



1DT301, Computer Technology I

Lecture #6,

Wednesday, September 18, 2019

- Interrupt, cont.
- Introduction to lab 3
- Simulator example
- Program example from exam

1DT301, Computer Technology I, autumn 2019. Lab. 3: Interrupts.

Goal for this lab:

Learn how to use interrupts for digital inputs.

Development environment: AVR Studio4 or AVR Studio6

You have to write comments that explain the code in each program. The structure of the program has to be explained with Flow Charts for each program. (each task)

Presentation of results:

Present each task for the teacher when you have solved the task. A written report should be submitted after each lab, containing the code and a brief description of results. The report must also include flowcharts for all programs. The report should be sent to the lab teacher within 1 week, thus before next week. Use text in the programs (comments) to explain the function. Each program must have a header like the example below.

1DT301, Computer Technology I
Date: 2015-09-03

Author:

Student name 1
Student name 2

Lab number: 3

Title: How to use interrupts

Hardware: STK600, CPU ATmega2560

Function: Describe the function of the program, so that you can understand even if you're viewing this in a year from now!

Input ports: Describe the function of used ports, for example on-board connected to PORTA.

Output ports: Describe the function of used ports, for example on-board connected to PORTB.

Subroutines: If applicable.
Included files: m2560def.inc

Other information:

Changes in program: (Description and date)



Task 1:

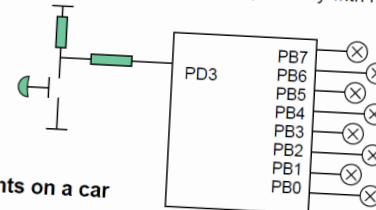
Write a program that turns **ON** and **OFF** a LED with a push button. The LED will be extinguished when pressing the button.

The program will use Interrupt. Connect the push buttons to PORT D.

The program should have a main program that runs in a loop and wait for the interrupts. An interrupt routine is called when the push button is pressed. Each time the button is pressed, the lamp should switch from **'OFF'** to **'ON'**, or from **'ON'** to **'OFF'**.

Task 2: Switch – Ringcounter / Johnsoncounter, with interrupt

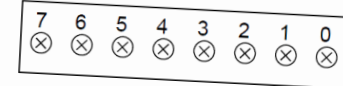
Write a program that by means of a switch can choose to flash 8 LEDs either in the form of a ring counter or in the form of a Johnson counter. Use the switch SW0 connected to PORTD to switch between the two counters. Each time the button is pressed, a shift between the two counters should take place. By using interrupts you'll swap directly with no delay.



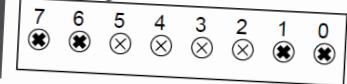
Task 3: Rear lights on a car

Interrupt.

Program that simulates the rear lights on a car
The 8 LEDs should behave like the rear lights.



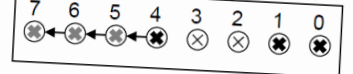
Normal light



Turning right. 3-2-1-0 ring counter.



Turning left. 4-5-6-7 ring counter.



Function:

Normal light: LED 0, 1, 6 and 7 'ON'.

Turning right:

LED 6 – 7 on, LED 0 – 3 blinking as RING counter.

Turning left:

LED 0 – 1 on, LED 4 – 7 blinking as RING counter.

See manual for how to set Interrupt registers: EICRA and EIMSK

Task 4: Rear lights on a car, with light for brakes

Add function for the stop light to the previous task. When braking, all LEDs light up, if blink on the right or left is not going on.

Turning right and brake:

LED 4 – 7 on, LED 0 – 3 blinking as RING counter.

Turning left and brake:

LED 0 – 3 on, LED 4 – 7 blinking as RING counter.
Use INT2 for the Brake.



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 ⁽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
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 ⁽²⁾	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 ⁽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
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

- Notes:
1. When the BOOTSZ 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⁽¹⁾

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

```

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

```

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.

