Lesson Interrupt

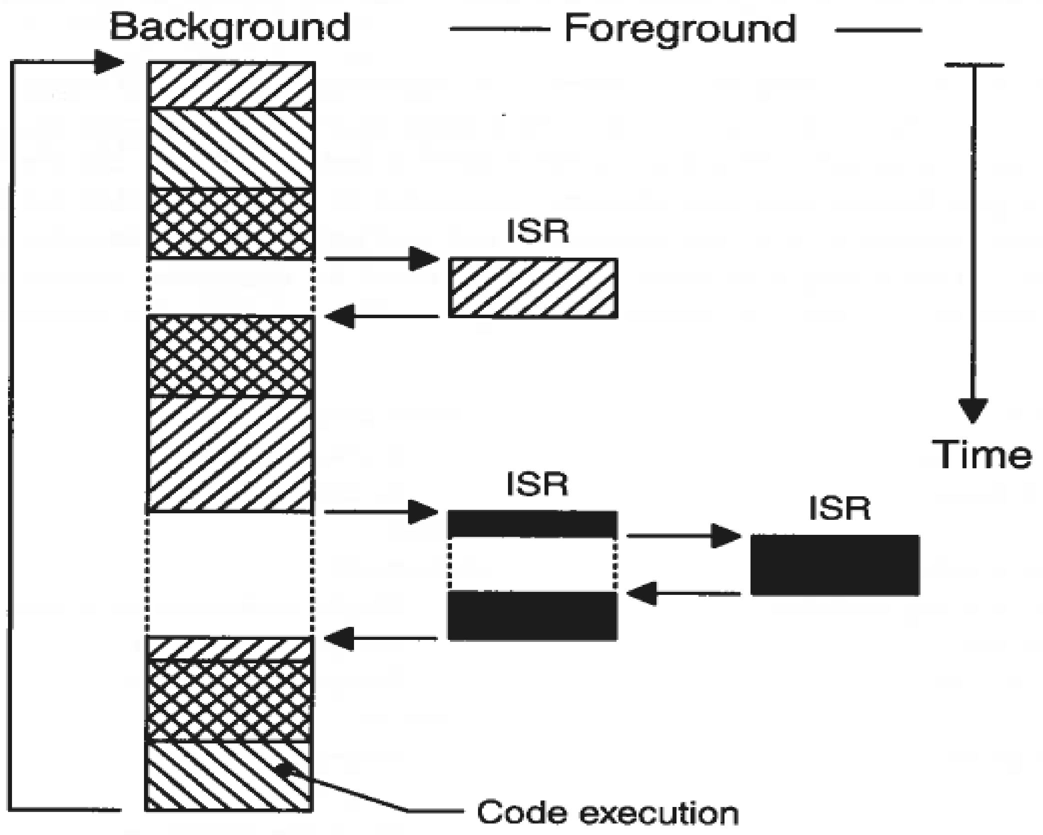
Communicating with I/O Devices

- The OS/App needs to know when:
  - The I/O device has completed an operation
  - The I/O operation has encountered an error
- This can be accomplished in two different ways:
  - Polling
  - Interrupt

Communicating with I/O Devices.....

- Polling:
  - The I/O device put information in a status register
  - The OS/App periodically check the status register
- I/O Interrupt:
  - An I/O interrupt is an externally stimulated event, asynchronous to instruction execution but does allow instruction completion
  - Whenever an I/O device needs attention from the processor, it interrupts the processor from what it is currently doing

Foreground/Background



Interrupt

- Enabling/Disabling
- Protection of critical sections against interrupt
- Reentrance software (Thread safe)
- Priority
- Hardware interrupt
- Software interrupt
- Interrupt Service routines (ISR)

Interrupt routines

- Must save processor state and registers
- Clear the interrupt reason
- Handle the interrupt – as fast as possible
- Restore the processor state and registers
- Return to normal execution

