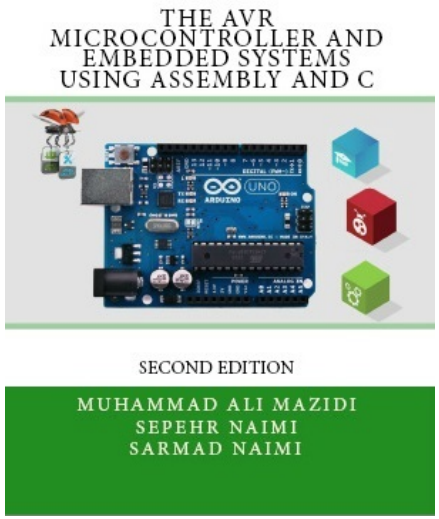
Introduction to Assembly
Chapter 2

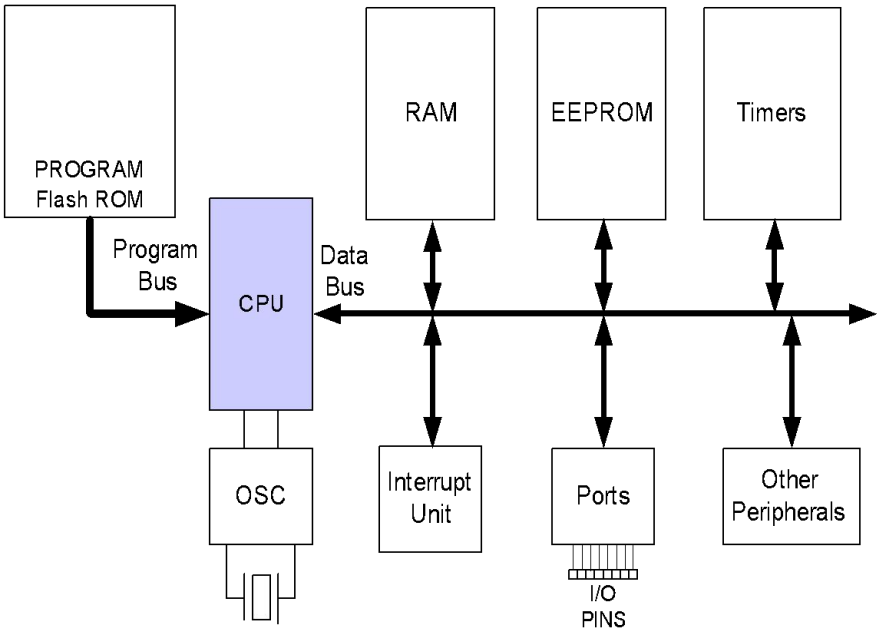


Sepehr Naimi

www.NicerLand.com
www.MicroDigitalEd.com

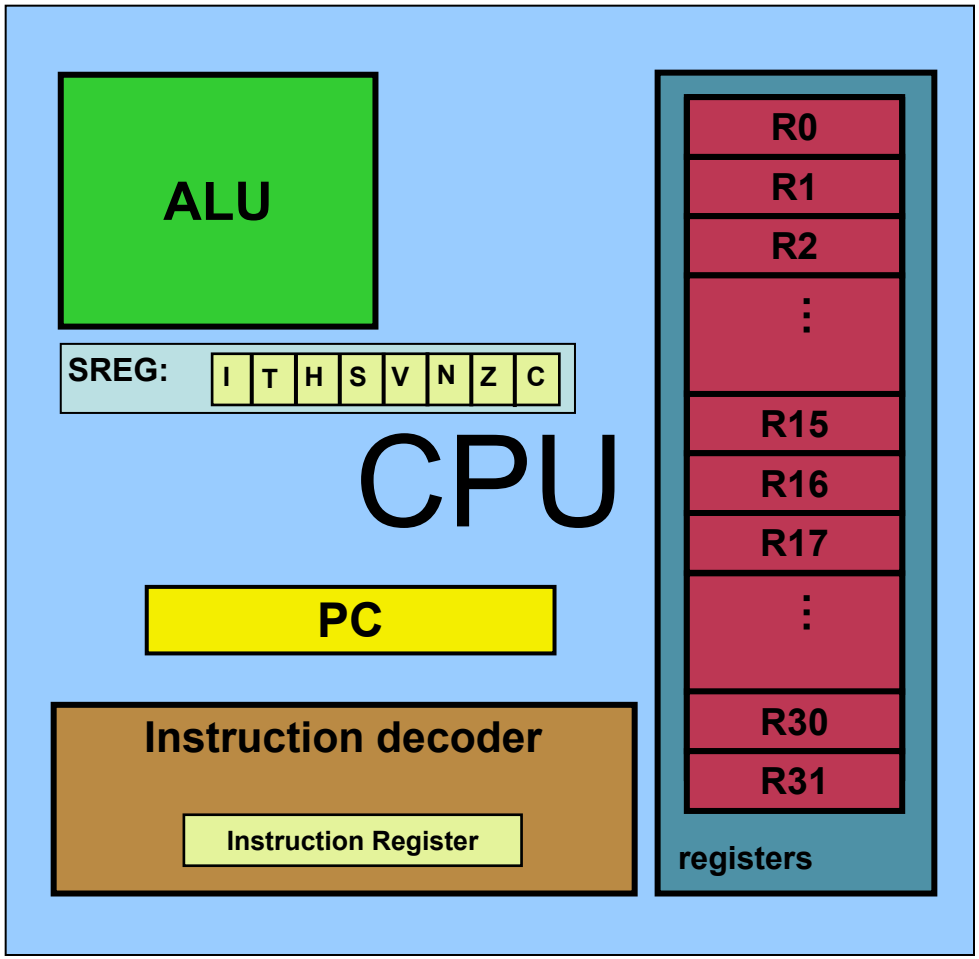
Topics

- AVR's CPU
  - Its architecture
  - Some simple programs
- Data Memory access
- Program memory
- RISC architecture



AVR's CPU

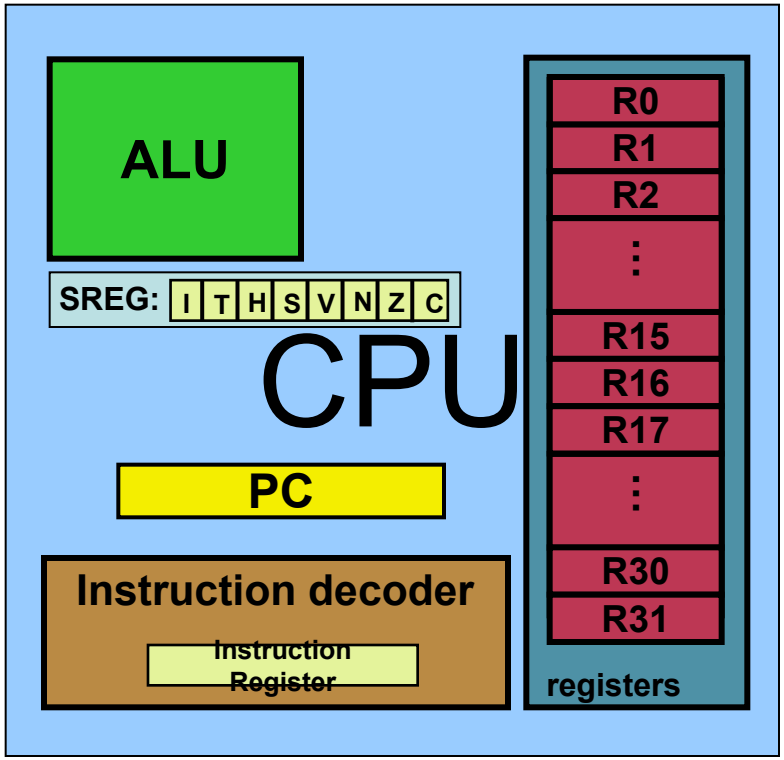
- AVR's CPU
  - ALU
  - 32 General Purpose registers (R0 to R31)
  - PC register
  - Instruction decoder



