Microprocessor System Design
AVR Microcontroller

Omid Fatemi

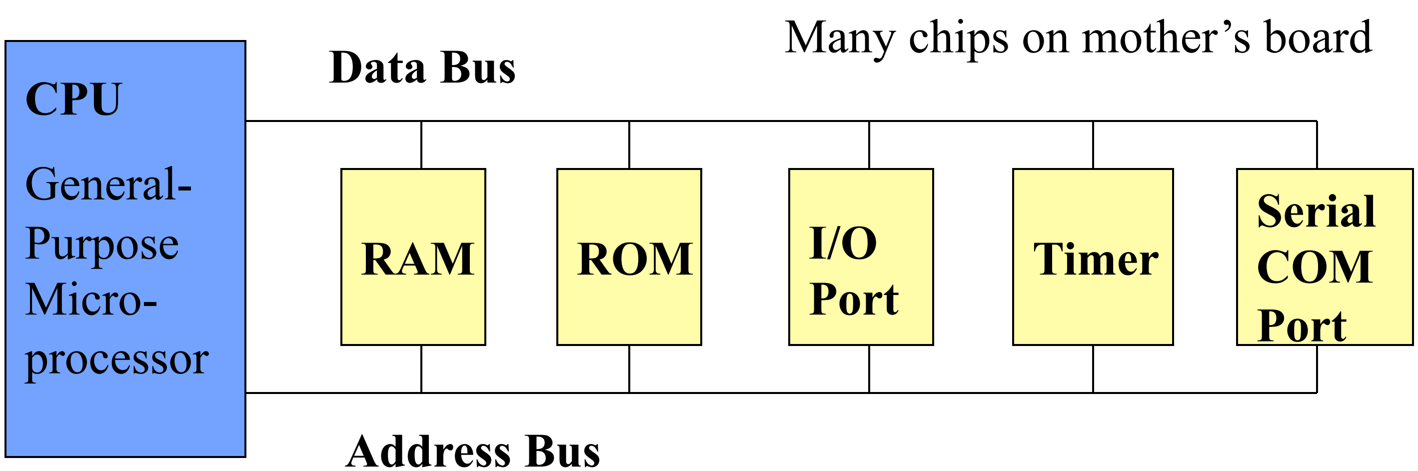
Contents

- **Introduction**

Introduction

General-purpose microprocessor

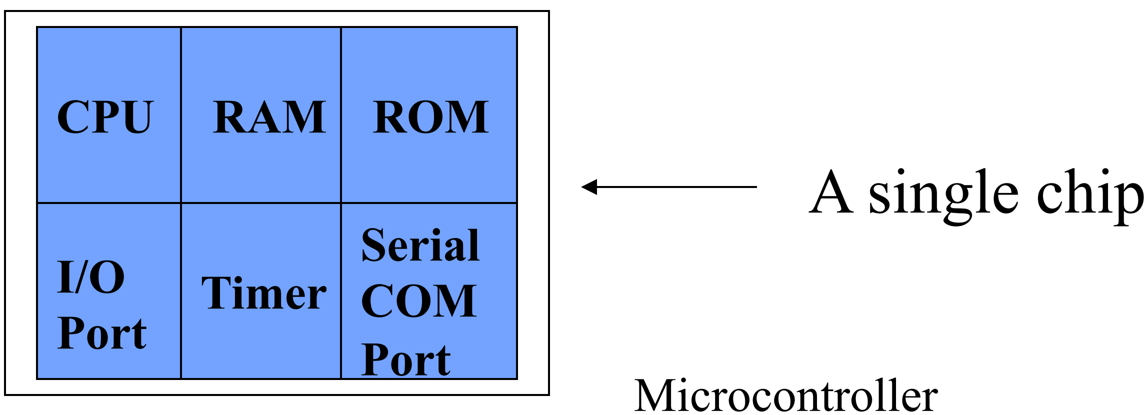
- **CPU for Computers**
- **No RAM, ROM, I/O on CPU chip itself**
- **Example Intel's x86, Motorola's 680x0**



General-Purpose Microprocessor System

Microcontroller

- A smaller computer
- On-chip RAM, ROM, I/O ports...
- Example AVR, Intel's 8051, Zilog's Z8 and PIC 16X



Microprocessor vs. Microcontroller

- **Microprocessor**
- **CPU is stand-alone, RAM, ROM, I/O, timer are separate**
- **designer can decide on the amount of ROM, RAM and I/O ports.**
- **More consumption power**
- **More computing power**
- **versatility**
- **general-purpose**

Microcontroller

- CPU, RAM, ROM, I/O and timer are all on a single chip
- fix amount of on-chip ROM, RAM, I/O ports
- for applications in which cost, power and space are critical
- Less applications

Embedded System

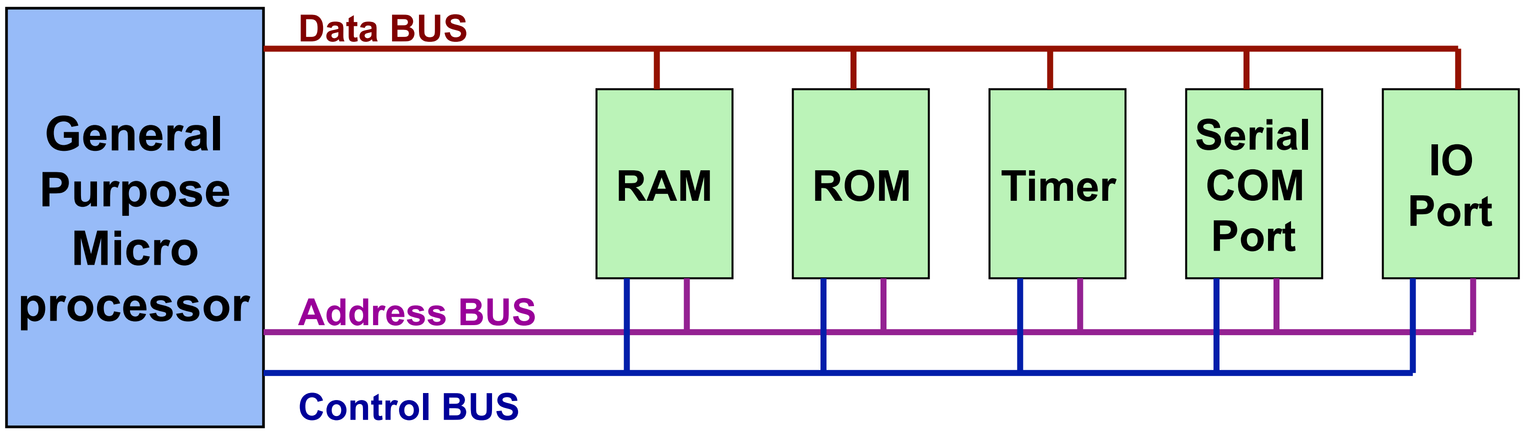
- Embedded system means the processor is embedded into that application.
- An embedded product uses a microprocessor or microcontroller to do one task only.
- In an embedded system, there is only one application software that is typically burned into ROM.
- Example printer, keyboard, video game player

Three Criteria in Choosing a Microcontroller

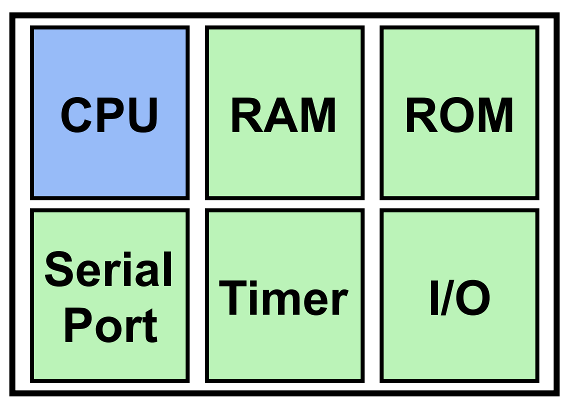
- **meeting the computing needs of the task efficiently and cost effectively**
  - speed, the amount of ROM and RAM, the number of I/O ports and timers, size, packaging, power consumption
  - easy to upgrade
  - cost per unit
- **availability of software development tools**
  - assemblers, debuggers, C compilers, in circuit emulator, simulator, technical support
- **wide availability and reliable sources for the microcontroller.**

