EE-222: Microprocessor Systems

AVR Interrupts

Instructor: Dr. Arbab Latif

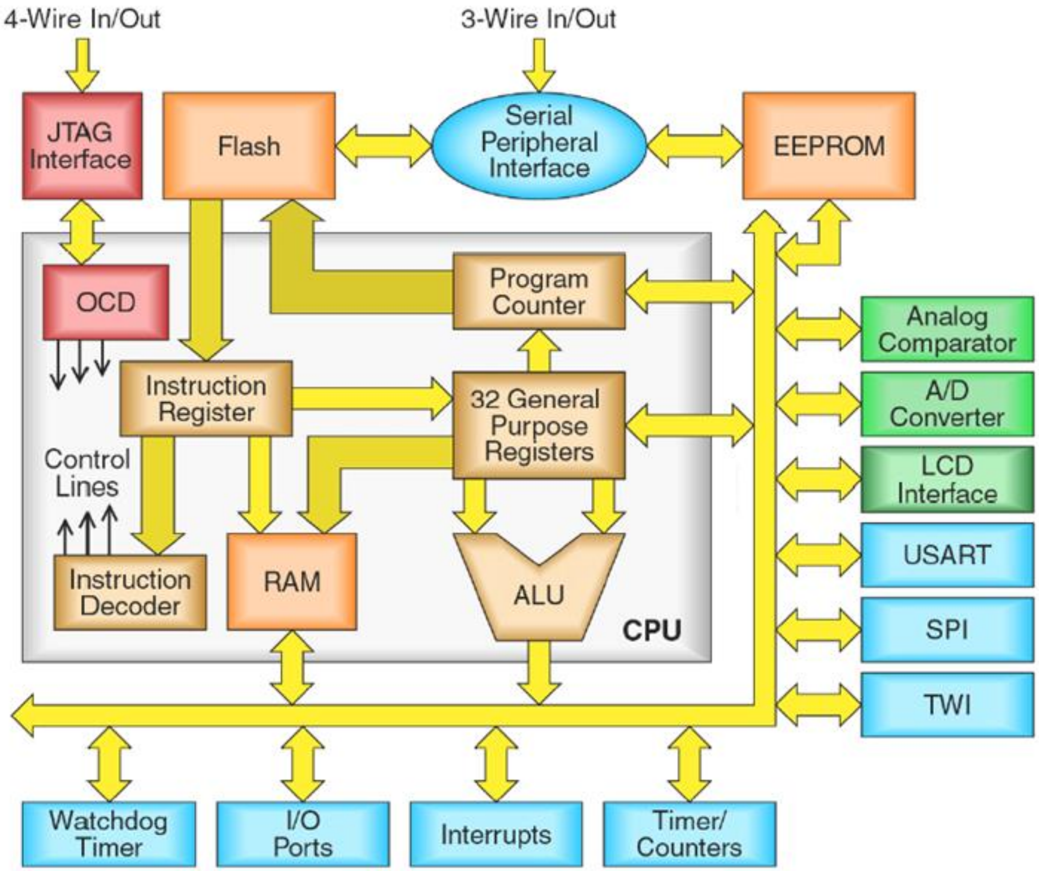
NUST

SCHOOL OF ELECTRICAL ENGINEERING &
COMPUTER SCIENCE (SEECS)

Interrupt vs Polling

I/O Services

- A single microcontroller can serve several devices.
- Two ways:
  - Polling method
  - Interrupt method



Polling Vs. Interrupt

- Polling
  - Ties down the CPU

```
while (true)
{
    if(PIND.2 == 0)
          //do something;
}
```