Some simple instructions
1. Loading values into the general purpose registers

LDI (**Load Immediate**)

- LDI Rd, k
  - Its equivalent in high level languages:
    Rd = k
- Example:
  - LDI R16,53
    - R16 = 53
  - LDI R19,$27
  - LDI R23,0x27
    - R23 = 0x27
  - LDI R23,0b11101100



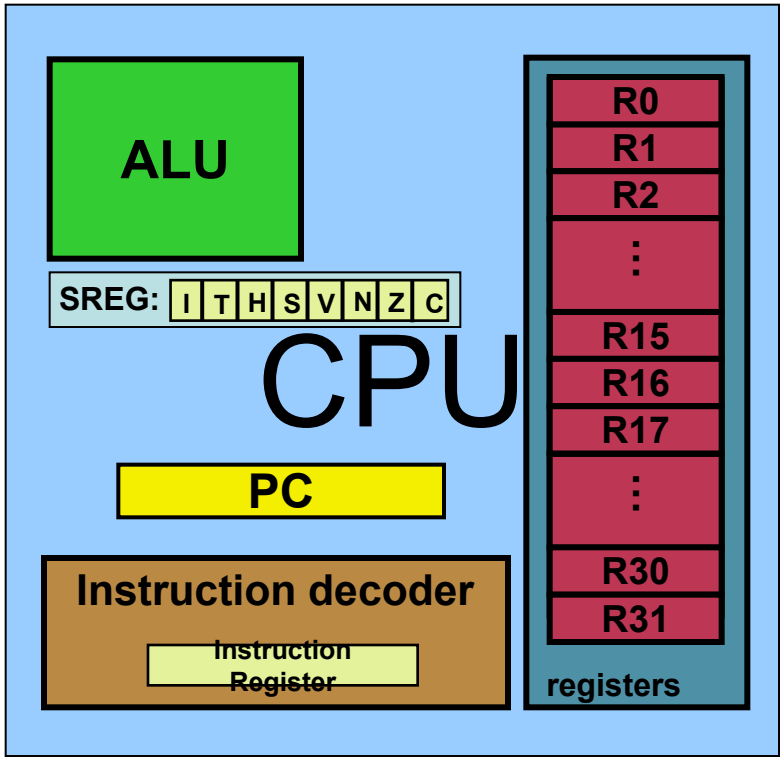
Some simple instructions

2. Arithmetic calculation

- There are some instructions for doing Arithmetic and logic operations; such as:

  ADD, SUB, MUL, AND, etc.

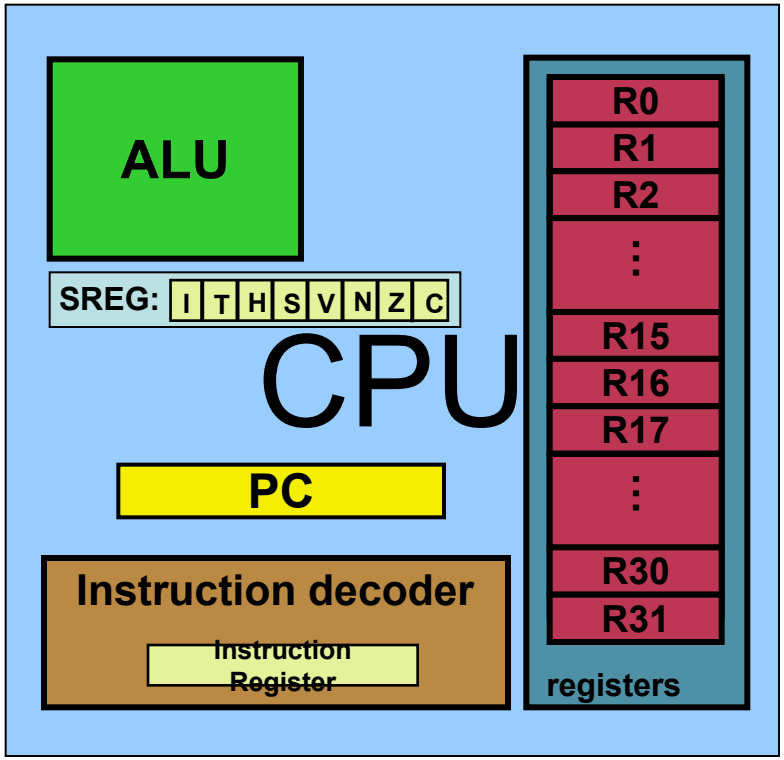
- ADD Rd,Rs
  - Rd = Rd + Rs
  - Example:
  - ADD R25, R9
    - R25 = R25 + R9
  - ADD R17,R30
    - R17 = R17 + R30



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

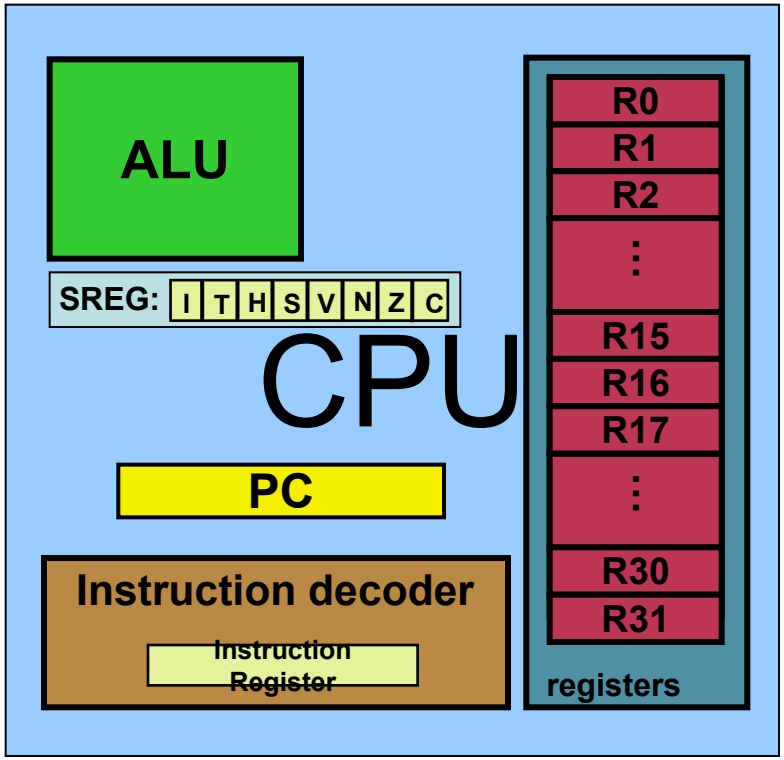
```
LDI R16, 19     ;R16 = 19
LDI R20, 95     ;R20 = 95
LDI R21, 5      ;R21 = 5
ADD R16, R20    ;R16 = R16 + R20
ADD R16, R21    ;R16 = R16 + R21
```

```
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
```

