    mov ACSR,a                  ; setup the ACSR register to select fSYS/8 as the A/D clock

Start_conversion:
    clr ADCR.7
    set ADCR.7                  ; reset A/D
    clr ADCR.7                  ; start A/D

Polling_EOC:
    sz ADCR.6                   ; poll the ADCR register EOCB bit to detect end of A/D conversion
    jmp polling_EOC             ; continue polling
    mov a,ADR                   ; read conversion result from the high byte ADR register
    mov adr_buffer,a            ; save result to user defined register
            :
            :
    jmp start_conversion        ; start next A/D conversion
```

Example: using Interrupt method to detect end of conversion

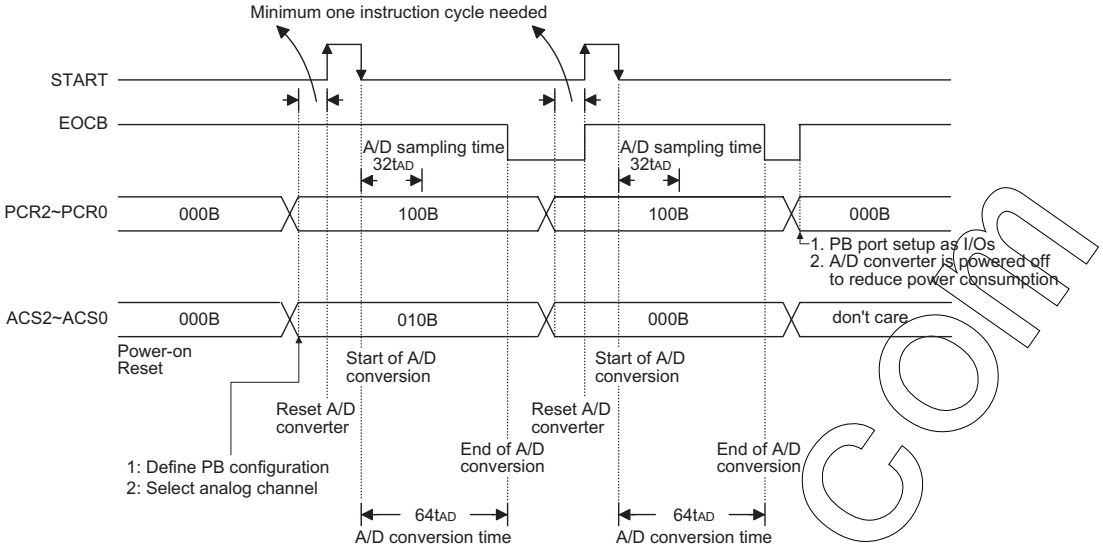
```
    set INTC0.0                 ; interrupt global enable
    set INTC0.3                 ; enable A/D interrupt in interrupt control register
    mov a,00100000B
    mov ADCR,a                  ; setup ADCR register to configure Port PB0~PB3 as A/D inputs and select
                                ; AN0 to be connected to the A/D converter
    mov a,00000001B
    mov ACSR,a                  ; setup the ACSR register to select fSYS/8 as the A/D clock

start_conversion:
    clr ADCR.7
    set ADCR.7                  ; reset A/D
    clr ADCR.7                  ; start A/D
            :
            :

; interrupt service routine
EOC_service routine:
    mov a_buffer,a              ; save ACC to user defined register
    mov a,ADR                   ; read conversion result from the high byte ADR register
    mov adr_buffer,a            ; save result to user defined register

    clr ADCR.7
    set ADCR.7                  ; reset A/D
    clr ADCR.7                  ; start A/D

    mov a,a_buffer              ; restore ACC from temporary storage
    reti
```



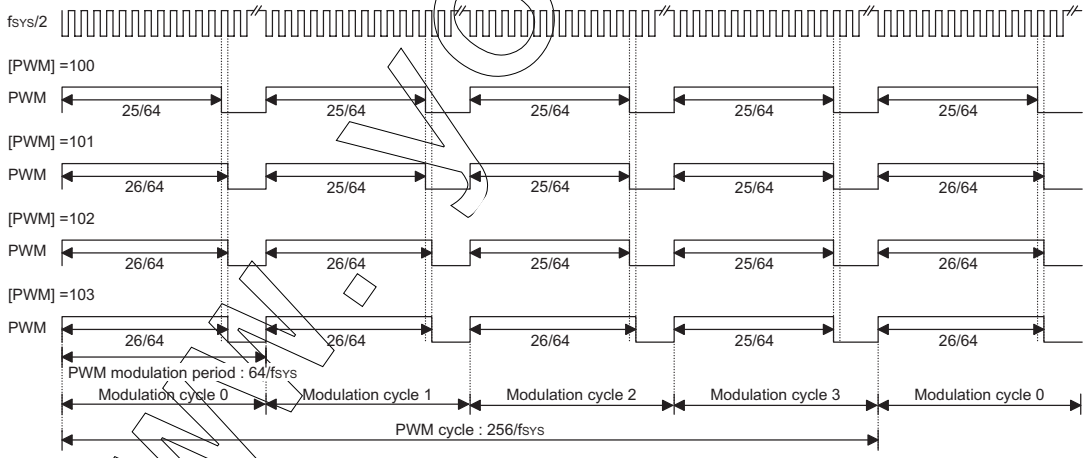

Note: A/D clock must be $f_{SYS}/2$, $f_{SYS}/8$ or $f_{SYS}/32$

**A/D Conversion Timing**

## PWM

The micro-controller provides 4 channels (6+2) bits PWM outputs shared with PD0~PD3. The PWM channels has their data register. The PWMs uses a PWM counter whose stages are 8 (stage 1~stage 8: $f_{SYS}/2^1$ ~ $f_{SYS}/2^8$). The frequency source of the PWM counter comes from $f_{SYS}$. The PWM register is an eight bits register. The waveforms of PWM outputs are as shown. Once the PDi (i=0~3) is selected as the PWMi output and the output function of PDi is enabled, writing "1" to PDi data register will enable the PWMi output function. Otherwise the PDi will stay at "0". The PWM modulation frequency, PWM cycle frequency and PWM cycle duty are summarized in the following table.