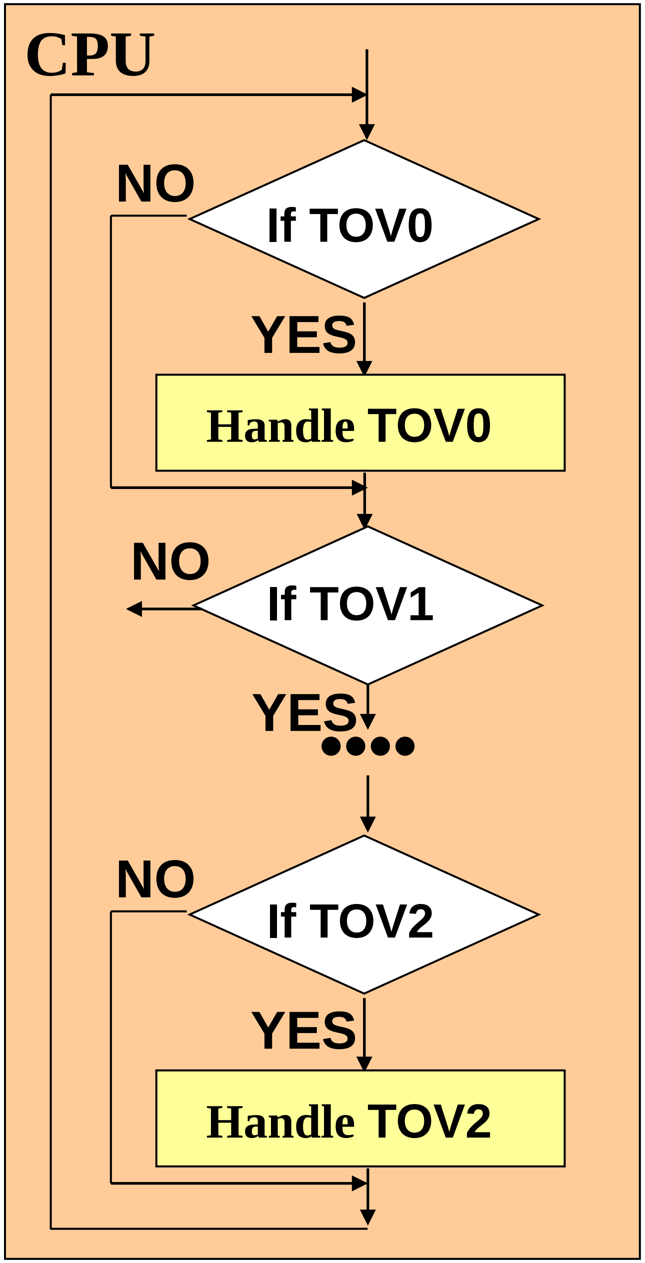
- Interrupt
  - Efficient CPU use
  - Has priority
  - Can be masked

```
main( )
{
    Do your common task
}

whenever PIND.2 is 0 then
    do something
```

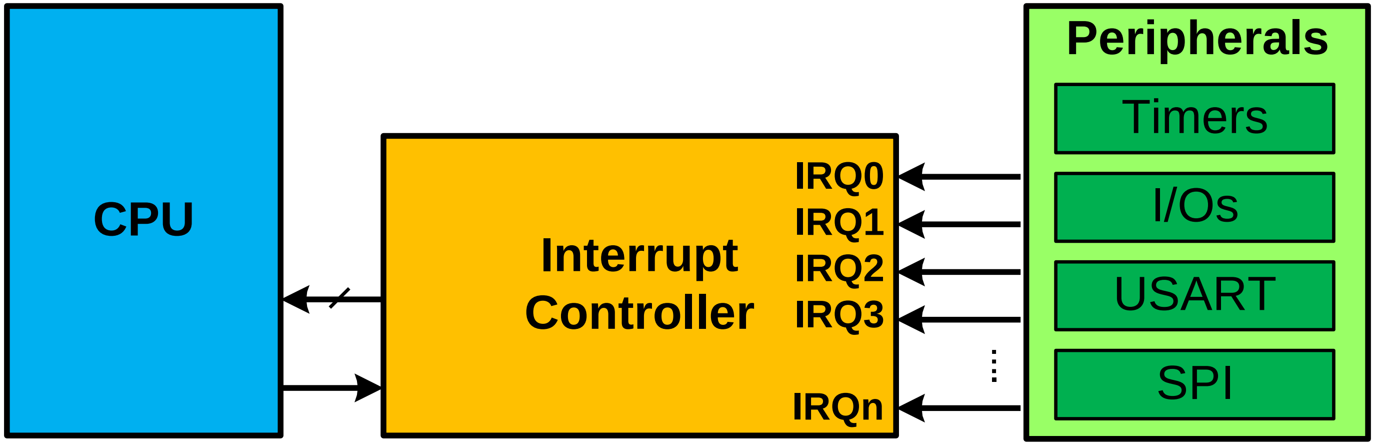
Polling Method

- What if you had to use all three timers simultaneously?
- The microcontroller continuously monitors the TOVx status of a given timer.
- When the condition is met, it performs the service.
- After that, it moves on to monitor the next device until every one is serviced.
- The microcontroller check all devices in a round-robin fashion.



