## Technical Document

- Tools Information
- FAQs
- Application Note
  - HA0086E HT48E MCU Series - Using Assembly Language to Write to the 1K EEPROM Data Memory
  - HA0087E HT48E MCU Series - Using C Language to Write to the 1K EEPROM Data Memory
  - HA0088E HT48E MCU Series - Using Assembly Language to Write to the 2K EEPROM Data Memory
  - HA0089E HT48E MCU Series - Using C Language to Write to the 2K EEPROM Data Memory

## Features

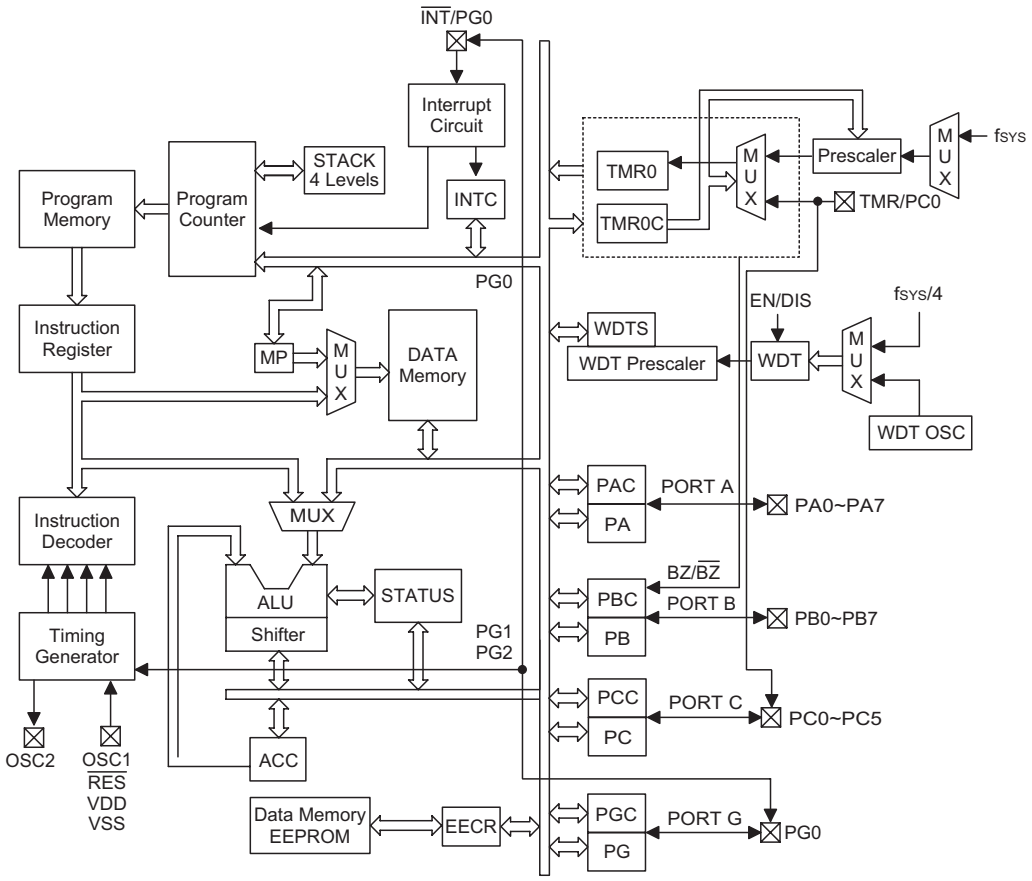
- Operating voltage:
  $f_{SYS}$=4MHz: 2.2V~5.5V
  $f_{SYS}$=8MHz: 3.3V~5.5V
- Low voltage reset function
- 23 bidirectional I/O lines (max.)
- 1 interrupt input shared with an I/O line
- 8-bit programmable timer/event counter with overflow interrupt and 8-stage prescaler
- On-chip crystal and RC oscillator
- Watchdog Timer
- 1,000 erase/write cycles MTP program memory
- 2048×14 program memory ROM (MTP)
- 128×8 data memory EEPROM
- 96×8 data memory RAM
- Buzzer driving pair and PFD supported
- HALT function and wake-up feature reduce power consumption
- 4-level subroutine nesting
- Up to 0.5μs instruction cycle with 8MHz system clock at $V_{DD}$=5V
- Bit manipulation instruction
- 14-bit table read instruction
- 63 powerful instructions
- $10^6$ erase/write cycles EEPROM data memory
- EEPROM data retention > 10 years
- All instructions in one or two machine cycles
- In system programming (ISP)
- 24/28-pin SKDIP/SOP package

## General Description

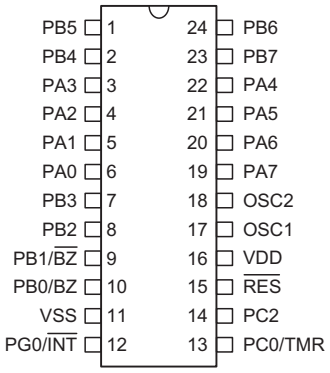
The HT48E30 is an 8-bit high performance, RISC architecture microcontroller device specifically designed for multiple I/O control product applications.

The advantages of low power consumption, I/O flexibility, timer functions, oscillator options, HALT and wake-up functions, watchdog timer, buzzer driver, as well as low cost, enhance the versatility of these devices to suit a wide range of application possibilities such as industrial control, consumer products, subsystem controllers, etc.

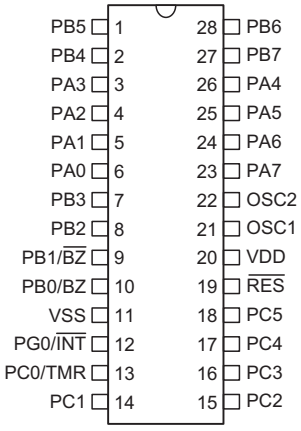
## Block Diagram




## Pin Assignment




**HT48E30**
**— 24 SKDIP-A/SOP-A**




**HT48E30**
**— 28 SKDIP-A/SOP-A**

## Pad Description

| Pad Name | I/O | Options | Description |
|---|---|---|---|
| PA0~PA7 | I/O | Pull-high*<br>Wake-up<br>CMOS/Schmitt trigger<br>Input | Bidirectional 8-bit input/output port. Each bit can be configured as a wake-up input by options. Software instructions determine the CMOS output or Schmitt trigger or CMOS input (depends on options) with pull-high resistor (determined by 1-bit pull-high options). |
| PB0/BZ<br>PB1/$\overline{BZ}$<br>PB2~PB7 | I/O | Pull-high*<br>PB0 or BZ<br>PB1 or $\overline{BZ}$ | Bidirectional 8-bit input/output port. Software instructions determine the CMOS output or Schmitt trigger input with pull-high resistor (determined by pull-high options).<br>The PB0 and PB1 are pin-shared with the BZ and $\overline{BZ}$, respectively. Once the PB0 or PB1 is selected as buzzer driving outputs, the output signals come from an internal PFD generator (shared with timer/event counter). |
| VSS | — | — | Negative power supply, ground |
| PG0/$\overline{INT}$ | I/O | Pull-high* | Bidirectional I/O lines. Software instructions determine the CMOS output or Schmitt trigger input with pull-high resistor (determined by 1-bit pull-high options). This external interrupt input is pin-shared with PG0. The external interrupt input is activated on a high to low transition. |
| PC0/TMR<br>PC1~PC5 | I/O | Pull-high* | Bidirectional I/O lines. Software instructions determine the CMOS output or Schmitt trigger input with pull-high resistor (determined by 1-bit pull-high options). The timer input are pin-shared with PC0. |
| $\overline{RES}$ | I | — | Schmitt trigger reset input. Active low. |
| VDD | — | — | Positive power supply |
| OSC1<br>OSC2 | I<br>O | Crystal or RC | OSC1and OSC2 are connected to an RC network or Crystal (determined by options) for the internal system clock. In the case of RC operation, OSC2 is the output terminal for 1/4 system clock. |

Note: "*" The pull-high resistors of each I/O port (PA, PB, PC, PG) are controlled by a 1-bit option.
CMOS or Schmitt trigger option of port A is controlled by a 1-bit option.

## 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 (Crystal OSC) | 3V | No load, $f_{SYS}$=4MHz | — | 0.6 | 1.5 | mA |
| | | 5V | | — | 2 | 4 | mA |
| $I_{DD2}$ | Operating Current (RC OSC) | 3V | No load, $f_{SYS}$=4MHz | — | 0.8 | 1.5 | mA |
| | | 5V | | — | 2.5 | 4 | mA |
| $I_{DD3}$ | Operating Current<br>(Crystal OSC, RC OSC) | 5V | No load, $f_{SYS}$=8MHz | — | 4 | 8 | mA |

| Symbol | Parameter | Test Conditions | | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|---|---|
| | | $V_{DD}$ | Conditions | | | | |
| $I_{STB1}$ | Standby Current (WDT Enabled) | 3V | No load*, system HALT | — | — | 10 | μA |
| | | 5V | | — | — | 15 | μA |
| $I_{STB2}$ | Standby Current (WDT Disabled) | 3V | No load*, system HALT | — | — | 3 | μA |
| | | 5V | | — | — | 5 | μA |
| $V_{IL1}$ | Input Low Voltage for I/O Ports | — | — | 0 | — | $0.3V_{DD}$ | V |
| $V_{IH1}$ | Input High Voltage for I/O Ports | — | — | $0.7V_{DD}$ | — | $V_{DD}$ | 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 |
| $V_{LVR}$ | Low Voltage Reset Voltage | — | LVR enabled | 2.7 | 3.0 | 3.3 | V |
| $I_{OL}$ | I/O Port Sink Current | 3V | $V_{OL}=0.1V_{DD}$ | 4 | 8 | — | mA |
| | | 5V | $V_{OL}=0.1V_{DD}$ | 10 | 20 | — | mA |
| $I_{OH}$ | I/O Port Source Current | 3V | $V_{OH}=0.9V_{DD}$ | –2 | –4 | — | mA |
| | | 5V | $V_{OH}=0.9V_{DD}$ | –5 | –10 | — | mA |
| $R_{PH}$ | Pull-high Resistance | 3V | — | 20 | 60 | 100 | kΩ |
| | | 5V | — | 10 | 30 | 50 | kΩ |

Note: "*" All tests are conducted with the I/O pins setup as outputs and set to a low value.

