EE2016: Microprocessor Lab

Experiment 9: Interrupts in Atmel AVR Atmega through Assembly Programming

Ajeet E S, EE22B086
Amogh Agrawal, EE22B087

June 5, 2024

Contents

List of Listings

1 Aim

Using Atmel AVR assembly language programming, implement interrupts and DIP switches control in Atmel Atmega microprocessor.

Aims of this experiment are:

1. Generate an external (logical) hardware interrupt using an emulation of a push button switch.
2. Write an ISR (Interrupt Service Routine) to switch ON an LED for a few seconds (10 secs) and then switch OFF. (The lighting of the LED could be verifed by monitoring the signal to switch it ON).
3. If there is time, you could try this also: Use the 16 bit timer (via interrupt) to make an LED blink with a duration of 1 second.

Also, one needs to implement all of the above, in AVR assembly.

2 Observations and Code: Google Drive Link

The link to the codes and video for the Experiment are uploaded here:
Code and Video

3 Problem 1: Implement Interrupt using INT1

3.1 Problem Statement

You are given the file "EE2016F23Exp9int1.asm" which is an assembly program which implements interrupt using INT1. Fill in the blanks in the assembly code.

3.2 Approach

- The ISR for INT1 is set at the label int1_ISR by using the following command (ISR for INT1 is set in the memory address 0x04):

```
.ORG 0x0004 ;
RJMP int1_ISR;
```

- At the program origin, we make the program jump to the label reset. Here, the following are done:

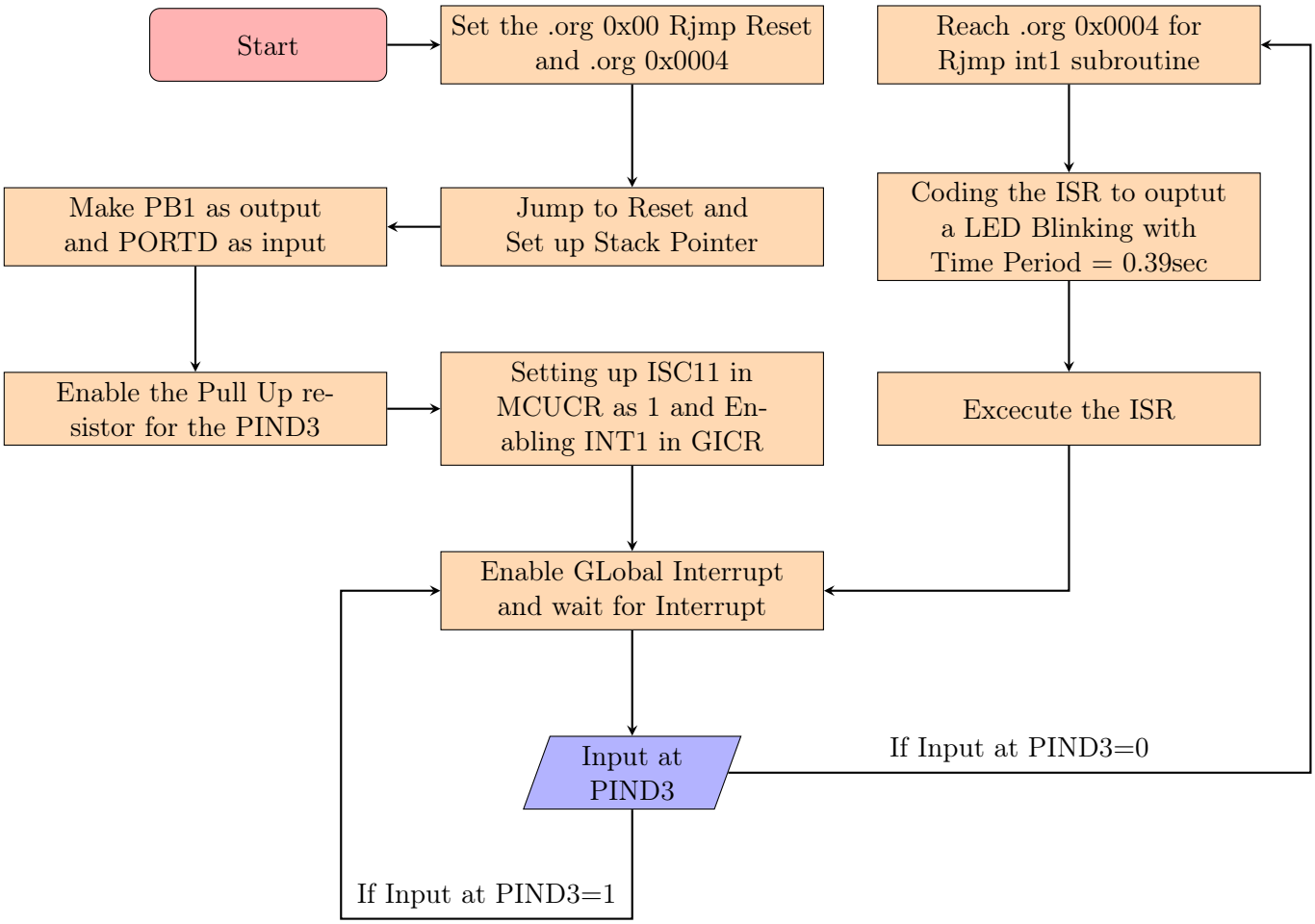
  1. Store the memory address of RAMEND to Stack Pointer.
  2. Make PB1 as output and make all of PortD as input and take the input from PIND3, enabling internal pull-up resistor.
  3. Set ISC11 in MCUCR to 1, that is, make a falling edge of interrupt pin generate an interrupt.
  4. Enable Interrupt for INT1 in GICR.
  5. Enable all interrupts using SEI.

