Microcontrollers: AVR
(from here for session 25)

Amir Mahdi Hosseini Monazzah

Room 332,
School of Computer Engineering,
Iran University of Science and Technology,
Tehran, Iran.
monazzah@iust.ac.ir

Fall 2020

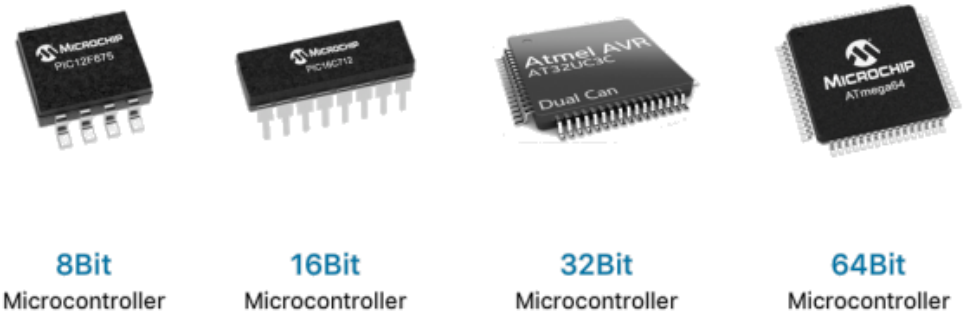
* Parts of slides are adopted from Dr. Konstantinos Tatas's "Microprocessors" lecture slides Fredrick University

Overview

- Introduction
  - Definition
- AVR families
  - Different types of AVRs
  - Architecture
  - Assembly
  - How to write programs for AVRs
  - Programming AVRs
- Case study introduction
  - ATmega32

Definition

- A **microcontroller** (**MCU** for *microcontroller unit*, or **UC** for *µ-controller*)
  - A small computer on a single integrated circuit.
  - It is similar to, but less sophisticated than, a system on a chip (SoC)
    - SoC may include a microcontroller as one of its components.

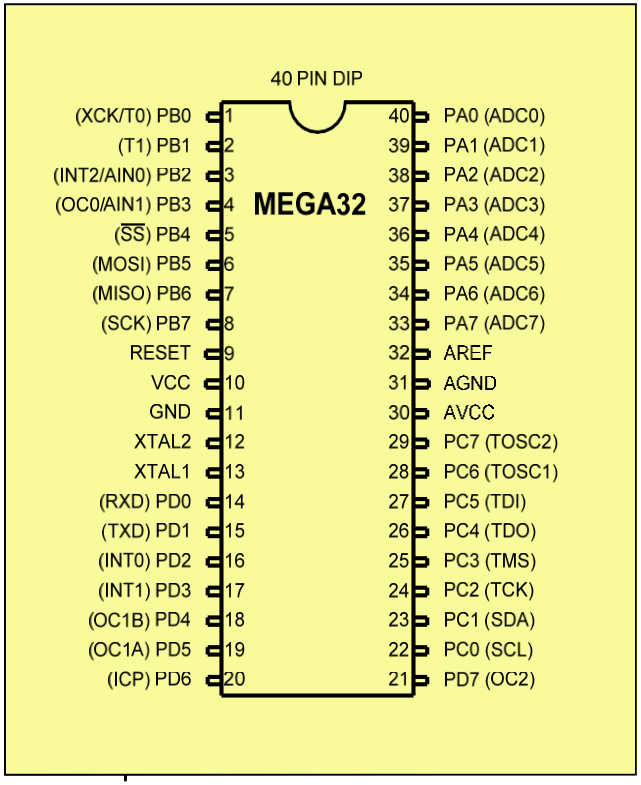


Definition

- A microcontroller contains
  - One or more CPUs (processor cores)
  - Memory
    - Ferroelectric RAM
    - NOR flash
    - OTP ROM
    - RAM
  - Programmable input/output peripherals.
- Microcontrollers are designed for embedded applications

Introduction

- Widely-used microcontroller
  - Different families
    - Based on the application and Flash memory capacity
  - Classic AVR
    - e.g. AT90S2313, AT90S4433
  - Mega
    - e.g. ATmega8, ATmega32, ATmega128
  - Tiny
    - e.g. ATtiny13, ATtiny25
  - Special Purpose AVR
    - e.g. AT90PWM216,AT90USB1287

