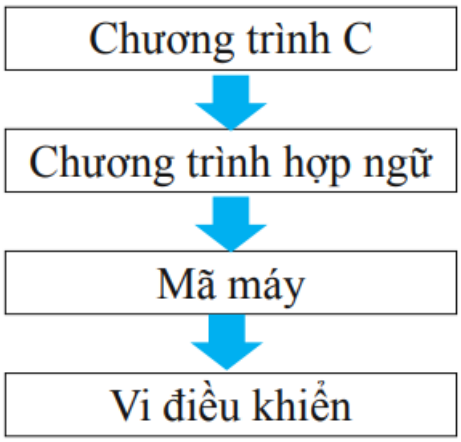
Nạp chương trình cho PIC



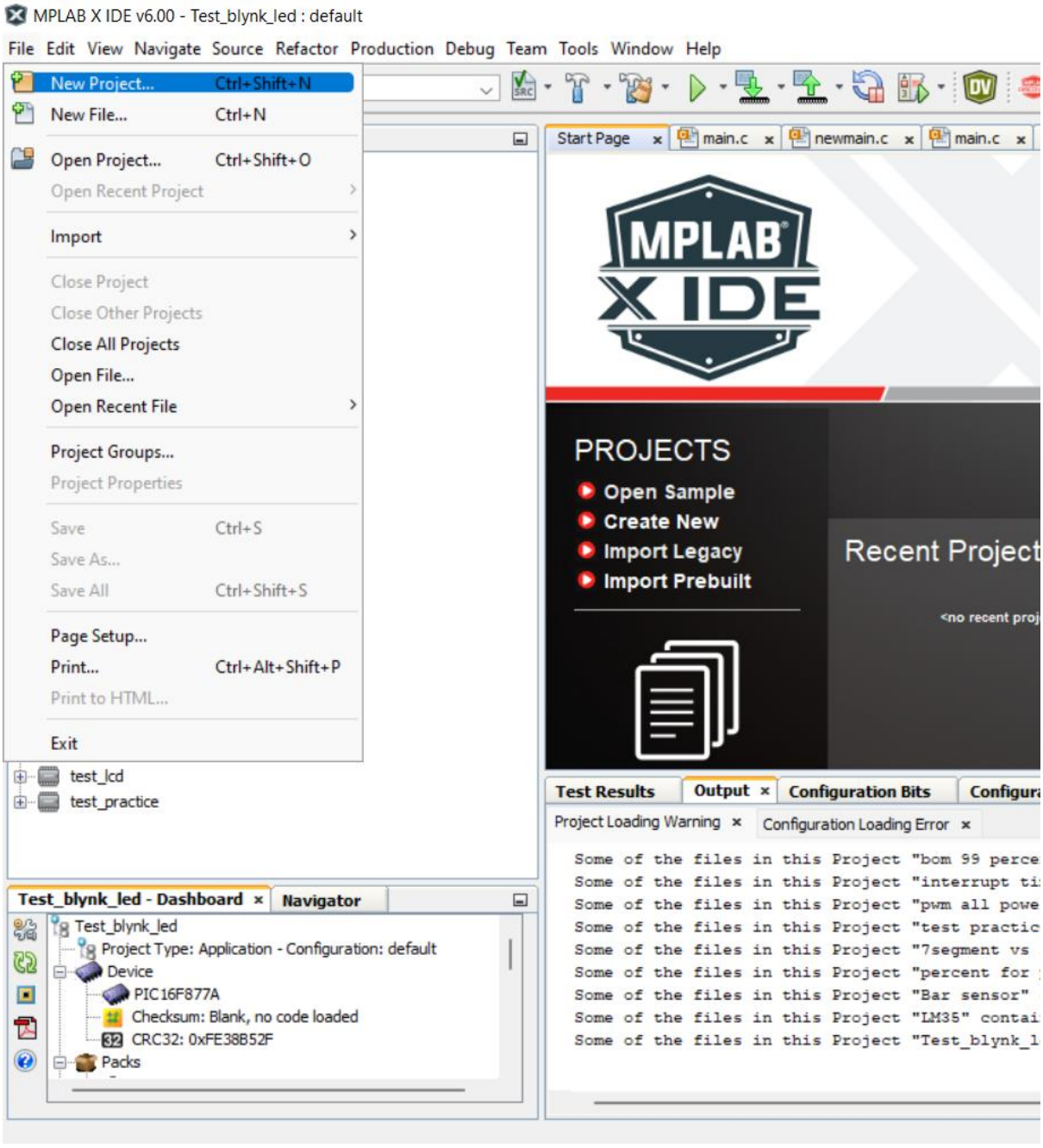
Mạch nạp cho PIC .

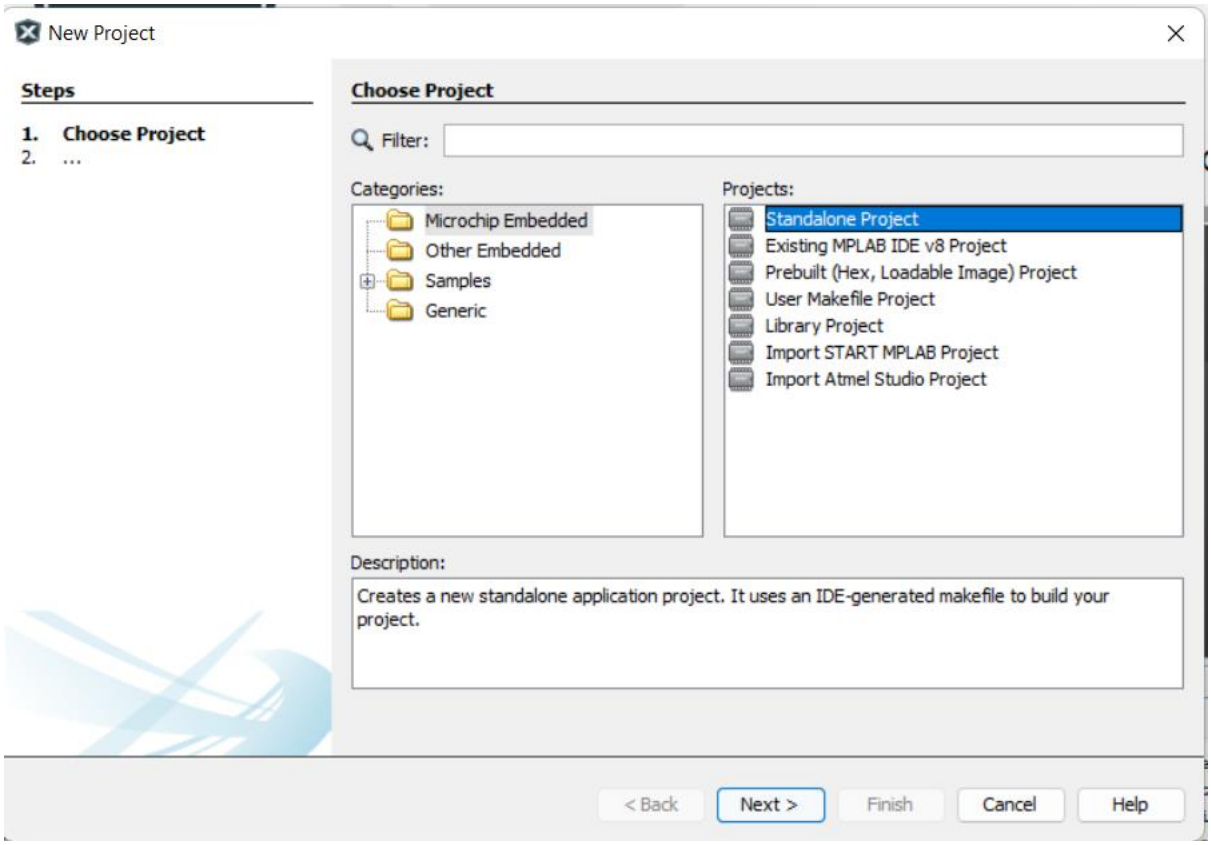
- Mạch nạp chính hãng Microchip: PICSTART Plus, MPLAB ICD 2, MPLAB PM3, PRO MATE II,… Giá thành cao, nạp nhiều chế độ
- Mạch nạp do bên thứ 3 sản xuất: P16PRO40, mạch nạp Universal của Williem
- Tự làm mạch nạp.

Sử dụng Mplab

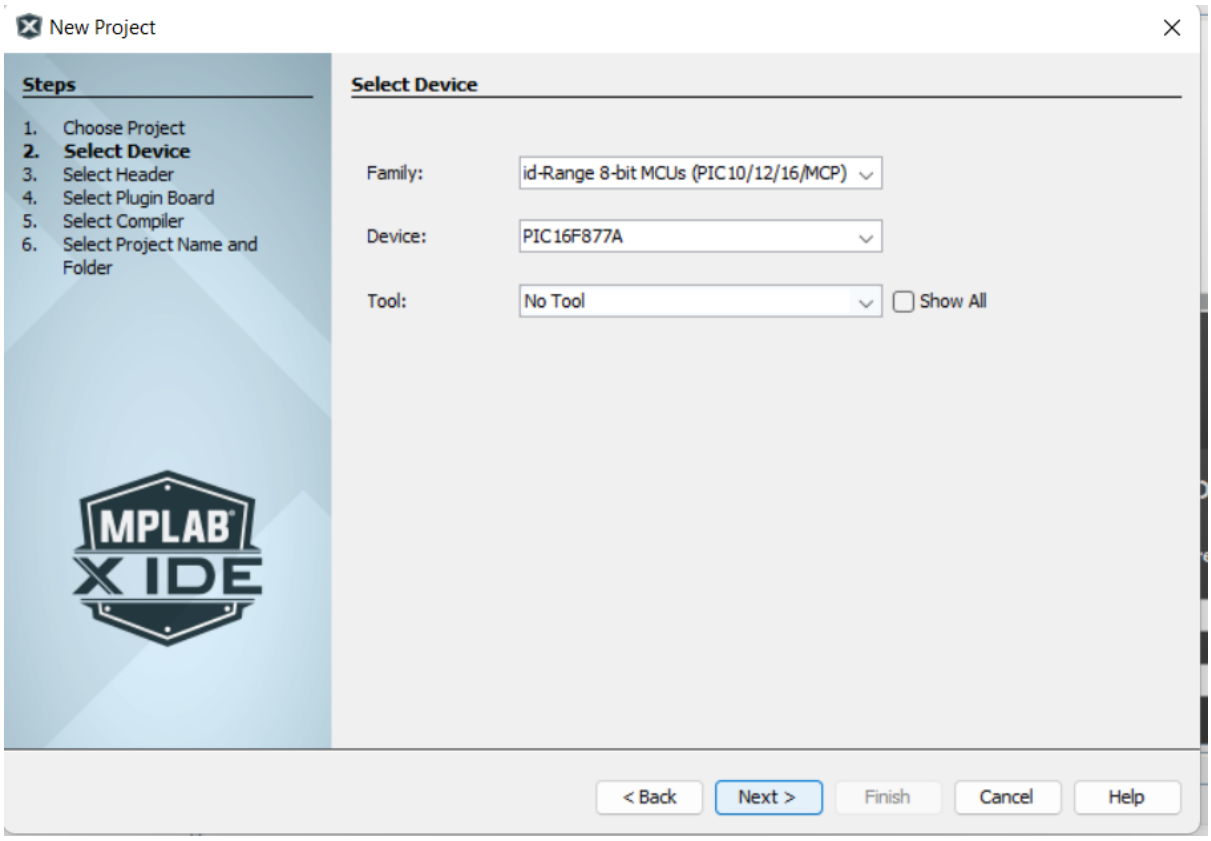
- Tạo Project mới

  Bước 1: Mở phần mềm **MPLABxIDE,** chọn **File -> New project** (hoặc tổ hợp phím Ctrl + Shift + N) **-> Microchip Embedded -> Standalone Project**, sau đó bấm **Next**

