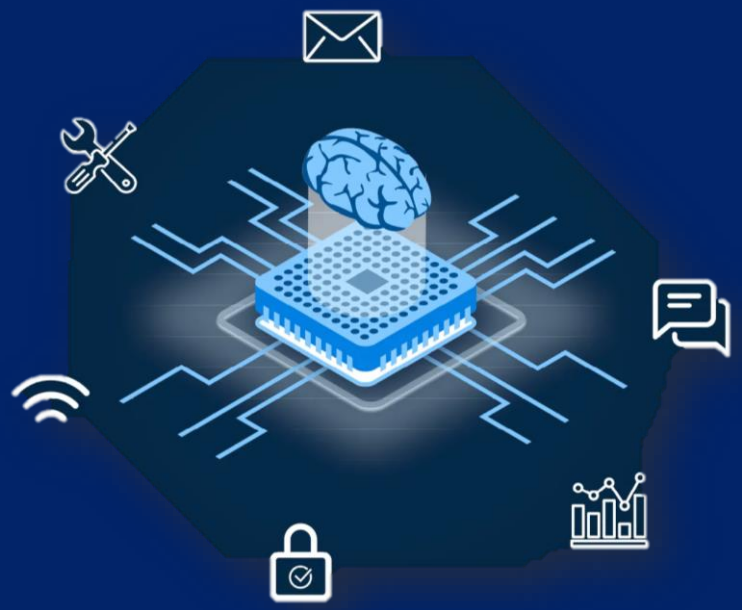
Today session Objective

D-BUGERZ

Start Coding with GPIO and Interface with some hardware components



Our First Peripherals (GPIO & Interrupt)

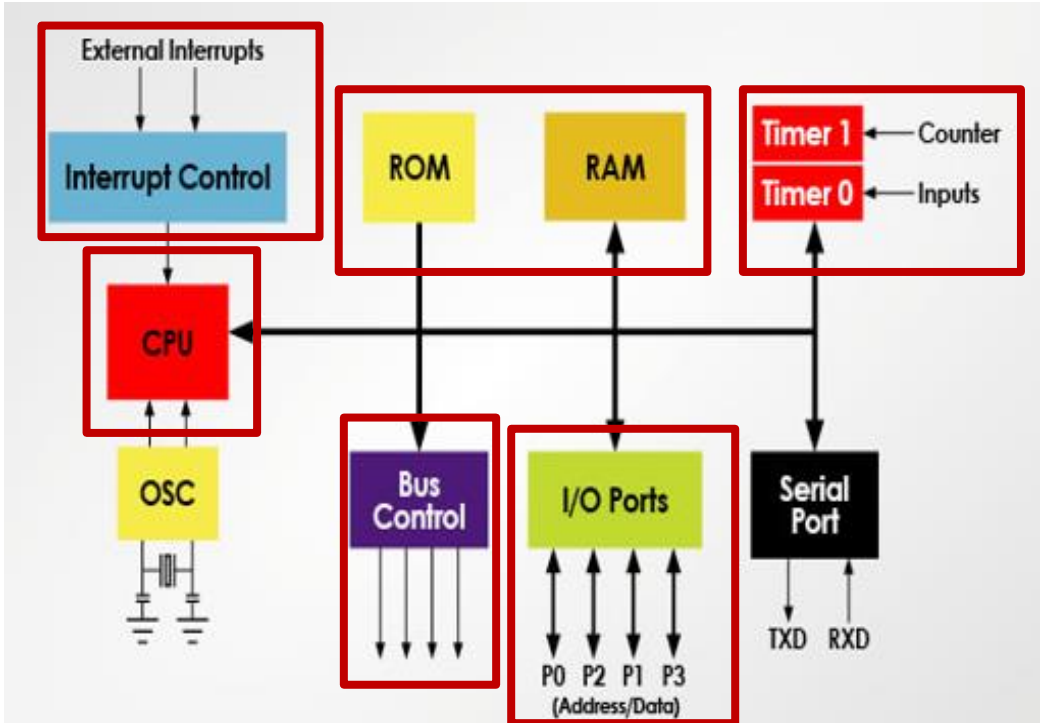
D-BUGERZ

- Recall Last session
- Recall Important C programming keys
- Introduction to GPIO
- Start Coding with GPIO
- Introduction to Interrupt

Recall Last session

D-BUGERZ

1. Central Processing Unit (CPU)
2. Memory Units
3. Input and Output Ports (GPIO or DIO)
4. Timers
5. Watch-Dog Timer
6. Buses
7. Interrupt Unit



Recall Last session

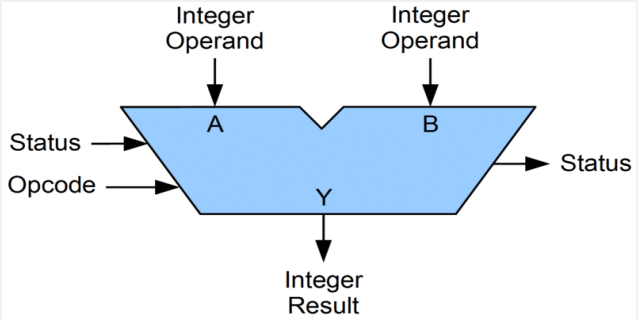
D-BUGERZ

Component of CPU

1.
2.
3.
4.
5.

CPU

ALU



Control unit

Instruction	Op - Code
ADD	0000 0000
SUB	0000 0001

Registers

	7 0	Addr.
	R0	$00
	R1	$01
	R2	$02
	...	
	R13	$0D
General	R14	$0E
Purpose	R15	$0F
Working	R16	$10
Registers	R17	$11
	...	
	R26	$1A
	R27	$1B
	R28	$1C
	R29	$1D
	R30	$1E
	R31	$1F

Recall Last session

D-BUGERZ

1.3 Registers

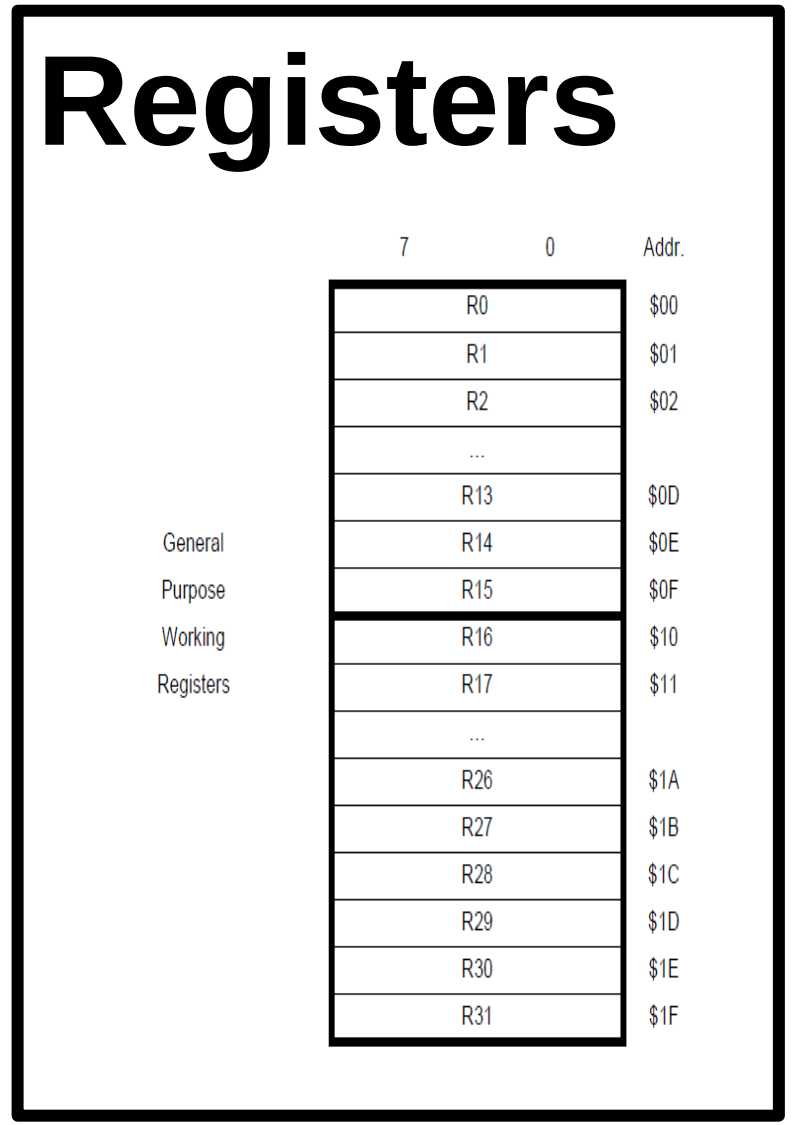
1. A processor register (CPU register) is one of a small set of data holding places that are part of the computer processor.

2. A register may hold an instruction, a storage address, or any kind of data (such as a bit sequence or individual characters). Some instructions specify registers as part of the instruction. For example, an instruction may specify that the contents of two defined registers be added together and then placed in a specified register.

3.

4.

5.



Recall Last session

D-BUGERZ

Port A Data Register – PORTA

Bit	7	6	5	4	3	2	1	0	
	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0	PORTA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

1.

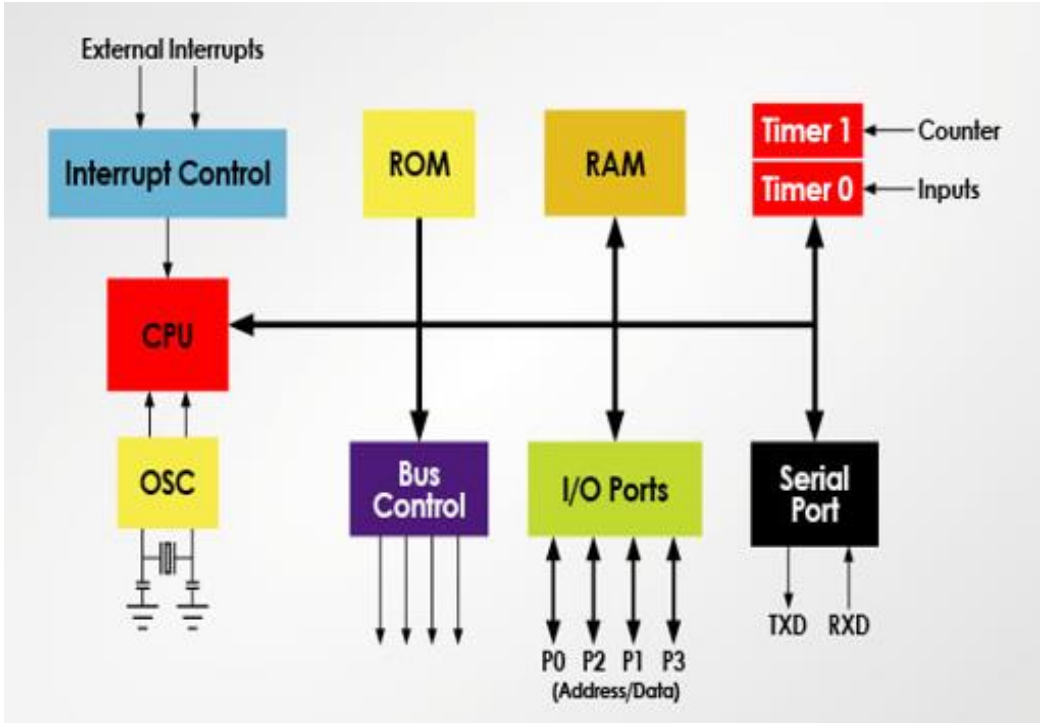
2.