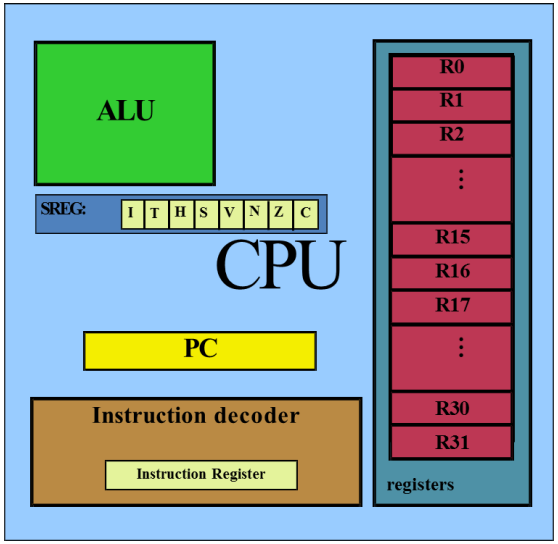


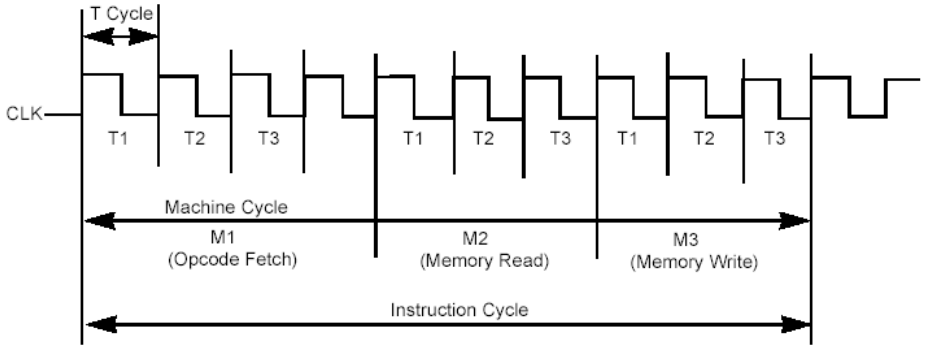
AVR's CPU

- CPU:
  - RISC architecture
  - 131 instructions
  - Most instructions are executed in a single cycle
  - 32 general-purpose registers
  - 64 IO registers
  - Many useful peripherals
- Very low-power
  - Less than 10mW power consumption

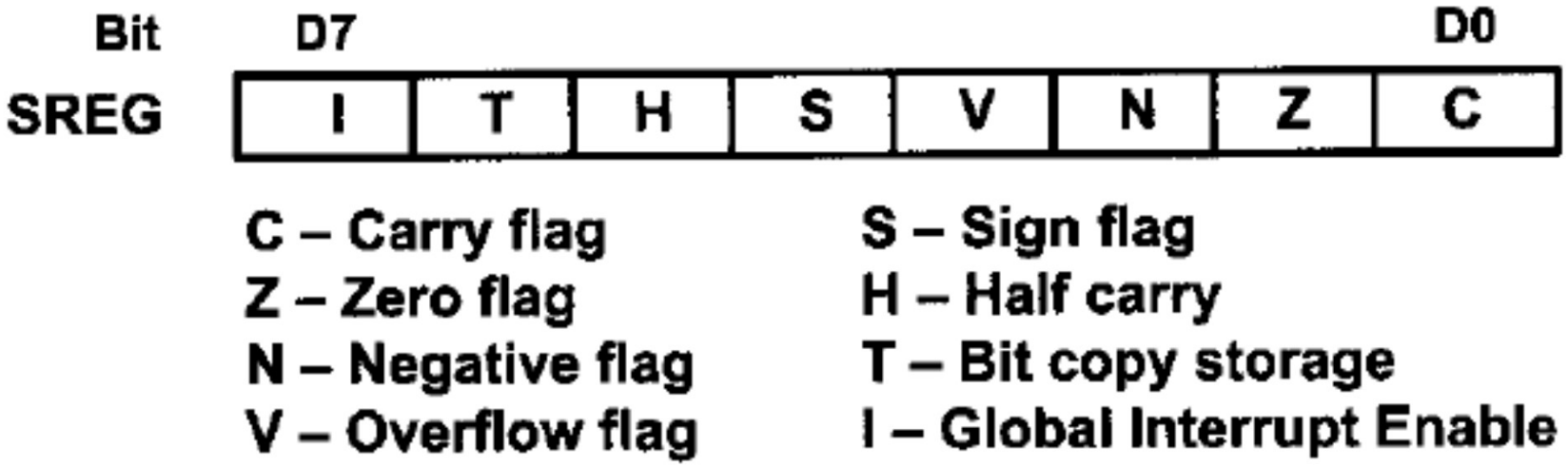
AVR's CPU

- AVR's CPU
  - ALU
  - 32 general purpose registers (R0 to R31)
  - PC register
  - Instruction decoder
- Almost the same for all families
- AVR cycles
  - Fetch
  - Decode
  - Execute
    - Operand fetch
    - Execute
    - Write-back