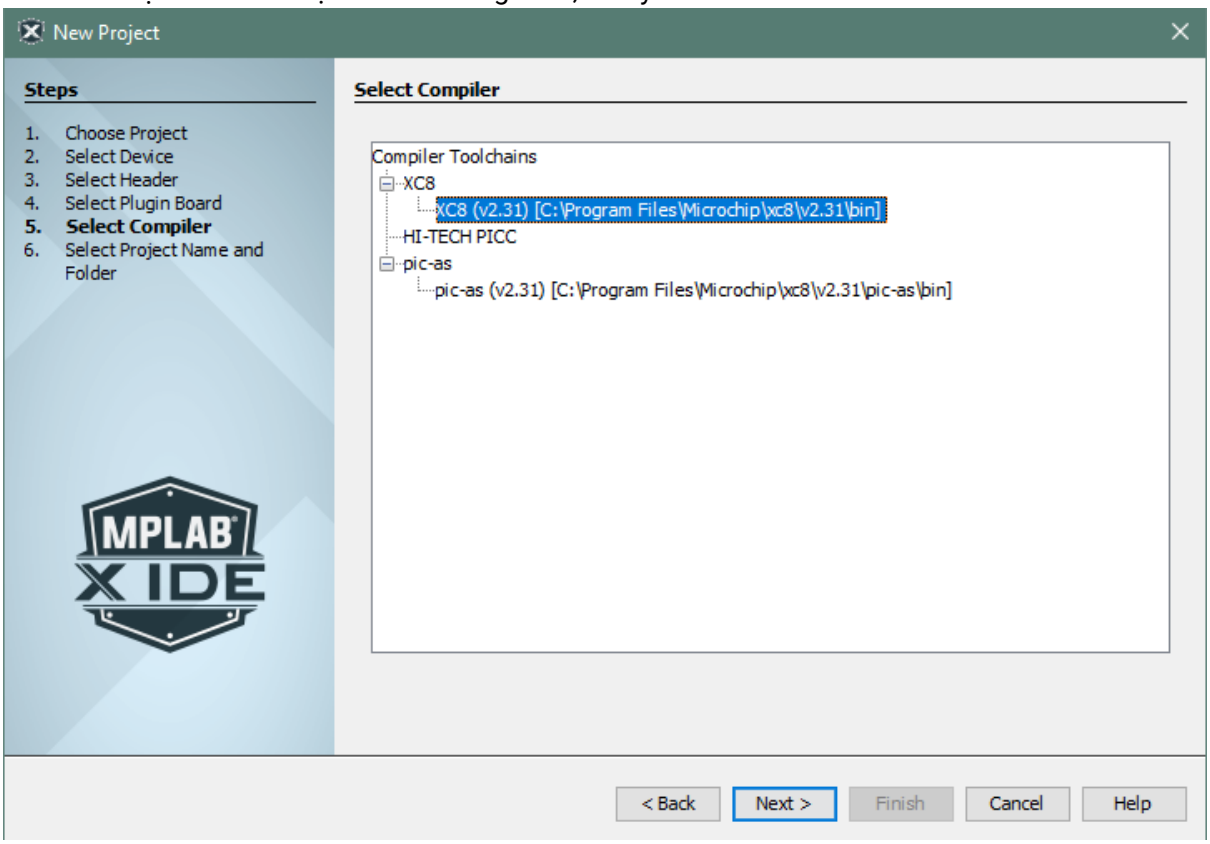




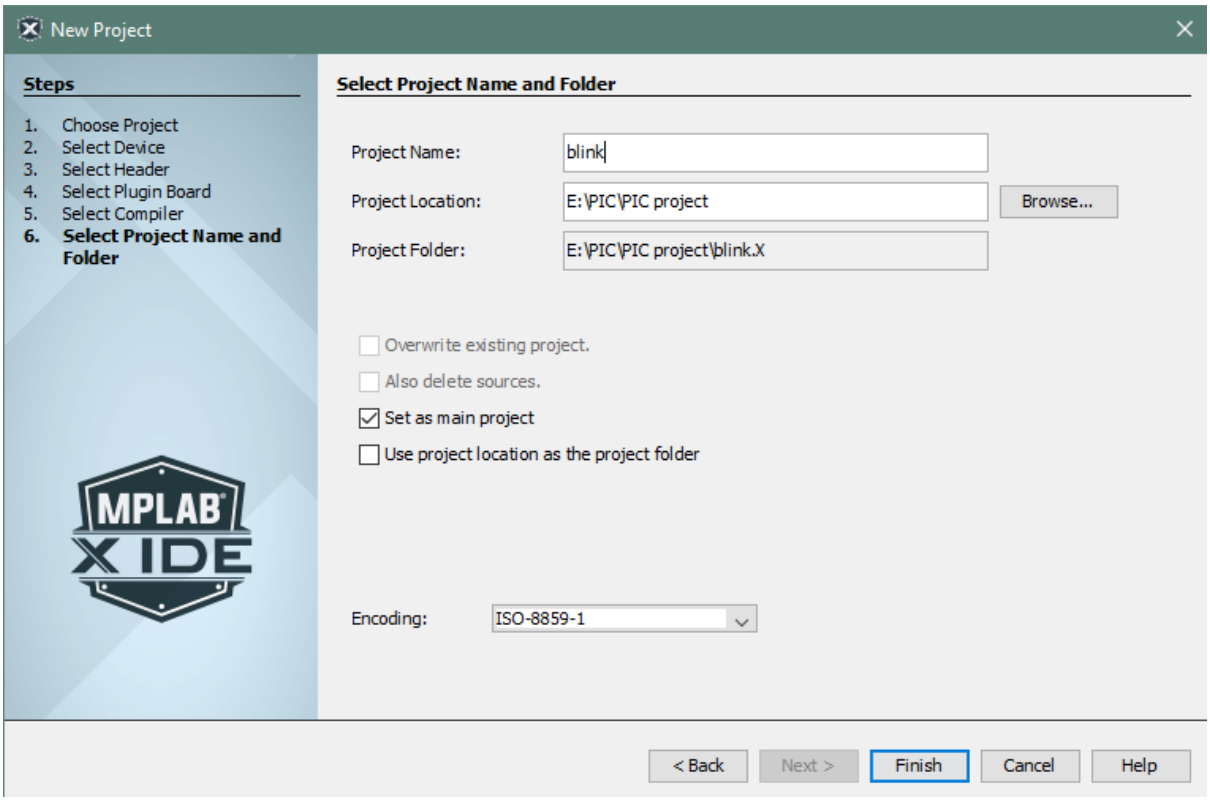
Bước 2: Chọn chip PIC16F887A để lập trình



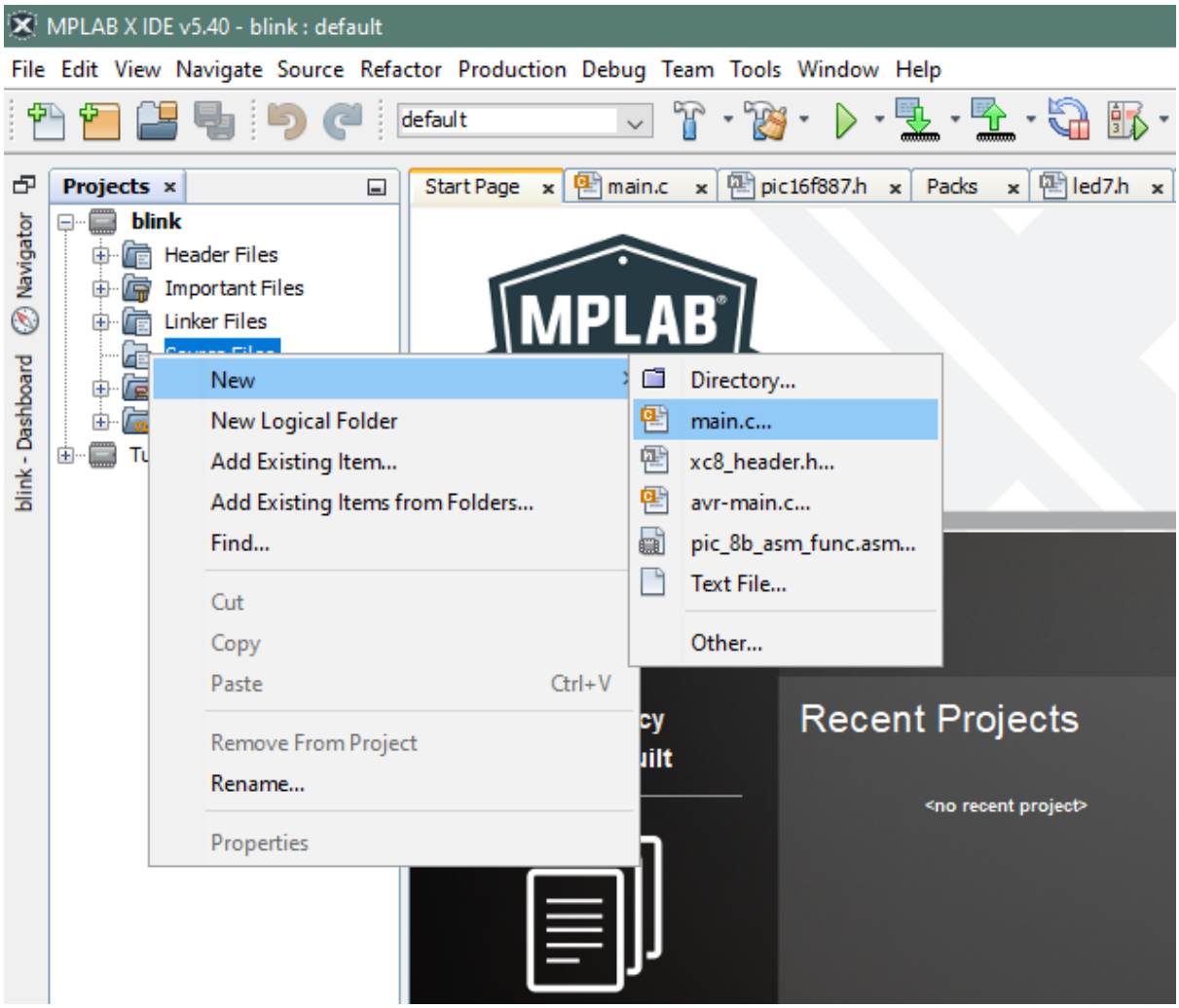
Bước 3: Chọn trình biên dịch cho chương trình, ở đây là **XC8**. Sau đó bấm **Next**



