# HT48RA0
# 8-Bit Microcontroller

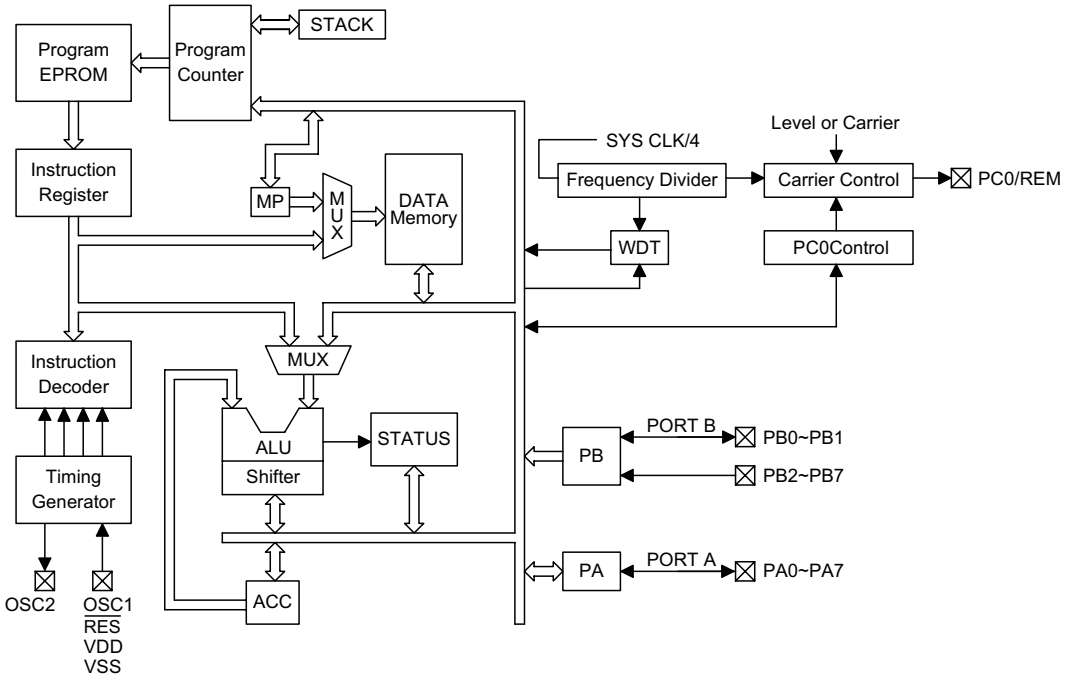
## Features

- Operating voltage: 2.9V~5.2V
- Ten bidirectional I/O lines
- Six schmitt trigger input lines
- One carrier output (1/2 or 1/3 duty)
- On-chip crystal and RC oscillator
- Watchdog timer
- 1K×14 program EPROM
- 32×8 data RAM
- Low voltage reset function
- Halt function and wake-up feature reduce power consumption
- 62 powerful instructions
- Up to 1μs instruction cycle with 4MHz system clock
- All instructions in 1 or 2 machine cycles
- 14-bit table read instructions
- One-level subroutine nesting
- Bit manipulation instructions
- 20-pin DIP-A/SOP-A package
  24-pin SOP-A package

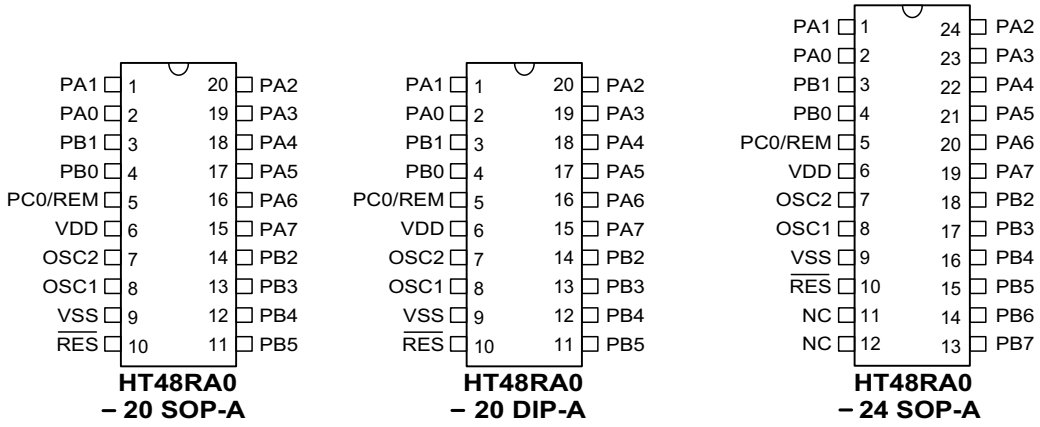
## General Description

The HT48RA0 is an 8-bit high performance RISC-like microcontroller specifically designed for multiple I/O product applications. The device is particularly suitable for use in products such as remote controllers, fan/light controllers, washing machine controllers, scales, toys and various subsystem controllers. A halt feature is included to reduce power consumption.

