



# 1DT301, Computer Technology I

Lecture #5,

Tuesday, September 17, 2019

- Interrupts
- Introduction to lab 3



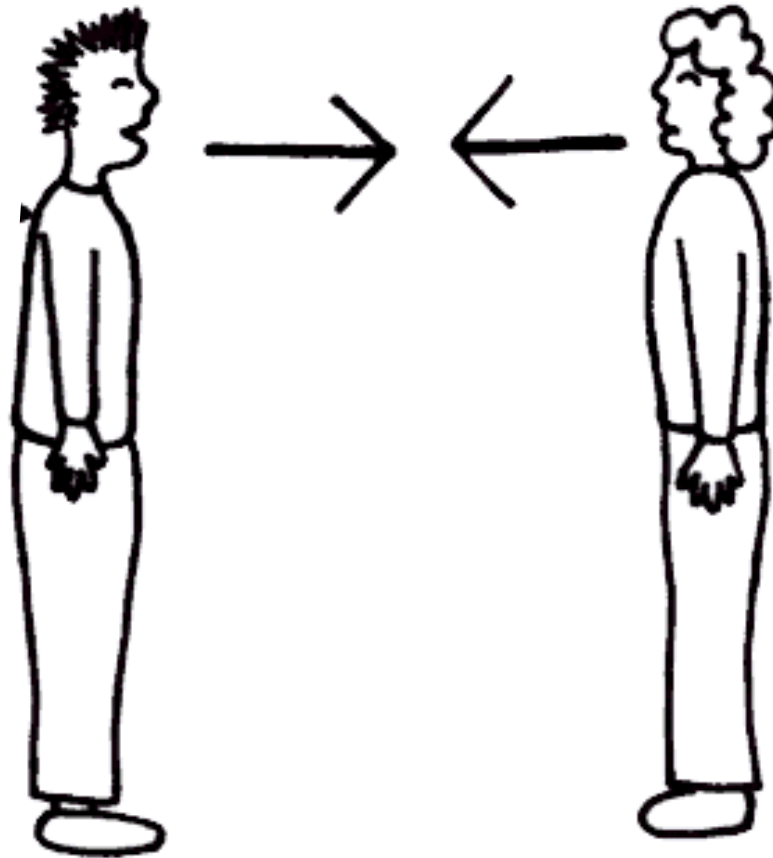
# AVR Interrupts

## *Assembly Language Programming*

Based on PP from Dr. Tim Margush, University of Akron

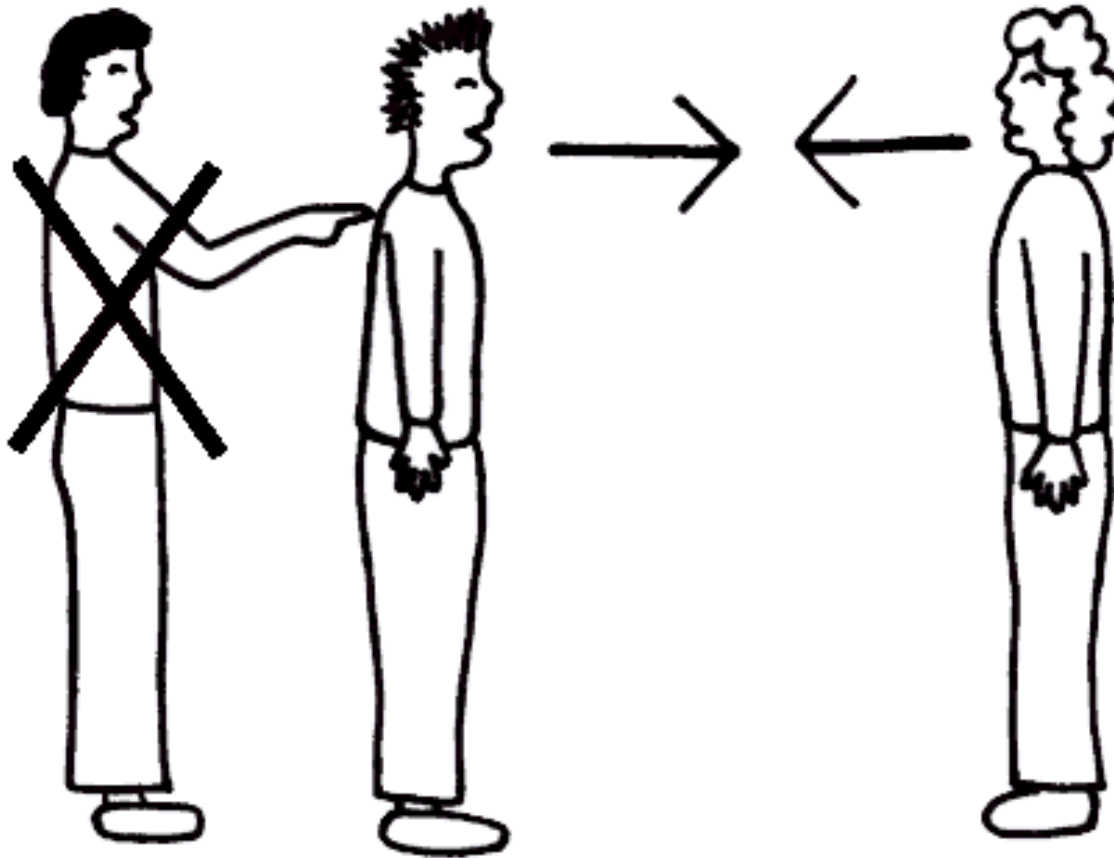


# What is interrupt?





# What is interrupt?





Time Out is a kind of interrupt.

