Fab Lab Ismailia represent :

Embedded Systems Workshop

by : Mohammed hemed

4- INTERRUPT

INTERRUPT

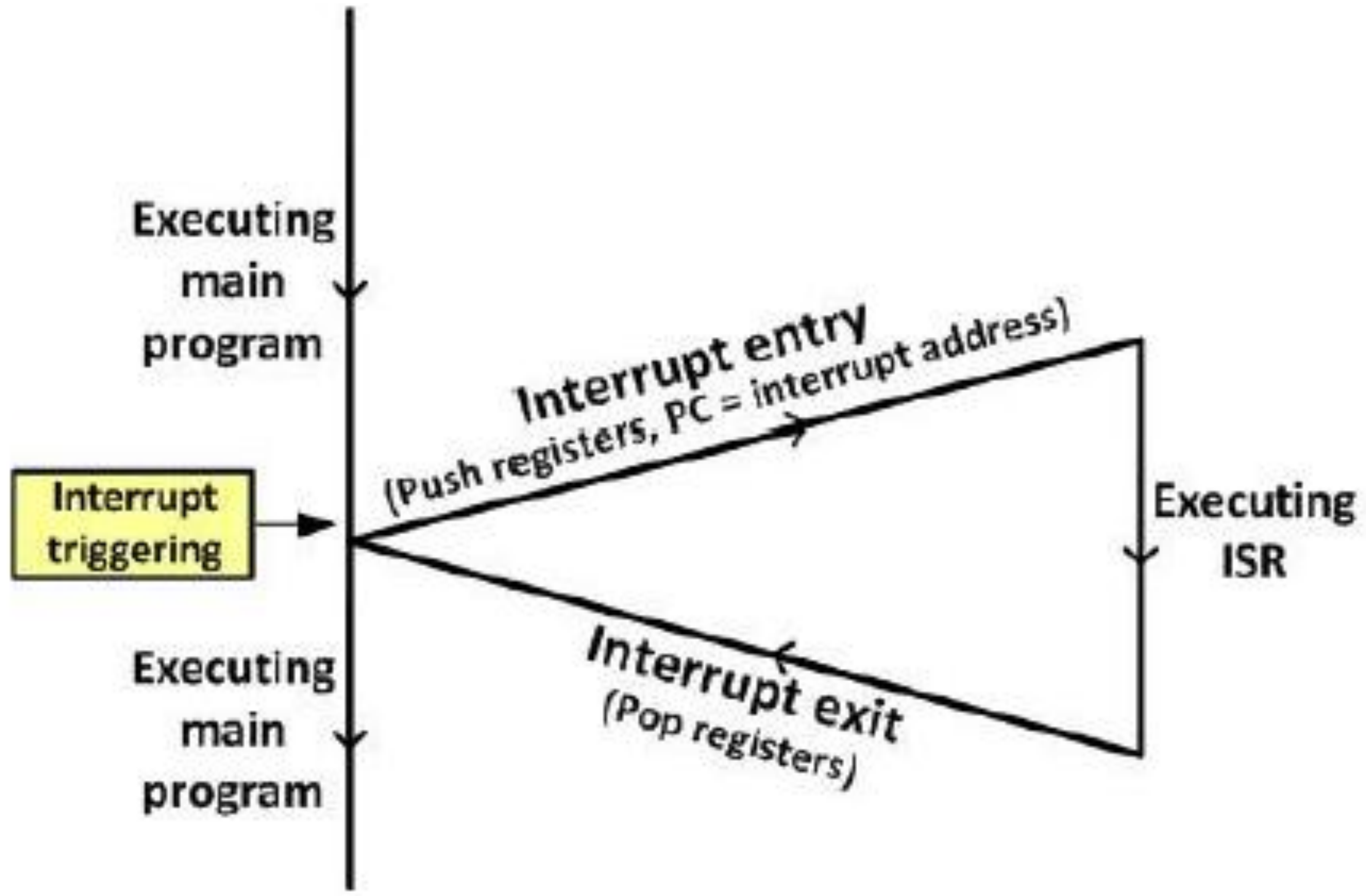


Figure 6-5: Main Program gets Interrupted

Ways of serving peripherals

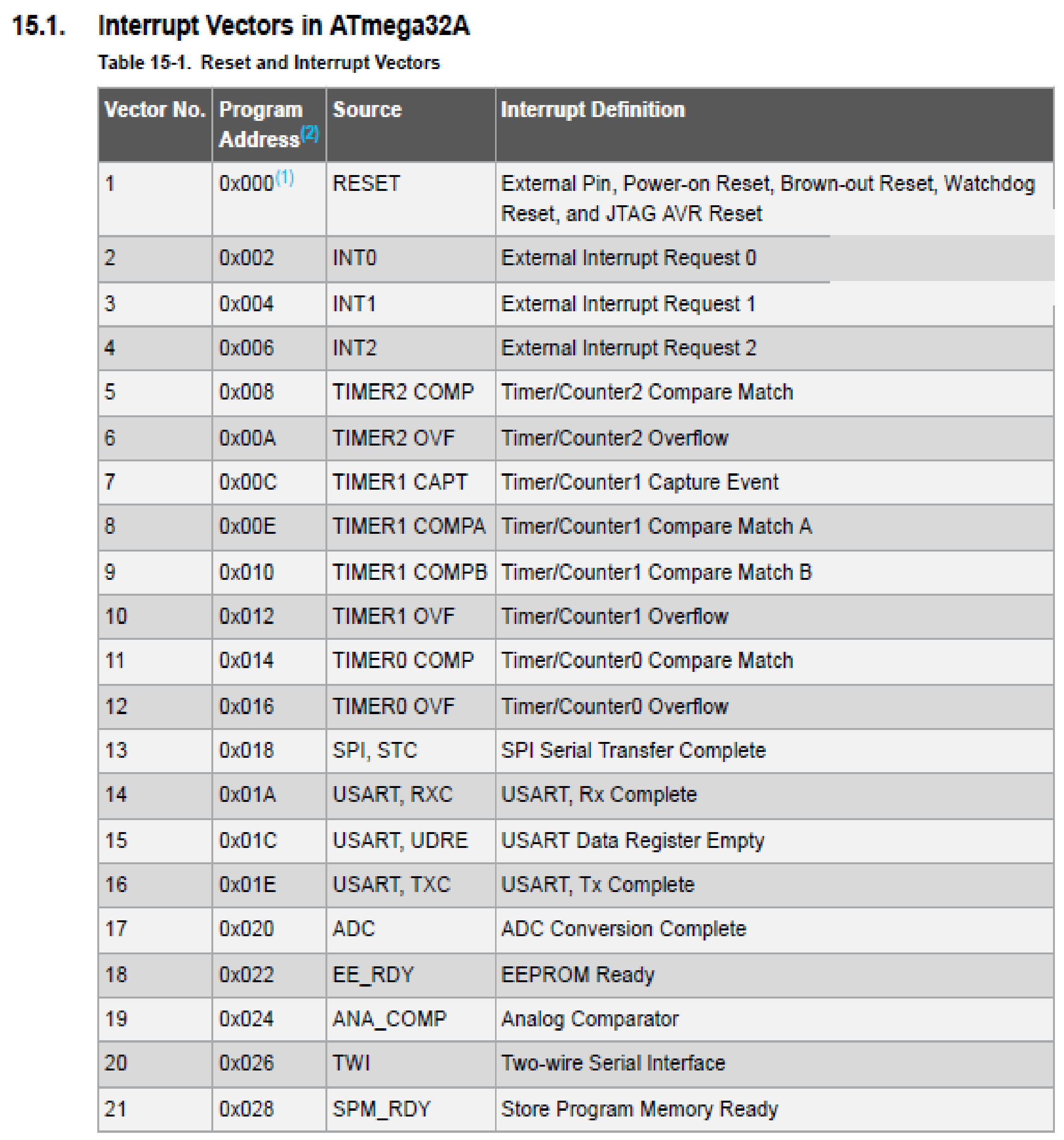
- A single uC can serve several peripherals (devices), there are two methods by which peripherals receive service : Interrupt - polling .
- In Interrupt method , whenever any device needs a uC service , the device notifies the uC by sending an interrupt signal , when uC receiving interrupt signal , uC stops whatever it’s doing and serves the device . The program associated with the interrupt is called Interrupt Service Routine (ISR), or interrupt handler .
- In Polling : the uC continuously monitors the status of a given device , when the status condition is met , it performs the service , after that it moved on to monitor the next device until each one is serviced .

why polling isn’t efficient ?
why the interrupt is better ?

- via Interrupt uC can serve several devices based on priority assigned to it , polling can’t do that as it checks all devices in a round-robin fashion .
- via interrupt uC can ignore (mask) a device request for service , polling can’t.
- via interrupt uC can do many other things (even if sleeping to save power consumption) , instead of tying down the uC to keep monitor all devices , wasting CPU time - Power .

