TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP.HCM
KHOA ĐIỆN TỬ
BỘ MÔN VIỄN THÔNG

VI XỬ LÝ 2

Biên soạn: Nguyễn Đình Phú

TP.HCM 2007

LỜI NÓI ĐẦU

Ở vi xử lý 1 chúng ta đã khảo sát các vi xử lý cơ bản 8 bit của INTEL và Zilog và vi điều khiển họ MCS-51 chủ yếu là AT80C51 hay mới hơn là AT89C51 và AT89S51 hoặc AT89S52.

Do mới bắt đầu tiếp cận nên việc giảng dạy cho người học các loại vi xử lý và các loại vi điều khiển ở trên là phù hợp về nội dung và tạo điều kiện thuận lợi để người học dễ tiếp cận với chi phí đầu tư thấp vì các vi xử lý và các vi điều khiển đó có giá thành không cao.

Sau khi đã nắm bắt được các kiến thức cơ bản của vi xử lý 1 thì người học dễ dàng tiếp cận vớii các vi xử lý vi điều khiển mạnh hơn, tích hợp nhiều chức năng hơn để giải quyết các yêu cầu thực tế đa dạng hơn rất nhiều.

Trong tài liệu này sẽ giới thiệu vi điều khiển họ PIC của hãng Microchip được sử dụng rất nhiều trong các thiết bị điều khiển công nghiệp.

Tài liệu được chia làm 3 chương: chương 1 thiết kế các ứng dùng dùng vi điều khiển họ AT89S từ các yêu cầu thực tế. Trong chương này tác giả đưa ra các yêu cầu điều khiển trong thực tế và hướng dẫn phân tích yêu cầu, các thiết kế và viết các chương trình. Qua các bài thiết kế này nhằm giúp cho người học biết cách thiết kế một hệ thống từ yêu cầu thực tế.

Chương 2 giới thiệu về vi điều khiển PIC chủ yếu là chip 16F877A đang được sử dụng nhiều. Trong chương này giới thiệu cấu trúc, các khối tích hợp bên trong.

Chương 3 giới thiệu về ngôn ngữ lập trình cho PIC và giới thiệu các mạch lập trình cho PIC và các chương trình ví dụ nhằm giúp người học làm quen với PIC.

Hãng ATMEL có nhiều vi điều khiển như họ AVR và họ ATMEGA cũng có các tính năng mạnh như họ PIC đã trình bày ở trên và hãng MOTOROLA cũng có các vi điều khiển được sử dụng khá phổ biến như do thời gian môn học có hạn nên tác giả không trình bày.

Xin cảm ơn các bạn bè đồng nghiệp và các bạn sinh viên đã tốt nghiệp đóng góp vào tập giáo trình này.

Mọi đóng góp xây dựng xin hãy gởi về theo địa chỉ phu_nd@yahoo.com - xin chân thành cảm ơn.

MỤC LỤC

Chương 1

THIẾT KẾ CÁC ỨNG DỤNG DÙNG VI ĐIỀU KHIỂN

LIỆT KÊ CÁC HÌNH

BÀI THIẾT KẾ SỐ 1:
"DO DÒNG ĐIỆN GIỜ CAO ĐIỂM VÀ GIỜ THẤP ĐIỂM".

ĐẶT VẤN ĐỀ

Trong hệ thống cung cấp điện thì việc thay đổi tải diễn ra liên tục và có 2 khoảng thời gian trong một ngày đó là thời gian cao điểm và thời gian thấp điểm.

Thời gian cao điểm vào khoảng 18 giờ đến 20 giờ và cao điểm nhất là 17 giờ – khi đó tất cả mọi người sử dụng rất nhiều thiết bị điện trong sinh hoạt gia đình – dòng điện đạt đến đỉnh điểm trong ngày và làm giảm điện áp và nếu công suất cung cấp không đủ sẽ sinh ra hiện tượng quá tải – làm hư hỏng thiết bị cung cấp, mất điện và gây thiệt hại.

Thời gian thấp điểm vào khoảng 2 giờ sáng khi mọi người đã yên giấc ngủ và tất cả các thiết bị sinh hoạt đã tắt – dòng điện hạ thấp nhất trong ngày và điện áp có tăng trở lại.

Các kỹ sư và công nhân trong các nhà máy, trạm phân phối, trạm cung cấp cần phải khảo sát và nắm bắt được sự biến động đó để kiểm soát chúng, tránh sự qúa tải gây thiệt hại cho thiết bị điện.

Một số các trạm điện phải yêu cầu nhân viên trực trạm tiến hành đo dòng điện tại các biến áp vào thời điểm giờ cao điểm và giờ thấp điểm để thống kê, để kiểm soát.

