Vi điều khiển PIC - Học nhanh đi vào ứng dụng

Lời khuyên lựa chọn bộ công cụ làm việc với PIC

Mạch nạp: Falleaf PG2C - PIC Tutorial
Chương trình nạp: IC - PROG
Bootloader: Tiny bootloader
Debugger: ICD2 Clone
Chương trình dịch: MPLAB IDE - CCS C

Lựa chọn PIC:

- Mới học: PIC16F628A hoặc PIC16F88
- Học tổng hợp: PIC16F877A
- Làm đề tài: PIC16F876A
- Cần mạnh hơn: PIC18F458
Các loại trên đều có thể dùng PG2C và IC-PROG 1.05D

- Điều khiển động cơ: PIC18F4331, PIC18F4431 (ICD2)
- Lập trình thuật toán: dsPIC30Fxxxx (dùng ICD2)

Để tránh mất thời gian các câu hỏi thường được lặp đi lặp lại về PIC, các bạn mới học về PIC lưu ý bài viết này. Bài viết này sẽ được update liên tục khi có các thông tin mới.

0) Một vài điều cơ bản về PIC

- PIC16F84 là dòng PIC phổ biến nhất được khuyến khích cho những người mới học. Tuy nhiên, gần đây, dòng PIC16F628A ra đời, giá thấp hơn, nhiều chức năng hơn, và thực sự là dòng PIC Flash. Nó được hầu hết các chuyên gia khuyên dùng để bắt đầu thực hành về PIC. Hầu hết các tutorial mới đều bắt đầu chọn PIC16F628A. Tuy nhiên, hiện nay dòng PIC16F88 mới ra đời, cũng như sự ra đời của PIC16F628A, PIC16F88 có nhiều chức năng hơn PIC16F628A, giá cả không chênh lệch là bao (khoảng 5000 đến 10000 đồng tại Việt Nam), và nó hỗ trợ gần như toàn bộ chức năng của một vi điều khiển hiện đại. Do vậy, chúng tôi khuyên các bạn nên chọn PIC16F628A hoặc PIC16F88 để bắt đầu học về PIC

Thời điểm tháng 05 năm 2005
- Giá hiện nay của dòng PIC 18 chân dao động từ 20.000 đồng đến 50.000 đồng mỗi con
- Giá dòng PIC16Fxxxx dao động từ 40.000 đồng đến 150.000 đồng
- Giá dòng PIC18Fxxxx dao động từ 100.000 đồng đến 300.000 đồng
- Giá dòng dsPIC dao động từ 150.000 đồng đến 350.000 đồng hoặc hơn
- Giá dòng rfPIC dao động từ 50.000 đồng đến 100.000 đồng

Đánh giá các dòng PIC
- Dòng PIC nhiều chân nhất là dòng PIC18Fxxxx, có những con số chân lên đến 80 chân
- Dòng PIC ít chân nhất là dòng PIC10Fxxx, chỉ có 6 chân
- Dòng PIC phổ biến nhất là dòng PIC16F877A (đủ mạnh về tính năng, 40 chân, bộ nhớ đủ cho hầu hết các ứng dụng thông thường)
- Dòng PIC mà chúng tôi đánh giá cao nhất là dòng PIC16F876A (28 chân, chức năng không khác gì so với PIC16F877A, nhưng nhỏ gọn hơn nhiều, và số chân cũng không quá ít như PIC16F88).
- Dòng PIC hỗ trợ giao tiếp USB là dòng PIC18F2550 và PIC18F4550
- Dòng PIC điều khiển động cơ mạnh nhất là dòng PIC18F4x31
- Khi cho rằng mình chuyên nghiệp hơn, các bạn nên dùng PIC18F458
- dsPIC chúng tôi khuyên không nên dùng và không nên nghĩ tới khi mới học, bản thân chúng tôi cũng chưa có điều kiện làm việc với dsPIC mặc dù về lập trình thì dsPIC hoàn toàn giống với PIC thông thường.
- Dòng PIC tàng hình là dòng PIC17xxxxx, hiện nay đã không còn được sản xuất

1) Mạch nạp PIC, Bootloaders và các chương trình nạp tương ứng
Mạch nạp
http://www.olimex.com/
Trang web này cung cấp rất nhiều loại mạch nạp của PIC, có sơ đồ nguyên lý đầy đủ, và tất cả các hướng dẫn liên quan đến việc cài đặt và sử dụng mạch nạp. Trong tài liệu hướng dẫn PIC Tutorial, chúng tôi chọn sử dụng mạch nạp PG2C để hướng dẫn.
http://siscobf.webcindario.com/winpic800.htm
Hơi khó coi một chút vì nó là tiếng Tây Ban Nha hay sao đó? Nhưng không vấn đề gì, các bạn download về, tự động sẽ hiểu phải làm thế nào. Tôi vẫn chủ trương, người chưa biết gì dùng PG2C.
In Circuit Debugger
http://www.stolz.de.be/
ICD2 Clone, nạp được hầu hết các loại PIC hiện có, hỗ trợ debug trong mạch và quan trọng nhất là nạp được cho dòng dsPIC30F
Bootloader
http://www.ac.ugal.ro/staff/ckiku/software/picbootloader.htm
Đây là bộ tinybootloader, là bộ bootloader xịn nhất cho đến bây giờ mà tôi biết.
http://www.dontronics.com/rfarmer.html
http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1824&appnote=en012031
Microchip bootloader, chỉ hỗ trợ dòng 16F, nhưng là bootloader chính thức của hãng, cung cấp miễn phí
Chương trình nạp
http://www.ic-prog.com/
2) Các chương trình dịch
Chương trình MPLAB IDE : http://www.microchip.com/
Chương trình CCS C (phiên bản 3.222 có crack): download tại đây
Chương trình HT PIC (phiên bản 8.05PL2 ngày 27/9/2004, có crack): download tại đây
Chương trình HT PIC18 (phiên bản demo): http://www.hitech.com/
Hướng dẫn cài đặt: MPLAB, CCS C, HT PIC, HT PIC18 , download tất cả
3) Các tài liệu hướng dẫn
- Chúng tôi đăng toàn văn các tài liệu hướng dẫn trong luồng TÀI LIỆU HƯỚNG DẪN TIẾNG ANH để các bạn tiện download.
Lưu ý rằng, chúng tôi đăng những tài liệu này bằng file .pdf để thuận tiện cho việc download, đọc trên máy và in ấn. Chúng tôi không muốn đăng file .doc vì lý do không muốn các bạn mới học thuận tay copy and paste. Chúng tôi hy vọng rằng thời gian đầu mới học, các bạn nên kiên nhẫn học từng dòng lệnh, cách trình bày để hiểu rõ nội dung. Ngoài ra, theo những đánh giá cá nhân, những tài liệu hướng dẫn này không giống như một thư viện source code, cách thực hiện tối ưu hoá từng đề tài một, nên cũng không phù hợp với các bạn mới học.
- Tài liệu hướng dẫn tiếng Việt đang được thực hiện, và sẽ đăng từng phần trong luồng TÀI LIỆU HƯỚNG DẪN TIẾNG VIỆT.
Tài liệu này cũng cung cấp dạng file .pdf để tránh sao chép, vì lý do chúng tôi muốn soạn thảo hoàn thiện tài liệu này trước khi công bố, và đây cũng là mục đích chính của diễn đàn picvietnam.
4) Hướng dẫn mạch nạp Falleaf PG2C - PIC Tutorial
- Tài liệu hướng dẫn này được đăng tại luồng Falleaf PG2C - PIC Tutorial
- Các bạn có thể tìm mua mạch nạp này và đĩa CD đi kèm thông qua
phungtbinh@yahoo.com (Hà Nội)
myfrienddang@yahoo.com (TPHCM)
với giá 35.000 đồng/bộ
5) Các địa chỉ tìm source code của PIC
http://www.piclist.com/ (địa chỉ nhiều source code của PIC nhất trên đời)
6) Các forum tiếng Anh về PIC
- Forum chuyên về MPASM, có sự tham gia của Nigel Goodwin:
http://www.electro-tech-online.com/
- Forum chuyên về CCS C, do chính CCS C info xây dựng:
http://www.ccsinfo.com/forum/viewforum.php?f=1
- Forum hướng dẫn của Olimex và SparkFun:
http://www.sparkfun.com/
Trang web này hướng dẫn các mạch do Olimex cung cấp, hay nói cách khác SparkFun là forum

của Olimex.

7) http://www.microchip.com/

Trang web chính của Microchip PIC, cung cấp:
- datasheet
- diễn đàn chính của Microchip PIC (nhưng không sôi nổi lắm)
- môi trường soạn thảo và trình dịch MPLAB (luôn có phiên bản mới nhất)
- bán các linh kiện (PIC, dsPIC, rfPIC, mạch nạp, chương trình dịch, linh kiện analog...)
- bootloader chính thức của PIC dùng cho 16F877A và 16F876A

8) Một số trang web mua bán các công cụ hỗ trợ PIC, các sản phẩm từ PIC...

http://www.ccsinfo.com/ (bán chương trình CCS C cho PIC)
http://www.dontronics.com/dt101.html (bán một số sản phẩm điện tử)
http://www.digikey.com/ (bán một số sản phẩm điện tử)
http://www.phanderson.com/PIC/PICC/index.html (địa chỉ mua trình dịch và thư viện source code)
http://www.diendandientu.com/ (trong luồng Mua Bán Linh Kiện có một số người buôn bán lẻ các sản phẩm PIC)

9) Các đề tài thực hiện với PIC

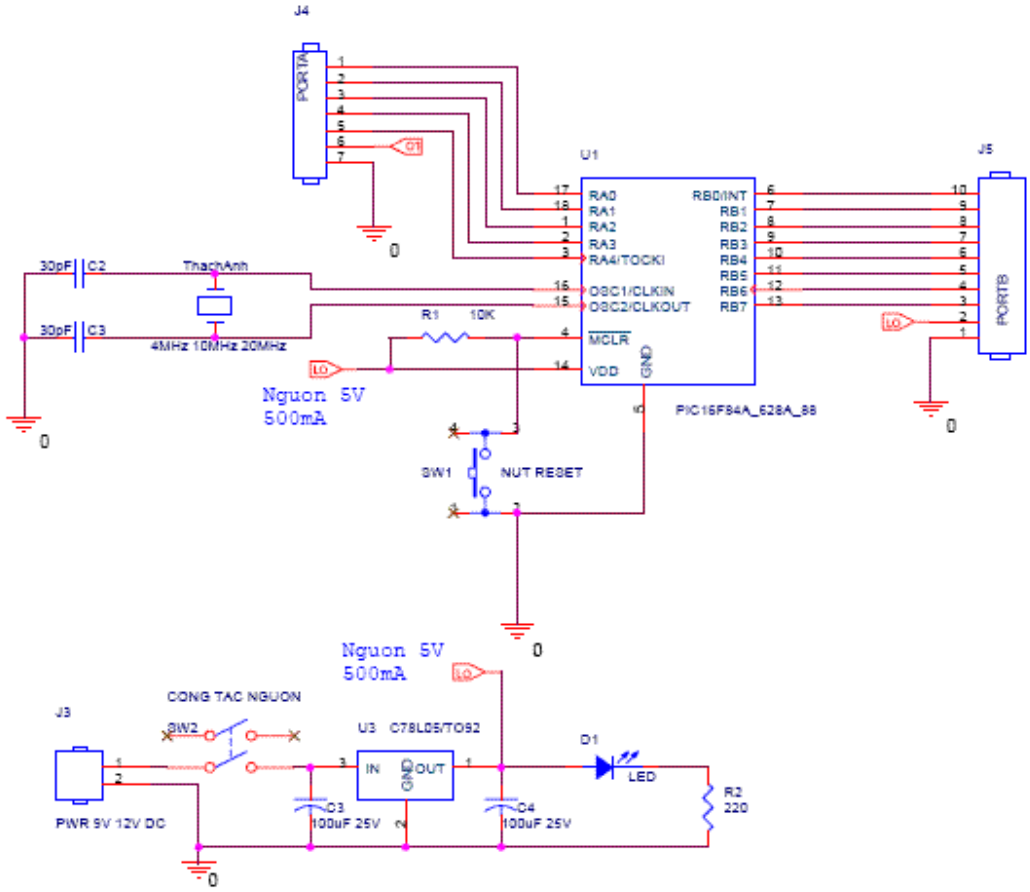
http://www.bobblick.com/techref/projects/propclock/propclock.html
Đồng hồ quay, dùng đèn led và hiện tượng lưu ảnh để hiển thị giờ, dùng tín hiệu xung trên các mấu rotor để xác định thời gian hiển thị.
http://www.seattlerobotics.org/encoder/may97/picchip.html

10) Email hỗ trợ thực hành PIC

Các bạn có thể email cho tôi khi gặp vấn đề cần tư vấn về PIC qua địa chỉ:
falleaf.pic@gmail.com
Khi gửi email, mong các bạn gửi kèm theo mạch nguyên lý, chương trình đã thực hiện, và các thông tin như: bạn sử dụng hệ điều hành gì? bạn dùng mạch nạp nào? bạn dùng chương trình dịch gì? bạn dùng chương trình nạp gì? Các lỗi báo cụ thể.... và tất nhiên các vấn đề các bạn muốn hỏi.
Tôi không hứa có thể trả lời tất cả email của các bạn, tuy nhiên tôi và các bạn của tôi sẽ cố gắng hết sức để giúp đỡ các bạn thực hiện đề tài trên vi điều khiển PIC.
Ngoài ra, chúng tôi rất thích trao đổi về các quan điểm thiết kế, ý tưởng sáng tạo nhất là về PIC, robotics, haptic device, biomedical equipements, radio frequency devices... Chúng tôi rất mong có được sự ủng hộ và chia sẻ của các bạn.

CƠ BẢN VỀ PIC

Dưới đây là hình mạch chạy của PIC16F84A, PIC16F628A và PIC16F88. Tất cả các PIC này đều có vị trí chân tương ứng nhau, và thậm chí có thể nói PIC16F628A tương thích PIC16F84A và PIC16F88 tương thích với hai loại còn lại. Có nghĩa là trong các ứng dụng của PIC16F84A, khi thay đổi bằng PIC16F88, hay PIC16F628A đều được.



Tất nhiên, 3 loại vi dòng PIC trên đây có thể tương thích với nhiều dòng PIC cũ hơn, nhưng vì thị trường PIC Việt Nam phổ biến với 3 loại PIC này, cho nên chúng tôi chỉ đề cập đến 3 loại PIC này mà thôi.