- The program waits at the label "indefiniteloop", waiting for an interrupt:

```
indefiniteloop: RJMP indefiniteloop
```

- In the "int1_ISR" Routine, the LED is made to blink 10 times with the duty cycle as given in the code.

3.3 Flowchart



3.4 Code

The code is given below:

```
    .nolist ; turn list file generation OFF

    .include "m8def.inc"
    .list /* turn it ON again */

    .ORG 0 ; set program origin
    RJMP reset ; on reset, program starts here

    .ORG 0x0004 ;
    RJMP int1_ISR;

reset:
```

```
    LDI R16, LOW(RAMEND) ;
    OUT SPL, R16
    LDI R16, HIGH(RAMEND) /* Guess, why it is done ??? */
    OUT SPH, R16
    LDI R16, 0x02 ; make PB1 OUTput
    OUT DDRB, R16; /* fill in here */
    LDI R16, 0x00 ; fill in here
    OUT DDRD, R16 ; make PORTD input
    LDI R16,0x08 ; /* enable internal pull-up resistor. This avoids the high impe */
    OUT PORTD, R16 ; /* -dance state while reading data from external world */
    IN R16, MCUCR ;
    ORI R16, 1<<ISC11 ; why it is Ored?
    OUT MCUCR, R16 ;
    IN R16, GICR ; enable INT1 interrupt
    ORI R16, 1<<INT1 ;
    OUT GICR, R16 ; /* fill in here */
    LDI R16, 0x00 ;
    OUT PORTB, R16 ;
    SEI ; /* what does it do? */
indefiniteloop: RJMP indefiniteloop /* Stay here. Expecting interrupt? */

    /* reset - the main - loop ends here */
int1_ISR: ; INT1 interrupt handler or ISR
    IN R16, SREG ; save status register SREG
    PUSH R16 ; /* Fill in here. save the status register contents into the stack memory. */
    /* StckPntr decremented. StckPntr tracks the lower end of ACTIVE stack. PC is */
    /* PUSHed automatically, by default. No need for explicit instruction */

    LDI R16,0x0a ; blink led 10 times by storing R0 a value of 10 & decrementing
    MOV R0, R16 /* to zero realises the LED blinking 10 times */
back5:

    LDI R16,0x02 ; Turn on LED
    OUT PORTB, R16 /* LED connected to penultimate bit (B1) of PORTB */

delay1:
    LDI R16,0xFF ; delay
back2:
    LDI R17,0xFF ; /* back2 loop starts.. adds delay */
back1:
    DEC R17 /* fill in. Innermost delay loop. Each execution cycle is of T_clk */
    BRNE back1 /* branch if not equal - means go on in loop till R* goes 0 */
```

```
    DEC R16 /* for each inner loop run, an equal amount of delay in OUTer loop */
    BRNE back2 /* how many clock cycles for ON period? */
    /* Till this time LED is ON. First part of duty cycle ends */
    LDI R16,0x00 ; Turn off LED
    OUT PORTB, R16; /* fill-in here */

delay2:
    LDI R16,0xFF ; delay - OFF period. Second part of duty cycle starts
back3:
    LDI R17,0xFF
back4:
    DEC R17
    BRNE back4
    DEC R16
    BRNE back3
    DEC R0 /* Fill in here. Initially R0 = 10. Completes ONE duty cycle */
BRNE back5 ; /* R0-- till 0. 10 times blinking */
    POP R16 ; retrieve status register. The stack's lower end is incremented
    OUT SREG, R16 /* meaning stckPntr++; In "pop R16" instruction, the topend
    /* stack location's value is espewed OUT and is stored IN R16 */
    RETI ; go back to main program and set I = 1 (enabling interrupts as the current ISR is executed
```

