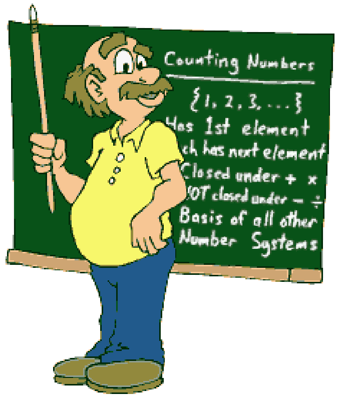
Ingeniørhøjskolen i Århus

IECA

Embedded Computer Architecture

Lesson 5: Assembly programming



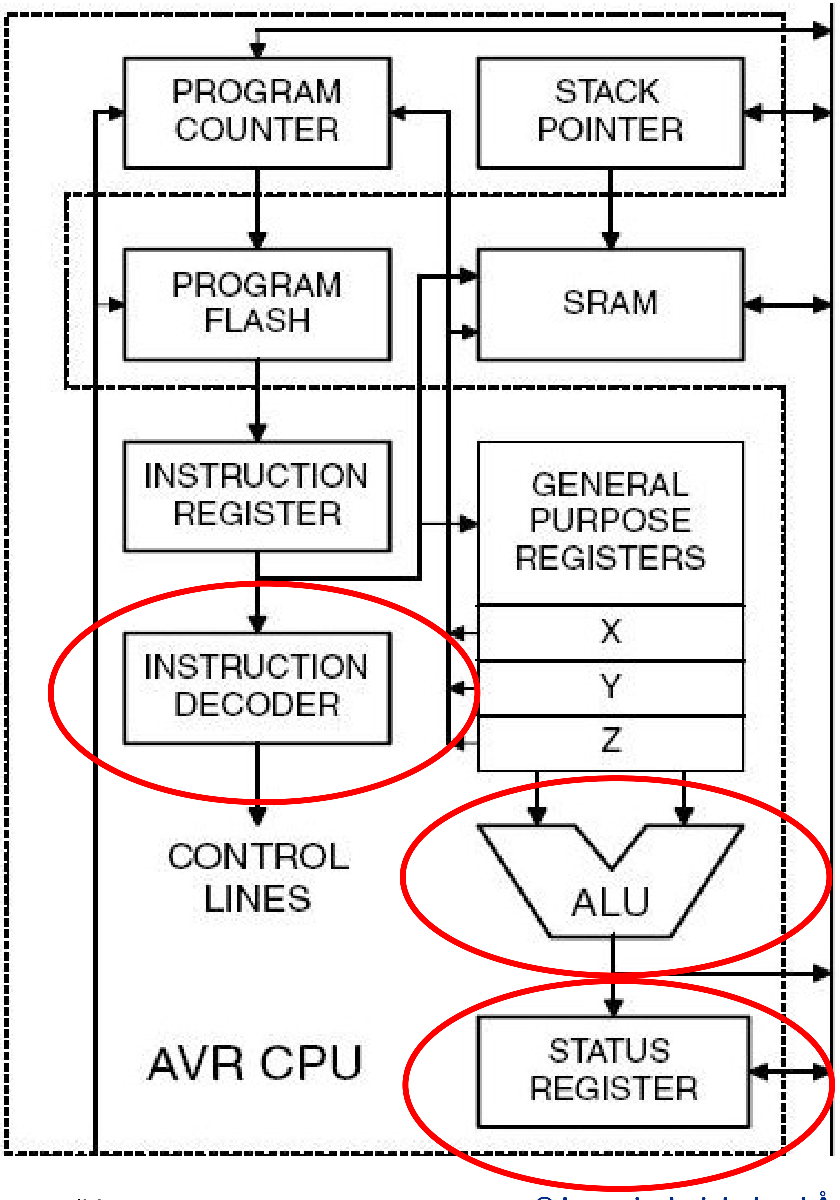
Version: 29-10-2015, Henning Hargaard

Mega32 CPU and the machine codes

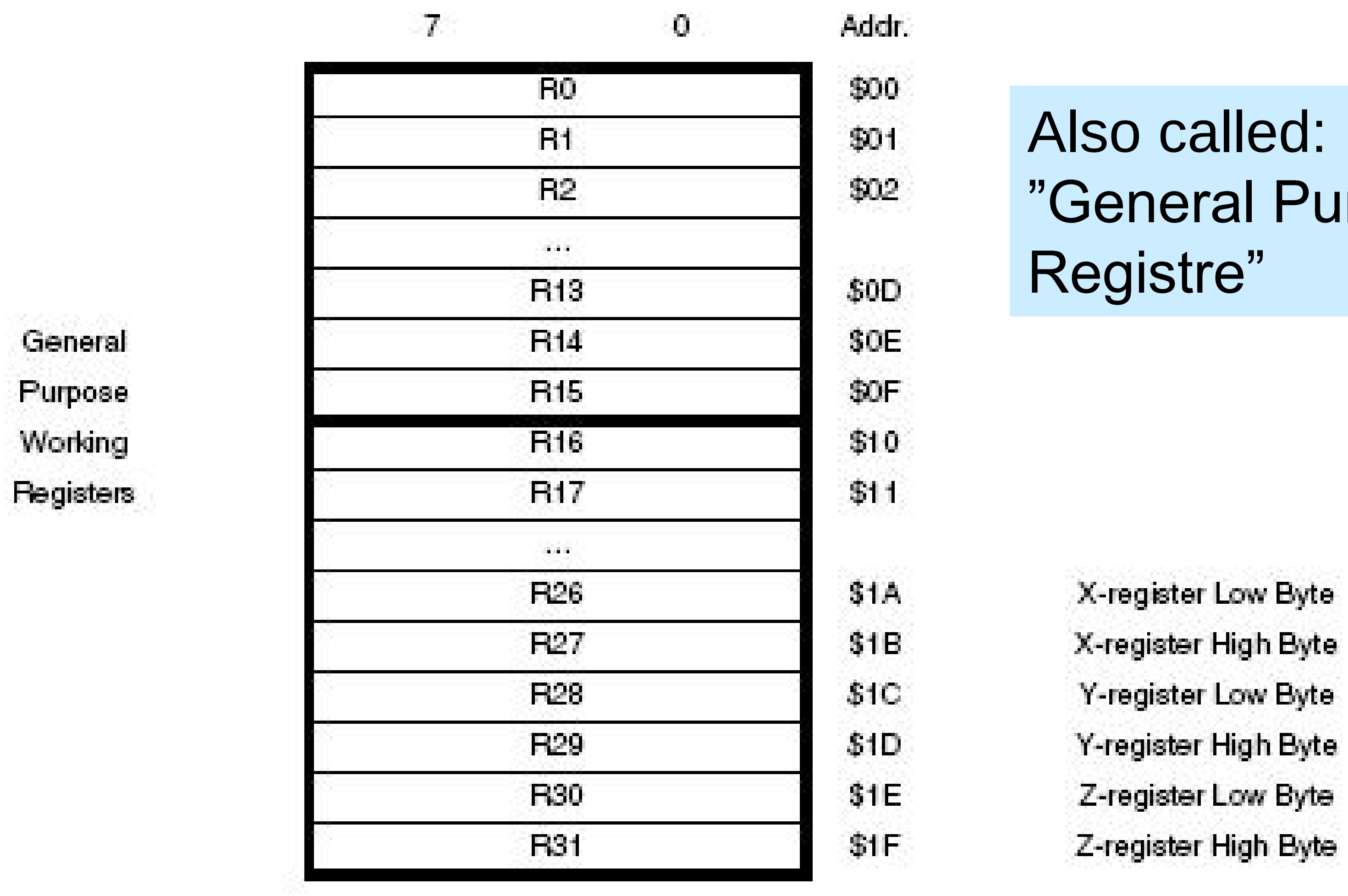
Instructions are binary codes (all 0's and 1's) !

The ALU does the calculations.

The status register changes while executing <u>some</u> instructions.



Mega32 Working Registers



General Purpose Working Registers

7 … 0	Addr.	
R0	\$00	
R1	\$01	
R2	\$02	
...		
R13	\$0D	
R14	\$0E	
R15	\$0F	
R16	\$10	
R17	\$11	
...		
R26	\$1A	X-register Low Byte
R27	\$1B	X-register High Byte
R28	\$1C	Y-register Low Byte
R29	\$1D	Y-register High Byte
R30	\$1E	Z-register Low Byte
R31	\$1F	Z-register High Byte

Also called:
”General Purpose Registre”

Instruction Groups

- Arithmetic and logical
- Branch
- Data transfer
- Bit- and bit test

LDI (Load Immediate)

Description:

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

Operation:

(i) $Rd \leftarrow K$

	Syntax:	Operands:	Program Counter:
(i)	LDI Rd,K	$16 \le d \le 31, 0 \le K \le 255$	$PC \leftarrow PC + 1$

16-bit Opcode:

1110	KKKK	dddd	KKKK

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
```

CLR (Clear register)

Description:

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

Operation:

(i) Rd ← Rd ⊕ Rd

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

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

0010	01dd	dddd	dddd

Example:

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

SER (Set all bits in register)

Description:

Loads $FF directly to register Rd.

Operation:

(i) Rd ← $FF

	Syntax:	Operands:	Program Counter:
(i)	SER Rd	$16 \le d \le 31$	PC ← PC + 1

16-bit Opcode:

1110	1111	dddd	1111

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
```

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	$0 \le d \le 31$, $0 \le r \le 31$	PC ← PC + 1

16-bit Opcode:

0010	11rd	dddd	rrrr

Example:

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

COM (Ones Complement)

Description:

This instruction performs a One's Complement of register Rd.

Operation:

(i) Rd ← $FF - Rd

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

16-bit Opcode:

1001	010d	dddd	0000

Example:

```
        com     r4      ; Take one's complement of r4
        breq    zero    ; Branch if zero
        ...
zero:   nop             ; Branch destination (do nothing)
```

ADD (Add without Carry)

Description:

Adds two registers without the C Flag and places the result in the destination register Rd.

