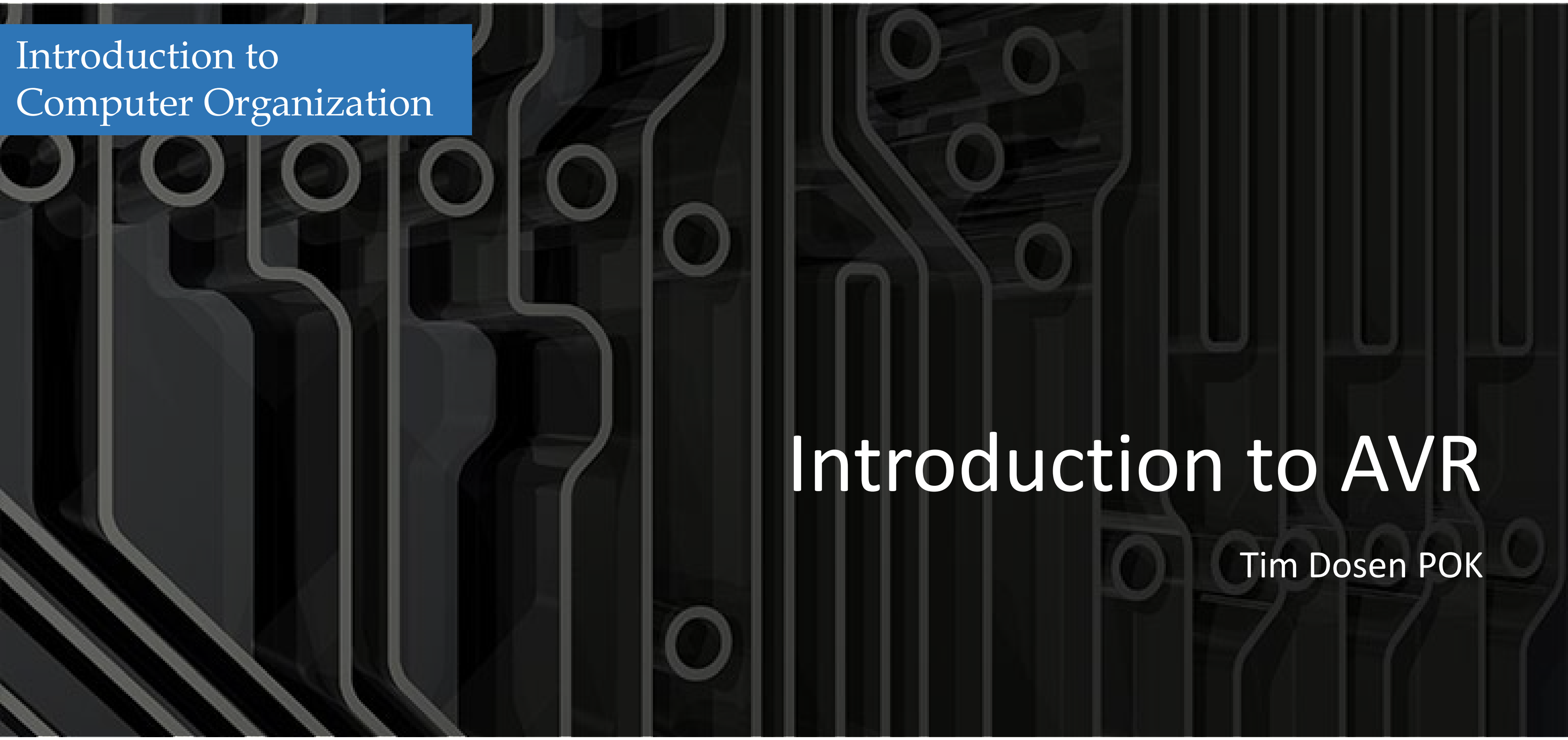


Introduction to Computer Organization

Introduction to AVR

Tim Dosen POK

Outline

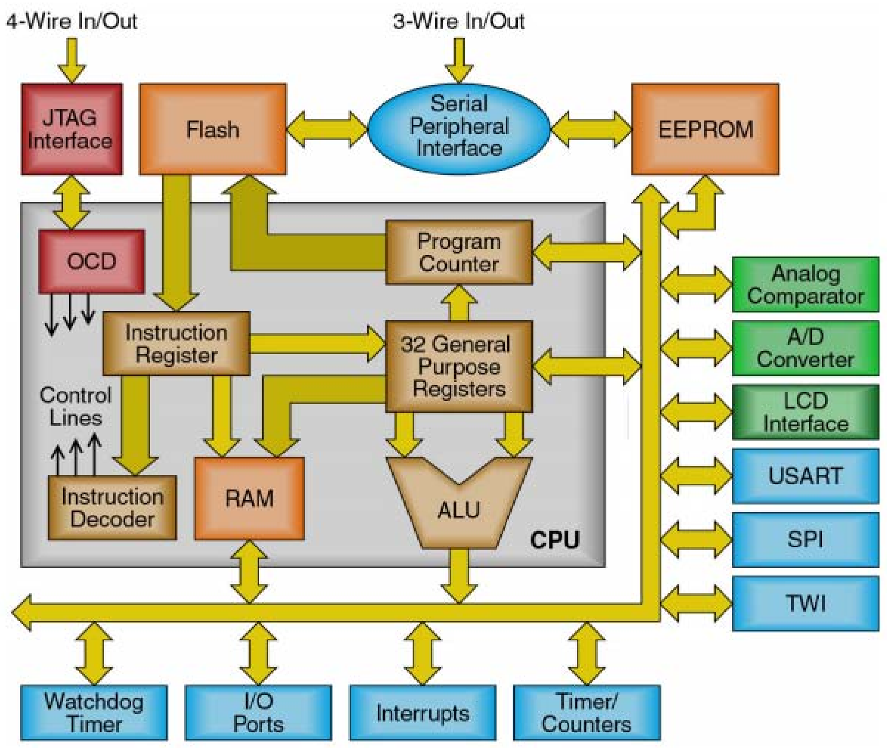
- AVR Architecture
- Register
- AVR instructions
- AVR program examples

AVR

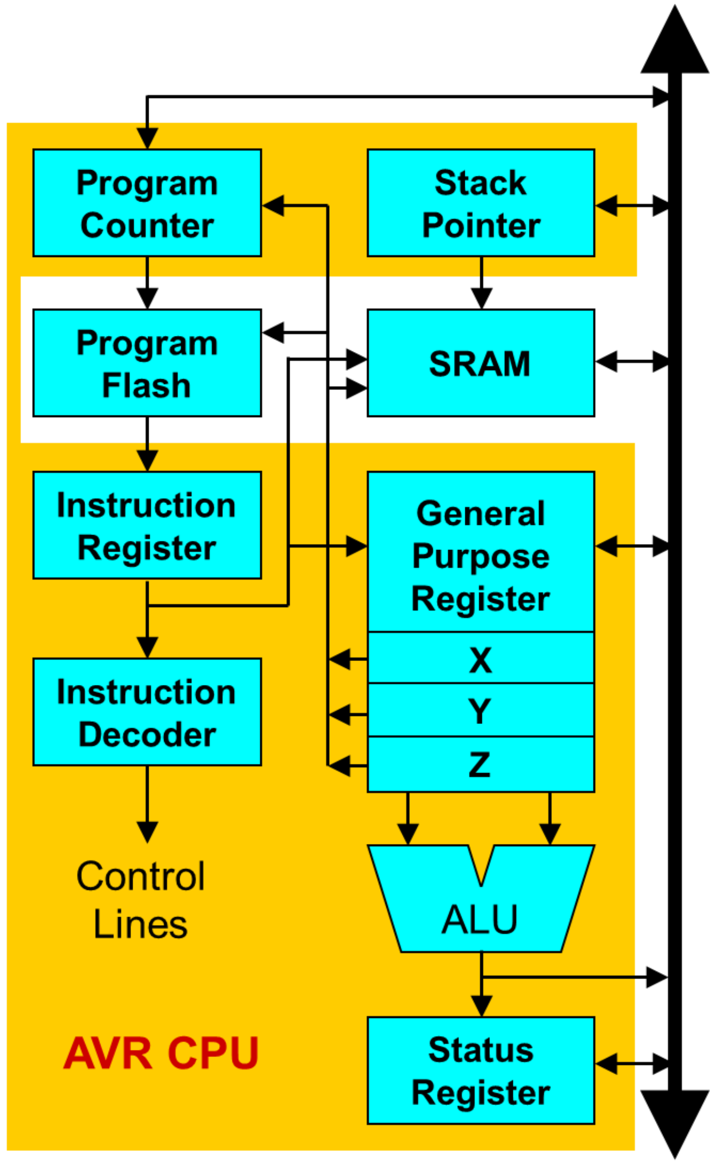
- Developed by **A**lf-Egil Bogen and **V**egard Wollan
- Is a **R**ISC processor
- **A**dvanced **V**irtual **R**isc processor

AVR

- Single cycle execution
  - One instruction per external clock
  - Low power consumption
- 32 Working Registers
  - All Directly connected to ALU!