3.

4.

5.

- All peripherals have registers and you as programmer your role to write and read from this registers to know the status of your controller
- That what we will start with in this session



Recall Last session

D-BUGERZ

1.

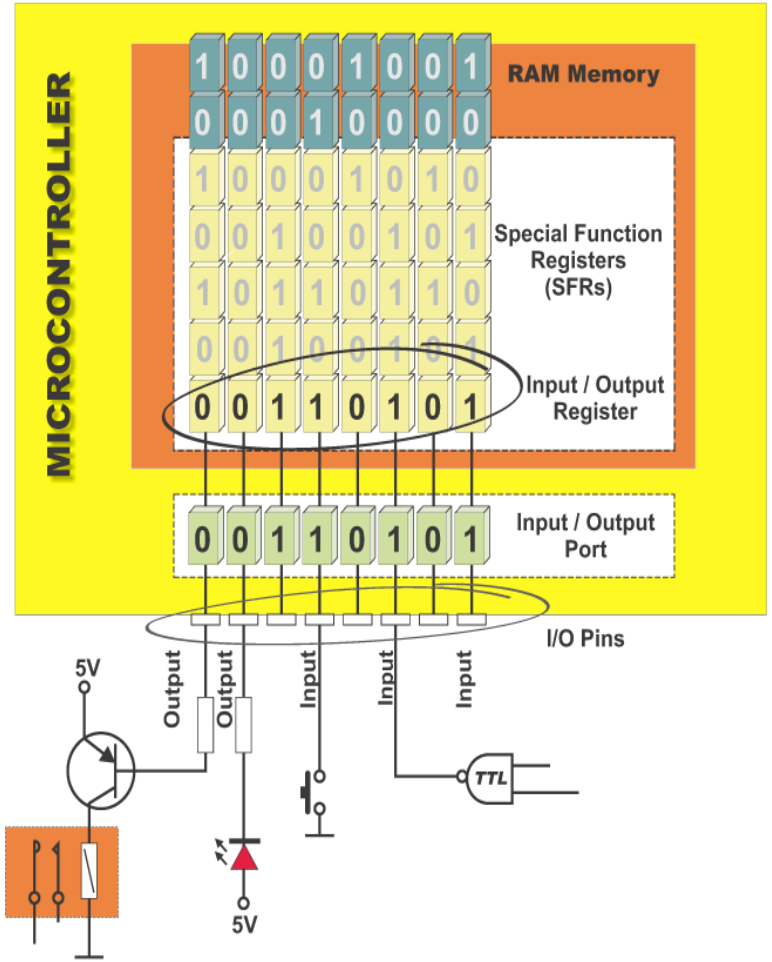
2.

3.

4.

5.

- The microcontroller has to be connected to additional external electronics and peripherals.
- For that reason, each microcontroller has one or more registers (called “port” in this case) to which it’s connected.
- These ports are bidirectional could be output or input.
- Each I/O port is under control of another SFR, which means that each bit of that register determines state of the corresponding microcontroller pin.
- **<u>Suppose you want your device to turn on and off three signal LEDs you must to write 1 in register corresponding to this pin</u>**



Recall Last session

D-BUGERZ

- Clock signal is a particular type of signal that oscillates between a high and a low state, Circuits use the clock signal for synchronization may become active at either the rising edge, falling edge, or, in the case of double data rate, both in the rising and in the falling edges of the clock cycle.

1.

- Microcontroller clock generator source could be internal or external.

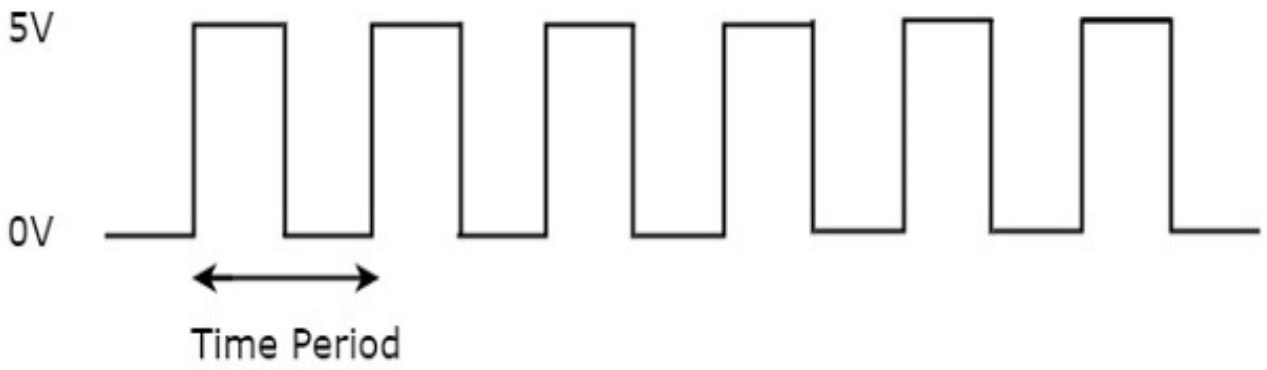
2.

- **Any Microcontroller have it's Clock and you can modify this number to your needs as programmer and this clock is the main parameter of controller speed you can modify by 1 MHZ , 8 MHZ , 12 MHZ , 16MHZ**

3.

4.

5.



Recall Last session

D-BUGERZ

1.

2.

3.

4.

5.

D-BUGERZ

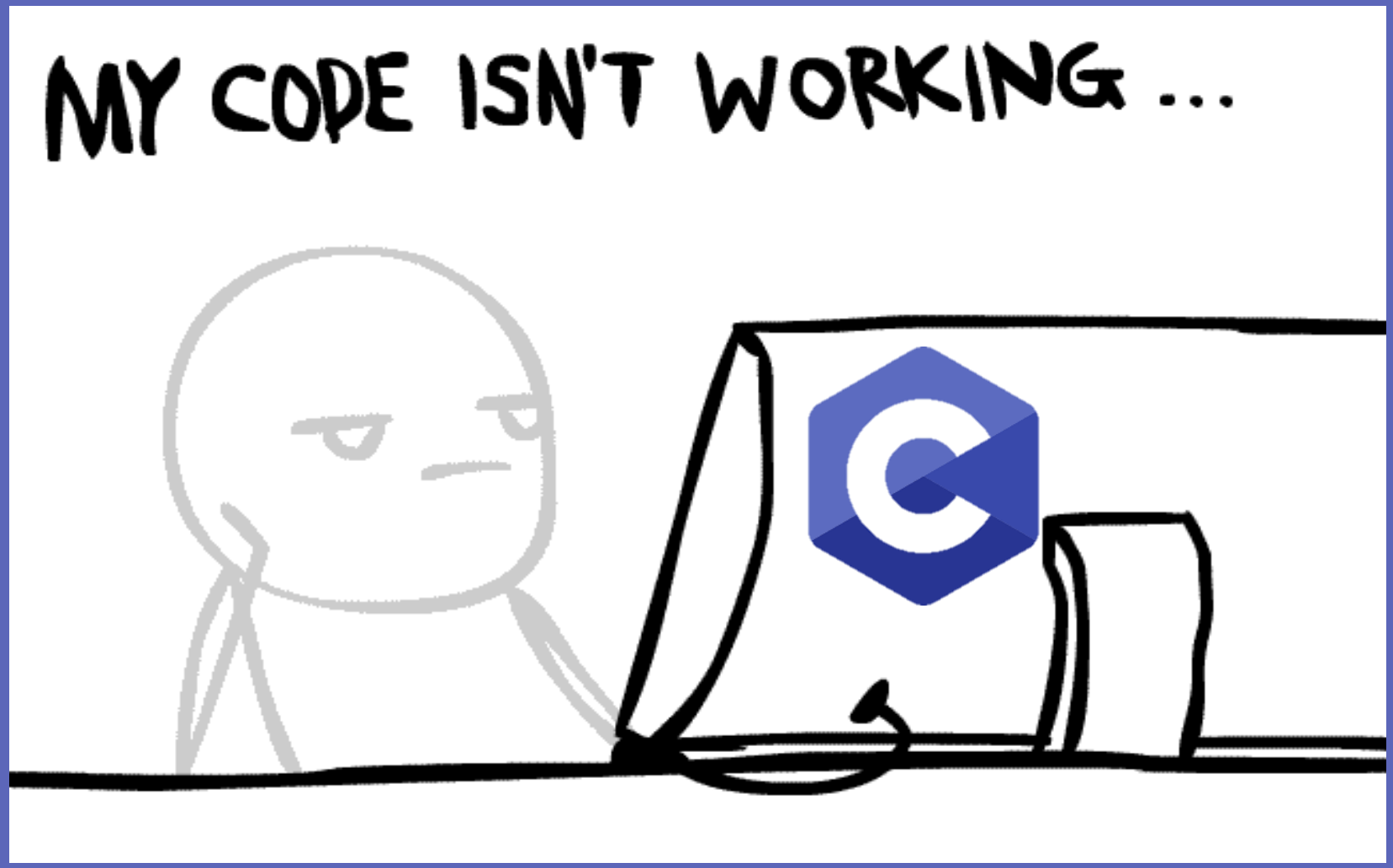
Recall Last session

Recall Important C programming keys

Introduction to GPIO

Start Coding with GPIO

Introduction to Interrupt



1- #Includes

D-BUGERZ

1.
2.
3.
4.
5.

Start here X Ex2.c X

```
1
2
3   #include <stdio.h>
4
5   int main()
6   {
7       char name[14]="Mahmoud Kafafy";
8       float grade=97.5;
9       printf("my name is :%s\nmy grade : %f \n",name,grade);
10      return 0;
11  }
12
```

- In C programming, #include <stdio.h> is a preprocessor directive that tells the compiler to include the contents of the standard input-output library (stdio.h) in the program. This library provides functions for input and output operations, such as reading from the keyboard and writing to the screen.