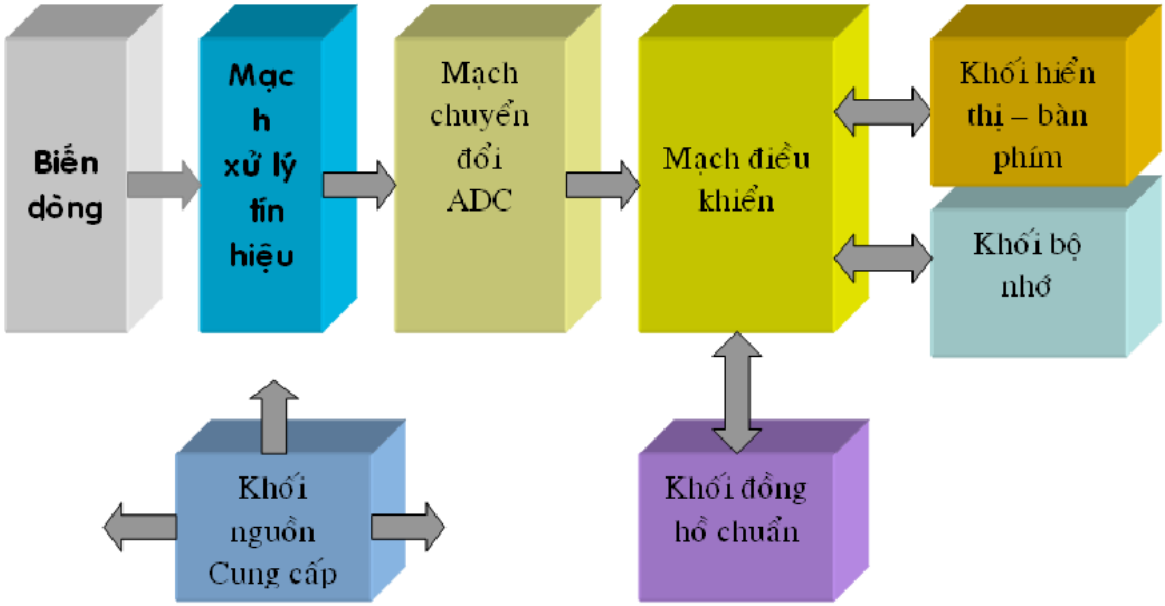
Việc đo dòng được tiến hành vào ban đêm và trên những trụ điện cao ở những nơi xa thì rất khó khăn và nguy hiểm đến tính mạng nhân viên.

GIẢI QUYẾT VẤN ĐỀ

Với máy đo dòng tự động lưu lại giá trị đo trong một tháng, 2 tháng hoặc nhiều hơn sẽ giúp cho nhân viên có thể đi lấy kết quả đo vào ban ngày an toàn hơn và nếu kết quả đo chính xác và máy đo có thể giao tiếp với máy tính thì dữ liệu có được hoàn toàn chính xác, nếu có thêm chương trình vẽ biểu đồ hay thống kê sẽ giúp đánh giá sự biến thiên của tải và đánh giá công suất cung cấp của biến áp hay đường dây có đủ công suất hay không và khắc phục nhanh.

1. THIẾT KẾ SƠ ĐỒ KHỐI CỦA HỆ THỐNG:

Sơ đồ khối của máy đo do người nghiên cứu thiết kế như hình 1:



Hình 1-1. sơ đồ khối của hệ thống.

2. PHÂN TÍCH CHỨC NĂNG CÁC KHỐI:

a. Khối biến dòng điện:

- Với một dòng điện cung cấp rất lớn để có thể đo được chúng ta phải sử dụng biến dòng để chuyển đổi dòng điện cao thành dòng điện thấp hơn nhiều lần tùy thuộc vào hệ số của biến dòng.
- Việc lựa chọn biến dòng để thực hiện tùy thuộc vào dòng điện đang đo nằm trong khoảng nào và đáp ứng của mạch điện tử ADC có thể là bao nhiêu.

b. Khối xử lý tín hiệu:

- Tín hiệu của cuộn thứ cấp thường mắc nối tiếp với một điện trở để chuyển đổi dòng điện thành điện áp và nếu bằng với điện áp chuyển đổi của ADC thì không xử lý nữa và nếu chưa bằng thì phải xử lý cho bằng cách sử dụng thêm mạch khuếch đại hoặc mạch hạn chế biên độ.

c. Khối chuyển đổi ADC:

- Tín hiệu đo dạng tương tự phải được chuyển thành tín hiệu số để có thể hiển thị và lưu trữ – giao tiếp với máy tính . Bộ chuyển đổi ADC sử dụng càng nhiều bit càng tốt.

d. Khối hiển thị và bàn phím:

- Khối này hiển thị các thông tin đo được để kiểm tra đúng hay sai.
- Bàn phím dùng để ra lệnh nhằm xem lại kết quả đo.

e. Khối nhớ:

- Khối này dùng để lưu lại các giá trị đã đo được trong vòng một tháng hoặc nhiều tháng tùy thuộc vào bộ nhớ đang sử dụng có dung lượng lớn hay nhỏ.

