ICT 5307: Embedded System Design

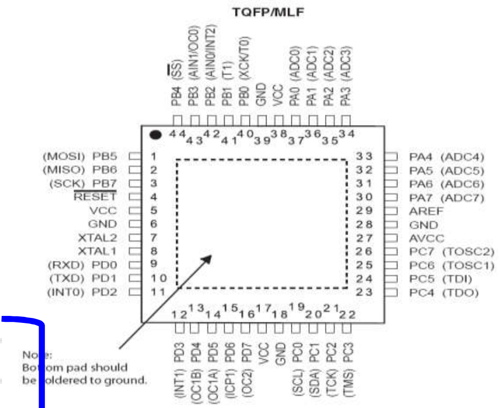
Lecture 2

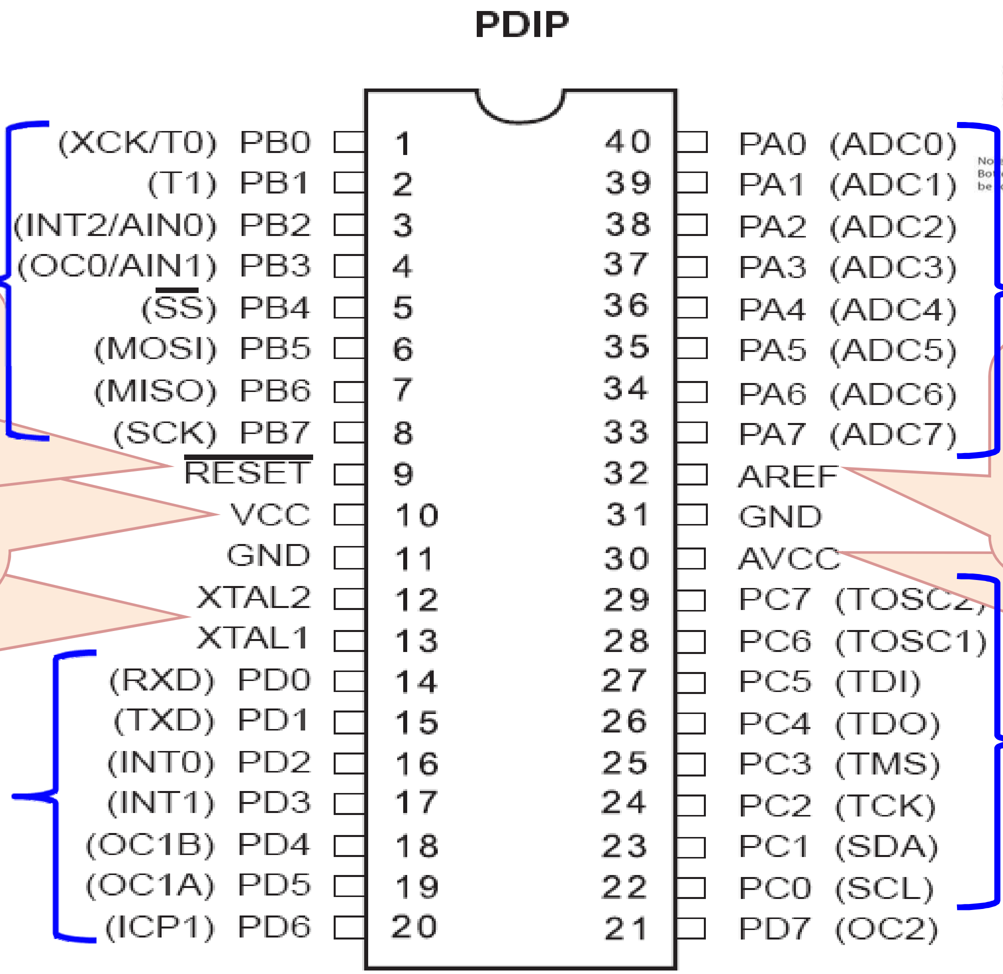
AVR Architecture and Organization
Input-Output Ports

Professor S.M. Lutful Kabir

IICT, BUET

ATMega32 Pin out & Descriptions





Port B

Clears all the registers and restart the execution of program

b.

RC oscillator

Port A

Reference voltage for ADC

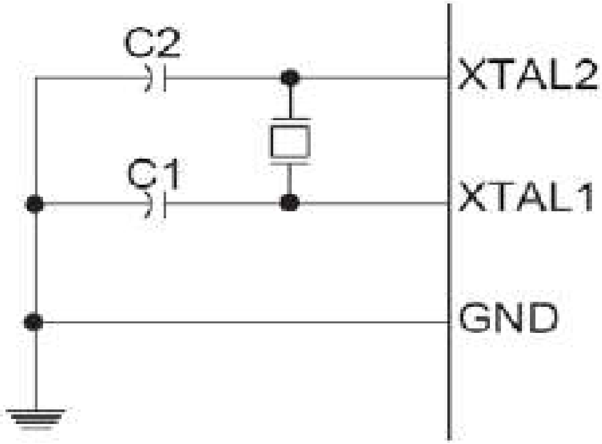
Connect it to VCC

Port C

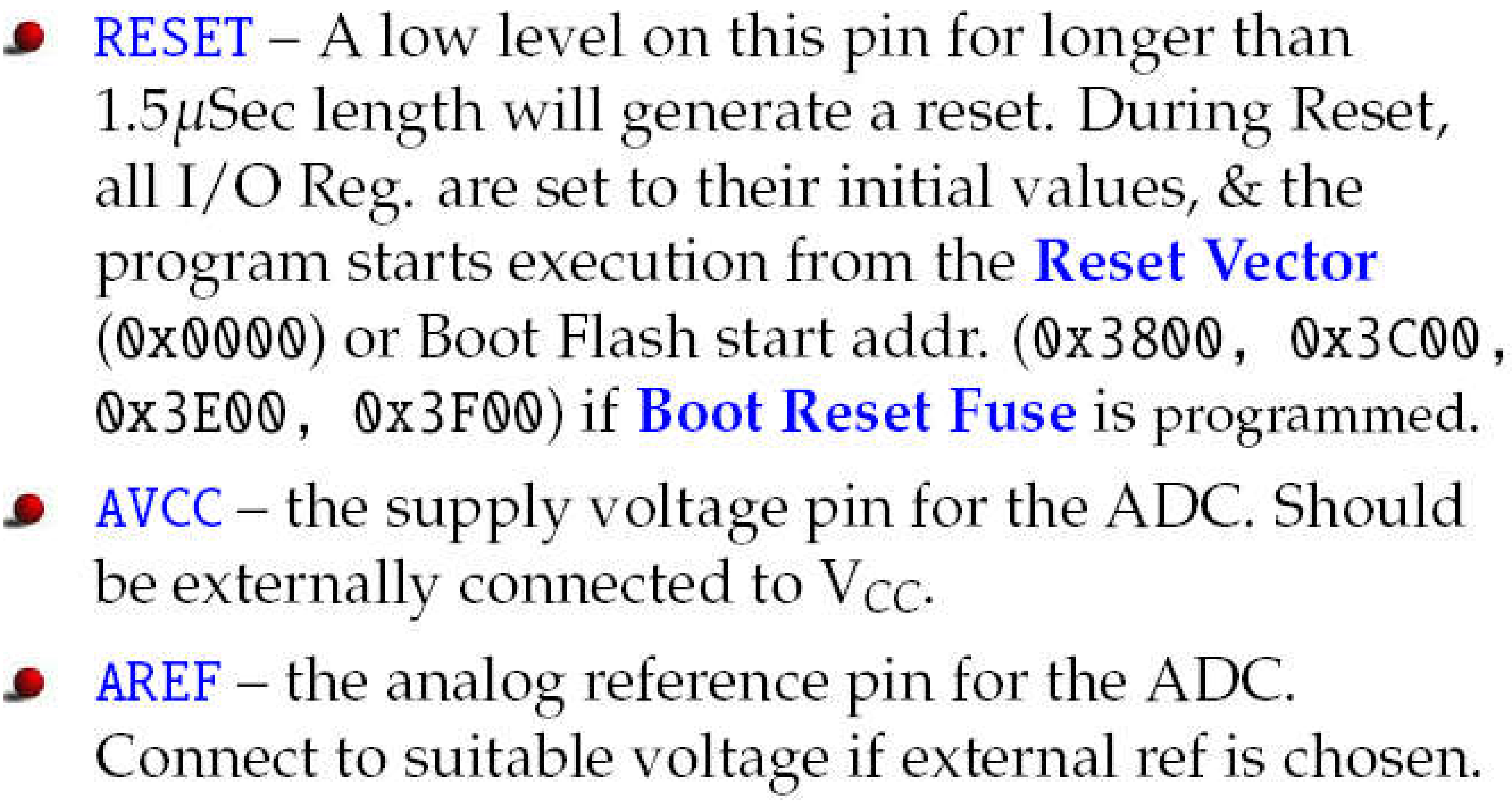
Port D

ATMega32 Pin out & Descriptions

- VCC & GND – Digital supply voltage for ATmega32. Typical operating voltages are 2.7 – 5.5V. Should never exceed 6.0V !
- XTAL1 & XTAL2 – Connections to ceramic/crystal on-chip oscillator for generating internal clocks. Crystal connection as below where typical values for C1,C2 are 22pF. Max crystal freq. is 16MHz.

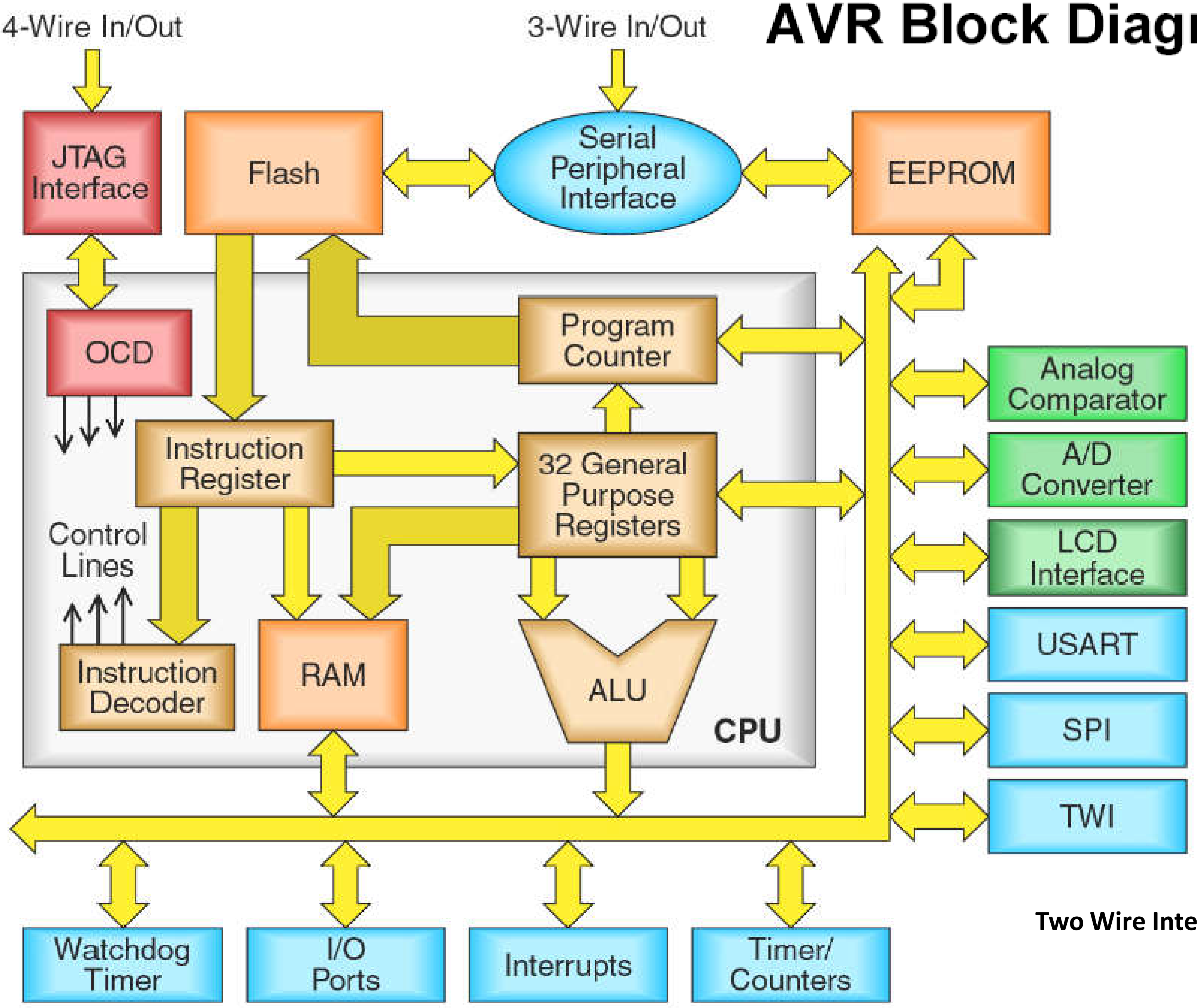


ATMega32 Pin out & Descriptions



- RESET – A low level on this pin for longer than 1.5μSec length will generate a reset. During Reset, all I/O Reg. are set to their initial values, & the program starts execution from the **Reset Vector** (`0x0000`) or Boot Flash start addr. (`0x3800, 0x3C00, 0x3E00, 0x3F00`) if **Boot Reset Fuse** is programmed.
- AVCC – the supply voltage pin for the ADC. Should be externally connected to V_{CC}.
- AREF – the analog reference pin for the ADC. Connect to suitable voltage if external ref is chosen.

