# HT47C10L

## R-F Type Low Voltage 8-Bit Mask MCU

### Technical Document

- Tools Information
- FAQs
- Application Note
  - HA0029E Using the Time Base Function in the HT47R20A-1
  - HA0030E Using the RTC in the HT47R20A-1
  - HA0034E Using the Buzzer Function in the HT47R20A-1
  - HA0036E Using the PFD Function in the HT47R20A-1
  - HA0045E Distinguishing between the Different Devices in the HT47 MCU Series

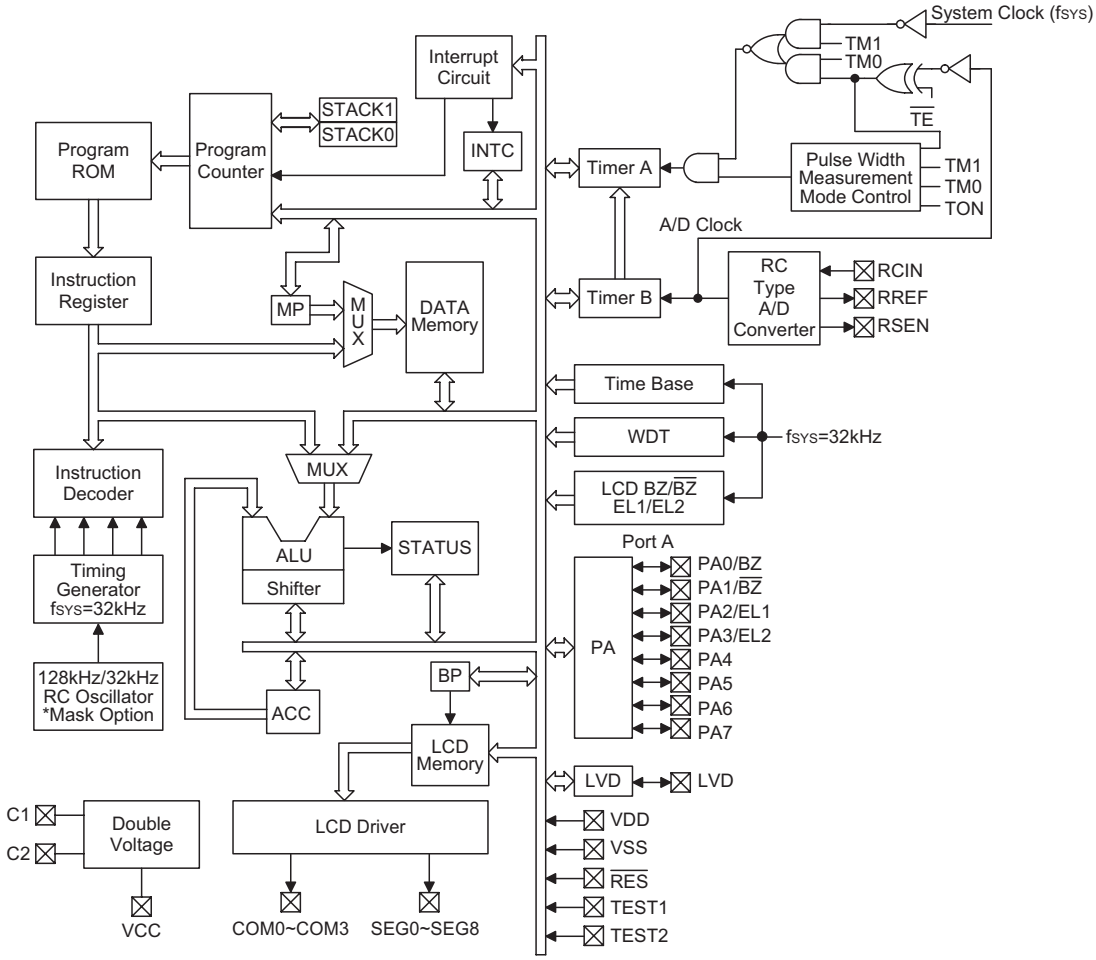
### Features

- Operating voltage: 1.2V~2.2V
- Eight bidirectional I/O lines
- On-chip 32kHz/128kHz built-in RC oscillator (Mask option; 128kHz is selected especially for EL driving)
- Watchdog Timer
- 1K×16 program memory ROM
- 32×8 data memory RAM
- One time base (TB)
- One buzzer output
- One EL output
- One externally adjustable low voltage detector
- HALT function and wake-up feature reduce power consumption
- One LCD driver with 9×4 segments, 1/4 duty, 1/2 bias
- RC type A/D converter
- Two-level subroutine nesting
- Bit manipulation instruction
- 16-bit table read instruction
- Up to 122μs instruction cycle with 32768Hz system clock
- All instructions in one or two machine cycles
- 63 powerful instructions
- 44-pin QFP package

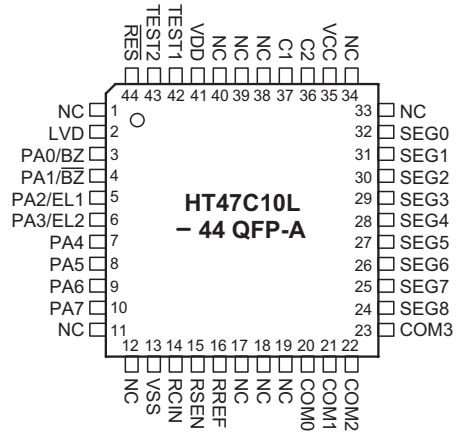
### General Description

The HT47C10L is an 8-bit high performance RISC-like microcontroller. Its single cycle instruction and two-stage pipeline architecture make it suitable for high speed applications. The device is suited for clinical thermometers.

## Block Diagram




## Pin Assignment



## Pin Description

| Pin Name | I/O | Function |
|---|---|---|
| $\overline{RES}$ | I | Schmitt trigger reset input. Active low |
| PA0/BZ<br>PA1/$\overline{BZ}$ | I/O<br>I/O | Bidirectional 2-bit input/output port. Each bit can be a wake-up input. The PA0 and PA1 are pin-shared with the BZ and $\overline{BZ}$, respectively. Once the PA0 and PA1 are selected as buzzer driving outputs, the output signals come from an internal buzzer clock generator. Software instructions determine the CMOS output or Schmitt trigger input with pull-high resistor |
| PA2/EL1<br>PA3/EL2 | I/O<br>I/O | Bidirectional 2-bit input/output port. Each bit can be a wake-up input. The PA2 and PA3 are pin-shared with the EL1 and EL2, respectively. Once the PA2 and PA3 are selected as EL driving outputs, the output signals come from an internal EL clock generator.<br>Software instructions determine the CMOS output or Schmitt trigger input with pull-high resistor |
| PA4~PA7 | I/O | Bidirectional 4-bit input/output port. Each bit can be a wake-up input.<br>Software instructions determine the CMOS output or Schmitt trigger input with pull-high resistor |
| VSS | — | Negative power supply, ground |
| VCC, C1, C2 | — | For double voltage. VCC=2×VDD<br>VCC: LCD power supply voltage, a capacitor has to be connected between VCC and VSS.<br>C1, C2: Switching pins for VCC, a capacitor has to be connected between C1 and C2 |
| SEG8~SEG0<br>COM3~COM0 | O | LCD driver outputs for LCD panel segments and commons. |
| VDD | — | Positive power supply |
| LVD | B | Low voltage detector. A resistor has to be connected between VSS and LVD |
| RCIN | I | RC type A/D converter input pin for RC oscillation. |
| RREF | O | RC type A/D converter output pin for reference resistor oscillation. |
| RSEN | O | RC type A/D converter output pin for sensor resistor oscillation |
| TEST1<br>TEST2 | I | TEST mode input pin with pull-high resistor.<br>Let open in normal mode |
| TRIM1~TRIM6 | I | TEST mode input pin. Let open in normal mode |