- To write Printf, Scanf

1- #Includes

D-BUGERZ

1.
2.
3.
4.
5.

main.c

```
1
2
3  #include <stdio.h>
4
5  #define PI 3.14
6
7  int main()
8  {
9      float radious,area,circumference;
10     scanf("%f",&radious);
11
12     /* Calculate the circle area */
13     area=PI*radious*radious;
14
15     /* Calculate the circle circumference */
16     circumference=2*PI*radious;
17
18     printf("Circle Area=%f\nCircle Circumference=
19     return 0;
20 }
21
```

- Constant to use in another part in code

1- #Includes

D-BUGERZ

1.
2.
3.
4.
5.

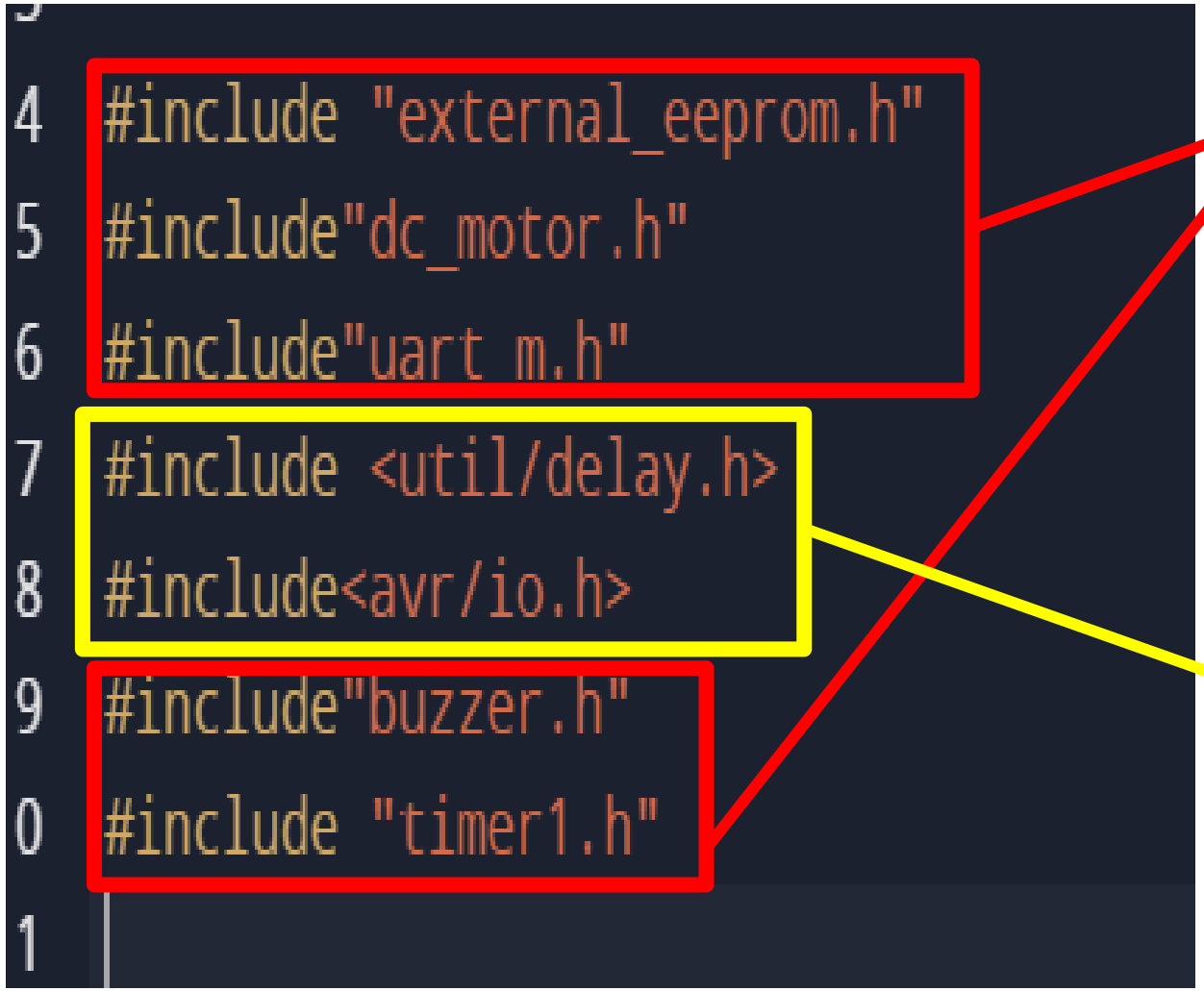
- There are 3 ways to define constants in C

```
 */

#define binary                  01000001
#define hexadecimal             0x3f66
#define real_number             25


```

1- #Includes

D-BUGERZ

1.
2.
3.
4.
5.



- Include from another files you made

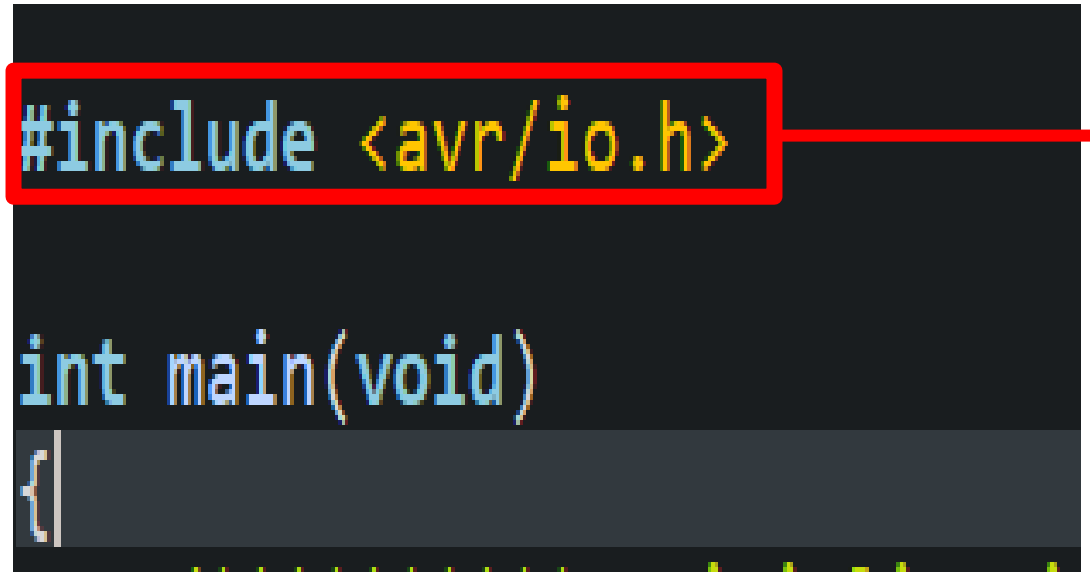
- Include from standard libraries

1- #Includes

D-BUGERZ

We will see at first Exercise

1.
2.
3.
4.
5.



- The avr/io.h header file is used in AVR microcontroller programming with the AVR-GCC compiler. This header file provides access to the I/O registers of the AVR microcontroller, allowing you to manipulate hardware peripherals such as GPIO pins, timers, and serial communication interfaces.

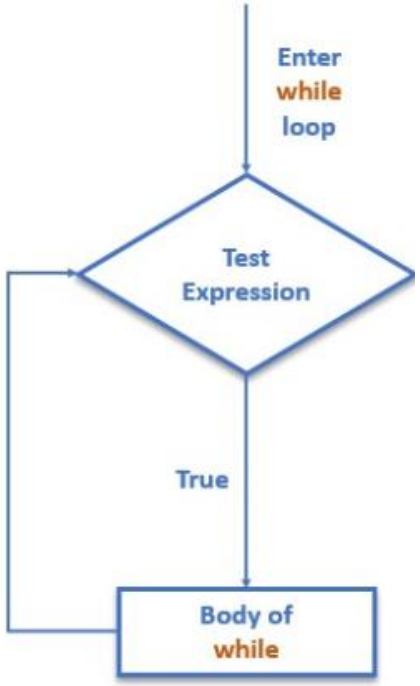
2- While(1)

D-BUGERZ

1.
2.
3.
4.
5.

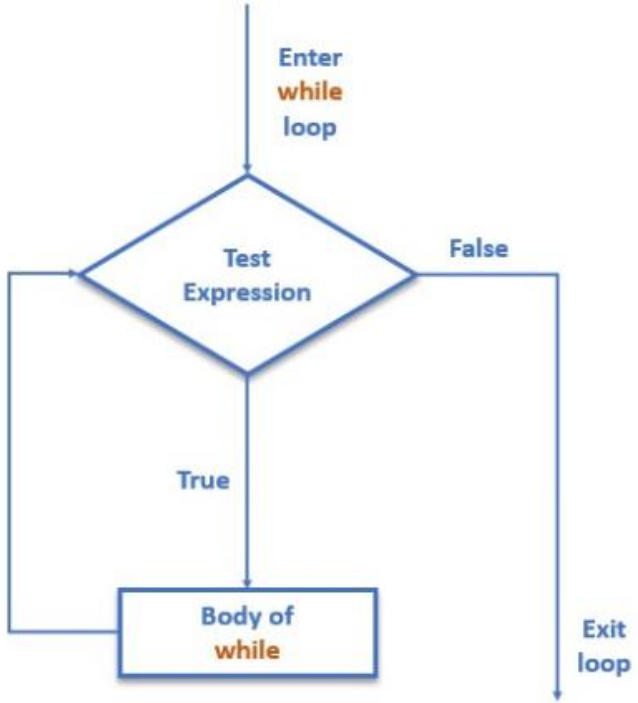
While(1)

- Infinite loop



While(x==1)

- If x is variable and we need to check on it



2- While(1)

D-BUGERZ

1.
2.
3.
4.
5.

```c
#include  <stdio.h>

int main() {
    int num = 5 , i, sum = 0;

    i = 1;

    while (i) {
        sum = sum + i;
        i++;
    }

    printf("Sum: %d", sum);

    return 0;
}
```

```
Output
/tmp/IJi6AfAfNz.o
```

Stuck in the loop (No Output)

```c

#include  <stdio.h>

int main() {
    int num = 5 , i, sum = 0;

    i = 1;

    while (i  <= num) {
        sum = sum + i;
        i++;
    }

    printf("Sum: %d", sum);

    return 0;
}
```