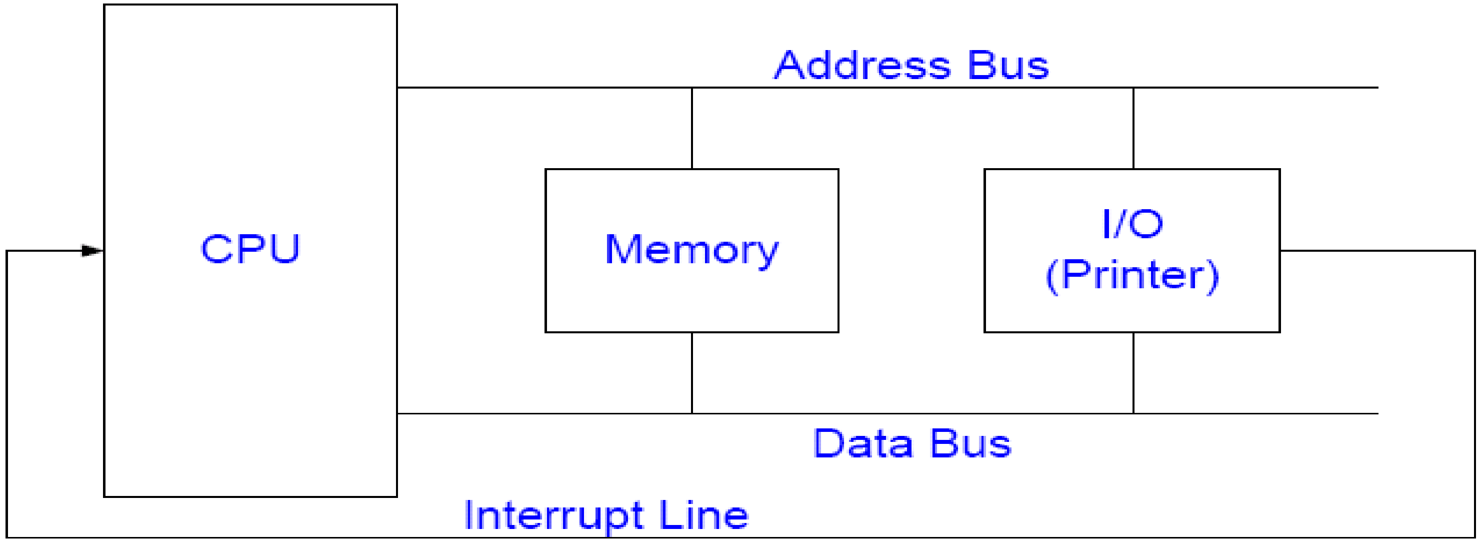
Types of interrupts

- Hardware vs Software
  - Hardware: I/O, clock tick, power failure, exceptions
  - Software: INT instruction (80386)
- External vs Internal Hardware Interrupts
  - External interrupts are generated by CPU's interrupt pins
  - Internal interrupts (exceptions): div by zero, single step, page fault,
  - bad op-code, stack overflow, protection, ...

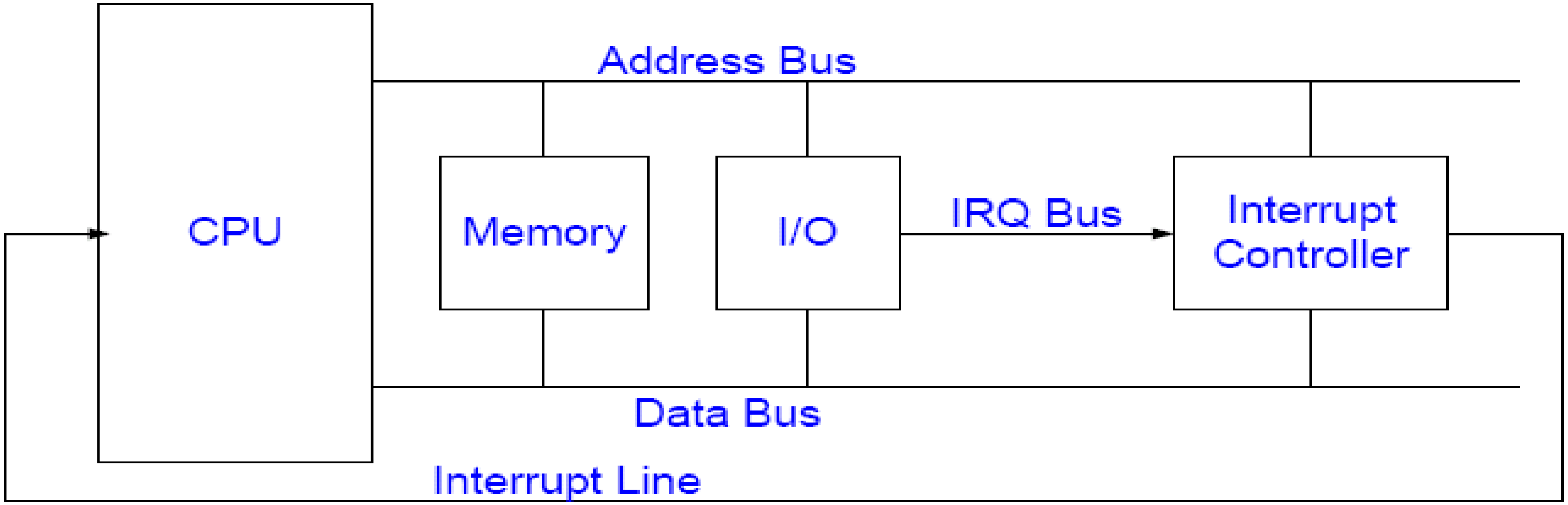
Types of interrupts ....