Some simple instructions

2. Arithmetic calculation

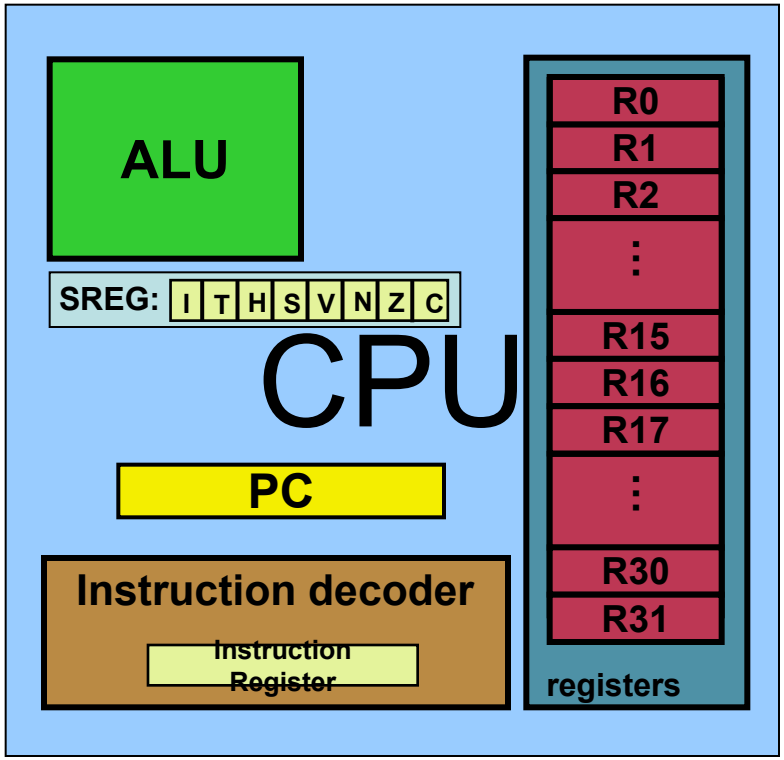
- SUB Rd,Rs
  - Rd = Rd - Rs
- Example:
  - SUB R25, R9
    - R25 = R25 - R9
  - SUB R17,R30
    - R17 = R17 - R30



Some simple instructions

2. Arithmetic calculation

- INC Rd
  - Rd = Rd + 1
- Example:
  - INC R25
    - R25 = R25 + 1
- DEC Rd
  - Rd = Rd - 1
- Example:
  - DEC R23
    - R23 = R23 - 1

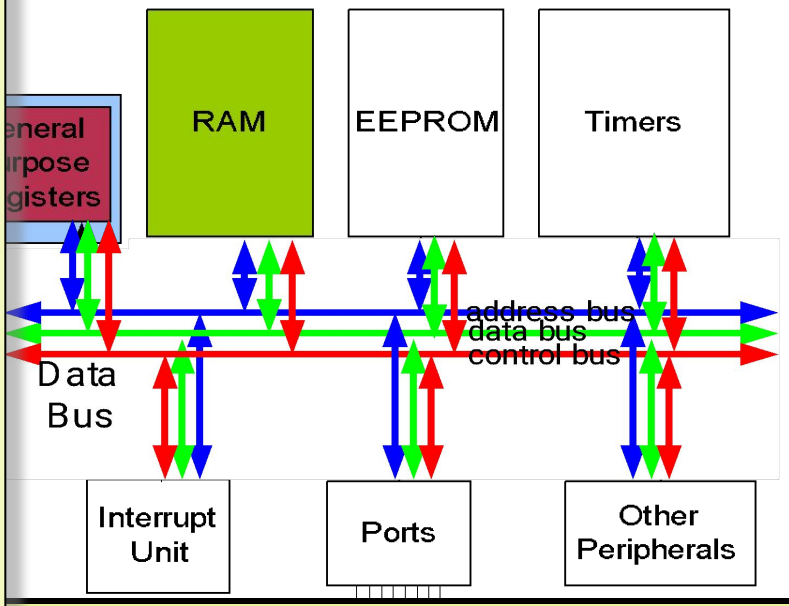


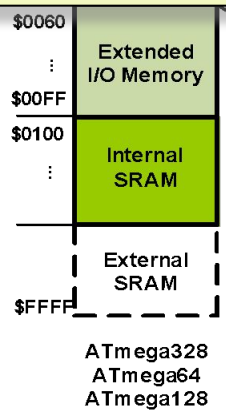
Data Address Space

Address Mem.	Address I/O	Name
$20	$00	-
$21	$01	-
$22	$02	-
$23	$03	PINB
$24	$04	DDRB
$25	$05	PORTB
$26	$06	PINC
$27	$07	DDRC
$28	$08	PORTC
$29	$09	PIND
$2A	$0A	DDRD
$2B	$0B	PORTD
$2C	$0C	-
$2D	$0D	-
$2E	$0E	-
$2F	$0F	-
$30	$10	-
$31	$11	-
$32	$12	-
$33	$13	-
$34	$14	-
$35	$15	TIFR0

Address Mem.	Address I/O	Name
$36	$16	TIFR1
$37	$17	TIFR2
$38	$18	-
$39	$19	-
$3A	$1A	-
$3B	$1B	PCIFR
$3C	$1C	EIFR
$3D	$1D	EIMSK
$3E	$1E	GPIOR0
$3F	$1F	EECR
$40	$20	EEDR
$41	$21	EEARL
$42	$22	EEARH
$43	$23	GTCCR
$44	$24	TCCR0A
$45	$25	TCCR0B
$46	$26	TCNT0
$47	$27	OCR0A
$48	$28	OCR0B
$49	$29	-
$4A	$2A	GPIOR1
$4A	$2A	GPIOR2

Address Mem.	Address I/O	Name
$4C	$2C	SPCR0
$4D	$2D	SPSR0
$4E	$2E	SPDR0
$4F	$2F	-
$50	$30	ACSR
$51	$31	DWDR
$52	$32	-
$53	$33	SMCR
$54	$34	MCUSR
$55	$35	MCUCR
$56	$36	-
$57	$37	SPMCSR
$58	$38	-
$59	$39	-
$5A	$3A	-
$5B	$3B	-
$5C	$3C	-
$5D	$3D	SPL
$5E	$3E	SPH
$5F	$3F	SREG