**TIME  
OUT**



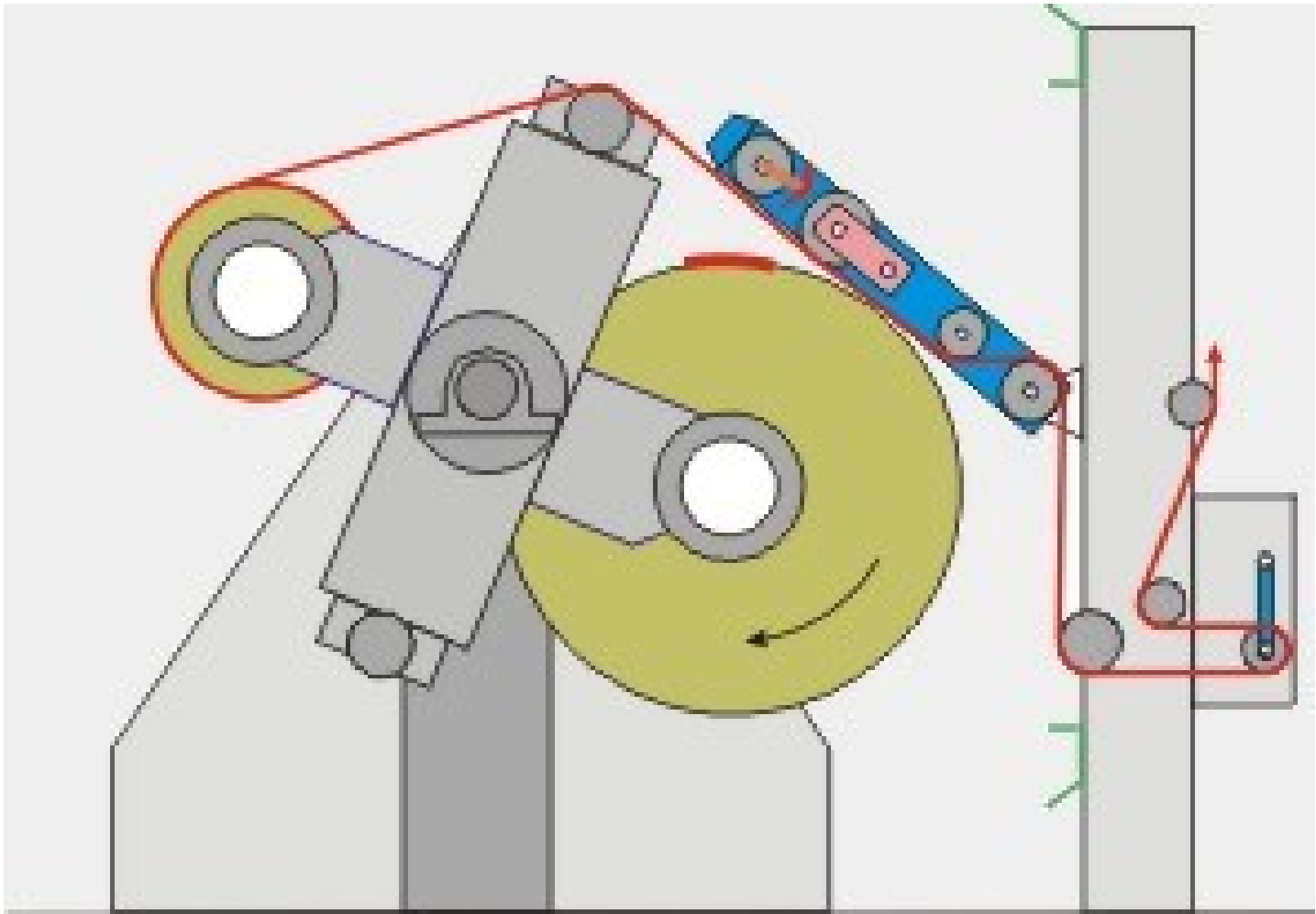


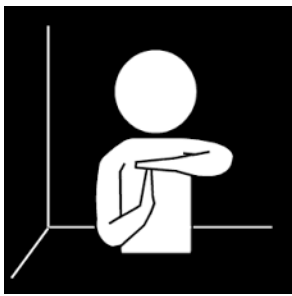
# Paper reel unwinder



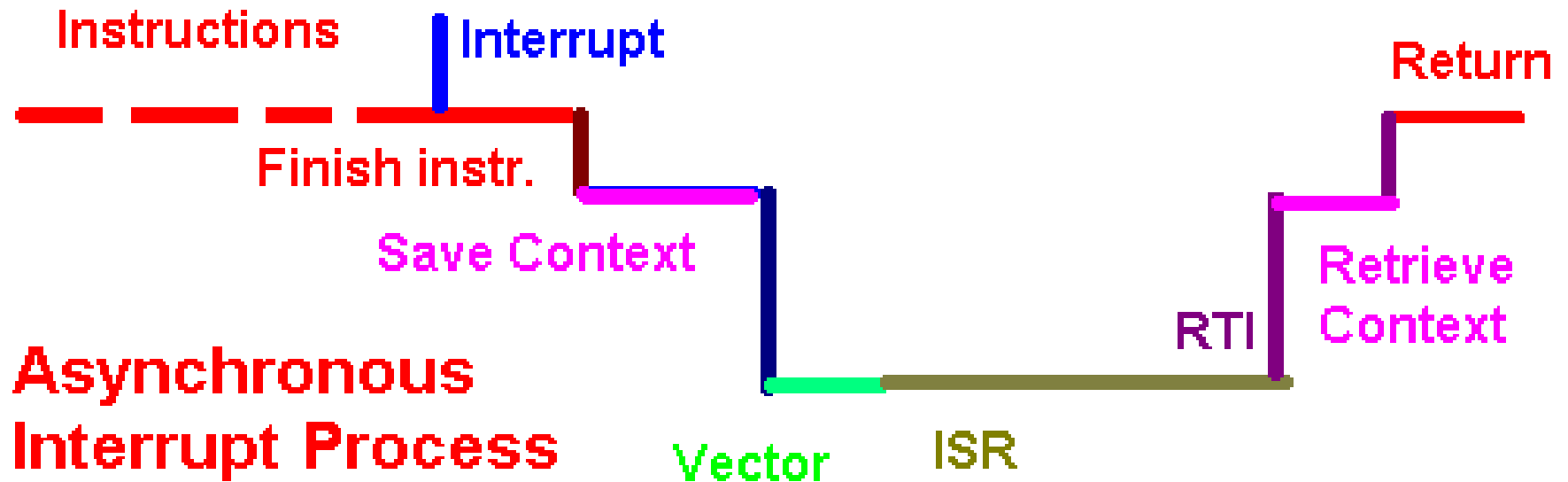


# Paper reel unwinder

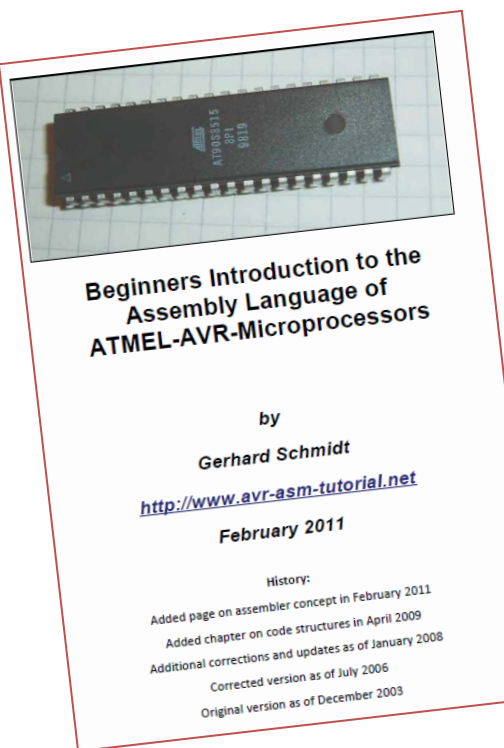




# Timing diagram







Page 37 – 38.

## 8.6 Interrupts and program execution

Very often we have to react on hardware conditions or other events. An example is a change on an input pin. You can program such a reaction by writing a loop, asking whether a change on the pin has occurred. This method is called polling, its like a bee running around in circles searching for new flowers. If there are no other things to do and reaction time does not matter, you can do this with the processor. If you have to detect short pulses of less than a  $\mu s$  duration this method is useless. In that case you need to program an interrupt.

An interrupt is triggered by some hardware conditions. All hardware interrupts are disabled at reset time by default, so the condition has to be enabled first. The respective port bits enabling the component's interrupt ability are set first. The processor has a bit in its status register enabling him to respond to the interrupt of all components, the Interrupt Enable Flag. Enabling the general response to interrupts requires the following instruction:

*SEI ; Set Int Enable Bit*

Each single interrupt requires additional port manipulation to be enabled.

If the interrupting condition occurs, e. g. a change on the port bit, the processor pushes the actual program counter to the stack (which must be enabled first! See initiation of the stackpointer in the Stack section of the SRAM description). Without that, the processor wouldn't be able to return back to the location, where the interrupt occurred (which could be any time and anywhere within program execution). After that, processing jumps to the predefined location, the interrupt vector, and executes the instructions there. Usually the instruction there is a JUMP instruction to the interrupt service routine, located somewhere in the code. The interrupt vector is a processor-specific location and depending from the hardware component and the condition that leads to the interrupt. The more hardware components and the more conditions, the more vectors. The different vectors for some older AVR types are listed in the following table. (The first vector isn't an interrupt but the reset vector, performing no stack operation!)

Name	Interrupt Vector Address			Triggered by
	2313	2323	8515	
RESET	0000	0000	0000	Hardware Reset, Power-On-Reset, Watchdog Reset
INT0	0001	0001	0001	Level change on the external INT0 pin
INT1	0002	-	0002	Level change on the external INT1 pin
TIMER1CAPT	0003	-	0003	Capture event on Timer/Counter 1
TIMER1COMPA	-	-	0004	Timer/Counter 1 = Compare value A
TIMER1 COMPB	-	-	0005	Timer/Counter 1 = Compare value B
TIMER1 COMP1	0004	-	-	Timer/Counter 1 = Compare value 1
TIMER1 OVf	0005	-	0006	Timer/Counter 1 Overflow
TIMER0 OVf	0006	0002	0007	Timer/Counter 0 Overflow
SPI STC	-	-	0008	Serial Transmit Complete
UART TX	0007	-	0009	UART char in receive buffer available
UART UDRE	0008	-	000A	UART transmitter ran empty
UART TX	0009	-	000B	UART All Sent
ANA_COMP	-	-	000C	Analog Comparator

Note that the capability to react to events is very different for the different types. The addresses are sequential, but not identical for different types. Consult the data sheet for each AVR type.

The higher a vector in the list the higher is its priority. If two or more components have an interrupt condition pending at the same time, the up most vector with the lower vector address wins. The lower int has to wait until the upper int was served. To disable lower ints from interrupting during the execution of its service routine the first executed int disables the processor's I-flag. The service routine must re-enable this flag after it is done with its job.

For re-setting the I status bit there are two ways. The service routine can end with the instruction:

*RETI*



doc2549\_ATmega2560.pdf  
page 105 – 117.

## Features

- High Performance, Low Power Atmel® AVR® 8-Bit Microcontroller
- Advanced RISC Architecture
  - 135 Powerful Instructions – Most Single Clock Cycle Execution
  - 32 × 8 General Purpose Working Registers
  - Fully Static Operation
  - Up to 16 MIPS Throughput at 16MHz
  - On-Chip 2-cycle Multiplier
- High Endurance Non-volatile Memory Segments
  - 64K/128K/256KBytes of In-System Self-Programmable Flash
  - 4Kbytes EEPROM
  - 8Kbytes Internal SRAM
  - Write/Erase Cycles:10,000 Flash/100,000 EEPROM
  - Data retention: 20 years at 85°C/ 100 years at 25°C
  - Optional Boot Code Section with Independent Lock Bits
    - In-System Programming by On-chip Boot Program
    - True Read-While-Write Operation
  - Programming Lock for Software Security
    - Endurance: Up to 64Kbytes Optional External Memory Space
- Atmel® QTouch® library support
  - Capacitive touch buttons, sliders and wheels
  - QTouch and QMatrix® acquisition
  - Up to 64 sense channels
- JTAG (IEEE std. 1149.1 compliant) Interface
  - Boundary-scan Capabilities According to the JTAG Standard
  - Extensive On-chip Debug Support
  - Programming of Flash, EEPROM, Fuses, and Lock Bits through the JTAG Interface
- Peripheral Features
  - Two 8-bit Timer/Counters with Separate Prescaler and Compare Mode
  - Four 16-bit Timer/Counter with Separate Prescaler, Compare- and Capture Mode
  - Real Time Counter with Separate Oscillator
  - Four 8-bit PWM Channels
  - Six/Twelve PWM Channels with Programmable Resolution from 2 to 16 Bits (ATmega1281/2561, ATmega640/1280/2560)
  - Output Compare Modulator
  - 8/16-channel, 10-bit ADC (ATmega1281/2561, ATmega640/1280/2560)
  - Two/Four Programmable Serial USART (ATmega1281/2561, ATmega640/1280/2560)
  - Master/Slave SPI Serial Interface
  - Byte Oriented 2-wire Serial Interface
  - Programmable Watchdog Timer with Separate On-chip Oscillator
  - On-chip Analog Comparator
  - Interrupt and Wake-up on Pin Change
- Special Microcontroller Features
  - Power-on Reset and Programmable Brown-out Detection
  - Internal Calibrated Oscillator
  - External and Internal Interrupt Sources
  - Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby, and Extended Standby
- I/O and Packages
  - 54/86 Programmable I/O Lines (ATmega1281/2561, ATmega640/1280/2560)
  - 64-pad QFN/MLF, 64-lead TQFP (ATmega1281/2561)
  - 100-lead TQFP, 100-ball CBGA (ATmega640/1280/2560)
  - RoHS/Fully Green
- Temperature Range:
  - -40°C to 85°C Industrial
- Ultra-Low Power Consumption
  - Active Mode: 1MHz, 1.8V; 500µA
  - Power-down Mode: 0.1µA at 1.8V
- Speed Grade:
  - ATmega640V/ATmega1280V/ATmega1281V:
    - 0 - 4MHz @ 1.8V - 5.5V, 0 - 8MHz @ 2.7V - 5.5V
  - ATmega2560V/ATmega2561V:
    - 0 - 2MHz @ 1.8V - 5.5V, 0 - 8MHz @ 2.7V - 5.5V
  - ATmega640/ATmega1280/ATmega1281:
    - 0 - 8MHz @ 2.7V - 5.5V, 0 - 16MHz @ 4.5V - 5.5V
  - ATmega2560/ATmega2561:
    - 0 - 16MHz @ 4.5V - 5.5V