Sau khi các bạn có mạch nạp, chương trình nạp, MPLAB IDE, CCS C hoặc HT PIC, các bạn làm mạch chạy này. Kể từ đây khi thiết kế cách mạch test, hoặc các thiết bị ngoại vi khác, cần thử nghiệm, các bạn chỉ việc thiết kế mạch ngoài, sau đó cắm vào các chân ra và chạy thử.
Khi mạch chạy tốt, các bạn muốn thiết kế được hoàn chỉnh, các bạn chỉ việc copy mạch chạy từ Orcad và dán vào mạch nguyên lý của thiết bị của bạn. Xoá các chân header đi, và nối dây vào trong mạch chạy PIC. Như vậy, chúng ta không phải tốn thời gian thiết kế cho PIC nữa.

Một vài điểm lưu ý về mạch như sau:

- Nguồn chỉ dùng cho PIC, tuyệt đối không dùng bộ nguồn này cho thiết bị ngoại vi. Nếu thiết bị ngoại vi cần nguồn, các bạn thiết kế bộ nguồn riêng. Một số thiết bị ngoại vi quá đơn giản, và tốn ít dòng, các bạn có thể dùng nguồn chung (khoảng 100mA)
- Tôi không khuyến khích dùng dao động nội của PIC, bởi vì dao động nội chỉ chạy được ở 4MHz, và không ổn định như dùng thạch anh ngoài. Một số đề tài công nghiệp, họ dùng thạch anh chuẩn công nghiệp 4 chân, nên chúng ta cũng tạo thói quen dùng thạch anh ngoài, không cần quá tận dụng 2 chân của PIC.

- Mạch reset này là mạch reset đơn giản nhất của PIC, và tạo chế độ reset power on. Một số ứng

dụng của PIC yêu cần mạch brownout reset, các bạn có thể tham khảo trong datasheet. Nhưng tôi thiết nghĩ, những đề tài thông thường, không cần dùng mạch brownout reset này.

- Chúng ta thống nhất chuẩn thiết kế cho các header là nối vào các chân của PIC theo thứ tự hai chân ngoài cùng là Rx0 và GND. Mục đích là để khi chạy mạch in, chân GND có thể được xếp ra phía ngoài, chân Rx0 để quy định cho tất cả các port khác nhau, vì có port chỉ có 3 chân, có port 5 chân, 8 chân... Nếu lấy chân RB7 làm chuẩn chẳng hạn, thì sẽ rất khó giải thích khi lấy chân RA4 đặt ra phía ngoài. Vì vậy RA0 và RB0 chúng ta lấy làm chuẩn. Điều này cũng đã được thực hiện trong một số tutorial, và gần như là quy ước bất thành văn khi thực hiện các mạch phát triển cho vi điều khiển. Chân VDD (5V) được nối vào, nhằm sử dụng cho các ứng dụng cần có điện áp ngõ vào, nhưng không cao lắm như ở trên đã nói (100mA). Tuyệt đối không thiết kế chân VSS (GND) và chân VDD (5V) ở hai đầu của header, tránh tình trạng đôi khi chúng ta không để ý cắm nhầm, có thể làm hỏng PIC, hoặc hỏng luôn cả thiết bị ngoại vi.
- Các nút bấm và công tắc, tôi thiết kế là các nút bấm 4 chân, vì hiện nay trên thị trường hầu như chỉ bán loại nút bấm này, và loại nút bấm này chắc chắn hơn loại 2 chân trước đây. Các bạn cũng lưu ý sau này khi thiết kế nút bấm cũng nên thiết kế nút bấm 4 chân.

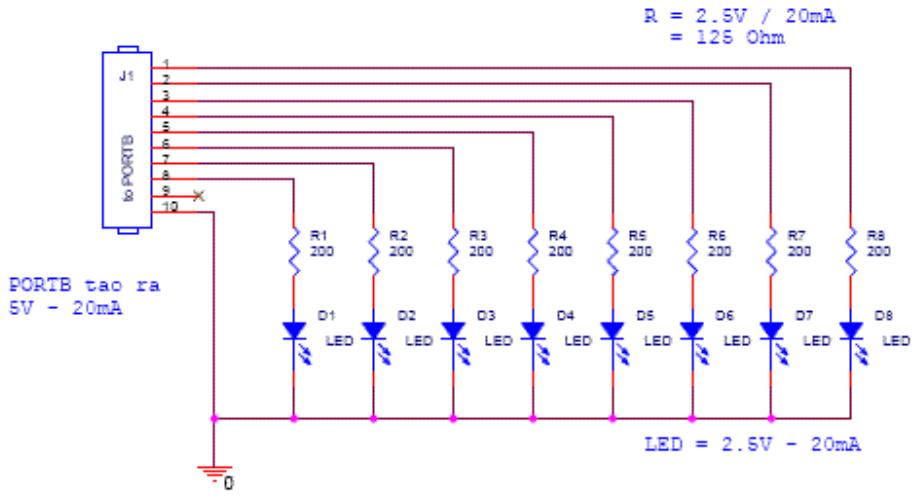
- Con ổn áp 78L05 khác với con 7805. Nó là dạng TO92, tức là nó giống như con transistor thông thường, nên rất nhỏ, chứ không phải dạng 3 chân và có tấm tản nhiệt phía sau như con 7805. Do vậy, mạch thiết kế sẽ nhỏ đi khá nhiều.

- Ở đây, tôi không chạy ra mạch in, vì rằng tôi muốn dành công việc này cho các bạn sinh viên mới học. Sau khi các bạn làm xong mạch in, nếu các bạn có thể chia sẻ với chúng tôi thì thật là tuyệt vời. Chỉ có một điều lưu ý là, chúng ta thường không cắm trực tiếp vi điều khiển vào mạch để hàn, mà chúng ta cắm qua một socket để có thể gỡ ra lập trình lại, và để đảm bảo không bị cháy PIC khi hàn. Do vậy, khi cắm socket, các bạn sẽ có thể nhét hai tụ nối ở thạch anh vào bên trong socket, khi cắm PIC lên, nó sẽ che hai cái tụ đó đi, và mạch của các bạn sẽ gọn gàng hơn. Socket loại 18 chân không thể nhét thạch anh và điện trở nối từ chân MCLR đến VDD vào bên trong được, nhưng sau này khi dùng PIC 28 hoặc 40 chân, các bạn nên nhét tất cả vào bên dưới socket để cho mạch gọn gàng hơn.

- Một điểm cuối cùng, chúng tôi không thiết kế phần nạp bằng ICSP, bởi vì chúng tôi không muốn làm cho các bạn mới học PIC cảm thấy bối rối. Chúng ta sẽ thực hiện mạch chạy PIC với các chân ICSP và bootloader sau.

Học vi điều khiển PIC trong 1 ngày

Bài tập 1: Bật tắt đèn LED



Cực dương của LED được nối với điện trở, điện trở được nối với các chân vi điều khiển. Cực âm của LED được nối với GND của vi điều khiển. Như vậy, khi chân vi điều khiển ở mức cao, tức là 5V, đèn LED sẽ sáng. Khi chân vi điều khiển ở mức thấp (0V) đèn LED sẽ tắt.

Lưu ý trong hình: Giá trị của điện trở được xác định dựa vào dòng tối đa của vi điều khiển, điện áp và dòng điện tối đa của đèn LED. Như vậy, giá trị nhỏ nhất của điện trở được dùng được tính toán như trong hình. R = 125 Ohm.

Tuy nhiên, để đảm bảo hoạt động của đèn LED, chúng ta nâng giá trị điện trở lên thành 200 Ohm. Đèn LED khi sáng quá, chỉ cần sờ tay vào nó, hoặc các va chạm mạnh, hoặc trường hợp bị tĩnh điện, đèn LED có thể bị hư ngay. Hiện tượng này dễ thấy nhất là ở các LED cực sáng dùng trong các bảng hiệu hoặc biển báo giao thông, các đèn LED cực sáng chỉ cần chạm tay vào, sẽ có hiện tượng tĩnh điện và nổ ngay. Với các LED thường và dùng trong thí nghiệm, khó xảy ra hiện tượng này, tuy nhiên chất lượng sản xuất của các đèn LED cũng không đảm bảo, do vậy chúng ta chọn giải pháp an toàn là trên hết. Hơn nữa, chúng ta cũng không cần đèn LED quá sáng.

Để bắt đầu bài tập 1, chúng ta tìm hiểu sơ qua về cấu trúc một chương trình viết bằng MPASM như sau:

Bất cứ một chương trình ASM nào, cũng được bắt đầu bằng việc giới thiệu về chương trình, tên chương trình, người thực hiện chương trình, ngày thực hiện chương trình, ngày hoàn tất, người kiểm tra lại chương trình, ngày kiểm tra chương trình, phiên bản của chương trình, mô tả phần cứng của mạch giao tiếp và một số chú thích. Vì vậy, tôi đưa ra đây một form mà tôi cho rằng hợp lý, từ đây về sau, các bạn chỉ cần cắt dán form này, thay đổi nội dung từng mục để làm phần mở đầu.
Chúng ta quy định một số quy ước sau:
;========== dùng để phân cách các phần chính của chương trình
;---------- dùng để phân cách các chương trình con của chương trình

Code:

```
;================================================  =======
; Ten chuong trinh     : Mach test den LED_1
; Nguoi thuc hien      : Falleaf
; Ngay thuc hien       : 23/05/2005
; Phien ban    : 1.0
; Mo ta phan cung      : Dung PIC16F628A - thach anh 10MHz
;              : LED giao tiep voi PORTB
;              : Cuc am cua LED noi voi GND
;              : RB0 - RB7 la cac chan output
;----------------------------------------------------------------
; Ngay hoan thanh      : 23/05/2005
; Ngay kiem tra: 23/05/2005
; Nguoi kiem tra       : Doan Hiep
;----------------------------------------------------------------
; Chu thich    : Mo ta cac diem khac nhau cua cac phien ban khac nhau
;              : hoac cac chu thich khac
;              : vd, dung che do Power On Reset, PORTB = 00000000
;              : hoac, chuong trinh viet cho PIC Tutorial
;              : hoac, chuong trinh nay hoan toan mien phi va co the dung cho
;              : moi muc dich khac nhau
;================================================  =======
```

Mặc dù chưa chắc rằng đoạn chú thích này có thể ngắn hơn chương trình các bạn viết, và như vậy việc viết chú thích dài hơn việc viết chương trình? Không, thực sự các chú thích này rất quan trọng, vì sau 1, 2, 3 năm, các bạn nhìn lại, các bạn sẽ vẫn còn hiểu được mình đã làm gì. Có thể khi mới bắt đầu, các bạn thấy công việc ghi chú này là nhàm chán, chính vì vậy, tôi đã cung cấp form của ghi chú này, các bạn sau đó chỉ cần cắt và dán. Tôi hy vọng rằng các bạn nên tạo thói quen đưa đoạn chú thích này vào chương trình để các bạn trở nên chuyên nghiệp hơn khi làm việc với vi điều khiển, cụ thể ở đây là PIC.

Tất nhiên, đây là bài học đầu tiên, do vậy các chú thích sẽ được ghi rất chi tiết, nhất là khi mô tả phần cứng. Sau này, với các mạch phức tạp hơn, các bạn không thể ghi chú quá chi tiết như thế này được, các bạn chỉ ghi chú những điểm chính thôi. Cũng tất nhiên, khi lập trình với CCS C hay HT PIC, các bạn cũng nên ghi chú như vậy trong chương trình chính, nhưng chúng ta chưa bàn đến CCS C và HT PIC ở đây.
Phần thứ hai các bạn cần học, đó là khởi tạo PIC. Phần này là phần bắt buộc theo sau phần ghi chú, bởi vì chương trình dịch cần phải hiểu bạn đang làm việc với con PIC nào, làm việc với nó như thế nào?

Code:

```
;================================================  ======
                TITLE           "Mach test LED_1"
                PROCESSOR       P16F628A
                INCLUDE         <P16F628A.inc>
                __CONFIG        _CP_OFF & _PWRTE_ON & _WDT_OFF &_HS_OSC

;================================================  ======
```

Các bạn sẽ thấy rằng có một số từ khoá như sau:
TITLE: dùng để các bạn ghi chú thích tên chương trình. TITLE là ten chương trình chính. Cú pháp ghi TITLE như trên. Nhớ phải có dấu nháy kép khi viết tên chương trình.
PROCESSOR: dùng để khai báo dòng vi điều khiển mà các bạn sử dụng. Các bạn lưu ý, trong MPLAB quy định, không viết đầy đủ tên PIC16F628A mà chỉ viết P16F628A, vì trong chương trình dịch đã quy định như vậy.
INCLUDE: dùng để đưa thêm vào các file mà bạn viết trong chương trình. Mặc định, trong MPLAB đường dẫn đến thư mục chứa file P16F628A.inc đã có sẵn. Nếu bạn đặt file ở nơi khác không phải

trong thư mục bạn đang làm việc, hoặc các file include khong phải là file .inc có sẵn của MPLAB, thì các bạn phải chỉ đường dẫn rõ ràng. Lưu ý rằng, để MPASM dịch được, các bạn phải đặt đường dẫn từ thư mục gốc đến hết tên file (kể cả phần mở rộng của file) không được quá 60 ký tự.
__CONFIG: dùng để thiết lập các chế độ hoạt động của PIC. Các bạn có thể xem để hiểu thêm về các chế độ hoạt động này trong tài liệu

PICmicro
Mid Range MCU Family
Reference Manual

Section 27. Device Configuration Bits
Table 27-1 page 27-7

Tài liệu này có thể download trên trang web của microchip http://www.microchip.com/, keyword: MidRange Manual.
Mỗi directive để đặt chế độ, cách nhau một ký tự &.
Nếu ghi chế độ hoạt động vào đây, các chế độ hoạt động sẽ ở trạng thái mặc định khi khởi động. Các bạn cũng có cách khác để đặt chế độ hoạt động bằng cách tác động trực tiếp vào các thanh ghi khởi tạo. Tuy nhiên, việc này là việc làm không cần thiết, khi chúng ta đã có các directive để viết tắt.
Như vậy, chúng ta đặt ở đây chế độ _CP_OFF, tức là khôngđặt chế độ bảo vệ source code khi nạp vào PIC, sau khi nạp vào sẽ có thể đọc ngược lại từ PIC ra. Chúng ta không cần bảo vệ chương trình này, để bạn có thể đọc ngược bằng IC-PROG và kiểm tra lại.
Chế độ _PWRITE_ON, tức là cho timer 0 chạy khi Power On Reset. Thực ra timer0 có chạy hay không cũng không quan trọng, vì nó chẳng liên quan gì đến công việc của chúng ta. Nếu sau này muốn dùng timer0, thì các bạn vẫn phải khởi tạo lại giá trị cho nó, chứ đâu thể sử dụng giá trị ngẫu nhiên của nó được, thành ra cứ để cho nó chạy, sau này cần dùng khỏi phải khởi tạo.
_WDT_OFF, tại thời điểm này, tôi tắt Watch Dog Timer vì lý do các bạn chưa nên tìm hiểu phần này vội.
_HS_OSC, chúng ta dùng thạch anh 10MHz, tức là chạy chế độ dao động HS. Tham khảo tại:

datasheet PIC16F628A
Section 14. Special Features of the CPU
14.2. Oscillator Configuration
Page 95