## A.C. Characteristics

Ta=25°C

| Symbol | Parameter | Test Conditions | | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|---|---|
| | | $V_{DD}$ | Conditions | | | | |
| $f_{SYS1}$ | System Clock (Crystal OSC) | — | 2.2V~5.5V | 400 | — | 4000 | kHz |
| | | — | 3.3V~5.5V | 400 | — | 8000 | kHz |
| $f_{SYS2}$ | System Clock (RC OSC) | — | 2.2V~5.5V | 400 | — | 4000 | kHz |
| | | — | 3.3V~5.5V | 400 | — | 8000 | kHz |
| $f_{TIMER}$ | Timer I/P Frequency (TMR) | — | 2.2V~5.5V | 0 | — | 4000 | kHz |
| | | — | 3.3V~5.5V | 0 | — | 8000 | kHz |
| $t_{WDTOSC}$ | Watchdog Oscillator Period | 3V | — | 45 | 90 | 180 | μs |
| | | 5V | — | 32 | 65 | 130 | μs |
| $t_{WDT1}$ | Watchdog Time-out Period (WDT OSC) | 3V | Without WDT prescaler | 11 | 23 | 46 | ms |
| | | 5V | | 8 | 17 | 33 | ms |
| $t_{WDT2}$ | Watchdog Time-out Period (System Clock) | — | Without WDT prescaler | — | 1024 | — | $t_{SYS}$ |
| $t_{RES}$ | External Reset Low Pulse Width | — | — | 1 | — | — | μs |
| $t_{SST}$ | System Start-up Timer Period | — | Wake-up from HALT | — | 1024 | — | $t_{SYS}$ |
| $t_{INT}$ | Interrupt Pulse Width | — | — | 1 | — | — | μs |

## Functional Description

### Execution Flow

The HT48E30 system clock is derived from either a crystal or an RC oscillator and is internally divided into four non-overlapping clocks. One instruction cycle consists of four system clock cycles.

Instruction fetching and execution are pipelined in such a way that a fetch takes an instruction cycle while decoding and execution takes the next instruction cycle. This pipelining scheme ensures that instructions are effectively executed in one cycle. If an instruction changes the contents of the program counter, two cycles are required to complete the instruction.

### Program Counter – PC

The program counter (PC) controls the sequence in which the instructions stored in the program ROM are executed and its contents specify a full range of program memory.

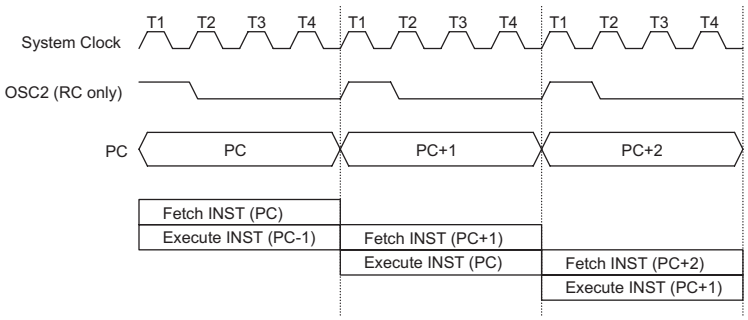
After accessing a program memory word to fetch an instruction code, the contents of the program counter are incremented by one. The program counter then points to the memory word containing the next instruction code.

When executing a jump instruction, conditional skip execution, loading into the PCL register, subroutine call or return from subroutine, initial reset, internal interrupt, external interrupt or return from interrupt, the PC 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 byte of the program counter (PCL) is a readable and writeable register (06H). Moving data into the PCL performs a short jump. The destination will be within the current program ROM page.

When a control transfer takes place, an additional dummy cycle is required.




**Execution Flow**

| Mode | Program Counter | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | *10 | *9 | *8 | *7 | *6 | *5 | *4 | *3 | *2 | *1 | *0 |
| Initial Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| External Interrupt | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| Timer/Event Counter Overflow | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| Skip | Program Counter+2 | | | | | | | | | | |
| Loading PCL | *10 | *9 | *8 | @7 | @6 | @5 | @4 | @3 | @2 | @1 | @0 |
| Jump, Call Branch | #10 | #9 | #8 | #7 | #6 | #5 | #4 | #3 | #2 | #1 | #0 |
| Return from Subroutine | S10 | S9 | S8 | S7 | S6 | S5 | S4 | S3 | S2 | S1 | S0 |

**Program Counter**

Note: *10~*0: Program counter bits
#10~#0: Instruction code bits
S10~S0: Stack register bits
@7~@0: PCL bits

### In System Programming

In system programming allows programming and reprogramming of HT48EXX microcontroller on application circuit board, this will save time and money, both during development in the lab. Using a simple 3-wire interface, the ISP communicates serially with the HT48EXX microcontroller, reprogramming program memory and EEPROM data memory on the chip.

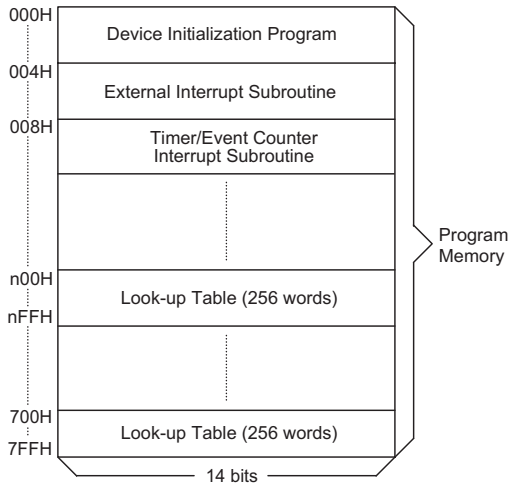
| Pin Name | Function | Description |
|---|---|---|
| PA0 | SDATA | Serial data input/output |
| PA4 | SCLK | Serial clock input |
| RES | RESET | Device reset |
| VDD | VDD | Power supply |
| VSS | VSS | Ground |

**ISP Pin Assignments**

### Program Memory – ROM

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 2048×14 bits, addressed by the program counter and table pointer.

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




Note: n ranges from 0 to 7

**Program Memory**

- Location 000H
  This area is reserved for program initialization. After a chip reset, the program always begins execution at location 000H.
- Location 004H
  This area is reserved for the external interrupt service program. If the $\overline{INT}$ input pin is activated, the interrupt is enabled and the stack is not full, the program begins execution at location 004H.
- Location 008H
  This area is reserved for the timer/event counter interrupt service program. If a timer interrupt results from a timer/event counter overflow, and if the interrupt is enabled and the stack is not full, the program begins execution at location 008H.
- Table location
  Any location in the program memory space can be used as look-up tables. The instructions "TABRDC [m]" (the current page, one 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 TBLH (08H). Only the destination of the lower-order byte in the table is well-defined, the other bits of the table word are transferred to the lower portion of TBLH, and the remaining 2-bits words are read as "0". The Table Higher-order byte register (TBLH) is read only. The table pointer (TBLP) is a read/write register (07H), which indicates the table location. Before accessing the table, the location must be placed in the 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 the TBLH in the main routine are likely to be changed by the table read instruction used in the ISR. Errors can occur. In other words, 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 the 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 has been backed up. All table related instructions require two cycles to complete the operation. These areas may function as normal program memory depending upon the requirements.

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

**Table Location**

Note: *10~*0: Table location bits P10~P8: Current program counter bits

@7~@0: Table pointer bits

### Stack Register – STACK

This is a special part of the memory which is used to save the contents of the Program Counter only. The stack is organized into 4 levels and is neither part of the data nor part of the program space, and is neither readable nor writeable. The activated level is indexed by the Stack Pointer (SP) and is neither readable nor writeable. At a subroutine call or interrupt acknowledge signal, 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.

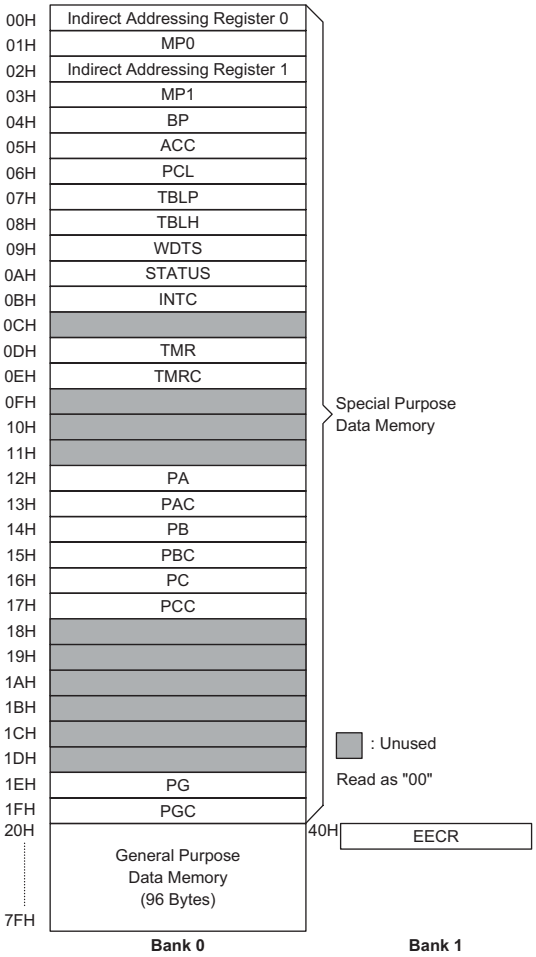
If the stack is full and a non-masked interrupt takes place, the interrupt request flag will be recorded but the acknowledge signal will be inhibited. When the stack pointer is decremented (by RET or RETI), the interrupt will be serviced. This feature prevents stack overflow allowing the programmer to use the structure more easily. In a 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 4 return addresses are stored).