Example: What does t

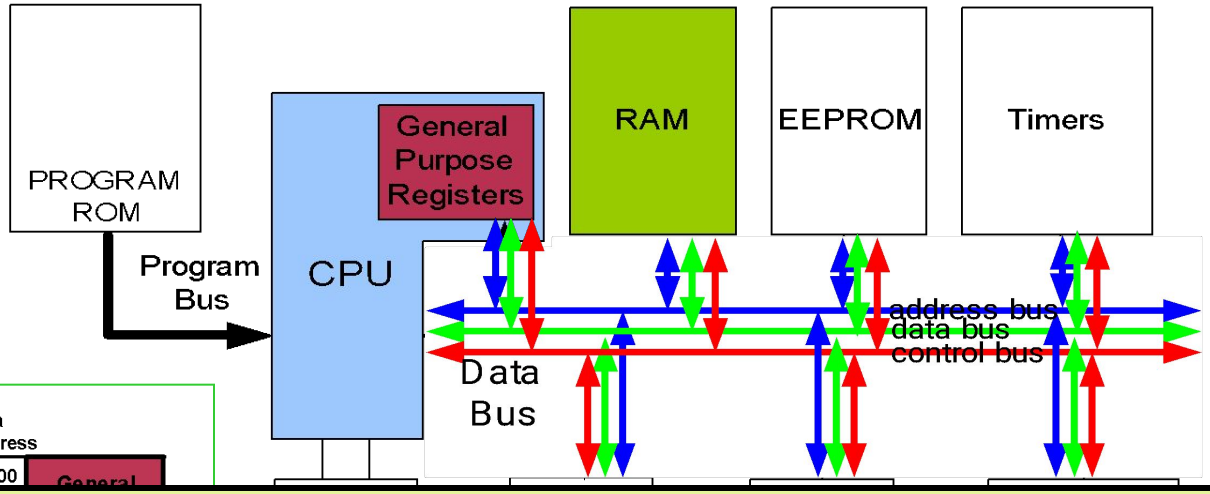
```
LDS   R20,2
```

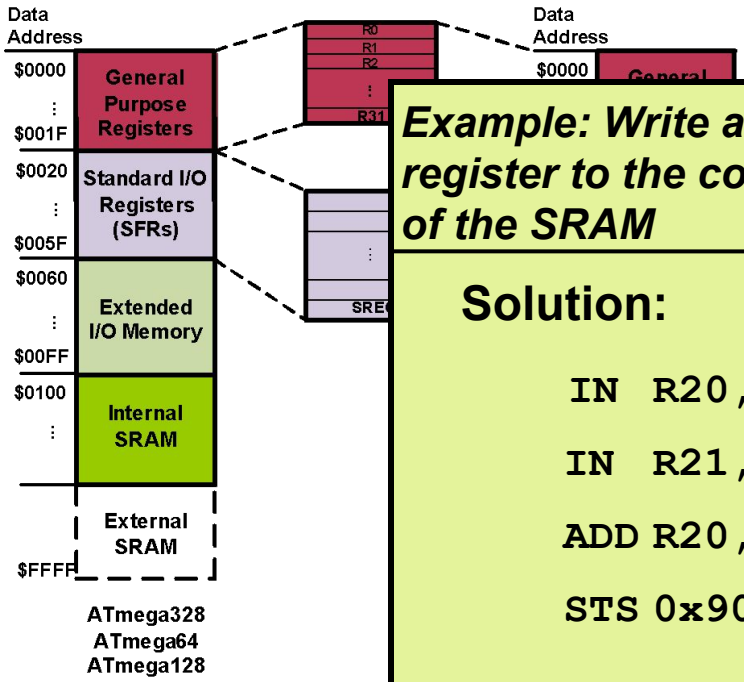
Answer:

It copies the conte

Example: Store 0x53 into the SPH register. The address of SPH is 0x5E

Data Address Space





Example: Write a program that adds the contents of the PINC I/O register to the contents of PIND and stores the result in location 0x90 of the SRAM

Solution:

```
IN  R20,PINC   ;R20 = PINC
IN  R21,PIND   ;R21 = PIND
ADD R20,R21    ;R20 = R20 + R21
STS 0x90,R20   ;[0x90] = R20
```

Machine Language

- ADD R0,R1

000011 00 0000 0001
opcode *operand*

- LDI R16, 2
  LDI R17, 3
  ADD R16, R17

1110 0000 0000 0010
1110 0000 0001 0011
0000 1111 0000 0001

Status Register (SREG)

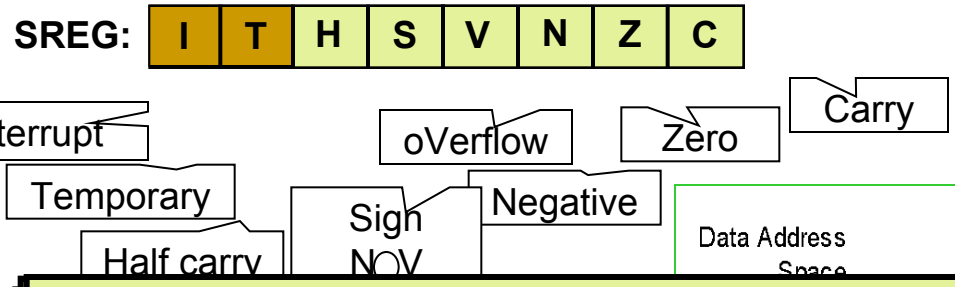


Table 2-5: AVR Branch (Jump) Instructions Using Flag Bits	
Instruction	**Action**
BRLO	Branch if C = 1
BRSH	Branch if C = 0
BREQ	Branch if Z = 1
BRNE	Branch if Z = 0
BRMI	Branch if N = 1
BRPL	Branch if N = 0
BRVS	Branch if V = 1
BRVC	Branch if V = 0

Example: Show the status of the C, H, and Z flags afte
subtraction of 0x9C from 0x9C in the following instru

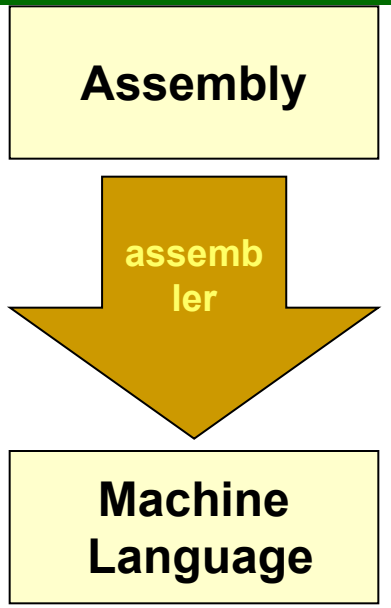
```
LDI R20, 0x9C
LDI R21, 0x9C
SUB R20, R21    ;subtract R21 from R20
```