General Purpose Microprocessors vs. Microcontrollers

- **General Purpose Microprocessors**



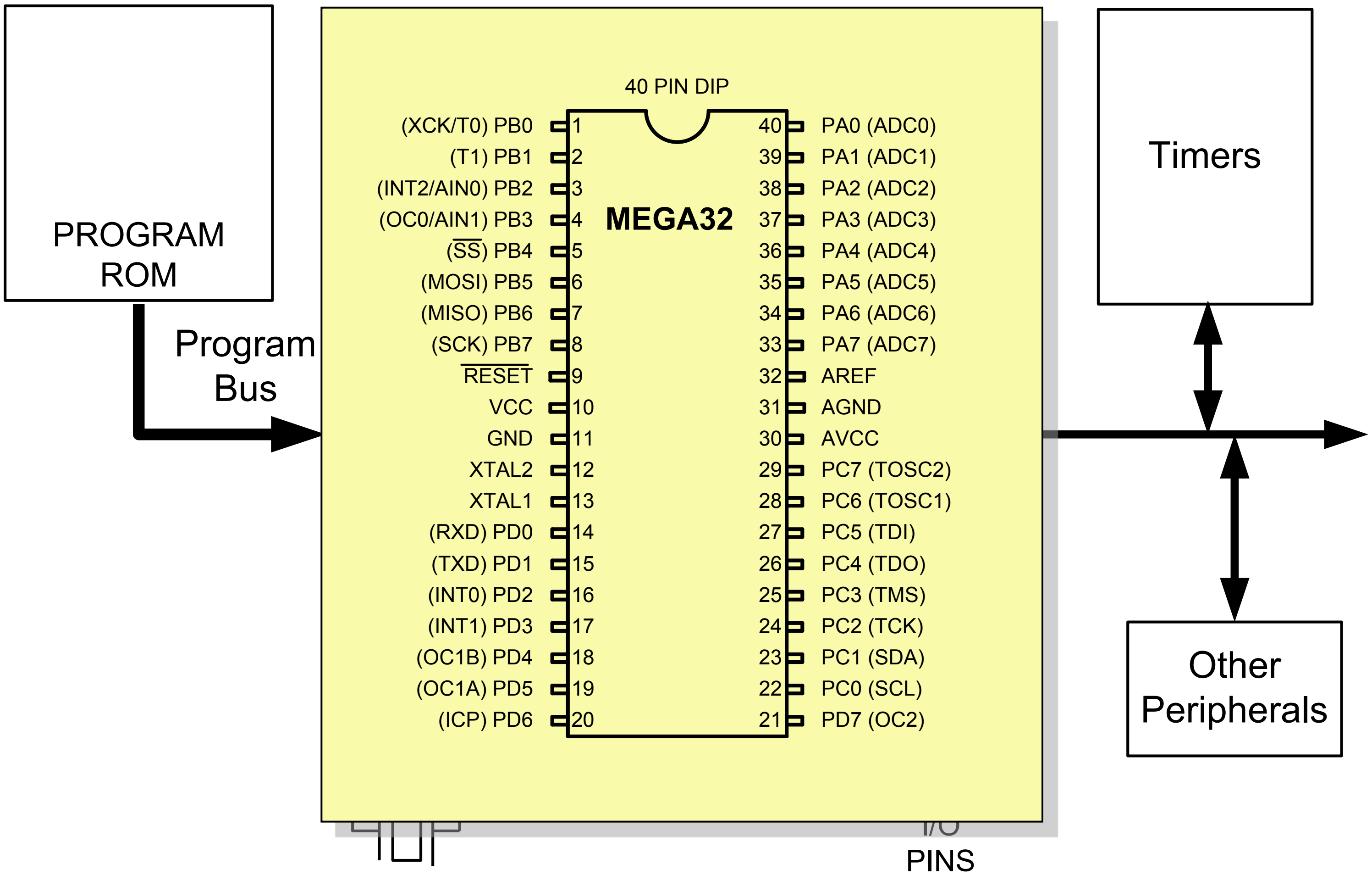
- Microcontrollers



Most common microcontrollers

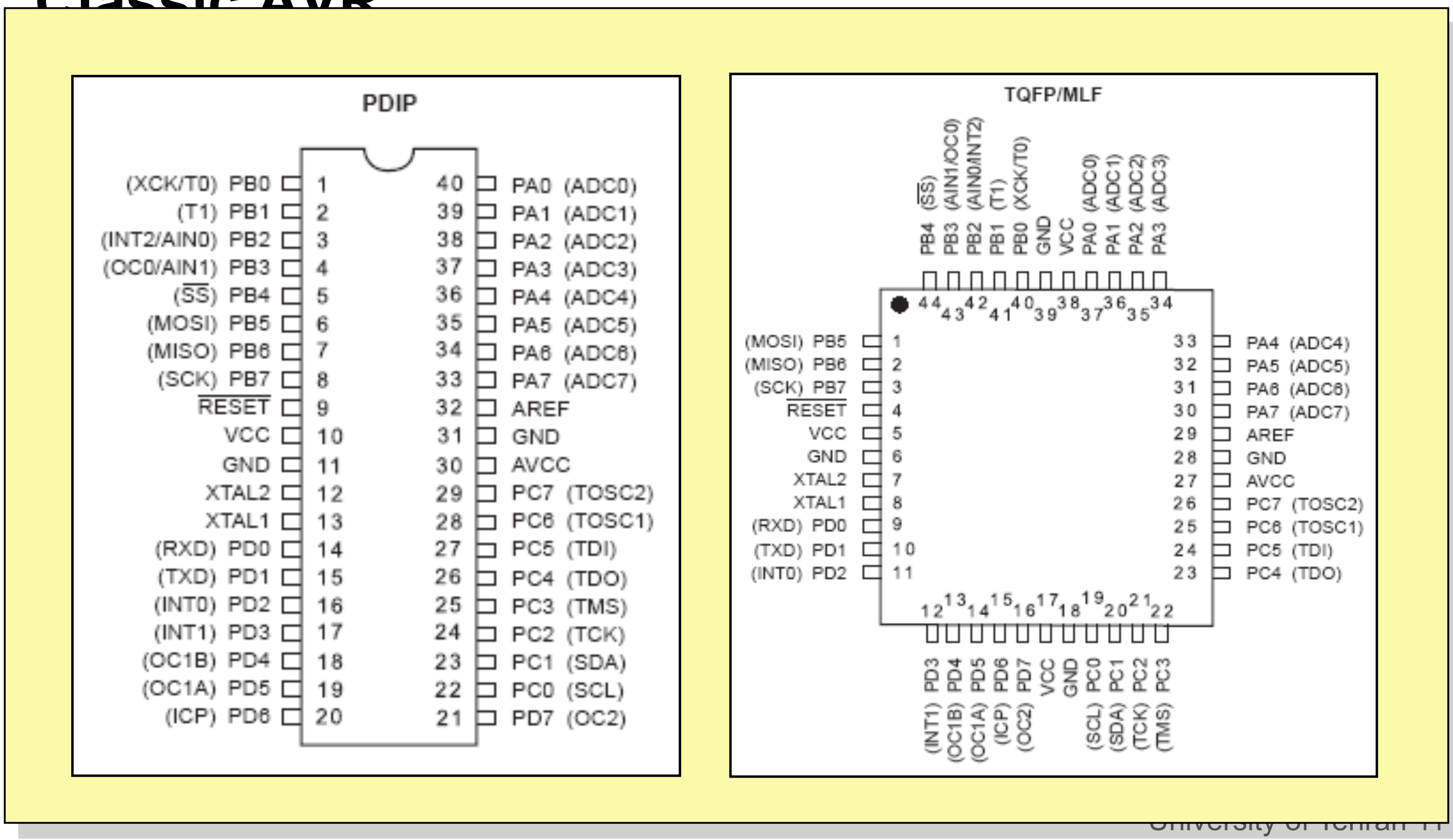
- **8-bit microcontrollers**
  - **AVR**
  - **PIC**
  - **HCS12**
  - **8051**
- **32-bit microcontrollers**
  - **ARM**
  - **PIC32**