## Absolute Maximum Ratings

Supply Voltage ..... $V_{SS}$–0.3V to $V_{SS}$+2.5V

Input Voltage ..... $V_{SS}$–0.3V to $V_{DD}$+0.3V

Storage Temperature ..... –50°C to 125°C

Operating Temperature ..... –40°C to 85°C

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

## D.C. Characteristics

Ta=25°C

| Symbol | Parameter | Test Conditions | | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|---|---|
| | | $V_{DD}$ | Conditions | | | | |
| $V_{DD}$ | Operating Voltage | — | — | 1.2 | 1.5 | 2.2 | V |
| $V_{CC}$ | LCD Voltage | — | VCC=2×VDD | 2.4 | 3 | 4.4 | V |
| $V_{LVD}$ | Low Voltage Detector Voltage | — | *$R_{LVD}$=30kΩ | 1.25 | 1.3 | 1.35 | V |
| $I_{DD}$ | Operating Current | 1.5V | No load, $f_{OSC}$=128kHz $f_{SYS}$=32kHz, A/D Off, LVD disable | — | 9 | 20 | μA |
| | | | No load, $f_{OSC}$=128kHz $f_{SYS}$=32kHz, A/D On, LVD disable *R=30kΩ, *C=2200pF | — | 26 | 50 | μA |
| | | | No load, $f_{OSC}$=32kHz $f_{SYS}$=32kHz, A/D Off, LVD disable | — | 5 | 10 | μA |
| | | | No load, $f_{OSC}$=32kHz $f_{SYS}$=32kHz, A/D On, LVD disable *R=30kΩ, *C=2200pF | — | 23 | 40 | μA |
| $I_{LVD}$ | LVD Current | 1.5V | LVD enable | — | 50 | 100 | μA |
| $I_{STB1}$ | Standby Current (LVD Disable, LCD Off) | 1.5V | No load, system HALT A/D Off, LVD Off | — | — | 1 | μA |
| $I_{STB2}$ | Standby Current (LCD On) | 1.5V | No load, $f_{OSC}$=128kHz $f_{SYS}$=32kHz, A/D Off, LVD disable | — | 7 | 15 | μA |
| | | | No load, $f_{OSC}$=32kHz $f_{SYS}$=32kHz, A/D Off, LVD disable | — | 2.5 | 5 | μA |
| $V_{IL1}$ | Input Low Voltage for I/O Ports | — | — | 0 | — | $0.3V_{DD}$ | V |
| $V_{IH1}$ | Input High Voltage for I/O Ports | — | — | $0.8V_{DD}$ | — | $V_{DD}$ | V |
| $V_{IL2}$ | Input Low Voltage ($\overline{RES}$) | — | — | 0 | — | $0.4V_{DD}$ | V |
| $V_{IH2}$ | Input High Voltage ($\overline{RES}$) | — | — | $0.9V_{DD}$ | — | $V_{DD}$ | V |
| $I_{OL1}$ | Sink Current PA0 (BZ), PA1 ($\overline{BZ}$), PA2 (EL1), PA3 (EL2), PA4~PA7 | 1.5V | $V_{OL}$=0.15V | 0.5 | 0.8 | — | mA |
| $I_{OH1}$ | Source Current PA0 (BZ), PA1 ($\overline{BZ}$), PA2 (EL1), PA3 (EL2), PA4~PA7 | 1.5V | $V_{OH}$=1.35V | –0.3 | –0.6 | — | mA |
| $I_{OL2}$ | Common Output Sink Current | 1.5V | $V_{OL}$=0.3V (1/2 bias) | 50 | 100 | — | μA |
| $I_{OH2}$ | Common Output Source Current | 1.5V | $V_{OH}$=2.7V (1/2 bias) | –50 | –100 | — | μA |
| $I_{OL3}$ | Segment Output Sink Current | 1.5V | $V_{OL}$=0.3V (1/2 bias) | 50 | 100 | — | μA |
| $I_{OH3}$ | Segment Output Source Current | 1.5V | $V_{OH}$=2.7V(1/2 bias) | –50 | –100 | — | μA |
| $R_{PH1}$ | Pull-high Resistance of I/O Ports | 1.5V | $V_{IL}$=0V | 75 | 150 | 300 | kΩ |
| $R_{PH2}$ | Pull-high Resistance of $\overline{TEST}$ | 1.5V | $V_{IL}$=0V | 75 | 150 | 300 | kΩ |

Note: *R means the resistance of RC type A/D converter

*C means the capacitance of RC type A/D converter

*$R_{LVD}$ value may be different for different lot

## A.C. Characteristics

Ta=25°C

| Symbol | Parameter | Test Conditions | | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|---|---|
| | | $V_{DD}$ | Conditions | | | | |
| $f_{32K}$ | Oscillator Clock (32kHz option) | 1.5V | — | 26 | 32 | 40 | kHz |
| $f_{128K}$ | Oscillator Clock (128kHz option) | 1.5V | — | 102 | 128 | 160 | kHz |
| $t_{RES}$ | External Reset Low Pulse Width | 1.5V | — | 100 | — | — | μs |
| $f_{AD}$ | A/D Converter Frequency | 1.5V | — | — | — | 50 | kHz |

## Functional Description

### Execution Flow

The HT47C10L system clock is derived from an about 32kHz built-in RC oscillator. The system clock is internally divided into four non-overlapping clocks (T1, T2, T3 and T4). 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. However, the pipelining scheme causes each instruction to effectively execute in one cycle. If an instruction changes the program counter, two cycles are required to complete the instruction.

### Program Counter – PC

The 10-bit program counter (PC) controls the sequence in which the instructions stored in the program ROM are executed and its contents specify a maximum of 1024 addresses.

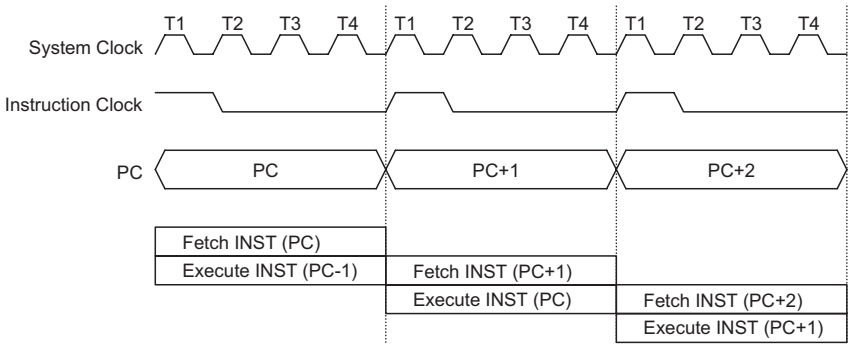
After accessing a program memory word to fetch an instruction code, the contents of the program counter are incremented by one. The program counter then points to the memory word containing the next instruction code.

When executing a jump instruction, conditional skip execution, loading PCL register, subroutine call, initial reset, internal interrupt, external interrupt or return from subroutine, the PC manipulates the program transfer by loading the address corresponding to each instruction.

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

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

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




**Execution Flow**

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

**Program Counter**

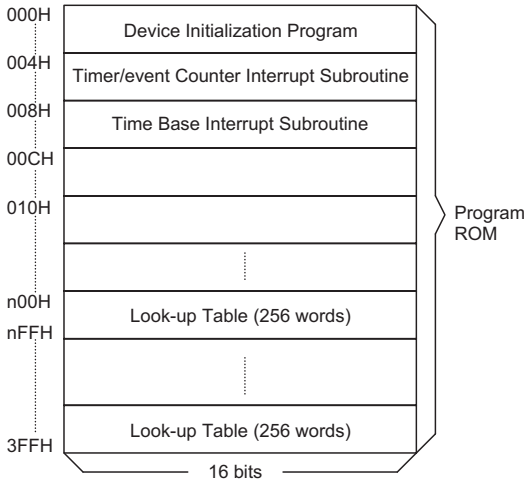
Note: *9~*0: Program counter bits
#9~#0: Instruction code bits
S9~S0: Stack register bits
@7~@0: PCL bits

**Program Memory – ROM**

The program memory is used to store the program instructions, which are to be executed. It also contains data, table, and interrupt entries, and is organized into 1024×16 bits, addressed by the program counter and table pointer.

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

- Location 000H
  This area is reserved for the initialization program. After chip reset, the program always begins execution at location 000H.
- Location 004H
  This area is reserved for the timer/event counter interrupt service program. If timer interrupt results from a timer/event counter A or B overflow, and if the interrupt is enabled and the stack is not full, the program begins execution at location 004H.
- Location 008H
  This area is reserved for the time base interrupt service program. If a time base interrupt occurs, and if the interrupt is enabled and the stack is not full, the program begins execution at location 008H.
- Table location
  Any location in the ROM space can be used as look-up tables. The instructions "TABRDC [m]" (the current page, 1 page=256 words) and "TABRDL [m]" (the last page) transfer the contents of the lower-order byte to the specified data memory, and the higher-order byte to TBLH (08H). Only the destination of the lower-order byte in the table is well-defined, the higher-order byte of the table word are transferred to the TBLH. The table higher-order byte register (TBLH) is read only. The table pointer (TBLP) is a read/write register (07H), which indicates the table location. Before accessing the table, the location must be placed in TBLP. The TBLH is read only and cannot be restored. If the main routine and the ISR (interrupt service routine) both employ the table read instruction, the contents of the TBLH in the main routine are likely to be changed by the table read instruction used in the ISR. Errors can occur. In other words using the table read instruction in the main routine and the ISR simultaneously should be avoided. However, if the table read instruction has to be applied in both the main routine and the ISR, the interrupt is supposed to be disabled prior to the table read instruction. It will not be enabled until the TBLH has been backed up. All table related instructions need two cycles to complete the operation. These areas may function as normal program memory depending upon the requirements.




