BÀI 1: NHẬP MÔN VI ĐIỀU KHIỂN PIC

I. PHẦN LÝ THUYẾT

1. Giới thiệu về vi điều khiển PIC

1.1 Giới thiệu về vi điều khiển

Bộ vi điều khiển viết tắt là Micro-controller là mạch tích hợp trên một chip có thể lập trình được, dùng để điều khiển hoạt động của hệ thống .Theo các tập lệnh của người lập trình, bộ vi điều khiển tiến hành đọc, lưu trữ thông tin, xử lý thông tin, đo thời gian và tiến hành đóng mở một cơ cấu nào đó .

Trong các thiết bị điện và điện tử, các bộ vi điều khiển điều khiển hoạt động của ti vi, máy giặt, đầu đọc lase, lò vi ba, điện thoại …Trong hệ thống sản xuất tự động, bộ vi điều khiển sử dụng trong robot, các hệ thống đo lường giám sát .Các hệ thống càng thông minh thì vai trò của vi điều khiển ngày càng quan trọng. Hiện nay trên thị trường có rất nhiều họ vi điều khiển như: 6811 của Motorola, 8051 của Intel, Z8 của Zilog, PIC của Microchip Technology .

1.2 Giới thiệu về vi điều khiển PIC

PIC bắt nguồn từ chữ viết tắt của “Programmable Intelligent Computer” (Máy tính khả trình thông minh) là sản phẩm của hãng General Instrument đặt cho dòng sản phẩm đầu tiên của họ là PIC 1650. Lúc này Pic dùng để giao tiếp với các thiết bị ngoại vi cho máy chủ 16 bit CP1600, vì vậy người ta gọi PIC với tên là “ Peripheral Interface Controller” (bộ điều khiển giao tiếp ngoại vi) .

Năm 1985 General Instrument bán bộ phận vi điện tử của họ, và chủ sở hữu mới (Microchip Technology) huỷ bỏ hầu hết các dự án – lúc đó đã quá lỗi thời. Tuy nhiên PIC được bổ sung EEPROM để tạo thành một bộ điều khiển vào ra khả trình. Ngày nay có rất nhiều dòng PIC được sản xuất với hàng loạt các modul ngoại vi được tích hợp sẵn (như :USART, PWM, ADC…) với bộ nhớ chương trình từ 512 word đến 32k word.

PIC sử dụng tập lệnh RISC, với dòng PIC low-end (độ dài mã lệnh 12 Bit ví dụ PIC12Cxxx) và mid-range (độ dài mã lệnh 14 bit , ví dụ PIC16Fxxx), tập lệnh bao gồm khoảng 35 lệnh, và 70 lệnh đối với dòng PIC high-end(có độ dài mã lệnh 16bit PIC18Fxxxx). Tập lệnh bao gồm các lệnh tính toán trên các thanh ghi, và các hằng số, hoặc các vị trí ô nhớ, cũng như có các lệnh điều kiện, nhảy/ gọi hàm, và các lệnh quay trở về, nó cũng có các chức năng phần cứng khác như ngắt hoặc sleep(chế độ hoạt động tiết kiệm điện). Microchip cung cấp môi trường lập trình MPLAB0, nó bao gồm phần mềm mô phỏng và trình dịch ASM

Hiện nay có khá nhiều dòng PIC và có rất nhiều khác biệt về phần cứng, nhưng chúng ta có thể điểm qua một vài nét như sau :

- 8/16/24/32 bit CPU, xây dựng theo kiến trúc Harvard
- Flash và Rom có thể tuỳ chọn 256 byte đến 256 kbybe
- Bộ nhớ nội EEPROM - có thể ghi/ xoá lên tới hàng triệu lần
- Các cổng xuất/nhập (mức lôgic thường từ 0v đến 5v, ứng với mức logic 0 và 1, dòng khoảng vài chục mA)
- 8/16 bit timer
- Modun giao tiếp ngoại vi nối tiếp không đồng bộ: USART
- Modun giao tiếp ngoại vi song song (kiểu máy in)
- Bộ chuyển đổi ADC 10 bit nội gồm 8 kênh đầu vào
- Module ngoại vi MSSP dùng cho các giao tiếp I2C, SPI
- Modul CCP có chức năng
    - Comparator (so sánh)
    - Capture
    - PWM: dùng trong điều khiển động cơ

Một số dòng vi điều khiển PIC hỗ trợ thêm:

- Hỗ trợ điều khiển động cơ 3 pha, 1 pha
- Hỗ trợ giao tiếp USB
- Hỗ trợ điều khiển Ethernet
- Hỗ trợ giao tiếp CAN

- Hỗ trợ giao tiếp LIN
- Hỗ trợ giao tiếp IRDA
- DSP những tính năng xử lý tín hiệu số

1.3 Các thành phần cơ bản của một mạch ứng dụng vi điều khiển PIC:

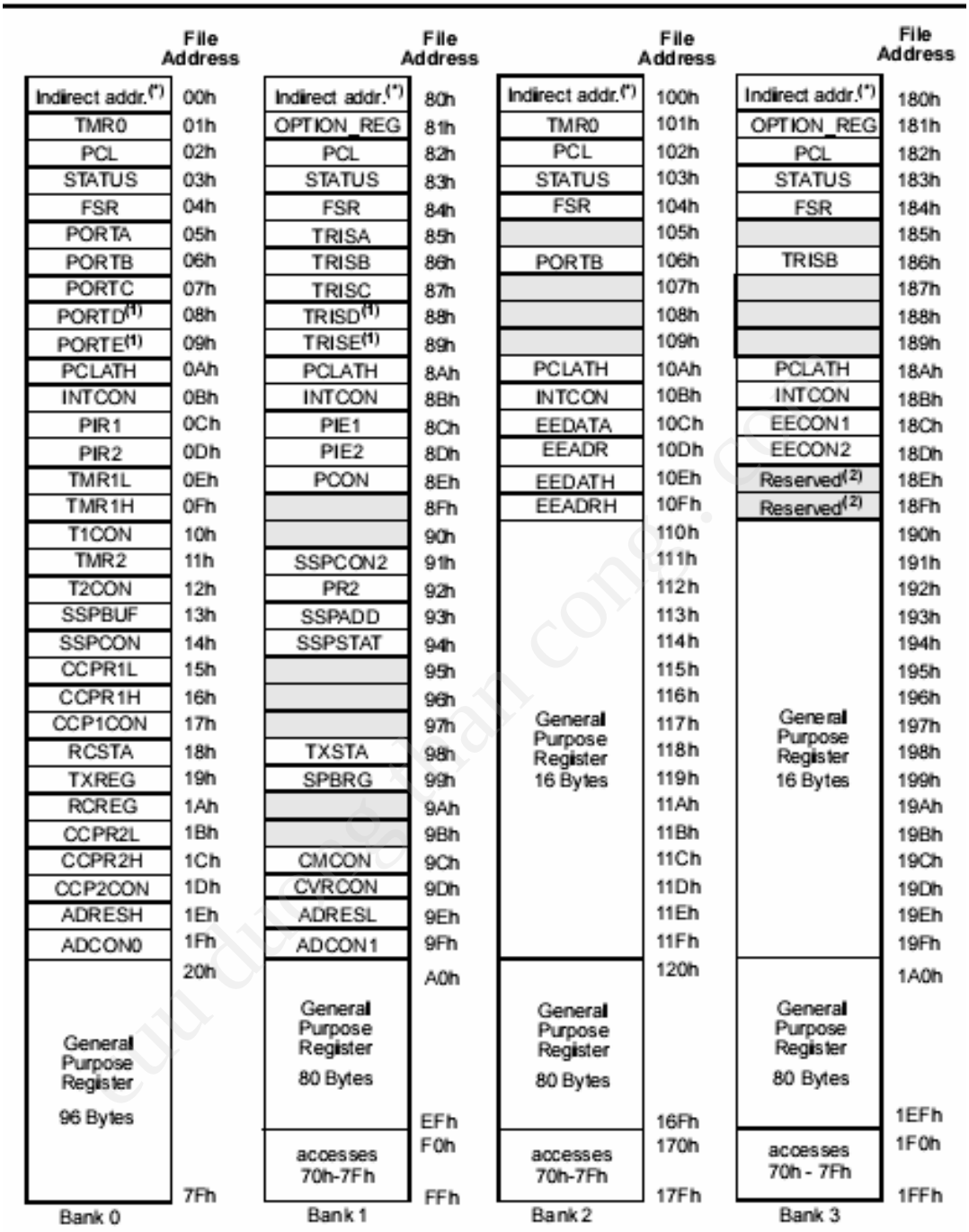
1.4 Các khái niệm cơ bản:

Cổng xuất nhập I/O:

Trong vi điều khiển PIC16F877A có 5 cổng:

- Cổng A gồm 6 chân: RA0, RA1.. RA5
- Cổng B gồm 8 chân: RB0, RB1,..RB7
- Cổng C gồm 8 chân: RC0, RC1, ..RC7
- Cổng D gồm 8 chân: RD0, RD1,..RD7
- Cổng E gồm 3 chân: RE0, RE1, RE2

Mỗi cổng thực chất được quản lý bởi các thanh ghi PORTA, PORTB, PORTC, PORTD, PORTE nằm trong bộ nhớ RAM của vi điều khiển. Xem hình sau:



File Address	Bank 0	File Address	Bank 1	File Address	Bank 2	File Address	Bank 3
00h	Indirect addr.(*)	80h	Indirect addr.(*)	100h	Indirect addr.(*)	180h	Indirect addr.(*)
01h	TMR0	81h	OPTION_REG	101h	TMR0	181h	OPTION_REG
02h	PCL	82h	PCL	102h	PCL	182h	PCL
03h	STATUS	83h	STATUS	103h	STATUS	183h	STATUS
04h	FSR	84h	FSR	104h	FSR	184h	FSR
05h	PORTA	85h	TRISA	105h		185h	
06h	PORTB	86h	TRISB	106h	PORTB	186h	TRISB
07h	PORTC	87h	TRISC	107h		187h	
08h	PORTD(1)	88h	TRISD(1)	108h		188h	
09h	PORTE(1)	89h	TRISE(1)	109h		189h	
0Ah	PCLATH	8Ah	PCLATH	10Ah	PCLATH	18Ah	PCLATH
0Bh	INTCON	8Bh	INTCON	10Bh	INTCON	18Bh	INTCON
0Ch	PIR1	8Ch	PIE1	10Ch	EEDATA	18Ch	EECON1
0Dh	PIR2	8Dh	PIE2	10Dh	EEADR	18Dh	EECON2
0Eh	TMR1L	8Eh	PCON	10Eh	EEDATH	18Eh	Reserved(2)
0Fh	TMR1H	8Fh		10Fh	EEADRH	18Fh	Reserved(2)
10h	T1CON	90h		110h		190h	
11h	TMR2	91h	SSPCON2	111h		191h	
12h	T2CON	92h	PR2	112h		192h	
13h	SSPBUF	93h	SSPADD	113h		193h	
14h	SSPCON	94h	SSPSTAT	114h		194h	
15h	CCPR1L	95h		115h		195h	
16h	CCPR1H	96h		116h		196h	
17h	CCP1CON	97h		117h	General Purpose Register 16 Bytes	197h	General Purpose Register 16 Bytes
18h	RCSTA	98h	TXSTA	118h		198h	
19h	TXREG	99h	SPBRG	119h		199h	
1Ah	RCREG	9Ah		11Ah		19Ah	
1Bh	CCPR2L	9Bh		11Bh		19Bh	
1Ch	CCPR2H	9Ch	CMCON	11Ch		19Ch	
1Dh	CCP2CON	9Dh	CVRCON	11Dh		19Dh	
1Eh	ADRESH	9Eh	ADRESL	11Eh		19Eh	
1Fh	ADCON0	9Fh	ADCON1	11Fh		19Fh	
20h	General Purpose Register 96 Bytes	A0h	General Purpose Register 80 Bytes	120h	General Purpose Register 80 Bytes	1A0h	General Purpose Register 80 Bytes
7Fh		EFh	accesses 70h-7Fh	16Fh	accesses 70h-7Fh	1EFh	accesses 70h - 7Fh
		F0h		170h		1F0h	
		FFh		17Fh		1FFh	

Bộ nhớ RAM của vi điều khiển PIC 16F877A gồm 4 bank nhớ. Nhìn vào các bank nhớ ta có thể thấy các thanh ghi được đặt tên và các thanh ghi đa mục đích (General Purpose Register).

- Các thanh ghi được đặt tên là các thanh ghi đặc biệt dùng để điều khiển, quản lý hoặc thể hiện trạng thái của các khối chức năng trong vi điều khiển ví dụ PORTA

là đại diện cho các chân cổng A, PORTB là đại diện cho các chân cổng B v.v. Các thanh ghi này có địa chỉ xác định và không được dùng cho các mục đích khác

- Các thanh ghi đa mục đích được dùng để đặt biến trong một chương trình ứng dụng của vi điều khiển. Nhìn vào bản đồ bộ nhớ RAM, ta thấy biến có thể đặt từ địa chỉ 20F đến 7Fh trong bank nhớ 0, A0h-EFh, 120h-16Fh, 1A0h-1EFh.

Trở lại vấn đề về các cổng, tới đây ta có thể đưa ra nhận xét:

Thanh ghi PORTA phản ánh trạng thái của các chân cổng A, nghĩa là muốn tín hiệu đầu ra của các chân cổng A như thế nào ta chỉ việc đưa giá trị vào các bit tương ứng trên thanh ghi PORTA. Cũng như khi đọc giá trị của thanh ghi PORTA ta sẽ biết được trạng thái của các chân cổng A.

Ví dụ:

Muốn RA0 ở mức logic 1 (mức 5V), RA1 ở mức logic 0 (mức 0V), RA2 ở mức logic 1, RA3 ở mức logic 0, RA4 ở mức logic 1, RA5 ở mức logic 1, ta chỉ việc gán giá trị 000110101 cho thanh ghi PORTA.

X	X	1	1	0	1	0	1
		RA5	RA4	RA3		RA2	RA1

RA0

X: không quan tâm.

Tương tự như vậy với PORTB,PORT C,PORTD,PORTE.

Tính đa chức năng của một chân trên vi điều khiển:

Nhìn vào sơ đồ chân của vi điều khiển, ta có thể thấy một số chân của vi điều khiển có tên gồm nhiều phần với dấu gạch chéo. Ví dụ: RA0/AN0, RC7/RX/DT, RC6/TX/CK

Đây chính là tính đa chức năng của một chân trên vi điều khiển hay còn gọi là sự dồn kênh.

Ý nghĩa của nó là:

Bình thường nếu không được cài đặt thì tất cả các chân trên 5 cổng A, B, C, D, E là các chân vào ra số I/O.

Nếu trong chương trình ta có cài đặt một chức năng nào đó như RS232, ADC hoặc PWM v.v thì các chân tương ứng với chức năng đó sẽ hoạt động theo chức năng đó. Khi đó chân này sẽ không được dùng làm chân vào ra số như bình thường nữa.

Ví dụ: bình thường chân RA0/ANO là chân vào ra số RA0, nếu chức năng ADC với kênh vào tín hiệu analog là kênh 0 được cài đặt khi đó chân RA0 /AN0 sẽ là chân vào của bộ ADC, tức là hoạt động theo chức năng AN0.

Tương tự như vậy, khi cài đặt giao tiếp với thiết bị ngoại vi theo chuẩn RS232, chân vào ra số RC7/RX/DT sẽ hoạt động như đầu vào dữ liệu RS232 tức là chức năng RX của chân này.

Cài đặt vào/ra cho các chân vào ra số trên các cổng:

Các chân vào/ra số trên vi điều khiển PIC *phải được cài đặt là chân vào hoặc chân ra thì mới hoạt động đúng chức năng*. Việc một chân trên cổng X (X=A,B,..E) được qui định là đầu ra hay đầu vào phụ thuộc vào bit tương ứng trên thanh ghi TRISX (X=A,B,..E) là 0 hay 1.

Ví dụ: Muốn 4 chân thấp (bit thấp) trên cổng B (RB0-RB3) là chân vào, 4 chân cao (bit cao) trên cổng B (RB4-RB7) là chân ra thì giá trị các bit trên thanh ghi TRISB sẽ là:

0	0	0	0	1	1	1	1

Gợi ý để nhớ là:

Để chân RB.m (m=0-7) là đầu ra, tức **O**utput thì giá trị TRISB.m là **0**

Là đầu vào, tức **I**nput thì giá trị TRISB.m là **1**

Tương tự như vậy đối với các chân trên các cổng còn lại

2. Ngôn ngữ lập trình cho vi điều khiển PIC- CCS:

2.1 Các ngôn ngữ lập trình cho vi điều khiển PIC:

Ngôn ngữ lập trình cho vi điều khiển PIC có 2 loại:

- Ngôn ngữ lập trình cấp thấp- Hợp ngữ: có phần mềm MPLAB
- Ngôn ngữ lập trình bậc cao: có nhiều loại, được phát triển theo ngôn ngữ C, như: CCS, HTPIC, PIC BASIC v.v

Ưu điểm của hợp ngữ là giúp người học và lập trình hiểu rõ hơn về cấu trúc bên trong của vi điều khiển PIC, cũng như tối ưu hóa bộ nhớ chương trình. Tuy nhiên, nhìn chung phương pháp tiếp cận hợp ngữ là khó và khả năng phát triển ứng dụng là hạn chế, mất

thời gian. Vì vậy, khóa học sẽ tập trung vào sử dụng ngôn ngữ bậc cao mà cụ thể là CCS để nghiên cứu và phát triển các ứng dụng trên PIC.

