Linnæus University
Sweden

Instructions

Exam in Computer Technology, 1DT301. 2019-10-28.

Time: 08.00 - 13.00
Allowed tools: Only language dictionaries.

Total score: 50 credits. Pass: 20 credits. Grade 4: 30 credits. Grade 5: 40 credits.
Grade **F**: 0–19, Grade **E**: 20–25, Grade **D**: 26–32, Grade **C**: 33–38, Grade **B**: 39–44: Grade **A**: 45-50.

Write clearly and legibly! Answer in Swedish or English!

CPU: ATMEL AVR ATMega2560 in all tasks, unless otherwise indicated.

1.

a) Convert these numbers to decimal numbers:

1111_2 (2-complement form, 4-bit number, signed)

Answer: $-2^3 + 2^2 + 2^1 + 2^0 = -8 + 4 + 2 + 1 = -1$

1111_2 (unsigned number)

Answer: $2^3 + 2^2 + 2^1 + 2^0 = 8 + 4 + 2 + 1 = 15$

1111_8 (octal number)

Answer: $8^3 + 8^2 + 8^1 + 8^0 = 512 + 64 + 8 + 1 = 585$

1111_{16} (hexadecimal number) **4p**

Answer: $16^3 + 16^2 + 16^1 + 16^0 = 4096 + 256 + 16 + 1 = 4369$

b) Convert the decimal number **-3823_{10}** to a binary number in two's complement form, with **13 bits**. After that, convert the binary number to a **hexadecimal value**. Give the answer in **hexadecimal**! Show your calculations. **2p**

Answer: +3823 = 1110 1110 1111.
With 13 bits: 0 1110 1110 1111
Ones-complement, 13 bits: 1 0001 0001 0000.
Add 1: 1 0001 0001 0000 + 1 = 1 0001 0001 0001 = 1111_{16}

Division:	Quote:	Rest:	
3823/2	1911	1	lsb
1911/2	955	1	
955/2	477	1	
477/2	238	1	
238/2	119	0	
119/2	59	1	
59/2	29	1	
29/2	14	1	
14/2	7	0	
7/2	3	1	
3/2	1	1	
1/2	0	1	msb

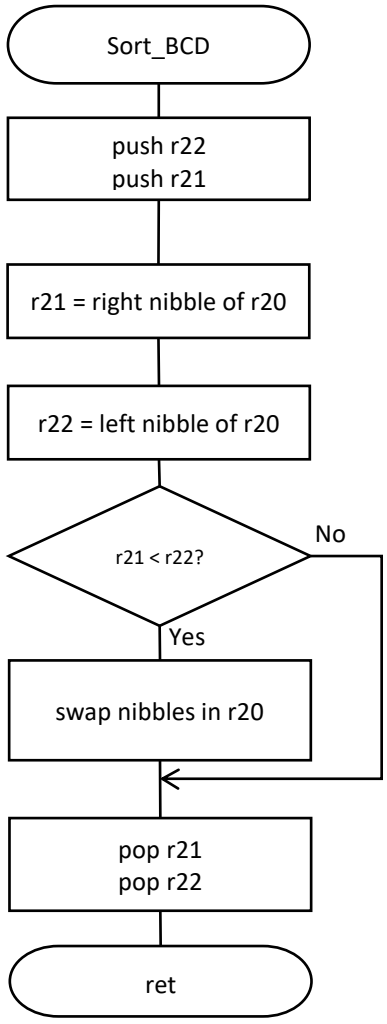
2. Machine code

Below is part of a program, cut out from the disassembler. The machine code is printed with the most significant byte first, ie as it is described in the Instruction Set Manual.
Example: For the instruction **ret**, machine code is **9508**.
On lines B –11 the assembler code is removed. Branch or rjump-instructions should be given with an offset to PC, example: BRNE PC – 0x06.

a) Recreate the assembler code by interpreting machine code. Use the enclosed examples from the Instruction Set Manual. **4p**

b) Make a flowchart of the complete program and explain the function of the program. **4p**

```
@00000003: Sort_BCD
11:        push left                    ;save registers on stac
+00000003:    936F         PUSH       R22             Push regis
12:        push right
+00000004:    935F         PUSH       R21             Push regis
13:        rjmp bcd
+00000005:    C004         RJMP       PC+0x0005       Relative j
@00000006: swap_n
15:        swap r20
+00000006:    9542         SWAP       R20             Swap nibbl
@00000007: return
18:        pop right                     ;restore used registers
+00000007:    915F         POP        R21             Pop regist
19:        pop left
+00000008:    916F         POP        R22             Pop regist
20:        ret                                ;return from subrou
+00000009:    9508         RET                        Subroutine
@0000000A: bcd
23:        mov right, r20              ;r21 = right BCD number
+0000000A:    2F54         MOV        R21,R20         Copy regis
24:        andi right, 0x0F
+0000000B:    705F          ANDI       R21,0x0F        Logical A
26:        mov left, r20               ;r22 = left BCD number
+0000000C:    2F64         MOV        R22,R20         Copy regis
27:        andi left, 0xF0
+0000000D:    7F60         ANDI       R22,0xF0        Logical AN
28:        swap left                     ;move number to right n
+0000000E:    9562         SWAP       R22             Swap nibbl
30:        cp right, left                ;compare BCD numbers
+0000000F:    1756         CP         R21,R22         Compare
31:        brlt swap_n                 ;jump if RN is less than L
+00000010:    F3AC         BRLT       PC-0x0A         Branch if
32:        rjmp return
+00000011:    CFF5         RJMP       PC-0x000A       Relative j
```



Answer: A subroutine that compare left and right nibble of r20. (A nibble is 4 bits of a byte)
The two nibbles are compared and sorted so that the highest number will be in the right nibble.

3. Assembler programming.

Below is an assembler program for a subroutine. The subroutine is called from a timer interrupt.

