# HT49RA0/HT49CA0
# Remote Type 8-Bit MCU with LCD

## Technical Document

- Tools Information
- FAQs
- Application Note
  - HA0075E MCU Reset and Oscillator Circuits Application Note

## Features

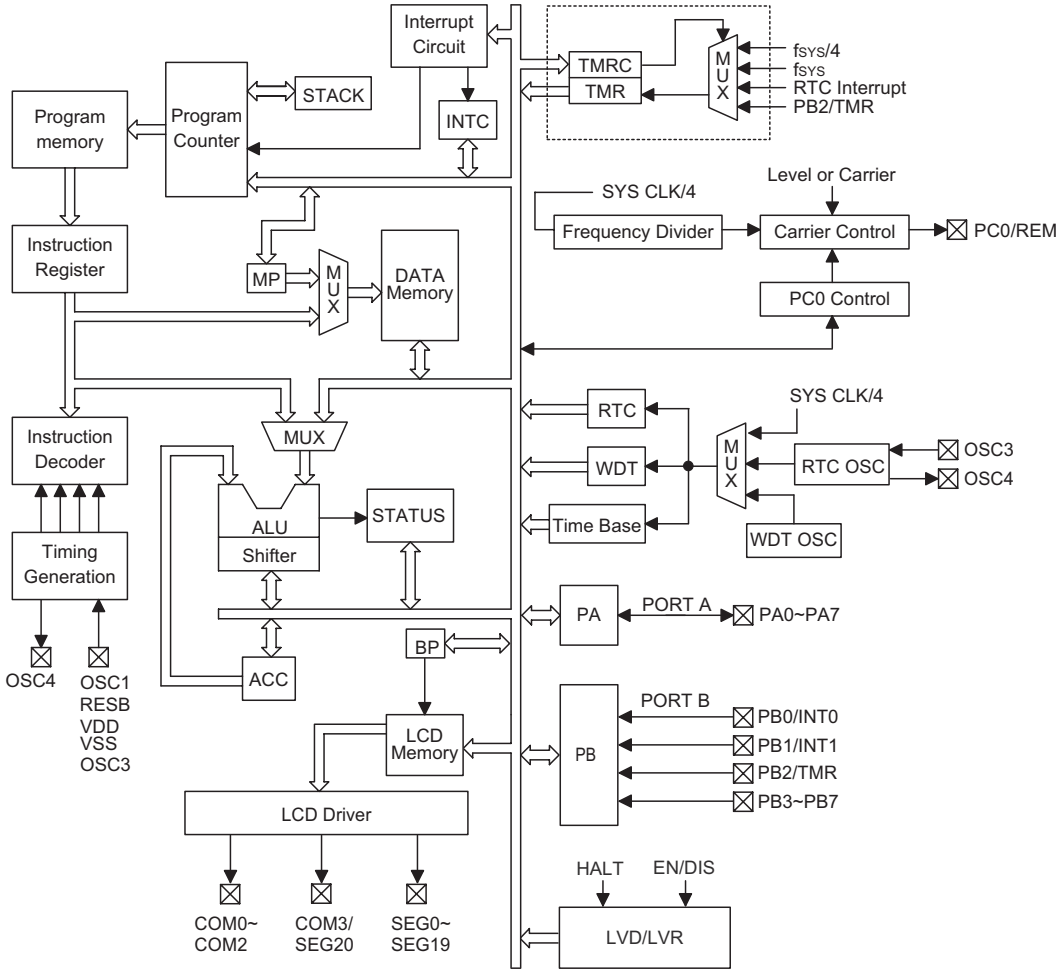
- Operating voltage: 2.0V~3.6V
- 8 bidirectional I/O lines and 8 input lines
- Two external interrupt input
- One 8-bit programmable timer/event counter
- LCD driver with 21×2, 21×3 or 20×4 segments
- 2K×14 program memory
- 96×8 data memory RAM
- Real Time Clock (RTC)
- 8-bit prescaler for RTC
- One carrier output (1/2 or 1/3 duty)
- Software LCD, RTC control
- On-chip RC and 32768Hz crystal oscillator
- Watchdog Timer
- HALT function and wake-up feature reduce power consumption
- 4-level subroutine nesting
- Bit manipulation instruction
- 14-bit table read instruction
- Up to 1μs instruction cycle with 4MHz system clock
- 63 powerful instructions
- All instructions in 1 or 2 machine cycles
- Low voltage reset/detector function
- 52-pin QFP package

## General Description

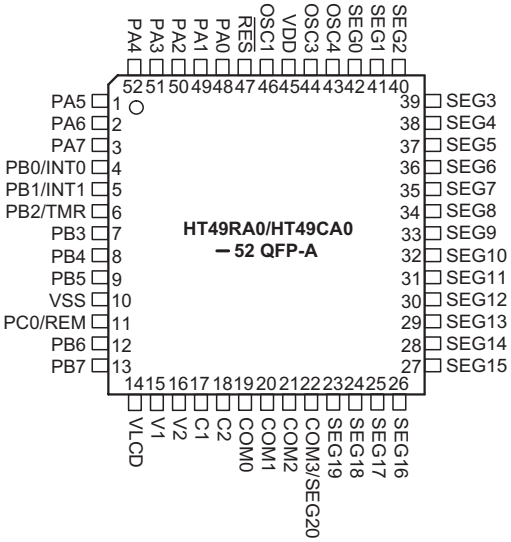
The HT49RA0/HT49CA0 are 8-bit high performance, RISC architecture microcontroller devices specifically designed for multiple I/O control product applications. The mask version HT49CA0 is fully pin and functionally compatible with the OTP version HT49RA0 device.

The advantages of low power consumption, I/O flexibility, timer functions, oscillator options, watchdog timer, HALT and wake-up functions, as well as low cost, enhance the versatility of this device to suit a wide range of application possibilities such as industrial control, consumer products, and particularly suitable for use in products such as infrared LCD remote controllers and various subsystem controllers.

## Block Diagram




## Pin Assignment



## Pin Description

| Pin Name | I/O | Options | Description |
|---|---|---|---|
| PA0~PA7 | I/O | — | Bidirectional 8-bit input/output port with pull-high resistors. Each bit can be determined as NMOS output or Schmitt trigger input by software instructions. |
| PB0/INT0<br>PB1/INT1<br>PB2/TMR<br>PB3~PB7 | I | Wake-up | 8-bit Schmitt trigger input port with pull-high resistors. Eash bit can be configured as a wake-up input by code option.<br>Pins PB0, PB1 and PB2 are pin-shared with INT0, INT1 and TMR respectively. |
| PC0/REM | I/O | Level or Carrier<br>Pull-high | Level or carrier output pin<br>PC0 can be set as CMOS output pin or carrier output pin by code option. |
| V1, V2, C1, C2 | I | — | Voltage pump |
| VLCD | | | LCD power supply<br>$V_{LCD}$ should be larger than $V_{DD}$ for correct operation i.e. $V_{LCD} \geq V_{DD}$. |
| COM0~COM2<br>COM3/SEG20 | O | 1/2, 1/3 or 1/4<br>Duty | COM3/SEG20 can be set as a segment or as a common output driver for LCD panel by options. COM0~COM2 are output for LCD panel plate. |
| SEG0~SEG11 | O | — | LCD driver outputs for LCD panel segments. |
| SEG12~SEG19 | O | SEG12~SEG19<br>CMOS output | LCD driver outputs for LCD panel segments.<br>SEG12~SEG19 can be optioned as logical outputs. |
| OSC1 | I | RC | OSC1 connect a resistor to GND for internal system clock. |
| OSC3<br>OSC4 | I<br>O | — | Real time clock oscillator. OSC3 and OSC4 are connected to a 32768Hz crystal oscillator for timing purpose. It can't be used as a system clock. No capacitor is built-in. If RTC is not selected as fs. OSC3,OSC4 should be left floating. |
| $\overline{RES}$ | I | — | Schmitt trigger reset input, active low. |
| VSS | — | — | Negative power supply, ground |
| VDD | — | — | Positive power supply |

## 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
$I_{OL}$ Total .....150mA
Total Power Dissipation .....500mW

Storage Temperature .....–50°C to 125°C
Operating Temperature.....–40°C to 85°C
$I_{OH}$ Total.....–100mA

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 | — | — | 2.0 | — | 3.6 | V |
| $I_{DD}$ | Operating Current<br>(RC OSC) | 3V | No load, $f_{SYS}$=4MHz | — | 0.7 | 1.5 | mA |
| $I_{STB1}$ | Standby Current<br>(*$f_S$=T1) | 3V | No load, system HALT,<br>LCD off at HALT | — | 0.1 | 1 | μA |
| $I_{STB2}$ | Standby Current<br>(*$f_S$=32.768kHz OSC) | 3V | No load, system HALT,<br>LCD On at HALT, C type | — | 2.5 | 5 | μA |

| Symbol | Parameter | Test Conditions | | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|---|---|
| | | $V_{DD}$ | Conditions | | | | |
| $I_{STB3}$ | Standby Current (*$f_S$=WDT RC OSC) | 3V | No load, system HALT LCD On at HALT, C type | — | 2 | 5 | μA |
| $V_{IL1}$ | Input Low Voltage for I/O Ports, TMR, INT0 and INT1 | 3V | — | 0 | — | $0.3V_{DD}$ | V |
| $V_{IH1}$ | Input High Voltage for I/O Ports, TMR, INT0 and INT1 | 3V | — | $0.7V_{DD}$ | — | $V_{DD}$ | V |
| $V_{IL2}$ | Input Low Voltage ($\overline{RES}$) | 3V | — | 0 | — | $0.4V_{DD}$ | V |
| $V_{IH2}$ | Input High Voltage ($\overline{RES}$) | 3V | — | $0.9V_{DD}$ | — | $V_{DD}$ | V |
| $I_{OL1}$ | I/O Port & REM Sink Current | 3V | $V_{OL}$=$0.1V_{DD}$ | 4 | 8 | — | mA |
| $I_{OH1}$ | I/O Port & REM Source Current | 3V | $V_{OH}$=$0.9V_{DD}$ | –5 | –7 | — | mA |
| $I_{OL2}$ | LCD Common and Segment Current | 3V | $V_{OL}$=$0.1V_{DD}$ | 210 | 420 | — | μA |
| $I_{OH2}$ | LCD Common and Segment Current | 3V | $V_{OH}$=$0.9V_{DD}$ | –80 | –160 | — | μA |
| $R_{PH}$ | Pull-high Resistance of I/O Ports | 3V | — | 100 | 150 | 200 | kΩ |
| $V_{LVR}$ | Low Voltage Reset Voltage | — | — | 2.0 | 2.1 | 2.2 | V |
| $V_{LVD}$ | Low Voltage Detector Voltage | — | — | 2.2 | 2.3 | 2.4 | V |
| $V_{POR}$ | VDD Start Voltage to Ensure Power-on Reset | — | — | — | — | 100 | mV |
| $R_{POR}$ | VDD Rise Rate to Ensure Power-on Reset | — | — | 0.035 | — | — | V/ms |

Note: "*" for the value of VA refer to the LCD driver section.

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

"*$f_S$" please refer to WDT clock option

## A.C. Characteristics

Ta=25°C

| Symbol | Parameter | Test Conditions | | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|---|---|
| | | $V_{DD}$ | Conditions | | | | |
| $f_{SYS}$ | System Clock | — | 2.0V~3.6V, 4MHz ± 3%, Temp.= 0°C ~ 50°C | — | 4000 | — | kHz |
| $f_{RTCOSC}$ | RTC Frequency | — | — | — | 32768 | — | Hz |
| $f_{TIMER}$ | Timer I/P Frequency (TMR) | 3V | — | 0 | — | 4000 | kHz |
| $t_{WDTOSC}$ | Watchdog Oscillator Period | 3V | — | 45 | 90 | 180 | μs |
| $t_{RES}$ | External Reset Low Pulse Width | — | — | 1 | — | — | μs |
| $t_{LVR}$ | Low Voltage Width to Reset | — | — | 0.25 | 1 | 2 | ms |
| $t_{SST}$ | System Start-up Timer Period | — | Wake-up from HALT | — | 1024 | — | *$t_{SYS}$ |
| $t_{INT}$ | Interrupt Pulse Width | — | — | 1 | — | — | μs |

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

## Functional Description

### Execution Flow

The system clock is derived from RC oscillator. It 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 one instruction cycle while decoding and execution takes the next instruction cycle. The pipelining scheme causes each instruction to effectively execute in a cycle. If an instruction changes the value of the program counter, two cycles are required to complete the instruction.