Ưu điểm của CCS là:

- Kế thừa tất cả đặc điểm của ngôn ngữ C- là ngôn ngữ cơ bản, quen thuộc mà sinh viên đã được đào tạo
- Xây dựng sẵn các hàm phục vụ cho việc sử dụng dễ dàng các khối chức năng đặc biệt của Vi điều khiển PIC như khối ADC, PWM, RS232, SPI
- Có khả năng kết hợp với ngôn ngữ hợp ngữ, tạo sự mềm dẻo trong phát triển ứng dụng
- Khả năng phát triển, nâng cấp ứng dụng là dễ dàng
- Ngày càng được cập nhật với nhiều tính năng ưu việt và hiệu quả hơn.

2.2 Cơ bản về ngôn ngữ lập trình CCS:

2.2.1 Ví dụ về một chương trình viết trên ngôn ngữ CCS:

```
// Đây là chú thích chương trình
//Bắt đầu các chỉ thị tiền xử lý của chương trình
#include<16f877a.h> // cho file định nghĩa thiết bị 16f877a.h vào chương trình
#fuses HS,NOLVP,NOWDT// Cấu hình cho vi điều khiển PIC
#use delay (clock=4000000)   // dùng thạch anh tần số 4MHz
// Khai báo biến hằng
byte const MAP[10] = {0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,0x80,0x90};
//=========================================
//=========================================
// Bắt đầu chương trình con hiển thị
void display(int n)
{
  char b; // khai báo biến b
  b=((MAP[n/10]) ^ 0x00);
  if ((n/10)==(0)) b=0xff;
  output_b(b); // sử dụng hàm xuất giá trị ra cổng B
  output_low(PIN_A4);// sử dụng hàm đưa giá trị chân RA4 xuống mức thấp
```

```
  delay_ms(2);// Sử dụng hàm tạo trễ 2 ms
  output_high(PIN_A4); // sử dụng hàm đưa giá trị chân RA4 lên mức cao
  output_b((MAP[n%10]) ^ 0x00);
  output_low(PIN_A5);
  delay_ms(2);
  output_high(PIN_A5);
}
// Kết thúc chương trình con hiển thị
//=============================================
// Bắt đầu chương trình chính
// Đây là nơi vi điều khiển bắt đầu chạy lệnh
//=============================================
void main()
{
  int i,count;
  count=0;
  while(TRUE)
  {
    for (i=0;i<=50;i++)
      display(count);     // dispay 50 times
    count=(count==99) ? 1: count+1;
  }
}
```

2.2.2 Cấu trúc của một chương trình viết bằng CCS:

2.2.2.1 Khai báo tiền xử lý:

Bắt đầu một chương trình viết bằng ngôn ngữ CCS là phần khai báo tiền xử lý:

1. Đầu tiên là phần khai báo file header: *#include <tên chip dùng.h>*

   Ví dụ: # include <16f877a.h>

   Việc khai báo này thực chất là chép cả file 16f877a.h vào chương trình này.

Vậy nội dung của file .h này là gì? Nội dung của file này sẽ định nghĩa tất cả các tên cũng như định nghĩa các hằng sẽ được dùng trong các hàm chức năng của CCS.

Ví dụ trong chương trình cho ở trên, trong dòng lệnh :

output_low(PIN_A4);// sử dụng hàm đưa giá trị chân RA4 xuống mức thấp

thì hằng PIN_A4 được định nghĩa trong file 16f877a.h là **#define PIN_A4 44, CCS hiểu đây là bit thứ 4 của thanh ghi có địa chỉ 04h (thanh ghi PORTA).**

Có thể mở file: C:\Program Files\PICC\Devices\16f877a.h để hiểu thêm

2. Thứ hai là phần khai báo cấu hình: *#fuses HS,NOLVP,NOWDT*

   Vi điều khiển PIC có thể hoạt động ở nhiều chế độ khác nhau, cũng như cách cấu hình phần cứng của nó cũng có nhiều chế độ. Mỗi chế độ như vậy được khai báo và dùng sẽ ảnh hưởng đến hoạt động của các khối chức năng, cũng như các chân của vi điều khiển. Vì vậy, đối với mỗi ứng dụng, ta phải khai báo cấu hình cho phù hợp.

   Trong ví dụ trên, khai báo cấu hình cho bộ dao động kiểu HS, không sử dụng chức năng Watchdog Timer, và lập trình điện áp thấp.

3. Thứ ba là phần khai báo tần số của thạch anh dùng cho ứng dụng, tốc độ này phải phù hợp với thạch anh ta dùng trong mạch.

   # USE Delay(clock=tần số thạch anh)

   Ví dụ: *# USE Delay(clock=4000000)// Khai báo dùng thạch anh 4 MHz*

   Điều lưu ý là chúng ta chỉ dùng được hàm tạo thời gian trễ delay_ms(), delay_us() sau khi có khai báo này trong chương trình.

4. Ngoài ra, khi sử dụng bất cứ khối chức năng đặc biệt nào trong vi điều khiển PIC ta phải dùng chỉ thị tiền xử lý #USE để khai báo. Các khối chức năng đặc biệt là RS232, PWM, SPI, I2C ..v.v

   Ví dụ: #use rs232(baud=9600, xmit=PIN_C6,rcv=PIN_C7)

2.2.2.2 Phần khai báo biến toàn cục:

Sau phần khai báo tiền xử lý là phần khai báo biến toàn cục nếu có.

Cũng nhắc lại biến toàn cục là biến được sử dụng trong toàn bộ chương trình, cả chương trình chính và chương trình con. Điều này khác với biến cục bộ là biến được khai báo

trong các chương trình con, hàm con, chương trình chính, các biến này chỉ được tạo ra và sử dụng trong chính mỗi chương trình khai báo nó.

Nhân đây cũng giới thiệu qua dạng dữ liệu dùng trong CCS. Có một số các dạng cơ bản như sau:

Khai báo biến:

Int1,short: dạng logic 1,0

Int,Int8,byte: dạng số nguyên 8 bit, nếu khai báo dạng số nguyên 8 bit có dấu thêm signed

Int16,long: dạng số nguyên 16 bit

Int32: dạng số nguyên 32 bit

Char: dạng kí tự 8 bit

Float: dạng số thực 32 bit

Ví dụ: int8 a; // Khai báo a là biến số nguyên 8 bit

Khai báo hằng số:

Ví dụ: int8 const a=231;

Khai báo mảng hằng số:

Ví dụ: int const a[5]= {1, 23, 3, 4, 5}

2.2.2.3 Phần khai báo, định nghĩa các chương trình con:

Chương trình con là chương trình sẽ được gọi từ chương trình chính hoặc chương trình con khác.

Chương trình con phải được khai báo và định nghĩa trong một file chương trình ứng dụng.

Phần khai báo phải đặt trước chương trình chính (trong file)

Phần định nghĩa có thể định nghĩa ngay trong khai báo hoặc được đặt bất kì nơi nào sau phần khai báo biến toàn cục nếu có.

Ngoài các chương trình con bình thường ra, còn có chương trình con phục vụ ngắt được đặt sau khai báo tiền xử lý: # int_tên ngắt. Phần này sẽ được bàn kĩ hơn trong các bài học sau.

2.2.2.3 Phần chương trình chính:

Bắt đầu bằng:

void main() {

}

Cũng nhắc lại là trong một chương trình CCS , vi điều khiển sẽ chạy từ chương trình chính, hay nói cách khác là từ dòng lệnh đầu tiên sau void main().

Các chương trình con chỉ được gọi tại các lời gọi chương trình con trong chương trình chính hoặc từ các chương trình con khác. Chương trình con phục vụ ngắt chỉ được chạy khi có ngắt xảy ra, và ngắt đó được cho phép (sẽ bàn kĩ hơn trong các bài học sau).

2.2.3 Các cấu trúc thuật toán của ngôn ngữ CCS:

Cấu trúc thuật toán của ngôn ngữ CCS kế thừa 100% từ ngôn ngữ C. Ở đây xin nhắc lại một số các cấu trúc hay dùng:

- *Cấu trúc IF:*
    - If (biểu thức)
      Lệnh1;
      Else lệnh2;

  Ví dụ: if (x==25)
  x=1;
  else
  x=x+1;
- *Cấu trúc lặp While:*
    - While (biểu thức)
      {
      Các lệnh;
      }

  Ví dụ: While (count<20)
  {
  Output_B(count);
  Count=count+1;
  }

  Chú ý: while(1) sẽ thực hiện các lệnh trong khối while này mãi mãi
- *Cấu trúc lặp FOR:*

- For (biểu thức 1, biểu thức 2, biểu thức 3)

  {

  Các lệnh;

  }

  Ví dụ: for (i=1;i<=10;++i)

  A=a+i;

- *Cấu trúc lựa chọn SWITCH:*
  - SWITCH (biến)

    {

    Case giá trị 1: lệnh 1;

    Break;

    Case giá trị 2: lệnh 2;

    Break;

    ……

    Case giá trị n: lệnh n

    Break;

    [default: lệnh n+1; break;]

    }

Ví dụ: switch (cmd)

```
{   case 0:printf("cmd 0");
                break;
    case 1:printf("cmd 1");
                break;
   default:printf("bad cmd");
                break;
}
```

Chú ý: lệnh break được dùng để thoát lập tức khỏi một vòng lặp For, While, Switch

2.2.4 Các toán tử cơ bản trong CCS:

Hoàn toàn tương tự trong C.

Nhắc lại một số toán tử hay dùng:

- += ý nghĩa là cộng thêm một giá trị và lấy kết quả

Ví dụ: a+=2 nghĩa là a= a+2

Tương tự đối với các phép toán trừ, chia, nhân

-++ ý nghĩa là cộng thêm 1 đơn vị vào biến

Ví dụ: a++; tức là a=a+1

- &&: phép AND
- ||: phép OR
- !: phép NOT
- !=: không bằng
- >>n: dịch trái n bit
- <<n: dịch phải n bit

2.2.5 Các hàm số học cơ bản trong CCS:

Hoàn toàn giống trong C.

Nhắc lại một số hàm cơ bản:

Abs (): lấy trị tuyệt đối

Ceil(): làm tròn theo hướng tăng

Floor(): làm tròn theo hướng giảm

Pow(): lũy thừa

Sqrt(): lấy căn

Chi tiết tra help CCS : trong tab contents, chọn Built-In-Function

Chú ý là khi sử dụng các hàm này cần khai báo file header tương ứng

2.2.6 Các hàm vào ra cơ bản trong CCS:

- **Output_low (*chân*)**: cho *chân* xuống mức logic thấp (mức điện áp 0V), chân có thể là:

PIN_A0, PIN_A1, ..PIN_B0, PIN_B1,.. PIN_C0,..v.v Nói chung là các chân có tên trong file C:\Program Files\PICC\Devices\16f877a.h

Ví dụ: Output_low(PIN_D0) đưa chân RD0 của PIC xuống mức thấp

- **Output_high(*chân*)**: cho *chân* lên mức logic thấp (mức điện áp 0V), chân có thể là:

  PIN_A0, PIN_A1, ..PIN_B0, PIN_B1,.. PIN_C0,..v.v Nói chung là các chân có tên trong file C:\Program Files\PICC\Devices\16f877a.h

  Ví dụ: Output_high(PIN_a5) đưa chân Ra5 của PIC lên mức cao
- **Output_bit(*chân,giá trị*)**: là lệnh tổng hợp 2 lệnh trên, xuất *giá trị* ra *chân.* Giá trị có thể là 0 (mức thấp) hoặc là 1 (mức cao). Tên chân tương tự như 2 lệnh trên

  Ví dụ: output_bit(PIN_E0,1); đưa chân E0 lên mức 1
- **Output_X(*giá trị*)**: lệnh này đưa *giá trị* ra cổng X. X có thể là A, B, C, D, E.

  Ví dụ: Output_A(0x21); đưa giá trị 0x21 ra cổng A
- **Biến=Input_X():** đưa giá trị của cổng X vào B**iến**. X là A, hoặc B, C, D, E

  Ví dụ: bien1= Input_A()
- **Biến=Input(*chân*)**: lệnh này đưa giá trị của *chân* vào biến. Chân tương tự như trong các lệnh.

  Ví dụ: bien2= Input(PIN_A2);

2.2.7 Các hàm tạo trễ:

Các hàm tạo trễ gồm delay_cycles(), delay_us(), delay_ms. Tạo một khoảng thời gian trễ từ lúc lệnh được thực hiện. Chú ý là phải sử dụng khai báo tiền xử lý # use delay(clock=tần số) thì mới dùng được các lệnh này.

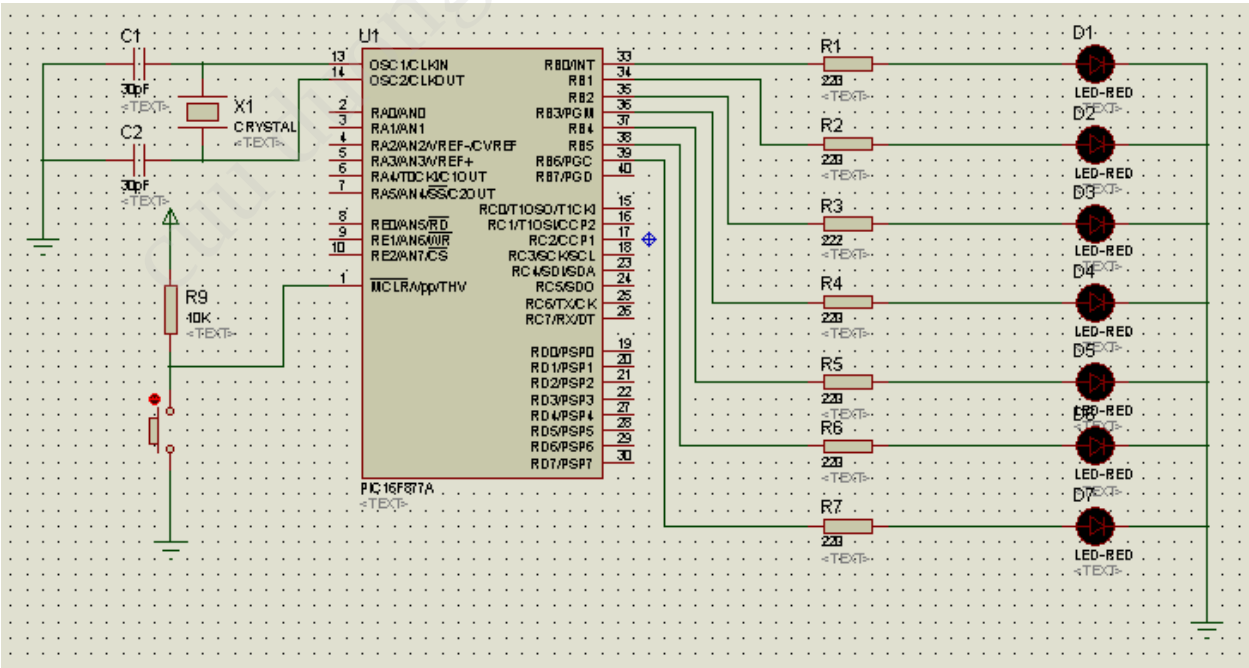
- **delay_cycles(số chu kỳ)**: tạo trễ một khoảng thời gian bằng số chu kỳ. Số chu kỳ=0-255
- **delay_us(số micro giây)**: tạo trễ số micro giây. Số micro giây =0-65535
- **delay_ms(số mili giây)**: tạo trễ số mili giây. Số mili giây= 0-65535

II. PHẦN THỰC HÀNH

Mục tiêu:

- Làm quen với ngôn ngữ lập trình CCS, phần mềm mô phỏng Proteus, kit phát triển
- Hiểu được các phần cơ bản của một mạch thực thi vi điều khiển PIC
- Hiểu được việc xuất nhập dữ liệu trên các chân vi điều khiển
- Ôn lại tư duy lập trình, thuật toán và cấu trúc một chương trình viết trên C
- Làm quen với một số lệnh cơ bản trong CCS

Bài tập 1.1: Dùng phần mềm mô phỏng Proteus thiết kế mạch như hình vẽ sau:



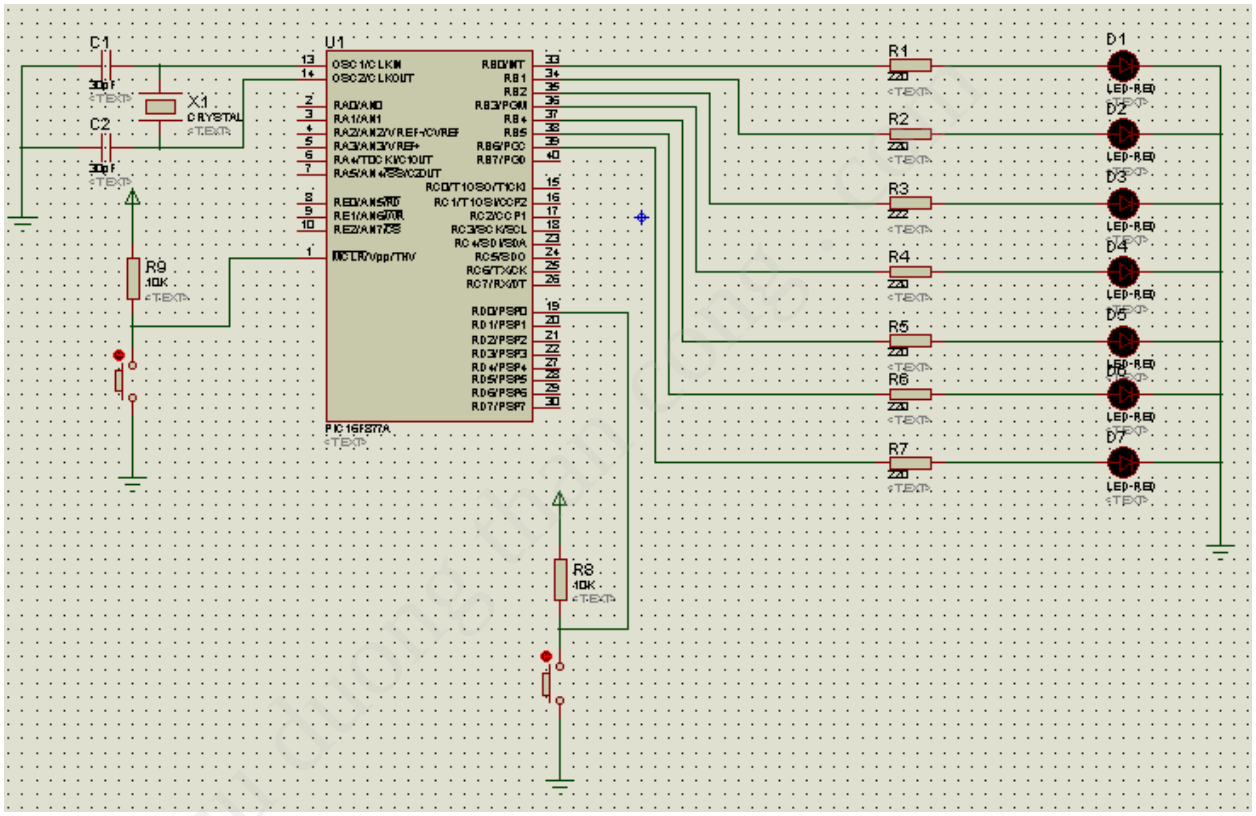
a. Viết chương trình bật đèn led D1

b. Viết chương trình bật đèn led D2, tạo trễ 1s, tắt đèn led D2, lặp lại quá trình này

c. Viết chương trình bật đèn D1, D2, …D7 theo giá trị của cổng B từ 0x00 đến 0xFF rồi lặp lại quá trình đó.

d. Viết chương trình bật các đèn led theo trình tự như sau: D1 sáng, D1 tắt đồng thời D2 sáng, D2 tắt đồng thời D3 sáng v.v. Và lặp lại

Bài tập 1.2: Dùng Proteus thiết kế mạch như hình vẽ



Viết chương trình hiển thị các led theo qui luật sau:

- Ban đầu không có đèn nào sang
- Mỗi lần bấm vào phím bấm nối với chân RD0, đèn sẽ sáng theo số lần bấm: bấm 1 lần đèn D1 sáng, bấm 2 lần đèn D2 sáng,… bấm 8 lần đèn D8 sáng, bấm 9 lần quay lại chu trình trên.

Bài 1.3: Viết chương trình mô phỏng hào quang ở chùa trong ngày lễ.

BÀI 2: CÁC PHƯƠNG PHÁP HIỂN THỊ TRONG CÁC THIẾT BỊ DÙNG VI ĐIỀU KHIỂN

PHẦN LÝ THUYẾT:

Hiện nay, trong hầu hết các thiết bị nhúng đều có sử dụng các khối hiển thị. Mục đích cho người dùng giám sát, cài đặt và hiển thị các thông số của thiết bị cũng như đối tượng cần giám sát điều khiển.

Có rất nhiều phương pháp hiển thị, có thể kể ra như sau:

- Hiển thị cảnh báo, báo lỗi: thông thường dùng led đơn. Có thể hiển thị theo kiểu dùng nhiều màu khác nhau hoặc bật tắt v.v
- Hiển thị số liệu: dùng led 7 đoạn, LCD hoặc LCD đồ họa v.v
- Hiển thị trên máy tính: dùng các phần mềm điều khiển giám sát, kết nối thiết bị và máy tính thông qua chuẩn RS232 hoặc các chuẩn mạng (giám sát từ xa)

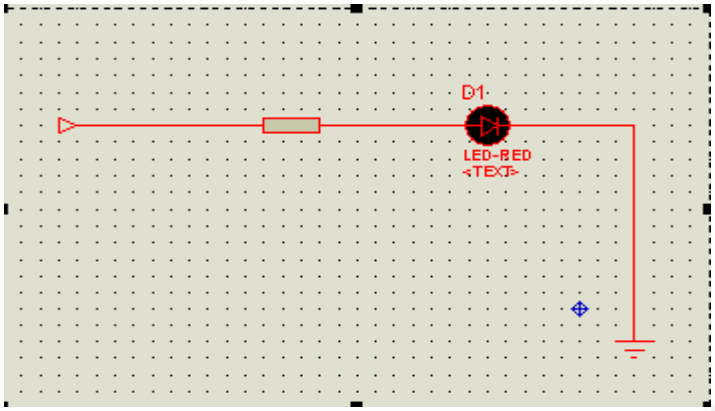
Trong bài này sẽ giới thiệu 2 cách hiển thị đầu, phần hiển thị bằng máy tính sẽ được đề cập trong bài học về chuẩn giao tiếp RS232.

2.1 Hiển thị bằng led đơn:

Đây là cách hiển thị đơn giản nhất.

Thông thường cách hiển thị này dùng để báo một trạng thái nào đấy của thiết bị như trạng thái làm việc của nguồn (lỗi hoặc không lỗi), cũng như các khối chức năng khác.

Có rất nhiều loại led đơn dùng để hiển thị. Phương pháp đơn giản như sau:



Hình 2.1: Hiển thị led đơn

Các led này sáng khi được cấp áp cỡ 2 V, dòng 10-20mA.

Nếu dùng một chân ra từ vi điều khiển để bật tắt led, phải dùng thêm điện trở hạn dòng,hạn áp.

Tính toán như sau:

Muốn bật đèn, ta cho chân ra vi điều khiển lên mức cao nối với đầu vào của mạch trên. Như ta biết, chân ra vi điều khiển ở mức logic cao có điện áp 5V.

Cho điện áp rơi trên led là 2V, dòng qua là 15mA.

Suy ra, điện áp rơi trên trở là 3V. Dòng qua led chính là dòng qua điện trở và bằng 15mA.

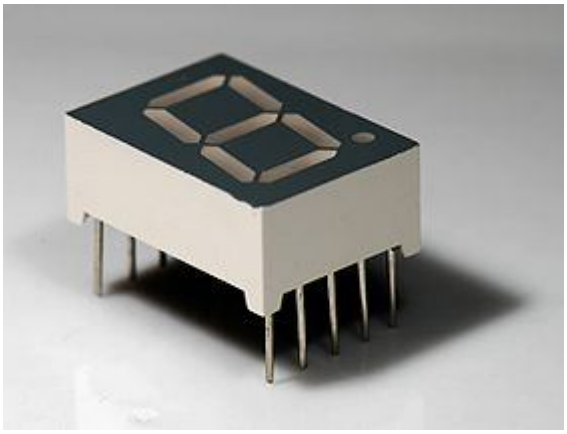
Suy ra, điện trở dùng: R=3/15mA=200 ohm.

Chọn điện trở tiêu chuẩn 220 ohm

(Điện trở tiêu chuẩn: 10, 11, 12, 13, 15, 16, 18, 20, 22, 24, 27, 30, 33, 36, 39, 43, 47, 51, 56, 62, 68, 72, 82, 91 và các bội số)

2.2 Hiển thị bằng led bảy đoạn- 7 segment led:

2.2.1 Cấu tạo của led 7 đoạn:

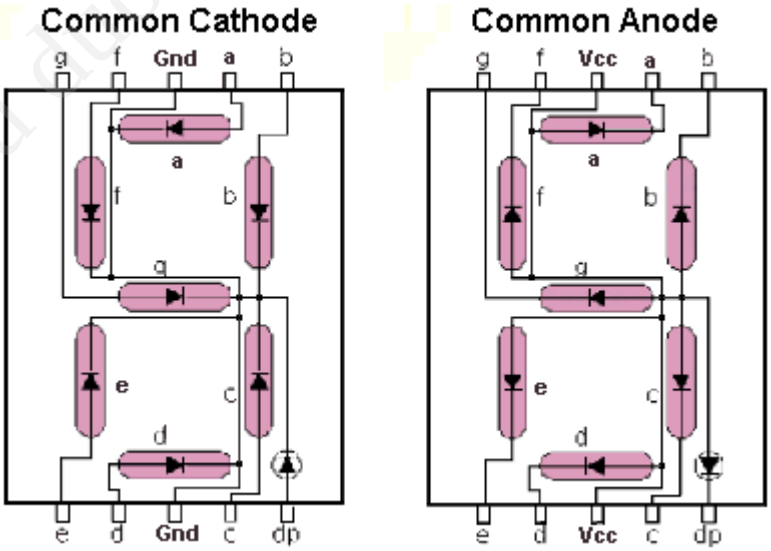


Hình 2.2: Led 7 đoạn

Một led 7 đoạn thực ra là gồm 7 led đơn nối với nhau (8 led đơn nêu có thêm dấu chấm-dp).

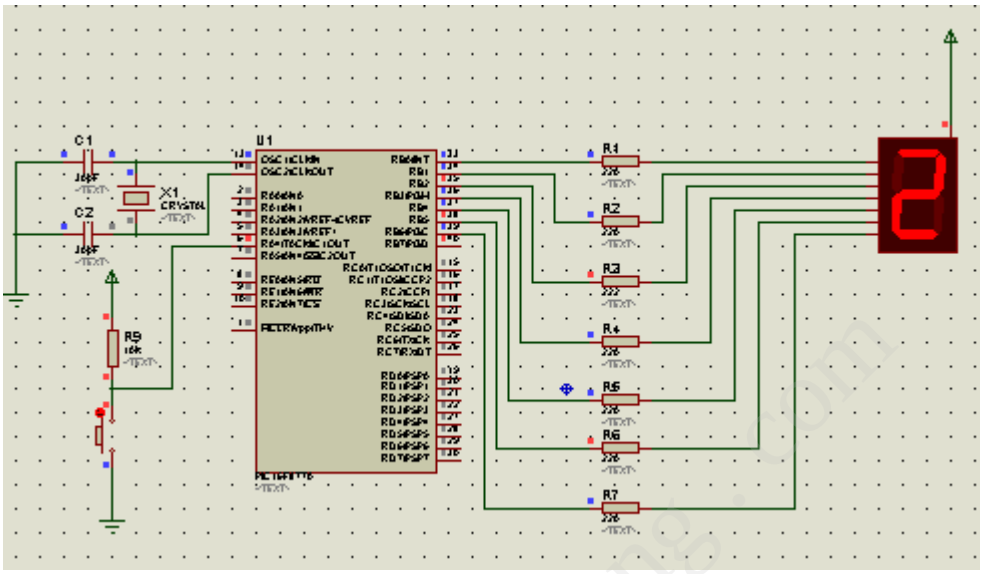
Có 2 loại:

- Chung catod: các đầu catod (cực âm) được nối chung với nhau và nối với đất, các đầu anod a,b,c,d,e,f,g,h được đưa ra ngoài (các chân) nhận tín hiệu điều khiển. Khi cấp điện áp 5v cho mỗi đầu anod, led tương ứng với đầu đó sẽ sáng
- Chung anod: các đầu anod (cực âm) được nối chung với nhau và nối với nguồn, các đầu catod a,b,c,d,e,f,g,h được đưa ra ngoài (các chân) nhận tín hiệu điều khiển. Muốn led đơn nào sáng chỉ việc đưa chân catod của led tương ứng xuống mức 0V.



Hình 2.3: Cấu tạo của 2 loại led 7 đoạn

2.2.2 Hiển thị 1 led 7 đoạn dùng vi điều khiển:



Như đã giới thiệu ở phần trên, thực chất led 7 đoạn gồm 8 hoặc 7 led đơn nối với nhau. Vì vậy để điều khiển thanh led đơn sáng, cách thực hiện phần cứng như hình 2.1.

Cụ thể hơn, như dùng led chung anod như hình vẽ trên. Mỗi đầu vào a,b,c,d,e,f,g,h được nối với một chân của vi điều khiển, tương ứng là RB0, RB1, ..RB7, thông qua các điện trở phân áp 200 ohm, đầu anod chung được nối với nguồn. Để led đơn sáng đơn giản ta đưa chân vi điều khiển nối với led đó xuống mức thấp.

Như trên hình 2.4 trên, để led 7 đoạn hiển thị số 2 thì các led a,b,d,e,g sáng; các led c, f tắt. Giá trị sáng tương ứng chân vi điều khiển nối vào ở mức 0, giá trị tắt tương ứng với chân vi điều khiển nối với ở mức 1.

Do đó nội dung của thanh ghi PORTB là:

0	0	1	0	0	1	0	0

Đây là mã led 7 đoạn của số 2

Như vậy, chúng ta lưu ý một điều rằng, dữ liệu xuất ra led 7 đoạn là mã led tương ứng với số cần xuất

Mã led tương ứng với các số từ 0 đến 9 là:

0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,0x80,0x90

Cách điều khiển led 7 đoạn chung catod thì ngược lại.

2.2.3 Hiển thị nhiều led 7 đoạn dùng vi điều khiển:

Trong thực tế, ta phải dùng nhiều led 7 đoạn để hiển thị.

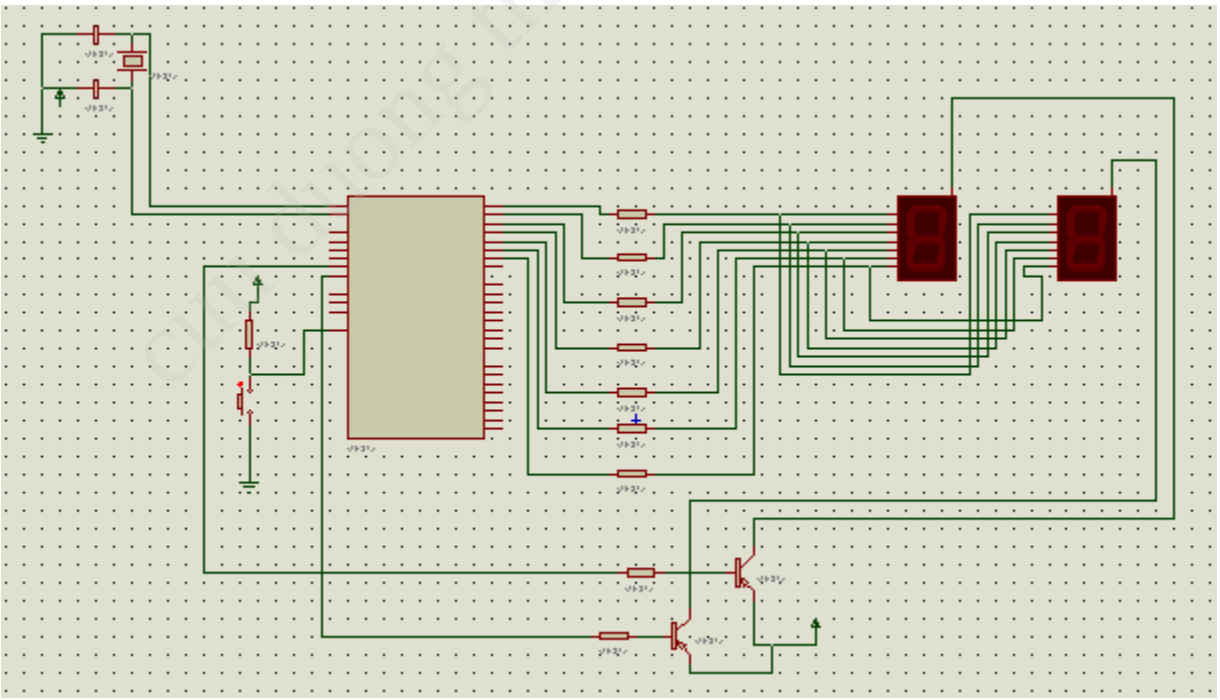
Vậy giải quyết việc hiển thị nhiều led như thế nào?