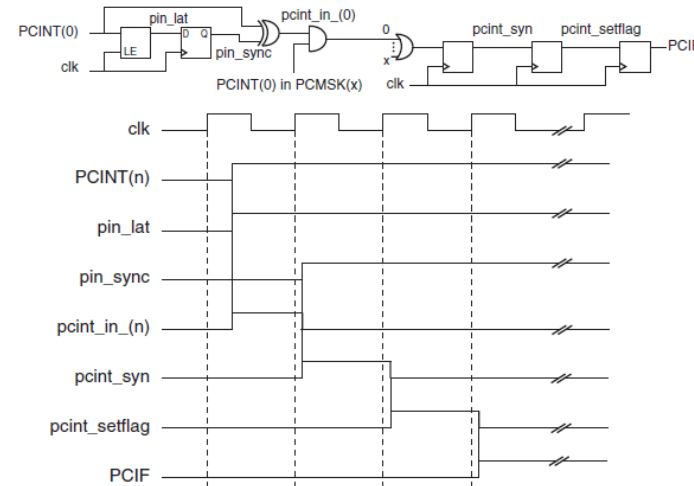
The External Interrupts can be triggered by a falling or rising edge or a low level. This is set up as indicated in the specification for the External Interrupt Control Registers – EICRA (INT3:0) and EICRB (INT7:4). When the external interrupt is enabled and is configured as level triggered, the interrupt 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⁽¹⁾

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⁽¹⁾

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.

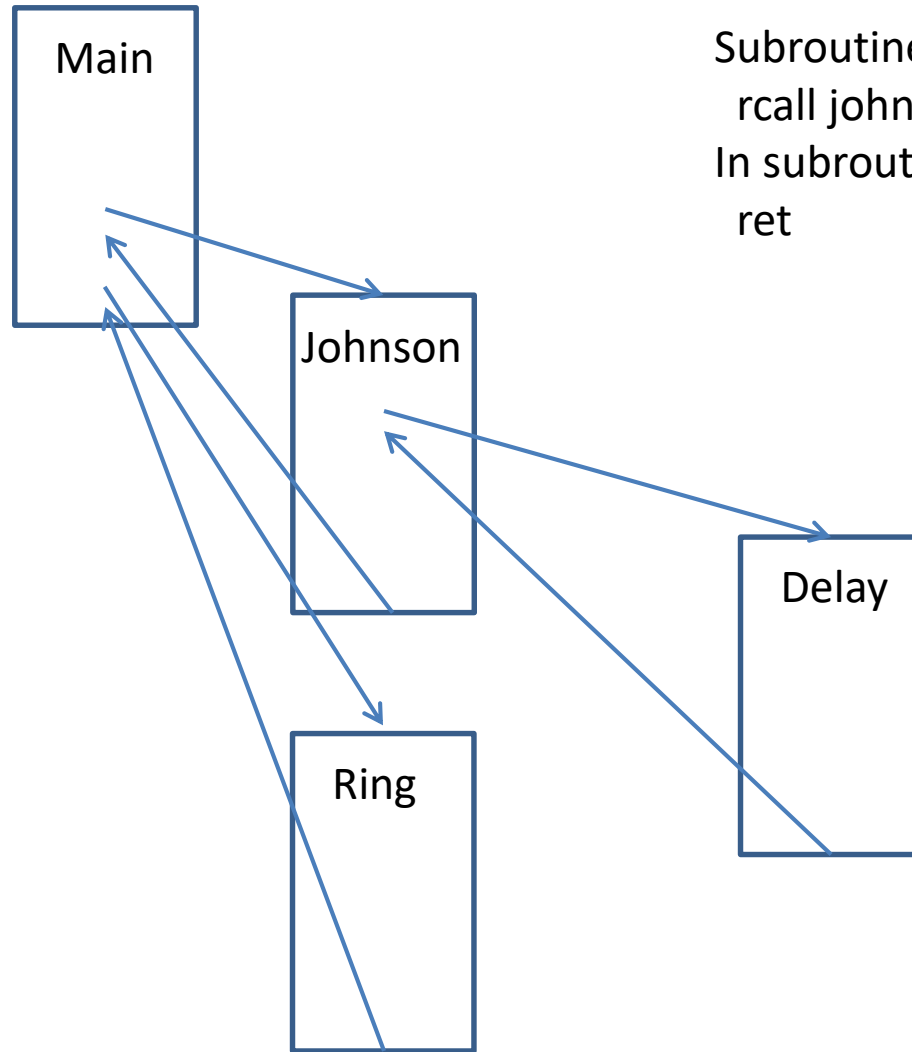
• Bit 1 – PCIE1: Pin Change Interrupt Enable 1

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.