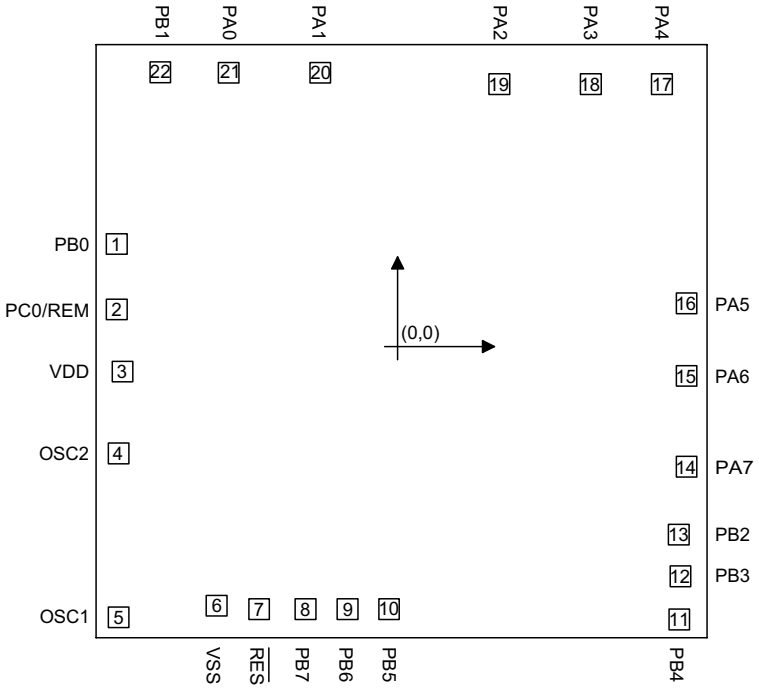
## Block Diagram



## Pin Assignment




## Pad Assignment




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

## Pad Description

| Pad Name | I/O | Code Option | Function |
|---|---|---|---|
| PB0, PB1 | I/O | Wake-up or none | 2-bit bidirectional input/output lines with pull-high resistors. Each bit can be determined as NMOS output or schmitt trigger input by software instructions. Each bit can also be configured as wake-up input by code option. |
| PC0/REM | O | Level or carrier | Level or carrier output pin<br>PC0 can be set as CMOS output pin or carrier output pin by code option. |
| VDD | — | — | Positive power supply |
| OSC2<br>OSC1 | I<br>O | Crystal or RC | OSC1, OSC2 are connected to an RC network or a crystal (determined by code option) for the internal system clock. In the case of RC operation, OSC2 is the output terminal for 1/4 system clock (NMOS open drain output). |
| VSS | — | — | Negative power supply, GND |
| $\overline{RES}$ | I | — | Schmitt trigger reset input. Active low. |
| PB2~PB7 | I | Wake-up or none | 6-bit schmitt trigger input lines with pull-high resistors. Each bit can be configured as a wake-up input by code option. |
| 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. |

## Absolute Maximum Ratings

Supply Voltage.....–0.3V to 5.5V

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

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

Operating Temperature .....–25°C to 70°C

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

## D.C. Characteristics

Ta=25°C

| Symbol | Parameter | Test Conditions | | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|---|---|
| | | $V_{DD}$ | Conditions | | | | |
| $V_{DD}$ | Operating Voltage | — | $f_{SYS}$=2MHz | $V_{LVR}$ | — | 5.2 | V |
| $V_{DD}$ | Operating Voltage | — | $f_{SYS}$=4MHz | 3 | — | 5.2 | V |
| $I_{DD}$ | Operating Current | 3V | No load, $f_{SYS}$=4MHz | — | 0.7 | 1.5 | mA |
| $I_{STB}$ | Standby Current | 3V | No load, system halt | — | — | 1 | μA |
| $V_{IL1}$ | Input Low Voltage for I/O Ports | 3V | — | 0 | — | 1.05 | V |
| $V_{IH1}$ | Input High Voltage for I/O Ports | 3V | — | 1.95 | — | 3 | V |
| $V_{IL2}$ | Input Low Voltage ($\overline{RES}$) | 3V | — | — | 1.5 | — | V |
| $V_{IH2}$ | Input High Voltage ($\overline{RES}$) | 3V | — | — | 2.4 | — | V |
| $I_{OL}$ | I/O Ports Sink Current | 3V | $V_{OL}$=0.3V | 1.5 | 2.5 | — | mA |
| $I_{OH}$ | I/O Ports Source Current | 3V | $V_{OH}$=2.7V | –1 | –1.5 | — | mA |
| $R_{PH1}$ | Pull-high Resistance of PA Port, PB0~PB1 and $\overline{RES}$ | 3V | — | — | 60 | — | kΩ |
| $R_{PH2}$ | Pull-high Resistance of PB2~PB7 | 3V | — | — | 60 | — | kΩ |
| $V_{LVR}$ | Low Voltage Reset | 3V | — | 2.3 | 2.6 | 2.9 | V |

## A.C. Characteristics

Ta=25°C

| Symbol | Parameter | Test Conditions | | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|---|---|
| | | $V_{DD}$ | Conditions | | | | |
| $f_{SYS}$ | System Clock | 3V | — | 400 | — | 4000 | kHz |
| $t_{RES}$ | External Reset Low Pulse Width | — | — | 1 | — | — | μs |
| $t_{SST}$ | System Start-up timer Period | — | Power-up or wake-up from halt | — | 1024 | — | $t_{SYS}$ |

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

## Functional Description

### Execution flow

The HT48RA0 system clock can be derived from a crystal/ceramic resonator 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. However, the pipelining scheme causes each instruction to effectively execute within 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 program EPROM 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 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.