Interrupt Service Routine (ISR)

- For every Interrupt type there is a fixed location in memory that holds the address of its ISR , the group of memory locations set aside to hold the addresses of ISRs is called the interrupt vector table .
- From page 61 in data sheet



15.1. Interrupt Vectors in ATmega32A

Table 15-1. Reset and Interrupt Vectors

Vector No.	Program Address[2]	Source	Interrupt Definition
1	0x000[1]	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	0x002	INT0	External Interrupt Request 0
3	0x004	INT1	External Interrupt Request 1
4	0x006	INT2	External Interrupt Request 2
5	0x008	TIMER2 COMP	Timer/Counter2 Compare Match
6	0x00A	TIMER2 OVF	Timer/Counter2 Overflow
7	0x00C	TIMER1 CAPT	Timer/Counter1 Capture Event
8	0x00E	TIMER1 COMPA	Timer/Counter1 Compare Match A
9	0x010	TIMER1 COMPB	Timer/Counter1 Compare Match B
10	0x012	TIMER1 OVF	Timer/Counter1 Overflow
11	0x014	TIMER0 COMP	Timer/Counter0 Compare Match
12	0x016	TIMER0 OVF	Timer/Counter0 Overflow
13	0x018	SPI, STC	SPI Serial Transfer Complete
14	0x01A	USART, RXC	USART, Rx Complete
15	0x01C	USART, UDRE	USART Data Register Empty
16	0x01E	USART, TXC	USART, Tx Complete
17	0x020	ADC	ADC Conversion Complete
18	0x022	EE_RDY	EEPROM Ready
19	0x024	ANA_COMP	Analog Comparator
20	0x026	TWI	Two-wire Serial Interface
21	0x028	SPM_RDY	Store Program Memory Ready