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

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

```
label_1:
```

```
;  
|
```

```
inc r20           ; increase counter in r20  
cpi r20, 10       ; check if equal 20  
brne label_1      ; if not, go back
```

BRNE – Branch if Not Equal

Description:

Conditional relative branch. Tests the Zero Flag (Z) and branches relatively to PC if Z is cleared. If the instruction is executed immediately after any of the instructions CP, CPI, SUB or SUBI, the branch will occur if and only if the unsigned or signed binary number represented in Rd was not equal to the unsigned or signed binary number represented in Rr. This instruction branches relatively to PC in either direction ($PC - 63 \leq \text{destination} \leq PC + 64$). The parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBC 1,k).

Operation:

- (i) If $Rd \neq Rr$ ($Z = 0$) then $PC \leftarrow PC + k + 1$, else $PC \leftarrow PC + 1$

Syntax:

- (i) BRNE k

Operands:

$-64 \leq k \leq +63$

Program Counter:

$PC \leftarrow PC + k + 1$

$PC \leftarrow PC + 1$, if condition is false

16-bit Opcode:

1111	01kk	kkkk	k001
------	------	------	------

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
eor    r27,r27    ; Clear r27
loop:  inc    r27    ; Increase r27
...
cpi    r27,5      ; Compare r27 to 5
brne   loop       ; Branch if r27<>5
nop                    ; Loop exit (do nothing)
```

Words: 1 (2 bytes)

Cycles: 1 if condition is false

2 if condition is true

CPI – Compare with Immediate

Description:

This instruction performs a compare between register Rd and a constant. The register is not changed. All conditional branches can be used after this instruction.

Operation:

(i) $Rd - K$

Syntax:

(i) $CPI\ Rd, K$

Operands:

$16 \leq d \leq 31, 0 \leq K \leq 255$

Program Counter:

$PC \leftarrow PC + 1$

16-bit Opcode:

0011	KKKK	dddd	KKKK
------	------	------	------

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	\Leftrightarrow	\Leftrightarrow	\Leftrightarrow	\Leftrightarrow	\Leftrightarrow	\Leftrightarrow

H: $\overline{Rd3} \bullet K3 + K3 \bullet R3 + R3 \bullet \overline{Rd3}$
Set if there was a borrow from bit 3; cleared otherwise

S: $N \oplus V$, For signed tests.

V: $Rd7 \bullet \overline{K7} \bullet \overline{R7} + \overline{Rd7} \bullet K7 \bullet R7$
Set if two's complement overflow resulted from the operation; cleared otherwise.

N: $R7$
Set if MSB of the result is set; cleared otherwise.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result is \$00; cleared otherwise.

C: $\overline{Rd7} \bullet K7 + K7 \bullet R7 + R7 \bullet \overline{Rd7}$
Set if the absolute value of K is larger than the absolute value of Rd; cleared otherwise.

R (Result) after the operation.

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
```




Assembler

- 5p

[illegible]

```
subr_01:      push counter
               push temp
               clr counter
               andi r20, 0x7F
               mov temp, r20
```

```
shift:      lsr temp
            brcc next
            inc counter
```

```
next:          cpi temp, 0
               brne shift

               lsr counter
               brcc end
               ori r20, 0x80
```

```
end:
    pop temp
    pop counter
    ret
```

5.

- Analyze the program below and explain how the program works and what it will do. Input data in register R20. Explain what will happen with this data and where to find the result.
- Make a flowchart of the program that explains the function.

[illegible]

```

subr_01:      push counter      ; Store registers on stack, counter and temp
              push temp        ; counter = 0000 0000
              clr counter
              andi r20, 0x7F
              mov temp, r20

shift:        lsr temp
              brcc next
              inc counter

next:         cpi temp, 0
              brne shift

              lsr counter
              brcc end
              ori r20, 0x80

end:          pop temp
              pop counter
              ret

```

ANDI – Logical AND with Immediate

Description:

Performs the logical AND between the contents of register Rd and a constant and places the result in the destination register Rd.