Ví dụ: để hiển thị số 35 bằng 2 led 7 đoạn.

Đối chiếu với cách hiển thị 1 led 7 đoạn, ta nghĩ đơn giản chỉ là dùng 1 cổng hiển thị số 3, 1 cổng khác hiển thị số 5.

Như vậy ta mất 2 cổng. Hiển thị 4 led thì mất 4 cổng => toàn bộ chân trên vi điều khiển dùng cho việc hiển thị led…Không còn chân để giao tiếp với các thiết bị khác như bàn phím, đầu vào số khác v.v Không khả thi!

Ta có phương pháp tiết kiệm chân hơn để giải quyết:



Hình 2.5: Hiển thị 2 led 7 đoạn

Các chân dữ liệu (chân sẽ nhận mã led từ vi điều khiển) được nối tương ứng với nhau và nối vào 1 cổng của vi điều khiển, chẳng hạn như cổng B

Chân nguồn của 2 led được điều khiển bởi 2 chân trên vi điều khiển, chẳng hạn chân RA4 và RA5 như trên hình, thông qua cực B của 2 transistor pnp.

Quá trình hiển thị con số 35 trên 2 led sẽ như sau:

- Cho chân RA4 (chân nối với led hàng chục) xuống mức thấp, transistor thứ nhất mở do tiếp giáp BE thuận, chân RA5 lên mức cao (chân nối với led hàng đơn vị), transistor thứ hai không mở. Vậy chỉ có led hàng chục được cấp nguồn.
- Cho cổng B xuất dữ liệu mã led của số 3. Chỉ có led hàng chục được cấp nguồn nên chỉ có led này sáng
- Tạo thời gian trễ 10-20ms
- Điều khiển tương tự cho led hàng đơn vị được cấp nguồn, led hàng chục không cấp nguồn, xuất dữ liệu mã led số 5 ra cổng B. Led đơn vị hiển thị số 5.
- Tạo thời gian trễ 10-20ms
- Quay lại bước thứ nhất

Như vậy, số 3 hiển thị 10ms, số 5 hiển thị 10ms và quay vòng như vậy. Thời gian này rất nhanh, do hiệu ứng của mắt, ta cảm giác như số 35 hiển thị cùng lúc. Bài toán được giải quyết, ta chỉ mất có 10 chân để điều khiển 2 led.

Cách hiển thị nhiều led cũng tương tự như vậy.

Cũng giải thích thêm lí do dùng transistor nối vào RA4, RA5. Do chân vi điều khiển có dòng khoảng vài chục mA, đây là chân cấp nguồn cho led 7 đoạn, mỗi led đơn trong Led 7 đoạn mất 20mA vậy cả led 7 đoạn mất trên 100mA. Vì vậy ta phải dùng transistor để khuếch đại dòng.

2.3 Hiển thị dùng LCD:

Hình 2.6: Các chân LCD

2.3.1 Các chân cơ bản của LCD 2 dòng 16 kí tự:

- VSS: Chân đất
- VCC: Chân nguồn
- VEE: Chân hiệu chỉnh độ sáng của LCD
- RS:
  - =0: LCD sẽ nhận lệnh từ vi điều khiển
  - =1: LCD sẽ nhận kí tự từ vi điều khiển để hiển thị
- R/W:
  - =1: Vi điều khiển đọc dữ liệu từ LCD
  - =0: Vi điều khiển ghi dữ liệu lên LCD

Thông thường Vi điều khiển chủ yếu ghi dữ liệu lên LCD nên chân này thường nối đất

- E: Chân cho phép (Enable). Sau khi lệnh hoặc dữ liệu đã được chuẩn bị trên đường dữ liệu, tạo xung mức cao-mức thấp sẽ bắt đầu quá trình LCD nhận dữ liệu hoặc lệnh từ vi điều khiển.
- D0-D7: các chân dữ liệu, nơi vi điều khiển truyền lệnh hoặc dữ liệu lên LCD.

2.3.2 Khởi tạo LCD:

LCD có nhiều độ làm việc, có thể kể ra như sau:

- Chế độ 1 dòng hay 2 dòng
- Chế độ giao tiếp 4 bit hay 8 bit
- Chế độ font 5*8 hoặc 5*10
- Ngoài ra còn có thể thay đổi vị trí hiển thị kí tự v.v

Vì vậy, trước khi bắt đầu quá trình hiển thị một chuỗi kí tự nào đó, ta cần quá trình khởi tạo để

cài đặt các chế độ này. Vi điều khiển thực hiện quá trình khởi tạo này bắt cách ghi đến LCD một chuỗi các lệnh.

Căn cứ vào chức năng của các chân vi điều khiển được giới thiệu ở trên, ta đưa ra qui trình của việc gửi một lệnh từ Vi điều khiển đến LCD:

- Cho chân R/W=0 để xác định đây là ghi xuống LCD (thông thường chân này được nối đất, nên mặc định chân này ở mức 0, ta không cần quan tâm đến nữa)
- Cho chân RS=0 để xác định đây là lệnh mà vi điều khiển gửi xuống LCD (phân biệt với RS=1, gửi kí tự hiển thị)
- Gửi **mã lệnh** xuống LCD theo các đường dữ liệu (RD0-RD7 nếu dùng chế độ 8 bit, R4-R7 nếu dùng chế độ 4 bit)
- Đưa chân E (chân cho phép- Enable) lên mức cao, mức 1
- Tạo trễ vài chu kì lệnh
- Đưa chân E xuống mức thấp, mức 0

Mã lệnh như đã giới thiệu trong phần trên tùy thuộc vào từng lệnh, ở đấy giới thiệu một số lệnh cơ bản như sau:

. Lệnh cài đặt chế độ làm việc:

0	0	1	DL	N	F	-	-

- DL:
  - = 1: 8 bit
  - = 0: 4 bit
- N:
  - = 1: 2 dòng
  - = 0 1 dòng
- F:
  - = 1: font 5x10 dot
  - = 0: font 5x8 dot

. Lệnh đặt chế độ tăng giảm địa chỉ:

0	0	0	0	0	1	I/D	S

- I/D:
  - = 1 tăng địa chỉ
  - = 0 giảm địa chỉ
- S:
  - =1: Cài đặt di chuyển cùng địa chỉ

. Lệnh đặt chế độ hiển thị:

0	0	0	0	1	D	C	B

- D: Cho phép hiển thị
- C: cài đặt hiển thị con trỏ
- B: nhấp nháy vị trí kí tự

. Lệnh đặt vị trí hiển thị của kí tự:

1	ĐC	ĐC	ĐC	ĐC	ĐC	ĐC	ĐC

- Địa chỉ dòng 1: 00- 0F
- Địa chỉ dòng 2: 40-4F

Vì vậy, muốn hiển thị đầu dòng thứ nhất, mã lệnh sẽ là 0x80

muốn hiển thị đầu dòng thứ hai, mã lệnh sẽ là 0xC0

. Lệnh xóa màn hình: mã lệnh 0x01

. Lệnh trở về đầu dòng thứ nhất: mã lệnh 0x02

Chi tiết có thể xem datasheet đi kèm

2.3.2 Ghi kí tự lên LCD để hiển thị:

Sau khi thực hiện quá trình khởi tạo để gửi các lệnh cài đặt chế độ làm việc của LCD, kí tự sẽ được hiển thị lên LCD bất kì khi nào vi điều khiển muốn gửi.

Quá trình gửi kí tự gồm các bước sau:

- Cho chân R/W=0 để xác định đây là ghi xuống LCD (thông thường chân này được nối đất, nên mặc định chân này ở mức 0, ta không cần quan tâm đến nữa)
- Cho chân RS=1 để xác định đây là kí tự mà vi điều khiển gửi xuống LCD (phân biệt với RS=0, gửi lệnh)
- Gửi **mã ascii của kí tự cần hiển thị** xuống LCD theo các đường dữ liệu (RD0-RD7 nếu dùng chế độ 8 bit, R4-R7 nếu dùng chế độ 4 bit)
- Đưa chân E (chân cho phép- Enable) lên mức cao, mức 1
- Tạo trễ vài chu kì lệnh
- Đưa chân E xuống mức thấp, mức 0

2.4 Giới thiệu một thư viện cho LCD 4 bit và bài tập ứng dụng:

File: LCD_lib_4bit:

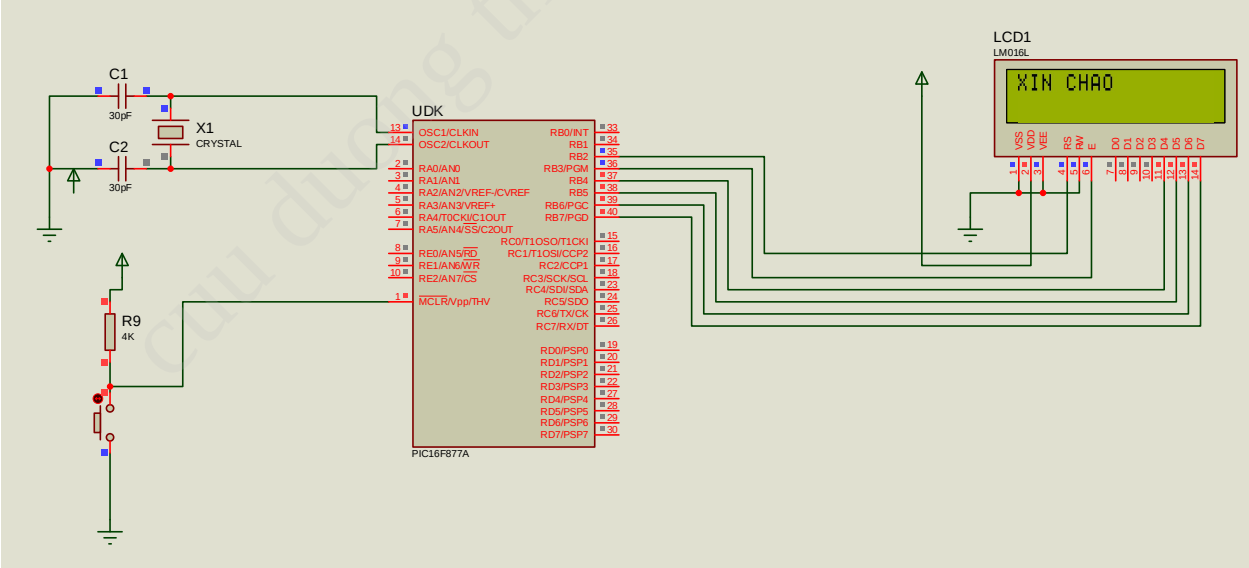
```
#include <stddef.h>

#define LCD_RS          PIN_B2

#define LCD_EN          PIN_B3

#define LCD_D4          PIN_B4

#define LCD_D5          PIN_B5

#define LCD_D6          PIN_B6

#define LCD_D7          PIN_B7

#define Line_1          0x80

#define Line_2          0xC0

#define Clear_Scr       0x01

#separate void LCD_Init ();// ham khoi tao LCD

#separate void LCD_SetPosition ( unsigned int cX );//Thiet lap vi tri con tro

#separate void LCD_PutChar ( unsigned int cX );// Ham viet1kitu/1chuoi len LCD

#separate void LCD_PutCmd ( unsigned int cX) ;// Ham gui lenh len LCD

#separate void LCD_PulseEnable ( void );// Xung kich hoat

#separate void LCD_SetData ( unsigned int cX );// Dat du lieu len chan Data


//khoi tao LCD**********************************************

#separate void LCD_Init ()

   {

   LCD_SetData ( 0x00 );
```

```
   delay_ms(20);       /*Cho cho lcd khoi tao */
   output_low ( LCD_RS );// che do gui lenh
   LCD_SetData ( 0x03 );   /* khoi tao che do 4 bit */
   LCD_PulseEnable();
   LCD_PulseEnable();
   LCD_PulseEnable();
   LCD_SetData ( 0x02 );   /* tao giao dien 4 bit */
   LCD_PulseEnable();      /* send dual nibbles hereafter, MSN first */
   LCD_PutCmd ( 0x2C );    /* function set (all lines, 5x7 characters) */
   LCD_PutCmd ( 0x0C );    /* display ON, cursor off, no blink */
   LCD_PutCmd ( 0x06 );    /* entry mode set, increment & scroll left */
   LCD_PutCmd ( 0x01 );    /* clear display */

   // Init for BarGraph

   }

#separate void LCD_SetPosition ( unsigned int cX )
   {
   /* this subroutine works specifically for 4-bit Port A */
   LCD_SetData ( swap ( cX ) | 0x08 );
   LCD_PulseEnable();
```

```
   LCD_SetData ( swap ( cX ) );
   LCD_PulseEnable();
   }

#separate void LCD_PutChar ( unsigned int cX )
   {
   /* this subroutine works specifically for 4-bit Port A */
      output_high ( LCD_RS );
      LCD_PutCmd( cX );
      output_low ( LCD_RS );
   }
#separate void LCD_PutCmd ( unsigned int cX )
   {
   LCD_SetData ( swap ( cX ) );     /* send high nibble */
   LCD_PulseEnable();
   LCD_SetData ( swap ( cX ) );     /* send low nibble */
   LCD_PulseEnable();
   }
#separate void LCD_PulseEnable ( void )
   {
   output_high ( LCD_EN );
   delay_us ( 3 );         // was 10
```

```
    output_low ( LCD_EN );
    delay_ms ( 3 );         // was 5
    }
#separate void LCD_SetData ( unsigned int CX )
    {
    output_bit ( LCD_D4, CX & 0x01 );
    output_bit ( LCD_D5, CX & 0x02 );
    output_bit ( LCD_D6, CX & 0x04 );
    output_bit ( LCD_D7, CX & 0x08 );
    }
```

Chương trình ứng dụng:



Hình 2.7: Ví dụ về LCD

```
#include <16f877A.h>
#fuses HS, NOLVP, NOWDT, NOPROTECT
```

```
#use   delay (clock=4000000) //Use built-in function: delay_ms() & delay_us()

#include "LCD_LIB_4BIT.c"



VOID MAIN()


{

  LCD_INIT();

  LCD_PUTCHAR('X');

  DELAY_MS(1000);

  LCD_PUTCHAR('I');

  DELAY_MS(1000);

  LCD_PUTCHAR('N');

  DELAY_MS(1000);

  LCD_PUTCHAR(' ');

  DELAY_MS(1000);

  LCD_PUTCHAR('C');

  DELAY_MS(1000);

  LCD_PUTCHAR('H');

  DELAY_MS(1000);

  LCD_PUTCHAR('A');

  DELAY_MS(1000);
```

```
LCD_PUTCHAR('O');
```

PHẦN THỰC HÀNH:

Bài 2.2: Hiển thị 1 led 7 đoạn

Bài 2.3: Thiết kế mạch trên Proteus như hình 2.5

a. Lập trình cho 2 led hiển thị số 35
b. Lập trình hiển thị số đếm từ 0-99

Bài 2.4 Thiết kế mạch trên Proteus như hình 2.7

Lập trình thực hiển hiển thị họ tên, thay đổi các chế độ hiển thị bằng cách gửi lệnh lên LCD

BÀI 3: BỘ ĐỊNH THỜI - TIMER

3.1 Giới thiệu:

Trong các ứng dụng của vi điều khiển trong thực tế, việc định thời (tạo một khoảng thời gian giữa 2 sự kiện) các thao tác là việc thường xuyên xảy ra.

Để thực hiện việc này, ta có 2 cách:

- Dùng các lệnh thực hiện các vòng lặp để tạo ra khoảng thời gian. Nguyên tắc tạo ra khoảng thời gian này đơn giản như sau: nếu vi điều khiển mất một khoảng thời gian x để thực hiện một lệnh, việc lặp lại một lệnh n lần sẽ mất n*x thời gian.
  Trong chương trình, cách này được dùng nhiều với thể hiện là các lệnh Delay_ms(), Delay_us()
- Dùng các bộ định thời Timer để tạo ra khoảng thời gian trễ.

Trong bài này, ta sẽ đi vào nguyên cứu các bộ timer. Một chế độ quan trọng nữa của Timer là khi nó hoạt động như bộ đếm. Trong ứng dụng này, timer hoạt động như một bộ đếm, có nhiệm vụ đếm số các xung đi vào một chân cụ thể trên vi điều khiển. Chế độ bộ đếm này có nhiều ứng dụng trong thực tế như đếm số vòng quay của động cơ (phản hồi từ bộ đo tốc độ động cơ-encoder), đếm số sản phẩm trên một dây chuyền v.v.