**8-bit Atmel  
Microcontroller  
with  
64K/128K/256K  
Bytes In-System  
Programmable  
Flash**

**ATmega640/V  
ATmega1280/V  
ATmega1281/V  
ATmega2560/V  
ATmega2561/V**

**Preliminary**

2549N-AVR-05/11





14.1 Interrupt Vectors in  
ATmega640/1280/1281/2560/  
2561



14.2 Reset and Interrupt  
Vector placement



14.3 Moving Interrupts  
Between Application and Boot  
Section



14.4 Register Description



15. External Interrupts



15.1 Pin Change Interrupt  
Timing



15.2 Register Description



16. 8-bit Timer/Counter0 with  
PWM



17. 16-bit Timer/Counter

## Features

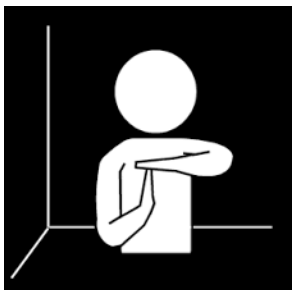
- High Performance, Low Power Atmel® AVR® 8-Bit Microcontroller
- Advanced RISC Architecture
  - 135 Powerful Instructions – Most Single Clock Cycle Execution
  - 32 x 8 General Purpose Working Registers
  - Fully Static Operation
  - Up to 16 MIPS Throughput at 16MHz
  - On-Chip 2-cycle Multiplier
- High Endurance Non-volatile Memory Segments
  - 64K/128K/256KBytes of In-System Self-Programmable Flash
  - 4Kbytes EEPROM
  - 8Kbytes Internal SRAM
  - Write/Erase Cycles: 10,000 Flash/100,000 EEPROM
  - Data retention: 20 years at 85°C/ 100 years at 25°C
  - Optional Boot Code Section with Independent Lock Bits
    - In-System Programming by On-chip Boot Program
    - True Read-While-Write Operation
  - Programming Lock for Software Security
    - Endurance: Up to 64Kbytes Optional External Memory Space
- Atmel® QTouch® library support
  - Capacitive touch buttons, sliders and wheels
  - QTouch and QMatrix® acquisition
  - Up to 64 sense channels
- JTAG (IEEE std. 1149.1 compliant) Interface
  - Boundary-scan Capabilities According to the JTAG Standard
  - Extensive On-chip Debug Support
  - Programming of Flash, EEPROM, Fuses, and Lock Bits through the JTAG Interface
- Peripheral Features
  - Two 8-bit Timer/Counters with Separate Prescaler and Compare Mode
  - Four 16-bit Timer/Counter with Separate Prescaler, Compare- and Capture Mode
  - Real Time Counter with Separate Oscillator
  - Four 8-bit PWM Channels
  - Six/Twelve PWM Channels with Programmable Resolution from 2 to 16 Bits (ATmega1281/2561, ATmega640/1280/2560)
  - Output Compare Modulator
  - 8/16-channel, 10-bit ADC (ATmega1281/2561, ATmega640/1280/2560)
  - Two/Four Programmable Serial USART (ATmega1281/2561, ATmega640/1280/2560)
  - Master/Slave SPI Serial Interface
  - Byte Oriented 2-wire Serial Interface
  - Programmable Watchdog Timer with Separate On-chip Oscillator
  - On-chip Analog Comparator
  - Interrupt and Wake-up on Pin Change
- Special Microcontroller Features
  - Power-on Reset and Programmable Brown-out Detection
  - Internal Calibrated Oscillator
  - External and Internal Interrupt Sources
  - Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby, and Extended Standby
- I/O and Packages
  - 54/86 Programmable I/O Lines (ATmega1281/2561, ATmega640/1280/2560)
  - 64-pad QFN/MLF, 64-lead TQFP (ATmega1281/2561)
  - 100-lead TQFP, 100-ball CBGA (ATmega640/1280/2560)
  - RoHS/Fully Green
- Temperature Range:
  - -40°C to 85°C Industrial
- Ultra-Low Power Consumption
  - Active Mode: 1MHz, 1.8V: 500µA
  - Power-down Mode: 0.1µA at 1.8V
- Speed Grade:
  - ATmega640V/ATmega1280V/ATmega1281V:
    - 0 - 4MHz @ 1.8V - 5.5V, 0 - 8MHz @ 2.7V - 5.5V
  - ATmega2560V/ATmega2561V:



8-bit Atmel  
Microcontroller  
with  
64K/128K/256K  
Bytes In-System  
Programmable  
Flash

ATmega640/V  
ATmega1280/V  
ATmega1281/V  
ATmega2560/V  
ATmega2561/V

Preliminary



## 14. Interrupts

This section describes the specifics of the interrupt handling as performed in ATmega640/1280/1281/2560/2561. For a general explanation of the AVR interrupt handling, refer to “Reset and Interrupt Handling” on page 18.

### 14.1 Interrupt Vectors in ATmega640/1280/1281/2560/2561

Table 14-1. Reset and Interrupt Vectors

Vector No.	Program Address <sup>(2)</sup>	Source	Interrupt Definition
1	\$0000 <sup>(1)</sup>	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 <sup>(3)</sup>	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
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



Table 14-1. Reset and Interrupt Vectors (Continued)

Vector No.	Program Address <sup>(2)</sup>	Source	Interrupt Definition
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 <sup>(3)</sup>	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 <sup>(3)</sup>	TIMER5 CAPT	Timer/Counter5 Capture Event
48	\$005E	TIMER5 COMPA	Timer/Counter5 Compare Match A
49	\$0060	TIMER5 COMPB	Timer/Counter5 Compare Match B
50	\$0062	TIMER5 COMPC	Timer/Counter5 Compare Match C
51	\$0064	TIMER5 OVF	Timer/Counter5 Overflow
52	\$0066 <sup>(3)</sup>	USART2 RX	USART2 Rx Complete
53	\$0068 <sup>(3)</sup>	USART2 UDRE	USART2 Data Register Empty
54	\$006A <sup>(3)</sup>	USART2 TX	USART2 Tx Complete
55	\$006C <sup>(3)</sup>	USART3 RX	USART3 Rx Complete
56	\$006E <sup>(3)</sup>	USART3 UDRE	USART3 Data Register Empty
57	\$0070 <sup>(3)</sup>	USART3 TX	USART3 Tx Complete

- Notes:
1. When the BOOTRST Fuse is programmed, the device will jump to the Boot Loader address at reset, see "Memory Programming" on page 335.
  2. When the IVSEL bit in MCUCR is set, Interrupt Vectors will be moved to the start of the Boot Flash Section. The address of each Interrupt Vector will then be the address in this table added to the start address of the Boot Flash Section.
  3. Only available in ATmega640/1280/2560.



## 14.2 Reset and Interrupt Vector placement

Table 14-2 shows Reset and Interrupt Vectors placement for the various combinations of BOOTRST and IVSEL settings. If the program never enables an interrupt source, the Interrupt Vectors are not used, and regular program code can be placed at these locations. This is also the case if the Reset Vector is in the Application section while the Interrupt Vectors are in the Boot section or vice versa.

Table 14-2. Reset and Interrupt Vectors Placement<sup>(1)</sup>

BOOTRST	IVSEL	Reset Address	Interrupt Vectors Start Address
1	0	0x0000	0x0002
1	1	0x0000	Boot Reset Address + 0x0002
0	0	Boot Reset Address	0x0002
0	1	Boot Reset Address	Boot Reset Address + 0x0002

Note: 1. The Boot Reset Address is shown in Table 29-7 on page 328 through Table 29-15 on page 332. For the BOOTRST Fuse "1" means unprogrammed while "0" means programmed.

The most typical and general program setup for the Reset and Interrupt Vector Addresses in ATmega640/1280/1281/2560/2561 is:

Address	Labels	Code	Comments
0x0000		jmp RESET	; Reset Handler
0x0002		jmp INT0	; IRQ0 Handler
0x0004		jmp INT1	; IRQ1 Handler
0x0006		jmp INT2	; IRQ2 Handler
0x0008		jmp INT3	; IRQ3 Handler
0x000A		jmp INT4	; IRQ4 Handler
0x000C		jmp INT5	; IRQ5 Handler
0x000E		jmp INT6	; IRQ6 Handler
0x0010		jmp INT7	; IRQ7 Handler
0x0012		jmp PCINT0	; PCINT0 Handler
0x0014		jmp PCINT1	; PCINT1 Handler
0x0016		jmp PCINT2	; PCINT2 Handler
0x0018		jmp WDT	; Watchdog Timeout Handler
0x001A		jmp TIM2_COMPA	; Timer2 CompareA Handler
0x001C		jmp TIM2_COMPB	; Timer2 CompareB Handler
0x001E		jmp TIM2_OVF	; Timer2 Overflow Handler
0x0020		jmp TIM1_CAPT	; Timer1 Capture Handler
0x0022		jmp TIM1_COMPA	; Timer1 CompareA Handler
0x0024		jmp TIM1_COMPB	; Timer1 CompareB Handler
0x0026		jmp TIM1_COMPC	; Timer1 CompareC Handler
0x0028		jmp TIM1_OVF	; Timer1 Overflow Handler
0x002A		jmp TIM0_COMPA	; Timer0 CompareA Handler
0x002C		jmp TIM0_COMPB	; Timer0 CompareB Handler
0x002E		jmp TIM0_OVF	; Timer0 Overflow Handler
0x0030		jmp SPI_STC	; SPI Transfer Complete Handler
0x0032		jmp USART0_RXC	; USART0 RX Complete Handler
0x0034		jmp USART0_UDRE	; USART0, UDR Empty Handler
0x0036		jmp USART0_TXC	; USART0 TX Complete Handler
0x0038		jmp ANA_COMP	; Analog Comparator Handler
0x003A		jmp ADC	; ADC Conversion Complete Handler
0x003C		jmp EE_RDY	; EEPROM Ready Handler
0x003E		jmp TIM3_CAPT	; Timer3 Capture Handler