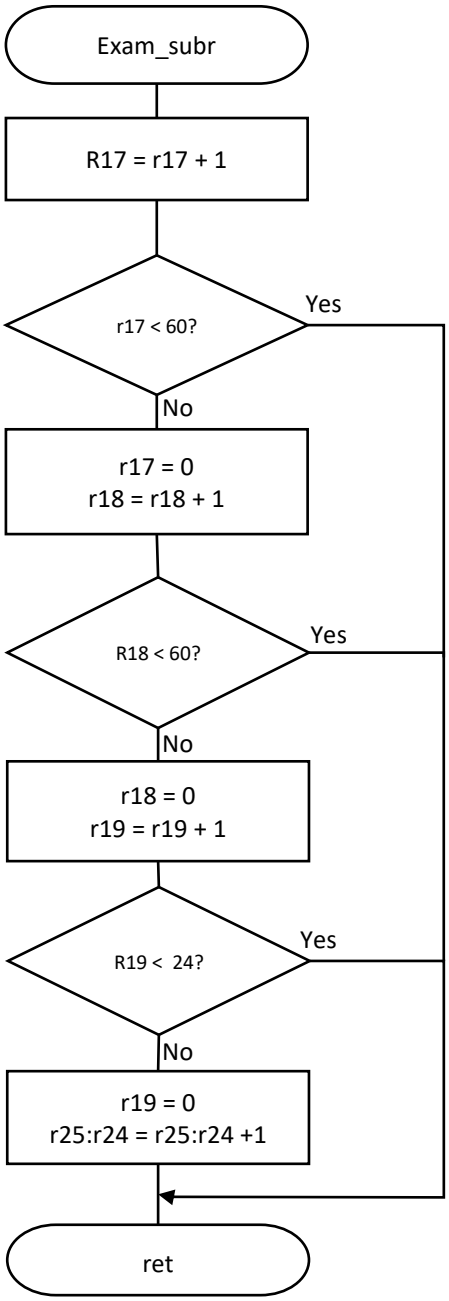
a) Make a flowchart of the assembler program below. **4p**

b) Analyze the program and explain what the program will do, the function. Which registers will be used as input/outputs and what do you think they will contain? **4p**

Answer: A watch. Timer interrupt each second. r17 contains number of seconds, r18 number of minutes, r19 number of hours and r25:r24 number of days.

```
exam_subr:
    inc r17
    cpi r17,0x3c
    brlo exam
    clr r17
    inc r18
    cpi r18, 0b00111100
    brlt exam
    eor r18,r18
    inc r19
    cpi r19, 0x18
    brlt exam
    andi r19, 0x00
    adiw r24,1
exam:
    ret
```

should be:
reti



4. Interrupts.

Se text from manual doc2549_ATmega2560 below.

a) Write 2 lines of assembler code to set the flags in register EICRA to get this function:
Interrupt when pin 3 on port D goes from low to high.
Interrupt when pin 2 on port D goes from high to low.
Interrupt when pin 1 on port D goes from high to low or from low to high.
Interrupt when pin 0 on port D goes has low level.
The register EICRA is placed on address 0x69. **6p**

Answer:
ldi r16, 0xE4
sts EICRA, r16

```
ldi r16, 0b11100100        ; ISC31=1, ISC31=1, ISC21=1, ISC21=0
sts EICRA, r16             ; ISC11=0, ISC11=1, ISC01=0, ISC01=0
```

b) Which other flags have to be set to get the external interrupts in a) working? **2p**

Answer: Interrupt flags in register EIMSK, INT3, INT2, INT1 & INT0 and Global Interrupt Enable Flag in SREG.
ldi r16, 0x0F
out EIMSK, r16
sei

```
ldi r16, 0b00001111        ; set INT3, INT2, INT1 &
out EIMSK, r16             ; in register EIMSK
sei                        ; set global interrupt e
```

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 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	
(0x69)	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.

Alternate Functions of Port D

The Port D pins with alternate functions are shown in Table 13-12.

Table 13-12. Port D Pins Alternate Functions

Port Pin	Alternate Function
PD7	T0 (Timer/Counter0 Clock Input)
PD6	T1 (Timer/Counter1 Clock Input)
PD5	XCK1 (USART1 External Clock Input/Output)
PD4	ICP1 (Timer/Counter1 Input Capture Trigger)
PD3	INT3/TXD1 (External Interrupt3 Input or USART1 Transmit Pin)
PD2	INT2/RXD1 (External Interrupt2 Input or USART1 Receive Pin)
PD1	INT1/SDA (External Interrupt1 Input or TWI Serial DAta)
PD0	INT0/SCL (External Interrupt0 Input or TWI Serial CLock)

Table 15-1. Interrupt Sense Control[1]

ISCn1	ISCn0	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, 1or 0.
When changing the ISCn1/ISCn0 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

5. Assembler program.

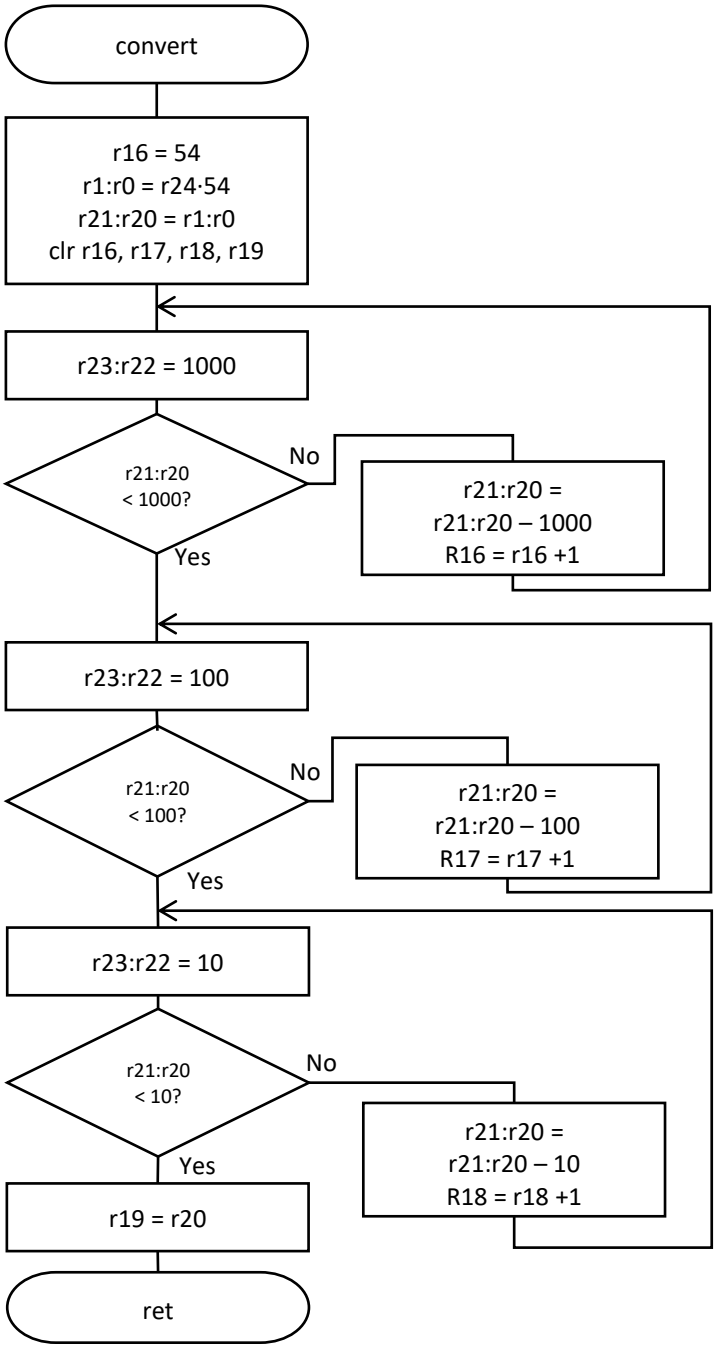
Analyze the program below. It is a subroutine that will do some kind of conversion. Try to figure out what! The input value to the subroutine is in register **r24**. Outputs in register **r16, r17, r18** and **r19**.