- Synchronous vs Asynchronous Hardware Int.
  - Synchronous interrupts occur at exactly the same place every time the program is executed. E.g., bad opcode, div by zero, illegal memory address, software int.
  - Asynchronous interrupts occur at unpredictable times relative to the program
    E.g., I/O, clock ticks

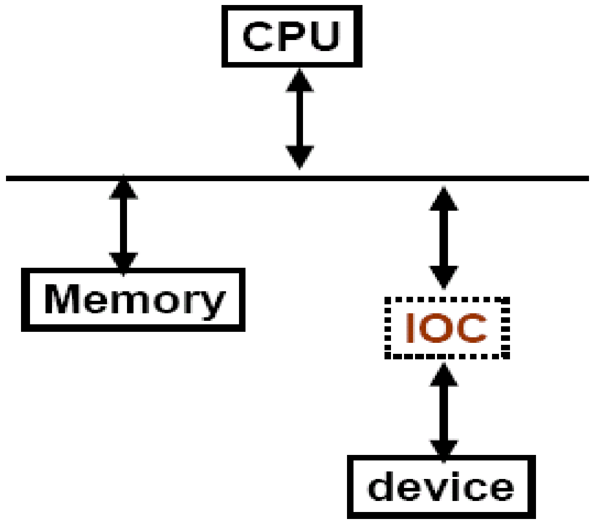
Hardware

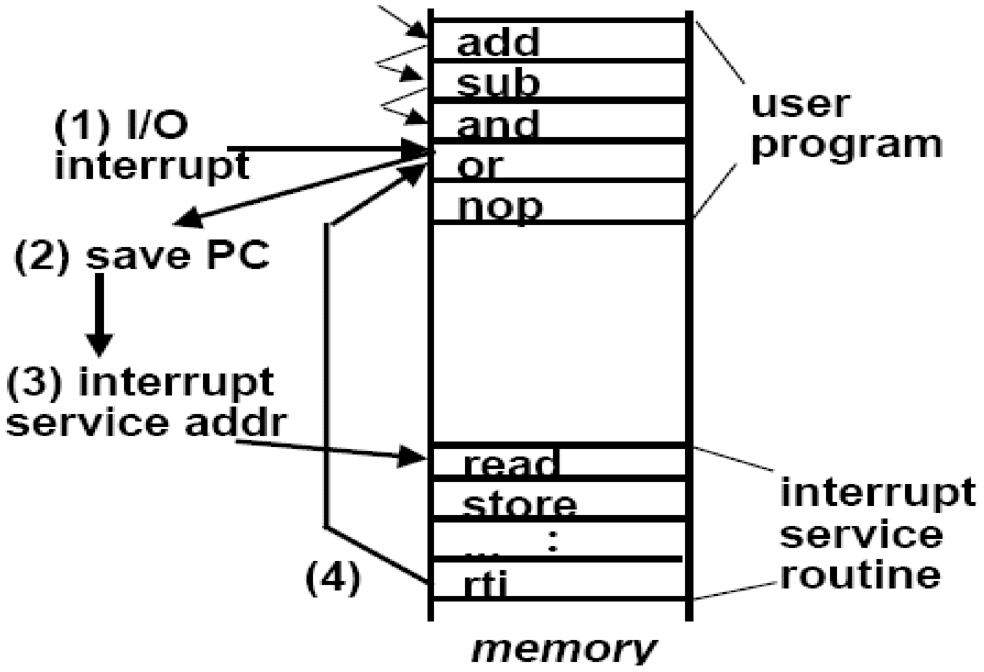


Hardware…..