### Program Counter – PC

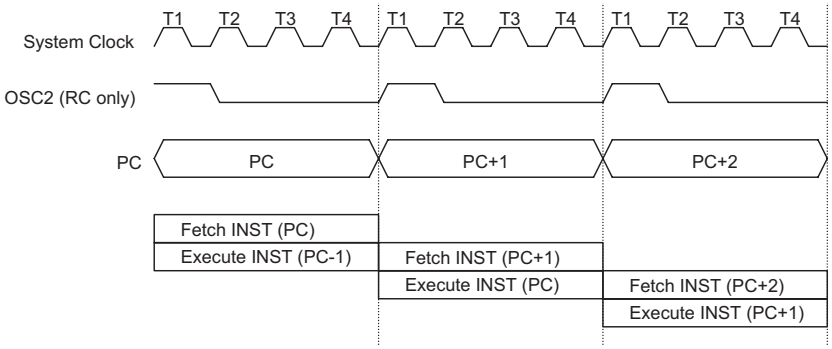
The program counter (PC) is of 11 bits wide and controls the sequence in which the instructions stored in the program ROM are executed. The contents of the PC can specify a maximum of 2048 addresses.

After accessing a program memory word to fetch an instruction code, the value of the PC is incremented by one. The PC then points to the memory word containing the next instruction code.

When executing a jump instruction, conditional skip execution, loading a PCL register, a subroutine call, an initial reset, an internal interrupt, an external interrupt, or returning from a subroutine, 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 a proper instruction; otherwise proceed with the next instruction.

The lower byte of the PC (PCL) is a readable and writeable register (06H). Moving data into the PCL performs a short jump. The destination is within 256 locations.




**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 | 0 | 1 | 0 | 0 |
| External Interrupt 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| Timer/Event Counter overflow | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| Time Base Interrupt | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| RTC Interrupt | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 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

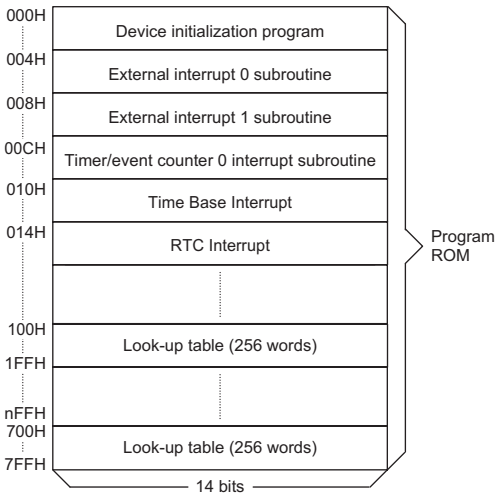
When a control transfer takes place, an additional dummy cycle is required.

**Program Memory – ROM**

The program memory (ROM) 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 which are addressed by the program counter and table pointer.

Certain locations in the ROM are reserved for special usage:

- Location 000H
  Location 000H is reserved for program initialization. After chip reset, the program always begins execution at this location.
- Location 004H
  Location 004H is reserved for the external interrupt service program. If the INT0 input pin is activated, and the interrupt is enabled, and the stack is not full, the program begins execution at location 004H.
- Location 008H
  Location 008H is reserved for the external interrupt service program also. If the INT1 input pin is activated, and the interrupt is enabled, and the stack is not full, the program begins execution at location 008H.
- Location 00CH
  Location 00CH 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 00CH.
- Location 010H
  Location 010H is reserved for the Time Base interrupt service program. If a Time Base interrupt occurs, and the interrupt is enabled, and the stack is not full, the program begins execution at location 010H.
- Location 014H
  Location 014H is reserved for the real time clock interrupt service program. If a real time clock interrupt occurs, and the interrupt is enabled, and the stack is not full, the program begins execution at location 014H.
- Table location
  Any location in the ROM can be used as a look-up table. The instructions "TABRDC [m]" (the current page, 1 page=256 words) and "TABRDL [m]" (the last page) transfer the contents of the lower-order byte to the specified data memory, and the contents of the higher-order byte to TBLH (Table Higher-order byte register) (08H). Only the destination of the lower-order byte in the table is well-defined; the other bits of the table word are all transferred to the lower portion of TBLH, and the remaining 2 bit is read as "0". The TBLH is read only, and the table pointer (TBLP) is a read/write register (07H), indicating the table location. Before accessing the table, the location should be placed in TBLP. All the table related instructions require 2 cycles to complete the operation. These areas may function as a normal ROM depending upon the user's requirements.




Note: n ranges from 0 to 6

**Program Memory**

| Instruction(s) | 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
@7~@0: Table pointer bits
P10~P8: Current program Counter bits

**Stack Register – STACK**

The stack register is a special part of the memory used to save the contents of the program counter. The stack is organized into 4 levels and is neither part of the data nor part of the program, and is neither readable nor writeable. Its activated level is indexed by a stack pointer (SP) and is neither readable nor writeable. At a commencement of a subroutine call or an interrupt acknowledgment, the contents of the program counter is pushed onto the stack. At the end of the subroutine or interrupt routine, signaled by a return instruction (RET or RETI), the contents of the program counter is restored to its previous value from the stack. After chip reset, the SP will point to the top of the stack.

If the stack is full and a non-masked interrupt takes place, the interrupt request flag is recorded but the acknowledgment is still inhibited. Once the SP is decremented (by RET or RETI), the interrupt is serviced. This feature prevents stack overflow, allowing the programmer to use the structure easily. Likewise, if the stack is full, and a "CALL" is subsequently executed, a stack overflow occurs and the first entry is lost (only the most recent 4 return addresses are stored).

**Data Memory – RAM**

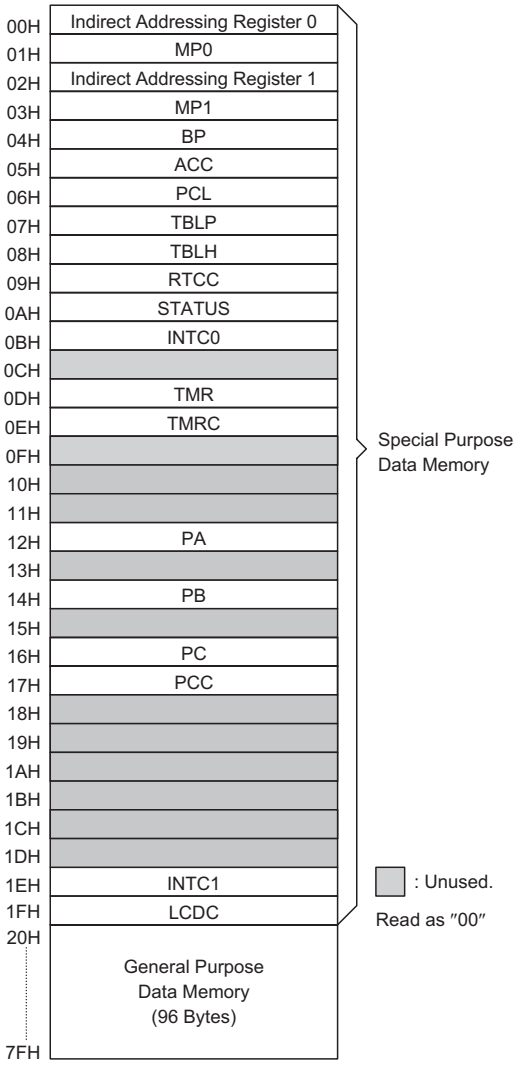
The data memory is divided into two functional group : special function registers (20×8) and general purpose data memory (96×8). Most of them are read/write, but some are read only.

The unused space before 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 command. The areas in the RAM can directly handle arithmetic, logic, increment, decrement, and rotate operations. Except some dedicated bits, each bit in the RAM can be set and reset by "SET [m].i" and "CLR [m].i". They are also indirectly accessible through the Memory pointer register 0 (MP0;01H) or the Memory pointer register 1 (MP1;03H).

**Indirect Addressing Register**

Location 00H and 02H are indirect addressing registers that are not physically implemented. Any read/write operation of [00H] and [02H] accesses 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, while MP1 can be applied to data memory and LCD display memory.




**RAM Mapping**

**Accumulator – ACC**

The accumulator (ACC) is related to the ALU operations. It is also mapped to location 05H of the RAM and is capable of operating with immediate data. The data movement between two data memory locations must pass through the ACC.

**Arithmetic and Logic Unit – ALU**

This circuit performs 8-bit arithmetic and logic operations and 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 etc.)

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

**Status Register – STATUS**

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

Except the TO and PDF flags, bits in the status register can be altered by instructions similar to other registers. Data written into the status register does not alter the TO or PDF flags. Operations related to the status register, however, may yield different results from those intended. The TO and PDF flags can only be changed by a Watchdog Timer overflow, chip power-up, or clearing the Watchdog Timer and executing the ″HALT″ instruction. The Z, OV, AC, and C flags reflect the status of the latest operations.

On entering the interrupt sequence or executing the subroutine call, the status register will not be automatically pushed onto the stack. If the contents of the status is important, and if the subroutine is likely to corrupt the status register, the programmer should take precautions and save it properly.

**Interrupts**

The devices provides two external interrupts, one internal timer/event counter interrupts, an internal time base interrupt, and an internal real time clock interrupt. The interrupt control register 0 (INTC0;0BH) and interrupt control register 1 (INTC1;1EH) both contain the interrupt control bits that are used to set the enable/disable status and interrupt request flags.

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

All these interrupts can support a wake-up function. As an interrupt is serviced, a control transfer occurs by pushing the contents of the program counter onto the stack followed by a branch to a subroutine at the specified location in the ROM. Only the contents of the program counter is pushed onto the stack. If the contents of the register or of the status register (STATUS) is 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 or low to high or both transition of INT0 or INT1, and the related interrupt request flag (EIF0;bit 4 of INTC0, EIF1;bit 5 of INTC0) is set as well. After the interrupt is enabled, the stack is not full, and the external interrupt is active, a subroutine call to location 04H or 08H occurs. The interrupt request flag (EIF0 or EIF1) and EMI bits are all cleared to disable other interrupts.

| Bit No. | Label | Function |
|---|---|---|
| 0 | C | C is set if the 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 the 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 the operation results in a carry into the highest-order bit but not a carry out of the highest-order bit, or vice versa; otherwise OV is cleared. |
| 4 | PDF | PDF is cleared by either a system power-up or executing the ″CLR WDT″ instruction. 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. TO is set by a WDT time-out. |
| 6, 7 | — | Unused bit, read as ″0″ |

**Status (0AH) Register**

The internal Timer/Event Counter interrupt is initialized by setting the Timer/Event Counter interrupt request flag (TF;bit 6 of INTC0), which is normally caused by a timer overflow. After the interrupt is enabled, and the stack is not full, and the TF bit is set, a subroutine call to location 0CH occurs. The related interrupt request flag (TF) is reset, and the EMI bit is cleared to disable further interrupts.

The time base interrupt is initialized by setting the time base interrupt request flag (TBF;bit 4 of INTC1), that is caused by a regular time base signal. After the interrupt is enabled, and the stack is not full, and the TBF bit is set, a subroutine call to location 10H occurs. The related interrupt request flag (TBF) is reset and the EMI bit is cleared to disable further interrupts.

The real time clock interrupt is initialized by setting the real time clock interrupt request flag (RTF; bit 5 of INTC1), that is caused by a regular real time clock signal. After the interrupt is enabled, and the stack is not full, and the RTF bit is set, a subroutine call to location 14H occurs. The related interrupt request flag (RTF) is reset and the EMI bit is cleared to disable further interrupts.