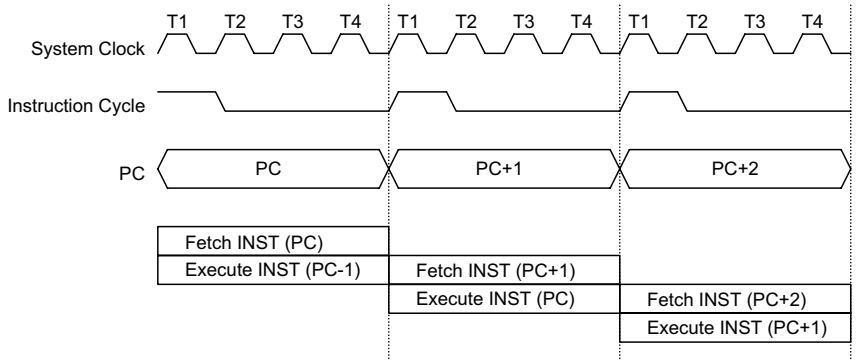
### Program memory – EPROM

The program memory is used to store the program instructions which are to be executed. It also contains data and table and is organized into 1024×14 bits, addressed by the program counter and table pointer.

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

- Location 000H

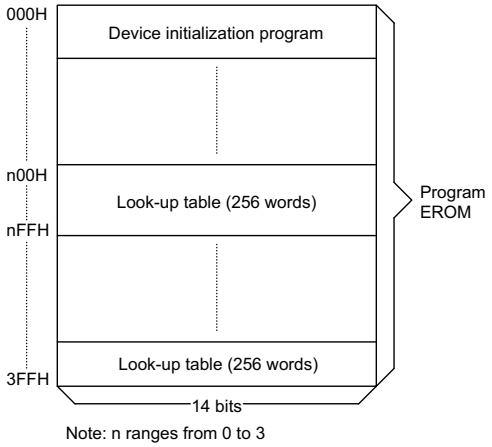
  This area is reserved for the initialization program. After chip reset, the program always begins execution at location 000H.




Execution flow

• Table location

Any location in the EPROM 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 other bits of the table word are transferred to the lower portion of TBLH, the remaining 2 bits are read as "0". The Table Higher-order byte register (TBLH) is read only. The table pointer (TBLP) is a read/write register (07H), where P indicates the table location. Before accessing the table, the location must be placed in TBLP. The TBLH is read only and cannot be restored. All table related instructions need 2 cycles to complete the operation. These areas may function as normal program memory depending upon the requirements.




Program memory

### Stack register – STACK

This is a special part of the memory used to save the contents of the program counter (PC) only. The stack is organized into one level 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 the contents of the program counter are pushed onto the stack. At the end of a subroutine signaled by a return instruction (RET), 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 "CALL" is subsequently executed, stack overflow occurs and the first entry will be lost (only the most recent return address is stored).

| 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 |
| Skip | **PC+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

### Data memory – RAM

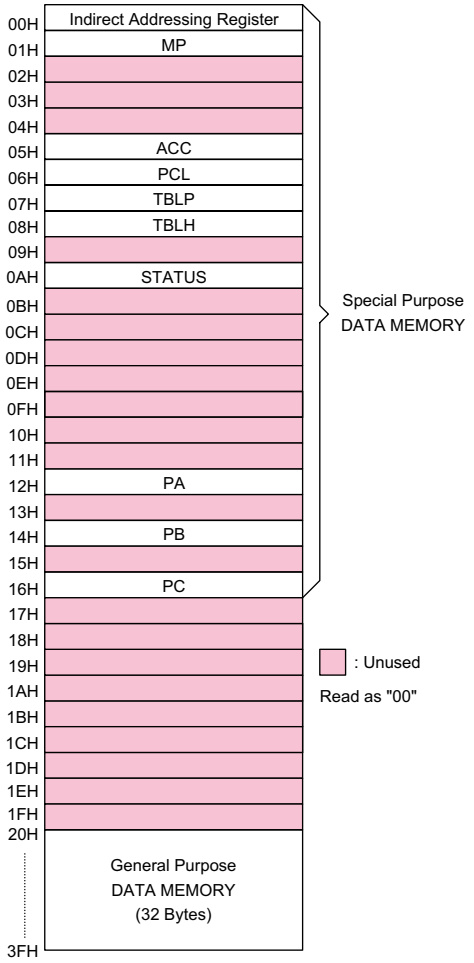
The data memory is designed with 42×8 bits. The data memory is divided into two functional groups: special function registers and general purpose data memory (32×8). Most of them are read/write, but some are read only.

The special function registers include the indirect addressing register (00H), the memory pointer register (MP;01H), 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 status register (STATUS;0AH) and the I/O registers (PA;12H, PB;14H, PC;16H). The remaining space before the 20H is reserved for future expanded usage and reading these locations will return the result 00H. The general purpose data memory, addressed from 20H to 3FH, is used for data and control information under instruction command.

All data memory areas can handle arithmetic, logic, increment, decrement and rotate operations directly. Except for some dedicated bits, each bit in the data memory can be set and reset by the SET [m].i and CLR [m].i instructions, respectively. They are also indirectly accessible through memory pointer register (MP;01H).

### Indirect addressing register

Location 00H is an indirect addressing register that is not physically implemented. Any read/write operation of [00H] accesses data memory pointed to by MP (01H). Reading loca-




RAM mapping

| 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: Table location bits

@7~@0: Table pointer bits

P9~P8: Current program counter bits

tion 00H itself indirectly will return the result 00H. Writing indirectly results in no operation.