AVR Block Diagram



4-Wire In/Out

3-Wire In/Out

JTAG Interface

Flash

Serial Peripheral Interface

EEPROM

OCD

Program Counter

Instruction Register

32 General Purpose Registers

Control Lines

Instruction Decoder

RAM

ALU

CPU

Analog Comparator

A/D Converter

LCD Interface

USART

SPI

TWI

Watchdog Timer

I/O Ports

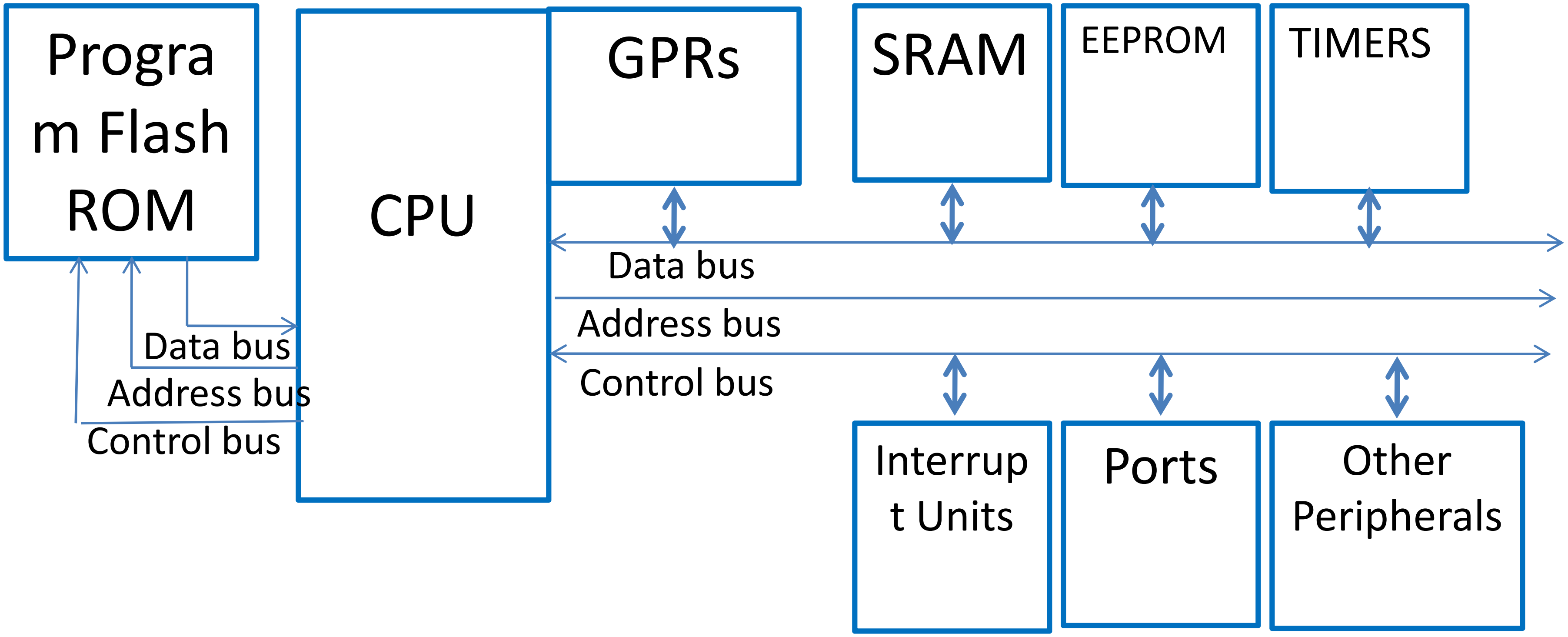
Interrupts

Timer/ Counters

On Chip Debugger

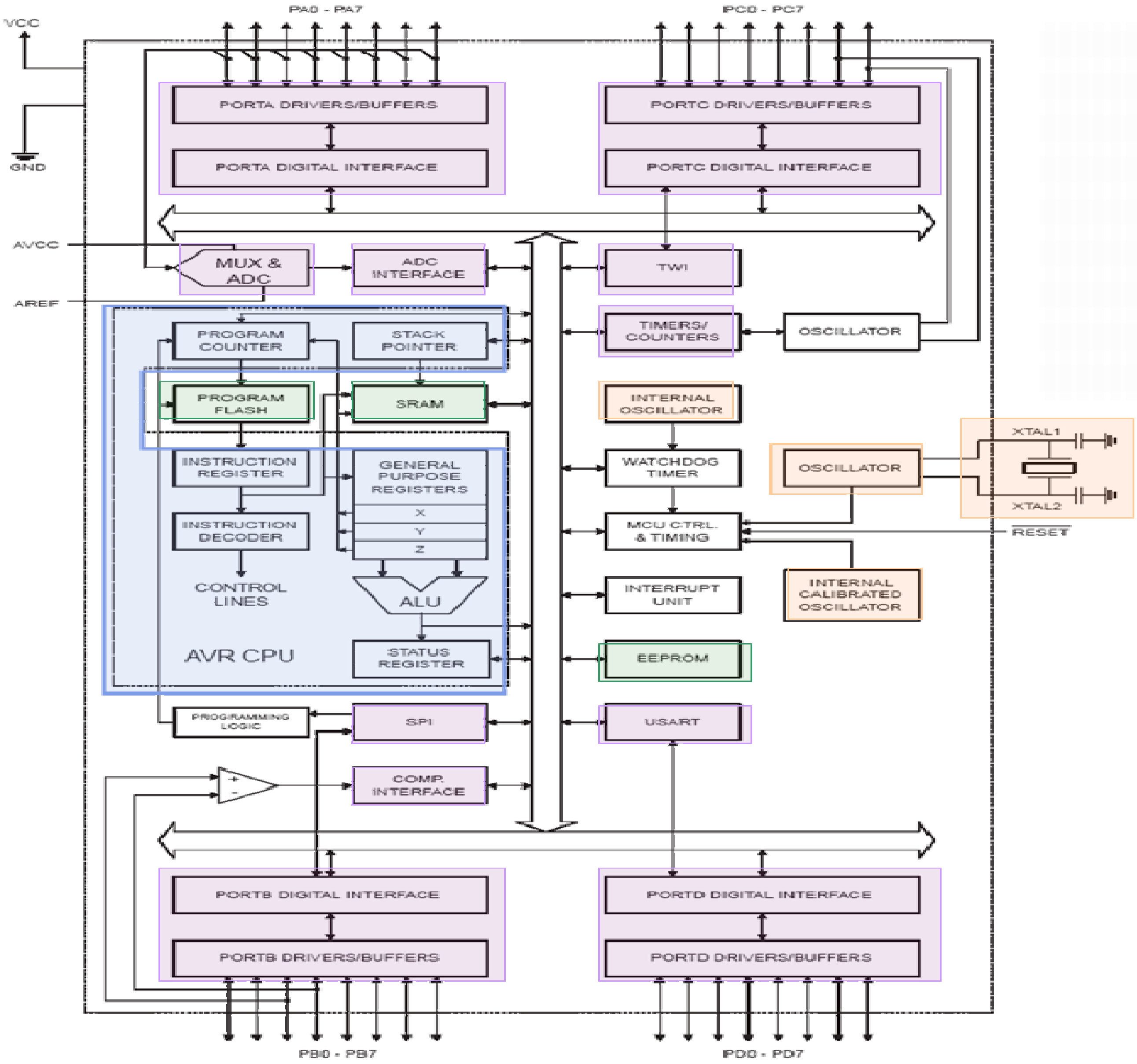
Two Wire Interface

Harvard Architecture in AVR



ATMega32 Architecture

- Native data size is 8 bits (1 byte).
- Uses 16-bit data addressing allowing it to address 2^{16} = 65536 unique addresses.
- Has three separate on-chip memories
  - 2KB SRAM
  - 1KB EEPROM
  - 32KB Flash
- I/O ports A-D
  - Digital input/output
  - Analog input
  - Serial/Parallel
  - Pulse accumulator



ATMega32 Programmer Model: Memory

1. **2KB SRAM**

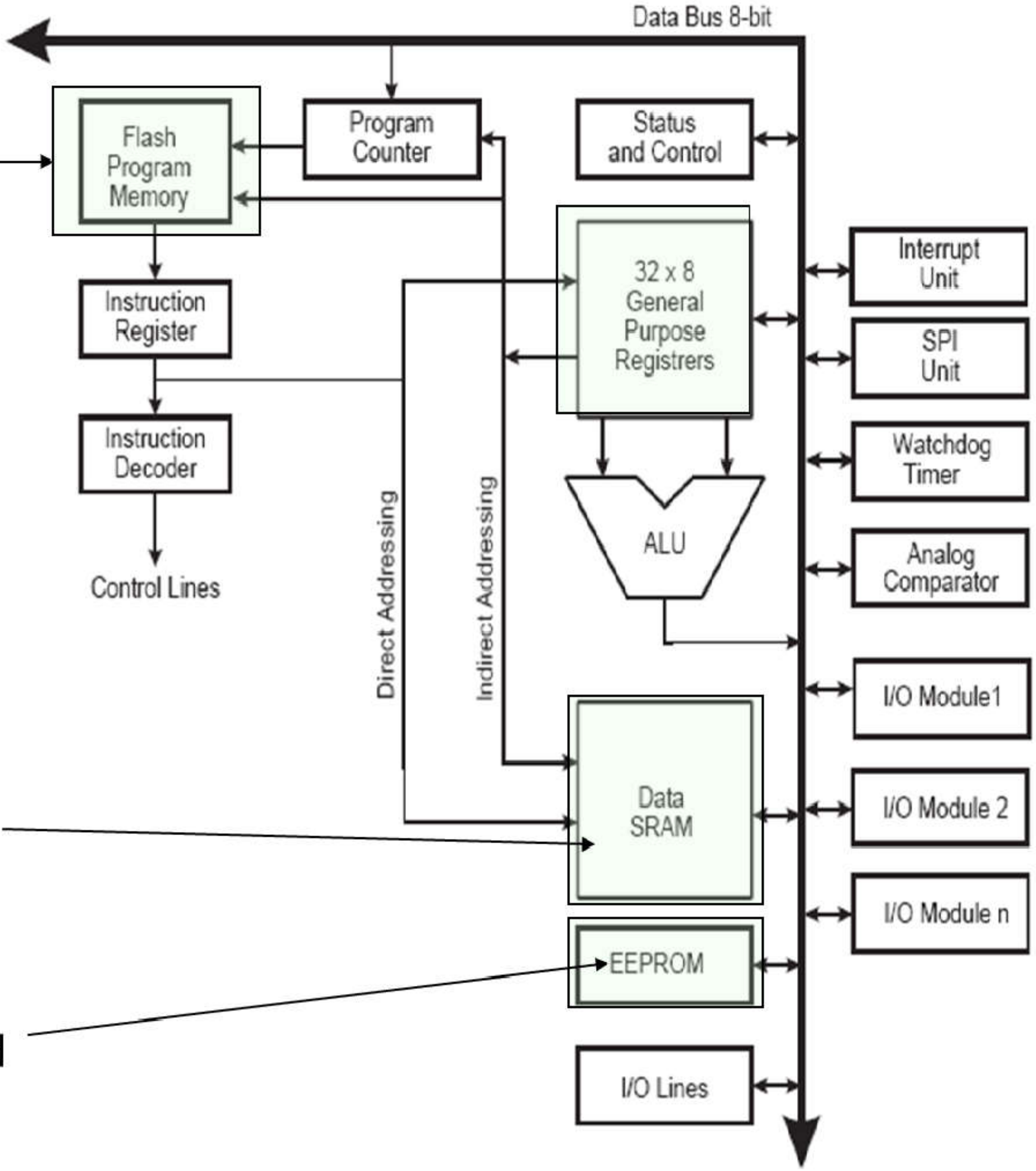
– For temporary data storage
– Memory is lost when power is shut off (volatile)
– Fast read and write