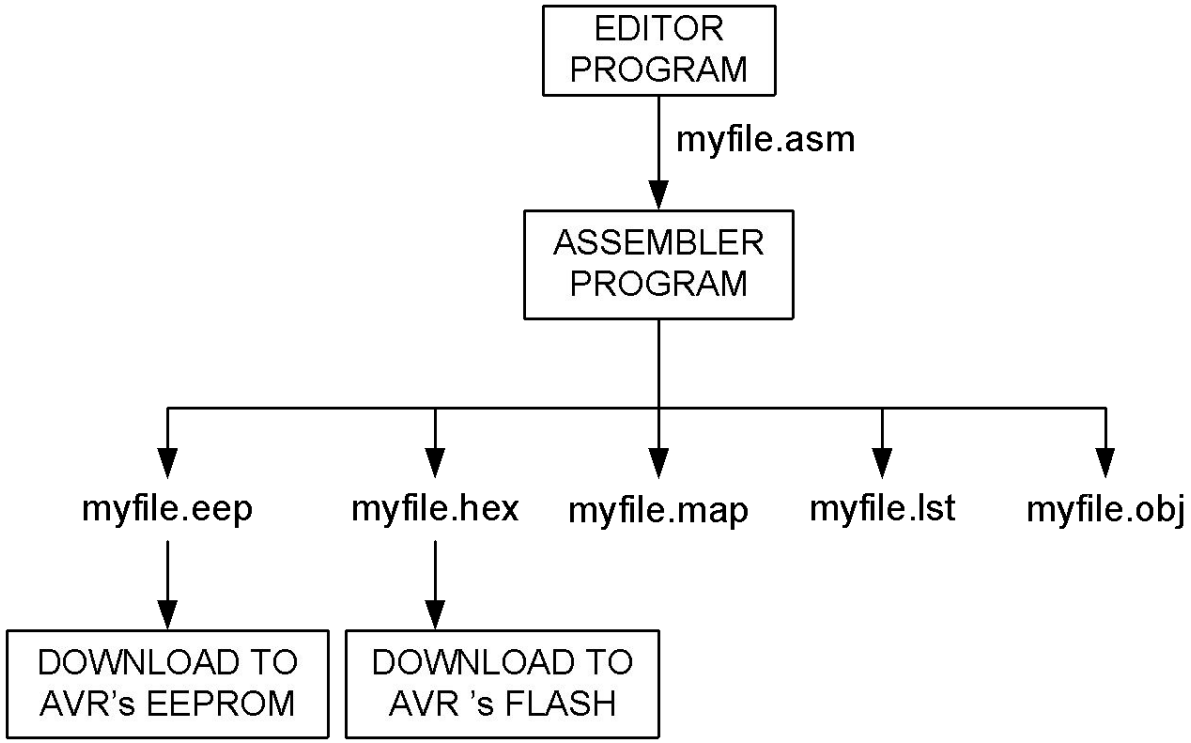
Solution:

$9C 1001 1100
- $9C 1001 1100
$00 0000 0000 R20 = $00

C = 0 because R21 is not bigger than R20 and there is no borrow from D8 bit.
Z = 1 because the R20 is zero after the subtraction.
H = 0 because there is no borrow from D4 to D3.

Assembler


Assembly
assemb
ler
Machine
Language


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

Assembler Directives

.EQU and .SET

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

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

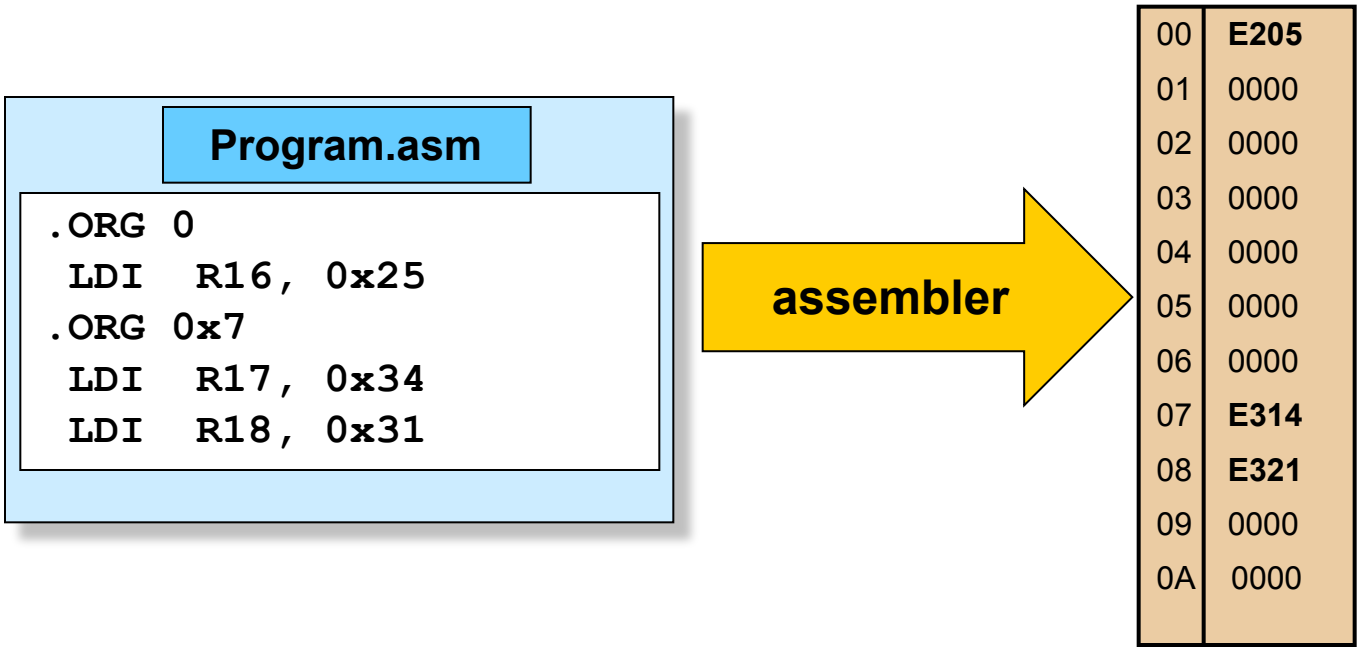
- .SET *name* = *value*
  - *Example:*

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

Assembler Directives

.ORG

- .ORG *address*



Assembler Directives

.INCLUDE

- .INCLUDE "*filename.ext*"