f. Khối thời gian thực:

- Việc đo dòng điện thực hiện vào đúng 2 khoảng thời gian 19 giờ và 2 giờ trong tất cả các lần đo. Khối này tạo ra thời gian thực hoạt động như một đồng hồ thực và vẫn hoạt

đúng khi nguồn cung cấp không còn. Khối điều khiển sẽ lấy thời gian từ hệ thống này để tiến hành việc đo đúng theo thời gian đã qui định.

g. Khối điều khiển:

- Khối này là thành phần chính trong hệ thống sẽ điều khiển khối ADC thực hiện qúa trình chuyển đổi tín hiệu tương tự thành tín hiệu số – lưu trữ dữ liệu vào bộ nhớ – hiển thị kết quả ra màn hình và giao tiếp với máy tính.

3. THIẾT KẾ MẠCH:

Sau khi đã thiết kế sơ đồ khối ta tiến hành chọn linh kiện và thiết kế sơ đồ nguyên lý.

Khối điều khiển được chọn là vi điều khiển 89S52 có 32 đường điều khiển IO và dung lượng bộ nhớ là 8 kbyte, có giao tiếp nối tiếp với máy tính.

Khối ADC được chọn là ADC 12 bit 7109. Với 12 bit nên cấp chuyển đổi có 4096 cấp – điều này phù hợp với sự thay đổi khá lớn của dòng điện cần đo.

Khối hiển thị được chọn là LCD vì hiển thị được nhiều thông tin. Trong hệ thống này chọn LCD có 16 kí tự và 2 hàng.

Khối bộ nhớ: kết qủa đo nằm trong phạm vi 4095A nên số bit cần sử dụng để biểu diễn kết quả đo này là 16 bit – 2 byte, một ngày đo 2 lần và sử dụng 4 ô nhớ byte để lưu trữ . Một tháng 31 ngày sẽ sử dụng 124 ô nhớ byte để lưu với số lượng bộ nhớ như vậy sử dụng loại bộ nhớ nối tiếp họ 24C08 có dung lượng 1kbyte = 1024 ô nhớ byte.

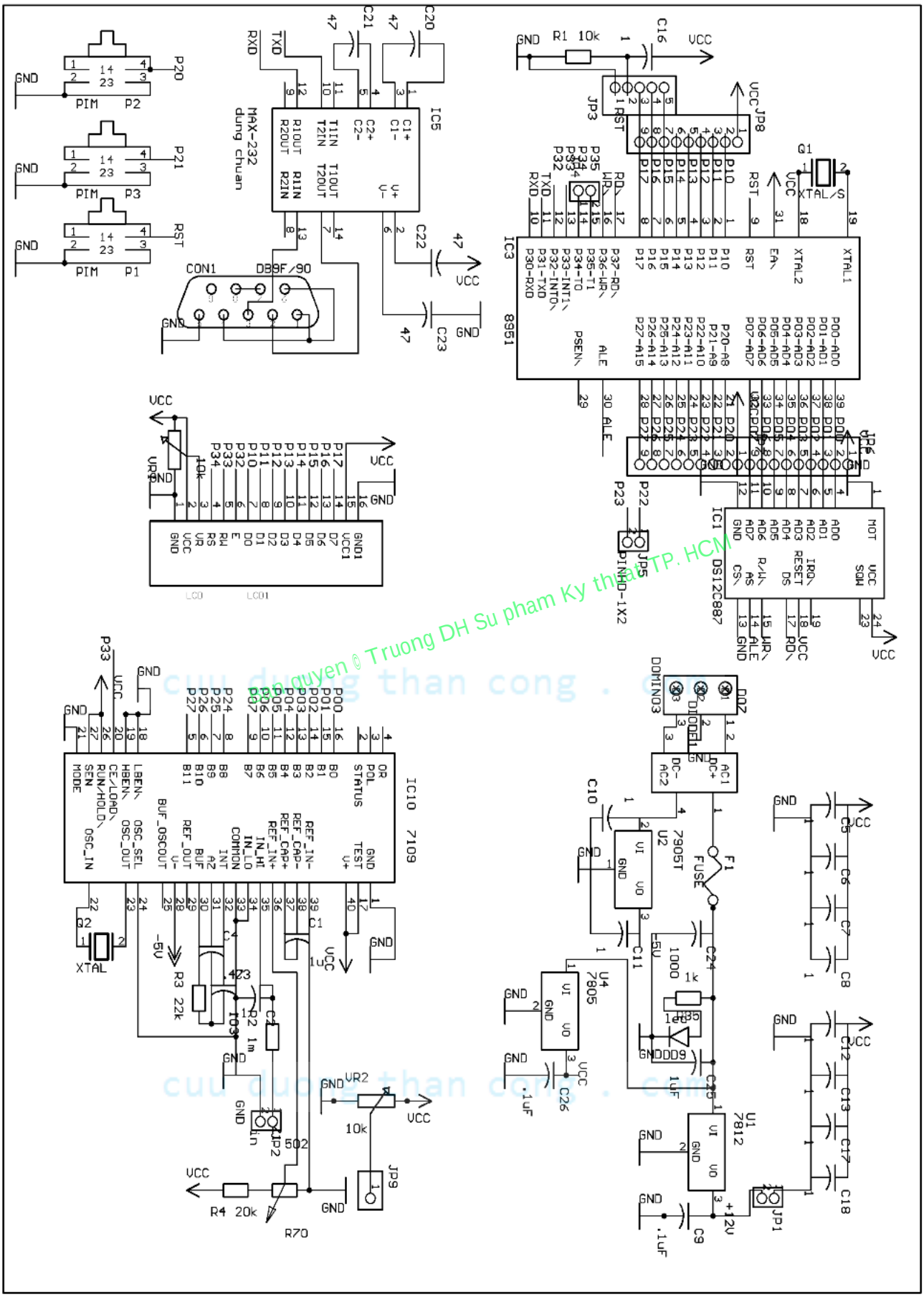
Khối bàn phím chỉ dùng để xem lại kết quả đo nên ta chỉ cần dùng 2 phím để đọc lần lược từng kết quả đo đã lưu trong bộ nhớ.