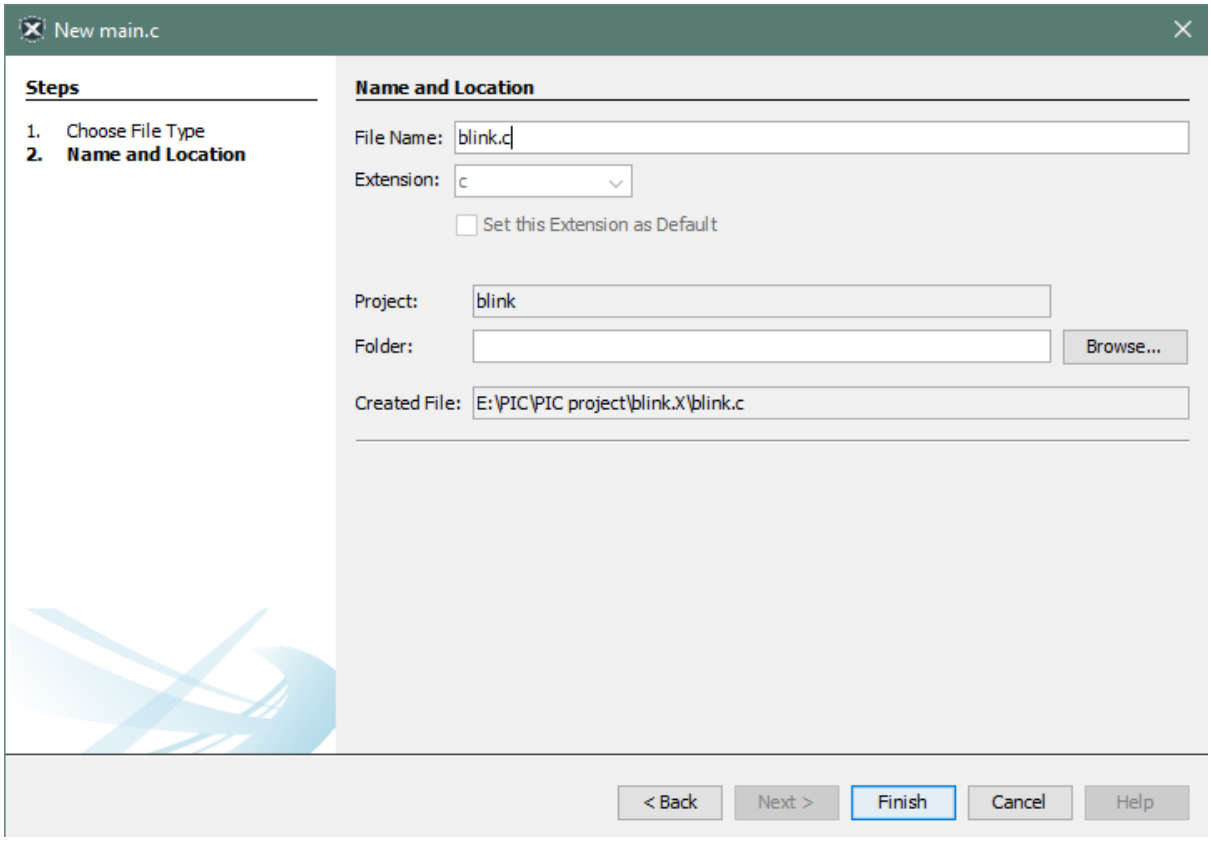
Bước 4: Đặt tên cho project và chọn nơi lưu project. Sau đó nhấn **finish**



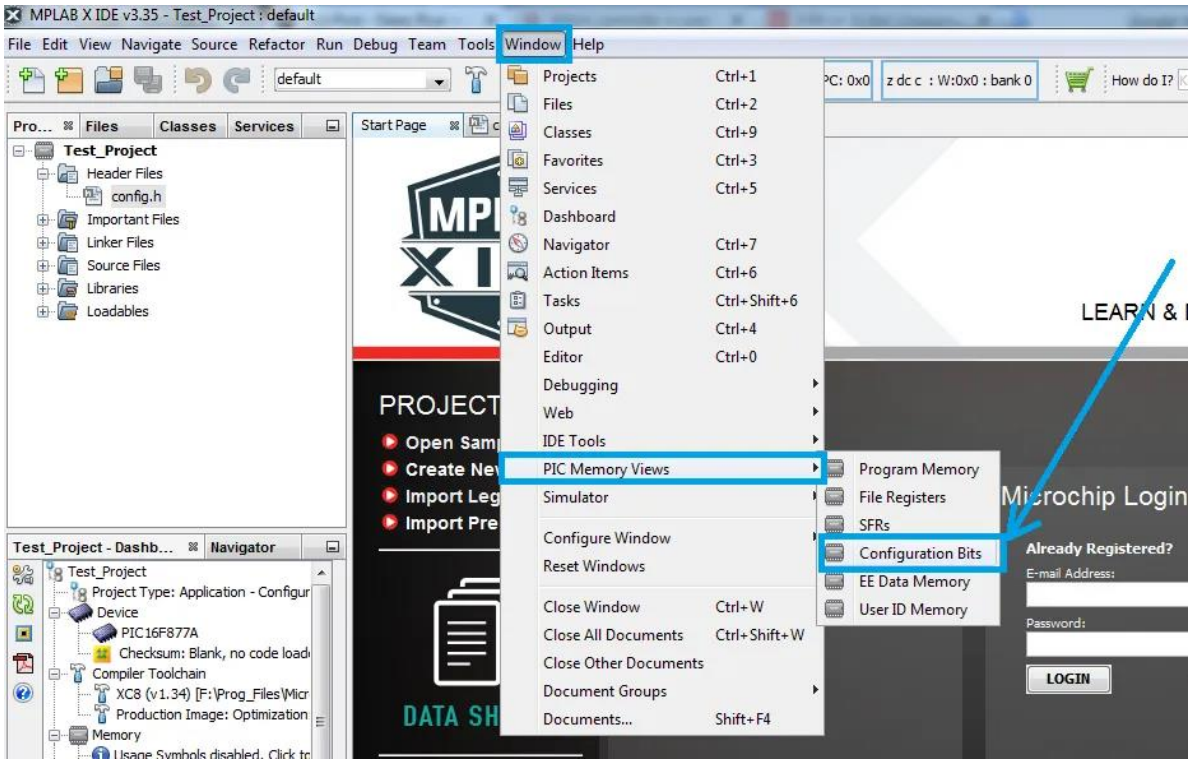
Bước 5: Sau khi tạo xong thư mục chứa project, click chuột phải vào **Source file**, chọn **New -> main.c**

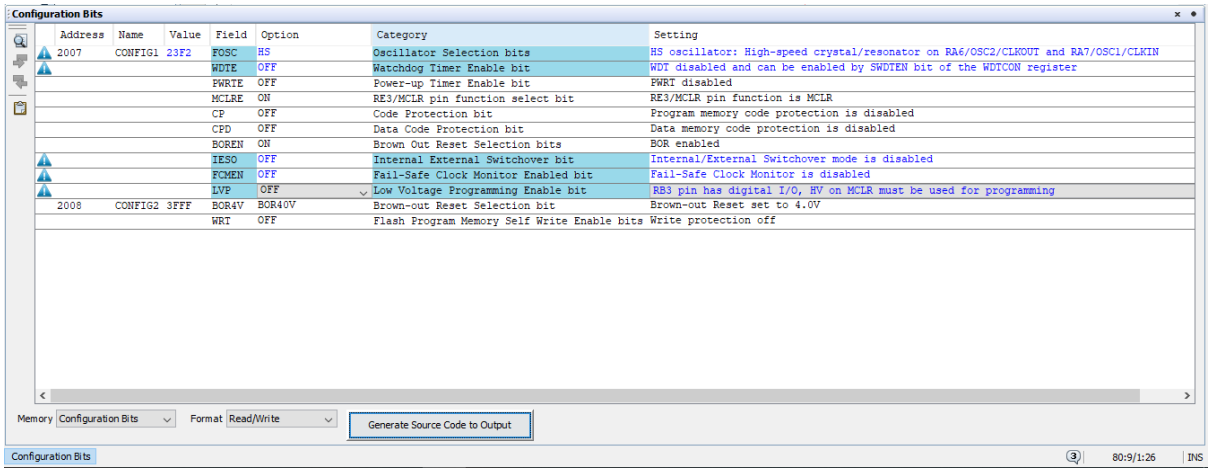


Bước 6: Tiến hành đặt tên cho file và nhấn **finish** để bắt đầu vào giao diện code

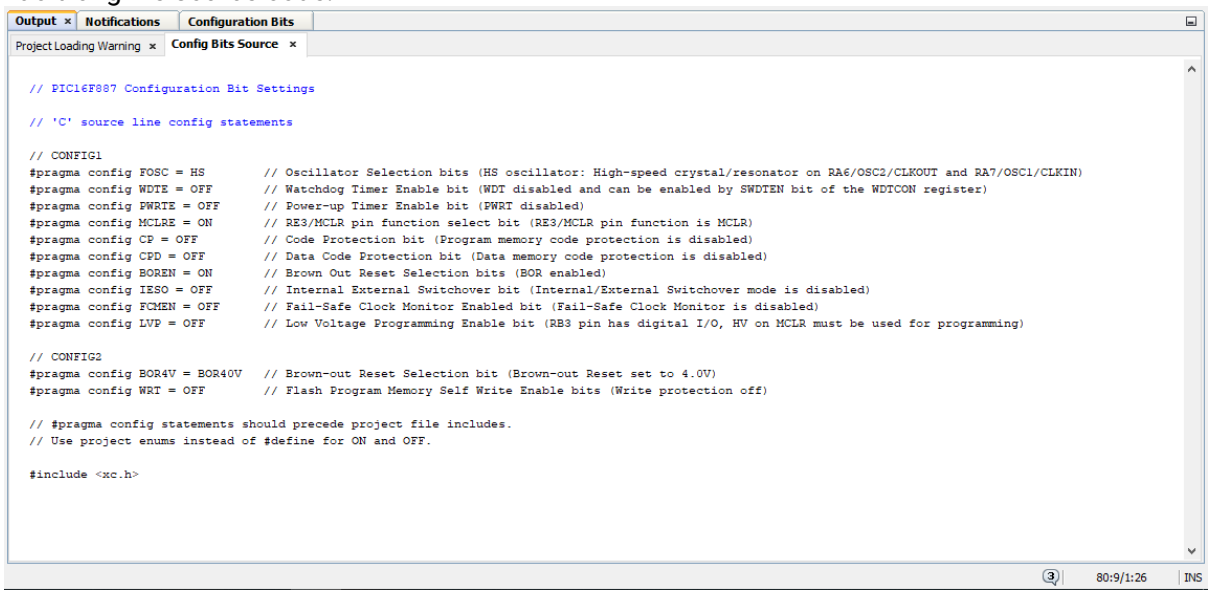


Bước 7: Trước khi tiến hành viết code, mình sẽ cài đặt các thông số của chip bằng cách chọn tab **Production -> Set Configuration Bit,** IDE sẽ hiển thị 1 bảng chọn các thông số ở dưới. Tuy nhiên, vì mới tìm hiểu về PIC nên chúng ta chưa cần quá quan tâm về những thông số này.
Ở cột **Option,** bạn có thể lựa chọn các thông số phù hợp cho chip. Mình chọn các thông số như trên hình:





Sau khi chỉnh các thông số xong, bấm vào **Generate Source Code to Output,** IDE sẽ hiển thị đoạn code tương ứng với các thông số vừa cài đặt. Lúc này bạn có thể copy đoạn code này lại, paste vào trong file source code.