AVR internal architecture



AVR different groups

- Classic AVR
-
-
-



Let’s get familiar with the AVR part numbers

ATmega128

Atmel | group | Flash =128K

ATtiny44

Atmel | Tiny group | Flash =4K

AT90S4433

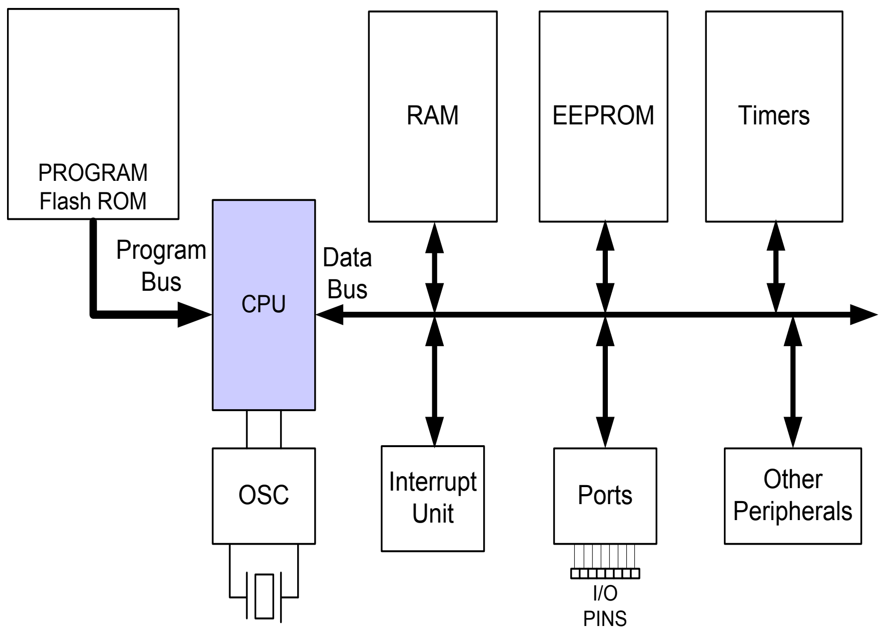
Atmel | Classic group | Flash =4K

Introduction to Assembly
Chapter 2

The AVR microcontroller and embedded systems
using assembly and c

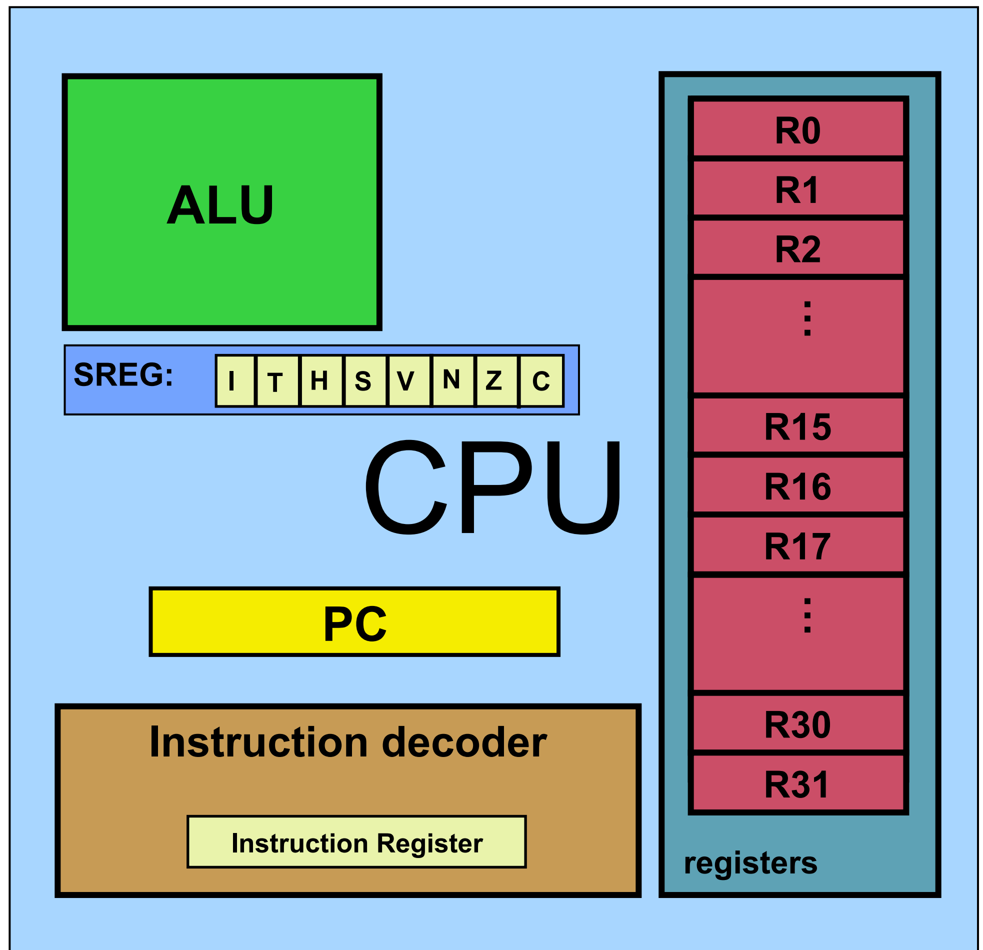
Topics

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



AVR's CPU

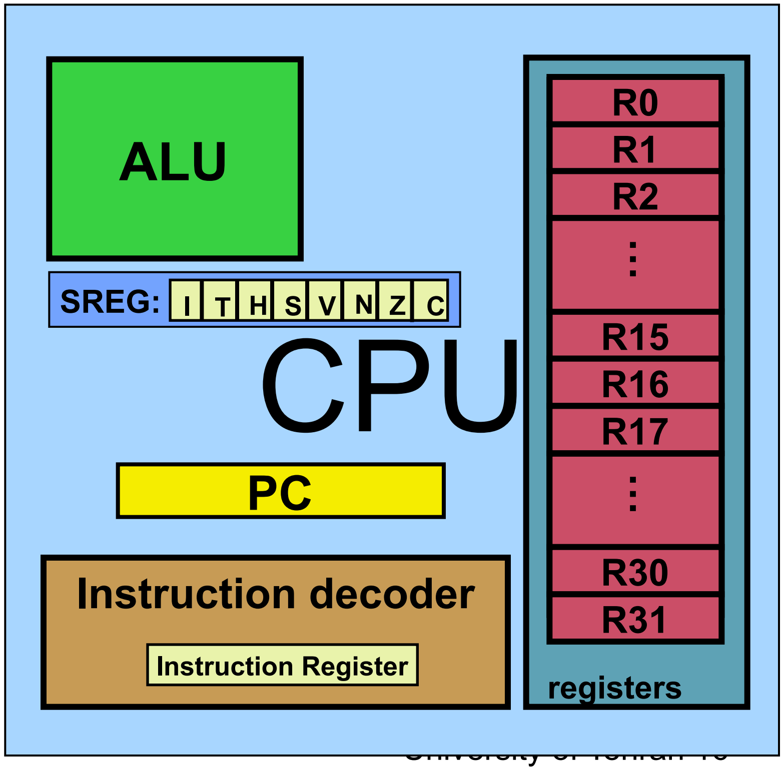
- **AVR's CPU**
  - ALU
  - 32 General Purpose registers (R0 to R31)
  - PC register
  - Instruction decoder



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 R19,132
  - LDI R23,0x27
    - » R23 = 0x27