### Data Memory – RAM

The data memory has a capacity of 115×8 bits and is divided into two functional groups: special function registers and general purpose data memory (96×8). Most are read/write, but some are read only.

The special function registers include the Indirect addressing registers (IAR0;00H), Timer/event counter (TMR;0DH), Timer/event counter control register (TMRC;0EH), Program counter lower-order byte register (PCL;06H), Memory pointer registers (MP;01H), Accumulator (ACC;05H), Table pointer (TBLP;07H), Table higher-order byte register (TBLH;08H), Status register (STATUS;0AH), Interrupt control register (INTC;0BH), Watchdog Timer option setting register (WDTS;09H), I/O registers (PA;12H, PB;14H, PC;16H, PG;1EH) and I/O control registers (PAC;13H, PBC;15H, PCC;17H, PGC;1FH). The remaining space before the 20H is reserved for future expanded usage and reading these locations will return the result ″00H″. The general purpose data memory, addressed from 20H to 7FH, 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 reset by ″SET [m].i″ and ″CLR [m].i″. They are also indirectly accessible through memory pointer registers (MP). The control register of the EEPROM data memory is located at [40H] in Bank 1.




**RAM Mapping**

### Indirect Addressing Register

Location 00H and 02H are indirect addressing registers that are not physically implemented. Any read/write operation on [00H] and [02H] access the RAM pointed to by MP0 (01H) and MP1 (03H), respectively. Reading location 00H or 02H indirectly returns the result 00H. While, writing it indirectly leads to no operation.

The function of data movement between two indirect addressing registers is not supported. The memory pointer registers, MP0 and MP1, are both 7-bit registers used to access the RAM by combining corresponding indirect addressing registers. MP0 can only be applied to data memory in Bank 0, while MP1 can be applied to data memory in Bank 0 and Bank 1.

### 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 a 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 during 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 may corrupt the status register, precautions must be taken to save it properly.

### Interrupt

The device provides an external interrupt and internal timer/event counter interrupts. The Interrupt Control Register (INTC;0BH) 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 the EMI bit). This scheme may prevent any further interrupt nesting. Other interrupt requests may occur during this interval but only the interrupt request flag is recorded. If a certain interrupt requires servicing within the service routine, the EMI bit and the corresponding bit of the INTC may be set 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 decremented. If immediate service is desired, the stack must be prevented from becoming full.

All these kinds of interrupts have a wake-up capability. As an interrupt is serviced, a control transfer occurs by pushing the program counter onto the stack, followed by a branch to a subroutine at specified location in the program memory. Only the program counter is pushed onto the stack. If the contents of the register or status register (STATUS) are altered by the interrupt service program which corrupts the desired control sequence, the contents should be saved in advance.

External interrupts are triggered by a high to low transition of the $\overline{\text{INT}}$ and the related interrupt request flag (EIF; bit 4 of INTC) will be set. When the interrupt is enabled, the stack is not full and the external interrupt is active, a subroutine call to location 04H will occur. The interrupt

| Bit No. | Label | Function |
|---|---|---|
| 0 | C | 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. |
| 1 | AC | 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. |
| 2 | Z | Z is set if the result of an arithmetic or logic operation is zero; otherwise Z is cleared. |
| 3 | OV | OV is set if an 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. |
| 4 | PDF | PDF is cleared by a system power-up or executing the ″CLR WDT″ instruction.<br>PDF is set by executing the ″HALT″ instruction. |
| 5 | TO | TO is cleared by a system power-up or executing the ″CLR WDT″ or ″HALT″ instruction.<br>TO is set by a WDT time-out. |
| 6~7 | — | Unused bit, read as ″0″ |

**Status (0AH) Register**

| Bit No. | Label | Function |
|---|---|---|
| 0 | EMI | Controls the master (global) interrupt (1= enabled; 0= disabled) |
| 1 | EEI | Controls the external interrupt (1= enabled; 0= disabled) |
| 2 | ETI | Controls the Timer/Event Counter 0 interrupt (1= enabled; 0= disabled) |
| 3, 6~7 | — | Unused bit, read as ″0″ |
| 4 | EIF | External interrupt request flag (1= active; 0= inactive) |
| 5 | TF | Internal Timer/Event Counter 0 request flag (1= active; 0= inactive) |

**INTC (0BH) Register**

request flag (EIF) and EMI bits will be cleared to disable other interrupts.

The internal timer/event counter interrupt is initialized by setting the timer/event counter interrupt request flag (TF; bit 5 of INTC), caused by a timer overflow. When the interrupt is enabled, the stack is not full and the TF bit is set, a subroutine call to location 08H will occur. The related interrupt request flag (TF) will be reset and the EMI bit cleared to disable further interrupts.

During the execution of an interrupt subroutine, other interrupt acknowledge signals are held until the ″RETI″ instruction is executed or the EMI bit and the related interrupt control bit are set to 1 (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 the rising edges of two consecutive T2 pulses, will be serviced on the latter of the two T2 pulses, if the corresponding interrupts are enabled. In the case of simultaneous requests the following table shows the priority that is applied. These can be masked by resetting the EMI bit.

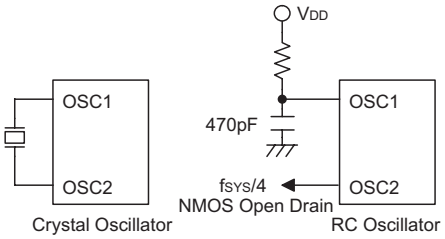
| Interrupt Source | Priority | Vector |
|---|---|---|
| External Interrupt | 1 | 04H |
| Timer/Event Counter Overflow | 2 | 08H |

The timer/event counter interrupt request flag (TF), external interrupt request flag (EIF), enable timer/event counter interrupt bit (ETI), enable external interrupt bit (EEI) and enable master interrupt bit (EMI) constitute an interrupt control register (INTC) which is located at 0BH in the data memory. EMI, EEI, ETI are used to control the enabling/disabling of interrupts. These bits prevent the requested interrupt from being serviced. Once the interrupt request flags (TF, EIF) are set, they will remain in the INTC register until the interrupts are serviced or cleared by a software instruction.

It is recommended that a program does not use the ″CALL subroutine″ within the interrupt subroutine. 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.

**Oscillator Configuration**

There are 2 oscillator circuits in the microcontroller.




**System Oscillator**

All of them are designed for system clocks, namely, external RC oscillator and external Crystal oscillator, which are determined by options. No matter what oscillator type is selected, the signal provides the system clock. The HALT mode stops the system oscillator and ignores an external signal to conserve power.

If an RC oscillator is used, an external resistor between OSC1 and VDD is required and the resistance must range from 24kΩ to 1MΩ. The system clock, divided by 4, is available on OSC2, which can be used to synchronize external logic. The RC oscillator provides the most cost effective solution. However, the frequency of oscillation may vary with VDD, temperatures and the chip itself due to process variations. It is, therefore, not suitable for timing sensitive operations where an accurate oscillator frequency is desired.

If a Crystal oscillator is used, a crystal across OSC1 and OSC2 is needed to provide the feedback and phase shift required for the oscillator. No other external components are required. In stead of a crystal, a resonator can also be connected between OSC1 and OSC2 to obtain a frequency reference, but two external capacitors in OSC1 and OSC2 are required.

The WDT oscillator is a free running on-chip RC oscillator, and no external components are required. Even if the system enters the power down mode and the system clock is stopped, the oscillator still works within a period of 65μs at 5V. The WDT oscillator can be disabled by options to conserve power.

**Watchdog Timer – WDT**

The WDT clock source is implemented by a dedicated RC oscillator (WDT oscillator), instruction clock (system clock divided by 4), determines the options. This timer is designed to prevent a software malfunction or sequence from jumping to an unknown location with unpredictable 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.

Once the internal WDT oscillator (RC oscillator with a period of 65μs at 5V normally) is selected, it is first divided by 256 (8-stage) to get the nominal time-out period of 18.6ms at 5V. This time-out period may vary with temperatures, VDD and process variations. By invoking the WDT prescaler, longer time-out periods can be realized. Writing data to WS2, WS1, WS0 (bit 2,1,0 of the WDTS) can give different time-out periods. If WS2, WS1, and WS0 are all equal to 1, the division ratio is up to 1:128, and the maximum time-out period is 2.4s at 5V seconds. If the WDT oscillator is disabled, the WDT clock may still come from the instruction clock and operates in the same manner except that in the HALT state the WDT may stop counting and lose its protecting purpose. In this situation the logic can only be restarted by an external logic. The high nibble and bit 3 of the WDTS are reserved for user's defined flags, which can be used to indicate some specified status.