Stack

- stack is a section of RAM used by the CPU to store information temporarily , this information could be data or an address . The CPU need this storage area because there are only a limited number of registers .

- The stack is used In interrupts and function calls :

- Saving the (Return address) address of the next instruction after returning from a function or ISRs.
- Saving the Status register (Flag Register)and GPR (General Purpose Registers) contents otherwise the ISR or function call will override them , resulting in unpredictable results .

what happened in executing an interrupt ?

- Activate the interrupt .
- uC finishes the current instruction and save the address of the next instruction Program Counter (PC) contents on the stack .
- uC jumps to a fixed location in memory (Interrupt vector table) which directs the uC to address of ISR .
- uC starts to execute ISR until it reaches the last instruction of the subroutine which is return from interrupt (RETI).
- uC return to the place where it interrupted . First it gets PC address from the stack then it continue executing program .

Sources of interrupts in AVR

- One External Reset interrupt :

- Reset in Embedded is different from Restart in the computers (which switch off the power and switch on again) , but Reset in uC mean : interrupt whatever the uC do and go to the first instruction in PC (0x00) , as well as making all registers = zeros , but the power still on , the reason for designing the reset in this way :

if turn off power and turn on again , it take about 60 ms , and it's a big

number in the Embedded world , where reset take less than 1 uSec which is faste

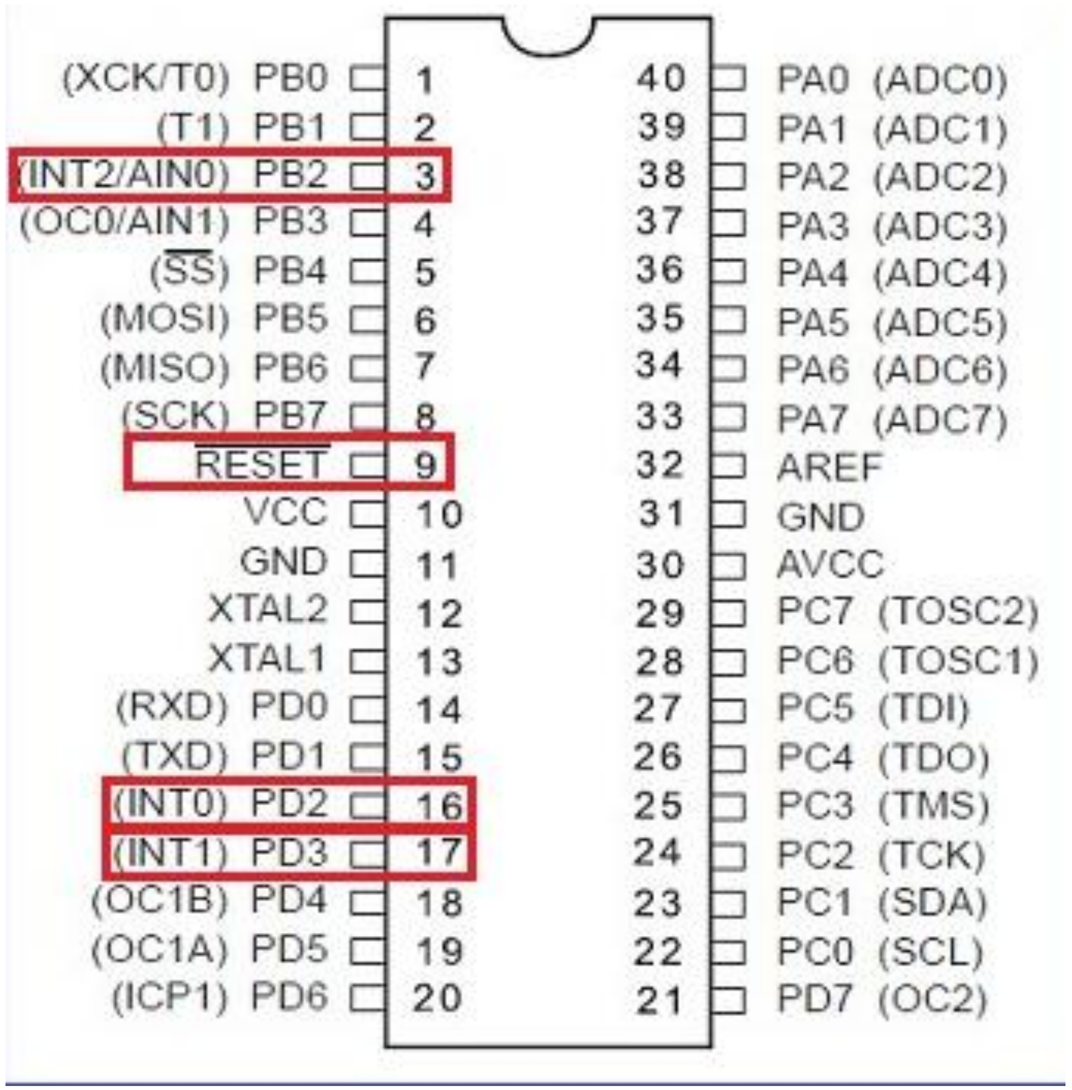
than restart about 60,000 :)

