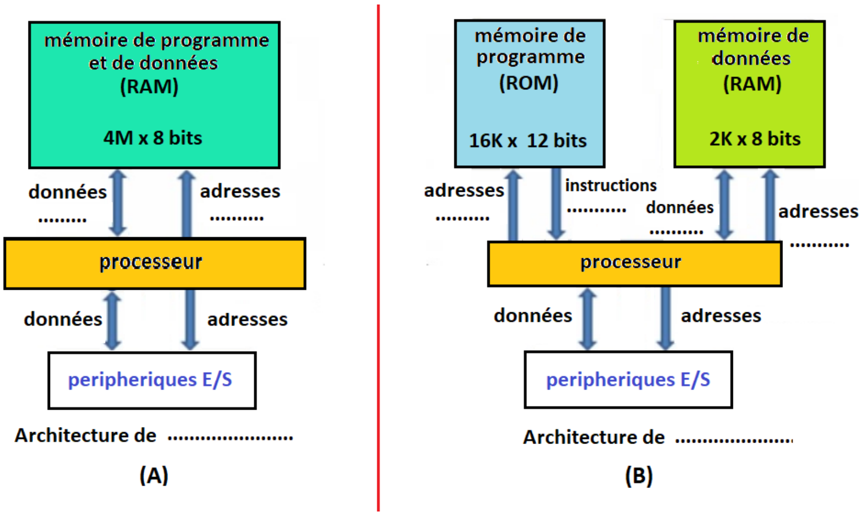
Tutorial exercises No 8

Exercise 01:

1. Determine the type of architecture represented by each block diagram.
2. Determine the size of the buses (address, data, instructions) for each block diagram.



3. Where can microcontrollers be found?
4. Do all microcontrollers necessarily have to be of Harvard architecture?
5. List 6 components that can be found in a microcontroller.

Exercise 02:
Course Questions

1. What type of architecture does the PIC16F84 have (RISC/CISC)? Provide the total number of its instructions.
2. What are the bit sizes of the instructions and data memory of the PIC16F84?
3. What circuits are essential for the functioning of the microcontroller?
4. If the external clock of the PIC16F84 microcontroller has an 8MHz quartz, determine the instruction cycle time.
5. What type of stack does the PIC16F84 have (FIFO, LIFO, etc.) and how many levels does it contain?
6. What is the size of the program memory in the PIC16F84?
7. How many peripheral interrupts can be handled by the PIC16F84? List them.
8. Which bit, when set to '1', is used to enable all interrupts in the PIC16F84?
9. If the TMR0 interrupt is enabled (**T0IE = 1**), determine when the interrupt occurs.
10. Which instruction signals the end of an interrupt subroutine and forces the processor to return to the main program?

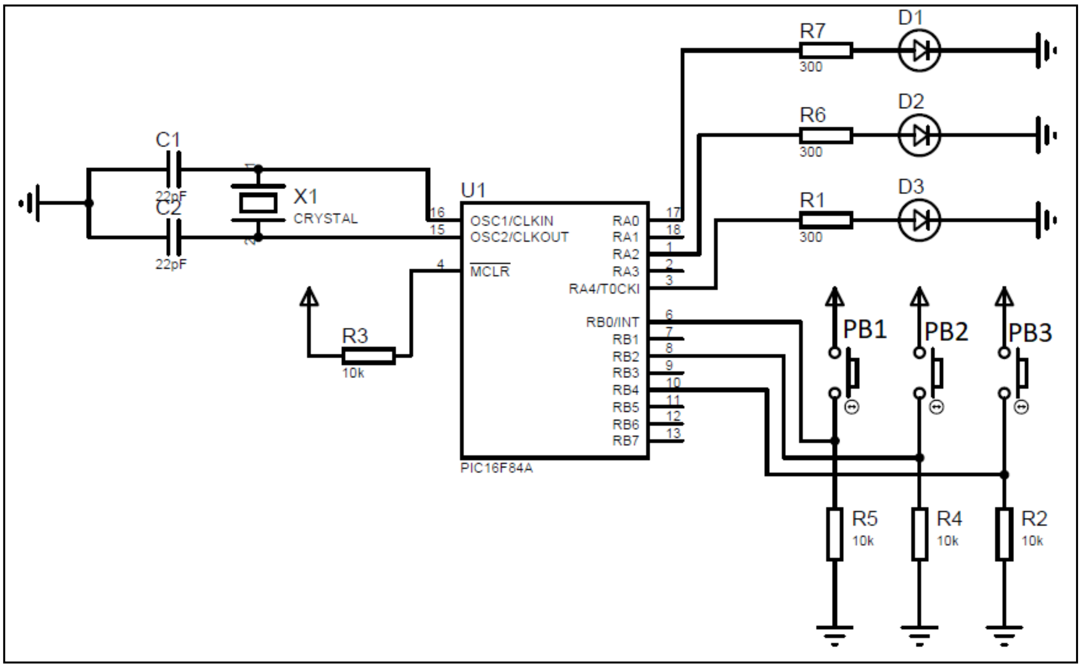
Exercise 03:
Answer with (True) or (False) to the following statements and correct the false ones.

1. In a Harvard architecture, the processor can easily read/write data and access instructions at any time.
2. When powering up the PIC16F84, the processor points to address **0000H** in the program memory.
3. A high voltage level on the (MCLR) pin causes a reset of the PIC16F84.
4. If a peripheral triggers an interrupt, the processor jumps to address **0004H**.
5. The **TRISA** and **TRISB** registers determine the direction of **PORTA** and **PORTB**, respectively.
6. The **SLEEP mode** is used to minimize power consumption in battery-powered applications.

7. An interrupt on the **RB0/INT** pin can wake up the PIC from **SLEEP mode**.
8. Upon powering up the PIC16F84, the **TMR0** register increments automatically.
9. The PIC16F84 stack allows saving the return address of the **PC** during subroutine calls (**CALL**) or when an interrupt occurs.
10. The programmer must clear the flag signaling the interrupt after handling it.
11. The **RETFIE** instruction restores the **PC** register content from the stack, allowing the processor to resume the program it interrupted.
12. The PIC16F84 can store memory data in the stack.
13. In a Von Neumann architecture, execution speed is faster because the processor fetches data and instructions simultaneously.

Exercise 04:

Consider the diagram below, consisting of 3 push buttons (**PB1**, **PB2**, and **PB3**) connected to **RB0**, **RB2**, and **RB4**, respectively, and three LEDs (**D1**, **D2**, and **D3**) connected to **RA0**, **RA2**, and **RA4**, respectively. The setup is managed by a PIC16F84 microcontroller.



We will describe the operation of the above setup for the button **PB1** and the LED **D1**, but the reasoning is the same for the other components.

- If the button **PB1** is pressed, the input **RB0** is read as a logical '1', the output **RA0** must be set to a high voltage level ≈5Volts (logical '1'), and the LED lights up.
- If the button **PB1** is released, the input **RB0** is read as a logical '0', the output **RA0** must be set to a low voltage level (logical '0'), and the LED turns off.

Write a program to read the push buttons **PB1**, **PB2**, and **PB3** and display their states on **D1**, **D2**, and **D3**, respectively. Below is a proposed structure for the assembler program, which can be adopted by the programmer:

```
org         0x000                         ; Start address after reset
            goto    init                  ; Address 0: initialize

init
; INITIALIZATION OF PORTS
            .
```

```
        .
        .
        .
        goto    start                   ; Jump to the main program

; MAIN PROGRAM
Start
        .
        .
        .
        goto    start                   ; Loop
        END                                     ; End of program directive
```