| PWMi Modulation Frequency | PWMi Cycle Frequency | PWMi Cycle Duty |
|---|---|---|
| $f_{SYS}/64$ | $f_{SYS}/256$ | [PWM]/256 |




**PWM Mode**

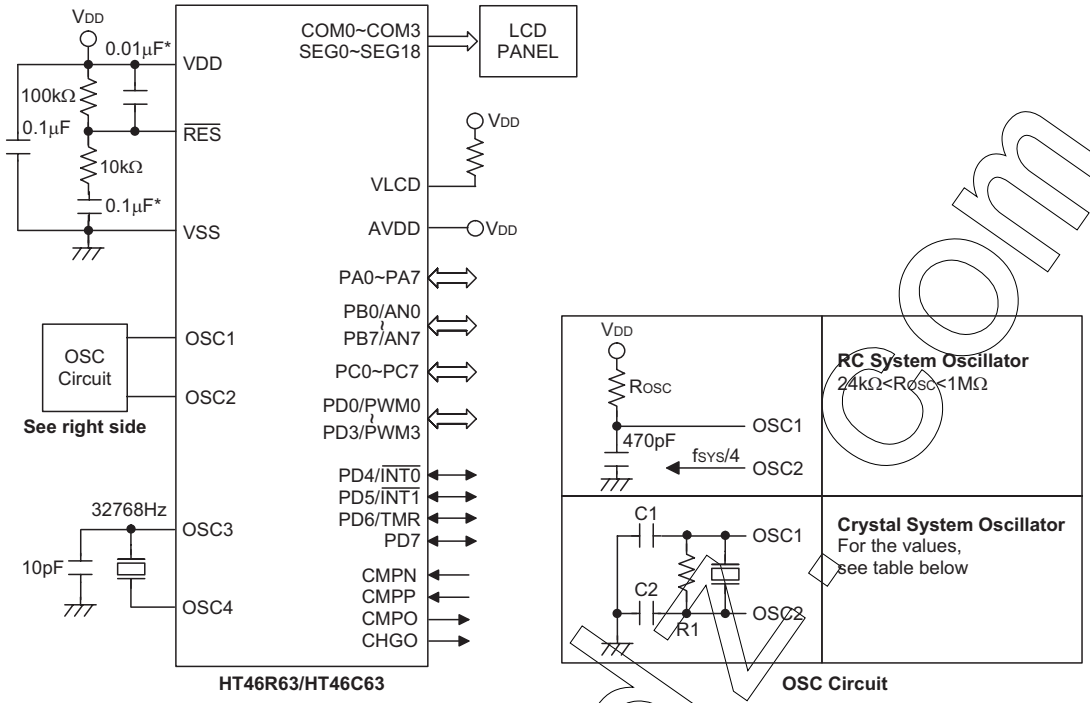
**Options**

The following table shows all kinds of options in the microcontroller. All of the options must be defined to ensure proper system function.

| No. | Options |
|---|---|
| 1 | PA wake-up enable or disable (1/0) options |
| 2 | WDT/LCD/RTC/Time Base Clock Source ($f_S$):<br>RTCOSC(32768Hz crystal), T1D or WDTOSC (*1) |
| 3 | CLR WDT instructions: 1/2 |
| 4 | WDT enable or disable |
| 5 | PA pull-high enable or disable (1 option : 4 bits (0~3/4~7)) |
| 6 | PB pull-high enable or disable (1 option : 4 bits (0~3/4~7)) |
| 7 | PC pull-high enable or disable (1 option : 4 bits (0~3/4~7)) |
| 8 | PD pull-high enable or disable (1 option : 4 bits (0~3/4~7)) |
| 9 | $\overline{INT0}$ or $\overline{INT1}$ trigger edge: disable; high to low; low to high; low to high or high to low. |
| 10 | COM3 or SEG19 (1/4 or 1/3 duty) |
| 11 | LCD on/off at halt mode |
| 12 | enable or disable Comparator |
| 13 | enable or disable PWMi function for PDi (bit optional) |
| 14 | $f_S/2^{12}$~$f_S/2^{15}$: Time base period |
| 15 | SEG7~SEG18 logical or LCD output<br>(1 option: 4 bits (SEG7~SEG10/SEG11~SEG14/SEG15~SEG18)) |
| 16 | System oscillators: external RC/ external crystal |
| 17 | enable or disable RTCOSC(32.768kHz crystal) or WDTOSC at HALT mode |
| 18 | LCD bias current: Low/Middle/High driving current |
| 19 | LCD driver clock selection.<br>There are seven types of frequency signals for the LCD driver circuits: $f_S/2^2$~$f_S/2^8$, "$f_S$" stands for the clock source selection by options. |

Note: "*1" T1D is stopped at HALT; RTCOSC(32.768kHz crystal) and WDT OSC are stopped or non-stopped at HALT decided by option(18).

## Application Circuits




The following table shows the C1, C2 and R1 value according different crystal values.

| Crystal or Resonator | C1, C2 | R1 |
|---|---|---|
| 4MHz Crystal | 0pF | 10kΩ |
| 4MHz Resonator (3 pin) | 0pF | 12kΩ |
| 4MHz Resonator (2 pin) | 10pF | 12kΩ |
| 3.58MHz Crystal | 0pF | 10kΩ |
| 3.58MHz Resonator (2 pin) | 25pF | 10kΩ |
| 2MHz Crystal & Resonator (2 pin) | 25pF | 10kΩ |
| 1MHz Crystal | 35pF | 27kΩ |
| 480kHz Resonator | 300pF | 9.1kΩ |
| 455kHz Resonator | 300pF | 10kΩ |
| 429kHz Resonator | 300pF | 10kΩ |

Note: The resistance and capacitance for reset circuit should be designed in such a way as to ensure that the VDD is stable and remains within a valid operating voltage range before bringing $\overline{RES}$ to high.

"*" Make the length of the wiring, which is connected to the $\overline{RES}$ pin as short as possible, to avoid noise interference.

