

ĐẠI HỌC QUỐC GIA TP.HCM
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH



**THÍ NGHIỆM
VI XỬ LÝ - VI ĐIỀU KHIỂN**

BM Kỹ thuật Máy tính

2016

Bài 1 : Giới thiệu MPLAB IDE và KIT PIC

Nội dung :

- Nd1. Tạo project trên MPLAB IDE.
- Nd2. Viết chương trình ASM.
- Nd3. Dịch và nạp chương trình vào vi điều khiển PIC.
- Nd4. Chạy và gỡ rối chương trình.

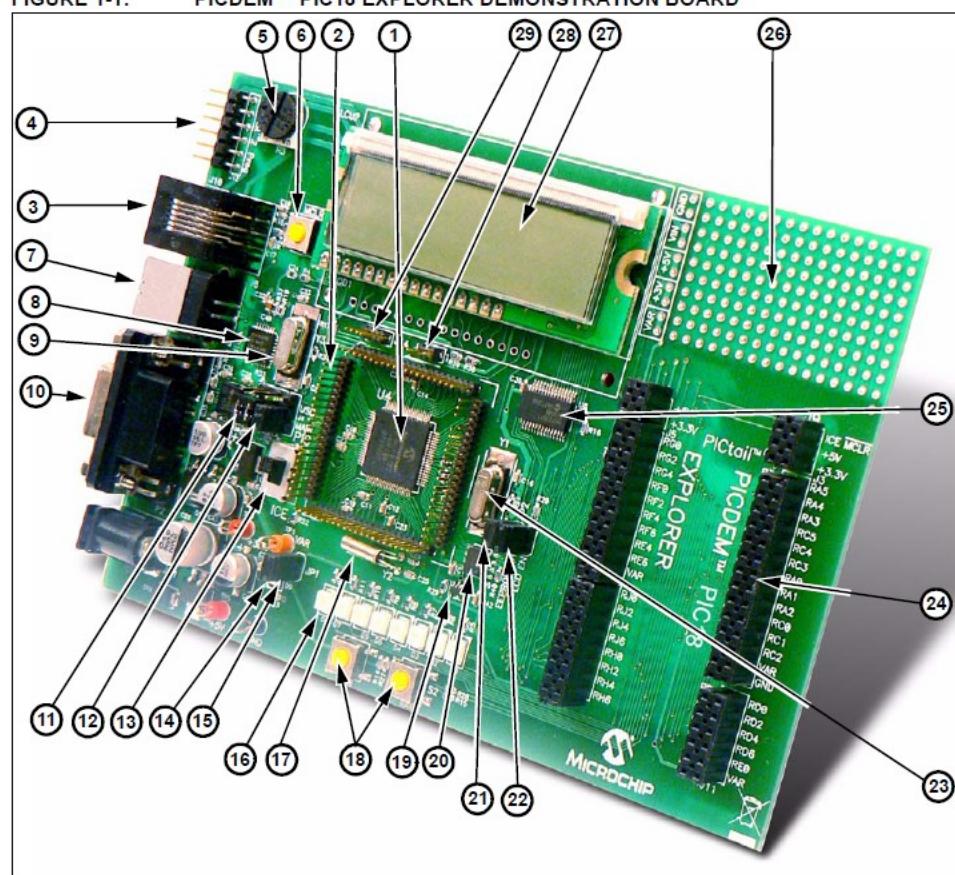
1.1 Phần cứng thí nghiệm Pickit3 và PICDEM PIC18.

Bộ hỗ trợ lập trình dùng với máy tính PICkitTM3.



Kit thí nghiệm PICDEM PIC18 có các đặc điểm như hình sau :

FIGURE 1-1: PICDEM™ PIC18 EXPLORER DEMONSTRATION BOARD



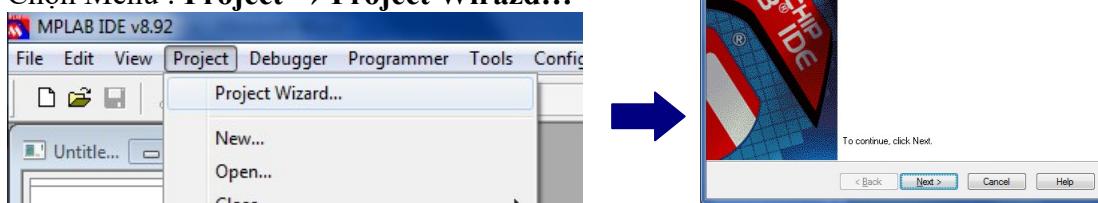
1. Vi điều khiển PIC18F8722.
2. Chân cắm vi điều khiển thứ hai (Plug-In Modules PIMs).
3. Đầu kết nối với bộ lập trình ICD.

4. Đầu kết nối với bộ lập trình PICkit™.
5. Biến trở $10\text{K}\Omega$ dùng cho tín hiệu nhập tương tự.
6. Nút nhấn reset.
7. Đầu cắm USB dùng cho chuẩn giao tiếp RS-232.
8. PIC18LF2450 - dùng để đổi giao tiếp RS-232 ra USB nối với PC.
9. Thạch anh 12MHz dùng cho PIC18F2450.
10. Đầu giao tiếp RS-232 DB9.
11. Jumper J13 chọn đường giao tiếp RS-232 thông qua USB hay DB9.
12. Jumper J4 chọn lập trình PIC18F2450 hay PIC chính trên mạch.
13. Chuyển mạch S4 chọn PIC18F8722 hay PIM.
14. LED nguồn.
15. Jumper J1 dùng để ngắt nguồn 8 LED chỉ thị PORTD.
16. Tám LED chỉ thị cho PORTD.
17. Thạch anh 32.768 kHz dùng cho bộ tạo xung clock cho Timer1.
18. Hai nút nhấn tạo tín hiệu nhập.
19. Cảm biến nhiệt MPC9701A.
20. 25LC256 SPI EEPROM.
21. Jumper J2 cho phép EEPROM.
22. Jumper J3 cho phép LCD.
23. Thạch anh 10 MHz dùng cho PIC chính.
24. PICtail™ daughter board connector socket.
25. Bộ mở rộng giao tiếp SPI dùng cho màn hình LCD ngoài MCP23S17.
26. Vùng lắp thêm linh kiện.
27. Màn hình LCD.
28. J2, 3 chân, dùng để chọn nguồn 3.3V hay 5V.
29. J14, 4 chân, dùng với PIM nếu cần để lấy 3.3V, 5V, V_{IN} và MCLR.

1.2 Môi trường phát triển MPLAB

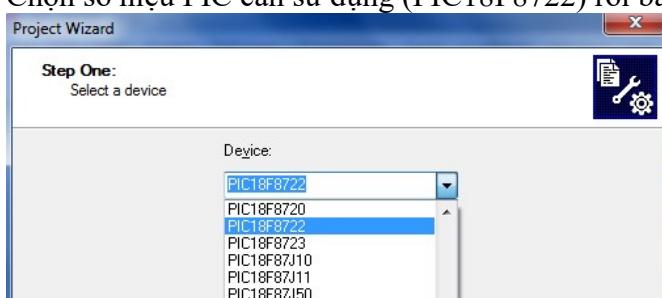
Bước 1. Chạy phần mềm MPLAB V8.92.

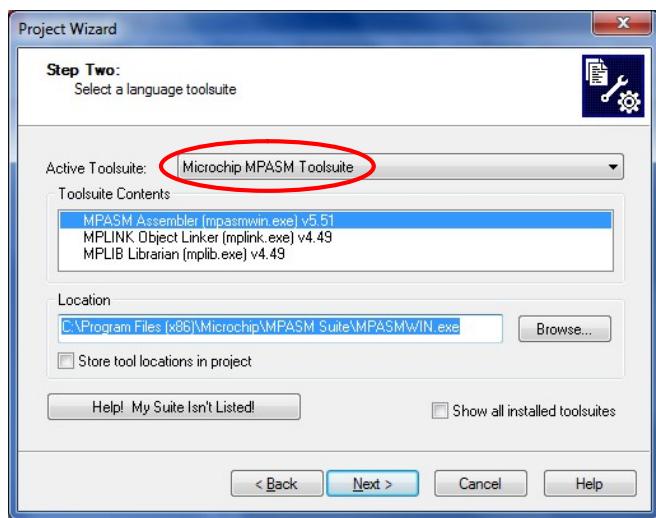
Bước 2. Chọn Menu : Project → Project Wizard...



Chọn Next ở cửa sổ Welcome

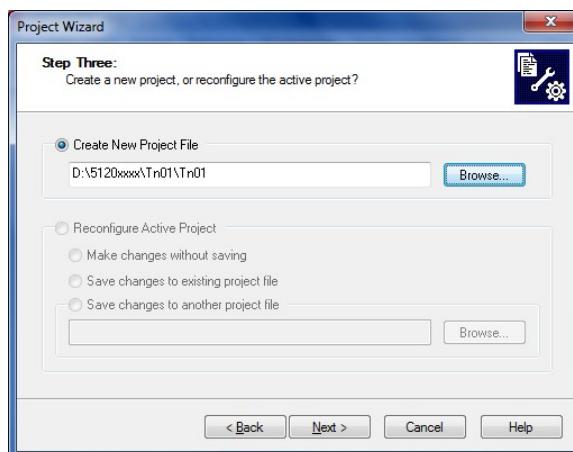
Bước 3. Chọn số hiệu PIC cần sử dụng (PIC18F8722) rồi bấm Next:



Bước 4. Chọn ngôn ngữ lập trình **Microchip MPASM Toolsuite** tại ô Active Toolsuite.**Bước 5.** Chọn thư mục lưu dự án, tên dự án.

Sinh viên nên chọn ổ đĩa dữ liệu (thường là ổ D:), tạo thư mục có tên là mã số sinh viên và vào trong tạo thư mục con tên Tn01 cho bài thí nghiệm 1 (các bài 2, 3, ... sau này cũng sẽ lưu vào thư mục Tn02, Tn03, ...).

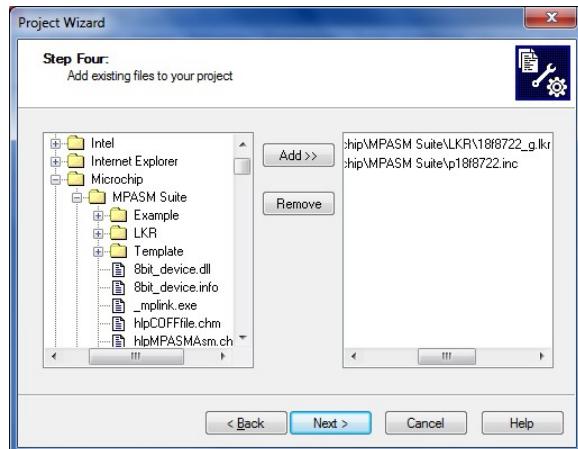
Việc tạo thư mục con như trên giúp sinh viên dễ dàng sao chép dự án ra USB drive lúc kết thúc buổi thí nghiệm hoặc buổi thi.



Bước 6. Thêm header file **p18f8722.inc** và linker script **18f8722_g.lkr** vào dự án với đường dẫn:

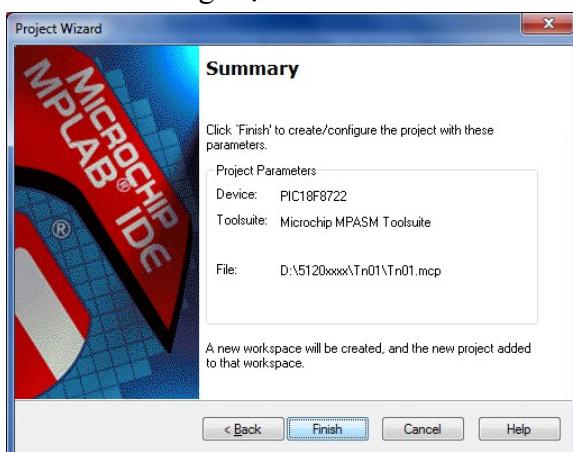
C:\Program Files\Microchip\MPASM Suite\LKR\18f8722_g.lkr

C:\Program Files\Microchip\MPASM Suite\p18f8722.inc

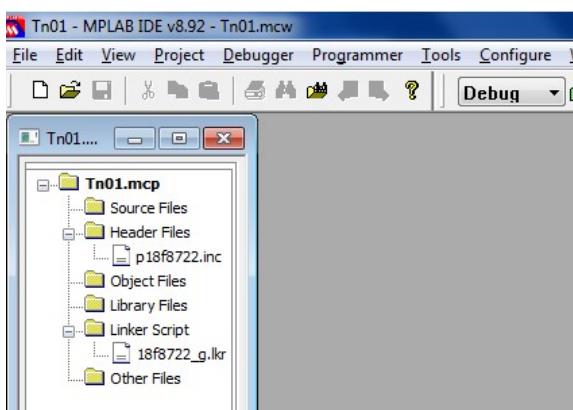


Sinh viên có thể chép 2 file này vào thư mục MSSV để thêm vào các dự án sau này nhanh chóng hơn.

Màn hình cuối cùng hiện ra như sau :



Click **Finish**. Ta sẽ được một project như hình sau:



Bước 7. Thêm Source file vào dự án theo cách sau :

- Tạo cửa sổ mới : **File → New**
- Lưu cửa sổ mới vào thư mục dự án với tên Tn01.asm (phải có .asm) : **File → Save As...**
- Click chuột phải lên **Source Files** trong cửa sổ dự án, chọn **Add Files...**, thêm file Tn01.asm vừa tạo vào.