SREG flags



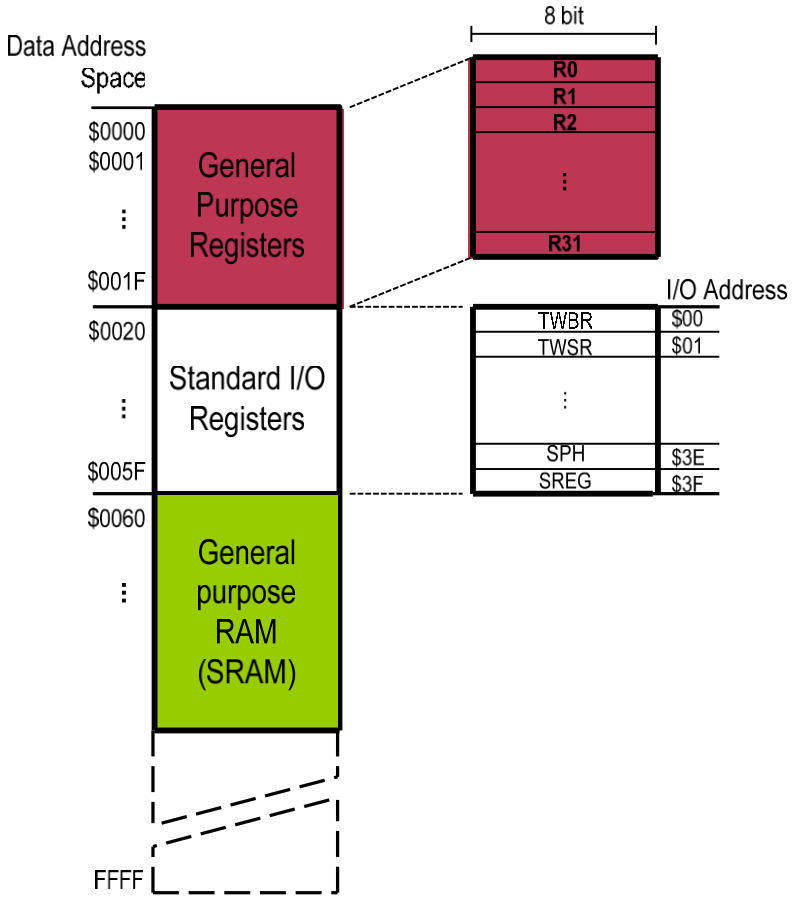
- Not all instructions affect flags
  - Example: load instruction has nothing to do with flags
- S-bit == XOR(N-bit, V-bit) for signed test

Decision making by flags

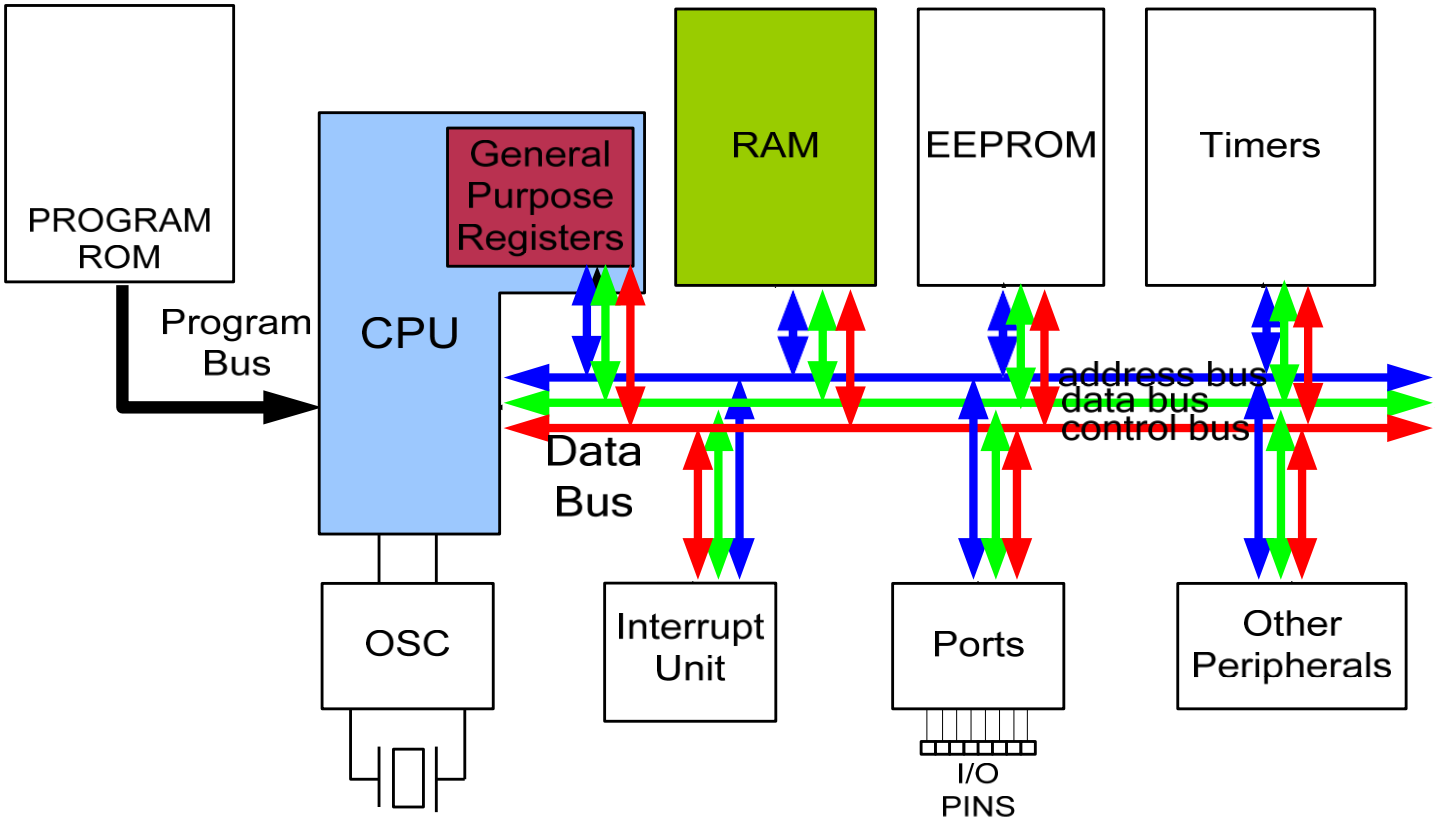
Instruction	Action
BRLO	Branch if C = 1
BRSH	Branch if C = 0
BREQ	Branch if Z = 1
BRNE	Branch if Z = 0
BRMI	Branch if N = 1
BRPL	Branch if N = 0
BRVS	Branch if V = 1
BRVC	Branch if V = 0