## Instruction Set Summary

| Mnemonic | Description | Instruction Cycle | Flag Affected |
|---|---|---|---|
| **Arithmetic** | | | |
| ADD A,[m] | Add data memory to ACC | 1 | Z,C,AC,OV |
| ADDM A,[m] | Add ACC to data memory | 1(1) | Z,C,AC,OV |
| ADD A,x | Add immediate data to ACC | 1 | Z,C,AC,OV |
| ADC A,[m] | Add data memory to ACC with carry | 1 | Z,C,AC,OV |
| ADCM A,[m] | Add ACC to data memory with carry | 1(1) | Z,C,AC,OV |
| SUB A,x | Subtract immediate data from ACC | 1 | Z,C,AC,OV |
| SUB A,[m] | Subtract data memory from ACC | 1 | Z,C,AC,OV |
| SUBM A,[m] | Subtract data memory from ACC with result in data memory | 1(1) | Z,C,AC,OV |
| SBC A,[m] | Subtract data memory from ACC with carry | 1 | Z,C,AC,OV |
| SBCM A,[m] | Subtract data memory from ACC with carry and result in data memory | 1(1) | Z,C,AC,OV |
| DAA [m] | Decimal adjust ACC for addition with result in data memory | 1(1) | C |
| **Logic Operation** | | | |
| AND A,[m] | AND data memory to ACC | 1 | Z |
| OR A,[m] | OR data memory to ACC | 1 | Z |
| XOR A,[m] | Exclusive-OR data memory to ACC | 1 | Z |
| ANDM A,[m] | AND ACC to data memory | 1(1) | Z |
| ORM A,[m] | OR ACC to data memory | 1(1) | Z |
| XORM A,[m] | Exclusive-OR ACC to data memory | 1(1) | Z |
| AND A,x | AND immediate data to ACC | 1 | Z |
| OR A,x | OR immediate data to ACC | 1 | Z |
| XOR A,x | Exclusive-OR immediate data to ACC | 1 | Z |
| CPL [m] | Complement data memory | 1(1) | Z |
| CPLA [m] | Complement data memory with result in ACC | 1 | Z |
| **Increment & Decrement** | | | |
| INCA [m] | Increment data memory with result in ACC | 1 | Z |
| INC [m] | Increment data memory | 1(1) | Z |
| DECA [m] | Decrement data memory with result in ACC | 1 | Z |
| DEC [m] | Decrement data memory | 1(1) | Z |
| **Rotate** | | | |
| RRA [m] | Rotate data memory right with result in ACC | 1 | None |
| RR [m] | Rotate data memory right | 1(1) | None |
| RRCA [m] | Rotate data memory right through carry with result in ACC | 1 | C |
| RRC [m] | Rotate data memory right through carry | 1(1) | C |
| RLA [m] | Rotate data memory left with result in ACC | 1 | None |
| RL [m] | Rotate data memory left | 1(1) | None |
| RLCA [m] | Rotate data memory left through carry with result in ACC | 1 | C |
| RLC [m] | Rotate data memory left through carry | 1(1) | C |
| **Data Move** | | | |
| MOV A,[m] | Move data memory to ACC | 1 | None |
| MOV [m],A | Move ACC to data memory | 1(1) | None |
| MOV A,x | Move immediate data to ACC | 1 | None |
| **Bit Operation** | | | |
| CLR [m].i | Clear bit of data memory | 1(1) | None |
| SET [m].i | Set bit of data memory | 1(1) | None |

| Mnemonic | Description | Instruction Cycle | Flag Affected |
|---|---|---|---|
| **Branch** | | | |
| JMP addr | Jump unconditionally | 2 | None |
| SZ [m] | Skip if data memory is zero | 1(2) | None |
| SZA [m] | Skip if data memory is zero with data movement to ACC | 1(2) | None |
| SZ [m].i | Skip if bit i of data memory is zero | 1(2) | None |
| SNZ [m].i | Skip if bit i of data memory is not zero | 1(2) | None |
| SIZ [m] | Skip if increment data memory is zero | 1(3) | None |
| SDZ [m] | Skip if decrement data memory is zero | 1(3) | None |
| SIZA [m] | Skip if increment data memory is zero with result in ACC | 1(2) | None |
| SDZA [m] | Skip if decrement data memory is zero with result in ACC | 1(2) | None |
| CALL addr | Subroutine call | 2 | None |
| RET | Return from subroutine | 2 | None |
| RET A,x | Return from subroutine and load immediate data to ACC | 2 | None |
| RETI | Return from interrupt | 2 | None |
| **Table Read** | | | |
| TABRDC [m] | Read ROM code (current page) to data memory and TBLH | 2(1) | None |
| TABRDL [m] | Read ROM code (last page) to data memory and TBLH | 2(1) | None |
| **Miscellaneous** | | | |
| NOP | No operation | 1 | None |
| CLR [m] | Clear data memory | 1(1) | None |
| SET [m] | Set data memory | 1(1) | None |
| CLR WDT | Clear Watchdog Timer | 1 | TO,PDF |
| CLR WDT1 | Pre-clear Watchdog Timer | 1 | TO(4),PDF(4) |
| CLR WDT2 | Pre-clear Watchdog Timer | 1 | TO(4),PDF(4) |
| SWAP [m] | Swap nibbles of data memory | 1(1) | None |
| SWAPA [m] | Swap nibbles of data memory with result in ACC | 1 | None |
| HALT | Enter power down mode | 1 | TO,PDF |

Note: x: Immediate data

m: Data memory address

A: Accumulator

i: 0~7 number of bits

addr: Program memory address

√: Flag is affected

–: Flag is not affected

(1): If a loading to the PCL register occurs, the execution cycle of instructions will be delayed for one more cycle (four system clocks).

(2): If a skipping to the next instruction occurs, the execution cycle of instructions will be delayed for one more cycle (four system clocks). Otherwise the original instruction cycle is unchanged.

(3): (1) and (2)

(4): The flags may be affected by the execution status. If the Watchdog Timer is cleared by executing the CLR WDT1 or CLR WDT2 instruction, the TO and PDF are cleared.
Otherwise the TO and PDF flags remain unchanged.

## Instruction Definition

**ADC A,[m]** Add data memory and carry to the accumulator

Description The contents of the specified data memory, accumulator and the carry flag are added simultaneously, leaving the result in the accumulator.

Operation ACC ← ACC+[m]+C

Affected flag(s)

| TO | PDF | OV | Z | AC | C |
|---|---|---|---|---|---|
| — | — | √ | √ | √ | √ |

**ADCM A,[m]** Add the accumulator and carry to data memory

Description The contents of the specified data memory, accumulator and the carry flag are added simultaneously, leaving the result in the specified data memory.