Bước 8. Trong cửa sổ Tn01.asm, đánh đoạn code sau đây vào để hoàn tất dự án:

```

=====
; Chuong trinh: IO port.
; Ngu dung: Su dung nut nhan RA5 de thay doi LED hien thi.
; Ban dau, hien thi gia tri H'XX' ra LED va cho nhan nut RA5.
; Khi nhan RA5, tang gia tri hien ra LED len 1.
; Khi nhan RA5 khong lam gi ca.

=====
list      p=PIC18F8722
#include  p18f8722.inc
CONFIG    OSC = HS, WDT = OFF, LVP = OFF
#define    LED      LATD
#define    LED_IO   TRISD
#define    NUT      PORTA, RA5
#define    NUT_IO   TRISA, RA5
code     H'000000'
goto    start

;vung dinh nghia du lieu (neu co)
udata_acs
TRI_BD  equ     H'00'           ;XX phai la so cu the
dem    equ     0

;vung dinh nghia cac chuong trinh con
PRG      code
start   rcall   init       ;khoi dong cac thanh ghi va bien trong RAM
;chuong trinh chinh
main    btfsc   NUT        ;nut RA5 duoc nhan ?
        bra     main      ;khong: tiep tuc cho nhan
        rcall   xuat_led ;co: goi chuong trinh con tang gia tri port LED
swoff   btfss   NUT        ;nha nut RA5 ?
        bra     swoff    ; khong: tiep tuc cho nha
                    ; co: khong lam gi ca
        bra     main      ;Tro len kiem tra nhan RA5
;chuong trinh khoi dong ban dau
init    movlw   H'0F'
        movwf   ADCON1   ;cau hinh RA5 la Digital (mac dinh la Analog)
        bsf    NUT_IO   ;cau hinh nut RA5 la cong nhap
        clrf   LED_IO   ;cau hinh LED la cong xuat
        movlw   TRI_BD   ;TRI_BD vao WREG
        movwf   LED      ;WREG ra LED
        return
; cac chuong trinh con khac (neu co)
xuat_led incf    LED
        return
end      ; chi thi ket thuc module hop ngu

```

1.3 Dịch chương trình và nạp code vào vi điều khiển PICBước 9. Cấu hình dự án dùng menu **Configure → Configuration Bits ...** và bỏ chọn như sau:

→ Configuration Bits set in code.

Address	Value	Field	Ca
300001	07	OSC	Oscillator Selection
		FCMEN	Fail-Safe Clock Moni
		IESO	Internal/External Os
300002	1F	PWRT	Power-up Timer Enabl

Chọn giá trị cho các mục sau :

- OSC : HS oscillator
- WDT : WDT disabled (control is placed on the SWDTEN bit)
- LVP : Single-Supplly ICSP disabled

Sau đó bật lại ô chọn :

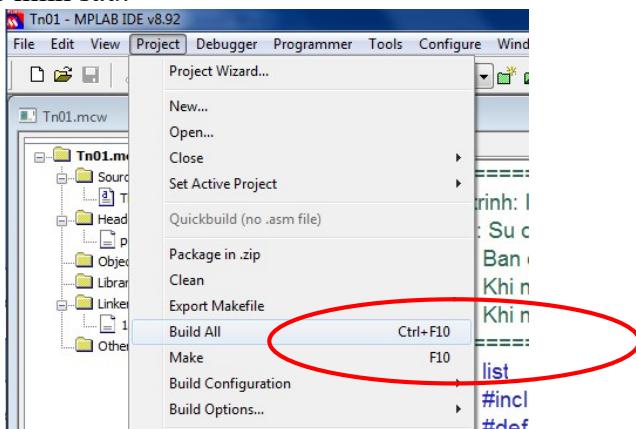
Address	Value	Field	Ca
300001	02	OSC	Oscillator Selection
		FCMEN	Fail-Safe Clock Moni

Ta cũng có thể thực hiện cấu hình bằng chỉ thị CONFIG ở đầu chương trình như sau :

CONFIG OSC = HS, WDT = OFF, LVP = OFF

Bước 10. Sau khi hoàn tất bước 9, ta đã có dự án hoàn chỉnh, sẵn sàng để dịch, nạp ra mạch PICDEM PIC18 và chạy thử.

Để dịch chương trình, ta dùng chức năng: **Project → Build All** (Ctrl-F10) hoặc **Make** (F10) như hình sau:



Nếu việc dịch thất bại (BUILD FAILED), ta cần tìm dòng báo lỗi để sửa.

```
Clean: Deleted file "D:\5120xxx\Tn01\Tn01.cor".
Clean: Deleted file "D:\5120xxx\Tn01\Tn01.hex".
Clean: Done.
Executing: "C:\Program Files (x86)\Microchip\MPASM Suite\MPASMWIN.exe" /q /p18F8722 "Tn01.asm" /l "Tn01.lst" /e "Tn01.err" /o "Tn01.o" /d __DEBUG=1
Error[107] D:\5120xxx\TN01\TN01.ASM 17 : Illegal digit (X in hexadecimal)
Error[107] D:\5120xxx\TN01\TN01.ASM 17 : Illegal digit (X in hexadecimal)
Halting build on first failure as requested.

Debug build of project 'D:\5120xxx\Tn01\Tn01.mcp' failed.
Language tool versions: MPASMWIN.exe v5.51, mplink.exe v4.49, mplib.exe v4.49
Preprocessor symbol '__DEBUG' is defined.
Sat Mar 07 21:46:27 2015
```

BUILD FAILED

Việc tìm lỗi (nhấn kép chuột tại dòng error[xxx] để đến vị trí lỗi), sửa lỗi (thay XX bằng giá trị xác định) và dịch lại được làm cho đến khi dịch thành công.

```
MPLINK 4.49, Linker
Device Database Version 1.14
Copyright (c) 1998-2011 Microchip Technology Inc.
Errors : 0

MP2HEX 4.49, COFF to HEX File Converter
Copyright (c) 1998-2011 Microchip Technology Inc.
Errors : 0

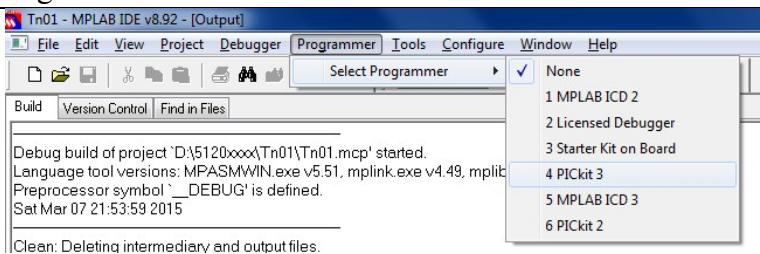
Loaded D:\5120xxx\Tn01\Tn01.cof.

Debug build of project 'D:\5120xxx\Tn01\Tn01.mcp' succeeded.
Language tool versions: MPASMWIN.exe v5.51, mplink.exe v4.49, mplib.exe v4.49
Preprocessor symbol '__DEBUG' is defined.
Sat Mar 07 21:54:01 2015
```

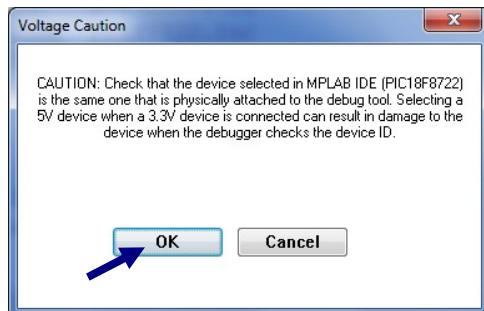
BUILD SUCCEEDED

Bước 11. Chuẩn bị nạp chương trình ra mạch ứng dụng, ta cần chọn Bộ lập trình (Programmer).

Sử dụng: **Programmer → Select Programmer → PICkit 3.**

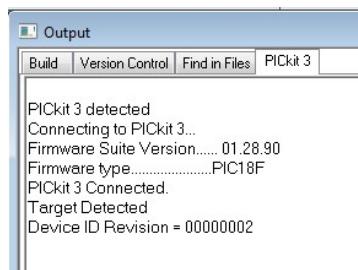


Sau khi chọn PICkit 3 xong thì ta sẽ thấy xuất hiện cửa sổ cảnh báo sử dụng nguồn 3.3V hay 5V như sau:

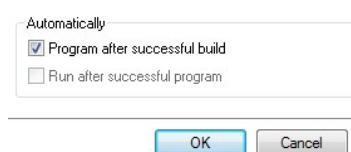


Nhấn OK để tiếp tục.

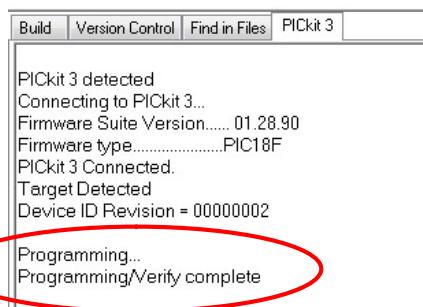
Thông tin xác nhận đã kết nối với **PICkit 3** xuất hiện trên cửa sổ Output như hình sau:



Một thao tác cần làm giúp ta có thể dịch chương trình, nạp và chạy tự động là vào menu **Programmer → Settings... → Program Memory** chọn mục Automatically như hình sau :



Bây giờ ta có thể dịch và nạp dự án với thao tác menu **Project → Build All** (hoặc nhấn Ctrl-F10, hoặc F10). Nếu thành công thì chương trình sẽ được dịch, nạp ra card PICDEM và chạy ngay.

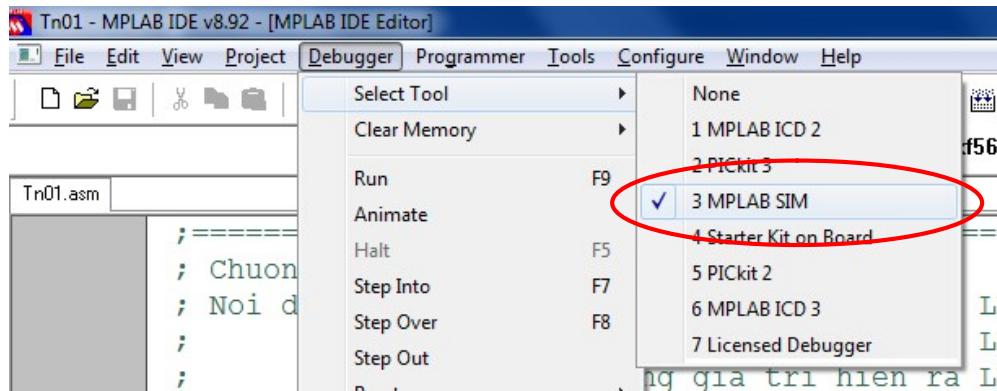


Trường hợp thành công mà chương trình chạy không đúng, ta phải sử dụng công cụ Debugger để tìm lỗi theo một trong hai cách sau.

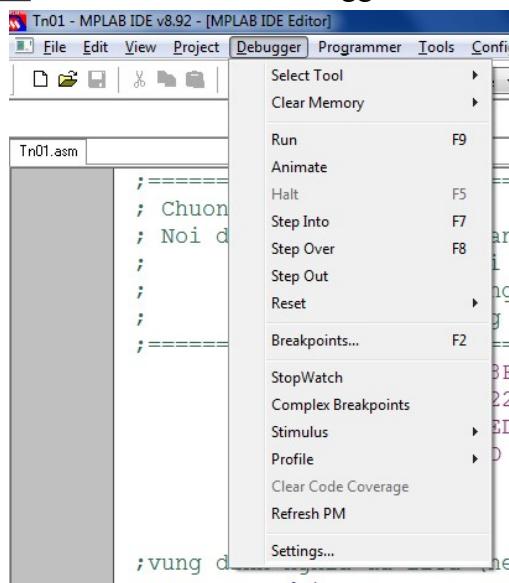
1.4 Debug dùng MpLab SIM

Bước 12. Chọn Debugger bằng menu

Debugger |Select Debugger |Mplab SIM



Bước 13. Tham khảo menu Debugger. Xuất hiện nhiều chức năng hỗ trợ debug.



Từ đây ta đã có thể mô phỏng được chương trình của mình một cách dễ dàng.

Các chức năng thường dùng :

- Run (F9): chạy chương trình, chương trình sẽ chạy liên tục đến khi nào có breakpoint thì dừng.
- Breakpoints (F2): tạo ra breakpoint tại vị trí hiện tại của con trỏ (cũng có thể "double click" vào hàng code mình mong muốn đặt Breakpoint).
- Step Into (F7): chạy từng bước, vào trong chương trình con (nếu gấp).
- Step Over (F8): chạy từng bước, gọi chương trình con cũng xem như 1 bước.
- Reset: trở về đầu chương trình.

Bước 14. Khi debug thì ta cũng cần phải biết giá trị của các thanh ghi cũng như bộ nhớ của chip như thế nào, để xem được các giá trị này thì chúng ta qua menu View.

Để xem được giá trị của các thanh ghi trong PIC ta chọn menu **View → File registers** sẽ xuất hiện cửa sổ như hình sau:

Address	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	ASCII
Hex	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
Symbolic																	
000	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
010	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
050	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
060	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Để xem được giá trị của các thanh ghi SFR thì ta chọn **View → Special Function Registers** sẽ xuất hiện cửa sổ như hình sau:

Address	SFR Name	Hex
F78	TMR4	0x00
F79	ECCP1DEL	0x00
F7C	BAUDCON2	0x40
F7D	SPBRGH2	0x00
F7E	BAUDCON1	0x40
F7F	SPBRGH1	0x00
F80	PORTA	0x00
F81	PORTB	0x00
F82	PORTC	0x00
F83	PORTD	0x56
F84	PORTE	0x00
F85	PORTF	0x00
F86	PORTG	0x00
F87	PORTH	0x00

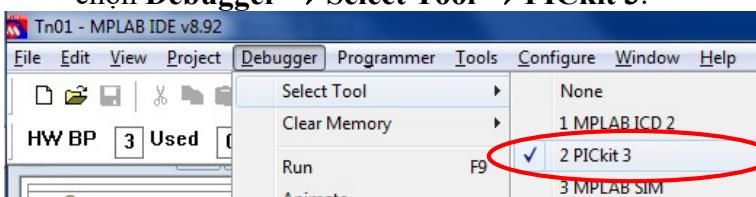
Hay để xem một vành đai mà ta quan tâm thì có thể dùng Watch để xem bằng cách vào menu **View → Watch** thì hình sau sẽ xuất hiện:

Add SFR	PORTD	Add Symbol	
Update	Address	Symbol Name	Value
	F83	PORTD	0x56

Muốn xem thanh ghi nào, ta chỉ việc chọn thanh ghi tương ứng trong combobox bên trên, sau đó nhấn Add SFR.

1.5 Debug onchip dùng PICkit 3.

Bước 15. Cũng giống như debug trên Mplab SIM, PICkit 3 cũng có những tính năng tương tự, chọn **Debugger → Select Tool → PICkit 3**.



Khi sử dụng debug trên PICkit 3 thì cần phải có mạch demo PICdem PIC18, và các hiện tượng xảy ra giống như khi chạy thực tế.

1.6 Bài làm thêm

Cài tiến chương trình Tn01.asm để khi nhấn RA5 1 lần sẽ thấy 8 LED PORTD sáng và nhấn RA5 lần nữa thì tắt tất cả, và cứ thế.