Một điểm lưu ý cuối cùng là các bạn phải sử dụng phím TAB để phân cách các cột của một chương trình viết bằng MPASM. Các dòng khởi tạo này được viết ở cột thứ 3. Các directive __CONFIG, TITLE, PROCESSOR, INCLUDE được viết vào cột thứ 3. Còn chi tiết khởi tạo được viết vào cột thứ tư.
Cột thứ nhất dùng để viết các [NHÃN], cột thứ hai để viết mã lệnh, cột thứ ba lại dùng để viết chi tiết các tham số của lệnh, và cột thứ tư bỏ trống để tạo khoảng cách với cột thứ năm. Cột thứ năm dùng để viết các chú thích.
Các chú thích bắt đầu bằng dấu chấm phẩy (. Trên một dòng, tất cả các ký tự viết sau dấu chấm phẩy đều vô nghĩa. Chính vì vậy, khi viết phần chú thích ban đầu, các bạn thấy rằng tất cả nội dung đó đều bắt đầu bằng dấu chấm phẩy. Như vậy, một dòng lệnh được cụ thể như sau:
Code:

```
NHÃN            LỆNH    thamso1,        thamso2         ; chú thích dòng lệnh
```

Bây giờ chúng ta dành chút thời gian cho lý thuyết, các bạn mở datasheet PIC16F628A trang 15, Section 4. Memory Organization
Chúng ta sẽ thấy rằng tổ chức bộ nhớ chương trình của PIC được chia ra làm mấy phần như sau:
- Pointer
- Stack
- Interrupt vector
- Program memory

Chúng ta tạm thời chưa bàn đến pointer và stack.
Interrupt vector được đặt ở địa chỉ 0x0004
Program memory được đặt ở địa chỉ 0x0005
Vậy từ địa chỉ 0x0000 đến địa chỉ 0x0003 chúng ta làm được gì?

Khi PIC được reset, nó lập tức nhảy về địa chỉ 0x0000. Rồi cứ sau một chu kỳ máy, nó nhảy đến địa chỉ tiếp theo, xem xem trong địa chỉ đó yêu cầu nó làm gì, nó thực hiện việc đó, xong rồi lại nhảy tiếp. Cứ làm như thế cho đến khi hết chương trình. Tất nhiên, khi chúng ta thực hiện một số lệnh điều khiển vị trí nhảy, thì nó sẽ nhảy không theo thứ tự nữa, nhưng việc này chưa bàn vội. Chúng ta trước mắt chỉ cần biết rằng nó cứ nhảy như vậy cho đến hết chương trình.

Như vậy, nếu không sử dụng ngắt, thì chúng ta viết chương trình từ địa chỉ 0x0000 luôn, vì nó cứ thế là nhảy từ 0x0000 khi khởi động, cho đến hết chương trình. Tuy nhiên, nếu làm như vậy, sau này chúng ta sử dụng chương trình ngắt, thì chúng ta sẽ gặp trục trặc vì thói quen viết từ địa chỉ 0x0000.
Chính vì vậy, chúng ta nên đặt chương trình trong phần Program Memory như ý đồ thiết kế PIC.
Vậy, chương trình của chúng ta sẽ viết như sau:
Code:

```
;************************************************  ******************
                ORG     0x0000
                GOTO    MAIN


                ORG     0x0005
MAIN
.....

                END.
;************************************************  ******************
```

Đây sẽ là cấu trúc một chương trình mà chúng ta sẽ thực hiện. Directive ORG dùng để xác định địa chỉ mà chúng ta sẽ làm việc. Bây giờ chúng ta xem tiếp đến trang 16 của datasheet. Chúng ta thấy rằng, bộ nhớ dữ liệu của PIC16F628A được chia ra thành 4 BANK, hay chúng ta gọi tiếng Việt là 4 BĂNG. Trong 4 băng này, chúng ta thấy rõ nó được chia làm 3 phần. Phần thứ nhất là phần các thanh ghi có địa chỉ xác định (được ghi chú ở bên cạnh) và có tên tuổi rõ ràng. Những thanh ghi này được gọi là những thanh ghi đặc biệt của PIC. Tên của chúng, thực ra không có, một thanh ghi chỉ được xác định bằng địa chỉ của thanh ghi mà thôi.

Tuy nhiên, chúng ta đã làm động tác include file P16F628A.inc, file này đã định nghĩa sẵn tên các thanh ghi này, và là quy ước của MPLAB, đồng thời cũng là quy ước chung cho tất cả người dùng PIC. Chúng ta có thể thay đổi, sửa chữa những định nghĩa này, tuy nhiên việc làm đó vừa không cần thiết, lại vừa gây ra rất nhiều khó khăn khi làm việc nhóm.

Vậy các bạn phải hiểu, những tên thanh ghi này xem như là không thay đổi trong PIC, và chúng ta sử dụng nó như nó đã tồn tại vài chục năm nay.

Phần thứ hai, đó là phần General Purpose Register. Chúng ta gọi nó là các Thanh Ghi Dùng Chung. Những thanh ghi này chưa được định nghĩa, và vì thế nó cũng không có tên. Những thanh ghi này có giá trị như các biến trong chương trình mà chúng ta sẽ sử dụng.

Phần thứ ba, đó là các thanh ghi nằm ở địa chỉ 70h đến 7Fh, và vị trí tương ứng của nó ở băng 1, 2, 3. Các thanh ghi tương ứng đó ở bank1, 2, 3 sẽ tương thích với các thanh ghi từ 70h đến 7Fh ở băng 0. Tuy nhiên, chúng ta tạm thời chưa quan tâm đến phần này.
Bây giờ chúng ta học viết chương trình
Code:

```
;************************************************  ******************************
                ORG     0x0000
                GOTO    MAIN


                ORG     0x0005

MAIN
```

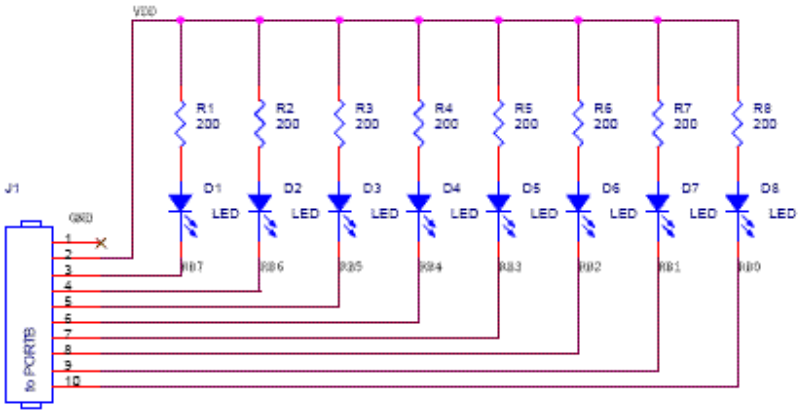
```
                BANKSEL TRISB           ; bank select
                CLRF            TRISB           ; trisb = 00000000
                                                ; portb = output
                BANKSEL PORTB
                BSF             PORTB,  0       ; rb0 = 1
                                                ; RB0 = 5V
                GOTO            $               ; dung chuong trinh tai day
                                                ; vong lap tai cho^~
                                                ; khong bao gio ket thuc
                END                             ; lenh bat buoc de ket thuc
;================================================  ==============================
```

Rồi, như vậy, chúng ta đã thực hiện xong một chương trình viết bằng MPASM cho PIC16F628A.

Phân tích chương trình, chúng ta sẽ thấy, mới khởi động, chương trình gặp lệnh goto main, nó sẽ nhảy đến nhãn MAIN. Ở nhãn MAIN, nó gặp lệnh banksel, tức là lệnh bank select. Có nghĩa là nó sẽ chuyển sang hoạt động ở băng có chứa thanh ghi TRISB. Vì sao? Bởi vì ban đầu khởi động, PIC luôn nằm ở băng 0. Nhưng thanh ghi TRISB lại nằm ở băng 1, vì thế cần phải chuyển sang băng 1 để làm việc. Thực ra chúng ta cũng có cách để yêu cầu PIC chuyển sang băng 1 một cách đích danh, chứ không phải là chuyển sang băng có thanh ghi trisb như chúng ta vừa làm. Nhưng việc này là không cần thiết, cả hai việc làm đều giống nhau. Chính vì vậy, chúng ta chọn cách viết nào cho dễ nhớ là được. Sau khi chuyển sang băng 1. Chúng ta dùng lệnh CLRF để xoá thanh ghi TRISB.
Tức là TRISB = 00000000

Chúng ta lưu ý một điều rằng, thanh ghi TRISB có công dụng quy định PORTB sẽ có những chân nào là chân xuất, chân nào là chân nhập. Chúng ta nhớ thêm một điều nữa, số 0 giống chữ O, và số 1 giống chữ I. Như vậy, khi TRISB = 00000000 tức là PORTB sẽ là OOOOOOOO, tức có nghĩa là tất cả các chân của portB đều là Output. Nếu TRISB = 01010101 thì PORTB sẽ là OIOIOIOI. Có nghĩa là RB0 sẽ là Input, RB1 là Output, RB2 là Input, RB3 là Output.. cứ như thế cho đến RB7 là Output. Lưu ý rằng RB0 đến RB7 được tính từ phải sang trái.
Sau đó, chúng ta lại thực hiện lệnh Banksel portb, tức là chúng ta lại nhảy về băng 0 (băng chứa thanh ghi portb). Tất cả các lệnh làm thay đổi giá trị của thanh ghi portb, sẽ làm thay đổi tín hiệu điện ở bên ngoài chân của PORT B. Sau khi chuyển sang băng 0, chúng ta thực hiện lệnh BSF PORTB,0. Có nghĩa là chúng ta set bit ở vị trí 0 của portb, tức là chúng ta cho RB0 = 1.
Có nghĩa là ở ngoài chân RB0 sẽ mang giá trị điện áp 5V. Khi đó, đèn LED nối với RB0 sẽ sáng.
Các bạn sẽ thấy mach ngoài hoạt động như thế này:
Khi bật điện lên, PIC được reset. Nó lập tức bật sáng đèn LED ở RB0, rồi sau đó giữ nguyên như vậy, không làm gì cả.
Bây giờ các bạn lưu chương trình vừa viết thành LED_1.asm vào một thư mục nào đó.
Nhấn Alt - F10, chương trình sẽ dịch LED_1.asm thành LED_1.hex
Các bạn dùng mạch nạp PG2C và chương trình nạp IC-PROG để nạp vào PIC (tham khảo Hướng dẫn mạch nạp Falleaf PG2C - PIC Tutorial).
Công việc của các bạn như sau:
0) Chạy thử chương trình ban đầu
1) Thay đổi lệnh BSF PORTB, 0 bằng lệnh BSF PORTB, 1. Nạp lại chương trình mới vào PIC. Bạn sẽ thấy bây giờ đèn LED không sáng ở vị trí RB0 nữa mà sáng ở vị trí RB1.
2) Thay lệnh BSF PORTB,0 bằng đoạn lệnh
MOVLW b'11110000'
MOVWF PORTB
Bạn sẽ thấy các các chân từ RB0 đến RB3 sẽ tắt đèn, và các chân từ RB4 đến RB7 đèn sẽ sáng.
3) Bạn thay lệnh CLRF TRISB bằng đoạn lệnh
CLRF TRISB
BSF TRISB, 0
và giữ nguyên lệnh
BSF PORTB, 0
Các bạn sẽ thấy rằng đèn LED trong trường hợp này sẽ không sáng nữa.
Bởi vì các bạn đã làm cho TRISB = 00000001. Như vậy, RB0 trở thành chân Input. Khi RB0 trở thành chân Input, thì lệnh BSF PORTB, 0 sẽ không còn tác dụng nữa. RB0 lúc này không thể thay đổi giá trị bằng chương trình, nó chỉ có thể nhận giá trị điện áp từ bên ngoài vào.

4) Trong trường hợp mạch này, các bạn sẽ làm thế nào?



Kết luận: Qua bài học này, các bạn đã học được các nội dung sau:
- Làm một mạch chạy PIC
- Cấu trúc một chương trình PIC
- Lập trình từ máy tính, nạp vào PIC, và cho PIC hoạt động
- Hiểu được hoạt động xuất nhập của PIC, chức năng của thanh ghi TRISA, TRISB, PORTA, PORTB, hiểu được các lệnh CLRF (xoá thanh ghi bất kỳ), MOVLW (ghi một giá trị bất kỳ vào thanh ghi W), MOVWF (ghi giá trị của thanh ghi W vào một thanh ghi khác), BSF (bật một bit trong một thanh ghi bất kỳ), GOTO (nhảy đến một nhãn bất kỳ), GOTO $ (nhảy tại chỗ), BANKSEL (chon băng trong bộ nhớ chương trình, chứa một thanh ghi bất kỳ), ORG định địa chỉ trong bộ nhớ chương trình. Hiện nay các bạn chưa học đến làm thế nào để Input, nhưng có thể các bạn sẽ thực hiện dễ dàng bằng việc thay LED bằng một nút bấm. Hoặc giả, các bạn muốn đèn LED nhấp nháy, về nguyên tắc các bạn có thể thực hiện bật tắt liên tục đèn LED bằng lệnh BSF và BCF. Nhưng làm như thế nó nháy quá nhanh, không thể thấy được.

Bài học sau, chúng ta sẽ học cách viết hàm Delay, và các bạn có thể thực hiện việc làm cho đèn LED nhấp nháy, làm cho dãy đèn từ RB0 đến RB7 chạy qua chạy lại...

Chúc các bạn may mắn trong bài học đầu tiên, và chúc các bạn thành công với PIC!

******* &&& *******

Thanh ghi W

Trong bài này, chúng ta nói đôi nét về thanh ghi W để các bạn nắm rõ hơn phương thức hoạt động của PIC.