Khối thời gian thực sử dụng IC Dallas DS12C887. Chúng ta phải sử dụng đồng hồ thời gian thực để vì chúng hoạt động chính xác và vẫn hoạt động khi mất điện, việc đo dòng với yêu cầu phải theo đúng giờ đã qui định và để giảm bớt sự phức tạp của chương trình nên giải pháp sử dụng đồng hồ thực là tốt nhất.

Khối nguồn cung cấp gồm có nguồn +5V và – 5V.

Từ sơ đồ khối và các linh kiện đã chọn ta phải nắm rõ hoạt động của từng linh kiện và cách kết nối sau đó ta tiến hành thiết kế sơ đồ nguyên lý cho hệ thống.

Sơ đồ mạch của hệ thống như hình 2.

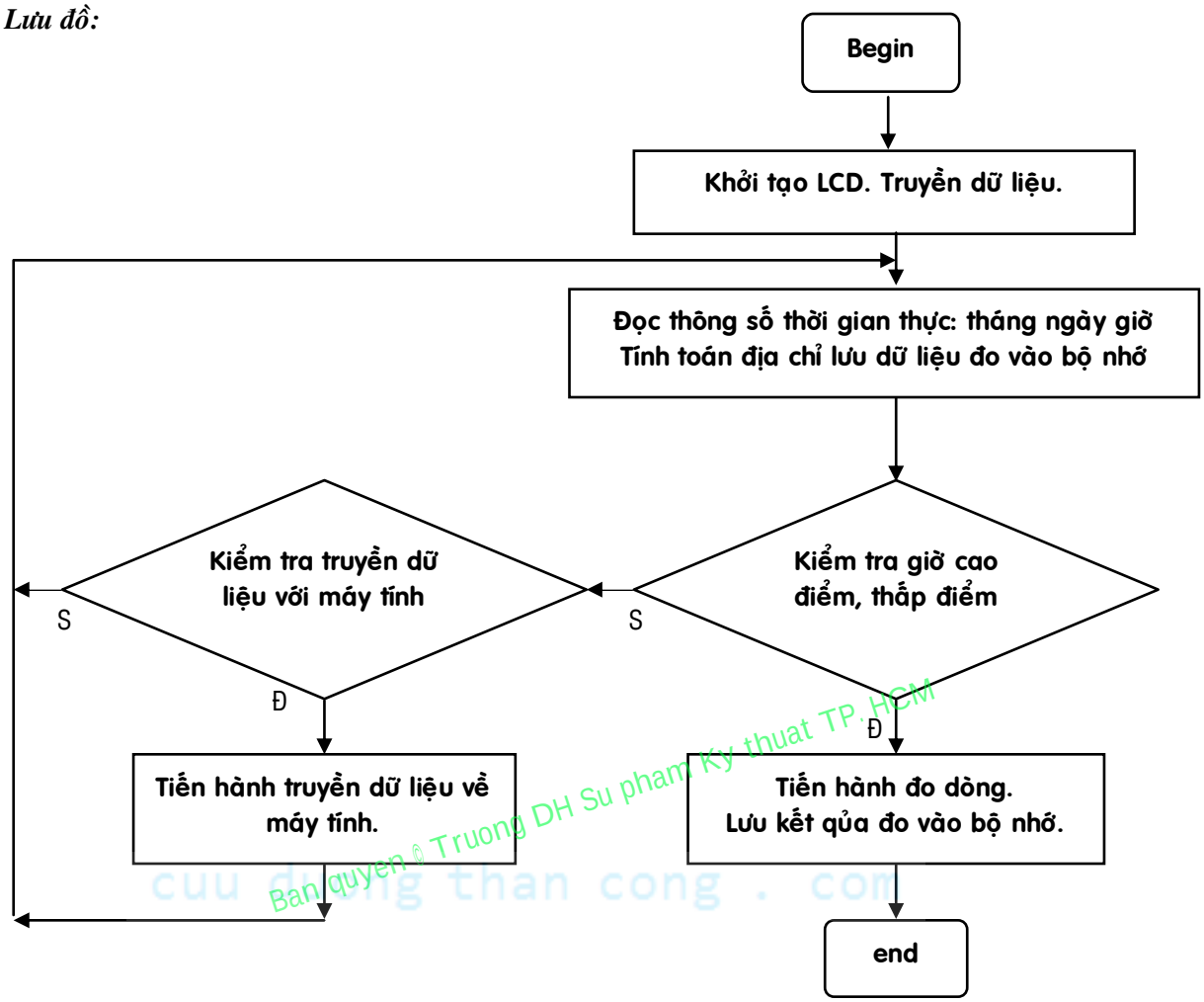


Hình 1-2. Sơ đồ nguyên lý.

4. VIẾT CHƯƠNG TRÌNH CHO HỆ THỐNG:

Sau khi đã thiết kế xong ta tiến hành viết lưu đồ và chương trình điều khiển cho máy đo.

Lưu đồ:



Dựa vào lưu đồ điều khiển ta tiến hành viết chương trình cho hệ thống đo.

```
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;chuong trinh dong dong gio cao diem thap diem
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
                pmode   bit     p2.0            ;dinh nghia phim mode
                phim    bit     p2.1            ;dinh nghia phim
                pcon    equ     87h

                giay    equ     r2
                phut    equ     r3
                gio     equ     r4
                ngay    equ     r5
                thang   equ     r6

                kqbytel equ     11h
                kqbyteh equ     12h

                bienmode        equ     10h

;cac vung nho se su dung de luu thong tin
;               30h ->  3fh de luu thong tin cho LCD hang thu 1
;               40h ->  4fh de luu thong tin cho LCD hang thu 2

;bang phan chia bo nho 24C08 de luu tru ket qua do
;tu ngay 1 den ngay 9
;       010H -> 011h luu ket qua do gio thap diem ngay 1
;       012H -> 013h luu ket qua do gio thap cao  ngay 1
```

```
;       020H -> 021h luu ket qua do gio thap diem ngay 2
;       022H -> 023h luu ket qua do gio thap cao  ngay 2

;       030H -> 031h luu ket qua do gio thap diem ngay 3
;       032H -> 033h luu ket qua do gio thap cao  ngay 3

;       040H -> 041h luu ket qua do gio thap diem ngay 4
;       042H -> 043h luu ket qua do gio thap cao  ngay 4

;       050H -> 051h luu ket qua do gio thap diem ngay 5
;       052H -> 053h luu ket qua do gio thap cao  ngay 5

;       060H -> 061h luu ket qua do gio thap diem ngay 6
;       062H -> 063h luu ket qua do gio thap cao  ngay 6

;       070H -> 071h luu ket qua do gio thap diem ngay 7
;       072H -> 073h luu ket qua do gio thap cao  ngay 7

;       080H -> 081h luu ket qua do gio thap diem ngay 8
;       082H -> 083h luu ket qua do gio thap cao  ngay 8

;       090H -> 091h luu ket qua do gio thap diem ngay 9
;       092H -> 093h luu ket qua do gio thap cao  ngay 9

;       100H -> 101h luu ket qua do gio thap diem ngay 10
;       102H -> 103h luu ket qua do gio thap cao  ngay 10

;tu ngay 11 den ngay 20

;       110H -> 111h luu ket qua do gio thap diem ngay 11
;       112H -> 113h luu ket qua do gio thap cao  ngay 11

;       120H -> 121h luu ket qua do gio thap diem ngay 12
;       122H -> 123h luu ket qua do gio thap cao  ngay 12

;       130H -> 131h luu ket qua do gio thap diem ngay 13
;       132H -> 133h luu ket qua do gio thap cao  ngay 13

;       140H -> 141h luu ket qua do gio thap diem ngay 14
;       142H -> 143h luu ket qua do gio thap cao  ngay 14

;       150H -> 151h luu ket qua do gio thap diem ngay 15
;       152H -> 153h luu ket qua do gio thap cao  ngay 15

;       160H -> 161h luu ket qua do gio thap diem ngay 16
;       162H -> 163h luu ket qua do gio thap cao  ngay 16

;       170H -> 171h luu ket qua do gio thap diem ngay 17
;       172H -> 173h luu ket qua do gio thap cao  ngay 17

;       180H -> 181h luu ket qua do gio thap diem ngay 18
;       182H -> 183h luu ket qua do gio thap cao  ngay 18

;       190H -> 191h luu ket qua do gio thap diem ngay 19
;       192H -> 193h luu ket qua do gio thap cao  ngay 19

;       200H -> 201h luu ket qua do gio thap diem ngay 20
;       202H -> 203h luu ket qua do gio thap cao  ngay 20


;tu ngay 21 den ngay 30

;       210H -> 211h luu ket qua do gio thap diem ngay 21
;       212H -> 213h luu ket qua do gio thap cao  ngay 21
```

```
;           220H -> 221h luu ket qua do gio thap diem ngay 22
;           222H -> 223h luu ket qua do gio thap cao  ngay 22

;           230H -> 231h luu ket qua do gio thap diem ngay 23
;           232H -> 233h luu ket qua do gio thap cao  ngay 23

;           240H -> 241h luu ket qua do gio thap diem ngay 24
;           242H -> 243h luu ket qua do gio thap cao  ngay 24

;           250H -> 251h luu ket qua do gio thap diem ngay 25
;           252H -> 253h luu ket qua do gio thap cao  ngay 25

;           260H -> 261h luu ket qua do gio thap diem ngay 26
;           262H -> 263h luu ket qua do gio thap cao  ngay 26

;           270H -> 271h luu ket qua do gio thap diem ngay 27
;           272H -> 273h luu ket qua do gio thap cao  ngay 27

;           280H -> 281h luu ket qua do gio thap diem ngay 28
;           282H -> 283h luu ket qua do gio thap cao  ngay 28

;           290H -> 291h luu ket qua do gio thap diem ngay 29
;           292H -> 293h luu ket qua do gio thap cao  ngay 29

;           300H -> 301h luu ket qua do gio thap diem ngay 30
;           302H -> 303h luu ket qua do gio thap cao  ngay 30