Bài 2 : Khảo sát cổng xuất nhập - Vòng lặp trễ

Nội dung:

- Nd1. Khảo sát hoạt động của nút nhấn, LED.
- Nd2. Khảo sát các thanh điều khiển cổng xuất nhập.
- Nd3. Tính toán thời gian thực thi lệnh, viết chương trình con làm nhiệm vụ delay.
- Nd4. Viết chương trình kiểm tra nút nhấn và hiển thị kết quả kiểm tra ra LED.

Yêu cầu:

- Yc1. Viết chương trình xuất dữ liệu ra led bộ đếm 8 bit. Thời gian giữa các lần đếm lên 1 đơn vị là 500 ms.
- Yc2. Kiểm tra nút nhấn RA5. Khi nút RA5 được nhấn/nhả thì thực hiện toggle việc tăng/giảm giá trị đếm trên port LED.

2.1 Kiến thức liên quan

2.1.1 Các thanh ghi điều khiển cổng xuất nhập

Mỗi Port có ba thanh ghi điều khiển hoạt động chính:

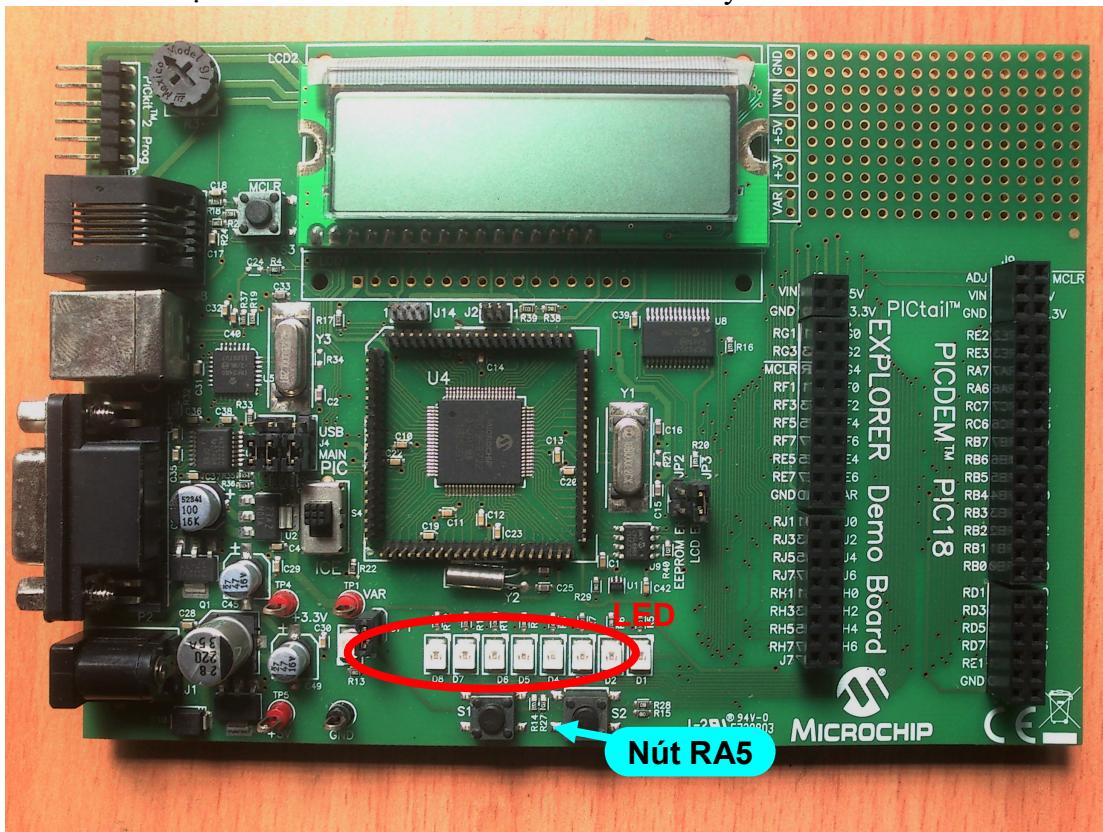
Các bit trong thanh ghi TRIS: thiết lập chân tương ứng là ngõ vào (logic 1) hoặc ngõ ra (logic 0).

Các bit trong thanh ghi PORT: đọc mức logic từ chân tương ứng.

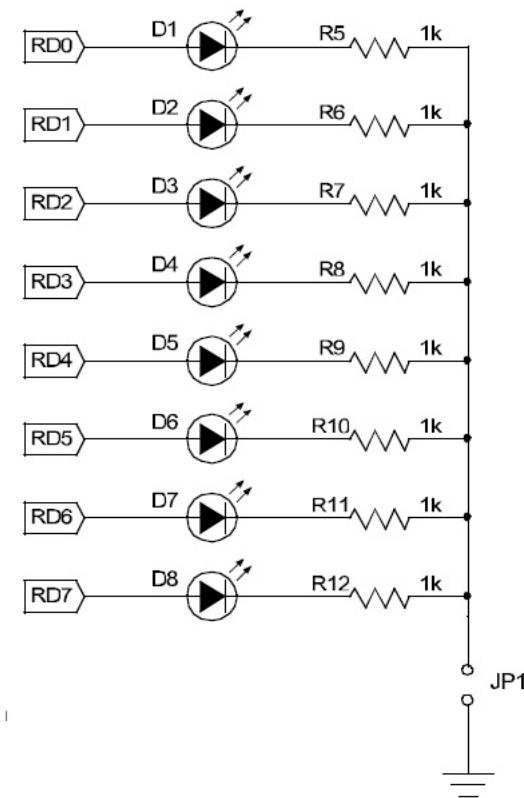
Các bit trong thanh ghi LAT: ghi mức logic ra chân tương ứng.

2.1.2 Kết nối mạch

Vị trí LED hiển thị và nút nhấn trên board như hình dưới đây:



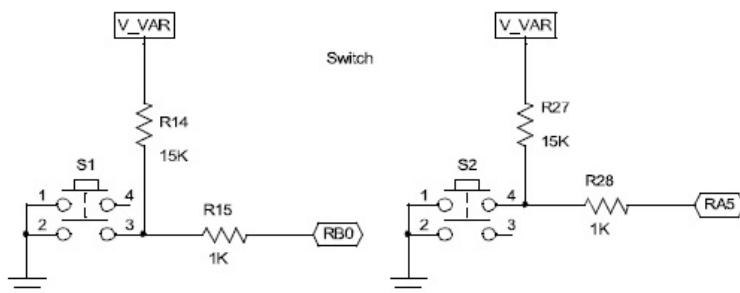
LED hiển thị có sơ đồ mạch như sau:



Muốn các LED sáng, các chân RD_i tương ứng phải lên mức 1.

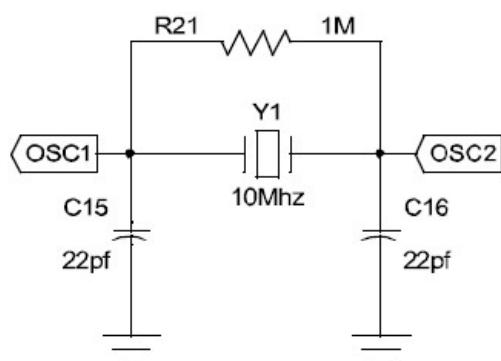
Jumper JP1 cho phép sử dụng PORTD với mục đích khác khi hở ra.

Các nút nhấn có kết nối như sau:



Nút nhấn RA5, RB0 khi được nhấn sẽ làm cho chân tương ứng ở mức **logic 0**. Cần thiết lập các chân RA5 và RB0 là **ngõ vào - digital**. Các thanh ghi liên quan là TRISA, TRISB và ADCON1.

Mạch sử dụng dao động bên trong với thạch anh Y1 có tần số 10 MHz.



2.2 Các bước hiện thực yêu cầu

Bước 1. Tạo dự án mới Tn02. Tạo file Tn02.asm và đánh code vào theo dạng chuẩn relocatable (xem lại bài Tn01).

Bước 2. Khởi tạo port LED và chương trình con **init**.

```
#define LED LATD
#define LED_IO TRISD

...
; Ham khai dong ban dau
init    clrf LED_IO ;cong LED xuat
        clrf LED      ;tat het LED
        return

; Vung du lieu RAM
        udata_acs
dem     res     .1
demla   res     .1
demlb   res     .1

...
; Ham lam tre 1000T
; Fosc=10MHz => T=4/Fosc=4/10.000.000=0.4μs
delay   movlw  .249
        movwf dem
        nop
lap1    nop
        decfsz dem
        bra    lap1
        return
```

Chu kỳ T được tính theo tần số xung dao động Fosc 10 MHz như sau :

$$T = 4 / F_{osc} = 4 / 10 \times 10^6 = 0.4 \mu s$$

$$t_{delay} = 1000T = 400 \mu s$$

Sinh viên tự tính ra thời gian trễ $t_{delay} = 1000T$ từ thời gian thi hành của các lệnh trong vòng lặp. (Tham khảo tập lệnh trong Data sheet)

Bước 4. Từ đây ta có thể tạo ra được hàm **delay500ms** bằng cách gọi hàm **delay** 1250 lần ($1250 = 5 \times 250$) như sau :

```
; Ham lam tre 500ms = 1250 x 0.4μs = 5 x 250 x 0.4μs
delay500ms
        movf dem ngoai,W
        movwf demla
lap2    movlw  .250      ;bat dau vong lap ngoai (5 lan)
        movwf demlb
lap3    call delay      ;bat dau vong lap trong (250 lan)
        decfsz demlb
        bra    lap3      ;ket thuc vong lap trong
        decfsz demla
        bra    lap2      ;ket thuc vong lap ngoai
        return
```

Bước 5. Viết chương trình cho hàm **main** thực hiện tăng trị hiển thị ra port LED lên 1 sau mỗi nửa giây :

```
; chuong trinh chinh
main   incf LED
        rcall delay500ms
        bra    main       ; lap lai sau moi 500 ms
```

Bước 6. Sinh viên viết lại chương trình hoàn chỉnh, yêu cầu đúng dạng chuẩn tái định.

Bước 7. Dịch chương trình, nạp xuống mạch và chạy thử để kiểm tra.

2.3 Các bước hiện thực yêu cầu 2

Bước 8. Tiếp tục yêu cầu 1, thêm vào code cần thiết để sử dụng nút nhấn RA5 như trong Tn01.

Trong module Tn02.asm, thêm vào các định nghĩa:

```
#define NUT      PORTA, RA5
#define NUT_IO    TRISA, RA5
```

Trong hàm **init**, thêm vào code:

```
init      movlw   H'0F'      ;chon ngo nhap RA5 la digital
          movwf   ADCON1
          bsf     NUT_IO     ;RA5 la ngo nhap
```

Bước 9. Để đổi chiều tăng/giảm đếm, ta cần định nghĩa thêm biến và hằng sau trong **udata_acs**:

```
chieu    res    .1
#define TANG    chieu, 0
```

với ý nghĩa cờ (flag) **TANG** là bit 0 của biến **chieu**. Cờ **TANG=1** là tăng số đếm trên LED, cờ **TANG=0** là giảm.

Bước 10. Đoạn code kiểm tra nút nhấn RA5 để đổi chiều được viết trong **main** như sau:

```
main      btfsc  NUT      ;kiem tra nhan nut RA5
          bra    main
          btg    TANG      ;doi chieu
          rcall  xuat_led ;tang/giam so dem tren LED
nha      btfss  NUT      ;cho nha nut RA5
          bra    nha
          bra    main      ; lap lai kiem tra nut RA5
```

; Hàm kiểm tra chiều để đếm tăng hoặc giảm trị LED

```
;Ham tang giam LED
xuat_led btfsc  TANG      ;co TANG=0 la giam so dem LED
          bra    tang_led ;co TANG=1, chuyen den tang so dem LED
giam_led decf   LED
          bra    ket_thuc
tang_led incf   LED
ket_thuc rcall  delay500ms;lam tre nua giay
          return
```

Bước 11. Sửa lại chương trình như trên, dịch và chạy thử.

2.4 Bài làm thêm

- Sửa lại code để giảm thời gian đếm xuống 200ms/lần đếm.
- Tạo hiệu ứng chạy LED light river với 2 LED sáng chạy từ trái sang phải. Nhấn RA5 để đổi chiều.

Bài 3 : Ngắt quãng và truy xuất ROM, RAM

Nội dung:

Nd1. Khảo sát ngắt quãng ngoài INT0 (thông qua nút nhấn RB0) gồm các phần: bố cục chương trình có vector ngắt, các điều kiện sử dụng ngắt quãng, xử lý ngắt quãng (isr: interrupt service routine).

Nd2. Xử lý tra bảng dữ liệu trong ROM và RAM.

Yêu cầu:

Yc1. Viết chương trình khởi tạo ngắt ngoài INT0 để nhận sự kiện nhấn nút RB0.

Tổ chức bảng dữ liệu trong ROM (có 4 dữ liệu).

Mỗi lần nhấn nút RB0, xử lý trong **isr** lấy từng byte dữ liệu trong bảng dữ liệu ROM theo thứ tự tăng dần để xuất ra port LED 8 bit.

Yc2. Lập lại chương trình với bảng dữ liệu trong RAM (cần chép trước từ ROM sang RAM).

Yc2. Lập lại chương trình với bảng dữ liệu trong K/LM (còn chưa trước từ ROM sang K/LM).
Yc3. Kết hợp thêm gọi hàm delay500ms để hiện dữ liệu tự động. Kiểm tra nút nhấn để đổi quy luật.