Arsitektur AVR

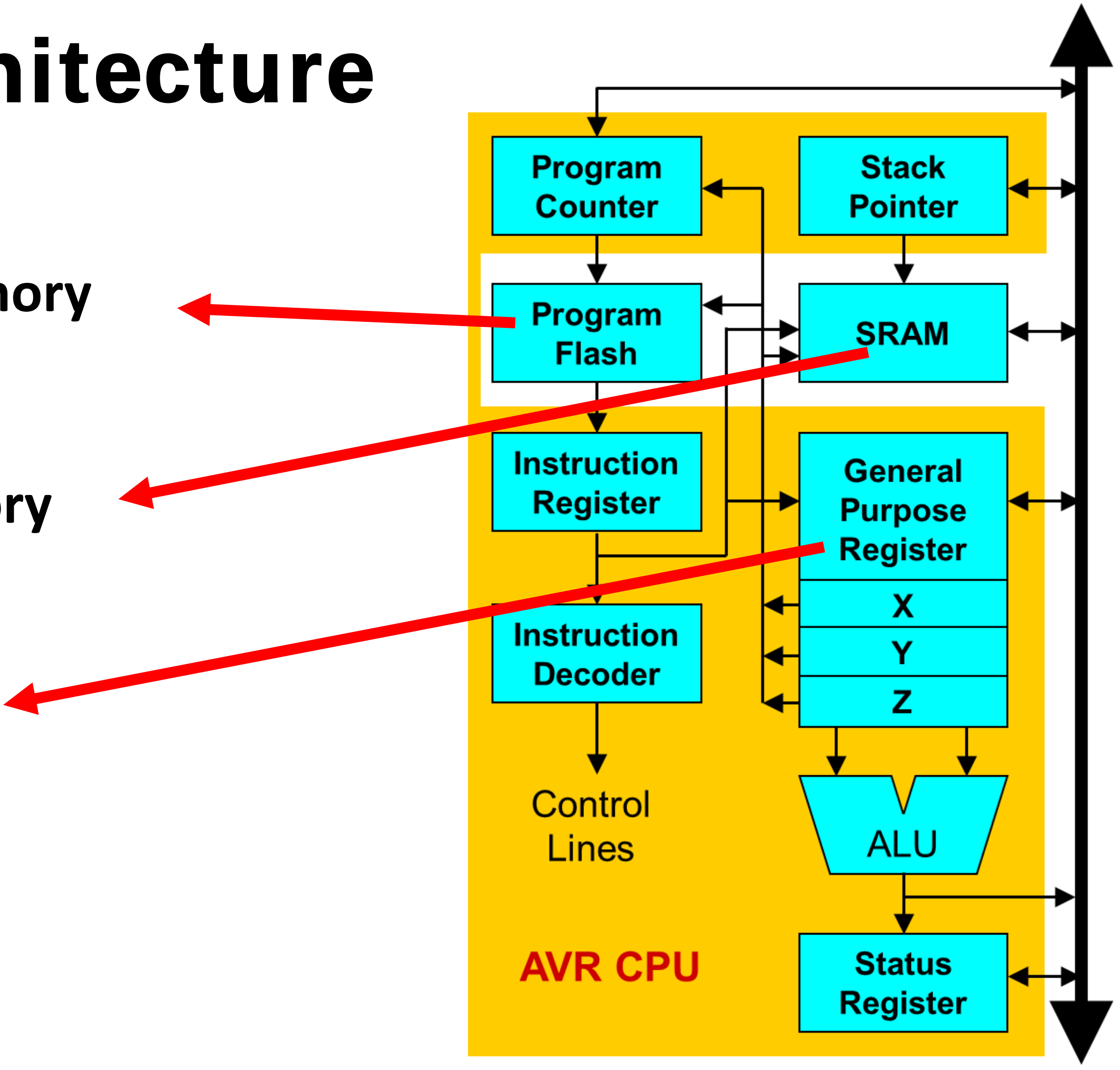


AVR Architecture

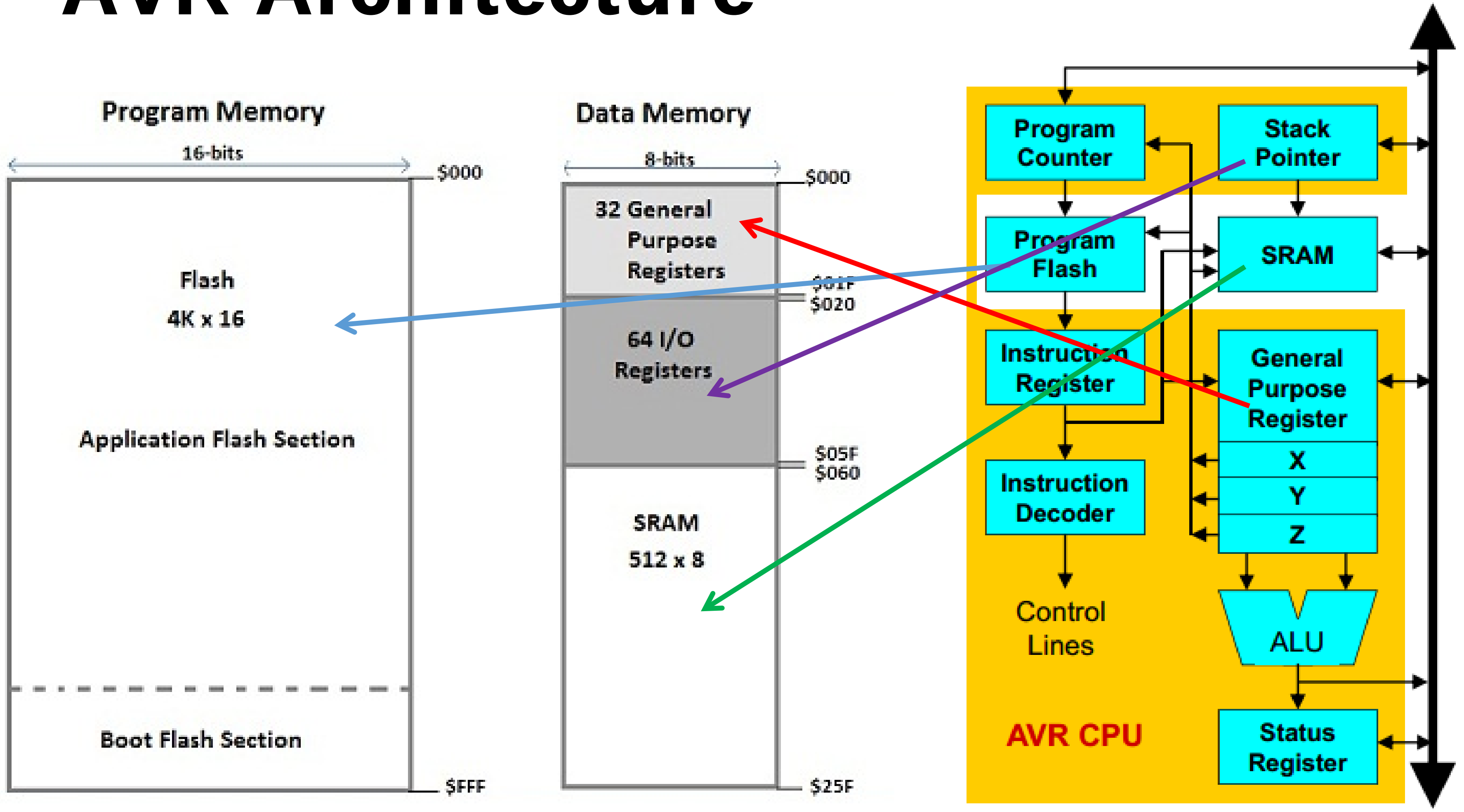
Program memory

Data memory

Register



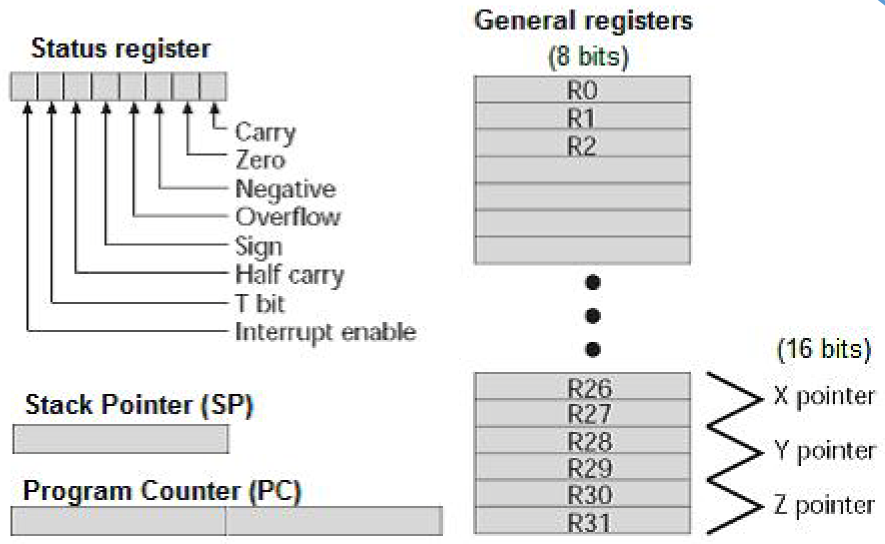
AVR Architecture

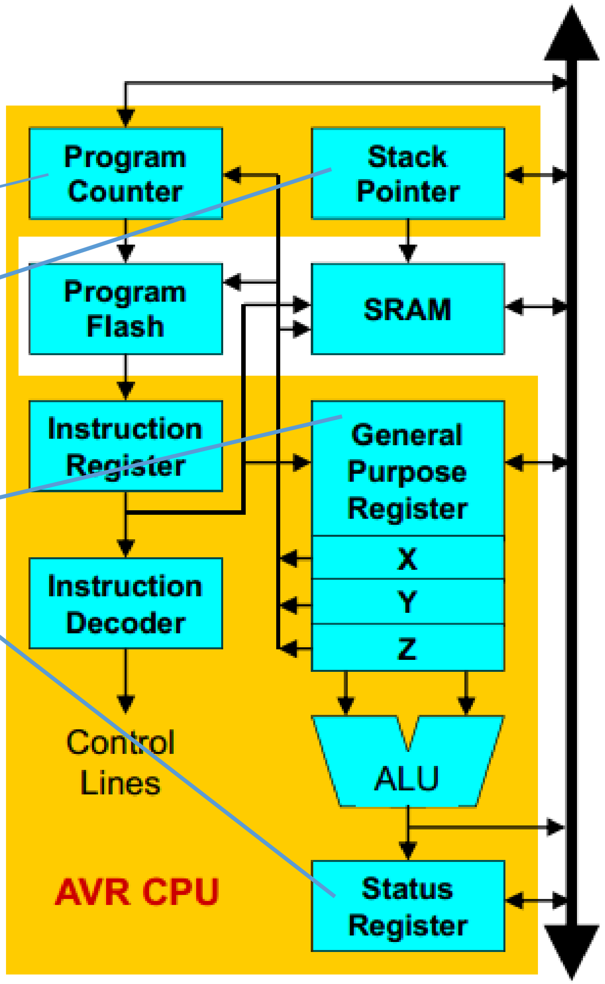


AVR Register

Types of AVR registers:

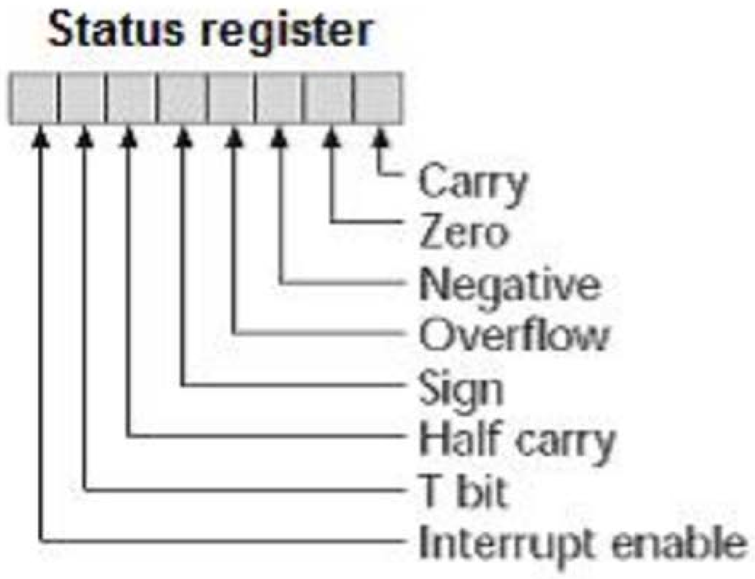
- 1 Program Counter (16 bit)
- 1 Status Register (8 bit)
- Stack Pointer (16 bit)
- 32 General Purpose Register (8 bit)





AVR Register: Status Register

- Contains 8 different flags
- Flag will be updated after every ALU operation
- Example:
  - If the result of the CPU operation is “0”, then flag ‘zero’ will be set to “1”

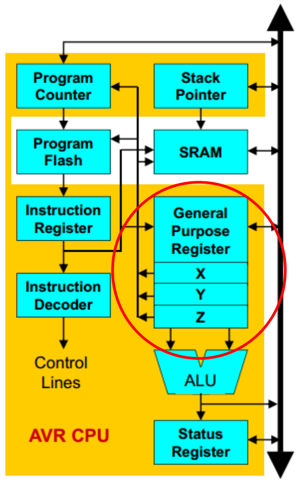


1

AVR Register: General Purpose