Listing 1: Code to Blink LED using INT1

3.5 Questions From Code

Q: `LDI R16, HIGH(RAMEND);` Guess, why it is done ???
A: This is store the top 8 bits of the address of the end of data memory.

Q: `ORI R16, 1<< ISC11;` Why it is Ored?
A: It is Ored to not change the rest of the bits of R16 except the bit at the same position as that of ICS11 in the MCUCR which is set to 1.

Q: `SEI;` What does it do?
A: It sets the global interrupt enable in SREG Register as 1.

Q: `indefiniteloop: rjmp indefiniteloop;` Stay here. Expecting interrupt?
A: Yes. The program stays here, waiting for an interrupt.

Q: What is the Duty Cycle and period of the LED Blinking?
A: Duty cycle is 50% while the LED blinking period is 390158 cycles, equivalent to .39sec.

3.6 Inferences

3.6.1 Interrupts

- It is a method used to serve a device using a microcontroller.

- In this, the device sends an interrupt signal to the microcontroller, upon which the microcontroller pauses all other processes and serves the device.
- This is better than Polling, in which the microcontroller keeps monitoring the output of a device in the fact that it doesn't waste time monitoring all devices, and makes the rest of the program wait only when a request is made.

3.6.2 Interrupt Service Routine(ISR):

- The ISR is the Routine that is called when an interrupt is raised.
- This routine is called only if Global Interrupt Enable is set using the SEI Command
- When an interrupt request occurs for INT1, the program goes to the memory address 0x04, the interrupt vector for the same

3.6.3 Interrupt Vector:

- The Microcontroller checks in a memory address called Interrupt Vector, to find the memory address at which the ISR is stored.
- The Interrupt Vectors for INT0, INT1 and INT2 are memory addresses 0x02,0x04 and 0x06
- Whenever an Interrupt occurs, the program goes to these memory addresses and searches for the memory address for the ISR.

3.6.4 Taking Care of Interrupts

Whenever an Interrupt Signal is received, the microcontroller does the following:

1. Finishes the instruction it is currently executing and store the address of the next instruction in the PC.
2. Jumps to the Interrupt Vector Table, which redirects the microcontroller to the ISR.
3. Executes the ISR until it encounters RETI, upon which it goes to the next instruction to be executed and continues normally.

3.6.5 Enabling Interrupts and Choosing Interrupt Signal

The following are important Registers in relation to Interrupts:

1. **SREG: Status Register**

Bit	D7							D0
SREG	I	T	H	S	V	N	Z	C

C – Carry flag
Z – Zero flag
N – Negative flag
V – Overflow flag
S – Sign flag
H – Half carry
T – Bit copy storage
I – Global Interrupt Enable

- Global Interrupt Enable (I), the 7th bit, must be set for all interrupts to be accepted.

2. **MCUCR: MCU Control Register**

SE	SM2	SM1	SM0	ISC11	ISC10	ISC01	ISC00

- The values of ISC11, ISC10, ISC01, ISC00 and the purposes are given below:

ISC01, ISC00 (Interrupt Sense Control bits) These bits define the level or edge on the external INT0 pin that activates the interrupt, as shown in the following table:

ISC01	ISC00		Description
0	0		The low level of INT0 generates an interrupt request.
0	1		Any logical change on INT0 generates an interrupt request.
1	0		The falling edge of INT0 generates an interrupt request.
1	1		The rising edge of INT0 generates an interrupt request.

ISC11, ISC10 These bits define the level or edge that activates the INT1 pin.

ISC11	ISC10		Description
0	0		The low level of INT1 generates an interrupt request.
0	1		Any logical change on INT1 generates an interrupt request.
1	0		The falling edge of INT1 generates an interrupt request.
1	1		The rising edge of INT1 generates an interrupt request.