Note: n ranges from 0 to 3

**Program Memory**

**Stack Register – STACK**

This is a special part of the memory which is used to save the contents of the Program Counter only. The stack is organized into two levels and is neither part of the data nor part of the program space, and is neither readable nor writeable. The activated level is indexed by the stack pointer (SP) and is neither readable nor writeable. At a subroutine call or interrupt acknowledgment, the contents of the program counter are pushed onto the stack. At the end of a subroutine or an interrupt routine, signaled by a return instruction (RET or RETI), the program counter is restored to its previous value from the stack. After a chip reset, the 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 will be recorded but the acknowledgment will be inhibited. When the stack pointer is decremented (by RET or RETI), the interrupt will be serviced. This feature prevents stack overflow al-

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

**Table Location**

Note: *9~*0: Bits of table location

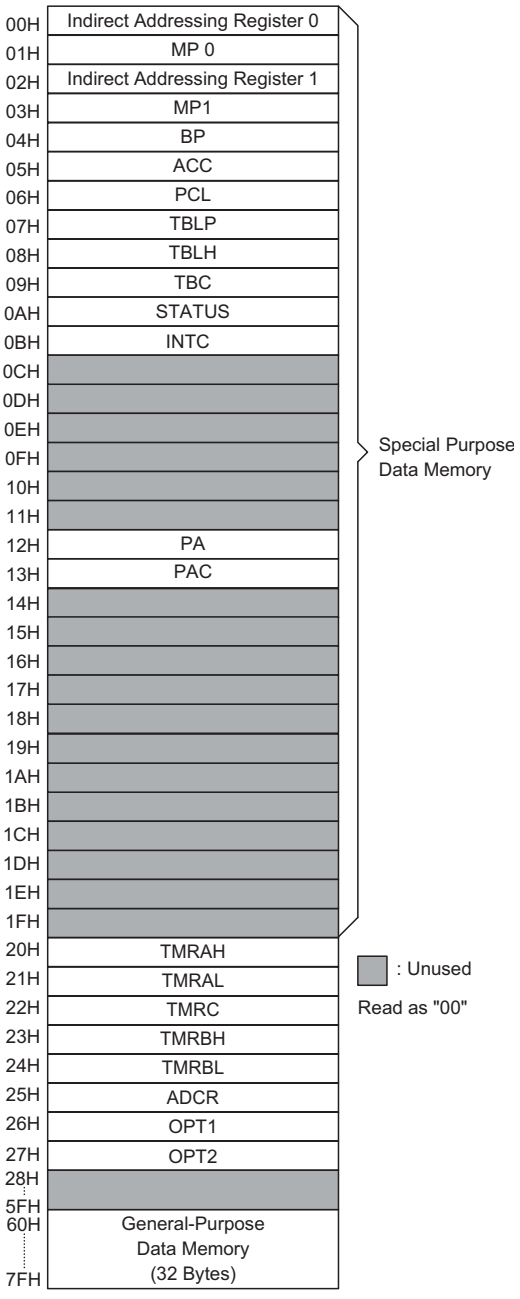
@7~@0: Bits of table pointer

P9~P8: Bits of current program counter

lowing the programmer to use the structure more easily. In a similar case, if the stack is full and a "CALL" is subsequently executed, stack overflow occurs and the first entry will be lost (only the most recent two return addresses is stored).

**Data Memory – RAM**

The data memory is designed with 54×8 bits. The data memory is divided into two functional groups: special function registers and general-purpose data memory (32×8). Most are read/write, but some are read only.




**RAM Mapping (Bank 0)**

The special function registers include the indirect addressing register 0 (00H), the memory pointer register 0 (MP0; 01H), the indirect addressing register 1 (02H), the memory pointer register 1 (MP1;03H), the bank pointer (BP;04H), the accumulator (ACC;05H), the program counter lower-order byte register (PCL;06H), the table pointer (TBLP;07H), the table higher-order byte register (TBLH;08H), the time base control register (TBC;09H), the status register (STATUS;0AH), the interrupt control register 0 (INTC;0BH), the I/O registers (PA;12H), I/O port control register (PAC;13H), the timer/event counter A higher-order byte register (TMRAH; 20H), the timer/event counter A lower-order byte register (TMRAL; 21H), the timer/event counter control register (TMRC; 22H), the timer/event counter B higher-order byte register (TMRBH; 23H), the timer/event counter B lower-order byte register (TMRBL; 24H), the RC oscillator type A/D converter control register (ADCR; 25H) and the option register (OPT1; 26H, OPT2; 27H).

The remaining space before the 60H are reserved for future expanded usage and reading these location will return the result 00H. The general-purpose data memory, addressed from 60H to 7FH, is used for data and control information under instruction command.

All data memory areas can handle arithmetic, logic, increment, decrement and rotate operations. Except for some dedicated bits, each bit in the data memory can be set and reset by the "SET [m].i" and "CLR [m].i" instruction, respectively. They are also indirectly accessible through memory pointer registers (MP0;01H, 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] access data memory pointed to by MP0 (01H) and MP1 (03H) respectively. Reading location 00H or 02H indirectly will return the result 00H. Writing indirectly results in no operation.

The function of data movement between two indirect addressing registers are not supported. The memory pointer registers, MP0 and MP1, are both 8-bit registers which can be used to access the data memory by combining corresponding indirect addressing registers.

MP0 only can be applied to data memory, while MP1 can be applied to data memory and LCD display memory.

**Accumulator**

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

### Arithmetic and Logic Unit – ALU

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

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

The ALU not only saves the results of a data operation but can change the status register.

### Status Register – STATUS

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

With the exception of the TO and PDF flags, bits in the status register can be altered by instructions like most other registers. Any data written into the status register will not change the TO or PDF flags. In addition it should be noted that operations related to the status register may give different results from those intended. The TO and PDF flags can only be changed by the watchdog timer overflow, system power-up, clearing the watchdog timer and executing the HALT instruction.

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

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

### Interrupts

The HT47C10L provides an internal timer/event counter interrupt and an internal time base interrupt. The interrupt control register (INTC;0BH) contains the interrupt control bits to set the enable/disable and interrupt request flags.

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

As an interrupt is serviced, a control transfer occurs by pushing the program counter onto the stack, followed by a branch to a subroutine at specified locations in the program memory. Only the program counter is pushed onto the stack. If the contents of the register and status register (STATUS) is altered by the interrupt service program which corrupts the desired control sequence, the contents must be saved first.

The internal timer/event counter interrupt is initialized by setting the timer/event counter interrupt request flag (TF; bit 4 of INTC), caused by a timer A or timer B overflow. When the interrupt is enabled, and the stack is not full and the TF bit is set, a subroutine call to location 04H will occur. The related interrupt request flag (TF) will be reset and the EMI bit cleared to disable further interrupts.

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

**STATUS (0AH) Register**

| Bit No. | Label | Function |
|---|---|---|
| 0 | EMI | Control the master or global interrupt (1=enabled; 0=disabled) |
| 1 | ETI | Control the timer/event counter interrupt (1=enabled; 0=disabled) |
| 2 | ETBI | Control the time base interrupt (1=enabled; 0=disabled) |
| 3 | — | Unused bit, read as "0" |
| 4 | TF | Timer/event counter interrupt request flag (1=active; 0=inactive) |
| 5 | TBF | Time base interrupt request flag (1=active; 0=inactive) |
| 6 | — | Unused bit, read as "0" |
| 7 | — | Unused bit, read as "0" |

**INTC (0BH) Register**

The time base interrupt is initialized by setting the time base interrupt request flag (TBF; bit 5 of INTC), caused by a regular time base signal. When the interrupt is enabled, and the stack is not full and the TBF bit is set, a subroutine call to location 08H will occur. The related interrupt request flag (TBF) will be reset and the EMI bit cleared to disable further interrupts.

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

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

| No. | Interrupt Source | Priority | Vector |
|---|---|---|---|
| a | Timer/event counter interrupt | 1 | 04H |
| b | Time base interrupt | 2 | 08H |

### Oscillator Configuration

The HT47C10L provides one built-in RC oscillator which frequency ($f_{OSC}$) is 32kHz or 128kHz decided by mask option. However, the CPU system clock ($f_{SYS}$) is always 32kHz. The HALT mode may stop the oscillator decided by software option. User should select 128kHz mask option for EL driving mode.

### Watchdog Timer – WDT

The clock source of the WDT ($f_S$) is $f_{SYS}$. The timer is designed to prevent a software malfunction or sequence jumping to an unknown location with unpredictable results. The Watchdog timer can be disabled by software option. If the Watchdog timer is disabled, all the executions related to the WDT result in no operation.