Memory address space

- A unified address space
- 32 general-purpose registers
  - Directly connected to the ALU
- 64 IO registers
  - To keep the data sent to/ received from peripherals
- Stack
  - Starting from the end of SRAM and grow up to lower addresses
  - Indexed by SPL-SPH registers

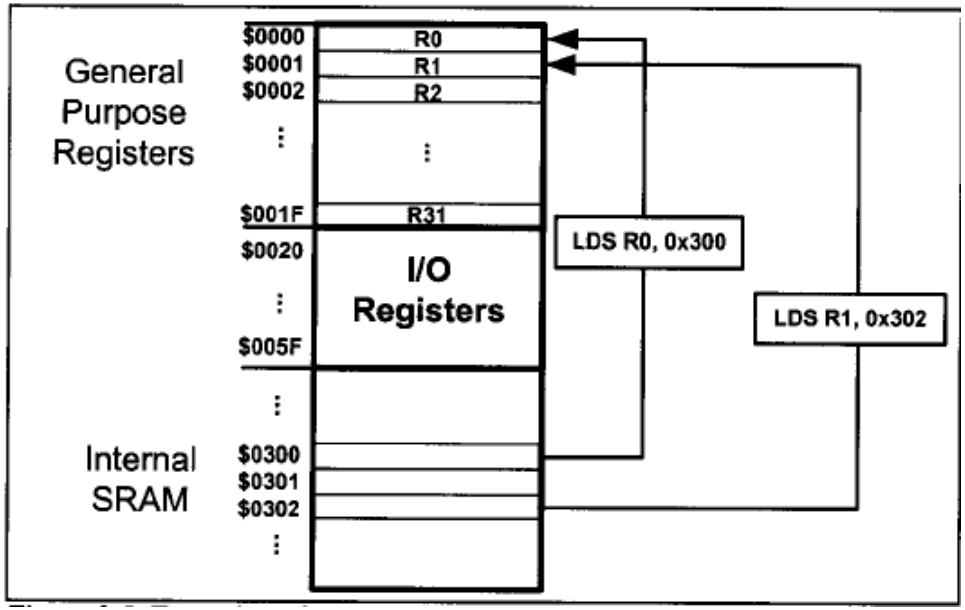


Internal busses



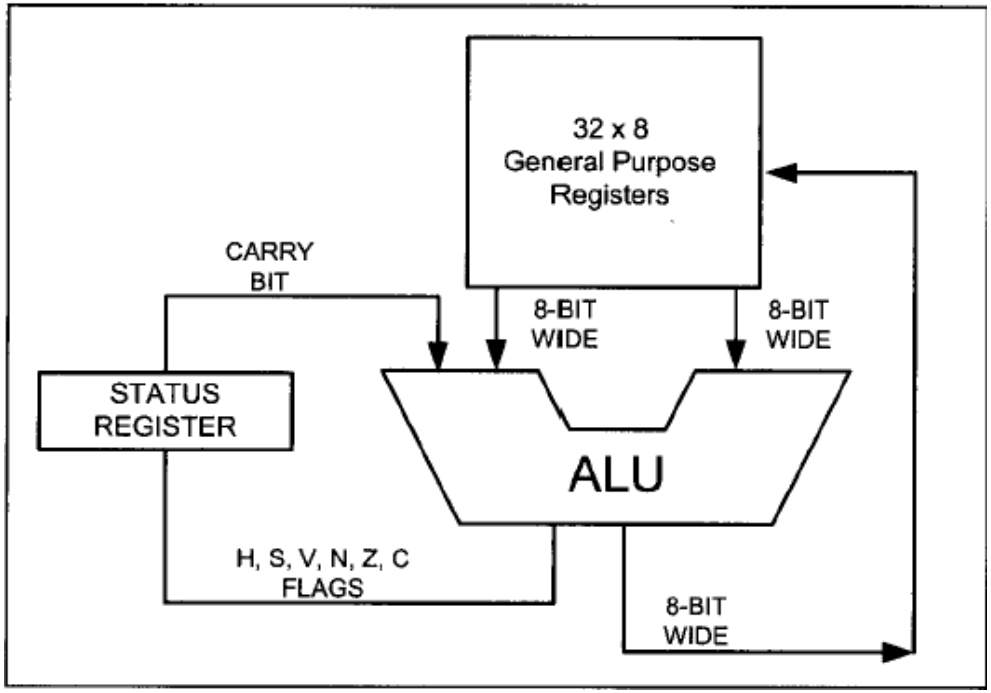
Some simple instructions

- 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
  - LDS (load from SRAM)
    - LDS R0,0x300
    - Load R0 by data from address 300



Some simple instructions

- 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



Some simple instructions

- Store

STS instruction (STore direct to data Space)

```
STS   K, Rr ;store register into location K
            ;K is an address between $0000 to $FFFF
```

- No immediate value store! Just stores can be done through registers

Some simple instructions

- IN and OUT

IN instruction (IN from I/O location)

```
IN      Rd, A ;load an I/O location to the GPR (0 ≤ d ≤31),(0 ≤ A ≤ 63)
```

OUT instruction (OUT to I/O location)

```
OUT A, Rr ;store register to I/O location (0 ≤ r ≤ 31),(0 ≤A ≤ 63)
```

- Each IO register can be addressed in two ways:
  - Memory address
  - IO address (relative to the beginning of the IO registers)
- “IN” and “OUT” use the IO address
- Example:
  - IN r1,0x16; // copy IO register no. 0x16 (memory address 0x36) to r1)

Some other instructions

MOV instruction

The MOV instruction is used to copy data among the GPR registers of R0–R31. It has the following format:

```
MOV   Rd,Rr          ;Rd = Rr (copy Rr to Rd)
                     ;Rd and Rr can be any of the GPRs
```

ALU instructions

Instruction		
ADD	Rd, Rr	ADD Rd and Rr
ADC	Rd, Rr	ADD Rd and Rr with Carry