Table 2-6: Some of the Common AVRs and Their Include Files

ATM	
ATme	.inc
ATme	.inc
ATme	f.inc
ATme	f.inc
ATme	f.inc
ATme	

M328def.inc

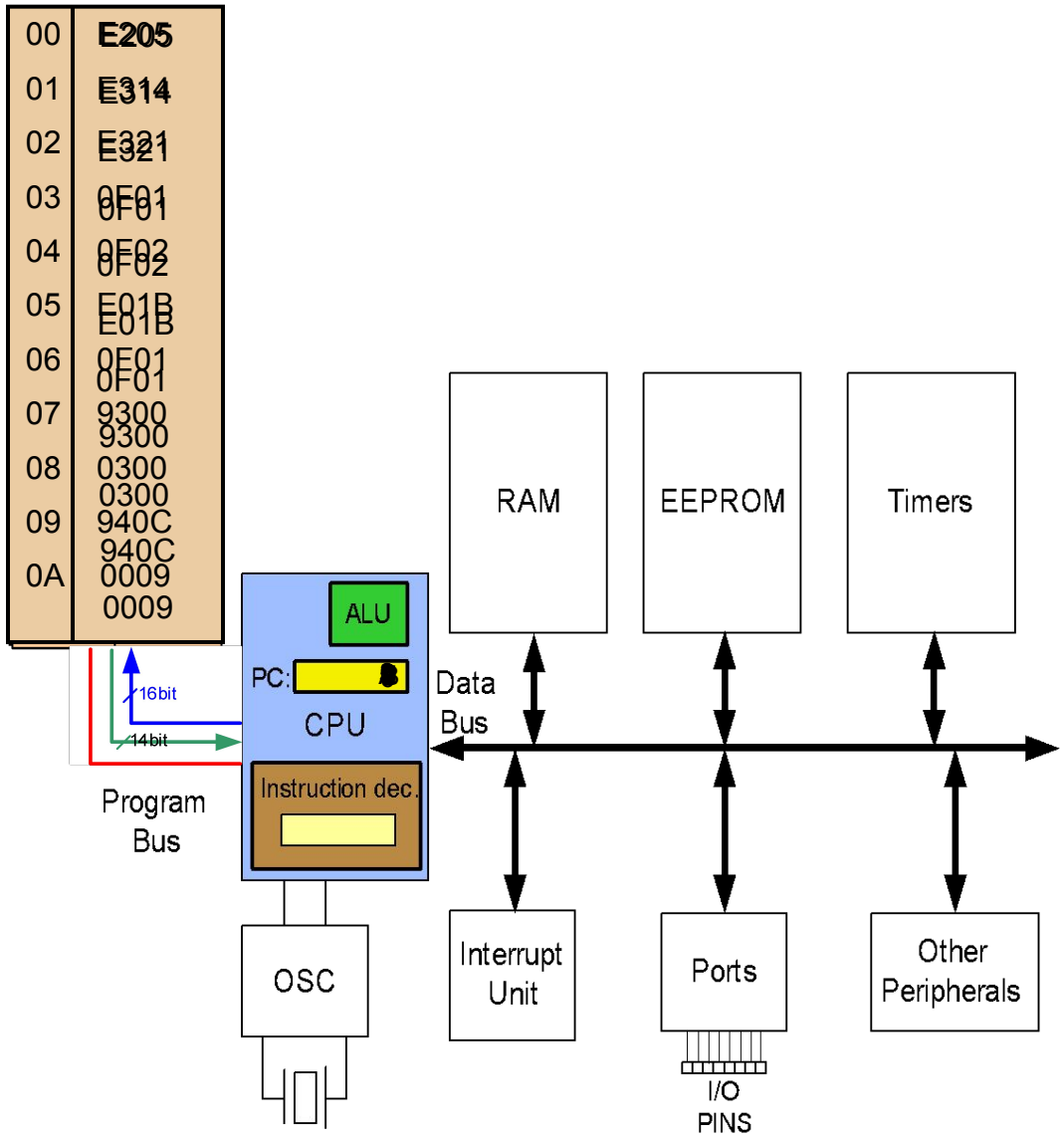
```
.equ    SREG    = 0x3f
.equ    SPL = 0x3d
.equ    SPH = 0x3e
....
```

Program.asm

```
    LDI    R20, 10
    OUT    SPL, R20
```

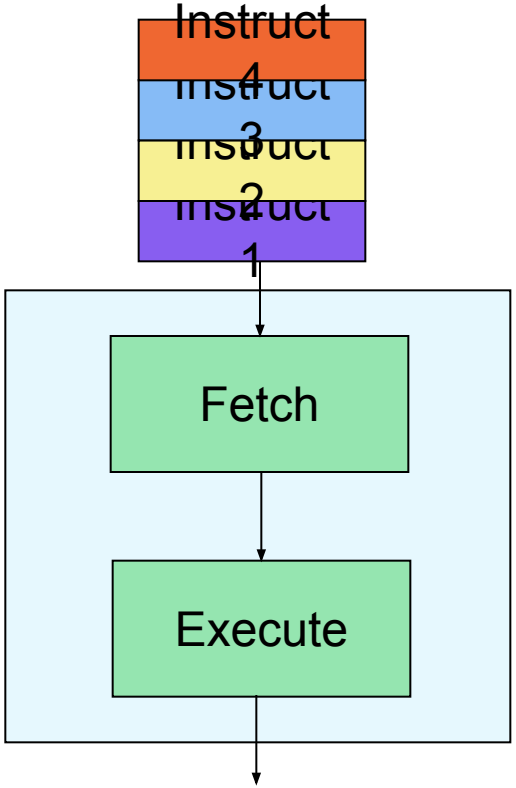
Flash memory and PC register

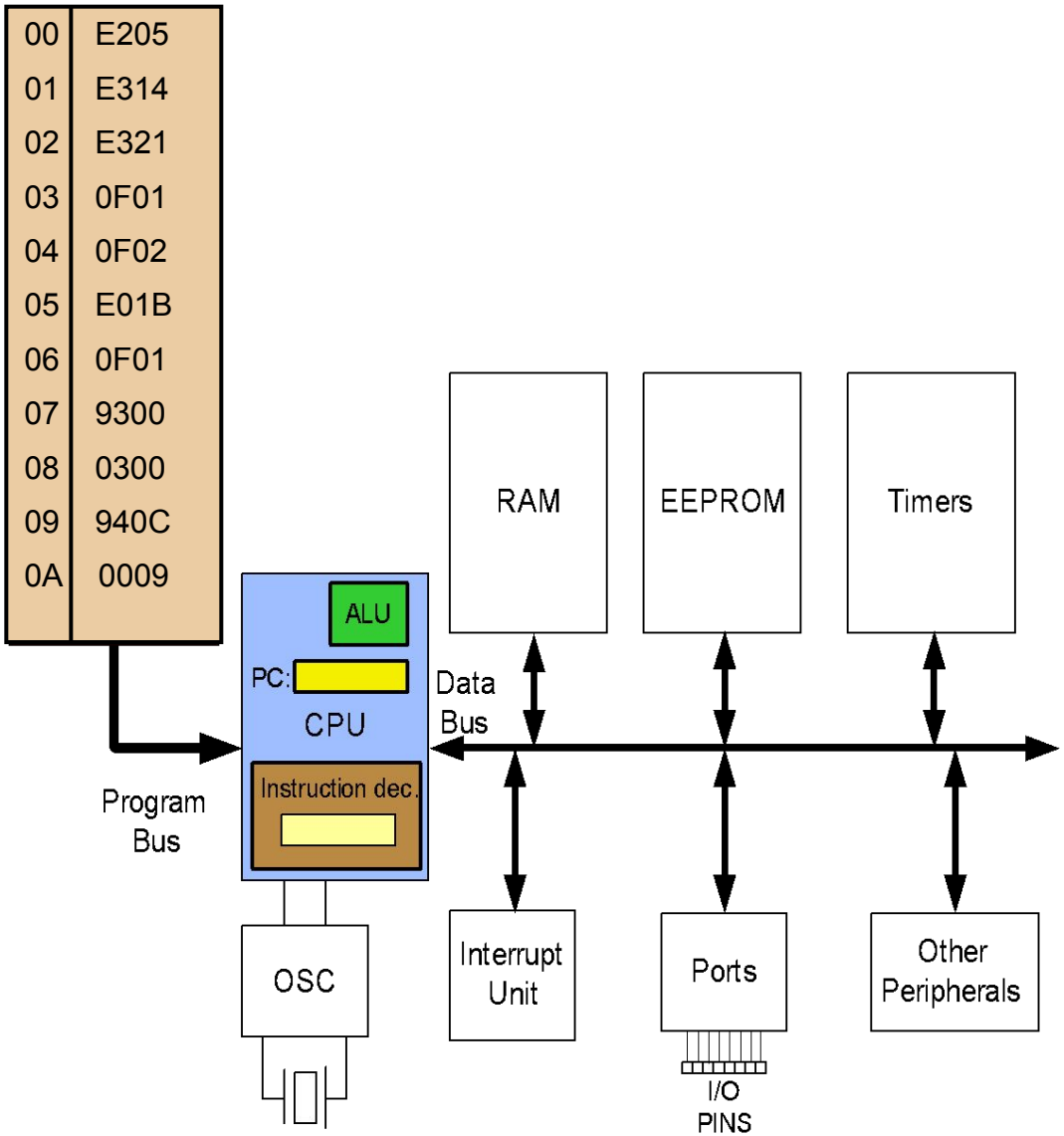
```
      LDI   R16,
0x25
      LDI   R17, $34
      LDI   R18,
0x31
      ADD   R16, R17
      ADD   R16, R18
      LDI   R17, 11
      ADD   R16, R17
      STS   SUM, R16
HERE: JMP  HERE
```