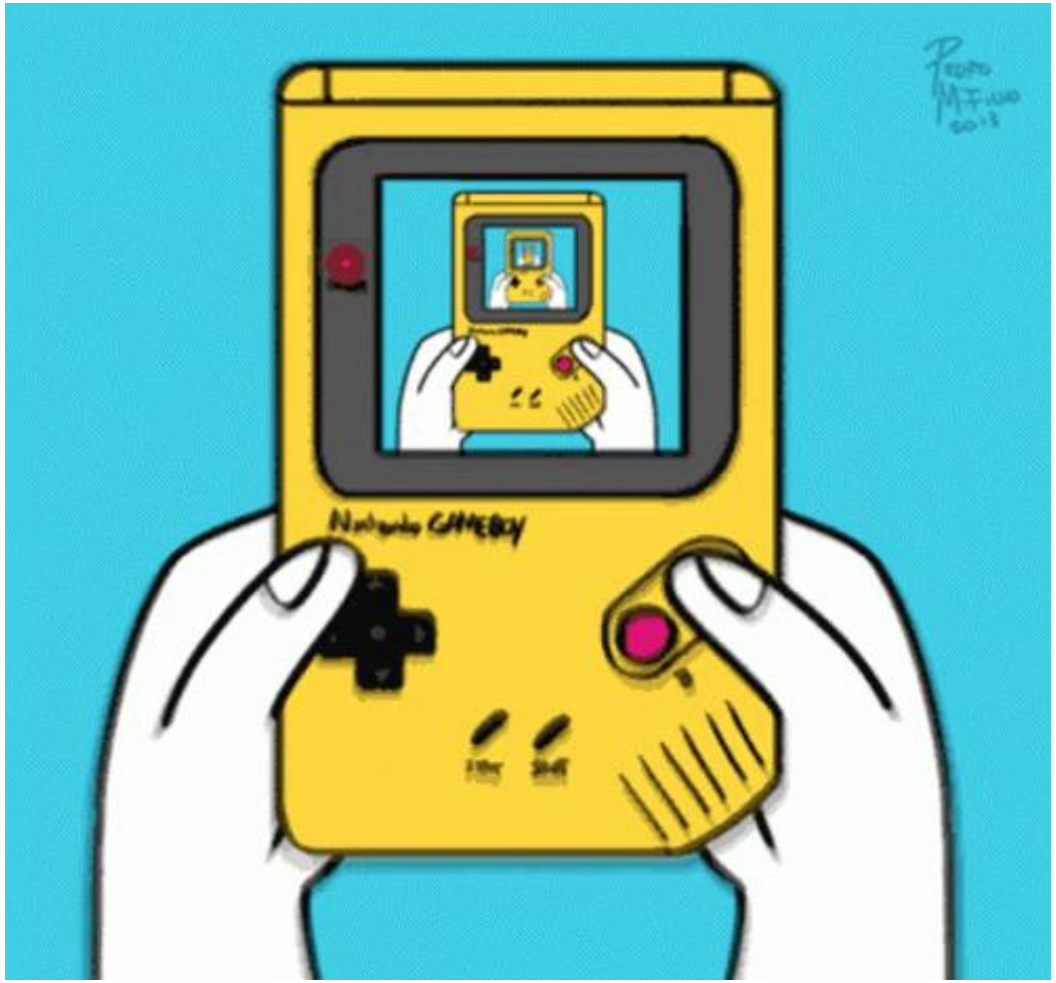
Output = 15

2- While(1)

D-BUGERZ

1.
2.
3.
4.
5.

- We use while(1) in embedded code to make infinite code which will always executable if the controller powered on
- To make code always work



D-BUGERZ

1.

2.

3.

4.

5.

3- Shift Operation

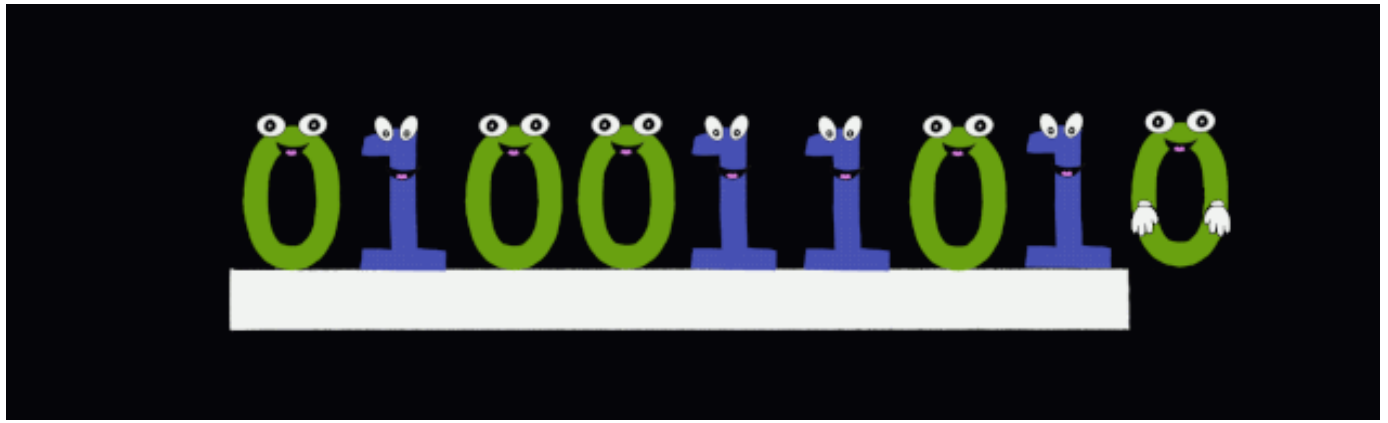
D-BUGERZ

1.

Shift Left

1 = 0000 0001

1 << 2 = 0000 0100



2.

1 << 5 = 0010 0000

3.

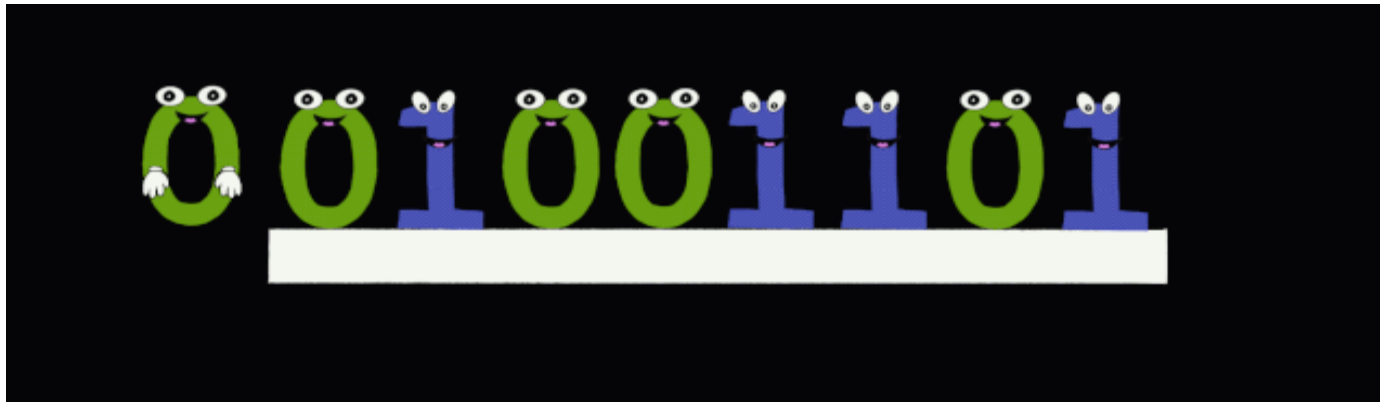
Shift right

4.

1 = 0000 0001

5.

1 >> 2 = 0000 0000



10 = 0000 1010

10 >> 2 = 0000 0010

4- Logic Operations

D-BUGERZ

1.

2.

3.

4.

5.

& (AND)

1001
&
1110
=
1000

| (or)

1001
|
1110
=
1111

! (NOT)

! 1 = 0

~ (compliment)

~ 1001 = 0110

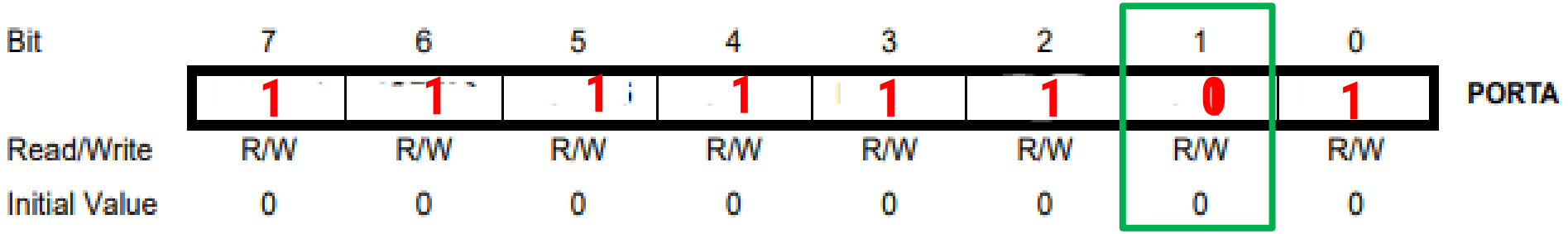
4- Logic Operations

D-BUGERZ

& (AND)

- we use **&** if we want to put 0 in register in specify

1. 1001 & 1110 = 1000



2. **Assume that Register have value 1111 1111**

3. **Initial Register Value 1111 1111**

4. **1 << 1 = 0000 0010**

5. **~ 1 << 1 = 1111 1101**

1 = 0000 0001

1 << 1 = 0000 0010

4- Logic Operations

D-BUGERZ

1.
2.
3.
4.
5.

& (AND)

1001
&
1110
=
1000

Bit	7	6	5	4	3	2	1	0	
	1	1	1	1	1	1	0	1	PORTA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Initial Register Value 1111 1111

1 << 1 = 0000 0010

~ 1 << 1 = 1111 1101

```
  1111 1111
&
  1111 1101
-------------
  1111 1101
```

1	= 0000 0001
1 << 1	= 0000 0010

4- Summarize AND &

D-BUGERZ

1.
2.
3.
4.
5.

& (AND)

1001
&
1110
=
1000

~ (compliment)

~ 1001 = 0110

- if we want to put 0 in register in any desired bit

```
PORTC =PORTC & (~(1<<1));
```

PORTC **1111 1111**

1 << 1 **= 0000 0010**

~ 1 << 1 **= 1111 1101**

1111 1111
&
1111 1101

1111 1101

4- Logic Operations

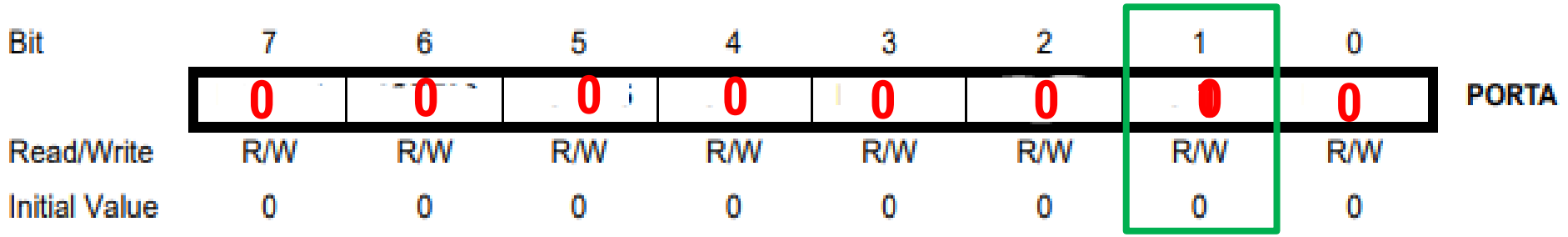
D-BUGERZ

1.
2.
3.
4.
5.

| (or)

1001
|
1110
=
1111

- we use | if we want to put 1 in register in specify bit



Assume that Register have value 0000 0000

Initial Register Value 0000 0000

1 << 1 = 0000 0010

1 = 0000 0001

1 << 1 = 0000 0010

0000 0000
| 0000 0010
0000 0010

4- Summarize Or |

D-BUGERZ

- if we want to put 1 in register in any desired bit

```
PORTC |= (1<<1);
```

1.