a) Make a flowchart of the program. **4p**

b) Explain what the program is doing and which values will be in the output registers. **4p**

```
convert:
            ldi     r16, 54
            mul     r24, r16
            mov     r21, r1
            mov     r20, r0
            clr     r16
            clr     r17
            clr     r18
            clr     r19

_m:         ldi     r22, low(1000)
            ldi     r23, high(1000)
            cp      r20, r22
            cpc     r21, r23
            brlt    _c
            subi    r20, low(1000)
            sbci    r21, high(1000)
            inc     r16
            rjmp _m
_c:
            ldi     r22, low(100)
            ldi     r23, high(100)
            cp      r20, r22
            cpc     r21, r23
            brlt    _d
            subi    r20, low(100)
            sbci    r21, high(100)
            inc     r17
            rjmp _c
_d:
            ldi     r22, low(10)
            ldi     r23, high(10)
            cp      r20, r22
            cpc     r21, r23
            brlt    _n
            subi    r20, low(10)
            sbci    r21, high(10)
            inc     r18
            rjmp    _d
_n:
            mov     r19, r20
            ret
```



Answer: Multiply r24 with 0,54 which is equal division with 1,852. r16 and r17 integer part, r18, r19 fractional part. Conversion from km/h to NM, Nautic Mile.
Ex: 49 km/h = 26,46 NM. R16=2, r17=6, r18= 4, r19= 6.
Distance from the equator to North Pole: 10'000 km = 90x60 NM = 5400 NM. Thus 1 km = 0.54 NM.
1 NM = 10'000/5400 km = 1,852 km.

C programming.

6. **(4p)**

a.

Assuming the following integer array

arr:	7	9	33	77	8

and the statement

```
arr[X] = 99;
```

What should X be set to if You want to replace 77 with 99?

b.

Turn the following sentence into a C statement:

A char pointer references memory location 332.

c.

Describe shortly how the union statement works.

d.

Assuming

```
int i = 5;
```

Write a C statement that shifts i left by two positions.

7. **(4p)**

Given the following assembly program

```
    ldi r16, 5
    ldi r17, 0
myLabel:
    add r17, r16
    dec r16
    brne myLabel
```

Create an equivalent C program.

Note! A main() is not required, only its body.

Note! You are not allowed to use labels (goto).

8. **(2p)**

Create a function having an unsigned char as a parameter and returning 1 if bits 2 and 5 are set otherwise 0.
Note! Use the return statement.

9. **(2p)**

Assuming the following

```
int i = 5;
int k = 8;
int *p1 = &i;
int *p2 = p1;

/* A */

i = 55;
*p1 = 88;

/* B */

*p1 = 444;
*p2 = 555;

/* C */
```

What different values does i and k have when A, B and C is reached?

6a.
3
6b.
char *p = 332;
6c.
See C lecture 4 or the *The C Book* chapter 6.3.
6d.
i = i << 2;

7.

```
void main()

{

int i = 5;       // r16

int sum = 0;  // r17

        while( i >= 0)

        {

                sum = sum + i;

                i = i – 1;

        }

}
```

8.
A verbose version:

```
unsigned char test(unsigned char b)

{

unsigned char pattern = 0b100100; // Bits 2 and 5 are set.

unsigned char is_set = 0; // Assume bits not set.


        if( (b & pattern) == pattern)

        {

                is_set = 1;

        }

        return is_set;

}
```

And a terse version:

```
unsigned char test1(unsigned char b)

{

        return (b & 0b100100) == 0b100100;

}
```

9.
A: i = 5, k = 8
B: i = 88, k = 8
C: i = 555, k = 8

From m2560def:

```
; ***** INTERRUPT VECTORS ********************************************************
.equ    INT0addr        = 0x0002        ; External Interrupt Request 0
.equ    INT1addr        = 0x0004        ; External Interrupt Request 1
.equ    INT2addr        = 0x0006        ; External Interrupt Request 2
.equ    INT3addr        = 0x0008        ; External Interrupt Request 3
.equ    INT4addr        = 0x000a        ; External Interrupt Request 4
```

```
.equ    PCMSK1  = 0x6c  ; MEMORY MAPPED
.equ    PCMSK0  = 0x6b  ; MEMORY MAPPED
.equ    EICRB   = 0x6a  ; MEMORY MAPPED
.equ    EICRA   = 0x69  ; MEMORY MAPPED
.equ    PCICR   = 0x68  ; MEMORY MAPPED
.equ    OSCCAL  = 0x66  ; MEMORY MAPPED
.equ    PRR1    = 0x65  ; MEMORY MAPPED
.equ    PRR0    = 0x64  ; MEMORY MAPPED
.equ    CLKPR   = 0x61  ; MEMORY MAPPED
.equ    WDTCSR  = 0x60  ; MEMORY MAPPED
```

```
.equ    EEDR    = 0x20
.equ    EECR    = 0x1f
.equ    GPIOR0  = 0x1e
.equ    EIMSK   = 0x1d
.equ    EIFR    = 0x1c
.equ    PCIFR   = 0x1b
.equ    TIFR5   = 0x1a
```

```
.equ    PORTE   = 0x0e
.equ    DDRE    = 0x0d
.equ    PINE    = 0x0c
.equ    PORTD   = 0x0b
.equ    DDRD    = 0x0a
.equ    PIND    = 0x09
.equ    PORTC   = 0x08
.equ    DDRC    = 0x07
.equ    PINC    = 0x06
.equ    PORTB   = 0x05
.equ    DDRB    = 0x04
.equ    PINB    = 0x03
.equ    PORTA   = 0x02
.equ    DDRA    = 0x01
.equ    PINA    = 0x00
```