3.1 Kiến thức liên quan

3.1.1 Tóm tắt các thanh ghi điều khiển ngắt

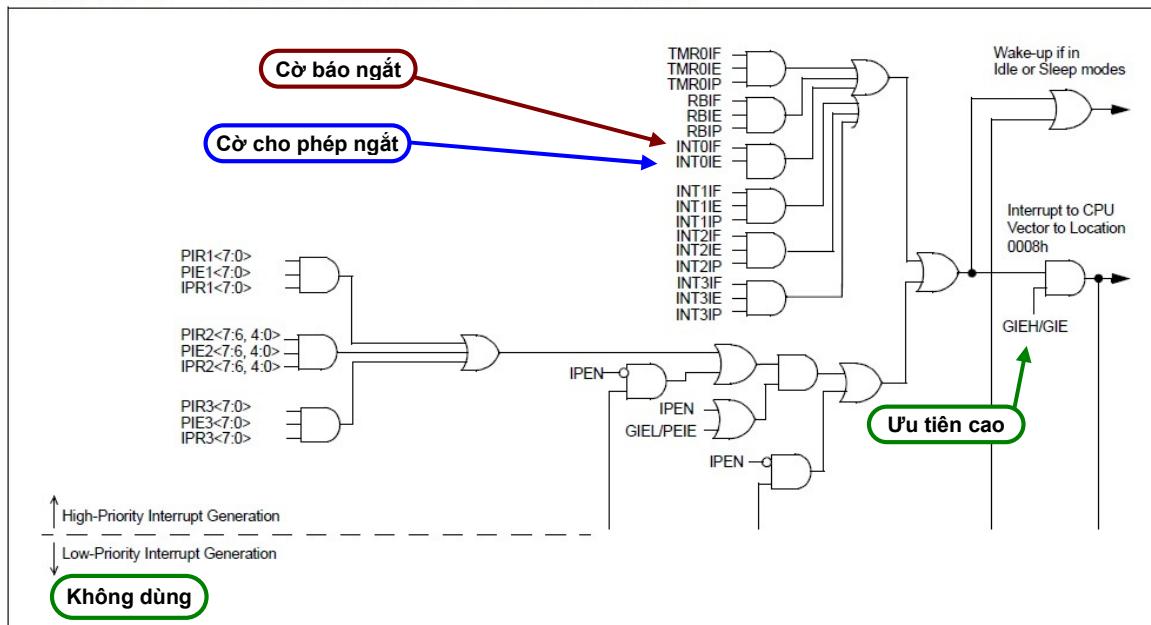
Thanh ghi INTCON:

REGISTER 10-1: INTCON: INTERRUPT CONTROL REGISTER

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/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF ⁽¹⁾
bit 7							bit 0

Sơ đồ điều khiển ngắt:

FIGURE 10-1: PIC18F8722 FAMILY INTERRUPT LOGIC



3.2 Các bước hiện thực yêu cầu 1

Bước 1. Tao dự án mới Tn03, file Interrupt.asm.

Bước 2. Trong cửa sổ code của Interrupt.asm, thêm vector ngắt như sau:

```
; =====
; Chuong trinh: Interrupt.asm
; Noid dung: Su dung nut nhan RB0 qua INT0.
; =====
        list      p=PIC18F8722
        #include  p18f8722.inc
        CONFIG   OSC = HS, WDT = OFF, LVP = OFF

        code      H'00000'
        goto     start
        org      0x08    ← Vector ngắt
        goto     isr_high
        org      0x18
        goto     isr_low
```

Bước 3. Thêm vào cấu hình LED 8 bit, nút nhán RB0 là ngõ nhập.

```
#define  LED          LATD
#define  LED_IO       TRISD
#define  NUTRB0      PORTB, RB0
#define  NUTRB0_IO   TRISB, RB0

Viết hàm init
init
    movlw   H'0F'
    movwf   ADCON1
    bsf     NUTRB0_IO
    clrf   LED_IO
    return
```

Bước 4. Cấu hình cho phép dùng ngắt ngoài INT0. Độ ưu tiên của INT0 được định sẵn là ưu tiên cao (vector H'08'). Đoạn code khởi động như sau:

```
int0_init bsf      RCON, IPEN           ;cho phep uu tien
          bcf      INTCON, INT0IF         ;xoa co ngat IF
          bsf      INTCON, INT0IE         ;cho phep ngat
          bsf      INTCON, GIEH           ;cho phep ngat uu tien cao
          return
```

Bước 5. Viết chương trình phục vụ cho ngắt ngoài INT0 (**isr_high**), trong đó thực hiện :

- Xóa cờ ngắt INT0IF.
- Gọi chương trình con **xuat_led** tra bảng lấy dữ liệu trong ROM (theo biến chỉ số **idx**) xuất ra LED.
- Kết thúc bằng lệnh trở về từ ngắt quãng (**retfie**).

```
isr_high bcf      INTCON, INT0IF
          rcall   xuat_led
          retfie

isr_low
          retfie
```

- Do không sử dụng ngắt ưu tiên thấp nên **isr_low** chỉ có một lệnh **retfie**.

Bước 6. Viết chương trình con **xuat_led** thực hiện công việc :

- Tra bảng lấy dữ liệu từ ROM, dùng thanh ghi **TBLPTR** (21 bit).
- Xuất byte nội dung ROM ra port LED.
- Gọi hàm **inc_idx** tăng chỉ số.
- Định nghĩa bảng dữ liệu (dùng chỉ thị **db**) ROM ngay sau chương trình con **xuat_led**.

xuat_led	movlw	upper_dulieu	;nap dia chi dau bang vao
	movwf	TBLPTRU	;thanh ghi 21 bit TBLPTR
	movlw	high_dulieu	
	movwf	TBLPTRH	
	movlw	low_dulieu	
	movwf	TBLPTRL	
	movf	idx,W	;chi so cua du lieu
	incf	WREG	
trabang	TBLRD*+		;vong lap doc bo qua cac
	decfsz	WREG	;byte dung truoc
	bra	trabang	
	movf	TABLAT,W	;WREG=dulieu[idx]
	movwf	LED	
	rcall	inc_idx	;dieu chinh chi so
	return		
dulieu	db	H'55', H'AA', H'C3', H'3C'	
inc_idx	incf	idx	
	movlw	MAXIDX	
	cpfslt	idx	
	clrf	idx	
	return		

Bước 7. Viết hàm điều chỉnh chỉ số inc_idx (chỉ số thay đổi xoay vòng từ 0 đến MAXIDX).

```

inc_idx    incf      idx
          movlw    MAXIDX
          cpfslt   idx
          clrf     idx
          return

```

3.3 Chương trình mẫu yêu cầu 1

Bước 8. Sinh viên viết lại chương trình hoàn chỉnh theo gợi ý sau

```

=====
; Chuong trinh: Interrupt.asm
; Nguon dung: Khoi tao ngat ngoai INT0 cho nut RB0.
;             Nhan nut RB0, tra bang lay du lieu trong ROM
;             xuat ra LED
=====
list      p=PIC18F8722
#include  p18f8722.inc
CONFIG   OSC = HS, WDT = OFF, LVP = OFF
#define    LED      LATD
#define    LED_IO   TRISD
#define    NUTRBO  PORTB,RB0
#define    NUTRBO_IO TRISB,RB0

code     H'00000'
goto    start
org     0x08
goto    isr_high
org     0x18
goto    isr_low

;Vung du lieu
udata_acs

;Vung viet code
PRG      code
start   rcall   init
        rcall   int0_init
main    bra     main
;Ham khai dong
init
        ;sinh vien viet code
        return
;Ham khai dong ngat INT0
int0_init
        ;sinh vien viet code
        return

```

```

;Ham tra bang lay du lieu tu ROM xuat ra LED
xuat_led
    ;sinh vien viet code
    return
dulieu db      H'55',H'AA',H'C3',H'3C'
;Ham tang chi so bang
inc_idx
    ;sinh vien viet code
    return
;Ham phuc vu ngat uu tien cao
isr_high bcf      INTCON,INT0IF
    rcall    xuat_led
    retfie
;Ham phuc vu ngat uu tien thap
isr_low  retfie
end

```

Bước 9. Chạy thử chương trình.

3.4 Các bước hiện thực yêu cầu 2

Bước 10. Thêm vào udata_acs bảng 4 byte như sau :

```

buffer res .4

```

Bước 11. Trong phần init, thêm lệnh gọi hàm rom2ram để chép bảng dữ liệu từ ROM (**bang_dl**) sang vùng **buffer** trong RAM.

```

rom2ram movlw upper dulieu      ;nap dia chi dau bang vao
        movwf TBLPTRU      ;thanh ghi 21 bit TBLPTR
        movlw high dulieu
        movwf TBLPTRH
        movlw low dulieu
        movwf TBLPTRL
        lfsr   FSR0,buffer      ;dau bang RAM
        movlw MAXIDX      ;so byte can chep
        TBLRD*+
        movff TABLAT,POSTINCO ;doc ROM
        decfsz WREG          ;chep sang RAM
        bra    chep
        return

```

Bước 12. Viết thêm hàm **xuat_led2** thực hiện tra bảng lấy dữ liệu từ RAM

```

xuat_led2 lfsr   FSR1,buffer      ;dau bang RAM
        movf   idx,W          ;chi so cua du lieu
        movf   PLUSW1,W        ;lay du lieu buffer[idx]
        movwf  LED
        rcall  inc_idx         ;dieu chinh chi so
        return

```

Bước 13. Dịch và chạy thử chương trình đã sửa.

3.5 Thực hiện yêu cầu 3

Bước 14. Thêm hàm **delay** và **delay500ms** như trong bài 2.

Bước 15. Chính lại hàm **main** để gọi **xuat_led** và **delay500ms**

```

main   rcall  xuat_led2
        rcall  delay500ms
        bra   main

```

Bước 16. Sửa lại code để khi nhấn nút RB0 thì gọi đến hàm **doi ql** (thay cho hàm **xuat_led2**) để chép các bộ dữ liệu khác nhau từ ROM sang RAM (thông qua các hàm **rom2ram1**, **rom2ram2**).

```

doi_ql    tstfsz    so_ql
          bra        ql_2
ql_1      rcall     rom2ram1
          incf      so_ql
          return
ql_2      rcall     rom2ram2
          clrf      so_ql
          return

```

Bước 17. Hoàn chỉnh chương trình và chạy thử.

3.6 Bài làm thêm

- a) Viết chương trình thực hiện chạy LED theo các quy luật sau:

Trạng thái 1:	○ ○ ○ ○ ○ ○ ○ ○	○ ○ ○ ○ ○ ○ ○ ○
Trạng thái 2:	● ○ ○ ○ ○ ○ ○ ●	● ○ ○ ○ ○ ○ ○ ○
Trạng thái 3:	● ● ○ ○ ○ ○ ● ●	● ● ○ ○ ○ ○ ○ ○
Trạng thái 4:	● ● ● ○ ○ ○ ● ●	● ● ● ○ ○ ○ ○ ○
Trạng thái 5:	● ● ● ● ○ ○ ○ ○	● ● ● ● ○ ○ ○ ○
Trạng thái 6:	● ● ● ○ ○ ○ ● ●	● ● ● ○ ○ ○ ○ ○
Trạng thái 7:	● ● ○ ○ ○ ○ ● ●	● ● ○ ○ ○ ○ ○ ○
Trạng thái 8:	● ○ ○ ○ ○ ○ ○ ●	● ○ ○ ○ ○ ○ ○ ○
Trạng thái 9:	trở về 1	○ ● ● ● ● ● ● ●
Trạng thái 10:		○ ○ ● ● ● ● ● ●
Trạng thái 11:		○ ○ ○ ● ● ● ● ●
Trạng thái 12:		○ ○ ○ ○ ● ● ● ●
Trạng thái 13:		○ ○ ○ ○ ○ ● ● ●
Trạng thái 14:		○ ○ ○ ○ ○ ○ ● ●
Trạng thái 15:		○ ○ ○ ○ ○ ○ ○ ●
Trạng thái 16:		trở về 1

- b) Viết chương trình hiện ra LED 4 quy luật khác nhau tự động và đổi bằng nút RB0.

Bài 4 : Khảo sát bộ định thời - Module

Nội dung:

Nd1. Khảo sát các chế độ hoạt động của các bộ định thời timer0 và timer1.

Nd2. Tạo module Timer.asm để khởi động và sử dụng bộ định thời.

Nd3. Khai thác module Lcd.asm để hiển thị ra màn hình LCD.

Yêu cầu:

Yc1. Sử dụng bộ timer0 tạo ngắt quãng thời gian thực 10ms theo công thức.

Yc2. Trên cơ sở ngắt thời gian 10ms, kết hợp thêm biến đếm phụ (với số lần đếm SODEM1S), viết hàm **delay1s** để có thời gian trễ 1s dùng cho việc điều khiển hiển thị LED.

Yc3. Thay đổi giá trị prescaler (4, 8), tính lại số đếm để thời gian xảy ra ngắt không đổi.

Yc4. Sử dụng module Lcd.asm có sẵn để hiển thị ra màn hình LCD các ký tự sau mỗi giây.

4.1 Các bước hiện thực yêu cầu 1

Bước 1. Tạo dự án mới Tn04. Tập tin Tn04.asm dạng relocatable có dùng vector ngắt.

Bước 2. Khởi động port LED.

Bước 3. Tạo mới module định thì có tên *Timer.asm*.

Bước 4. Tham khảo thanh ghi điều khiển bộ định thì Timer0 **T0CON** :

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
TMR0ON	T08BIT	T0CS	T0SE	PSA	T0PS2	T0PS1	T0PS0
0	0	0	0	0	0	0	0

Trong module Timer.asm, viết chương trình con khởi động **Timer0_init** gồm các bước càn làm như sau:

Chọn mức độ ưu tiên cho ngắt Timer0 [IPEN=1, TMR0IP].

Cho phép sử dụng ngắt Timer0 [TMR0IE=1].

Cho phép ngắt tổng thể [GIEH=1, GIEL=1].

Xoá cờ ngắt [TMR0IF=0] (làm trong hàm Timer0_reset).

Nạp số đếm vào bộ đếm (làm trong hàm Timer0_reset).

```

list      p=PIC18f8722
#include  P18F8722.INC
extern   Timer_process
SODEM10MS equ     .xxxxxx
PRG      code
;Cau hinh Timer0
global   Timer0_init
Timer0_init
    bsf     RCON, IPEN           ;su dung uu tien
    bcf     INTCON2, TMR0IP       ;Timer0 uu tien thap
    bsf     INTCON, TMROIE        ;cho phép ngắt Timer0
    bsf     INTCON, GIEH          ;cho phép ngắt uu tien cao
    bsf     INTCON, GIEL          ;cho phép ngắt uu tien thap
    clrf   T0CON                ;dem 16 bit, prescaler=2
    rcall  Timer0_reset
    return

```

Hàm Timer0_reset

```

Timer0_reset
    bcf      INTCON, TMROIF      ;xoa co ngat
    bcf      T0CON, TMR0ON      ;cam dem
    movlw   high (-SODEM10MS)   ;nap lai so dem
    movwf   TMR0H
    movlw   low (-SODEM10MS)
    movwf   TMR0L
    bsf      T0CON, TMR0ON      ;cho phep dem
    return

```

Bước 5. Số đếm SODEM10MS được tính từ công thức sau :

$$t_{\text{timer}} = (1/(f_{\text{oosc}}/4)) * \text{prescaler} * \text{SODEM10MS}$$

trong đó:

t_{timer} là đơn vị thời gian cần tạo ra, trong ví dụ này là 10ms.

f_{oosc} là tần số xung clock cấp cho CPU (với PICDEM PIC18 là 10 MHz).

prescaler = 2 (do lập trình các bit T0PS_{2,1,0} trong T0CON quy định)

Sinh viên tính ra giá trị của SODEM10MS và thay vào xxxx ở dòng code:

```
SODEM10MS EQU .xxxxx
```

Do bộ đếm thực hiện **đếm tăng** khi có xung clock nên giá trị nạp vào bộ đếm sử dụng số âm (**-SODEM10MS**).