2.

3.

4.

5.

PORTC **0000 0000**

1 << 1 **= 0000 0010**

```
  0000 0000
| 0000 0010
-------------
  0000 0010
```

D-BUGERZ

1.

2.

3.

4.

5.

BREAK 10 MIN

Recall Last session

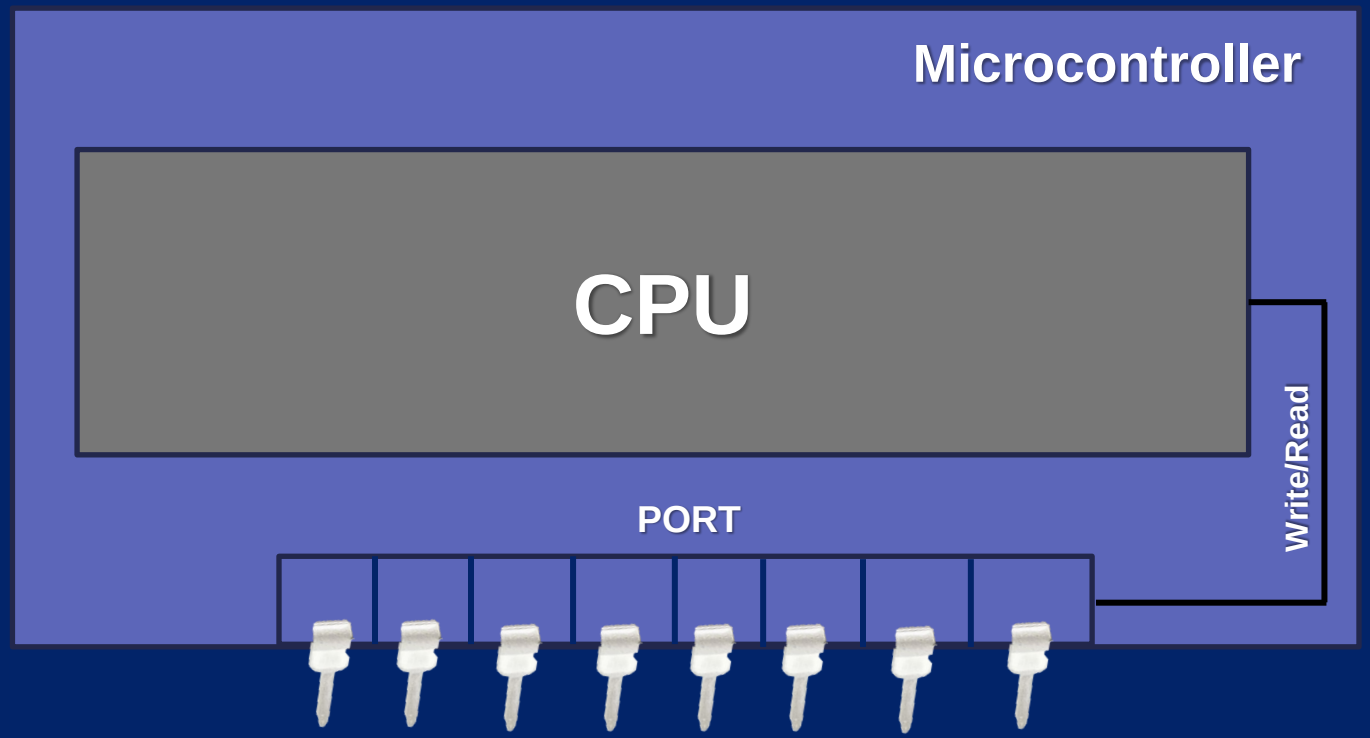
Recall Important C programming keys

Introduction to GPIO

Start Coding with GPIO

Introduction to Interrupt

Let’s Start with First Peripheral

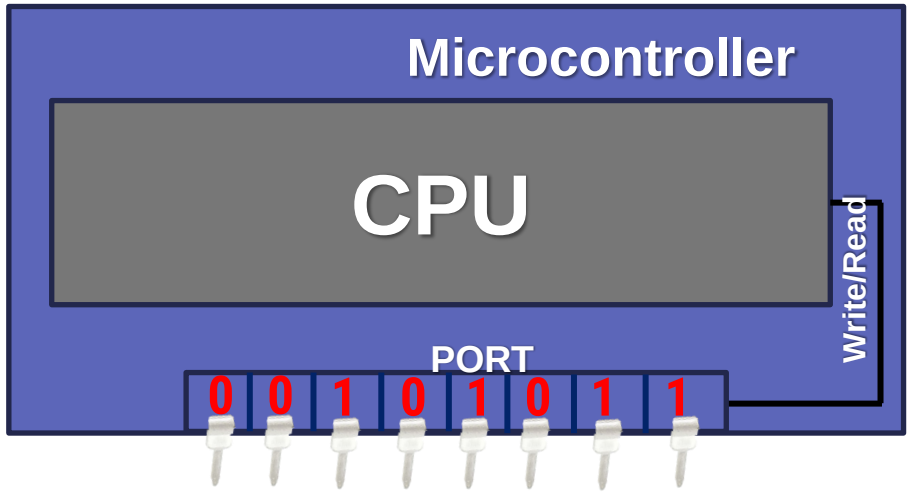


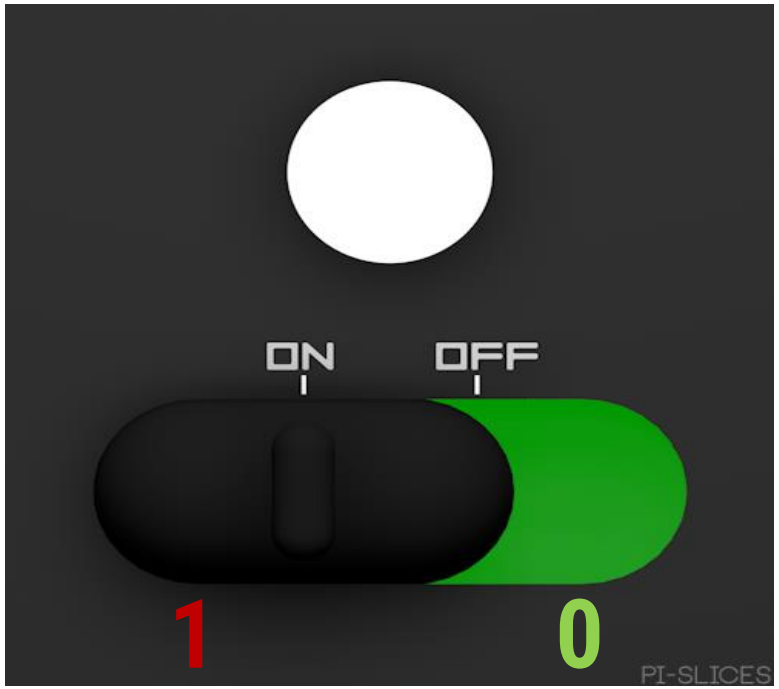
Introduction to GPIO

D-BUGERZ

1.
2.
3.
4.
5.

GPIO stands for **General Purpose Input/Output**. It's a standard interface used to connect microcontrollers to other electronic devices. For example, it can be used with sensors, diodes, displays, and System-on-Chip modules.



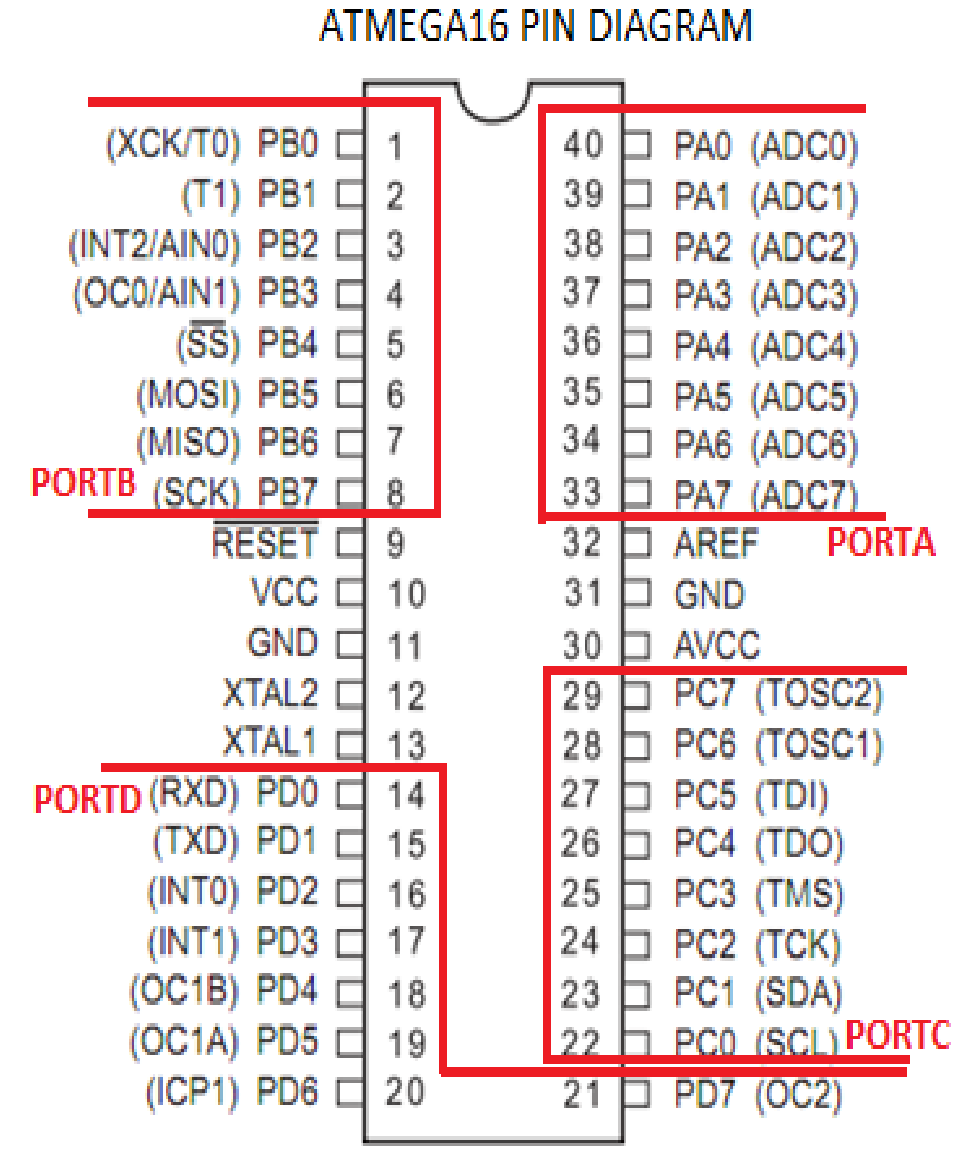


Introduction to GPIO

D-BUGERZ

1.
2.
3.
4.
5.