- Two interrupts set aside for each of the timers
- Three interrupts are set aside for external hardware interrupts :

- Pins PD2 – PD3 from PORTD , PB2 from PORTB = (INT0 , INT1 , INT2) respectively .

- Serial communication's USART (one for receive - two fpr transmit)
- SPI – I2C
- ADC

External interrupts



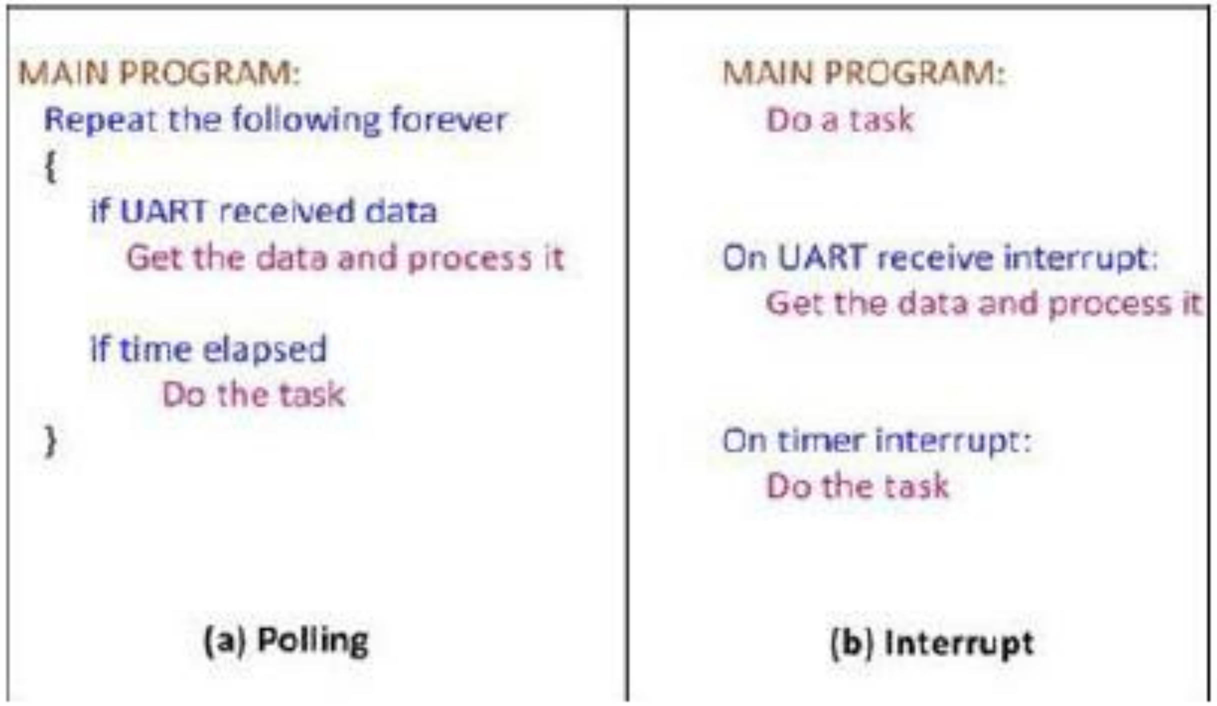
External Interrupts

- In Atmega32 we have three external interrupts ,

when a digital signal come to these pins the interrupt will happen .

- Steps of Enabling the External Interrupt :

- Set the pins to be used in interrupt as input from DDRx
- Set signal type which cause the interrupt according to sensor type or button which generates the interrupt .
- Enable receiving interrupt signal on the required pin : INT0 - INT1
- Enable the interrupt in general
- Write the ISR



```
MAIN PROGRAM:
  Repeat the following forever
  {
     if UART received data
        Get the data and process it

     If time elapsed
           Do the task
  }
```

(a) Polling

```
MAIN PROGRAM:
   Do a task

On UART receive interrupt:
   Get the data and process it

On timer interrupt:
   Do the task
```

(b) Interrupt

Edge-triggered vs level-triggered Interrupts

- There are two types of activation the external interrupts :