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

Interrupts occurring in the interval between the rising edges of two consecutive T2 pulses are serviced on the latter of the two T2 pulses if the corresponding interrupts are enabled. In the case of simultaneous requests, the priorities in the following table apply. These can be masked by resetting the EMI bit.

| Interrupt Source | Priority | Vector |
|---|---|---|
| External interrupt 0 | 1 | 04H |
| External interrupt 1 | 2 | 08H |
| Timer/Event Counter overflow | 3 | 0CH |
| Time base interrupt | 4 | 10H |
| Real time clock interrupt | 5 | 14H |

The EMI, EEI0, EEI1, ETI, ETBI and ERTI are all used to control the enable/disable status of interrupts. These bits prevent the requested interrupt from being serviced. Once the interrupt request flags (RTF, TBF, TF, EIF1, EIF0) are all set, they remain in the INTC1 or INTC0 respectively until the interrupts are serviced or cleared by a software instruction.

| Bit No. | Label | Function |
|---|---|---|
| 0 | EMI | Controls the master (global) interrupt (1=enabled; 0=disabled) |
| 1 | EEI0 | Controls the external interrupt 0 (1=enabled; 0=disabled) |
| 2 | EEI1 | Controls the external interrupt 1 (1=enabled; 0=disabled) |
| 3 | ETI | Controls the Timer/Event Counter interrupt (1=enabled; 0=disabled) |
| 4 | EIF0 | External interrupt 0 request flag (1=active; 0=inactive) |
| 5 | EIF1 | External interrupt 1 request flag (1=active; 0=inactive) |
| 6 | TF | Internal Timer/Event Counter request flag (1=active; 0=inactive) |
| 7 | — | Unused bit, read as ″0″ |

**INTC0 (0BH) Register**

| Bit No. | Label | Function |
|---|---|---|
| 0 | ETBI | Controls the time base interrupt (1=enabled; 0:disabled) |
| 1 | ERTI | Controls the real time clock interrupt (1=enabled; 0:disabled) |
| 2, 3 | — | Unused bit, read as ″0″ |
| 4 | TBF | Time base request flag (1=active; 0=inactive) |
| 5 | RTF | Real time clock request flag (1=active; 0=inactive) |
| 6, 7 | — | Unused bit, read as ″0″ |

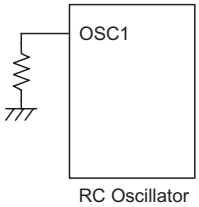
**INTC1 (1EH) Register**

It is recommended that a program not use the "CALL subroutine" within the interrupt subroutine. It's because interrupts often occur in an unpredictable manner or require to be serviced immediately in some applications. At this time, if only one stack is left, and enabling the interrupt is not well controlled, operation of the "call" in the interrupt subroutine may damage the original control sequence.

**Oscillator Configuration**

An external resistor between OSC1 and VSS in needed and the resistance is about 12kΩ. The RC oscillator provides a ±3% accuracy, the conditions are:

- $V_{DD}$=2.0V~3.6V
- Temp.= 0°C ~ 50°C
- $f_{SYS}$=4MHz




**System Oscillator**

**LCD/RTC OSC Control Register**

Two bit in (1FH) are for controlling LCD and RTC OSC.

| Bit No. | Label | Function |
|---|---|---|
| 0 | RTCEN | Controls RTC OSC enable (1=enabled; 0=disabled) |
| 1 | LCDEN | Controls LCD enable (1=enabled; 0=disabled) |
| 2~7 | — | Unused bit, read as "0" |

**LCDC (1FH) Register**

LCDEN and RTCEN may decide LCD and RTC On/Off condition on normal operation.

| $f_S$ Clock Source | LCD/WDT Control Bits | | | |
|---|---|---|---|---|
| | LCDEN, RTCEN=0, 0 | LCDEN, RTCEN=0, 1 | LCDEN, RTCEN=1, 0 | LCDEN, RTCEN=1, 1 |
| $f_{SYS}$/4 | LCD off, RTC off | LCD off, RTC off | LCD on, RTC off | LCD on, RTC off |
| WDT OSC | LCD off, RTC off | LCD off, RTC off | LCD on, RTC off | LCD on, RTC off |
| RTC OSC (WDT enable) | LCD off, RTC on | LCD off, RTC on | LCD on, RTC on | LCD on, RTC on |
| RTC OSC (WDT disable) | LCD off, RTC off | LCD off, RTC on | LCD on, RTC on | LCD on, RTC on |

**Watchdog Timer – WDT**

The WDT clock source is implemented by a dedicated RC oscillator (WDT oscillator) or an instruction clock (system clock/4) or a real time clock oscillator (RTC oscillator). The timer is designed to prevent a software malfunction or sequence from jumping to an unknown location with unpredictable results. The WDT can be disabled by option. But if the WDT is disabled, all executions related to the WDT lead to no operation.

The WDT time-out period is as $f_S/2^{16}$~$f_S/2^{15}$.

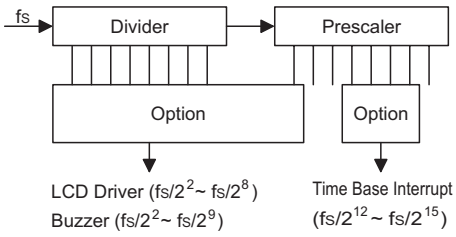
If the WDT clock source chooses the internal WDT oscillator, the time-out period may vary with temperature, VDD, and process variations. On the other hand, if the clock source selects the instruction clock and the ″HALT″ instruction is executed, WDT may stop counting and lose its protecting purpose, and the logic can only be restarted by an external logic.

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

The WDT overflow under normal operation initializes a ″chip reset″ and sets the status bit ″TO″. In the HALT mode, the overflow initializes a ″warm reset″, and only the program counter and stack pointer are reset to zero. To clear the contents of the WDT, there are three methods to be adopted, i.e., external reset (a low level to $\overline{RES}$), software instruction, and a ″HALT″ instruction. There are two types of software instructions; ″CLR WDT″ and the other set – ″CLR WDT1″ and ″CLR WDT2″. Of these two types of instruction, only one type of instruction can be active at a time depending on the options – ″CLR WDT″ times selection option. If the ″CLR WDT″ is selected (i.e., CLR WDT times equal one), any execution of the ″CLR WDT″ instruction clears the WDT. In the case that ″CLR WDT1″ and ″CLR WDT2″ are chosen (i.e., CLR WDT times equal two), these two instructions have to be executed to clear the WDT; otherwise, the WDT may reset the chip due to time-out.

**Time Base**

The time base offers a periodic time-out period to generate a regular internal interrupt. Its time-out period ranges from $f_S/2^{12}$ to $f_S/2^{15}$ selected by options. If time base time-out occurs, the related interrupt request flag (TBF; bit 4 of INTC1) is set. But if the interrupt is enabled, and the stack is not full, a subroutine call to location 10H occurs.



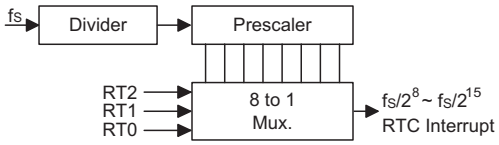

**Time Base**

**Real Time Clock – RTC**

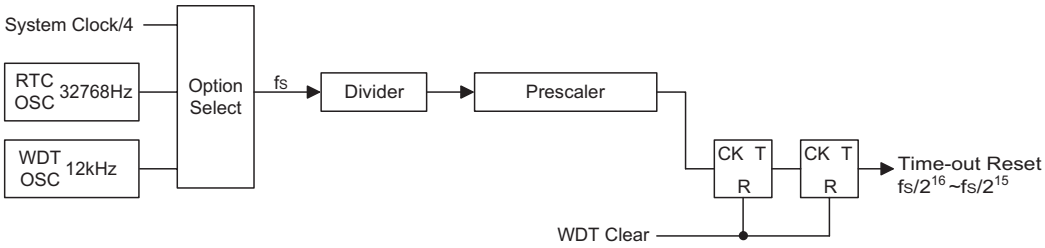
The real time clock (RTC) is operated in the same manner as the time base that is used to supply a regular internal interrupt. Its time-out period ranges from $f_S/2^8$ to $f_S/2^{15}$ by software programming . Writing data to RT2, RT1 and RT0 (bit2, 1, 0 of RTCC;09H) yields various time-out periods. If the RTC time-out occurs, the related interrupt request flag (RTF; bit 5 of INTC1) is set. But if the interrupt is enabled, and the stack is not full, a subroutine call to location 14H occurs. The real time clock time-out signal also can be applied to be a clock source of Timer/Event Counter for getting a longer time-out period.

| RT2 | RT1 | RT0 | RTC Clock Divided Factor |
|---|---|---|---|
| 0 | 0 | 0 | $2^{8*}$ |
| 0 | 0 | 1 | $2^{9*}$ |
| 0 | 1 | 0 | $2^{10*}$ |
| 0 | 1 | 1 | $2^{11*}$ |
| 1 | 0 | 0 | $2^{12}$ |
| 1 | 0 | 1 | $2^{13}$ |
| 1 | 1 | 0 | $2^{14}$ |
| 1 | 1 | 1 | $2^{15}$ |

Note: ″*″ not recommended to be used




**Real Time Clock**




**Watchdog Timer**

**Power Down Operation – HALT**

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

- The system oscillator turns off but the WDT OSC keeps running (if the WDT oscillator or the real time clock is selected).
- The contents of the on-chip RAM and of the registers remain unchanged.
- The WDT is cleared and start recounting (if the WDT clock source is from the WDT oscillator or the real time clock oscillator).
- All I/O ports maintain their original status.
- The PDF flag is set but the TO flag is cleared.

The system quits the HALT mode by an external reset, an interrupt, an external falling edge signal on port B, or a WDT overflow. An external reset causes device initialization, and the WDT overflow performs a "warm reset". After examining the TO and PDF flags, the reason for chip reset can be determined. The PDF flag is cleared by system power-up or by executing the "CLR WDT" instruction, and is set by executing the "HALT" instruction. On the other hand, the TO flag is set if WDT time-out occurs, and causes a wake-up that only resets the program counter and Stack Pointer, and leaves the others at their original state.

The port B wake-up and interrupt methods can be considered as a continuation of normal execution. Each bit in port B can be independently selected to wake-up the device by option. Awakening from an I/O port stimulus, the program resumes execution of the next instruction. On the other hand, awakening from an interrupt, two sequences may occur. If the related interrupt is disabled or the interrupt is enabled but the stack is full, the program resumes execution at the next instruction. But if the interrupt is enabled, and the stack is not full, the regular interrupt response takes place.

When an interrupt request flag is set before entering the "HALT" status, the system cannot be awaken using that interrupt.

If wake-up events occur, it takes 1024 $t_{SYS}$ (system clock period) to resume normal operation. In other words, a dummy period is inserted after the wake-up. If the wake-up results from an interrupt acknowledgment, the actual interrupt subroutine execution is delayed by more than one cycle. However, if the Wake-up results in the next instruction execution, the execution will be performed 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.

When at HALT state and $f_S=f_{SYS}/4$, LCD and RTC will be turned off no matter the bit value of (LCDEN, RTCEN).

**Reset**

There are three ways in which reset may occur.

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

The WDT time-out during HALT differs from other chip reset conditions, for it can perform a "warm reset" that resets only the program counter and stack pointer and leaves the other circuits at their original state. Some registers remain unaffected during any other reset conditions. Most registers are reset to the "initial condition" once the reset conditions are met. Examining the PDF and TO flags, the program can distinguish between different "chip resets".

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

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

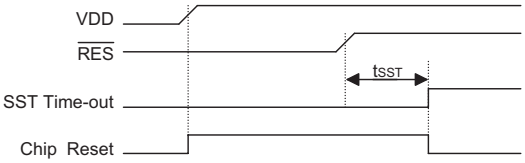
Note: "u" means unchanged

To guarantee that the system oscillator is started and stabilized, the SST (System Start-up Timer) provides an extra-delay of 1024 system clock pulses when the system awakes from the HALT state. Awaking from the HALT state, the SST delay is added.

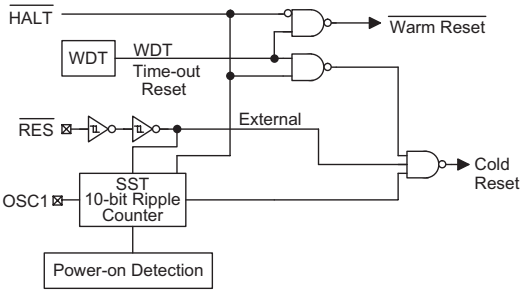
An extra SST delay is added during the power-up period and any wakeup from the HALT may enable only the SST delay.

The functional unit chip reset status is shown below.

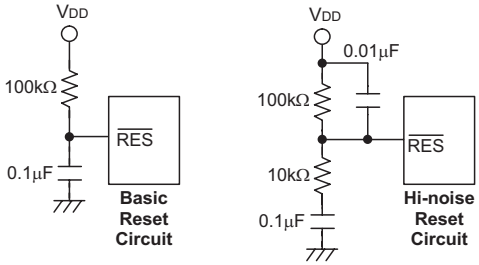
| | |
|---|---|
| Program Counter | 000H |
| Interrupt | Disabled |
| Prescaler, Divider | Cleared |
| WDT, RTC,<br>Time base | Cleared. After master reset,<br>WDT starts counting |
| Timer/Event Counter | Off |
| Input/output ports | Input mode |
| Stack Pointer | Points to the top of the stack |




**Reset Timing Chart**




**Reset Configuration**




**Reset Circuit**

Note: Most applications can use the Basic Reset Circuit as shown, however for applications with extensive noise, it is recommended to use the Hi-noise Reset Circuit.