Operation:

(i) Rd ← Rd + Rr

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

16-bit Opcode:

0000	11rd	dddd	rrrr

Example:

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

SUB (Subtract without Carry)

Description:

Subtracts two registers and places the result in the destination register Rd.

Operation:

(i) Rd ← Rd - Rr

	Syntax:	Operands:	Program Counter:
(i)	SUB Rd,Rr	$0 \le d \le 31$, $0 \le r \le 31$	PC ← PC + 1

16-bit Opcode:

0001	10rd	dddd	rrrr

Example:

```
        sub   r13,r12     ; Subtract r12 from r13
        brne  noteq       ; Branch if r12<>r13
        ...
noteq:  nop               ; Branch destination (do nothing)
```

INC (Increment)

Description:

Adds one -1- to the contents of register Rd and places the result in the destination register Rd.

The C Flag in SREG is not affected by the operation, thus allowing the INC instruction to be used on a loop counter in multiple-precision computations.

When operating on unsigned numbers, only BREQ and BRNE branches can be expected to perform consistently. When operating on two's complement values, all signed branches are available.

Operation:

(i) Rd ← Rd + 1

	Syntax:	Operands:	Program Counter:
(i)	INC Rd	$0 \le d \le 31$	PC ← PC + 1

16-bit Opcode:

1001	010d	dddd	0011

Example:

```
        clr     r22         ; clear r22
loop:   inc     r22         ; increment r22
        ...
        cpi     r22,$4F     ; Compare r22 to $4f
        brne    loop        ; Branch if not equal
        nop                 ; Continue (do nothing)
```

DEC (Decrement)

Description:

Subtracts one -1- from the contents of register Rd and places the result in the destination register Rd.

The C Flag in SREG is not affected by the operation, thus allowing the DEC instruction to be used on a loop counter in multiple-precision computations.

When operating on unsigned values, only BREQ and BRNE branches can be expected to perform consistently. When operating on two's complement values, all signed branches are available.

Operation:

(i) Rd ← Rd - 1

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

16-bit Opcode:

1001	010d	dddd	1010

Example:

```
        ldi   r17,$10    ; Load constant in r17
 loop:  add   r1,r2      ; Add r2 to r1
        dec   r17        ; Decrement r17
        brne  loop       ; Branch if r17<>0
        nop              ; Continue (do nothing)
```

Test (”socrative.com”: Room = MSYS)

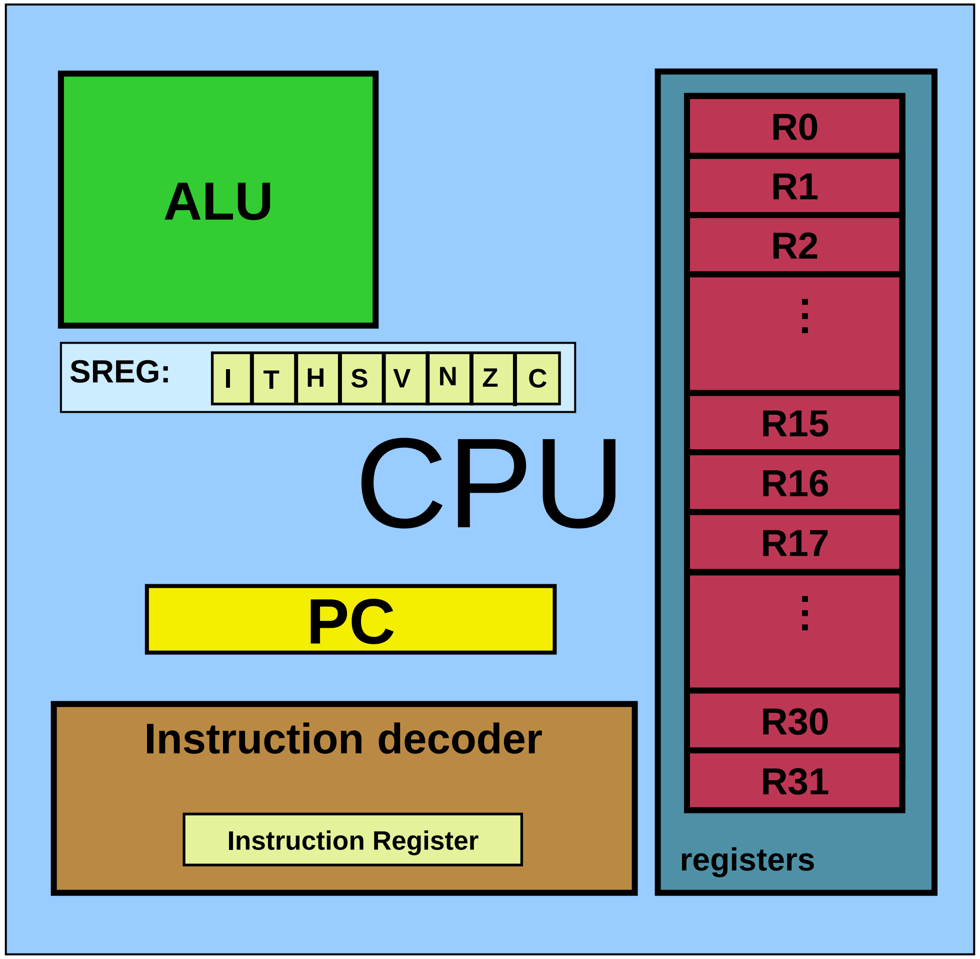
- What will be the content of R20 after this:

```
LDI R19,2
LDI R20,200
ADD R20,R19
INC R20
INC R20
```