Instruction Set Summary

Mnemonics	Operands	Description	Operation	Flags	#Clocks
ARITHMETIC AND LOGIC INSTRUCTIONS					
ADD	Rd, Rr	Add two Registers	Rd ← Rd + Rr	Z,C,N,V,H	1
ADC	Rd, Rr	Add with Carry two Registers	Rd ← Rd + Rr + C	Z,C,N,V,H	1
ADIW	Rdl,K	Add Immediate to Word	Rdh:Rdl ← Rdh:Rdl + K	Z,C,N,V,S	2
SUB	Rd, Rr	Subtract two Registers	Rd ← Rd - Rr	Z,C,N,V,H	1
SUBI	Rd, K	Subtract Constant from Register	Rd ← Rd - K	Z,C,N,V,H	1
SBC	Rd, Rr	Subtract with Carry two Registers	Rd ← Rd - Rr - C	Z,C,N,V,H	1
SBCI	Rd, K	Subtract with Carry Constant from Reg.	Rd ← Rd - K - C	Z,C,N,V,H	1
SBIW	Rdl,K	Subtract Immediate from Word	Rdh:Rdl ← Rdh:Rdl - K	Z,C,N,V,S	2
AND	Rd, Rr	Logical AND Registers	Rd ← Rd • Rr	Z,N,V	1
ANDI	Rd, K	Logical AND Register and Constant	Rd ← Rd • K	Z,N,V	1
OR	Rd, Rr	Logical OR Registers	Rd ← Rd v Rr	Z,N,V	1
ORI	Rd, K	Logical OR Register and Constant	Rd ← Rd v K	Z,N,V	1
EOR	Rd, Rr	Exclusive OR Registers	Rd ← Rd ⊕ Rr	Z,N,V	1
COM	Rd	One's Complement	Rd ← $FF – Rd	Z,C,N,V	1
NEG	Rd	Two's Complement	Rd ← $00 – Rd	Z,C,N,V,H	1
SBR	Rd,K	Set Bit(s) in Register	Rd ← Rd v K	Z,N,V	1
CBR	Rd,K	Clear Bit(s) in Register	Rd ← Rd • ($FF - K)	Z,N,V	1
INC	Rd	Increment	Rd ← Rd + 1	Z,N,V	1
DEC	Rd	Decrement	Rd ← Rd – 1	Z,N,V	1
TST	Rd	Test for Zero or Minus	Rd ← Rd • Rd	Z,N,V	1
CLR	Rd	Clear Register	Rd ← Rd ⊕ Rd	Z,N,V	1
SER	Rd	Set Register	Rd ← $FF	None	1
MUL	Rd, Rr	Multiply Unsigned	R1:R0 ← Rd x Rr	Z,C	2
MULS	Rd, Rr	Multiply Signed	R1:R0 ← Rd x Rr	Z,C	2
MULSU	Rd, Rr	Multiply Signed with Unsigned	R1:R0 ← Rd x Rr	Z,C	2
FMUL	Rd, Rr	Fractional Multiply Unsigned	R1:R0 ← (Rd x Rr) << 1	Z,C	2
FMULS	Rd, Rr	Fractional Multiply Signed	R1:R0 ← (Rd x Rr) << 1	Z,C	2
FMULSU	Rd, Rr	Fractional Multiply Signed with Unsigned	R1:R0 ← (Rd x Rr) << 1	Z,C	2
BRANCH INSTRUCTIONS					
RJMP	k	Relative Jump	PC ← PC + k + 1	None	2
IJMP		Indirect Jump to (Z)	PC ← Z	None	2
JMP	k	Direct Jump	PC ← k	None	3
RCALL	k	Relative Subroutine Call	PC ← PC + k + 1	None	3
ICALL		Indirect Call to (Z)	PC ← Z	None	3
CALL	k	Direct Subroutine Call	PC ← k	None	4
RET		Subroutine Return	PC ← STACK	None	4
RETI		Interrupt Return	PC ← STACK	I	4
CPSE	Rd,Rr	Compare, Skip if Equal	if (Rd = Rr) PC ← PC + 2 or 3	None	1 / 2 / 3
CP	Rd,Rr	Compare	Rd – Rr	Z, N,V,C,H	1
CPC	Rd,Rr	Compare with Carry	Rd – Rr – C	Z, N,V,C,H	1
CPI	Rd,K	Compare Register with Immediate	Rd – K	Z, N,V,C,H	1
SBRC	Rr, b	Skip if Bit in Register Cleared	if (Rr(b)=0) PC ← PC + 2 or 3	None	1 / 2 / 3
SBRS	Rr, b	Skip if Bit in Register is Set	if (Rr(b)=1) PC ← PC + 2 or 3	None	1 / 2 / 3
SBIC	P, b	Skip if Bit in I/O Register Cleared	if (P(b)=0) PC ← PC + 2 or 3	None	1 / 2 / 3
SBIS	P, b	Skip if Bit in I/O Register is Set	if (P(b)=1) PC ← PC + 2 or 3	None	1 / 2 / 3
BRBS	s, k	Branch if Status Flag Set	if (SREG(s) = 1) then PC←PC+k + 1	None	1 / 2
BRBC	s, k	Branch if Status Flag Cleared	if (SREG(s) = 0) then PC←PC+k + 1	None	1 / 2
BREQ	k	Branch if Equal	if (Z = 1) then PC ← PC + k + 1	None	1 / 2
BRNE	k	Branch if Not Equal	if (Z = 0) then PC ← PC + k + 1	None	1 / 2
BRCS	k	Branch if Carry Set	if (C = 1) then PC ← PC + k + 1	None	1 / 2
BRCC	k	Branch if Carry Cleared	if (C = 0) then PC ← PC + k + 1	None	1 / 2
BRSH	k	Branch if Same or Higher	if (C = 0) then PC ← PC + k + 1	None	1 / 2
BRLO	k	Branch if Lower	if (C = 1) then PC ← PC + k + 1	None	1 / 2
BRMI	k	Branch if Minus	if (N = 1) then PC ← PC + k + 1	None	1 / 2
BRPL	k	Branch if Plus	if (N = 0) then PC ← PC + k + 1	None	1 / 2
BRGE	k	Branch if Greater or Equal, Signed	if (N ⊕ V= 0) then PC ← PC + k + 1	None	1 / 2
BRLT	k	Branch if Less Than Zero, Signed	if (N ⊕ V= 1) then PC ← PC + k + 1	None	1 / 2
BRHS	k	Branch if Half Carry Flag Set	if (H = 1) then PC ← PC + k + 1	None	1 / 2
BRHC	k	Branch if Half Carry Flag Cleared	if (H = 0) then PC ← PC + k + 1	None	1 / 2
BRTS	k	Branch if T Flag Set	if (T = 1) then PC ← PC + k + 1	None	1 / 2
BRTC	k	Branch if T Flag Cleared	if (T = 0) then PC ← PC + k + 1	None	1 / 2
BRVS	k	Branch if Overflow Flag is Set	if (V = 1) then PC ← PC + k + 1	None	1 / 2
BRVC	k	Branch if Overflow Flag is Cleared	if (V = 0) then PC ← PC + k + 1	None	1 / 2