Fetch and execute

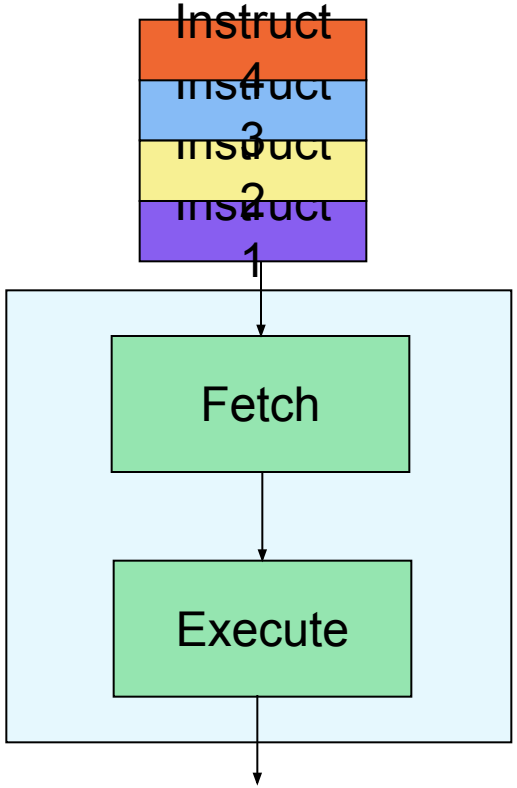
- Old Architectures

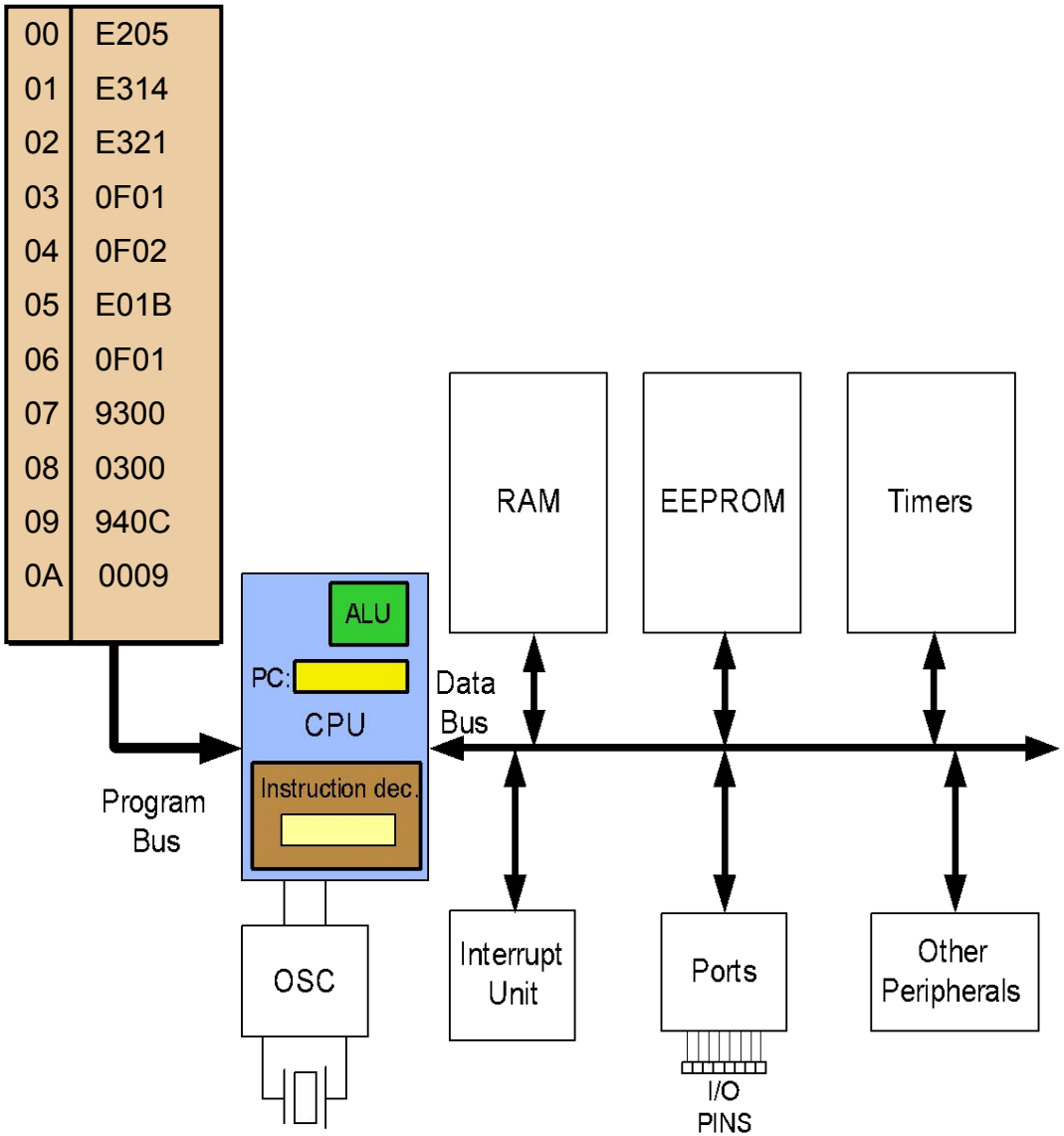




Pipelining

- Pipelining





How to speed up the CPU

- Increase the clock frequency
  - More frequency ☐ More power consumption & more heat
  - Limitations
- Change the architecture
  - Pipelining
  - RISC

Changing the architecture
RISC vs. CISC

- CISC (Complex Instruction Set Computer)
  - Put as many instruction as you can into the CPU
- RISC (Reduced Instruction Set Computer)
  - Reduce the number of instructions, and use your facilities in a more proper way.

