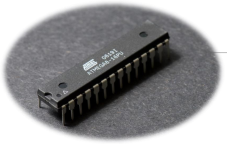
# CO321 : Embedded Systems

## Programming AVR microcontrollers



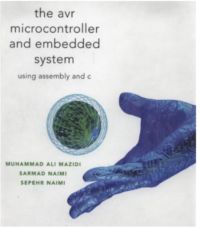
ISURU NAWINNE

Department of Computer Engineering
University of Peradeniya

## Outline

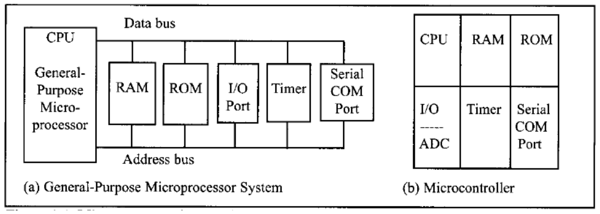
- Introduction to AVR microcontroller
- AVR architecture
- AVR programming in C
- AVR pin description and flashing

Reference :
The AVR microcontroller and embedded systems using assembly and C
chapter 1,2,7,8



## Introduction to AVR microcontroller

## Microcontroller vs microprocessor



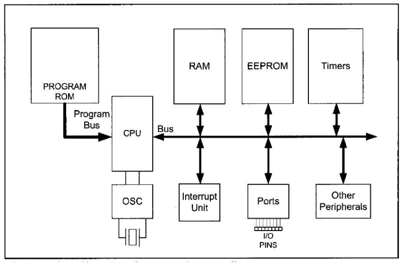
(a) General-Purpose Microprocessor System

(b) Microcontroller

## Different microcontrollers

- Microchip PIC
- Atmel AVR
- Zilog Z8
- Freescale Semiconductors (Formerly Motorola)
- Intel 8051

## A simplified view of an AVR microcontroller



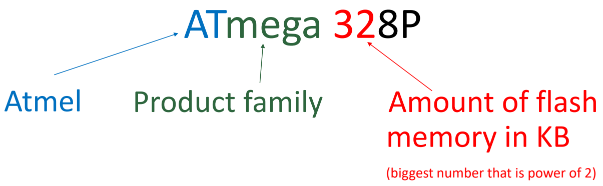
Harvard architecture :

Separate bus for instruction memory and data memory

## AVR families

| Family | Description |
|---|---|
| Classic (AT90Sxxxx) | Original AVR chip which is outdated now |
| Mega (ATmegaxxxx) | More than 120 instructions and lot of peripherals and hence suitable for most designs |
| Tiny (ATtinyxxxx) | Lesser instructions and smaller packages for low cost and low power applications |
| Special purpose | Considered a subset of other groups with special capabilities such as USB controller, Ethernet controller, LCD controller, etc... |

## Product naming scheme



ATmega 328P

Atmel

Product family

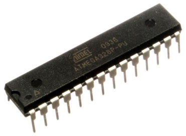
Amount of flash memory in KB

(biggest number that is power of 2)

There are some exceptions as well

## ATmega328P

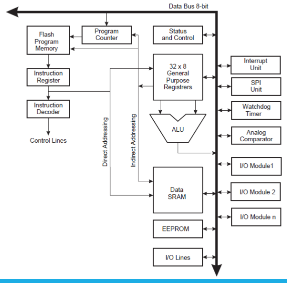
For our labs we use **ATmega328P** chip which is the microcontroller found on Arduino Uno.