Interrupt Method

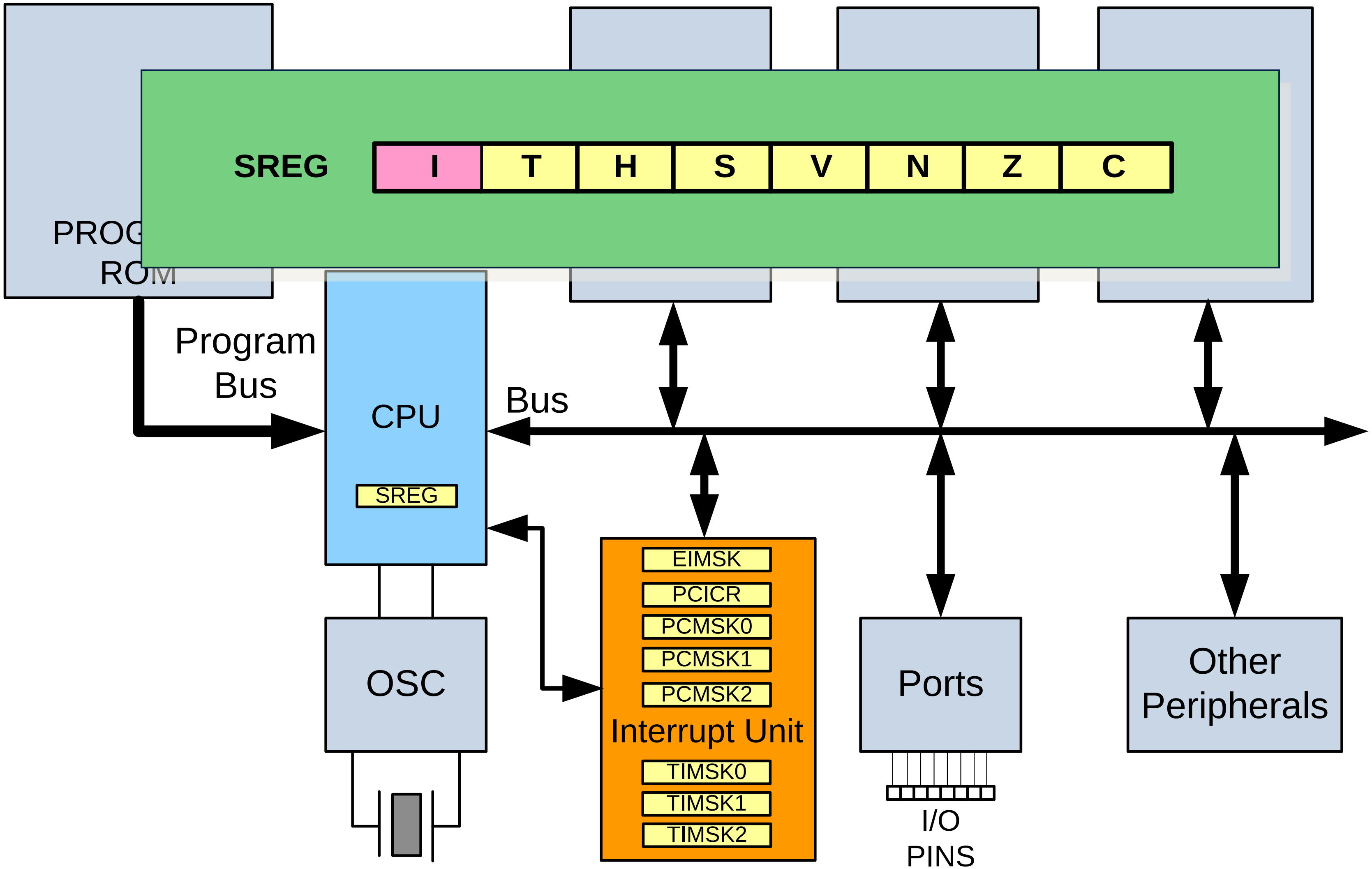
- An interrupt is an external or internal event that interrupts the microcontroller to inform it that a device needs its service.
- Whenever any device needs its service, the device notifies the microcontroller by sending it an interrupt signal.
- Upon receiving an interrupt signal, the microcontroller interrupts whatever it is doing and serves the device by executing the Interrupt Service Routine.