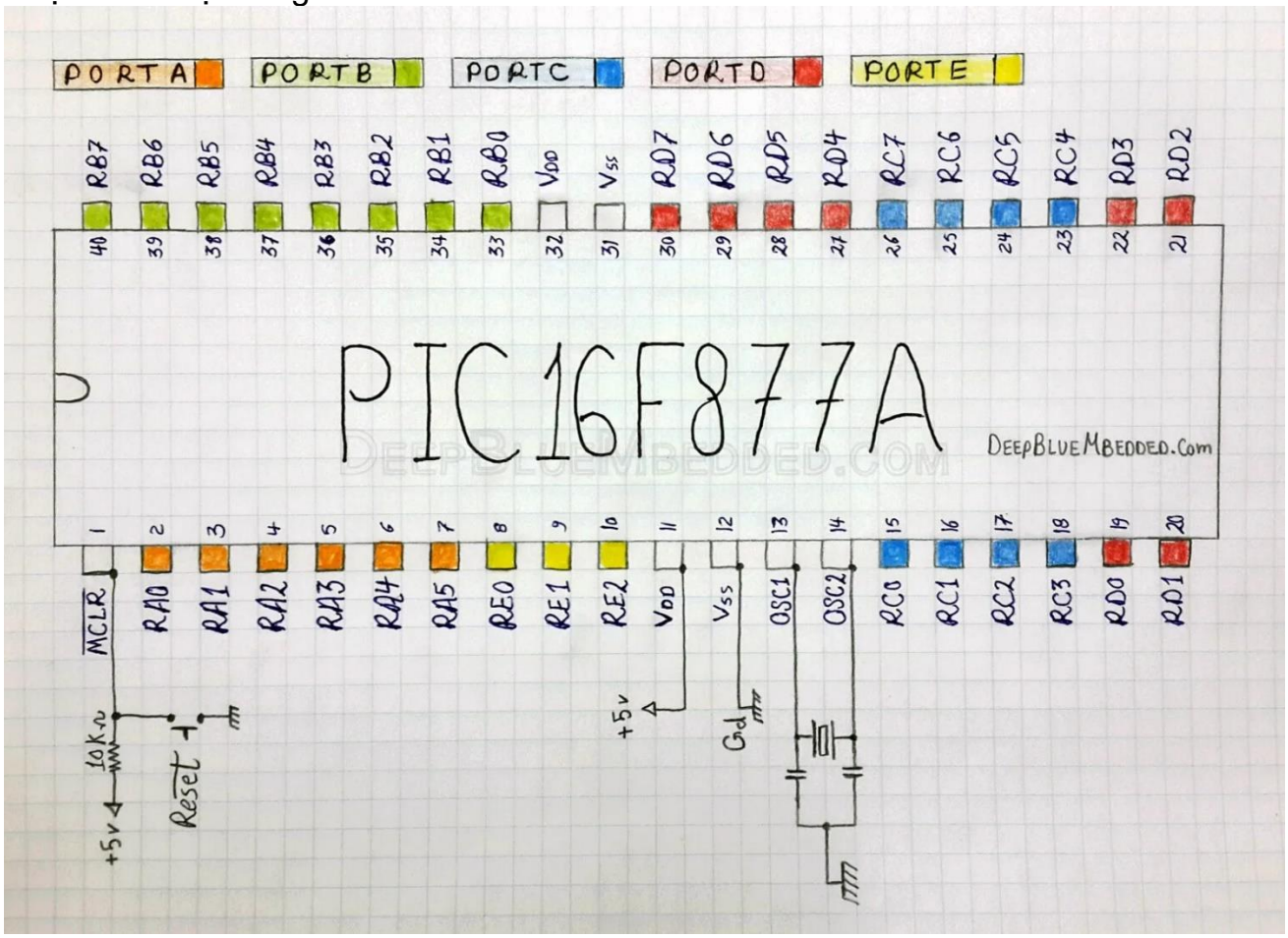


GPIO

GPIO là “**General Purpose Input/Output pins**“ Trên thực tế, hầu hết các chân trong một bộ vi điều khiển điển hình là chân GPIO ngoại trừ một số chân đặc biệt. Các chân đặc biệt (không phải GPIO) thường là những chân sau:

- Chân cung cấp điện: Vdd &Vss
- Chân OSCillator: OSC1 &OSC2. Một vài chân đó được sử dụng để cung cấp cho MCU đầu vào đồng hồ dao động mà nó cần.
- MCLR: Reset_pin . Được sử dụng để khởi động lại MCU có chủ đích.
- V_{USB}: nguồn điện USB bên ngoài.

Ngoại trừ các chân chức năng đặc biệt, tất cả các chân khác đều là GPIO. Như được hiển thị bên dưới từ bảng dữ liệu PIC16F877A, các chân GPIO được nhóm lại trong PORT.



Như bạn có thể nhận thấy rằng các chân [2 đến 7] được đặt tên là [RA0 đến RA5] và đó là lý do tại sao điều này được gọi là "**PORTA**". Trong MCU của chúng tôi có 5 cổng Đầu vào / Đầu ra kỹ thuật số [**A, B, C, D và E**]. Thông thường trong các bộ vi điều khiển 8-Bit có các cổng I / O, mỗi cổng có tối đa 8 chân. Tuy nhiên, một số cổng I / O có thể có ít hơn 8 chân như trong PIC16F877A của chúng tôi để làm ví dụ. Như bạn có thể thấy

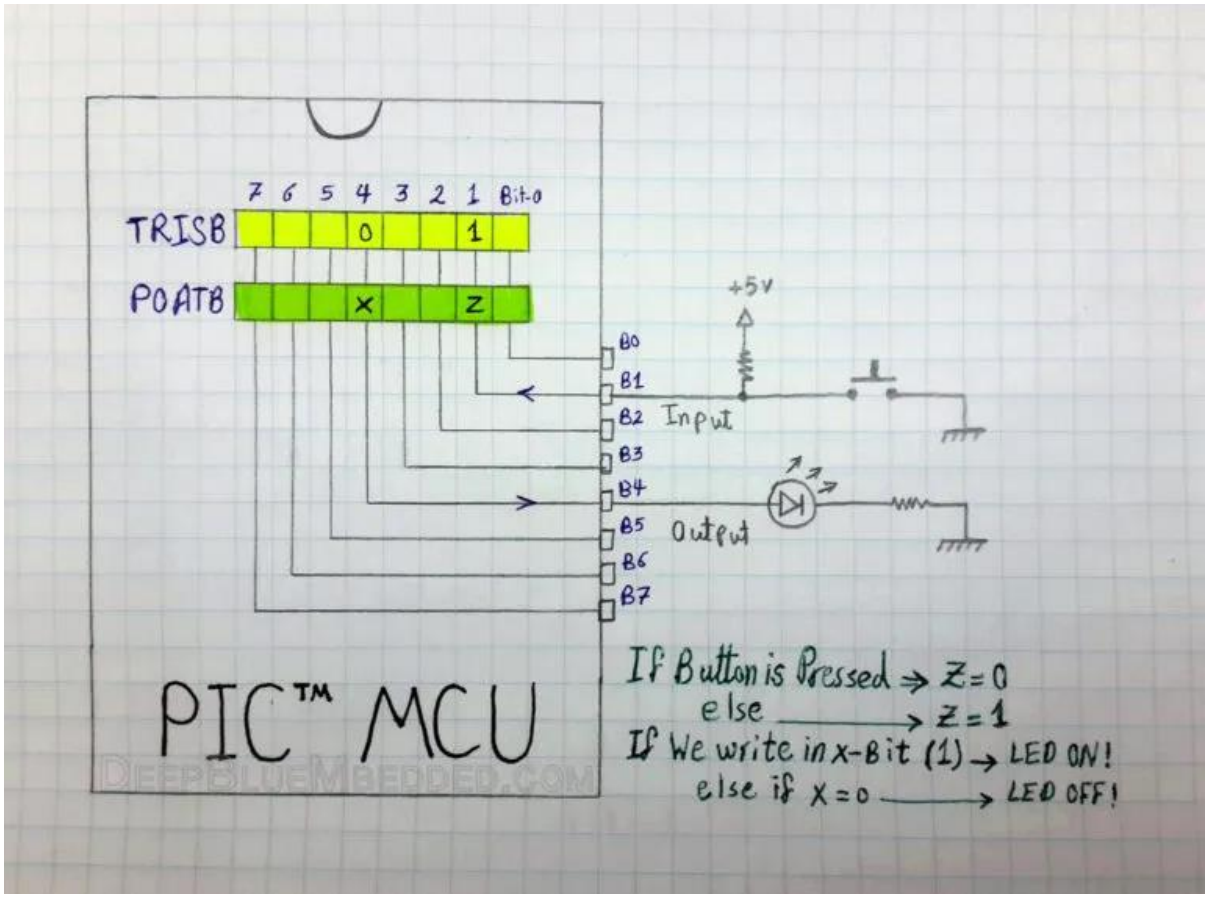
- PORTA: có 6 chân [2 đến 7]
- PORTB: có 8 chân [33 đến 40]
- PORTC: có 8 chân
- PORTD: có 8 chân
- PORTE: có 3 chân [8 đến 10]

Xác định thanh ghi GPIO

Về cơ bản, có một vài thanh ghi dành riêng cho mỗi cổng I / O để kiểm soát hoạt động của nó.

- **Trisx ·**
- **PORTx ·**

Các thanh ghi dành riêng cho các chân PORTA là (TRISA &PORTA) và các thanh ghi dành riêng cho các chân PORTB là (TRISB &PORTB), v.v.



Kiểm soát cổng I / O là một quá trình khá dễ dàng. Nó chỉ bao gồm hai bước liên tiếp.

1. Định cấu hình chân, xác định hướng dữ liệu, cho nó là **Đầu vào** hay **Đầu ra - (Input or Output)**
2. Read / Write . Nếu được định cấu hình để làm đầu vào, thì bạn sẽ Read trạng thái của nó nhiều lần trong chương trình của bạn. Nếu pin được định cấu hình thành chân đầu ra, thì bạn sẽ thúc đẩy trạng thái của nó (High - 5V / Low – 0V) trong chương trình của bạn.

Bất cứ khi nào bạn muốn sử dụng chân I / O, trước tiên bạn phải định cấu hình nó thành chân đầu ra hoặc chân đầu vào. Ở đây, chức năng của thanh ghi TRISx xuất hiện, vì giá trị chúng ta ghi vào thanh ghi đó chỉ định hướng của các chân tương ứng (đầu vào / đầu ra). **X ở** đây là viết tắt của bất kỳ cổng nào [A, B, C, D hoặc E].

- Ghi **1 (Cao)** vào một bit duy nhất của thanh ghi TRISx sẽ cấu hình chân tương ứng với bit này để làm chân **đầu vào (Input)**.
- Ghi **0 (Thấp)** vào một bit duy nhất của thanh ghi TRISx sẽ cấu hình chân tương ứng với bit này để làm chân **đầu ra (Output)**.

Điều đó có nghĩa là việc ghi các giá trị sau vào các bit này sẽ gây ra các hiệu ứng sau

Ví dụ

```
TRISB = 0x00;     // Configures the 8-pins of PORTB [RB0 to RB7] to be output pins
PORTB = 0xFF;     // Drives the 8-pins of PORTB [RB0 to RB7] to be High (logic 1)
```

TRIS**X** = 0x00: thiết lập cả PORT X ở trạng thái Output

TRISX = 0XFF: thiết lập cả PORT X ở trạng thái Input

TRISXbits.TRISXn = 0 : thiết lập chân thứ n của PORT X ở trạng thái Output

TRISXbits.TRISXn = 1 : thiết lập chân thứ n của PORT X ở trạng thái Input

X : là các PORT tương ứng trên PIC16F877A gồm A, B, C, D, E

n : là vị trí chân muốn thiết lập

```
1TRISBbits.TRISB0 = 0;
2TRISBbits.TRISB1 = 0;
3TRISBbits.TRISB2 = 0;
```

hoặc (tương đương)

```
1TRISB0 = 0;
2TRISB1 = 0;
3TRISB2 = 0;
```

Sau khi định cấu hình bất kỳ chân i / o nào, bạn sẽ có thể đọc trạng thái của nó (nếu đó là chân đầu vào) hoặc thay đổi trạng thái logic của nó (nếu đó là chân đầu ra). Ở đây, chức năng của thanh ghi **PORTx** xuất hiện, vì giá trị chúng ta ghi vào thanh ghi đó chỉ định trạng thái logic của các chân tương ứng (Cao 1 / Thấp 0).