The states of the registers are summarized below:

| Register | Reset (Power-on) | WDT Time-out (Normal Operation) | RES Reset (Normal Operation) | 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 |
| RTCC | --00 0111 | --00 0111 | --00 0111 | --00 0111 | --uu uuuu |
| STATUS | --00 xxxx | --1u uuuu | --uu uuuu | --01 uuuu | --11 uuuu |
| INTC0 | -000 0000 | -000 0000 | -000 0000 | -000 0000 | -uuu uuuu |
| TMR | xxxx xxxx | xxxx xxxx | xxxx xxxx | xxxx xxxx | uuuu uuuu |
| TMRC | 0000 1--- | 0000 1--- | 0000 1--- | 0000 1--- | uuuu u--- |
| PA | 1111 1111 | 1111 1111 | 1111 1111 | 1111 1111 | uuuu uuuu |
| PB | 1111 1111 | 1111 1111 | 1111 1111 | 1111 1111 | uuuu uuuu |
| PC | ---- ---1 | ---- ---1 | ---- ---1 | ---- ---1 | ---- ---u |
| PCC | ---- ---1 | ---- ---1 | ---- ---1 | ---- ---1 | ---- ---u |
| INTC1 | --00 --00 | --00 --00 | --00 --00 | --00 --00 | --uu --uu |
| LCDC | ---- --11 | ---- --11 | ---- --11 | ---- --11 | ---- --uu |

Note: "*" refers to warm reset
"u" means unchanged
"x" means unknown
"-" means unimplemented

**Timer/Event Counter**

One timer/event counters are implemented in the devices. It contains an 8-bit programmable count-up counter.

The timer/event counter clock source may come from the system clock or system clock/4 or RTC time-out signal or external source. System clock source or system clock/4 is selected by option.

Using 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.

There are two registers related to the timer/event counter, i.e., TMR (0DH) and TMRC (0EH). There are also two physical registers are mapped to TMR location; writing TMR places the starting value in the timer/event counter preload register, while reading it yields the contents of the timer/event counter. TMRC is timer/event counter control register used to define some options.

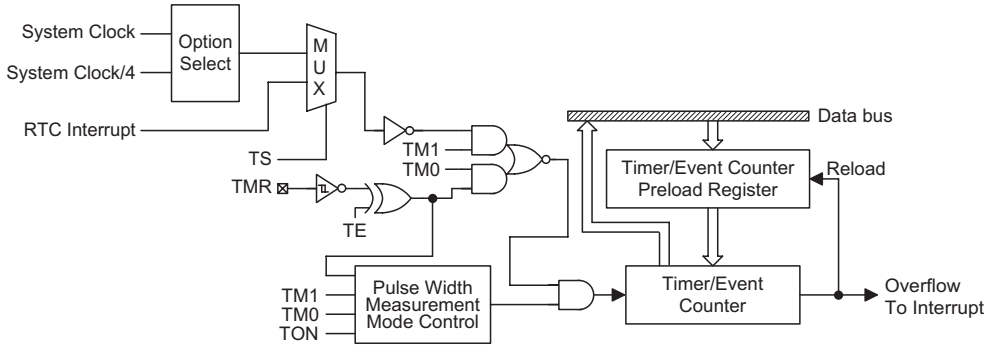
The TM0 and TM1 bits define the operation mode. The event count mode is used to count external events, which means that the clock source is from an external (TMR) pin. The timer mode functions as a normal timer with the clock source coming from the internal selected clock source. Finally, the pulse width measurement mode can be used to count the high or low level duration of the external signal (TMR), and the counting is based on the internal selected clock source.

In the event count or timer mode, the timer/event counter starts counting at the current contents in the timer/event counter and ends at FFH. Once an overflow occurs, the counter is reloaded from the timer/event counter preload register, and generates an interrupt request flag (TF;bit 6 of INTC0).

In the pulse width measurement mode with the values of the TON and TE bit equal to one, after the TMR has received a transient from low to high (or high to low if the TE bit is ″0″), it will start counting until the TMR returns to the original level and resets the TON. The measured result remains in the timer/event counter even if the activated transient occurs again. In other words, only one cycle measurement can be made until the TON is set. The cycle measurement will re-function as long as it receives further transient pulse. In this operation mode, the timer/event counter begins counting according not to the logic level but to the transient edges. In the case of counter overflows, the counter is reloaded from the timer/event counter preload register and issues an interrupt request, as in the other two modes, i.e., event and timer modes.

| Bit No. | Label | Function |
|---|---|---|
| 0~2 | — | Unused bit, read as ″0″ |
| 3 | TE | To define the TMR active edge of timer/event counter<br>(0=active on low to high; 1=active on high to low) |
| 4 | TON | To enable/disable timer counting<br>(0=disabled; 1=enabled) |
| 5 | TS | 2 to 1 multiplexer control inputs to select the timer/event counter clock source<br>(0=RTC outputs; 1= system clock or system clock/4) |
| 6<br>7 | TM0<br>TM1 | To define the operating mode (TM1, TM0)<br>01=Event count mode (External clock)<br>10=Timer mode (Internal clock)<br>11=Pulse Width measurement mode (External clock)<br>00=Unused |

**TMRC (0EH) Register**




**Timer/Event Counter**

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 is automatically cleared 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 disables the related interrupt service.

In the case of timer/event counter OFF condition, writing data to the timer/event counter preload register also reloads that data to the timer/event counter. But if the timer/event counter is turn on, data written to the timer/event counter is kept only in the timer/event counter preload register. The timer/event counter still continues its operation until an overflow occurs.

When the timer/event counter (reading TMR) is read, the clock is blocked to avoid errors. As this may results in a counting error, blocking of the clock should be taken into account by the programmer.

It is strongly recommended to load a desired value into the TMR register first, then turn on the related timer/event counter for proper operation. Because the initial value of TMR is unknown.

Due to the timer/event scheme, the programmer should pay special attention on the instruction to enable then disable the timer for the first time, whenever there is a need to use the timer/event function, to avoid unpredicatable result. After this procedure, the timer/event function can be operated normally.

**Carrier Generator**

The HT49RA0/HT49CA0 provides a carrier output which shares the pin with PC0. It can be selected to be a carrier output (REM) or level output pin (PC0) by code option. If the carrier output option is selected, setting PC0=″1″ to enable carrier output and setting PC0=″0″ to disable it at low level output.

The clock source of the carrier is implemented by instruction clock (system clock divided by 4) and processed by a frequency divider to yield various carry frequency.

$$\text{Carry Frequency} = \frac{\text{Clock Source}}{m \times 2^n}$$

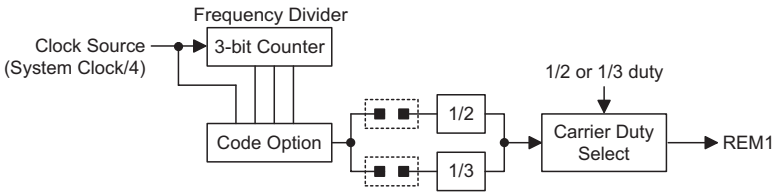
where m=2 or 3 and n=0~3, both are selected by code option. If m=2, the duty cycle of the carrier output is 1/2 duty. If m=3, the duty cycle of the carrier output can be 1/2 duty or 1/3 duty also determined by code option (with the exception of n=0).

Detailed selection of the carrier duty is shown below:

| $m \times 2^n$ | Duty Cycle |
|---|---|
| 2, 4, 8, 16 | 1/2 |
| 3 | 1/3 |
| 6, 12, 24 | 1/2 or 1/3 |

The following table shows examples of carrier frequency selection.

| $f_{SYS}$ | $f_{CARRIER}$ | Duty | $m \times 2^n$ |
|---|---|---|---|
| 455kHz | 37.92kHz | 1/3 only | 3 |
| | 56.9kHz | 1/2 only | 2 |




**Carrier/Level Output**

**Input/Output Ports**

There are an 8-bit bidirectional input/output port, a 8-bit input port and one-bit input/output port in the HT49RA0/HT49CA0, labeled as PA, PB and PC which are mapped to [12H], [14H], [16H] of the RAM respectively. Each bit of PA can be selected as NMOS output or Schmitt trigger with pull-high resistor by software instruction. PB0~PB7 can only be used for input operation (Schmitt trigger with pull-high resistors). PC is only one-bit input/output port shares the pin with carrier output.

When PA, PB and PC for the input operation, these ports are non-latched, that is, the inputs should be ready at the T2 rising edge of the instruction ″MOV A, [m]″ (m=12H or 14H). For PA and PC output operation, all data are latched and remain unchanged until the output latch is rewritten.

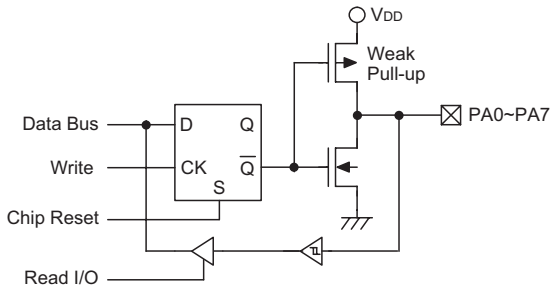
When the PA is used for input operation, it should be noted that before reading data from pads, a ″1″ should be written to the related bits to disable the NMOS device. That is, the instruction ″SET [m].i″ (i=0~7 for PA) is executed first to disable related NMOS device, and then ″MOV A, [m]″ to get stable data.

After chip reset, PA, PB and PC remain at a high level input line.

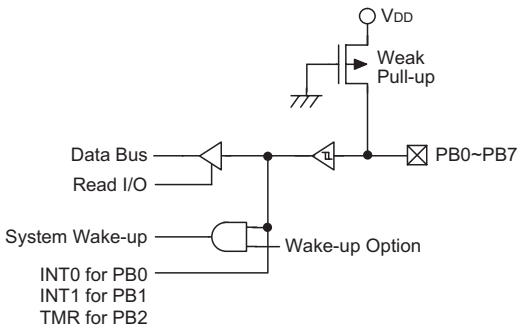
Each bit of PA and PC output latches can be set or cleared by the ″SET [m].i″ and ″CLR [m].i″ (m=12H or 16H) instructions respectively.

Some instructions first input data and then follow the output operations. For example, ″SET [m].i″, ″CLR [m]″, ″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 to the accumulator.

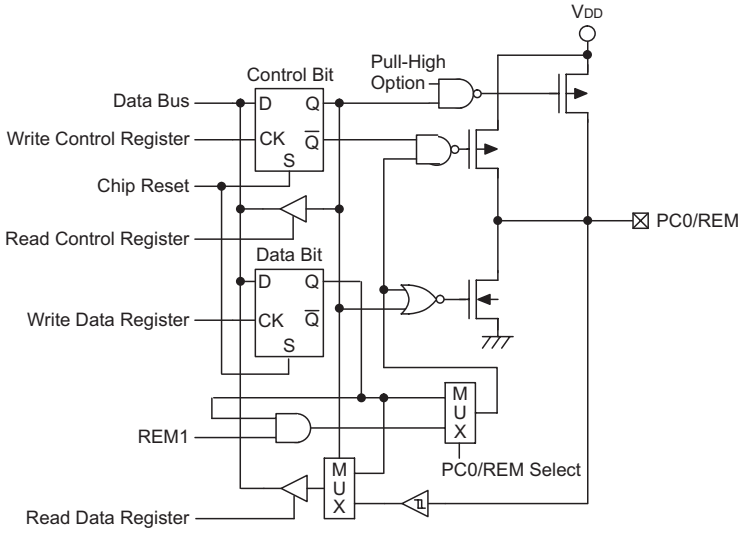
Each line of PB has a wake-up capability to the device by code option. The highest seven bits of PC are not physically implemented, on reading them a ″0″ is returned and writing results in a no-operation.




**PA Input/Output Ports**




**PB Input Ports**

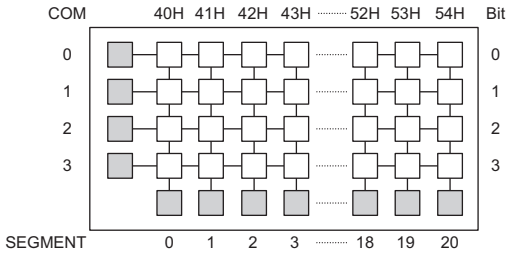



**PC Input/Output Ports**

### LCD Display Memory

The devices provides an area of embedded data memory for LCD display. This area is located from 40H to 54H of the RAM at Bank 1. Bank pointer (BP; located at 04H of the RAM) is the switch between the RAM and the LCD display memory. When the BP is set as ″1″, any data written into 40H~54H will effect the LCD display. When the BP is cleared to ″0″, any data written into 40H~54H means to access the general purpose data memory.

The LCD display memory can be read and written to only by indirect addressing mode using MP1. When data is written into the display data area, it is automatically read by the LCD driver which then generates the corresponding LCD driving signals. To turn the display on or off, a ″1″ or a ″0″ is written to the corresponding bit of the display memory, respectively. The figure illustrates the mapping between the display memory and LCD pattern for the devices.




