

**HOLTEK**



**HT82K68A**

**Preliminary**

**Multimedia Keyboard Encoder Body**

### Features

- Operating voltage: 2.2V~5.5V
- 34(42-pin)/32(40-pin) bidirectional I/O lines
- One 8-bit programmable timer counter with overflow interrupts
- Crystal or RC oscillator
- Watchdog Timer
- 3K×16 program ROM
- 160×8 data RAM
- HALT function and wake-up feature reduce power consumption
- Six-level subroutine nesting
- Bit manipulation instructions
- 16-bit table read instructions
- 63 powerful instructions
- All instructions in 1 or 2 machine cycles
- 40-pin DIP/42-pin DIP/DICE form package

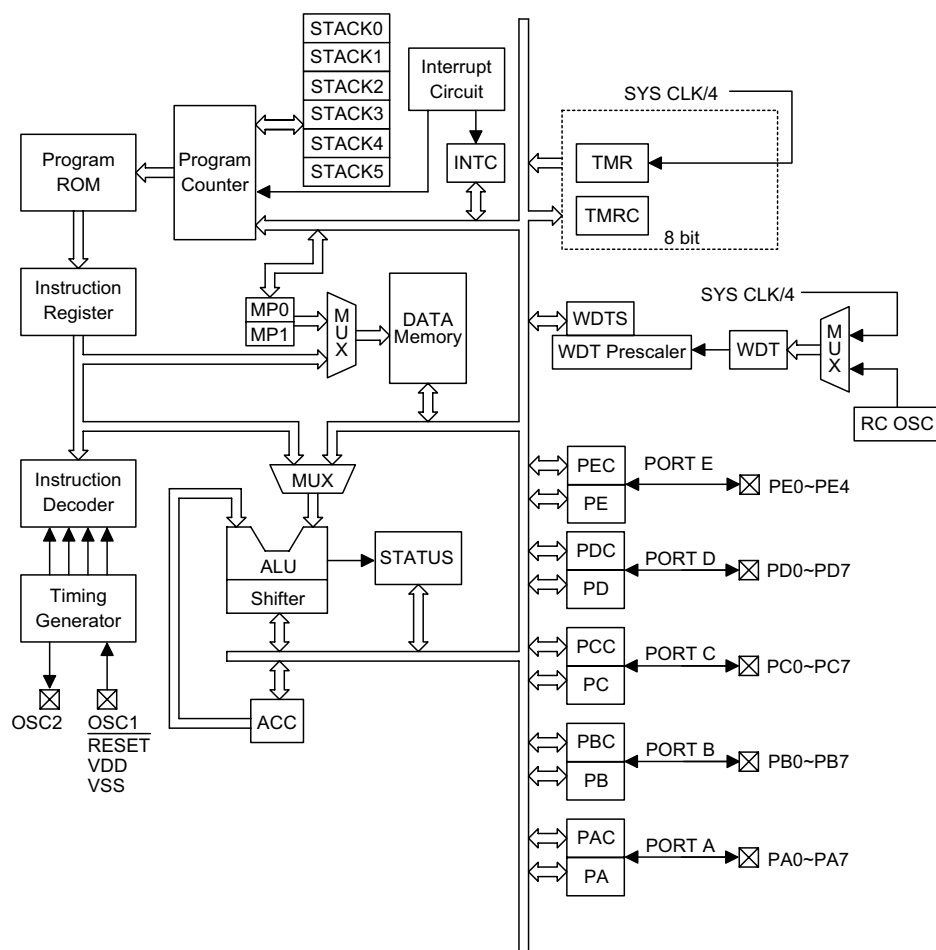
### General Description

The HT82K68A is an 8-bit high performance peripheral interface IC, designed for multiple I/O products and multimedia applications. It supports interface to a low speed PC with

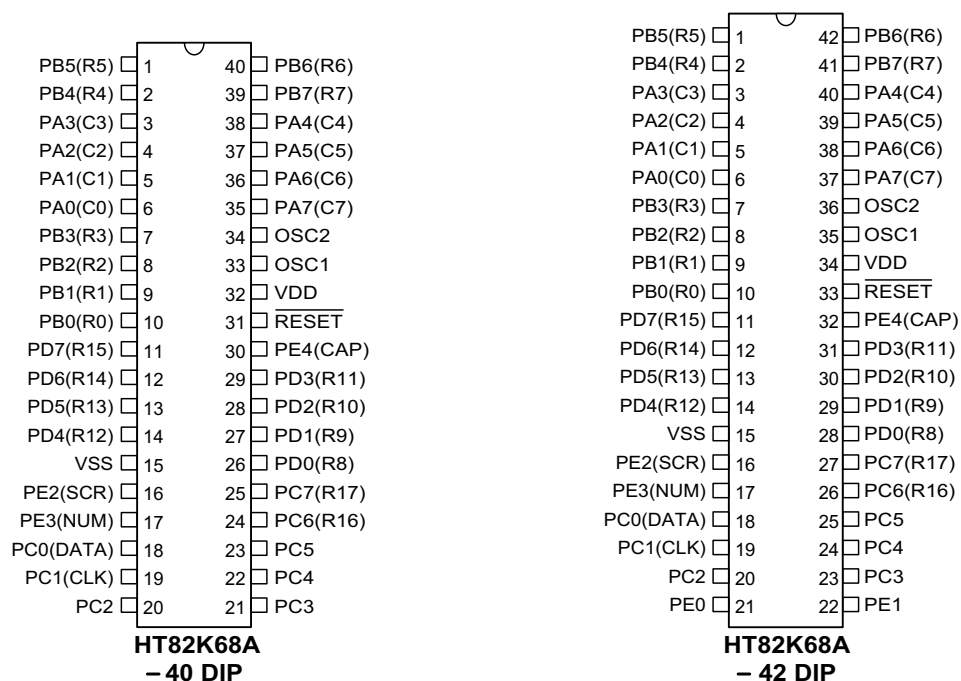
multimedia keyboard or wireless keyboard in Windows 95, Windows 98 or Windows 2000 environment. A HALT feature is included to reduce power consumption.