Operation [m] ← ACC+[m]+C

Affected flag(s)

| TO | PDF | OV | Z | AC | C |
|---|---|---|---|---|---|
| — | — | √ | √ | √ | √ |

**ADD A,[m]** Add data memory to the accumulator

Description The contents of the specified data memory and the accumulator are added. The result is stored in the accumulator.

Operation ACC ← ACC+[m]

Affected flag(s)

| TO | PDF | OV | Z | AC | C |
|---|---|---|---|---|---|
| — | — | √ | √ | √ | √ |

**ADD A,x** Add immediate data to the accumulator

Description The contents of the accumulator and the specified data are added, leaving the result in the accumulator.

Operation ACC ← ACC+x

Affected flag(s)

| TO | PDF | OV | Z | AC | C |
|---|---|---|---|---|---|
| — | — | √ | √ | √ | √ |

**ADDM A,[m]** Add the accumulator to the data memory

Description The contents of the specified data memory and the accumulator are added. The result is stored in the data memory.

Operation [m] ← ACC+[m]

Affected flag(s)

| TO | PDF | OV | Z | AC | C |
|---|---|---|---|---|---|
| — | — | √ | √ | √ | √ |

**AND A,[m]** Logical AND accumulator with data memory

Description Data in the accumulator and the specified data memory perform a bitwise logical_AND operation. The result is stored in the accumulator.

Operation ACC ← ACC "AND" [m]

Affected flag(s)

| TO | PDF | OV | Z | AC | C |
|---|---|---|---|---|---|
| — | — | — | √ | — | — |

**AND A,x** Logical AND immediate data to the accumulator

Description Data in the accumulator and the specified data perform a bitwise logical_AND operation. The result is stored in the accumulator.

Operation ACC ← ACC "AND" x

Affected flag(s)

| TO | PDF | OV | Z | AC | C |
|---|---|---|---|---|---|
| — | — | — | √ | — | — |

**ANDM A,[m]** Logical AND data memory with the accumulator

Description Data in the specified data memory and the accumulator perform a bitwise logical_AND operation. The result is stored in the data memory.

Operation [m] ← ACC "AND" [m]

Affected flag(s)

| TO | PDF | OV | Z | AC | C |
|---|---|---|---|---|---|
| — | — | — | √ | — | — |

**CALL addr** Subroutine call

Description The instruction unconditionally calls a subroutine located at the indicated address. The program counter increments once to obtain the address of the next instruction, and pushes this onto the stack. The indicated address is then loaded. Program execution continues with the instruction at this address.

Operation Stack ← PC+1
PC ← addr

Affected flag(s)

| TO | PDF | OV | Z | AC | C |
|---|---|---|---|---|---|
| — | — | — | — | — | — |

**CLR [m]** Clear data memory

Description The contents of the specified data memory are cleared to 0.

Operation [m] ← 00H

Affected flag(s)

| TO | PDF | OV | Z | AC | C |
|---|---|---|---|---|---|
| — | — | — | — | — | — |

**CLR [m].i** Clear bit of data memory

Description The bit i of the specified data memory is cleared to 0.

Operation [m].i ← 0

Affected flag(s)

| TO | PDF | OV | Z | AC | C |
|---|---|---|---|---|---|
| — | — | — | — | — | — |

**CLR WDT** Clear Watchdog Timer

Description The WDT is cleared (clears the WDT). The power down bit (PDF) and time-out bit (TO) are cleared.

Operation WDT ← 00H
PDF and TO ← 0

Affected flag(s)

| TO | PDF | OV | Z | AC | C |
|---|---|---|---|---|---|
| 0 | 0 | — | — | — | — |

**CLR WDT1** Preclear Watchdog Timer

Description Together with CLR WDT2, clears the WDT. PDF and TO are also cleared. Only execution of this instruction without the other preclear instruction just sets the indicated flag which implies this instruction has been executed and the TO and PDF flags remain unchanged.

Operation WDT ← 00H*
PDF and TO ← 0*

Affected flag(s)

| TO | PDF | OV | Z | AC | C |
|---|---|---|---|---|---|
| 0* | 0* | — | — | — | — |

**CLR WDT2** Preclear Watchdog Timer

Description Together with CLR WDT1, clears the WDT. PDF and TO are also cleared. Only execution of this instruction without the other preclear instruction, sets the indicated flag which implies this instruction has been executed and the TO and PDF flags remain unchanged.

Operation WDT ← 00H*
PDF and TO ← 0*

Affected flag(s)

| TO | PDF | OV | Z | AC | C |
|---|---|---|---|---|---|
| 0* | 0* | — | — | — | — |

**CPL [m]** Complement data memory

Description Each bit of the specified data memory is logically complemented (1's complement). Bits which previously contained a 1 are changed to 0 and vice-versa.

Operation [m] ← $\overline{[m]}$

Affected flag(s)

| TO | PDF | OV | Z | AC | C |
|---|---|---|---|---|---|
| — | — | — | √ | — | — |

**CPLA [m]** Complement data memory and place result in the accumulator

Description Each bit of the specified data memory is logically complemented (1′s complement). Bits which previously contained a 1 are changed to 0 and vice-versa. The complemented result is stored in the accumulator and the contents of the data memory remain unchanged.

Operation ACC ← $\overline{[m]}$

Affected flag(s)

| TO | PDF | OV | Z | AC | C |
|---|---|---|---|---|---|
| — | — | — | √ | — | — |

**DAA [m]** Decimal-Adjust accumulator for addition

Description The accumulator value is adjusted to the BCD (Binary Coded Decimal) code. The accumulator is divided into two nibbles. Each nibble is adjusted to the BCD code and an internal carry (AC1) will be done if the low nibble of the accumulator is greater than 9. The BCD adjustment is done by adding 6 to the original value if the original value is greater than 9 or a carry (AC or C) is set; otherwise the original value remains unchanged. The result is stored in the data memory and only the carry flag (C) may be affected.