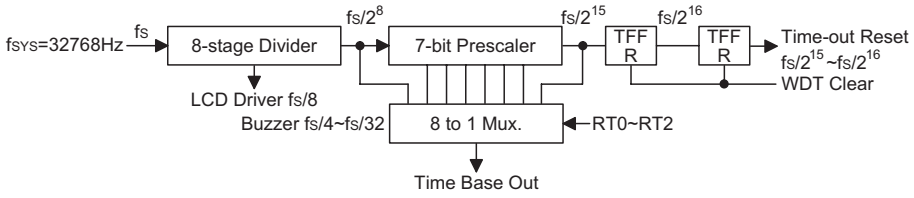
The "HALT" instruction is executed, WDT still counts if $f_{OSC}$ is on and can wake-up from HALT mode due to the WDT time-out.

The WDT overflow under normal operation will initialize "chip reset" and set the status bit TO. Whereas in the HALT mode, the overflow will initialize a "warm reset" only the Program Counter and SP are reset to zero. To clear the contents of WDT, three methods are adopted, external reset (a low level to $\overline{RES}$), software instruction, or a "HALT" instruction. The software instruction is "CLR WDT". Any execution of the "CLR WDT" instruction will clear the WDT. The WDT may reset the chip because of the time-out.

The WDT time-out period ranges from $f_S/2^{15}$~$f_S/2^{16}$. The "CLR WDT" instruction only clear the last two-stage of the WDT.

### Multi-function Timer

The HT47C10L provides a multi-function timer for the WDT and time base but with different time-out periods. The multi-function timer consists of an 8-stage divider and a 7-bit prescaler, with the clock source coming from $f_{SYS}$. The multi-function timer also provides a fixed frequency signal ($f_S/8$) for the LCD driver circuits, and buzzer output.




**Multi-function Timer**

**Time Base – TB**

The time base 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 (bits 2, 1, 0 of TBC;09H) yields various time-out periods. If a time base time-out occurs, the related interrupt request flag (TBF; bit 5 of INTC) is set. But if the interrupt is enabled, and the stack is not full, a subroutine call to location 08H occurs. When the "HALT" instruction is executed, the time base still works and can wake-up from HALT mode if $f_{OSC}$ is on. If the TBF is set 1 before entering the HALT mode, the wake-up function will be disabled.

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

**Power Down Operation – HALT**

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

- The $f_{OSC}$ and $f_{SYS}$ will still work or stop depend on STANDBY option (Option register bit 5), but T1 will turn off.
- The contents of the on-chip RAM and registers remain unchanged.
- The WDT will be cleared and recount again.
- All I/O ports maintain their original status.
- The PDF flag is set and the TO flag is cleared.
- LCD driver can be off or on depend on STANDBY option (Option register bit 5)
- The time base will stop or run depends on STANDBY option (Option register bit 5).

The port A wake-up and interrupt methods can be considered as a continuation of normal execution. Awakening from an I/O port stimulus, the program will resume execution of the next instruction. If awakening from an interrupt, two sequences may happen. If the related interrupt is disabled or the interrupt is enabled but the stack is full, the program will resume execution at the next instruction. If the interrupt is enabled and the stack is not full, a regular interrupt response takes place.

If an interrupt request flag is set to "1" before entering the HALT mode the wake-up function of the related interrupt will be disabled.

If the wake-up results from an interrupt acknowledgment, the actual interrupt subroutine execution will be delayed by more than one cycle. However, if the wake-up results in the next instruction execution, the execution will be performed immediately.

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

**Reset**

There are three ways in which a reset may occur.

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

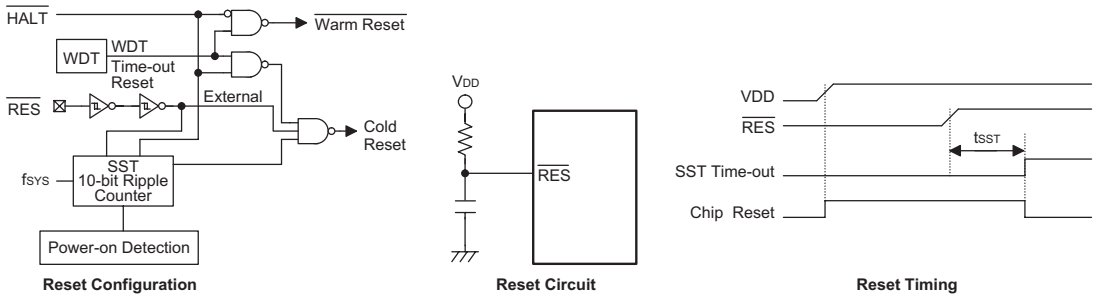
The WDT time-out during HALT mode is different from other chip reset conditions, since it can perform a warm reset that just resets the Program Counter and SP leaving the other circuits in their original state. Some registers remain unchanged during other reset conditions. Most registers are reset to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

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

Note: "u" means "unchanged"

The chip-reset status of the functional units are shown below.

| | |
|---|---|
| Program Counter | 000H |
| Interrupt | Disabled |
| Prescaler, Divider | Cleared |
| WDT, Time Base | Clear. After master reset, begin counting |
| Timer/event Counter | Off |
| Input/output Ports | Input mode |
| Stack Pointer | Points to the top of the stack |




**Reset Configuration** **Reset Circuit** **Reset Timing**

**Reset**

The states of the registers are summarized in the following table:

| Register | Reset (Power On) | WDT time-out (Normal Operation) | RES reset (Normal Operation) | RES reset (HALT) | WDT time-out (HALT) |
|---|---|---|---|---|---|
| TMRAH | xxxx xxxx | xxxx xxxx | xxxx xxxx | xxxx xxxx | uuuu uuuu |
| TMRAL | xxxx xxxx | xxxx xxxx | xxxx xxxx | xxxx xxxx | uuuu uuuu |
| TMRC | -000 1--- | -000 1--- | -000 1--- | -000 1--- | -uuu u--- |
| TMRBH | xxxx xxxx | xxxx xxxx | xxxx xxxx | xxxx xxxx | uuuu uuuu |
| TMRBL | xxxx xxxx | xxxx xxxx | xxxx xxxx | xxxx xxxx | uuuu uuuu |
| ADCR | ---x 0000 | ---x 0000 | ---x 0000 | ---x 0000 | ---u uuuu |
| Program Counter | 000H | 000H | 000H | 000H | 000H* |
| MP0 | xxxx xxxx | uuuu uuuu | uuuu uuuu | uuuu uuuu | uuuu uuuu |
| MP1 | xxxx xxxx | uuuu uuuu | uuuu uuuu | uuuu uuuu | uuuu uuuu |
| ACC | xxxx xxxx | uuuu uuuu | uuuu uuuu | uuuu uuuu | uuuu uuuu |
| TBLP | xxxx xxxx | uuuu uuuu | uuu uuuu | uuuu uuuu | uuuu uuuu |
| TBLH | xxxx xxxx | uuuu uuuu | uuuu uuuu | uuuu uuuu | uuuu uuuu |
| STATUS | --00 xxxx | --1u uuuu | --uu uuuu | --01 uuuu | --11 uuuu |
| INTC | --00 -000 | --00 -000 | --00 -000 | --00 -000 | --uu -uuu |
| TBC | ---- -111 | ---- -111 | ---- -111 | ---- -111 | ---- -uuu |
| PA | 1111 1111 | 1111 1111 | 1111 1111 | 1111 1111 | uuuu uuuu |
| PAC | 1111 1111 | 1111 1111 | 1111 1111 | 1111 1111 | uuuu uuuu |
| OPT1 | --00 0010 | --00 0010 | --00 0010 | --00 0010 | uuuu uuuu |
| OPT2 | 0000 0000 | 0000 0000 | 0000 0000 | 0000 0000 | uuuu uuuu |

Note: "*" refers to "warm reset"

"u" means "unchanged"

"x" means "unknown"

**Timer/Event Counter**

One 16-bit timer/event counter or RC type A/D converter is implemented in the HT47C10L. The ADC/$\overline{TM}$ bit (bit 1 of ADCR register) decides whether timer A and timer B are composed of one 16-bit timer/event counter or timer A and timer B are composed of RC type A/D converter.

The TMRAL, TMRAH, TMRBL, TMRBH composed of one 16-bit timer/event counter, when ADC/$\overline{TM}$ bit is "0". The TMRBL and TMRBH are timer/event counter preload registers for lower-order byte and higher-order byte respectively.

The timer/event counter clock source comes from system clock ($f_{SYS}$) or external source (A/D clock from pad:RCIN). The external clock input allows the user to count external events, count external RC type A/D clock, measure time intervals or pulse widths, or generate an accurate time base.