AND	Rd, Rr	AND Rd with Rr
EOR	Rd, Rr	Exclusive OR Rd with Rr
OR	Rd, Rr	OR Rd with Rr
SBC	Rd, Rr	Subtract Rr from Rd with carry
SUB	Rd, Rr	Subtract Rr from Rd without carry

Rd and Rr can be any of the GPRs.

Single operand instructions

Instruction		
CLR	Rd	Clear Register Rd
INC	Rd	Increment Rd
DEC	Rd	Decrement Rd
COM	Rd	One's Complement Rd
NEG	Rd	Negative (two's complement) Rd
ROL	Rd	Rotate left Rd through carry
ROR	Rd	Rotate right Rd through carry
LSL	Rd	Logical Shift Left Rd
LSR	Rd	Logical Shift Right Rd
ASR	Rd	Arithmetic Shift Right Rd
SWAP	Rd	Swap nibbles in Rd

Instruction size

- Almost all instructions are 2 bytes
- Example:

```
LDI Rd, K    ;load register Rd with value K
```

$16 \leq d \leq 31, 0 \leq K \leq 255$

  - 1110: machine code for LDI
  - KKKK…= immediate value
  - dddd= destination register

Instruction size

- Another examples:

```
ADD Rd, Rr    ;Add Rd to Rr
```

0000	11rd	dddd	rrrr

$0 \le d \le 31,\ 0 \le r \le 31$

```
IN Rd, A   ;load from Address A of I/O memory into register Rd
```

1011	0AAd	dddd	AAAA

$0 \le d \le 31, 0 \le A \le 63$

Instruction size

- LDS Rd, k ;Load from memory location k to register Rd

1001	000d	dddd	0000
kkkk	kkkk	kkkk	kkkk

$0 \le d \le 31, 0 \le k \le 65535$

JMP k ;Jump to address k

1001	010k	kkkk	110k
kkkk	kkkk	kkkk	kkkk

$0 \le k \le 4M$

Directives

- Like directives in high level languages
- For code readability
- Not instructions that generate machine code after compile
- Example: EQU
  - To set a constant value
  - Equivalent to CONSTANT in C++

```
.EQU   COUNT = 0x25
       ...   ....
       LDI   R21, COUNT          ;R21 = 0x25
```

Directives

.ORG (origin)

The .ORG directive is used to indicate the beginning of the address. It can be used for both code and data.