1- level triggered

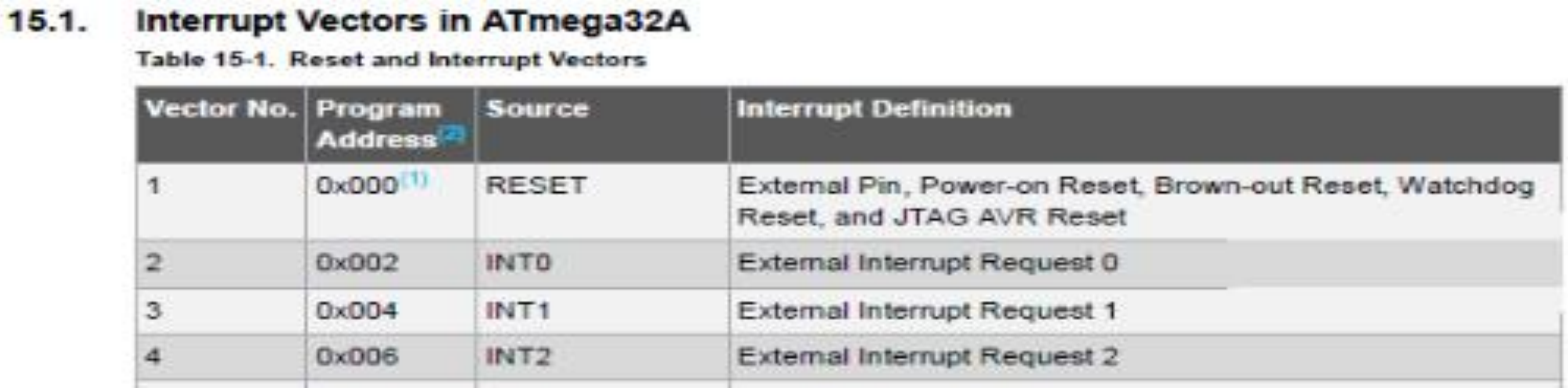
2- Edge triggered : INT2 is only edge triggered

Interrupt priority

- If two interrupts are activated in the same time , the interrupt with the higher priority is served first. We know which one is prior from the Interrupt vector table , the interrupt that has a lower address has a higher priority :

(address of INT0 is 2 - address of INT2 is 6)

so if the two external interrupts are occurred at the same time , INT0 is served first .



15.1. Interrupt Vectors in ATmega32A

Table 15-1. Reset and Interrupt Vectors

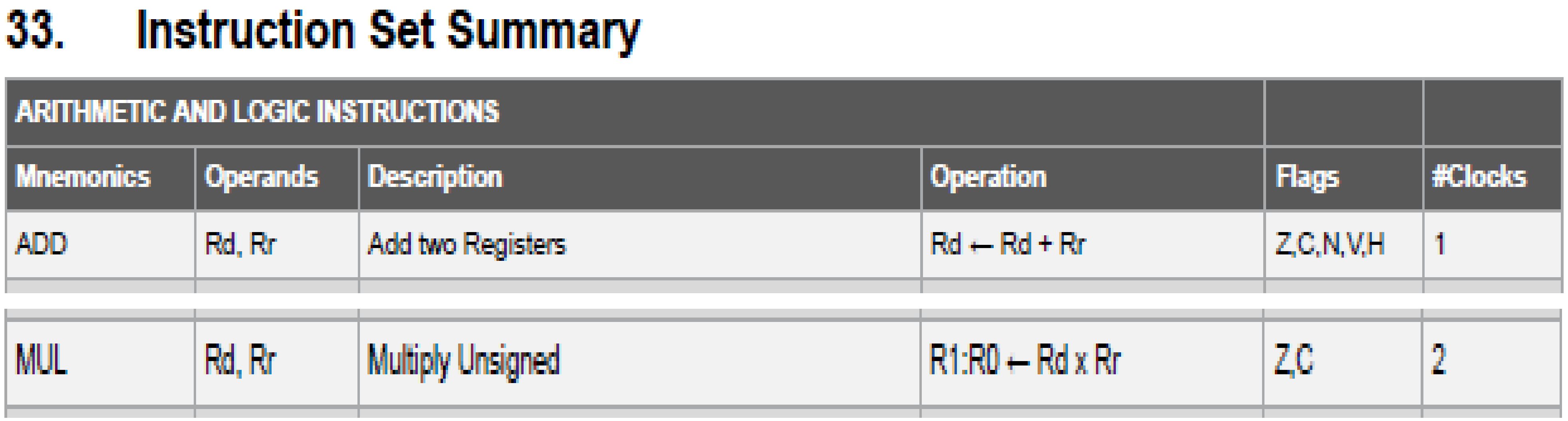
Vector No.	Program Address[2]	Source	Interrupt Definition
1	0x000[1]	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	0x002	INT0	External Interrupt Request 0
3	0x004	INT1	External Interrupt Request 1
4	0x006	INT2	External Interrupt Request 2

Interrupt latency

- The time from the moment an interrupt is activated to the moment the CPU starts to execute the ISR is called interrupt latency = 4 machine cycle times .