;           310H -> 311h luu ket qua do gio thap diem ngay 31
;           312H -> 313h luu ket qua do gio thap cao  ngay 31

;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
                org     0000h
                sjmp    mmain
                org     0023h
                ljmp    ngatnhan
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;chtrinh chinh
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
mmain:          mov     sp,#70h
                mov     ie,#10010000b           ;cho phep ngat truyen du lieu
                mov     bienmode,#0             ;do va cho phep hien thi ket qua do

                lcall   khoitao_trxd            ;goi chtr con khoi tao truyen du lieu giua 2 vdk
                lcall   khoitao_lcd             ;goi chtr khoi tao LCD
                lcall   copydata1               ;copy du lieu hien thi 1

main3:          lcall   docdongho               ;goi chtr con doc thoi gian
                lcall   giaima                  ;goi chtr con giai ma thong so thoi gian d hien thi
                lcall   hienthichung            ;goi chtr con hien thi thong tin ra LCD

main:           cjne    gio,#2h,main1           ;kiem tra gio thap diem
                cjne    phut,#0h,main1          ;kiem tra gio thap diem
                cjne    giay,#0h,main1          ;kiem tra gio thap diem
                ljmp    dogio_thapdiem          ;nhay den do dong gio thap diem


main1:          cjne    gio,#19h,main2          ;kiem tra gio cao diem
                cjne    phut,#00h,main2 ;kiem tra gio cao diem
                cjne    giay,#00h,main2 ;kiem tra gio cao diem
                ljmp    dogio_caodiem           ;nhay den do dong gio cao diem

main2:          jb      pmode,main3             ;kiem tra phim mode
                jnb     pmode,$         ;cho buon phim
                mov     bienmode,#1             ;do binh thuong nhung hien thi ket qua do trong bo nho
                ljmp    main3                   ;nhay ve lai tu dau
```

```
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

dogio_thapdiem:
                mov     a,ngay          ;ket qua do vao a
                anl     a,#0f0h         ;giu nguyen hang chuc ngay
                swap    a               ;chuyen xuong 4 bit thap
                mov     dph,a           ;chuyen sang cho dph

                mov     a,ngay          ;ket qua do vao a
                anl     a,#0fh          ;giu nguyen hang don vi ngay
                swap    a               ;chuyen len 4 bit cao
                mov     dpl,a           ;chuyen sang cho dpl

                lcall   docadc7109      ;goi chtr con doc du lieu so tu ADC 7109
                lcall   luuketquado
                lcall   hex_to_bcd
                lcall   giaima_kqdo     ;goi giai ma ket qua do
                ljmp    main3

;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
dogio_caodiem:  mov     a,ngay          ;ket qua do vao a
                anl     a,#0f0h         ;giu nguyen hang chuc ngay
                swap    a               ;chuyen xuong 4 bit thap
                mov     dph,a           ;chuyen sang cho dph

                mov     a,ngay          ;ket qua do vao a
                anl     a,#0fh          ;giu nguyen hang don vi ngay
                swap    a               ;chuyen len 4 bit cao
                add     a,#8
                mov     dpl,a           ;chuyen sang cho dpl

                lcall   docadc7109      ;goi chtr con doc du lieu so tu ADC 7109
                lcall   luuketquado
                lcall   hex_to_bcd
                lcall   giaima_kqdo     ;goi giai ma ket qua do

                ljmp    main3

;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;chuong trinh con doc gio phut giay
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
docdongho:      mov     r0,#0
                movx    a,@r0           ;doc giay
                mov     giay,a

                mov     r0,#2
                movx    a,@r0           ;doc phut
                mov     phut,a

                mov     r0,#4
                movx    a,@r0           ;doc gio
                mov     gio,a

                mov     r0,#7
                movx    a,@r0           ;doc ngay cua thang
                mov     ngay,a

                mov     r0,#8
                movx    a,@r0           ;doc thang
                mov     thang,a
                ret
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;chtr con giai ma thoi gian doc
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