Operation If ACC.3~ACC.0 >9 or AC=1
then [m].3~[m].0 ← (ACC.3~ACC.0)+6, AC1=$\overline{AC}$
else [m].3~[m].0 ← (ACC.3~ACC.0), AC1=0
and
If ACC.7~ACC.4+AC1 >9 or C=1
then [m].7~[m].4 ← ACC.7~ACC.4+6+AC1,C=1
else [m].7~[m].4 ← ACC.7~ACC.4+AC1,C=C

Affected flag(s)

| TO | PDF | OV | Z | AC | C |
|---|---|---|---|---|---|
| — | — | — | — | — | √ |

**DEC [m]** Decrement data memory

Description Data in the specified data memory is decremented by 1.

Operation [m] ← [m]−1

Affected flag(s)

| TO | PDF | OV | Z | AC | C |
|---|---|---|---|---|---|
| — | — | — | √ | — | — |

**DECA [m]** Decrement data memory and place result in the accumulator

Description Data in the specified data memory is decremented by 1, leaving the result in the accumulator. The contents of the data memory remain unchanged.

Operation ACC ← [m]−1

Affected flag(s)

| TO | PDF | OV | Z | AC | C |
|---|---|---|---|---|---|
| — | — | — | √ | — | — |

**HALT** Enter power down mode

Description This instruction stops program execution and turns off the system clock. The contents of the RAM and registers are retained. The WDT and prescaler are cleared. The power down bit (PDF) is set and the WDT time-out bit (TO) is cleared.

Operation PC ← PC+1
PDF ← 1
TO ← 0

Affected flag(s)

| TO | PDF | OV | Z | AC | C |
|---|---|---|---|---|---|
| 0 | 1 | — | — | — | — |

**INC [m]** Increment data memory

Description Data in the specified data memory is incremented by 1

Operation [m] ← [m]+1

Affected flag(s)

| TO | PDF | OV | Z | AC | C |
|---|---|---|---|---|---|
| — | — | — | √ | — | — |

**INCA [m]** Increment data memory and place result in the accumulator

Description Data in the specified data memory is incremented by 1, leaving the result in the accumulator. The contents of the data memory remain unchanged.

Operation ACC ← [m]+1

Affected flag(s)

| TO | PDF | OV | Z | AC | C |
|---|---|---|---|---|---|
| — | — | — | √ | — | — |

**JMP addr** Directly jump

Description The program counter are replaced with the directly-specified address unconditionally, and control is passed to this destination.

Operation PC ←addr

Affected flag(s)

| TO | PDF | OV | Z | AC | C |
|---|---|---|---|---|---|
| — | — | — | — | — | — |

**MOV A,[m]** Move data memory to the accumulator

Description The contents of the specified data memory are copied to the accumulator.

Operation ACC ← [m]

Affected flag(s)

| TO | PDF | OV | Z | AC | C |
|---|---|---|---|---|---|
| — | — | — | — | — | — |

**MOV A,x** Move immediate data to the accumulator

Description The 8-bit data specified by the code is loaded into the accumulator.

Operation ACC ← x

Affected flag(s)

| TO | PDF | OV | Z | AC | C |
|---|---|---|---|---|---|
| — | — | — | — | — | — |

**MOV [m],A** Move the accumulator to data memory

Description The contents of the accumulator are copied to the specified data memory (one of the data memories).

Operation [m] ←ACC

Affected flag(s)

| TO | PDF | OV | Z | AC | C |
|---|---|---|---|---|---|
| — | — | — | — | — | — |

**NOP** No operation

Description No operation is performed. Execution continues with the next instruction.

Operation PC ← PC+1

Affected flag(s)

| TO | PDF | OV | Z | AC | C |
|---|---|---|---|---|---|
| — | — | — | — | — | — |

**OR A,[m]** Logical OR accumulator with data memory

Description Data in the accumulator and the specified data memory (one of the data memories) perform a bitwise logical_OR operation. The result is stored in the accumulator.

Operation ACC ← ACC ″OR″ [m]

Affected flag(s)

| TO | PDF | OV | Z | AC | C |
|---|---|---|---|---|---|
| — | — | — | √ | — | — |

**OR A,x** Logical OR immediate data to the accumulator

Description Data in the accumulator and the specified data perform a bitwise logical_OR operation. The result is stored in the accumulator.

Operation ACC ← ACC ″OR″ x

Affected flag(s)

| TO | PDF | OV | Z | AC | C |
|---|---|---|---|---|---|
| — | — | — | √ | — | — |

**ORM A,[m]** Logical OR data memory with the accumulator

Description Data in the data memory (one of the data memories) and the accumulator perform a bitwise logical_OR operation. The result is stored in the data memory.

Operation [m] ←ACC ″OR″ [m]

Affected flag(s)

| TO | PDF | OV | Z | AC | C |
|---|---|---|---|---|---|
| — | — | — | √ | — | — |

**RET** Return from subroutine

Description The program counter is restored from the stack. This is a 2-cycle instruction.

Operation PC ← Stack

Affected flag(s)

| TO | PDF | OV | Z | AC | C |
|---|---|---|---|---|---|
| — | — | — | — | — | — |

**RET A,x** Return and place immediate data in the accumulator

Description The program counter is restored from the stack and the accumulator loaded with the specified 8-bit immediate data.

Operation PC ← Stack
ACC ← x

Affected flag(s)

| TO | PDF | OV | Z | AC | C |
|---|---|---|---|---|---|
| — | — | — | — | — | — |

**RETI** Return from interrupt

Description The program counter is restored from the stack, and interrupts are enabled by setting the EMI bit. EMI is the enable master (global) interrupt bit.

Operation PC ← Stack
EMI ← 1

Affected flag(s)

| TO | PDF | OV | Z | AC | C |
|---|---|---|---|---|---|
| — | — | — | — | — | — |

**RL [m]** Rotate data memory left

Description The contents of the specified data memory are rotated 1 bit left with bit 7 rotated into bit 0.

Operation [m].(i+1) ← [m].i; [m].i:bit i of the data memory (i=0~6)
[m].0 ← [m].7

Affected flag(s)

| TO | PDF | OV | Z | AC | C |
|---|---|---|---|---|---|
| — | — | — | — | — | — |

**RLA [m]** Rotate data memory left and place result in the accumulator

Description Data in the specified data memory is rotated 1 bit left with bit 7 rotated into bit 0, leaving the rotated result in the accumulator. The contents of the data memory remain unchanged.