- ATmega16/32 Microcontrollers has 32 programmable I/O pins constituting four ports.
- Each port has 8 pins. The pins of these four ports can be used as general-purpose inputs/outputs.
- Each I/O pin could be Input or Output (Multiplexed Direction). This could be controlled through the I/O Port direction register.
- Each Microcontroller pin could be assigned to different I/O functions. Pin could be configured for digital I/O, analog input, timer I/O, or serial I/O. For example PDO in ATmega16/32 microcontrollers could be digital I/O or UART serial input (Multiplexed Functionality).



Introduction to GPIO

D-BUGERZ

1.
2.
3.
4.
5.

- Each PORT is controlled by 3 registers:

1. DDRX (Data Direction Register)

This register used to decide the **direction** of the pins, i.e. whether the pins will act **as input pins or as output pins**.

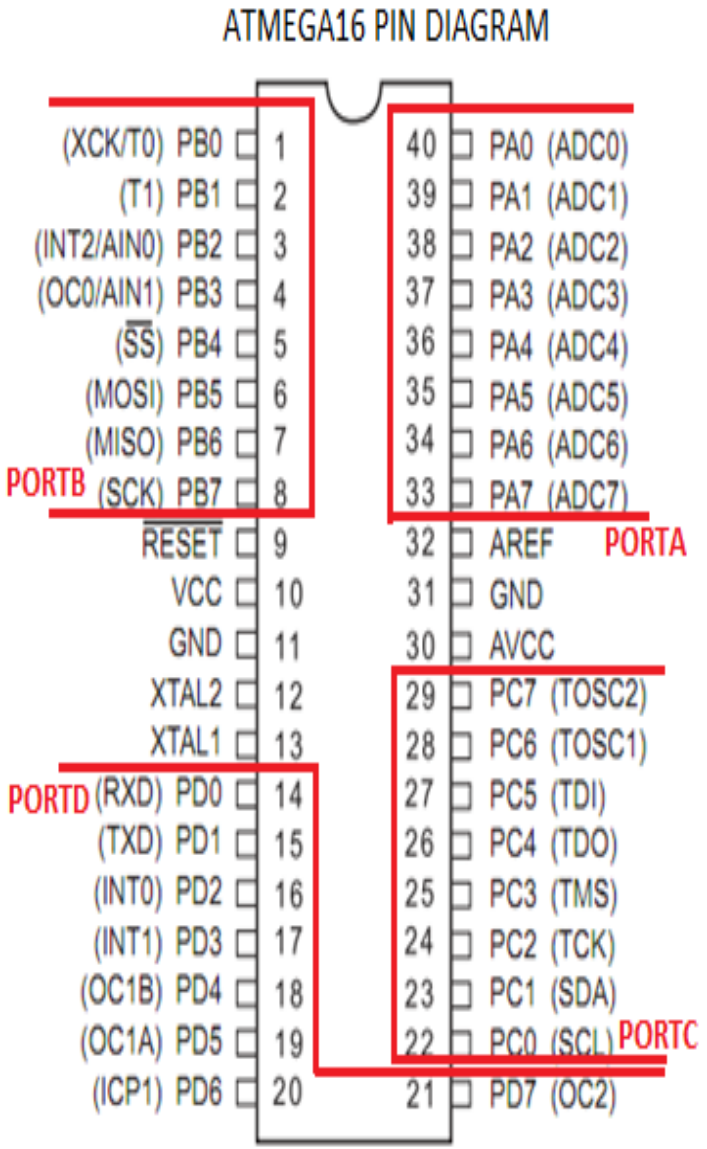
2. PORTX (Output Register)

This register is used to **set the logic** on the output pins HIGH or LOW.

3. PINX (Input Register)

This register is used to **read the logic** level on the port input pins.

- Note: x could be A, B, C, or D depending on which port registers are being addressed.



Introduction to GPIO

D-BUGERZ

1.
2.
3.
4.
5.
6.

PORTA

Port A Data Register – PORTA

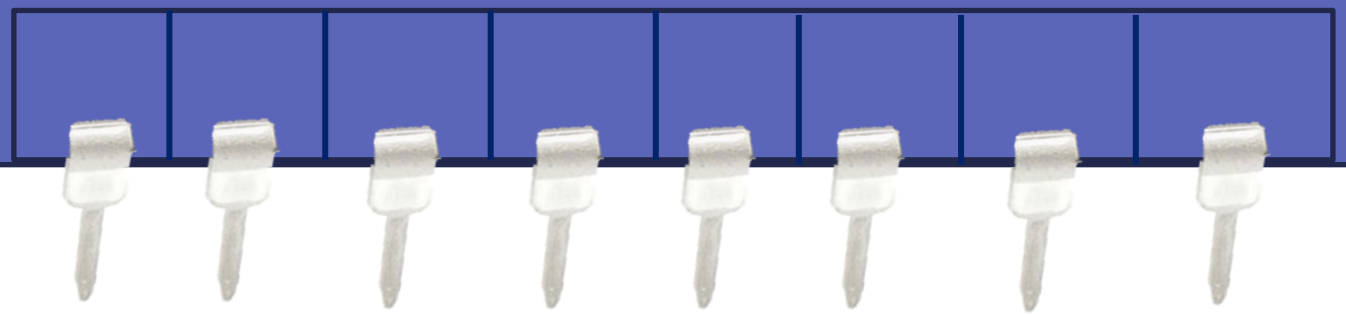
Bit	7	6	5	4	3	2	1	0	
	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0	PORTA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Port A Data Direction Register – DDRA

Bit	7	6	5	4	3	2	1	0	
	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0	DDRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Port A Input Pins Address – PINA

Bit	7	6	5	4	3	2	1	0	
	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0	PINA
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	



Introduction to GPIO

D-BUGERZ

- Example 1

Want Pin 2 in PORTA as Output and make output 1 on it

1.

2. DDRA = 0000 0100 (4 in decimal)(0x4 in HEXA)

3. PORTA = 0000 0100 (4 in decimal)(0x4 in HEXA)

4.

5.

Port A Data Register – PORTA

Bit	7	6	5	4	3	2	1	0	
	0	0	0	0	0	1	0	0	PORTA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Port A Data Direction Register – DDRA

Bit	7	6	5	4	3	2	1	0	
	0	0	0	0	0	1	0	0	DDRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Port A Input Pins Address – PINA

Bit	7	6	5	4	3	2	1	0	
	0	0	0	0	0	0	0	0	PINA
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

Introduction to GPIO

D-BUGERZ

- Example 2

Want Pin 5 in PORTA as Output and make output 0 on it

1.

2. DDRA = 0010 0000 (32 in decimal)(0x20 in HEXA)

3. PORTA = 0000 0000 (0 in decimal)(0x0 in HEXA)

4.

5.

Port A Data Register – PORTA

Bit	7	6	5	4	3	2	1	0	
	0	0	0	0	0	0	0	0	PORTA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Port A Data Direction Register – DDRA

Bit	7	6	5	4	3	2	1	0	
	0	0	1	0	0	0	0	0	DDRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Port A Input Pins Address – PINA

Bit	7	6	5	4	3	2	1	0	
	0	0	0	0	0	0	0	0	PINA
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

Introduction to GPIO

D-BUGERZ

❑ Example 3

1. Want Pin 3 in PORTA as Output and pin 7 as input and make output 1 on Pin 3 and check input on bit 7

2.

3. DDRA = 0000 1000 (8 in decimal)(0x8 in HEXA)

DDRA = 0000 1000 (8 in decimal)(0x8 in HEXA)

4. PORTA = 0000 1000 (8 in decimal)(0x8 in HEXA)

5. To check on input :

```
if(PINA & (1<<7));
```

Port A Data Register – PORTA

Bit	7	6	5	4	3	2	1	0	
	0	0	0	0	0	0	1	0	PORTA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Port A Data Direction Register – DDRA

Bit	7	6	5	4	3	2	1	0	
	0	0	0	0	1	0	0	0	DDRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Port A Input Pins Address – PINA

Bit	7	6	5	4	3	2	1	0	
	0	0	0	0	0	0	0	0	PINA
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

We Don't write in this register we only read from it

Introduction to GPIO

D-BUGERZ

1. 2. 3. 4. 5.

To check on input :

```
If(PINA & (1<<7));
```

This bit will change with external device as when we press on push button

```
  1000 0000
&
  1000 0000
-----------
  1000 0000 = 128
```

if(128) equivalent to if (TRUE)

SO will enter condition

Port A Data Register – PORTA

Bit	7	6	5	4	3	2	1	0	
	0	0	0	0	0	0	1	0	PORTA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Port A Data Direction Register – DDRA

Bit	7	6	5	4	3	2	1	0	
	0	0	0	0	1	0	0	0	DDRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Port A Input Pins Address – PINA

Bit	7	6	5	4	3	2	1	0	
	0	0	0	0	0	0	0	0	PINA
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

We Don't write in this register we only read from it

Question??

D-BUGERZ

Want Pin 7 & Pin 2 in PORTA as Output and pin 0 as input and make output 1 on Pin 7 and 0 on Pin 2 and check input on bit 0 ??

Pin 7 DDRA = 1000 0000 (128 in decimal)(0x80 in HEXA)

Pin 2 DDRA = 1000 0100 (132 in decimal)(0x84 in HEXA)

Pin 0 DDRA = 1000 0100 (132 in decimal)(0x84 in HEXA)

Pin 7 PORTA = 1000 0000 (128 in decimal)(0x80 in HEXA)

Pin 2 PORTA = 1000 0000 (128 in decimal)(0x80 in HEXA)

To check on input Pin 0 :

Pin 0 `if(PINA & (1<<0));`

Port A Data Register – PORTA

Bit	7	6	5	4	3	2	1	0	
	1	0	0	0	0	0	0	0	PORTA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Port A Data Direction Register – DDRA

Bit	7	6	5	4	3	2	1	0	
	1	0	0	0	0	1	0	0	DDRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Port A Input Pins Address – PINA

Bit	7	6	5	4	3	2	1	0	
	0	0	0	0	0	0	0	0	PINA
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

We Don't write in this register we only read from it

D-BUGERZ

1.