Khái niệm thanh ghi W:

Thanh ghi W là thanh ghi làm việc (Working register), và hầu hết mọi lệnh của PIC đều liên quan đến thanh ghi W này, lấy thí dụ như ADDLW (cộng một số vào giá trị đã có trong thanh ghi W), SUBWF (trừ giá trị của thanh ghi W cho một thanh ghi khác), XORLW (lấy XOR của một số và thanh ghi W)... Và các bạn để ý rằng, tổng số lệnh có thể tương tác với thanh ghi W là 23/35 lệnh, gần như chiếm toàn bộ tập lệnh của PIC. Vậy chúng ta ghi nhận điều thứ nhất, khi PIC làm việc, gần như luôn luôn tương tác với thanh ghi W.

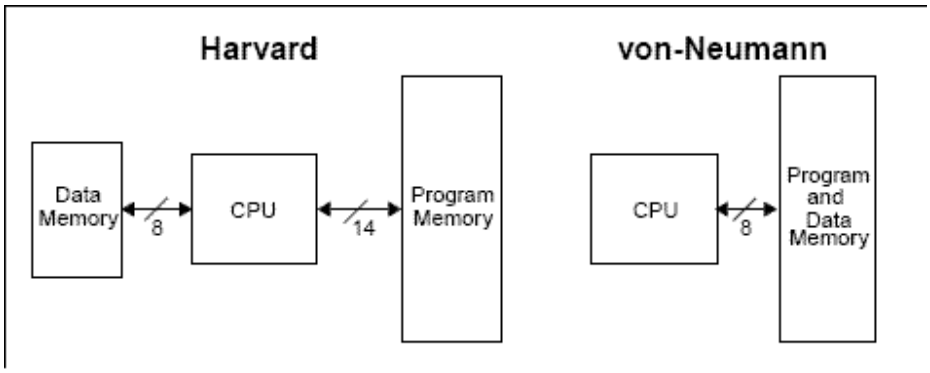
Điều thứ hai, các bạn nhìn trong bản đồ bộ nhớ dữ liệu của PIC, các bạn sẽ thấy là thanh ghi W là thanh ghi không có mặt ở bất kỳ băng nào của bộ nhớ dữ liệu, trong khi đó thanh ghi STATUS có mặt ở cả 4 băng. Các bạn lại thấy một điều rằng, thanh ghi W và thanh ghi STATUS có thể được truy nhật từ tất cả các băng, và từ bất kỳ đâu trong chương trình, và vì vậy chúng trở thành những thanh ghi toàn cục nhất. Điểm khác biệt giữa chúng ra sao? Đâu là sự khác biệt giữa thanh ghi W và các thanh ghi khác?

Điểm thứ ba, trong tập lệnh của PIC, không có lệnh nào cho phép tương tác trực tiếp giữa một thanh ghi trong bộ nhớ dữ liệu dùng chung với một giá trị thêm vào, mà đều phải thông qua thanh

ghi W. Như vậy, thanh ghi W là cầu nối của hầu hết các phép toán được thực hiện trên các thanh ghi nằm trong bộ nhớ dữ liệu. Như vậy, thanh ghi W vô cùng quan trọng trong hoạt động của PIC.

Nhắc lại kiến trúc Harvard và Von Newmann:

Hình sau sẽ gợi lại cho các bạn nhớ về kiến trúc Harvard và Von Newmann, trong đó các bạn luôn nhớ rằng có sự phân biệt giữa bộ nhớ dữ liệu và bộ nhớ chương trình. Các bạn thấy rằng bus bộ nhớ chương trình của PIC midrange chỉ có 14 bit.



Với đặc điểm này, chúng ta sẽ phân tích vì sao cần phải có thanh ghi W, và sau đó chúng ta sẽ phân tích tất cả các hoạt động của thanh ghi W trong một chương trình viết bằng PIC, nếu có thể. Những gì còn lại, chúng ta sẽ xem trong bài tập lệnh của PIC midrange.

Vì sao cần phải có thanh ghi W?

Bạn sẽ làm thế nào để tính phép toán sau: lấy giá trị a của thanh ghi A cộng với giá trị b của thanh ghi B và đặt vào thanh ghi A? Một giới hạn của tập lệnh PIC là không cho phép cộng hai thanh ghi và đặt vào một thanh ghi khác. Do đó, các bạn sẽ phải thực hiện thao tác sau:

Chuyển giá trị b từ thanh ghi B vào thanh ghi W, sau đó lấy giá trị của thanh ghi W (lúc này là b) cộng với giá trị a ở thanh ghi A, sau đó gán lại vào thanh ghi A. Đoạn code được thực hiện như sau:

Code:

```
MOVF    B,      W               ; chuyển giá trị của thanh ghi B vào thanh ghi W
ADDWF   A,      F               ; cộng giá trị của thanh ghi A với giá trị b của
thanh ghi W và gán lại vào A
```

Khi các thanh ghi A và B không nằm trong cùng một băng, khi thao tác với từng thanh ghi, các bạn chỉ việc đổi về băng chứa các thanh ghi đó là xong. Một đoạn lệnh hoàn chỉnh có thể thực hiện cho bất kỳ 2 thanh ghi nào được viết như sau:

Code:

```
                BANKSEL B
                MOVF            B,      W
                BANKSEL A
                ADDWF           A,      F
```

Đoạn chương trình này cũng minh hoạ luôn cho việc thanh ghi W là một thanh ghi toàn cục, khi chúng ta thao tác với thanh ghi B ở một băng bất kỳ, nhưng khi chuyển giá trị b từ thanh ghi B vào thanh ghi W rồi, thì chúng ta không cần quan tâm rằng giá trị đó nằm ở đâu, chỉ cần chuyển về băng chứa thanh ghi A thì lệnh cộng sẽ được thực hiện một cách dễ dàng.

Một thí dụ khác về lệnh cộng, nhưng không phải là cộng giá trị nằm trong 2 thanh ghi, mà là cộng giá trị a của thanh ghi A với một số k cho trước nào đó, giả sử k = 5 và lưu vào thanh ghi A.

Chúng ta thấy rằng, hoàn toàn trong tập lệnh không có lệnh cộng trực tiếp một thanh ghi với một số, mà chỉ có lệnh cộng một số với thanh ghi W. Như vậy chúng ta phải thực hiện thao tác sau: chuyển giá trị a từ thanh ghi A vào thanh ghi W, cộng thanh ghi W với hằng số k = 5, sau đó chuyển giá trị mới của thanh ghi W trở lại thanh ghi A. Điều này được thực hiện như sau:

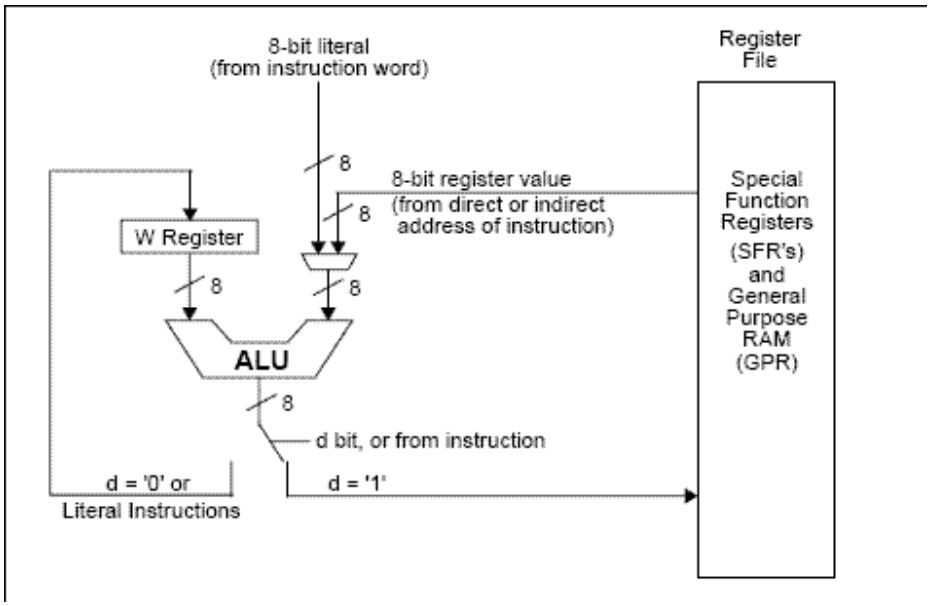
Code:

```
MOVF            A,      W
ADDLW           d'5'
MOVWF           A
```

Trong thí dụ này, chúng ta sẽ không thấy W là một biến tạm nữa, mà trở thành một thanh ghi dùng để lưu kết quả cộng với một con số. Đến bây giờ, thì chúng ta sẽ giả thích rõ hơn vì sao chúng ta phải làm như vậy.

Chúng ta thấy rõ ràng rằng, một dòng lệnh của PIC midrange, được mô tả bằng 14 bit. Điều này có nghĩa là, khi thực hiện một lệnh cộng, không thể nào dòng lệnh đó vừa lưu địa chỉ của thanh ghi A, vừa lưu giá trị 8 bit của hằng số k được, vì một thanh ghi trong dòng PIC midrange cần tối thiểu 7 bit để biểu diễn địa chỉ thanh ghi, và một hằng số chiếm 8 bit. Nó vượt quá con số 14 bit cho phép để mã hoá lệnh. Chính vì vậy, không thể thực hiện lệnh cộng trực tiếp từ một thanh ghi với một số được. Quay lại thí dụ ở trên, chúng ta cũng thấy rằng không thể thực hiện việc cộng hai thanh ghi với nhau, nếu như cần lưu 2 địa chỉ thanh ghi, chúng ta sẽ mất 14 bit, và như vậy không có các bit mã hoá mô tả lệnh cần thực hiện là gì.

Đây chính là điểm khác biệt giữa tập lệnh RISC và tập lệnh CISC. Tập lệnh CISC có thể thực hiện lệnh phức, vì nó có thể tạo ra một lệnh dài 8 bit, 16 bit, 24 bit... và là bộ số của 8 bit. Do đó, nếu cần cộng 2 thanh ghi 8 bit, nó hoàn toàn có thể tạo ra một lệnh dài 24 bit, trong đó 8 bit dùng để mã hoá, 8 bit dành cho địa chỉ của thanh ghi thứ nhất, 8 bit dành cho địa chỉ cua thanh ghi thứ 2. Trong khi đó, tập lệnh RISC là tập lệnh rút gọn, cho dù nó là lệnh gì, nó cũng luôn luôn chỉ có 14 bit (đối với PIC midrange).
Thanh ghi W giống như một thanh ghi mặc định duy nhất, vì vậy, khi thực hiện, bộ xử lý trung tâm có thể giải mã được nếu lệnh đó có cần thao tác với thanh ghi W hay không, mà không cần lưu địa chỉ của thanh ghi W bên trong đoạn mã lệnh.
Chúng ta xem hình dưới đây để biết được bộ xử lý logic hoạt động như thế nào với thanh ghi W.



Vậy chúng ta đã thấy rõ sự cần thiết của thanh ghi W, bởi vì chúng ta cần có một thanh ghi tạm cho các công việc tính toán, và chúng ta cần mã hoá thanh ghi mà không cần tốn quá nhiều bit, vậy thì thanh ghi W vừa là thanh ghi có tính toàn cục, vừa là thanh ghi tạm, vừa là thanh ghi không cần thiết nhiều bit để biểu diễn địa chỉ.

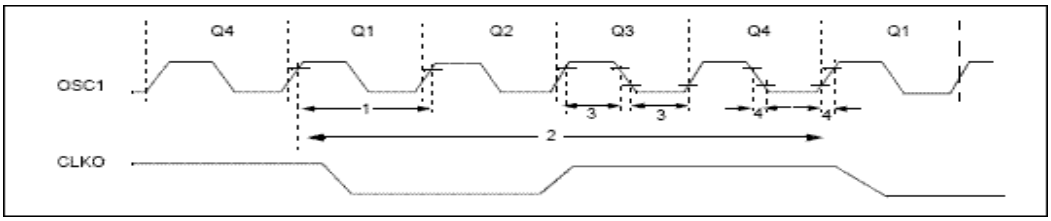
Các bạn đã biết vì sao chúng ta phải cần thanh ghi W, bây giờ chúng ta cần biết thanh ghi W hoạt động như thế nào trong các chương trình của PIC.

BÀI 2 - DELAY FUNCTION

Qua bài học thứ nhất, chúng ta đã học về cách bật tắt một đèn LED. Bây giờ nếu muốn làm cho đèn LED nhấp nháy, có nghĩa là chúng ta bật đèn LED, sau đó chờ một khoảng thời gian, và tắt đèn led đó đi, sau đó lại chờ một khoảng thời gian nữa và lại bật đèn led lên. Muốn thực hiện việc này, chúng ta phải tìm cách làm một hàm delay (delay - tiếng Anh có nghĩa là trễ, chậm lại) Hàm DELAY là một hàm rất thông dụng khi lập trình thời gian thực. Nguyên lý của hàm delay là dùng thời gian thực hiện các lệnh của vi điều khiển để làm thời gian trễ. Như các bạn đã biết (nếu chưa biết thì bây giờ biết.. hihi), mỗi lệnh của vi điều khiển, khi thực hiện, cần phải tốn một khoảng thời gian nào đó. Nếu một việc làm mà không tốn thời gian thì đúng là vô lý. Vậy thời gian thực hiện một lệnh của PIC là bao lâu?

Như trong bài học đầu tiên chúng ta đã đề cập, chúng ta sử dụng thạch anh từ 4MHz đến 10MHz và đến 20MHz. Thạch anh này tạo ra các dao động xung nhịp chính xác để duy trì những khoảng thời gian xác định cho vi điều khiển hoạt động.
Chúng ta xem hình sau để hiểu được nguyên lý tạo dao động bên trong vi điều khiển:

Hình 1:



Thạch anh tạo dao động trên các chân OSC, đưa vào bên trong PIC. PIC sẽ đếm 4 nhịp trên dao động thạch anh, và để thực hiện một lệnh. Như vậy, thời gian thực hiện một lệnh chính là 4 nhịp dao động của thạch anh.
Chúng ta thường gọi thời gian thực hiện một lệnh của PIC là một chu kỳ máy (đoạn số 2 trên hình). Vậy một chu kỳ máy bằng bao nhiêu, nếu chúng ta sử dụng thạch anh 10MHz cho PIC?

Code:

```
Tần số dao động của thạch anh:
F_osc = 10MHz
Chu kỳ của dao động thạch anh:
T_osc = 1/10.000.000 s
Chu kỳ máy
T_instruction = 4 * T_osc = 4/10.000.000 s = 0.0000004 s = 0.0004 ms = 0.4 us = 400
ns
```

Như vậy, một lệnh máy được thực hiện trong vòng 0.4 micro giây, hay 400 nano giây.
Tương tự, khi các bạn dùng thạch anh 4MHz, chu kỳ máy sẽ là 1us, và dùng thạch anh 20MHz, chu kỳ máy sẽ là 200 nano giây.