Datasheet :
http://www.atmel.com/images/atmel-8271-8-bit-avr-microcontroller-atmega48a-48pa-88a-88pa-168a-168pa-328-328p_datasheet_complete.pdf

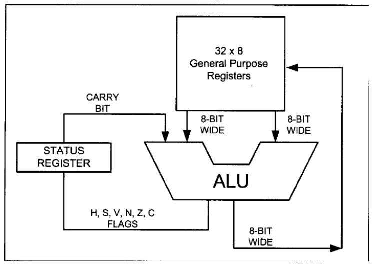
8 bit processor, 131 instructions, 32 general purpose registers
32KB FLASH, 2KB SRAM, 1KB EEPROM
3 timers, 10 bit ADC, USART, SPI and many other peripherals

## ATmega328P block diagram



# AVR architecture

## AVR CPU core

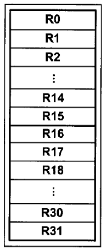


## General Purpose Registers (GPRs)

- 32 GPRs
- All are 8-bit
- Located at lowest memory addresses
- Can be used by any arithmetic or logical instruction
- GPR in AVR are the same as the accumulator in other microprocessors

Example usage by add instruction :

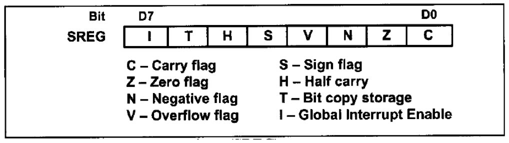
ADD R16,R17 ; R16=R16+R17



| R0 |
|---|
| R1 |
| R2 |
| ⋮ |
| R14 |
| R15 |
| R16 |
| R17 |
| R18 |
| ⋮ |
| R30 |
| R31 |

## Status Register (SReg)

- This is the flag register in AVR
- 8-bit register
- Corresponding flags in status register are set by the execution of arithmetic or logical instructions



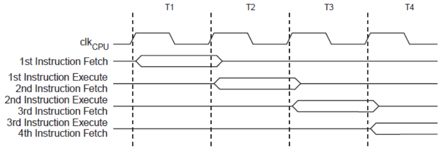
| Bit | D7 | | | | | | | D0 |
|---|---|---|---|---|---|---|---|---|
| SREG | I | T | H | S | V | N | Z | C |

C – Carry flag
Z – Zero flag
N – Negative flag
V – Overflow flag
S – Sign flag
H – Half carry
T – Bit copy storage
I – Global Interrupt Enable

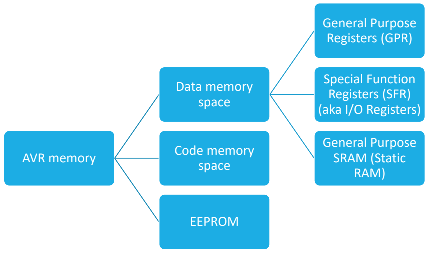
## Instruction set architecture

- AVR is RISC
- Most instructions take single clock cycle
- 2 stage pipeline

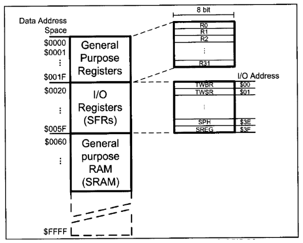
The Parallel Instruction Fetches and Instruction Executions



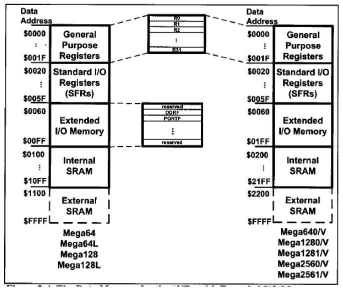
## AVR memory architecture


AVR memory
Data memory space
Code memory space
EEPROM
General Purpose Registers (GPR)
Special Function Registers (SFR) (aka I/O Registers)
General Purpose SRAM (Static RAM)

## AVR Data Memory

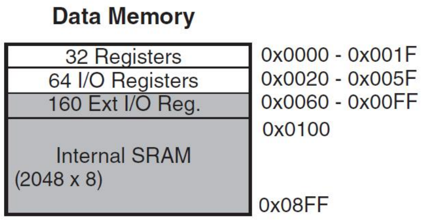


Without extended I/O



With extended I/O

## Data memory in 328P



## Data memory

**GPR**

- 32 bytes of data memory space from 00-FF in memory space (already discussed)