2.

3.

4.

5.

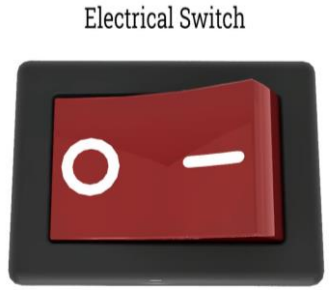
Introduction to GPIO

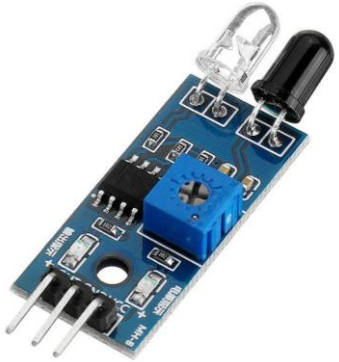
D-BUGERZ

1.
2.
3.
4.
5.

❑ Input devices

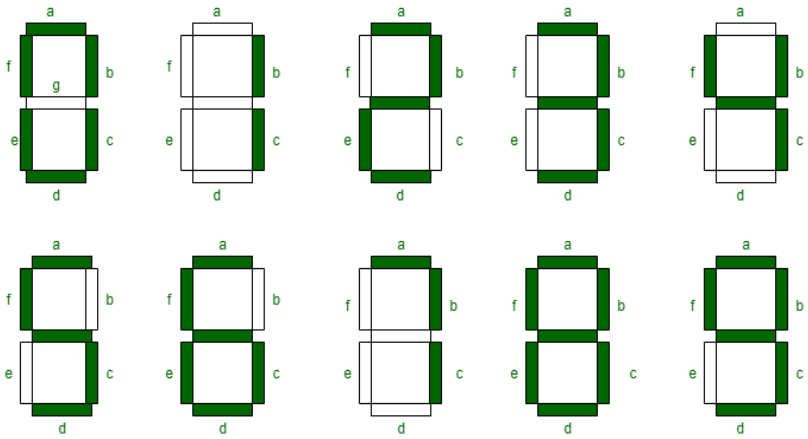
1- Switches(Push button, Keypad etc.)
2- Digital /Analogue sensor
3- Signal from another microcontroller





❑ Output devices

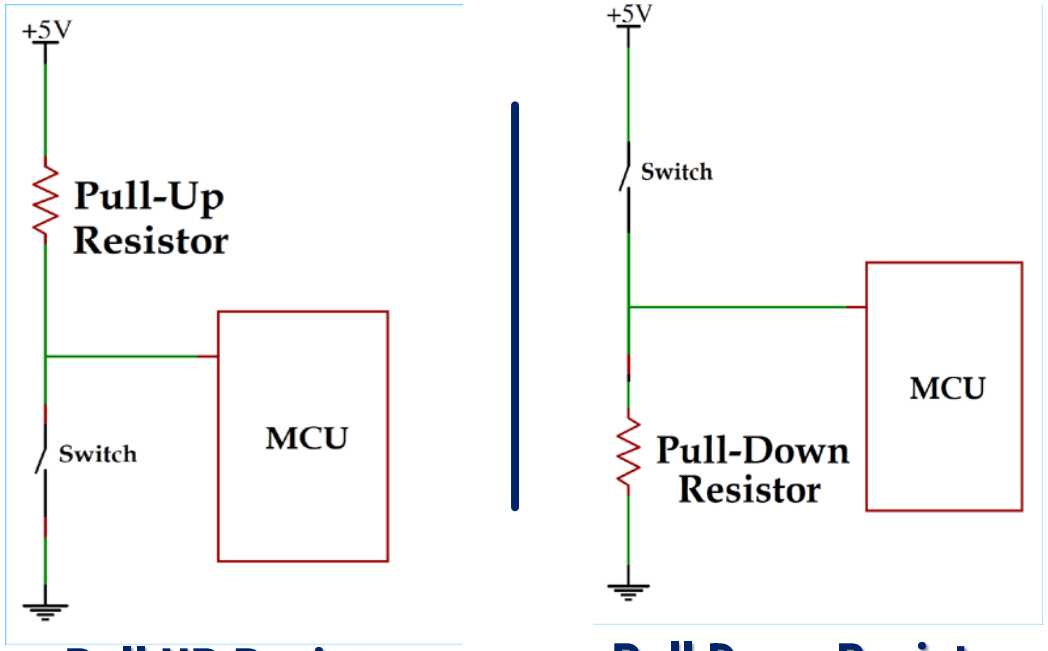
1- LED
2- 7-SEGMENT
3- LCD display
4- Motors
5- Buzzer



Connection

1.
2.
3.
4.
5.

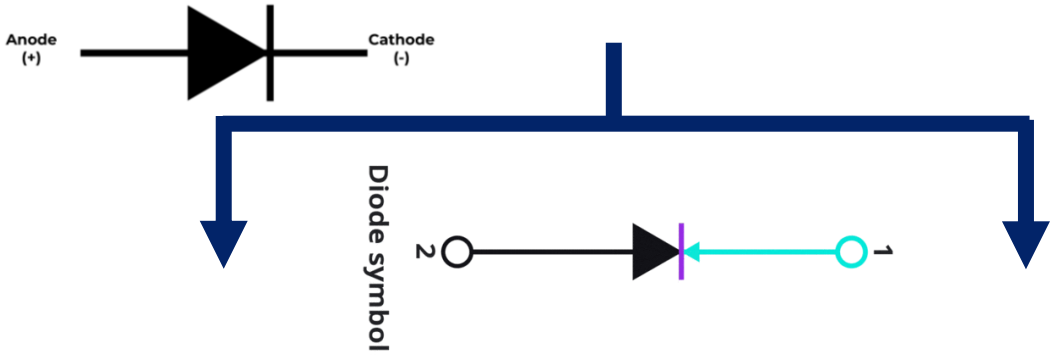
Switches(Push button, Keypad etc.)

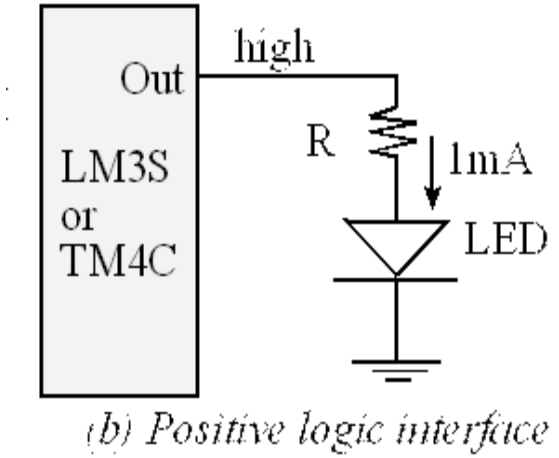


Pull UP Resistor

Pull Down Resistor

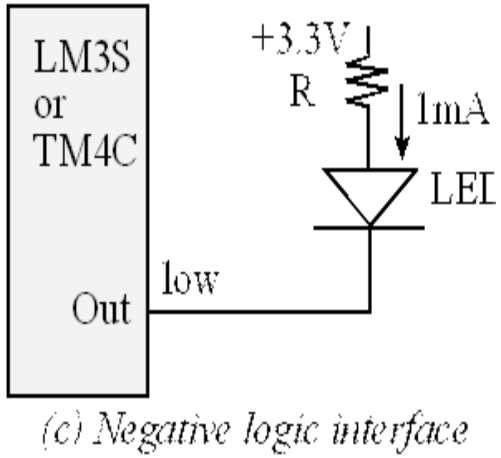
LED





(b) Positive logic interface

Positive Logic



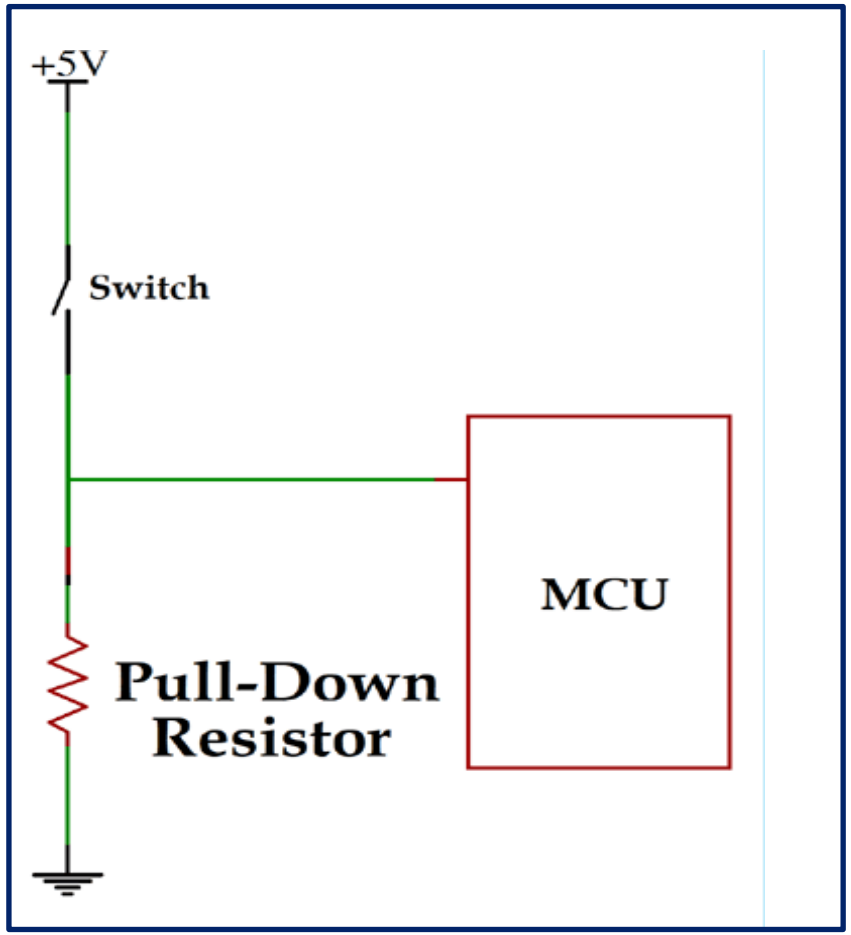
(c) Negative logic interface

Negative Logic

Connection

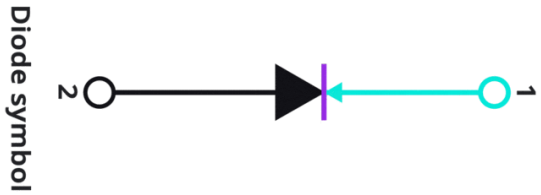
1.
2.
3.
4.
5.