The Advantage of Interrupts

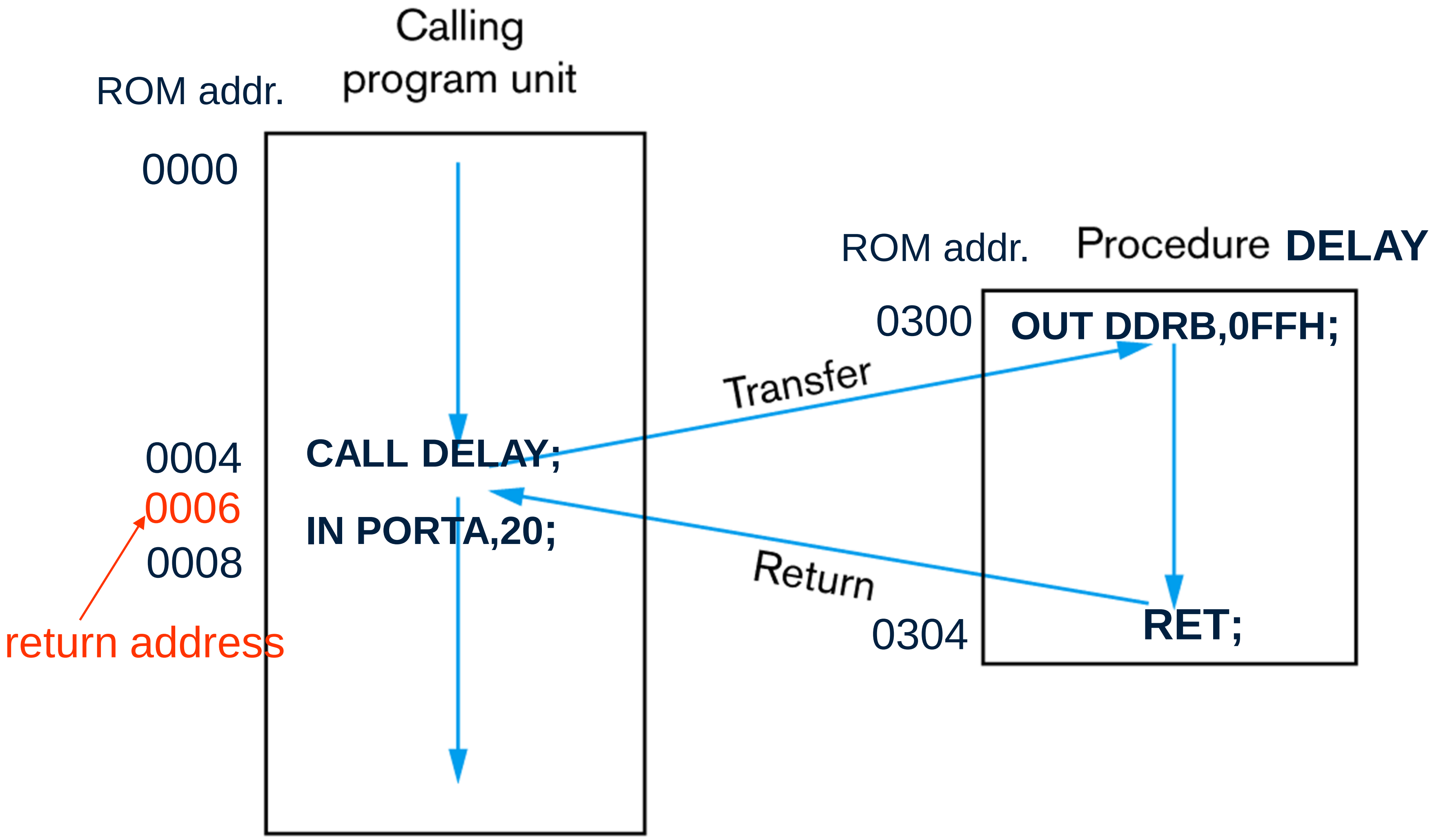
- The use of microcontroller is more efficient.
  - In polling system, `SBRS R20,TOV0` wastes much of the microcontroller’s time.
- The microcontroller can monitor many devices simultaneously.
- Each device can get service based on the priority assigned to it.
- The microcontroller can ignore (mask) a device request.

Interrupt control unit in AVR



Steps in Executing an Interrupt

Review of Call A Subroutine



Steps in Executing an AVR Interrupt (1/2)

- Upon activation of an interrupt, the microcontroller goes through the following steps:
  1. It finishes the instruction it is executing and saves the address of the next instruction (PC) on the stack.
  2. It jumps to a fixed location in memory based on the interrupt vector table. [see couple of slides down]
  3. The microcontroller gets the address of the ISR from the interrupt vector table and jumps to it.
  4. The microcontroller starts to execute the interrupt service routine until it reaches the last instruction of the subroutine which is RETI (return from interrupt).