Ngoài ra, việc định nghĩa hằng số SODEM10MS thay vì dùng số trực tiếp giúp ta dễ dàng thay đổi giá trị đếm sau này khi muốn thay đổi thời gian xảy ra ngắt quãng.

Bước 6. Viết chương trình phục vụ ngắt quãng cho Timer0 (**Timer0_isr**) thực hiện các việc sau :

Xóa cờ ngắt (TMR0IF).

[hàm Timer0_reset]

Cảm đếm (xóa TMR0ON).

[hàm Timer0_reset]

Nạp lại số đếm (TMR0H - TMR0L).

[hàm Timer0_reset]

Cho phép đếm lại (lập TMR0ON).

[hàm Timer0_reset]

Gọi chương trình con xử lý định kỳ **Timer_process** (10ms thực hiện 1 lần).

Tham khảo code sau:

```

;Chuong trinh phuc vu ngat quang Timer0
global   Timer0_isr
Timer0_isr
    rcall    Timer0_reset
    rcall    Timer_process
    return
    end

```

Bước 7. Hàm Timer_process sẽ được viết trong module chính Tn04.asm nên trong module Timer.asm chỉ cần khai báo:

```
extern   Timer_process
```

ở đầu module như sau là được rồi.

Bước 8. Chỉ thị **global** được dùng kết hợp với **extern** cho phép các module có thể trao đổi tên biến, tên hàm để sử dụng. Muốn thế, ta tạo thêm module **Timer.inc** có nội dung như sau để dùng với #include trong module chính Tn04.asm:

```
extern   Timer0_init, Timer0_isr
```

4.2 Các bước hiện thực yêu cầu 2

Bước 9. Trong module Tn04.asm, thêm vào:

```
#include "Timer.inc"
```

Bước 10. Viết hàm Timer_process như sau:

```

;Chuong trinh xu ly dinh ky cua Timer (10ms/lan)
    global    Timer_process
Timer_process
    decfsz   dem      ;dem them n lan
    return
    bsf      FLAG_1S   ;da du thoi gian, bat co hieu
    movlw    SODEM1S  ;nap lai n
    movwf    dem
    return

```

Bước 11. Định nghĩa cờ **FLAG_1S** sử dụng theo bit.

```

#define  FLAG_1S  flags,0
udata_acs
dem   res     .1
flags res     .1

```

Bước 12. Trong **main**, ta kiểm tra theo dõi cờ **FLAG_1S** để thực hiện thay đổi LED.

```

PRG      code
;main program
start   rcall   init
        rcall   Timer0_init
main    btfss   FLAG_1S
        bra    main
        bcf    INTCON,GIEH
        bcf    FLAG_1S
        bsf    INTCON,GIEH
        incf   LED
        bra    main

```

Bước 13. Ta cũng có thể dùng **Timer_process** để tạo ra hàm **Delay1s** như cách sau:

```

;Ham lam tre n lan 10ms
Delay1s  movlw    SODEM1S
        movwf   dem
D11s1   tstfsz  dem
        bra    D11s1
        return
;Chuong trinh xu ly dinh ky cua Timer (10ms/lan)
    global    Timer_process
Timer_process
    tstfsz  dem      ;neu dem>0 moi giam
    decf    dem      ;giam dem
    return

```

4.3 Các bước hiện thực yêu cầu 3

Bước 14. Tham khảo thanh ghi T0CON để thay đổi giá trị **prescaler = 8**.

Bước 15. Tính lại hằng số **SODEM10MS** để giữ nguyên thời gian $t_{timer} = 10ms$.

Bước 16. Chạy lại chương trình, có thể kết hợp với tra bảng dữ liệu như trong bài Tn03.

4.4 Các bước hiện thực yêu cầu 4

Bước 17. Module Lcd.asm và Lcd.inc cho phép sử dụng màn hình LCD với các hàm hỗ trợ sau đây:

Hàm **Lcd_init** : khởi động màn hình LCD, gọi 1 lần ở đầu chương trình.

Hàm **Lcd_cls** : xoá trống màn hình.

Hàm **Lcd_gotoxy** dùng chung với 2 biến lcd_row (dòng: 0÷1) và lcd_col (cột: 0÷15) để dời con trỏ màn hình đến vị trí (dòng,cột). Màn hình có 2 dòng, 16 cột.

Hàm **Lcd_putc** : hiện ký tự ASCII trong biến **lcd_wr** ra vị trí con trỏ.

Ví dụ: muốn hiện ký tự 'A' ra vị trí hàng 1, cột 7, ta làm như sau:

```

PRG      code
;main program
start   rcall    init
        rcall    Lcd_init
main    bra     main
        movlw   .1
        movwf   lcd_row
        movlw   .7
        movwf   lcd_col
        rcall   Lcd_gotoxy
        movlw   'A'
        movwf   lcd_wr
        rcall   Lcd_putc
        bra     main1
main1

```

Bước 18. Viết chương trình hiện ký tự tăng dần ($0 \div 255$) và chạy khắp màn hình (200ms chạy 1 vị trí) với các quy luật chạy:

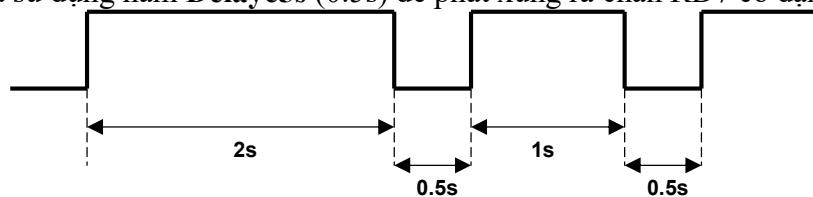
Từ trái sang phải, từ trên xuống dưới.

Từ trái sang phải, từ dưới lên trên.

Từ phải sang trái, từ trên xuống dưới.

4.5 Bài tập

- Làm lại bài thí nghiệm nhưng sử dụng Timer1 thay vì Timer0.
- Viết và sử dụng hàm **Delay5s** (0.5s) để phát xung ra chân RD7 có dạng như sau:



- Dùng bộ định thời tạo xung vuông chu kì 10ms, duty cycle 30%.
- Viết hàm xuất một chuỗi ASCIIIZ trong ROM/RAM ra LCD.

Bài 5 : Bàn phím

Nội dung:

- Nd1. Khảo sát cấu tạo, hoạt động của ma trận phím.
- Nd2. Tìm hiểu kỹ thuật "quét hàng / đọc cột", định vị phím nhấn.
- Nd3. Chống rung phím bằng phần mềm (bài làm thêm).

Yêu cầu:

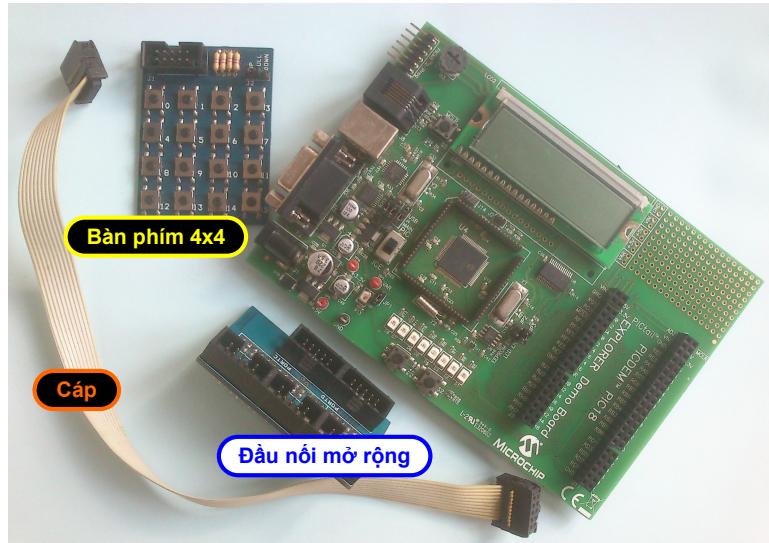
- Yc1. Dự án Tn05, file Tn05.asm, code sử dụng ngắt Timer.
- Yc2. Kiểm tra phím trên từng hàng.
- Yc3. Viết chương trình con kiểm tra quét hàng theo chỉ số.
- Yc4. Kết hợp với ngắt thời gian 10ms gọi chương trình con kiểm tra quét hàng, định vị phím, xuất số thứ tự phím ra LED 8-bit (hoặc LCD).

5.1 Các bước hiện thực yêu cầu 1

Bước 1. Tạo dự án mới Tn05, tập tin nguồn Tn05.asm, đánh vào code có vector ngắt.

Bước 2. Khởi động: port LED 8-bit, Timer0, portB (xem yêu cầu bên dưới).

Bước 3. Kết nối mạch:



Cắm connector mở rộng vào card PICDEM PIC18 như trong hình.



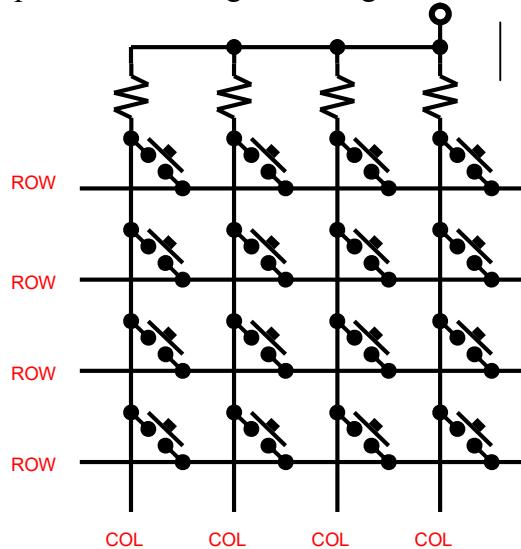
Kết nối : bàn phím - dây cáp - portB của connector mở rộng như hình sau:



Rút Jumper LCD EN trên card PICDEM PIC18 ra và cắm lại 1 chân thay vì 2 chân
(nghĩa là không dùng LCD vì trùng portB).

Bước 4. Kết nối hàng cột với PortB.

Ma trận 16 phím có cấu tạo gồm 4 hàng x 4 cột như hình bên dưới.



Cột có trở kéo lên để đảm bảo mức logic 1 khi phím không được nhấn.

Sử dụng PortB để kết nối với ma trận 16 phím (4 hàng x 4 cột) :

Cột COL1 - COL4 : cổng nhập, nối vào các bit RB₀₋₃.

Hàng ROW1 - ROW4 : cổng xuất, nối vào RB₄₋₇.

5.2 Kiểm tra bàn phím

Bước 5. Cấu hình PortB cho 4 bit cao là xuất, 4 bit thấp là nhập (sinh viên phải xác định giá trị cho thanh ghi TRISB).

Các định nghĩa hàng

```

#define LED LATD
#define LED_IO TRISD
#define KEY_IO TRISB
#define ALLCOL PORTB
#define ALLROW LATB

#define COL1 PORTB, RB0
#define COL2 PORTB, RB1
#define COL3 PORTB, RB2
#define COL4 PORTB, RB3
#define ROW1 LATB, RB4
#define ROW2 LATB, RB5
#define ROW3 LATB, RB6
#define ROW4 LATB, RB7

```

Hàm init

```

init
    movlw H'OF'      ;Digital input
    movwf ADCON1
    movlw H'XX'      ;Sinh vien xac dinh gia tri XX
    movwf KEY_IO     ;cau hinh keyboard
    clrf LED_IO
    clrf LED
    ;khoi dong cac bien can thiet
    return

```

Bước 6. Thực hiện kiểm tra phím trên 1 hàng. Làm cho hàng 1 xuống mức 0, các hàng khác lên mức 1. Ta có thể làm riêng từng bit (dùng ROW1 - ROW4) hoặc làm một lần trên cả port (dùng ALLROW). Sau đó, đọc dữ liệu về từ cột, chuyển sang port LED để quan sát kết quả (chủ yếu là 4 bit thấp).

```

main
    bcf ROW1      ;hoac ALLROW=B'11101111'
    bsf ROW2
    bsf ROW3
    bsf ROW4

main1
    movf ALLCOL,W
    movwf LED
    bra main1

```

Bước 7. Dịch, chạy, nhấn phím trên hàng 1 và quan sát port LED.

Bước 8. Thực hiện tiếp kiểm tra trên các hàng còn lại.

5.3 Quét toàn bàn phím:

Bước 9. Thêm vào module sử dụng Timer0 tạo thời khoản 10ms. Trong Timer_process thực hiện kiểm tra phím theo các bước kế tiếp.

Bước 10. Sử dụng một biến row_idx (giá trị = 0, 1, 2, 3) để ghi nhớ vị trí hàng đang được kiểm tra. Như vậy, dữ liệu xuất ra cổng ALLROW (còn được gọi là mã quét scan_code) sẽ thay đổi tùy theo chỉ số hàng. Việc xác định scan_code có thể được thực hiện bằng nhiều cách như: tra bảng, kiểm tra giá trị row_idx rồi gán scan_code hoặc dùng vòng lặp quay trai. Sau đây là hàm **GetScancode** sử dụng cách kiểm tra row_idx và gán trị scan_code.

```

GetScancode
    movf      row_idx,W
    incf      WREG
    dcfsnz   WREG
    bra       getrow1
    dcfsnz   WREG
    bra       getrow2
    dcfsnz   WREG
    bra       getrow3
    getrow4  movlw   B'01111111'
    bra       getend
    getrow3  movlw   B'10111111'
    bra       getend
    getrow2  movlw   B'11011111'
    bra       getend
    getrow1  movlw   B'11101111'
    getend   movwf   scan_code
    return

```

Bước 11. Xác định số thứ tự phím nhấn và xuất ra LED.