The memory pointer register MP (01H) is a 6-bit register. The bit 7~6 of MP is undefined and reading will return the result "1". Any writing operation to MP will only transfer the lower 6-bit data to MP.

### Accumulator

The accumulator closely relates to ALU operations. It is also mapped to location 05H of the data memory and is capable of carrying out immediate data operations. Data movement between two data memory locations has to 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 also changes the contents of the status register.

### Status register – STATUS

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

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

| Labels | Bits | Function |
|---|---|---|
| C | 0 | 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. |
| AC | 1 | 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. |
| Z | 2 | Z is set if the result of an arithmetic or logic operation is zero; otherwise Z is cleared. |
| OV | 3 | OV is set if the operation results in a carry into the highest-order bit but not a carry out of the highest-order bit, or vice versa; otherwise OV is cleared. |
| PD | 4 | PD is cleared when either a system power-up or executing the CLR WDT instruction. PD is set by executing the HALT instruction. |
| TO | 5 | 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 | Undefined, read as "0" |
| — | 7 | Undefined, read as "0" |

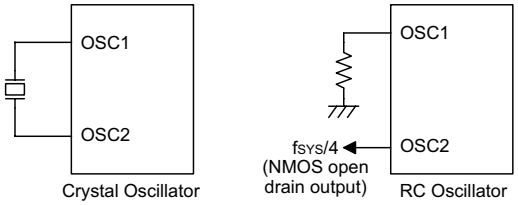
Status register

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

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

### Oscillator configuration

There are two oscillator circuits in the HT48RA0.




System oscillator

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

If an RC oscillator is used, an external resistor between OSC1 and VSS in needed and the resistance must range from 51kΩ to 1MΩ. The system clock, divided by 4, is available on OSC2, which can be used to synchronize external logic. The RC oscillator provides the most cost effective solution. However, the frequency of the oscillation may vary with $V_{DD}$, temperature and the chip itself due to process variations. It is, therefore, not suitable for timing sensitive operations where accurate oscillator frequency is desired.

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

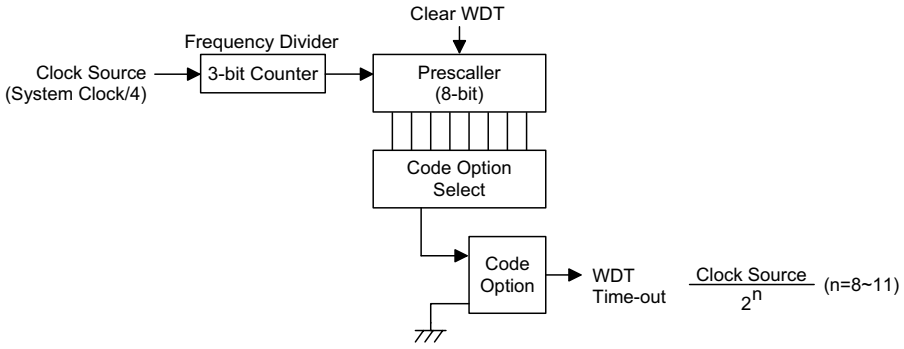
### Watchdog timer – WDT

The clock source of the WDT is implemented by instruction clock (system clock divided by 4). The clock source is processed by a frequency divider and a prescaller to yield various time out periods.

$$\text{WDT time out period} = \frac{\text{Clock Source}}{2^n}$$

Where n= 8~11 selected by code option.

This 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 code option. If the Watchdog Timer is disabled, all the executions related to the WDT result in no opera-




Watchdog timer

tion and the WDT will lose its protection purpose. In this situation the logic can only be restarted by an external logic.

A WDT overflow under normal operation will initialize "chip reset" and set the status bit "TO". To clear the contents of the WDT prescaler, three methods are adopted; external reset (a low level to $\overline{RES}$), software instructions, or a HALT instruction. There are two types of software instructions. One type is the single instruction "CLR WDT", the other type comprises two instructions, "CLR WDT1" and "CLR WDT2". Of these two types of instructions, only one can be active depending on the code option – "CLR WDT times selection option". If the "CLR WDT" is selected (i.e.. CLRWDT times equal one), any execution of the CLR WDT instruction will clear the WDT. In case "CLR WDT1" and "CLR WDT2" are chosen (i.e.. CLRWDT times equal two), these two instructions must be executed to clear the WDT; otherwise, the WDT may reset the chip due to a time-out.

### Power down operation – HALT

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

- The system oscillator turns off and the WDT stops.
- The contents of the on-chip RAM and registers remain unchanged.
- WDT prescaler are cleared.
- All I/O ports maintain their original status.
- The PD flag is set and the TO flag is cleared.

The system can quit the HALT mode by means of an external reset or an external falling edge signal on port B. An external reset causes a device initialization. Examining the TO and PD flags, the reason for chip reset can be determined. The PD flag is cleared when the system powers up or execute the CLR WDT instruction and is set when the HALT instruction is executed. The TO flag is set if the WDT time-out occurs, and causes a wake-up that only resets the PC (Program Counter) and SP, the others keep their original status.

The port B wake-up can be considered as a continuation of normal execution. Each bit in port B can be independently selected to wake up the device by the code option. Awakening from an I/O port stimulus, the program will resume execution of the next instruction.

Once a wake-up event(s) occurs, it takes 1024 $t_{SYS}$ (system clock period) to resume normal operation. In other words, a dummy cycle period will be inserted after the wake-up.