A: 200
B: 202
C: 203
D: 204

The AVR CPU

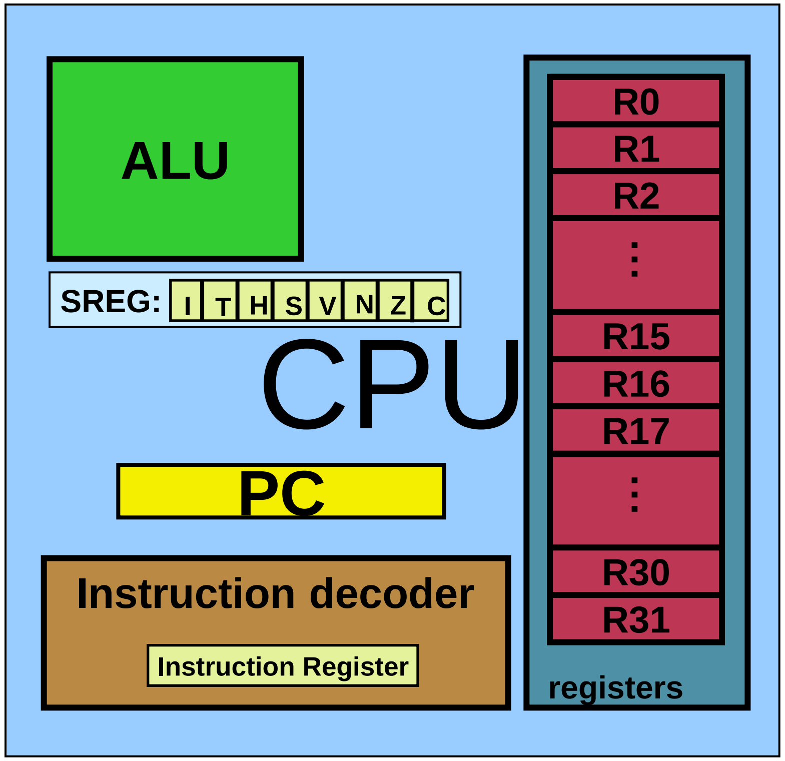
- AVR's CPU
  - ALU
  - 32 General Purpose registers (R0 to R31)
  - PC register
  - Instruction decoder



A simple program

- Write a program that calculates 19 + 95

```
LDI R16, 19     ;R16 = 19
LDI R20, 95     ;R20 = 95
ADD R16, R20    ;R16 = R16 + R20
```



A simple program

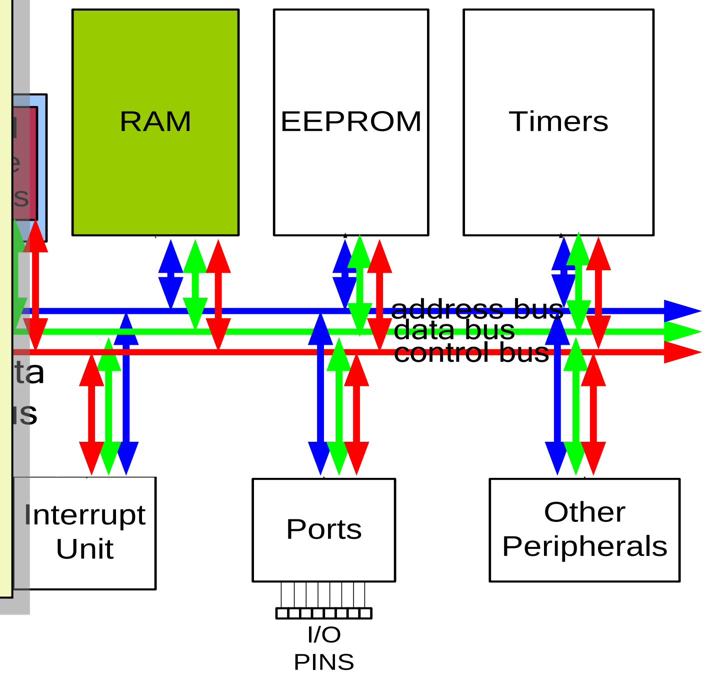
- Write a program that calculates 19 + 95 + 5