Operation ACC.(i+1) ← [m].i; [m].i:bit i of the data memory (i=0~6)
ACC.0 ← [m].7

Affected flag(s)

| TO | PDF | OV | Z | AC | C |
|---|---|---|---|---|---|
| — | — | — | — | — | — |

**RLC [m]** Rotate data memory left through carry

Description The contents of the specified data memory and the carry flag are rotated 1 bit left. Bit 7 replaces the carry bit; the original carry flag is rotated into the bit 0 position.

Operation [m].(i+1) ← [m].i; [m].i:bit i of the data memory (i=0~6)
[m].0 ← C
C ← [m].7

Affected flag(s)

| TO | PDF | OV | Z | AC | C |
|---|---|---|---|---|---|
| — | — | — | — | — | √ |

**RLCA [m]** Rotate left through carry and place result in the accumulator

Description Data in the specified data memory and the carry flag are rotated 1 bit left. Bit 7 replaces the carry bit and the original carry flag is rotated into bit 0 position. The rotated result is stored in the accumulator but the contents of the data memory remain unchanged.

Operation ACC.(i+1) ← [m].i; [m].i:bit i of the data memory (i=0~6)
ACC.0 ← C
C ← [m].7

Affected flag(s)

| TO | PDF | OV | Z | AC | C |
|---|---|---|---|---|---|
| — | — | — | — | — | √ |

**RR [m]** Rotate data memory right

Description The contents of the specified data memory are rotated 1 bit right with bit 0 rotated to bit 7.

Operation [m].i ← [m].(i+1); [m].i:bit i of the data memory (i=0~6)
[m].7 ← [m].0

Affected flag(s)

| TO | PDF | OV | Z | AC | C |
|---|---|---|---|---|---|
| — | — | — | — | — | — |

**RRA [m]** Rotate right and place result in the accumulator

Description Data in the specified data memory is rotated 1 bit right with bit 0 rotated into bit 7, leaving the rotated result in the accumulator. The contents of the data memory remain unchanged.

Operation ACC.(i) ← [m].(i+1); [m].i:bit i of the data memory (i=0~6)
ACC.7 ← [m].0

Affected flag(s)

| TO | PDF | OV | Z | AC | C |
|---|---|---|---|---|---|
| — | — | — | — | — | — |

**RRC [m]** Rotate data memory right through carry

Description The contents of the specified data memory and the carry flag are together rotated 1 bit right. Bit 0 replaces the carry bit; the original carry flag is rotated into the bit 7 position.

Operation [m].i ← [m].(i+1); [m].i:bit i of the data memory (i=0~6)
[m].7 ← C
C ← [m].0

Affected flag(s)

| TO | PDF | OV | Z | AC | C |
|---|---|---|---|---|---|
| — | — | — | — | — | √ |

**RRCA [m]** Rotate right through carry and place result in the accumulator

Description Data of the specified data memory and the carry flag are rotated 1 bit right. Bit 0 replaces the carry bit and the original carry flag is rotated into the bit 7 position. The rotated result is stored in the accumulator. The contents of the data memory remain unchanged.

Operation ACC.i ← [m].(i+1); [m].i:bit i of the data memory (i=0~6)
ACC.7 ← C
C ← [m].0

Affected flag(s)

| TO | PDF | OV | Z | AC | C |
|---|---|---|---|---|---|
| — | — | — | — | — | √ |

**SBC A,[m]** Subtract data memory and carry from the accumulator

Description The contents of the specified data memory and the complement of the carry flag are subtracted from the accumulator, leaving the result in the accumulator.

Operation ACC ← ACC+$\overline{[m]}$+C

Affected flag(s)

| TO | PDF | OV | Z | AC | C |
|---|---|---|---|---|---|
| — | — | √ | √ | √ | √ |

**SBCM A,[m]** Subtract data memory and carry from the accumulator

Description The contents of the specified data memory and the complement of the carry flag are subtracted from the accumulator, leaving the result in the data memory.

Operation [m] ← ACC+$\overline{[m]}$+C

Affected flag(s)

| TO | PDF | OV | Z | AC | C |
|---|---|---|---|---|---|
| — | — | √ | √ | √ | √ |

**SDZ [m]** Skip if decrement data memory is 0

Description The contents of the specified data memory are decremented by 1. If the result is 0, the next instruction is skipped. If the result is 0, the following instruction, fetched during the current instruction execution, is discarded and a dummy cycle is replaced to get the proper instruction (2 cycles). Otherwise proceed with the next instruction (1 cycle).

Operation Skip if ($\overline{[m]}$–1)=0, [m] ← ([m]–1)

Affected flag(s)

| TO | PDF | OV | Z | AC | C |
|---|---|---|---|---|---|
| — | — | — | — | — | — |

**SDZA [m]** Decrement data memory and place result in ACC, skip if 0

Description The contents of the specified data memory are decremented by 1. If the result is 0, the next instruction is skipped. The result is stored in the accumulator but the data memory remains unchanged. If the result is 0, the following instruction, fetched during the current instruction execution, is discarded and a dummy cycle is replaced to get the proper instruction (2 cycles). Otherwise proceed with the next instruction (1 cycle).

Operation Skip if ([m]–1)=0, ACC ← ([m]–1)

Affected flag(s)

| TO | PDF | OV | Z | AC | C |
|---|---|---|---|---|---|
| — | — | — | — | — | — |

**SET [m]** Set data memory

Description Each bit of the specified data memory is set to 1.

Operation [m] ← FFH

Affected flag(s)

| TO | PDF | OV | Z | AC | C |
|---|---|---|---|---|---|
| — | — | — | — | — | — |

**SET [m]. i** Set bit of data memory

Description Bit i of the specified data memory is set to 1.

Operation [m].i ← 1

Affected flag(s)

| TO | PDF | OV | Z | AC | C |
|---|---|---|---|---|---|
| — | — | — | — | — | — |

**SIZ [m]** Skip if increment data memory is 0

Description The contents of the specified data memory are incremented by 1. If the result is 0, the following instruction, fetched during the current instruction execution, is discarded and a dummy cycle is replaced to get the proper instruction (2 cycles). Otherwise proceed with the next instruction (1 cycle).