2. **1KB EEPROM**

– For persistent data storage
– Memory contents retain when power is off (non-volatile)
– Fast read; slow write
– Can write individual byte

3. **32KB Flash Program Memory**

– Used to store program code
– Memory contents retain when power is off (non-volatile)
– Fast to read; slow to write
– Can only write entire “blocks” of memory at a time
– organized in 16-bit words (16KWords)



Programs are stored here

Temporary data is stored here

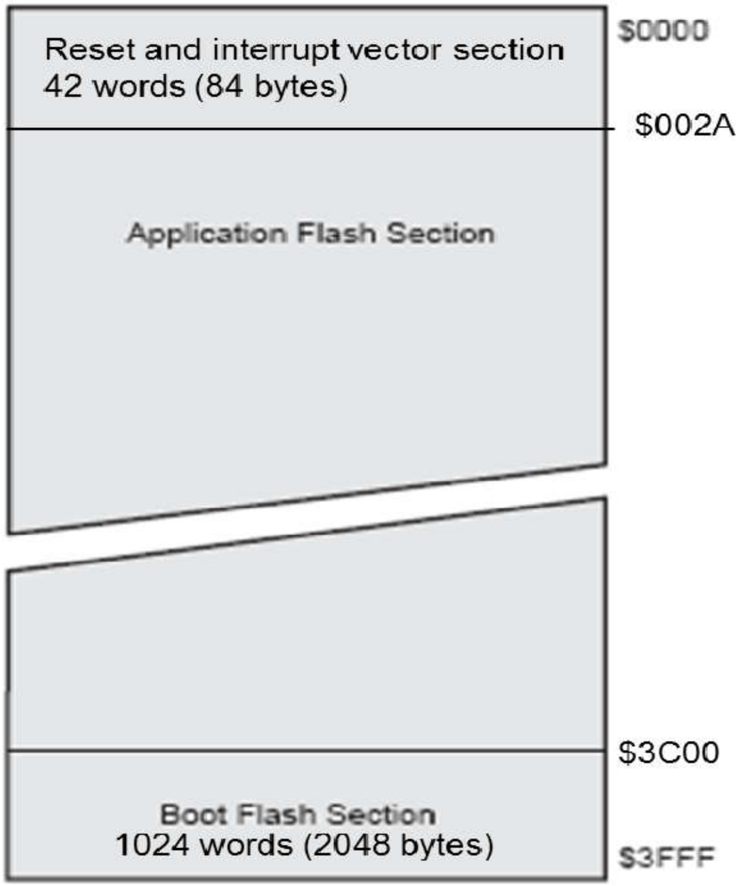
Permanent data is stored here

ATMega32 Programmer Model: Memory

Type	Flash		RAM		EEPROM	
	F_END	Size, kB	RAM-END	Size, kB	E_END	Size, kB
Atmega8	$0FFF	8	$045F	1	$1FF	0.5
Atmega32	**$3FFF**	**32**	**$085F**	**2**	**$3FF**	**1**
Atmega64	$7FFF	64	$10FF	4	$7FF	2
Atmega128	$FFFF	128	$10FF	4	$FFF	4

ATMega32 Programmer Model: Program Memory

Flash Memory Layout



- There are 32KB of program memory (Flash memory)
  - Organized as 16K 2-byte **words**
  - Because program instructions are either 2 or 4 bytes long
- Each word (not byte) in memory has a unique address
  - Beginning address $0000
  - Ending address $3FFF
- Some memory is reserved or protected
  - First 42 words (reserved)
  - Last 1024 words (protected)

ATMega32 Programmer Model: Data Memory

EEPROM

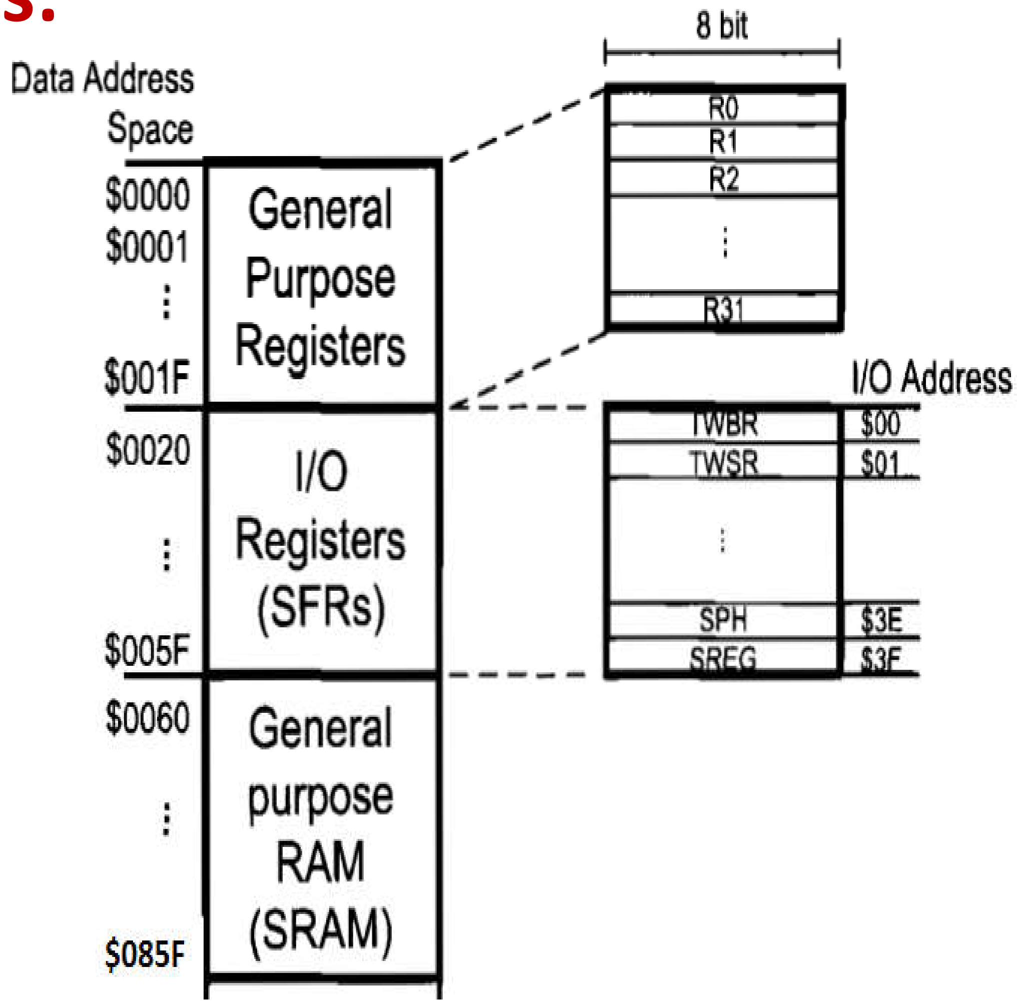
- ATmega32 contains 1024 bytes of data EEPROM memory.
- It is organized as a separate data space, in which single bytes can be read and written.
- The EEPROM has an endurance of at least 100,000 write/erase cycles.
- Different chip have different size of EEPROM memory

Chip	Bytes	Chip	Bytes	Chip	Bytes
ATmega8	512	ATmega16	512	ATmega32	1024
ATmega64	2048	ATmega128	4096	ATmega256RZ	4096
ATmega640	4096	ATmega1280	4096	ATmega2560	4096

ATMega32 Programmer Model: Data Memory

The data memory is composed of three parts:

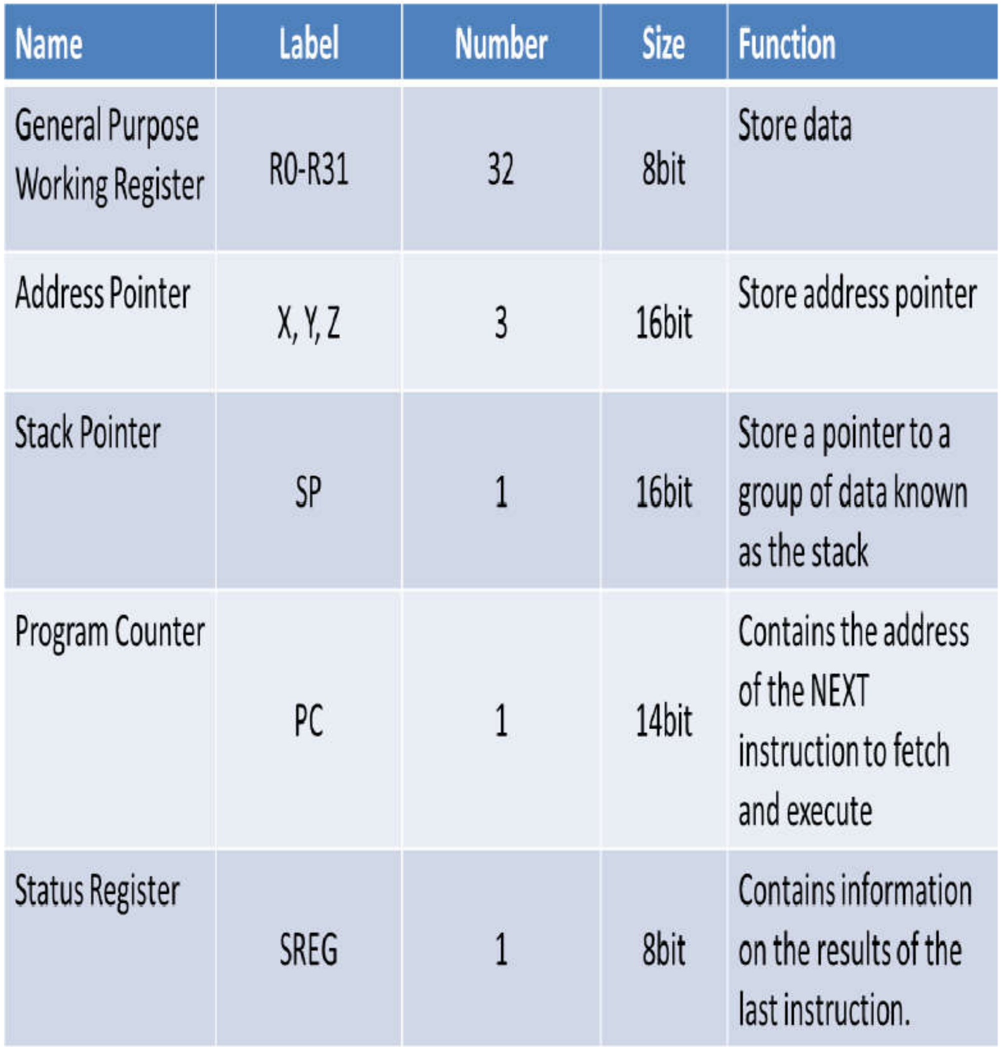
- **GPRs (general purpose registers),**
- **Special Function Registers (SFRs), and**
- **Internal data SRAM.**