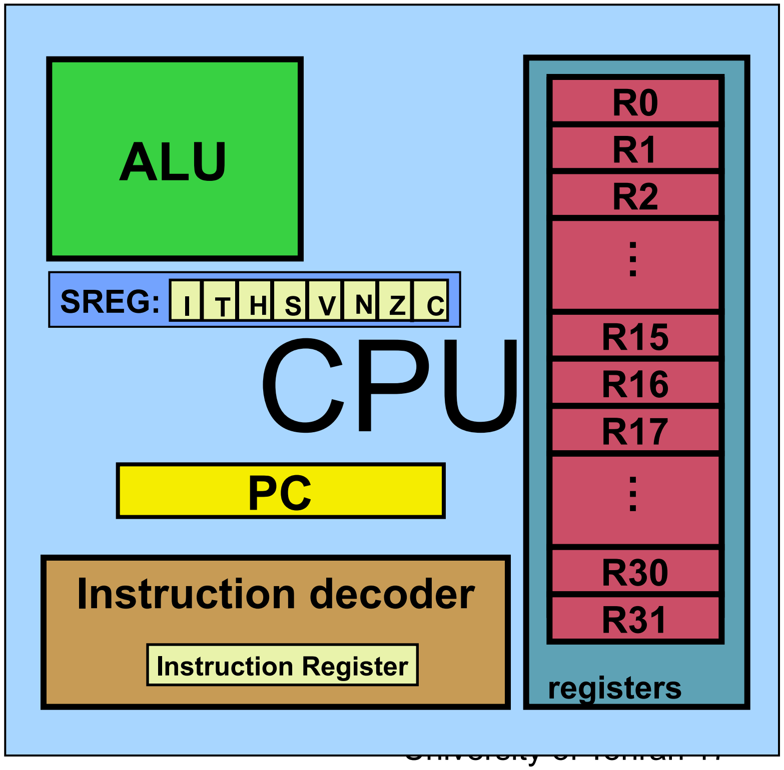
Some simple instructions

2. Arithmetic calculation

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

  ADD, SUB, MUL, AND, etc.

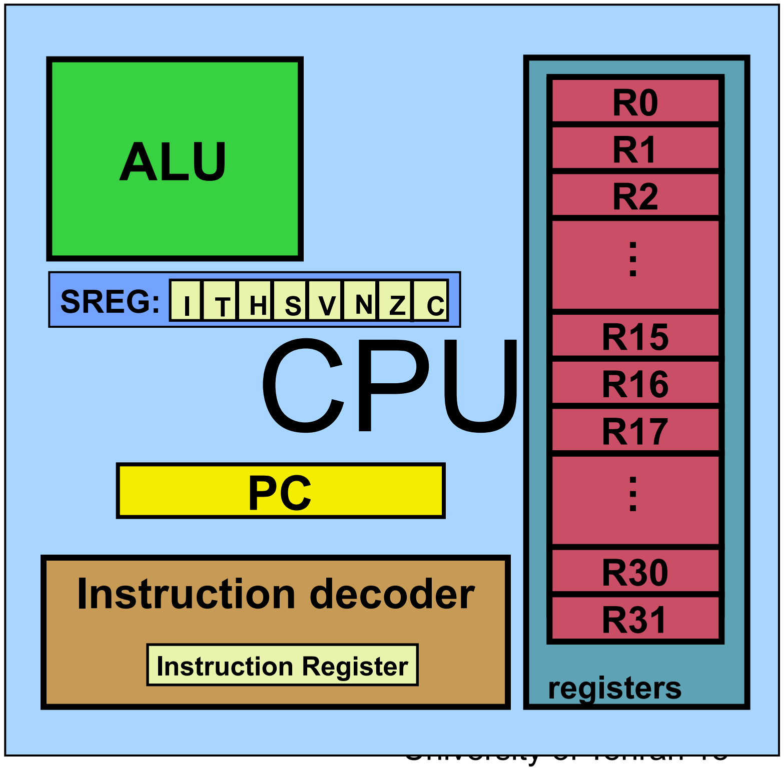
- ADD Rd,Rs
  - Rd = Rd + Rs
  - Example:
  - ADD R25, R9
    - R25 = R25 + R9
  - ADD R17,R30
    - R17 = R17 + R30



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**

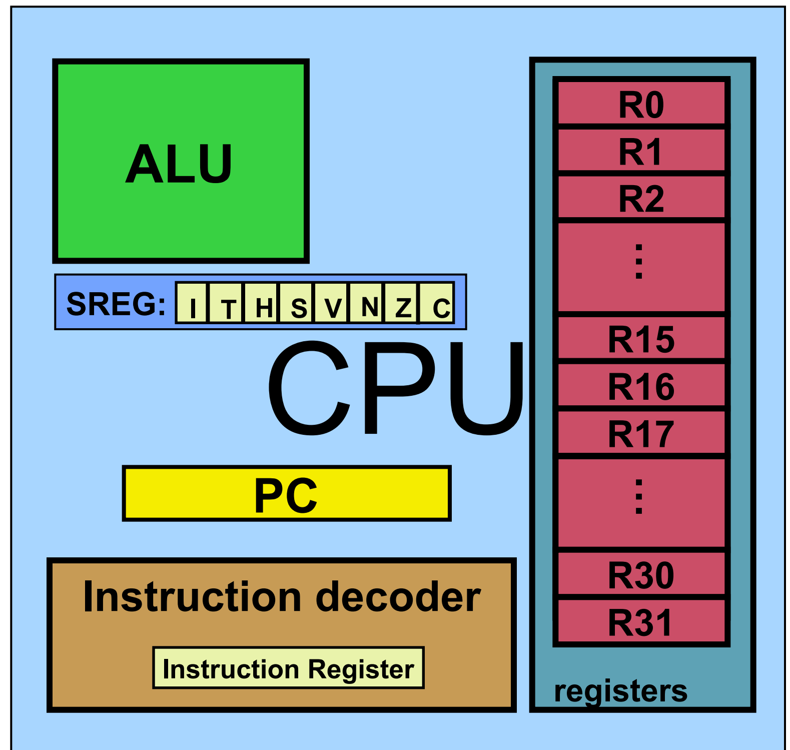
```
LDI     R16, 19        ;R16 = 19
LDI     R20, 95        ;R20 = 95
LDI     R21, 5         ;R21 = 5
ADD     R16, R20       ;R16 = R16 + R20
ADD     R16, R21       ;R16 = R16 + R21
```

```
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

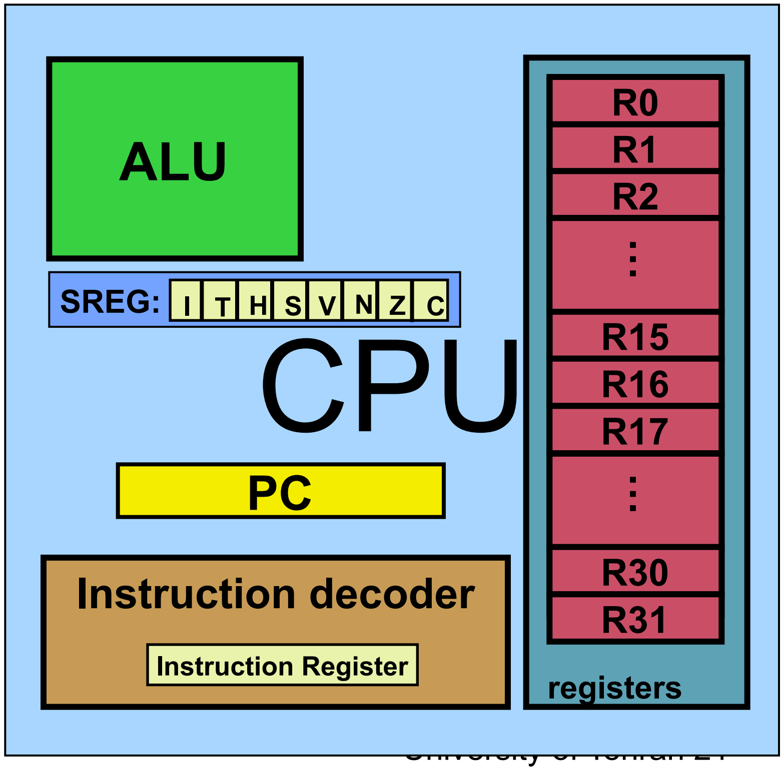


Some simple instructions

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**

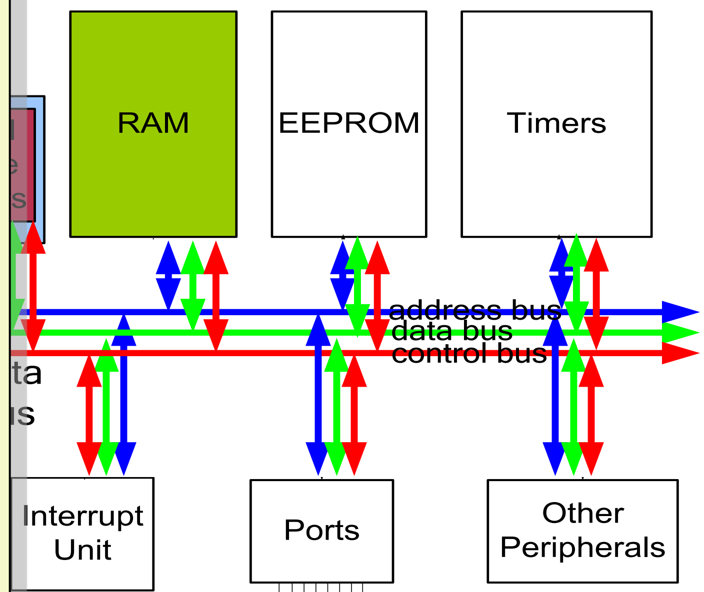


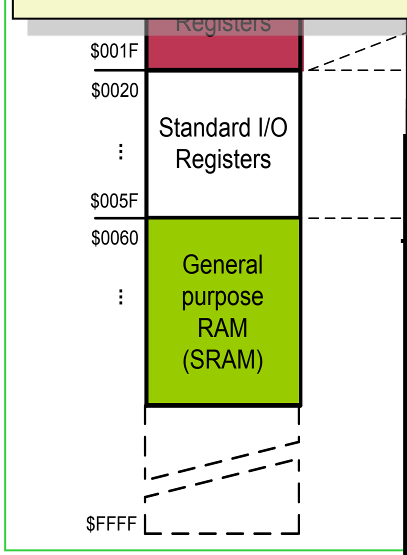
Address I/O	Address Mem.	Name
$00	$20	TWBR
$01	$21	TWSR
$02	$22	TWAR
$03	$23	TWDR
$04	$24	ADCL
$05	$25	ADCH
$06	$26	ADCSRA
$07	$27	ADMUX
$08	$28	ACSR
$09	$29	UBRRL
$0A	$2A	UCSRB
$0B	$2B	UCSRA
$0C	$2C	UDR
$0D	$2D	SPCR
$0E	$2E	SPSR
$0F	$2F	SPDR
$10	$30	PIND
$11	$31	DDRD
$12	$32	PORTD
$13	$33	PINC
$14	$34	DDRC
$15	$35	PORTC

Address I/O	Address Mem.	Name
$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

Address I/O	Address Mem.	Name
$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

Space





Example: Add conten

Example: What does t