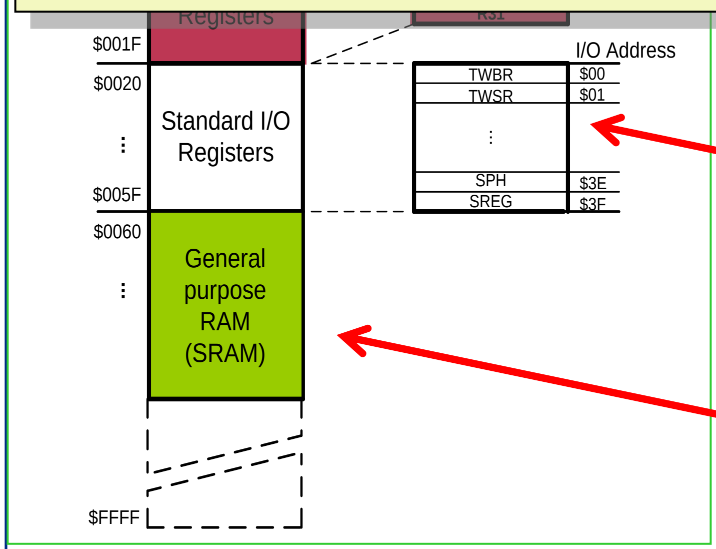
```
LDI     R16, 19          ;R16 = 19
LDI     R20, 95          ;R20 = 95
LDI     R21, 5           ;R21 = 5
ADD     R16, R20         ;R16 = R16 + R20
ADD     R16, R21         ;R16 = R16 + R21
```

```
LDI     R16, 19          ;R16 = 19
LDI     R20, 95          ;R20 = 95
ADD     R16, R20         ;R16 = R16 + R20
LDI     R20, 5           ;R20 = 5
ADD     R16, R20         ;R16 = R16 + R20
```

Space

Address		Name
I/O	Mem.	
$00	$20	TWBR
$01	$21	TWSR
$02	$22	TWAR
$03	$23	TWDR
$04	$24	ADCL
$05	$25	ADCH
$06	$26	ADCSRA
$07	$27	ADMUX
$08	$28	ACSR
$09	$29	UBRRL
$0A	$2A	UCSRB
$0B	$2B	UCSRA
$0C	$2C	UDR
$0D	$2D	SPCR
$0E	$2E	SPSR
$0F	$2F	SPDR
$10	$30	PIND
$11	$31	DDRD
$12	$32	PORTD
$13	$33	PINC
$14	$34	DDRC
$15	$35	PORTC
$16	$36	PINB
$17	$37	DDRB
$18	$38	PORTB
$19	$39	PINA
$1A	$3A	DDRA
$1B	$3B	PORTA
$1C	$3C	EECR
$1D	$3D	EEDR
$1E	$3E	EEARL
$1F	$3F	EEARH
$20	$40	UBRRC
		UBRRH
$21	$41	WDTCR
$22	$42	ASSR
$23	$43	OCR2
$24	$44	TCNT2
$25	$45	TCCR2
$26	$46	ICR1L
$27	$47	ICR1H
$28	$48	OCR1BL
$29	$49	OCR1BH
$2A	$4A	OCR1AL
$2B	$4B	OCR1AH
$2C	$4C	TCNT1L
$2D	$4D	TCNT1H
$2E	$4E	TCCR1B
$2F	$4F	TCCR1A
$30	$50	SFIOR
$31	$51	OCDR
		OSCCAL
$32	$52	TCNT0
$33	$53	TCCR0
$34	$54	MCUCSR
$35	$55	MCUCR
$36	$56	TWCR
$37	$57	SPMCR
$38	$58	TIFR
$39	$59	TIMSK
$3A	$5A	GIFR
$3B	$5B	GICR
$3C	$5C	OCR0
$3D	$5D	SPL
$3E	$5E	SPH
$3E	$5E	SREG





IN and OUT

LDS and STS

LDS (Load Direct from Data Space)

Description:

Loads one byte from the data space to a register. 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 LDS 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) Rd ← (k)

	Syntax:	Operands:	Program Counter:
(i)	LDS Rd,k	$0 \le d \le 31$, $0 \le k \le 65535$	PC ← PC + 2

32-bit Opcode:

1001	000d	dddd	0000
kkkk	kkkk	kkkk	kkkk

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
```

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) $\leftarrow$ Rr

	Syntax:	**Operands:**	**Program Counter:**
(i)	STS k,Rr	$0 \leq r \leq 31$, $0 \leq k \leq 65535$	PC $\leftarrow$ PC + 2

32-bit Opcode:

1001	001d	dddd	0000
kkkk	kkkk	kkkk	kkkk

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
```

IN (Load an I/O Location to Register)

Description:

Loads data from the I/O Space (Ports, Timers, Configuration Registers etc.) into register Rd in the Register File.

Operation:

(i) Rd ← I/O(A)

	Syntax:	Operands:	Program Counter:
(i)	IN Rd,A	$0 \le d \le 31$, $0 \le A \le 63$	PC ← PC + 1

16-bit Opcode:

1011	0AAd	dddd	AAAA

Example:

```
        in      r25,$16       ; Read Port B
        cpi     r25,4         ; Compare read value to constant
        breq    exit          ; Branch if r25=4
        ...
exit:   nop                   ; Branch destination (do nothing)
```

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	$0 \le r \le 31$, $0 \le A \le 63$	PC ← PC + 1

16-bit Opcode:

1011	1AAr	rrrr	AAAA

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
```

Assembler Directives .EQU and .SET

- .EQU *name = value*
  - *Example:*

```
.EQU    COUNT = 0x25
LDI     R21, COUNT                ;R21 = 0x25
LDI     R22, COUNT + 3            ;R22 = 0x28
```

- .SET *name = value*
  - *Example:*