Hàng 1: số từ 0 đến 3.

Hàng 2: số từ 4 đến 7.

Hàng 3: số từ 8 đến 11.

Hàng 4: số từ 12 đến 15.

Có thể sử dụng công thức : key_code = row_idx * 4 + chỉ số cột

Như vậy, cần xác định chỉ số cột (0-3) của phím được nhấn từ dữ liệu đọc được từ ALLCOL bằng cách kiểm tra các bit từ 0 đến 3 và gán trị vào key_code như hàm Getkey sau:

```

Getkey
    movf      ALLCOL,W
    movwf   key
    clrf   key_code
    btfss  key,0      ; kiem tra cot 1
    bra    Getkeyend
    incf   key_code   ; kiem tra cot 2
    btfss  key,1
    bra    Getkeyend
    incf   key_code   ; kiem tra cot 3
    btfss  key,2
    bra    Getkeyend
    incf   key_code   ; kiem tra cot 4
    btfss  key,3
    bra    Getkeyend
    setf   key_code   ; khong nhan, key_code=H'FF'
    return

Getkeyend
    movf      row_idx,W
    rlnkf   WREG      ;row_idx X 4
    rlnkf   WREG
    addwf   key_code
    return

```

Bước 12. Sau khi lấy được key_code, gọi hàm Inc_rowidx để tăng chỉ số hàng lên.

```

Inc_rowidx
    incf   row_idx      ; tang chi so
    movlw  MAXIDX
    cpfslt row_idx      ; neu row_idx=MAXIDX cho ve 0
    clrf   row_idx
    return

```

Bước 13. Hàm Timer_process.

```

global      Timer_process
Timer_process
    rcall   GetScancode
    movwf  ALLROW      ;quet hang tuy theo row_idx
;kiem tra cot
    rcall   Getkey      ;doc du lieu cot, tinh key_code
    rcall   Inc_rowidx
    incf   key_code,W
    tstfsz WREG        ;=0:khong co phim nhan
    rcall   Key_process ;xu ly phim nhan
    return

```

Bước 14. Cuối cùng, hàm Key_process có nhiệm vụ xử lý các phím nhấn tùy theo chức năng được gán cho phím. Trong bài thí nghiệm này chỉ đơn giản là hiện key_code ra LED. Tuy nhiên, sinh viên có thể chọn một vài phím nào đó để có thể làm các công việc ở các bài thí nghiệm trước.

5.4 Bài tập

- Sử dụng kỹ thuật tra bảng để lấy scan_code.
- Sử dụng vòng lặp quay trai để tính scan_code.
- Đổi qua kết nối bàn phím với PortD để có thể sử dụng LCD ở PortB.

Bài 6 : Kỹ thuật quét LED

Nội dung:

Nd1. Khảo sát cấu tạo, hoạt động của LED ma trận và LED 7 đoạn.

Nd2. Tìm hiểu kỹ thuật quét LED ma trận và LED 7 đoạn.

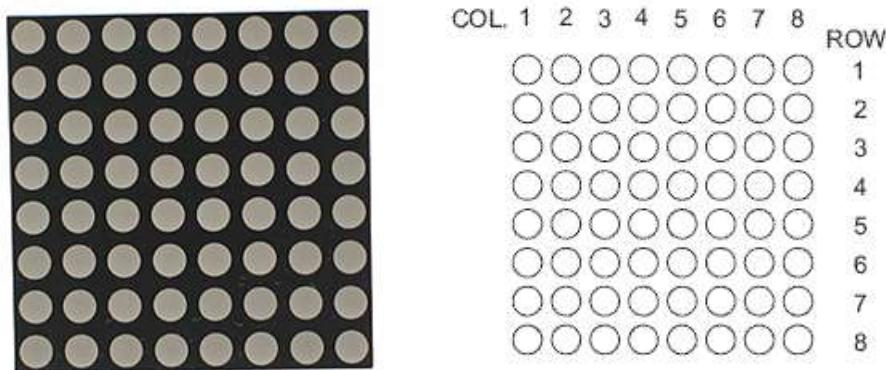
Yêu cầu:

Yc1. Viết chương trình cho phép hiển thị giá trị ra LED ma trận.

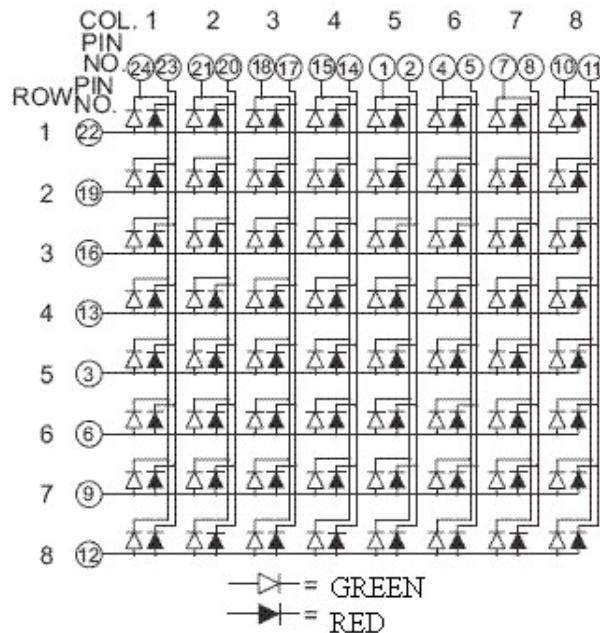
Yc2. Viết chương trình cho phép hiển thị giá trị ra LED 7 đoạn.

6.1 Cấu tạo LED ma trận và LED 7 đoạn:

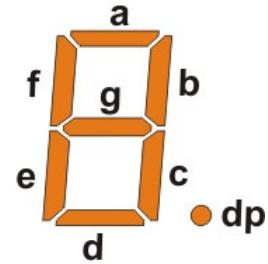
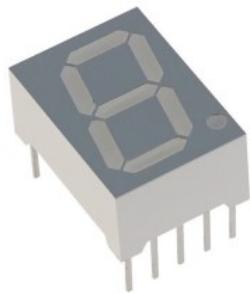
LED ma trận 8x8 hai màu được bố trí thành 8 hàng và 8 cột. Mỗi điểm có hai LED.



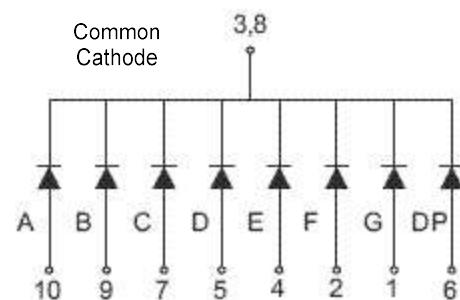
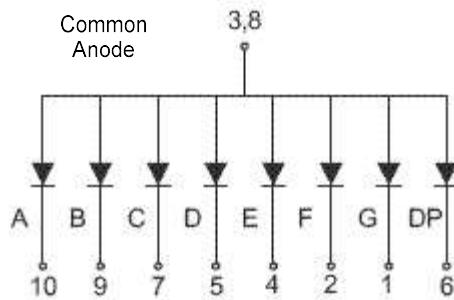
Các LED trên cùng một hàng nối chung Anode, các LED cùng loại trên cùng một cột nối chung Cathode.



LED 7 đoạn gồm có 7 đoạn được đánh dấu: a, b, c, d, e, f, g và một điểm dp.



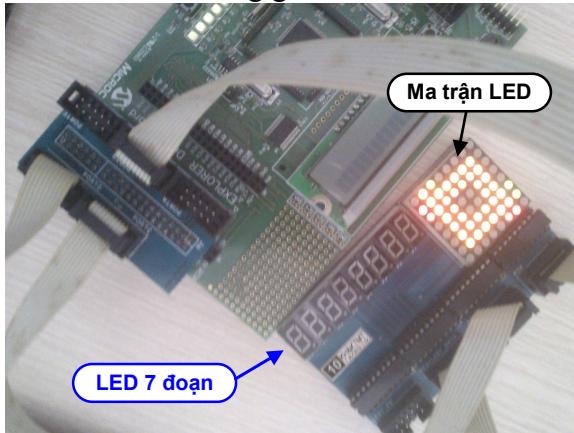
LED 7 đoạn có hai loại là Common Anode và Common Cathode, tương ứng các LED nối chung Anode hay nối chung Cathode.



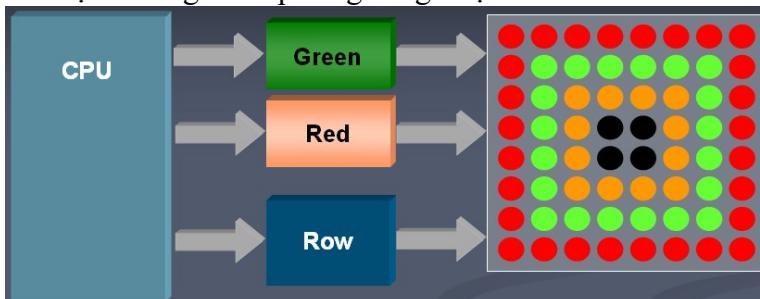
6.2 Thực hiện yêu cầu 1: xuất dữ liệu ra 1 hàng LED ma trận

Bước 1. Kết nối mạch.

Mạch LED mở rộng gồm có 8 LED 7 đoạn và 1 LED ma trận hai màu xanh/đỏ (xem hình).



Ma trận LED giao tiếp song song được thiết kế theo hình sau:



Trong đó, các cổng Green, Red, và Row đều là các cổng 8 bit xuất. Ta dùng 3 sợi cáp để kết nối với connector theo như sau:

PortC : Green (dữ liệu cột xanh)

PortD : Red (dữ liệu cột đỏ)

PortB : Row (chọn hàng 0-7)

Bước 2. Tạo dự án mới Tn06, tập tin nguồn Tn06.asm.

Bước 3. Khởi động các port và nút nhấn RA5.

```
#define ROW LATB
#define ROW_IO TRISB
#define GCOL LATC
#define GCOL_IO TRISC
#define RCOL LATD
#define RCOL_IO TRISD
#define NUTNHAN PORTA, RA5
#define NUT_IO TRISA, RA5
```

Bước 4. Để xuất dữ liệu cho ma trận LED sáng hàng 0 như hình sau:



Điều khiển trộn màu tuân theo quy luật sau:

Màu đỏ xuất hiện là do LED xanh tắt (bit dữ liệu xanh=0), LED đỏ sáng (bit dữ liệu đỏ=1).

Màu xanh là do LED xanh sáng, LED đỏ tắt.

Màu cam là do LED xanh, LED đỏ cùng sáng.

Màu đen là do LED xanh, LED đỏ cùng tắt.

Sinh viên cần xác định dữ liệu xanh G_DATA (8 bit) và dữ liệu đỏ R_DATA (8 bit) sao cho khi trộn các bit tương ứng lại sẽ cho ra màu theo mẫu yêu cầu.

Trong **main** đánh đoạn code thực hiện quy trình sau:

Xoá tất cả các hàng (nghĩa là tắt hết các hàng, tuy nhiên do hiện tượng lưu ảnh trên LED nên ta vẫn thấy LED sáng).

Xuất G_DATA ra cổng dữ liệu xanh.

Xuất R_DATA ra cổng dữ liệu đỏ.

Chọn hàng 0 bằng cách xuất dữ liệu B'00000001' ra cổng chọn hàng.

Bước 5. Chạy chương trình và quan sát.

Bước 6. Thêm code kiểm tra nút RA5 thực hiện xoay vòng chọn hàng từ hàng 0 đến hàng 7 như sau:

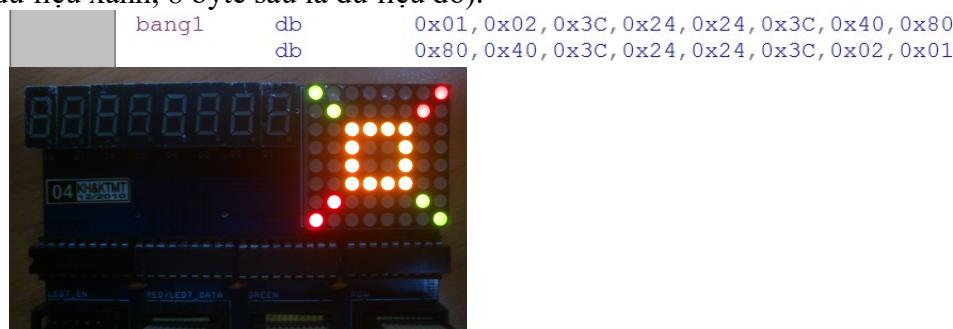
```
main1    btfsc    NUTNHAN
        bra     main1
        rlnclf   ROW
nhaphim  btfss    NUTNHAN
        bra     nhaphim
        bra     main1
```

Bước 7. Chạy chương trình và quan sát.

6.3 Thực hiện yêu cầu 1: xuất hình ra LED ma trận

Bước 8. Để xuất 1 khung hình có 8 hàng ra ma trận LED, ta cần luân phiên xuất dữ liệu ra từng hàng từ hàng 0 đến hàng 7 theo chỉ số **idx**.

Dữ liệu của 8 hàng sẽ được tổ chức theo bảng trong ROM thành 16 byte liên tiếp (8 byte đầu là dữ liệu xanh, 8 byte sau là dữ liệu đỏ).



Định nghĩa trong RAM vùng đệm dữ liệu led 16 byte (gồm **gbuff** là 8 byte đầu, **rbuf** là 8 byte cuối).

```
udata_acs
MAXIDX    equ      .8
idx        res      .1
gbuff     res      .8
rbuf      res      .8
rsel      res      .1
dem       res      .1
```

Thực chép 16 byte dữ liệu từ **Bang1** trong ROM sang **gbuff** của RAM.

```
rom2ram
        movlw   upper bang1
        movwf   TBLPTRU
        movlw   high bang1
        movwf   TBLPTRH
        movlw   low bang1
        movwf   TBLPTRL
        lfsr    FSR0, gbuff
        movlw   .16
        movwf   dem

c大海1
        TBLRD*+
        movf    TABLAT, W
        movwf   POSTINCO
        decfsz dem
        bra   大海1
        return
```

Bước 9. Thêm module Timer sử dụng Timer0 10 ms. Trong Timer_process thực hiện quy trình quét LED theo chỉ số **idx**.