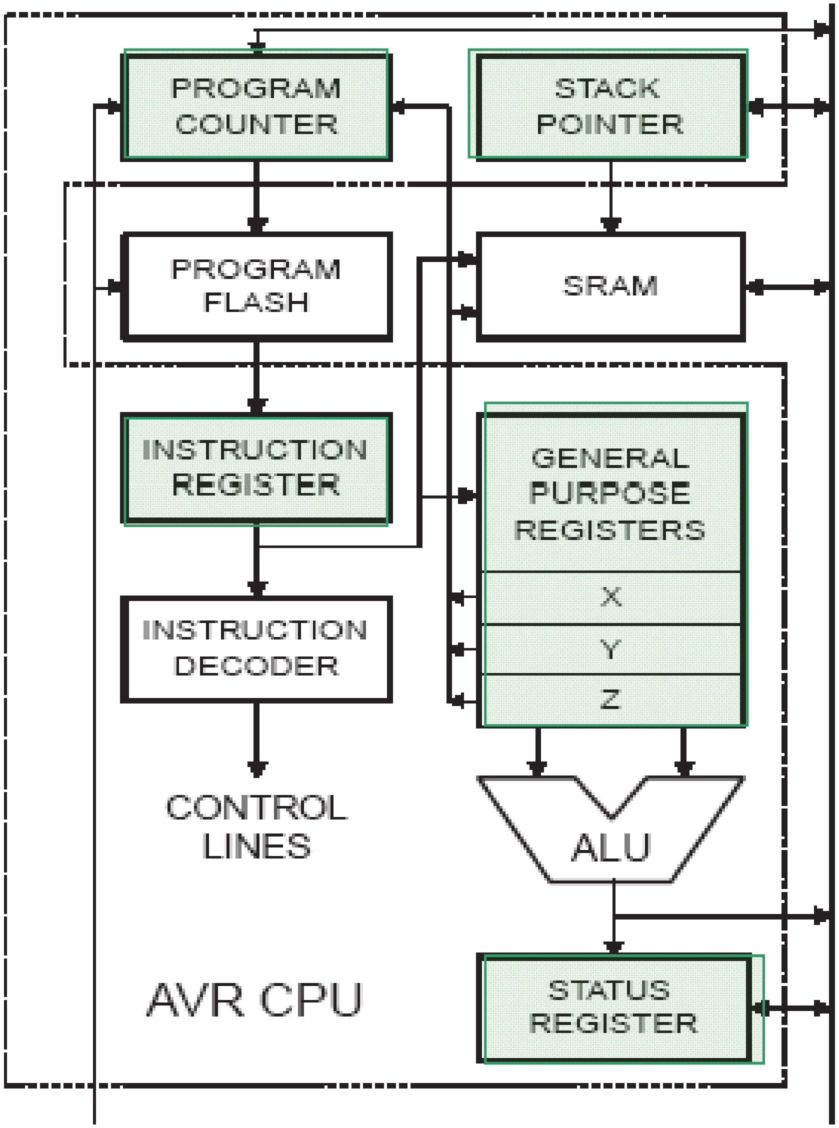
ATMega32 Programmer Model: Internal SRAM

- Internal data SRAM is widely used for storing data and parameters by AVR programmers and C compilers.
- Each location of the SRAM can be accessed directly by its address.
- Each location is 8 bit wide and can be used to store any data we want.
- Size of SRAM varies from chip to chip, even among members of the same family.

ATMega32 Programmer Model: Registers



Name	Label	Number	Size	Function
General Purpose Working Register	R0-R31	32	8bit	Store data
Address Pointer	X, Y, Z	3	16bit	Store address pointer
Stack Pointer	SP	1	16bit	Store a pointer to a group of data known as the stack
Program Counter	PC	1	14bit	Contains the address of the NEXT instruction to fetch and execute
Status Register	SREG	1	8bit	Contains information on the results of the last instruction.



LDI (Load immediate)

- **LDI Rd, K**; load Rd (destination) with immediate value K and d must be between 16 and 31)
- Examples:
  - LDI R20, 0x25
  - LDI R31, 0x87
  - LDI R25, 0x79
- LDI R5, 0x34 ; invalid instruction

ADD (Addition) Instruction

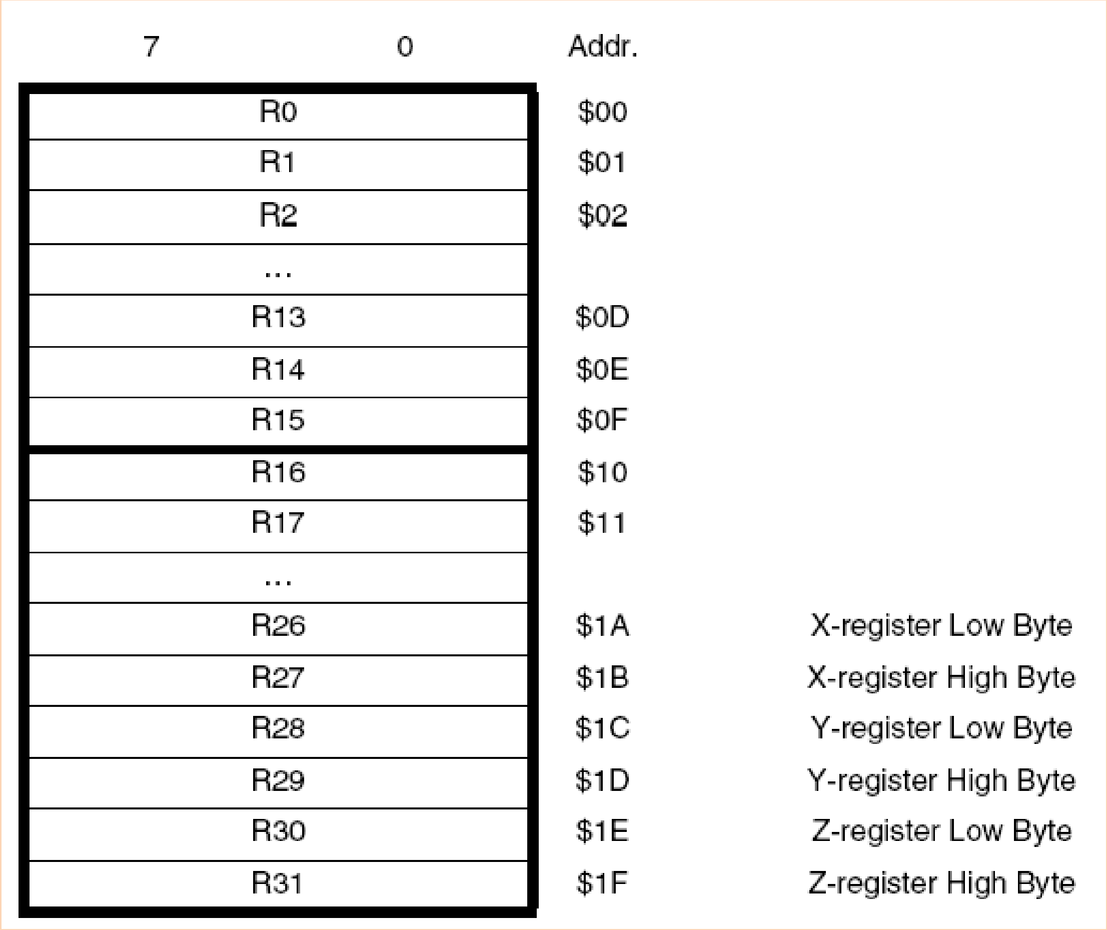
- **ADD Rd, Rr** ; ADD Rr to Rd and store the result in Rd
- Adds the content of Rr with that of Rd and the result is stored in Rd
  - LDI R16, 0x25
  - LDI R17, 0x 34
  - ADD R16, R17

ATMega32 Programmer Model: Registers (GPRs)

- The fast-access Register file contains 32 x 8-bit general purpose working registers with a single clock cycle access time.
- This allows single-cycle Arithmetic Logic Unit (ALU) operation.
- In a typical ALU operation, two operands are output from the Register file, the operation is executed, and the result is stored back in the Register file –in one clock cycle.”

ATMega32 Programmer Model: Registers (GPRs)

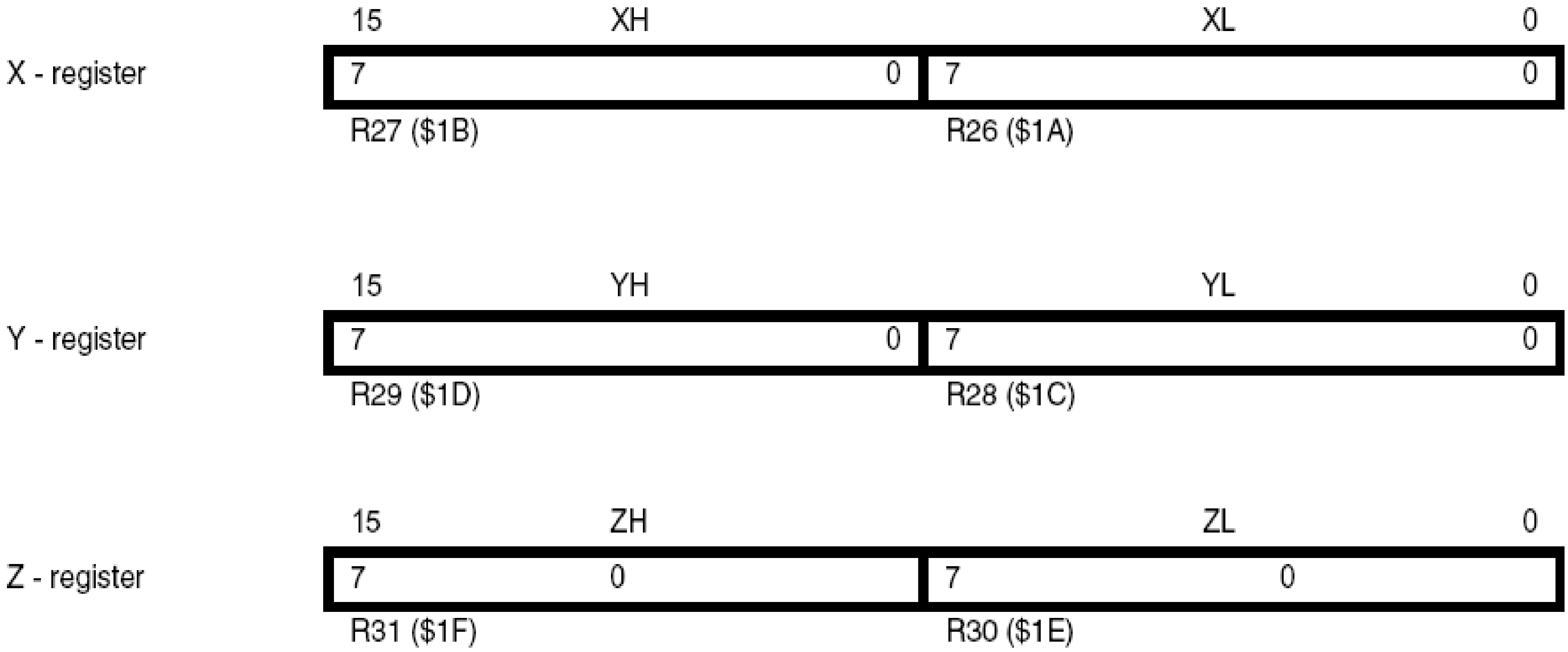
Six of the 32 registers can be used as three 16-bit indirect address register pointers : X-register, Y-register and Z-register



7 0	Addr.	
R0	$00	
R1	$01	
R2	$02	
...		
R13	$0D	
R14	$0E	
R15	$0F	
R16	$10	
R17	$11	
...		
R26	$1A	X-register Low Byte
R27	$1B	X-register High Byte
R28	$1C	Y-register Low Byte
R29	$1D	Y-register High Byte
R30	$1E	Z-register Low Byte
R31	$1F	Z-register High Byte

ATMega32 Programmer Model: Registers (GPRs)

- ❑ In the different addressing modes these address registers have functions as fixed displacement, automatic increment, and automatic decrement



Using Instructions with the Data memory

- LDI and ADD instructions are limited to General purpose registers (R16 to R31)
- To store or retrieve data to or from any memory location LDS and STS commands is used.
- **LDS Rd, Ad** ; Load register Rd with the content of memory location Ad [$0000 - $FFFF] within data memory (GPR, SFR & SRAM)
- Examples: LDS R0, 0x0300, LDS R3, 0x0302
- Similarly **STS 0x0300, R0** does the reverse job

ATMega32 Programmer Model: I/O Registers (SFRs)

- The I/O memory is dedicated to specific functions such as status register, timers, serial communication, I/O ports, ADC and etc.
- Function of each I/O memory location is fixed by the CPU designer at the time of design. (because it is used for control of the microcontroller and peripherals)
- AVR I/O memory is made of 8 bit registers.
- All of the AVRs have at least 64 bytes of I/O memory location. (This section is called standard I/O memory)
- In other microcontrollers, the I/O registers are called SFRs (Special Function Registers)

ATMega32 Programmer Model: I/O Registers (SFRs)

Address		Name
I/O	Mem.	
$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
$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
$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

IN and OUT Instructions

- The IN instruction tells the CPU to load one byte from an I/O register to the GPR
- **IN Rd, Aio** ; load content of an I/O location (0-63H) to a GPR Rd
- *Note: Each of GPR has two addresses: One is data memory global address and other is I/O address relative to beginning of I/O space*
- **OUT Aio, Rd** ; does the opposite job of IN instruction