```
.SET    COUNT = 0x25
LDI     R21, COUNT                ;R21 = 0x25
LDI     R22, COUNT + 3            ;R22 = 0x28
.SET    COUNT = 0x19
LDI     R21, COUNT                ;R21 = 0x19
```

Assembler Directives .INCLUDE

- .INCLUDE “*filename.ext*”

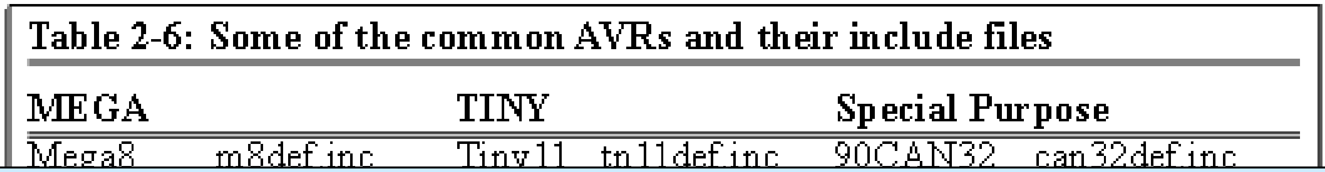


Table 2-6: Some of the common AVRs and their include files

MEGA		TINY		Special Purpose	
Mega8	m8def.inc	Tiny11	tn11def.inc	90CAN32	can32def.inc

M32def.inc

```
.equ    SREG    = 0x3f
.equ    SPL     = 0x3d
.equ    SPH     = 0x3e
....
.equ    INT_VECTORS_SIZE = 42    ; size in words
```

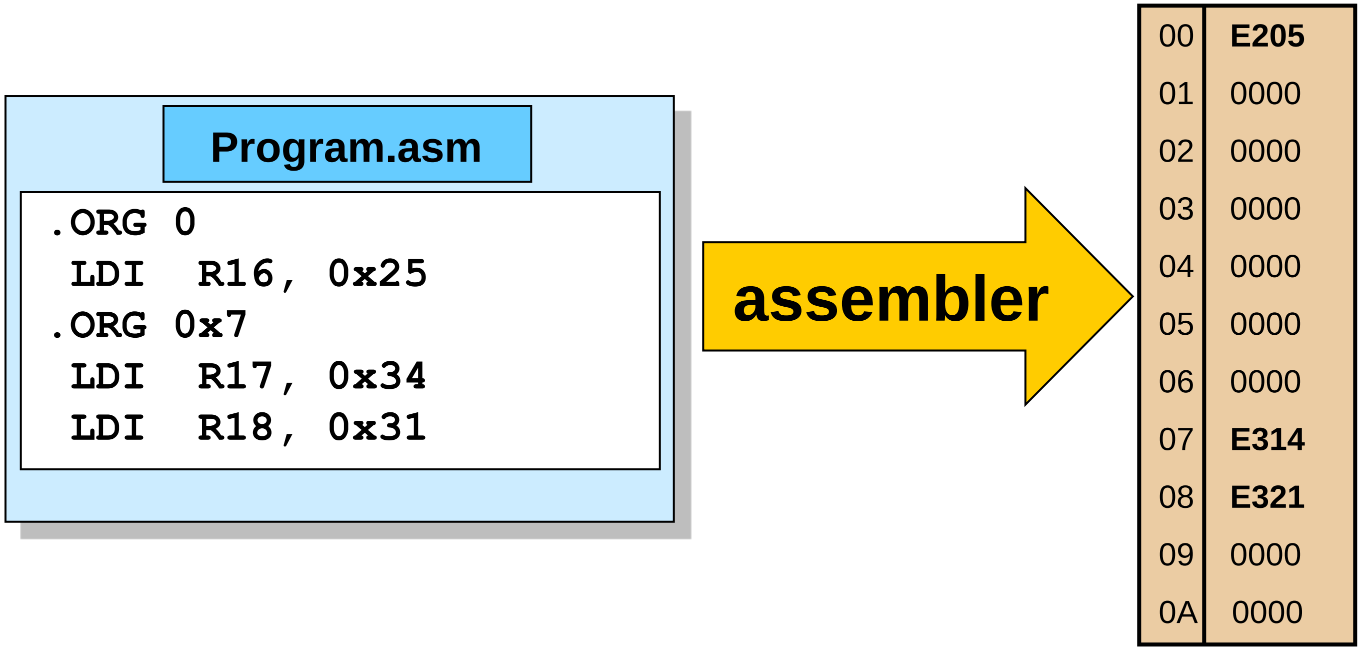
Program.asm