## ATmega640/1280/1281/2560/2561

```

0x0040      jmp     TIM3_COMPA      ; Timer3 CompareA Handler
0x0042      jmp     TIM3_COMPB      ; Timer3 CompareB Handler
0x0044      jmp     TIM3_COMPC      ; Timer3 CompareC Handler
0x0046      jmp     TIM3_OVF        ; Timer3 Overflow Handler
0x0048      jmp     USART1_RXC      ; USART1 RX Complete Handler
0x004A      jmp     USART1_UDRE     ; USART1,UDR Empty Handler
0x004C      jmp     USART1_TXC      ; USART1 TX Complete Handler
0x004E      jmp     TWI             ; 2-wire Serial Handler
0x0050      jmp     SPM_RDY         ; SPM Ready Handler
0x0052      jmp     TIM4_CAPT       ; Timer4 Capture Handler
0x0054      jmp     TIM4_COMPA      ; Timer4 CompareA Handler
0x0056      jmp     TIM4_COMPB      ; Timer4 CompareB Handler
0x0058      jmp     TIM4_COMPC      ; Timer4 CompareC Handler
0x005A      jmp     TIM4_OVF        ; Timer4 Overflow Handler
0x005C      jmp     TIM5_CAPT       ; Timer5 Capture Handler
0x005E      jmp     TIM5_COMPA      ; Timer5 CompareA Handler
0x0060      jmp     TIM5_COMPB      ; Timer5 CompareB Handler
0x0062      jmp     TIM5_COMPC      ; Timer5 CompareC Handler
0x0064      jmp     TIM5_OVF        ; Timer5 Overflow Handler
0x0066      jmp     USART2_RXC      ; USART2 RX Complete Handler
0x0068      jmp     USART2_UDRE     ; USART2,UDR Empty Handler
0x006A      jmp     USART2_TXC      ; USART2 TX Complete Handler
0x006C      jmp     USART3_RXC      ; USART3 RX Complete Handler
0x006E      jmp     USART3_UDRE     ; USART3,UDR Empty Handler
0x0070      jmp     USART3_TXC      ; USART3 TX Complete Handler

```

```

;
0x0072  RESET:  ldi     r16, high(RAMEND) ; Main program start
0x0073          out     SPH,r16          ; Set Stack Pointer to top of RAM
0x0074          ldi     r16, low(RAMEND)
0x0075          out     SPL,r16
0x0076          sei                      ; Enable interrupts
0x0077          <instr> xxx

```

When the BOOTRST Fuse is unprogrammed, the Boot section size set to 8Kbytes and the IVSEL bit in the MCUCR Register is set before any interrupts are enabled, the most typical and general program setup for the Reset and Interrupt Vector Addresses is:

```

Address  Labels Code      Comments
0x00000  RESET:  ldi     r16,high(RAMEND); Main program start
0x00001          out     SPH,r16          ; Set Stack Pointer to top of RAM
0x00002          ldi     r16,low(RAMEND)
0x00003          out     SPL,r16
0x00004          sei                      ; Enable interrupts
0x00005          <instr>  xxx
;
.org 0x1F002
0x1F002      jmp     EXT_INT0      ; IRQ0 Handler
0x1F004      jmp     EXT_INT1      ; IRQ1 Handler
...          ...                  ;
0x1F070      jmp     USART3_TXC    ; USART3 TX Complete Handler

```



## ATmega640/1280/1281/2560/2561

When the BOOTRST Fuse is programmed and the Boot section size set to 8Kbytes, the most typical and general program setup for the Reset and Interrupt Vector Addresses is:

```

Address  Labels Code          Comments
.org 0x0002
0x00002      jmp    EXT_INT0      ; IRQ0 Handler
0x00004      jmp    EXT_INT1      ; IRQ1 Handler
...          ...      ;
0x00070      jmp    USART3_TXC    ; USART3 TX Complete Handler
;
.org 0x1F000
0x1F000 RESET: ldi    r16,high(RAMEND); Main program start
0x1F001      out    SPH,r16      ; Set Stack Pointer to top of RAM
0x1F002      ldi    r16,low(RAMEND)
0x1F003      out    SPL,r16
0x1F004      sei                    ; Enable interrupts
0x1F005      <instr> xxx

```

When the BOOTRST Fuse is programmed, the Boot section size set to 8Kbytes and the IVSEL bit in the MCUCR Register is set before any interrupts are enabled, the most typical and general program setup for the Reset and Interrupt Vector Addresses is:

```

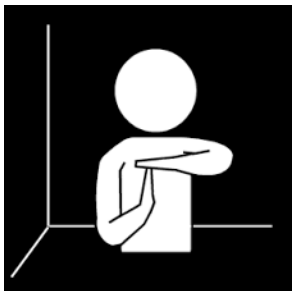
Address  Labels Code          Comments
;
.org 0x1F000
0x1F000      jmp    RESET        ; Reset handler
0x1F002      jmp    EXT_INT0      ; IRQ0 Handler
0x1F004      jmp    EXT_INT1      ; IRQ1 Handler
...          ...      ;
0x1F070      jmp    USART3_TXC    ; USART3 TX Complete Handler
;
0x1F072 RESET: ldi    r16,high(RAMEND) ; Main program start
0x1F073      out    SPH,r16      ; Set Stack Pointer to top of RAM
0x1F074      ldi    r16,low(RAMEND)
0x1F075      out    SPL,r16
0x1F076      sei                    ; Enable interrupts
0x1F077      <instr> xxx

```

### 14.3 Moving Interrupts Between Application and Boot Section

The MCU Control Register controls the placement of the Interrupt Vector table, see Code Example below. For more details, see ["Reset and Interrupt Handling" on page 18](#).





## Assembly Code Example

```
Move_interrupts:
; Get MCUCR
in r16, MCUCR
mov r17, r16

; Enable change of Interrupt Vectors
ori r16, (1<<IVCE)
out MCUCR, r16
; Move interrupts to Boot Flash section
ori r16, (1<<IVSEL)
out MCUCR, r17
ret
```

## C Code Example

```
void Move_interrupts(void)
{
uchar temp;
/* Get MCUCR*/
temp = MCUCR;
/* Enable change of Interrupt Vectors */
MCUCR = temp | (1<<IVCE);
/* Move interrupts to Boot Flash section */
MCUCR = temp | (1<<IVSEL);
}
```

## 14.4 Register Description

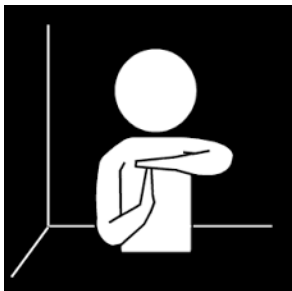
### 14.4.1 MCUCR – MCU Control Register

Bit	7	6	5	4	3	2	1	0	
0x35 (0x55)	JTD	–	–	PUD	–	–	IVSEL	IVCE	MCUCR
Read/Write	R/W	R	R	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

#### • Bit 1 – IVSEL: Interrupt Vector Select

When the IVSEL bit is cleared (zero), the Interrupt Vectors are placed at the start of the Flash memory. When this bit is set (one), the Interrupt Vectors are moved to the beginning of the Boot Loader section of the Flash. The actual address of the start of the Boot Flash Section is determined by the BOOTSZ Fuses. Refer to the section [“Memory Programming”](#) on page 335 for details. To avoid unintentional changes of Interrupt Vector tables, a special write procedure must be followed to change the IVSEL bit (see [“Moving Interrupts Between Application and Boot Section”](#) on page 109):

1. Write the Interrupt Vector Change Enable (IVCE) bit to one.
2. Within four cycles, write the desired value to IVSEL while writing a zero to IVCE.



---

## ATmega640/1280/1281/2560/2561

Interrupts will automatically be disabled while this sequence is executed. Interrupts are disabled in the cycle IVCE is set, and they remain disabled until after the instruction following the write to IVSEL. If IVSEL is not written, interrupts remain disabled for four cycles. The I-bit in the Status Register is unaffected by the automatic disabling.

**Note:** If Interrupt Vectors are placed in the Boot Loader section and Boot Lock bit BLB02 is programmed, interrupts are disabled while executing from the Application section. If Interrupt Vectors are placed in the Application section and Boot Lock bit BLB12 is programmed, interrupts are disabled while executing from the Boot Loader section. Refer to the section ["Memory Programming" on page 335](#) for details on Boot Lock bits.

- **Bit 0 – IVCE: Interrupt Vector Change Enable**

The IVCE bit must be written to logic one to enable change of the IVSEL bit. IVCE is cleared by hardware four cycles after it is written or when IVSEL is written. Setting the IVCE bit will disable interrupts, as explained in the IVSEL description.



### 15. External Interrupts

The External Interrupts are triggered by the INT7:0 pin or any of the PCINT23:0 pins. Observe that, if enabled, the interrupts will trigger even if the INT7:0 or PCINT23:0 pins are configured as outputs. This feature provides a way of generating a software interrupt.

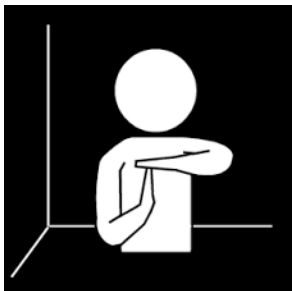
The Pin change interrupt PCI2 will trigger if any enabled PCINT23:16 pin toggles, Pin change interrupt PCI1 if any enabled PCINT15:8 toggles and Pin change interrupts PCI0 will trigger if any enabled PCINT7:0 pin toggles. PCMSK2, PCMSK1 and PCMSK0 Registers control which pins contribute to the pin change interrupts. Pin change interrupts on PCINT23:0 are detected asynchronously. This implies that these interrupts can be used for waking the part also from sleep modes other than Idle mode.

The External Interrupts can be triggered by a falling or rising edge or a low level. This is set up as indicated in the specification for the External Interrupt Control Registers – EICRA (INT3:0) and EICRB (INT7:4). When the external interrupt is enabled and is configured as level triggered, the interrupt will trigger as long as the pin is held low. Note that recognition of falling or rising edge interrupts on INT7:4 requires the presence of an I/O clock, described in [“Overview” on page 40](#). Low level interrupts and the edge interrupt on INT3:0 are detected asynchronously. This implies that these interrupts can be used for waking the part also from sleep modes other than Idle mode. The I/O clock is halted in all sleep modes except Idle mode.