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

- Registers are special storages with 8 bits capacity.
- A register can store either:
  - numbers from 0 to 255 (unsigned),
  - numbers from -128 to +127 (signed),
  - ASCII-coded character
  - Any 8-bit data

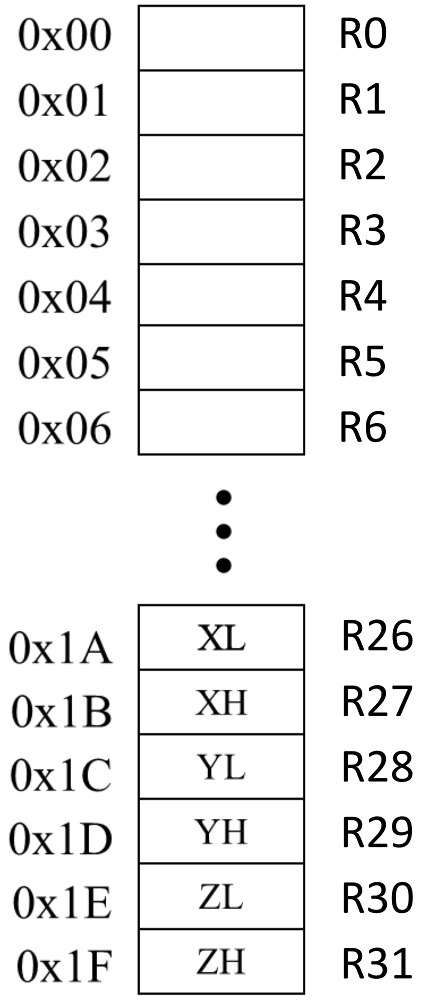


AVR Register: General Purpose

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

- Special characteristics of registers:
  - They can be used directly in assembler commands.
  - Operations with their content require only a single command word.
  - They are connected directly to the ALU.
  - Can be used as source and target for calculations.
  - R26-R31 can be combined, resulting 3 16-bit Registers (X, Y, and Z)