Thực hiện kỹ thuật tra bảng RAM để lấy dữ liệu LED và xuất ra cổng dữ liệu tương ứng.

```
Timer_process
        clrf    ROW      ;tat het cac hang
        lfsr    FSR1, gbuff
        lfsr    FSR2, rbuf
        movf    idx, W   ;du lieu xanh
        movf    PLUSW1, W
        movwf   GCOL
        movf    idx, W   ;du lieu do
        movf    PLUSW2, W
        movwf   RCOL
        movf    rsel, W  ;chon hang
        movwf   ROW
        rlncl  rsel      ;hang ke tiep
        incf    idx      ;tang chi so
        movlw   MAXIDX
        cpfslt idx
        clrf    idx
        return
```

Bước 10. Chạy chương trình và quan sát.

Bước 11. Điều chỉnh lại thời gian quét để ma trận LED không bị chớp.

6.4 Thực hiện yêu cầu 2: hiển thị số ra LED 7 đoạn.

Bước 12. Kết nối lại dây cáp như sau:

PortB : LED_EN (SELECT)

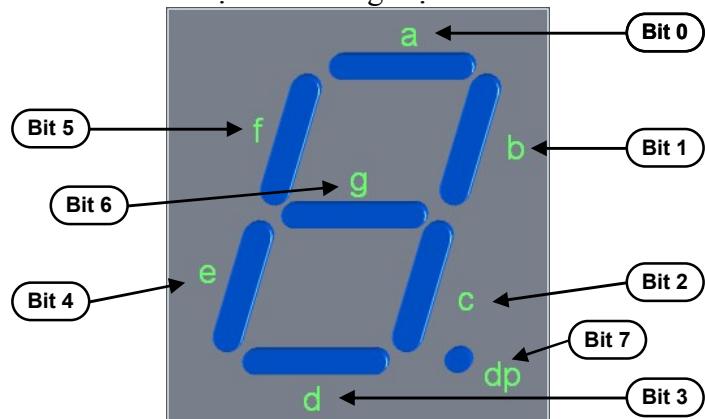
PortD : RED (SMENT)

Bước 13. Tạo file mới Tn06B.asm, chuẩn bị code sử dụng ngắt Timer0.

Bước 14. Dữ liệu xuất ra 8 LED 7 đoạn gồm 8 byte (chứa số từ 0 đến 9).

Bước 15. Tuy nhiên, do đặc thù của LED 7 đoạn là các nét dữ liệu (a-g) và chấm dp nên để hiện được số từ 0 đến 9, dữ liệu phải qua bước giải mã 7 đoạn như sau:

Kết nối nét và bit dữ liệu theo mạch như hình sau:



Sáng số 0: các nét a-b-c-d-e-f = 1, các g-dp = 0, dữ liệu sẽ là B'00111111'=H'3F'.
Sáng số 1: dữ liệu là H'06'.

```
...
    gm_7doan
        movwf      TBLPTRL
        movlw      low Bang_7s
        addwf      TBLPTRL
        movlw      high Bang_7s
        movwf      TBLPTRH
        clrf       WREG
        addwfc      TBLPTRH
        movlw      upper Bang_7s
        movwf      TBLPTRU
        TBLRD*
        movf       TABLAT,W
        return
    Bang_7s   db      H'3F',H'06',H'5B',H'4F',H'66',H'6D',H'7D',H'07'
                db      H'7F',H'6F',H'77',H'7C',H'39',H'5E',H'79',H'71'
```

Bước 16. Đoạn code xuất số 9 ra vị trí LED4 như sau:

```
SO      equ     .9
LED4    equ     B'00010000'

PRG
start  rcall   init
main   clrf    SELECT      ;tat cac LED
        movlw   SO          ;du lieu
        rcall   gm_7doan    ;giai ma 7 doan
        movwf   SMENT
        movlw   LED4        ;chon LED4 sang
        movwf   SELECT
        bra    main1
```

Bước 17. Kết hợp Timer0 10 ms để hiển thị số trên cả 8 LED.

6.5 Bài tập

- Viết chương trình hiển thị hình thay đổi theo quy luật lên ma trận LED, 0.5s thay đổi 1 lần. (Gợi ý: tạo nhiều frame hình, 0.5s đổi frame một lần bằng cách chép frame từ ROM sang buffer RAM).
- Viết dự án thực hiện số chạy (3 số) từ trái sang phải màn hình LED 7 đoạn.
- Viết chương trình hiện đồng hồ (hh.mm.ss) lên LED 7 đoạn.

Bài 7 : Khảo sát bộ truyền nhận nối tiếp

Nội dung:

- Nd1. Khảo sát bộ truyền nhận nối tiếp UART của PIC.
- Nd2. Trao đổi thông tin giữa 2 PICDEM PIC18 với nhau, sử dụng cáp chéo.
- Nd3. Trao đổi thông tin giữa PICDEM PIC18 với PC qua cổng COM (RS-232C), phần mềm Terminal, sử dụng cáp thẳng.

Yêu cầu:

- Yc1. Viết chương trình sử dụng khói truyền nhận nối tiếp UART của PIC. Lập trình chọn chế độ hoạt động, truyền trực tiếp, nhận qua ngắt quãng và hiện ra LCD.
- Yc2. Thực hiện truyền nhận ký tự giữa 2 PICDEM PIC18 với nhau.
- Yc3. Thực hiện truyền nhận ký tự giữa PICDEM PIC18 với PC.

7.1 Thực hiện yêu cầu 1:

Bước 1. Tạo dự án mới Tn07, tập tin nguồn Tn07.asm.

Bước 2. Khởi động LCD, nút nhấn RA5.

Bước 3. Tạo module Serial.asm chứa các hàm:

Serial_init : khởi động UART.

Serial_isr : xử lý ngắt quãng, gọi hàm Serial_process (xử lý ký tự nhận được) và Rcerr_process (xử lý khi có lỗi trong quá trình nhận ký tự) trong module Tn07.asm.

Send_char : truyền một ký tự ra cổng UART.

Bước 4. Khởi động khói EUART1 để hoạt động ở chế độ sau:

Chế độ truyền bất đồng bộ (Asynchronous), 8 bit, High speed bằng thanh ghi TXSTA1.

REGISTER 20-1: TXSTAx: TRANSMIT STATUS AND CONTROL REGISTER

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-1	R/W-0
CSRC	TX9	TXEN	SYNC	SEND8	BRGH	TRMT	TX9D
bit 7							bit 0

Cấu hình yêu cầu có TXSTA1 = B'00100100'.

Cấu hình chế độ nhận bằng thanh ghi RCSTA1.

REGISTER 20-2: RCSTAx: RECEIVE STATUS AND CONTROL REGISTER

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	R-0	R-x
SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D
bit 7							bit 0

Cấu hình yêu cầu có RCSTA1 = B'10010000'.

Tốc độ truyền nhận 9600 bit/s. Sinh viên tra bảng số liệu có sẵn trong data sheet (table 20-3, p.253) để chọn giá trị cho bộ phát xung SPBRG.

Cấu hình ngắt nhận dữ liệu với mức ưu tiên cao.

Định nghĩa một số hằng

```
#define SSP_IN    TRISC, RC7
#define SSP_OUT   TRISC, RC6
#define TX_CFG    B'00100100'
#define RC_CFG    B'10010000'
```

Hàm Serial_init

```

Serial_init
    bsf      SSP_IN          ;ngõ nhận dữ liệu nối tiếp
    bcf      SSP_OUT         ;ngõ truyền dữ liệu nối tiếp
    movlw   TX_CFG          ;cấu hình truyền
    movwf   TXSTA1
    movlw   RC_CFG          ;cấu hình nhận
    movwf   RCSTA1
    movlw   BAUDRATE        ;tốc độ truyền/nhận
    movwf   SPBRG
    bsf      RCON, IPEN      ;cho phép ưu tiên
    bsf      IPR1, RC1IP     ;nhận ưu tiên cao
    bcf      PIR1, RC1IF     ;xóa bỏ ngắt nhận
    bsf      PIE1, RC1IE     ;cho phép ngắt nhận
    bsf      INTCON, GIEH     ;cho phép ngắt toàn cục
    bsf      INTCON, GIEL
    return

```

Bước 5. Hàm **Serial_isr** thực hiện các việc sau:

Kiểm tra xác nhận có ngắt nhận dữ liệu nối tiếp (cờ RC1IF), nếu có thì xoá cờ ngắt và làm tiếp các bước kế.

Kiểm tra 2 bit lỗi FERR và OERR trong thanh ghi RCSTA1. Nếu có lỗi thì thực hiện xoá lỗi và gọi **Rcerr_process**, không lỗi thì gọi **Serial_process**.

```

Serial_isr
    btfss   PIR1, RC1IF
    return
    movlw   RCERROR         ;không phải ngắt nhận
    andwf   RCSTA1, W
    btfss   STATUS, Z
    bra     Rcv_error
    rcall   Serial_process
    return
    Rcv_error
        bcf    RCSTA1, CREN    ;xoá lỗi
        bsf    RCSTA1, CREN
        rcall  Rcerr_process
        return

```

Bước 6. Hàm **Send_char** thực hiện truyền 1 ký tự trong biến **tx_char**:

```

global  Send_char
    Send_char
        movf   tx_char, W
        movwf  TXREG1
    send1
        btfss  PIR1, TX1IF    ;truyền xong TX1IF=1
        bra    send1
    return

```

Bước 7. Viết hàm **Serial_process** thực hiện các việc sau:

Nhận ký tự từ RCREG1.

Hiện ký tự nhận được ra LCD ở hàng 2, từ trái sang phải, đến cuối thì quay trở về đầu hàng.

Xuất ký tự ra LED.

Bước 8. Viết hàm **Rcerr_process** thực hiện xuất trị H'FF' ra LED.**Bước 9.** Kiểm tra nút RA5, gọi hàm **Truyen** thực hiện:

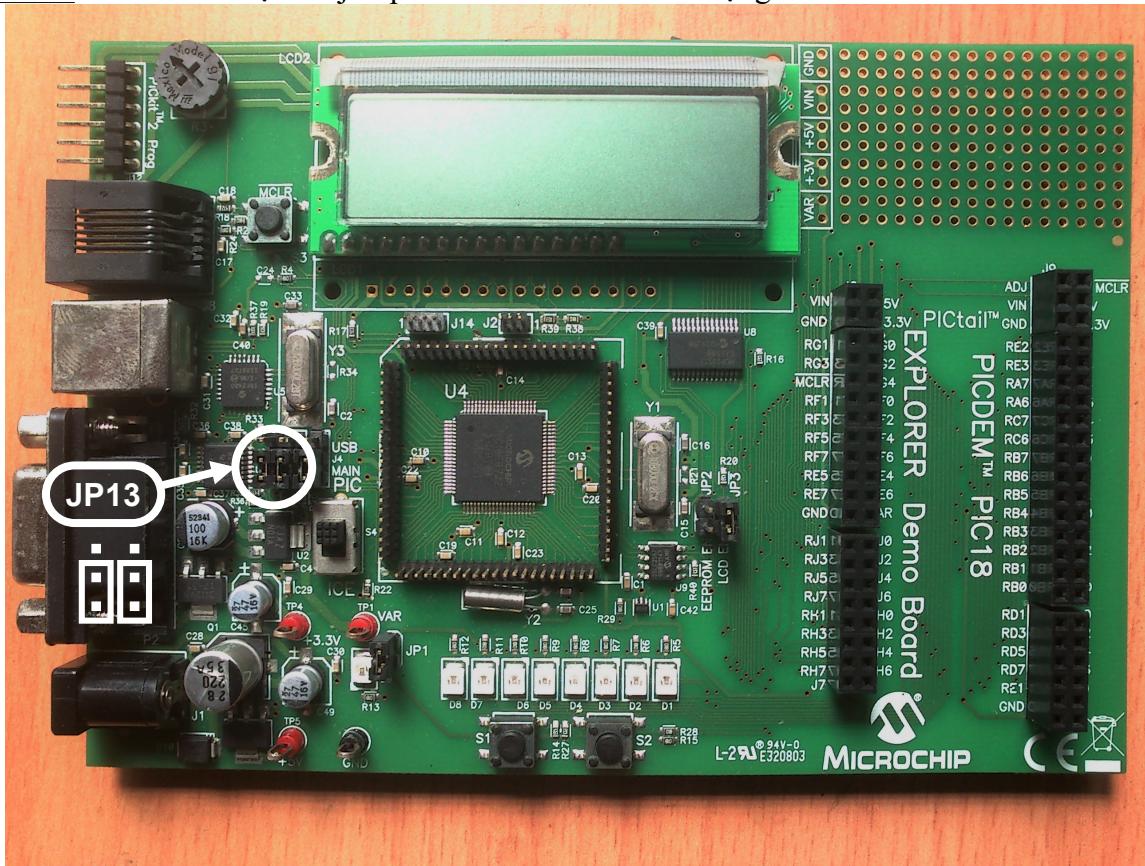
Truyền ký tự trong biến **char** (gọi hàm **Send_char** để truyền).

Hiện ký tự đã truyền ra LCD ở hàng 1, từ trái sang phải, đến cuối thì quay trở về đầu hàng.

Tăng mã ký tự trong biến **char**.

7.2 Thực hiện yêu cầu 2: truyền/nhận PICDEM PIC18 với nhau

Bước 10. Kiểm tra và đặt các jumper JP13 như hình để sử dụng connector DB9.



Bước 11. Dùng cáp chéo kết nối 2 connector DB9 của 2 mạch PICDEM PIC18 với nhau, chạy chương trình và quan sát.

7.3 Thực hiện yêu cầu 3: truyền/nhận PICDEM PIC18 với PC

Bước 12. Nếu PC có đầu kết nối DB9 thì dùng cáp thẳng nối giữa PICDEM PIC18 với PC.

Bước 13. Trên PC, chạy chương trình Terminal.exe để thực hiện truyền nhận.

Bước 14. Với PC không hỗ trợ kết nối DB9, thì việc truyền nhận sẽ được thực hiện qua cổng USB (Sinh viên làm thêm).

7.4 Bài tập

- Thực hiện truyền nhận với tốc độ 1200 bit/s, 2400 bit/s.
- Viết chương trình điều khiển PICDEM PIC18 từ xa thông qua UART từ một PICDEM PIC18 khác.

Bài 8 : Khảo sát khói chuyển đổi A-D

Nội dung:

Nd1. Khảo sát hoạt động khói chuyển đổi A-D.

Nd2. Khảo sát các thanh ghi điều khiển hoạt động khói chuyển đổi A-D.