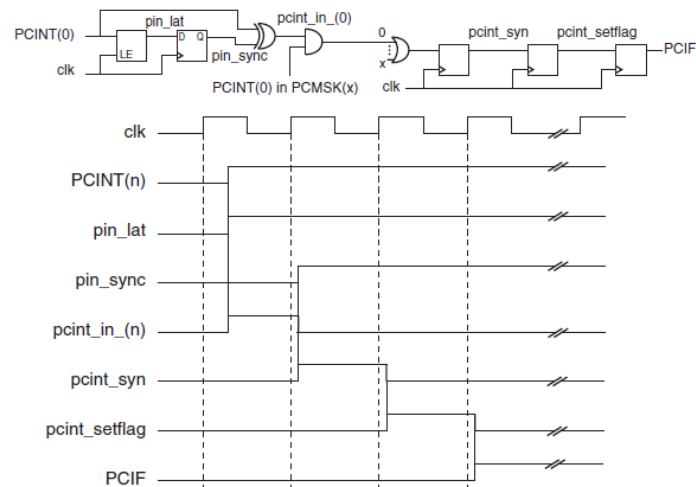
Note that if a level triggered interrupt is used for wake-up from Power-down, the required level must be held long enough for the MCU to complete the wake-up to trigger the level interrupt. If the level disappears before the end of the Start-up Time, the MCU will still wake up, but no interrupt will be generated. The start-up time is defined by the SUT and CKSEL Fuses as described in [“System Clock and Clock Options” on page 40](#).

#### 15.1 Pin Change Interrupt Timing

An example of timing of a pin change interrupt is shown in [Figure 15-1 on page 113](#).



**Figure 15-1.** Normal pin change interrupt.



## 15.2 Register Description

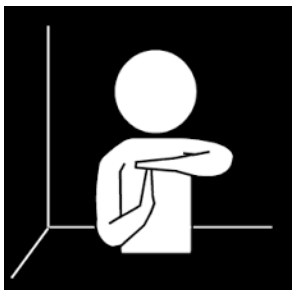
### 15.2.1 EICRA – External Interrupt Control Register A

The External Interrupt Control Register A contains control bits for interrupt sense control.

Bit	7	6	5	4	3	2	1	0	
(0x09)	ISC31	ISC30	ISC21	ISC20	ISC11	ISC10	ISC01	ISC00	EICRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

#### • Bits 7:0 – ISC31, ISC30 – ISC00, ISC00: External Interrupt 3 - 0 Sense Control Bits

The External Interrupts 3 - 0 are activated by the external pins INT3:0 if the SREG I-flag and the corresponding interrupt mask in the EIMSK is set. The level and edges on the external pins that activate the interrupts are defined in [Table 15-1 on page 114](#). Edges on INT3:0 are registered asynchronously. Pulses on INT3:0 pins wider than the minimum pulse width given in [Table 15-2 on page 114](#) will generate an interrupt. Shorter pulses are not guaranteed to generate an interrupt. If low level interrupt is selected, the low level must be held until the completion of the currently executing instruction to generate an interrupt. If enabled, a level triggered interrupt will generate an interrupt request as long as the pin is held low. When changing the ISCn bit, an interrupt can occur. Therefore, it is recommended to first disable INTn by clearing its Interrupt Enable bit in the EIMSK Register. Then, the ISCn bit can be changed. Finally, the INTn interrupt flag should be cleared by writing a logical one to its Interrupt Flag bit (INTFn) in the EIFR Register before the interrupt is re-enabled.



**Table 15-1. Interrupt Sense Control<sup>(1)</sup>**

ISn1	ISn0	Description
0	0	The low level of INTn generates an interrupt request
0	1	Any edge of INTn generates asynchronously an interrupt request
1	0	The falling edge of INTn generates asynchronously an interrupt request
1	1	The rising edge of INTn generates asynchronously an interrupt request

Note: 1. n = 3, 2, 1 or 0.  
When changing the ISn1/ISn0 bits, the interrupt must be disabled by clearing its Interrupt Enable bit in the EIMSK Register. Otherwise an interrupt can occur when the bits are changed.

**Table 15-2. Asynchronous External Interrupt Characteristics**

Symbol	Parameter	Condition	Min	Typ	Max	Units
$t_{INT}$	Minimum pulse width for asynchronous external interrupt			50		ns

## 15.2.2 EICRB – External Interrupt Control Register B

Bit	7	6	5	4	3	2	1	0	
(0x6A)	ISC71	ISC70	ISC61	ISC60	ISC51	ISC50	ISC41	ISC40	EICRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

### • Bits 7:0 – ISC71, ISC70 – ISC41, ISC40: External Interrupt 7 - 4 Sense Control Bits

The External Interrupts 7 - 4 are activated by the external pins INT7:4 if the OREG1 flag and the corresponding interrupt mask in the EIMSK is set. The level and edges on the external pins that activate the interrupts are defined in Table 15-3. The value on the INT7:4 pins are sampled before detecting edges. If edge or toggle interrupt is selected, pulses that last longer than one clock period will generate an interrupt. Shorter pulses are not guaranteed to generate an interrupt. Observe that CPU clock frequency can be lower than the XTAL frequency if the XTAL divider is enabled. If low level interrupt is selected, the low level must be held until the completion of the currently executing instruction to generate an interrupt. If enabled, a level triggered interrupt will generate an interrupt request as long as the pin is held low.

**Table 15-3. Interrupt Sense Control<sup>(1)</sup>**

ISn1	ISn0	Description
0	0	The low level of INTn generates an interrupt request
0	1	Any logical change on INTn generates an interrupt request
1	0	The falling edge between two samples of INTn generates an interrupt request
1	1	The rising edge between two samples of INTn generates an interrupt request

Note: 1. n = 7, 6, 5 or 4.  
When changing the ISn1/ISn0 bits, the interrupt must be disabled by clearing its Interrupt Enable bit in the EIMSK Register. Otherwise an interrupt can occur when the bits are changed.



## 15.2.3 EIMSK – External Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
0x1D (0x3D)	INT7	INT6	INT5	INT4	INT3	INT2	INT1	INT0	EIMSK
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

### • Bits 7:0 – INT7:0: External Interrupt Request 7 - 0 Enable

When an INT7:0 bit is written to one and the I-bit in the Status Register (SREG) is set (one), the corresponding external pin interrupt is enabled. The Interrupt Sense Control bits in the External Interrupt Control Registers – EICRA and EICRB – defines whether the external interrupt is activated on rising or falling edge or level sensed. Activity on any of these pins will trigger an interrupt request even if the pin is enabled as an output. This provides a way of generating a software interrupt.

## 15.2.4 EIFR – External Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0	
0x1C (0x3C)	INTF7	INTF6	INTF5	INTF4	INTF3	INTF2	INTF1	INTF0	EIFR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

### • Bits 7:0 – INTF7:0: External Interrupt Flags 7 - 0

When an edge or logic change on the INT7:0 pin triggers an interrupt request, INTF7:0 becomes set (one). If the I-bit in SREG and the corresponding interrupt enable bit, INT7:0 in EIMSK, are set (one), the MCU will jump to the interrupt vector. The flag is cleared when the interrupt routine is executed. Alternatively, the flag can be cleared by writing a logical one to it. These flags are always cleared when INT7:0 are configured as level interrupt. Note that when entering sleep mode with the INT3:0 interrupts disabled, the input buffers on these pins will be disabled. This may cause a logic change in internal signals which will set the INTF3:0 flags. See [“Digital Input Enable and Sleep Modes” on page 74](#) for more information.

## 15.2.5 PCICR – Pin Change Interrupt Control Register

Bit	7	6	5	4	3	2	1	0	
(0x68)	–	–	–	–	–	PCIE2	PCIE1	PCIE0	PCICR
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

### • Bit 2 – PCIE2: Pin Change Interrupt Enable 1

When the PCIE2 bit is set (one) and the I-bit in the Status Register (SREG) is set (one), pin change interrupt 2 is enabled. Any change on any enabled PCINT23:16 pin will cause an interrupt. The corresponding interrupt of Pin Change Interrupt Request is executed from the PC12 Interrupt Vector. PCINT23:16 pins are enabled individually by the PCMSK2 Register.

### • Bit 1 – PCIE1: Pin Change Interrupt Enable 1

When the PCIE1 bit is set (one) and the I-bit in the Status Register (SREG) is set (one), pin change interrupt 1 is enabled. Any change on any enabled PCINT15:8 pin will cause an interrupt. The corresponding interrupt of Pin Change Interrupt Request is executed from the PC11 Interrupt Vector. PCINT15:8 pins are enabled individually by the PCMSK1 Register.



- **Bit 0 – PCIE0: Pin Change Interrupt Enable 0**

When the PCIE0 bit is set (one) and the I-bit in the Status Register (SREG) is set (one), pin change interrupt 0 is enabled. Any change on any enabled PCINT7:0 pin will cause an interrupt. The corresponding interrupt of Pin Change Interrupt Request is executed from the PCIO Interrupt Vector. PCINT7:0 pins are enabled individually by the PCMSK0 Register.

## 15.2.6 PCIFR – Pin Change Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0	
0x1B (0x3B)	–	–	–	–	–	PCIF2	PCIF1	PCIF0	PCIFR
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 2 – PCIF2: Pin Change Interrupt Flag 1**

When a logic change on any PCINT23:16 pin triggers an interrupt request, PCIF2 becomes set (one). If the I-bit in SREG and the PCIE2 bit in PCICR are set (one), the MCU will jump to the corresponding Interrupt Vector. The flag is cleared when the interrupt routine is executed. Alternatively, the flag can be cleared by writing a logical one to it.

- **Bit 1 – PCIF1: Pin Change Interrupt Flag 1**

When a logic change on any PCINT15:8 pin triggers an interrupt request, PCIF1 becomes set (one). If the I-bit in SREG and the PCIE1 bit in PCICR are set (one), the MCU will jump to the corresponding Interrupt Vector. The flag is cleared when the interrupt routine is executed. Alternatively, the flag can be cleared by writing a logical one to it.