**I/O memory (SFR)**

- Dedicated for specific functions such as status register, timers, serial communication, I/O ports, ADC and so on
- I/O memory is made up of 8-bit registers

**SRAM**

- Used for storing data and parameters
- Called the scratch pad
- Each location is 8-bit wide

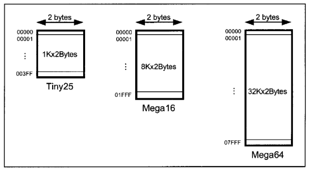
## EEPROM

Used for storing data that are rarely changed or should not lost when power is off

## Program memory

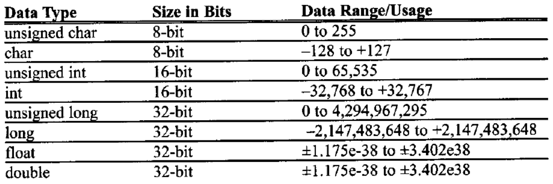
- Stores the program code
- Made up of flash memory
- Each memory location is 2-bytes wide

Atmega 328P has 32KB of Flash memory



# AVR programming in C

## Datatypes in AVR C

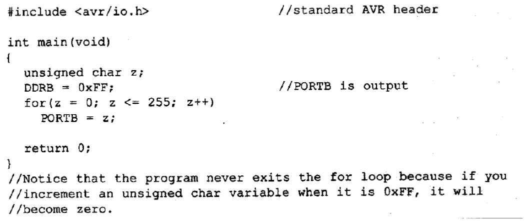


| Data Type | Size in Bits | Data Range/Usage |
|---|---|---|
| unsigned char | 8-bit | 0 to 255 |
| char | 8-bit | –128 to +127 |
| unsigned int | 16-bit | 0 to 65,535 |
| int | 16-bit | –32,768 to +32,767 |
| unsigned long | 32-bit | 0 to 4,294,967,295 |
| long | 32-bit | –2,147,483,648 to +2,147,483,648 |
| float | 32-bit | ±1.175e-38 to ±3.402e38 |
| double | 32-bit | ±1.175e-38 to ±3.402e38 |

Memory in a microcontroller is limited. Hence use the suitable data type.

## Example 1

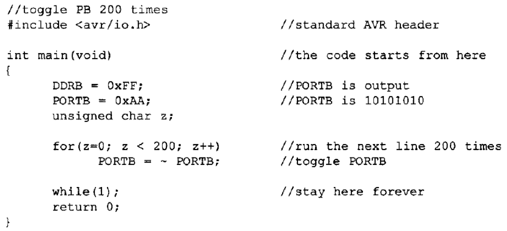
Write an AVR C program to send values 00-FF to Port B.



```
#include <avr/io.h>                    //standard AVR header

int main(void)
{
  unsigned char z;
  DDRB = 0xFF;                         //PORTB is output
  for(z = 0; z <= 255; z++)
    PORTB = z;

  return 0;
}
//Notice that the program never exits the for loop because if you
//increment an unsigned char variable when it is 0xFF, it will
//become zero.
```

## Example 2

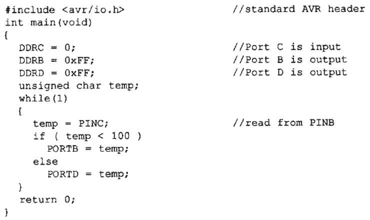
Write an AVR C program to toggle all the bits of Port B 200 times.



```
//toggle PB 200 times
#include <avr/io.h>                    //standard AVR header

int main(void)                         //the code starts from here
{
      DDRB = 0xFF;                     //PORTB is output
      PORTB = 0xAA;                    //PORTB is 10101010
      unsigned char z;

      for(z=0; z < 200; z++)           //run the next line 200 times
            PORTB = ~ PORTB;           //toggle PORTB

      while(1);                        //stay here forever
      return 0;
}
```

## Example 3

Write an AVR C program to get a byte of data from Port C. If it is less than 100, send it to Port B, otherwise send it to Port D.



```
#include <avr/io.h>                    //standard AVR header
int main(void)
{
  DDRC = 0;                            //Port C is input
  DDRB = 0xFF;                         //Port B is output
  DDRD = 0xFF;                         //Port D is output
  unsigned char temp;
  while(1)
  {
    temp = PINC;                       //read from PINB
    if ( temp < 100 )
      PORTB = temp;
    else
      PORTD = temp;
  }
  return 0;
}
```

## Setting a single bit

Set only bit 4 of Port B without disturbing other pins of Port B.