Operation Skip if ([m]+1)=0, [m] ← ([m]+1)

Affected flag(s)

| TO | PDF | OV | Z | AC | C |
|---|---|---|---|---|---|
| — | — | — | — | — | — |

**SIZA [m]** Increment data memory and place result in ACC, skip if 0

Description The contents of the specified data memory are incremented by 1. If the result is 0, the next instruction is skipped and the result is stored in the accumulator. The data memory remains unchanged. If the result is 0, the following instruction, fetched during the current instruction execution, is discarded and a dummy cycle is replaced to get the proper instruction (2 cycles). Otherwise proceed with the next instruction (1 cycle).

Operation Skip if ([m]+1)=0, ACC ← ([m]+1)

Affected flag(s)

| TO | PDF | OV | Z | AC | C |
|---|---|---|---|---|---|
| — | — | — | — | — | — |

**SNZ [m].i** Skip if bit i of the data memory is not 0

Description If bit i of the specified data memory is not 0, the next instruction is skipped. If bit i of the data memory is not 0, the following instruction, fetched during the current instruction execution, is discarded and a dummy cycle is replaced to get the proper instruction (2 cycles). Otherwise proceed with the next instruction (1 cycle).

Operation Skip if [m].i≠0

Affected flag(s)

| TO | PDF | OV | Z | AC | C |
|---|---|---|---|---|---|
| — | — | — | — | — | — |

**SUB A,[m]** Subtract data memory from the accumulator

Description The specified data memory is subtracted from the contents of the accumulator, leaving the result in the accumulator.

Operation ACC ← ACC+$\overline{[m]}$+1

Affected flag(s)

| TO | PDF | OV | Z | AC | C |
|---|---|---|---|---|---|
| — | — | √ | √ | √ | √ |

**SUBM A,[m]** Subtract data memory from the accumulator

Description The specified data memory is subtracted from the contents of the accumulator, leaving the result in the data memory.

Operation [m] ← ACC+$\overline{[m]}$+1

Affected flag(s)

| TO | PDF | OV | Z | AC | C |
|---|---|---|---|---|---|
| — | — | √ | √ | √ | √ |

**SUB A,x** Subtract immediate data from the accumulator

Description The immediate data specified by the code is subtracted from the contents of the accumulator, leaving the result in the accumulator.

Operation ACC ← ACC+$\overline{x}$+1

Affected flag(s)

| TO | PDF | OV | Z | AC | C |
|---|---|---|---|---|---|
| — | — | √ | √ | √ | √ |

**SWAP [m]** Swap nibbles within the data memory

Description The low-order and high-order nibbles of the specified data memory (1 of the data memories) are interchanged.

Operation [m].3~[m].0 ↔ [m].7~[m].4

Affected flag(s)

| TO | PDF | OV | Z | AC | C |
|---|---|---|---|---|---|
| — | — | — | — | — | — |

**SWAPA [m]** Swap data memory and place result in the accumulator

Description The low-order and high-order nibbles of the specified data memory are interchanged, writing the result to the accumulator. The contents of the data memory remain unchanged.

Operation ACC.3~ACC.0 ← [m].7~[m].4
ACC.7~ACC.4 ← [m].3~[m].0

Affected flag(s)

| TO | PDF | OV | Z | AC | C |
|---|---|---|---|---|---|
| — | — | — | — | — | — |

**SZ [m]** Skip if data memory is 0

Description If the contents of the specified data memory are 0, the following instruction, fetched during the current instruction execution, is discarded and a dummy cycle is replaced to get the proper instruction (2 cycles). Otherwise proceed with the next instruction (1 cycle).

Operation Skip if [m]=0

Affected flag(s)

| TO | PDF | OV | Z | AC | C |
|---|---|---|---|---|---|
| — | — | — | — | — | — |

**SZA [m]** Move data memory to ACC, skip if 0

Description The contents of the specified data memory are copied to the accumulator. If the contents is 0, the following instruction, fetched during the current instruction execution, is discarded and a dummy cycle is replaced to get the proper instruction (2 cycles). Otherwise proceed with the next instruction (1 cycle).

Operation Skip if [m]=0

Affected flag(s)

| TO | PDF | OV | Z | AC | C |
|---|---|---|---|---|---|
| — | — | — | — | — | — |

**SZ [m].i** Skip if bit i of the data memory is 0

Description If bit i of the specified data memory is 0, the following instruction, fetched during the current instruction execution, is discarded and a dummy cycle is replaced to get the proper instruction (2 cycles). Otherwise proceed with the next instruction (1 cycle).

Operation Skip if [m].i=0

Affected flag(s)

| TO | PDF | OV | Z | AC | C |
|---|---|---|---|---|---|
| — | — | — | — | — | — |

**TABRDC [m]** Move the ROM code (current page) to TBLH and data memory

Description The low byte of ROM code (current page) addressed by the table pointer (TBLP) is moved to the specified data memory and the high byte transferred to TBLH directly.

Operation [m] ← ROM code (low byte)
TBLH ← ROM code (high byte)

Affected flag(s)

| TO | PDF | OV | Z | AC | C |
|---|---|---|---|---|---|
| — | — | — | — | — | — |

**TABRDL [m]** Move the ROM code (last page) to TBLH and data memory

Description The low byte of ROM code (last page) addressed by the table pointer (TBLP) is moved to the data memory and the high byte transferred to TBLH directly.

Operation [m] ← ROM code (low byte)
TBLH ← ROM code (high byte)

Affected flag(s)

| TO | PDF | OV | Z | AC | C |
|---|---|---|---|---|---|
| — | — | — | — | — | — |

**XOR A,[m]** Logical XOR accumulator with data memory

Description Data in the accumulator and the indicated data memory perform a bitwise logical Exclusive_OR operation and the result is stored in the accumulator.

Operation ACC ← ACC ″XOR″ [m]

Affected flag(s)

| TO | PDF | OV | Z | AC | C |
|---|---|---|---|---|---|
| — | — | — | √ | — | — |

**XORM A,[m]** Logical XOR data memory with the accumulator

Description Data in the indicated data memory and the accumulator perform a bitwise logical Exclusive_OR operation. The result is stored in the data memory. The 0 flag is affected.