## Block Diagram



## Pin Assignment



## Pin Description

| Pin Name | I/O | Mask Option                     | Description  |
|----------|-----|---------------------------------|--|
| PA0~PA7  | I/O | Wake-up<br>Pull-high<br>or None | Bidirectional 8-bit input/output port. Each bit can be configured as a wake-up input by mask option. Software* instructions determine the CMOS output or schmitt trigger input with or without pull-high resistor 12K. |
| PB0~PB7  | I/O | Pull-high<br>or None            | Bidirectional 8-bit input/output port. Software* instructions determine the CMOS output or schmitt trigger input with or without pull-high resistor.   |
| PC0      | I/O | Wake-up<br>Pull-high<br>or None | This pin is an I/O port. NMOS open drain output with pull-high resistor and can be used as DATA or CLOCK line of PS2. This pin can be configured as a wake-up input by mask option.                                    |
| PC1      | I/O | Wake-up<br>Pull-high<br>or None | This pin is an I/O port. NMOS open drain output with pull-high resistor and can be used as DATA or CLOCK line of PS2. This pin can be configured as a wake-up input by mask option.                                    |

| Pin Name           | I/O    | Mask Option                     | Description   |
|--------------------|--------|---------------------------------|---|
| PC2~PC3            | I/O    | Wake-up<br>Pull-high<br>or None | Bidirectional 2-bit input/output port. Each bit can be configured as a wake-up input by mask option. Software* instructions determine the CMOS output or schmitt trigger input with or without pull-high resistor.                                    |
| PC4~PC7            | I/O    | Pull-high<br>or None            | Bidirectional 4-bit input/output port. Software* instructions determine the CMOS output or schmitt trigger input with or without pull-high resistor.  |
| PD0~PD7            | I/O    | Pull-high<br>or None            | Bidirectional 8-bit input/output port. Software* instructions determine the CMOS output or schmitt trigger input with or without pull-high resistor.  |
| PE0~PE1            | I/O    | Pull-high<br>or None            | Bidirectional input/output port. Software* instruction determine the CMOS output or schmitt trigger input with or without pull-high resistor.   |
| PE2                | O      |                                 | This pin is an NMOS output structure. The pad can function as LED (SCR) drivers for the keyboard. $I_{OL}=14mA$ , $@V_{OL}=3.2V$  |
| PE3                | O      |                                 | This pin is an NMOS output structure. The pad can function as LED (NUM) drivers for the keyboard. $I_{OL}=14mA$ , $@V_{OL}=3.2V$  |
| PE4                | O      |                                 | This pin is an NMOS output structure. The pad can function as LED (CAP) drivers for the keyboard. $I_{OL}=14mA$ , $@V_{OL}=3.2V$  |
| VDD                | —      | —                               | Positive power supply   |
| VSS                | —      | —                               | Negative power supply, ground   |
| $\overline{RESET}$ | I      | —                               | Chip reset input. Active low. Built-in power-on reset circuit to reset the entire chip. Chip can also be externally reset via RESET pin   |
| OSC1<br>OSC2       | I<br>O | Crystal or<br>RC                | OSC1, OSC2 are connected to an RC network or a crystal for the internal system clock. In the case of RC operation, OSC2 is the output terminal for the 1/4 system clock; A 110k $\Omega$ resistor is connected to OSC1 to generate a 2 MHz frequency. |

Note: \*: Software means the HT-IDE (Holtek Integrated Development Environment) can be configured by mask option.

### Absolute Maximum Ratings

|                      |                                |                             |                |
|----------------------|--------------------------------|-----------------------------|----------------|
| Supply Voltage ..... | -0.3V to 5.5V                  | Storage Temperature .....   | -50°C to 125°C |
| Input Voltage .....  | $V_{SS}-0.3V$ to $V_{DD}+0.3V$ | 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 <sub>DD</sub> | Conditions                       |      |      |      |      |
| V <sub>DD</sub>   | Operating Voltage  | —               | —                                | 2.2  | —    | 5.5  | V    |
| I <sub>DD1</sub>  | Operating Current<br>(Crystal OSC)                             | 3V              | No load, f <sub>SYS</sub> = 2MHz | —    | 0.7  | 1.5  | mA   |
|                   |  | 5V              |                                  | —    | 2    | 5    | mA   |
| I <sub>DD2</sub>  | Operating Current<br>(RC OSC)                                  | 3V              | No load, f <sub>SYS</sub> = 2MHz | —    | 0.5  | 1    | mA   |
|                   |  | 5V              |                                  | —    | 2    | 5    | mA   |
| I <sub>STB1</sub> | Standby Current<br>(WDT enabled)                               | 3V              | No load, system HALT             | —    | —    | 8    | μA   |
|                   |  | 5V              |                                  | —    | —    | 15   | μA   |
| I <sub>STB2</sub> | Standby Current<br>(WDT Disabled)                              | 3V              | No load, system HALT             | —    | —    | 3    | μA   |
|                   |  | 5V              |                                  | —    | —    | 6    | μA   |
| V <sub>IL</sub>   | Input Low Voltage for<br>I/O Ports                             | 3V              | —                                | 0    | —    | 0.9  | V    |
|                   |  | 5V              | —                                | 0    | —    | 1.5  | V    |
| V <sub>IH</sub>   | Input High Voltage for<br>I/O Ports                            | 3V              | —                                | 2.1  | —    | 3    | V    |
|                   |  | 5V              | —                                | 3.5  | —    | 5    | V    |
| V <sub>IL1</sub>  | Input Low Voltage<br>(RESET)                                   | 3V              | —                                | 0    | —    | 0.7  | V    |
|                   |  | 5V              | —                                | 0    | —    | 1.3  | V    |
| V <sub>IH1</sub>  | Input High Voltage<br>(RESET)                                  | 3V              | —                                | 2.4  | —    | 3    | V    |
|                   |  | 5V              | —                                | 4.0  | —    | 5    | V    |
| I <sub>OL</sub>   | I/O Port Sink Current  | 5V              | V <sub>OL</sub> = 0.5V           | 7    | 12   | —    | mA   |
| I <sub>OH</sub>   | I/O Port Source Current  | 5V              | V <sub>OH</sub> = 4.5V           | -2.5 | -4.5 | —    | mA   |
| I <sub>LED</sub>  | LED Sink Current<br>(SCR, NUM, CAP)                            | 5V              | V <sub>OL</sub> =3.2V            | 10   | 14   | 18   | mA   |
| t <sub>POR</sub>  | Power-on Reset Time  | 5V              | —                                | 120  | 150  | 180  | ms   |
| R <sub>PH</sub>   | Internal Pull-high<br>Resistance of PA, PB, PC,<br>PD, PE Port | 5V              | —                                | 5    | 12   | 20   | kΩ   |
| R <sub>PH1</sub>  | Internal Pull-high<br>Resistance of DATA, CLK                  | 5V              | —                                | 2    | 4.7  | 8    | kΩ   |
| Δf/f              | Frequency Variation  | 5V              | Crystal                          | —    | —    | ±1   | %    |
| Δf/f1             | Frequency Variation  | 5V              | RC                               | —    | —    | ±10  | %    |