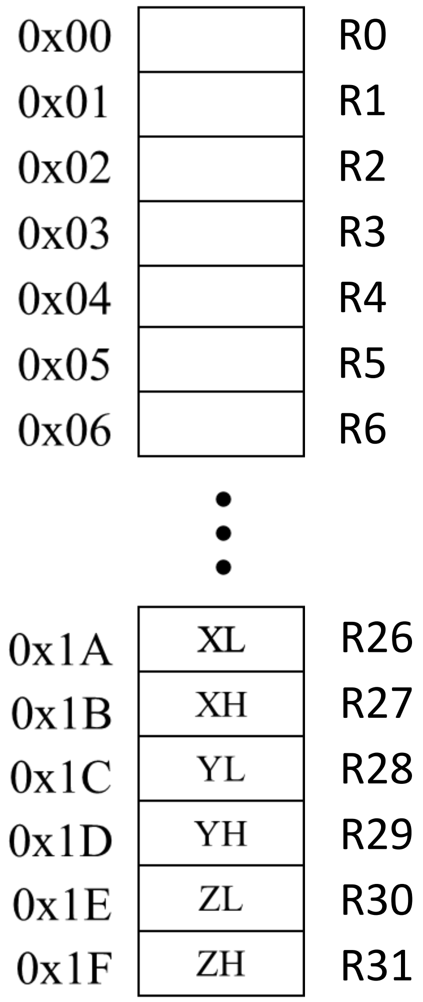
AVR Register



AVR has 32 register

R0, R1, R2, … R31

AVR Register



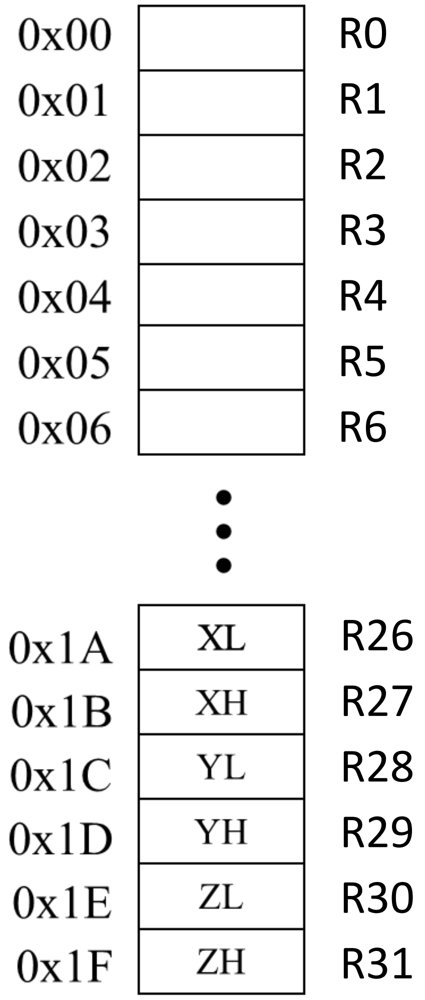
AVR memiliki 32 register

R0, R1, R2, … R31

Example 1:

```
    LDI R16,150
```

AVR Register



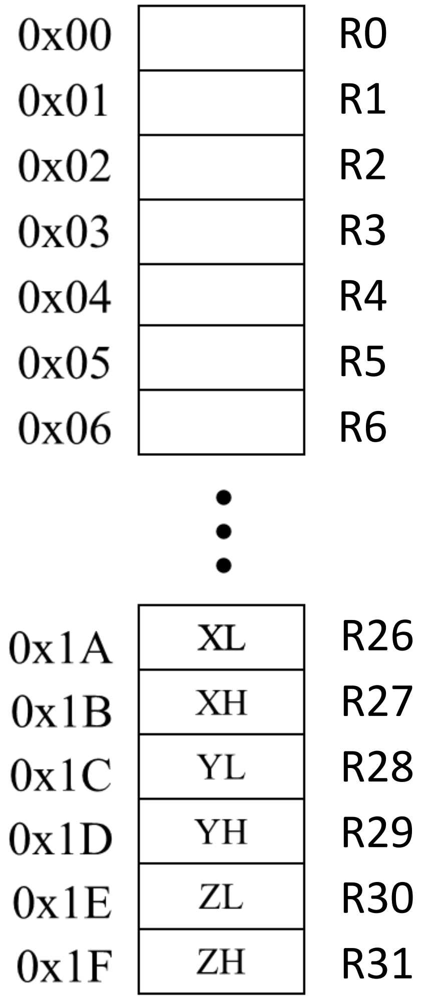
AVR memiliki 32 register

R0, R1, R2, … R31

Example 1:

```
    LDI R16,150
```

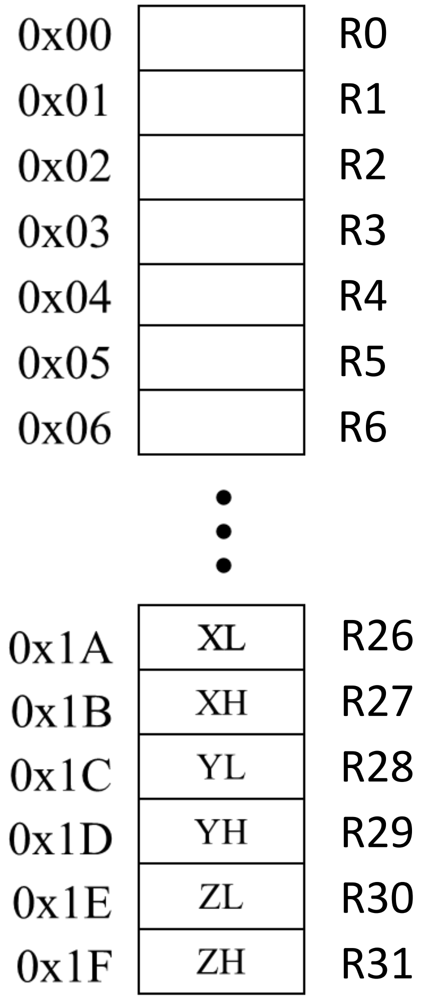
AVR Register



Example 3:

```
LDI R15,150
```

AVR Register

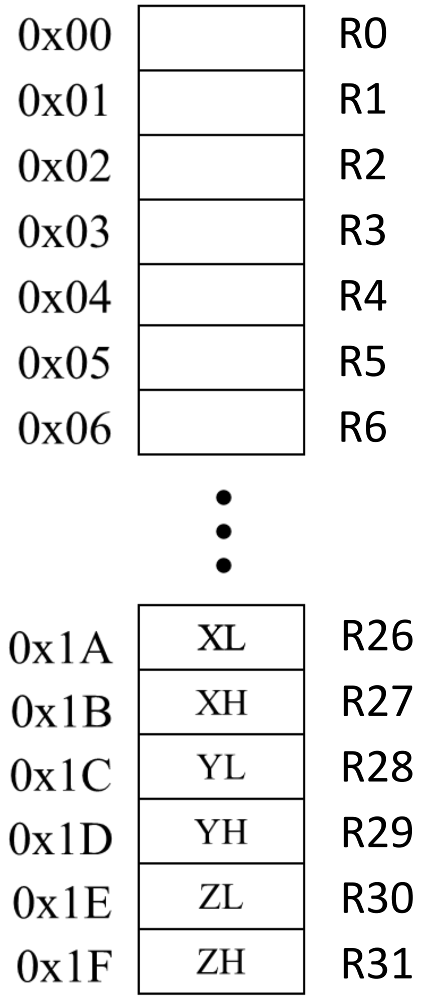


Example 3:

```
    LDI R15,150
```

$\longrightarrow$ **ERROR**

AVR Register



Contoh 3:

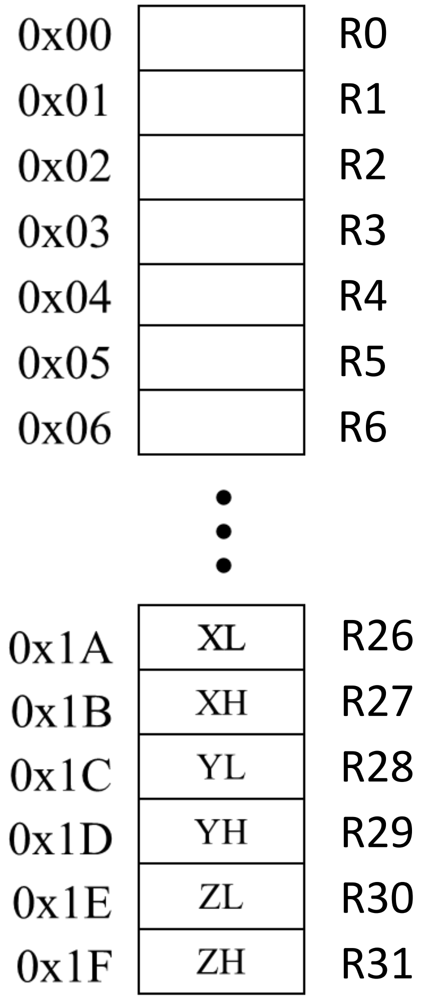
```
LDI R15,150
```

$\xrightarrow{\quad\quad}$ ERROR

Only R16-R31 that can be used by instructions with a constant (immediate) in it, such as:

- LDI Rx,K
- ANDI Rx, K
- CPI Rx, K
- SBCI
- SUBI

AVR Register



Example 2:

```
    LDI R16,150
    MOV R15, R16
```

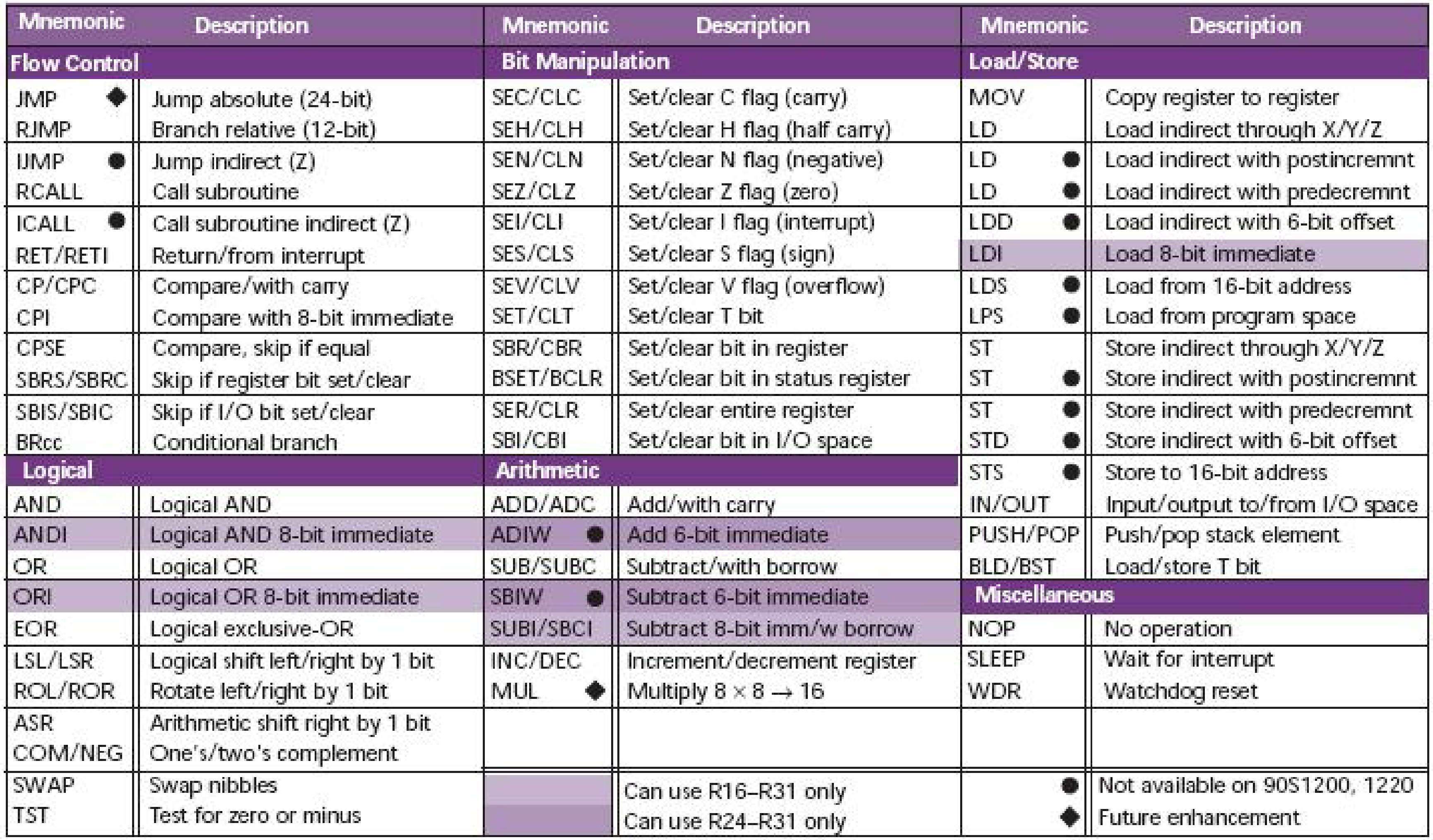
Recommended use of the registers

- Define names for registers with **.DEF** directives.
- For example: .def reg1=r17
- If you need **pointer** access, reserve R26 to R31 for that purpose.
  [Memory and Adderssing]
- 16-bit **counter** are best located in R25:R24. [Interrupt]
- If you need to have access to **single bits** within certain registers (e.g. for testing flags), use R16 to R23 for that purpose.
- If you need to read from the **program memory**, e.g. fixed tables, reserve Z (R31:R30) and R0 for that purpose.

AVR Instruction

- Can have 2 operands, 1 operand, or 0 operand
- Example:
  - Addition: ADD R1, R2 (2 Operands)
  - 2’s negation: NEG R5 (1 Operand)
  - Load Program Memory: LPM (0 Operand)

AVR



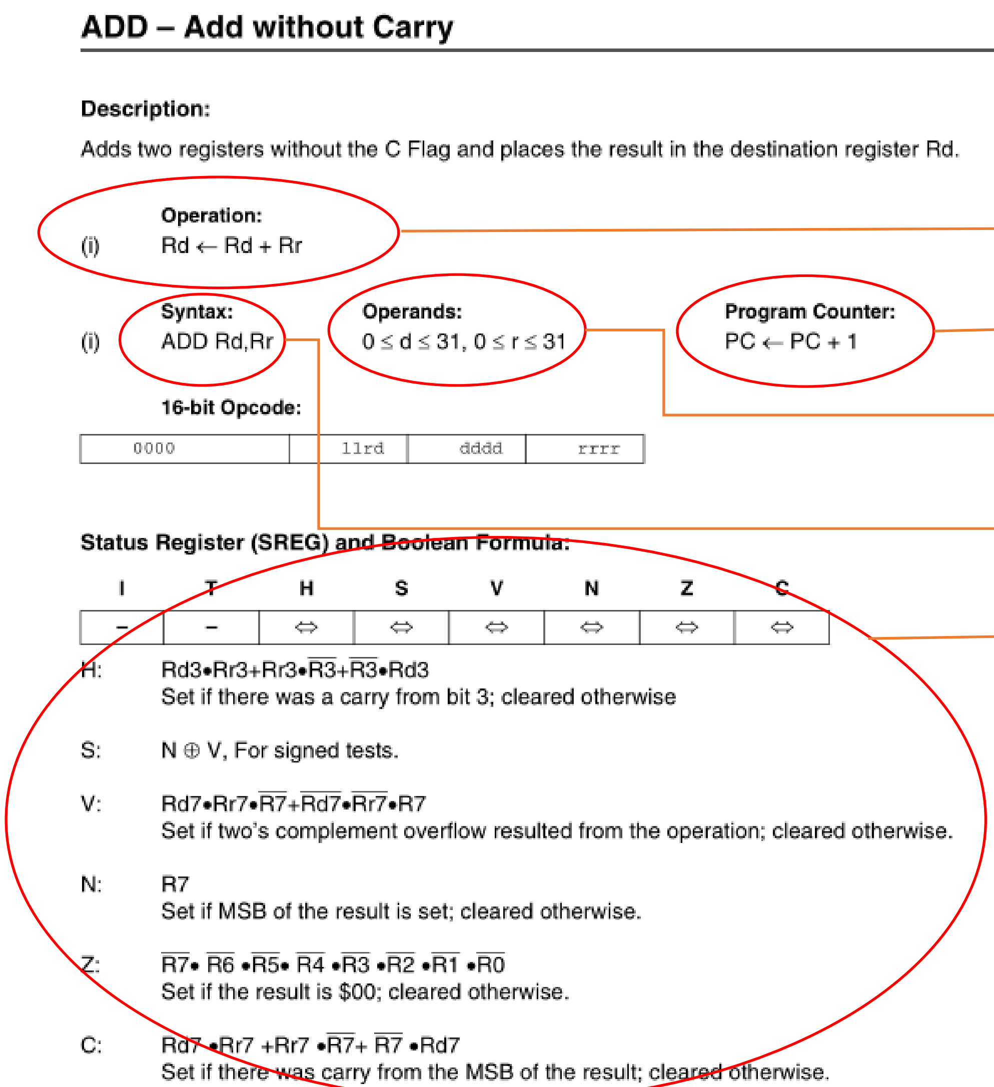
Mnemonic	Description	Mnemonic	Description	Mnemonic	Description
Flow Control		**Bit Manipulation**		**Load/Store**	
JMP ◆	Jump absolute (24-bit)	SEC/CLC	Set/clear C flag (carry)	MOV	Copy register to register
RJMP	Branch relative (12-bit)	SEH/CLH	Set/clear H flag (half carry)	LD	Load indirect through X/Y/Z
IJMP ●	Jump indirect (Z)	SEN/CLN	Set/clear N flag (negative)	LD ●	Load indirect with postincremnt
RCALL	Call subroutine	SEZ/CLZ	Set/clear Z flag (zero)	LD ●	Load indirect with predecremnt
ICALL ●	Call subroutine indirect (Z)	SEI/CLI	Set/clear I flag (interrupt)	LDD ●	Load indirect with 6-bit offset
RET/RETI	Return/from interrupt	SES/CLS	Set/clear S flag (sign)	LDI	Load 8-bit immediate
CP/CPC	Compare/with carry	SEV/CLV	Set/clear V flag (overflow)	LDS ●	Load from 16-bit address
CPI	Compare with 8-bit immediate	SET/CLT	Set/clear T bit	LPS ●	Load from program space
CPSE	Compare, skip if equal	SBR/CBR	Set/clear bit in register	ST	Store indirect through X/Y/Z
SBRS/SBRC	Skip if register bit set/clear	BSET/BCLR	Set/clear bit in status register	ST ●	Store indirect with postincremnt
SBIS/SBIC	Skip if I/O bit set/clear	SER/CLR	Set/clear entire register	ST ●	Store indirect with predecremnt
BRcc	Conditional branch	SBI/CBI	Set/clear bit in I/O space	STD ●	Store indirect with 6-bit offset
Logical		**Arithmetic**		STS ●	Store to 16-bit address
AND	Logical AND	ADD/ADC	Add/with carry	IN/OUT	Input/output to/from I/O space
ANDI	Logical AND 8-bit immediate	ADIW ●	Add 6-bit immediate	PUSH/POP	Push/pop stack element
OR	Logical OR	SUB/SUBC	Subtract/with borrow	BLD/BST	Load/store T bit
ORI	Logical OR 8-bit immediate	SBIW ●	Subtract 6-bit immediate	**Miscellaneous**	
EOR	Logical exclusive-OR	SUBI/SBCI	Subtract 8-bit imm/w borrow	NOP	No operation
LSL/LSR	Logical shift left/right by 1 bit	INC/DEC	Increment/decrement register	SLEEP	Wait for interrupt
ROL/ROR	Rotate left/right by 1 bit	MUL ◆	Multiply 8 × 8 → 16	WDR	Watchdog reset
ASR	Arithmetic shift right by 1 bit				
COM/NEG	One's/two's complement				
SWAP	Swap nibbles	▨	Can use R16–R31 only	●	Not available on 90S1200, 1220
TST	Test for zero or minus	▩	Can use R24–R31 only	◆	Future enhancement