Quay trở lại với việc nếu chúng ta cần thực hiện một việc gì đó giống như nhấp nháy đèn LED, thì chúng ta cần PIC phải dừng lại, không làm gì cả để chờ chúng ta. Nếu như lệnh NOP (lệnh không làm gì) sẽ giúp chúng ta chờ 0.4 us, mà chúng ta cần chờ 1 giây, thì chúng ta viết bao nhiêu lệnh NOP cho đủ? Thay vì như vậy, chúng ta viết một vòng lặp để cho vi điều khiển làm một việc vô thưởng vô phạt nào đó N lần, và mỗi lần như vậy nó tốn T chu kỳ máy. Như vậy, sau khi kết thúc việc làm vô thưởng vô phạt đó, vi điều khiển đã chờ chúng ta N * T chu kỳ máy.
Để viết một vòng lặp như vậy, trước tiên chúng ta học cách đặt biến.
Một biến được đặt trong PIC, thực chất là một tên gọi chung cho một hoặc nhiều thanh ghi các giá trị. Trong phần này, chúng ta chỉ đơn giản làm đặt biến có nghĩa là đặt tên cho một thanh ghi. Thực ra, chúng ta hoàn toàn không cần đặt tên, mà có thể gọi trực tiếp địa chỉ của thanh ghi, nhưng nếu làm như vậy, sau này, khi chương trình phức tạp dần lên, chúng ta sẽ dễ bị lẫn lộn các biến.

Khi đặt biến, thanh ghi này nằm ở đâu? Nó sẽ nằm trong bộ nhớ chương trình và cụ thể,

nó sẽ nằm trong vùng nhớ dùng chung mà chúng ta đã đề cập trong bài học trước.
Vậy làm thế nào để đặt biến? Có rất nhiều cách đặt biến, và trong phần này, tôi sẽ hướng dẫn các bạn cách đặt biến mà tôi cho rằng rõ ràng nhất.

Code:

```
;************************************************  ****************

                        ORG 0x020
COUNT_L RES     1
COUNT_H RES     1
COUNT_N RES     3

;************************************************  ****************
```

Các bạn vừa làm gì?

Directive ORG dùng để xác định địa chỉ vùng nhớ. Các bạn lưu ý rằng, khi xác định địa chỉ vùng nhớ ở đây, chính là các bạn xác định địa chỉ vùng nhớ dữ liệu, chứ không phải địa chỉ vùng nhớ lập trình. Những gì các bạn viết phía bên dưới, sẽ giúp cho trình dịch hiểu được rằng các bạn đang làm việc trong vùng nhớ lập trình, hay vùng nhớ dữ liệu
Directive RES quy định việc đặt biến. Số 1 phía sau xác định rằng biến có tên COUNT_L chiếm 1 thanh ghi 8 bit, tức là 1 byte. Tiếp theo, các bạn lại đặt biến tên là COUNT_H. Như vậy, biến COUNT_H cũng chiếm 1 byte.

Câu hỏi đặt ra là các thanh ghi này nằm ở đâu?

Các bạn lưu ý, khi các bạn dùng directive ORG, là các bạn đã xác định nơi bắt đầu đặt biến. Như vậy, biến COUNT_L sẽ có độ dài 1 byte, và được đặt ở địa chỉ 0x020 tức là địa chỉ đầu tiên của vùng nhớ dữ liệu dùng chung trong băng 0 (20h)
Vì COUNT_L đã chiếm 1 byte. Do đó, biến COUNT_H sẽ chiếm byte tiếp theo, và địa chỉ đầu tiên của COUNT_H sẽ là 21h, nhưng COUNT_H cũng chỉ có 1 byte, cho nên nó chính là thanh ghi ở địa chỉ 21h. Đến biến COUNT_N, tương tự, địa chỉ đầu tiên của nó sẽ là 22h. Biến COUNT_N chiếm 3 thanh ghi, như vậy, biến COUNT_N sẽ nằm từ 22h, 23h đến 24h. Nếu tiếp tục đặt thêm các biến khác, các biến đó sẽ bắt đầu từ địa chỉ 25h, cứ như thế.

Nếu hiểu nôm na theo cách này, bạn có thể sẽ dễ hiểu nó hơn, một hằng là một giá trị. Giá trị đó có thể nằm trong thanh ghi dữ liệu (bộ nhớ dữ liệu), nhưng cũng có thể nằm trong lệnh điều khiển (bộ nhớ chương trình). Điều này khẳng định rằng, hằng là một giá trị.
Một khi bạn đặt một tên nào đó, để đại diện cho một hằng số, có nghĩa là thay vì bạn viết cái giá trị đó, thì bạn viết cái tên đại diện đó, để dễ nhớ. Chẳng hạn, bạn viết chữ pi, đại diện cho hằng số có giá trị 3.1415926....

Trong khi đó, nếu bạn đặt một biến pi, thì có nghĩa là bạn xác định địa chỉ của thanh ghi dữ liệu nào đó, mà mỗi khi bạn truy xuất đến biến pi, có nghĩa là bạn đang thao tác với thanh ghi ở địa chỉ mà biến pi đại diện. Ví dụ: bạn đặt biến pi ở thanh ghi 0x20 chẳng hạn. Điều đó có nghĩa là khi ban làm gì với biến pi, chính là bạn đang làm việc với thanh ghi ở địa chỉ 0x20.
Nhưng bạn sẽ thấy rằng, vậy biến pi và hằng số pi có gì khác nhau? Bây giờ biến pi và hằng pi cũng đều mang giá trị cả. Nhưng các bạn nên nhớ, trong câu lệnh lúc nào vị trí của biến (thanh ghi) F, và vị trí của hằng số k (trong cấu trúc một câu lệnh MPASM, tôi sẽ post lại bài này từ dddt). có sự phân biệt rõ ràng.

Vậy tùy theo vị trí bạn đặt nó ở đâu, nó sẽ là biến, hoặc là hằng. Nếu là biến, nó chỉ mang giá trị của dịa chỉ của thanh ghi nằm trong bộ nhớ dữ liệu, nếu là hằng, nó nằm đâu cũng được kể cả ở bộ nhớ dữ liệu và bộ nhớ chương trình.
Vậy muốn đặt biến ở các băng khác thì làm thế nào? Các bạn cứ lấy địa chỉ đầu của vùng nhớ dữ liệu dùng chung của băng đó và viết như sau:

Code:

```
;************************************************
                        ORG     0x0A0h

COUNT_X RES     10
```

```
;================================================
```

Tóm lại, để chuẩn hoá một chương trình, các bạn chép đoạn code này vào, và sau đó không bao giờ còn phải viết lại nữa:

Code:

```
;================================================  ======================

;----------------------------------
; Bien nam o Bank0
;----------------------------------
                      ORG     0x020

COUNT_L RES     1
COUNT_H RES     1

;----------------------------------
; Bien nam o Bank1
;----------------------------------

                      ORG     0x0A0

COUNT1_L        RES     1

;----------------------------------
; Bien nam o Bank2
;----------------------------------
                      ORG     0x120

;================================================  ======================
```

Như vậy, một chương trình tổng quát bây giờ sẽ trở thành như thế nào?

Code:

```
;================================================  ======================
; Phần chú thích ban đầu
;
;================================================  ======================
; Phần khởi tạo vi điều khiển
                      TITLE
                      PROCESSOR
                      INCLUDE
                        CONFIG

;================================================  ======================
; Phần đặt biến

;-------------------------------------
; Biến ở băng 0
;-------------------------------------
                      ORG     0x020
;-------------------------------------
; Biến ở băng 1
;-------------------------------------
                      ORG     0x0A0
;-------------------------------------
; Biến ở băng 2
;-------------------------------------
                      ORG     0x120
```

```
;================================================  ========================
; Phần chương trình chính
                        ORG     0x0000
                        GOTO    MAIN


                        ORG     0x0005
MAIN
;                       những dòng lệnh được viết ở đây
                        END
;================================================  ========================
```

Như vậy, chúng ta đã biết cách viết một chương trình đầy đủ dành cho vi điều khiển PIC bằng ngôn ngữ MPASM. Các bạn cần chú ý thêm, nếu phía trên chỗ biến ở băng 2, các bạn không đặt biến gì cả, thì các bạn cứ để nguyên như vậy, vì ngay bên dưới, các bạn đã đặt lại địa chỉ 0x0000, nó chẳng ảnh hưởng gì đến chương trình. Cũng giống như, nếu bạn không viết gì ở đoạn ORG 0x0000 và GOTO MAIN, mà bạn để ngay dòng ORG 0x0005 thì chương trình vẫn chạy bình thường. Đơn giản là từ đoạn 0x0000 đến 0x0004, PIC sẽ không làm gì cả. Chúng tôi đang cố gắng từng bước hình thành cho bạn kết cấu chương trình viết bằng MPASM, mỗi ngày một hoàn thiện hơn, để các bạn nắm rõ lý do vì sao các chương trình được viết như vậy, và chúng ta cùng thống nhất với nhau ở điểm này khi viết chương trình. Nếu các bạn tin tưởng vào việc tạo ra một chuẩn viết chương trình MPASM cho Việt Nam, thì các bạn là người đang đặt nền móng cho nó. Tôi cũng có tham vọng này, cho nên các quy cách ký hiệu tôi cố gắng dùng một chuẩn thống nhất, và mong rằng các bạn cùng tôi làm việc này, để sau này tất cả mọi người khi làm việc cùng với nhau có thể hiểu và truyền tải ý tưởng một cách nhanh nhất.

Kể từ nay, các bạn đã biết cách đặt biến, biết cách viết phần khởi tạo, chúng ta sẽ chỉ còn bàn tới việc viết ở phần chương trình chính như thế nào nữa mà thôi.

Code:

```
;================================================  ============================
                        ORG     0x0000
                        GOTO    MAIN

                        ORG     0x0005
MAIN
                        BANKSEL TRISB
                        CLRF            TRISB           ; đặt portb là output

                        MOVLW           D'255'
                        MOVWF           COUNT_L ; COUNT_L là 1 byte

                        BANKSEL PORTB
LOOP                    BSF             PORTB,  0
                        CALL            DELAY
                        BCF             PORTB,  0
                        CALL            DELAY
                        GOTO            LOOP

;================================================  ============================
; Các chương trình con
;================================================  ============================
DELAY                   DECFSZ          COUNT L,        F
                        GOTO            DELAY
                        RETURN

;================================================  ============================
                        GOTO            $
```

```
                        END
;================================================  ============================
```

Các bạn vừa làm gì với đoạn chương trình trên?

Điểm thứ nhất các bạn nên chú ý, đó là việc tôi thêm phần các chương trình con vào trong phần chương trình chính. Phần cuối chương trình tôi vẫn luôn để là GOTO $ và kết thúc với lệnh END. Tạm thời các bạn cứ viết như vậy để khoá chương trình ở dòng GOTO $, khi chương trình nhảy đến đó, nó sẽ thực hiện vòng lặp vô cùng tại chỗ, còn lệnh END là lệnh bắt buộc.

Việc này giúp chúng ta phần tách rạch ròi phần chương trình con và chương trình chính để tránh nhầm lẫn. Bởi vì ở đây chúng ta mới bắt đầu các bài học cơ bản, cho nên tôi cho rằng các chương trình của các bạn viết là ngắn, nên chúng ta chưa đi xa hơn về việc phân bổ vị trí này. Các bạn chỉ đơn giản hiểu là chúng ta cần phải bỏ đoạn chương trình con ở đâu đó, và chúng ta nên tách thêm một phần nữa để dành riêng cho việc viết chương trình con. Việc làm này về sau sẽ rất có lợi, nhưng tạm thời chúng ta khoan bàn tới, và chúng ta cứ viết như vậy đã.

Phân tích về đoạn chương trình con này, chúng ta thấy chương trình con luôn bao gồm như sau:

Code:

```
[NHÃN]
                        các câu lệnh
                        RETURN
```

Lưu ý rằng ở trên, chúng ta gọi chương trình con CALL DELAY. Như vậy, việc gọi hàm được thực hiện bằng lệnh CALL [NHÃN].

Con trỏ chương trình sẽ nhảy về [NHÃN] được gọi. Nó thực hiện các lệnh nằm từ nhãn đó trở đi. Thực hiện cho đến khi gặp lệnh RETURN, nó sẽ quay trở về và thực hiện lệnh tiếp theo ngay bên dưới lệnh CALL. Ở đây, chúng ta gặp phải một vấn đề, đó là khái niệm Top of Stack. Tuy nhiên, chúng ta tạm gác nó lại cho bài học sau, còn bây giờ các bạn chỉ cần nắm được việc thực hiện lệnh CALL bao giờ cũng đi kèm với một nhãn. Con trỏ nhảy tới nhãn và thực hiện các lệnh bên trong đó, đến khi gặp lệnh RETURN thì nó nhảy trở về vị trí nằm sau lệnh CALL đó và thực hiện tiếp công việc đang làm.

Vì bỏ qua khái niệm Top of Stack, cho nên đề nghị các bạn không đặt ra câu hỏi nếu trong các lệnh thực hiện, nó lại có một lệnh CALL gọi đi chỗ khác thì làm thế nào? Chúng ta sẽ giải quyết vấn đề này ở phần sau.

Thế bên trong hàm DELAY chúng ta làm những gì?

Lưu ý rằng, ở trên chương trình chính, sau khi đã khởi tạo PORTB là ngõ output, các bạn thấy chúng ta đã ghi giá trị d'255' vào biến COUNT_L. Cách viết giá trị như sau:

b'11001010' để xác định số nhị phân

d'234' để xác định số thập phân

0xF3 để xác định số thập lục phân

Lưu ý:

Số nhị phân chỉ có các giá trị 0 và 1, và tối đa dài 8 bit. Số thập phân chỉ có thể có giá trị từ 0 đến 255, và số thập lục phân chỉ có giá trị từ 00 đến FF

Quay trở lại, biến COUNT_L đang mang giá trị 255.

Khi thực hiện hàm DELAY, các bạn thực hiện lệnh DECFSZ (DECrement File, Skip if Zero), có nghĩa là nó sẽ giảm giá trị của một thanh ghi nào đó một đơn vị. Nếu sau khi giảm xong, mà kết quả là 0, thì nó sẽ nhảy cách ra một ô nhớ trong bộ nhớ chương trình, và thực hiện lệnh tiếp theo đó. Nếu giá trị sau khi giảm một đơn vị chưa bằng 0, thì nó sẽ thực hiện lệnh liền kề với nó.