- During the time PC register is pushed on the stack and I-bit of SREG is cleared , causing all interrupts to be disabled , the duration of an interrupt latency can be affected by the type of instruction that the CPU executing when the interrupt occurs , CPU finishes the execution of the current instruction whatever its (execution time) : (MUL) instruction take two clock cycles

  but ADD and most of instructions take 2 , so it depend .

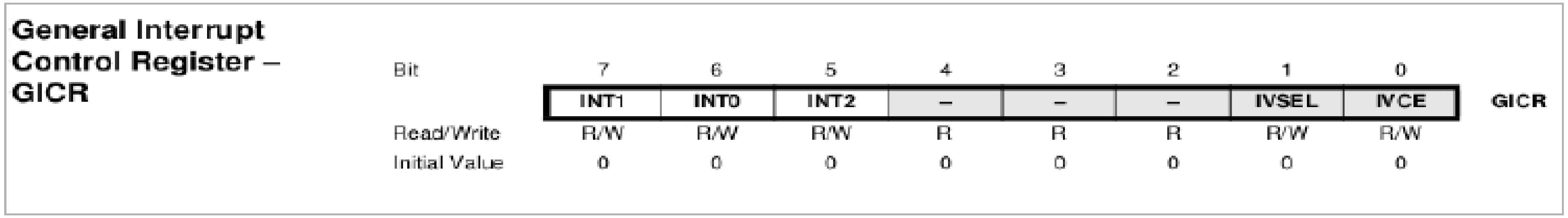


33. Instruction Set Summary

ARITHMETIC AND LOGIC INSTRUCTIONS					
Mnemonics	Operands	Description	Operation	Flags	#Clocks
ADD	Rd, Rr	Add two Registers	Rd ← Rd + Rr	Z,C,N,V,H	1
MUL	Rd, Rr	Multiply Unsigned	R1:R0 ← Rd x Rr	Z,C	2

The important used Registers

- GICR (General Interrupt Control Register) : this register has three important bits : INT0 – INT1 – INT2 , each one used to enable receiving interrupt on that pin . If we want to enable INT0 : GICR |= (1<<INT0)

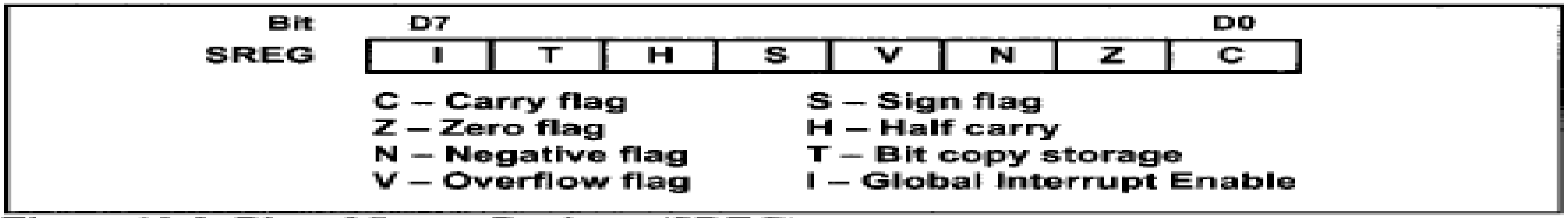


General Interrupt Control Register – GICR

Bit	7	6	5	4	3	2	1	0	
	INT1	INT0	INT2	–	–	–	IVSEL	IVCE	GICR
Read/Write	R/W	R/W	R/W	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- SREG : status register as we know has flag bits , as well as it has Global Interrupt Enable bit via assembly instruction SEI OR : SREG |= (1<<7);