3. **MCUCSR: MCU Control and Status Register**

- The value of ISC2 in this register. This bit is used to choose which level and edge is taken as an interrupt.

JTD	ISC2	-	JTRF	WDRF	BORF	EXTRF	PORF

ISC2 This bit defines whether the INT2 interrupt activates on the falling edge or the rising edge.

ISC2		Description
0		The falling edge of INT2 generates an interrupt request.
1		The rising edge of INT2 generates an interrupt request.

4 Problem 2: Implement Interrupt using INT0

4.1 Problem Statement

Perform 4-bit addition of two unsigned nibbles from an 8-bit dip input switch (set by TAs) and display the result obtained in LEDs.

4.2 Approach

Essentially, this code uses the same logic as the previous one, but the only difference is that this operates on INT0 and not INT1.

- The ISR for INT0 is set at the label int0_ISR by using the following command (ISR for INT0 is set in the memory address 0x02):

```
.ORG 0x0002 ;
RJMP int0_ISR;
```

- At the program origin, we make the program jump to the label reset. Here, the following are done:

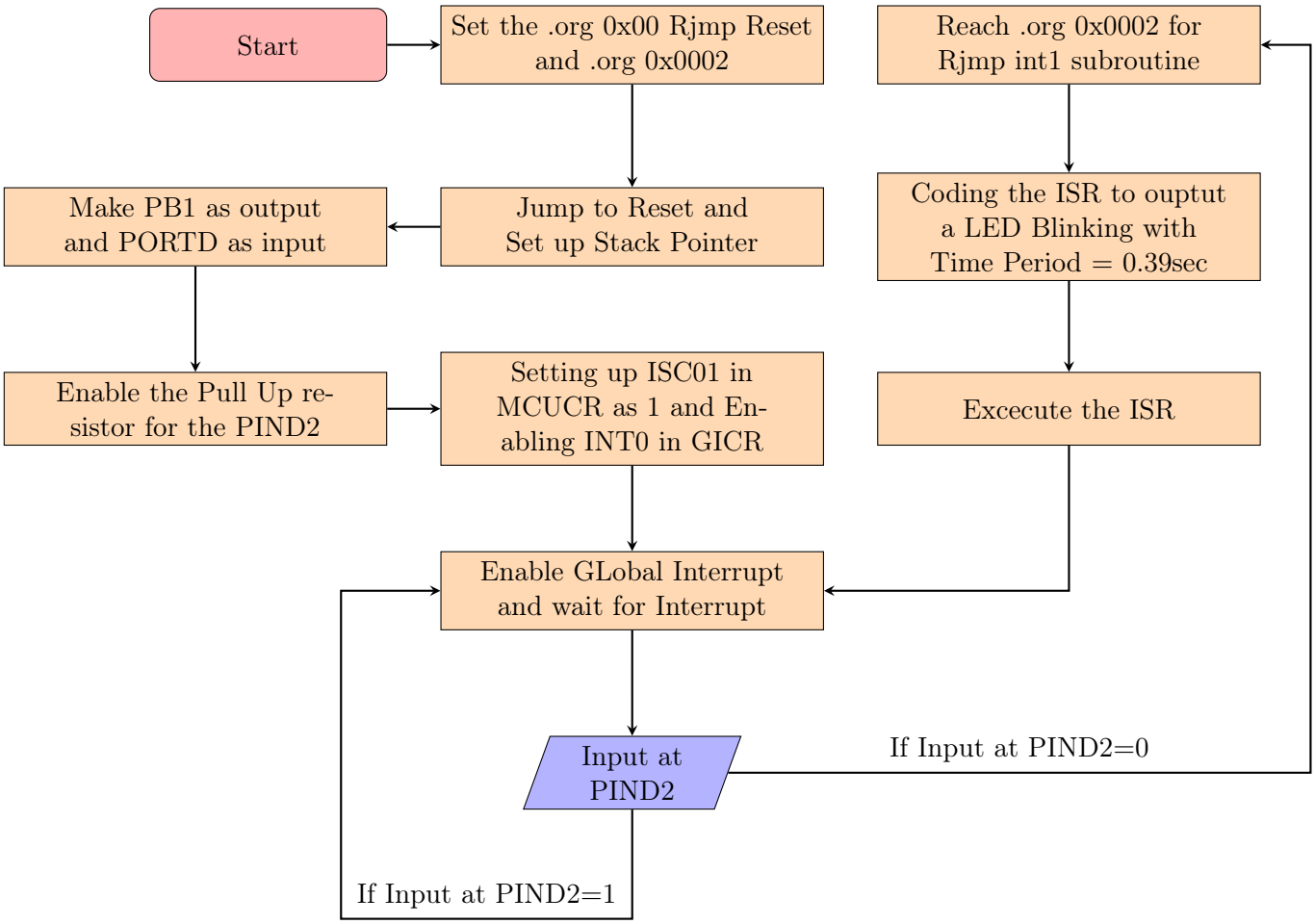
  1. Store the memory address of RAMEND to Stack Pointer.
  2. Make PB1 as output and make all of PortD as input and take the input from PIND2, enabling internal pull-up resistor.
  3. Set ISC01 in MCUCR to 1, that is, make a falling edge of interrupt pin generate an interrupt.
  4. Enable Interrupt for INT0 in GICR.
  5. Enable all interrupts using SEI.