Address Bus
CPU
Memory
I/O
IRQ Bus
Interrupt Controller
Data Bus
Interrupt Line

Software


CPU
Memory
IOC
device


add
sub
and
or
nop
user program
(1) I/O interrupt
(2) save PC
(3) interrupt service addr
read
store
...
:
rti
(4)
interrupt service routine
memory

Interrupt Sequence

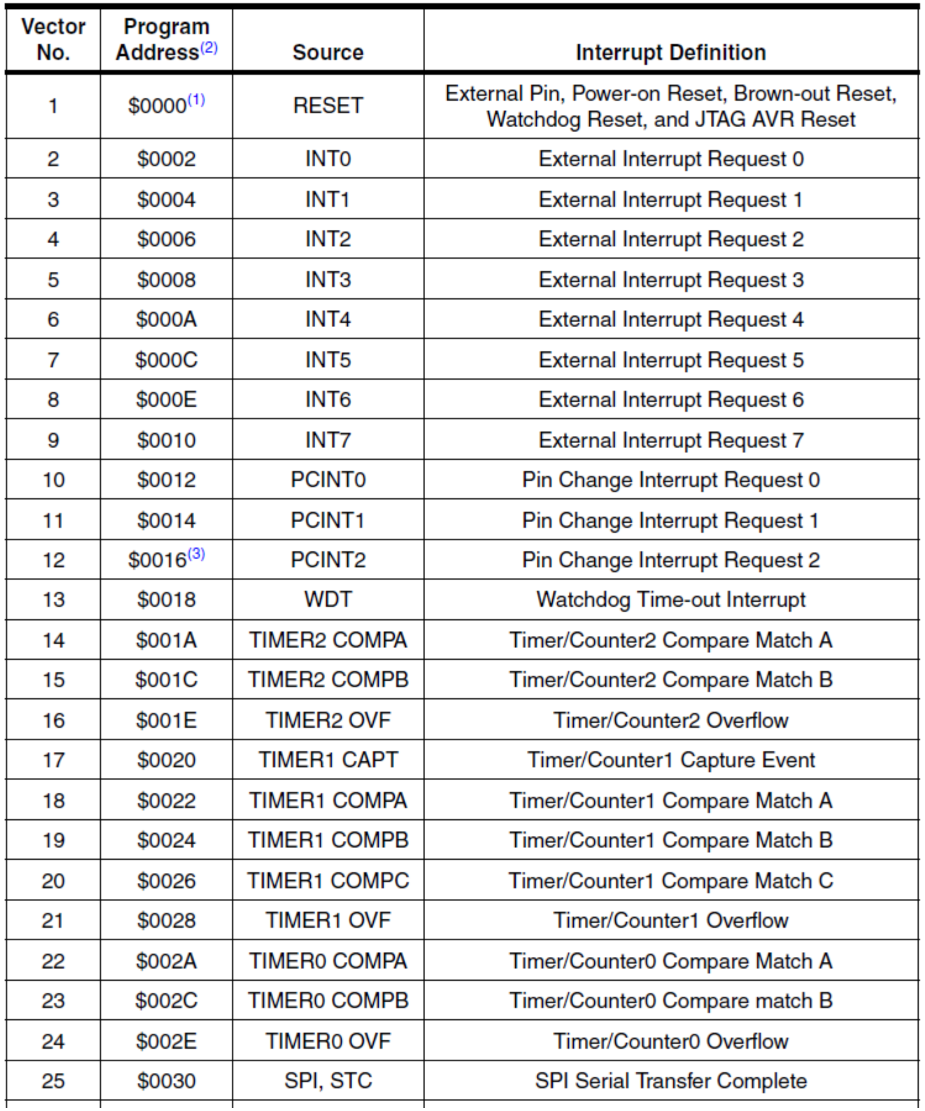
- Device sends signal to interrupt controller
- Controller uses IRQ# for interrupt # and priority
- Controller sends signal to CPU if the CPU is not already processing an interrupt with higher priority
- CPU finishes executing the current instruction
- CPU saves FLAGS & return address on the stack
- CPU gets interrupt # from controller using I/O ops
- CPU finds “gate” in Interrupt Description Table
- CPU switches to Interrupt Service Routine (ISR). This may include a change in privilege level
- IF cleared

Interrupt Sequence ....