or by just call the function sei(); which is in the interrupt library



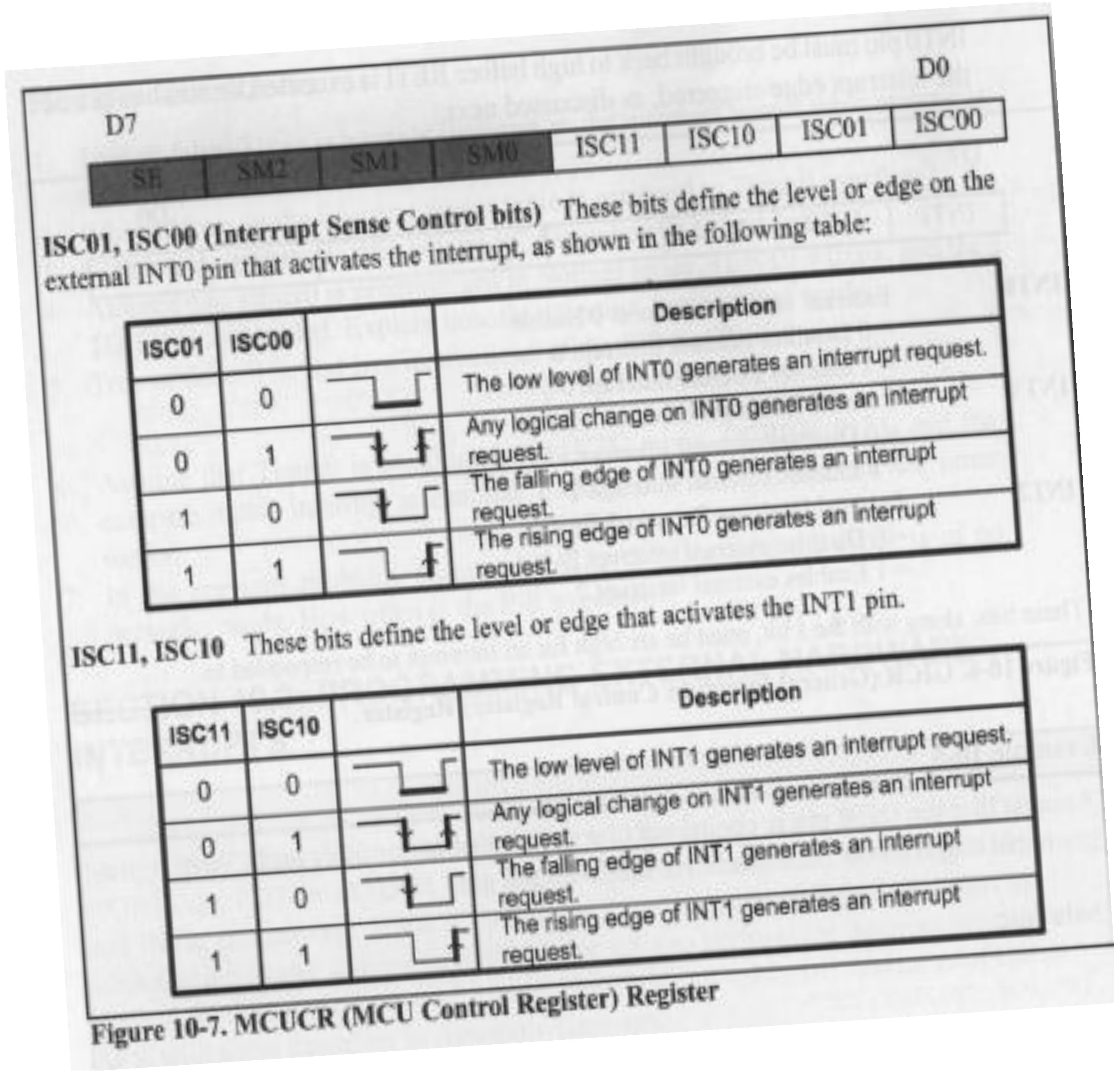
Bit	D7							D0
SREG	I	T	H	S	V	N	Z	C

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

Figure 10-2. Bits of Status Register (SREG)

- MCUCR (uC control register) :

- to specify the type of the signal to be interrupted via Interrupt Sense Control bits (ISC00 - ISC01) for INT0
(ISC10 - ISC11) for INT1
- So if we want LOW level generate INT0 :
MCUCR &= (~(1<<ISC00)) | (~(1<<ISC01)) ;
If you want to write zeros to these bits , you can do it , as the default value for each bit = 0 .
- If we want any logical change generate INT0 :
 MCUCR |= (1<<ISC00) ;
 MCUCR &= (~(1<<ISC01)) ;
- If we want falling edge generate INT0 :
 MCUCR |= (1<<ISC01) ;
 MCUCR &= (~(1<<ISC00)) ;
- If we want Rising edge generate INT0 :
 MCUCR |= (1<<ISC01) |(1<<ISC00);



D7							D0
SE	SM2	SM1	SM0	ISC11	ISC10	ISC01	ISC00

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.

Figure 10-7. MCUCR (MCU Control Register) Register

- MCUCSR (uC Control and Status Register) : which has ISC2 to define whether INT2 on the falling or rising edge.
- After Enabling I-bit in SREG & Enable external interrupt on INT2 via GICR :