There are six registers related to the timer/event counter operating mode. TMRAH ([20H]), TMRAL ([21H]), TMRC ([22H]), TMRBH ([23H]), TMRBL ([24H]) and ADCR ([25H]). Writing to TMRBL only writes the data into a low byte buffer, and writing to TMRBH will write the data and the contents of the low byte buffer into the time/event counter preload register (16-bit) simultaneously. The timer/event counter preload register is changed by writing to TMRBH operations and writing to TMRBL will keep the timer/event counter preload register unchanged.

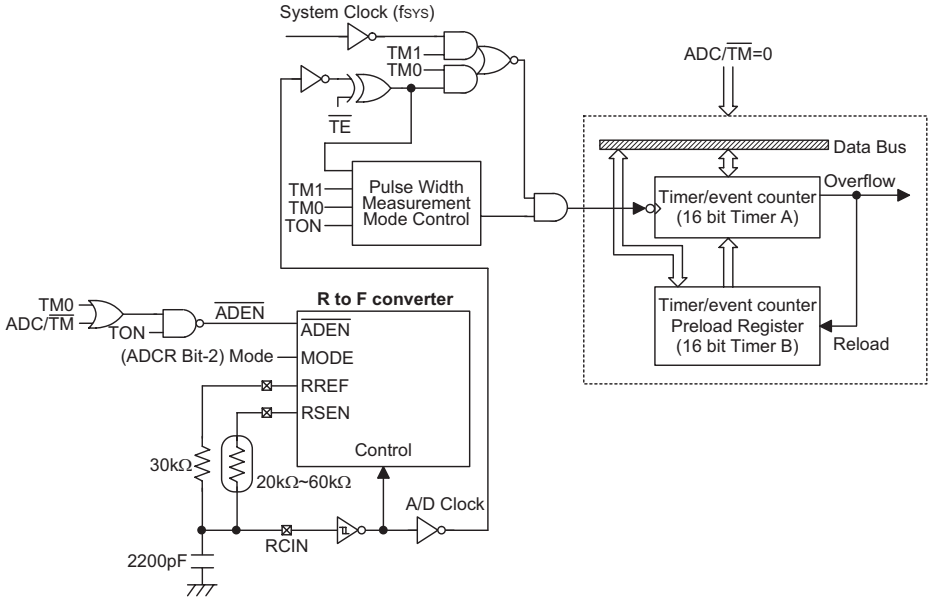
Reading TMRAH will also latch the TMRAL into the low byte buffer to avoid the false timing problem. Reading TMRAL returns the contents of the low byte buffer. In other words, the low byte of the timer/event counter can not be read directly. It must read the TMRAH first to make the low byte contents of timer/event counter be latched into the buffer.

The TMRC is the timer/event counter control register, which defines the timer/event counter options. The timer/event counter control register define the operating mode, counting enable or disable and active edge. Writing to timer B location puts the starting value in the timer/event counter preload register, while reading timer A yields the contents of the timer/event counter. Timer B is timer/event counter preload register.

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 (A/D clock) comes from an external (RCIN) pin. The timer mode functions as a normal timer with the clock source coming from the internal clock source ($f_{SYS}$). Finally, the pulse width measurement mode can be used to count the high or low level duration of the external signal (A/D clock from pad:RCIN). The counting is based on the system clock ($f_{SYS}$).

In the event count, A/D clock or internal timer mode, once the timer/event counter starts counting, it will count from the current contents in the timer/event counter (TMRAH and TMRAL) to FFFFH. Once overflow occurs, the counter is reloaded from the timer/event counter preload register (TMRBH and TMRBL) and generates the corresponding interrupt request flag (TF; bit 4 of INTC) at the same time.

In the pulse width measurement mode with the TON and TE bits equal to one, once the RCIN has received a transient from low to high (or high to low if the TE bit is 0) it will start counting until the A/D Clock returns to the original level and resets the TON. The measured result will remain in the timer/event counter even if the activated transient occurs again. In other words, only one cycle measurement can be done. Until setting the TON,




**Timer/Event Counter**

the cycle measurement will function again as long as it receives further transient pulse. Note that in this operation mode, the timer/event counter starts counting not according to the logic level but according to the transient edges. In the case of counter overflow, the counter is reloaded from the timer/event counter preload register and issues interrupt request just like the other two modes.

To enable the counting operation, the timer on bit (TON; bit 4 of TMRC) should be set to 1. In the pulse width measurement mode, the TON will automatically be cleared after the measurement cycle is completed. But in the other two modes, the TON can only be reset by instructions.

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 turns On, data written to the timer/event counter preload register is kept only in the timer/event counter preload register. The timer/event counter will still operate until overflow occurs.

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

It is strongly recommended to load first the desired value into TMRBL, TMRBH, TMRAL, and TMRAH registers then turn on the related timer/event counter for proper operation. Because the initial value of TMRBL, TMRBH, TMRAL and TMRAH are unknown.

| 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 (0= disabled; 1= enabled) |
| 5<br>6 | TM0<br>TM1 | To define the operating mode (TM1, TM0)<br>10= Timer mode (Internal clock: $f_{SYS}$)<br>01= Event counter mode (External clock: A/D clock from pad RCIN)<br>11= Pulse width measurement mode (RCIN, $f_{SYS}$)<br>00= Unused |
| 7 | — | Unused bit, read as ″0″ |

**TMRC (22H) Register**

Example for Timer/event counter mode (disable interrupt):

```
clr tmrc
clr adcr.1                      ; set timer mode
clr intc.4                      ; clear timer/event counter interrupt request flag
mov a, low (65536-1000)         ; give timer initial value
mov tmrbl, a                    ; count 1000 time and then overflow
mov a, high (65536-1000)
mov tmrbh, a

mov a, 01010000b                ; timer clock source=fSYS and timer on
mov tmrc, a

p10:
clr wdt
snz intc.4                      ; polling timer/event counter interrupt request flag
jmp p10
clr intc.4                      ; clear timer/event counter interrupt request flag
                                ; program continue
```

**RC Type A/D Converter**

RC type A/D converter is implemented in the HT47C10L. The A/D converter contains two 16-bit programmable count-up counters and the timer A clock source comes from the system clock ($f_{SYS}$=32kHz). The timer B clock source comes from the external RC oscillator. The TMRAL, TMRAH, TMRBL, TMRBH are composed of the A/D converter when ADC/$\overline{TM}$ bit (bit 1 of ADCR register) is ″1″.

The A/D converter timer B clock source may come from RREF~RCIN oscillation, RSEN~RCIN oscillation or RCIN external clock input. The timer A clock source is the system clock by setting (TM1, TM0=1, 0).

There are six registers related to the A/D converter, i.e., TMRAH, TMRAL, TMRC, TMRBH, TMRBL and ADCR. The internal timer clock is input to TMRAH and TMRAL, the A/D clock is input to TMRBH and TMRBL. The OVB/$\overline{OVA}$ bit (bit 0 of ADCR register) decides whether timer A overflows or timer B overflows, then the TF bit is set and timer interrupt occurs. When the A/D converter mode timer A or timer B overflows, the TON bit is reset and stop counting. Writing TMRAH/TMRBH makes the starting value be placed in the timer A/timer B and reading TMRAH/TMRBH gets the contents of the timer A/timer B. Writing TMRAL/TMRBL only writes the data into a low byte buffer, and writing TMRAH/TMRBH will write the data and the contents of the low byte buffer into the timer A/timer B (16-bit) simultaneously. The timer A/timer B is changed by writing TMRAH/TMRBH operations and writing TMRAL/TMRBL will keep timer A/timer B unchanged.