**A.C. Characteristics**
 $T_a=25^{\circ}\text{C}$ 

| Symbol              | Parameter                                  | Test Conditions |                                  | Min. | Typ. | Max. | Unit             |
|---------------------|--|-----------------|----------------------------------|------|------|------|------------------|
|                     |  | V <sub>DD</sub> | Conditions                       |      |      |      |                  |
| f <sub>SYS1</sub>   | System Clock<br>(Crystal OSC)              | 3V              | —                                | —    | 2    | —    | MHz              |
|                     |  | 5V              | —                                | —    | 2    | —    | MHz              |
| f <sub>SYS2</sub>   | System Clock (RC OSC)                      | 3V              | —                                | —    | 2    | 12   | MHz              |
|                     |  | 5V              | —                                | —    | 2    | 20   | MHz              |
| t <sub>WDTOSC</sub> | Watchdog Oscillator                        | 3V              | —                                | 45   | 90   | 180  | μs               |
|                     |  | 5V              | —                                | 35   | 65   | 130  | μs               |
| t <sub>WDT1</sub>   | Watchdog Time-out<br>Period (RC)           | 3V              | Without WDT<br>prescaler         | 12   | 23   | 45   | ms               |
|                     |  | 5V              |                                  | 9    | 17   | 35   | ms               |
| t <sub>WDT2</sub>   | Watchdog Time-out<br>Period (System Clock) | —               | Without WDT<br>prescaler         | —    | 1024 | —    | t <sub>SYS</sub> |
| t <sub>RES</sub>    | External Reset Low<br>Pulse Width          | —               | —                                | 1    | —    | —    | μs               |
| t <sub>SST</sub>    | System Start-up Timer<br>Period            | —               | Power-up or<br>wake-up from HALT | —    | 1024 | —    | t <sub>SYS</sub> |
| t <sub>INT</sub>    | Interrupt Pulse Width                      | —               | —                                | 1    | —    | —    | μs               |

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

### Functional Description

#### Execution flow

The HT82K68A system clock is derived from either a crystal or an RC oscillator. The system clock 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 12-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 4096 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, exter-

nal 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.

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

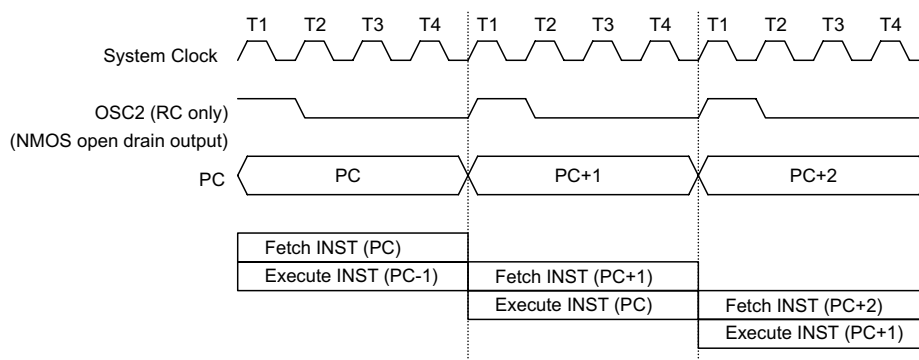
#### Program memory – ROM

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 with  $3072 \times 16$  bits, addressed by the program counter and table pointer.

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

- Location 000

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



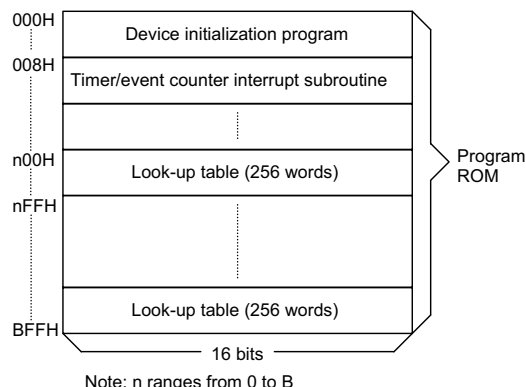
Execution flow

- Location 008H

This area is reserved for the timer counter interrupt service program. If timer interrupt results from a timer counter overflow, and if the interrupt is enabled and the stack is not full, the program begins execution at location 008H.

- Table location

Any location in the 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 other bits of the table word are transferred to the lower portion of TBLH, the remaining 1 bit is read as 0. The Table Higher-order byte register (TBLH) is read only. 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. How-



Program memory

ever, 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. 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. All table related instructions need 2 cycles to complete the operation. These areas may function as normal program memory depending upon the requirements.

| Mode                   | Program Counter |     |    |    |    |    |    |    |    |    |    |    |
|------------------------|-----------------|-----|----|----|----|----|----|----|----|----|----|----|
|                        | *11             | *10 | *9 | *8 | *7 | *6 | *5 | *4 | *3 | *2 | *1 | *0 |
| Initial reset          | 0               | 0   | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
| Timer counter overflow | 0               | 0   | 0  | 0  | 0  | 0  | 0  | 0  | 1  | 0  | 0  | 0  |
| Skip                   | PC+2            |     |    |    |    |    |    |    |    |    |    |    |
| Loading PCL            | *11             | *10 | *9 | *8 | @7 | @6 | @5 | @4 | @3 | @2 | @1 | @0 |
| Jump, call branch      | #11             | #10 | #9 | #8 | #7 | #6 | #5 | #4 | #3 | #2 | #1 | #0 |
| Return from subroutine | S11             | S10 | S9 | S8 | S7 | S6 | S5 | S4 | S3 | S2 | S1 | S0 |

Note: \*11~\*0: Program counter bits  
#11~#0: Instruction code bits

S11~S0: Stack register bits  
@7~@0: PCL bits



### Stack register – STACK

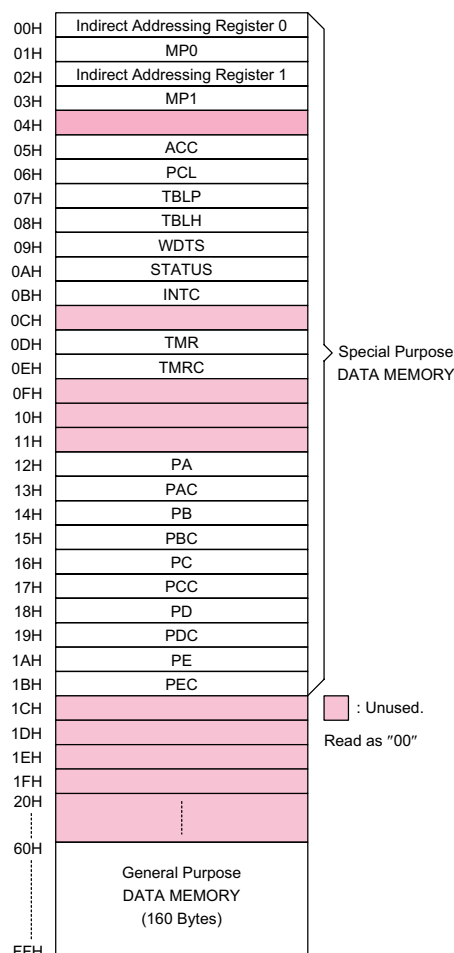
This is a special part of the memory which is used to save the contents of the program counter (PC) only. The stack is organized into six levels and is neither part of the data nor part of the program space, and is neither readable nor writeable. The activated level is indexed by the stack pointer (SP) and is neither readable nor writeable. At a subroutine call or interrupt acknowledgement, 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 acknowledgement will be inhibited. When the stack pointer is decremented (by RET or RETI), the interrupt will be serviced. This feature prevents stack overflow allowing the programmer to use the structure more easily. In a similar case, if the stack is full and a "CALL" is subsequently executed, stack overflow occurs and the first entry will be lost (only the most recent four return addresses are stored).