Như vậy, vòng lặp được thực hiện như sau:

Code:

```
COUNT_L = 255 (ở trên đã đặt)

DELAY     COUNT_L = COUNT_L - 1
if COUNT_L <> 0
        GOTO DELAY
if COUNT_L = 0
        RETURN
```

Code:

```
Lệnh DECFSZ     [File], F/W
```

Nếu phía sau dấu phẩy, chúng ta để W, thì kết quả sẽ lưu vào thanh ghi W, và [File] không thay đổi giá trị gì hết. Nhưng ở đây, chúng ta muốn thực hiện như đoạn mã giả ở trên, nên chúng ta phải để là F. COUNT_L sẽ giảm dần từ 255 đến 1, trong quá trình đó nó cứ chạy lên DELAY, rồi giảm COUNT_L một đơn vị, xong lại nhảy về DELAY, lại thực hiện việc giảm 1 đơn vị của COUNT_L Khi COUNT_L = 1 nó lại giảm 1 đơn vị, lúc này COUNT_L = 0. Và nó không thực hiện lệnh GOTO nữa, mà thay bằng lệnh NOP, sau đó nó thực hiện lệnh RETURN, có nghĩa là quay về lại lệnh CALL ở trên. Như vậy, các bạn đã hiểu rõ hàm DELAY rồi. Nhưng quan trọng nhất là làm sao tính toán được thời gian hao tốn của đoạn vòng lặp này kể từ khi bắt đầu thực hiện lệnh CALL, vì thực ra chúng ta muốn là muốn biết chính xác thời gian thực hiện lệnh của nó.

Thời gian thực hiện của lệnh CALL DELAY là bao lâu?

Lệnh CALL khi thực hiện tốn 2 chu kỳ máy, như vậy chúng ta ghi chú là (2) ở đây.
Lệnh DECFSZ tốn 1 chu kỳ máy khi giá trị trả về khác 0. Như vậy, trong quá trình thực hiện giảm từ 255 xuống 1, nó thực hiện 255 - 1 = 254 lần. Mỗi lần thế này nó tốn 1 chu kỳ máy, chúng ta ký hiệu (254) ở đây. Khi thực hiện lệnh GOTO, lệnh GOTO tốn 2 chu kỳ máy, vậy nó cũng thực hiện 254 lần, chúng ta ký hiệu (254 x 2 = 506) ở đây.
Khi COUNT_L = 1, nó vẫn thực hiện lệnh DECFSZ, vậy nó tốn thêm 1 chu kỳ máy nữa (1). Sau khi thực hiện lệnh này, kết quả trả về là 0, vậy nó sẽ thực hiện một lệnh NOP (1), và sau đó thực hiện lệnh RETURN, lệnh RETURN tốn 2 chu kỳ máy (2)
Kết quả:
(2) + (254) + (508) + (1) + (1) + (2) = 768 chu kỳ máy
Nếu chúng ta dùng thạch anh 10MHz, 1 chu kỳ máy tốn 0.4 us, có nghĩa là lệnh CALL DELAY tốn 768 * 0.4 us tức là khoảng 1/3000 giây.
Chúng ta khoan bàn đến việc xa hơn, vậy thì chúng ta đã biết cách tính thời gian hao tốn của hàm DELAY rồi. Nhưng nếu tính như thế này thì quá mất công, chúng ta có thể chuyển nó thành công thức cụ thể như sau:
CALL = 2
DELAY (COUNT_L) = [COUNT_L - 1] * (DECFSZ + GOTO) + 1 + 1
RETURN = 2
Các bạn nên nhớ công thức này để sau này phát triển lên tính các công thức khác.
Có lẽ hôm nay chúng ta tạm dừng bài học ở đây
Các bạn lưu ý, tôi có tính sai một đoạn phía trên, vì quáng gà hay sao đó, tính từ 255 xuống 1 giảm chỉ có 253 lần. Đúng là phải 254 lần. Như từ 2 giảm xuống 1 thì chỉ có 1 lần thôi. Xin thành thật cáo lỗi với các bạn.

Tổng kết

Các bạn đã học được gì ngày hôm nay?
- Các bạn đã hiểu được khái niệm chu kỳ máy, dao động thạch anh tạo ra, PIC sẽ thực hiện 1 lệnh trong vòng 4 dao động của thạch anh. Như vậy, chu kỳ máy của PIC sẽ là chu kỳ dao động của thạch anh nhân với 4, hay tần số PIC sẽ bằng tần số thạch anh chia 4.
- Các bạn đã học được cách đặt biến trong một chương trình viết bằng MPASM, các bạn đã có thể đặt biến ở bất kỳ băng nào các bạn muốn
- Sau đó, các bạn bổ sung phần đặt biến này vào trong sườn chương trình lần trước đã học, các bạn hoàn thiện hơn sườn một chương trình viết bằng MPASM
- Các bạn lại thêm vào sườn chương trình đó phần các chương trình con, vậy tôi thông báo với các bạn rằng các bạn chỉ còn thiếu 2 phần nữa là ngắt (Interrupt) và bảng (Table) nữa, là các bạn đã có thể có một sườn chương trình viết bằng MPASM hoàn chỉnh. Các bạn sẽ không phải đợi lâu để hoàn tất sườn chương trình này.
- Các bạn học được cách dùng hàm CALL và RETURN, nó luôn luôn đi kèm từng cặp với nhau.
- Các bạn học thêm các lệnh: BCF, CALL, RETURN, DECFSZ

Tài liệu tham khảo:

Các bạn tham khảo datasheet PIC16F84A, PIC16F628A và PIC16F88 để biết thêm chi tiết về cấu trúc bộ nhớ dữ liệu, vì có cái thì có băng 2, có cái không có, có cái lại có băng 3, băng 4.... Nhớ chú ý phần tập lệnh để đọc hiểu thêm về các lệnh vừa học (Instruction Set). Các bạn có thể dùng keyword: DELAY để tìm trong trang http://www.piclist.com/ những đoạn chương trình con viết về hàm DELAY, làm thế nào để viết hàm DELAY dài hơn?...

Lưu ý cuối cùng, đó là các bạn đang chuẩn bị trở thành một người viết PIC chuyên nghiệp, do đó, các bạn cần phải nhớ các chân nào của PIC để thiết kế mạch và điều khiển, các bạn nên in hình sơ đồ chân của PIC ra để dán lên trước bàn làm việc. Các bạn có thể download bản in tại đây (có trong datasheet, nhưng tôi muốn gửi trực tiếp cho các bạn để các bạn đỡ mất công).

Bài tập làm thêm:

1) Các bạn thấy rằng, nếu thời gian DELAY quá ngắn, trên thực tế các bạn sẽ khó thấy đèn LED nhấp nháy. Vì vậy, thay vì viết một hàm CALL DELAY, các bạn viết một dọc 20 dòng CALL DELAY liên tiếp nhau, các bạn sẽ thấy sự khác biệt

2) Nhưng nếu viết 20 dòng CALL DELAY thì cũng như viết 20 dòng lệnh NOP, vậy có nghĩa là các bạn vẫn có thể thực hiện một vòng lặp, trong đó lặp lại 20 lần, và trong vòng lặp các bạn thực hiện hàm DELAY. Như vậy, các bạn phải viết một hàm DELAY_NGOAI để bên trong thực hiện hàm DELAY_TRONG. Chính vì vậy, tôi gợi ý cho các bạn tìm trong trang web http://www.piclist.com/ để tìm các source code hàm DELAY, và các bạn sẽ biết phải làm sao để viết hàm DELAY chờ lâu hơn. Quan trọng nhất là các bạn phải chỉ ra được công thức tính toán thời gian của hàm DELAY mà các bạn viết. (bài tập tính điểm)

3) Bây giờ các bạn có thể điều khiển một đèn LED, vậy nếu muốn 8 đèn LED nháy theo thứ tự nào đó chẳng hạn, các bạn sẽ làm thế nào? (bài tập tính điểm)

Ghi chú: (bài tập tính điểm) là những bài tập mà chúng tôi sẽ cộng đồn vào để tặng PIC cho các bạn nào tham gia giải bài như thông báo về việc bán PIC.

***** &&& *****

Bài 3: Ngắt (interrupt)

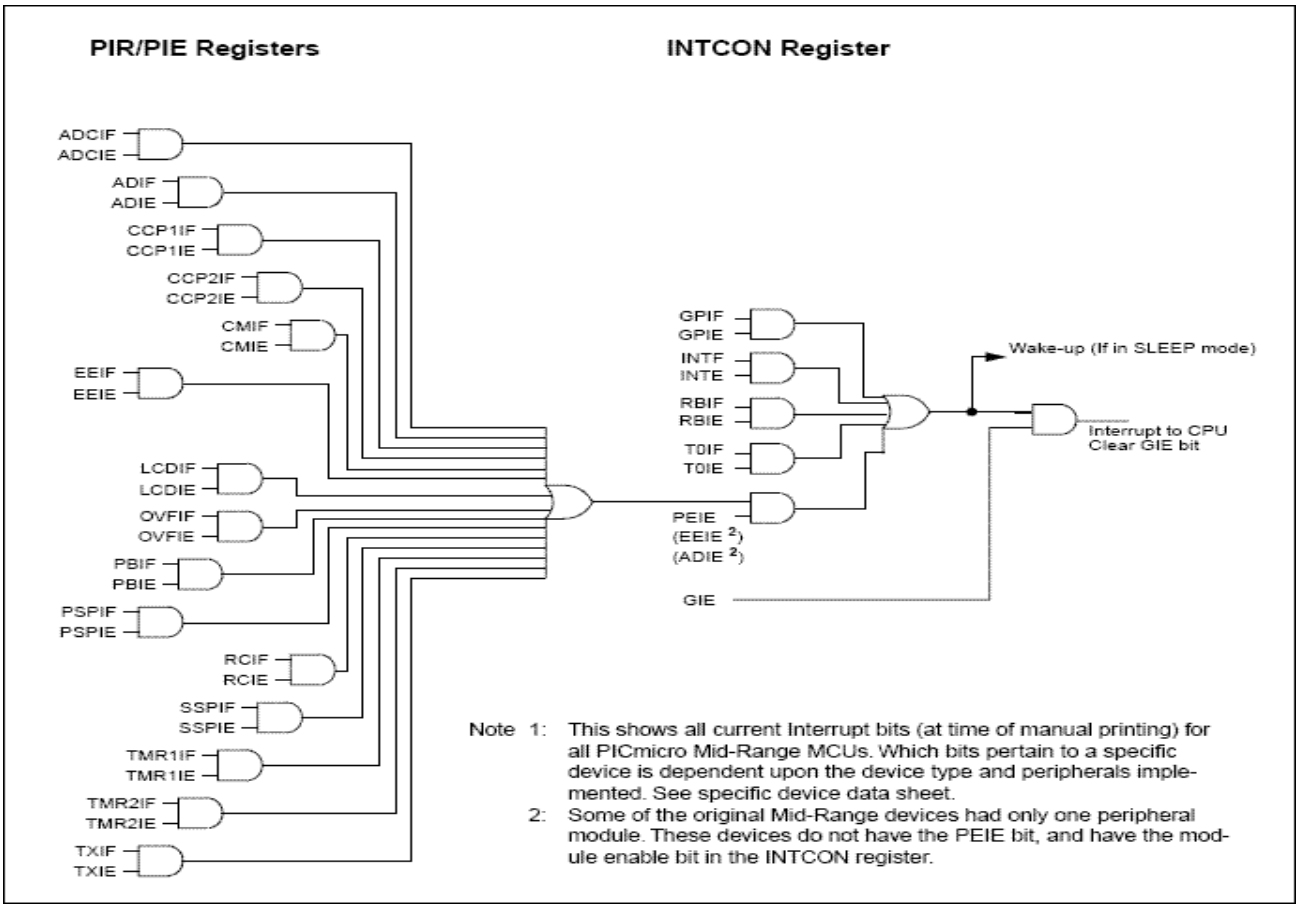
Giới thiệu

Khái niệm ngắt là một khái niệm rất phổ biến trong tất cả các hệ thống vi điều khiển, vi xử lý và máy tính. Vậy ngắt là gì? Các bạn hình dung hình ảnh chúng ta đang đi xe máy trên bờ ruộng, con đường đi rất dài và rất thẳng, bỗng nhiên có một con bò chạy ngang, húc chúng ta xuống ruộng. Cả xe và người lao xuống ruộng. Chúng ta lồm cồm bò dậy, phủi quần áo, chửi đổng lên một cái vì chẳng biết chửi ai, thế là chúng ta đem ông trời ra chửi. Sau đó, chúng ta dắt xe máy lên bờ ruộng, tại cái chỗ mà chúng ta bị húc té xuống, rồi chúng ta lấy xe chạy tiếp. Nếu lỡ có một con bò nào khác, lại húc chúng ta.. thì....Hoạt động ngắt cũng giống như vậy, khi chúng ta đang chạy một chương trình chính nào đó, bỗng nhiên có một sự kiện xảy ra, chúng ta phải dừng việc chúng ta đang làm lại, và giải quyết cái sự việc xảy ra đó. Cuối cùng, chúng ta lại quay trở về cái chỗ mà chúng ta đã tạm dừng lại lúc nãy và tiếp tục công việc đang làm.

Khái niệm ngắt chỉ đơn giản như vậy, tuy nhiên, đối với vi điều khiển nói chung, và PIC nói riêng, ngắt có thể do rất nhiều nguồn xảy ra, và với mỗi nguồn ngắt khác nhau, chúng ta có thể định trước rằng trong ngắt đó chúng ta sẽ làm việc gì. Cũng như khi đi trên bờ ruộng, chúng ta có thể bị bò húc, cũng có thể bị trâu húc, cũng có thể bị vấp cục đã, cũng có thể bị lọt ổ gà... Và nếu như bị bò húc thì chúng ta chửi ông trời, bị trâu húc chúng ta mắng ông trăng, bị vấp cục đá chúng ta tự trách mình xui xẻo, và đến khi vấp ổ gà... thì chúng ta vô nhà thương...

Các nguồn ngắt trong PIC:

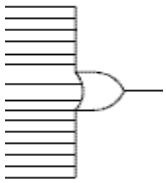
Số lượng và loại nguồn ngắt trong PIC rất đa dạng, và rất khác nhau ở mỗi dòng PIC. Do vậy không thể liệt kê hết ra đây tất cả các dòng PIC và tất cả các loại ngắt trong từng dòng được. Chúng ta chỉ đưa ra đây sơ đồ tổng quát của các nguồn ngắt, và đi sâu vào một số loại ngắt phổ biến.