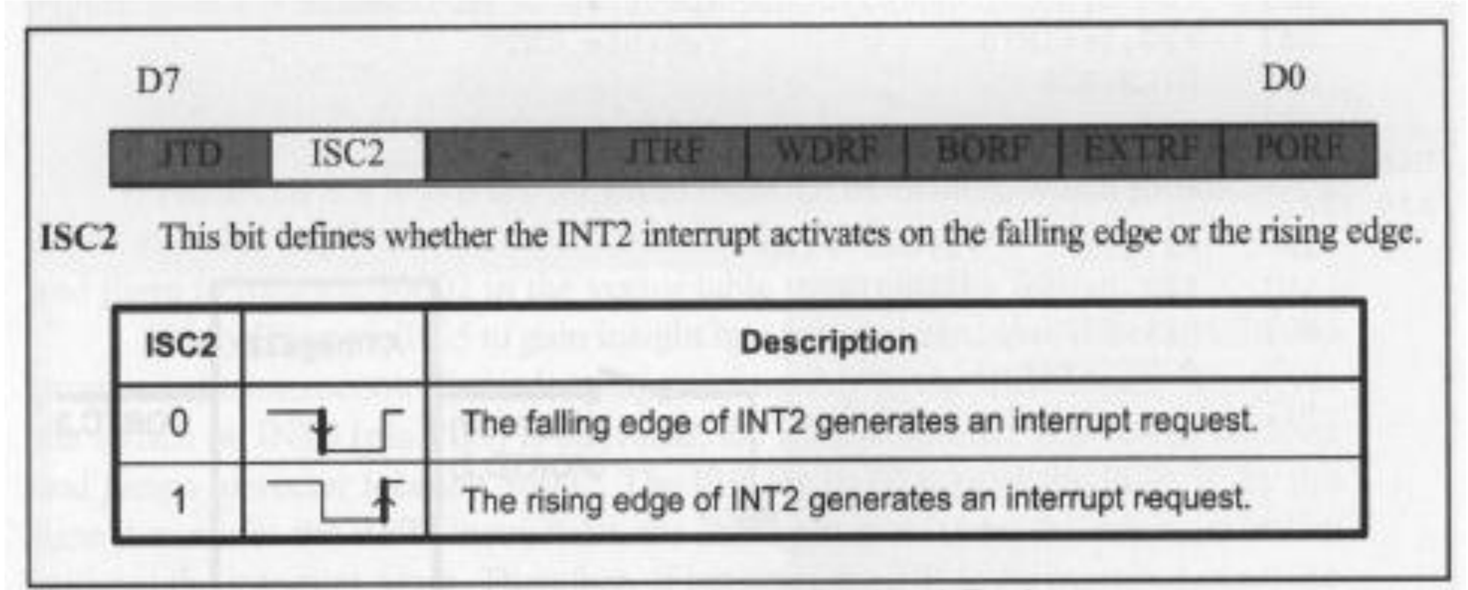
- If ISC2 bit is set to zero a falling edge on INT2 activates :

  MCUCSR &=(~(1<<ISC2));

- If ISC2 bit is set to one a rising edge on INT2 activates :

  MCUCSR |= (1<<ISC2);

- Because of INT2 is only edge-triggered it has only one Interrupt Sense Control bit (ISC2) setting it mean the rising edge of INT2 generates an Interrupt request , clearing it mean the falling edge of INT2 generates an Interrupt request .



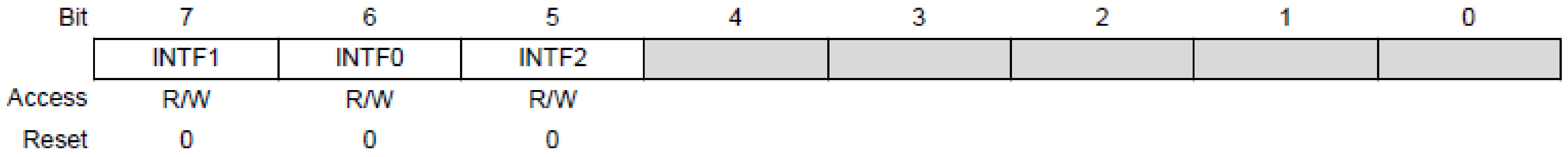
JTD	ISC2	-	JTRF	WDRF	BORF	EXTRF	PORF

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.

Figure 10-8. MCUCSR (MCU Control and Status Register) Register

- GIFR (General Interrupt Flag Register) : Has three flags for each interrupt do the same thing so for example we take INTF0 : when an event on the INT0 pin triggers an interrupt request, INTF0 becomes set (one). If the I-bit in SREG and the INT0 bit in GICR are set (one), the MCU will jump to the corresponding Interrupt Vector. the flag is cleared when the interrupt routine is executed.
- If you have nested interrupts : the flag can be cleared by writing a logical one to it. This flag is always cleared when INT0 is configured as a level interrupt.



Bit	7	6	5	4	3	2	1	0
	INTF1	INTF0	INTF2					
Access	R/W	R/W	R/W					
Reset	0	0	0					

GIFR General interrupt Flag Register

Putting all together

1- write a code that :

- Enable external INT0
- Choose Rising Mode
- When interrupt occurs change a led state which connected to PD1

2- write a code that :

- Enable external INT1 with Falling mode
- When interrupt occurs change a led state which connected to PD1

- You can put all these together by making a library for external interrupts

References :

- Books :
- Simply AVR - > Abdallah Ali
- The AVR microcontroller & Embedded Systems using Assembly & c -- > Mazidi
- ATMEGA 32A Datasheet
- PIC microcontroller -- > Milan Verle
- Websites :
- https://www.sparkfun.com
- http://maxembedded.com
- https://www.tutorialspoint.com/cprogramming
- https://stackoverflow.com
- https://www.quora.com
- https://www.lucidchart.com

Any questions ?

- Instructor : Mohammed Hemed

- Embedded Systems developer at fab lab Ismailia

Repository link of Embedded workshop Material:

https://github.com/FabLab-Ismailia/Embedded-Systems-Workshop

Contact me :

- Gmail :

mohammedhemed23@gmail.com

- Linkedin :

https://www.linkedin.com/in/mohammedhemed