Mnemonics	Operands	Description	Operation	Flags	#Clocks
BRIE	k	Branch if Interrupt Enabled	if (I = 1) then PC ← PC + k + 1	None	1 / 2
BRID	k	Branch if Interrupt Disabled	if (I = 0) then PC ← PC + k + 1	None	1 / 2
DATA TRANSFER INSTRUCTIONS					
MOV	Rd, Rr	Move Between Registers	Rd ← Rr	None	1
MOVW	Rd, Rr	Copy Register Word	Rd+1:Rd ← Rr+1:Rr	None	1
LDI	Rd, K	Load Immediate	Rd ← K	None	1
LD	Rd, X	Load Indirect	Rd ← (X)	None	2
LD	Rd, X+	Load Indirect and Post-Inc.	Rd ← (X), X ← X + 1	None	2
LD	Rd, - X	Load Indirect and Pre-Dec.	X ← X - 1, Rd ← (X)	None	2
LD	Rd, Y	Load Indirect	Rd ← (Y)	None	2
LD	Rd, Y+	Load Indirect and Post-Inc.	Rd ← (Y), Y ← Y + 1	None	2
LD	Rd, - Y	Load Indirect and Pre-Dec.	Y ← Y - 1, Rd ← (Y)	None	2
LDD	Rd,Y+q	Load Indirect with Displacement	Rd ← (Y + q)	None	2
LD	Rd, Z	Load Indirect	Rd ← (Z)	None	2
LD	Rd, Z+	Load Indirect and Post-Inc.	Rd ← (Z), Z ← Z+1	None	2
LD	Rd, -Z	Load Indirect and Pre-Dec.	Z ← Z - 1, Rd ← (Z)	None	2
LDD	Rd, Z+q	Load Indirect with Displacement	Rd ← (Z + q)	None	2
LDS	Rd, k	Load Direct from SRAM	Rd ← (k)	None	2
ST	X, Rr	Store Indirect	(X) ← Rr	None	2
ST	X+, Rr	Store Indirect and Post-Inc.	(X) ← Rr, X ← X + 1	None	2
ST	- X, Rr	Store Indirect and Pre-Dec.	X ← X - 1, (X) ← Rr	None	2
ST	Y, Rr	Store Indirect	(Y) ← Rr	None	2
ST	Y+, Rr	Store Indirect and Post-Inc.	(Y) ← Rr, Y ← Y + 1	None	2
ST	- Y, Rr	Store Indirect and Pre-Dec.	Y ← Y - 1, (Y) ← Rr	None	2
STD	Y+q,Rr	Store Indirect with Displacement	(Y + q) ← Rr	None	2
ST	Z, Rr	Store Indirect	(Z) ← Rr	None	2
ST	Z+, Rr	Store Indirect and Post-Inc.	(Z) ← Rr, Z ← Z + 1	None	2
ST	-Z, Rr	Store Indirect and Pre-Dec.	Z ← Z - 1, (Z) ← Rr	None	2
STD	Z+q,Rr	Store Indirect with Displacement	(Z + q) ← Rr	None	2
STS	k, Rr	Store Direct to SRAM	(k) ← Rr	None	2
LPM		Load Program Memory	R0 ← (Z)	None	3
LPM	Rd, Z	Load Program Memory	Rd ← (Z)	None	3
LPM	Rd, Z+	Load Program Memory and Post-Inc	Rd ← (Z), Z ← Z+1	None	3
SPM		Store Program Memory	(Z) ← R1:R0	None	-
IN	Rd, P	In Port	Rd ← P	None	1
OUT	P, Rr	Out Port	P ← Rr	None	1
PUSH	Rr	Push Register on Stack	STACK ← Rr	None	2
POP	Rd	Pop Register from Stack	Rd ← STACK	None	2
BIT AND BIT-TEST INSTRUCTIONS					
SBI	P,b	Set Bit in I/O Register	I/O(P,b) ← 1	None	2
CBI	P,b	Clear Bit in I/O Register	I/O(P,b) ← 0	None	2
LSL	Rd	Logical Shift Left	Rd(n+1) ← Rd(n), Rd(0) ← 0	Z,C,N,V	1
LSR	Rd	Logical Shift Right	Rd(n) ← Rd(n+1), Rd(7) ← 0	Z,C,N,V	1
ROL	Rd	Rotate Left Through Carry	Rd(0)←C,Rd(n+1)← Rd(n),C←Rd(7)	Z,C,N,V	1
ROR	Rd	Rotate Right Through Carry	Rd(7)←C,Rd(n)← Rd(n+1),C←Rd(0)	Z,C,N,V	1
ASR	Rd	Arithmetic Shift Right	Rd(n) ← Rd(n+1), n=0..6	Z,C,N,V	1
SWAP	Rd	Swap Nibbles	Rd(3..0)←Rd(7..4),Rd(7..4)←Rd(3..0)	None	1
BSET	s	Flag Set	SREG(s) ← 1	SREG(s)	1
BCLR	s	Flag Clear	SREG(s) ← 0	SREG(s)	1
BST	Rr, b	Bit Store from Register to T	T ← Rr(b)	T	1
BLD	Rd, b	Bit load from T to Register	Rd(b) ← T	None	1
SEC		Set Carry	C ← 1	C	1
CLC		Clear Carry	C ← 0	C	1
SEN		Set Negative Flag	N ← 1	N	1
CLN		Clear Negative Flag	N ← 0	N	1
SEZ		Set Zero Flag	Z ← 1	Z	1
CLZ		Clear Zero Flag	Z ← 0	Z	1
SEI		Global Interrupt Enable	I ← 1	I	1
CLI		Global Interrupt Disable	I ← 0	I	1
SES		Set Signed Test Flag	S ← 1	S	1
CLS		Clear Signed Test Flag	S ← 0	S	1
SEV		Set Twos Complement Overflow.	V ← 1	V	1
CLV		Clear Twos Complement Overflow	V ← 0	V	1
SET		Set T in SREG	T ← 1	T	1
CLT		Clear T in SREG	T ← 0	T	1
SEH		Set Half Carry Flag in SREG	H ← 1	H	1