Operation:

(i) $Rd \leftarrow Rd \bullet K$

Syntax:

(i) ANDI Rd,K

Operands:

$16 \leq d \leq 31, 0 \leq K \leq 255$

Program Counter:

$PC \leftarrow PC + 1$

16-bit Opcode:

0111	KKKK	dddd	KKKK
------	------	------	------

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	\Leftrightarrow	0	\Leftrightarrow	\Leftrightarrow	–

S: $N \oplus V$, For signed tests.

V: 0
Cleared

N: R7
Set if MSB of the result is set; cleared otherwise.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result is \$00; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```
andi r17,$0F ; Clear upper nibble of r17
andi r18,$10 ; Isolate bit 4 in r18
andi r19,$AA ; Clear odd bits of r19
```

Words: 1 (2 bytes)

Cycles: 1

5.

- Analyze the program below and explain how the program works and what it will do. Input data in register R20. Explain what will happen with this data and where to find the result. 5p
- Make a flowchart of the program that explains the function. 5p

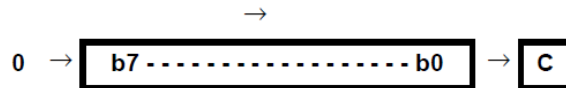
[illegible]

LSR – Logical Shift Right

Description:

Shifts all bits in Rd one place to the right. Bit 7 is cleared. Bit 0 is loaded into the C Flag of the SREG. This operation effectively divides an unsigned value by two. The C Flag can be used to round the result.

Operation:



Syntax: **Operands:** **Program Counter:**
(i) LSR Rd $0 \leq d \leq 31$ $PC \leftarrow PC + 1$

16-bit Opcode:

1001	010d	dddd	0110
------	------	------	------

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	↔	↔	0	↔	↔

S: $N \oplus V$, For signed tests.

V: $N \oplus C$ (For N and C after the shift)

N: 0

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result is \$00; cleared otherwise.

C: Rd0
Set if, before the shift, the LSB of Rd was set; cleared otherwise.

R (Result) equals Rd after the operation.

5.

- 5p

5p

subr_01:

```
; result in r20
```

```
; shift r20 right 1 bit, bit 0 -> Carry flag
```

```
ori r20, 0x80
```

```
ret
```


BRCC – Branch if Carry Cleared

Description:

Conditional relative branch. Tests the Carry Flag (C) and branches relatively to PC if C is cleared. This instruction branches relatively to PC in either direction ($PC - 64 \leq \text{destination} \leq PC + 64$). The parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBC 0,k).

Operation:

- (i) If $C = 0$ then $PC \leftarrow PC + k + 1$, else $PC \leftarrow PC + 1$

Syntax:

- (i) BRCC k

Operands:

$$-64 \leq k \leq +63$$

Program Counter:

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

$$PC \leftarrow PC + 1, \text{ if condition is false}$$

16-bit Opcode:

1111	01kk	kkkk	k000
------	------	------	------

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
add    r22,r23    ; Add r23 to r22
brcc   nocarry    ; Branch if carry cleared
...
nocarry: nop      ; Branch destination (do nothing)
```

5.

- Analyze the program below and explain how the program works and what it will do. Input data in register R20. Explain what will happen with this data and where to find the result.
- Make a flowchart of the program that explains the function.

[illegible]

```
subr_01:
```

```
push counter      ; Store registers on stack, counter and temp
push temp
clr counter       ; counter = 0000 0000
andi r20, 0x7F    ; mask r20 with 0111 1111
mov temp, r20     ; result in r20, copy to temp
```

shift:

```
lsr temp           ; shift r20 right 1 bit, bit 0 -> Carry flag
brcc next          ; check if Carry flag = 0, if = 0 jump to next
inc counter        ; if Carry flag = 1, increase counter
```

next:

```

cpi temp, 0          ; check if temp = 0
brne shift          ; temp not 0, jump to shift

```

lsl counter	; temp = 0, shift counter 1 step right, bit 0 to carry
brcc end	; if Carry flag = 0, jump to end
ori r20, 0x80	; if carry flag =1, set bit 7 i r20

end:

```
pop temp           ; Reset registers from stack, counter and temp
pop counter
ret                ; Return value in Register r20
```

5.

- Analyze the program below and explain how the program works and what it will do. Input data in register R20. Explain what will happen with this data and where to find the result.
- Make a flowchart of the program that explains the function.