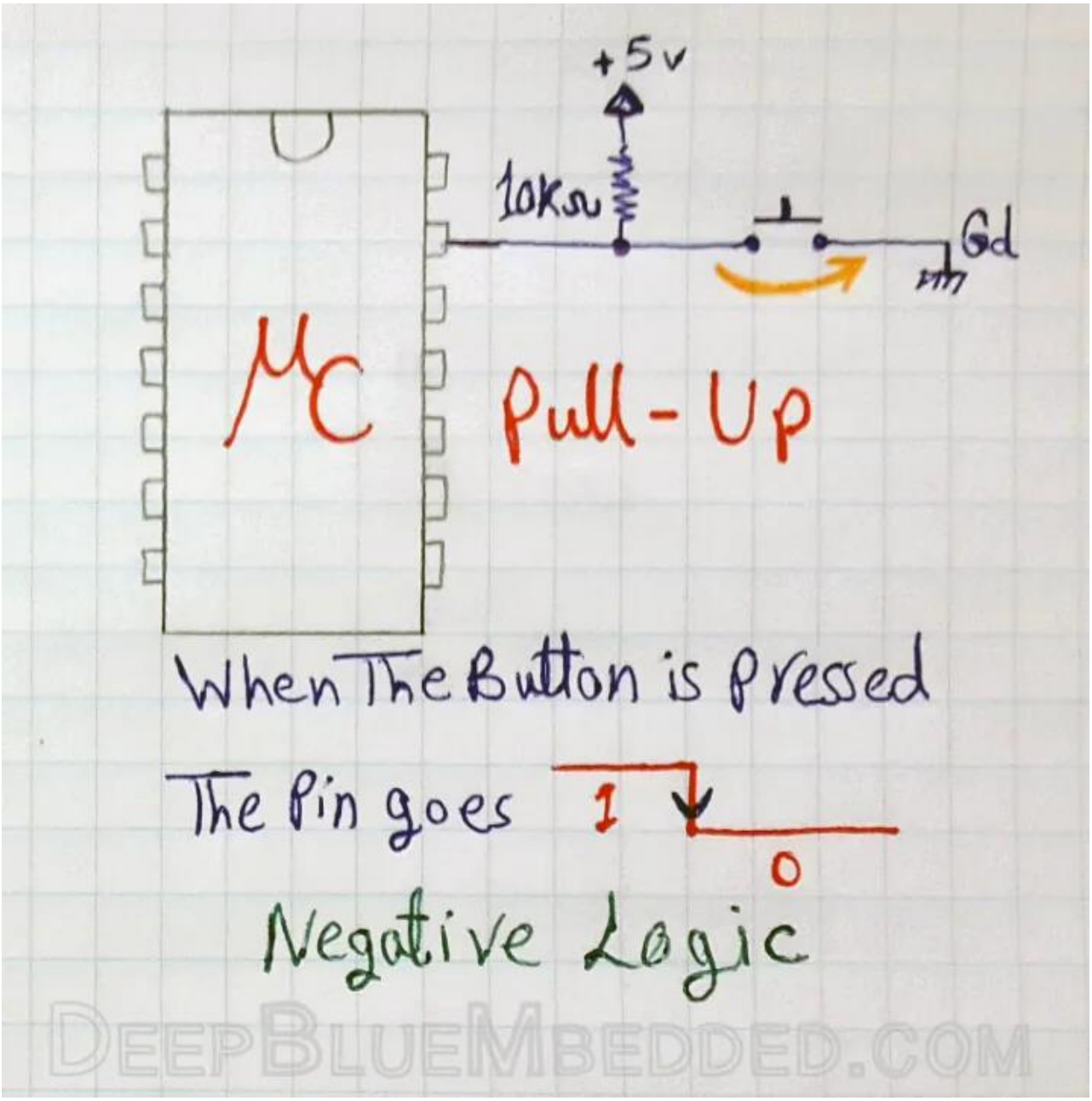
- Viết **1 (Cao)** vào một bit duy nhất của thanh ghi PORTx đặt chân tương ứng thành **Cao**.
- Viết **0 (Thấp)** vào một bit duy nhất của thanh ghi PORTx đặt chân tương ứng thành **Thấp**.

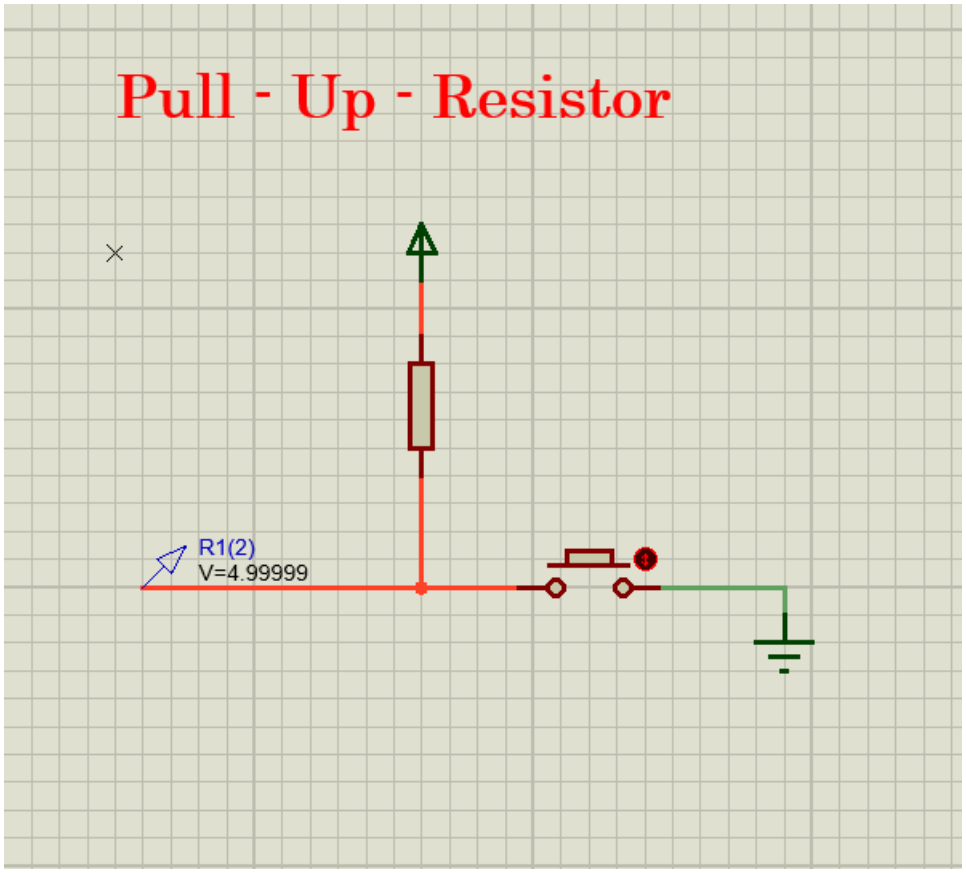
PULL UP & PULL DOWN RESISTOR

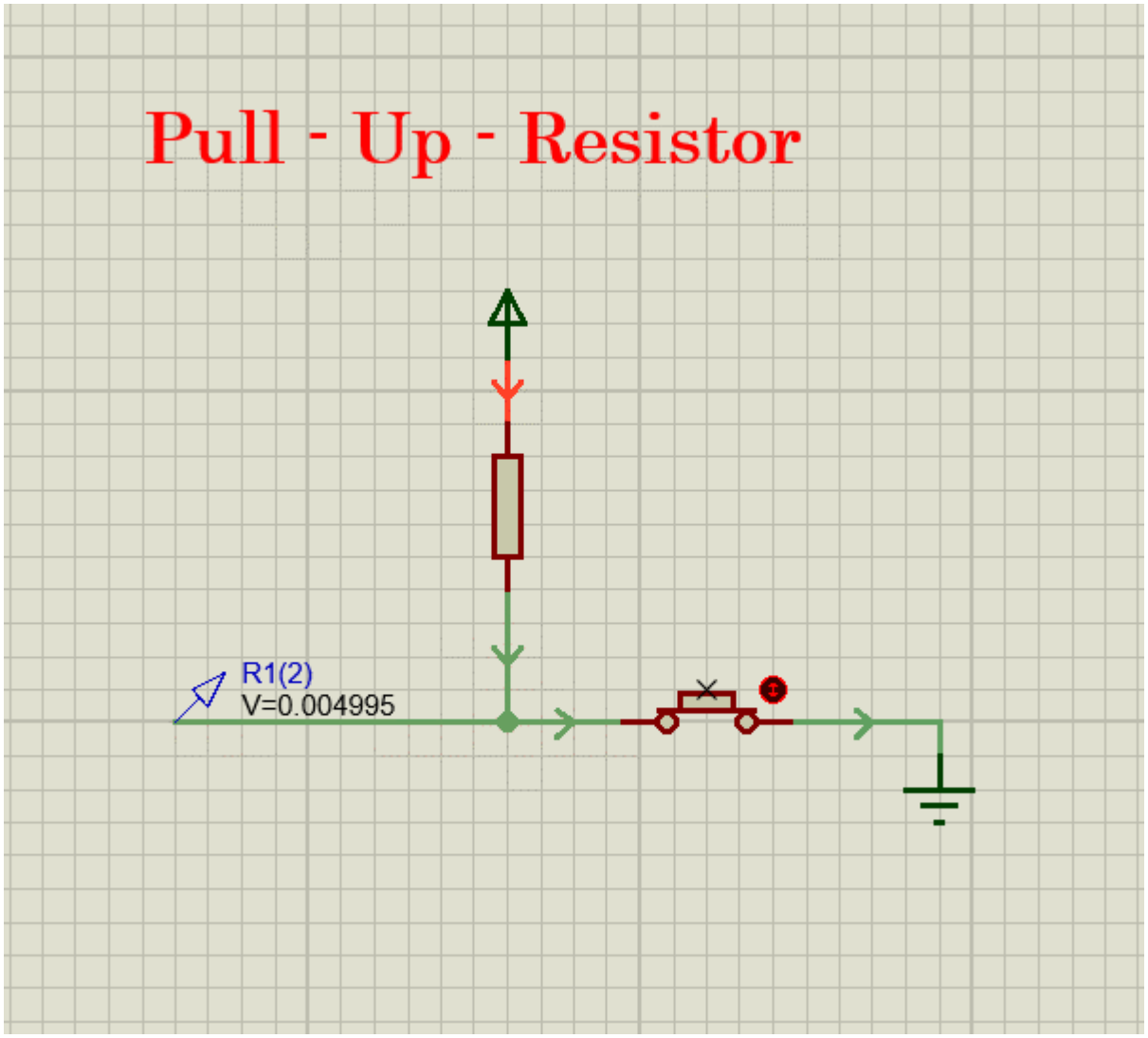
(Trở kéo lên – Trở kéo xuống)

- **Chân đầu vào Pull-Up**
  Trong cấu hình này, chân luôn được kéo lên Cao (logic 1) cho đến khi một sự kiện xảy ra để lái nó Xuống thấp (đến 0). Hầu hết tất cả những gì bạn cần có là một điện trở 10k ohm được kết nối giữa chân đầu vào và Vdd (+ 5v) như hình dưới đây.