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

### Reset

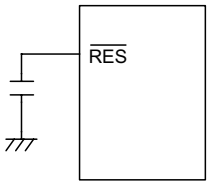
There are three ways in which a reset can occur:

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

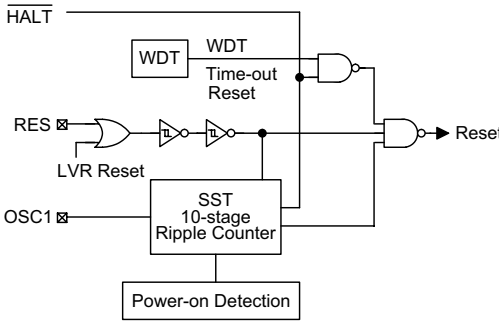
Some registers remain unchanged during reset conditions. Most registers are reset to the "initial condition" when the reset conditions are met. By examining the PD and TO flags, the program can distinguish between different "chip resets".

| TO | PD | 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 |

Note: "u" means "unchanged".



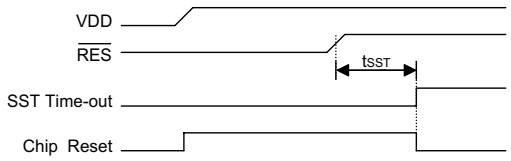

Reset circuit




Reset configuration

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

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




Reset timing chart

The functional unit chip reset status is shown below.

| | |
|---|---|
| PC | 000H |
| WDT Prescaler | Clear |
| Input/output ports | Input mode |
| SP | Points to the top of the stack |
| Carrier Output | Low level |

The chip reset status of the registers is summarized in the following table:

| Register | Reset (power on) | WDT time-out (normal operation) | $\overline{RES}$ reset (normal operation) | $\overline{RES}$ reset (HALT) |
|---|---|---|---|---|
| PC (Program Counter) | 000H | 000H | 000H | 000H |
| MP | -xxx xxxx | -uuu uuuu | -uuu uuuu | -uuu uuuu |
| ACC | xxxx xxxx | uuuu uuuu | uuuu uuuu | uuuu uuuu |
| TBLP | xxxx xxxx | uuuu uuuu | uuuu uuuu | uuuu uuuu |
| TBLH | --xx xxxx | --uu uuuu | --uu uuuu | --uu uuuu |
| STATUS | --00 xxxx | --1u uuuu | --uu uuuu | --01 uuuu |
| PA | 1111 1111 | 1111 1111 | 1111 1111 | 1111 1111 |
| PB | 1111 1111 | 1111 1111 | 1111 1111 | 1111 1111 |
| PC | ---- ---1 | ---- ---1 | ---- ---1 | ---- ---1 |

Note: "u" means "unchanged"
"x" means "unknown"

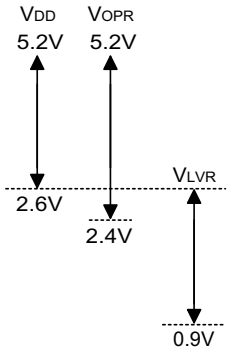
### Low voltage reset – LVR

The HT48RA0 provides 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~2.6V, such as changing a battery, the LVR will automatically reset the device internally.

The LVR includes the following specifications:

- The low voltage (0.9V~2.6V) has to remain in their original state to exceed 1 ms. If the low voltage state does not exceed 1 ms, the LVR will ignore it and do not perform a reset function.
- The LVR uses the "OR" function with the external $\overline{RES}$ signal to perform chip reset.
- During HALT mode, if the LVR occurs, the device will wake-up and the PD flag will be set as "1", the same as the external $\overline{RES}$.

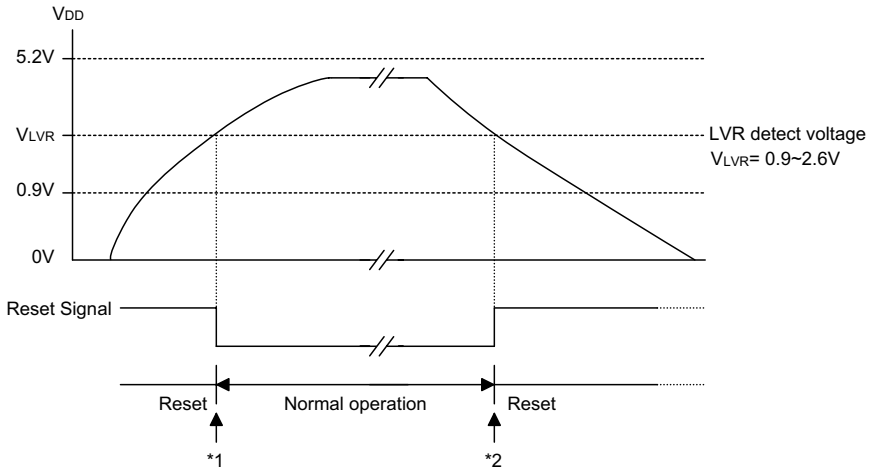
Because the operating voltage ($V_{DD}$) is 2.6V~5.2V and the LVR operating voltage ($V_{LVR}$) is 0.9V~2.6V, therefore one margin voltage about 0.1V is needed for proper chip operation. The relationship between $V_{DD}$ and $V_{LVR}$ is shown below.




Note: $V_{OPR}$ is the voltage range for proper chip operation at 2MHz system clock.