If the device operates in a noisy environment, using the on-chip RC oscillator (WDT OSC) is strongly recommended, since the HALT will stop the system clock.

| WS2 | WS1 | WS0 | Division Ratio |
|---|---|---|---|
| 0 | 0 | 0 | 1:1 |
| 0 | 0 | 1 | 1:2 |
| 0 | 1 | 0 | 1:4 |
| 0 | 1 | 1 | 1:8 |
| 1 | 0 | 0 | 1:16 |
| 1 | 0 | 1 | 1:32 |
| 1 | 1 | 0 | 1:64 |
| 1 | 1 | 1 | 1:128 |

**WDTS (09H) Register**

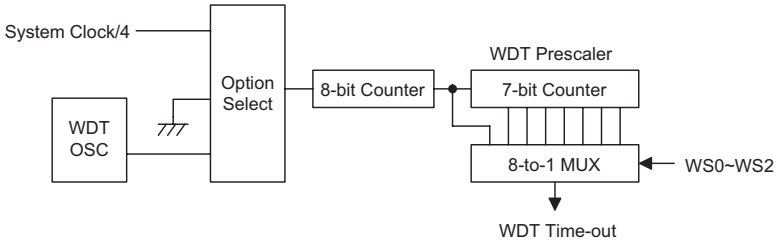
The WDT overflow under normal operation will initialize a "chip reset" and set the status bit "TO". But in the HALT mode, the overflow will initialize a "warm reset" and only the Program Counter and SP are reset to zero. To clear the contents of WDT (including the WDT prescaler), three methods are adopted; external reset (a low level to $\overline{RES}$), software instruction and a "HALT" instruction. The software instruction includes "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 option – "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. CLRWDT 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.

**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 WDT oscillator remains running (if the WDT oscillator is selected).
- The contents of the on chip RAM and registers remain unchanged.
- WDT and WDT prescaler will be cleared and recounted again (if the WDT clock is from the WDT oscillator).
- 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 a system power-up or executing the "CLR WDT" instruction and is set when executing the "HALT" instruction. The TO flag is set if a WDT time-out occurs, and causes a wake-up that only resets the Program Counter and SP; the others remain in their original status.




**Watchdog Timer**

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 options. Awakening from an I/O port stimulus, the program will resume execution of the next instruction. If it awakens from an interrupt, two sequence may occur. 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, a 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 (system clock period) to resume to normal operation. In other words, a dummy period will be inserted after a wake-up. If the wake-up results from an interrupt acknowledge signal, 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.

**Reset**

There are three ways in which a reset can occur:

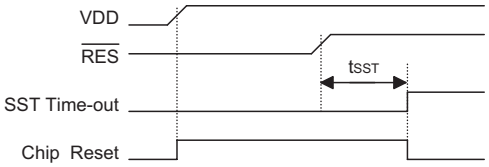
- $\overline{\text{RES}}$ reset during normal operation
- $\overline{\text{RES}}$ reset during HALT
- WDT time-out reset during normal operation

The time-out during HALT is different from other chip reset conditions, since it can perform a "warm reset" that resets only the Program Counter 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".

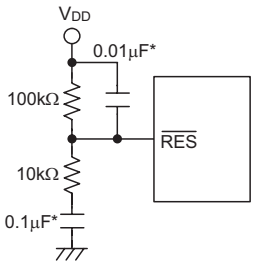
| TO | PDF | RESET Conditions |
|---|---|---|
| 0 | 0 | $\overline{\text{RES}}$ reset during power-up |
| u | u | $\overline{\text{RES}}$ reset during normal operation |
| 0 | 1 | $\overline{\text{RES}}$ wake-up HALT |
| 1 | u | WDT time-out during normal operation |
| 1 | 1 | WDT wake-up HALT |

Note: "u" stands for unchanged

To guarantee that the system oscillator is started and stabilized, the SST (System Start-up Timer) provides an extra delay of 1024 system clock pulses when the system reset (power-up, WDT time-out or $\overline{\text{RES}}$ reset) or the system awakes from the HALT state.

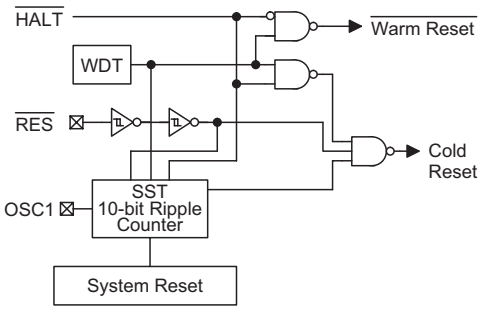



**Reset Timing Chart**




**Reset Circuit**

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




**Reset Configuration**

When a system reset occurs, the SST delay is added during the reset period. Any wake-up from HALT will enable an SST delay.

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

The functional unit chip reset status are shown below.

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

The registers status is summarized in the following table.

| Register | Reset (Power-on) | WDT Time-out (Normal Operation) | $\overline{RES}$ Reset (Normal Operation) | $\overline{RES}$ Reset (HALT) | WDT Time-out (HALT)* |
|---|---|---|---|---|---|
| MP0 | -xxx xxxx | -uuu uuuu | -uuu uuuu | -uuu uuuu | -uuu uuuu |
| MP1 | -xxx xxxx | -uuu uuuu | -uuu uuuu | -uuu uuuu | -uuu uuuu |
| BP | 0000 0000 | 0000 0000 | 0000 0000 | 0000 0000 | uuuu uuuu |
| ACC | xxxx xxxx | uuuu uuuu | uuuu uuuu | uuuu uuuu | uuuu uuuu |
| Program Counter | 000H | 000H | 000H | 000H | 000H |
| TBLP | xxxx xxxx | uuuu uuuu | uuuu uuuu | uuuu uuuu | uuuu uuuu |
| TBLH | --xx xxxx | --uu uuuu | --uu uuuu | --uu uuuu | --uu uuuu |
| WDTS | 0000 0111 | 0000 0111 | 0000 0111 | 0000 0111 | uuuu uuuu |
| STATUS | --00 xxxx | --1u uuuu | --uu uuuu | --01 uuuu | --11 uuuu |
| INTC | --00 0000 | --00 0000 | --00 0000 | --00 0000 | --uu uuuu |
| TMR | xxxx xxxx | xxxx xxxx | xxxx xxxx | xxxx xxxx | uuuu uuuu |
| TMRC | 00-0 1000 | 00-0 1000 | 00-0 1000 | 00-0 1000 | uu-u uuuu |
| 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 | --11 1111 | --11 1111 | --11 1111 | --11 1111 | --uu uuuu |
| PCC | --11 1111 | --11 1111 | --11 1111 | --11 1111 | --uu uuuu |
| PG | ---- ---1 | ---- ---1 | ---- ---1 | ---- ---1 | ---- ---u |
| PGC | ---- ---1 | ---- ---1 | ---- ---1 | ---- ---1 | ---- ---u |
| EECR | 1000 ---- | 1000 ---- | 1000 ---- | 1000 ---- | uuuu ---- |

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

**Timer/Event Counter**

Timer/event counters (TMR) is implemented in the microcontroller. The timer/event counter contains an 8-bit programmable count-up counter and the clock may come from an external source or from the system clock.

Using the internal clock sources, there are 2 reference time-bases for the timer/event counter. The internal clock source can be selected as coming from $f_{SYS}$ or by options. Using an external clock input allows the user to count external events, measure time internals or pulse widths, or generate an accurate time base. While using the internal clock allows the user to generate an accurate time base.

The timer/event counter can generate PFD signals by using external or internal clock and the PFD frequency is determine by the equation $f_{INT}/[2\times(256\text{-}N)]$.

There are 2 registers related to the timer/event counter; TMR ([0DH]), TMRC ([0EH]). Two physical registers are mapped to TMR location; writing TMR makes the starting value be placed in the timer/event counter preload register and reading TMR retrieves the contents of the timer/event counter. The TMRC is a timer/event counter control register, which defines some options.

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 the $f_{INT}$ clock. 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 the $f_{INT}$ clock.

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 FFH. Once overflow occurs, the counter is reloaded from the timer/event counter preload register and generates the interrupt request flag (TF; bit 5 of INTC) at the same time.

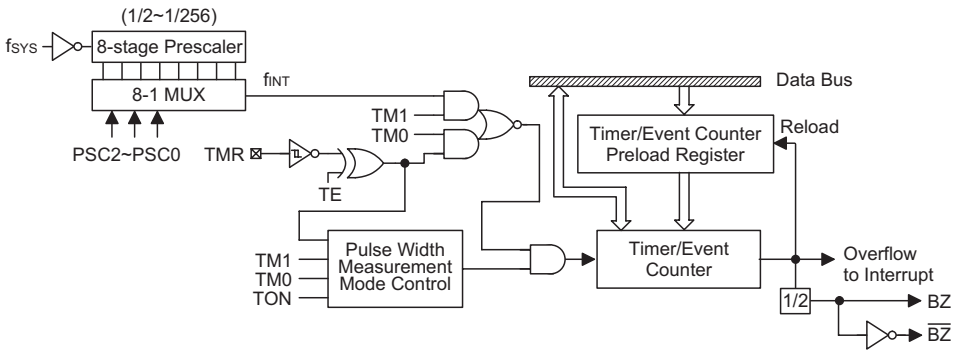
In the pulse width measurement mode with the TON and TE bits equal to one, once the TMR has received a transient from low to high (or high to low if the TE bits is "0")

it will start counting until the TMR returns to the original level and resets the TON. The measured result will remain in the timer/event counter even if the activated transient 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 transient pulse. Note that, in this operating mode, the timer/event counter starts counting not according to the logic level but according to the transient 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 completed. 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 disable the corresponding interrupt services.