```
LDS  R20,2
```

Answer:

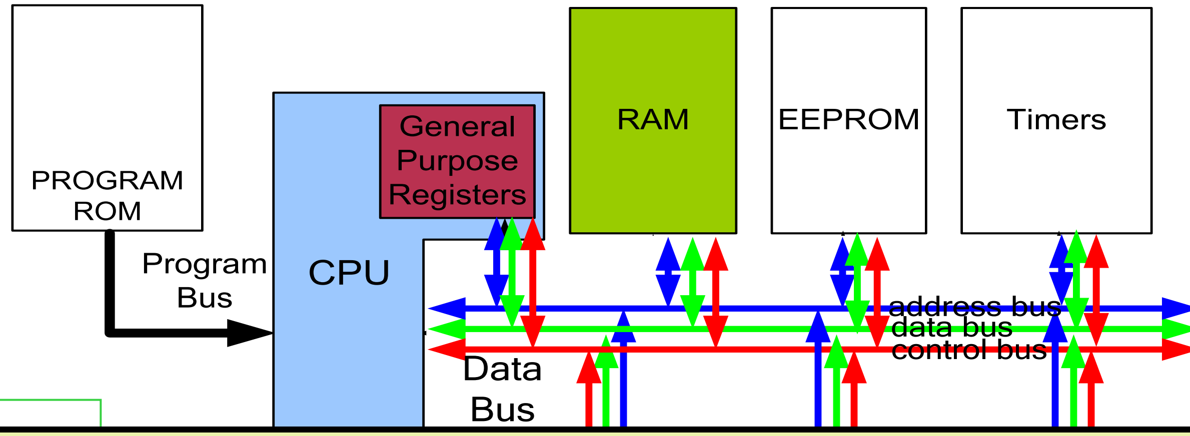
It copies the conte

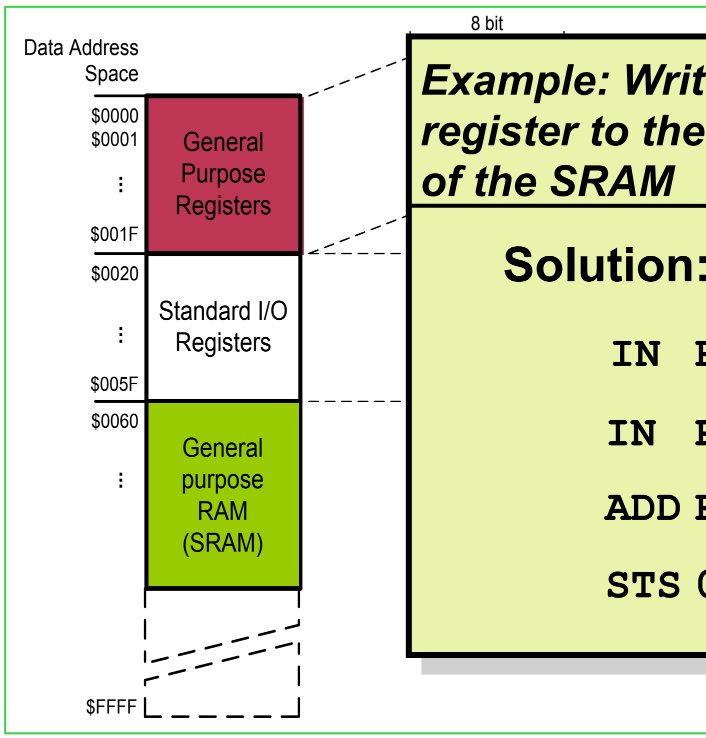
Example: Store 0x53 into the SPH register. The address of SPH is 0x5E

Solution:

```
LDI    R20, 0x53          ;R20 = 0x53
STS    0x5E, R20          ;SPH = R20
```

Data Address Space





Example: Write a program that adds the contents of the PINC IO 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
```

Status Register (SREG)