Vi điều khiển PIC16F877A có 3 bộ Timer:

- Timer0: 8 bit (số đếm tối đa của nó là 255), hoạt động ở 2 chế độ định thời và bộ đếm.
- Timer1: 16 bit (số đếm tối đa của nó là 65535), hoạt động ở 2 chế độ định thời và bộ đếm.
- Timer2: 8 bit, hoạt động phục vụ chức năng PWM (Pulse Width Modulation- Điều chế độ rộng xung)

Trong bài này ta đi vào khảo sát 2 bộ Timer0 và Timer1, Timer2 sẽ được khảo sát trong bài về PWM.

Để tiện cho việc khảo sát, ta đi vào nguyên lý hoạt động cơ bản của các bộ timer ở hai chế độ: định thời và bộ đếm.

3.2 Nguyên lý hoạt động cơ bản của một bộ Timer:

3.2.1 Chế độ định thời:

Mỗi bộ timer có một hoặc nhiều thanh ghi chứa giá trị đếm của nó (tùy thuộc vào độ dài của timer), ta giả sử tên thanh ghi là TMR có độ dài là n byte, hay giá trị đếm tối đa là $2^{8n}-1$. Khi giá trị của TMR đạt đến giá trị này, vi điều khiển sẽ set bit cờ của bộ timer đó lên mức 1. Người dùng sẽ biết được thời điểm này bằng cách kiểm tra bit cờ. Đồng thời TMR sẽ tự động xóa về giá trị 0.

Khi được cài đặt hoạt động trong chế độ định thời, Giá trị của thanh ghi TMR sẽ tự động tăng lên 1 đơn vị sau mỗi chu kì lệnh của vi điều khiển. Khi giá trị của TMR đạt đến giá trị tối đa, bit cờ của Timer sẽ được set lên mức 1 và TMR bị xóa, TMR=0.

Giả sử vi điều khiển dùng thạch anh tần số 4MHz, như vậy:

$$\textbf{\textit{Chu kỳ lệnh= 4 chu kì thạch anh=}}\frac{4}{4*10^6}\textbf{\textit{=1µs}}$$

Vậy TMR sẽ tự động tăng lên 1 đơn vị sau 1µs.

Nếu ban đầu ta cho TMR= x. Thì sau khoảng thời gian $2^{8n}-1-x$ ***(µs)*** giá trị TMR sẽ đạt giá trị tối đa của nó là $2^{8n}-1$. Thời điểm này được xác định thông qua trạng thái của bit cờ.

Ngược lại, ta muốn thực hiện định thời khoảng thời gian t sau một sự kiện 1 như sau:

- Sự kiện 1
- Tạo khoảng thời gian trễ t
- Sự kiện 2

Ta làm các bước:

- Sự kiện 1
- Gán giá trị ban đầu cho TMR = $2^{8n}-1-t$
- Kiểm tra bit cờ
- Khi bit cờ = 1, thực hiện sự kiện 2.

Thật vậy, sau 1 ***(µs)*** TMR tăng lên 1 đơn vị, để tăng giá trị cho TMR từ $2^{8n}-1-t$ đến giá trị tối đa $2^{8n}-1$ (khi bit cờ được set lên 1) mất $2^{8n}-1-(2^{8n}-1$ -t) = t ***(µs)***

Vậy khoảng thời gian từ sau sự kiện 1 (khi TMR bắt đầu được gán) đến sự kiện 2 (ngay sau khi bit cờ được set) là t đúng như yêu cầu của ta.

Ví dụ: Để định thời 200 ***(μs)*** dùng Timer0 (8 bit; n=1; giá trị tối đa là 255) ta cho TMR0= 255-200=55 rồi bắt đầu cho đếm lên.

3.2.2 Chế độ bộ đếm:

Khi được cài đặt trong này, một chân chức năng trên vi điều khiển sẽ trở thành chân đầu vào xung của bộ đếm. Ví dụ: chân RA4 đối với Timer0 và RC0 đối với Timer1. Hoạt động của nó có nét giống với chế độ định thời.

Khi được cài đặt hoạt động trong chế độ bộ đếm, Giá trị của thanh ghi TMR sẽ tự động tăng lên 1 đơn vị khi có một xung vào chân đầu vào xung của timer đó. Khi giá trị của TMR0 đạt đến giá trị tối đa, bit cờ của Timer sẽ được set lên mức 1 và TMR bị xóa, TMR=0.

Như vậy, về cách hoạt động trong chế độ này chỉ khác với chế độ định thời ở chỗ, thay vì TMR tự động tăng lên sau mỗi chu kì lệnh, thì TMR tăng lên khi có một xung đi vào chân đầu vào xung của Timer đó.

Dạng xung được xác định là sườn âm hay sườn dương phụ thuộc vào việc cài đặt bit chọn dạng xung tương ứng trên thanh ghi của vi điều khiển.

Nguyên lý hoạt động định thời và bộ đếm này cũng đúng với các bộ vi điều khiển, vi xử lý khác.

Ta đi vào khảo sát cụ thể 2 Timer0 và Timer1 của vi điều khiển PIC.

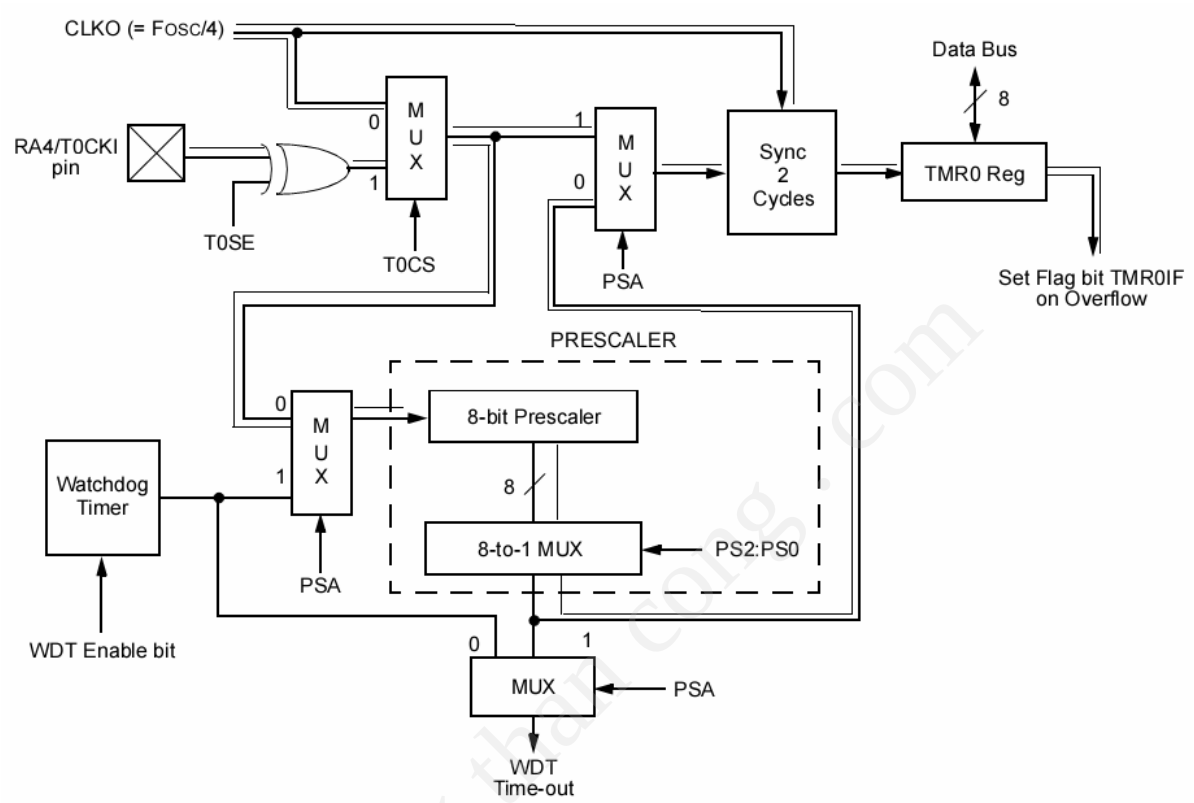
3.3 Timer 0:

3.3.1 Nguyên lý hoạt động:

Timer 0 có độ dài 8 bit – Thanh ghi chứa giá trị đếm là TMR0 (số đếm tối đa là 255). Hoạt động ở 2 chế độ là định thời và bộ đếm.

Đẻ hoạt động ở chế độ định thời, ta cho bit T0CS (bit 5 của thanh ghi Option_Reg) ở mức 0. Khi đó, giá trị của thanh ghi TMR0 sẽ tự động tăng lên 1 đơn vị sau mỗi chu kì lệnh của vi điều khiển.

Đẻ hoạt động ở chế độ bộ đếm, ta cho bit T0CS (bit 5 của thanh ghi Option_Reg) ở mức 1. Khi đó, giá trị của thanh ghi TMR0 sẽ tự động tăng lên 1 đơn vị sau khi có một xung đi vào chân RA4 của vi điều khiển. Việc chọn xung là dạng sườn lên hay sườn xung phụ thuộc vào bit T0SE (bit 4 của thanh ghi Option_Reg là 0 hay 1.



Khi giá trị TMR0 đạt đến giá trị tối đa 255 (0xFF) nó sẽ bị xóa về 0 đồng thời bit cờ TMR0IF (bit 5 của thanh ghi INTCON) được set lên 1, báo cho chương trình biết đã có sự tràn TMR.

Lưu ý là người lập trình phải xóa TMR0IF bằng chương trình (để ghi nhận đúng các lần tràn kế tiếp).

Ngoài ra, để tăng thời gian định thời hoặc số xung đếm đầu vào tối đa của Timer 0, vi điều khiển còn cho phép việc lựa chọn tỉ lệ Prescale cho đầu vào của Timer 0 bằng 3 bit PS2-PS0 (bit 2-0 của thanh ghi

Option_ Reg). Tỉ lệ như sau:

PS2:PS0: Prescaler Rate Select bits

Bit Value	TMR0 Rate	WDT Rate
000	1 : 2	1 : 1
001	1 : 4	1 : 2
010	1 : 8	1 : 4
011	1 : 16	1 : 8
100	1 : 32	1 : 16
101	1 : 64	1 : 32
110	1 : 128	1 : 64
111	1 : 256	1 : 128

Ta có thể hiểu đơn giản như sau:

Nếu tỉ lệ Prescale là 1:64 thì:

- Trong chế độ định thời, TMR0 sẽ tăng lên 1 đơn vị sau 64 chu kì lệnh
- Trong chế độ định thời, TMR0 sẽ tăng lên 1 đơn vị sau khi có 64 xung đi vào chân RA4

Như vậy, thời gian định thời tối đa cũng như số xung đếm được khi TMR0 tràn sẽ tăng lên 64 lần.

Tương tự với các tỉ lệ khác.

Thực ra, Timer 0 chia sẻ bộ chia tần số Prescale với một chế độ khác của vi điều khiển- chế độ Watchdog Timer. Việc chọn chế độ này được thực hiện khi cho bit PSA (bit 3 của thanh ghi Option_Reg) giá trị 0. Chế độ này sẽ được nghiên cứu sau. Ở đây ta chỉ quan tâm đến bộ chia tần số dành cho Timer 0 khi PSA=1. Vì vậy trong sơ đồ trên ta chỉ quan tâm đến các đường đến ghi chú thêm bằng đường thẳng mỏng.

3.3.2 Các thanh ghi liên quan:

Thanh ghi TMR0 (Địa chỉ 01h):

Chứa giá trị đếm hiện tại của Timer 0

Thanh ghi Option_Reg (Địa chỉ 81h):

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
$\overline{\text{RBPU}}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit 7							bit 0

T0CS: Bit chọn chế độ

= 0: TMR0 hoạt động chế độ định thời

= 1: TMR0 hoạt động chế độ bộ đếm

T0SE: Bit chọn dạng xung cho chế độ bộ đếm

= 0: Xung sườn lên

= 1: Xung sườn xuống

PSA: Bit chọn chế độ cho bộ chia tần số PresCale là WatchDog_Timer hay Timer 0.

= 0: Bộ chia tần số dành cho Timer 0

= 1: Bộ chia tần số dành cho Watch_Dog Timer

PS2-PS0: 3 bit chọn tỉ lệ chia tần số như đã giới thiệu ở phần trên

Thanh ghi INTCON (0Bh):

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF
bit 7							bit 0

Bit TMR0IE: Bit này bằng 1 cho phép ngắt Timer 0. Sự kiện ngắt xảy ra khi có sự tràn TMR0 từ 255 xuống 0.

Bit TMR0IF: Bit cờ xác nhận giá trị TMR0 đã bị tràn từ 255 về 0

3.3.3 Các lệnh CCS dùng cho thanh ghi Timer 0:

Lệnh cài đặt chế độ: Setup_timer_0 (chế độ|tỉ lệ chia tần số)

Chế độ có thể là một trong những từ sau:

- RTCC_INTERNAL: chế độ định thời
- RTCC_EXT_L_TO_H: chế độ bộ đếm với xung dạng sườn lên
- RTCC_EXT_H_TO_L: chế độ bộ đếm với xung dạng sườn xuống

Tỉ lệ chia tần số là một trong những từ sau:

- RTCC_DIV_2, RTCC_DIV_4, RTCC_DIV_8, RTCC_DIV_16, RTCC_DIV_32, RTCC_DIV_64, RTCC_DIV_128, RTCC_DIV_256: tương ứng với tỉ lệ chia

Lệnh gán giá trị đầu cho thanh ghi TMR0: Set_timer_0 (giá trị)

Giá trị nằm trong khoảng từ 0-255

Lệnh đọc giá trị hiện tại của thanh ghi TMR0: biến = Get_timer_0()

3.4 Timer 1:

3.4.1 Nguyên lý hoạt động:

Timer 1 có độ dài 16 bit. Giá trị của bộ đếm timer 1 được lưu trong 2 thanh ghi 8 bit TMR1H và TMR1L. Timer 1 cũng có 2 chế độ được cài đặt bởi bit TMR1CS (bit 1 của thanh ghi T1CON):

- TMR1CS = 0: Chế độ định thời
- TMR1CS = 1: Chế độ bộ đếm. Có 2 dạng được phân biệt bởi bit T1SYNC (bit 2 của T1CON)
    - T1SYNC = 0: Bộ đếm đồng bộ
    - T1SYNC = 1: Bộ đếm không đồng bộ

Trong chế độ bộ đếm, đầu vào cho xung có thể nối vào chân RC0 hoặc RC1 tùy thuộc vào bit T1OSCEN (bit 3 của T1CON)

- T1OSCEN = 1: Đầu vào xung nối với RC1
- T1OSCEN = 0: Đầu vào xung nối với RC0

Chú ý là khác với Timer 0, đầu vào xung ở Timer 1 phải có dạng sườn xuống.

Chế độ đồng bộ của chế độ bộ đếm nghĩa là nếu như vi điều khiển đang ở chế độ ngủ thì giá trị của thanh ghi TMR1L và TMR2L không tăng lên.

Tỉ lệ bộ chia tần số Prescal của timer 1 được chọn bởi 2 bit T1CKPS1-T1CKPS0 (Bit 5-4 của T1CON). Tỉ lệ chia lớn nhất là 1:8.

3.4.2 Các thanh ghi liên quan:

Thanh ghi TMR1L (Địa chỉ 0Eh) và TMR1H (Địa chỉ 0Fh):

Thanh ghi chứa 8 bit thấp và 8 bit cao của giá trị đếm 16 bit của timer 1.

Thanh ghi điều khiển Timer 1- T1CON (Địa chỉ 10h):

-	-	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC	TMR1CS	TMR1ON

T1CKPS1-T1CKPS0: Chọn tỉ lệ bộ chia tần số:

T1CKPS1-T1CKPS0	Tỉ lệ chia tần số
00	1:1
01	1:2
10	1:4
11	1:8

T1OSCEN: bit chọn đầu vào xung là RC1 hay RC0

- =1: RC1
- =0: RC0

T1SYNC: bit chọn chế độ bộ đếm đồng bộ

- =1: Không đồng bộ
- =0: Không đồng bộ

TMR1CS: bit chọn chế độ định thời hay chế độ bộ đếm

- =1: Chế độ bộ đếm
- =0: Chế độ bộ định thời

TMR1ON: bit bật hay tắt Timer 1

- =1: Bật Timer 1
- =0: Tắt Timer 1

Thanh ghi PIR1:

Kqt	Kqt	kqt	kqt	Kqt	kqt	kqt	TMR1IF

Bit TMR1IF là bit cờ của timer 1. Bit được set lên giá trị 1 khi xảy ra tràn 2 thanh ghi TMR1L và TMR2H từ 65535 về 0.