### Carrier

The HT48RA0 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="0" to enable carrier output and setting PC0="1" to disable it at low level output.




Low voltage reset

*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: Since the low voltage has to maintain in its original state and exceed 1 ms, therefore 1 ms delay is needed to enter the reset mode.

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:

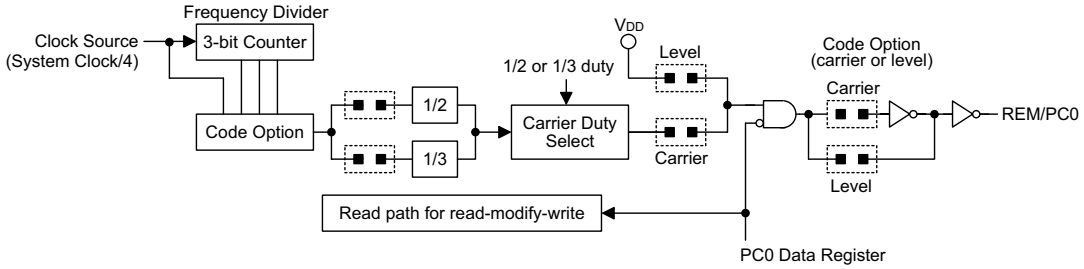
| $m \times 2^n$ | Duty Cycle |
|---|---|
| 2, 4, 8, 16 | 1/2 |
| 3 | 1/3 |
| 6, 12, 24 | 1/2 or 1/3 |

### Input/output ports

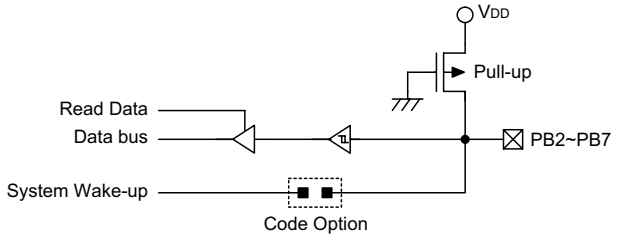
There are an 8-bit bidirectional input/output port, a 6-bit input with 2-bit I/O port and one-bit output port in the HT48RA0, labeled PA, PB and PC which are mapped to [12H], [14H], [16H] of the RAM, respectively. Each bit of PA can be selected as NMOS output or schmitt trigger with pull-high resistor by software instruction. PB0~PB1 have the same structure with PA, while PB2~PB7 can only be used for input operation (schmitt trigger with pull-high resistors). PC is only one-bit output port shares the pin with carrier output. If the level option is selected, the PC is CMOS output.

Both PA and PB 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, PB0~PB1 and PC output operation, all data are latched and remain unchanged until the output latch is rewritten.

When the PA and PB0~PB1 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, i=0~1 for PB) is executed first to disable related NMOS device, and then "MOV A, [m]" to get stable data.




Carrier/Level output



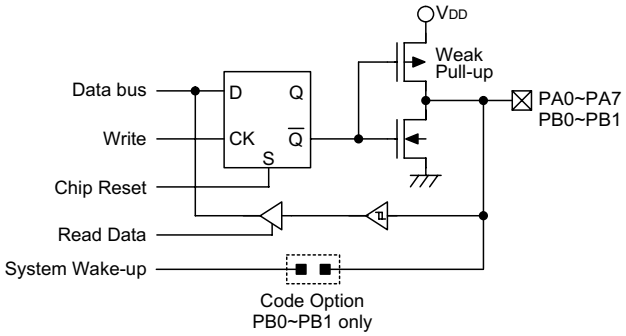

PB input lines

After chip reset, PA and PB remain at a high level input line while PC remain at high level output, if the level option is selected.

Each bit of PA, PB0~PB1 and PC output latches can be set or cleared by the "SET [m].i" and "CLR [m].i" (m=12H, 14H 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.