Method 1 : PORTB = PORTB | 0b00010000

Method 2 : PORTB = PORTB | 16

Method 3 : PORTB = PORTB | 0x10

Method 4 : PORTB = PORTB | (1<<4)

Next slides are only demonstrated via method 4 but any method is usable.

## Clearing a single bit

Clear only bit 4 of Port B without disturbing other pins of Port B.

PORTB = PORTB & ~(1<<4)

## Toggling a single bit

Toggle only bit 4 of Port B without disturbing other pins of Port B.

PORTB = PORTB ^ (1<<4)

## Checking a single bit

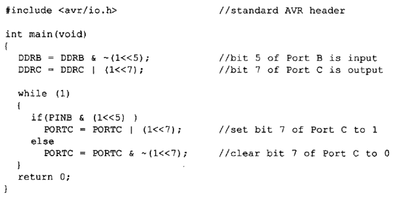
Check if bit 4 of Port B is set to 1.

Method 1 : if ((PORTB >> 4 ) & 1){……….}

Method 2 : if ( PORTB & (1<<4) ) {……….}

## Example

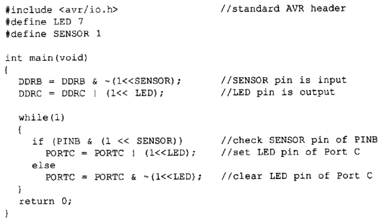
Write an AVR C program to get the status of bit 5 of port B and send it to pin 7 of port C continuously.



```
#include <avr/io.h>                   //standard AVR header

int main(void)
{
  DDRB = DDRB & ~(1<<5);              //bit 5 of Port B is input
  DDRC = DDRC | (1<<7);               //bit 7 of Port C is output

  while (1)
  {
    if(PINB & (1<<5) )
      PORTC = PORTC | (1<<7);         //set bit 7 of Port C to 1
    else
      PORTC = PORTC & ~(1<<7);        //clear bit 7 of Port C to 0
  }
  return 0;
}
```

## Example

A door sensor is connected to the Port B pin 1, and an LED is connected to Port C pin 7. Write an AVR C program to monitor the door sensor and, when it opens, turn on the LED.



```
#include <avr/io.h>                   //standard AVR header
#define LED 7
#define SENSOR 1

int main(void)
{
  DDRB = DDRB & ~(1<<SENSOR);         //SENSOR pin is input
  DDRC = DDRC | (1<< LED);            //LED pin is output

  while(1)
  {
    if (PINB & (1 << SENSOR))         //check SENSOR pin of PINB
      PORTC = PORTC | (1<<LED);       //set LED pin of Port C
    else
      PORTC = PORTC & ~(1<<LED);      //clear LED pin of Port C
  }
  return 0;
}
```

## Changing multiple bits

Set only bit 4 and bit 7 of Port B without disturbing other pins of Port B.