In the case of timer/event counter OFF condition, writing data to the timer/event counter preload register will also reload that data to the timer/event counter. But if the timer/event counter is turned on, data written to it will only be kept in the timer/event counter preload register. The timer/event counter will still operate until overflow occurs. When the timer/event counter (reading TMR) is read, the clock will be blocked to avoid errors. As clock blocking may results in a counting error, this must be taken into consideration by the programmer.

| Bit No. | Label | Function |
|---|---|---|
| 0~2 | PSC0~PSC2 | Defines the prescaler stages, PSC2, PSC1, PSC0=<br>000: $f_{INT}=f_{SYS}/2$<br>001: $f_{INT}=f_{SYS}/4$<br>010: $f_{INT}=f_{SYS}/8$<br>011: $f_{INT}=f_{SYS}/16$<br>100: $f_{INT}=f_{SYS}/32$<br>101: $f_{INT}=f_{SYS}/64$<br>110: $f_{INT}=f_{SYS}/128$<br>111: $f_{INT}=f_{SYS}/256$ |
| 3 | TE | Defines the TMR active edge of the timer/event counter:<br>In Event Counter Mode (TM1,TM0)=(0,1):<br>1:count on falling edge;<br>0:count on rising edge<br>In Pulse Width measurement mode (TM1,TM0)=(1,1):<br>1: start counting on the rising edge, stop on the falling edge;<br>0: start counting on the falling edge, stop on the rising edge |
| 4 | TON | Enable or disable timer 0 counting (0=disabled; 1=enabled) |
| 5 | — | Unused bit, read as "0" |
| 6<br>7 | TM0<br>TM1 | Defines 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 (0EH) Register**




**Timer/Event Counter**

The bit0~bit2 of the TMRC can be used to define the pre-scaling stages of the internal clock sources of timer/event counter. The definitions are as shown. The overflow signal of the timer/event counter can be used to generate PFD signals for buzzer driving.

**Input/Output Ports**

There are 23 bidirectional input/output lines in the microcontroller, labeled from PA to PC and PG, which are mapped to the data memory of [12H], [14H], [16H] and [1EH], respectively. All of these I/O ports can be used for 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 1EH). 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, PGC) to control the input/output configuration. With this control register, CMOS output or Schmitt trigger input with or without pull-high resistor structures can be reconfigured dynamically under software control. To function as an input, the corresponding latch of the control register must write a ″1″. The input source also depends on the control register. If the control register bit is ″1″, the input will read the pad state. If the control register bit is ″0″, the contents of the latches will move to the internal bus. The latter is possible in the ″read-modify-write″ instruction.

For output function, CMOS is the only configuration. These control registers are mapped to locations 13H, 15H, 17H and 1FH.

After a chip reset, these input/output lines remain at high levels or floating state (depending on the pull-high options). Each bit of these input/output latches can be set or cleared by ″SET [m].i″ and ″CLR [m].i″ (m=12H, 14H, 16H or 1EH) instructions.

Some instructions first input data and then follow the output operations. For example, ″SET [m].i″, ″CLR [m].i″, ″CPL [m]″, ″CPLA [m]″ read the entire port states into the CPU, execute the defined operations (bit-operation), 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 highest 7-bit of port G are not physically implemented; on reading them a ″0″ is returned whereas writing then results in no operation. See Application note.

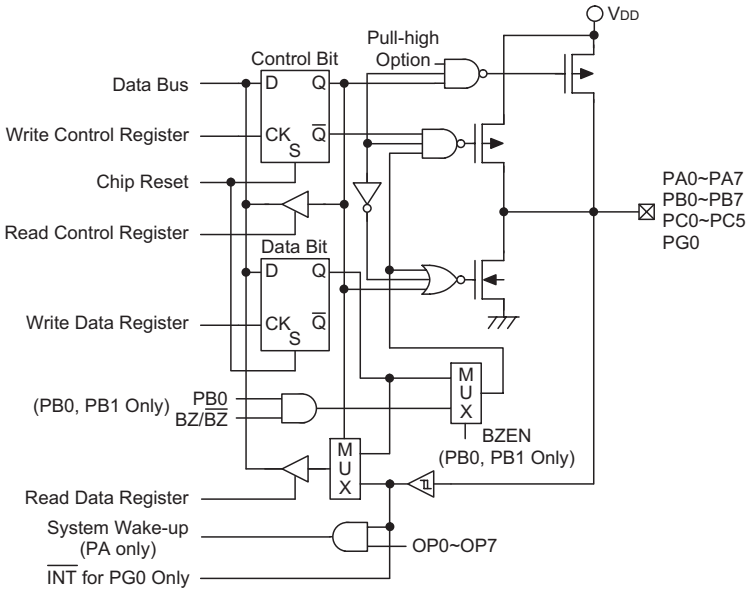
There is a pull-high option available for all I/O lines (bit option). Once the pull-high option of an I/O line is selected, the I/O line has a pull-high resistor. Otherwise, the pull-high resistor is absent. It should be noted that a non-pull-high I/O line operating in input mode will cause a floating state.

The PB0 and PB1 are pin-shared with BZ and $\overline{BZ}$ signals, respectively. If the BZ/$\overline{BZ}$ option is selected, the output signal in output mode of PB0/PB1 will be the PFD signal generated by timer/event counter 0 overflow signal. The input mode always remain in its original functions. Once the BZ/$\overline{BZ}$ option is selected, the buzzer output signals are controlled by the PB0 data register only.

The I/O functions of PB0/PB1 are shown below.

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| PB0 I/O | I | I | O | O | O | O | O | O | O | O |
| PB1 I/O | I | O | I | I | I | O | O | O | O | O |
| PB0 Mode | x | x | C | B | B | C | B | B | B | B |
| PB1 Mode | x | C | x | x | x | C | C | C | B | B |
| PB0 Data | x | x | D | 0 | 1 | $D_0$ | 0 | 1 | 0 | 1 |
| PB1 Data | x | D | x | x | x | $D_1$ | D | D | x | x |
| PB0 Pad Status | I | I | D | 0 | B | $D_0$ | 0 | B | 0 | B |
| PB1 Pad Status | I | D | I | I | I | $D_1$ | D | D | 0 | B |

Note: ″I″ input, ″O″ output, ″D, $D_0$, $D_1$″ data,
″B″ buzzer option, BZ or $\overline{BZ}$, ″x″ don't care
″C″ CMOS output




**Input/Output Ports**

The PG0 is pin-shared with $\overline{INT}$.

It is recommended that unused or not bonded out I/O lines should be set as output pins by software instruction to avoid consuming power under input floating state.

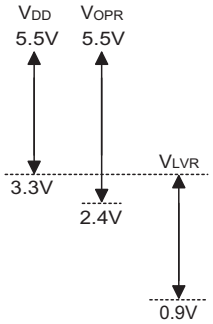
**Low Voltage Reset – LVR**

The HT48E30 provides a low voltage reset circuit in order to monitor the supply voltage of the device. If the supply voltage is within the range 0.9V~$V_{LVR}$, such as when changing a battery, the LVR will automatically reset the device internally.

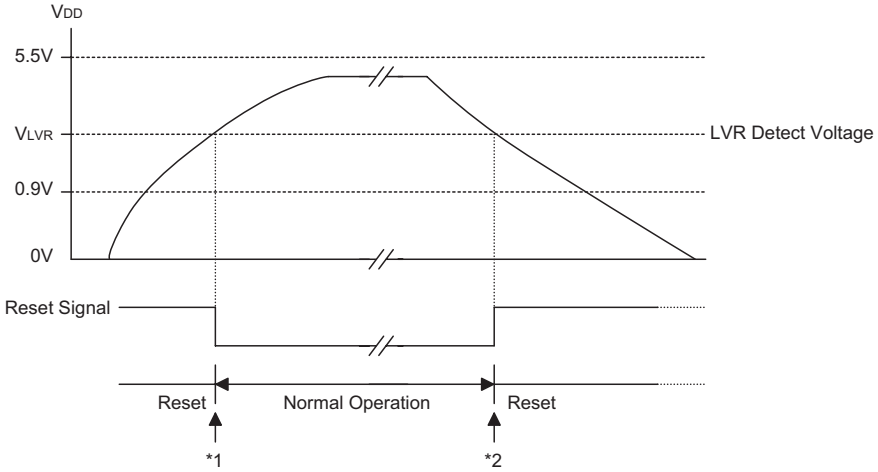
The LVR includes the following specifications:

- The low voltage (0.9V~VLVR) has to remain in its original state for longer than 1ms. If the low voltage state does not exceed 1ms, the LVR will ignore it and will not perform a reset function.
- The LVR uses an ″OR″ function with the external $\overline{RES}$ signal to perform a chip reset.

The relationship between $V_{DD}$ and $V_{LVR}$ is shown below.




Note: $V_{OPR}$ is the voltage range for proper chip operation at 4MHz system clock.




**Low Voltage Reset**

Note: *1: To make sure that the system oscillator has stabilized, the SST provides an extra delay of 1024 system clock pulses before starting the normal operation.

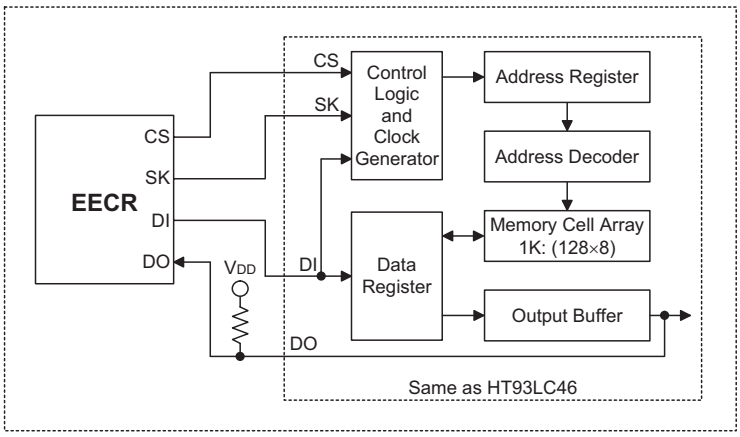
*2: Since low voltage state has to be maintained its original state for longer than 1ms, therefore after 1ms delay, the device enters the reset mode.