MOV Instruction

- MOV instruction to used to copy data from one to other registers in between R0 to R31
- **MOV Rd, Rr** : both d and r can be in between 0 to 31

ALU Instructions Using two GPRs

Instr.	operands	Meaning
ADD	Rd, Rr	Add Rd and Rc
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 Rd from Rr without Carry

ALU Instructions Using one GPR as Operand

Instr.	operands	Meaning
CLR	Rd	Clear Register Rd
INC	Rd	Increment Rd
DEC	Rd	Decrement Rd
COM	Rd	One’s Complement Rd
NEG	Rd	Two’s Complement Rd
ROL	Rd	Rotate Left Rd with Carry
ROR	Rd	Rotate Right Rd with Carry
LSL	Rd	Logical Shift Left Rd
LSR	Rd	Logical Shift Right rd
SWAP	Rd	Swap nibble in Rd

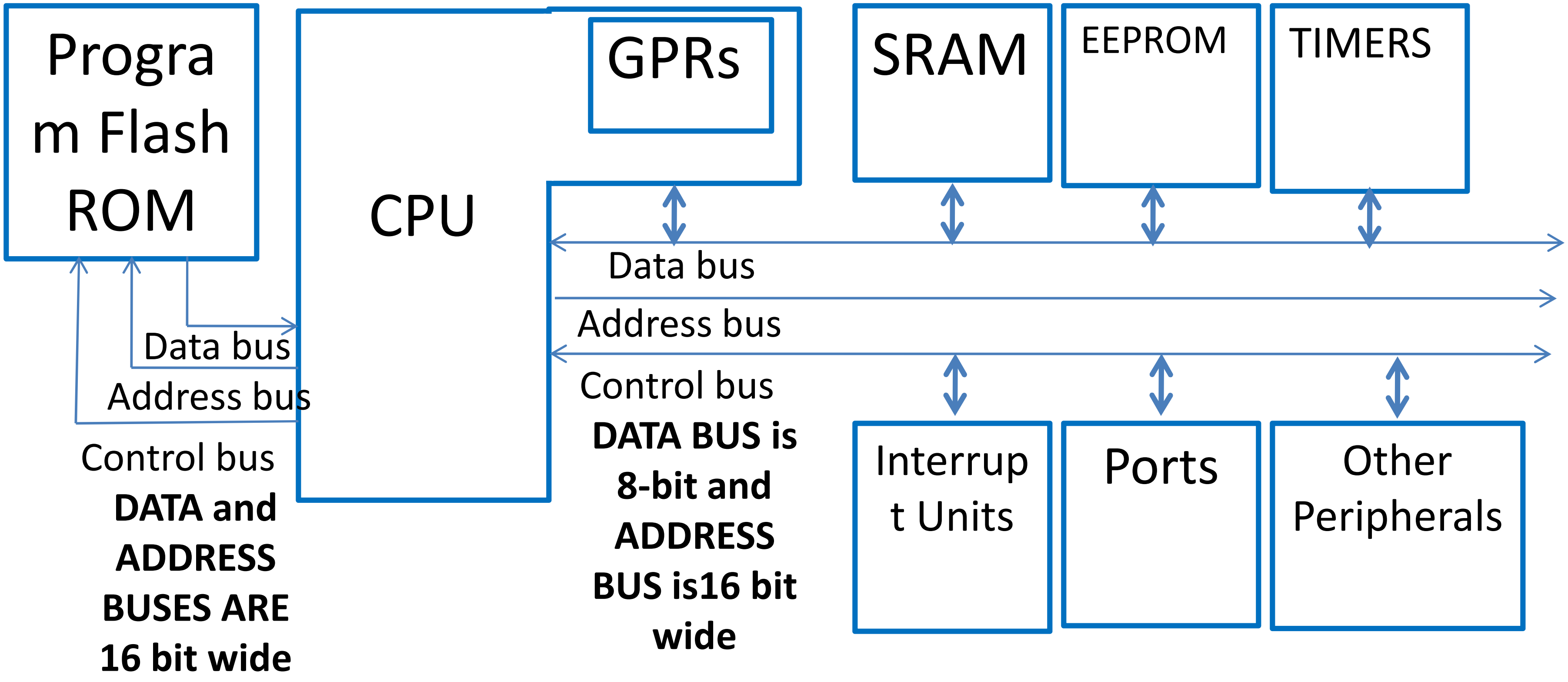
Program Counter in the AVR

- Program Counter, in short PC is the most important register in AVR
- PC is used by the CPU to point to the address of the next instruction to be executed
- As the CPU fetches the opcode from the Program ROM, PC is incremented automatically to point to the next instruction.
- The wider the program counter the more memory locations a CPU can access.

Program Counter in the AVR

- In AVR, each flash memory location is 2 bytes wide (it is called a “word”)
- Atmega32 has 32K flash program ROM, it is organized in 16K X 2 bytes, in other word a total of 16K locations.
- So, the program counter in Atmega32 is 14 bit wide because 2^{14}=16K.
- The PC in AVR family can be as high as 22 bit wide.

Communication between Program ROM and CPU



DATA and ADDRESS BUSES ARE 16 bit wide

DATA BUS is 8-bit and ADDRESS BUS is16 bit wide

Where the AVR wakes up

- When AVR is powered up or Signal is applied to RESET pin, the PC has the value of 0x0000
- For this reason in AVR system, the first opcode must be burned into memory location 0x0000 of Program ROM.

A Sample AVR program written in Assembly

```
.EQU  SUM=0x300     ; SRAM loc 0x300 for SUM
.ORG 0000           ; start at address 0
LDI  R16, 0x25      ; R16=0x25
LDI  R17, 0x34      ; R17=0x34
LDI  R18, 0x31      ; R18=0x31
ADD  R16, R17       ; add R17 to R16
ADD  R16, R18       ; add R18 to R16
LDI  R17, 0x0B      ; R17=0x0B
ADD  R16, R17       ; add R17 to R16
STS  SUM, R16       ; save the result in loc 0x300
HERE: JMP HERE      ; stay here forever
```

List file corresponding to the sample program

MEM. LOC	M/C CODE	INSTRUCTION
0000	E205	LDI R16, 0x25
0001	E314	LDI R17, 0x34
0002	E321	LDI R18, 0x31
0003	0F01	ADD R16, R17
0004	0F02	ADD R16, R18
0005	E01B	LDI R17, 0x0B
0006	0F01	ADD R16, R17
0007	9300 0300	STS SUM, R16
0009	940C 0009 HERE:	JMP HERE

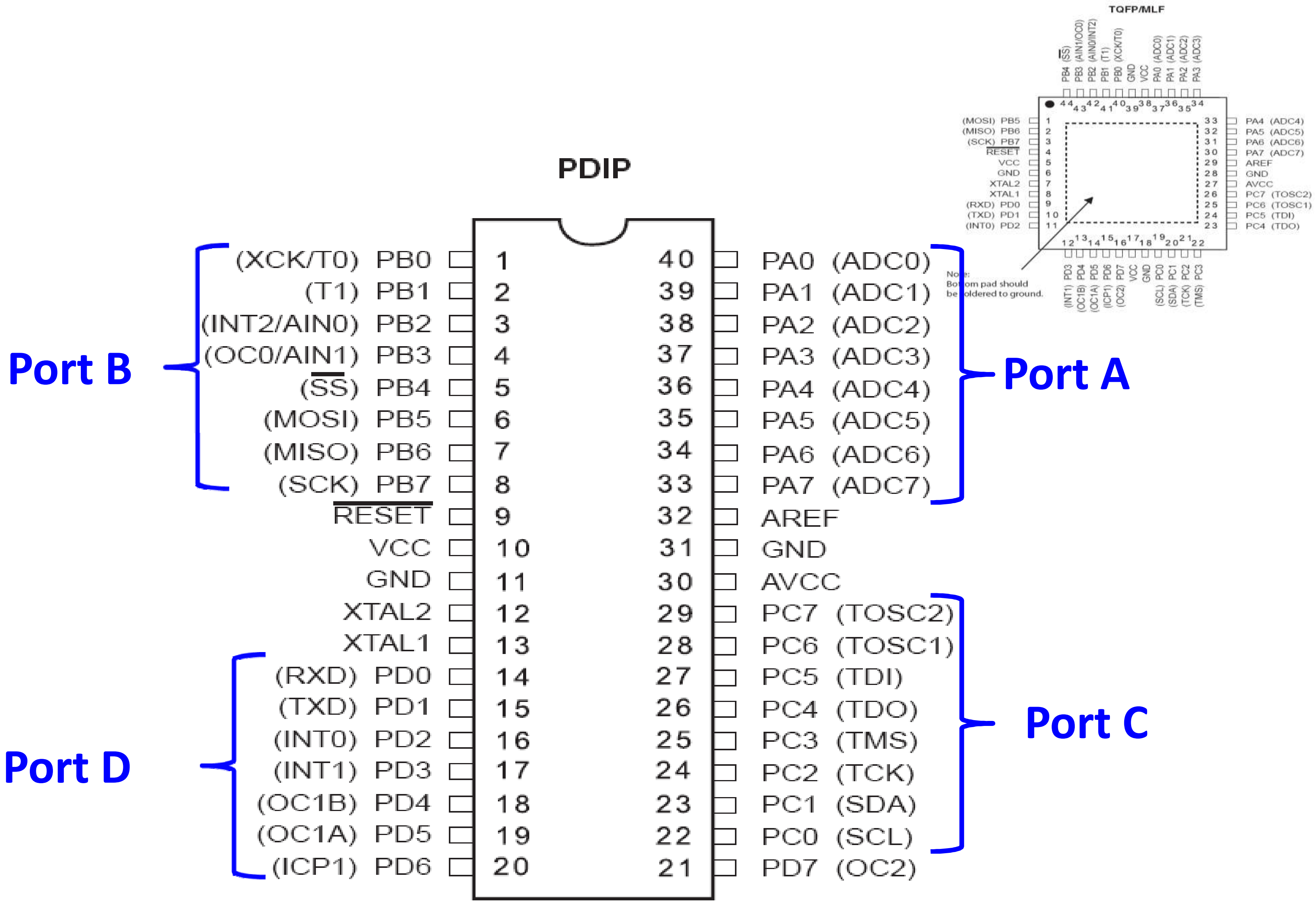
Placing code in Program ROM

- The list shows that address 0000 contains E205 which is the opcode for moving a value to R16 along with the operand is 0x25, called m/c code
- Similarly the machine code “E315” is located in ROM memory location 0001 and so on
- E205 and E315 both are 2 byte (1 word) wide instructions
- STS and JMP instructions are 4 byte wide
- That is why, the opcode for instruction “STS SUM, R16” is located at address 0007 and its **address** is at 0008
- And, the opcode for “JMP HERE” and its target address are located in locations 0009 and 000A

Executing a Program Instruction by Instruction

- When AVR is powered up, the PC (Program Counter) has 0000 and starts to fetch instruction located at 0000.
- In the example program first instruction is E205, CPU will place the data 0x25 to R16.
- The PC will then be incremented automatically to 0001 (location of the next instruction)
- In this way, when PC reaches at 0007 “STS SUM, R16” instruction will be executed
- And since it is 4 bytes (two word) instruction, PC will be incremented by 2, i.e., the new value of PC will be 0009