```
giaima:     mov   a,gio
            anl   a,#0f0h
            swap  a
            add   a,#30h
            mov   30h,a          ;o nho luu tru ma hien thi hang chuc gio

            mov   a,gio
            anl   a,#0fh
            add   a,#30h
            mov   31h,a          ;o nho luu tru ma hien thi hang don vi gio

            mov   a,phut
            anl   a,#0f0h
            swap  a
            add   a,#30h
            mov   33h,a          ;o nho luu tru ma hien thi hang chuc phut

            mov   a,phut
            anl   a,#0fh
            add   a,#30h
            mov   34h,a          ;o nho luu tru ma hien thi hang don vi phut

            mov   a,giay
            anl   a,#0f0h
            swap  a
            add   a,#30h
            mov   36h,a          ;o nho luu tru ma hien thi hang chuc giay

            mov   a,giay
            anl   a,#0fh
            add   a,#30h
            mov   37h,a          ;o nho luu tru ma hien thi hang don vi giay

            ret
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;chtr con chuyen ket qua do duoc: kqbytel, kqbyteh sang so BCD
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
hex_to_bcd:
            mov   a,kqbyteh
            mov   b,#10
            div   ab
            mov   18h,b          ;luu hang don vi

            mov   b,#10
            div   ab
            mov   19h,a          ;luu hang chuc

            mov   a,b
            swap  a
            orl   18h,a          ;cat hang chuc vao o nho

            mov   r1,kqbytel
            cjne  r1,#0,hex1
            ret

hex1:       mov   a,18h
            add   a,#56h
            da    a
            mov   18h,a
            mov   a,19h
            addc  a,#2
            da    a
            mov   19h,a
            djnz  r1,hex1
            ret
```

```
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;chtr con giai ma ket qua do duoc: kqbytel, kqbyteh
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
giaima_kqdo:    mov     a,19h                   ;byte cao
                anl     a,#0f0h
                swap    a
                add     a,#30h
                mov     48h,a

                mov     a,19h
                anl     a,#0fh
                add     a,#30h
                mov     49h,a

                mov     a,18h                   ;byte thap
                anl     a,#0f0h
                swap    a
                add     a,#30h
                mov     4ah,a

                mov     a,18h
                anl     a,#0fh
                add     a,#30h
                mov     4bh,a
                ret
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;chtr con hien thi thong tin ra LCD MAC DINH KHI KHOI DONG
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
COPYDATA1:      mov     dptr,#fldata1           ;nap dia chi bat dau
                lcall   copydata                ;goi ch tr con copy 32 byte
                lcall   hienthichung
                ret

fldata1:  DB    '               '
fldata2:  DB    'DONG:          '

;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;lay du lieu tu bo nho chuong trinh vao bo nho ram
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
copydata:       mov     r0,#30h
                mov     r1,#0

copydatax:      mov     a,r1
                movc    a,@a + dptr             ;lay data
                mov     @r0,a                   ;cat data
                inc     r1
                inc     r0
                cjne    r1,#33,copydatax
                ret

;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;Chuong trinh con hien thi noi dung  tren LCD cua2 vung nho
;30H->3Fh hang 1; 40H-> 4Fh hang 2;
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
hienthichung:   MOV     A,#080h                 ;set DDRAM
                LCALL   KTAO
                mov     r1,#16
                MOV     R0,#30H

fline:          lcall   Write
                djnz    r1,fline
                mov     a,#0c0h                 ;set DDRAM
                LCALL   KTAO

                mov     r1,#16
```

```
                MOV     R0,#40H
sline:          lcall   Write
                djnz    r1,sline
                ret
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;chtr con goi data hien thi ra LCD
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
write:          MOV     byteout,@R0
                Lcall   data_byte
                inc     r0
                reT
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;chtr con khoi tao LCD