```
.INCLUDE “M32DEF.INC”
    LDI   R20, 10

    OUT   SPL, R20
```

Assembler Directives .ORG

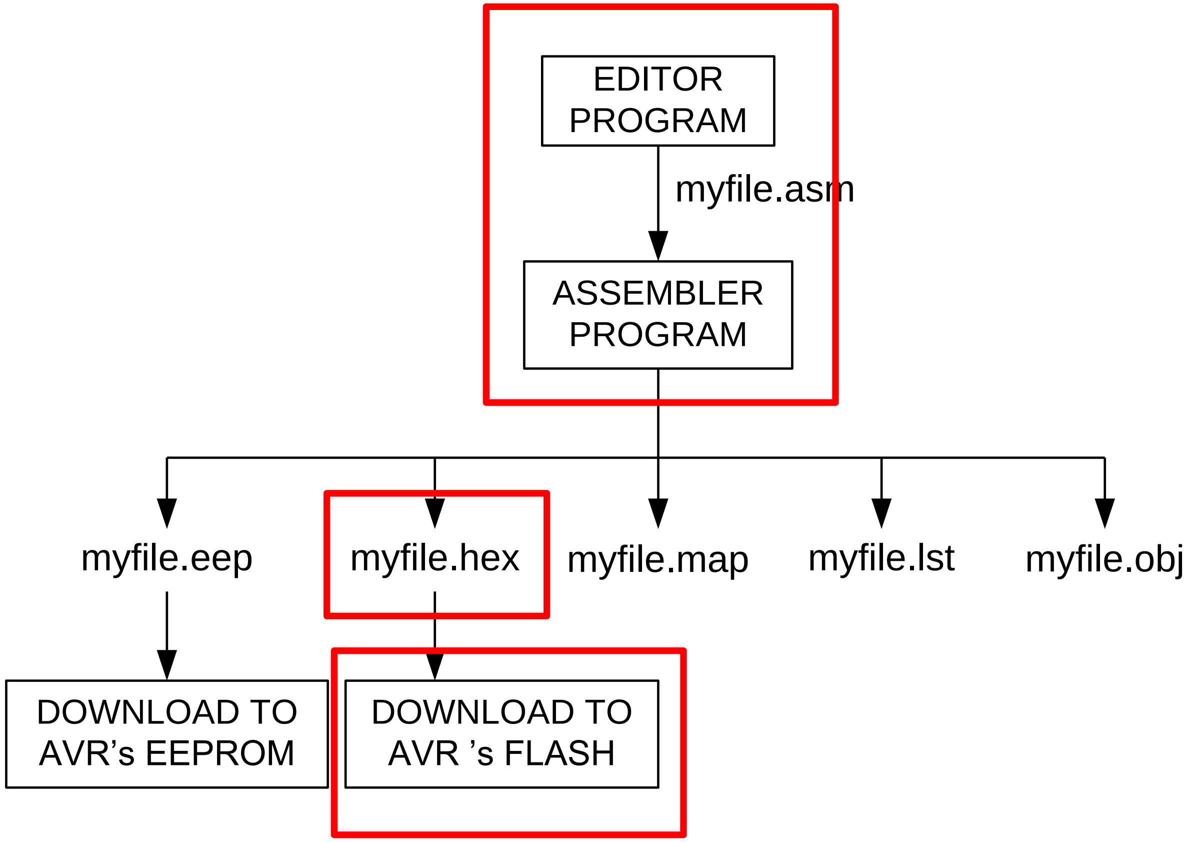
- .ORG *address*

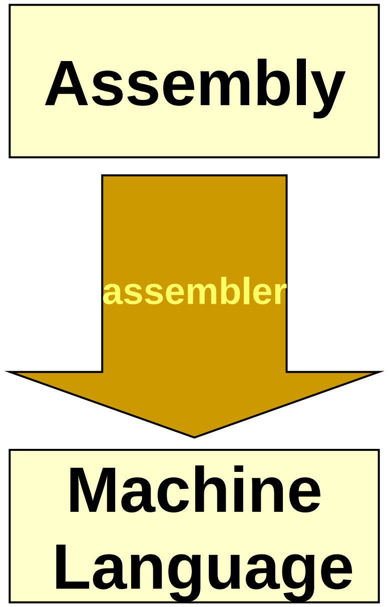