**Display Memory**

### LCD Driver Output

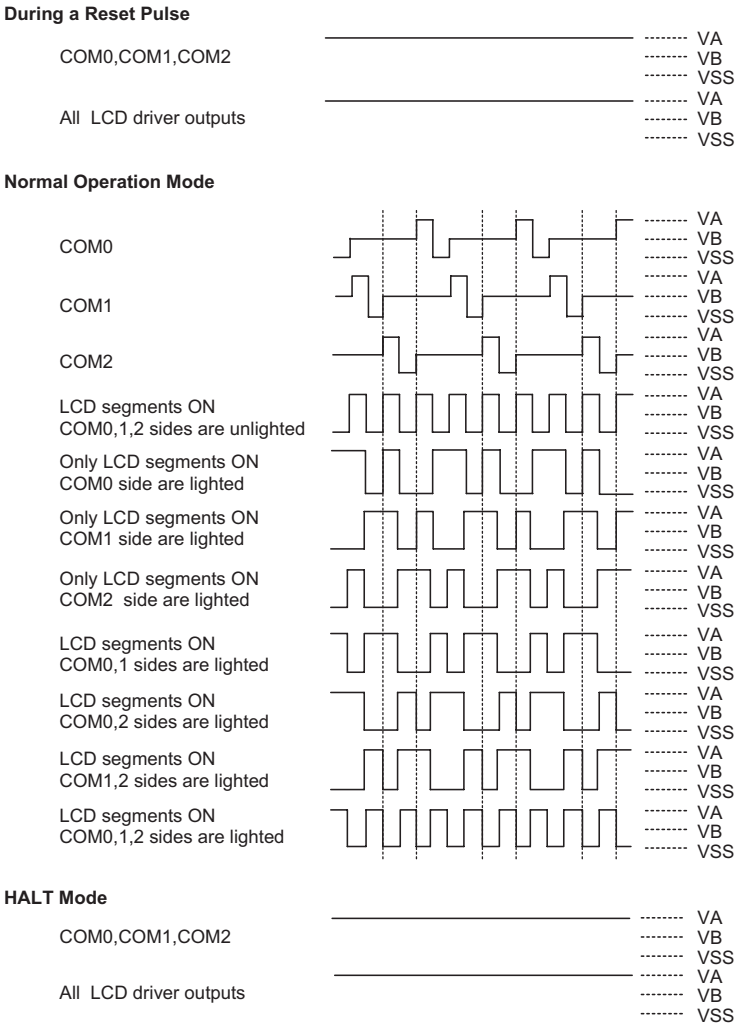
The output number of the LCD driver device can be 21×2, 21×3 or 20×4 by option. The bias type LCD driver can be ″C″ type only. A capacitor mounted between C1 and C2 pins is needed. If 1/2 bias is selected, a capacitor mounted between V2 pin and ground is required. If 1/3 bias is selected, two capacitors are needed for V1 and V2 pins. All the capacitance of capacitors used for LCD bias generator is suggested to use the 0.1μF. The relationships between LCD bias types, bias levels and V1 and V2 connection are listed in the table.

| Bias Level | Bias Type | C1/C2 | V1 | V2 | VLCD |
|---|---|---|---|---|---|
| 1/2 | C | 0.1μF | x | 0.1μF | VLCD |
| 1/3 | C | 0.1μF | 0.1μF | 0.1μF | VLCD |

There is a clock source needed for LCD driver. The LCD clock source comes from the general purpose prescaler and is decided by code options. The LCD clock frequency should be selected as near to 4kHz either from 32768 RTC or WDT or $f_{SYS}/4$ clock source.

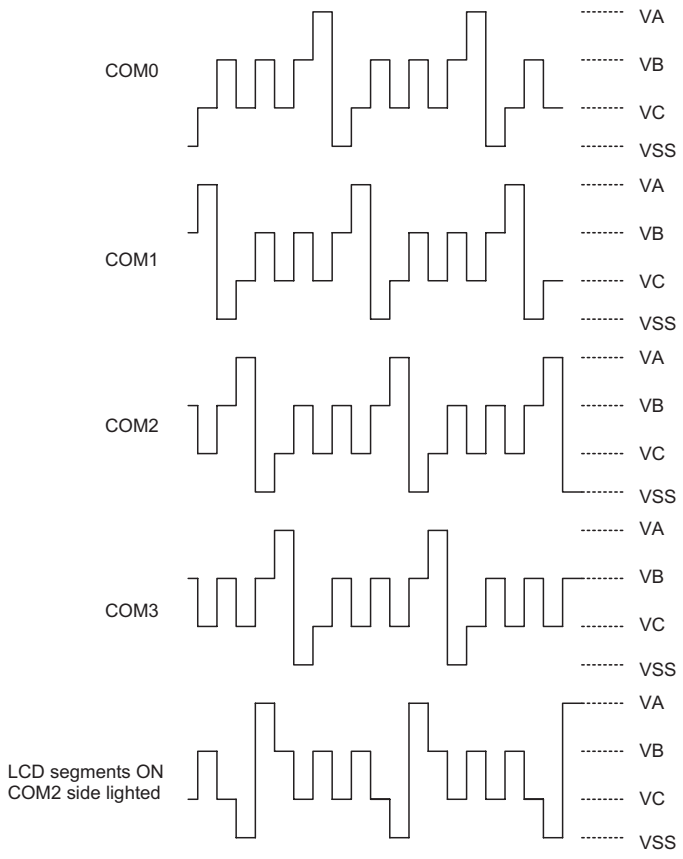
The options of LCD clock frequency are listed in the following table.

| $f_S$ Clock Source | LCD Clock Selection |
|---|---|
| WDT Oscillator | $WDT/2^2$ |
| RTC Oscillator | $RTC/2^3$ |
| $f_{SYS}/4$ | $f_{SYS}/2^4$~$f_{SYS}/2^{10}$ |




Note: "∗" Omit the COM2 signal, if the 1/2 duty LCD is used.
VA=VLCD, VB=VLCDx1/2

**LCD Driver Output (1/3 Duty, 1/2 Bias, C Type)**




Note: VA=VLCDx1.5, VB=VLCD, VC=VLCDx1/2

**LCD Driver Output (1/4 Duty, 1/3 Bias, C Type)**

**Low Voltage Reset/Detector Functions**

There is a low voltage detector (LVD) and a low voltage reset circuit (LVR) implemented in the microcontroller. These two functions can be enabled/disabled by options. Once the LVD options is enabled, the user can use the RTCC.3 to enable/disable (1/0) the LVD circuit and read the LVD detector status (0/1) from RTCC.5; otherwise, the LVD function is disabled.

The RTCC register definitions are listed below.

| Bit No. | Label | Function |
|---|---|---|
| 0~2 | RT0~RT2 | 8 to 1 multiplexer control inputs to select the real clock prescaler output |
| 3 | LVDC* | LVD enable/disable (1/0) |
| 4 | QOSC | 32768Hz OSC quick start-up oscillating<br>0/1: quickly/slowly start |
| 5 | LVDO | LVD detection output (1/0)<br>1: low voltage detected, read only |
| 6, 7 | — | Unused bit, read as ″0″ |

**RTCC (09H) Register**

Once the LVD function is enabled the reference generator should be enabled; otherwise the reference generator is controlled by LVR code option. The relationship among LVR and LVD options and LVDC are as shown.

LVDC can read/write, LVDO is read only.

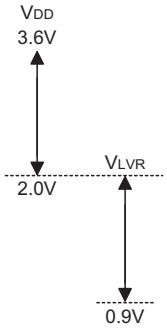
| LVD | LVR | LVDC | VREF Generator | LVR Comparator | LVD Comparator |
|---|---|---|---|---|---|
| Enable | Enable | On | Enable | Enable | Enable |
| Enable | Enable | Off | Enable | Enable | Disable |
| Enable | Disable | On | Enable | Disable | Enable |
| Enable | Disable | Off | Disable | Disable | Disable |
| Disable | Enable | X | Enable | Enable | Disable |
| Disable | Disable | X | Disable | Disable | Disable |

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

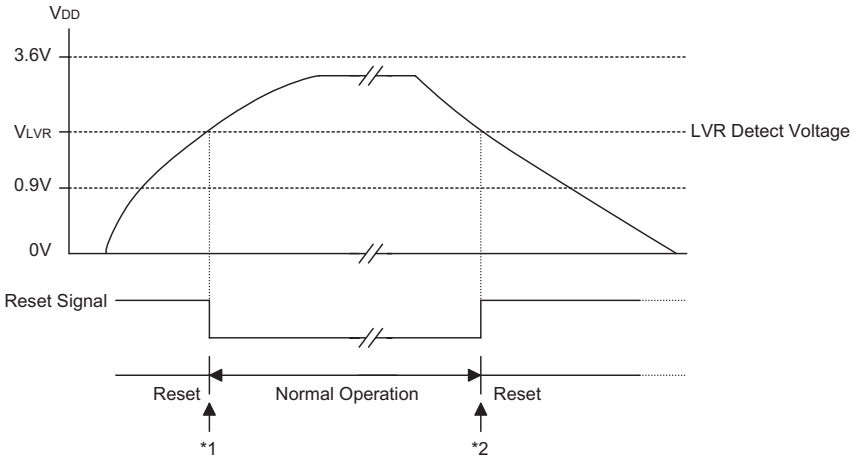
The LVR includes the following specifications:

- The low voltage (0.9V~$V_{LVR}$) state must exists for more than 1ms, while the other circuits remain in their original state. If the low voltage state does not exceed 1ms, the LVR will ignore it and do not perform a reset function.
- The LVR uses the ″OR″ function with the power-on reset 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 entering the normal operation.

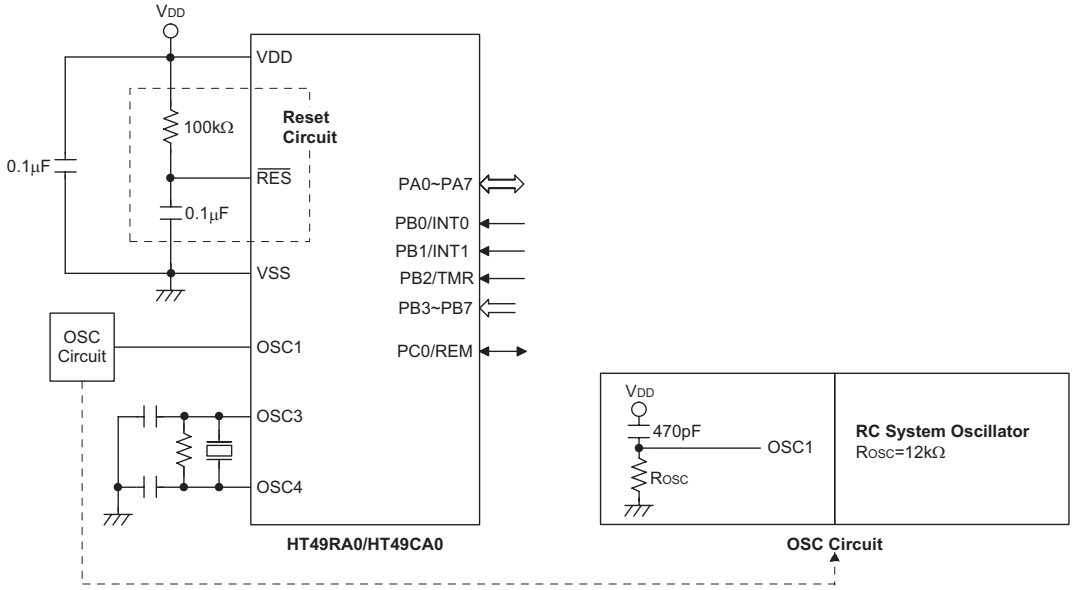
″*2″ Low voltage state has to be maintained for over 1ms, then after a 1ms delay the device enters the reset mode.