PORTB = PORTB | ((1<<7)|(1<<4))

## AVR pin description and flashing

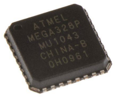
## Different package types

ATmega microcontrollers come in different packages

- DIP (Dual in-line package)
- MLF (Micro Lead Frame Package)
- QFP (Quad Flat Package)

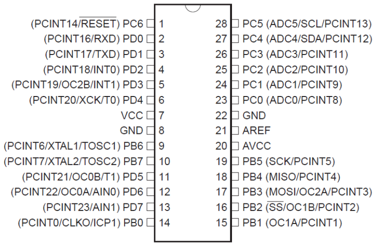
DIP

QPF



MLF

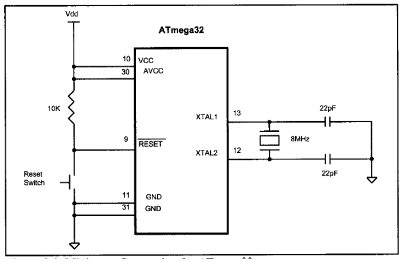
## Atmel 328P DIP



## Pin descriptions

| Pin | Description |
|---|---|
| VCC | Power supply pin. The typical voltage source is 5V |
| AVCC | Supply voltage pin for A/D converter. It should be connected even if A/D is not used |
| GND | Two pins used for ground |
| XTAL1 and XTAL2 | To connect quartz crystal oscillator as the clock source |
| RESET | When LOW pulse is microcontroller will reset |

## Minimal connection for ATmega 32



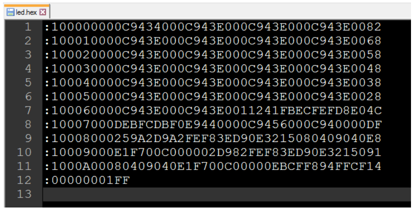
## Ports

| Port | Description |
|---|---|
| Port B (PB7:0) | 8-bit bi-directional I/O port |
| Port C (PC5:0) | 7-bit bi-directional I/O port |
| Port D (PD7:0) | 8-bit bi-directional I/O port |

Note that most pins have alternate functions which change depending on the configuration.

## HEX files for AVR

Intel HEX is a widely used file format designed to standardize the loading (transferring) of executable machine code into a chip



```
:100000000C9434000C943E000C943E000C943E0082
:100010000C943E000C943E000C943E000C943E0068
:100020000C943E000C943E000C943E000C943E0058
:100030000C943E000C943E000C943E000C943E0048
:100040000C943E000C943E000C943E000C943E0038
:100050000C943E000C943E000C943E000C943E0028
:100060000C943E000C943E0011241FBECFEFD8E04C
:10007000DEBFCDBF0E9440000C9456000C940000DF
:100080000259A2D9A2FEF83ED90E3215080409040E8
:10009000E1F700C000002D982FEF83ED90E3215091
:1000A00080409040E1F700C00000EBCFF894FFCF14
:00000001FF
```

## Intel HEX file

Since the programmer (loader) uses the HEX file to download opcode into Flash, following information are provided

- Number of bytes of information to be loaded
- Information to be loaded
- Starting address where information must be placed

## Intel HEX file

Each line of HEX file has six parts

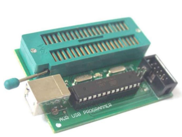
:BBAAAATTHHHHH.......HHHHCC

- Each line starts with a ':'
- BB tells how many bytes are in the line (in hexadecimal)
- AAAA is the 16-bit address at which the loader should place the first byte
- TT is the type. 00 means there are more lines to come after this line, 01 means the last line
- HH.....H is the real information
- CC is the checksum for everything in the line

## AVR programming

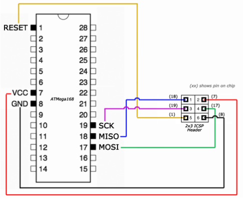
There are 3 ways to load a program to the flash memory

- Parallel programming
  - Chip is programmed before being inserted to the circuit or the chip is removed and reprogrammed
  - ZIF (Zero insertion force) sockets are used



## AVR programming

- ISP (In-circuit Serial Programming)
  - Chip is programmed while it is on the circuit



## AVR programming

- Bootloader
  - A piece of code burned into microcontroller's flash
  - It communicates with the user's board via serial port / USB/ network
  - Drawback : requires a communication port and program space on the microcontroller
  - Advantage : convenience
  - Arduino has bootloader
  - We use this method for programming