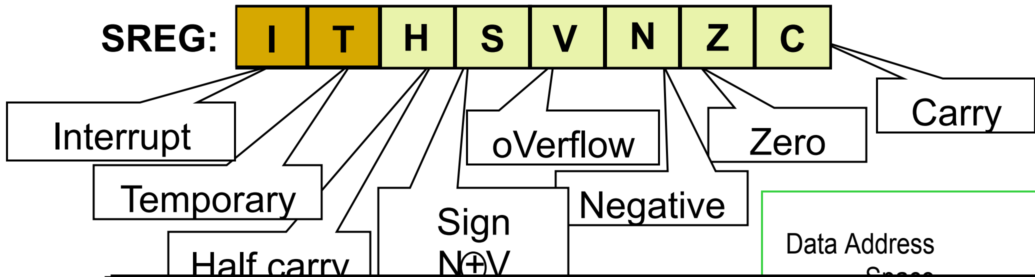


Table 2-5: AVR Branch (Jump) Instructions Using Flag Bits

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 fla... subtraction of 0x9C from 0x9C in the following

```
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 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

•

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***

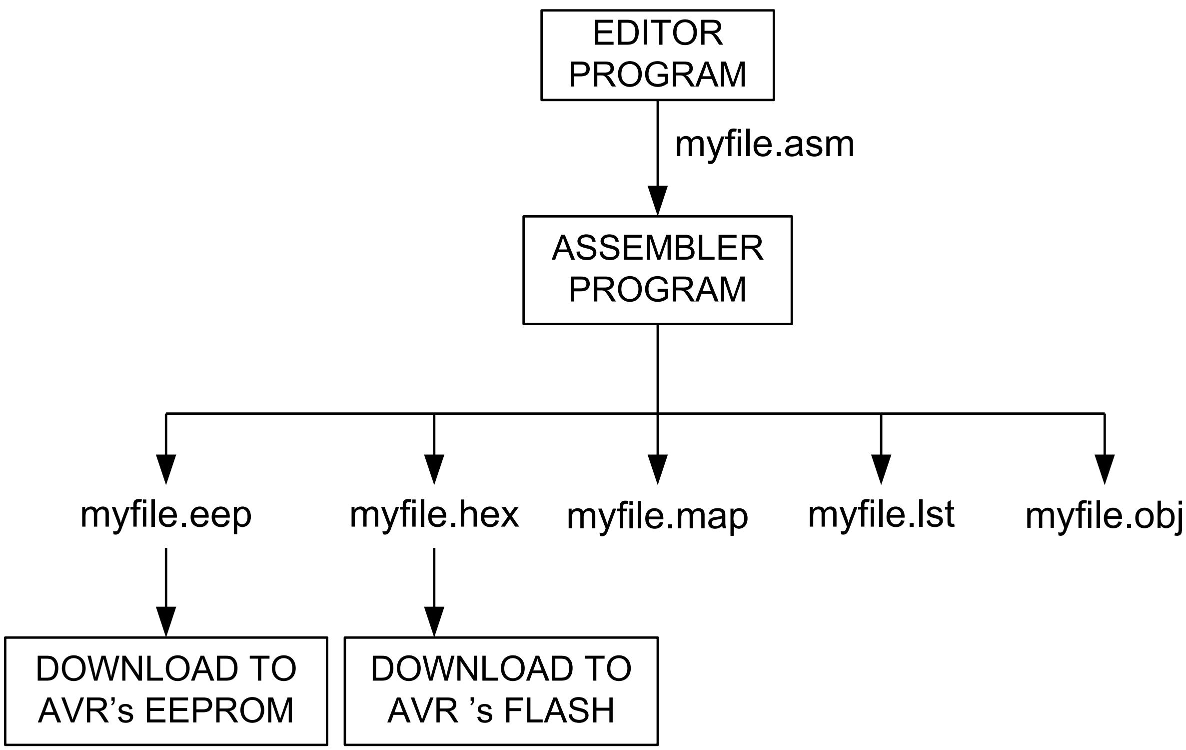
Program.asm

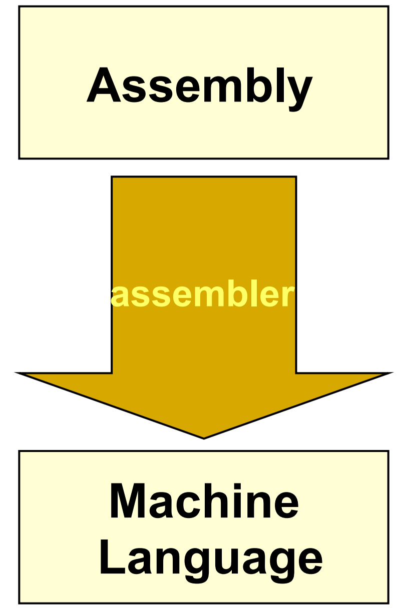
```
.ORG 0
 LDI  R16, 0x25
.ORG 0x7
 LDI  R17, 0x34