**Options**

The following shows the options in the devices. All these options should be defined in order to ensure proper system functioning.

| Item | Options |
|---|---|
| **I/O Options** | |
| 1 | PB0~PB7: wake-up enable or disable (bit option) |
| 2 | PC0: CMOS output or carrier output (bit option) |
| 3 | PC0: Pull-high enable or disable (bit option) |
| **LCD Options** | |
| 4 | LCD clock: $f_S/2^2$, $f_S/2^3$, $f_S/2^4$, $f_S/2^5$, $f_S/2^6$, $f_S/2^7$, $f_S/2^8$ |
| 5 | LCD duty: 1/2, 1/3, 1/4 |
| 6 | LCD bias: 1/2, 1/3 |
| 7 | LCD segment 12~15 output or CMOS output(Nibble Option) |
| 8 | LCD segment 16~19 output or CMOS output(Nibble Option) |
| **Interrupt Options** | |
| 9 | INT0 function: enable or disable |
| 10 | Triggering edge: rising, falling or both |
| 11 | INT1 function: enable or disable |
| 12 | Triggering edge: rising, falling or both |
| **Oscillator Options** | |
| 13 | $f_S$ internal clock source: RTC oscillator, WDT oscillator or $f_{SYS}/4$ |
| **Timer Options** | |
| 14 | Timer/Event Counter clock source: $f_{SYS}$ or $f_{SYS}/4$ |
| **Time Base Options** | |
| 15 | Time Base division ratio: $f_S/2^{12}$, $f_S/2^{13}$, $f_S/2^{14}$, $f_S/2^{15}$ |
| **Watchdog Options** | |
| 16 | WDT enable or disable |
| 17 | CLRWDT instructions: 1 or 2 instructions |
| **LVD/LVR Options** | |
| 18 | Low Voltage Detect: enable or disable |
| 19 | LVR function: enable or disable |
| **Carrier Options** | |
| 20 | Carrier duty: 1/2 duty or 1/3 duty |
| 21 | Carrier frequency: $f_{SYS}/8$, $f_{SYS}/16$, $f_{SYS}/32$, $f_{SYS}/64$ for 1/2 duty cycle |
| 22 | Carrier frequency: $f_{SYS}/12$, 1/3 duty cycle |
| 23 | Carrier frequency: $f_{SYS}/24$, $f_{SYS}/48$, $f_{SYS}/96$ for 1/2 duty or 1/3 duty cycle |

## Application Circuits

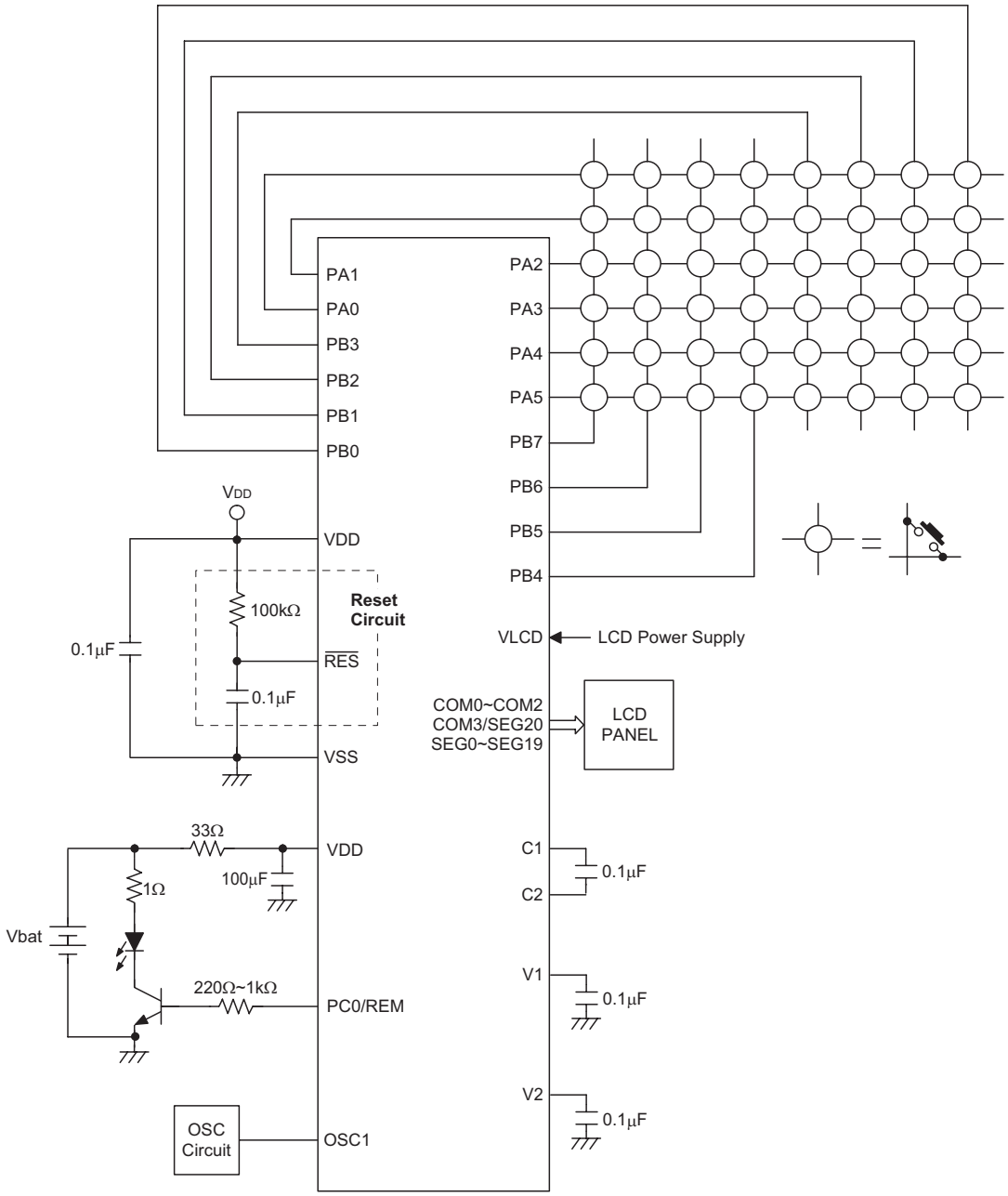



Note: 1. Reset circuit

The reset circuit resistance and capacitance values should be chosen to ensure that VDD is stable and remains within its operating voltage range before the $\overline{RES}$ pin reaches a high level. Ensure that the length of the wiring connected to the $\overline{RES}$ pin is kept as short as possible, to avoid noise interference.

2. For applications where noise may interfere with the reset circuit and for details on the oscillator external components, refer to Application Note HA0075E for more information.

**Example**


PA1
PA0
PB3
PB2
PB1
PB0
PA2
PA3
PA4
PA5
PB7
PB6
PB5
PB4
VDD
VDD
100kΩ
Reset
Circuit
0.1μF
RES
0.1μF
VLCD
LCD Power Supply
COM0~COM2
COM3/SEG20
SEG0~SEG19
LCD
PANEL
VSS
33Ω
VDD
C1
0.1μF
C2
100μF
1Ω
Vbat
V1
0.1μF
220Ω~1kΩ
PC0/REM
V2
0.1μF
OSC
Circuit
OSC1
HT49RA0/HT49CA0

## Instruction Set

### Introduction

Central to the successful operation of any microcontroller is its instruction set, which is a set of program instruction codes that directs the microcontroller to perform certain operations. In the case of Holtek microcontrollers, a comprehensive and flexible set of over 60 instructions is provided to enable programmers to implement their application with the minimum of programming overheads.

For easier understanding of the various instruction codes, they have been subdivided into several functional groupings.

### Instruction Timing

Most instructions are implemented within one instruction cycle. The exceptions to this are branch, call, or table read instructions where two instruction cycles are required. One instruction cycle is equal to 4 system clock cycles, therefore in the case of an 8MHz system oscillator, most instructions would be implemented within 0.5μs and branch or call instructions would be implemented within 1μs. Although instructions which require one more cycle to implement are generally limited to the JMP, CALL, RET, RETI and table read instructions, it is important to realize that any other instructions which involve manipulation of the Program Counter Low register or PCL will also take one more cycle to implement. As instructions which change the contents of the PCL will imply a direct jump to that new address, one more cycle will be required. Examples of such instructions would be ″CLR PCL″ or ″MOV PCL, A″. For the case of skip instructions, it must be noted that if the result of the comparison involves a skip operation then this will also take one more cycle, if no skip is involved then only one cycle is required.

### Moving and Transferring Data

The transfer of data within the microcontroller program is one of the most frequently used operations. Making use of three kinds of MOV instructions, data can be transferred from registers to the Accumulator and vice-versa as well as being able to move specific immediate data directly into the Accumulator. One of the most important data transfer applications is to receive data from the input ports and transfer data to the output ports.

### Arithmetic Operations

The ability to perform certain arithmetic operations and data manipulation is a necessary feature of most microcontroller applications. Within the Holtek microcontroller instruction set are a range of add and subtract instruction mnemonics to enable the necessary arithmetic to be carried out. Care must be taken to ensure correct handling of carry and borrow data when results exceed 255 for addition and less than 0 for subtraction. The increment and decrement instructions INC, INCA, DEC and DECA provide a simple means of increasing or decreasing by a value of one of the values in the destination specified.

### Logical and Rotate Operations

The standard logical operations such as AND, OR, XOR and CPL all have their own instruction within the Holtek microcontroller instruction set. As with the case of most instructions involving data manipulation, data must pass through the Accumulator which may involve additional programming steps. In all logical data operations, the zero flag may be set if the result of the operation is zero. Another form of logical data manipulation comes from the rotate instructions such as RR, RL, RRC and RLC which provide a simple means of rotating one bit right or left. Different rotate instructions exist depending on program requirements. Rotate instructions are useful for serial port programming applications where data can be rotated from an internal register into the Carry bit from where it can be examined and the necessary serial bit set high or low. Another application where rotate data operations are used is to implement multiplication and division calculations.

### Branches and Control Transfer

Program branching takes the form of either jumps to specified locations using the JMP instruction or to a subroutine using the CALL instruction. They differ in the sense that in the case of a subroutine call, the program must return to the instruction immediately when the subroutine has been carried out. This is done by placing a return instruction RET in the subroutine which will cause the program to jump back to the address right after the CALL instruction. In the case of a JMP instruction, the program simply jumps to the desired location. There is no requirement to jump back to the original jumping off point as in the case of the CALL instruction. One special and extremely useful set of branch instructions are the conditional branches. Here a decision is first made regarding the condition of a certain data memory or individual bits. Depending upon the conditions, the program will continue with the next instruction or skip over it and jump to the following instruction. These instructions are the key to decision making and branching within the program perhaps determined by the condition of certain input switches or by the condition of internal data bits.

### Bit Operations

The ability to provide single bit operations on Data Memory is an extremely flexible feature of all Holtek microcontrollers. This feature is especially useful for output port bit programming where individual bits or port pins can be directly set high or low using either the "SET [m].i" or "CLR [m].i" instructions respectively. The feature removes the need for programmers to first read the 8-bit output port, manipulate the input data to ensure that other bits are not changed and then output the port with the correct new data. This read-modify-write process is taken care of automatically when these bit operation instructions are used.

### Table Read Operations

Data storage is normally implemented by using registers. However, when working with large amounts of fixed data, the volume involved often makes it inconvenient to store the fixed data in the Data Memory. To overcome this problem, Holtek microcontrollers allow an area of Program Memory to be setup as a table where data can be directly stored. A set of easy to use instructions provides the means by which this fixed data can be referenced and retrieved from the Program Memory.

### Other Operations

In addition to the above functional instructions, a range of other instructions also exist such as the "HALT" instruction for Power-down operations and instructions to control the operation of the Watchdog Timer for reliable program operations under extreme electric or electromagnetic environments. For their relevant operations, refer to the functional related sections.

### Instruction Set Summary

The following table depicts a summary of the instruction set categorised according to function and can be consulted as a basic instruction reference using the following listed conventions.