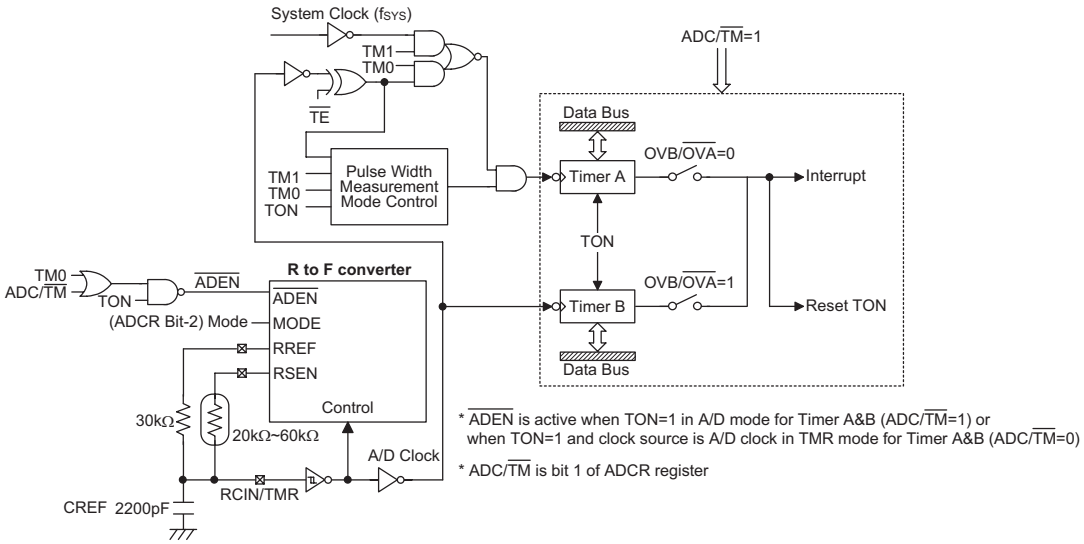
Reading TMRAH/TMRBH will also latch the TMRAL/TMRBL into the low byte buffer to avoid the false timing problem. Reading TMRAL/TMRBL returns the contents of the low byte buffer. In other word, the low byte of timer A/timer B can not be read directly. It must read the TMRAH/TMRBH first to make the low byte contents of timer A/timer B be latched into the buffer.

The bit2 of ADCR decides which resistor and capacitor compose an oscillation circuit and input to TMRBH and TMRBL.

The TM0 and TM1 bits of TMRC define the clock source of timer A. It is suggested that the clock source of timer A use the system clock.

The TON bit (bit 4 of TMRC) is set ″1″ the timer A and timer B will start counting until timer A or timer B overflows, the timer/event counter generates the interrupt request flag (TF ; bit 4 of INTC) and the timer A and timer B stop counting and reset the TON bit to ″0″ at the same time.

If the TON bit is ″1″, the TMRAH, TMRAL, TMRBH and TMRBL cannot be read or written to. Only when the timer/event counter is off and when the instruction ″MOV″ is used could those four registers be read or written to.




**RC Type A/D Converter**

| Bit No. | Label | Function |
|---|---|---|
| 0 | OVB/$\overline{OVA}$ | In the RC type A/D converter mode, this bit is used to define the timer/event counter interrupt which comes from timer A overflow or timer B overflow.<br>(0= timer A overflow; 1= timer B overflow)<br>In the timer/event counter mode, this bit is void. |
| 1 | ADC/$\overline{TM}$ | To define 16-bit timer/event counter or RC type A/D converter is enable.<br>(0= timer/event counter enable; 1= A/D converter is enable) |
| 2 | MODE | To define the A/D converter operating mode<br>0= RREF~CREF oscillation (reference resistor and reference capacitor)<br>1= RSEN~CREF oscillation (resistor sensor and reference capacitor) |
| 3 | BON | Low voltage detector disable/enable (0=disable; 1=enable) |
| 4 | BLF | Low voltage flag (0=battery power good; 1=battery low) |
| 5~7 | — | Unused bit, read as ″0″ |

**ADCR (25H) Register**

Example for RC type AD converter mode (Timer A overflow):

```
clr tmrc
clr adcr.1                          ; set timer mode
clr intc.4                          ; clear timers/event counter interrupt request flag
mov a, low (65536-1000)             ; give timer A initial value
mov tmrbl, a                        ; count 1000 time and then overflow
mov a, high (65536-1000)
mov tmrbh, a

mov a, 00000010b                    ; RREF~CREF; set RC type ADC mode; set Timer A overflow
mov  adcr, a
mov a, 00h                          ; give timer B initial value
mov tmrbl, a
mov a, 00h
mov tmrbh, a

mov a, 01010000b                    ; timer A clock source=fSYS and timer on
mov tmrc, a

p10:
clr wdt
snz intc.4                          ; polling timer/event counter interrupt request flag
jmp p10

clr intc.4                          ; clear timer/event counter interrupt request flag
                                    ; program continue
```

Example for RC type AD converter mode (Timer B overflow):

```
clr tmrc
clr adcr.1                      ; set timer mode
clr intc.4                      ; clear timer/event counter interrupt request flag
mov a, 00h                      ; give timer A initial value
mov tmrbl, a
mov a, 00h
mov tmrbh, a

mov a, 00000011b                ; RREF~CREF; set RC type ADC mode; set Timer B overflow
mov adcr,a

mov a, low (65536-1000)         ; give timer B initial value
mov tmrbl, a                    ; count 1000 time and then overflow
mov a, high (65536-1000)
mov tmrbh, a

mov a, 00110000b                ; timer A clock source=fSYS and timer on
mov tmrc, a

p10:
clr wdt
snz intc.4                      ; polling timer/event counter interrupt request flag
jmp p10

clr intc.4                      ; clear timer/event counter interrupt request flag
                                ; program continue
```

**Input/Output Ports**

There is 8-bit bidirectional input/output port in the microcontroller, labeled PA which is mapped to the data memory of [12H]. All of these I/O lines can be used as input and output operations. For the input operation, these lines are non-latching, that is, the inputs must be ready at the T2 rising edge of instruction "MOV A, [m]" (m=12H). For output operation, all the data is latched and remain unchanged until the output latch is rewritten.

Each I/O line has its own control register (PAC) to control the input/output configuration. With this control register, CMOS output or Schmitt trigger input with pull-high resistor structures can be reconfigured dynamically (i.e., on-the fly) under software control. To function as an input, the corresponding latch of the control register has to be set as "1". The pull-high resistor will be exhibited automatically. The input sources also depend on the control register. If the control register bit is "1", the input will read the pad state ("mov" and readmodify-write instructions). If the control register bit is "0", the contents of the latches will move to internal data bus ("mov" and read-modify-write instructions). The input paths (pad state or latches) of read-modify-write instructions are dependent on the control register bits. For output function, CMOS is the only configuration. This control register is mapped to locations 13H.

After chip reset, these input/output lines stay at high levels (pull-high). Each bit of these input/output latches can be set or cleared by "SET [m].i" (m=12H) instructions. Some instructions first input data and then follow the output operations. For example, "SET [m].i", "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 bit of the port A has the capability of waking-up the device.

The PA0 and PA1 are pin-shared with BZ and $\overline{BZ}$, respectively. If the BZ mode is selected, the output signal in output mode of PA0 (or PA1) will be BZ (or $\overline{BZ}$) signal. The input mode always remains its original functions. The 4kHz buzzer output signals (in output mode) are controlled by the PA0 and PA1 data registers. The truth table of PA0/BZ and PA1/$\overline{BZ}$ are listed below.

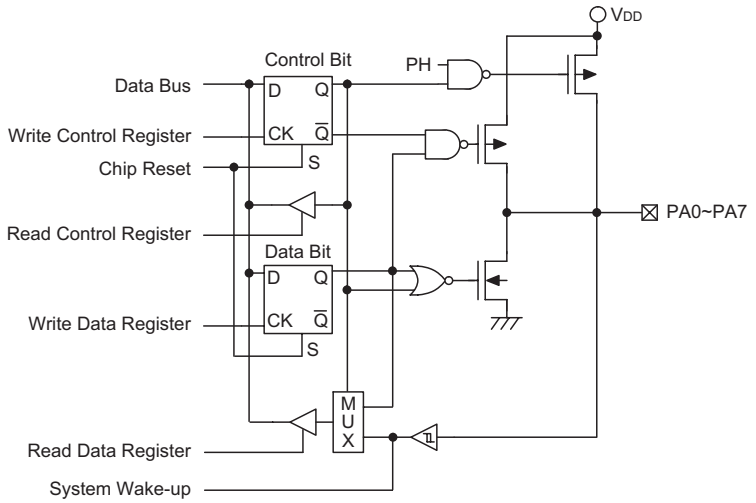
| PA1 Data Register | PA0 Data Register | PA1, PA0 Pad Function |
|---|---|---|
| 0 (CLR PA.1) | 0 (CLR PA.0) | PA0=BZ, PA1=$\overline{BZ}$ |
| 1 (SET PA.1) | 0 (CLR PA.0) | PA0=BZ, PA1=0 |
| X | 1 (SET PA.0) | PA0=0, PA1=0 |