3.4.3 Các lệnh CCS liên quan đến Timer 1:

Lệnh cài đặt chế độ: Setup_timer_1 (chế độ|tỉ lệ chia tần số)

Chế độ có thể là một trong những từ sau:

- T1_DISABLE: không sử dụng, tắt Timer 1
- T1_INTERNAL: chế độ định thời
- T1_EXTERNAL: chế độ bộ đếm không đồng bộ
- T1_EXTERNAL_SYNC: chế độ bộ đồng bộ

Tỉ lệ chia tần số là một trong những từ sau:

- T1_DIV_1, T1_DIV_2, T1_DIV_4, T1_DIV_8: tương ứng với tỉ lệ chia

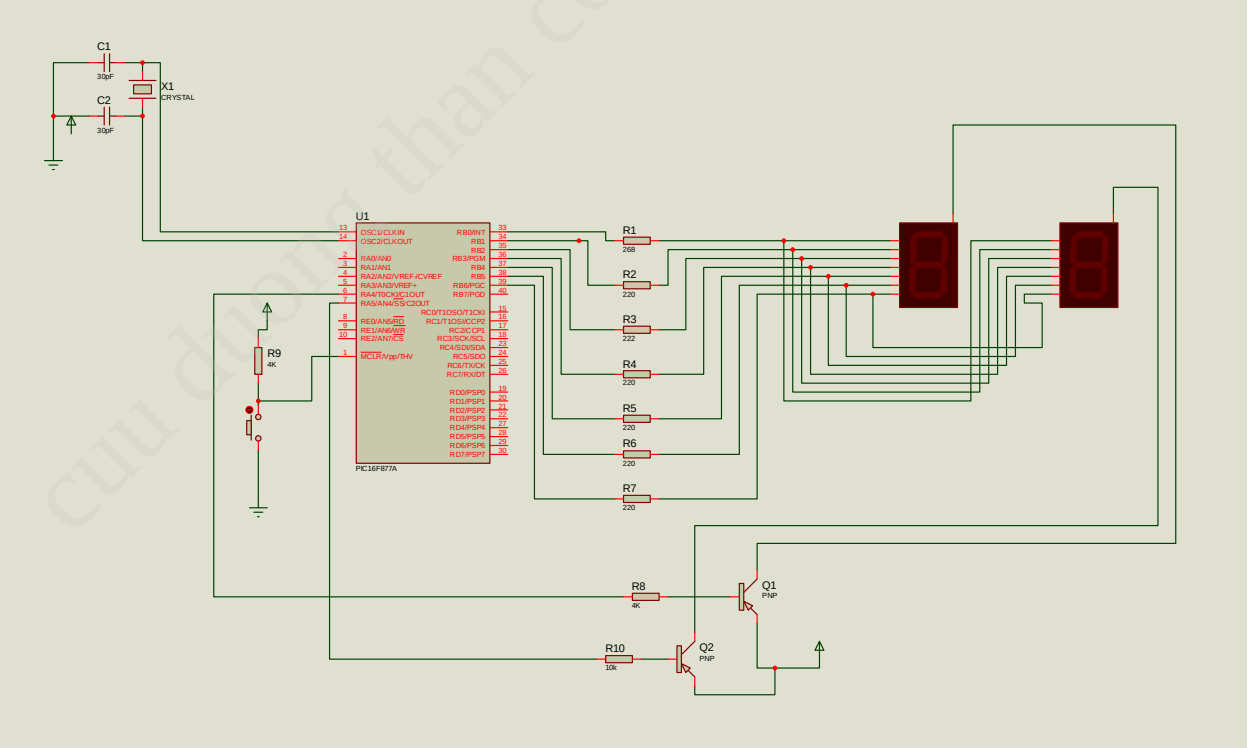
Lệnh gán giá trị đầu cho cặp thanh ghi TMR1L và TMR1H: Set_timer_1 (giá trị)

Giá trị nằm trong khoảng từ 0-65535

Lệnh đọc giá trị hiện tại của cặp thanh ghi TMR1L và TMR1H: biến = Get_timer_1()

BÀI TẬP:

3.1 Cho sơ đồ mạch như sau:

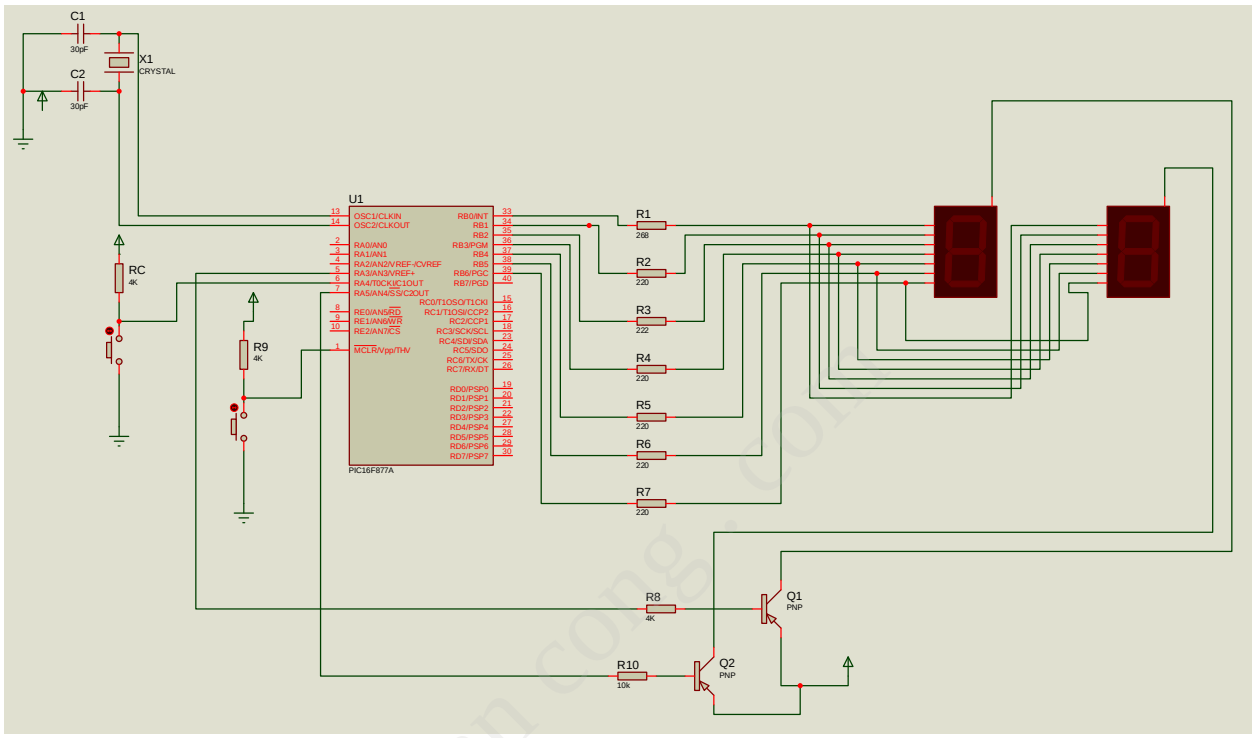


a. 2 led 7 đoạn hiển thị số đếm, giá trị số đếm tự động tăng lên từ 0-99 sau khoảng thời gian định thời bởi Timer 0

b. Tăng thời gian định thời bằng cách sử dụng Timer 1

c. Định thời 1 s dùng Timer 1

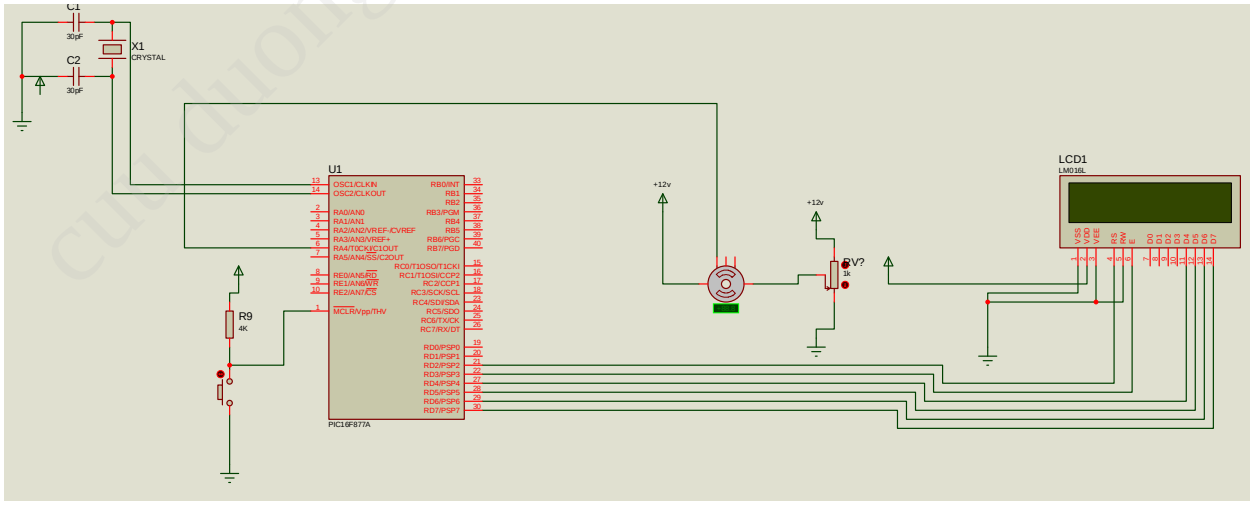
3.2 Cho sơ đồ mạch như sau:



Trong đó phím bấm nối với chân đầu vào xung của Timer 0, RA4.

2 led bảy đoạn hiển thị số lần bấm phím

3.3 Cho sơ đồ mạch đo tốc độ động cơ như sau:



Động cơ được thay đổi tốc độ nhờ biến trở. Tốc độ động cơ được đo bởi encoder nằm ngay trong động cơ và tín hiệu được truyền tới chân RA4, chân đầu vào xung của Timer 0. Dùng Timer 0 chế độ bộ đếm, Timer 1 chế độ định thời để đọc tốc độ động cơ.

Giá trị đọc được hiển thị lên LCD.

BÀI 4: NGẮT

4.1 Ngắt là gì:

Ngắt hiểu theo nghĩa đơn giản là các sự kiện ngẫu nhiên làm gián đoạn quá trình của một sự kiện đang xảy ra. Để có thể dễ hiểu khái niệm mới này ta cùng đưa ra một ví dụ trong thực tế như sau:

Ví dụ: Trong giờ học trên lớp, ta đang học bài, có chuông điện thoại hoặc có bạn gọi, ta phải dừng hoạt động học bài lại để trả lời điện thoại hoặc ra gặp bạn. Sự kiện điện thoại reo chuông, hay bạn bè gọi được gọi là *sự kiện ngắt*, việc ta trả lời điện thoại hay ra gặp bạn là *chương trình phục vụ ngắt.* Việc đang học bài được xem là *chương trình chính.*

Ngắt được thực hiện khi và chỉ khi cài đặt cho phép nó. Như trong ví dụ trên, nếu sự kiện ngắt-điện thoại reo xảy ra, nếu giáo viên và bản thân ta cho phép mình trả lời điện thoại khi đang học bài thì khi có điện thoại ta mới nghe.

Vi điều khiển cũng có ngắt. Cách xử lý của nó cũng tương tự như ví dụ trên.

Cụ thể hoạt động của vi điều khiển khi có sự kiện ngắt xảy ra và ngắt đó đã được cho phép:

- *Thực hiện nốt lệnh đang thực hiện*
- *Dừng chương trình đang thực hiện*
- *Lưu lại địa chỉ của lệnh kế tiếp trong chương trình đang thực hiện vào bộ nhớ stack*
- *Nhảy tới địa chỉ 0x04 trong bộ nhớ chương trình*
- *Tại đây, vi điều khiển sẽ thực hiện chương trình con phục vụ ngắt do người lập trình đã lập trình từ trước.*
- *Sau khi thực hiện xong chương trình con phục vụ ngắt, vi điều khiển lấy lại địa chỉ của lệnh kế tiếp đã được lưu và thực hiện tiếp chương trình đang thực hiện dở lúc chưa có ngắt*

Như vậy, cách phản ứng của vi điều khiển là khá tương đồng với cách xử lý của con người trong thực tế. Như trong ví dụ trên, khi ta đang học bài, khi có ngắt, tức có điện thoại-sự kiện ngắt, ta đọc nốt từ cuối cùng, nhớ dòng đang đọc ở trang thứ mấy, đánh dấu, trả lời điện thoại (chương trình con phục vụ ngắt), trả lời xong ta trở lại học bài ở dòng, trang đã được đánh dấu.

Tới đây ta tổng hợp lại các thuật ngữ dùng cho xử lý ngắt trong vi điều khiển:

- **Nguồn ngắt:** nguồn ngắt là nguyên nhân gây ra ngắt. Như trong ví dụ trên, nguồn ngắt có thể

  Là điện thoại gọi hoặc bạn gọi

- **Sự kiện ngắt:** khi nguồn ngắt xảy ra
- **Chương trình con phục vụ ngắt:** là chương trình vi điều khiển xử lý khi có sự kiện ngắt xảy ra do người lập trình lập trình ra
  Ví dụ như ta trả lời hoặc chạy ra khỏi phòng gặp bạn
- **Vecto ngắt:** tức địa chỉ 0x04 nơi vi điều khiển chạy tới sau khi lưu địa chỉ trả về
- **Bit cho phép ngắt:** tức việc cho phép vi điều khiển chạy chương trình con phục vụ ngắt khi có sự kiện ngắt xảy ra. Trong vi điều khiển PIC, mỗi ngắt có bit cho phép của nó. Bit này tận cùng bằng chữ E (enable), nằm trong các thanh ghi chuyên dụng. Muốn cho phép ngắt đó, ta phải đưa bit cho phép ngắt tương ứng lên giá trị 1. Ngắt chỉ thực sự được cho phép ngắt khi ta cho bit cho phép ngắt toàn cục GIE (Global Interrupt Enable) lên mức 1. Ta hình dùng như sau: khi có sự kiện ngắt- điện thoại gọi, nếu ta cho phép mình nghe điên thoại (tức bit cho phép ngắt của ngắt đó được set lên 1) đồng thời thầy giáo cho phép (bit cho phép ngắt toàn cục GIE được lên mức 1) thì ta mới nghe điện thoại (cho chương trình con phục vụ ngắt hoạt động).
  Một số các ngắt khác, như các ngắt ngoại vi bao gồm ADC, PWM v.v Muốn cho phép nó còn phải đưa bit cho phép ngắt ngoại vi lên mức 1.
- **Cờ ngắt:** là bit phản ánh trạng thái của sự kiện ngắt. Mỗi ngắt có một bit cờ. Khi bit cờ này bằng 1 nghĩa là sự kiện ngắt tương ứng với cờ đó xảy ra. Ta hình dung như tiếng chuông của điện thoại là cờ ngắt, chuông rung báo có sự kiện ngắt- có điện thoại xảy ra. Các bit này tận cùng bằng từ F (Flag- cờ). Lưu ý là dù một ngắt có được cho phép hay không thì cờ ngắt vẫn được set lên 1 khi có sự kiện ngắt xảy ra. (Dù ta có được phép nghe điện thoại hay không thì chuông điện thoại vẫn cứ reo).