Table conventions:

x: Bits immediate data

m: Data Memory address

A: Accumulator

i: 0~7 number of bits

addr: Program memory address

| Mnemonic | Description | Cycles | 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[Note] | 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[Note] | Z, C, AC, OV |
| SUB A,x | Subtract immediate data from the 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[Note] | 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, result in Data Memory | 1[Note] | Z, C, AC, OV |
| DAA [m] | Decimal adjust ACC for Addition with result in Data Memory | 1[Note] | C |
| **Logic Operation** | | | |
| AND A,[m] | Logical AND Data Memory to ACC | 1 | Z |
| OR A,[m] | Logical OR Data Memory to ACC | 1 | Z |
| XOR A,[m] | Logical XOR Data Memory to ACC | 1 | Z |
| ANDM A,[m] | Logical AND ACC to Data Memory | 1[Note] | Z |
| ORM A,[m] | Logical OR ACC to Data Memory | 1[Note] | Z |
| XORM A,[m] | Logical XOR ACC to Data Memory | 1[Note] | Z |
| AND A,x | Logical AND immediate Data to ACC | 1 | Z |
| OR A,x | Logical OR immediate Data to ACC | 1 | Z |
| XOR A,x | Logical XOR immediate Data to ACC | 1 | Z |
| CPL [m] | Complement Data Memory | 1[Note] | 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[Note] | Z |
| DECA [m] | Decrement Data Memory with result in ACC | 1 | Z |
| DEC [m] | Decrement Data Memory | 1[Note] | Z |

| Mnemonic | Description | Cycles | Flag Affected |
|---|---|---|---|
| **Rotate** | | | |
| RRA [m] | Rotate Data Memory right with result in ACC | 1 | None |
| RR [m] | Rotate Data Memory right | 1[Note] | None |
| RRCA [m] | Rotate Data Memory right through Carry with result in ACC | 1 | C |
| RRC [m] | Rotate Data Memory right through Carry | 1[Note] | C |
| RLA [m] | Rotate Data Memory left with result in ACC | 1 | None |
| RL [m] | Rotate Data Memory left | 1[Note] | None |
| RLCA [m] | Rotate Data Memory left through Carry with result in ACC | 1 | C |
| RLC [m] | Rotate Data Memory left through Carry | 1[Note] | C |
| **Data Move** | | | |
| MOV A,[m] | Move Data Memory to ACC | 1 | None |
| MOV [m],A | Move ACC to Data Memory | 1[Note] | None |
| MOV A,x | Move immediate data to ACC | 1 | None |
| **Bit Operation** | | | |
| CLR [m].i | Clear bit of Data Memory | 1[Note] | None |
| SET [m].i | Set bit of Data Memory | 1[Note] | None |
| **Branch** | | | |
| JMP addr | Jump unconditionally | 2 | None |
| SZ [m] | Skip if Data Memory is zero | 1[Note] | None |
| SZA [m] | Skip if Data Memory is zero with data movement to ACC | 1[note] | None |
| SZ [m].i | Skip if bit i of Data Memory is zero | 1[Note] | None |
| SNZ [m].i | Skip if bit i of Data Memory is not zero | 1[Note] | None |
| SIZ [m] | Skip if increment Data Memory is zero | 1[Note] | None |
| SDZ [m] | Skip if decrement Data Memory is zero | 1[Note] | None |
| SIZA [m] | Skip if increment Data Memory is zero with result in ACC | 1[Note] | None |
| SDZA [m] | Skip if decrement Data Memory is zero with result in ACC | 1[Note] | 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 table (current page) to TBLH and Data Memory | 2[Note] | None |
| TABRDL [m] | Read table (last page) to TBLH and Data Memory | 2[Note] | None |
| **Miscellaneous** | | | |
| NOP | No operation | 1 | None |
| CLR [m] | Clear Data Memory | 1[Note] | None |
| SET [m] | Set Data Memory | 1[Note] | None |
| CLR WDT | Clear Watchdog Timer | 1 | TO, PDF |
| CLR WDT1 | Pre-clear Watchdog Timer | 1 | TO, PDF |
| CLR WDT2 | Pre-clear Watchdog Timer | 1 | TO, PDF |
| SWAP [m] | Swap nibbles of Data Memory | 1[Note] | None |
| SWAPA [m] | Swap nibbles of Data Memory with result in ACC | 1 | None |
| HALT | Enter power down mode | 1 | TO, PDF |

Note: 1. For skip instructions, if the result of the comparison involves a skip then two cycles are required, if no skip takes place only one cycle is required.
2. Any instruction which changes the contents of the PCL will also require 2 cycles for execution.
3. For the ″CLR WDT1″ and ″CLR WDT2″ instructions the TO and PDF flags may be affected by the execution status. The TO and PDF flags are cleared after both ″CLR WDT1″ and ″CLR WDT2″ instructions are consecutively executed. Otherwise the TO and PDF flags remain unchanged.

## Instruction Definition

| | |
|---|---|
| **ADC A,[m]** | Add Data Memory to ACC with Carry |
| Description | The contents of the specified Data Memory, Accumulator and the carry flag are added. The result is stored in the Accumulator. |
| Operation | ACC ← ACC + [m] + C |
| Affected flag(s) | OV, Z, AC, C |
| **ADCM A,[m]** | Add ACC to Data Memory with Carry |
| Description | The contents of the specified Data Memory, Accumulator and the carry flag are added. The result is stored in the specified Data Memory. |
| Operation | [m] ← ACC + [m] + C |
| Affected flag(s) | OV, Z, AC, C |
| **ADD A,[m]** | Add Data Memory to ACC |
| 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) | OV, Z, AC, C |
| **ADD A,x** | Add immediate data to ACC |
| Description | The contents of the Accumulator and the specified immediate data are added. The result is stored in the Accumulator. |
| Operation | ACC ← ACC + x |
| Affected flag(s) | OV, Z, AC, C |
| **ADDM A,[m]** | Add ACC to Data Memory |
| Description | The contents of the specified Data Memory and the Accumulator are added. The result is stored in the specified Data Memory. |
| Operation | [m] ← ACC + [m] |
| Affected flag(s) | OV, Z, AC, C |
| **AND A,[m]** | Logical AND Data Memory to ACC |
| 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) | Z |
| **AND A,x** | Logical AND immediate data to ACC |
| Description | Data in the Accumulator and the specified immediate data perform a bitwise logical AND operation. The result is stored in the Accumulator. |
| Operation | ACC ← ACC ″AND″ x |
| Affected flag(s) | Z |
| **ANDM A,[m]** | Logical AND ACC to Data Memory |
| 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) | Z |

| | |
|---|---|
| **CALL addr** | Subroutine call |
| Description | Unconditionally calls a subroutine at the specified address. The Program Counter then increments by 1 to obtain the address of the next instruction which is then pushed onto the stack. The specified address is then loaded and the program continues execution from this new address. As this instruction requires an additional operation, it is a two cycle instruction. |
| Operation | Stack ← Program Counter + 1<br>Program Counter ← addr |
| Affected flag(s) | None |
| **CLR [m]** | Clear Data Memory |
| Description | Each bit of the specified Data Memory is cleared to 0. |
| Operation | [m] ← 00H |
| Affected flag(s) | None |
| **CLR [m].i** | Clear bit of Data Memory |
| Description | Bit i of the specified Data Memory is cleared to 0. |
| Operation | [m].i ← 0 |
| Affected flag(s) | None |
| **CLR WDT** | Clear Watchdog Timer |
| Description | The TO, PDF flags and the WDT are all cleared. |
| Operation | WDT cleared<br>TO ← 0<br>PDF ← 0 |
| Affected flag(s) | TO, PDF |
| **CLR WDT1** | Pre-clear Watchdog Timer |
| Description | The TO, PDF flags and the WDT are all cleared. Note that this instruction works in conjunction with CLR WDT2 and must be executed alternately with CLR WDT2 to have effect. Repetitively executing this instruction without alternately executing CLR WDT2 will have no effect. |
| Operation | WDT cleared<br>TO ← 0<br>PDF ← 0 |
| Affected flag(s) | TO, PDF |
| **CLR WDT2** | Pre-clear Watchdog Timer |
| Description | The TO, PDF flags and the WDT are all cleared. Note that this instruction works in conjunction with CLR WDT1 and must be executed alternately with CLR WDT1 to have effect. Repetitively executing this instruction without alternately executing CLR WDT1 will have no effect. |
| Operation | WDT cleared<br>TO ← 0<br>PDF ← 0 |
| Affected flag(s) | TO, PDF |

| | |
|---|---|
| **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 | $[m] \leftarrow \overline{[m]}$ |
| Affected flag(s) | Z |
| **CPLA [m]** | Complement Data Memory with result in ACC |
| 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 | $ACC \leftarrow \overline{[m]}$ |
| Affected flag(s) | Z |
| **DAA [m]** | Decimal-Adjust ACC for addition with result in Data Memory |
| Description | Convert the contents of the Accumulator value to a BCD ( Binary Coded Decimal) value resulting from the previous addition of two BCD variables. If the low nibble is greater than 9 or if AC flag is set, then a value of 6 will be added to the low nibble. Otherwise the low nibble remains unchanged. If the high nibble is greater than 9 or if the C flag is set, then a value of 6 will be added to the high nibble. Essentially, the decimal conversion is performed by adding 00H, 06H, 60H or 66H depending on the Accumulator and flag conditions. Only the C flag may be affected by this instruction which indicates that if the original BCD sum is greater than 100, it allows multiple precision decimal addition. |
| Operation | $[m] \leftarrow ACC + 00H$ or<br>$[m] \leftarrow ACC + 06H$ or<br>$[m] \leftarrow ACC + 60H$ or<br>$[m] \leftarrow ACC + 66H$ |
| Affected flag(s) | C |
| **DEC [m]** | Decrement Data Memory |
| Description | Data in the specified Data Memory is decremented by 1. |
| Operation | $[m] \leftarrow [m] - 1$ |
| Affected flag(s) | Z |
| **DECA [m]** | Decrement Data Memory with result in ACC |
| Description | Data in the specified Data Memory is decremented by 1. The result is stored in the Accumulator. The contents of the Data Memory remain unchanged. |
| Operation | $ACC \leftarrow [m] - 1$ |
| Affected flag(s) | Z |
| **HALT** | Enter power down mode |
| Description | This instruction stops the program execution and turns off the system clock. The contents of the Data Memory and registers are retained. The WDT and prescaler are cleared. The power down flag PDF is set and the WDT time-out flag TO is cleared. |
| Operation | $TO \leftarrow 0$<br>$PDF \leftarrow 1$ |
| Affected flag(s) | TO, PDF |

| | |
|---|---|
| **INC [m]** | Increment Data Memory |
| Description | Data in the specified Data Memory is incremented by 1. |
| Operation | [m] ← [m] + 1 |
| Affected flag(s) | Z |
| **INCA [m]** | Increment Data Memory with result in ACC |
| Description | Data in the specified Data Memory is incremented by 1. The result is stored in the Accumulator. The contents of the Data Memory remain unchanged. |
| Operation | ACC ← [m] + 1 |
| Affected flag(s) | Z |
| **JMP addr** | Jump unconditionally |
| Description | The contents of the Program Counter are replaced with the specified address. Program execution then continues from this new address. As this requires the insertion of a dummy instruction while the new address is loaded, it is a two cycle instruction. |
| Operation | Program Counter ← addr |
| Affected flag(s) | None |
| **MOV A,[m]** | Move Data Memory to ACC |
| Description | The contents of the specified Data Memory are copied to the Accumulator. |
| Operation | ACC ← [m] |
| Affected flag(s) | None |
| **MOV A,x** | Move immediate data to ACC |
| Description | The immediate data specified is loaded into the Accumulator. |
| Operation | ACC ← x |
| Affected flag(s) | None |
| **MOV [m],A** | Move ACC to Data Memory |
| Description | The contents of the Accumulator are copied to the specified Data Memory. |
| Operation | [m] ← ACC |
| Affected flag(s) | None |
| **NOP** | No operation |
| Description | No operation is performed. Execution continues with the next instruction. |
| Operation | No operation |
| Affected flag(s) | None |
| **OR A,[m]** | Logical OR Data Memory to ACC |
| Description | Data in the Accumulator and the specified Data Memory perform a bitwise logical OR operation. The result is stored in the Accumulator. |
| Operation | ACC ← ACC ″OR″ [m] |
| Affected flag(s) | Z |

| | |
|---|---|
| **OR A,x** | Logical OR immediate data to ACC |
| Description | Data in the Accumulator and the specified immediate data perform a bitwise logical OR operation. The result is stored in the Accumulator. |
| Operation | ACC ← ACC ″OR″ x |
| Affected flag(s) | Z |
| **ORM A,[m]** | Logical OR ACC to Data Memory |
| Description | Data in the specified Data Memory 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) | Z |
| **RET** | Return from subroutine |
| Description | The Program Counter is restored from the stack. Program execution continues at the restored address. |
| Operation | Program Counter ← Stack |
| Affected flag(s) | None |
| **RET A,x** | Return from subroutine and load immediate data to ACC |
| Description | The Program Counter is restored from the stack and the Accumulator loaded with the specified immediate data. Program execution continues at the restored address. |
| Operation | Program Counter ← Stack<br>ACC ← x |
| Affected flag(s) | None |
| **RETI** | Return from interrupt |
| Description | The Program Counter is restored from the stack and the interrupts are re-enabled by setting the EMI bit. EMI is the master interrupt global enable bit. If an interrupt was pending when the RETI instruction is executed, the pending Interrupt routine will be processed before returning to the main program. |
| Operation | Program Counter ← Stack<br>EMI ← 1 |
| Affected flag(s) | None |
| **RL [m]** | Rotate Data Memory left |
| Description | The contents of the specified Data Memory are rotated left by 1 bit with bit 7 rotated into bit 0. |
| Operation | [m].(i+1) ← [m].i; (i = 0~6)<br>[m].0 ← [m].7 |
| Affected flag(s) | None |
| **RLA [m]** | Rotate Data Memory left with result in ACC |
| Description | The contents of the specified Data Memory are rotated left by 1 bit with bit 7 rotated into bit 0. The rotated result is stored in the Accumulator and the contents of the Data Memory remain unchanged. |
| Operation | ACC.(i+1) ← [m].i; (i = 0~6)<br>ACC.0 ← [m].7 |
| Affected flag(s) | None |