- **Bit 0 – PCIF0: Pin Change Interrupt Flag 0**

When a logic change on any PCINT7:0 pin triggers an interrupt request, PCIF0 becomes set (one). If the I-bit in SREG and the PCIE0 bit in PCICR are set (one), the MCU will jump to the corresponding Interrupt Vector. The flag is cleared when the interrupt routine is executed. Alternatively, the flag can be cleared by writing a logical one to it.

## 15.2.7 PCMSK2 – Pin Change Mask Register 2

Bit	7	6	5	4	3	2	1	0	
(0x6D)	PCINT23	PCINT22	PCINT21	PCINT20	PCINT19	PCINT18	PCINT17	PCINT16	PCMSK2
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7:0 – PCINT23:16: Pin Change Enable Mask 23:16**

Each PCINT23:16-bit selects whether pin change interrupt is enabled on the corresponding I/O pin. If PCINT23:16 is set and the PCIE2 bit in PCICR is set, pin change interrupt is enabled on the corresponding I/O pin. If PCINT23:16 is cleared, pin change interrupt on the corresponding I/O pin is disabled.

## 15.2.8 PCMSK1 – Pin Change Mask Register 1

Bit	7	6	5	4	3	2	1	0	
(0x6C)	PCINT15	PCINT14	PCINT13	PCINT12	PCINT11	PCINT10	PCINT9	PCINT8	PCMSK1
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	



- **Bit 7:0 – PCINT15:8: Pin Change Enable Mask 15:8**

Each PCINT15:8-bit selects whether pin change interrupt is enabled on the corresponding I/O pin. If PCINT15:8 is set and the PCIE1 bit in EIMSK is set, pin change interrupt is enabled on the corresponding I/O pin. If PCINT15:8 is cleared, pin change interrupt on the corresponding I/O pin is disabled.

## 15.2.9 PCMSK0 – Pin Change Mask Register 0

Bit	7	6	5	4	3	2	1	0	
(0x6B)	PCINT7	PCINT6	PCINT5	PCINT4	PCINT3	PCINT2	PCINT1	PCINT0	PCMSK0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7:0 – PCINT7:0: Pin Change Enable Mask 7:0**

Each PCINT7:0 bit selects whether pin change interrupt is enabled on the corresponding I/O pin. If PCINT7:0 is set and the PCIE0 bit in PCICR is set, pin change interrupt is enabled on the corresponding I/O pin. If PCINT7:0 is cleared, pin change interrupt on the corresponding I/O pin is disabled.



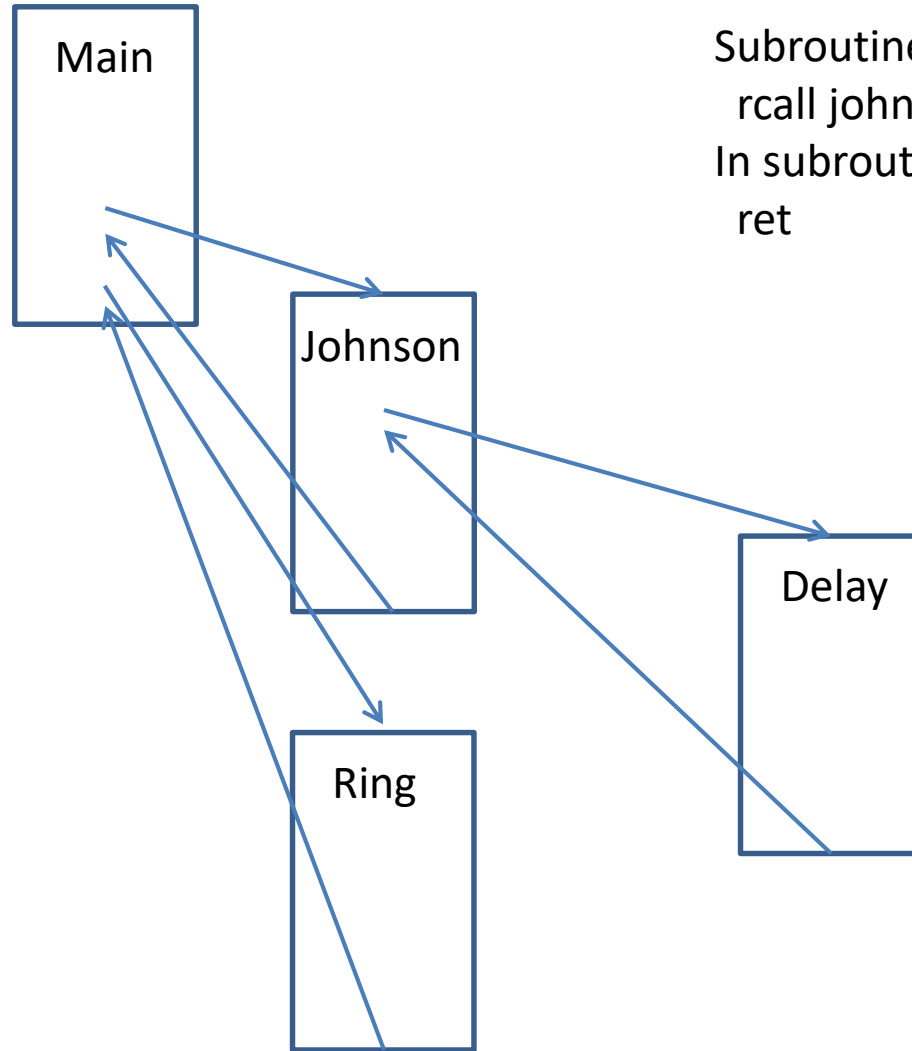


# What is an Interrupt

- A condition or event that interrupts the normal flow of control in a program
- Interrupt hardware inserts a function call between instructions to service the interrupt condition
- When the interrupt handler is finished, the normal program resumes execution



# Program structure, example.



Subroutine call, example:

`rcall johnsson`

In subroutine:

`ret`



# Interrupt Sources

- Interrupts are generally classified as
  - internal or external
  - software or hardware
- An external interrupt is triggered by a device originating off-chip
- An internal interrupt is triggered by an on-chip component



# Interrupt Sources

- Hardware interrupts occur due to a change in state of some hardware
- Software interrupts are triggered by the execution of a machine instruction



# Interrupt Handler

- An interrupt handler (or interrupt service routine) is a function ending with the special return from interrupt instruction (RETI)
- Interrupt handlers are not explicitly called; their address is placed into the processor's program counter by the interrupt hardware



# AVR Interrupt System

- The ATMega2560 can respond to 57 different interrupts (21 for ATMega16)
- Interrupts are numbered by priority from 1 to 57
  - The reset interrupt is interrupt number 1
- Each interrupt invokes a handler at a specific address in program memory
  - The reset handler is located at address \$0000

# JMP – Jump

## Description:

Jump to an address within the entire 4M (words) Program memory. See also RJMP.

This instruction is not available in all devices. Refer to the device specific instruction set summary.

### Operation:

(i)  $PC \leftarrow k$

### Syntax:

(i) `JMP k`

### Operands:

$0 \leq k < 4M$

### Program Counter:

$PC \leftarrow k$

### Stack:

Unchanged

### 32-bit Opcode:

1001	010k	kkkk	110k
kkkk	kkkk	kkkk	kkkk

## Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

## Example:

```
mov    r1,r0    ; Copy r0 to r1
jmp     farplc   ; Unconditional jump
...
farplc: nop      ; Jump destination (do nothing)
```

Words: 2 (4 bytes)

Cycles: 3

## RJMP – Relative Jump

### Description:

Relative jump to an address within  $PC - 2K + 1$  and  $PC + 2K$  (words). In the assembler, labels are used instead of relative operands. For AVR microcontrollers with Program memory not exceeding 4K words (8K bytes) this instruction can address the entire memory from every address location.

### Operation:

- (i)  $PC \leftarrow PC + k + 1$

### Syntax:

- (i) RJMP k

### Operands:

$$-2K \leq k < 2K$$

### Program Counter:

$$PC \leftarrow PC + k + 1$$

### Stack

Unchanged

### 16-bit Opcode:

1100	kkkk	kkkk	kkkk
------	------	------	------

### Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

### Example:

```
    cpi    r16,$42    ; Compare r16 to $42
    brne   error      ; Branch if r16 <> $42
    rjmp    ok         ; Unconditional branch
error:  add    r16,r17  ; Add r17 to r16
        inc    r16      ; Increment r16
ok:     nop           ; Destination for rjmp (do nothing)
```

9/16/ Words: 1 (2 bytes)

Cycles: 2





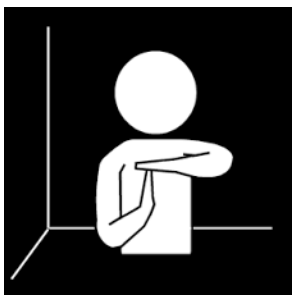
# Interrupt Vectors

- The interrupt handler for interrupt  $k$  is located at address  $2(k-1)$  in program memory
  - Address \$0000 is the reset interrupt
  - Address \$0002 is external interrupt 0
  - Address \$0004 is external interrupt 1
- Because there is room for only one or two instructions, each interrupt handler begins with a jump to another location in program memory where the rest of the code is found
  - *jmp handler* is a 32-bit instruction, hence each handler is afforded 2 words of space in this low memory area



# Interrupt Vector Table

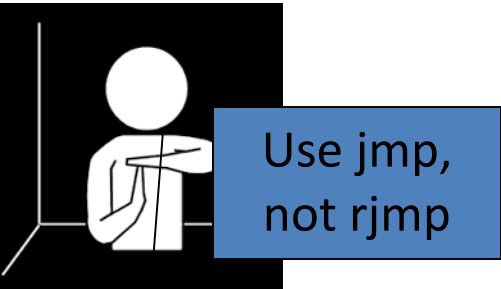
- The 21 instructions at address \$0000 through \$0029 comprise the interrupt vector table
- These jump instructions vector the processor to the actual service routine code
  - A long JMP is used so the code can be at any address in program memory
- An interrupt handler that does nothing could simply have an RETI instruction in the table