### Data memory – RAM

The data memory is designed with  $184 \times 8$  bits. It is divided into two functional groups: special function registers and general purpose data memory ( $160 \times 8$ ). Most of them are read/write, but some are read only.

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 Accumulator



RAM mapping

(ACC;05H), the Program Counter Lower-byte register (PCL;06H), the Table Pointer (TBLP;07H), the Table Higher-order byte register (TBLH;08H), the Watchdog Timer option Setting register (WDTS;09H), the Status register

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

Note: \*11~\*0: Table location bits  
@7~@0: Table location bits

P11~P8: Current program counter bits

(STATUS;0AH), the Interrupt Control register (INTC;0BH), the timer counter register (TMR;0DH), the timer counter control register (TMRC;0EH), the I/O registers (PA;12H, PB;14H, PC;16H, PD;18H, PE;1AH) and the I/O control registers (PAC;13H, PBC;15H, PCC;17H, PDC;19H, PEC;1BH). The remaining space before the 60H is reserved for future expanded usage and reading these locations will get the result 00H. The general purpose data memory, addressed from 60H to FFH, 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 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] can access the 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 is not supported. The memory pointer registers, MP0 and MP1, are 8-bit registers which can be used to access the data memory by combining corresponding indirect addressing registers.

#### **Accumulator**

The accumulator is closely related to the 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, SZ, SDZ ....)

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

#### **Status register – Status**

The 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 watch dog time-out flag (TO). The status register not only records the status information but also 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 registers. Any data written into the status register will not change the TO or PD flags. 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 system power up, Watchdog Timer overflow, executing the HALT instruction and clearing the Watchdog Timer.

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

In addition, on entering an interrupt sequence or executing a 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, precaution must be taken to save it properly.

### Interrupt

The HT82K68A provides an internal timer counter interrupt. The interrupt control register (INTC;0BH) contains the interrupt control bits to set not only the enable/disable status but also the 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 occur during this interval but only the interrupt request flag is recorded. If a certain interrupt requires servicing within the service routine, the EMI bit and the corresponding bit of the INTC may be set to allow interrupt nesting. If the stack is full, the interrupt request will not be acknowledged, even if the related interrupt is enabled, until the SP is decremented. If immediate service is desired, the stack must be prevented from becoming full.

All these kinds of interrupt have the wake-up capability. As an interrupt is serviced, a control transfer occurs by pushing the program counter onto the stack followed by a branch to a subroutine at the specified location in the program

memory. Only the program counter is pushed onto the stack. If the contents of the register and Status register (STATUS) are altered by the interrupt service program which corrupt the desired control sequence, the contents should be saved in advance.

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

During the execution of an interrupt subroutine, other interrupt acknowledgements 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, a RET or RETI instruction may be invoked. RETI will set the EMI bit to enable an interrupt service, but RET will not.

Interrupts occurring in the interval between the rising edges of two consecutive T2 pulses,

| 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 if 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

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       | Vector |
|-----|------------------------|--------|
| a   | Timer counter overflow | 08H    |

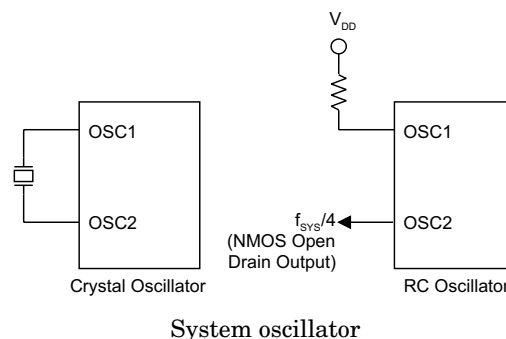
The timer counter interrupt request flag (T0F), enable timer counter bit (ET0I), and enable master interrupt bit (EMI) constitute an interrupt control register (INTC) which is located at 0BH in the data memory. EMI, ET0I, are used to control the enabling/disabling of interrupts. These bits prevent the requested interrupt from being serviced. Once the interrupt request flags (T0F) are set, they will remain in the INTC register until the interrupts are serviced or cleared by a software instruction.

It is suggested that a program does not use the "CALL subroutine" within the interrupt subroutine. Because interrupts often occur in an unpredictable manner or need to be serviced immediately in some applications, if only one stack is left and enabling the interrupt is not well

controlled, once the "CALL subroutine" operates in the interrupt subroutine it will damage the original control sequence.

### Oscillator configuration

There are two oscillator circuits in HT82K68A.



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

| Register      | Bit No. | Label | Function  |
|---------------|---------|-------|---|
| INTC<br>(0BH) | 0       | EMI   | Controls the master (global) interrupt<br>(1= enabled; 0= disabled) |
|               | 1       | —     | Undefined, read as "0"; programming must be "0"                     |
|               | 2       | ET0I  | Controls the timer counter interrupt<br>(1= enabled; 0= disabled)   |
|               | 3       | —     | Undefined, read as "0"  |
|               | 4       | —     | Undefined, read as "0"; programming must be "0"                     |
|               | 5       | T0F   | Internal timer counter request flag<br>(1= active; 0= inactive)     |
|               | 6       | —     | Undefined, read as "0"  |
|               | 7       | —     | Unused bit, read as "0"   |

INTC register

If an RC oscillator is used, an external resistor between OSC1 and VDD is 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 VDD, 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 needed for 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.

The WDT oscillator is a free running on-chip RC oscillator, and no external components are required. Even if the system enters the power down mode, the system clock is stopped, but the WDT oscillator still works for a period of approximately 78 μs. The WDT oscillator can be disabled by mask option to conserve power.

### Watchdog Timer – WDT

The WDT clock source is implemented by a dedicated RC oscillator (WDT oscillator) or instruction clock (system clock divided by 4), decided by mask options. 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 mask option. If the Watchdog Timer is disabled, all the executions related to the WDT results in no operation.

Once the internal WDT oscillator (RC oscillator normally with a period of 78μs) is selected, it is first divided by 256 (8-stages) to get the nominal time-out period of approximately 20 ms. This time-out period may vary with temperature, VDD and process variations. By invoking the WDT prescaler, longer time-out periods can be realized. Writing data to WS2, WS1, WS0 (bit 2,1,0 of the WDTS) can give different

time-out periods. If WS2, WS1, WS0 are all equal to 1, the division ratio is up to 1:128, and the maximum time-out period is 2.6 seconds.