Steps in Executing an AVR Interrupt (2/2)

- Executing steps (continuous):

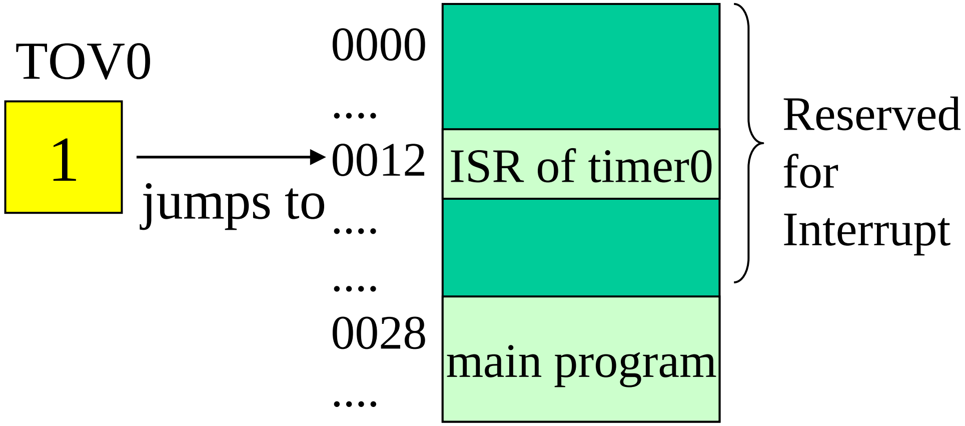
  5. Upon executing the RETI instruction, the microcontroller returns to the place where it was interrupted.

     - First, it gets the program counter (PC) address from the stack by popping the top two bytes of the stack into the PC.
     - Then it starts to execute from that address.

Interrupt Service Routine (ISR)

Interrupt Service Routine (ISR)

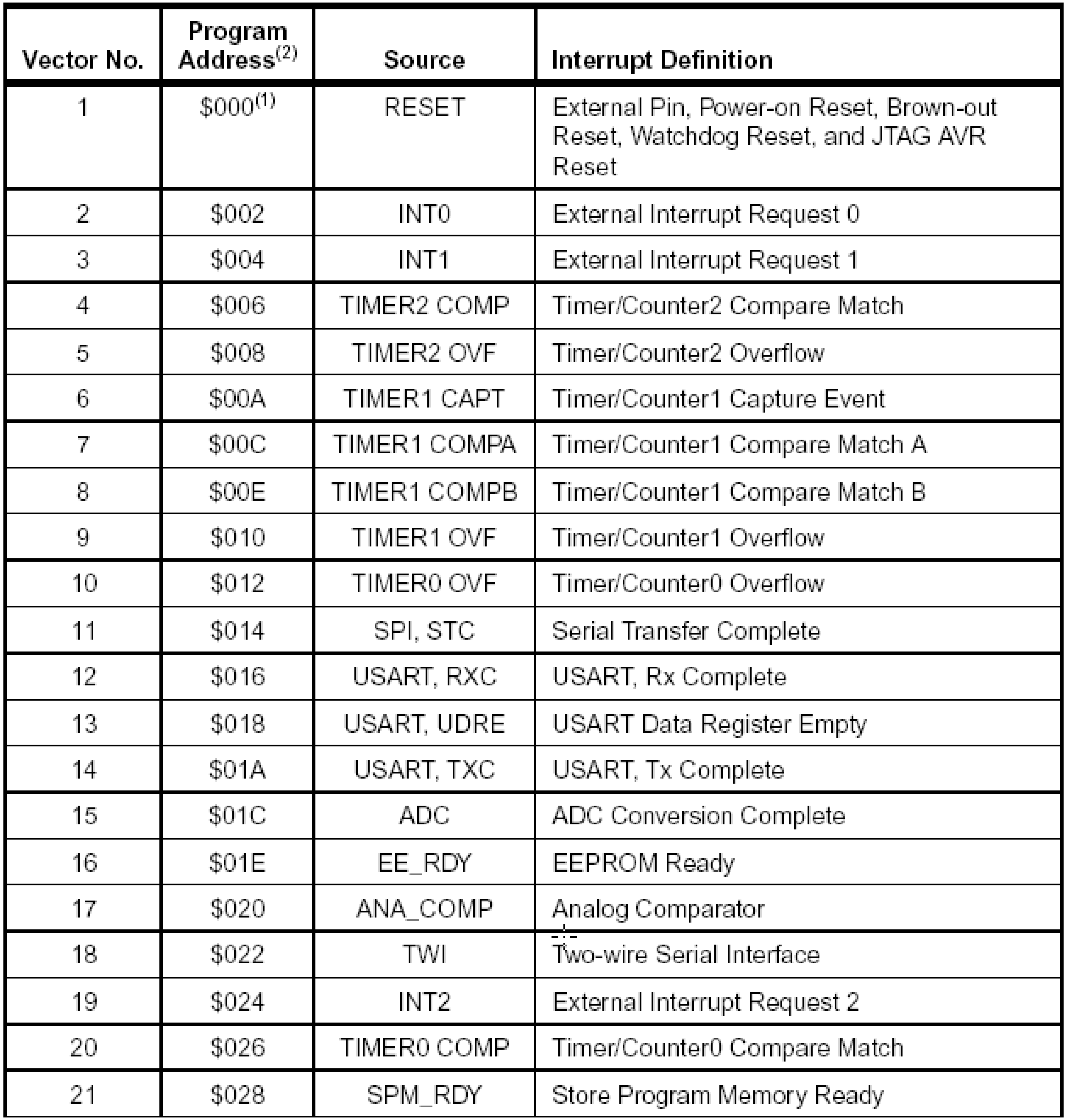
- ISRs are similar to normal subroutines.
- ISRs are generated by programs to handle interrupt events.
- For every interrupt, there must be an ISR.
- For every interrupt event, its corresponding ISR is held at a fixed location in memory.