AVR Operation Code (Opcode)

- Instruction formats of AVR $\neq$ MIPS
- Every AVR instruction has a unique opcode

AVR Instruction

- Refer to “**AVR Instruction**” documentation



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: ADD Rd,Rr — **Operands:** $0 \le d \le 31, 0 \le r \le 31$ — **Program Counter:** 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: $N \oplus V$, 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 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: $Rd7 \bullet Rr7 + Rr7 \bullet \overline{R7} + \overline{R7} \bullet Rd7$
Set if there was carry from the MSB of the result; cleared otherwise.

Add operation

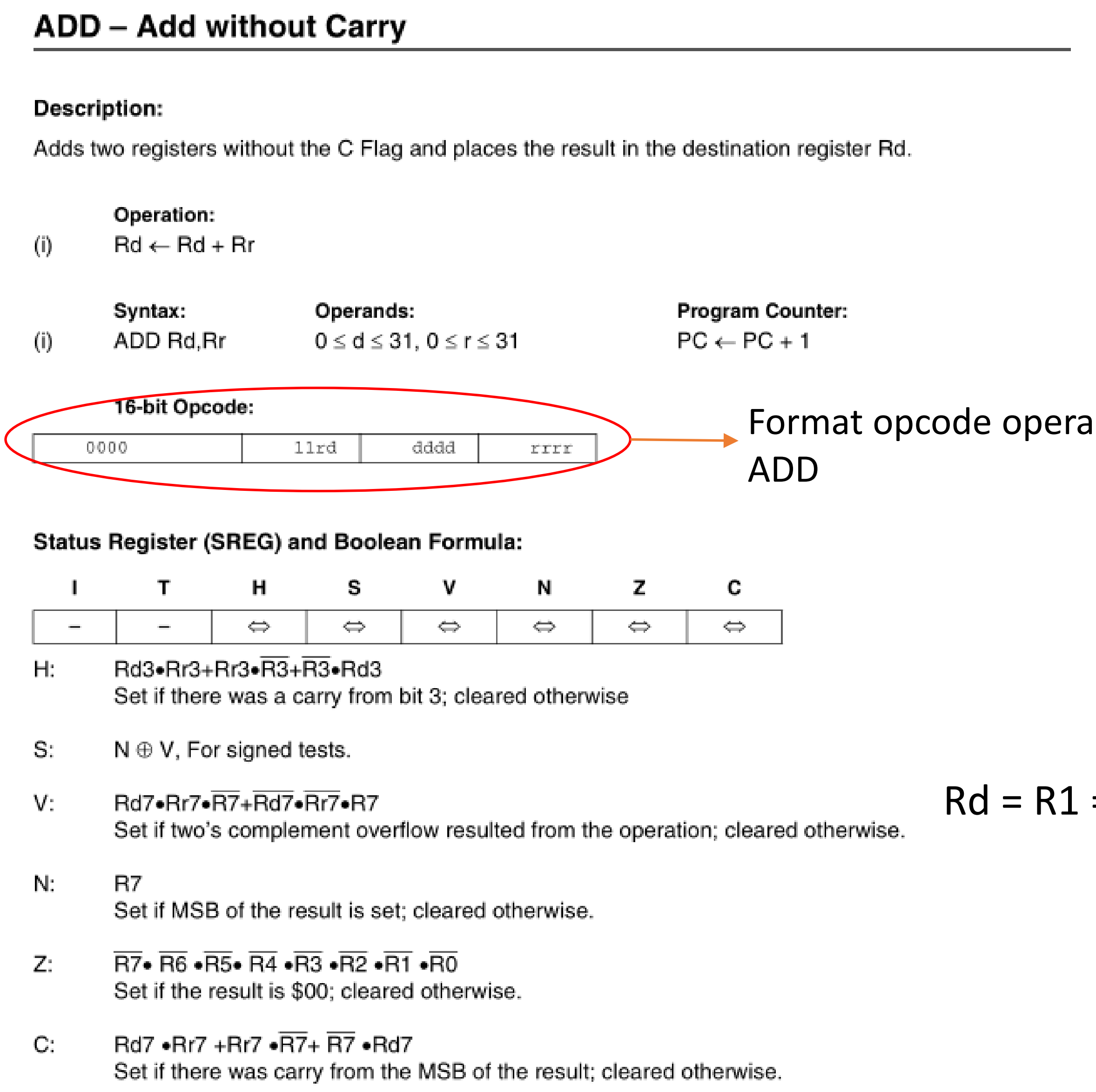
Impact on the Program Counter

Register(s) used by the instruction

The syntax

Impact on the **Status Register**

Contoh: AVR Opcode



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

Format opcode operasi ADD

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: $N \oplus V$, 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 otherwise.

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

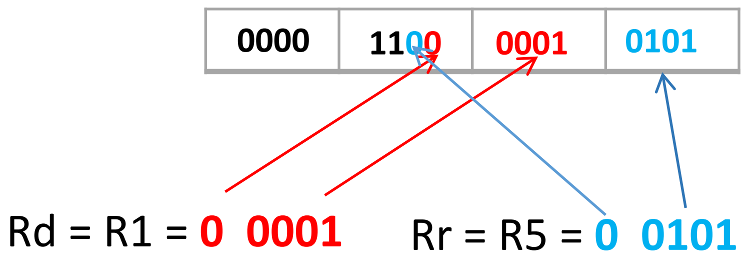
Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result is \$00; cleared otherwise.