**OPTION Register: BZ Mode Enable**

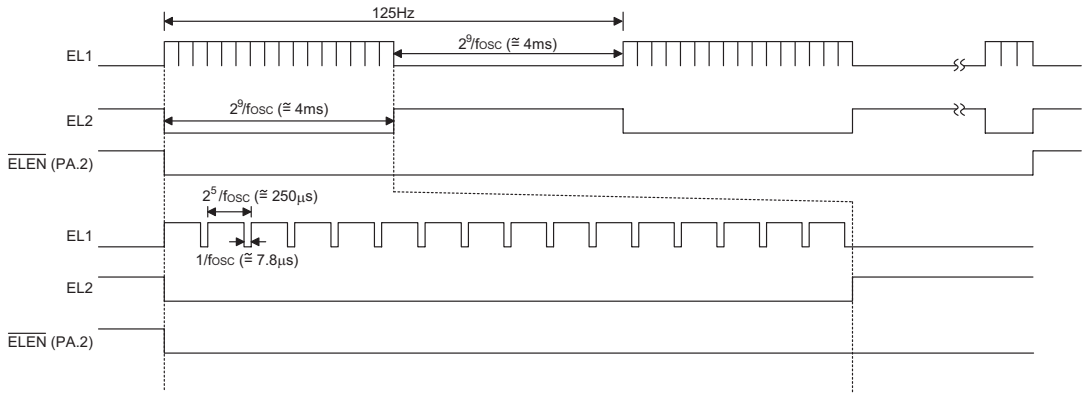
The PA2 and PA3 are pin-shared with EL1 and EL2 signals, respectively. If the EL mode is selected, the output signal in output mode of PA2 (or PA3) will be the EL1 (or EL2) signal. The input mode always remains its original functions. The EL output signals (in output mode) are controlled by the PA2 data register only. The truth table of PA2/EL1 and PA3/EL2 are listed below.

| PA3 Data Register | PA2 Data Register | PA3, PA2 Pad Function |
|---|---|---|
| 0 or 1 | 0 (CLR PA.2) | PA2=EL1, PA3=EL2 |
| 0 or 1 | 1 (SET PA.2) | PA2=0, PA3=1 |

**OPTION Register: EL Mode Enable**




Note: BZ mode and EL mode functions are not shown in this diagram

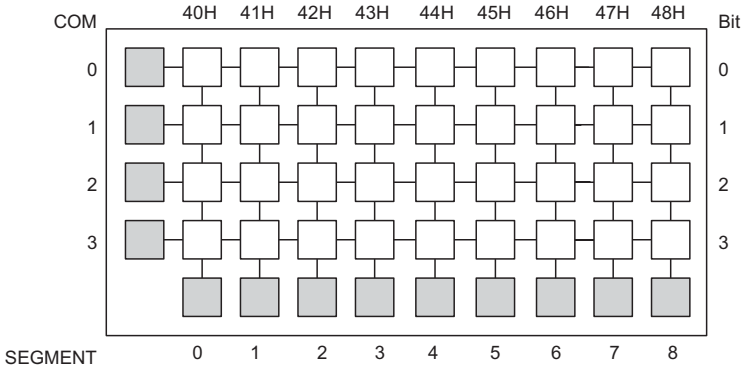



**EL Timing**

**LCD Display Memory**

The HT47C10L provides an area of embedded data memory for LCD display. The LCD display memory is designed into 9×4 bits. This area is located from 40H to 48H of the RAM at Bank 1. Bank pointer (BP; located at 04H of the data memory) is the switch between the general data memory and the LCD display memory. When the BP is set "1" any data written into 40H~48H will effect the LCD display (indirect addressing mode using MP1). When the BP is cleared "0", any data written into 40H~48H has to access the general-purpose data memory. The LCD display memory can be read and written 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 HT47C10L.




**Display Memory (Bank 1)**

### LCD Driver Output

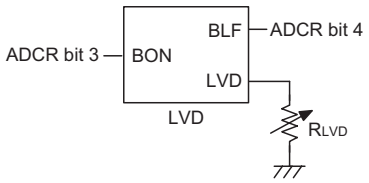
The output number of the HT47C10L LCD driver is 9×4 (1/4 duty). The bias type LCD driver is ″C″ type (1/2 bias). A capacitor has to be connected between C1 and C2.

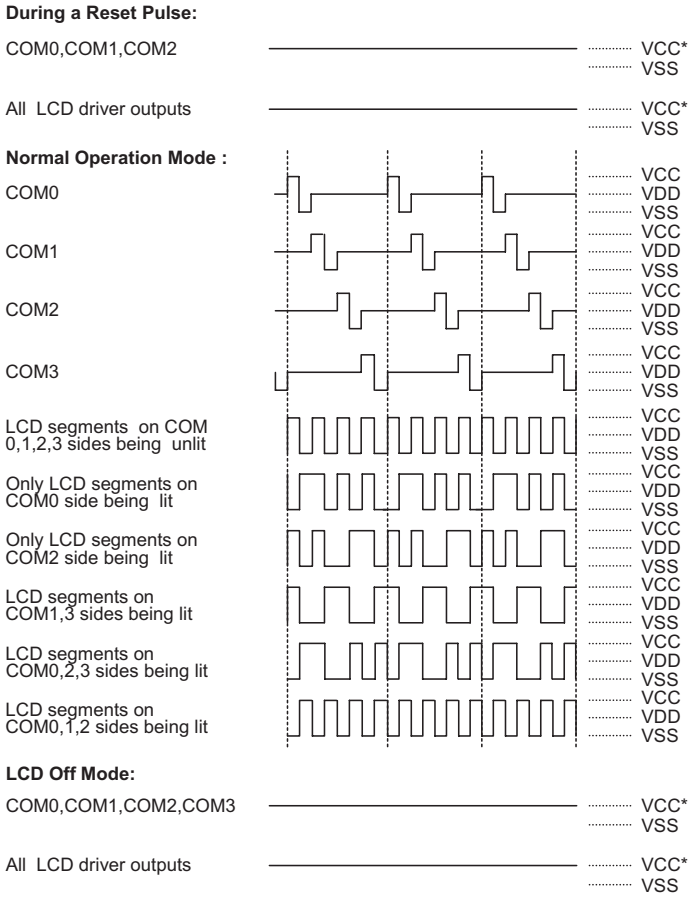
LCD driver on/off at HALT depends on STANDBY option (Option register bit 5)

### Low Voltage Detect – LVD

The HT47C10L provides a low voltage detector for battery system application. If the LVD is on and the battery voltage is lower than the specified value, the low voltage flag (BLF; bit 4 of ADCR register) is set. The specified value may be set as 1.3V±0.05V by changing suitable external $R_{LVD}$ for a same lot. The low voltage detector circuit can be turn On or Off by writing a ″1″ or a ″0″ to BON (bit 3 of ADCR register). The BLF is invalid when the BON is cleared as ″0″.

Set BON=0 after checking the voltage to prevent from DC current consumption of LVD.




**LCD Driver Output (1/4 Duty, 1/2 Bias)**

Note: ″VCC″ is ≅ $2V_{DD}$ at normal operation mode.

″VCC*″ is ≅ $V_{DD}$–0.2V with LCD off or reset.

**Option Register**

The following shows many kinds of mask options in the HT47C10L. All these options should be defined in order to ensure proper system functioning.

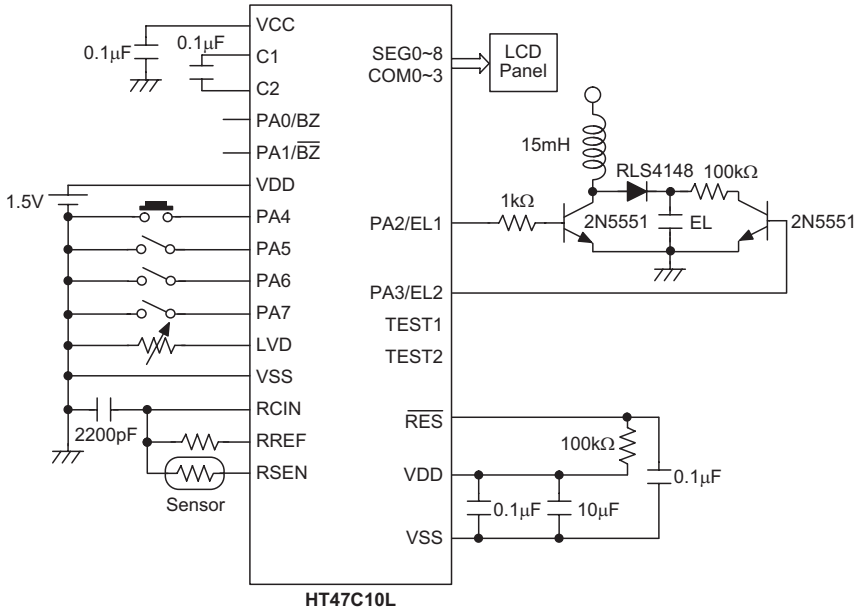
| Bit No. | Label | Function | Reset State |
|---|---|---|---|
| 0 | $\overline{\text{WDTEN}}$ | WDT enable or disable selection (0: enable; 1: disable) | 0 |
| 1<br>2 | BZFREQ0,<br>BZFREQ1 | Buzzer output frequency selection<br>BZFREQ1~BZFREQ0<br>00: $f_{SYS}/2^2$<br>01: $f_{SYS}/2^3$<br>10: $f_{SYS}/2^4$<br>11: $f_{SYS}/2^5$ | 01 |
| 3 | BZMODE | To define the PA0 and PA1 output function<br>0=Normal output<br>1=Buzzer output. PA0 is BZ output, PA1 is $\overline{\text{BZ}}$ output. | 0 |
| 4 | ELMODE | To define the PA2 and PA3 output function<br>0=Normal output<br>1=EL output. PA2 is EL1 output, PA3 is EL2 output. | 0 |
| 5 | STANDBY | Oscillator/LCD are on or off when CPU HALT<br>0=Oscillator/LCD is off at HALT<br>1=Oscillator/LCD is on at HALT | 0 |
| 6~7 | — | Unused bit, read as "0" | 00 |