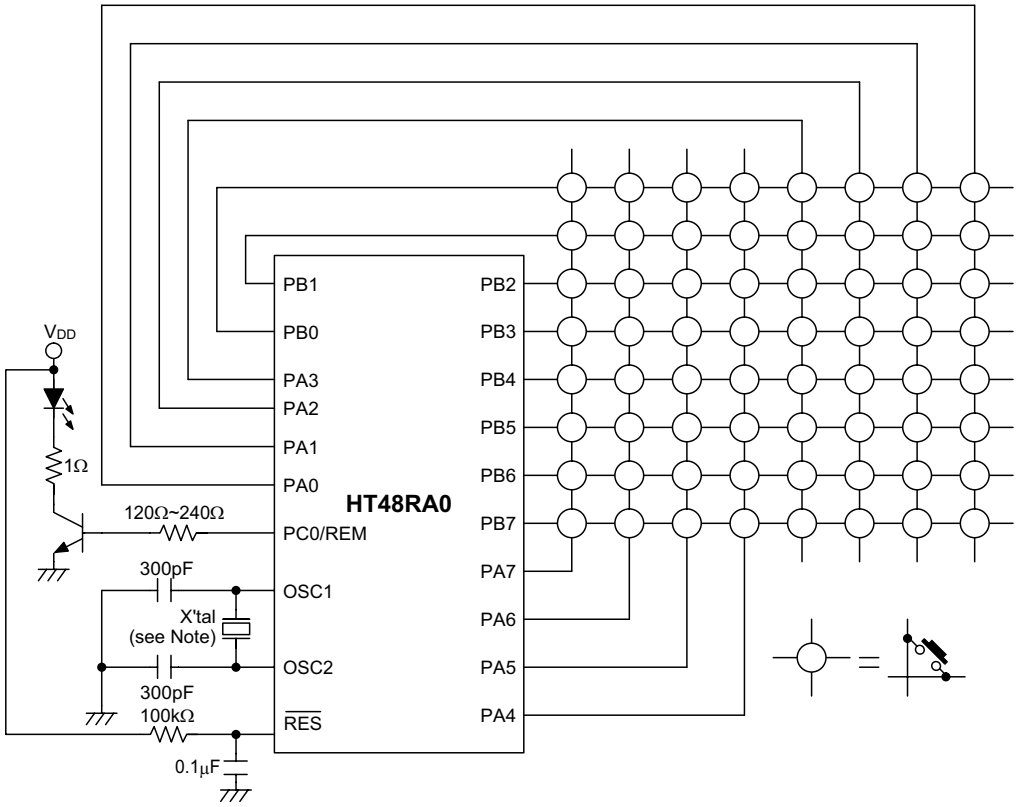

PA, PB Input/output lines

### Code option

The following table shows eight kinds of code option in the HT48RA0. All the code options must be defined to ensure proper system functioning.

| No. | Code option |
|---|---|
| 1 | WDT time-out period selection<br>Time-out period= $\frac{\text{Clock Source}}{2^n}$ where n=8~11. |
| 2 | WDT enable/disable selection. This option is to decide whether the WDT timer is enabled or disabled. |
| 3 | CLRWDT times selection. This option defines how to clear the WDT by instruction. "One time" means that the CLR WDT instruction can clear the WDT. "Two times" means only if both of the CLR WDT1 and CLR WDT2 instructions have been executed, the WDT can be cleared. |
| 4 | Wake-up selection. This option defines the wake-up activity function. External input pins (PB only) all have the capability to wake-up the chip from a HALT. |
| 5 | Carrier/level output selection. This option defines the activity of PC0 to be carrier output or level output. |
| 6 | Carry frequency selection.<br>Carry frequency= $\frac{\text{Clock Source}}{(2 \text{ or } 3) \times 2^n}$ where n=0~3. |
| 7 | Carrier duty selection. There are two types of selection: 1/2 duty or 1/3 duty.<br>If carrier frequency= Clock Source /(2, 4, 8 or 16), the duty cycle will be 1/2 duty.<br>If carrier frequency= Clock Source /3, the duty cycle will be 1/3 duty.<br>If carrier frequency= Clock Source /(6, 12 or 24), the duty cycle can be 1/2 duty or 1/3 duty. |
| 8 | OSC type selection. This option is to decide if an RC or Crystal oscillator is chosen as system clock. If the Crystal oscillator is selected, the XST (Crystal Start-up Timer) default is activated, otherwise the XST is disabled. |

## Application Circuits




Note: It is recommended that a 0.1μF decoupling capacitor is placed between VSS and VDD. If the crystal has a value above 1MHz the capacitors are not required.

## 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 register 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 with 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 |

| Mnemonic | Description | Instruction Cycle | Flag Affected |
|---|---|---|---|
| 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 |
| 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 |

| Mnemonic | Description | Instruction Cycle | Flag Affected |
|---|---|---|---|
| Table Read | | | |
| TABRDC [m] | Read EPROM code (current page) to data memory and TBLH | 2(1) | None |
| TABRDL [m] | Read EPROM 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,PD |
| CLR WDT1 | Pre-clear Watchdog timer | 1 | TO(4),PD(4) |
| CLR WDT2 | Pre-clear Watchdog timer | 1 | TO(4),PD(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,PD |

Note: x: 8 bits immediate data

m: 7 bits data memory address

A: accumulator

i: 0~7 number of bits

addr: 11 bits program memory address

√: Flag(s) is affected

–: Flag(s) is not affected

(1): If a loading to the PCL register occurs, the execution cycle of instructions will be delayed for one more cycle (4 system clocks).

(2): If a skip to the next instruction occurs, the execution cycle of instructions will be delayed for one more cycle (4 system clocks). Otherwise the original instruction cycle(s) 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 is set and the PD is cleared. Otherwise the TO and PD 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)

| TC2 | TC1 | TO | PD | 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)

| TC2 | TC1 | TO | PD | 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)

| TC2 | TC1 | TO | PD | 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)

| TC2 | TC1 | TO | PD | 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)

| TC2 | TC1 | TO | PD | 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)

| TC2 | TC1 | TO | PD | 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)

| TC2 | TC1 | TO | PD | 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)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | √ | — | — |

**CALL addr** Subroutine call

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

Operation Stack ← PC+1
PC ← addr

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — |

**CLR [m]** Clear data memory

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

Operation [m] ← 00H

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — |

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

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

Operation [m].i ← 0

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — |

**CLR WDT** Clear watchdog timer

Description The WDT and the WDT Prescaler are cleared (re-counting from zero). The power down bit (PD) and time-out bit (TO) are cleared.

Operation WDT and WDT Prescaler ← 00H
PD and TO ← 0

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | 0 | 0 | — | — | — | — |

**CLR WDT1** Preclear watchdog timer

Description The TD, PD flags, WDT and the WDT Prescaler has cleared (re-counting from zero), if the other preclear WDT instruction has been executed. Only execution of this instruction without the other preclear instruction sets the indicated flag which implies that this instruction has been executed and the TO and PD flags remain unchanged.