- The program waits at the label "indefiniteloop", waiting for an interrupt:

```
indefiniteloop: RJMP indefiniteloop
```

- In the "int0_ISR" Routine, the LED is made to blink 10 times

4.3 Flowchart



4.4 Code

The code to perform 4- bit addition is given below:

```
    .nolist ; turn list file generation OFF

    .include "m8def.inc"
    .list /* turn it ON again */

    .ORG 0 ; set program origin
    RJMP reset ; on reset, program starts here

    .ORG 0x0002 ;
    RJMP int0_ISR;

reset:
    LDI R16, LOW(RAMEND) ;
    OUT SPL, R16
    LDI R16, HIGH(RAMEND) /* Guess, why it is done ??? */
    OUT SPH, R16
    LDI R16, 0x02 ; make PB1 OUTput
    OUT DDRB, R16; /* fill in here */
    LDI R16, 0x00 ; fill in here
```

```
    OUT DDRD, R16 ; make PORTD input
    LDI R16,0x04 ; /* enable internal pull-up resistor. This avoids the high impe */
    OUT PORTD, R16 ; /* -dance state while reading data from external world */
    IN R16, MCUCR ;
    ORI R16, 1<<ISC01 ; why it is Ored?
    OUT MCUCR, R16 ;
    IN R16, GICR ; enable INT0 interrupt
    ORI R16, 1<<INT0 ;
    OUT GICR, R16 ; /* fill in here */
    LDI R16, 0x00 ;
    OUT PORTB, R16 ;
    SEI ; /* what does it do? */
indefiniteloop: RJMP indefiniteloop /* Stay here. Expecting interrupt? */

 /* reset - the main - loop ends here */
int0_ISR: ; INT1 interrupt handler or ISR
    IN R16, SREG ; save status register SREG
    PUSH R16 ; /* Fill in here. save the status register contents into the stack memory. */
    /* StckPntr decremented. StckPntr tracks the lower end of ACTIVE stack. PC is */
    /* PUSHed automatically, by default. No need for explicit instruction */

    LDI R16,0x0a ; blink led 10 times by storing R0 a value of 10 & decrementing
    MOV R0, R16 /* to zero realises the LED blinking 10 times */
back5:
    LDI R16,0x02 ; Turn on LED
    OUT PORTB, R16 /* LED connected to penultimate bit (B1) of PORTB */

delay1:
    LDI R16,0xFF ; delay
back2:
    LDI R17,0xFF ; /* back2 loop starts.. adds delay */
back1:
    DEC R17 /* fill in. Innermost delay loop. Each execution cycle is of T_clk */
    BRNE back1 /* branch if not equal - means go on in loop till R* goes 0 */
    DEC R16 /* for each inner loop run, an equal amount of delay in OUTer loop */
    BRNE back2 /* how many clock cycles for ON period? */
    /* Till this time LED is ON. First part of duty cycle ends */
    LDI R16,0x00 ; Turn off LED
    OUT PORTB, R16; /* fill-in here */

delay2:
```

```
    LDI R16,0xFF ; delay - OFF period. Second part of duty cycle starts
back3:
    LDI R17,0xFF
back4:
    DEC R17
    BRNE back4
    DEC R16
    BRNE back3
    DEC R0 /* Fill in here. Initially R0 = 10. Completes ONE duty cycle */
BRNE back5 ; /* R0-- till 0. 10 times blinking */

    POP R16 ; retrieve status register. The stack's lower end is incremented
    OUT SREG, R16 /* meaning stckPntr++; In "pop R16" instruction, the topend
    /* stack location's value is espewed OUT and is stored IN R16 */
    RETI ; go back to main program and set I = 1 (enabling interrupts as the current ISR is executed)

```

Listing 2: Code to Blink LED using INT0

4.5 Questions From Code

Q: `LDI R16, HIGH(RAMEND);` Guess, why it is done ???
A: This is store the top 8 bits of the address of the end of data memory.