[illegible]

subr_01:

```
push counter
push temp
clr counter
andi r20, 0x7F
mov temp, r20
```

shift:

```
lsr temp
brcc next
inc counter
```

```
next:
```

```

    cpi temp, 0
    brne shift

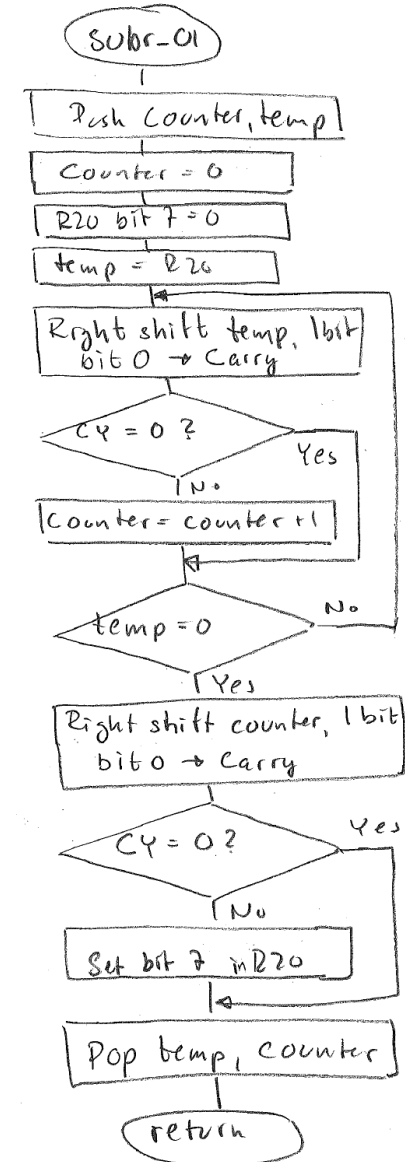
```

```
lsr counter
brcc end
ori r20, 0x80
```

end:

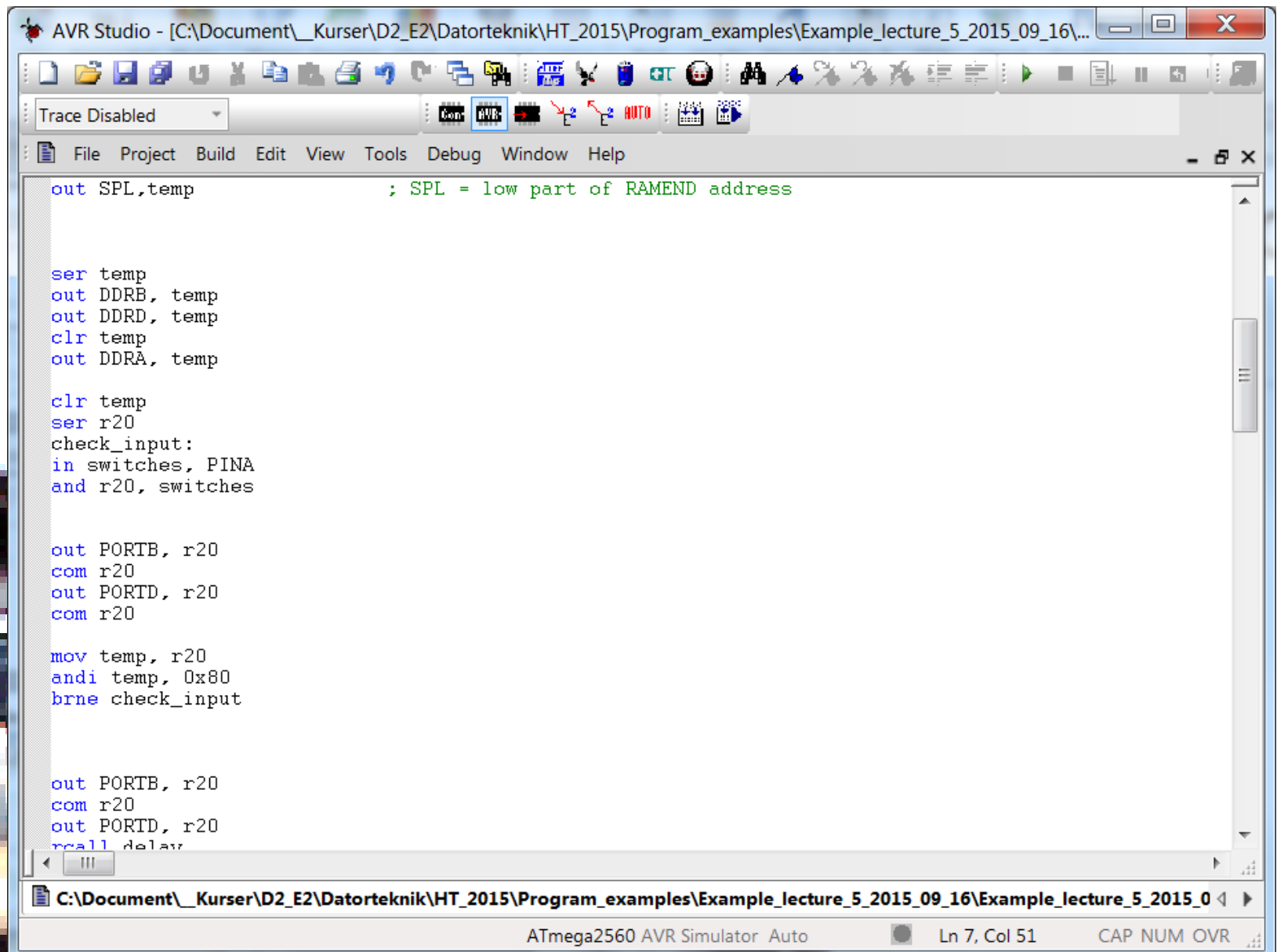
```
pop temp
pop counter
ret
```

The program counts the number of ones in r20, bit 0 -6. If the number is odd, the bit 7 in r20 will be set to one, otherwise bit 7 will remain zero. The program is thus checking parity and sets the parity bit to Even parity.



[illegible]

Testprogram



The screenshot shows the AVR Studio IDE with a file named 'Example_lecture_5_2015_09_16_0'. The assembly code is as follows:

```
out SPL,temp                ; SPL = low part of RAMEND address

ser temp
out DDRB, temp
out DDRD, temp
clr temp
out DDRA, temp

clr temp
ser r20
check_input:
in switches, PINA
and r20, switches

out PORTB, r20
com r20
out PORTD, r20
com r20

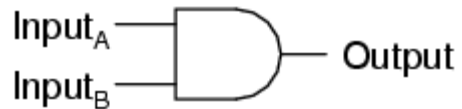
mov temp, r20
andi temp, 0x80
brne check_input

out PORTB, r20
com r20
out PORTD, r20
rcall delay
```

The status bar at the bottom indicates 'ATmega2560 AVR Simulator Auto', 'Ln 7, Col 51', and 'CAP NUM OVR'.

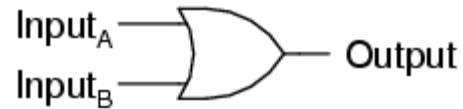
AND, OR, XOR

2-input AND gate



A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

OR gate



A	B	Output
0	0	0
0	1	1
1	0	1
1	1	1

Exclusive-OR gate



A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

AND – Logical AND

Description:

Performs the logical AND between the contents of register Rd and register Rr and places the result in the destination register Rd.

Operation:

(i) $Rd \leftarrow Rd \bullet Rr$

Syntax:

(i) AND Rd,Rr

Operands:

$0 \leq d \leq 31, 0 \leq r \leq 31$

Program Counter:

$PC \leftarrow PC + 1$

16-bit Opcode:

0010	00rd	dddd	rrrr
------	------	------	------

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	\Leftrightarrow	0	\Leftrightarrow	\Leftrightarrow	–

S: $N \oplus V$, For signed tests.

V: 0
Cleared

N: R7
Set if MSB of the result is set; cleared otherwise.