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

- Contoh:

ADD R1, R5

Opcode:



Sample Program

```
.include “m8515def.inc"


forever:
    ldi     R26, 04
    ldi     R27, $02
    ld      R16, X
    andi    R16, $63
    st      X, R16

    rjmp    forever
```

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

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
```

LDI – Load Immediate

$\boxed{\text{ldi} \quad \text{R26, 04}}$

Rd = 26 = 1 1010

K = 04 = 0000 0100

$\boxed{1110 \;\; 0000 \;\; 1010 \;\; 0100 = \text{E0A4}}$

Description:

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

Operation:

(i) Rd ← K

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

16-bit Opcode:

1110	KKKK	dddd	KKKK

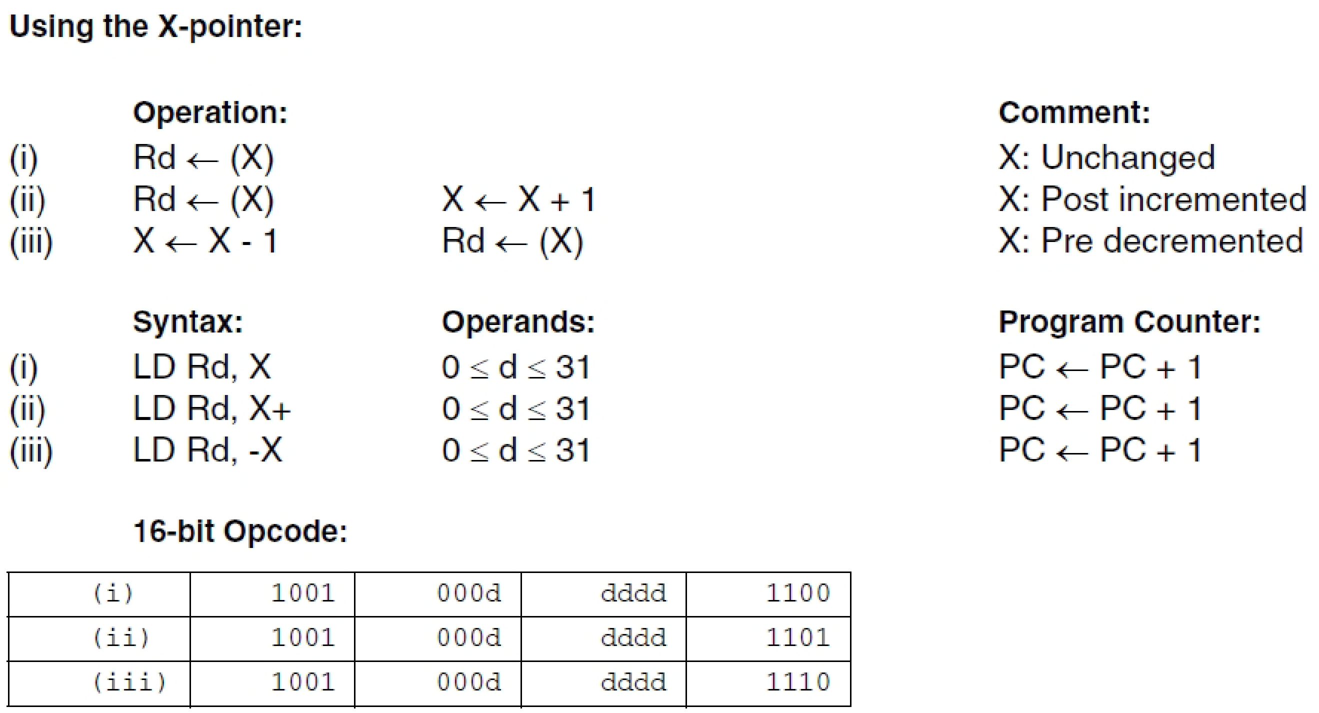
Status Register (SREG) and Boolean Formula:

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
```

LD – Load Indirect from Data Space to Register using Index X



Using the X-pointer:

	Operation:		**Comment:**
(i)	Rd ← (X)		X: Unchanged
(ii)	Rd ← (X)	X ← X + 1	X: Post incremented
(iii)	X ← X - 1	Rd ← (X)	X: Pre decremented

	Syntax:	**Operands:**	**Program Counter:**
(i)	LD Rd, X	$0 \leq d \leq 31$	PC ← PC + 1
(ii)	LD Rd, X+	$0 \leq d \leq 31$	PC ← PC + 1
(iii)	LD Rd, -X	$0 \leq d \leq 31$	PC ← PC + 1

16-bit Opcode:

(i)	1001	000d	dddd	1100
(ii)	1001	000d	dddd	1101
(iii)	1001	000d	dddd	1110

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:

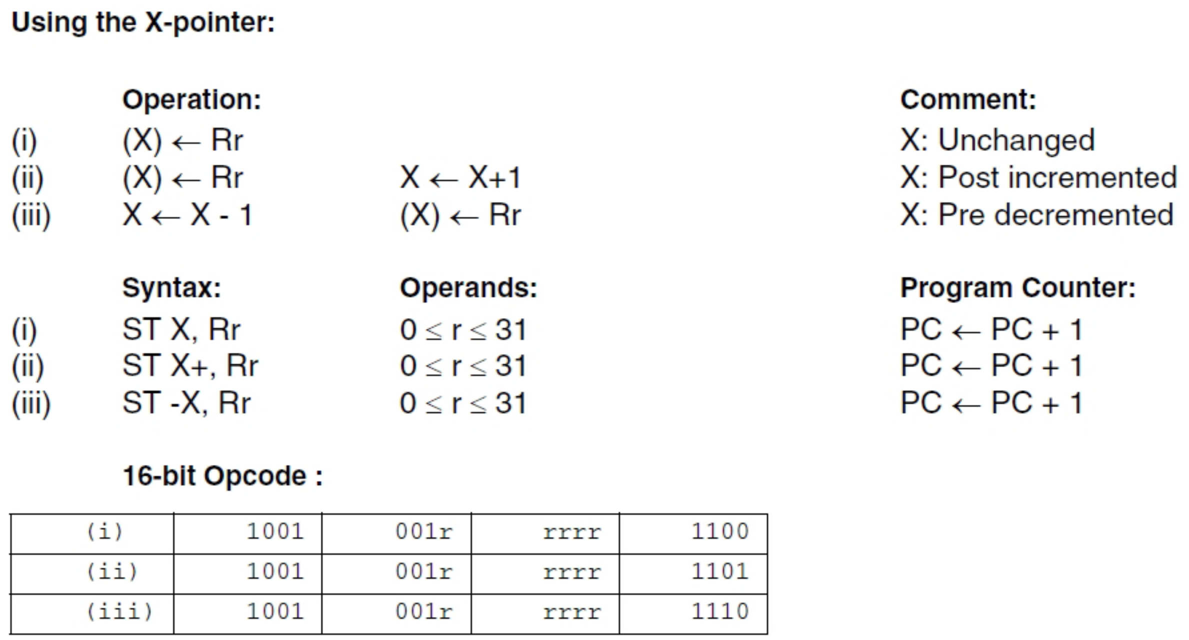
(i) $Rd \leftarrow Rd \bullet K$

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

16-bit Opcode:

0111	KKKK	dddd	KKKK

ST – Store Indirect From Register to Data Space using Index X



Using the X-pointer:

	Operation:		Comment:
(i)	$(X) \leftarrow Rr$		X: Unchanged
(ii)	$(X) \leftarrow Rr$	$X \leftarrow X+1$	X: Post incremented
(iii)	$X \leftarrow X - 1$	$(X) \leftarrow Rr$	X: Pre decremented

	Syntax:	Operands:	Program Counter:
(i)	ST X, Rr	$0 \leq r \leq 31$	$PC \leftarrow PC + 1$
(ii)	ST X+, Rr	$0 \leq r \leq 31$	$PC \leftarrow PC + 1$
(iii)	ST -X, Rr	$0 \leq r \leq 31$	$PC \leftarrow PC + 1$

16-bit Opcode :

(i)	1001	001r	rrrr	1100
(ii)	1001	001r	rrrr	1101
(iii)	1001	001r	rrrr	1110

RJMP – Relative Jump

Description:

Relative jump to an address within PC - 2K +1 and PC + 2K (words). For AVR microcontrollers with Program memory not exceeding 4K words (8K bytes) this instruction can address the entire memory from every address location. See also JMP.