Yêu cầu:

Yc1. Viết chương trình (không dùng ngắt quãng) đọc và hiển thị liên tục giá trị điện áp thay đổi bởi biến trả ra LCD.

Yc2. Làm lại yêu cầu 1 nhưng dùng ngắt quãng AD để đọc kết quả.

Yc3. Làm lại yêu cầu 2 nhưng dùng thêm ngắt quãng Timer để kiểm soát thời gian AD một giây một lần.

8.1 Các bước hiện thực yêu cầu 1.

Bước 1. Tạo dự án mới Tn08, tập tin nguồn Tn08.asm.

Bước 2. Khởi động LCD, nút nhấn RA5.

Bước 3. Tạo module Adc.asm và viết hàm Adc_init thực hiện cấu hình các thanh ghi:

ADCON0: Chọn kênh AD (ANi), cho phép AD (ADON), bắt đầu AD (cho GO/DONE=1) và kết thúc AD (kiểm tra GO/DONE=0).

REGISTER 21-1: ADCON0: A/D CONTROL REGISTER

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	CHS3 ⁽¹⁾	CHS2 ⁽¹⁾	CHS1 ⁽¹⁾	CHS0 ⁽¹⁾	GO/DONE	ADON
bit 7							bit 0

ADCON1: Cho phép ngõ nhập ANi là Analog.

REGISTER 21-2: ADCON1: A/D CONTROL REGISTER 1

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	VCFG1	VCFG0	PCFG3	PCFG2	PCFG1	PCFG0
bit 7							bit 0

ADCON2: ADFM chọn chỉnh biên kết quả 16 bit (trái=10 bit cao / phải=10 bit thấp); ADCS1 chọn nguồn xung lấy mẫu.

REGISTER 21-3: ADCON2: A/D CONTROL REGISTER 2

R/W-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
ADFM	—	ACQT2	ACQT1	ACQT0	ADCS2	ADCS1	ADCS0
bit 7							bit 0

Cấu hình ngõ nhập cho kênh được chọn ANi.

```

AD_CFG0  equ      H'01' ;chon AN0, GO=0, ADON=1
AD_CFG1  equ      H'0E' ;AN0=Analog
AD_CFG2  equ      H'81' ;ADFM=1(right justified), CLK=FOSC/8
Adc_init
    movlw   AD_CFG0 ;chon kenh
    movwf   ADCON0
    movlw   AD_CFG1 ;chon analog
    movwf   ADCON1
    movlw   AD_CFG2 ;chon clock, chinh bien ket qua
    movwf   ADCON2
    bsf    AN0_IO
    return

```

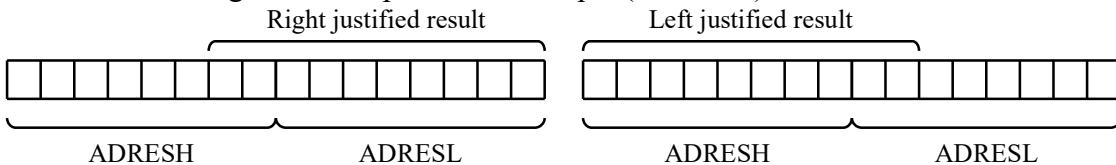
Bước 4. Để bắt đầu quá trình chuyển đổi AD, ta phải ra lệnh bằng cách cho bit GO/DONE=1. Sau đó, chờ cho đến khi bộ ADC chuyển đổi xong (kiểm tra bit GO/DONE=0) thì đọc kết quả trong 2 thanh ghi ADRESH-ADRESL cất vào biến 2 byte ad_res. Gọi hàm Adc_process để xuất kết quả ra LCD.

```

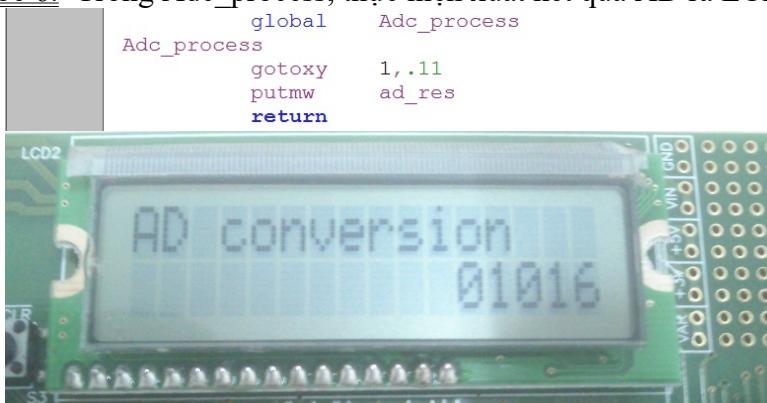
    Adc_go    bsf      ADCONO, GO      ;GO=1, bat dau qua trinh AD
    Adc_wait  btfsc   ADCONO, DONE    ;cho DONE=0, AD xong
    bra      Adc_wait
    movf     ADRESH, W      ;doc ket qua cao
    movwf    ad_res+1
    movf     ADRESL, W      ;doc ket qua thap
    movwf    ad_res
    rcall   Adc_process
    return

```

Bước 5. Do ta chọn chỉnh biến phải kết quả nên ADRESH giữ 2 bit cao của kết quả và ADRESL giữ 8 bit thấp nhất của kết quả (xem hình).



Bước 6. Trong Adc_process, thực hiện xuất kết quả AD ra LCD (xem hình).



8.2 Các bước hiện thực yêu cầu 2.

Bước 7. Thêm phần khởi động ngắt quãng AD vào cuối hàm Adc_init như sau:

```

    bsf      RCON, IPEN      ;su dung uu tien
    bsf      IPR1, ADIP      ;ngat AD uu tien cao
    bcf      PIR1, ADIF      ;xoa co ngat
    bsf      PIE1, ADIE      ;cho phép ngat AD
    bsf      INTCON, GIEH    ;cho phép ngat toàn cục
    bsf      INTCON, GIEL
    bsf      ADCONO, GO      ;bat dau AD luon
    return

```

Bước 8. Viết thêm hàm Adc_isr thực hiện:

Xác nhận có ngắt AD bằng cách kiểm tra xem ADIF có bằng 1 hay không. ADIF=1 thì làm tiếp các bước kế, ADIF=0 trở về.

Xóa cờ ngắt.

Đọc kết quả AD vào biến ad_res.

Gọi hàm Adc_process để xử lý kết quả.

Cuối cùng, bật bit GO/DONE lên 1 cho phép AD tiếp để lấy kết quả liên tục.

Bước 9. Chạy thử chương trình.

8.3 Các bước hiện thực yêu cầu 3.

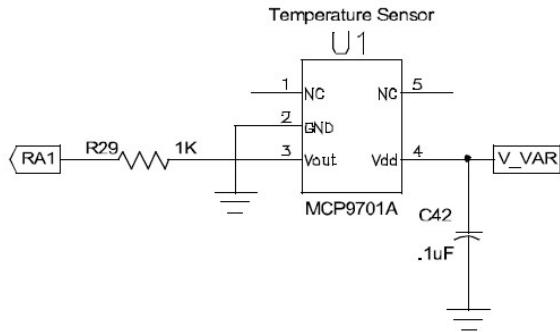
Bước 10. Thêm module Timer vào, trong Timer_process định thời 1 giây thì bật bit GO/DONE lên 1. Như vậy, ta sẽ **không bật** bit GO/DONE lên 1 trong hàm Adc_isr nữa.

Bước 11. Chạy thử và quan sát sự thay đổi giá trị kết quả trên LCD.

8.4 Bài tập

- Thực hiện AD trên kênh AN5 (port RF0). Nối dây trên connector mở rộng giữa chân RA0 với chân RF0 để có thể sử dụng biến trỏ RA0 cho kênh RF0.

- b) Viết chương trình dùng biến trỏ RA0 để điều khiển LED PORTD sáng từ 1 LED đến 8 LED tùy theo vị trí biến trỏ.
- c) Sử dụng module ADC của Pic để đo nhiệt độ trong phòng với cảm biến MCP9701A có sẵn, dùng LCD để hiển thị giá trị nhiệt độ.



Bài 9 : Khảo sát khối phát xung PWM

Nội dung:

Nd1. Khảo sát khối chức năng phát xung PWM.

Yêu cầu:

Yc1. Viết chương trình sử dụng chức năng PWM phát xung có dạng như yêu cầu.

Yc2. Dùng xung PWM điều khiển độ sáng của LED.

9.1 Các bước hiện thực PWM

Bước 1. Tạo dự án Th09, release, tạo file Tn09.asm, cấu hình dự án (OSC, WDT, LVP).

Bước 2. Khởi động nút nhấn RA5.

Bước 3. Viết hàm khởi động Pwm_init việc phát xung theo dạng xung yêu cầu như sau:

Chọn IO cho chân phát xung P1A (là chân RC2 [xem 8722_DS p.189]).

Chọn giá trị các bit của CCP1CON để phát xung theo chế độ: Single output P1A, phát xung active high (mức 1 trước, mức 0 sau).

REGISTER 17-1: CCPxCON: CCPx CONTROL REGISTER (CCP4 AND CCP5 MODULES)

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	DCxB1	DCxB0	CCPxM3	CCPxM2	CCPxM1	CCPxM0
bit 7							bit 0

bit 7-6 **PxM1:PxM0:** Enhanced PWM Output Configuration bits

If CCPxM<3:2> = 11:

00 = Single output: Px A modulated; Px B, Px C, Px D assigned as port pins

bit 5-4 **DCxB<1:0>:** PWM Duty Cycle bit 1 and bit 0

PWM mode:

These bits are the two LSbs of the 10-bit PWM duty cycle. The eight MSbs of the duty cycle are found in CCPRL.

bit 3-0 **CCPxM3:CCPxM0:** Enhanced CCP Mode Select bits

1100 = PWM mode: Px A, Px C active-high; Px B, Px D active-high

1101 = PWM mode: Px A, Px C active-high; Px B, Px D active-low

1110 = PWM mode: Px A, Px C active-low; Px B, Px D active-high

1111 = PWM mode: Px A, Px C active-low; Px B, Px D active-low

Khởi động Timer 2 bằng thanh ghi T2CON với postscale=1, prescale=4.

REGISTER 14-1: T2CON: TIMER2 CONTROL REGISTER

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	T2OUTPS3	T2OUTPS2	T2OUTPS1	T2OUTPS0	TMR2ON	T2CKPS1	T2CKPS0
bit 7							bit 0

bit 7 **Unimplemented:** Read as '0'

bit 6-3 **T2OUTPS<3:0>:** Timer2 Output Postscale Select bits

bit 2 **TMR2ON:** Timer2 On bit

1 = Timer2 is on

bit 1-0 **T2CKPS<1:0>:** Timer2 Clock Prescale Select bits

00 = Prescaler is 1

01 = Prescaler is 4

1x = Prescaler is 16

Khởi động period thông qua giá trị PR2.

Khởi động duty cycle thông qua giá trị 10 bit (CCPR1L₇₋₀:CCP1X:CCP1Y).

Hàm Pwm_init như sau:

```

    PWM_CFG  equ      H'0C'
    T2_CFG   equ      H'05'
    PR2_VAL  equ      .249
    DUTY8_VAL equ      .120
    #define  PWMOUT_IO TRISC, RC2

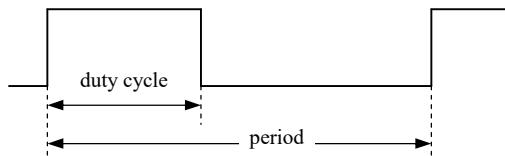
```

```

Pwm_init
    bcf      PWMOUT IO
    movlw   PWM_CFG
    movwf   CCP1CON
    movlw   T2_CFG
    movwf   T2CON
    movlw   PR2_VAL
    movwf   PR2
    movlw   DUTY8_VAL
    movwf   CCPR1L
    bcf      CCP1CON, CCP1X
    bcf      CCP1CON, CCP1Y
    return

```

Bước 4. Chu kỳ (period) và tỷ lệ xung active (duty cycle) được tính như cách sau:



$$\text{period} = (\text{PR2} + 1) * 4 * \text{Tosc} * \langle \text{Tmr2 prescale} \rangle$$

$$\text{duty cycle} = x * \text{Tosc} * \langle \text{Tmr2 prescale} \rangle$$

trong đó: x là giá trị 10 bit ghép lại từ các thành phần (CCPR1L:CCP1X:CCP1Y) theo tỉ lệ (8:1:1)

CCPR1L là thanh ghi 8 bit (CCPR1L= x chia 4 lấy phần nguyên).

CCP1X là bit 5 của thanh ghi CCP1CON (CCP1X:CCP1Y= x chia 4 lấy phần dư).

CCP1Y là bit 4 của thanh ghi CCP1CON.

Với số liệu trong mạch thí nghiệm ta có :

$$\text{Tosc} = 1 / 10 \text{ MHz}$$

$$\text{PR2} = 249$$

$$\text{CCPR1L} = 120$$

$$\text{CCP1X} = 0$$

$$\text{CCP1Y} = 0$$

$$4 * 10^4$$

$$1.92 * 10^{-4}$$

$$\text{Sinh viên xác định : period} = \dots \dots \dots ,$$

$$\text{duty cycle} = \dots \dots \dots$$

Bước 5. Chạy thử chương trình.

Bước 6. Dùng dao động kíp đo lại giá trị period và duty cycle đã tính được ở chân RC2

Bước 7. Sửa lại câu hình để phát xung theo hình sau, chú ý active low :

$$\boxed{\text{Prescale} = 16}$$



9.2 Thực hiện yêu cầu 2: điều chỉnh độ sáng LED

Bước 8. Dự án Tn09B, file SangLed.asm.

Bước 9. Thực hiện phát xung PWM active high với chu kỳ 1.024 ms, duty cycle là 1/8 chu kỳ.

Bước 10. Nhấn nút RA5 hoặc dùng timer 1s tăng duty cycle lên 1/8 chu kỳ mỗi lần. Nếu duty cycle tăng đến 8/8 chu kỳ rồi thì sau đó quay về 1/8 chu kỳ và cứ thế tiếp tục.

Bước 11. Chạy chương trình và cắm 1 đèn LED đã có hạn dòng vào chân RC2 để quan sát.

9.3 Bài tập

- a) Thực hiện phát xung sau ra RC2:



- b) Thực hiện điều khiển LED sáng dần lên rồi tối dần xuống với thời gian 100ms.

--- Kết ---