.INCLUDE directive

The .include directive tells the AVR assembler to add the contents of a file to our program (like the #include directive in C language)

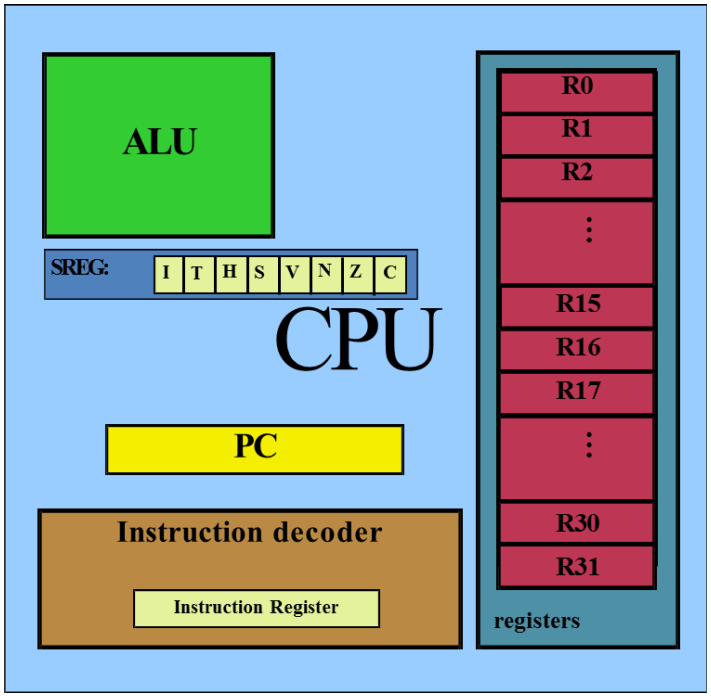
AVR data size

- AVR just has 8-bit data
- Programmer or compiler has to break larger data into 8-bit units