Hình 1: Các nguồn ngắt của dòng PIC Midrange
Chúng ta chú ý đến một số điểm sau:

1) Trong hình có các ký hiệu cổng logic AND và OR

Đây là cổng AND, có nghĩa là chỉ khi đầu vào của hai cổng này đều có giá trị là 1, thì đầu ra mới có giá trị là 1. Chúng ta quan sát một góc hình bên trái phía dưới TXIF và TXIE, chúng đi qua cổng AND, chỉ khi nào bit TXIE bật lên, và bit TXIF cũng được bật lên, thì lúc đó ngõ ra nối vào cổng OR phía trên mới có giá trị.



Đây là cổng OR, có nghĩa là chỉ cần một trong các tín hiệu ngõ vào có giá trị là 1, thì ngõ ra sẽ có giá trị là 1. Như vậy, nếu cả TXIE và TXIF đều có giá trị 1, thì ngõ ra sau cổng AND của chúng sẽ có giá trị 1 và ngõ ra sau cổng OR cũng có giá trị 1, bởi vì ít nhất cổng OR ở đây cũng có 1 ngõ vào có giá trị 1. Chúng ta cứ tiếp tục như vậy mà suy ra.

2) Điểm thứ hai, là các chữ đuôi IE và IF:

IE ở đây là viết tắt của chữ Interrupt Enable, và IF ở đây viết tắt của Interrupt Flag.
IE có nghĩa là chúng ta cho phép kích hoạt một loại ngắt nào đó xảy ra hay không. Đây là tín hiệu mà chúng ta có thể quy định ngay từ ban đầu. Mặc định, tất cả chúng đều có giá trị 0, chỉ khi nào chúng ta cho phép một ngắt nào đó xảy ra, thì về sau nó mới xảy ra ngắt đó thôi.
Cũng giống như, ban đầu trên bờ ruộng có dãy rào chắn, thì con bò không thể nào húc bạn té được, nếu bạn bỏ hàng rào ra, thì nếu có con bò húc bạn, bạn sẽ té. Nguyên lý này đơn giản như vậy thôi.

IF ở đây là các cờ ngắt. Tức là khi bạn bị bò húc, thì có một người cầm cờ giơ lên báo là

bạn đã bị bò húc, để những người dưới ruộng reo hò...hihi... Và tất nhiên, khi bạn không phá rào cản thì người trên ruộng vẫn có. Và khi con bò lao vào bạn, thì người ta cũng phất cờ lên như thường, nhưng bị cái rào cản nên bạn cứ thoải mái mà đi con đường của bạn, chẳng phải quan tâm đến việc té xuống, chửi bới hay trèo lên làm gì.
Cái rào cản chính là IE và cái sự việc cuối cùng mà bạn vẫn đi hay lồm cồm bò dậy đó chính là cái cổng AND mà chúng ta vừa nói trên kia.

3) Điểm thứ ba, các lớp ngắt:

Bạn thấy rằng, trong hình, rõ ràng có 2 lớp ngắt. Lớp thứ nhất nằm bên tay trái ngoài cùng, lớp thứ hai nằm ở giữa hình. Lớp thứ ba chỉ có một cổng AND nên chúng ta không kể tới làm gì.

Lớp thứ nhất được gọi là lớp ngắt ngoại vi.

Thực chất lớp này vì có quá nhiều nguồn ngắt, và các nguồn ngắt này đều là một số chuẩn giao tiếp, hoặc chức năng đặc biệt của PIC, cho nên người ta phân ra làm lớp ngắt ngoại vi. Để các ngắt ngoại vi hoạt động, trước tiên chúng ta phải cho phép ngắt ngoại vi, tức là bật bit PIE lên. Còn cụ thể muốn cho ngắt ngoại vi nào hoạt động, thì chúng ta bật ngắt đó lên. Trên sơ đồ các bạn cũng thấy rõ thông qua các cổng AND và OR.

Lớp thứ hai tạm gọi là lớp ngắt phổ thông.

Khi muốn dùng các nguồn ngắt phổ thông, chúng ta chỉ việc bật các bit IE của nguồn ngắt này. Tất nhiên, cuối cùng, chúng ta phải bật ngắt toàn cục GIE thì ngắt mới được phép xảy ra (kể cả ngắt ngoại vi và ngắt phổ thông. Khi đó, PIE được coi là một nguồn ngắt phổ thông. Điều này cũng giống như khi bạn chạy xe trên bờ ruộng, một hàng rào dài chạy dọc theo con đường, chính là ngắt toàn cục GIE. Lớp bên ngoài thứ hai là lớp ngắt phổ thông, bao gồm luôn cả ngắt ngoại vi PIE. Và ngoài cùng là các hàng rào thuộc lớp ngắt ngoại vi.

Nếu các bạn bật các nguồn ngắt, mà không bật ngắt toàn cục GIE thì cho dù ngắt có xảy ra, thì chương trình vẫn không dừng để thực hiện ngắt, giống như con bò có thể lao qua hàng rào ngoài cùng đã được mở, nhưng vẫn còn hàng rào trong cùng.

Như vậy, các bạn đã hiểu một cách tổng quan về hoạt động ngắt của PIC, những nguyên tắc phải bật hay tắt ngắt. Điểm lưu ý cuối cùng, đó là tôi muốn giới thiệu với các bạn rằng, chữ ký hiệu trong bảng, là tên các bit liên quan đến việc bật tắt ngắt. VD: bit PIE, INTE.. nằm trong thanh ghi INTCON (ngắt phổ thông), các bit quy định ngắt ngoại vi nằm trong các thanh ghi PIR và PIE.

Vectơ ngắt của PIC:

Như lần trước đã giới thiệu, vectơ ngắt của PIC nằm ở vị trí 0x0004 các bạn xem lại hình sau. Khác với khi bạn bị té ruộng, bạn té xuống ngay tại chỗ bạn bị húc, đối với vi điều khiển, khi xảy ra interrupt, nó sẽ nhảy về một địa chỉ cố định, và thực hiện công việc tại đó. Sau khi thực hiện xong, nó sẽ quay trở về vị trí mà từ đó nó đã thoát ra. Vị trí cố định mà nó sẽ nhảy về khi xảy ra ngắt là vị trí 0x0004.

Chương trình ngắt:

Lại quay về thí dụ té ruộng, có lẽ tôi thích cái thí dụ này vì nó có thể giúp bạn hình dung mọi thứ. Bây giờ các bạn hãy chia giai đoạn từ khi bị bò húc, té xuống ruộng, rồi bạn chửi đổng lên, rồi bạn lồm cồm bò lên. Vậy cho dù bạn bị bò húc, hay bị vấp ổ gà, thì chỉ có giai đoạn bạn chửi đổng lên là khác nhau, còn lại, giai đoạn bạn té xuống ruộng là té xuống ruộng, và sau đó thì bạn cũng bò lên. Vậy ngắt cũng giống thế, khi nhảy vào ngắt, bạn sẽ có một giai đoạn cần phải nhảy vào ngắt, và một giai đoạn nhảy ra khỏi ngắt, còn bên trong ngắt đó các bạn làm cái gì là nội dung cần thực hiện của từng nguồn ngắt. Tôi cung cấp ra đây đoạn chương trình ngắt chuẩn, từ nay về sau, các bạn chỉ cần copy đoạn chương trình này và sử dụng:

Code:

```
;================================================  ==================
                      ORG     0x0000
                      GOTO    MAIN
                      ORG     0x0004
                      GOTO    INTERRUPT
                      ORG     0x0005
MAIN
; đây là phần chương trình chính của các bạn

;================================================  ==================
```

```
INTERRUPT

                        RETFIE
;================================================  ==================
; Các chương trình con được viết ở đây

;================================================  ==================
                        GOTO    $
                        END
;================================================  ==================
```

Như vậy, một lần nữa, chúng ta bổ sung sườn chương trình của chúng ta một cách chi tiết hơn. Chúng ta vừa thêm vào một đoạn chương trình con INTERRUPT. Thực ra, gọi INTERRUPT là một chương trình con cũng không sai, nhưng vì nó khá đặc biệt, nên chúng ta cứ tách rời nó ra.

Khởi tạo và kết thúc ngắt:

Tôi cung cấp dưới đây đoạn chương trình khởi tạo và kết thúc ngắt đầy đủ cho PIC, từ nay về sau, khi muốn sử dụng ngắt, các bạn chỉ cần copy và paste đoạn code này lại, hoàn toàn không cần sửa chữa gì và cứ thế sử dụng. Tôi sẽ dành cho các bạn đặt câu hỏi về phần này để từ các câu hỏi, có thể giải thích rõ hơn vì sao chúng ta lại viết như vậy, từng điểm một. Nếu không, tôi không thể có thời gian để viết tất cả mọi vấn đề về ngắt ra đây được.

Code:

```
;================================================
==========================================
INTERRUPT

;------------------------------------------
;Doan ma bat buoc de vao ngat
;------------------------------------------

                        MOVWF   W_SAVE                  ;W_SAVE(bank unknown!) = W
                                SWAPF           STATUS, W
                                CLRF            STATUS                  ; force bank 0
for remainder of handler
                                MOVWF   STAT_SV         ; STAT_SV = swap_nibbles(
STATUS )
                                                                        ; STATUS = 0
                                MOVF    PCLATH, W
                                MOVWF   PCH_SV          ; PCH_SV = PCLATH
                                CLRF            PCLATH          ; PCLATH = 0
                                MOVF    FSR,    W
                                MOVWF   FSR_SV          ; FSR_SV = FSR
                                                        ; 10 cycles from interrupt to
here!

;----------------------------------
;Doan chuong trinh ngat
;----------------------------------

; cac ban se viet chuong trinh ngat o day

;------------------------------------------------
;Doan ma bat buoc de ket thuc ngat
;------------------------------------------------

                        MOVF    FSR_SV, W
                        MOVWF   FSR             ; FSR = FSR_SV
                                MOVF            PCH_SV, W
                                MOVWF   PCLATH          ; PCLATH = PCH_SV
                                SWAPF   STAT_SV,        W
```

```
                                MOVWF   STATUS          ; STATUS = swap_nibbles(
STAT SV )
                                SWAPF           W_SAVE, F
                        SWAPF   W_SAVE, W       ; W = swap(swap( W_SAVE )) (no change
Z bit)

                                BSF     INTCON, GIE
                        RETFIE
;*************************************************
*************************************************  =
```

Như vậy, chương trình ngắt được chia làm 3 phần chính.
Phần thứ nhất là phần bắt đầu vào ngắt, đây là đoạn chương trình bắt buộc, tất nhiên không hoàn toàn nghiêm ngặt như vậy, vì thực tế nhiều khi bạn không dùng đến tất cả các lệnh này, nhưng vì mục đích cung cấp các khái niệm cơ sở, và công cụ làm việc đầy đủ, tôi cung cấp cho bạn chương trình ngắt chi tiết. Phần thứ hai là phần chương trình ngắt của bạn. Khi xảy ra ngắt, bạn muốn làm cái gì, thì bạn bắt đầu viết từ phần này trở đi. Phần thứ ba là phần kết thúc ngắt, bạn cứ viết nguyên bản như vậy không cần sửa đổi. Tạm thời, sẽ không có các phân tích chi tiết giống như các bài học trước, các bạn có thể tự tìm hiểu thêm, nếu không, có thể đặt câu hỏi, và chúng ta sẽ từ từ tìm hiểu rõ hơn về ngắt của PIC. Thời gian tới đây, có lẽ tôi hơi bận, cho nên tôi không thể viết bài liên tục được, mong rằng các bạn cố gắng tìm hiểu và học tốt PIC. Đến giai đoạn này, các bạn đã có thể dùng con PIC, giống như một con 89C51 thông thường. Và các bạn thấy đấy, thực sự PIC chỉ cần 1 ngày để học.

Chúng ta vừa học xong 3 bài học cơ bản nhất của một con vi điều khiển: Điều khiển port, viết hàm delay và viết chương trình ngắt.
Phần thứ tư của bài viết chương trình ngắt, sẽ đi chi tiết vào các ngắt và giải thích rõ nghĩa từng ngắt. Nhưng thiết nghĩ, tôi nên kết hợp bài học này ở đây, và kết hợp phần thứ tư vào bài học sau: Nút bấm và các ngõ vào của PIC.

Tài liệu tham khảo:

http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1824&appnote=en011006

Bài 4. Nút bấm

Các bạn vừa biết khái niệm ngắt, và đã biết chương trình ngắt được viết như thế nào. Vậy bây giờ chúng ta chuyển đến bài tiếp theo về nút bấm.

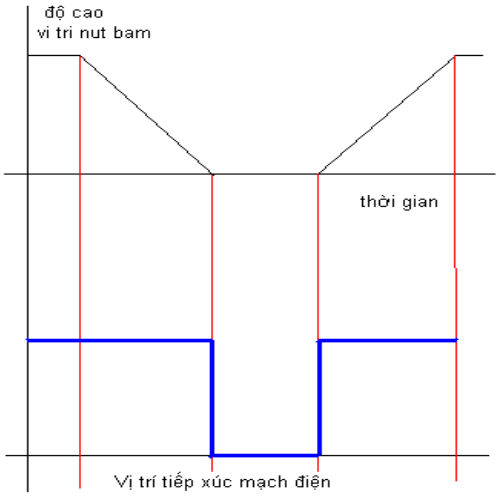
Công dụng của nút bấm

Nút bấm là một hình thức ra lệnh phổ biến nhất trên thế giới. Bạn gọi một cái thang máy, bạn bấm nút, bạn kêu cửa thì bấm chuông, bạn bật đèn thì bấm nút công tắc, và tôi đang ngồi viết cho bạn bằng cách bấm nút bàn phím...

Như vậy, bạn đã biết công dụng của cái nút bấm. Bây giờ các bạn sẽ học cách làm một cái nút bấm!!! Điều này có vẻ buồn cười, nhưng với vi điều khiển, và máy tính, khả năng xử lý các lệnh rất đa đạng. Bạn có thể bấm cùng một nút, nhưng lệnh sẽ khác nhau ở mỗi thời điểm, và mỗi trạng thái. Ví dụ, như bạn nhấp chuột máy tính, thực ra cũng là bạn nhấp nút bấm, nhưng bạn thấy rõ ràng rằng, ở những vị trí di chuyển chuột khác nhau, nút bấm của chuột sẽ đưa ra các mệnh lệnh khác nhau cho máy tính thực hiện.