Mnemonics	Operands	Description	Operation	Flags	#Clocks
CLH		Clear Half Carry Flag in SREG	H ← 0	H	1
MCU CONTROL INSTRUCTIONS					
NOP		No Operation		None	1
SLEEP		Sleep	(see specific descr. for Sleep function)	None	1
WDR		Watchdog Reset	(see specific descr. for WDR/timer)	None	1
BREAK		Break	For On-Chip Debug Only	None	N/A

BRLO – Branch if Lower (Unsigned)

Description:

Conditional relative branch. Tests the Carry Flag (C) and branches relatively to PC if C is set. 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 binary number represented in Rd was smaller than the unsigned binary number represented in Rr. This instruction branches relatively to PC in either direction (PC - 63 ≤ destination ≤ PC + 64). The parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBS 0,k).

Operation:

(i) If Rd < Rr (C = 1) then PC ← PC + k + 1, else PC ← PC + 1

	Syntax:	Operands:	Program Counter:
(i)	BRLO k	-64 ≤ k ≤ +63	PC ← PC + k + 1 PC ← PC + 1, if conditon is false

16-bit Opcode:

1111	00kk	kkkk	k000

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
        eor   r18,r18   ; Clear r18
loop:   inc   r18       ; Increase r18
        ...
        cpi   r18,$10   ; Compare r18 with $10
        brlo  loop      ; Branch if r18 < $10 (unsigned)
        nop             ; Exit from loop (do nothing)
```

Words: 1 (2 bytes)
Cycles: 1 if condition is false
2 if condition is true

RET – Return from Subroutine

Description:

Returns from subroutine. The return address is loaded from the STACK. The Stack Pointer uses a pre-increment scheme during RET.

Operation:

(i) PC(15:0) ← STACKDevices with 16 bits PC, 128K bytes Program memory maximum.
(ii) PC(21:0) ← STACKDevices with 22 bits PC, 8M bytes Program memory maximum.

	Syntax:	Operands:	Program Counter:	Stack:
(i)	RET	None	See Operation	SP←SP + 2, (2bytes,16 bits)
(ii)	RET	None	See Operation	SP←SP + 3, (3bytes,22 bits)

16-bit Opcode:

1001	0101	0000	1000

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
                call    routine   ; Call subroutine
                ...
routine:        push    r14       ; Save r14 on the Stack
                ...
                pop     r14       ; Restore r14
                ret               ; Return from subroutine
```

Words: 1 (2 bytes)
Cycles: 4 devices with 16-bit PC
5 devices with 22-bit PC

LDI – Load Immediate

Description:

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

Operation:

(i) Rd ← K

	Syntax:	Operands:	Program Counter:
(i)	LDI Rd,K	16 ≤ d ≤ 31, 0 ≤ K ≤ 255	PC ← PC + 1

16-bit Opcode:

1110	KKKK	dddd	KKKK

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

MOV – Copy Register

Description:

This instruction makes a copy of one register into another. The source register Rr is left unchanged, while the destination register Rd is loaded with a copy of Rr.

Operation:

(i) Rd ← Rr

	Syntax:	Operands:	Program Counter:
(i)	MOV Rd,Rr	0 ≤ d ≤ 31, 0 ≤ r ≤ 31	PC ← PC + 1

16-bit Opcode:

0010	11rd	dddd	rrrr

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
        mov   r16,r0    ; Copy r0 to r16
        call  check     ; Call subroutine
        ...
check:  cpi   r16,$11   ; Compare r16 to $11
        ...
        ret             ; Return from subroutine
```

Words: 1 (2 bytes)
Cycles: 1

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 ← PC + k + 1

	Syntax:	Operands:	Program Counter:	Stack
(i)	RJMP k	-2K ≤ k < 2K	PC ← PC + k + 1	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)
```

Words: 1 (2 bytes)
Cycles: 2

SWAP – Swap Nibbles

Description:

Swaps high and low nibbles in a register.

Operation:

(i) R(7:4) ← Rd(3:0), R(3:0) ← Rd(7:4)

	Syntax:	Operands:	Program Counter:
(i)	SWAP Rd	0 ≤ d ≤ 31	PC ← PC + 1

16-bit Opcode:

1001	010d	dddd	0010

Status Register and Boolean Formula:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

R (Result) equals Rd after the operation.

Example:

```
        inc   r1   ; Increment r1
        swap  r1   ; Swap high and low nibble of r1
        inc   r1   ; Increment high nibble of r1
        swap  r1   ; Swap back
```

bytes)

OUT – Store Register to I/O Location

Description:

Stores data from register Rr in the Register File to I/O Space (Ports, Timers, Configuration Registers etc.).

Operation:

(i) I/O(A) ← Rr

	Syntax:	Operands:	Program Counter:
(i)	OUT A,Rr	0 ≤ r ≤ 31, 0 ≤ A ≤ 63	PC ← PC + 1

16-bit Opcode:

1011	1AAr	rrrr	AAAA

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             ; Wait (do nothing)
        out   $18,r17   ; Write ones to Port B
```

Words: 1 (2 bytes)
Cycles: 1

STS – Store Direct to Data Space

Description:

Stores one byte from a Register to the data space. For parts with SRAM, the data space consists of the Register File, I/O memory and internal SRAM (and external SRAM if applicable). For parts without SRAM, the data space consists of the Register File only. The EEPROM has a separate address space.

A 16-bit address must be supplied. Memory access is limited to the current data segment of 64K bytes. The STS instruction uses the RAMPD Register to access memory above 64K bytes. To access another data segment in devices with more than 64K bytes data space, the RAMPD in register in the I/O area has to be changed.

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

Operation:

(i) (k) ← Rr

	Syntax:	Operands:	Program Counter:
(i)	STS k,Rr	0 ≤ r ≤ 31, 0 ≤ k ≤ 65535	PC ← PC + 2

32-bit Opcode:

1001	001d	dddd	0000
kkkk	kkkk	kkkk	kkkk

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
        lds   r2,$FF00  ; Load r2 with the contents of data space location $FF00
        add   r2,r1     ; add r1 to r2
        sts   $FF00,r2  ; Write back
```

ADIW – Add Immediate to Word

Description:

Adds an immediate value (0 - 63) to a register pair and places the result in the register pair. This instruction operates on the upper four register pairs, and is well suited for operations on the pointer registers.