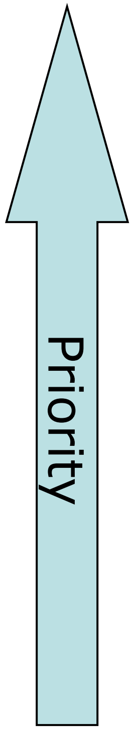
- ISR saves registers if necessary
- ISR, after initial processing, sets IF to allow interrupts
- ISR processes the interrupt
- ISR restores registers if necessary
- ISR sends End of Interrupt (EOI) to controller
- ISR returns from interrupt using IRET. EFLAGS (including IF) & return address restored
- CPU executes the next instruction
- Interrupt controller waits for next interrupt and manages pending interrupts

Interrupt on AVR

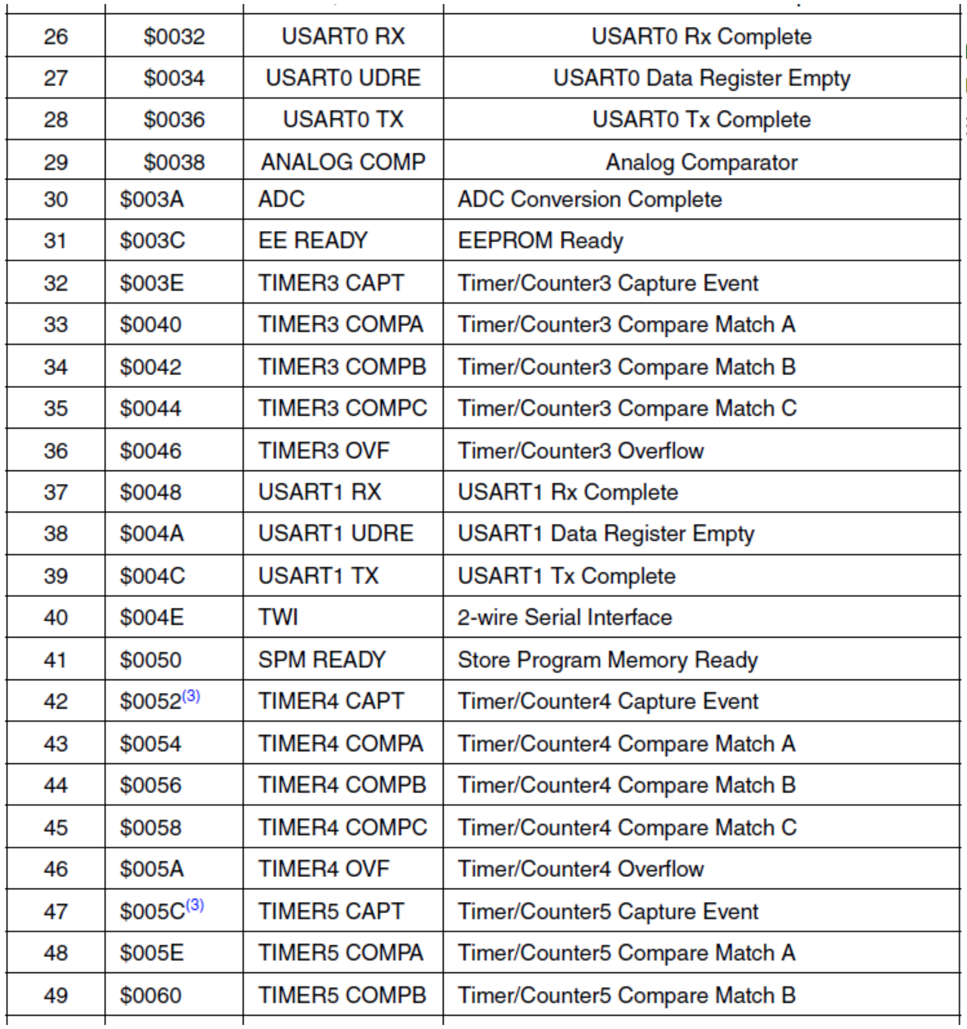
- Number and type of interrupt depends on MCU type
- ATMEGA1280 has 57 Vectors