If the WDT oscillator is disabled, the WDT clock may still come from the instruction clock and operate in the same manner except that in the HALT state the WDT may stop counting and lose its protecting purpose. In this situation the WDT logic can be restarted by external logic. The high nibble and bit 3 of the WDTS are reserved for user defined flags, which can be used to indicate some specified status.

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

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

WDTS register

The WDT overflow under normal operation will initialize "chip reset" and set the status bit TO. An overflow in the HALT mode, initializes a "warm reset" only when the PC and SP are reset to zero. To clear the contents of the WDT (including the WDT prescaler), three methods are adopted; external reset (a low level to RESET), software instruction(s), or a HALT instruction. There are two types of software instructions; CLR WDT and CLR WDT1/CLR WDT2. Of these two types of instruction, only one can be active depending on the mask option – "CLR WDT times selection option". If the "CLR WDT" is selected (ie. CLR WDT times equal one), any execution of the CLR WDT instruction will clear the WDT. In case "CLR WDT1" and "CLR WDT2" are chosen (ie. CLR WDT times equal two), these two instructions must be exe-

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

## Power down operation – HALT

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

- The system oscillator will turn off but the WDT oscillator keeps running (if the WDT oscillator is selected).
- The contents of the on chip RAM and registers remain unchanged.
- WDT and WDT prescaler will be cleared and recount again (if the WDT clock has come from the WDT oscillator).
- All I/O ports maintain their original status.
- The PD flag is set and the TO flag is cleared.

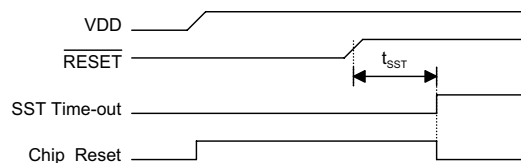
The system can leave the HALT mode by means of an external reset, an external falling edge signal on port A and port C [0:3] or a WDT overflow. An external reset causes a device initialization and the WDT overflow performs a "warm reset". Examining the TO and PD flags, the reason for chip reset can be determined. The PD flag is cleared when system power-up or executing 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 and SP, the others keep their original status.

The port A or port C [0:3] wake-up can be considered as a continuation of normal execution. Each bit in port A can be independently selected to wake up the device by mask option. Awakening from an I/O port stimulus, the program will resume execution of the next instruction.

## Reset

There are three ways in which a reset can occur:

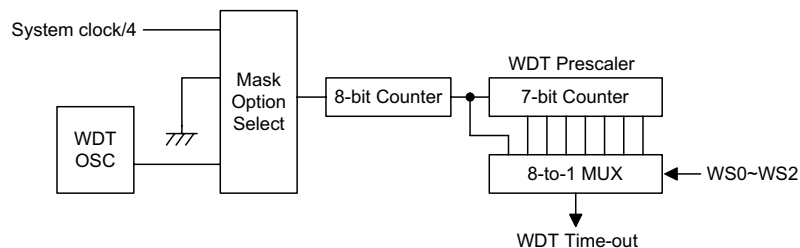
- RESET reset during normal operation
- RESET reset during HALT



### Reset timing chart

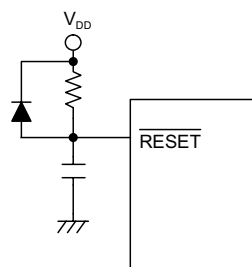
- WDT time-out reset during normal operation

The WDT time-out during HALT is different from other chip reset conditions, since it can perform a warm reset that just resets the PC and SP, leaving the other circuits to remain in their original state. Some registers remain unchanged during other reset conditions. Most registers are reset to the "initial condition"



## Watchdog Timer





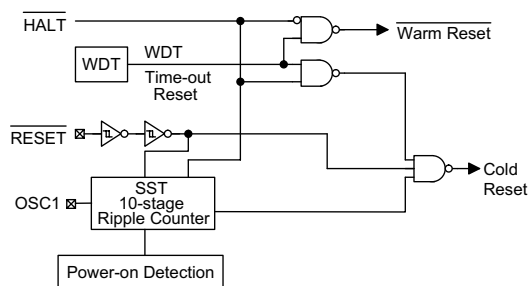
Reset circuit

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

Note: "u" means "unchanged"

To guarantee that the system oscillator 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 it awakes from the HALT state.



Reset configuration

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

The functional unit chip reset status is shown below.

|                    |  |
|--------------------|--|
| PC                 | 000H   |
| Prescaler          | Clear  |
| WDT                | Clear. After master reset, WDT begins counting |
| Timer counter      | Off  |
| Input/output ports | Input mode                                     |
| SP                 | Points to the top of the stack                 |

### Timer counter

A timer counter (TMR) is implemented in the HT82K68A. The timer counter contains an 8-bit programmable count-up counter and the clock may come from the system clock divided by 4.

Using the internal instruction clock, there is only one reference time-base.

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

In the timer mode, once the timer counter starts counting, it will count from the current contents in the timer counter to FFH. Once overflow occurs, the counter is reloaded from the timer counter preload register and generates the interrupt request flag (TF; bit 5 of INTC) at the same time.

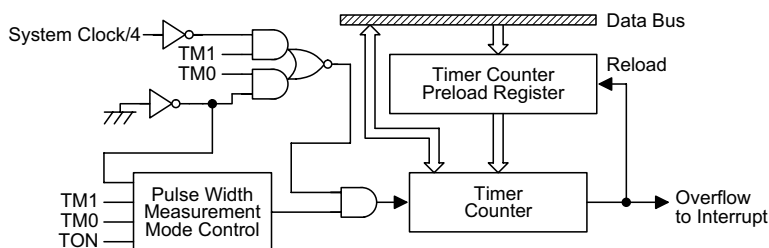
To enable the counting operation, the timer ON bit (TON; bit 4 of TMRC) should be set to 1. In the case of timer counter OFF condition, writing data to the timer counter preload register will also reload that data to the timer counter. But if the timer counter is turned on, data written to it will only be kept in the timer counter preload register. The timer counter will still operate until overflow occurs. When the timer counter (reading TMR) is read, the clock will be blocked to avoid errors. As clock blocking may results in a counting error, this must be taken into consideration by the programmer.