The Addresses of ISRs

- The group of memory locations set aside to hold the addresses of ISRs is called the interrupt vector table.

- See AVR Interrupt Vector Table in next slide.



Vector No.	Program Address[2]	Source	Interrupt Definition
1	\$000[1]	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	\$002	INT0	External Interrupt Request 0
3	\$004	INT1	External Interrupt Request 1
4	\$006	TIMER2 COMP	Timer/Counter2 Compare Match
5	\$008	TIMER2 OVF	Timer/Counter2 Overflow
6	\$00A	TIMER1 CAPT	Timer/Counter1 Capture Event
7	\$00C	TIMER1 COMPA	Timer/Counter1 Compare Match A
8	\$00E	TIMER1 COMPB	Timer/Counter1 Compare Match B
9	\$010	TIMER1 OVF	Timer/Counter1 Overflow
10	\$012	TIMER0 OVF	Timer/Counter0 Overflow
11	\$014	SPI, STC	Serial Transfer Complete
12	\$016	USART, RXC	USART, Rx Complete
13	\$018	USART, UDRE	USART Data Register Empty
14	\$01A	USART, TXC	USART, Tx Complete
15	\$01C	ADC	ADC Conversion Complete
16	\$01E	EE_RDY	EEPROM Ready
17	\$020	ANA_COMP	Analog Comparator
18	\$022	TWI	Two-wire Serial Interface
19	\$024	INT2	External Interrupt Request 2
20	\$026	TIMER0 COMP	Timer/Counter0 Compare Match
21	\$028	SPM_RDY	Store Program Memory Ready