Operation [m] ← ACC ″XOR″ [m]

Affected flag(s)

| TO | PDF | OV | Z | AC | C |
|---|---|---|---|---|---|
| — | — | — | √ | — | — |

**XOR A,x** Logical XOR immediate data to the accumulator

Description Data in the accumulator and the specified data perform a bitwise logical Exclusive_OR operation. The result is stored in the accumulator. The 0 flag is affected.

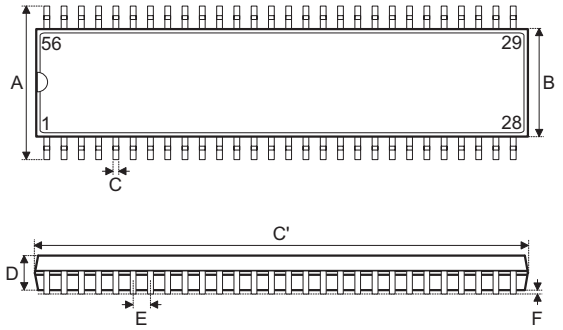
Operation ACC ← ACC ″XOR″ x

Affected flag(s)

| TO | PDF | OV | Z | AC | C |
|---|---|---|---|---|---|
| — | — | — | √ | — | — |

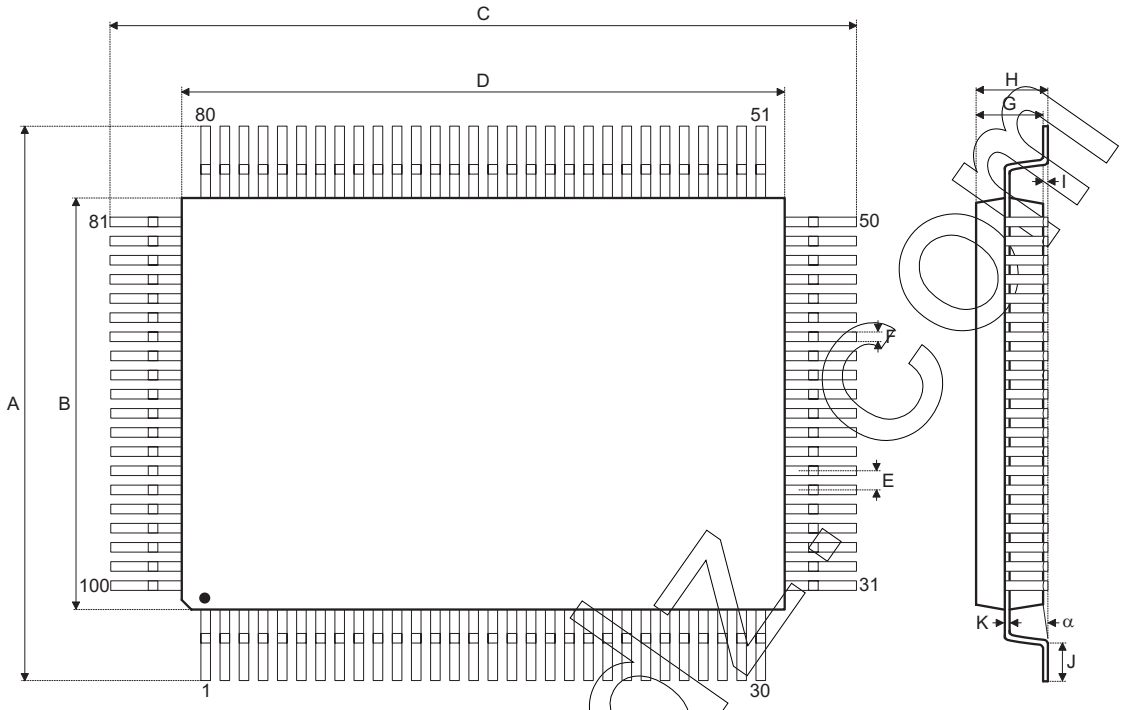
## Package Information

**56-pin SSOP (300mil) Outline Dimensions**




| Symbol | Dimensions in mil | | |
|---|---|---|---|
| | Min. | Nom. | Max. |
| A | 395 | — | 420 |
| B | 291 | — | 299 |
| C | 8 | — | 12 |
| C′ | 720 | — | 730 |
| D | 89 | — | 99 |
| E | — | 25 | — |
| F | 4 | — | 10 |
| G | 25 | — | 35 |
| H | 4 | — | 12 |
| α | 0° | — | 8° |

**100-pin QFP (14×20) Outline Dimensions**




| Symbol | Dimensions in mm | | |
|---|---|---|---|
| | Min. | Nom. | Max. |
| A | 18.50 | — | 19.20 |
| B | 13.90 | — | 14.10 |
| C | 24.50 | — | 25.20 |
| D | 19.90 | — | 20.10 |
| E | — | 0.65 | — |
| F | — | 0.30 | — |
| G | 2.50 | — | 3.10 |
| H | — | — | 3.40 |
| I | — | 0.10 | — |
| J | 1 | — | 1.40 |
| K | 0.10 | — | 0.20 |
| α | 0° | — | 7° |

**Holtek Semiconductor Inc. (Headquarters)**
No.3, Creation Rd. II, Science Park, Hsinchu, Taiwan
Tel: 886-3-563-1999
Fax: 886-3-563-1189
http://www.holtek.com.tw

**Holtek Semiconductor Inc. (Sales Office)**
4F-2, No. 3-2, YuanQu St., Nankang Software Park, Taipei 115, Taiwan
Tel: 886-2-2655-7070
Fax: 886-2-2655-7373
Fax: 886-2-2655-7383 (International sales hotline)

**Holtek Semiconductor (Shanghai) Inc.**
7th Floor, Building 2, No.889, Yi Shan Rd., Shanghai, China
Tel: 021-6485-5560
Fax: 021-6485-0313
http://www.holtek.com.cn

**Holtek Semiconductor (Hong Kong) Ltd.**
Block A, 3/F, Tin On Industrial Building, 777-779 Cheung Sha Wan Rd., Kowloon, Hong Kong
Tel: 852-2-745-8288
Fax: 852-2-742-8657

**Holmate Semiconductor, Inc.**
46712 Fremont Blvd., Fremont, CA 94538
Tel: 510-252-9880
Fax: 510-252-9885
http://www.holmate.com