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
```



Code memory

- The program after compiling is stored in a ROM (flash memory)
  - To keep the code even when the system is powered off
  - Data is stored in another memory (SRAM)
- In AVR each code memory location is 2-bytes
  - In Atmega32, 32K flash memory is organized as 16Kx16 words
  - Needs 14-bit PC

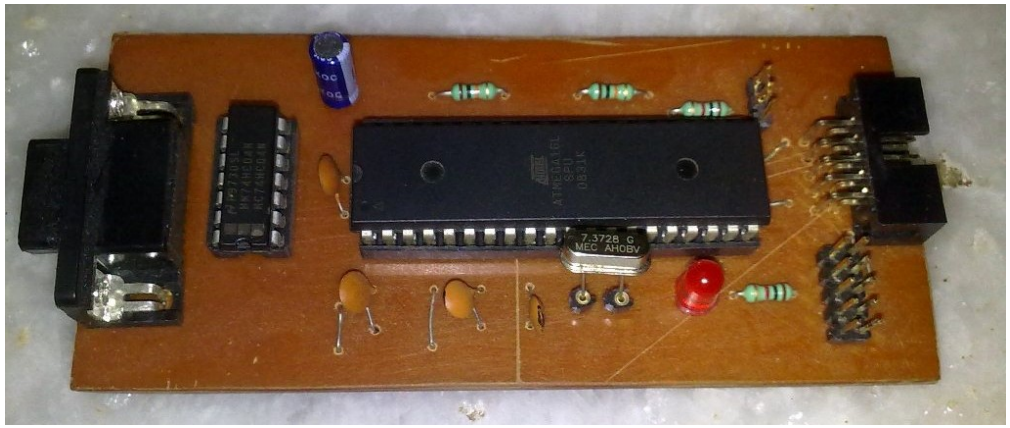
AVR programming

- Assembly: AVR Studio
- C: Code Vision

HP InfoTech
CodeVisionAVR

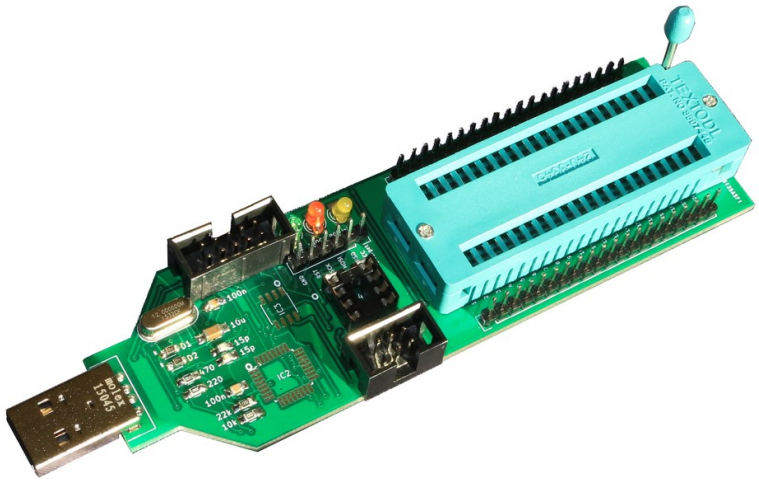
How to load our programs in AVRs?

- JTAG programming
  - Some chips have JTAG interface
  - Industry standard for debugging chips in circuit
  - Connect to special JTAG signals
  - Can program
    - FLASH
    - EEPROM
    - All fuses
    - Lock bits



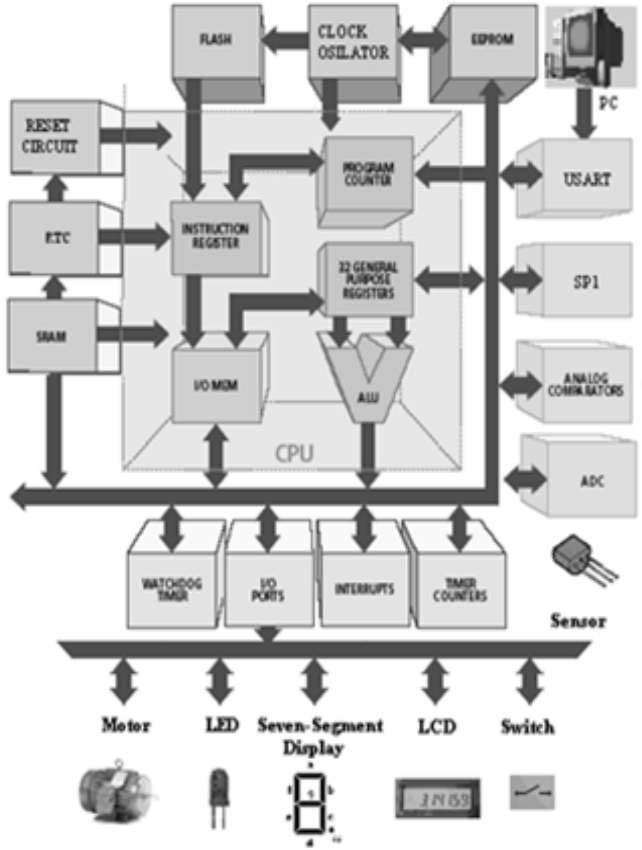
How to load our programs in AVRs?

- ISP (In-System Programmer) programming
  - Connect to 6 or 10 pin connector
  - SPI interface
  - Special cable required