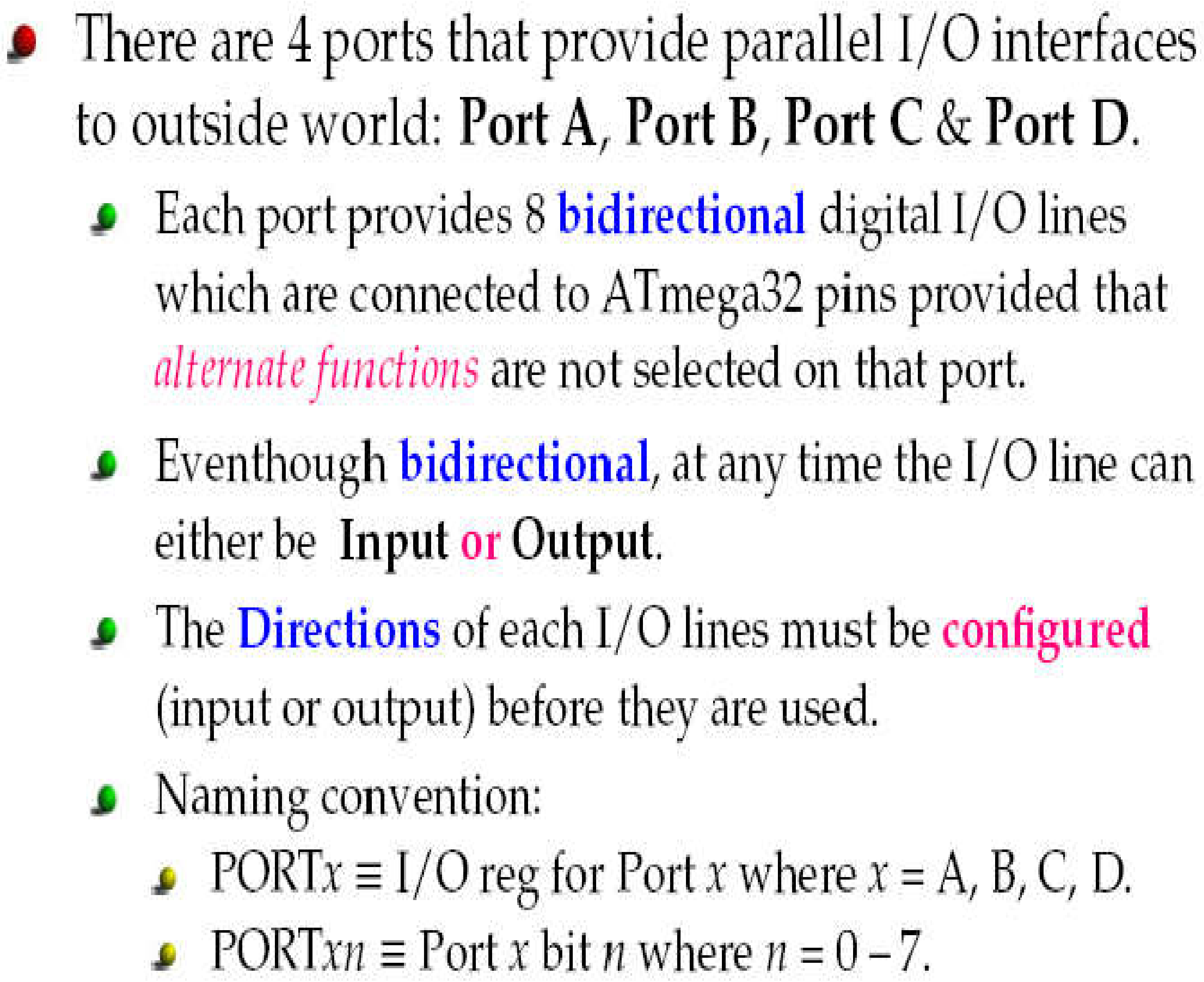
Ports of AVR ATmega32



Ports of AVR ATMega32

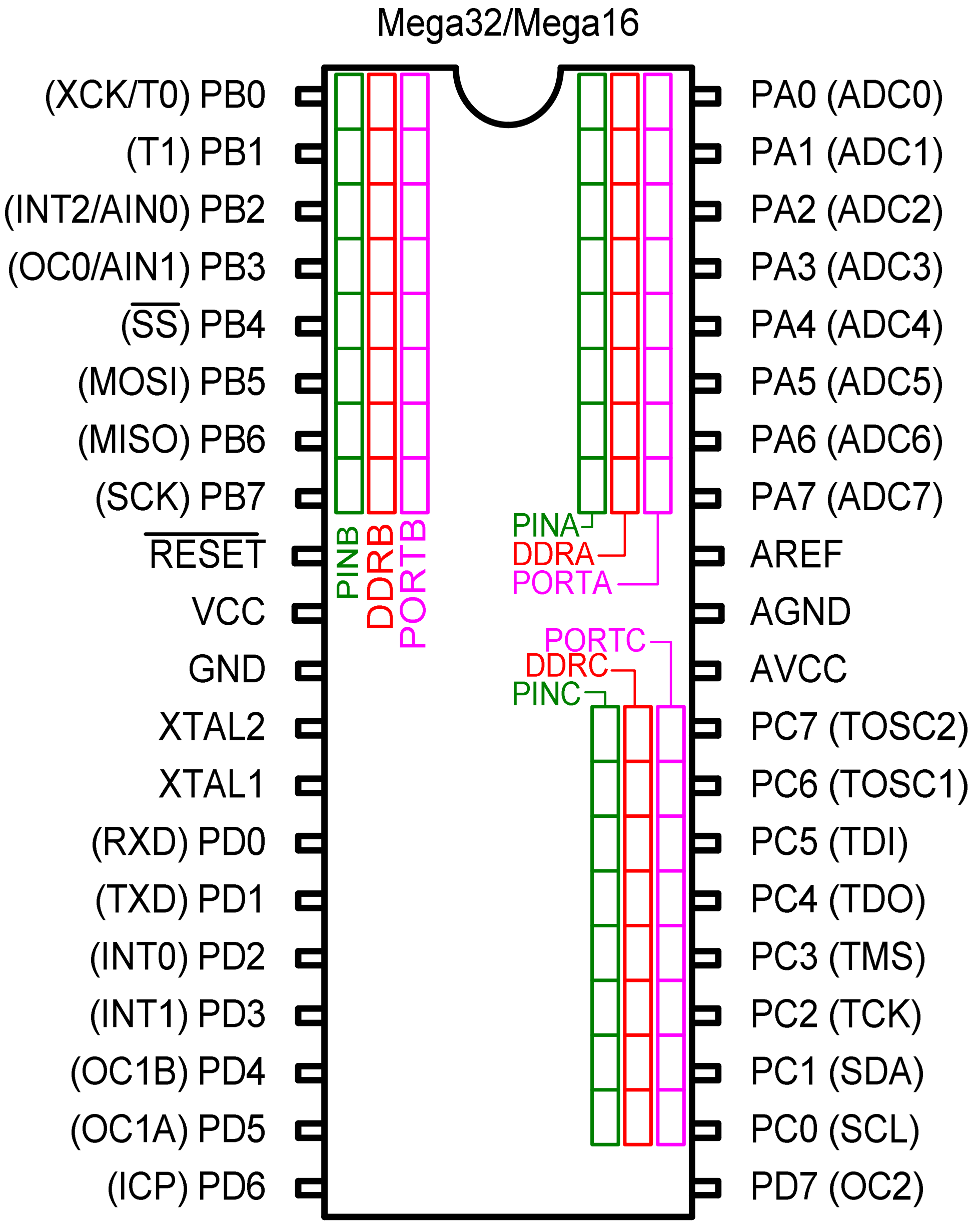
- ❑ Digital IO is the most fundamental mode of connecting a MCU to external world.
- ❑ The interface is done using what is called a PORT.
- ❑ **A port is the point where internal data from MCU chip comes out or external data goes in.**
- ❑ They are present in the form of PINs of the IC. Most of the PINs in ATmega32 are dedicated to this function and other pins are used for power supply, clock source etc .

ATMega32 Pin out & Descriptions



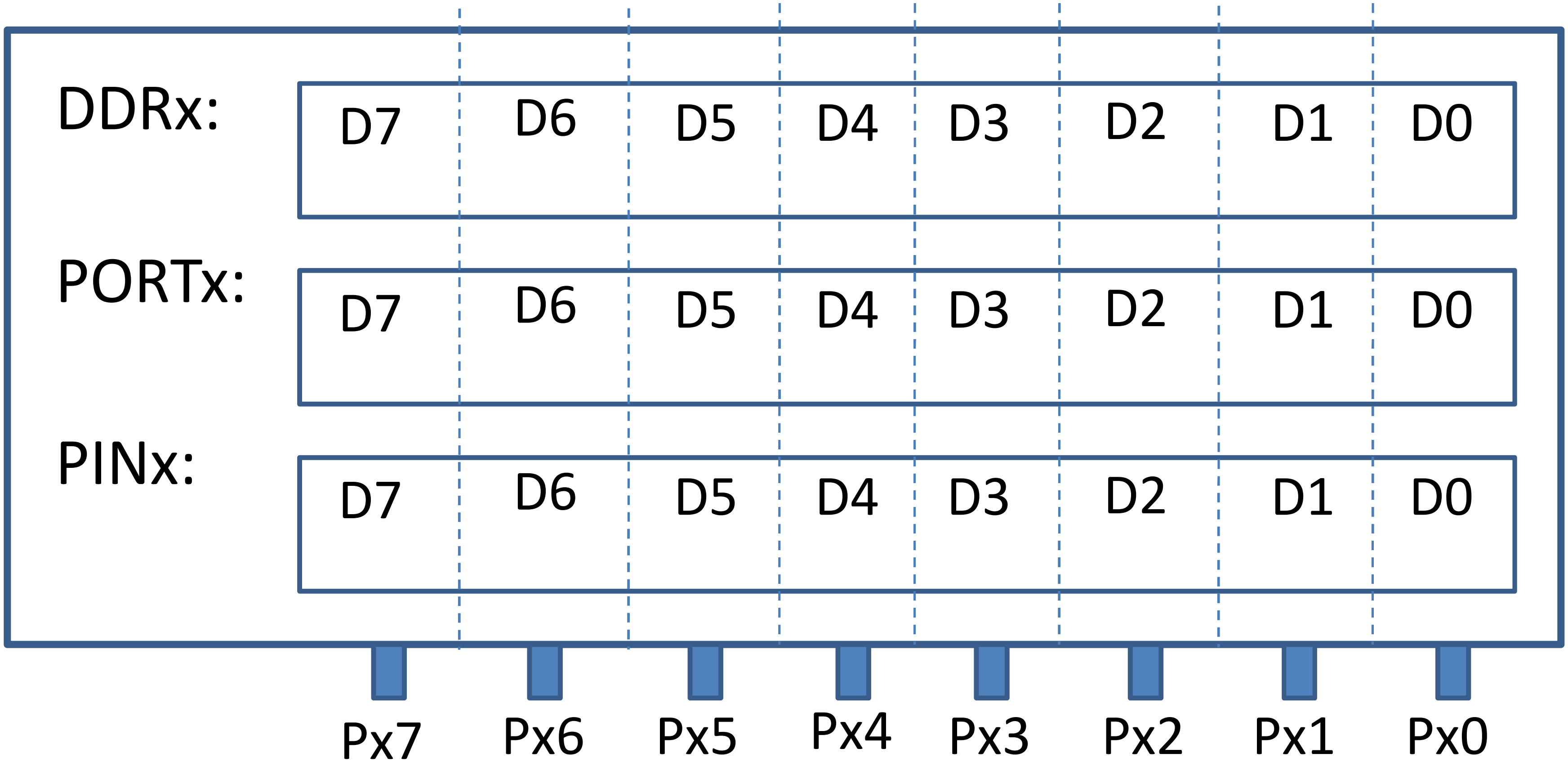
- There are 4 ports that provide parallel I/O interfaces to outside world: **Port A, Port B, Port C & Port D.**
  - Each port provides 8 **bidirectional** digital I/O lines which are connected to ATmega32 pins provided that *alternate functions* are not selected on that port.
  - Eventhough **bidirectional**, at any time the I/O line can either be **Input or Output.**
  - The **Directions** of each I/O lines must be **configured** (input or output) before they are used.
  - Naming convention:
    - PORT*x* ≡ I/O reg for Port *x* where *x* = A, B, C, D.
    - PORT*xn* ≡ Port *x* bit *n* where *n* = 0 – 7.

ATMega32 Pin out & Descriptions



I/O Registers related to Ports

- Each port has three I/O registers associated with it. They are designated as PORTx, DDRx and PINx
- For example, Port B has PORTB, DDRB and PINB registers.
- “DDR” stands for **D**ata **D**irection **R**egister, and PIN stands for **P**ort **IN**put.
- Also note that each of the I/O Register is 8-bit long and each of the Port has maximum of 8 pins.
- Each of I/O registers affects corresponding pin of the port.