Operation WDT and WDT Prescaler ← 00H*
PD and TO ← 0*

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | 0* | 0* | — | — | — | — |

**CLR WDT2** Preclear watchdog timer

Description The TO, PD flags, WDT and the WDT Prescaler are cleared (re-counting from zero), if the other preclear WDT instruction has been executed. Only execution of this instruction without the other preclear instruction sets the indicated flag which implies that this instruction has been executed and the TO and PD flags remain unchanged.

Operation WDT and WDT Prescaler ← 00H*
PD and TO ← 0*

Affected flag(s)

| TC2 | TC1 | TO | PD | 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 one are changed to zero and vice-versa.

Operation $[m] \leftarrow [\overline{m}]$

Affected flag(s)

| TC2 | TC1 | TO | PD | 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 one are changed to zero 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)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | √ | — | — |

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

Description The accumulator value is adjusted to the BCD (Binary Code 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)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | √ |

**DEC [m]** Decrement data memory

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

Operation [m] ← [m]–1

Affected flag(s)

| TC2 | TC1 | TO | PD | 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 one, leaving the result in the accumulator. The contents of the data memory remain unchanged.

Operation ACC ← [m]–1

Affected flag(s)

| TC2 | TC1 | TO | PD | 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 (PD) is set and the WDT time-out bit (TO) is cleared.

Operation PC ← PC+1
PD ← 1
TO ← 0

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | 0 | 1 | — | — | — | — |

**INC [m]** Increment data memory

Description Data in the specified data memory is incremented by one.

Operation [m] ← [m]+1

Affected flag(s)

| TC2 | TC1 | TO | PD | 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 one, leaving the result in the accumulator. The contents of the data memory remain unchanged.

Operation ACC ← [m]+1

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | √ | — | — |

**JMP addr** Directly jump

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

Operation PC ← addr

Affected flag(s)

| TC2 | TC1 | TO | PD | 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)

| TC2 | TC1 | TO | PD | 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)

| TC2 | TC1 | TO | PD | 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)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — |

**NOP** No operation

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

Operation PC ← PC+1

Affected flag(s)

| TC2 | TC1 | TO | PD | 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)

| TC2 | TC1 | TO | PD | 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)

| TC2 | TC1 | TO | PD | 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)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | √ | — | — |

**RET** Return from subroutine

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

Operation PC ← Stack

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — |

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

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

Operation PC ← Stack
ACC ← x

Affected flag(s)

| TC2 | TC1 | TO | PD | 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 (bit 0; register INTC).

Operation PC ← Stack
EMI ← 1

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — |

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

Description The contents of the specified data memory are rotated one 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)

| TC2 | TC1 | TO | PD | 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 one 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)

| TC2 | TC1 | TO | PD | 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 one 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)

| TC2 | TC1 | TO | PD | 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 one 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)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | √ |

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

Description The contents of the specified data memory are rotated one 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)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — |

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

Description Data in the specified data memory is rotated one 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)

| TC2 | TC1 | TO | PD | 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 one 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)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | √ |

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

Description Data of the specified data memory and the carry flag are rotated one 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)

| TC2 | TC1 | TO | PD | 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)

| TC2 | TC1 | TO | PD | 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)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | √ | √ | √ | √ |

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

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

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

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — |

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

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

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

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — |

**SET [m]** Set data memory

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

Operation [m] ← FFH

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — |

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

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

Operation [m].i ← 1

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — |

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

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

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

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — |

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

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

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

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — |

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

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

Operation Skip if [m].i≠0

Affected flag(s)

| TC2 | TC1 | TO | PD | 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)

| TC2 | TC1 | TO | PD | 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)

| TC2 | TC1 | TO | PD | 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)

| TC2 | TC1 | TO | PD | 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 (one of the data memories) are interchanged.

Operation [m].3~[m].0 ↔ [m].7~[m].4

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — |

**SWAPA [m]** Swap data memory-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)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — |

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

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

Operation Skip if [m]=0

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — |

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

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

Operation Skip if [m]=0, ACC ← [m]

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — |

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

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

Operation Skip if [m].i=0

Affected flag(s)

| TC2 | TC1 | TO | PD | 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)

| TC2 | TC1 | TO | PD | 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)

| TC2 | TC1 | TO | PD | 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)

| TC2 | TC1 | TO | PD | 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 zero flag is affected.

Operation [m] ← ACC ″XOR″ [m]

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | √ | — | — |

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

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

Operation ACC ← ACC ″XOR″ x

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | √ | — | — |

**Holtek Semiconductor Inc. (Headquarters)**
No.3 Creation Rd. II, Science-based Industrial Park, Hsinchu, Taiwan, R.O.C.
Tel: 886-3-563-1999
Fax: 886-3-563-1189

**Holtek Semiconductor Inc. (Taipei Office)**
5F, No.576, Sec.7 Chung Hsiao E. Rd., Taipei, Taiwan, R.O.C.
Tel: 886-2-2782-9635
Fax: 886-2-2782-9636
Fax: 886-2-2782-7128 (International sales hotline)

**Holtek Semiconductor (Hong Kong) Ltd.**
RM.711, Tower 2, Cheung Sha Wan Plaza, 833 Cheung Sha Wan Rd., Kowloon, Hong Kong
Tel: 852-2-745-8288
Fax: 852-2-742-8657