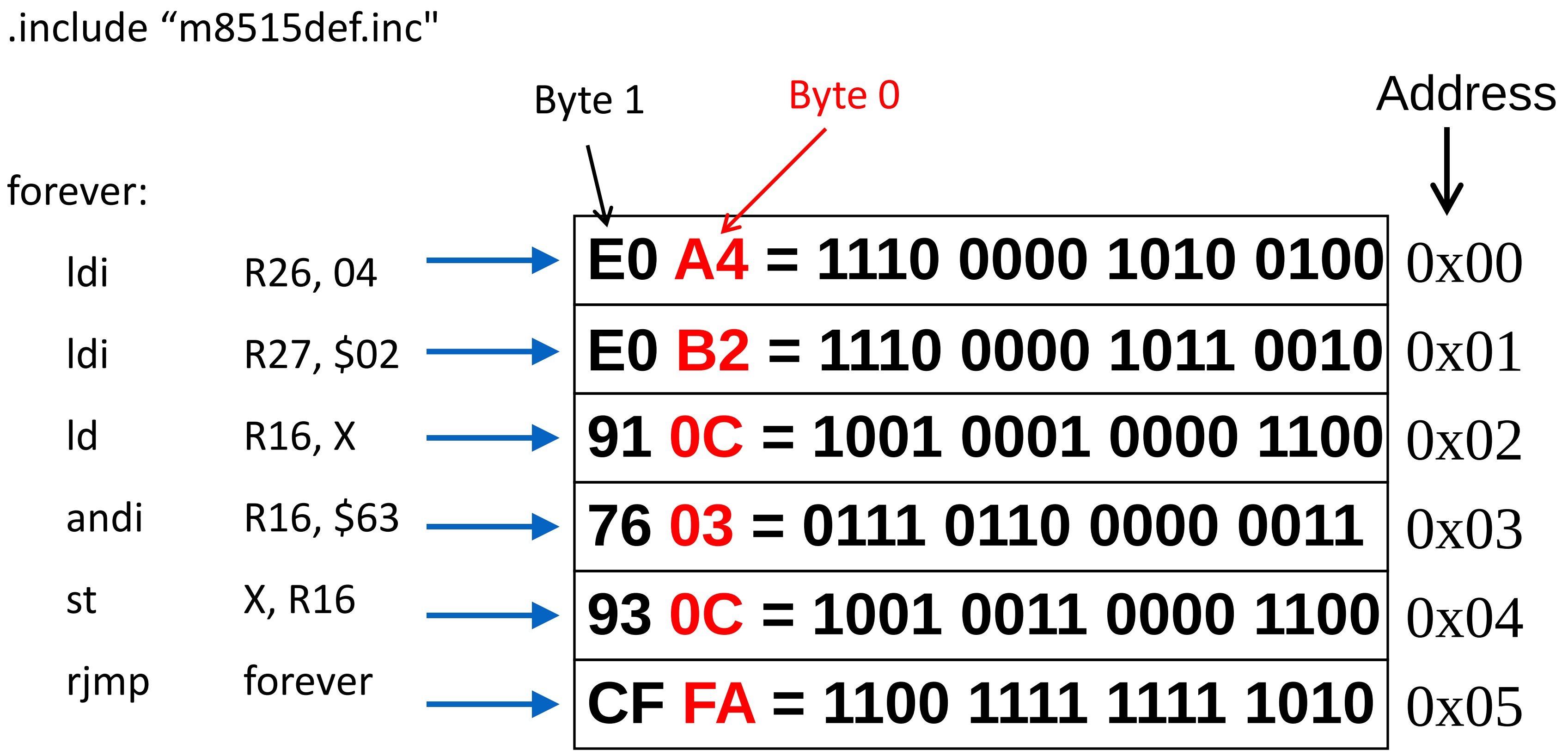
	Operation:
(i)	$PC \leftarrow PC + k + 1$

	Syntax:	**Operands:**	**Program Counter:**	**Stack**
(i)	RJMP k	$-2K \leq k < 2K$	$PC \leftarrow PC + k + 1$	Unchanged

16-bit Opcode:

1100	kkkk	kkkk	kkkk

Sample Program

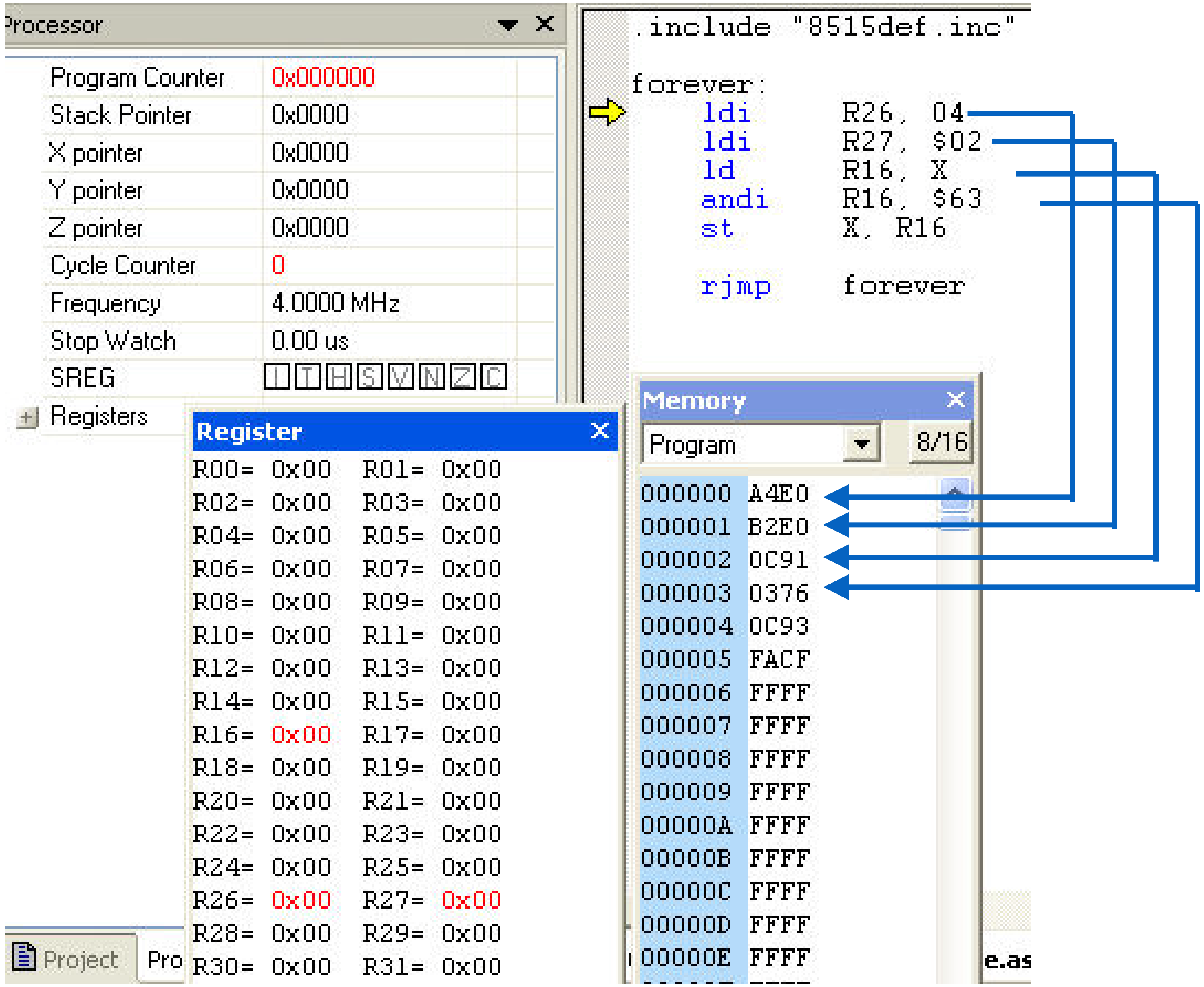


```
.include “m8515def.inc"

forever:
    ldi     R26, 04
    ldi     R27, $02
    ld      R16, X
    andi    R16, $63
    st      X, R16
    rjmp    forever
```

E0 A4 = 1110 0000 1010 0100	0x00
E0 B2 = 1110 0000 1011 0010	0x01
91 0C = 1001 0001 0000 1100	0x02
76 03 = 0111 0110 0000 0011	0x03
93 0C = 1001 0011 0000 1100	0x04
CF FA = 1100 1111 1111 1010	0x05

AVR Assembler



Processor	
Program Counter	0x000000
Stack Pointer	0x0000
X pointer	0x0000
Y pointer	0x0000
Z pointer	0x0000
Cycle Counter	0
Frequency	4.0000 MHz
Stop Watch	0.00 us
SREG	I T H S V N Z C
Registers	

```
.include "8515def.inc"

forever:
        ldi     R26, 04
        ldi     R27, $02
        ld      R16, X
        andi    R16, $63
        st      X, R16

        rjmp    forever
```

Register

```
R00= 0x00  R01= 0x00
R02= 0x00  R03= 0x00
R04= 0x00  R05= 0x00
R06= 0x00  R07= 0x00
R08= 0x00  R09= 0x00
R10= 0x00  R11= 0x00
R12= 0x00  R13= 0x00
R14= 0x00  R15= 0x00
R16= 0x00  R17= 0x00
R18= 0x00  R19= 0x00
R20= 0x00  R21= 0x00
R22= 0x00  R23= 0x00
R24= 0x00  R25= 0x00
R26= 0x00  R27= 0x00
R28= 0x00  R29= 0x00
R30= 0x00  R31= 0x00
```

Memory — Program — 8/16

```
000000 A4E0
000001 B2E0
000002 0C91
000003 0376
000004 0C93
000005 FACF
000006 FFFF
000007 FFFF
000008 FFFF
000009 FFFF
00000A FFFF
00000B FFFF
00000C FFFF
00000D FFFF
00000E FFFF
```