DDRx:	D7	D6	D5	D4	D3	D2	D1	D0
PORTx:	D7	D6	D5	D4	D3	D2	D1	D0
PINx:	D7	D6	D5	D4	D3	D2	D1	D0
	Px7	Px6	Px5	Px4	Px3	Px2	Px1	Px0

Confusion Regarding the Names of the Registers

- There is no confusion regarding the DDR register as its purpose lies in its meaning - **D**ata **D**irection **R**egister.
- But the PORT register is sometimes confused with the physical Port of the AVR microcontroller.
- PORT register is a register in I/O memory, whereas the physical Port (consisting of 8 numbers of physical pins) is port through which data is made out or in.
- Similarly, PIN (**P**ort **IN**put) register is sometimes confused with physical Pin of the port.
- Again PIN register is a register in I/O memory whereas physical Pins are terminal of the microcontroller through which data is made out or in

DDRx Register Role in Outputting Data

- Each of the Ports A-D in ATmega32 can be used for Input or Output.
- DDRx I/O register is used solely for the purpose of making a given port input or output port.
- For example, to make a port an output, we write 1s to the DDRx register.
- In other words, to output data to all of the pins of Port B, we must first put 0b11111111 (0xFF) into DDRB register to make all the pins output.

How data is sent to the output port?

- Once you set the direction of a port as output, the data that you want to send to the output, you have write that data into PORTx I/O register.
- If DDRC=0xFF and then PORTC=0x3B, the data at the output will also be 0x3B.
- That is -

0	0	1	1	1	0	1	1
PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0

DDRx Register Role in Inputting Data

- To make a port an input port, we must first put 0s into the DDRx register for that port.
- Then we can read the data which is residing at the pins of Port x.
- Notice that upon reset, all ports have the value 0x00 in their DDRx registers.
- That is all ports upon reset remains as input ports.

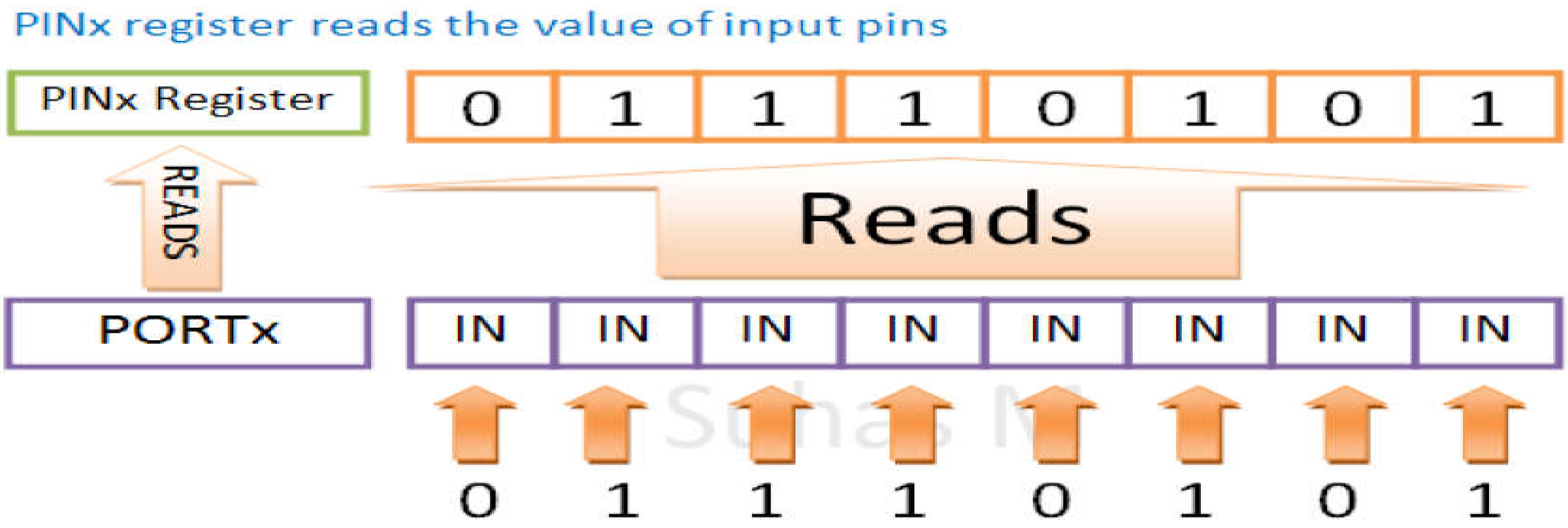
How data is read from the Input Port?

- Once you set the direction of a port as input, the data that you want to read from the input, you have place that data to port pins.
- If DDRB=0x00 and the data presented at the port pins of Port B port is 0x75, the data will go to PINB register of I/O memory. You have to read that reg.
- The content of PINx will be -

PINB Register	0	1	1	1	0	1	0	1
	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0

Reading Data from a Port

The PINx register gets the reading from the input pins of the MCU

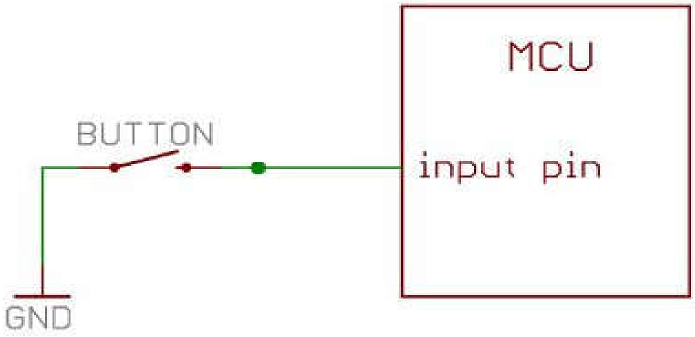


PINx register reads the value of input pins

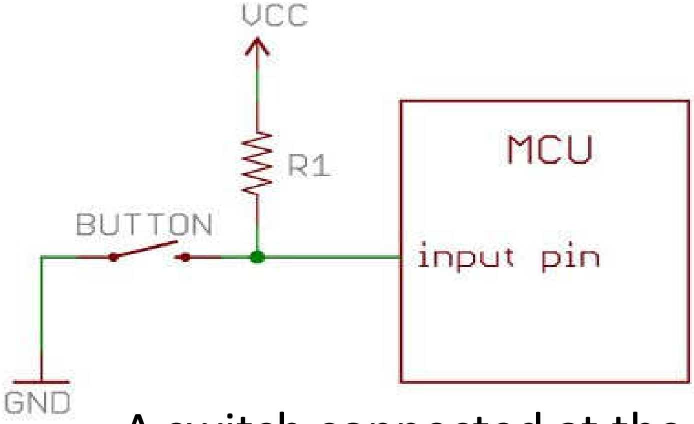
PINx Register	0	1	1	1	0	1	0	1
PORTx	IN	IN	IN	IN	IN	IN	IN	IN
	0	1	1	1	0	1	0	1

Role of Pull up resistors at the port pins

- Let us say you have configured a port pin as input pin.
- If there is nothing connected with the pin and your program reads the state of the pin, will it be high of low? It will be difficult to tell.
- This phenomena is referred to as floating state.
- In order to prevent this unknown state, pull up resistor is used.
- Pull ups are often used with button and switches.



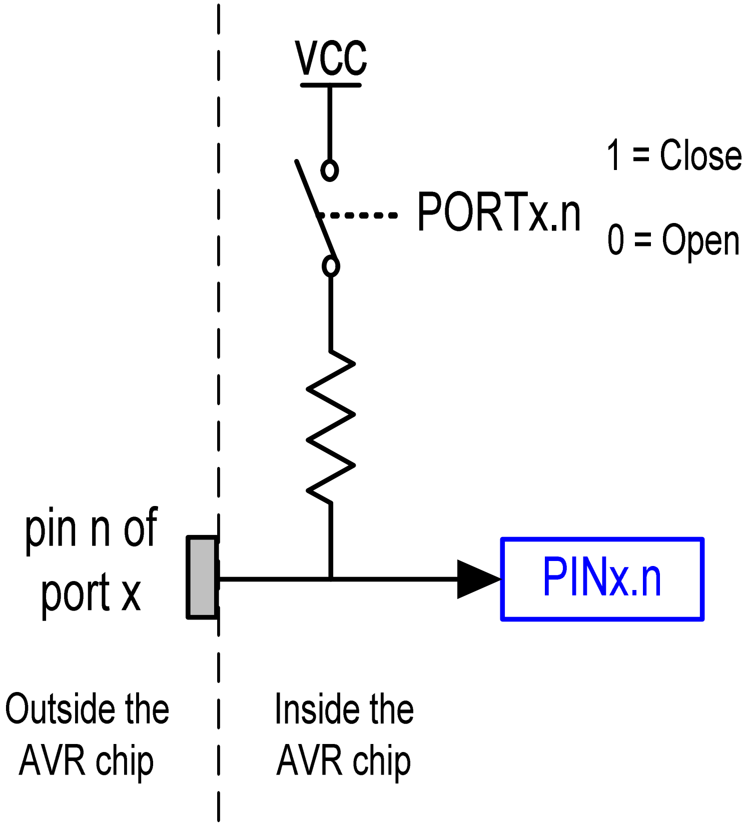
A switch connected at the input pin without pull up resistor



A switch connected at the input pin with pull up resistor

Internal pull up resistor in AVR and its control

- In AVR ATmega32 microcontroller, each of the port pins have a pull up resistor having the facility of connecting or disconnecting it with the pin.
- The provision of connection or disconnection is controlled by PORTx register. [see the figure]
- If we put 1s into bits of PORTx register, the pull up resistors are activated.

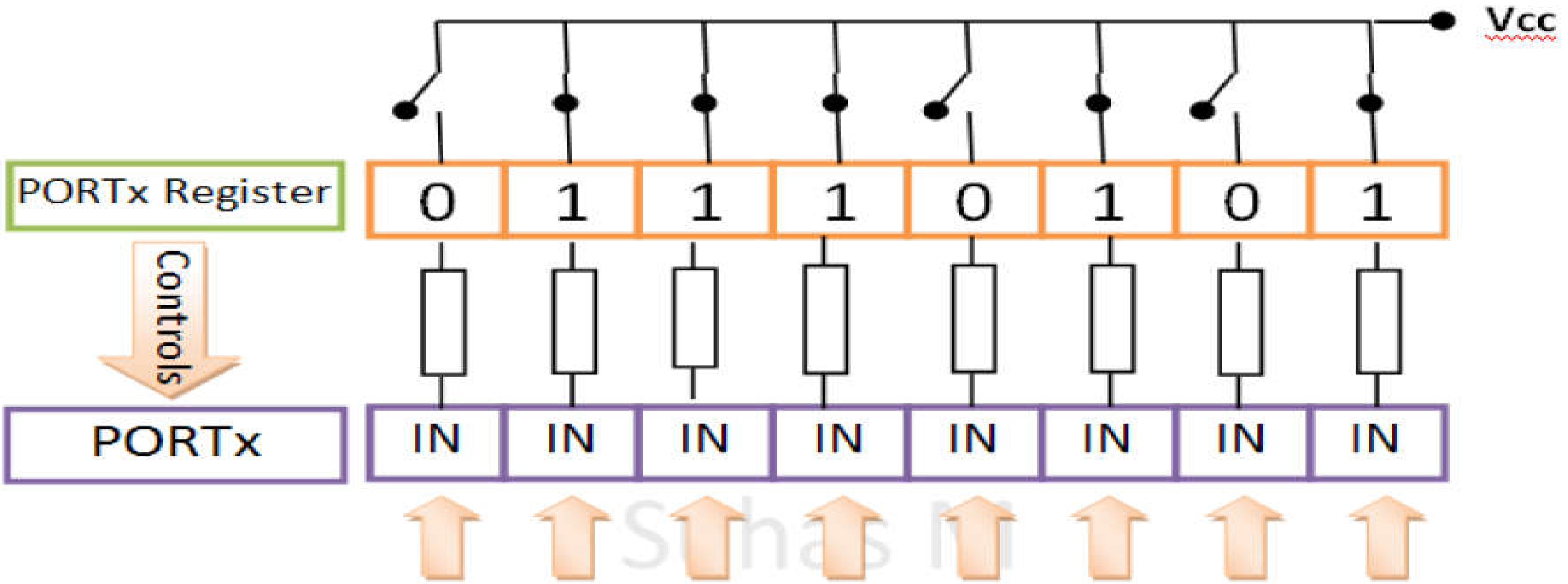


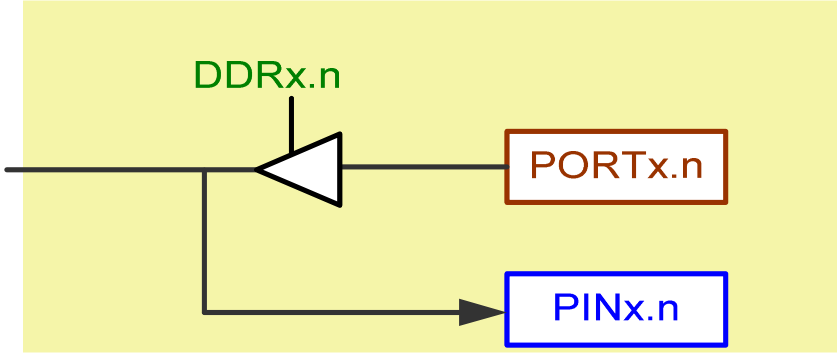
Remember that during output of data PORTx is the register in which data has to be written to make it out.