| | |
|---|---|
| **RLC [m]** | Rotate Data Memory left through Carry |
| Description | The contents of the specified Data Memory and the carry flag are rotated left by 1 bit. Bit 7 replaces the Carry bit and the original carry flag is rotated into bit 0. |
| Operation | [m].(i+1) ← [m].i; (i = 0~6)<br>[m].0 ← C<br>C ← [m].7 |
| Affected flag(s) | C |
| **RLCA [m]** | Rotate Data Memory left through Carry with result in ACC |
| Description | Data in the specified Data Memory and the carry flag are rotated left by 1 bit. Bit 7 replaces the Carry bit and the original carry flag is rotated into the bit 0. The rotated result is stored in the Accumulator and the contents of the Data Memory remain unchanged. |
| Operation | ACC.(i+1) ← [m].i; (i = 0~6)<br>ACC.0 ← C<br>C ← [m].7 |
| Affected flag(s) | C |
| **RR [m]** | Rotate Data Memory right |
| Description | The contents of the specified Data Memory are rotated right by 1 bit with bit 0 rotated into bit 7. |
| Operation | [m].i ← [m].(i+1); (i = 0~6)<br>[m].7 ← [m].0 |
| Affected flag(s) | None |
| **RRA [m]** | Rotate Data Memory right with result in ACC |
| Description | Data in the specified Data Memory and the carry flag are rotated right by 1 bit with bit 0 rotated into bit 7. The rotated result is stored in the Accumulator and the contents of the Data Memory remain unchanged. |
| Operation | ACC.i ← [m].(i+1); (i = 0~6)<br>ACC.7 ← [m].0 |
| Affected flag(s) | None |
| **RRC [m]** | Rotate Data Memory right through Carry |
| Description | The contents of the specified Data Memory and the carry flag are rotated right by 1 bit. Bit 0 replaces the Carry bit and the original carry flag is rotated into bit 7. |
| Operation | [m].i ← [m].(i+1); (i = 0~6)<br>[m].7 ← C<br>C ← [m].0 |
| Affected flag(s) | C |
| **RRCA [m]** | Rotate Data Memory right through Carry with result in ACC |
| Description | Data in the specified Data Memory and the carry flag are rotated right by 1 bit. Bit 0 replaces the Carry bit and the original carry flag is rotated into bit 7. The rotated result is stored in the Accumulator and the contents of the Data Memory remain unchanged. |
| Operation | ACC.i ← [m].(i+1); (i = 0~6)<br>ACC.7 ← C<br>C ← [m].0 |
| Affected flag(s) | C |

| | |
|---|---|
| **SBC A,[m]** | Subtract Data Memory from ACC with Carry |
| Description | The contents of the specified Data Memory and the complement of the carry flag are subtracted from the Accumulator. The result is stored in the Accumulator. Note that if the result of subtraction is negative, the C flag will be cleared to 0, otherwise if the result is positive or zero, the C flag will be set to 1. |
| Operation | ACC ← ACC – [m] – $\overline{C}$ |
| Affected flag(s) | OV, Z, AC, C |
| **SBCM A,[m]** | Subtract Data Memory from ACC with Carry and result in Data Memory |
| Description | The contents of the specified Data Memory and the complement of the carry flag are subtracted from the Accumulator. The result is stored in the Data Memory. Note that if the result of subtraction is negative, the C flag will be cleared to 0, otherwise if the result is positive or zero, the C flag will be set to 1. |
| Operation | [m] ← ACC – [m] – $\overline{C}$ |
| Affected flag(s) | OV, Z, AC, C |
| **SDZ [m]** | Skip if decrement Data Memory is 0 |
| Description | The contents of the specified Data Memory are first decremented by 1. If the result is 0 the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0 the program proceeds with the following instruction. |
| Operation | [m] ← [m] – 1<br>Skip if [m] = 0 |
| Affected flag(s) | None |
| **SDZA [m]** | Skip if decrement Data Memory is zero with result in ACC |
| Description | The contents of the specified Data Memory are first decremented by 1. If the result is 0, the following instruction is skipped. The result is stored in the Accumulator but the specified Data Memory contents remain unchanged. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0, the program proceeds with the following instruction. |
| Operation | ACC ← [m] – 1<br>Skip if ACC = 0 |
| Affected flag(s) | None |
| **SET [m]** | Set Data Memory |
| Description | Each bit of the specified Data Memory is set to 1. |
| Operation | [m] ← FFH |
| Affected flag(s) | None |
| **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) | None |

| | |
|---|---|
| **SIZ [m]** | Skip if increment Data Memory is 0 |
| Description | The contents of the specified Data Memory are first incremented by 1. If the result is 0, the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0 the program proceeds with the following instruction. |
| Operation | [m] ← [m] + 1<br>Skip if [m] = 0 |
| Affected flag(s) | None |
| **SIZA [m]** | Skip if increment Data Memory is zero with result in ACC |
| Description | The contents of the specified Data Memory are first incremented by 1. If the result is 0, the following instruction is skipped. The result is stored in the Accumulator but the specified Data Memory contents remain unchanged. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0 the program proceeds with the following instruction. |
| Operation | ACC ← [m] + 1<br>Skip if ACC = 0 |
| Affected flag(s) | None |
| **SNZ [m].i** | Skip if bit i of Data Memory is not 0 |
| Description | If bit i of the specified Data Memory is not 0, the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is 0 the program proceeds with the following instruction. |
| Operation | Skip if [m].i ≠ 0 |
| Affected flag(s) | None |
| **SUB A,[m]** | Subtract Data Memory from ACC |
| Description | The specified Data Memory is subtracted from the contents of the Accumulator. The result is stored in the Accumulator. Note that if the result of subtraction is negative, the C flag will be cleared to 0, otherwise if the result is positive or zero, the C flag will be set to 1. |
| Operation | ACC ← ACC – [m] |
| Affected flag(s) | OV, Z, AC, C |
| **SUBM A,[m]** | Subtract Data Memory from ACC with result in Data Memory |
| Description | The specified Data Memory is subtracted from the contents of the Accumulator. The result is stored in the Data Memory. Note that if the result of subtraction is negative, the C flag will be cleared to 0, otherwise if the result is positive or zero, the C flag will be set to 1. |
| Operation | [m] ← ACC – [m] |
| Affected flag(s) | OV, Z, AC, C |
| **SUB A,x** | Subtract immediate data from ACC |
| Description | The immediate data specified by the code is subtracted from the contents of the Accumulator. The result is stored in the Accumulator. Note that if the result of subtraction is negative, the C flag will be cleared to 0, otherwise if the result is positive or zero, the C flag will be set to 1. |
| Operation | ACC ← ACC – x |
| Affected flag(s) | OV, Z, AC, C |

| | |
|---|---|
| **SWAP [m]** | Swap nibbles of Data Memory |
| Description | The low-order and high-order nibbles of the specified Data Memory are interchanged. |
| Operation | [m].3~[m].0 ↔ [m].7 ~ [m].4 |
| Affected flag(s) | None |
| **SWAPA [m]** | Swap nibbles of Data Memory with result in ACC |
| Description | The low-order and high-order nibbles of the specified Data Memory are interchanged. The result is stored in the Accumulator. The contents of the Data Memory remain unchanged. |
| Operation | ACC.3 ~ ACC.0 ← [m].7 ~ [m].4<br>ACC.7 ~ ACC.4 ← [m].3 ~ [m].0 |
| Affected flag(s) | None |
| **SZ [m]** | Skip if Data Memory is 0 |
| Description | If the contents of the specified Data Memory is 0, the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0 the program proceeds with the following instruction. |
| Operation | Skip if [m] = 0 |
| Affected flag(s) | None |
| **SZA [m]** | Skip if Data Memory is 0 with data movement to ACC |
| Description | The contents of the specified Data Memory are copied to the Accumulator. If the value is zero, the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0 the program proceeds with the following instruction. |
| Operation | ACC ← [m]<br>Skip if [m] = 0 |
| Affected flag(s) | None |
| **SZ [m].i** | Skip if bit i of Data Memory is 0 |
| Description | If bit i of the specified Data Memory is 0, the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0, the program proceeds with the following instruction. |
| Operation | Skip if [m].i = 0 |
| Affected flag(s) | None |
| **TABRDC [m]** | Read table (current page) to TBLH and Data Memory |
| Description | The low byte of the program code (current page) addressed by the table pointer (TBLP) is moved to the specified Data Memory and the high byte moved to TBLH. |
| Operation | [m] ← program code (low byte)<br>TBLH ← program code (high byte) |
| Affected flag(s) | None |
| **TABRDL [m]** | Read table (last page) to TBLH and Data Memory |
| Description | The low byte of the program code (last page) addressed by the table pointer (TBLP) is moved to the specified Data Memory and the high byte moved to TBLH. |
| Operation | [m] ← program code (low byte)<br>TBLH ← program code (high byte) |
| Affected flag(s) | None |

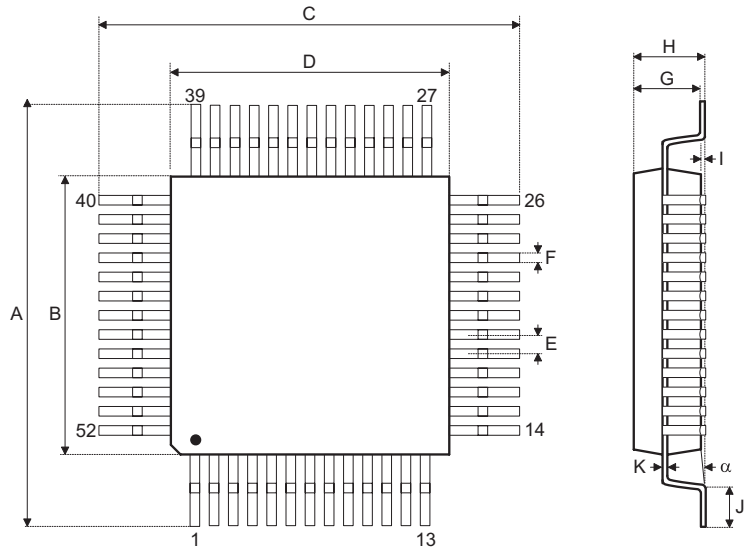
| | |
|---|---|
| **XOR A,[m]** | Logical XOR Data Memory to ACC |
| Description | Data in the Accumulator and the specified Data Memory perform a bitwise logical XOR operation. The result is stored in the Accumulator. |
| Operation | ACC ← ACC ″XOR″ [m] |
| Affected flag(s) | Z |

| | |
|---|---|
| **XORM A,[m]** | Logical XOR ACC to Data Memory |
| Description | Data in the specified Data Memory and the Accumulator perform a bitwise logical XOR operation. The result is stored in the Data Memory. |
| Operation | [m] ← ACC ″XOR″ [m] |
| Affected flag(s) | Z |

| | |
|---|---|
| **XOR A,x** | Logical XOR immediate data to ACC |
| Description | Data in the Accumulator and the specified immediate data perform a bitwise logical XOR operation. The result is stored in the Accumulator. |
| Operation | ACC ← ACC ″XOR″ x |
| Affected flag(s) | Z |

## Package Information

**52-pin QFP (14mm×14mm) Outline Dimensions**




| Symbol | Dimensions in mm | | |
|---|---|---|---|
| | Min. | Nom. | Max. |
| A | 17.3 | — | 17.5 |
| B | 13.9 | — | 14.1 |
| C | 17.3 | — | 17.5 |
| D | 13.9 | — | 14.1 |
| E | — | 1 | — |
| F | — | 0.4 | — |
| G | 2.5 | — | 3.1 |
| H | — | — | 3.4 |
| I | — | 0.1 | — |
| J | 0.73 | — | 1.03 |
| K | 0.1 | — | 0.2 |
| α | 0° | — | 7° |

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

**Holtek Semiconductor Inc. (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 (China) Inc. (Dongguan Sales Office)**
Building No. 10, Xinzhu Court, (No. 1 Headquarters), 4 Cuizhu Road, Songshan Lake, Dongguan, China 523808
Tel: 86-769-2626-1300
Fax: 86-769-2626-1311

**Holtek Semiconductor (USA), Inc. (North America Sales Office)**
46729 Fremont Blvd., Fremont, CA 94538, USA
Tel: 1-510-252-9880
Fax: 1-510-252-9885
http://www.holtek.com

Copyright © 2008 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.