Một số trạng thái nút bấm thông dụng

Trạng thái nút bấm ra lệnh tức thời, đó là khi bạn bấm nút, lập tức mọi trạng thái phải được kiểm tra và chương trình dừng lại để thực hiện lệnh từ nút bấm của bạn. Có nghĩa là bạn ra lệnh tại thời điểm bấm nút, và máy hiểu rằng bạn đã bấm nút. Còn việc xử lý thế nào thì hồi sau phân giải. Trạng thái chờ nút bấm, đó là chương trình bạn đang chạy, đến một giai đoạn nào đó, nó cần phải có sự ra lệnh của bạn bằng nút bấm, và chương trình chờ bạn bấm nút để chạy tiếp, hoặc bắt đầu một công việc nào đó sau khi chờ. Nhắc lại thao tác bấm nút một chút, cái nút của bạn đang ở trên cao, bạn bấm nó xuống thì nó sẽ có một giai đoạn nút bấm đi xuống, khi chạm vào mạch điện, hiển nhiên bạn muốn hay không muốn thì cũng phải có một khoảng thời gian bạn giữ cho nút bấm tiếp xúc với mạch điện, sau đó là giai đoạn bạn thả nút bấm ra.



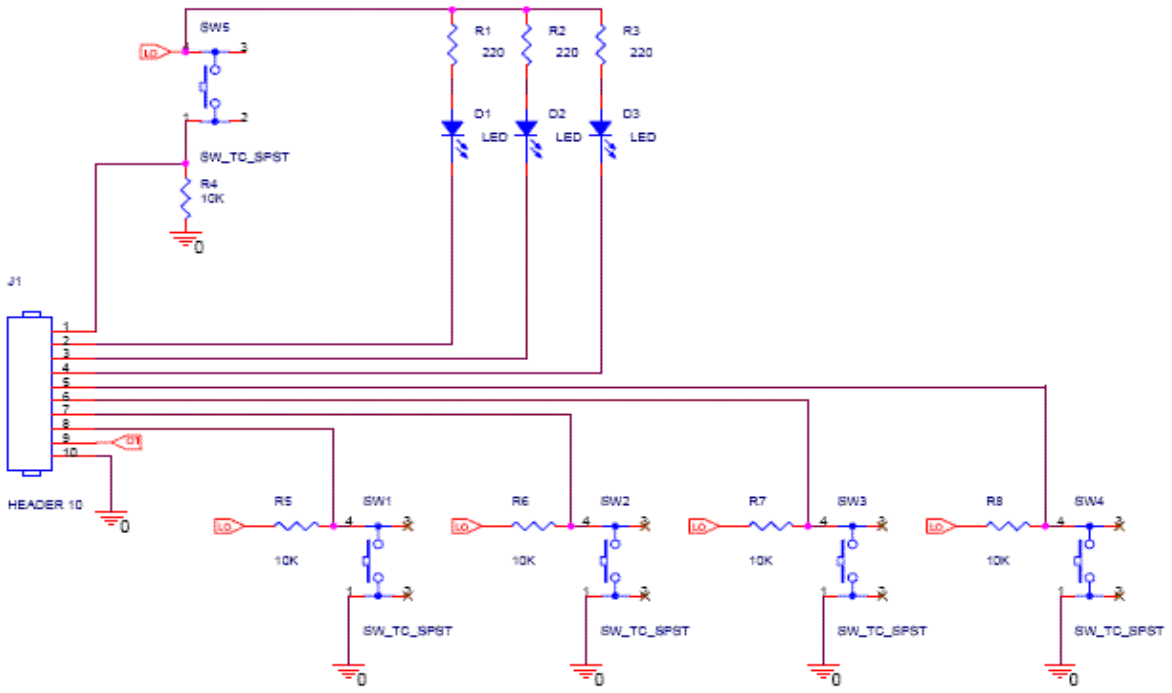
Theo dõi hình trên, chúng ta thấy. Khi bấm nút, có quá trình đi xuống của nút bấm, và quá trình đi lên của nút bấm. Nhưng thực tế, đối với mạch điện trong nút bấm, nó chỉ có thể nhận được trạng thái tiếp xúc hoặc không tiếp xúc, nên tín hiệu nhận được sẽ như đường màu xanh trong hình dưới. Chúng ta chỉ quan tâm đến trạng thái của đường màu xanh trong các ứng dụng của nút bấm.

Vậy, trạng thái nút bấm lại có thêm 3 trạng thái nữa là trạng thái bấm xuống, trạng thái giữ nút bấm, và trạng thái nhả nút bấm lên. Kết hợp với 2 trạng thái điều khiển trên, chúng ta có 6 trạng thái phổ biến của nút bấm. Các bạn lưu ý rằng, chúng ta có 6 trạng thái chứ không phải chỉ có 4 trạng thái, vì thực ra rất nhiều người cho rằng chỉ có 4 trạng thái khi cho rằng trạng thái chờ trong lúc giữ nút bấm không phải là trạng thái phổ biến. Nhưng nếu các bạn đã từng dùng điện thoại di động thì các bạn thấy số người dùng trạng thái chờ của nút bấm cũng không phải là con số nhỏ.

Ở đây, tôi muốn tán dóc một chút rằng, khi các bạn làm việc về khoa học kỹ thuật, và đến một khi các bạn khó có thể tìm ra đường hướng suy nghĩ để giải quyết một vấn đề khoa học kỹ thuật, hãy tìm mối liên hệ với nó trong khoa học xã hội. Chính vì vậy, các bạn thường thấy tôi hay đưa ra những ví dụ xã hội để minh hoạ cho vấn đề kỹ thuật cần được giải quyết.

Tôi sẽ dành việc ứng dụng từng trạng thái nút bấm phổ biến trong các ứng dụng cho các bạn, còn ở đây, tôi chỉ muốn nhân bài học này để tiếp tục bài học về interrupt mà chúng ta đã bỏ dở trước đó. Vậy chúng ta chỉ xét trạng thái khi bấm nút, lập tức lệnh sẽ được thực hiện, tức trạng thái tức thời của nút bấm.

Các bạn hãy làm bài tập thực hành, thực hiện một mạch điện tử như hình sau để chuẩn bị cho bài học của chúng ta.



Trong mạch điện này, chúng ta thấy có một vài điểm đặc biệt khi có 1 nút bấm nối giữa chân của PIC và nguồn, còn các nút bấm khác lại nối chân của PIC với đất.
Giữa nguồn và đất luôn có một điện trở 10K. Vì sao chúng ta phải nối mạch điện như vậy? Chúng ta tạm dừng bài học về nút bấm ở đây và theo dõi bài học cơ bản về điện tử tiếp theo.

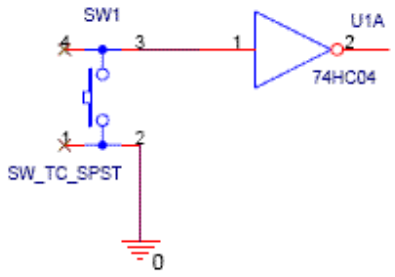
Điện tử cơ bản

Giới thiệu

Đây là phần rất cơ bản về điện tử, mà các bạn khi bắt đầu làm việc với vi điều khiển cần phải nắm rõ. Như đã nói, PIC tạo ra dòng điện khoảng 20mA và điện áp khoảng 5V, tương tự như vậy, nếu dòng ngõ vào quá cao so với 20mA và điện áp ngõ vào quá cao so với 5V, thì PIC sẽ bị hư. Vì vậy, bài học này trang bị cho các bạn một số khái niệm cơ bản về điện tử, để các bạn có thể nắm vững nguyên lý thiết kế mạch và tính toán các giá trị điện trở cần thiết. Đáng lẽ bài học này cần được thực hiện ngay từ đầu, tuy nhiên, tôi cho rằng bài tập đèn LED quá đơn giản, các bạn chưa biết gì cũng có thể hiểu được, nhưng nay, nếu như các bạn mới học về điện tử và vi điều khiển không được trang bị kiến thức cơ bản này, có thể làm cho các bạn lúng túng vì một số điểm không được làm rõ trong mạch điện tử.

Hiện tượng trôi điện áp

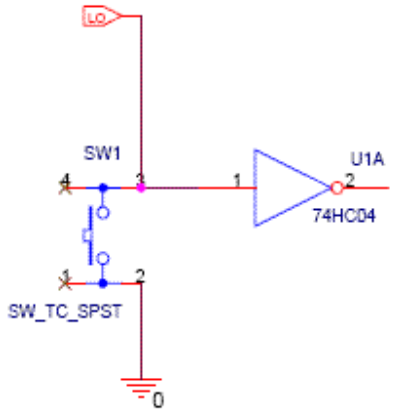
Các bạn xem hình sau:



Chúng ta cho rằng ngõ vào của PIC, cũng giống như ngõ vào của một linh kiện điện tử thông dụng là 74HC04. Thay vì vẽ một cái chân PIC, thì chúng tôi vẽ hình một con 74HC04 cho nó đơn giản, và để các bạn dễ hình dung. Nếu để một con PIC lên một hình thì quá phức tạp hình ảnh, và lại không cần thiết. Hơn nữa, bài viết này được tham khảo từ tài liệu Very Basic Circuits của Encoder, và trong trang web này, người ta sử dụng 74HC04 để làm thí dụ, tôi tôn trọng ví dụ này nên khi viết lại bài viết cũng sử dụng 74HC04 giống như họ.

Các bạn thấy, nếu như nút bấm được nhấn xuống, thì ngõ vào của 74HC04 hay PIC được nối với Mass. Như vậy, lúc đó PIC có thể đọc giá trị 0. Tuy nhiên, nếu nút nhấn được thả ra, chúng ta thấy rằng ngõ vào của PIC chẳng được nối với một linh kiện nào, vậy là điện áp ở chân của PIC sẽ trôi nổi không xác định được. Nếu không may mắn, điện áp trôi nổi này rơi vào vùng logic 0, rồi lại nhảy sang vùng logic 1... thì các bạn thấy rõ ràng chúng ta không thể xác định được nút bấm có được bấm hay không?!!

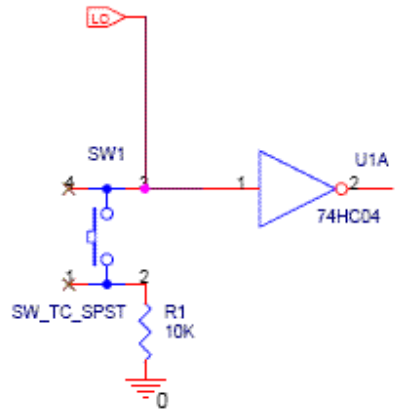
Vì vậy, để đảm bảo, nếu khi không bấm nút, PIC phải có điện áp tham khảo là 5V, sau khi bấm nút thì điện áp sẽ giảm xuống 0V, như vậy mức logic mới thật rõ ràng, không thể để trôi nổi như hình trên. Vậy chúng ta có hình dưới đây



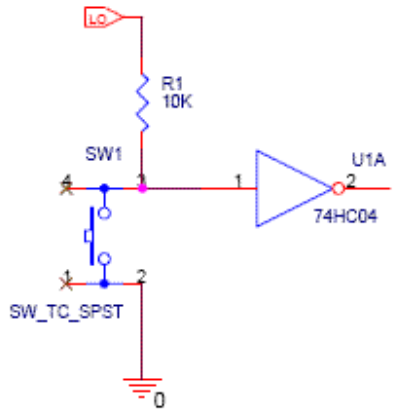
Công dụng của điện trở kéo lên

Các bạn lại thấy, nếu bây giờ không bấm nút, thì điện áp ngõ vào của PIC sẽ là 5V. Nhưng nếu bấm nút một cái, rõ ràng chúng ta gây ra ngắn mạch khi nối trực tiếp từ nguồn xuống đất. Chính vì vậy, chúng ta phải đưa thêm vào một điện trở giữa đất, nút bấm và nguồn.

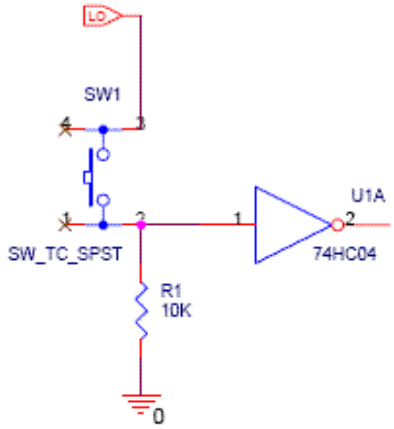
Có hai vấn đề đặt ra, đó là điện trở sẽ đặt ở đâu, và giá trị của nó bằng bao nhiêu.
Chúng ta xem hình này:



Các bạn sẽ bực mình ngay rằng, đúng là thằng ngốc mới đặt điện trở như thế này, bởi vì nếu đặt điện trở như vậy, dù bạn có bấm nút hay không bấm nút thì điện áp ngõ vào vẫn luôn luôn là 5V, vậy nút bấm trở nên vô nghĩa.
Thế thì chỉ còn một cách đặt điện trở như hình tiếp theo đây(H.a):



(H.a)



(H.b)

Vậy vấn đề còn lại là giá trị điện trở bằng bao nhiêu?
Các bạn sẽ thấy, PIC hoạt động ở 20mA và 5V trên các chân. Vì vậy, khi chưa bấm nút, nguồn 5V được nối với điện trở và đi vào chân của PIC. Nếu như trong một trường hợp nào đó chân của PIC chuyển từ chế độ input sang output, thì vấn đề xảy ra là dòng trên chân phải đảm bảo nhỏ hơn hoặc bằng 20mA. Như vậy, trong thiết kế trên, chúng ta xem dòng tại chân PIC nếu PIC đặt ở 0V là:
I = U/R = 5V/ 10000 Ohm = 5mA
Như vậy, thiết kế này đảm bảo cho hoạt động của PIC được an toàn.
Khi đóng nút bấm dòng 5mA này sẽ đi xuống đất, và chân của PIC được nối với đất.
Các bạn xem tiếp hình sau(H.b)
Trường hợp này, nút bấm được nối với nguồn 5V. Điện trở nối giữa chân của PIC với đất sẽ không làm cho PIC có hiện tượng trôi nổi điện áp, và khi đóng nút bấm thì dòng vẫn ở 5mA.

Tổng kết:

Qua bài học này, các bạn đã hiểu được cơ bản về khái niệm điện trở kéo lên (trường hợp điện trở nối với nguồn), và điện trở kéo xuống (trường hợp điện trở nối với đất). Giá trị điện trở được đặt ở đây nhằm loại bỏ hiện tượng ngắn mạch, và đảm bảo ngõ vào của PIC khoảng 20mA. Khi an toàn, cần thiết kế sao cho ngõ vào nhỏ.

To be continue …