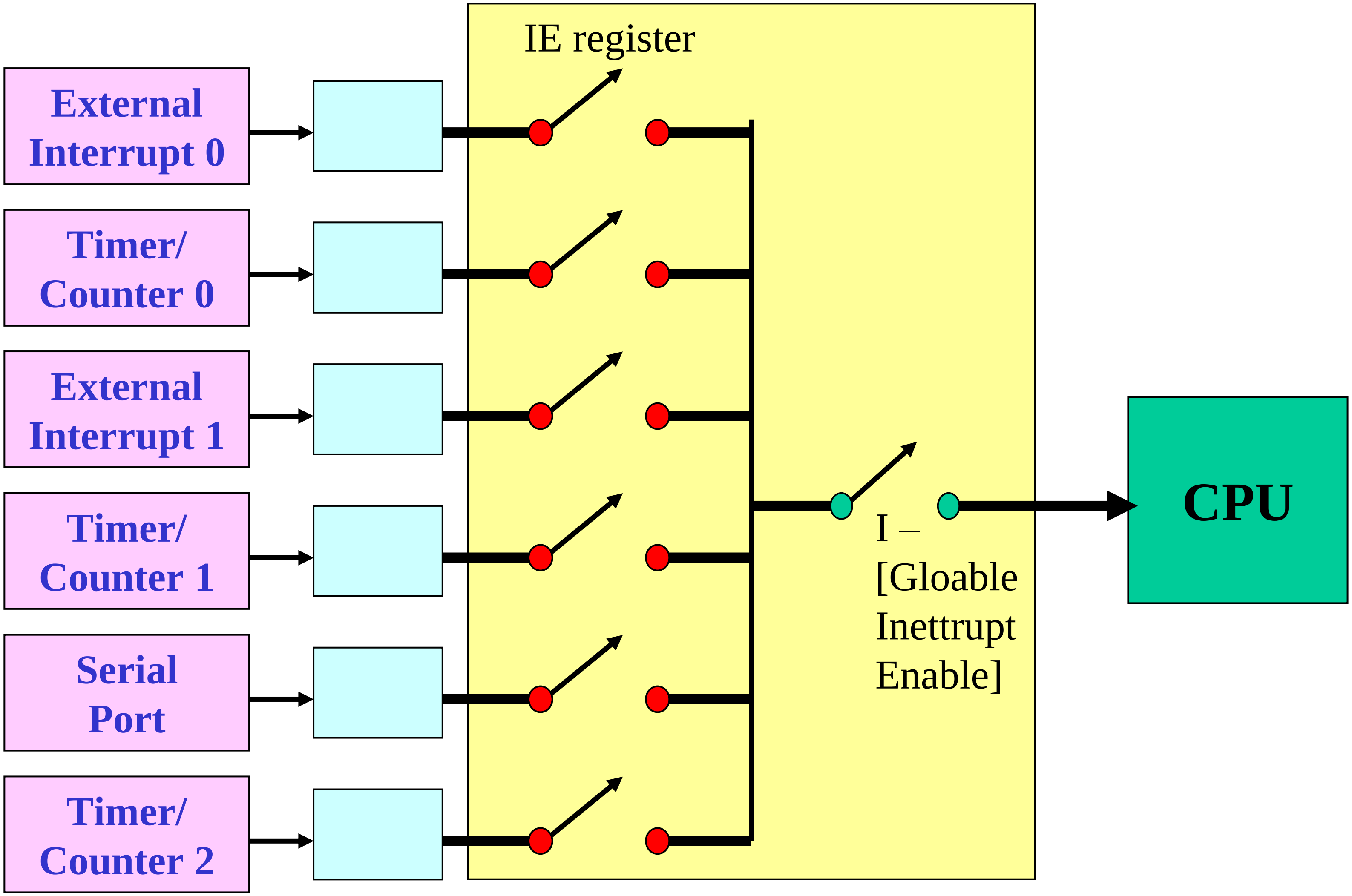
Interrupt Vector Table

- Normally, the ISR for an interrupt is too long to fit into the memory space allocated inside the interrupt vector table.

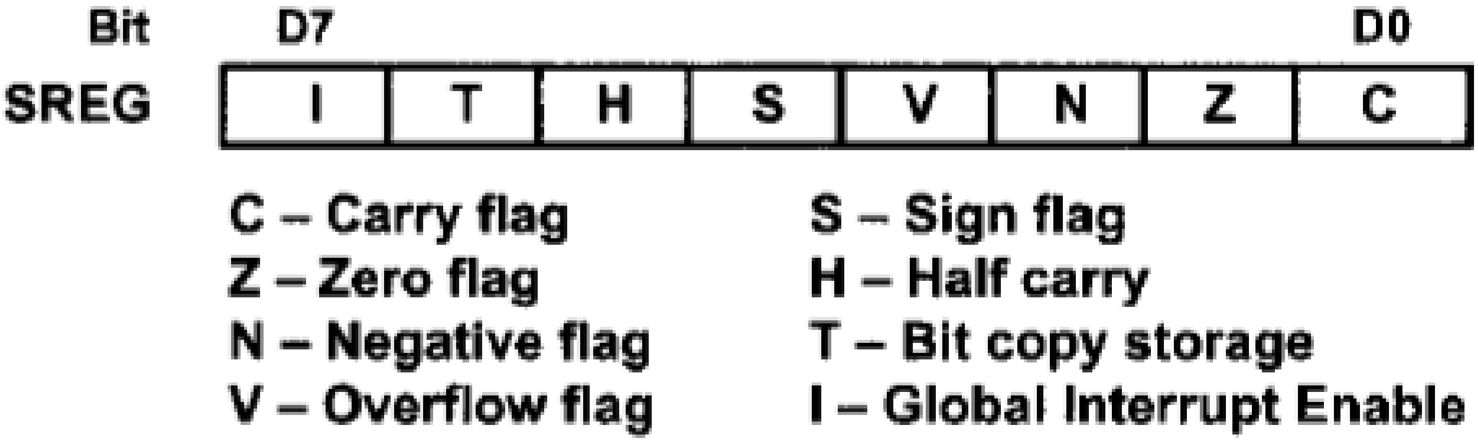
- For that reason, a JMP instruction is placed in the vector table to point to the address of the ISR.

How to Enable and Disable an Interrupt?