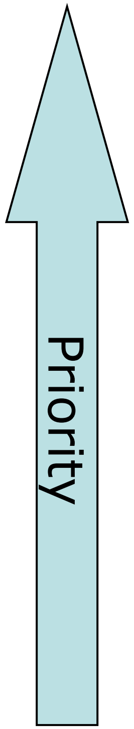
Vector No.	Program Address[2]	Source	Interrupt Definition
1	$0000[1]	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	$0002	INT0	External Interrupt Request 0
3	$0004	INT1	External Interrupt Request 1
4	$0006	INT2	External Interrupt Request 2
5	$0008	INT3	External Interrupt Request 3
6	$000A	INT4	External Interrupt Request 4
7	$000C	INT5	External Interrupt Request 5
8	$000E	INT6	External Interrupt Request 6
9	$0010	INT7	External Interrupt Request 7
10	$0012	PCINT0	Pin Change Interrupt Request 0
11	$0014	PCINT1	Pin Change Interrupt Request 1
12	$0016[3]	PCINT2	Pin Change Interrupt Request 2
13	$0018	WDT	Watchdog Time-out Interrupt
14	$001A	TIMER2 COMPA	Timer/Counter2 Compare Match A
15	$001C	TIMER2 COMPB	Timer/Counter2 Compare Match B
16	$001E	TIMER2 OVF	Timer/Counter2 Overflow
17	$0020	TIMER1 CAPT	Timer/Counter1 Capture Event
18	$0022	TIMER1 COMPA	Timer/Counter1 Compare Match A
19	$0024	TIMER1 COMPB	Timer/Counter1 Compare Match B
20	$0026	TIMER1 COMPC	Timer/Counter1 Compare Match C
21	$0028	TIMER1 OVF	Timer/Counter1 Overflow
22	$002A	TIMER0 COMPA	Timer/Counter0 Compare Match A
23	$002C	TIMER0 COMPB	Timer/Counter0 Compare match B
24	$002E	TIMER0 OVF	Timer/Counter0 Overflow
25	$0030	SPI, STC	SPI Serial Transfer Complete