**EEPROM Data Memory**

The 128×8 bits EEPROM data memory is readable and writable during normal operation. It is indirectly addressed through the control register EECR ([40H] in Bank 1). The EECR can be read and written to only by indirect addressing mode using MP1.

| Bit No. | Label | Function |
|---|---|---|
| 0~3 | — | Unused bit, read as ″0″ |
| 4 | CS | EEPROM data memory select |
| 5 | SK | Serial clock input to EEPROM data memory |
| 6 | DI | Serial data input to EEPROM data memory |
| 7 | DO | Serial data output from EEPROM data memory |

**EECR (40H) Register**

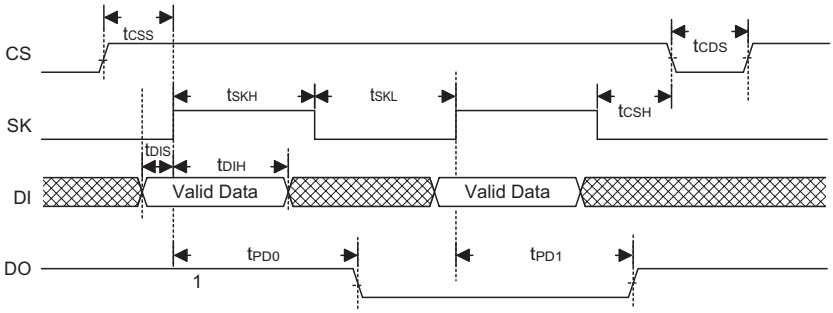



**EEPROM Data Memory Block Diagram**

The EEPROM data memory is accessed via a three-wire serial communication interface by writing to EECR. It is arranged into 128 words by 8 bits. The EEPROM data memory contains seven instructions: READ, ERASE, WRITE, EWEN, EWDS, ERAL and WRAL. These instructions are all made up of 10 bits data: 1 start bit, 2 op-code bits and 7 address bits.

By writing CS, SK and DI, these instructions can be given to the EEPROM. These serial instruction data presented at the DI will be written into the EEPROM data memory at the rising edge of SK. During the READ cycle, DO acts as the data output and during the WRITE or ERASE cycle, DO indicates the BUSY/READY status. When the DO is active for read data or as a BUSY/READY indicator the CS pin must be high; otherwise DO will be in a high state. For successful instructions, CS must be low once after the instruction is sent. After power-on, the device is by default in the EWDS state. And, an EWEN instruction must be performed before any ERASE or WRITE instruction can be executed.

The following are the functional descriptions and timing diagrams of all seven instructions.




**EECR A.C. Characteristics** Ta=25°C

| Symbol | Parameter | $V_{CC}$=5V±10% | | $V_{CC}$=2.2V±10% | | Unit |
|---|---|---|---|---|---|---|
| | | Min. | Max. | Min. | Max. | |
| $f_{SK}$ | Clock Frequency | 0 | 2 | 0 | 1 | MHz |
| $t_{SKH}$ | SK High Time | 250 | — | 500 | — | ns |
| $t_{SKL}$ | SK Low Time | 250 | — | 500 | — | ns |
| $t_{CSS}$ | CS Setup Time | 50 | — | 100 | — | ns |
| $t_{CSH}$ | CS Hold Time | 0 | — | 0 | — | ns |
| $t_{CDS}$ | CS Deselect Time | 250 | — | 250 | — | ns |
| $t_{DIS}$ | DI Setup Time | 100 | — | 200 | — | ns |
| $t_{DIH}$ | DI Hold Time | 100 | — | 200 | — | ns |
| $t_{PD1}$ | DO Delay to ″1″ | — | 250 | — | 500 | ns |
| $t_{PD0}$ | DO Delay to ″0″ | — | 250 | — | 500 | ns |
| $t_{SV}$ | Status Valid Time | — | 250 | — | 250 | ns |
| $t_{HZ}$ | DO Disable Time | 100 | — | 200 | — | ns |
| $t_{PR}$ | Write Cycle Time Per Word | — | 2 | — | 5 | ms |

### READ

The READ instruction will stream out data at a specified address on the DO. The data on DO changes during the low-to-high edge of SK. The 8 bits data stream is preceded by a logical ″0″ dummy bit. Irrespective of the condition of the EWEN or EWDS instruction, the READ command is always valid and independent of these two instructions. After the data word has been read the internal address will be automatically incremented by 1 allowing the next consecutive data word to be read out without entering further address data. The address will wrap around with CS High until CS returns to Low.

### EWEN/EWDS

The EWEN/EWDS instruction will enable or disable the programming capabilities. At both the power-on and power off state the device automatically entered the disable mode. Before a WRITE, ERASE, WRAL or ERAL instruction is given, the programming enable instruction EWEN must be issued, otherwise the ERASE/WRITE instruction is invalid. After the EWEN instruction is issued, the programming enable condition remains until power is turned off or an EWDS instruction is given. No data can be written into the EEPROM data memory in the programming disabled state. By so doing, the internal memory data can be protected.

### ERASE

The ERASE instruction erases data at the specified addresses in the programming enable mode. After the ERASE op-code and the specified address have been issued, the data erase is activated by the falling edge of CS. Since the internal auto-timing generator provides all timing signals for the internal erase, so the SK clock is not required. During the internal erase, we can verify the busy/ready status if CS is high. The DO will remain low but when the operation is over, the DO will return to high and further instructions can be executed.

### WRITE

The WRITE instruction writes data into the EEPROM data memory at the specified addresses in the programming enable mode. After the WRITE op-code and the specified address and data have been issued, the data writing is activated by the falling edge of CS. Since the internal auto-timing generator provides all timing signal for the internal writing, so the SK clock is not required. The auto-timing write cycle includes an automatic erase-before-write capability. So, it is not necessary to erase data before the WRITE instruction. During the internal writing, we can verify the busy/ready status if CS is high. The DO will remain low but when the operation is over, the DO will return to high and further instructions can be executed.

### ERAL

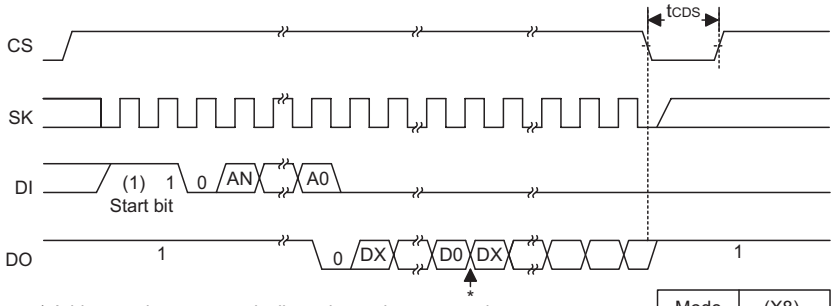
The ERAL instruction erases the entire 128×8 memory cells to a logical ″1″ state in the programming enable mode. After the erase-all instruction set has been issued, the data erase feature is activated by the falling edge of CS. Since the internal auto-timing generator provides all timing signal for the erase-all operation, so the SK clock is not required. During the internal erase-all operation, we can verify the busy/ready status if CS is high. The DO will remain low but when the operation is over, the DO will return to high and further instruction can be executed.

### WRAL

The WRAL instruction writes data into the entire 128×8 memory cells in the programming enable mode. After the write-all instruction set has been issued, the data writing is activated by the falling edge of CS. Since the internal auto-timing generator provides all timing signals for the write-all operation, so the SK clock is not required. During the internal write-all operation, we can verify the busy/ready status if CS is high. The DO will remain low but when the operation is over the DO will return to high and further instruction can be executed.