Q: `ORI R16, 1<< ISC01;` Why it is Ored?
A: It is Ored to not change the rest of the bits of R16 except the bit at the same position as that of ICS11 in the MCUCR which is set to 1.

Q: `SEI;` What does it do?
A: It sets the global interrupt enable in SREG Register as 1.

Q: `indefiniteloop: rjmp indefiniteloop;` Stay here. Expecting interrupt?
A: Yes. The program stays here, waiting for an interrupt.

Q: What is the Duty Cycle and period of the LED Blinking?
A: Duty cycle is 50% while the LED blinking period is 390158 cycles, equivalent to .39sec.

4.6 Inferences

4.6.1 Interrupts

- It is a method used to serve a device using a microcontroller.
- In this, the device sends an interrupt signal to the microcontroller, upon which the microcontroller pauses all other processes and serves the device.
- This is better than Polling, in which the microcontroller keeps monitoring the output of a device in the fact that it doesn't waste time monitoring all devices, and makes the rest of the program wait only when a request is made.

4.6.2 Interrupt Service Routine(ISR):

- The ISR is the Routine that is called when an interrupt is raised.
- This routine is called only if Global Interrupt Enable is set using the SEI Command
- When an interrupt request occurs for INT1, the program goes to the memory address 0x04, the interrupt vector for the same

4.6.3 Interrupt Vector:

- The Microcontroller checks in a memory address called Interrupt Vector, to find the memory address at which the ISR is stored.
- The Interrupt Vectors for INT0, INT1 and INT2 are memory addresses 0x02,0x04 and 0x06
- Whenever an Interrupt occurs, the program goes to these memory addresses and searches for the memory address for the ISR.

4.6.4 Taking Care of Interrupts

Whenever an Interrupt Signal is received, the microcontroller does the following:

1. Finishes the instruction it is currently executing and store the address of the next instruction in the PC.
2. Jumps to the Interrupt Vector Table, which redirects the microcontroller to the ISR.
3. Executes the ISR until it encounters RETI, upon which it goes to the next instruction to be executed and continues normally.

4.6.5 Enabling Interrupts and Choosing Interrupt Signal

The following are important Registers in relation to Interrupts:

1. **SREG: Status Register**

Bit	D7							D0
SREG	I	T	H	S	V	N	Z	C

C – Carry flag
Z – Zero flag
N – Negative flag
V – Overflow flag
S – Sign flag
H – Half carry
T – Bit copy storage
I – Global Interrupt Enable

   - Global Interrupt Enable (I), the 7th bit, must be set for all interrupts to be accepted.

2. **MCUCR: MCU Control Register**

SE	SM2	SM1	SM0	ISC11	ISC10	ISC01	ISC00

- The values of ISC11, ISC10, ISC01, ISC00 and the purposes are given below:

ISC01, ISC00 (Interrupt Sense Control bits) These bits define the level or edge on the external INT0 pin that activates the interrupt, as shown in the following table:

ISC01	ISC00		Description
0	0		The low level of INT0 generates an interrupt request.
0	1		Any logical change on INT0 generates an interrupt request.
1	0		The falling edge of INT0 generates an interrupt request.
1	1		The rising edge of INT0 generates an interrupt request.

ISC11, ISC10 These bits define the level or edge that activates the INT1 pin.

ISC11	ISC10		Description
0	0		The low level of INT1 generates an interrupt request.
0	1		Any logical change on INT1 generates an interrupt request.
1	0		The falling edge of INT1 generates an interrupt request.
1	1		The rising edge of INT1 generates an interrupt request.

3. **MCUCSR: MCU Control and Status Register**

- The value of ISC2 in this register. This bit is used to choose which level and edge is taken as an interrupt.

JTD	ISC2	-	JTRF	WDRF	BORF	EXTRF	PORF

ISC2 This bit defines whether the INT2 interrupt activates on the falling edge or the rising edge.

ISC2		Description
0		The falling edge of INT2 generates an interrupt request.
1		The rising edge of INT2 generates an interrupt request.