Z: $R7 \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result is \$00; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```
and  r2,r3      ; Bitwise and r2 and r3, result in r2
ldi  r16,1      ; Set bitmask 0000 0001 in r16
and  r2,r16     ; Isolate bit 0 in r2
```

Words: 1 (2 bytes)

Cycles: 1

OR – Logical OR

Description:

Performs the logical OR between the contents of register Rd and register Rr and places the result in the destination register Rd.

Operation:

(i) $Rd \leftarrow Rd \vee Rr$

Syntax:

(i) OR Rd,Rr

Operands:

$0 \leq d \leq 31, 0 \leq r \leq 31$

Program Counter:

$PC \leftarrow PC + 1$

16-bit Opcode:

0010	10rd	dddd	rrrr
------	------	------	------

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	\Leftrightarrow	0	\Leftrightarrow	\Leftrightarrow	–

S: $N \oplus V$, For signed tests.

V: 0
Cleared

N: R7
Set if MSB of the result is set; cleared otherwise.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result is \$00; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```
or      r15,r16    ; Do bitwise or between registers
bst     r15,6      ; Store bit 6 of r15 in T Flag
brts    ok         ; Branch if T Flag set
...
ok:     nop        ; Branch destination (do nothing)
```

Words: 1 (2 bytes)

Cycles: 1

EOR – Exclusive OR

Description:

Performs the logical EOR between the contents of register Rd and register Rr and places the result in the destination register Rd.

Operation:

(i) $Rd \leftarrow Rd \oplus Rr$

Syntax:

(i) EOR Rd,Rr

Operands:

$$0 \leq d \leq 31, 0 \leq r \leq 31$$

Program Counter:

$$PC \leftarrow PC + 1$$

16-bit Opcode:

0010	01rd	dddd	rrrr
------	------	------	------

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	\Leftrightarrow	0	\Leftrightarrow	\Leftrightarrow	–

S: $N \oplus V$, For signed tests.

V: 0
Cleared

N: R7
Set if MSB of the result is set; cleared otherwise.

Z: $\overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0}$
Set if the result is \$00; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```
eor    r4,r4    ; Clear r4
eor    r0,r22   ; Bitwise exclusive or between r0 and r22
```

Words: 1 (2 bytes)

Cycles: 1

16-bit Opcode, "hex code"

LDI – Load Immediate

Description:

Loads an 8 bit constant directly to register 16 to 31.

Operation:

(i) $Rd \leftarrow K$

Syntax:

(i) LDI Rd,K

Operands:

$16 \leq d \leq 31, 0 \leq K \leq 255$

Program Counter:

$PC \leftarrow PC + 1$

16-bit Opcode:

1110	KKKK	dddd	KKKK
------	------	------	------

Example: ldi r17, 0x0A,
d = 1, (offset from r16)
K = 0x0A, 0b0000 1010
Opcode: E01A

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
clr    r31      ; Clear Z high byte
ldi    r30,$F0  ; Set Z low byte to $F0
lpm                      ; Load constant from Program
                        ; memory pointed to by Z
```

Words: 1 (2 bytes)

Cycles: 1

16-bit Opcode, "hex code"

SER – Set all Bits in Register

Description:

Loads \$FF directly to register Rd.

Operation:

(i) $Rd \leftarrow \$FF$

Syntax:

(i) SER Rd

Operands:

$16 \leq d \leq 31$

Program Counter:

$PC \leftarrow PC + 1$

16-bit Opcode:

1110	1111	dddd	1111
------	------	------	------

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
clr r16      ; Clear r16
ser r17      ; Set r17
out $18,r16   ; Write zeros to Port B
nop          ; Delay (do nothing)
out $18,r17   ; Write ones to Port B
```

Words: 1 (2 bytes)

Cycles: 1

Example: ser r17,
d = 1, (offset from r16)
Opcode: EF1F

Same Opcode!

Compare with:
ldi r17, 0xFF
d = 1, (offset from r16)
K = 0xFF, 0b1111 1111
Opcode: EF1F

Intel HEX

From Wikipedia, the free encyclopedia

Intel HEX is a [file format](#) that conveys binary information in [ASCII](#) text form. It is commonly used for programming [microcontrollers](#), [EPROMs](#), and other types of programmable logic devices. In a typical application, a [compiler](#) or [assembler](#) converts a [program's source code](#) (such as in [C](#) or [assembly language](#)) to [machine code](#) and outputs it into a HEX file. The HEX file is then imported by a [programmer](#) to "burn" the machine code into a [ROM](#), or is transferred to the target system for loading and execution.^[1]

Contents [\[hide\]](#)

- 1 [Format](#)
 - 1.1 [Record structure](#)
 - 1.1.1 [Color legend](#)
 - 1.1.2 [Checksum calculation](#)
 - 1.2 [Text line terminators](#)
 - 1.3 [Record types](#)
 - 1.4 [Named formats](#)
- 2 [File example](#)
- 3 [See also](#)
- 4 [References](#)
- 5 [External links](#)

Format [\[edit \]](#)

Intel HEX consists of lines of [ASCII](#) text that are separated by [line feed](#) or [carriage return](#) characters or both. Each text line contains [hexadecimal](#) characters that [encode](#) multiple binary numbers. The binary numbers may represent data, [memory addresses](#), or other values, depending on their position in the line and the type and length of the line. Each text line is

Intel HEX consists of lines of [ASCII](#) text that are separated by [line feed](#) or [carriage return](#) characters or both. Each text line contains [hexadecimal](#) characters that [encode](#) multiple binary numbers. The binary numbers may represent data, [memory addresses](#), or other values, depending on their position in the line and the type and length of the line. Each text line is called a *record*.







Record structure [\[edit \]](#)

A [record](#) (line of text) consists of six [fields](#) (parts) that appear in order from left to right:

1. **Start code**, one character, an ASCII colon ':'.
2. **Byte count**, two hex digits, indicating the number of bytes (hex digit pairs) in the data field. The maximum byte count is 255 (0xFF). 16 (0x10) and 32 (0x20) are commonly used byte counts.
3. **Address**, four hex digits, representing the 16-bit beginning memory address offset of the data. The physical address of the data is computed by adding this offset to a previously established base address, thus allowing memory addressing beyond the 64 kilobyte limit of 16-bit addresses. The base address, which defaults to zero, can be changed by various types of records. Base addresses and address offsets are always expressed as [big endian](#) values.
4. **Record type** (see [record types](#) below), two hex digits, 00 to 05, defining the meaning of the data field.
5. **Data**, a sequence of n bytes of data, represented by $2n$ hex digits. Some records omit this field (n equals zero). The meaning and interpretation of data bytes depends on the application.
6. **Checksum**, two hex digits, a computed value that can be used to verify the record has no errors.

Color legend [\[edit \]](#)

As a visual aid, the fields of Intel HEX records are colored throughout this article as follows:

 Start code  Byte count  Address  Record type  Data  Checksum

Checksum calculation [\[edit \]](#)

A record's checksum byte is the [two's complement](#) (negative) of the [least significant byte](#) (LSB) of the sum of all decoded byte values in the record preceding the checksum. It is computed by summing the decoded byte values and extracting the LSB of the sum (*i.e.*, the data checksum), and then calculating the two's complement of the LSB (*e.g.*, by [inverting](#) its bits and adding one).

For example, in the case of the record `:0300300002337A1E`, the sum of the decoded byte values is $03 + 00 + 30 + 00 + 02 + 33 + 7A = E2$. The two's complement of `E2` is `1E`, which is the checksum byte appearing at the end of the record.

Hexcode, example 1

```
.include "m2560def.inc"
```

```
loop:
```

```
ldi r16, 0xaa ; load 0xAA to register r16
```

Hexcode:

```
:020000020000FC  
:020000000AEA0A  
:00000001FF
```

Checksum:

$02 + 00 + 00 + 00 + 0A + EA = 0xF6 \text{ \%FF} = 0xF6 = 0b\ 1111\ 0110.$

$1\text{-compl} = 0000\ 1001. 2\text{-compl} = 0000\ 1010 = 0x0A$

Hexcode, example 2

```
.include "m2560def.inc"
```

```
loop:
```

```
ldi r16, 0x0F           ; 0x0F to r16
out DDRB, r16           ; Set Data Direction Registers
ldi r17, 0b00110011     ; 0x33 to r17
out PORTB, r17          ; set outputs, port B
```

Hexcode:

```
:0200000020000FC
:0800000000FE004B913E315B988
:000000001FF
```

Checksum:

$08+00+00+00+0F+E0+04+B9+13+E3+15+B9=0x378 \%FF = 0x78 =$
 $0b\ 0111\ 1000. 1-compl = 1000\ 0111. 2-compl = 1000\ 1000 = 0x88$