 LDI  R18, 0x31
```

assembler →

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

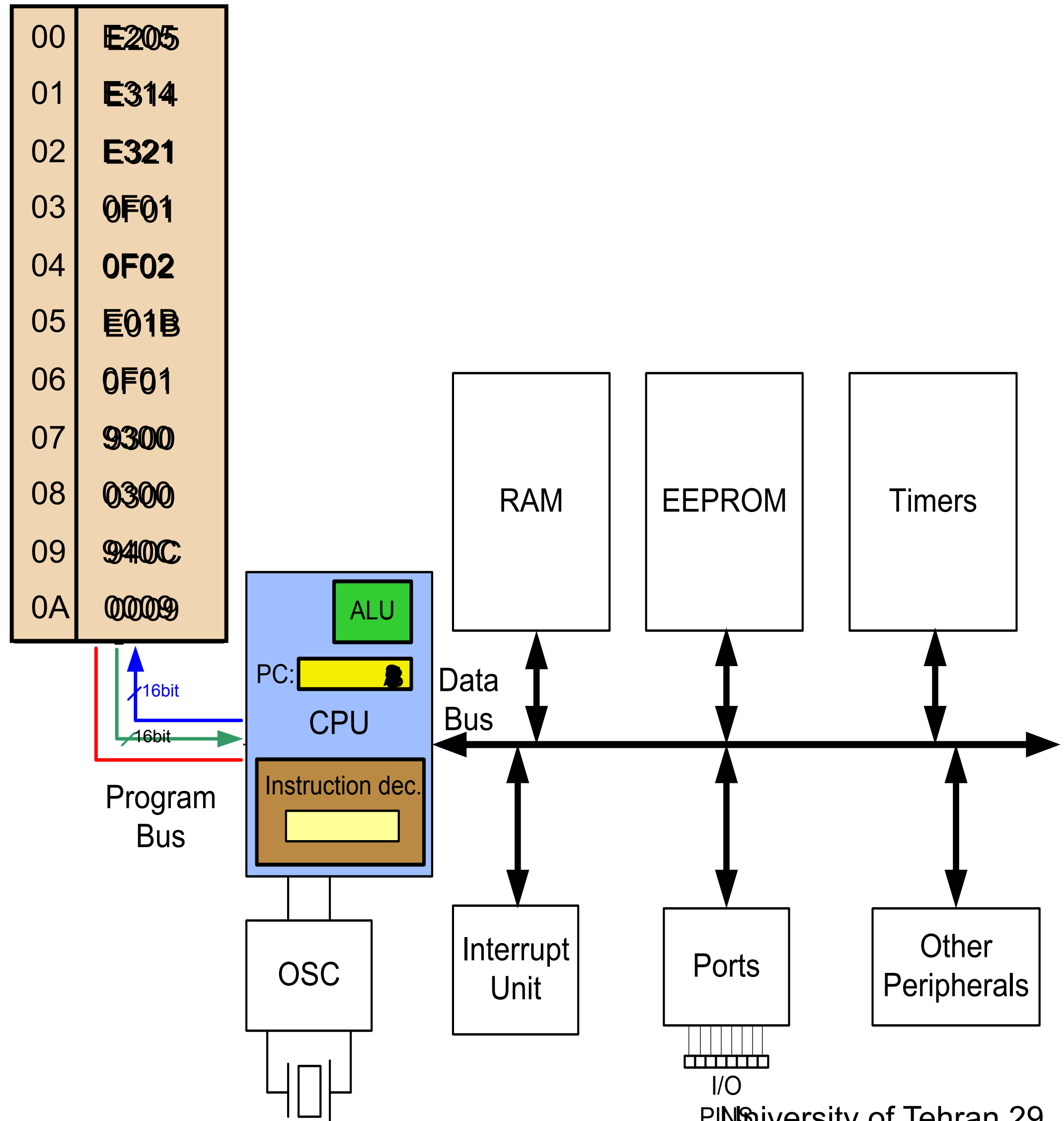
Assembler


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

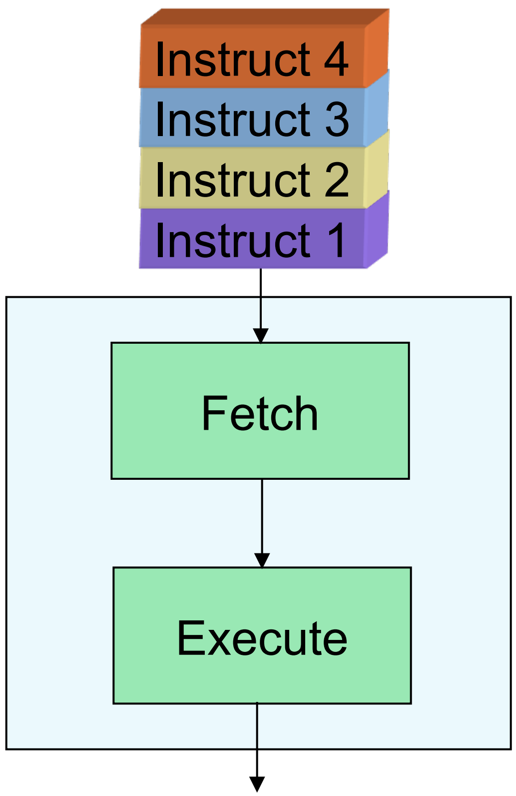
Flash memory and PC register

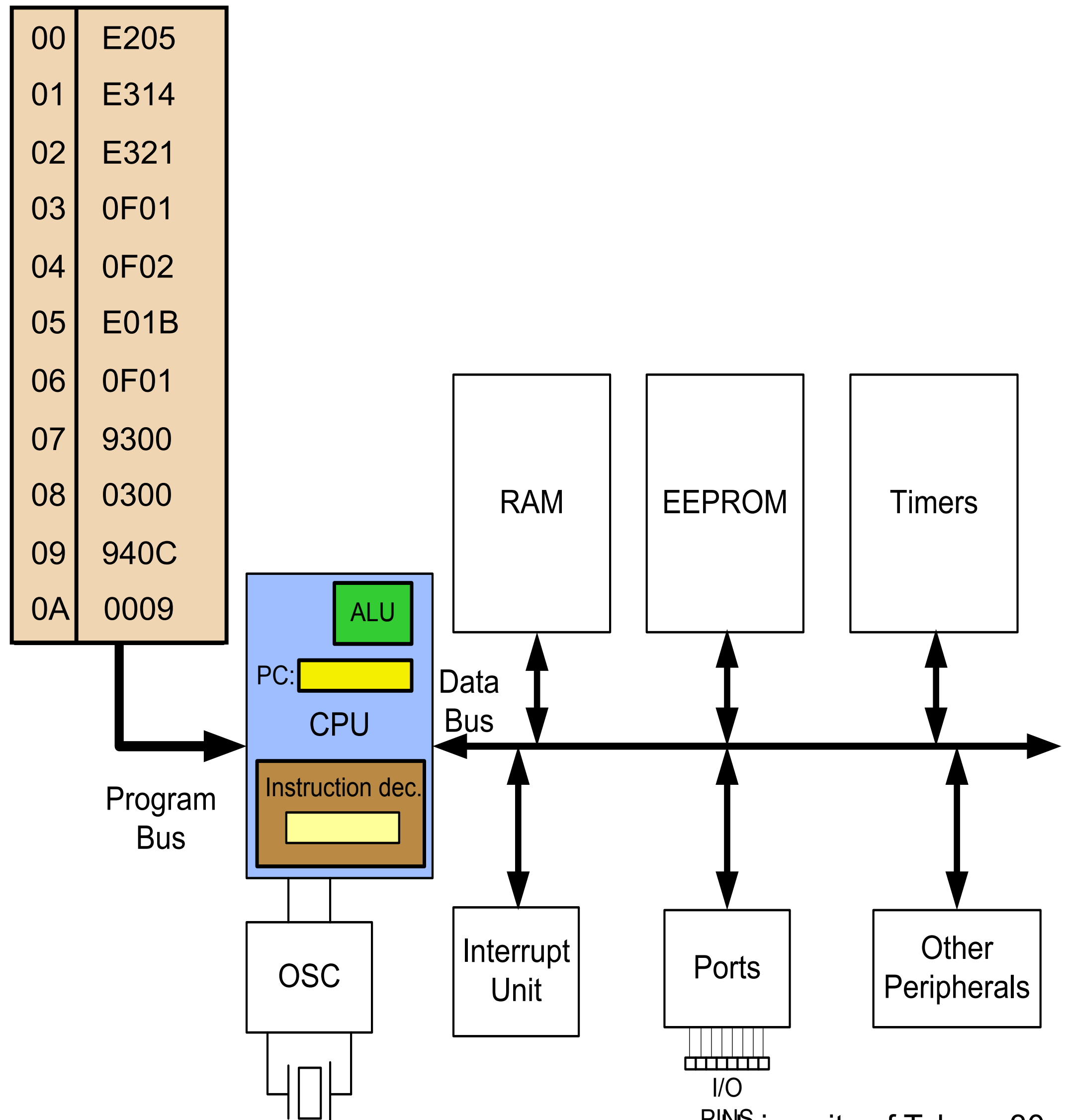
```
      LDI  R16, 0x25
      LDI  R17, $34
      LDI  R18, 0x31
      ADD  R16, R17
      ADD  R16, R18
      LDI  R17, 11
      ADD  R16, R17
      STS  SUM, R16
HERE:JMP  HERE
```



Fetch and execute