**OPT1 (26H) Register**

| Bit No. | Label | Function | Reset State |
|---|---|---|---|
| 0~7 | $\overline{\text{PH}}$ | PA0~PA7 pull-high option in input mode (0: enable; 1: disable) | 00H |

**OPT2 (27H) Register**

## Application Circuits



## Instruction Set Summary

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

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

Note: x: Immediate data

m: Data memory address

A: Accumulator

i: 0~7 number of bits

addr: Program memory address

√: Flag is affected

–: Flag is not affected

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

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

(3): (1) and (2)

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

## Instruction Definition

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

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

Operation ACC ← ACC+[m]+C

Affected flag(s)

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

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

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

Operation [m] ← ACC+[m]+C

Affected flag(s)

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

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

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

Operation ACC ← ACC+[m]

Affected flag(s)

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

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

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

Operation ACC ← ACC+x

Affected flag(s)

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

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

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

Operation [m] ← ACC+[m]

Affected flag(s)

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

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

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

Operation ACC ← ACC ″AND″ [m]

Affected flag(s)

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

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

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

Operation ACC ← ACC ″AND″ x

Affected flag(s)

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

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

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

Operation [m] ← ACC ″AND″ [m]

Affected flag(s)

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

**CALL addr** Subroutine call

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

Operation Stack ← Program Counter+1
Program Counter ← addr

Affected flag(s)

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

**CLR [m]** Clear data memory

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

Operation [m] ← 00H

Affected flag(s)

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

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

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

Operation [m].i ← 0

Affected flag(s)

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

**CLR WDT** Clear Watchdog Timer

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

Operation WDT ← 00H
PDF and TO ← 0

Affected flag(s)

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

**CLR WDT1** Preclear Watchdog Timer

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

Operation WDT ← 00H*
PDF and TO ← 0*

Affected flag(s)

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

**CLR WDT2** Preclear Watchdog Timer

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

Operation WDT ← 00H*
PDF and TO ← 0*

Affected flag(s)

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

**CPL [m]** Complement data memory

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

Operation [m] ← $[\overline{m}]$

Affected flag(s)

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

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

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

Operation ACC ← $\overline{[m]}$

Affected flag(s)

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

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

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

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

Affected flag(s)

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

**DEC [m]** Decrement data memory

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

Operation [m] ← [m]−1

Affected flag(s)

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

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

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

Operation ACC ← [m]−1

Affected flag(s)

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

**HALT** Enter power down mode

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

Operation Program Counter ← Program Counter+1
PDF ← 1
TO ← 0

Affected flag(s)

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

**INC [m]** Increment data memory

Description Data in the specified data memory is incremented by 1

Operation [m] ← [m]+1

Affected flag(s)

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

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

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

Operation ACC ← [m]+1

Affected flag(s)

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

**JMP addr** Directly jump

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

Operation Program Counter ←addr

Affected flag(s)

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

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

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

Operation ACC ← [m]

Affected flag(s)

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

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

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

Operation ACC ← x

Affected flag(s)

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

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

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

Operation [m] ←ACC

Affected flag(s)

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

**NOP** No operation

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

Operation Program Counter ← Program Counter+1

Affected flag(s)

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

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

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

Operation ACC ← ACC ″OR″ [m]

Affected flag(s)

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

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

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

Operation ACC ← ACC ″OR″ x

Affected flag(s)

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

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

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

Operation [m] ←ACC ″OR″ [m]

Affected flag(s)

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

**RET** Return from subroutine

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

Operation Program Counter ← Stack

Affected flag(s)

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

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

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

Operation Program Counter ← Stack
ACC ← x

Affected flag(s)

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

**RETI** Return from interrupt

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

Operation Program Counter ← Stack
EMI ← 1

Affected flag(s)

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

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

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

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

Affected flag(s)

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

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

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

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

Affected flag(s)

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

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

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

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

Affected flag(s)

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

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

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

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

Affected flag(s)

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

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

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

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

Affected flag(s)

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

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

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

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

Affected flag(s)

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

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

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

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

Affected flag(s)

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

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

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

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

Affected flag(s)

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

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

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

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

Affected flag(s)

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

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

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

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

Affected flag(s)

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

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

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

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

Affected flag(s)

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

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

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

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

Affected flag(s)

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

**SET [m]** Set data memory

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

Operation [m] ← FFH

Affected flag(s)

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

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

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

Operation [m].i ← 1

Affected flag(s)

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

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

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

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

Affected flag(s)

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

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

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

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

Affected flag(s)

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

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

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

Operation Skip if [m].i≠0

Affected flag(s)

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

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

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

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

Affected flag(s)

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

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

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

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

Affected flag(s)

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

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

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

Operation ACC ← ACC+$\overline{x}$+1

Affected flag(s)

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

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

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

Operation [m].3~[m].0 ↔ [m].7~[m].4

Affected flag(s)

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

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

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

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

Affected flag(s)

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

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

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

Operation Skip if [m]=0

Affected flag(s)

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

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

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

Operation Skip if [m]=0

Affected flag(s)

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

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

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

Operation Skip if [m].i=0

Affected flag(s)

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

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

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

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

Affected flag(s)

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

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

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

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

Affected flag(s)

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

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

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

Operation ACC ← ACC ″XOR″ [m]

Affected flag(s)

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

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

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

Operation [m] ← ACC ″XOR″ [m]

Affected flag(s)

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

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

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

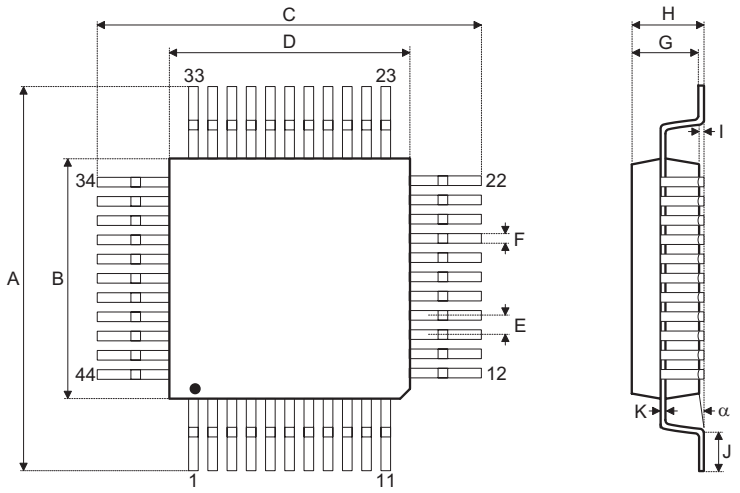
Operation ACC ← ACC ″XOR″ x

Affected flag(s)

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

## Package Information

**44-pin QFP (10×10) Outline Dimensions**




| Symbol | Dimensions in mm | | |
|---|---|---|---|
| | Min. | Nom. | Max. |
| A | 13 | — | 13.40 |
| B | 9.90 | — | 10.10 |
| C | 13 | — | 13.40 |
| D | 9.90 | — | 10.10 |
| E | — | 0.80 | — |
| F | — | 0.30 | — |
| G | 1.90 | — | 2.20 |
| H | — | — | 2.70 |
| I | — | 0.10 | — |
| J | 0.73 | — | 0.93 |
| K | 0.10 | — | 0.20 |
| α | 0° | — | 7° |

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

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

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

**Holtek Semiconductor Inc. (Shenzhen Sales Office)**
43F, SEG Plaza, Shen Nan Zhong Road, Shenzhen, China 518031
Tel: 0755-8346-5589
Fax: 0755-8346-5590
ISDN: 0755-8346-5591

**Holtek Semiconductor Inc. (Beijing Sales Office)**
Suite 1721, Jinyu Tower, A129 West Xuan Wu Men Street, Xicheng District, Beijing, China 100031
Tel: 010-6641-0030, 6641-7751, 6641-7752
Fax: 010-6641-0125

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