### Input/output ports

There are 32 bidirectional input/output lines in the HT82K68A, labeled from PA to PE, which are mapped to the data memory of [12H], [14H], [16H], [18H] and [1AH] respectively. All these I/O ports can be used for input and output operations. For input operation, these ports are non-latching, that is, the inputs must be ready at the T2 rising edge of instruction MOV A,[m] (m=12H, 14H, 16H, 18H or 1AH). For output operation, all data is latched and remains unchanged until the output latch is rewritten.

| Label      | Bits   | Function   |
|------------|--------|--|
| —          | 0~3    | Unused bits, read as "0"                                   |
| TON        | 4      | To enable/disable timer counting (0= disabled; 1= enabled) |
| —          | 5      | Unused bits, read as "0"                                   |
| TM0<br>TM1 | 6<br>7 | 10= Timer mode (internal clock)                            |

TMRC register



Timer counter



The state of the registers is summarized in the following table:

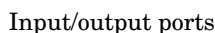
| Register | Reset<br>(power on) | WDT time-out<br>(normal<br>operation) | RESET reset<br>(normal<br>operation) | RESET reset<br>(HALT) | WDT<br>time-out<br>(HALT) |
|----------|---------------------|---------------------------------------|--------------------------------------|-----------------------|---------------------------|
| TMR      | xxxx xxxx           | uuuu uuuu                             | uuuu uuuu                            | uuuu uuuu             | uuuu uuuu                 |
| TMRC     | 00-0 1---           | 00-0 1---                             | 00-0 1---                            | 00-0 1---             | uu-u u---                 |
| PC       | 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                             | uuuu uuuu                            | uuuu uuuu             | uuuu uuuu                 |
| TBLH     | -xxx xxxx           | -uuu uuuu                             | -uuu uuuu                            | -uuu uuuu             | -uuu uuuu                 |
| STATUS   | --00 xxxx           | --1u uuuu                             | --uu uuuu                            | --01 uuuu             | --11 uuuu                 |
| INTC     | -000 0000           | -000 0000                             | -000 0000                            | -000 0000             | -uuu uuuu                 |
| WDTS     | 0000 0111           | 0000 0111                             | 0000 0111                            | 0000 0111             | uuuu uuuu                 |
| PA       | 1111 1111           | 1111 1111                             | 1111 1111                            | 1111 1111             | uuuu uuuu                 |
| PAC      | 1111 1111           | 1111 1111                             | 1111 1111                            | 1111 1111             | uuuu uuuu                 |
| PB       | 1111 1111           | 1111 1111                             | 1111 1111                            | 1111 1111             | uuuu uuuu                 |
| PBC      | 1111 1111           | 1111 1111                             | 1111 1111                            | 1111 1111             | uuuu uuuu                 |
| PC       | 1111 1111           | 1111 1111                             | 1111 1111                            | 1111 1111             | uuuu uuuu                 |
| PCC      | 1111 1111           | 1111 1111                             | 1111 1111                            | 1111 1111             | uuuu uuuu                 |
| PD       | 1111 1111           | 1111 1111                             | 1111 1111                            | 1111 1111             | uuuu uuuu                 |
| PDC      | 1111 1111           | 1111 1111                             | 1111 1111                            | 1111 1111             | uuuu uuuu                 |
| PE       | ---1 1111           | ---1 1111                             | ---1 1111                            | ---1 1111             | ---u uuuu                 |
| PEC      | ---1 1111           | ---1 1111                             | ---1 1111                            | ---1 1111             | ---u uuuu                 |

Note: "\*" means warm reset

"u" means unchanged

"x" means unknown

Each line of port A and port C [0:3] has the capability to wake-up the device.



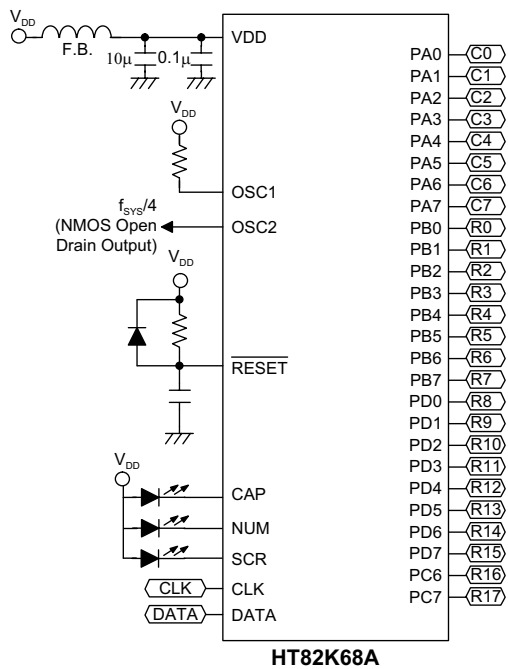
### Mask option

The following shows five kinds of mask option in the HT82K68A. All the mask options must be defined to ensure proper system function.

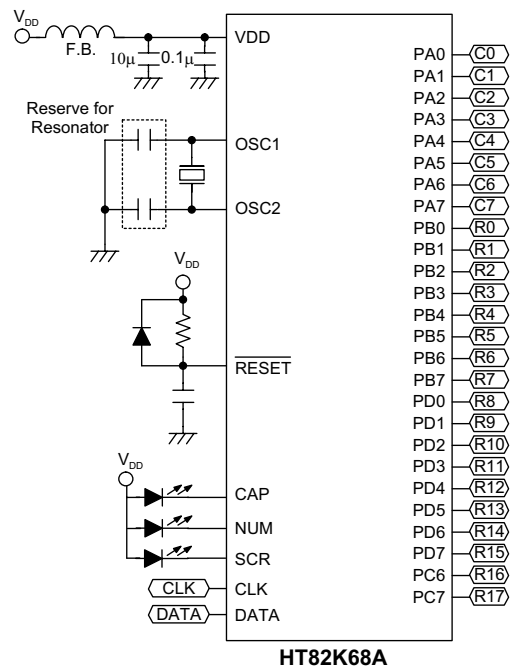
| No. | Mask Option   |
|-----|---|
| 1   | OSC type selection. This option is to decide if an RC or Crystal oscillator is chosen as system clock. If the Crystal oscillator is selected, the XST (Crystal Start-up Timer) default is activated, otherwise the XST is disabled.   |
| 2   | WDT source selection. There are three types of selection: on-chip RC oscillator, instruction clock or disable the WDT.  |
| 3   | CLRWDT times selection. This option defines the way 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, only then will the WDT be cleared. |
| 4   | Wake-up selection. This option defines the wake-up function activity. External I/O pins (PA and PC [0:3] only) all have the capability to wake-up the chip from a HALT.   |
| 5   | Pull-high selection. This option is to decide whether the pull-high resistance is visible or not in the input mode of the I/O ports. Each bit of an I/O port can be independently selected.   |
| 6   | Special power on reset. This option defines the function will reset the chip to prevent incorrect status.<br>If the special power on reset is enabled, the chip must not enter the HALT mode.   |