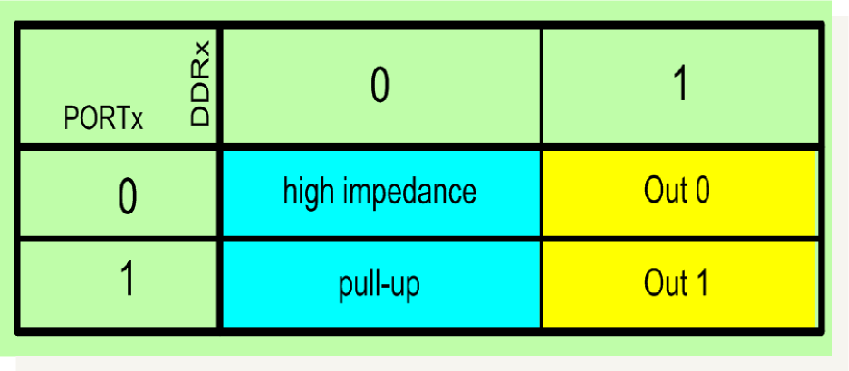
Activation of Pull Up Resistors

To activate / Deactivate pull up resistors-Data Register PORTx

If the pin is configured as input, PORTx manages the internal pull-up



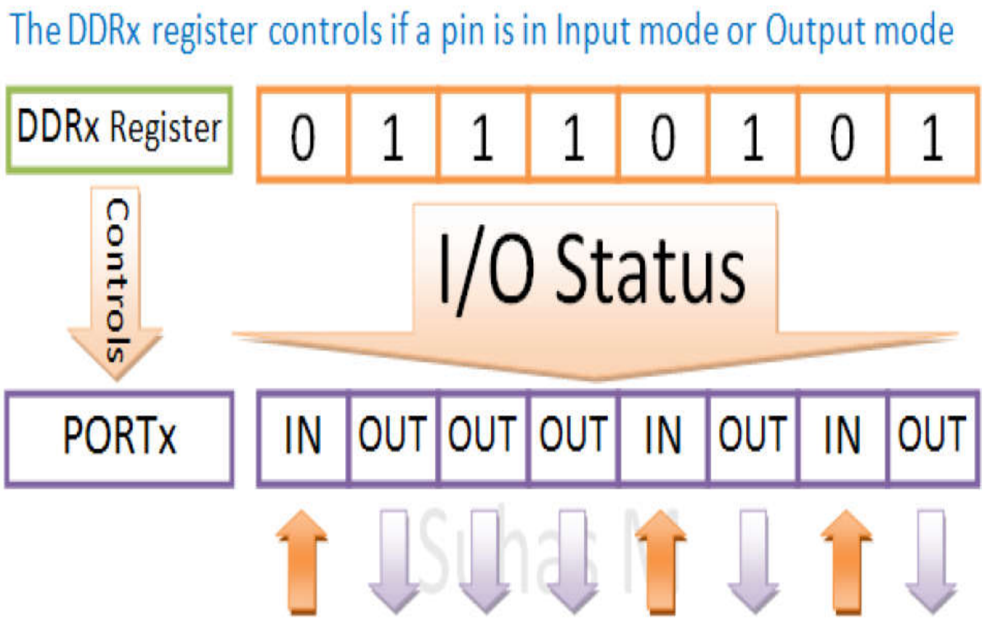




PORTx \ DDRx	0	1
0	high impedance	Out 0
1	pull-up	Out 1

Input and Output pins in a Single Port

Defining a pin as either Input or Output – The DDRx Registers


The DDRx register controls if a pin is in Input mode or Output mode
DDRx Register
0 1 1 1 0 1 0 1
Controls
I/O Status
PORTx
IN OUT OUT OUT IN OUT IN OUT

```
LDI   R20,0xFF   ;R20 = 0b01110101
OUT   PORTx,R20  ;PORTA = R20
OUT   DDRx,R20   ;DDRA = R20
```

DDRx = 0b01110101; /* Configuring I/O pins

- Each bit in the DDRx determines the direction of the corresponding bit in PORTx.
- Writing a ‘1’ to a bit in DDRx makes the bit of PORTx **Output**. Writing a ’0’ makes the bit **Input**.
- After *Reset*, all the bits in DDRx is cleared (‘0’).

Summary Action During Output

- **Step1:** If we want to send (out) data through a pin of a port, the corresponding bit of DDRx Register has to be set to ‘1’.
- **Step 2:** Then if we want to send ‘1’ write ‘1’ to the corresponding bit of PORTx Register and ‘0’ if we want to send ‘0’.
- **Note:** We can not send data (either ‘1’ or ‘0’) if we do not complete step 1. At START UP or UPON RESET DDRx register contains 0x00, that is all pins remain at input state.

Summary Action During Input

- **Step1:** If we want to read (in) data through a pin of a port, the corresponding bit of DDRx Register has to be set to ‘0’.
- **Step 2:** Whatever data is in that pin of the port the data will appear in the corresponding bit of PINx register, so we have read that bit of PINx.
- **Step 3:** Each pin has a provision of connecting a pull up resistor internally. If we want to connect that resistor, we have to write ‘1’ to the corresponding bit of PORTx register.
- **Note:** At START UP or UPON RESET, DDRx register contains 0x00. That is the port remains at Input state. But if it is used as OUTPUT at one stage of program, afterwards, we can not read data from a pin if we do not complete step 1.

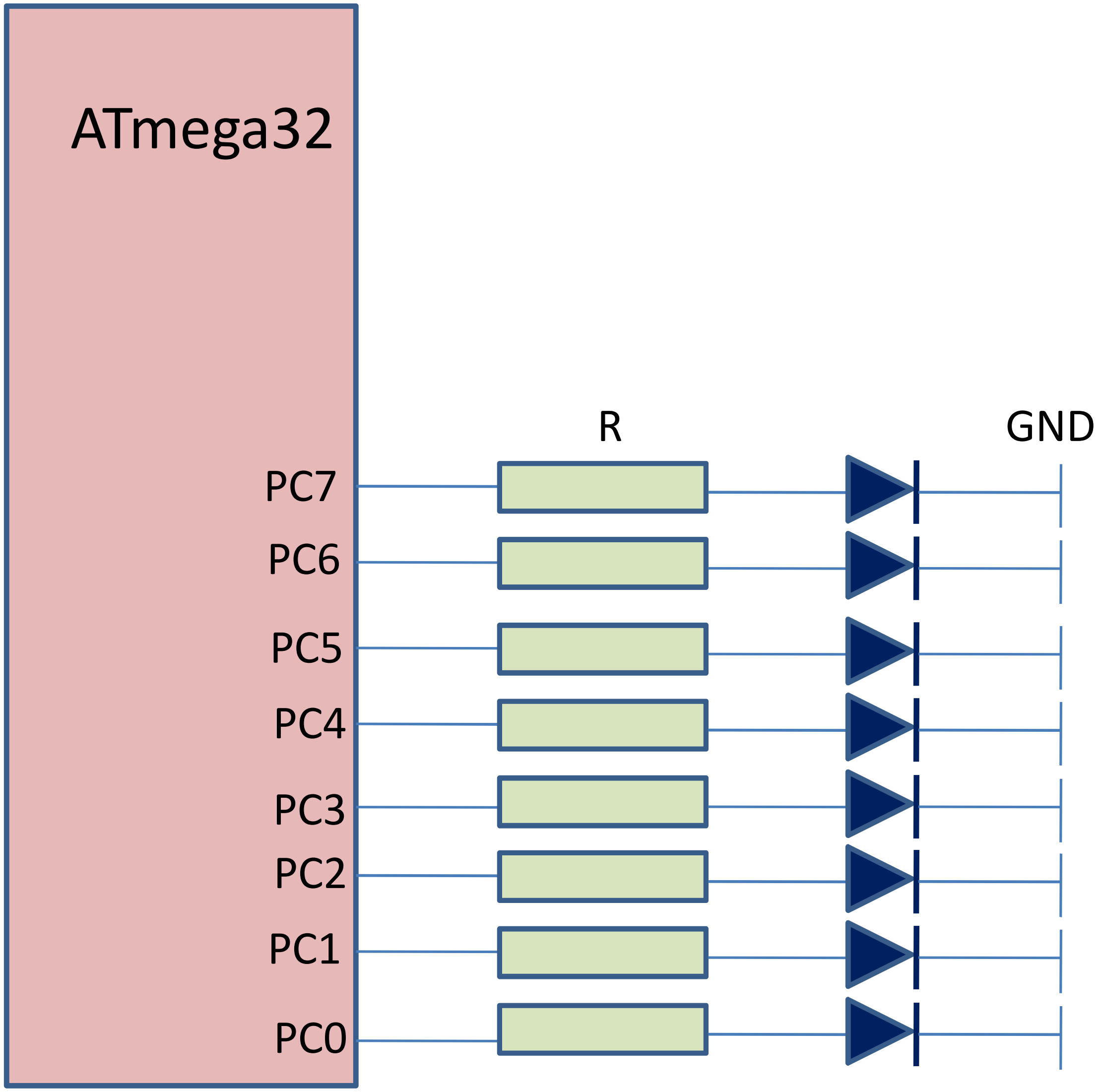
A Practical example of Outputting Data through a Port

- Let us assume that 8 LEDs are connected to 8-pins of PORTC of an Atmega32 chip
- We want to glow all of the eight LEDs.
- And they will be switched off after1 sec and it will be repeated continuously.
- Write a C code for the above mentioned output operations.

What is an LED?

- Light Emitting Diode (LED) is a special diode that emits light when electric voltage is applied to it. It is a common electronic equipment used in many devices for indication purpose.
- There are two leads of an LED that are used to supply input voltage. The longer lead is positive and known as ‘Post’, and the smaller is negative known as ‘Anvil’

Connection of LEDs with a Port



All LEDs glow and off at an interval of 1 sec. And the process is repeated.

0xFF	1	1	1	1	1	1	1	1
0x00	0	0	0	0	0	0	0	0

```
PORTC=0xFF
while (1)
     {
       PORTC=~PORTC;
       delay_ms(1000);
     }
}
```

Alternate LEDs will glow and toggle at every 1 sec

0xAA	1	0	1	0	1	0	1	0
0x55	0	1	0	1	0	1	0	1

```
PORTC=0xAA
while (1)
      {
        PORTC=~PORTC;
        delay_ms(1000);
      }
}
```

LED at MSB Position is ON, all others are off. After 1 sec, the next one is ON. Then the next one and so on. When it reaches at the LSB one, it starts from MSB again

PORTC	1	0	0	0	0	0	0	0
PORTC>>1	0	1	0	0	0	0	0	0

```
PORTC=0x80
while (1)
      {
        PORTC=PORTC>>1;
         delay_ms(1000);
         if (PORTC==0X01)
             PORTC=0X80;
      }
```

LED at MSB Position is ON, all others are off. After 1 sec, the next one is ON. Then the next one and so on. When it reaches at the LSB one, it REVERSES.

PORTC	0	0	0	0	0	0	0	1
PORTC<<1	0	0	0	0	0	0	1	0

```
PORTC=0x80
movement=0;
while (1)
     {
       if (movement==0) {
          PORTC=PORTC>>1;
           if (PORTC==0X01)
                    movement=1;}
```

```
          else
          {
           PORTC=PORTC<<1;
            if (PORTC==0x80)
                    movement=0;
            }
           delay_ms(1000);
      }
```

An Example of Inputting data

- Use Pin 0-7 of PORT D of an ATmega32 chip for inputting a data.
- Some of the pins are connected to VCC and some are made grounded.
- Read the data.
- Send the value to PORT C. [The port C is connected with LEDs]
- Write a C code for the above mentioned tasks.

Another Example

If Pin 0 of Port D is found to be zero send 0xAA to Port C, otherwise send 0x55 to Port C

```
while (1)
     {
     // Place your code here
     if (PIND.0==0)
     PORTC=0xAA;
     if (PIND.0==1)
     PORTC=0x55;
     }
}
```

Thanks