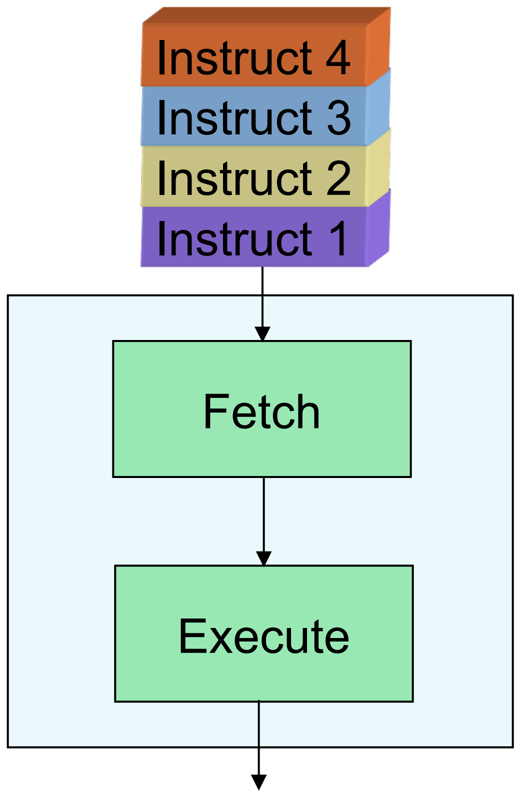
- **Old Architectures**

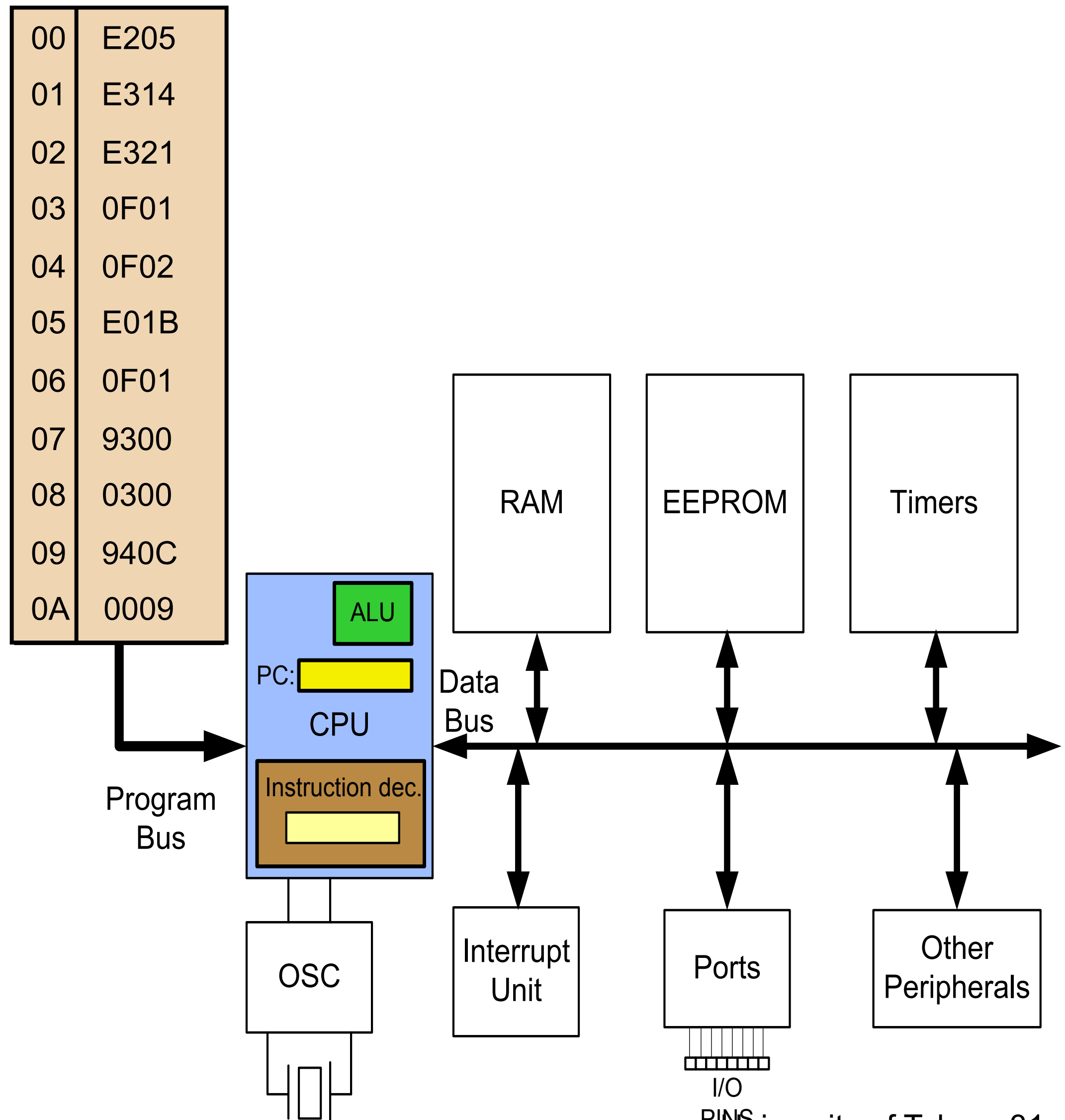




Pipelining

- **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**

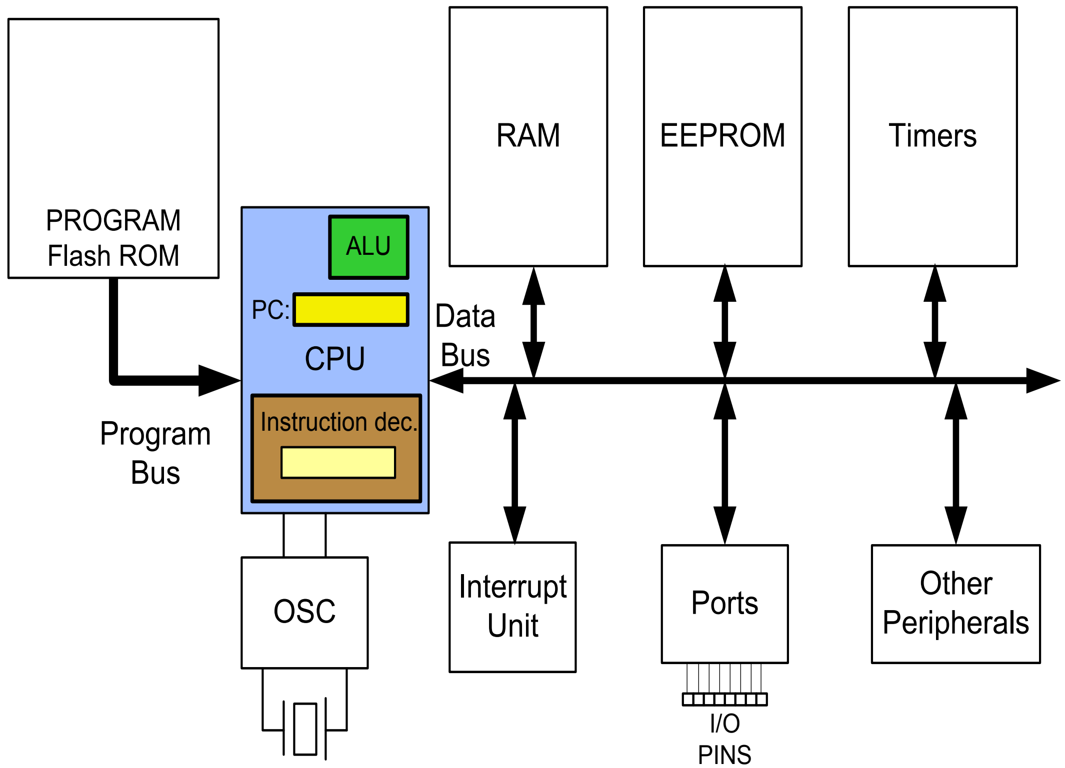
RISC architecture

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

RISC architecture

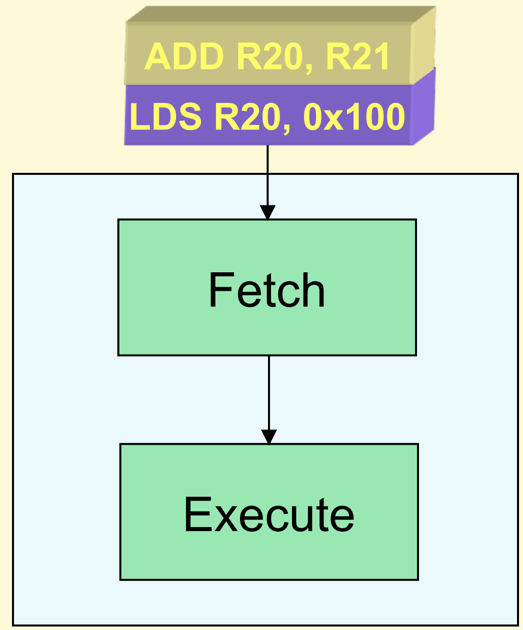
- **Feature 4: Load/Store**

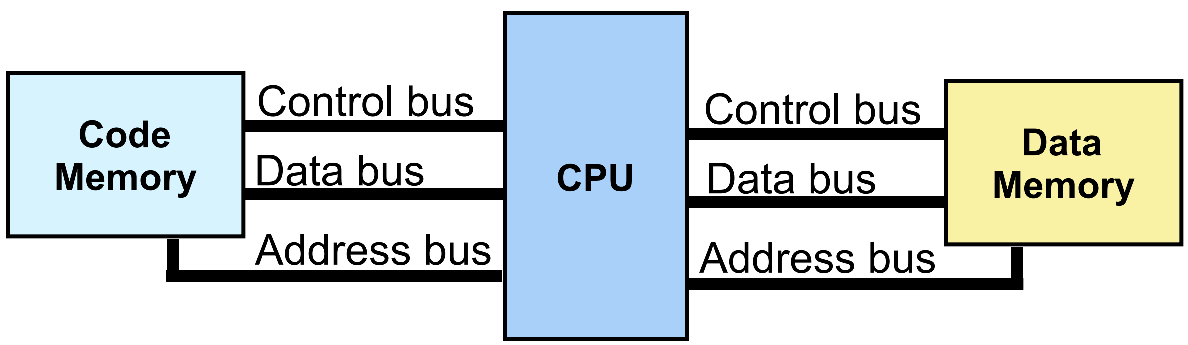
```
LDS R20, 0x200
LDS R21, 0x220
ADD R20, R21
STS 0x230, R20
```



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)**