### EECR Control Timing Diagrams

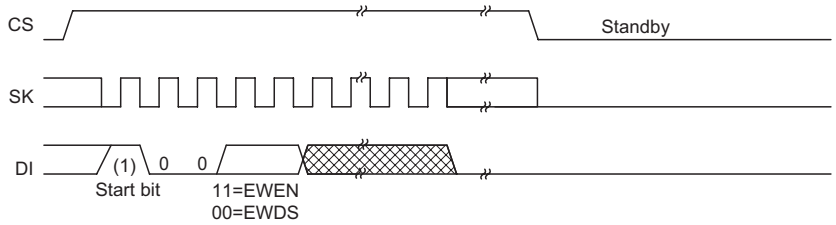
- READ



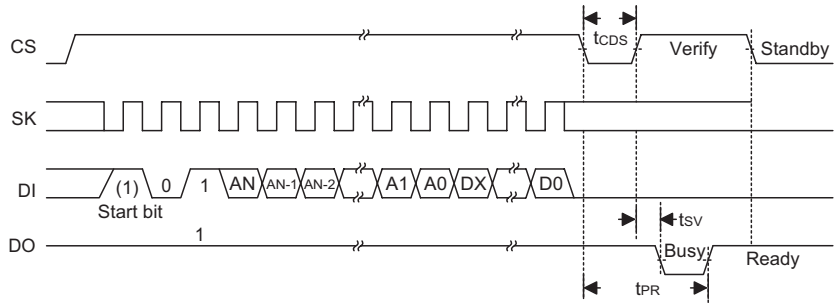

* Address pointer automatically cycles to the next word

| Mode | (X8) |
|---|---|
| AN | A6 |
| DX | D7 |

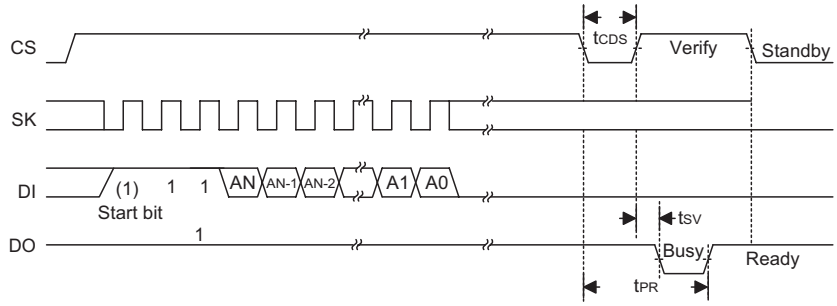
- EWEN/EWDS



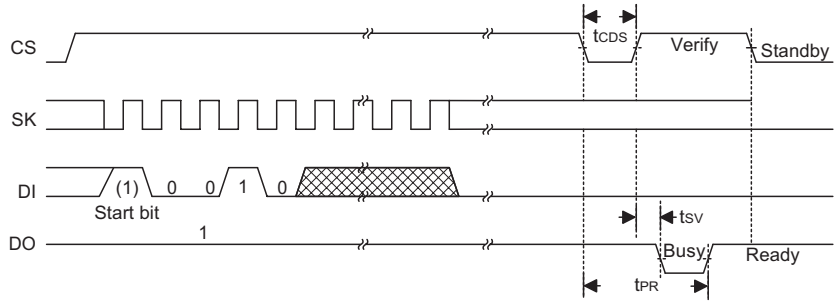

- WRITE



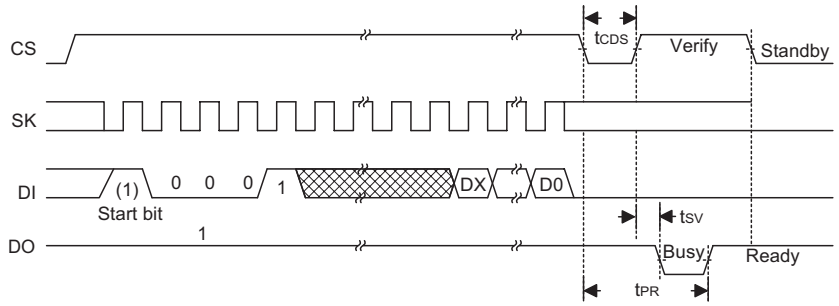

- ERASE




- ERAL



• WRAL




**EEPROM Data Memory Instruction Set Summary**

| Instruction | Comments | Start bit | Op Code | Address | Data |
|---|---|---|---|---|---|
| READ | Read data | 1 | 10 | A6~A0 | D7~D0 |
| ERASE | Erase data | 1 | 11 | A6~A0 | — |
| WRITE | Write data | 1 | 01 | A6~A0 | D7~D0 |
| EWEN | Erase/Write Enable | 1 | 00 | 11XXXXX | — |
| EWDS | Erase/Write Disable | 1 | 00 | 00XXXXX | — |
| ERAL | Erase All | 1 | 00 | 10XXXXX | — |
| WRAL | Write All | 1 | 00 | 01XXXXX | D7~D0 |

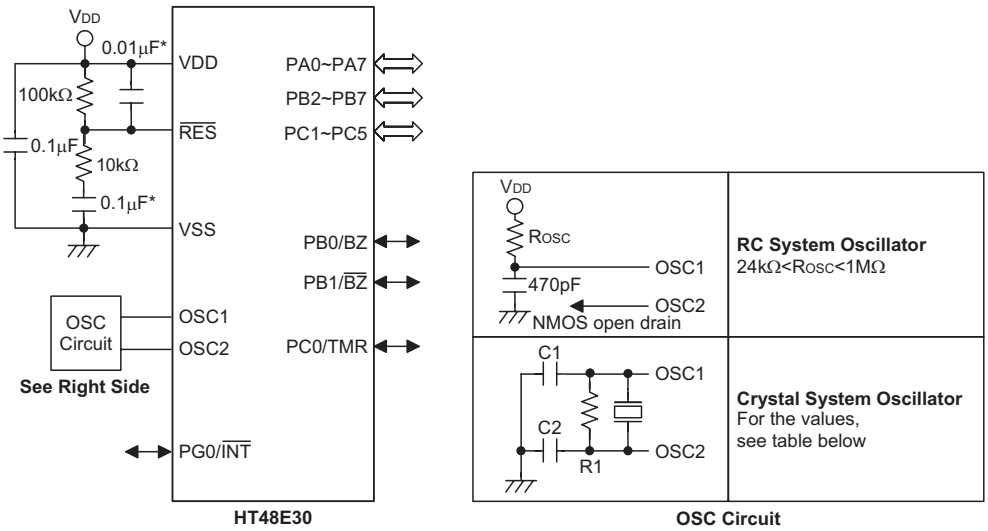
Note: ″X″ stands for ″don′t care″

**Options**

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

| No. | Options |
|---|---|
| 1 | WDT clock source: WDT oscillator or $f_{SYS}/4$ or disable |
| 2 | CLRWDT instructions: 1 or 2 instructions |
| 3 | Timer/event counter clock source: $f_{SYS}$ |
| 4 | PA bit wake-up enable or disable |
| 5 | PA CMOS or Schmitt input |
| 6 | PA, PB, PC, PG pull-high enable or disable (by port) |
| 7 | BZ/$\overline{BZ}$ enable or disable |
| 8 | LVR enable or disable |
| 9 | System oscillator: RC or crystal |

## Application Circuits




The following table shows the C1, C2 and R1 values corresponding to the different crystal values. (For reference only)

| Crystal or Resonator | C1, C2 | R1 |
|---|---|---|
| 4MHz Crystal | 0pF | 10kΩ |
| 4MHz Resonator | 10pF | 12kΩ |
| 3.58MHz Crystal | 0pF | 10kΩ |
| 3.58MHz Resonator | 25pF | 10kΩ |
| 2MHz Crystal & Resonator | 25pF | 10kΩ |
| 1MHz Crystal | 35pF | 27kΩ |
| 480kHz Resonator | 300pF | 9.1kΩ |
| 455kHz Resonator | 300pF | 10kΩ |
| 429kHz Resonator | 300pF | 10kΩ |
| The function of the resistor R1 is to ensure that the oscillator will switch off should low voltage conditions occur. Such a low voltage, as mentioned here, is one which is less than the lowest value of the MCU operating voltage. Note however that if the LVR is enabled then R1 can be removed. | | |

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 $\overline{RES}$ 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 ← Program Counter+1
Program Counter ← 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 Program Counter ← Program Counter+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 Program Counter ←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 Program Counter ← Program Counter+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 Program Counter ← 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 Program Counter ← 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 Program Counter ← 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 ([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.

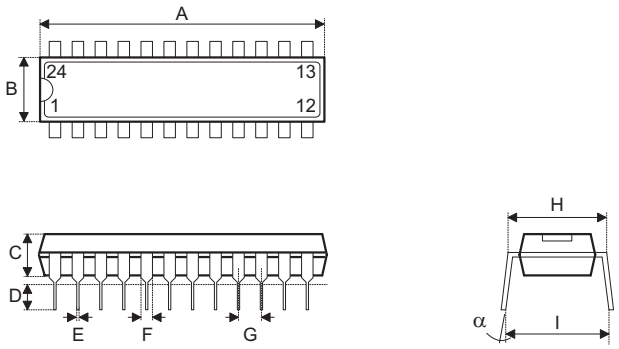
Operation ACC ← ACC ″XOR″ x

Affected flag(s)

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

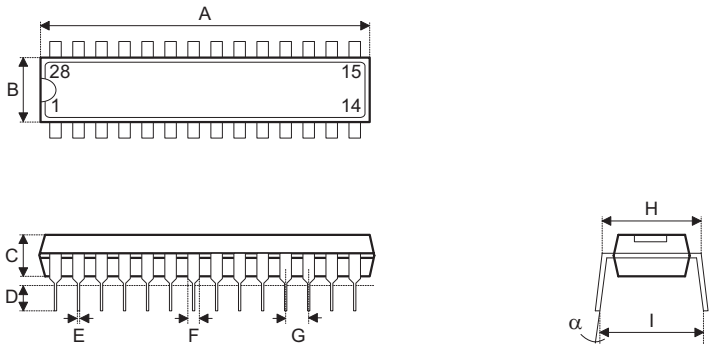
## Package Information

**24-pin SKDIP (300mil) Outline Dimensions**



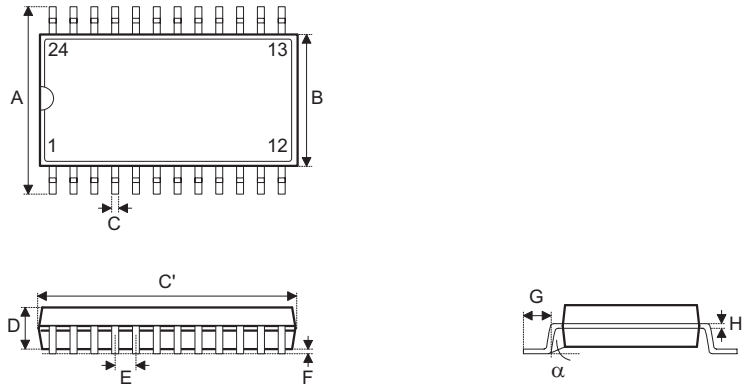

| Symbol | Dimensions in mil | | |
|---|---|---|---|
| | Min. | Nom. | Max. |
| A | 1235 | — | 1265 |
| B | 255 | — | 265 |
| C | 125 | — | 135 |
| D | 125 | — | 145 |
| E | 16 | — | 20 |
| F | 50 | — | 70 |
| G | — | 100 | — |
| H | 295 | — | 315 |
| I | 345 | — | 360 |
| α | 0° | — | 15° |