```
.ORG 0
 LDI  R16, 0x25
.ORG 0x7
 LDI  R17, 0x34
 LDI  R18, 0x31
```

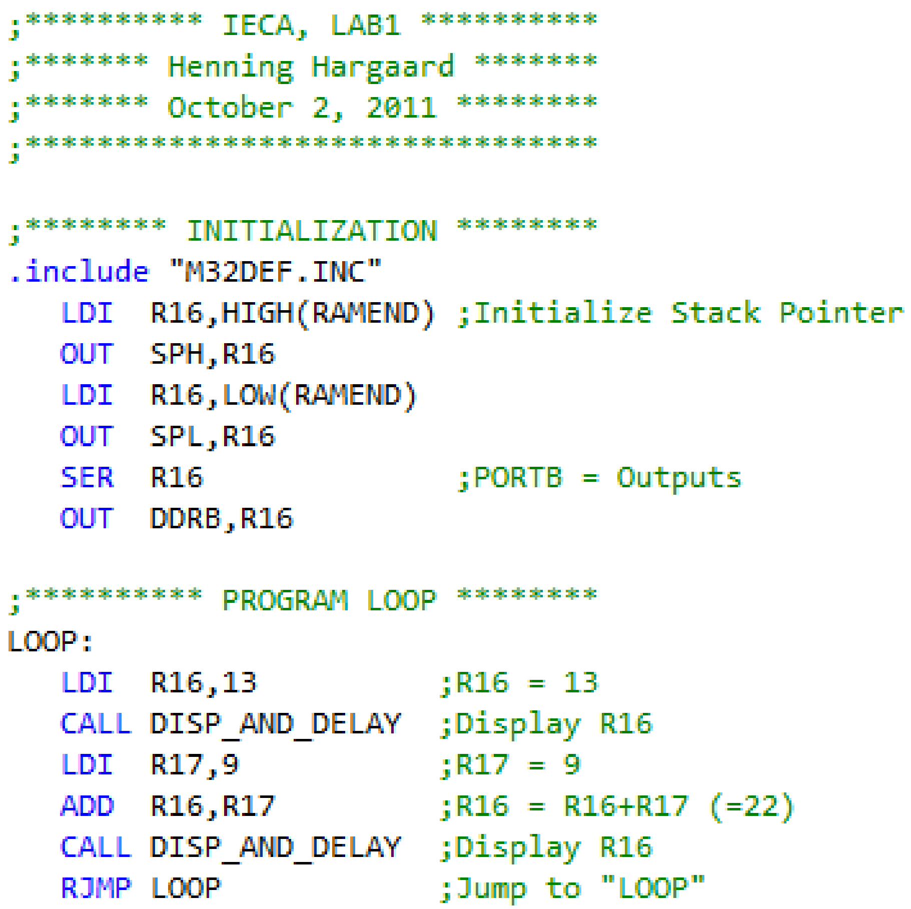
00	**E205**
01	0000
02	0000
03	0000
04	0000
05	0000
06	0000
07	**E314**
08	**E321**
09	0000
0A	0000

Assembler


EDITOR PROGRAM
myfile.asm
ASSEMBLER PROGRAM
myfile.eep
myfile.hex
myfile.map
myfile.lst
myfile.obj
DOWNLOAD TO AVR's EEPROM
DOWNLOAD TO AVR 's FLASH


Assembly
assembler
Machine Language

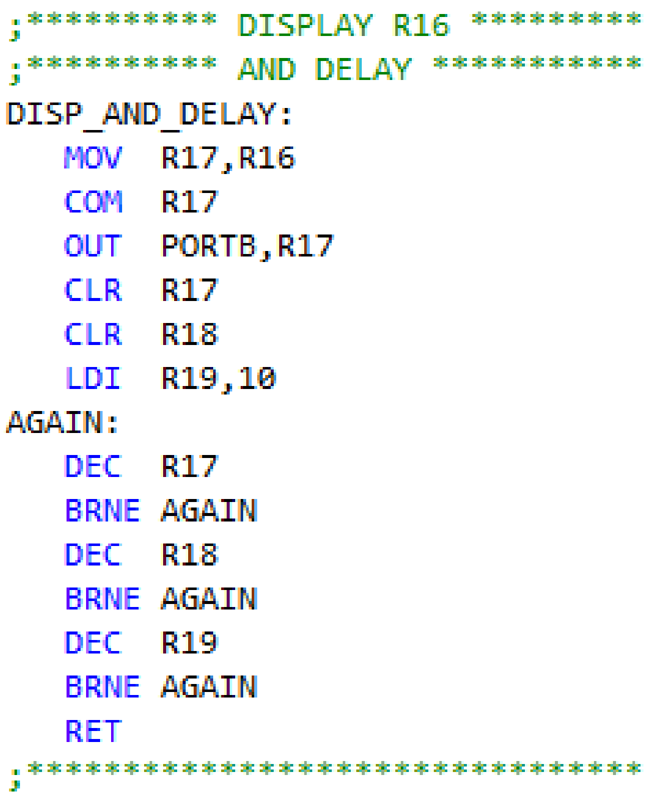
The LAB1 program



```
;********** IECA, LAB1 **********
;******* Henning Hargaard *******
;******* October 2, 2011 ********
;*******************************

;******** INITIALIZATION ********
.include "M32DEF.INC"
   LDI  R16,HIGH(RAMEND) ;Initialize Stack Pointer
   OUT  SPH,R16
   LDI  R16,LOW(RAMEND)
   OUT  SPL,R16
   SER  R16              ;PORTB = Outputs
   OUT  DDRB,R16

;********** PROGRAM LOOP ********
LOOP:
   LDI  R16,13          ;R16 = 13
   CALL DISP_AND_DELAY  ;Display R16
   LDI  R17,9           ;R17 = 9
   ADD  R16,R17         ;R16 = R16+R17 (=22)
   CALL DISP_AND_DELAY  ;Display R16
   RJMP LOOP            ;Jump to "LOOP"
```



```
;********** DISPLAY R16 *********
;********** AND DELAY ***********
DISP_AND_DELAY:
   MOV  R17,R16
   COM  R17
   OUT  PORTB,R17
   CLR  R17
   CLR  R18
   LDI  R19,10
AGAIN:
   DEC  R17
   BRNE AGAIN
   DEC  R18
   BRNE AGAIN
   DEC  R19
   BRNE AGAIN
   RET
;*******************************
```

....

End of lesson 5