## Application Circuits

### RC oscillator for multiple I/O applications



### Crystal oscillator or ceramic resonator for multiple I/O applications



### Instruction Set Summary

| Mnemonic                         | Description   | Flag Affected |
|----------------------------------|---|---------------|
| <b>Arithmetic</b>                |   |               |
| ADD A,[m]                        | Add data memory to ACC  | Z,C,AC,OV     |
| ADDM A,[m]                       | Add ACC to data memory  | Z,C,AC,OV     |
| ADD A,x                          | Add immediate data to ACC   | Z,C,AC,OV     |
| ADC A,[m]                        | Add data memory to ACC with carry                                   | Z,C,AC,OV     |
| ADCM A,[m]                       | Add ACC to register with carry                                      | Z,C,AC,OV     |
| SUB A,x                          | Subtract immediate data from ACC                                    | Z,C,AC,OV     |
| SUB A,[m]                        | Subtract data memory from ACC                                       | Z,C,AC,OV     |
| SUBM A,[m]                       | Subtract data memory from ACC with result in data memory            | Z,C,AC,OV     |
| SBC A,[m]                        | Subtract data memory from ACC with carry                            | Z,C,AC,OV     |
| SBCM A,[m]                       | Subtract data memory from ACC with carry with result in data memory | Z,C,AC,OV     |
| DAA [m]                          | Decimal adjust ACC for addition with result in data memory          | C             |
| <b>Logic Operation</b>           |   |               |
| AND A,[m]                        | AND data memory to ACC  | Z             |
| OR A,[m]                         | OR data memory to ACC   | Z             |
| XOR A,[m]                        | Exclusive-OR data memory to ACC                                     | Z             |
| ANDM A,[m]                       | AND ACC to data memory  | Z             |
| ORM A,[m]                        | OR ACC to data memory   | Z             |
| XORM A,[m]                       | Exclusive-OR ACC to data memory                                     | Z             |
| AND A,x                          | AND immediate data to ACC   | Z             |
| OR A,x                           | OR immediate data to ACC  | Z             |
| XOR A,x                          | Exclusive-OR immediate data to ACC                                  | Z             |
| CPL [m]                          | Complement data memory  | Z             |
| CPLA [m]                         | Complement data memory with result in ACC                           | Z             |
| <b>Increment &amp; Decrement</b> |   |               |
| INCA [m]                         | Increment data memory with result in ACC                            | Z             |
| INC [m]                          | Increment data memory   | Z             |
| DECA [m]                         | Decrement data memory with result in ACC                            | Z             |
| DEC [m]                          | Decrement data memory   | Z             |
| <b>Rotate</b>                    |   |               |
| RRA [m]                          | Rotate data memory right with result in ACC                         | None          |
| RR [m]                           | Rotate data memory right  | C             |
| RRCA [m]                         | Rotate data memory right through carry with result in ACC           | C             |
| RRC [m]                          | Rotate data memory right through carry                              | None          |
| RLA [m]                          | Rotate data memory left with result in ACC                          | None          |
| RL [m]                           | Rotate data memory left   | C             |
| RLCA [m]                         | Rotate data memory left through carry with result in ACC            | C             |
| RLC [m]                          | Rotate data memory left through carry                               | C             |

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

Note: x: 8-bit immediate data

m: 7-bit data memory address

A: Accumulator

i: 0~7 number of bits

addr: 12 bits program memory address

√: 0~7 number of bits

–: Flag(s) is not affected

\*: Flag(s) may be affected by the execution status

### Instruction Definition

#### ADC A,[m]

Description

Add data memory and carry to the accumulator

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

Operation

$ACC \leftarrow ACC + [m] + C$

Affected flag(s)

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

#### ADCM A,[m]

Description

Add the accumulator and carry to data memory

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

Operation

$[m] \leftarrow ACC + [m] + C$

Affected flag(s)

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

#### ADD A,[m]

Description

Add data memory to the accumulator

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

Operation

$ACC \leftarrow ACC + [m]$

Affected flag(s)

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

#### ADD A,x

Description

Add immediate data to the accumulator

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

Operation

$ACC \leftarrow ACC + x$

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

$[m] \leftarrow ACC + [m]$

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

$ACC \leftarrow ACC \text{ "AND" } [m]$

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

$ACC \leftarrow ACC \text{ "AND" } x$

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

$[m] \leftarrow ACC \text{ "AND" } [m]$

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  $\leftarrow$  PC+1  
PC  $\leftarrow$  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]  $\leftarrow$  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  $\leftarrow$  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  $\leftarrow$  00H  
PD and TO  $\leftarrow$  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  $\leftarrow$  00H\*

PD and TO  $\leftarrow$  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  $\leftarrow$  00H\*

PD and TO  $\leftarrow$  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 1 are changed to 0 and vice-versa.

Operation

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

Affected flag(s)

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

|                  |   |     |     |    |    |    |   |    |   |   |   |   |   |   |   |   |   |
|------------------|---|-----|-----|----|----|----|---|----|---|---|---|---|---|---|---|---|---|
| <b>CPLA [m]</b>  | 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        | $ACC \leftarrow [\overline{m}]$   |     |     |    |    |    |   |    |   |   |   |   |   |   |   |   |   |
| Affected flag(s) | <table><tr><td>TC2</td><td>TC1</td><td>TO</td><td>PD</td><td>OV</td><td>Z</td><td>AC</td><td>C</td></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>√</td><td>—</td><td>—</td></tr></table>   | TC2 | TC1 | TO | PD | OV | Z | AC | C | — | — | — | — | — | √ | — | — |
| TC2              | TC1   | TO  | PD  | OV | Z  | AC | C |    |   |   |   |   |   |   |   |   |   |
| —                | —   | —   | —   | —  | √  | —  | — |    |   |   |   |   |   |   |   |   |   |
| <b>DAA [m]</b>   | 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 $ACC.3 \sim ACC.0 > 9$ or $AC=1$<br>then $[m].3 \sim [m].0 \leftarrow (ACC.3 \sim ACC.0) + 6, AC1 = \overline{AC}$<br>else $[m].3 \sim [m].0 \leftarrow (ACC.3 \sim ACC.0), AC1 = 0$<br>and<br>If $ACC.7 \sim ACC.4 + AC1 > 9$ or $C=1$<br>then $[m].7 \sim [m].4 \leftarrow ACC.7 \sim ACC.4 + 6 + AC1, C=1$<br>else $[m].7 \sim [m].4 \leftarrow ACC.7 \sim ACC.4, C=C$  |     |     |    |    |    |   |    |   |   |   |   |   |   |   |   |   |
| Affected flag(s) | <table><tr><td>TC2</td><td>TC1</td><td>TO</td><td>PD</td><td>OV</td><td>Z</td><td>AC</td><td>C</td></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>√</td></tr></table>   | TC2 | TC1 | TO | PD | OV | Z | AC | C | — | — | — | — | — | — | — | √ |
| TC2              | TC1   | TO  | PD  | OV | Z  | AC | C |    |   |   |   |   |   |   |   |   |   |
| —                | —   | —   | —   | —  | —  | —  | √ |    |   |   |   |   |   |   |   |   |   |
| <b>DEC [m]</b>   | Decrement data memory   |     |     |    |    |    |   |    |   |   |   |   |   |   |   |   |   |
| Description      | Data in the specified data memory is decremented by 1.  |     |     |    |    |    |   |    |   |   |   |   |   |   |   |   |   |
| Operation        | $[m] \leftarrow [m] - 1$  |     |     |    |    |    |   |    |   |   |   |   |   |   |   |   |   |
| Affected flag(s) |   |     |     |    |    |    |   |    |   |   |   |   |   |   |   |   |   |

### DECA [m]

Description

Decrement data memory and place result in the accumulator

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

$ACC \leftarrow [m] - 1$

Affected flag(s)

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

### HALT

Description

Enter power down mode

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

$PC \leftarrow PC + 1$

$PD \leftarrow 1$

$TO \leftarrow 0$

Affected flag(s)

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

### INC [m]

Description

Increment data memory

Data in the specified data memory is incremented by 1.