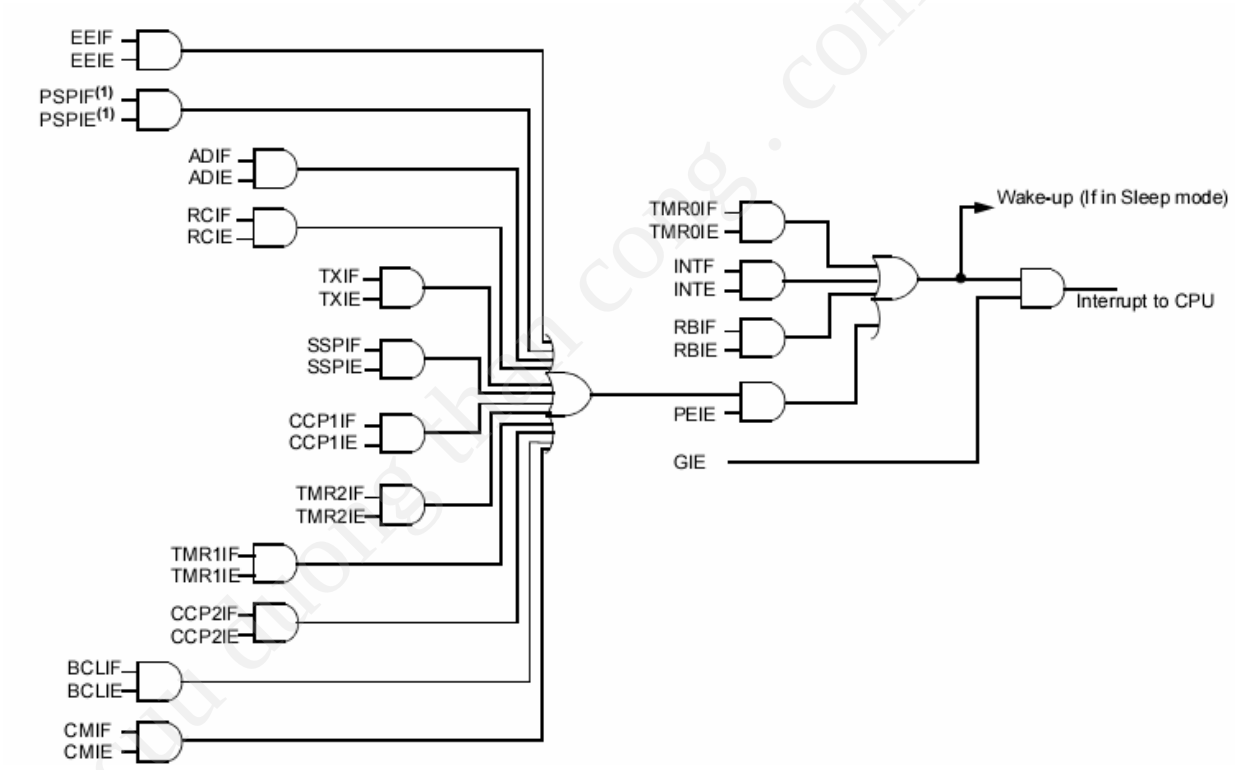
4.2 Các ngắt trong vi điều khiển PIC16F877A:

Vi điều khiển PIC16F877A có 15 nguồn ngắt. Được chia làm 2 lớp ngắt:

- **Lớp ngắt cơ bản:** bao gồm các ngắt cơ bản như ngắt tràn timer 0, ngắt ngoài, ngắt thay đổi trạng thái của các chân PortB (RB4-RB7). Bit cho phép ngắt và bit cờ tương ứng là

TMR0IE,TMR0IF; INTE, INTF; RBIE và RBIF. Để ý là để cho phép ngắt thực sự xảy ra phải có bit cho phép ngắt toàn cục GIE.

- **Lớp ngắt ngoại vi:** bao gồm các ngắt ngoại vi như ngắt tràn timer 1 (TMR1IE, TMR1IF), ngắt tràn Timer 2(TMR2IE, TMR2IF), ngắt hoàn thành việc chuyển đổi ADC (ADCIE, ADCIF), ngắt hoàn thành việc nhận kí tự trong truyền thông RS232 (RCIE, RCIF), ngắt hoàn thành việc truyền kí tự trong truyền thông RS232 (TXIE, TXIF) v.v Để ý là muốn thực sự cho phép các ngắt này ngoài bit cho phép ngắt toàn cục được set phải set cả bit cho phép ngắt ngoại vi PEIE.



4.3 Viết chương trình xử lý ngắt bằng CCS:

Ví dụ một chương trình có ngắt như sau:

```
#INCLUDE <16F877A.H>

#FUSES NOLVP, NOWDT, HS

#USE DELAY(CLOCK=8000000)

BYTE CONST MAP[10] = {0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,0x80,0x90};
```

```
VOID HIENTHI(INT A);
INT SODEM;
//CHUONG TRINH CON PHUC VU NGAT DAT SAU #INT_EXT
#INT_EXT
VOID NGATNGOAI()
{
  // XOA CO NGAT NGOAI
  CLEAR_INTERRUPT(INT_EXT);
  // CAM NGAT TRONG CHUONG TRINH CON PHUC VU NGAT
  DISABLE_INTERRUPTS(GLOBAL);
  SODEM++;
  // CHO PHEP NGAT TOAN CUC
  ENABLE_INTERRUPTS(GLOBAL);
}
VOID HIENTHI(INT A)
{
  INT HC, HDV;
  HC=A/10;
  HDV=A%10;
  OUTPUT_LOW(PIN_A4);
  OUTPUT_D(MAP[HC]);
  DELAY_MS(15);
```

```
  OUTPUT_HIGH(PIN_A4);
  OUTPUT_LOW(PIN_A5);
  OUTPUT_D(MAP[HDV]);
  DELAY_MS(15);
  OUTPUT_HIGH(PIN_A5);
}
VOID MAIN()
{
    // CAI DAT VAO RA CHO CONG B
    SET_TRIS_B(0xFF);
    // CHO PHEP NGAT NGOAI
    ENABLE_INTERRUPTS(INT_EXT);
    // CAI DAT SUON NGAT
    EXT_INT_EDGE(H_TO_L);
    // CHO PHEP NGAT TOAN CUC
    ENABLE_INTERRUPTS(GLOBAL);
   // VONG LAP DOI NGAT
    WHILE(1)
    {
    HIENTHI(SODEM);
    }
 }
```

Như vậy, lúc viết chương trình có dùng ngắt bằng CCS, ta có những lưu ý sau:

- Trong chương trình chính (main), chúng ta cài đặt ngắt: cho phép ngắt cụ thể, cho phép ngắt toàn cục. Đợi ngắt
- Chương trình con xử lý ngắt là chương trình con nằm ngay sau chỉ thị biên dịch #INT_XXX, trong đó XXX là tên của ngắt cụ thể. Ví dụ: #INT_EXT: ngắt ngoài
- Trong chương trình con xử lý ngắt: xóa cờ ngắt, cấm ngắt toàn cục đề phòng khi đang xử lý ngắt có ngắt xảy ra. Sau khi xử lý dữ liệu trong chương trình con xử lý ngắt, ta cho phép ngắt toàn cục lại.

Tên một số ngắt của PIC như sau:

- INT_EXT: ngắt ngoài
- INT_TIMER0: ngắt timer 0
- INT_TIMER1: ngắt timer 1
- INT_TIMER2: ngắt timer 2
- INT_RDA: ngắt nhận đủ kí tự trong truyền thông máy tính
- INT_RB: ngắt thay đổi trạng thái các chân RB7-RB4

Trong phần tiếp theo, ta sẽ khảo sát một số ngắt tiêu biểu như ngắt ngoài INT, ngắt thay đổi trạng thái các chân cao PORTB, ngắt tràn Timer 0, ngắt tràn Timer 1. Các ngắt ngoại vi khác sẽ được nhắc đến khi nghiên cứu các modun ngoại vi này.

4.4 Ngắt ngoài:

4.4.1 Hoạt động:

- *Nguồn ngắt:* là xung đi vào chân RB0 của vi điều khiển PIC
- *Sự kiện ngắt:* sự kiện ngắt xảy ra khi có xung đi vào chân RB0 của vi điều khiển. Xung là xung sườn dương hay sườn âm phụ thuộc bit cài đặt chọn dạng xung, bit INTEDG (bit 6 của thanh ghi PTION_REG) là 1 hay không.
- *Bit cho phép ngắt:* Để cho phép ngắt ngoài, bit cho phép ngắt ngoài INTIE (bit 4 của thanh ghi INTCON) phải được set lên 1. Ngoài ra, bit cho phép ngắt toàn cục GIE (bit 7 của thanh ghi INTCON) cũng phải được set lên 1.
- *Cờ ngắt:* bit cờ ngắt ngoài là bit INTIF (bit 1 của thanh ghi INTCON) được tự động set lên 1 khi có sự kiện ngắt ngoài xảy ra. Cờ này phải được xóa bằng chương trình (cụ thể là

trong chương trình con phục vụ ngắt) để vi điều khiển quản lý chính xác các lần ngắt kế tiếp.

4.4.2 Quản lý ngắt ngoài trong chương trình CCS:

Trong chương trình chính, cài đặt ngắt:

- Cài đặt chân RB0 là chân vào: SET_TRIS_B(0x01)
- Cài đặt dạng xung đầu vào là sườn dương hay sườn âm: EXT_INT_EDGE(H_TO_L) hoặc EXT_INT_EDGE(L_TO_H)
- Cho phép ngắt ngoài: ENABLE_INTERRUPTS(INT_EXT)
- Cho phép ngắt toàn cục: ENABLE_INTERRUPTS(GLOBAL)

Chương trình con phục vụ ngắt đặt sau chỉ định biên dịch #INT_EXT:

#INT_EXT

Định nghĩa chương trình con

Trong chương trình con phục vụ ngắt:

- Xóa cờ ngắt: CLEAR_INTERRUPT(INT_EXT)
- Cấm ngắt toàn cục, đề phòng lúc đang xử lý ngắt, lại có ngắt xảy ra:
  DISABLE_INTERRUPTS(GLOBAL)
- Xử lý ngắt đó: tùy thuộc vào ý đồ của người lập trình
- Cho phép ngắt toàn cục: ENABLE_INTERRUPTS(GLOBAL)

4.5 Ngắt Thay Đổi Trạng Thái Các Chân RB7-RB4:

4.5.1 Hoạt động:

- *Nguồn ngắt:* là trạng thái của một trong các chân RB7-RB4 của vi điều khiển PIC
- *Sự kiện ngắt:* sự kiện ngắt xảy ra khi có sự thay đổi trạng thái (1-0 hay 0-1) của một trong các chân RB7-RB4 của PortB
- *Bit cho phép ngắt:* Để cho phép ngắt này, bit cho phép ngắt RBIE (bit 3 của thanh ghi INTCON) phải được set lên 1. Ngoài ra, bit cho phép ngắt toàn cục GIE (bit 7 của thanh ghi INTCON) cũng phải được set lên 1.

- *Cờ ngắt:* bit cờ ngắt ngoài là bit RBIF (bit 0 của thanh ghi INTCON) được tự động set lên 1 khi có sự kiện ngắt ngoài xảy ra. Cờ này phải được xóa bằng chương trình (cụ thể là trong chương trình con phục vụ ngắt) để vi điều khiển quản lý chính xác các lần ngắt kế tiếp.

Lưu ý quan trọng: Khi sử dụng ngắt này trong các ứng dụng xử lý các xung đầu vào RB4-RB7, ví dụ như phím bấm chẳng hạn, ta cần lưu ý điểm sau. Giả sử như ban đầu phím bấm chưa bấm, đầu vào RB ở mức 1, khi bấm phím RB xuống mức 1, như vậy có 1 sự kiện ngắt xảy ra. Tuy nhiên, khi thả phím bấm ra, RB lên mức 1, tức cũng có một sự thay đổi trạng thái của chân RB7-RB4, nên cũng có ngắt xảy ra. Vì vậy, việc ấn và nhả phím bấm được tính 2 lần ngắt. Cần để ý điều này khi lập trình.

Vấn đề thứ 2 là cần sử dụng một lệnh đọc cổng B để loại bỏ trạng thái mismatch lúc xảy ra ngắt ở các chân này.

4.5.2 Quản lý ngắt RB trong chương trình CCS:

Trong chương trình chính, cài đặt ngắt:

- Cài đặt chân RB4-RB7 là chân vào: SET_TRIS_B(0xF0)
- Cho phép ngắt RB: ENABLE_INTERRUPTS(INT_RB)
- Cho phép ngắt toàn cục: ENABLE_INTERRUPTS(GLOBAL)

Chương trình con phục vụ ngắt đặt sau chỉ định biên dịch #INT_RB:

#INT_RB

Định nghĩa chương trình con

Trong chương trình con phục vụ ngắt:

- Xóa cờ ngắt: CLEAR_INTERRUPT(INT_RB)
- Cấm ngắt toàn cục, đề phòng lúc đang xử lý ngắt, lại có ngắt xảy ra:
  DISABLE_INTERRUPTS(GLOBAL)
- Đọc cổng B để loại bỏ trạng thái mismatch
- Xử lý ngắt đó: tùy thuộc vào ý đồ của người lập trình, chú ý đến số lần ngắt
- Cho phép ngắt toàn cục: ENABLE_INTERRUPTS(GLOBAL)

4.6 Ngắt Timer 0:

4.6.1 Hoạt động:

- *Nguồn ngắt:* là trạng thái tràn của thanh ghi bộ đếm timer 0, TMR0 vi điều khiển PIC
- *Sự kiện ngắt:* sự kiện ngắt xảy ra khi có sự tràn của TMR0, tức là khi TMR0=255 rồi bị xóa
- *Bit cho phép ngắt:* Để cho phép ngắt này, bit cho phép ngắt TMR0IE (bit 5 của thanh ghi INTCON) phải được set lên 1. Ngoài ra, bit cho phép ngắt toàn cục GIE (bit 7 của thanh ghi INTCON) cũng phải được set lên 1.
- *Cờ ngắt:* bit cờ ngắt ngoài là bit TMR0IF (bit 2 của thanh ghi INTCON) được tự động set lên 1 khi có sự kiện ngắt ngoài xảy ra. Cờ này phải được xóa bằng chương trình (cụ thể là trong chương trình con phục vụ ngắt) để vi điều khiển quản lý chính xác các lần ngắt kế tiếp.

4.6.2 Quản lý ngắt Timer 0 trong chương trình CCS:

Trong chương trình chính, cài đặt ngắt:

- Gán giá trị ban đầu cho thanh ghi TMR0, tùy thuộc vào thời gian mà người lập trình muốn (xem lại bài 3): SET_TIMER0(giá trị)
- Cho phép ngắt timer 0: ENABLE_INTERRUPTS(INT_TIMER0)
- Cho phép ngắt toàn cục: ENABLE_INTERRUPTS(GLOBAL)

Chương trình con phục vụ ngắt đặt sau chỉ định biên dịch #INT_TIMER0:

#INT_TIMER0

Định nghĩa chương trình con

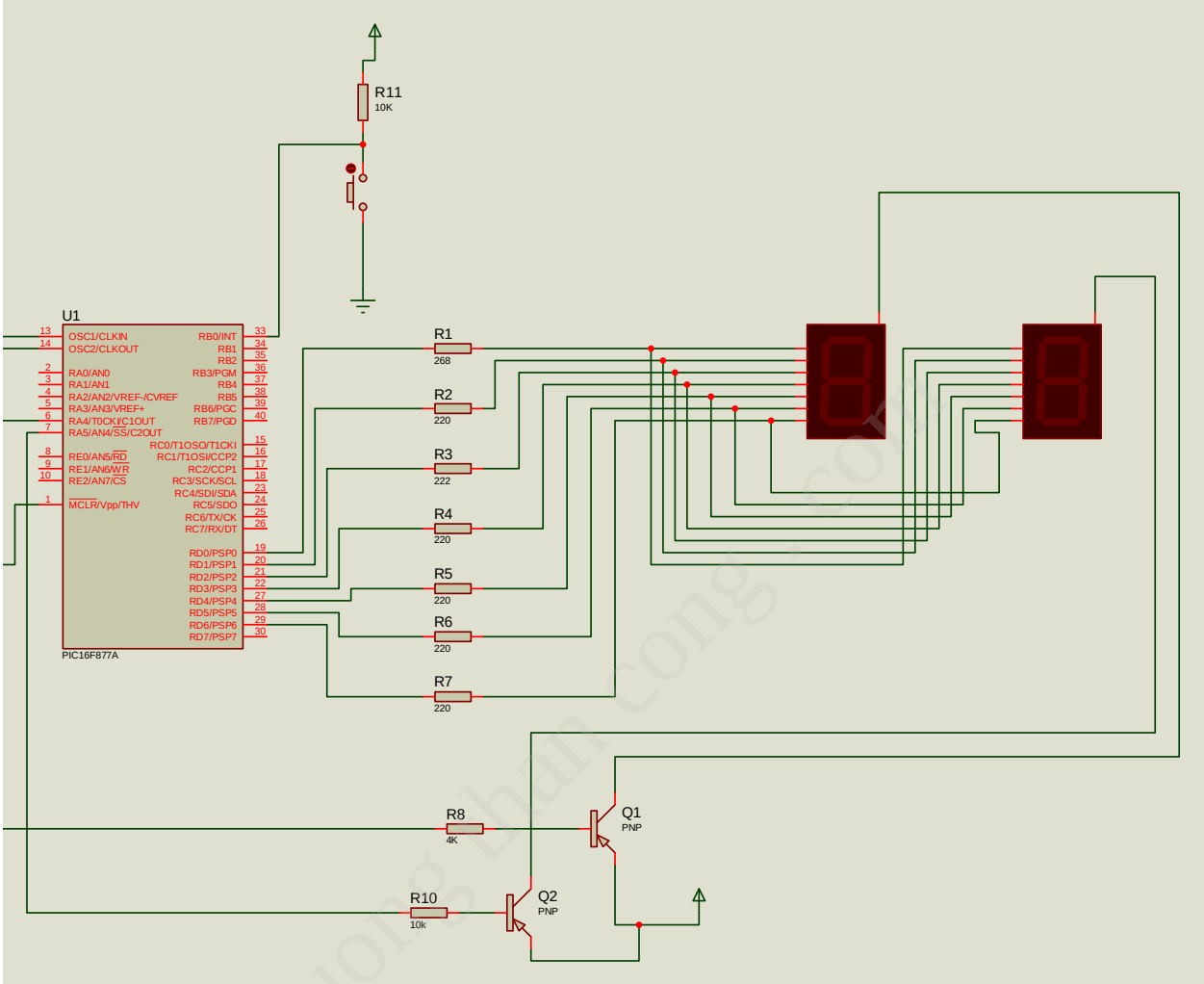
Trong chương trình con phục vụ ngắt:

- Xóa cờ ngắt: CLEAR_INTERRUPT(INT_TIMER0)
- Cấm ngắt toàn cục, đề phòng lúc đang xử lý ngắt, lại có ngắt xảy ra:
  DISABLE_INTERRUPTS(GLOBAL)
- Xử lý ngắt đó: tùy thuộc vào ý đồ của người lập trình, chú ý đến số lần ngắt
- Gán lại giá trị ban đầu cho thanh ghi TMR0 (tùy thuộc vào ý đồ của người lập trình): SET_TIMER0(giá trị)

- Cho phép ngắt toàn cục: ENABLE_INTERRUPTS(GLOBAL)

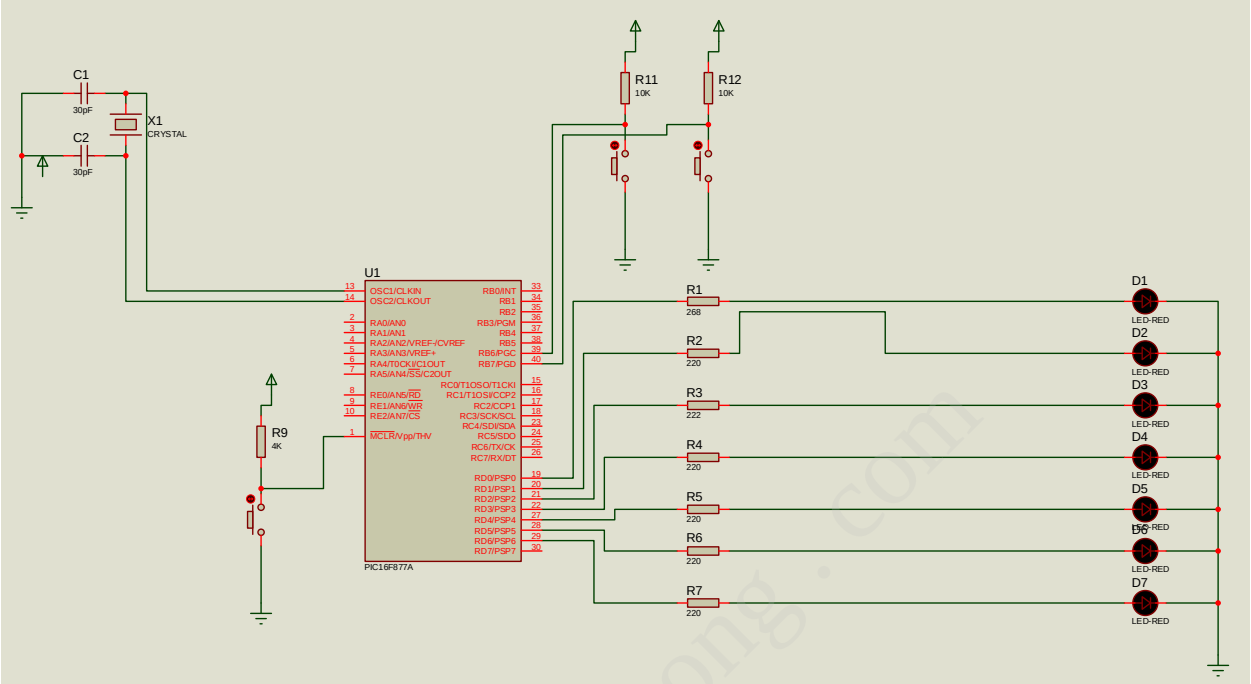
BÀI TẬP:

4.1 Cho sơ đồ mạch như hình vẽ:



Lập trình để mạch hoạt động như sau: 2 led 7 đoạn hiển thị số lần phím bấm

4.2 Cho sơ đồ mạch như sau:

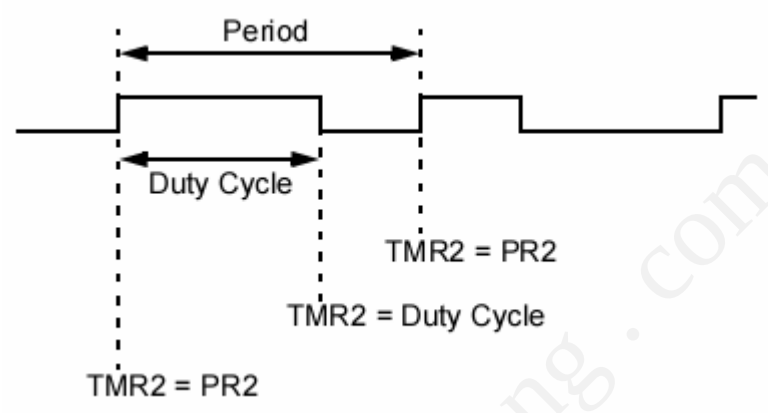


2 led 7 đoạn hiển thị số lần phím bấm

4.3 Làm lại các bài tập 3.1, 3.2, 3.3 nhưng sử dụng ngắt.

BÀI 5: ĐIỀU CHẾ ĐỘ RỘNG XUNG-PWM

5.1 Nguyên lý hoạt động:



Bộ điều chế độ rộng xung tạo xung hình chữ nhật trên 2 chân RC1/CCP2 và RC2/CCP1- giá trị xung ở 2 chân là ngược nhau (bù nhau). Thực ra đây là một chức năng của modun CCP gồm 3 chức năng: Comparation, Capture, PWM.

Nguyên lý tạo xung như sau:

Khi Thanh ghi bộ đếm của bộ định thời Timer 2 đạt giá trị bằng giá trị của thanh ghi PR2, đầu ra xung RC2/CCP1 được set lên mức cao. TMR2 được reset về 0, sau đó đếm lên, khi TMR2 đạt giá trị bằng độ rộng xung, chân RC2/CCP1 được reset về 0. TMR2 tiếp tục đếm lên cho đến khi bằng giá trị PR2 thì chu trình sẽ lặp lại như lúc đầu. Xung ra ở chân RC1/CC21 là bù của xung trên chân RC2/CCP1 .

5.2 Chu kỳ xung:

Để xác định chu kỳ xung ta đưa ra phân tích như sau:

Như đã tìm hiểu về cách làm việc của các bộ timer, ta biết: sau mỗi chu kì lệnh giá trị của TMR2 sẽ tăng lên 1 đơn vị. Nếu dùng bộ chia tần số, giả sử là 1: N thì sau N chu kì lệnh giá trị của TMR2 mới tăng lên 1 đơn vị.

Mỗi chu kì lệnh gồm 4 chu kì xung.

Chu kì xung chính là khoảng thời gian từ lúc TMR2=0 cho đến khi TMR2=PR2. Suy ra, để tăng PR2 đơn vị, hay chính là chu kì xung sẽ bằng:

$$Tpwm = (PR2 + 1) * 4 * Tosc * N$$

Ở đây, Tosc là chu kì xung của vi điều khiển hay chu kì của thạch anh

N là tỉ lệ chia tần số.

Ví dụ: với thạch anh Fosc=4Mhz, N=4, ta có chu kì xung tối đa đạt được là khi PR2=255 (PR2 là thanh ghi 8 bit)

T=(255+1)*4*(1/4Mhz)*4=1024µs

Tức tần số xung là: F=1/T=976Hz= 1KHz.

5.3 Độ rộng xung:

Độ rộng xung là giá trị 10 bit : trong đó 8 bit cao được ghi vào thanh ghi CCPR1L, và 2 bit thấp ghi vào 2 bit 5 và 4 của thanh ghi CCP1CON :

Độ rộng xung PWM được xác định theo công thức:

PWM Duty Cycle= (CCPR1L:CCP1CON<5:4>)*Tosc*N

Trong đó, CCPR1L là thanh ghi 8 bit, CCP1CON<5:4> là 2 bit 5 và 4 của thanh ghi điều khiển CCP1CON. N là tỉ lệ chia tần số.

5.4 Qui trình thực hiện xuất xung PWM:

Gồm các bước:

- Cài đăt modun CCP chức năng PWM: **setup_CCP1(CCP_PWM)**
- Cài đặt Timer 2: **setup_timer_2 (mode, period, 1)**

  Trong đó, mode có thể là: T2_DISABLED, T2_DIV_BY_1, T2_DIV_BY_4, T2_DIV_BY_16, nghĩa là tắt Timer2 hoặc định tỉ lệ bộ chia tần số

Period chính là giá trị của thanh ghi PR2 để xác định chu kì của xung

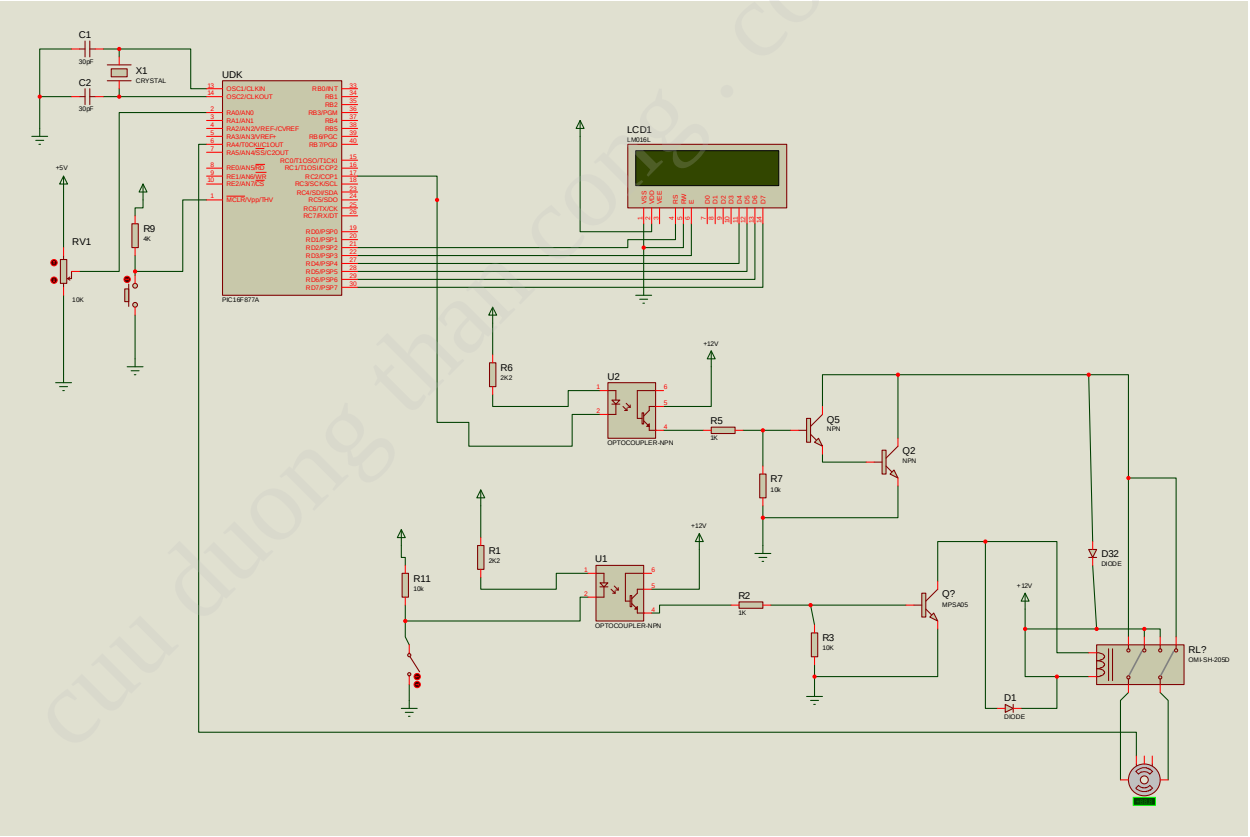
- Cài đặt độ rộng xung: **set_pwm1_duty(value);**

Trong đó, value là giá trị CCPR1L:CCP1CON<5:4>

Và độ rộng xung được tính theo công thức:

PWM Duty Cycle= value*Tosc*N

5.5 Ứng dụng PWM trong điều khiển tốc độ động cơ 1 chiều:



BÀI TẬP:

1. Lập trình để xuất xung PWM có tần số là 1KHZ, độ rộng xung là 50%
2. Ứng dụng PWM

CHƯƠNG TRÌNH BÀN PHÍM

```
#include <16F877A.h>

#fuses HS,NOWDT,NOPROTECT,NOLVP
#device *=16 adc=10
#use delay(clock=4000000)
#include <lcd_lib_4bit.c>
//#INCLUDE <KBD.C>
#define col0 PIN_B4
#define col1 PIN_B5
#define col2 PIN_B6
#define col3 PIN_B7
#define row0 PIN_B0
#define row1 PIN_B1
#define row2 PIN_B2
#define row3 PIN_B3

// Keypad layout:
char const KEYS[4][4] = {{'0','1','2','3'},
                         {'4','5','6','7'},
                         {'8','9','A','B'},
                         {'C','D','E','F'}
                        };

#define KBD_DEBOUNCE_FACTOR 100    // Set this number to apx n/333 where
                                   // n is the number of times you expect
```

```
                    // to call kbd_getc each second



#SEPARATE void kbd_init() {
}



short int ALL_ROWS (void)
{
  if (input (row0) & input (row1) & input (row2) & input (row3))
    return (0);
  else
    return (1);
}



char kbd_getc( ) {
  static byte kbd_call_count;
  static short int kbd_down;
  static char last_key;
  static byte col;

  byte kchar;
  byte row;
```

```
kchar='\0';
if(++kbd_call_count>KBD_DEBOUNCE_FACTOR) {
   switch (col) {
    case 0   :
         output_low(col0);

         output_high(col1);
         output_high(col2);
         output_high(col3);
         //DELAY_MS(1000);
            break;
    case 1   : output_high(col0);
         output_low(col1);
         output_high(col2);
         output_high(col3);
            break;
    case 2   : output_high(col0);
         output_high(col1);
         output_low(col2);
         output_high(col3);
            break;
    case 3   : output_high(col0);
         output_high(col1);
         output_high(col2);
         output_low(col3);
```

```
            break;
}

if(kbd_down)//NEU LAN TRUOC DO DA CO MOT PHIM DUOC BAM
{
 if(!ALL_ROWS())//DOI CHO DEN KHI PHIM DUOC NHA RA
 {
  kbd_down=false;
  kchar=last_key;
  last_key='\0';
 }
}
else //NEU KHONG CO PHIM NAO BAM TRUOC DO
{
  if(ALL_ROWS())//NEU CO PHIM BAM
  {
    if(!input (row0))
      row=0;
    else if(!input (row1))
      row=1;
    else if(!input (row2))
      row=2;
    else if(!input (row3))
      row=3;
    last_key =KEYS[row][col];
    kbd_down = true;
```

```
        }
        else // NEU KHONG CO PHIM NAO BAM, CHUYEN SANG QUET COT TIEP THEO
        {
          ++col;
          if(col>=4)
            col=0;
        }
      }
    kbd_call_count=0;//SAU 1 LAN QUET RESET KBD_CALL_COUNT VE 0, DE DOI LAN QUET TIEP
  }
 return(kchar);
}
void main()
{
INT C;
INT16 value;
SET_TRIS_D(0X00);
SET_TRIS_B(0X0F);
    lcd_putcmd(0x80);
    lcd_init();
     delay_ms(200);
OUTPUT_B(0XF0);


  PRINTF(LCD_PUTCHAR,"PHIM BAM: ");
```

```
  while( TRUE )

  {
   C=KBD_GETC();
   IF (C!='\0')
   {
   LCD_PUTCMD(0X01);
   PRINTF(LCD_PUTCHAR,"%C",C);
   }

}

}
```

Chương trình nhập số:

```
void main()
{
INT C;
INT16 DAT_THU;
INT16 value;
SET_TRIS_D(0X00);
SET_TRIS_B(0X0F);
lcd_putcmd(0x80);
lcd_init();
```

```
delay_ms(200);


 PRINTF(LCD_PUTCHAR,"PHIM BAM: ");
  WHILE( TRUE )


  {
   DAT_THU=0;
    DO
    {
    C=KBD_GETC();
     IF (C!='\0'& c!='A'&C!='B'& c!='C'&C!='D'& c!='D'&C!='E'&C!='F')
        {
                                    DAT_THU =DAT_THU*10;
                                    DAT_THU=DAT_THU+C-0X30;
                                      LCD_PUTCHAR(C);
        }
    }WHILE(C!='E');
   LCD_PUTCMD(0X01);
   PRINTF(LCD_PUTCHAR,"GIA TRI PHIM BAM:");
   LCD_PUTCMD(0XC2);
   PRINTF(LCD_PUTCHAR,"%6LU",DAT_THU);
```