# Interrupt Vector (ATMega16)

The most typical and general program setup for the Reset and Interrupt Vector Addresses in ATmega16 is:

Address	Labels	Code	Comments
\$000		jmp RESET	; Reset Handler
\$002		jmp EXT_INT0	; IRQ0 Handler
\$004		jmp EXT_INT1	; IRQ1 Handler
\$006		jmp TIM2_COMP	; Timer2 Compare Handler
\$008		jmp TIM2_OVF	; Timer2 Overflow Handler
\$00A		jmp TIM1_CAPT	; Timer1 Capture Handler
\$00C		jmp TIM1_COMPA	; Timer1 CompareA Handler
\$00E		jmp TIM1_COMPB	; Timer1 CompareB Handler
\$010		jmp TIM1_OVF	; Timer1 Overflow Handler
\$012		jmp TIM0_OVF	; Timer0 Overflow Handler
\$014		jmp SPI_STC	; SPI Transfer Complete Handler
\$016		jmp USART_RXC	; USART RX Complete Handler
\$018		jmp USART_UDRE	; UDR Empty Handler
\$01A		jmp USART_TXC	; USART TX Complete Handler
\$01C		jmp ADC	; ADC Conversion Complete Handler
\$01E		jmp EE_RDY	; EEPROM Ready Handler
\$020		jmp ANA_COMP	; Analog Comparator Handler
\$022		jmp TWSI	; Two-wire Serial Interface Handler
\$024		jmp EXT_INT2	; IRQ2 Handler
\$026		jmp TIM0_COMP	; Timer0 Compare Handler
\$028		jmp SPM_RDY	; Store Program Memory Ready Handler
		;	
\$02A	RESET:	ldi r16,high(RAMEND)	; Main program start
\$02B		out SPH,r16	; Set Stack Pointer to top of RAM
\$02C		ldi r16,low(RAMEND)	
\$02D		out SPL,r16	



Use jmp,  
not rjmp

# Typical IVT

```
.cseg
.org 0
jmp reset
jmp external_int_0
jmp external_int_1
.org UDREaddr
jmp transmitByte
    ..etc...
.org $2A
reset:
```

- If you omit some vectors, you must use .org to locate the vectors appropriately
  - The interrupt vector addresses are defined in the include file
- The \$2A address is just beyond the vector table
  - A lower address can be used if the corresponding interrupts are never enabled



# Interrupt Enabling

- Each potential interrupt source can be individually enabled or disabled
  - The reset interrupt is the one exception; it cannot be disabled
- The global interrupt flag must be set (enabled) in SREG, for interrupts to occur
  - Again, the reset interrupt will occur regardless



# Interrupt Actions

- If
  - global interrupts are enabled
  - AND a specific interrupt is enabled
  - AND the interrupt condition is present
- Then the interrupt will occur
- What actually happens?
  - At the completion of the current instruction,
    - the current PC is pushed on the stack
    - global interrupts are disabled
    - the proper interrupt vector address is placed in PC



# Return From Interrupt

- The RETI instruction will
  - pop the address from the top of the stack into the PC
  - set the global interrupt flag, re-enabling interrupts
- This causes the next instruction of the previously interrupted program to be executed
  - At least one instruction will be executed before another interrupt can occur



# Stack

- Since interrupts require stack access, it is essential that the reset routine initialize the stack before enabling interrupts
- Interrupt service routines should use the stack for temporary storage so register values can be preserved





# Status Register

- Interrupt routines MUST LEAVE the status register unchanged

`typical_interrupt_handler:`

```
    push r0                ; save r0 on Stack
    in r0, SREG             ; read Status Register
    push r0                ; save SREG on Stack
```

`...`

`...`

`...`

```
    pop r0                 ; fetch SREG from Stack
    out SREG, r0           ; restore SREG
    pop r0                 ; restore r0
    reti
```



# Interrupt Variations

- AVR Interrupts fall into two classes
  - Event based interrupts
    - Triggered by some event; must be cleared by taking some program action
  - Condition based interrupts
    - Asserted while some condition is true; cleared automatically when the condition becomes false



# Event-based Interrupts

- Even if interrupts are disabled, the corresponding interrupt flag may be set by the associated event
- Once set, the flag remains set, and will trigger an interrupt as soon as interrupts are enabled
  - This type of interrupt flag is cleared
    - by manually by writing a 1 to it
    - automatically when the interrupt occurs



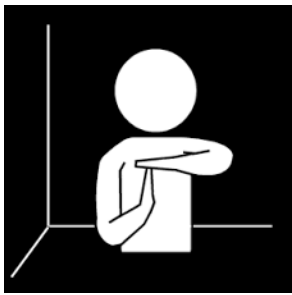
# Condition-based Interrupts

- Even if interrupts are disabled, the interrupt flag will be set when the associated condition is true
- If the condition becomes false before interrupts are enabled, the flag will clear and the interrupt will be missed
  - These flags are cleared when the condition becomes false
  - Some program action may be required to accomplish this



# Sample Interrupts

- Event-based
  - Edge-triggered external interrupts
  - Timer/counter overflows and output compare
- Condition-based
  - Level triggered external interrupts
  - USART Data Ready, Receive Complete
  - EEPROM Ready

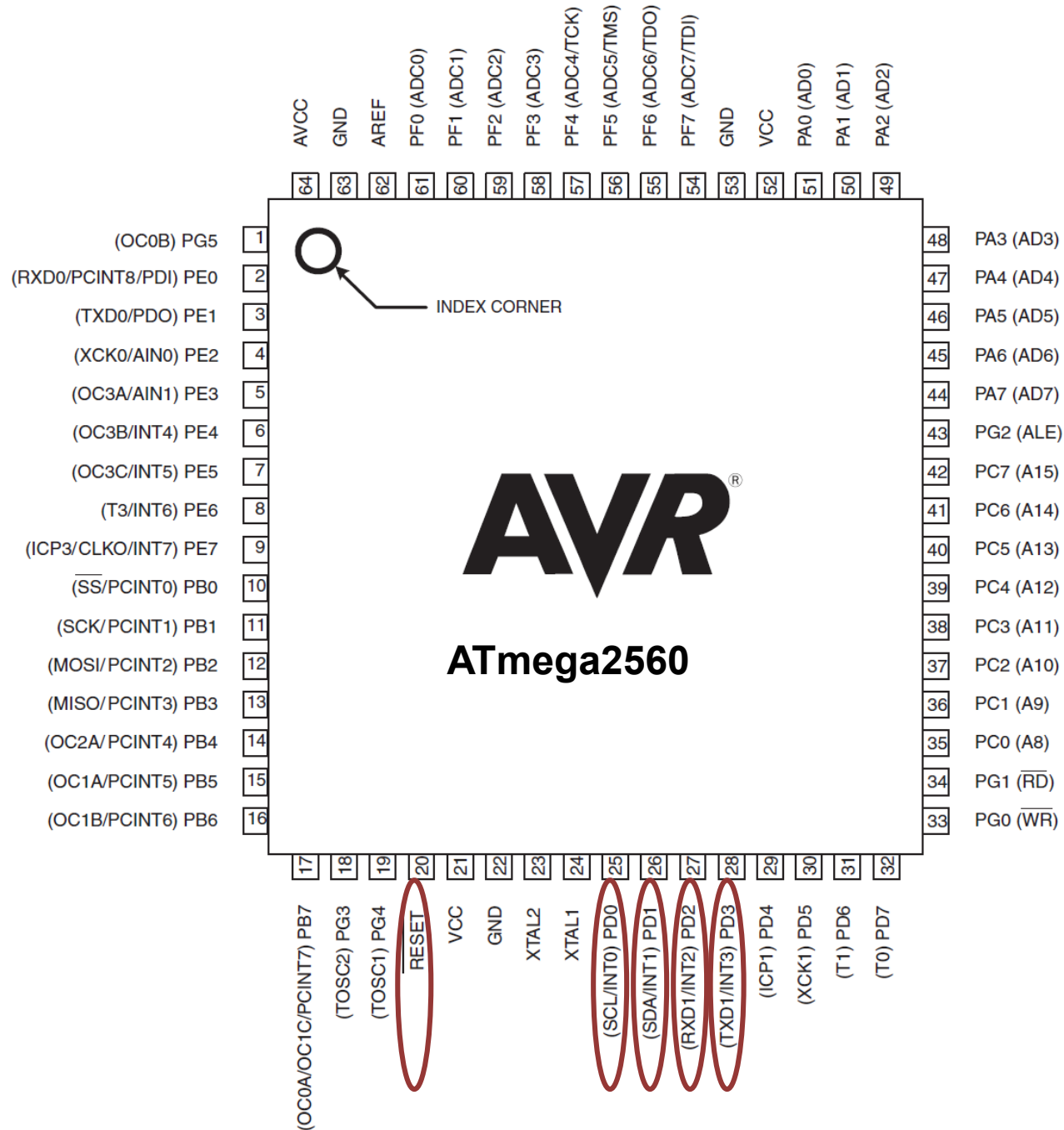
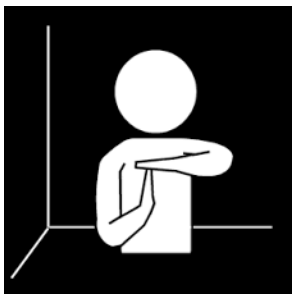