Operation

$[m] \leftarrow [m] + 1$

Affected flag(s)

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

### INCA [m]

Description

Increment data memory and place result in the accumulator

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

$ACC \leftarrow [m] + 1$

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  $PC \leftarrow \text{addr}$

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  $ACC \leftarrow [m]$

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  $ACC \leftarrow x$

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  $[m] \leftarrow ACC$

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  $PC \leftarrow PC+1$

Affected flag(s)

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

### OR A,[m]

Description

Logical OR accumulator with data memory

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

$ACC \leftarrow ACC \text{ "OR" } [m]$

Affected flag(s)

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

### OR A,x

Description

Logical OR immediate data to the accumulator

Data in the accumulator and the specified data perform a bitwise logical\_OR operation. The result is stored in the accumulator.

Operation

$ACC \leftarrow ACC \text{ "OR" } x$

Affected flag(s)

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

### ORM A,[m]

Description

Logical OR data memory with the accumulator

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

$[m] \leftarrow ACC \text{ "OR" } [m]$

Affected flag(s)

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

### RET

Description

Return from subroutine

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

Operation

$PC \leftarrow \text{Stack}$

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  $PC \leftarrow \text{Stack}$   
 $ACC \leftarrow x$

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  $PC \leftarrow \text{Stack}$   
 $EMI \leftarrow 1$

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 1 bit left with bit 7 rotated into bit 0.

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

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

Affected flag(s)

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

|                |  |
|----------------|--|
| <b>RLC [m]</b> | 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      | $[m].(i+1) \leftarrow [m].i$ ; $[m].i$ :bit i of the data memory (i=0~6)<br>$[m].0 \leftarrow C$<br>$C \leftarrow [m].7$   |

Affected flag(s)

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

|                 |   |
|-----------------|---|
| <b>RLCA [m]</b> | 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       | $ACC.(i+1) \leftarrow [m].i$ ; $[m].i$ :bit i of the data memory (i=0~6)<br>$ACC.0 \leftarrow C$<br>$C \leftarrow [m].7$  |

Affected flag(s)

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

|               |  |
|---------------|--|
| <b>RR [m]</b> | 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     | $[m].i \leftarrow [m].(i+1)$ ; $[m].i$ :bit i of the data memory (i=0~6)<br>$[m].7 \leftarrow [m].0$ |

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 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**  $ACC.(i) \leftarrow [m].(i+1)$ ;  $[m].i$ :bit i of the data memory (i=0~6)  
 $ACC.7 \leftarrow [m].0$

**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 1 bit right. Bit 0 replaces the carry bit; the original carry flag is rotated into the bit 7 position.

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

**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 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**  $ACC.i \leftarrow [m].(i+1)$ ;  $[m].i$ :bit i of the data memory (i=0~6)  
 $ACC.7 \leftarrow C$   
 $C \leftarrow [m].0$

**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  $ACC \leftarrow ACC + [\bar{m}] + C$

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  $[m] \leftarrow ACC + [\bar{m}] + C$

Affected flag(s)

| TC2 | TC1 | TO | PD | 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 (two cycles). Otherwise proceed with the next instruction (one cycle).

Operation Skip if  $([m]-1)=0$ ,  $[m] \leftarrow ([m]-1)$

Affected flag(s)

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

**SDZA [m]** Decrement data memory and place result in ACC, skip if 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  $([m]-1)=0$ ,  $ACC \leftarrow ([m]-1)$

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 1.

Operation

$[m] \leftarrow FFH$

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 1.

Operation

$[m].i \leftarrow 1$

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 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  $([m]+1)=0$ ,  $[m] \leftarrow ([m]+1)$

Affected flag(s)

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

### SIZA [m]

Increment data memory and place result in ACC, skip if 0

Description

The contents of the specified data memory are incremented by 1. 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 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  $([m]+1)=0$ ,  $ACC \leftarrow ([m]+1)$

Affected flag(s)

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

### SNZ [m].i

Description

Skip if bit "i" of the data memory is not 0

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 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)

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

### SUB A,[m]

Description

Subtract data memory from the accumulator

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

Operation

$ACC \leftarrow ACC + \overline{[m]} + 1$

Affected flag(s)

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

### SUBM A,[m]

Description

Subtract data memory from the accumulator

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

Operation

$[m] \leftarrow ACC + \overline{[m]} + 1$

Affected flag(s)

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

### SUB A,x

Description

Subtract immediate data from the accumulator

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

Operation

$ACC \leftarrow ACC + \overline{x} + 1$

Affected flag(s)

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

### SWAP [m]

Description

Swap nibbles within the data memory

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

Operation

$[m].3 \sim [m].0 \leftrightarrow [m].7 \sim [m].4$

Affected flag(s)

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

### SWAPA [m]

Description

Swap data memory-place result in the accumulator

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

$ACC.3 \sim ACC.0 \leftarrow [m].7 \sim [m].4$   
 $ACC.7 \sim ACC.4 \leftarrow [m].3 \sim [m].0$

Affected flag(s)

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

### SZ [m]

Description

Skip if data memory is 0

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 (2 cycle).

Operation

Skip if  $[m]=0$

Affected flag(s)

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

### SZA [m]

Description

Move data memory to ACC, skip if 0

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  $[m]=0$ ,  $ACC \leftarrow [m]$

Affected flag(s)

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

### SZ [m].i

#### Description

Skip if bit "i" of the data memory is zero

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)

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

### TABRDC [m]

#### Description

Move the ROM code (current page) to TBLH and data memory

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]

#### Description

Move the ROM code (last page) to TBLH and data memory

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]

#### Description

Logical XOR accumulator with data memory

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 0 flag is affected.

Operation

$[m] \leftarrow \text{ACC} \text{ "XOR" } [m]$

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 0 flag is affected.

Operation

$\text{ACC} \leftarrow \text{ACC} \text{ "XOR" } x$

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

Copyright © 2000 by HOLTEK SEMICONDUCTOR INC.

The information appearing in this Data Sheet is believed to be accurate at the time of publication. However, Holtek assumes no responsibility arising from the use of the specifications described. The applications mentioned herein are used solely for the purpose of illustration and Holtek makes no warranty or representation that such applications will be suitable without further modification, nor recommends the use of its products for application that may present a risk to human life due to malfunction or otherwise. Holtek reserves the right to alter its products without prior notification. For the most up-to-date information, please visit our web site at <http://www.holtek.com.tw>.