  - Can program
    - FLASH
    - EEPROM
    - Some fuses
    - Lock bits



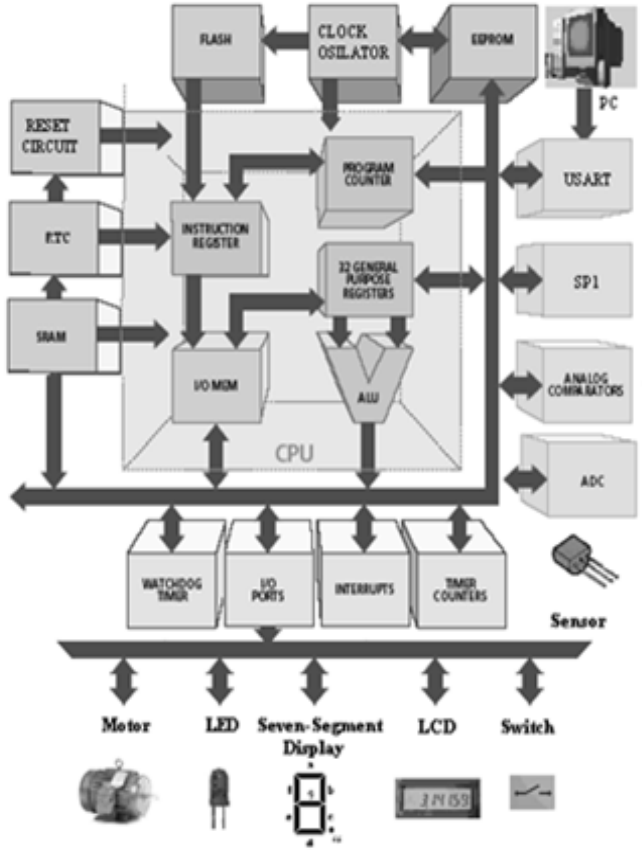
AVR microcontroller example: ATmega32

- 16 MHz clock frequency
- 40 pins
- 32 KB instruction memory (Flash)
- 1KB data memory (EEPROM)
- 2KB data memory (SRAM)
- SPI, USART, and I2C serial ports
- 3 timers
- 8 10-bit ADC channels



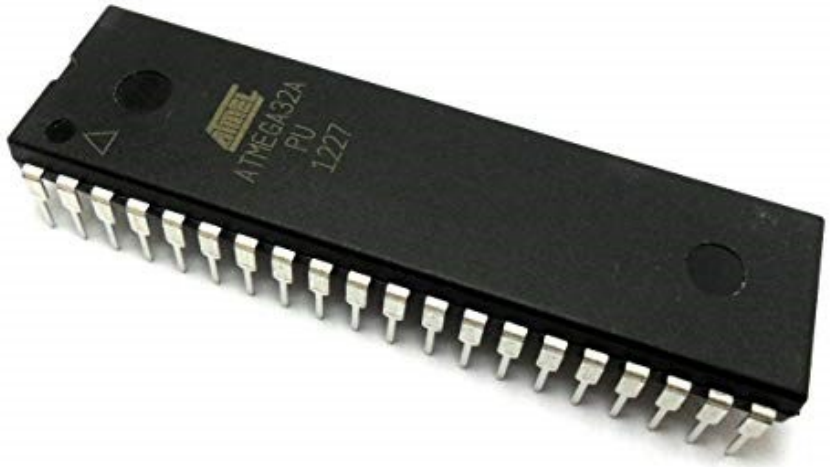
AVR microcontroller example: ATmega32

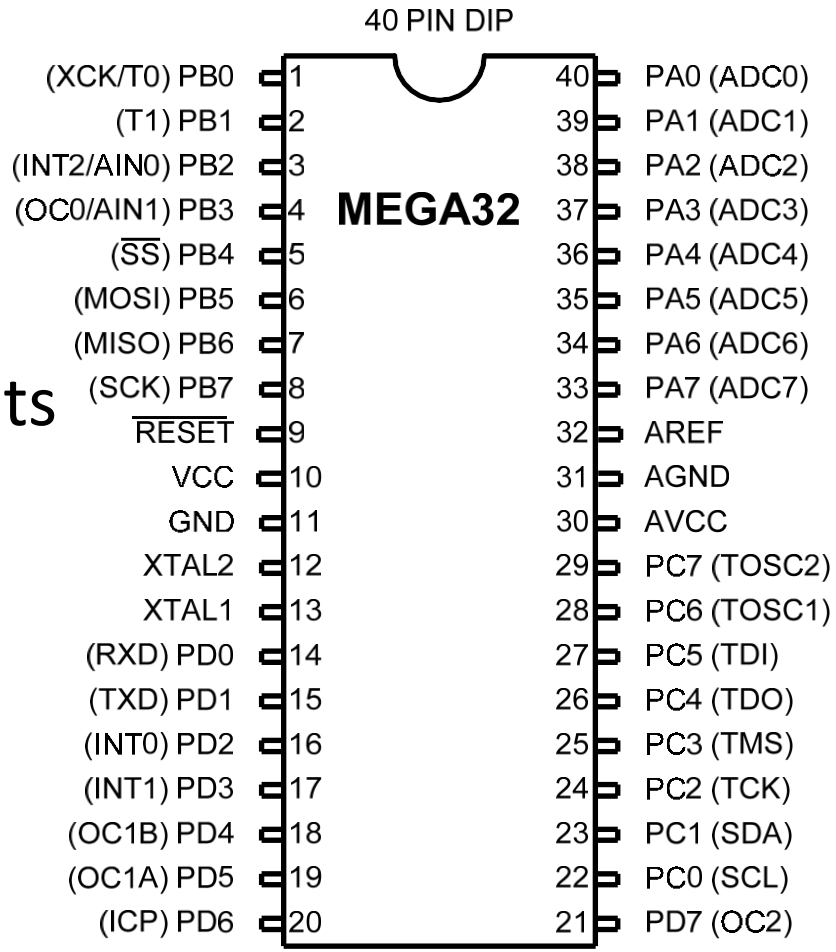
- Analog comparator
- 4 PWM ports
- .....



Pins

- 4×8-bit ports
- PA, PB, PC, PD
- Multiplexed with other in-outs





Fuses

- Fuses configure system parameters
  - Clock selection and options
  - Boot options
  - Some IO pin configurations
  - Reset options
- Three 8 bit fuse registers
  - Use caution! Some configurations can put the
  - Device in an unusable state!
  - We will cover some of the fuses in the following chapters
    - For more information, please refer to the datasheet of ATmega32!


The End