**28-pin SKDIP (300mil) Outline Dimensions**



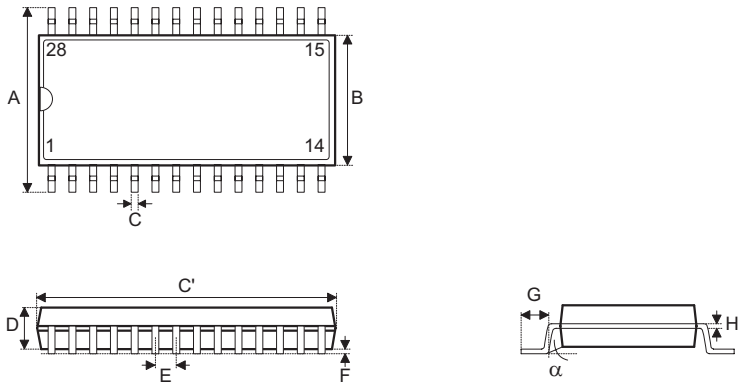

| Symbol | Dimensions in mil | | |
|---|---|---|---|
| | Min. | Nom. | Max. |
| A | 1375 | — | 1395 |
| B | 278 | — | 298 |
| C | 125 | — | 135 |
| D | 125 | — | 145 |
| E | 16 | — | 20 |
| F | 50 | — | 70 |
| G | — | 100 | — |
| H | 295 | — | 315 |
| I | 330 | — | 375 |
| α | 0° | — | 15° |

**24-pin SOP (300mil) Outline Dimensions**




| Symbol | Dimensions in mil | | |
|---|---|---|---|
| | Min. | Nom. | Max. |
| A | 394 | — | 419 |
| B | 290 | — | 300 |
| C | 14 | — | 20 |
| C′ | 590 | — | 614 |
| D | 92 | — | 104 |
| E | — | 50 | — |
| F | 4 | — | — |
| G | 32 | — | 38 |
| H | 4 | — | 12 |
| α | 0° | — | 10° |

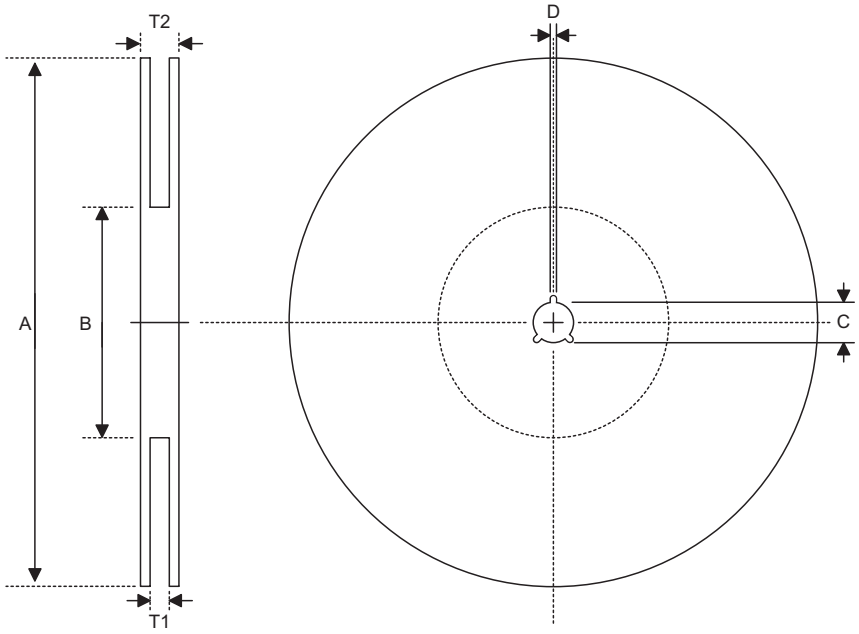
**28-pin SOP (300mil) Outline Dimensions**




| Symbol | Dimensions in mil | | |
|---|---|---|---|
| | Min. | Nom. | Max. |
| A | 394 | — | 419 |
| B | 290 | — | 300 |
| C | 14 | — | 20 |
| C′ | 697 | — | 713 |
| D | 92 | — | 104 |
| E | — | 50 | — |
| F | 4 | — | — |
| G | 32 | — | 38 |
| H | 4 | — | 12 |
| α | 0° | — | 10° |

## Product Tape and Reel Specifications

**Reel Dimensions**



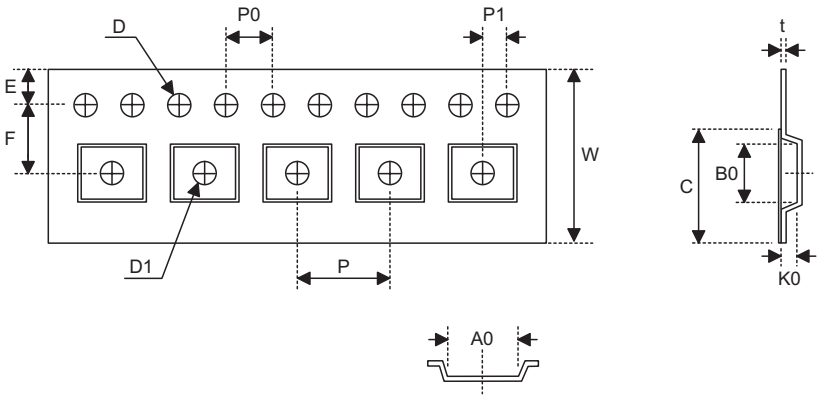

SOP 24W

| Symbol | Description | Dimensions in mm |
|---|---|---|
| A | Reel Outer Diameter | 330±1.0 |
| B | Reel Inner Diameter | 62±1.5 |
| C | Spindle Hole Diameter | 13.0+0.5<br>−0.2 |
| D | Key Slit Width | 2.0±0.5 |
| T1 | Space Between Flange | 24.8+0.3<br>−0.2 |
| T2 | Reel Thickness | 30.2±0.2 |

SOP 28W (300mil)

| Symbol | Description | Dimensions in mm |
|---|---|---|
| A | Reel Outer Diameter | 330±1.0 |
| B | Reel Inner Diameter | 62±1.5 |
| C | Spindle Hole Diameter | 13.0+0.5<br>−0.2 |
| D | Key Slit Width | 2.0±0.5 |
| T1 | Space Between Flange | 24.8+0.3<br>−0.2 |
| T2 | Reel Thickness | 30.2±0.2 |

**Carrier Tape Dimensions**




SOP 24W

| Symbol | Description | Dimensions in mm |
|---|---|---|
| W | Carrier Tape Width | 24.0±0.3 |
| P | Cavity Pitch | 12.0±0.1 |
| E | Perforation Position | 1.75±0.1 |
| F | Cavity to Perforation (Width Direction) | 11.5±0.1 |
| D | Perforation Diameter | 1.55+0.1 |
| D1 | Cavity Hole Diameter | 1.5+0.25 |
| P0 | Perforation Pitch | 4.0±0.1 |
| P1 | Cavity to Perforation (Length Direction) | 2.0±0.1 |
| A0 | Cavity Length | 10.9±0.1 |
| B0 | Cavity Width | 15.9±0.1 |
| K0 | Cavity Depth | 3.1±0.1 |
| t | Carrier Tape Thickness | 0.35±0.05 |
| C | Cover Tape Width | 21.3 |

SOP 28W (300mil)

| Symbol | Description | Dimensions in mm |
|---|---|---|
| W | Carrier Tape Width | 24.0±0.3 |
| P | Cavity Pitch | 12.0±0.1 |
| E | Perforation Position | 1.75±0.1 |
| F | Cavity to Perforation (Width Direction) | 11.5±0.1 |
| D | Perforation Diameter | 1.5+0.1 |
| D1 | Cavity Hole Diameter | 1.5+0.25 |
| P0 | Perforation Pitch | 4.0±0.1 |
| P1 | Cavity to Perforation (Length Direction) | 2.0±0.1 |
| A0 | Cavity Length | 10.85±0.1 |
| B0 | Cavity Width | 18.34±0.1 |
| K0 | Cavity Depth | 2.97±0.1 |
| t | Carrier Tape Thickness | 0.35±0.01 |
| C | Cover Tape Width | 21.3 |

**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. (Taipei 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 Inc. (Shanghai Sales Office)**
7th Floor, Building 2, No.889, Yi Shan Rd., Shanghai, China 200233
Tel: 021-6485-5560
Fax: 021-6485-0313
http://www.holtek.com.cn

**Holtek Semiconductor Inc. (Shenzhen Sales Office)**
43F, SEG Plaza, Shen Nan Zhong Road, Shenzhen, China 518031
Tel:0755-83465589
Fax:0755-83465590
ISDN : 0755-8346559

**Holtek Semiconductor Inc. (Beijing Sales Office)**
Suite 1721, Jinyu Tower, A129 West Xuan Wu Men Street, Xicheng District, Beijing, China 100031
Tel:010-66410030, 66417751, 66417752
Fax:010-66410125

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

Copyright © 2006 by HOLTEK SEMICONDUCTOR INC.

The information appearing in this Data Sheet is believed to be accurate at the time of publication. However, Holtek assumes no responsibility arising from the use of the specifications described. The applications mentioned herein are used solely for the purpose of illustration and Holtek makes no warranty or representation that such applications will be suitable without further modification, nor recommends the use of its products for application that may present a risk to human life due to malfunction or otherwise. Holtek's products are not authorized for use as critical components in life support devices or systems. Holtek reserves the right to alter its products without prior notification. For the most up-to-date information, please visit our web site at http://www.holtek.com.tw.