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

Operation:

(i) Rd+1:Rd ← Rd+1:Rd + K

	Syntax:	Operands:	Program Counter:
(i)	ADIW Rd+1:Rd,K	d ∈ {24,26,28,30}, 0 ≤ K ≤ 63	PC ← PC + 1

16-bit Opcode:

1001	0110	KKdd	KKKK

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	⇔	⇔	⇔	⇔	⇔

S: N ⊕ V, For signed tests.

V: $\overline{Rdh7} \bullet R15$
Set if two's complement overflow resulted from the operation; cleared otherwise.

N: R15
Set if MSB of the result is set; cleared otherwise.

Z: $\overline{R15} \bullet \overline{R14} \bullet \overline{R13} \bullet \overline{R12} \bullet \overline{R11} \bullet \overline{R10} \bullet \overline{R9} \bullet \overline{R8} \bullet \overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result is $0000; cleared otherwise.

C: $\overline{R15} \bullet Rdh7$
Set if there was carry from the MSB of the result; cleared otherwise.

R (Result) equals Rdh:Rdl after the operation (Rdh7-Rdh0 = R15-R8, Rdl7-Rdl0=R7-R0).

Example:

```
adiw r25:24,1  ; Add 1 to r25:r24
adiw ZH:ZL,63  ; Add 63 to the Z-pointer(r31:r30)
```

Words: 1 (2 bytes)
Cycles: 2

CLR – Clear Register

Description:

Clears a register. This instruction performs an Exclusive OR between a register and itself. This will clear all bits in th register.

Operation:

(i) Rd ← Rd ⊕ Rd

	Syntax:	Operands:	Program Counter:
(i)	CLR Rd	0 ≤ d ≤ 31	PC ← PC + 1

16-bit Opcode: (see EOR Rd,Rd)

0010	01dd	dddd	dddd

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	0	0	0	1	–

S: 0
Cleared

V: 0
Cleared

N: 0
Cleared

Z: 1
Set

R (Result) equals Rd after the operation.

Example:

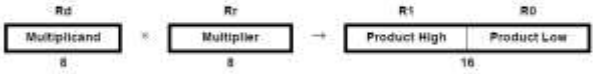
```
        clr   r18      ; clear r18
loop:   inc   r18      ; increase r18
        ...
        cpi   r18,$50  ; Compare r18 to $50
        brne  loop
```

Words: 1 (2 bytes)
Cycles: 1

MUL – Multiply Unsigned

Description:

This instruction performs 8-bit × 8-bit → 16-bit unsigned multiplication.



The multiplicand Rd and the multiplier Rr are two registers containing unsigned numbers. The 16-bit ur placed in R1 (high byte) and R0 (low byte). Note that if the multiplicand or the multiplier is selected from R will overwrite those after multiplication.

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

Operation:

(i) R1:R0 ← Rd × Rr (unsigned ← unsigned × unsigned)

	Syntax:	Operands:	Program Counter:
(i)	MUL Rd,Rr	0 ≤ d ≤ 31, 0 ≤ r ≤ 31	PC ← PC + 1

16-bit Opcode:

1001	11rd	dddd	rrrr

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	⇔	⇔

C: R15
Set if bit 15 of the result is set; cleared otherwise.

Z: $\overline{R15} \bullet \overline{R14} \bullet \overline{R13} \bullet \overline{R12} \bullet \overline{R11} \bullet \overline{R10} \bullet \overline{R9} \bullet \overline{R8} \bullet \overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result is $0000; cleared otherwise.

R (Result) equals R1,R0 after the operation.

Example:

```
mul  r5,r4    ; Multiply unsigned r5 and r4
movw r4,r0    ; Copy result back in r5:r4
```

Words: 1 (2 bytes)
Cycles: 2

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 ← Rd • K

	Syntax:	Operands:	Program Counter:
(i)	ANDI Rd,K	16 ≤ d ≤ 31, 0 ≤ K ≤ 255	PC ← PC + 1

16-bit Opcode:

0111	KKKK	dddd	KKKK

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	⇔	0	⇔	⇔	–

S: N ⊕ 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  r16,$10  ; Isolate bit 4 in r16
andi  r19,$AA  ; Clear odd bits of r19
```

Words: 1 (2 bytes)
Cycles: 1

Linnéuniversitetet

Name: _____

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 ← Rd ⊕ Rr

	Syntax:	Operands:	Program Counter:
(i)	EOR Rd,Rr	0 ≤ d ≤ 31, 0 ≤ r ≤ 31	PC ← PC + 1

16-bit Opcode:

0010	01rd	dddd	rrrr

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	⇔	0	⇔	⇔	–

S: N ⊕ 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:

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

Words: 1 (2 bytes)
Cycles: 1

CP – Compare

Description:

This instruction performs a compare between two registers Rd and Rr. None of the registers are changed. All conditional branches can be used after this instruction.

Operation:

(i) Rd - Rr

	Syntax:	Operands:	Program Counter:
(i)	CP Rd,Rr	0 ≤ d ≤ 31, 0 ≤ r ≤ 31	PC ← PC + 1

16-bit Opcode:

0001	01rd	dddd	rrrr

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	⇔	⇔	⇔	⇔	⇔	⇔

H: $\overline{Rd3} \bullet Rr3 + Rr3 \bullet R3 + R3 \bullet \overline{Rd3}$
Set if there was a borrow from bit 3; cleared otherwise.

S: N ⊕ V, For signed tests.

V: $Rd7 \bullet \overline{Rr7} \bullet \overline{R7} + \overline{Rd7} \bullet Rr7 \bullet R7$
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: $\overline{Rd7} \bullet Rr7 + Rr7 \bullet R7 + R7 \bullet \overline{Rd7}$
Set if the absolute value of the contents of Rr is larger than the absolute value of Rd; cleared otherwise.

R (Result) after the operation.

Example:

```
        cp    r4,r19    ; Compare r4 with r19
        brne  noteq     ; Branch if r4 <> r19
        ...
noteq:  nop             ; Branch destination (do nothing)
```

ADD – Add without Carry

Description:

Adds two registers without the C Flag and places the result in the destination regist

Operation:

(i) Rd ← Rd + Rr

	Syntax:	Operands:	Program Counter:
(i)	ADD Rd,Rr	0 ≤ d ≤ 31, 0 ≤ r ≤ 31	PC ← PC + 1

16-bit Opcode:

0000	11rd	dddd	rrrr

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	⇔	⇔	⇔	⇔	⇔	⇔

H: $Rd3 \bullet Rr3 + Rr3 \bullet \overline{R3} + \overline{R3} \bullet Rd3$
Set if there was a carry from bit 3; cleared otherwise

S: N ⊕ V, For signed tests.

V: $Rd7 \bullet Rr7 \bullet \overline{R7} + \overline{Rd7} \bullet \overline{Rr7} \bullet R7$
Set if two's complement overflow resulted from the operation; cleared other

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: $Rd7 \bullet Rr7 + Rr7 \bullet \overline{R7} + \overline{R7} \bullet Rd7$
Set if there was carry from the MSB of the result; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```
add   r1,r2     ; Add r2 to r1 (r1=r1+r2)
add   r28,r28   ; Add r28 to itself (r28=r28+r28)
```

Words: 1 (2 bytes)
Cycles: 1

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:	Operands:	Program Counter:
(i)	CPI Rd,K	16 ≤ d ≤ 31, 0≤ K ≤ 255	PC ← PC + 1

16-bit Opcode:

0011	KKKK	dddd	KKKK

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	⇔	⇔	⇔	⇔	⇔	⇔

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 ⊕ 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.

Example:

```
        cpi   r19,3     ; Compare r19 with 3
        brne  error     ; Branch if r19<>3
        ...
error:  nop             ; Branch destination (do nothing)
```

Words: 1 (2 bytes)
Cycles: 1

CPC – Compare with Carry

Description:

This instruction performs a compare between two registers Rd and Rr and also takes into account the previous carry. None of the registers are changed. All conditional branches can be used after this instruction.

Operation:

(i) Rd - Rr - C

	Syntax:	Operands:	Program Counter:
(i)	CPC Rd,Rr	0 ≤ d ≤ 31, 0 ≤ r ≤ 31	PC ← PC + 1

16-bit Opcode:

0000	01rd	dddd	rrrr

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
-	-	⇔	⇔	⇔	⇔	⇔	⇔

H: $\overline{Rd3} \bullet Rr3 + Rr3 \bullet R3 + R3 \bullet \overline{Rd3}$
Set if there was a borrow from bit 3; cleared otherwise

S: N ⊕ V, For signed tests.

V: $Rd7 \bullet \overline{Rr7} \bullet \overline{R7} + \overline{Rd7} \bullet Rr7 \bullet R7$
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: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0} \bullet Z$
Previous value remains unchanged when the result is zero; cleared otherwise.

C: $\overline{Rd7} \bullet Rr7 + Rr7 \bullet R7 + R7 \bullet \overline{Rd7}$
Set if the absolute value of the contents of Rr plus previous carry is larger than the absolute value of R... otherwise.

R (Result) after the operation.

Example:

```
                        ; Compare r3:r2 with r1:r0
        cp      r2,r0   ; Compare low byte
        cpc     r3,r1   ; Compare high byte
        brne    noteq   ; Branch if not equal
        ...
noteq:  nop             ; Branch destination (do nothing)
```

BRLT – Branch if Less Than (Signed)

Description:

Conditional relative branch. Tests the Signed Flag (S) and branches relatively to PC if S is set. 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 signed binary number represented in Rd was less than the signed binary number represented in Rr. This instruction branches relatively to PC in either direction (PC - 63 ≤ destination ≤ PC + 64). The parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBS 4,k).

Operation:

(i) If Rd < Rr (N ⊕ V = 1) then PC ← PC + k + 1, else PC ← PC + 1

	Syntax:	Operands:	Program Counter:
(i)	BRLT k	-64 ≤ k ≤ +63	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16-bit Opcode:

1111	00kk	kkkk	k100

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
        cp    r16,r1      ; Compare r16 to r1
        brlt  less        ; Branch if r16 < r1 (signed)
        ...
less:   nop               ; Branch destination (do nothing)
```

Words: 1 (2 bytes)
Cycles: 1 if condition is false
2 if condition is true

SUBI – Subtract Immediate

Description:

Subtracts a register and a constant and places the result in the destination register Rd. This instruction is working on Register R16 to R31 and is very well suited for operations on the X, Y and Z-pointers.

Operation:

(i) Rd ← Rd - K

	Syntax:	Operands:	Program Counter:
(i)	SUBI Rd,K	16 ≤ d ≤ 31, 0 ≤ K ≤ 255	PC ← PC + 1

16-bit Opcode:

0101	KKKK	dddd	KKKK

Status Register and Boolean Formula:

I	T	H	S	V	N	Z	C
-	-	⇔	⇔	⇔	⇔	⇔	⇔

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 ⊕ 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) equals Rd after the operation.

Example:

```
        subi    r22,$11 ; Subtract $11 from r22
        brne    noteq   ; Branch if r22<>$11
        ...
noteq:  nop             ; Branch destination (do nothing)
```

Words: 1 (2 bytes)
Cycles: 1

SBCI – Subtract Immediate with Carry

Description:

Subtracts a constant from a register and subtracts with the C Flag and places the result in the destination register Rd.

Operation:

(i) Rd ← Rd - K - C

	Syntax:	Operands:	Program Counter:
(i)	SBCI Rd,K	16 ≤ d ≤ 31, 0 ≤ K ≤ 255	PC ← PC + 1

16-bit Opcode:

0100	KKKK	dddd	KKKK

Status Register and Boolean Formula:

I	T	H	S	V	N	Z	C
-	-	⇔	⇔	⇔	⇔	⇔	⇔

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 ⊕ 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: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0} \bullet Z$
Previous value remains unchanged when the result is zero; cleared otherwise.

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

R (Result) equals Rd after the operation.

Example:

```
                        ; Subtract $4923 from r17:r16
        subi  r16,$23   ; Subtract low byte
        sbci  r17,$49   ; Subtract with carry high byte
```

Words: 1 (2 bytes)
Cycles: 1