  Trạng thái logic của chốt luôn là **1** cho đến khi nhấn nút, sau đó nó bị đoản mạch với mặt đất và trở thành **0**. Đây được gọi là đầu vào **Logic phủ định**, vì hành động của người dùng (nhấn nút) chuyển trạng thái pin kỹ thuật số từ Cao sang **Thấp**.

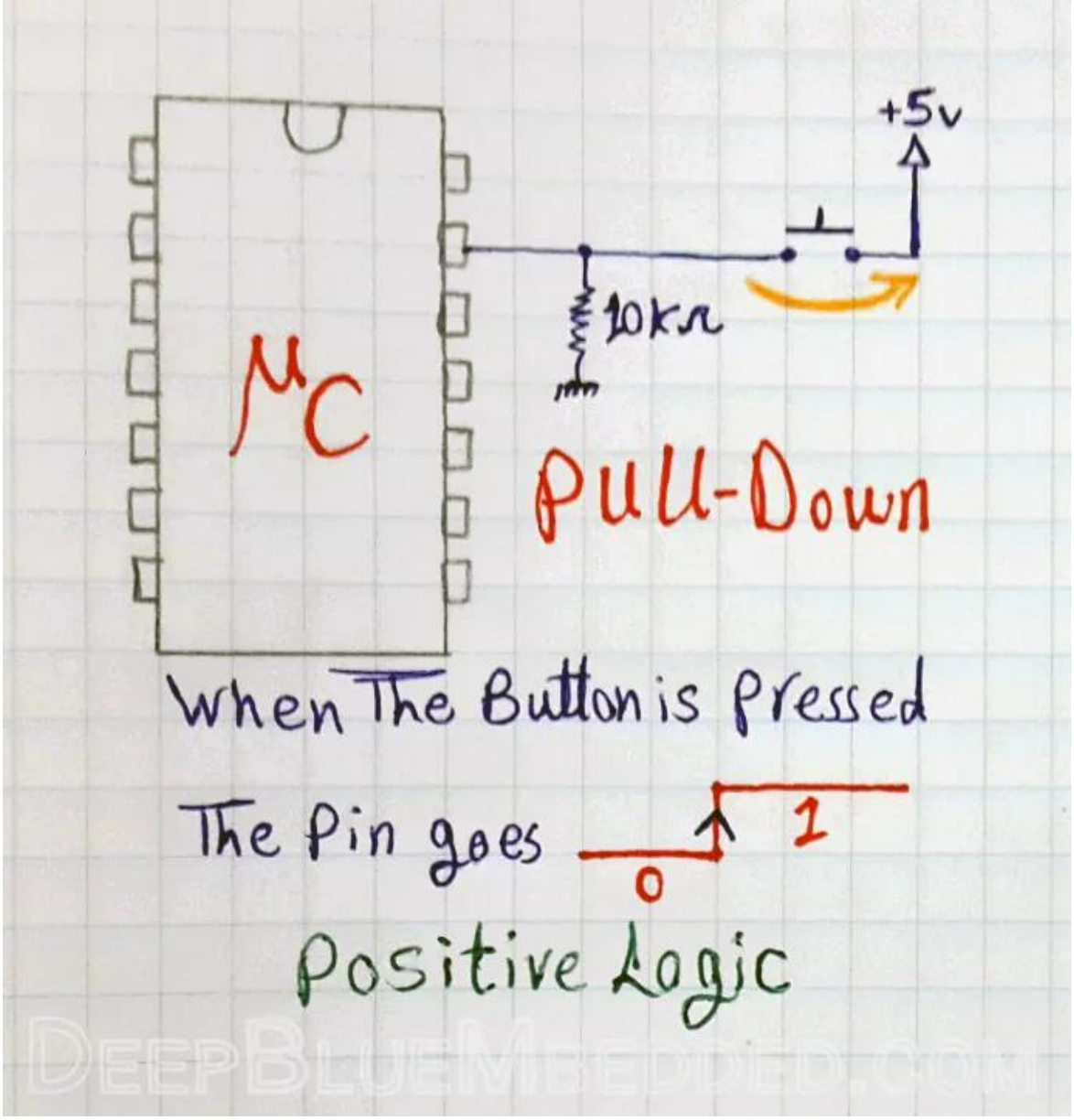

+5v
10KΩ
Gd
μC
Pull-Up
When The Button is Pressed
The Pin goes
1
0
Negative Logic
DEEPBLUEMBEDDED.COM

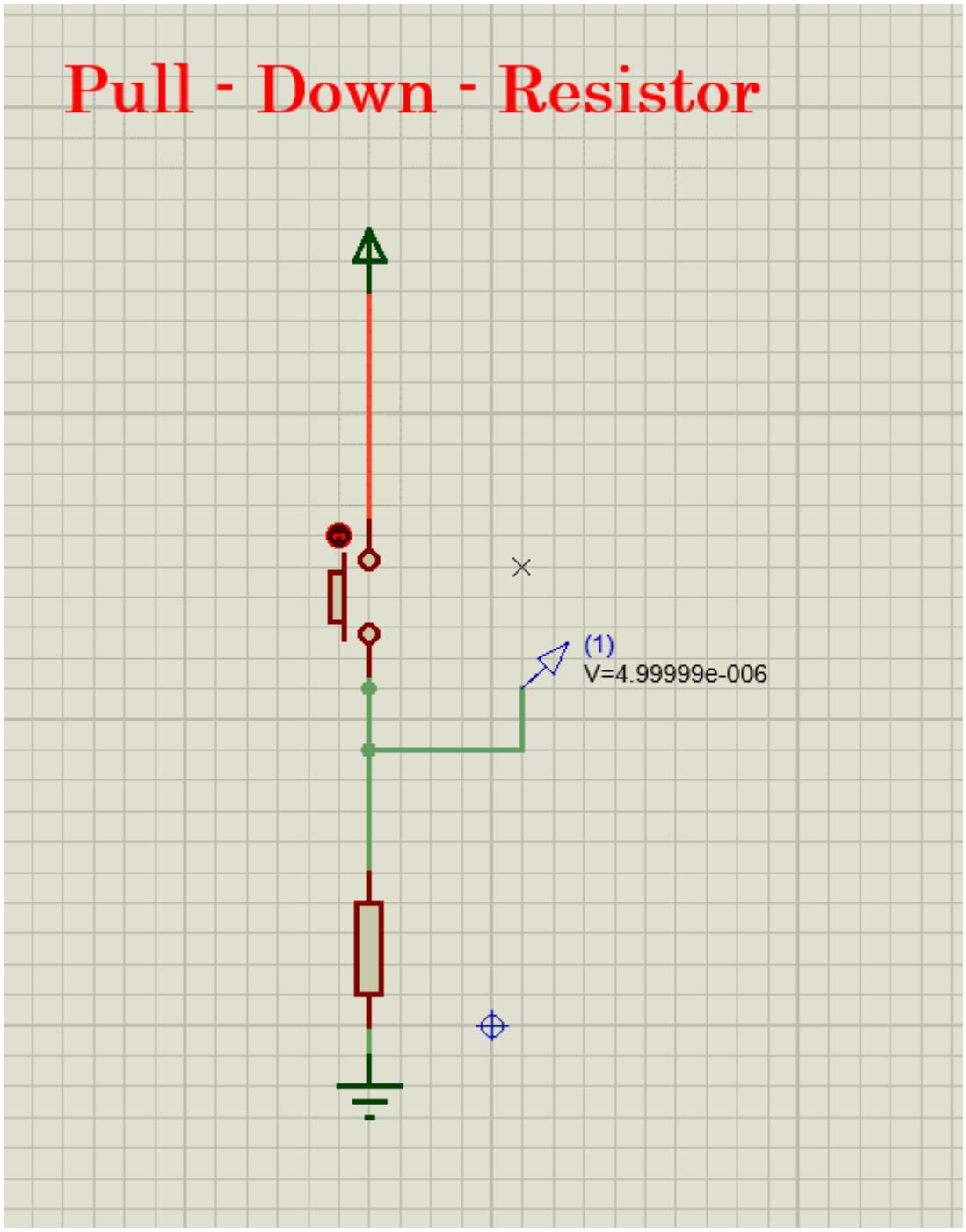

Pull - Up - Resistor
R1(2)
V=4.99999

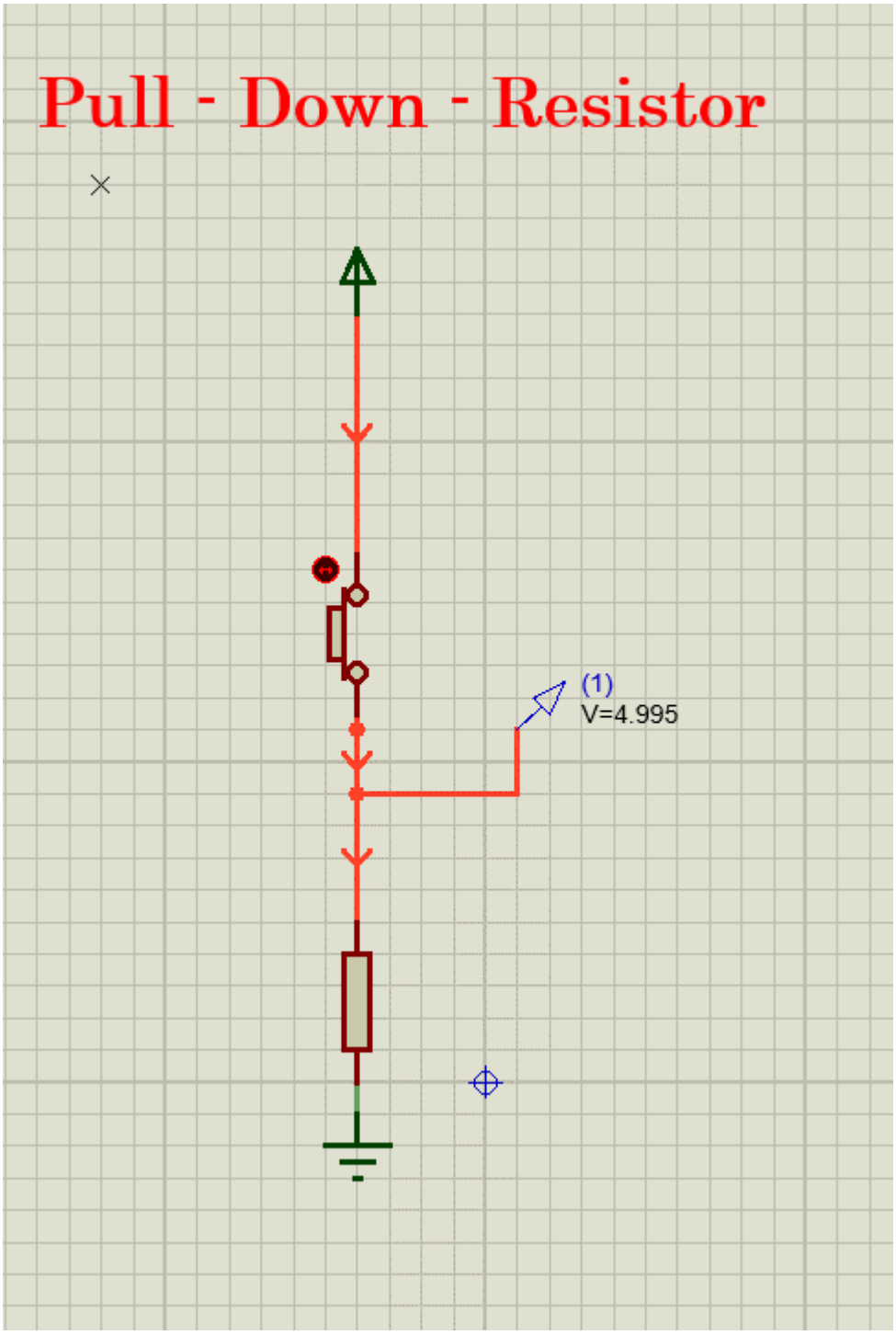

Pull - Up - Resistor
R1(2)
V=0.004995

- **Chân đầu vào Kéo xuống**
  Trong cấu hình này, chân luôn được kéo xuống Thấp (logic 0) cho đến khi một sự kiện xảy ra để lái nó Lên cao (đến 1). Hầu hết tất cả những gì bạn cần có là một điện trở 10k ohm được kết nối giữa chân đầu vào và Vss (0v hoặc Gd) như hình dưới đây.