RISC architecture

- Feature 1
  - RISC processors have a fixed instruction size. It makes the task of instruction decoder easier.
    - In AVR the instructions are 2 or 4 bytes.
  - In CISC processors instructions have different lengths
    - E.g. in 8051
      - CLR C ; a 1-byte instruction
      - ADD A, #20H ; a 2-byte instruction
      - LJMP HERE ; a 3-byte instruction

RISC architecture

- Feature 2: reduce the number of instructions
  - Pros: Reduces the number of used transistors
  - Cons:
    - Can make the assembly programming more difficult
    - Can lead to using more memory

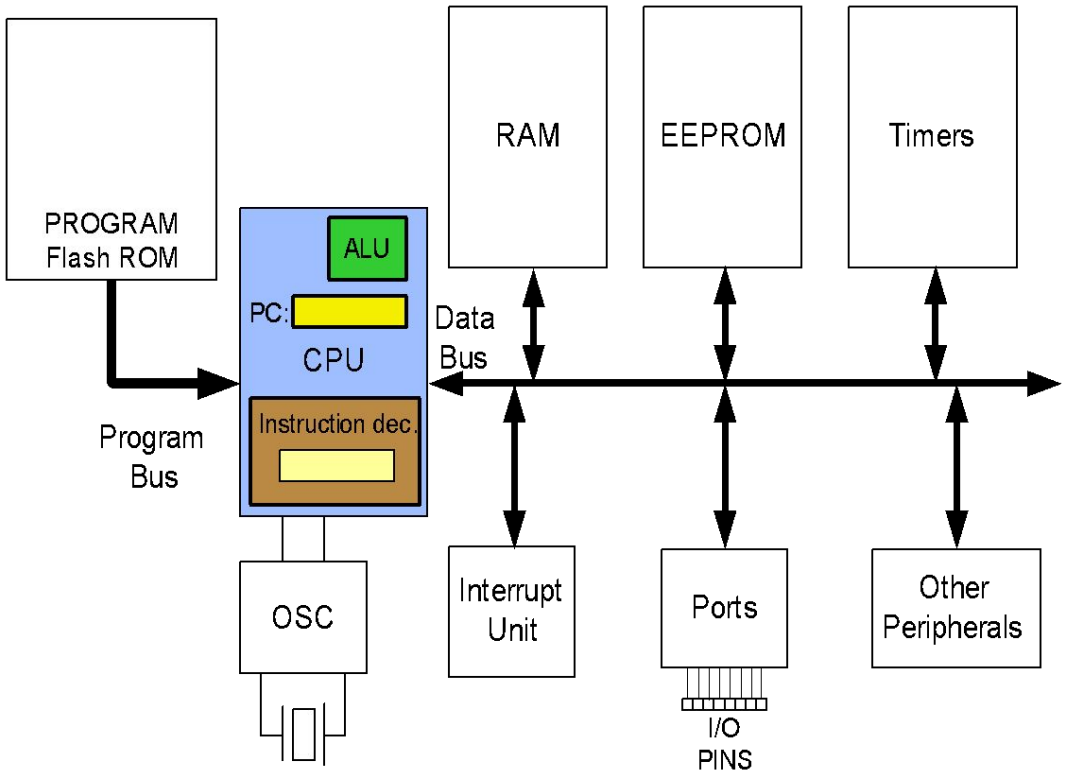
RISC architecture

- Feature 3: limit the addressing mode
  - Advantage
    - hardwiring
  - Disadvantage
    - Can make the assembly programming more difficult

RISC architecture

- Feature 4: Load/Store

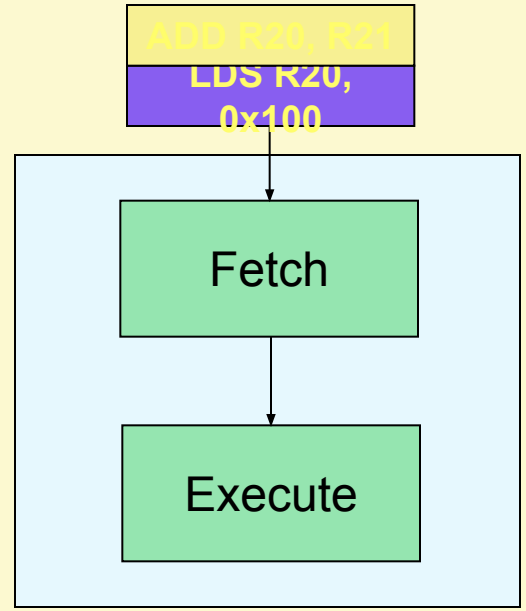
```
LDS R20, 0x200
LDS R21, 0x220
ADD R20, R21
STS 0x230, R20
```

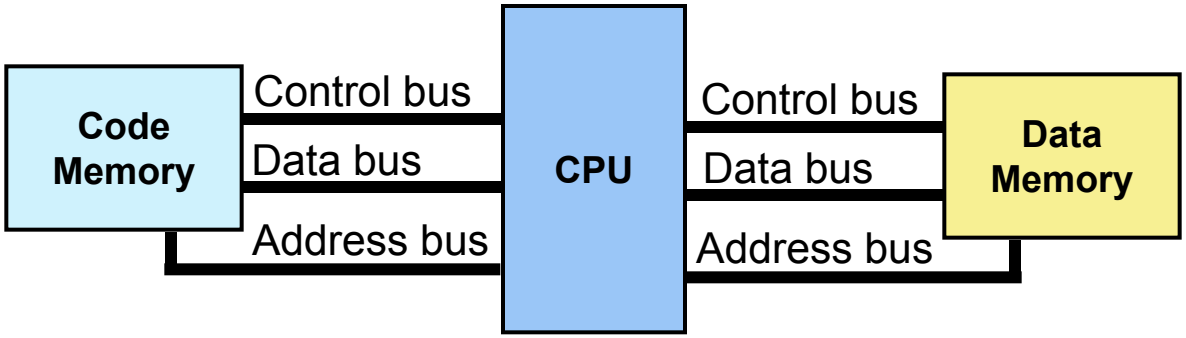


RISC architecture

-

```
LDS R20, 0x100 ; R20 = [0x100]
ADD R20,R21    ; R20 = R20 + R21
```





RISC architecture

- Feature 6: more than 95% of instructions are executed in 1 machine cycle

RISC architecture

- Feature 7
  - RISC processors have at least 32 registers. Decreases the need for stack and memory usages.
    - In AVR there are 32 general purpose registers (R0 to R31)