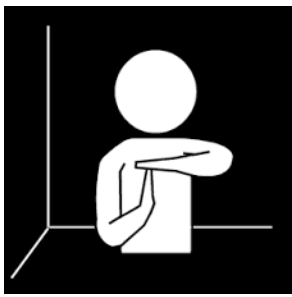
## PDIP

(XCK/T0) PB0	□	1	40	□	PA0 (ADC0)
(T1) PB1	□	2	39	□	PA1 (ADC1)
(INT2/AIN0) PB2	□	3	38	□	PA2 (ADC2)
(OC0/AIN1) PB3	□	4	37	□	PA3 (ADC3)
( $\overline{SS}$ ) PB4	□	5	36	□	PA4 (ADC4)
(MOSI) PB5	□	6	35	□	PA5 (ADC5)
(MISO) PB6	□	7	34	□	PA6 (ADC6)
(SCK) PB7	□	8	33	□	PA7 (ADC7)
$\overline{RESET}$	□	9	32	□	AREF
VCC	□	10	31	□	GND
GND	□	11	30	□	AVCC
XTAL2	□	12	29	□	PC7 (TOSC2)
XTAL1	□	13	28	□	PC6 (TOSC1)
(RXD) PD0	□	14	27	□	PC5 (TDI)
(TXD) PD1	□	15	26	□	PC4 (TDO)
(INT0) PD2	□	16	25	□	PC3 (TMS)
(INT1) PD3	□	17	24	□	PC2 (TCK)
(OC1B) PD4	□	18	23	□	PC1 (SDA)
(OC1A) PD5	□	19	22	□	PC0 (SCL)
(ICP1) PD6	□	20	21	□	PD7 (OC2)

ATMega16

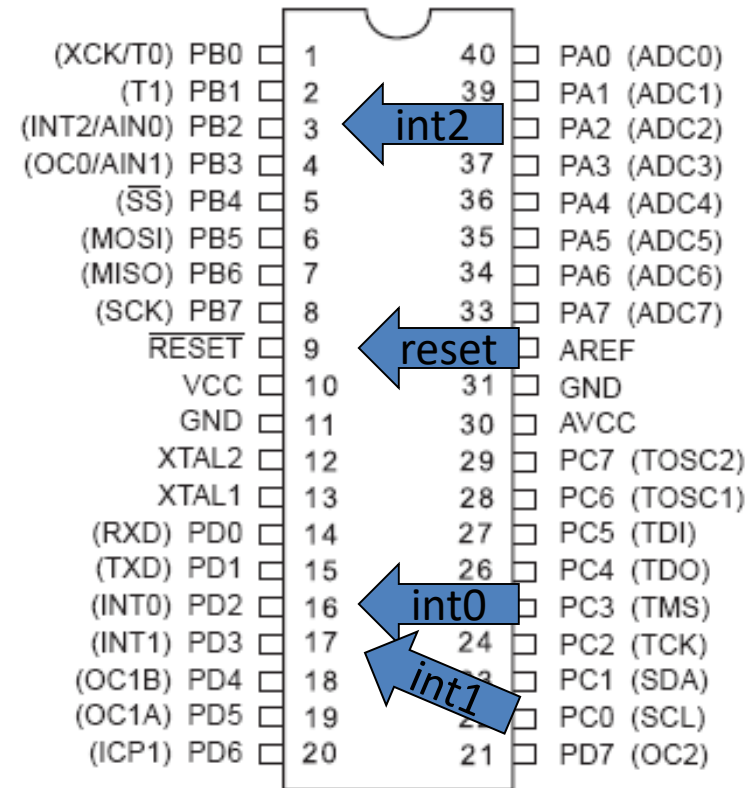
Figure 1-3. Pinout ATmega1281/2561





# External Interrupts

- The ATmega16 responds to 4 different external interrupts – signals applied to specific pins
- RESET (pin 9)
- INT0 (pin 16 – also PD2)
- INT1 (pin 17 – also PD3)
- INT2 (pin 3 – also PB3)

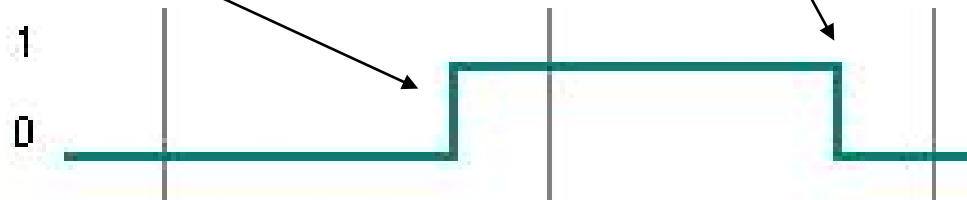






# Internal Interrupt Configuration

- Condition-based
  - while level is low
- Event-based triggers
  - level has changed (toggle)
  - falling (negative) edge (1 to 0 transition)
  - rising (positive) edge (0 to 1 transition)





# Level Triggers

- The processor samples the levels on pins INT0 and INT1 each clock cycle
- Very short pulses (less than one cycle) may go undetected (no change or edge is seen)
  - The low level interrupt will occur only if the pin is low at the end of the current instruction



# EIMSK – External Interrupt Mask Register

- General Interrupt Control Register

EIMSK – External Interrupt Mask Register

7	6	5	4	3	2	1	0
INT7	INT6	INT5	INT4	INT3	INT2	INT1	INT0

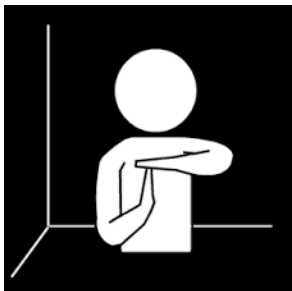
- Each external interrupt is enabled or disabled here

```
;Enable int1, disable int0  
in R16, EIMSK  
sbr R16, 1<<INT1  
cbr R16, 1<<INT0  
out EIMSK, R16
```



# Software Interrupt

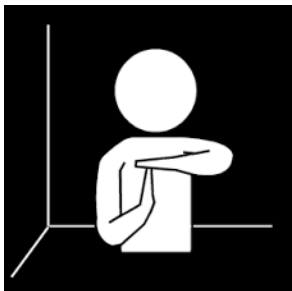
- If the external interrupt pins are configured as outputs, a program may assert 0 or 1 values on the interrupt pins
  - This action can trigger interrupts according to the external interrupt settings
- Since a program instruction causes the interrupt, this is called a software interrupt



```
.org 0x00
    rjmp start

.org INT1addr    ;rjmp to interrupt routine
    rjmp interrupt_int1









.org 0x30
start:
```



```
.org 0x00
    rjmp start

.org INT1addr    ;rjmp to interrupt routine
    rjmp interrupt_int1

.org 0x30
start:
..
..
interrupt_int1:    ;interrupt routine for int1
    push..
    ..
    ..
    pop..
    reti
```

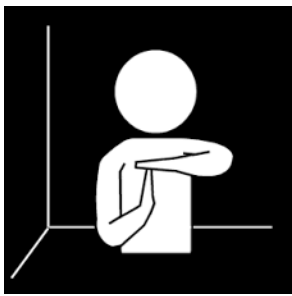
Name	Value
Program Counter	0x000002
Stack Pointer	0x21FC
X pointer	0x000000
Y pointer	0x000000
Z pointer	0x000000
Cycle Counter	52
Frequency	4.0000 MHz
Stop Watch	13.00 us
SREG	       

C:\Document\Kurser\D2 E2\Datorteknik\HT\_2019\Program examples\Lecture examp...

[illegible]

[illegible]





# Program example, cont.

```
C:\Document\_Kursen\D2_E2\Dator teknik\HT_2015\Program_examples\Test_Inte...
; Initialize SP, Stack Pointer
ldi r20, HIGH(RAMEND) ; R20 = high part of RAMEND address
out SPH,R20           ; SPH = high part of RAMEND address
ldi R20, low(RAMEND)  ; R20 = low part of RAMEND address
out SPL,R20           ; SPL = low part of RAMEND address

ldi r16, 0x00
out DDRD, r16         ; port D - input
out DDRA, r16         ; port A - input

ldi r16, 0xFF
;out PORTD, r16       ; set pull-up resistors on D input pin
out DDRE, r16         ; port E - output
out DDRB, r16         ; port E - output

ldi r16, 0b00000011   ; INTO and INT1 enable
out EIMSK, r16

ldi r16, 0b00001000   ; INT1 falling edge, INTO rising edge
sts EICRA, r16

sei                  ; Global interrupt enable

; main program
ldi r16, 1
main_program:
nop
rjmp main_program

interrupt_0:
interrupt_1:
com r16
out PORTB, r16
com r16
out PORTE, r16

lsl r16

ldi r22,200
delay_int:
dec r22
cpi r22,0
brne delay_int

reti
```

# RJMP – Relative Jump

## Description:

Relative jump to an address within  $PC - 2K + 1$  and  $PC + 2K$  (words). In the assembler, labels are used instead of relative operands. For AVR microcontrollers with Program memory not exceeding 4K words (8K bytes) this instruction can address the entire memory from every address location.

## Operation:

- (i)  $PC \leftarrow PC + k + 1$

## Syntax:

- (i) RJMP k

## Operands:

$$-2K \leq k < 2K$$

## Program Counter:

$$PC \leftarrow PC + k + 1$$

## Stack

Unchanged

## 16-bit Opcode:

1100	kkkk	kkkk	kkkk
------	------	------	------

1100 0000 0111 0001

## Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N
-	-	-	-	-	-

## Example:

```
    cpi    r16,$42    ; Compare r16 to $42
    brne   error      ; Branch if r16 <> $42
    rjmp    ok         ; Unconditional branch
error:  add    r16,r17  ; Add r17 to r16
        inc    r16      ; Increment r16
ok:     nop           ; Destination for rjmp (do nothing)
```

71 C0 in PM is C0 71  
(little endian!)

$$C0\ 71 = 1100\ 0000\ 0111\ 0001$$

$$k = 0111\ 0001_2 = 71_{16} = 113_{10}$$

Name	Value
Program Counter	0x000085
Stack Pointer	0x21FF
X pointer	0x000000
Y pointer	0x000000
Z pointer	0x000000
Cycle Counter	219
Frequency	4.0000 MHz
Stop Watch	54.75 us
SREG	
<b>Registers</b>	
R00	0x00
R01	0x00
R02	0x00
R03	0x00
R04	0x00
R05	0x00
R06	0x00
R07	0x00
R08	0x00
R09	0x00
R10	0x00
R11	0x00
R12	0x00
R13	0x00
R14	0x00
R15	0x00
R16	0x26
R17	0x03
R18	0x00
R19	0x00
R20	0xFF
R21	0x00
R22	0x00
R23	0x00
R24	0x00
R25	0x00
R26	0x00
R27	0x00
R28	0x00
R29	0x00
R30	0x00
R31	0x00

[illegible][illegible]