  Trạng thái logic của chân luôn là **0** cho đến khi nhấn nút, sau đó nó bị đoản mạch với Vdd (+ 5v) và trở thành **1**. Đây được gọi là đầu vào **Logic tích cực**, vì hành động của người dùng (nhấn nút) chuyển trạng thái chân kỹ thuật số từ Thấp sang **Cao**.

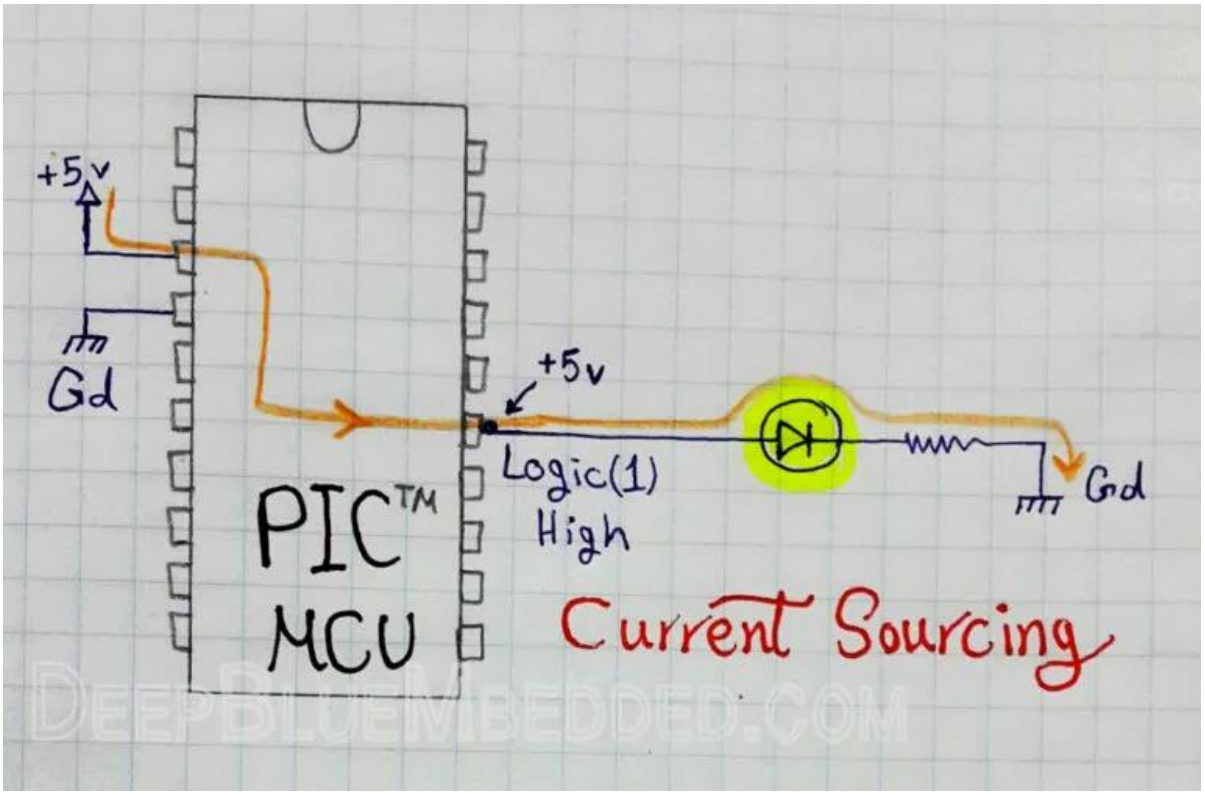



Pull - Down - Resistor
(1)
V=4.99999e-006


Pull - Down - Resistor
(1)
V=4.995

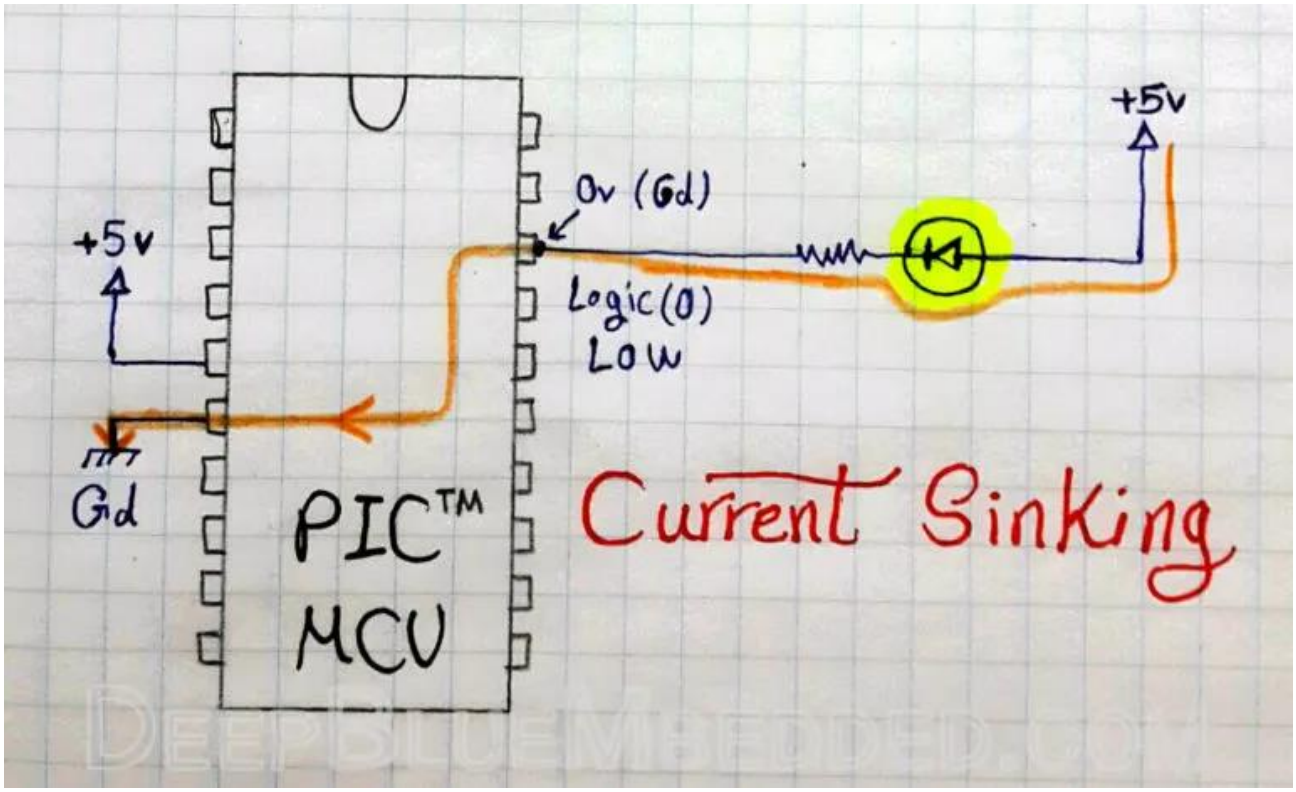
Ghi vào thanh ghi PORTx sẽ thay đổi trạng thái logic của chân i / o (chỉ khi chúng được định cấu hình làm chân đầu ra). Trạng thái logic của các chân đầu ra 0, 1 tương ứng với 0v, + 5v tương ứng. Điều đó có nghĩa là một chân đầu ra điển hình có thể đang tìm nguồn cung ứng dòng điện cho các thiết bị khác hoặc thay vào đó nó có thể là dòng điện chìm.

- **Nguồn cung ứng hiện tại của chân đầu ra** Trong cấu hình này, chân đầu ra được sử dụng làm nguồn (+ 5v) cung cấp dòng điện để điều khiển tải rất nhỏ (ví dụ: đèn LED, bóng bán dẫn, rơle, Optocouplers,v.v.).
  Một chân điển hình trong bộ vi điều khiển của chúng tôi có thể tạo nguồn lên đến 25 mA cho mỗi chân i / o và giá trị này thay đổi từ chip MCU sang chip MCU khác. Đây là cấu hình được sử dụng phổ biến nhất và nó phải rõ ràng trong sơ đồ dưới đây.



Chân đầu ra Chìm

hiện tại Trong cấu hình này, chân đầu ra được sử dụng như một (0v hoặc Mặt đất) để chìm dòng điện từ các tải nhỏ (ví dụ: đèn LED, Bóng bán dẫn, Rơle, Optocouplers, v.v.). Một chân điển hình trong bộ vi điều khiển của chúng tôi có thể chìm tới 25 mA cho mỗi chân i / o. Cấu hình này được hiển thị trong sơ đồ dưới đây.


+5v
0v (Gd)
+5v
Logic(0)
Low
Gd
PIC™
MCU
Current Sinking

***NOTE**

Bạn viết 1 con số ra giấy, ví dụ 12345678 đây là một số hệ 10 (decimal).
Trong con số kia, số 8 là hàng đơn vị và nó có khả năng ảnh hưởng đến giá trị số vừa viết là nhỏ nhất; con số 1 thì là số có ảnh hưởng lớn nhất.

Bây giờ quay lại MCU, tất cả số liệu điều tính bằng byte, bit trong đó 1 byte = 8 bit.