Enable/Disable Interrupts



Global Interrupt Enable



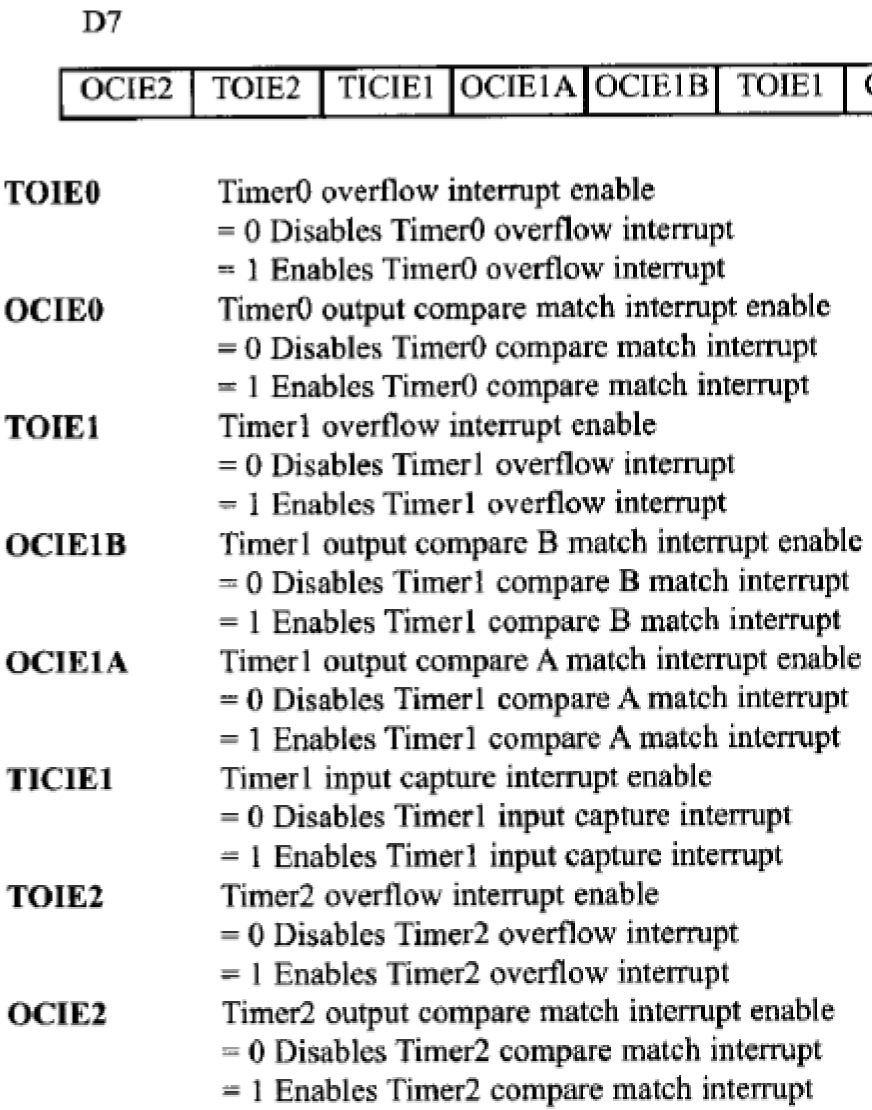
Bit	D7							D0
SREG	I	T	H	S	V	N	Z	C

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

```
SEI ; set I (enable interrupts globally)
```

Timer Interrupt Mask Register (TIMSK)



D7							D0
OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0

TOIE0 Timer0 overflow interrupt enable
= 0 Disables Timer0 overflow interrupt
= 1 Enables Timer0 overflow interrupt

OCIE0 Timer0 output compare match interrupt enable
= 0 Disables Timer0 compare match interrupt
= 1 Enables Timer0 compare match interrupt

TOIE1 Timer1 overflow interrupt enable
= 0 Disables Timer1 overflow interrupt
= 1 Enables Timer1 overflow interrupt

OCIE1B Timer1 output compare B match interrupt enable
= 0 Disables Timer1 compare B match interrupt
= 1 Enables Timer1 compare B match interrupt

OCIE1A Timer1 output compare A match interrupt enable
= 0 Disables Timer1 compare A match interrupt
= 1 Enables Timer1 compare A match interrupt

TICIE1 Timer1 input capture interrupt enable
= 0 Disables Timer1 input capture interrupt
= 1 Enables Timer1 input capture interrupt

TOIE2 Timer2 overflow interrupt enable
= 0 Disables Timer2 overflow interrupt
= 1 Enables Timer2 overflow interrupt

OCIE2 Timer2 output compare match interrupt enable
= 0 Disables Timer2 compare match interrupt
= 1 Enables Timer2 compare match interrupt

```
LDI R16, 1<<TOV0
OUT TIMSK, R16;
enable Timer0 overflow interrupts
```

Notices

- Interrupt is disable upon RESET.

- You can open the functionality of interrupt or not.

- You can choose to disable some interrupt events,
  - You do not need to write ISRs for them.

- Programmers must enable these interrupts before using them.

Recommended Reading

- The AVR Microcontroller and Embedded Systems: Using Assembly and C by Mazidi et al., Prentice Hall
  - Chapter 10

THANK YOU