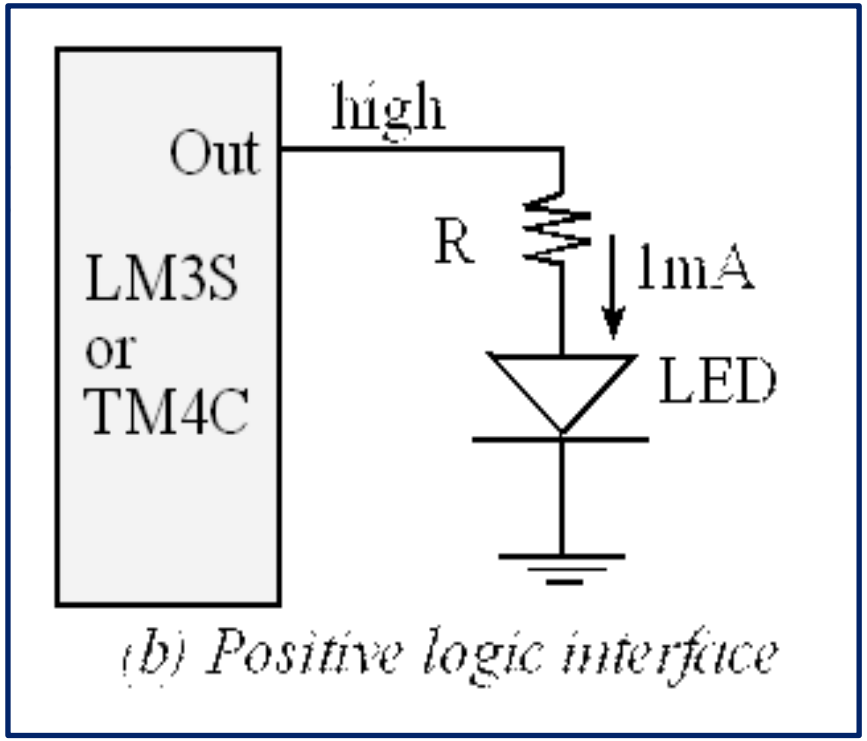
Switches(Push button, Keypad etc.)



Pull Down Resistor

LED





Positive Logic

D-BUGERZ

1.

2.

3.

4.

5.

D-BUGERZ

Recall Last session

Recall Important C programming keys

Introduction to GPIO

Start Coding with GPIO

Introduction to Interrupt

Coding by Interrupt

Let’s Start with Our First Embedded Code

Simple Application Structure Using AVR Library

D-BUGERZ

1.
2.
3.
4.
5.

```
7
8  #include <avr/io.h>
9
10 int main(void)
11 {
12     /********** Initialization Code **********/
13
14
15     /********** Super Loop **********/
16     while(1)
17     {
18         /********** Application Code **********/
19
20     }
21 }
22
23
```

AVR include Library

Initialization like you define pin is output or input or any initialize parameter you need

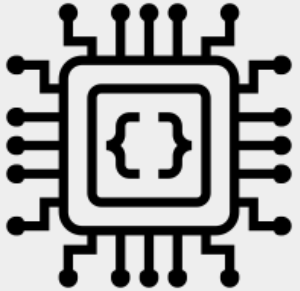
Your program which will execute till infinity

Exercise 1

D-BUGERZ

1.
2.
3.
4.
5.

Write Embedded C code using ATmega16/32 µC to control a Led using a Push Button.



Requirements:

1- Configure the µC clock with 16Mhz Crystal Oscillator.

2- The Push Button is connected to pin 2 in PORTD using Pull Down configuration.

3- The Led is connected to pin 1 in PORTC using Positive Logic configuration.

4- Turn on the Led when the Push Button status is pressed.

5- Turn off the Led when the Push Button status is released.

Delay

D-BUGERZ

- If you want to stop Code for interval of time

You must include #include <util/delay.h>

```
#include <util/delay.h> // for delay function
```

1.

2.

- Use in code to stop code for 1000ms

3.

```
_delay_ms(1000);
```

4.

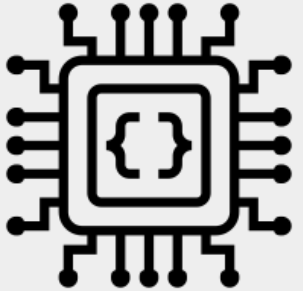
5.

Exercise 2

D-BUGERZ

1.
2.
3.
4.
5.

Write Embedded C code using ATmega16/32 µC to control a LED.



Requirements:

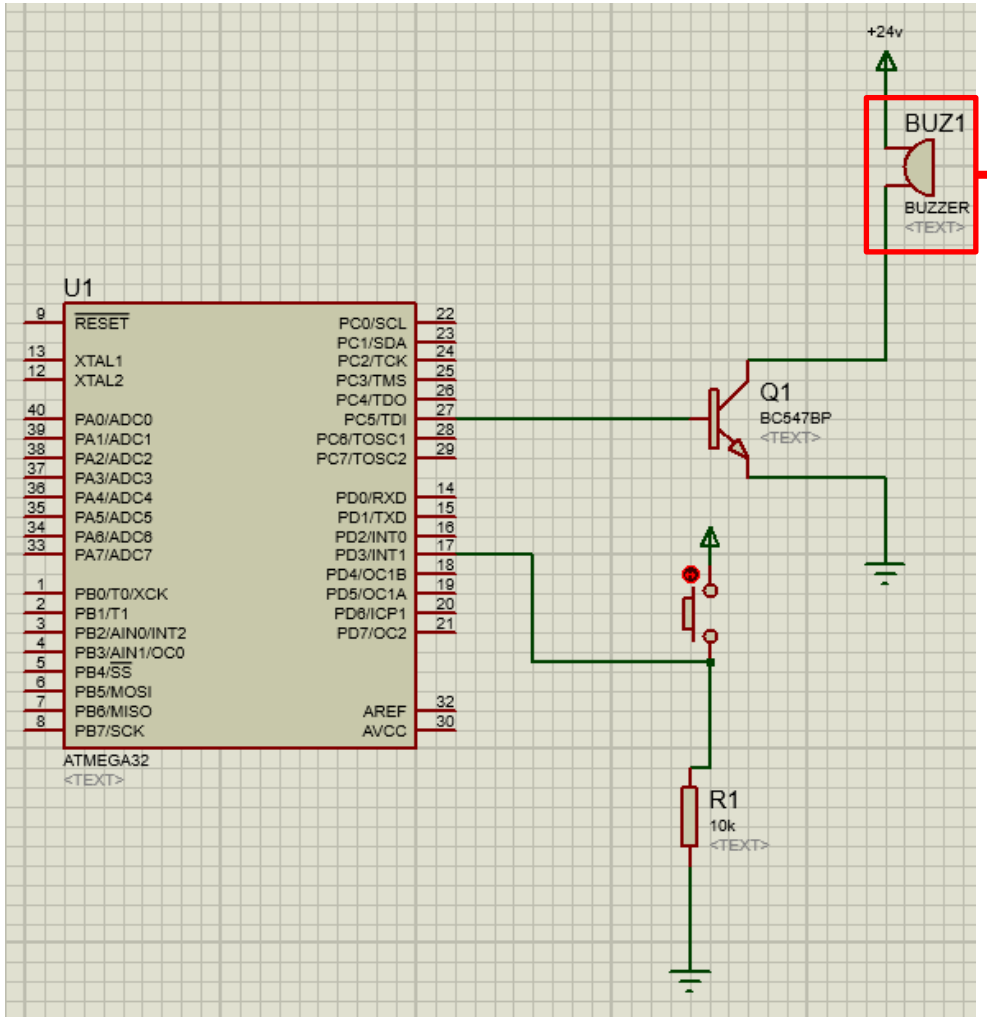
1- Configure the µc clock with 16Mhz Crystal Oscillator.
2- The led is connected to pin 2 in PORTC.
3- Connect the Led using Positive Logic configuration
4- Flash the led every 1 second.

Interfacing With Buzzer

D-BUGERZ

1.
2.
3.
4.
5.

- Buzzer is an electrical device, which is similar to a bell that makes a buzzing noise and is used for signalling. Typical uses of buzzers and beepers include alarm devices, timers and confirmation of user input such as a mouse click or keystroke

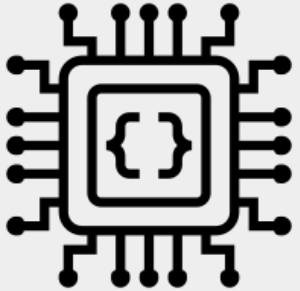


Exercise 3

D-BUGERZ

1.
2.
3.
4.
5.

Write Embedded C code using ATmega16/32 µC to control a buzzer using a push button.



Requirements:

1- Configure the µC clock with 16Mhz Crystal Oscillator.

2- The Push Button is connected to pin 3 in PORTD using Pull Down configuration.

3- The Buzzer is connected to pin 5 in PORTC.

4- The buzzer should make a noise when the push button is pressed and stay silent when the push button is released.

Challenge 1

D-BUGERZ

Write Embedded C code using ATmega16/32 µC to control two Leds using two Push Buttons.

Requirements:

1.
2.
3.
4.
5.

- ✓ Configure the PC clock with internal 1Mhz Clock.
- ✓ The Push Buttons 1 & 2 are connected to pin 0 & 1 in PORTB.
- ✓ Connect both Push Buttons using Pull Down configuration.
- ✓ The Led 1 & 2 is connected to pin 0 & 1 in PORTC.
- ✓ Connect both Leds using Positive Logic configuration.
- ✓ If Push Button 1 is pressed just turn on the first Led 1 only
- ✓ And if Push Button 2 is pressed just turn on Led 2 only.
- ✓ In case both Push Buttons are pressed both Leds are on.
- ✓ In case no Push Buttons are pressed both Leds are off.

Challenge 2

D-BUGERZ

Write Embedded C code using ATmega16/32 µC to control 3-LEDs.

1.

Requirements:

2.

- ✓ **Configure the µC clock with 16Mhz Crystal Oscillator.**
3. ✓ **Use the 3 LEDs at PC0, PC1 and PC2.**
- ✓ **LEDs are connected using Negative Logic configuration.**
4. ✓ **A roll action is perform using the LEDs each led for half second. The first LED is lit and roll down to the last LED then back to the first LED. This operation is done continuously.**
5. ✓ **After this Sequence make the buzzer ON for 2 second.**

D-BUGERZ

1.

2.

3.

4.

5.

Next Session

- Interface with PIR sensor
- Introduction to Interrupts
- Coding with External Interrupt
- Introduction to Timers

D-BUGERZ

Do you have any questions?

hello@mail.com
555-111-222
mydomain.com

D-BUGERZ