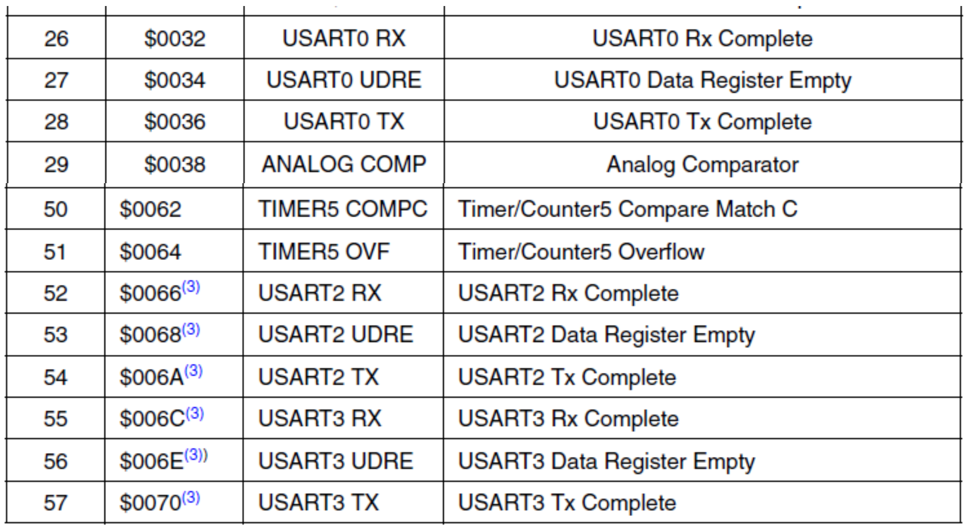
Interrupt on AVR



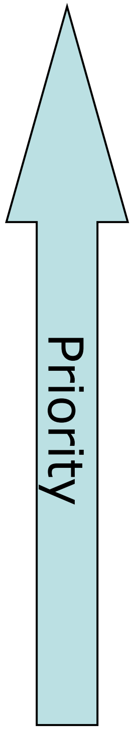
26	$0032	USART0 RX	USART0 Rx Complete
27	$0034	USART0 UDRE	USART0 Data Register Empty
28	$0036	USART0 TX	USART0 Tx Complete
29	$0038	ANALOG COMP	Analog Comparator
30	$003A	ADC	ADC Conversion Complete
31	$003C	EE READY	EEPROM Ready
32	$003E	TIMER3 CAPT	Timer/Counter3 Capture Event
33	$0040	TIMER3 COMPA	Timer/Counter3 Compare Match A
34	$0042	TIMER3 COMPB	Timer/Counter3 Compare Match B
35	$0044	TIMER3 COMPC	Timer/Counter3 Compare Match C
36	$0046	TIMER3 OVF	Timer/Counter3 Overflow
37	$0048	USART1 RX	USART1 Rx Complete
38	$004A	USART1 UDRE	USART1 Data Register Empty
39	$004C	USART1 TX	USART1 Tx Complete
40	$004E	TWI	2-wire Serial Interface
41	$0050	SPM READY	Store Program Memory Ready
42	$0052[(3)]	TIMER4 CAPT	Timer/Counter4 Capture Event
43	$0054	TIMER4 COMPA	Timer/Counter4 Compare Match A
44	$0056	TIMER4 COMPB	Timer/Counter4 Compare Match B
45	$0058	TIMER4 COMPC	Timer/Counter4 Compare Match C
46	$005A	TIMER4 OVF	Timer/Counter4 Overflow
47	$005C[(3)]	TIMER5 CAPT	Timer/Counter5 Capture Event
48	$005E	TIMER5 COMPA	Timer/Counter5 Compare Match A
49	$0060	TIMER5 COMPB	Timer/Counter5 Compare Match B



Interrupt on AVR



26	$0032	USART0 RX	USART0 Rx Complete
27	$0034	USART0 UDRE	USART0 Data Register Empty
28	$0036	USART0 TX	USART0 Tx Complete
29	$0038	ANALOG COMP	Analog Comparator
50	$0062	TIMER5 COMPC	Timer/Counter5 Compare Match C
51	$0064	TIMER5 OVF	Timer/Counter5 Overflow
52	$0066[(3)]	USART2 RX	USART2 Rx Complete
53	$0068[(3)]	USART2 UDRE	USART2 Data Register Empty
54	$006A[(3)]	USART2 TX	USART2 Tx Complete
55	$006C[(3)]	USART3 RX	USART3 Rx Complete
56	$006E[(3))]	USART3 UDRE	USART3 Data Register Empty
57	$0070[(3)]	USART3 TX	USART3 Tx Complete



AVR Software for interrupt handling

- Implement a ISR
- Setup and enable the interrupt source
- Enable the MCU's general interrupt

Case:

Setup external interrupt 1 (INT4/PE4) to interrupt on every level change

The ISR for INT1 handling

```
/**
 * Interrupt Service Routine (ISR) for
handling
 * INT4 interrupts.
 * Toggles PH0 for every interrupt
 */
ISR(INT4_vect) {
   // Toggle PH0
   PORTH = PORTH ^ _BV(PH0);
}
```

Setup and enable interrupt source

```
/**
 * Initialize the asyncron port pin toggler.
 * Setup INT4 (PE4) to interrupt on all level changes.
 * Setup PE4 to input.
 * Setup PH0 to output
  */
void init_port_toggler( void ) {
   // Set PE4 (INT4) to input
   DDRE &= ~_BV(DDE4);

   // Set PH0 to output
   DDRH |= _BV(DDH0);

   // Set INT4 to interrupt on every level change
   EICRB |= _BV(ISC40);

   // Enable INT4 in the Externalinterrupt register
   EIMSK |= _BV(INT4);
}
```

Enable the MCU’s general interrupt

```
/**
 * Main function
 * \return always 0
 */
int main( void ) {
   init_port_toggler();

   // Enable MCU interrupt (set I-flag)
   sei();


   // The main loop
   while (1) {
   }

   return 0;
}
```

How to protect your code against interrupts?

- Sometimes you can/will not allow your program to be interrupted
- We need a way to disable interrupts, but still be sure that all interrupts will be remembered and executed
- In the example we used `sei()` (set enable interrupt flag) to enable the MCU's general interrupt
- `cli()` (clear enable interrupt flag) can be used to disable all interrupts with a single instruction
- `sei()` and `cli()` manipulates with the global interrupt bit (I) in the MCU's status register (SREG)

How to protect your code against interrupts?

- The safe way to disable and enable interrupts
  - Disable interrupt

```
// disable interrupt
 uint8_t cSREG = SREG;
 cli();
```

  - First store the whole status register
  - Then disable the interrupt by clearing the interrupt bit

How to protect your code against interrupts?

- The safe way to disable and enable interrupts
  - Enable interrupt

```
// restore interrupt status
SREG = cSREG;
```

  - Restore the whole status register

- Using this method you will never enable interrupt by accident