Bây giờ mình có số 129 thể hiện dạng bit là 10000001.
Trong 8 bit đó, bit 1 bên phải có tầm ảnh hưởng nhỏ nhất (như số 8 ví dụ trên) gọi là LSB ~ bit thấp nhất. Bit 1 bên trái mang tầm ảnh hưởng cao nhất gọi là MSB ~ bit cao nhất. Còn ở giữa sẽ là các bit cao nhì, ba, bốn hoặc thấp nhì, ba …

int1	Số 1 bit = T hoặc F	float	Số thực 32 bit
int8	Số nguyên 1 byte	short	Mặc định kiểu int1
int16	Số nguyên 16 bit	byte	Mặc định kiểu int8
int32	Số nguyên 32 bit	int	Mặc định kiểu int8
char	Ký tự 8 bit	Long	Mặc định kiểu int16

- Có thể them signed hoặc unsigned để xác định số có dấu hay không
- Đối với PIC16F877A thì chỉ số mảng có kích thước tối đa là 256 byte

Các tiền tố

#include

Thêm các thư viện vào chương trình

#include<PIC16F877A.h>
#include “PIC16F877A.h”

Thư viện định nghĩa địa chỉ các cổng và thanh ghi

***Bắt buộc**

Phải có

- **#include <pic16f877a.h>**
- **#define _XTAL_FREQ X**

Với X là tần số thạch anh mình sử dụng

Câu lệnh “#define _XTAL_FREQ 8000000” xác định tần số xung nhịp của vi điều khiển (8MHz) được sử dụng để tính toán độ trễ trong hàm __delay_ms ()

- Câu lệnh “#include <xc.h>” để khai báo thư viện xc.h chứa định nghĩa của hàm __delay_ms(), các thanh ghi TRIS, PORT.

Chỉ thị tiền xử lí (macro - #define)

```
#define  LED  RB0
```

Thay thế tên LED cho RB0

Hàm delay

Lỗi hay sai :

Dùng 1 dấu _ thay vì 2 dấu __

Template

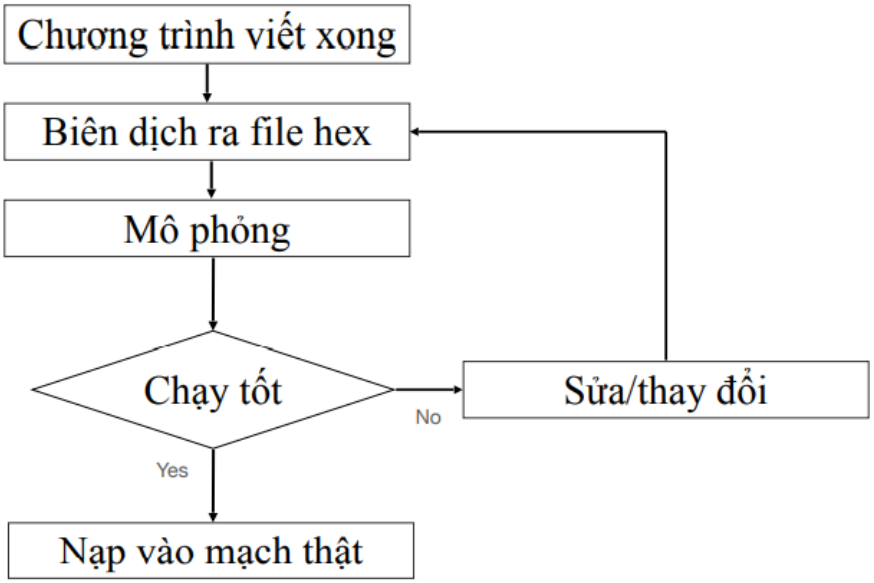
__delay_ms(X); // delay X mili s
__delay_us(X); // delay X micro s

Nếu có Watch Dog Timer thì
__delaywdt_ms(X);
__delaywdt_us(X);

Nếu muốn delay 1s

__delay_ms(1000);

Viết chương trình xong thì làm gì ?



Những linh kiện cần thiết trong Proteus.

1.**Điện trở** : res
2. **Biến trở**: pot hoặc pot-hg
3. **Tụ điện**: cap
4. **Tụ hóa**: cap-elec hoặc cap-pol
5. **Cuộn cảm**: inductor
6. **Diode**: diode hoặc tên diode, ví dụ 1N4007, 1N4148...
7. **Cầu diode**: bridge
8. **LED**: led
9. **Led 7 đoạn**: 7seg
10. **LED ma trận**: matrix
11. **Đèn**: lamp
12. **LCD 16x2** : LM016L
13. **Quang trở**: ldr
14. **Transistor**: gõ tên của transistor (ví dụ: 2N2222) hoặc dùng NPN, PNP, ...
15. **IC**: nhập tên của IC, ví dụ: 555, 7805, 7490, ...
16. **Loa**: Speaker hoặc Sounder
17. **Công tắc**: sw- hoặc switch
18. **Nút nhấn**: button
19. **Bàn phím**: keypad-
20. **Rơ-le**: relay
21. **Các cổng logic**: gõ tên cổng cần tìm, ví dụ AND, OR, NOT...
22. **Thạch anh**: crystal
23. **Đồng hồ đếm giờ** : 7SEG-MPX2-
24. **Điện trở thanh** : RESPACK
25. **Biến áp**: tran
26. **Nguồn AC**: vsin
27. **Nguồn DC**: battery hoặc cell
28. **Jack cắm 1 hàng các loại** : conn-sil
29. **Jack cắm 2 hàng các loại** : conn-dil
30. **Công cụ lật trạng thái 0/1**: Logic toggle
31. **Công cụ đo mức logic**: logic probe (big)

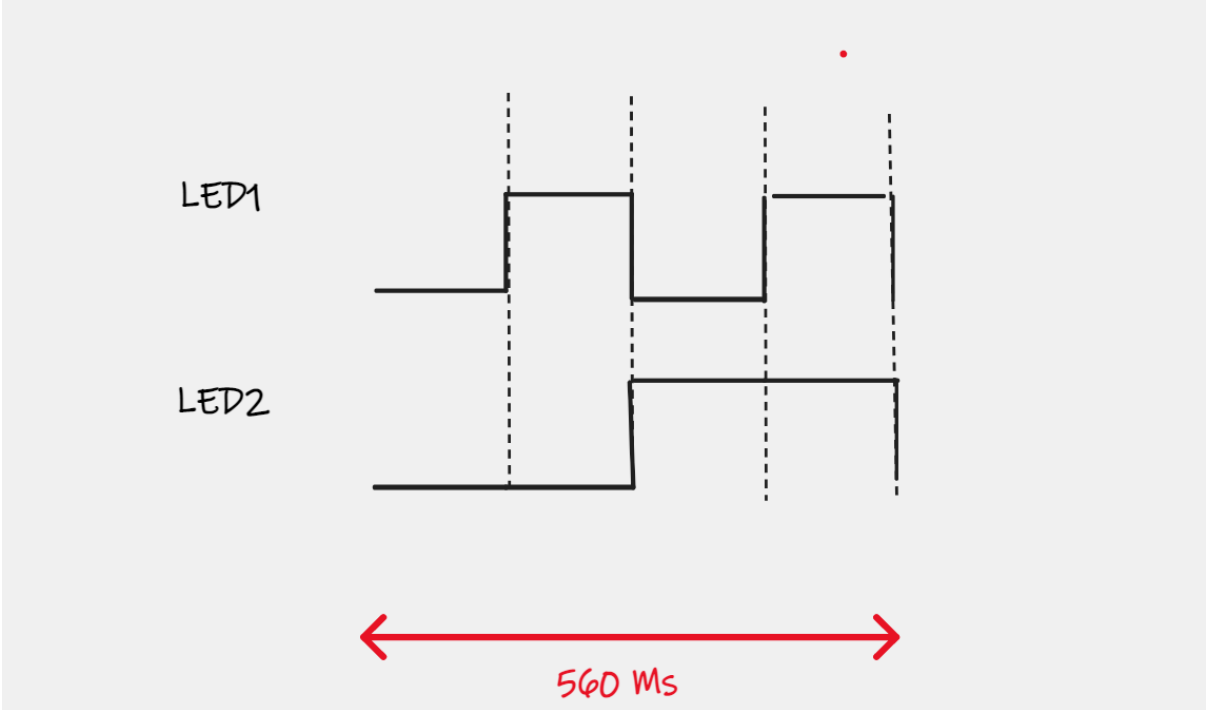
Lưu ý: Các linh kiện thuộc thư viện **Active** thì có thể thay đổi và hoạt động trong lúc mô phỏng được, còn các linh kiện khác chỉ có thể thay đổi thông số khi chưa mô phỏng.

Bài ví dụ code.

Ví Dụ 1. Sáng 8 led ở cổng B với chu kì 500ms.

```
void main()
{
// Thiet lap che do cho PORTB
TRISB = 0x00; //  0b0000 0000 Tat ca PORTB deu la cong xuat du lieu
PORTB = 0x00; // Tat het cac LED
While(1)
{
PORTB = 0xff; //0b11111111 Cho các LED sáng
delay_ms(250); // Tạo thời gian trễ 250ms
PORTB = 0x00;
delay_ms(250);
}
```

Ví Dụ 2. Sáng 2 led ở cổng RB (RB0 vs RB1) với bảng sơ đồ như sau.



```
// PIC16F877A Configuration Bit Settings
// 'C' source line config statements
// CONFIG
#pragma config FOSC = HS        // Oscillator Selection bits (HS oscillator)
#pragma config WDTE = ON        // Watchdog Timer Enable bit (WDT enabled)
#pragma config PWRTE = OFF      // Power-up Timer Enable bit (PWRT disabled)
#pragma config BOREN = ON       // Brown-out Reset Enable bit (BOR enabled)
#pragma config LVP = ON         // Low-Voltage (Single-Supply) In-Circuit
Serial Programming Enable bit (RB3/PGM pin has PGM function; low-voltage
programming enabled)
#pragma config CPD = OFF        // Data EEPROM Memory Code Protection bit
(Data EEPROM code protection off)
#pragma config WRT = OFF        // Flash Program Memory Write Enable bits
(Write protection off; all program memory may be written to by EECON control)
#pragma config CP = OFF         // Flash Program Memory Code Protection bit
(Code protection off)
// #pragma config statements should precede project file includes.
// Use project enums instead of #define for ON and OFF.
#include <xc.h>
#include <pic16f877a.h>
#define _XTAL_FREQ 20000000
void main(void) {
    TRISB = 0x00;// TRISbits.RB0 = 0;...
    while(1)
    {
        PORTBbits.RB0 = 0;
        PORTBbits.RB1 = 0;
        __delay_ms(140);
    //-----------------------
        PORTBbits.RB0 = 1;
        PORTBbits.RB1 = 0;
           __delay_ms(140);
      //---------
        PORTBbits.RB0 = 0;
        PORTBbits.RB1 = 1;
           __delay_ms(140);
     //----------------------
        PORTBbits.RB0 = 1;
        PORTBbits.RB1 = 1;
           __delay_ms(140);
    }

}
```

Bài tập.

Bài 1. Thiết kế sơ đồ ghép nối 4 LED đơn với vi điều kiển PIC16F877A tại chân bất kì tại cổng RB. Viết chương trình thực hiện.

Nháy sáng lần lượt các led với tần số 0.5Hz (Led 1 sáng tắt trong 0.5Hz tương tự với led 2,3,4). Hết vòng 3 led thì nháy theo chiều ngược lại. Quá trình thực hiện vô tận.

Bài 2. Thiết kế sơ đồ ghép nối 8 LED đơn ghép thành 1 hàng với vi điều kiển PIC16F877A tại chân bất kì tại cổng bất kì. Viết chương trình thực hiện.

Bật sáng lần lượt các led từ 2 đầu hàng tiễn vào giữa lần lượt cách nhau 0.5s sau đó cá led lại tắt lần lượt từ trái qua phải cách nhau 1s.

Bài 3. Thiết kế sơ đồ ghép nối 3 LED đơn với vi điều kiển PIC16F877A tại chân bất kì tại cổng RE. Viết chương trình thực hiện.

Sáng lần lượt led từ led1 tới led 3 với chu kì 0.1s. Sau đó nhấp nháy 2 lần 3 led với chu kì 1s. Lặp lại quá trình trên.