;chuong trinh dieu khien LCD 16X2 tren kit vi dieu khien LOAI NHO
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
                E               BIT     P3.5
                rw              BIT     P3.6
                rs              BIT     P3.7
                byteout         equ     p2

khoitao_lcd:    mov     0a2h,#0
                LCALL   khtaolcd        ;khoi tao lcd
                ret
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;chtr con khoi tao LCD
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
khtaolcd:       setb    e               ;Enable
                clr     rs              ;RS low
                clr     rw              ;RW low

                MOV     a,#38h          ;tu dieu khien LCD
                LCALL   KTAO
                LCALL   ddelay41        ;delay 4.1 mSec

                MOV     A,#38h          ;function set
                LCALL   KTAO
                LCALL   ddelay100       ;delay

                MOV     A,#38h          ;function
                LCALL   KTAO

                MOV     A,#0ch          ;tu dieu khien display on
                LCALL   KTAO
                MOV     A,#01h          ;tu dieu khien Clear display
                LCALL   KTAO

                MOV     A,#06h          ;tu dieu khien entry mode set
                LCALL   KTAO

                MOV     A,#80h          ;thiet lap dia chi LCD (set DD RAM)
                LCALL   KTAO
                RET

;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;chuong trinh con khoi tao LCD
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
KTAO:           mov     byteout,a
                lcall   command_byte
                RET

;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;Feed command/data to the LCD module
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
command_byte:
```

```
                clr     rs              ;RS low for a command byte
                ljmp    bdelay

data_byte:      setb    rs              ;RS high for a data byte
bdelay:         clr     rw              ;R/W low for a write mode
                clr     e
                nop

                setb    e               ;Enable pulse
                nop
                nop

                mov     byteout,#0ffh   ;configure port1 to input mode

                setb    rw              ;set RW to read
                clr     rs              ;set RS to command
                clr     e               ;generate enable pulse

                nop
                nop
                setb    e
                lcall   ddelay100
                ret

;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;chuong trinh con delay 4.1 ms
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
ddelay41:       mov     7eh,#90h
del412:         mov     7fh,#200
                djnz    7fh,$
                djnz    7eh,del412
                ret

;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;chuong trinh con delay 255 microgiay
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
ddelay100:      mov     7fh,#00
                djnz    7fh,$
                ret
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;chon va doc du lieu tu 7109 thu 1
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
docadc7109:     clr     p3.3            ;cho phep doc
                mov     kqbytel,p0      ;doc byte low

                mov     a,#00h          ;doc byte cao
                mov     c,p2.4
                rrc     a
                mov     c,p2.5
                rrc     a
                mov     c,p2.6
                rrc     a
                mov     c,p2.7
                rrc     a
                mov     kqbyteH,p0
                ret



;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;chtr con khoi tao truyen du lieu giua 2 vdk A va B
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
khoitao_trxd:   mov     th1,#0fah
                mov     tl1,#0fah
                anl     tmod,#0fh
```

```
                orl     tmod,#20h
                setb    tr1
                mov     scon,#50h
                setb    ti
                orl     pcon,#80h
                ret
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;chuong trinh con luu ket qua do
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
luuketquado:    lcall   ghibyte
                inc     dptr
                lcall   ghibyte
                ret

ghibyte:        ret

docbyte: ret

;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;chuong trinh con truyen ket qua do ve may tinh:
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
ngatnhan:       jnb     ri,$
                clr     ri
                mov     a,sbuf
                cjne    a,#35h,ngatend
                sjmp    ngatx

ngatend: reti


;goi ngay 1 den ngay 9

ngatx:          mov     dptr,#010h      ;dia chi bat dau

ngat2:          mov     15h,#4          ;so luong byte cua 1 ngay
                mov     a,15h
                lcall   sendbyte


ngat1:          lcall   docbyte
                lcall   sendbyte
                inc     dptr
                djnz    1h,ngat1

                mov     a,dpl           ;chuyen dia chi byte thap vao A
                anl     a,#0            ;xoa 4 bit thap
                swap    a
                inc     a
                swap    a               ;tra lai sau khi tang

                mov     dpl,a           ;chuyen sang ngay tiep theo
                cjne    a,#0ah,ngat2

;goi ngay 10 den ngay 19

                mov     dptr,#100h      ;dia chi bat dau

ngata2:         mov     15h,#4          ;so luong byte cua 1 ngay
                mov     a,15h
                lcall   sendbyte


ngata1:         lcall   docbyte
                lcall   sendbyte
                inc     dptr
```

```
                djnz    1h,ngata1

                mov     a,dpl           ;chuyen dia chi byte thap vao A
                anl     a,#0            ;xoa 4 bit thap
                swap    a
                inc     a
                swap    a               ;tra lai sau khi tang

                mov     dpl,a           ;chuyen sang ngay tiep theo
                cjne    a,#0ah,ngata2

;goi ngay 20 den ngay 29

                mov     dptr,#200h      ;dia chi bat dau

ngatb2:         mov     15h,#4          ;so luong byte cua 1 ngay
                mov     a,15h
                lcall   sendbyte


ngatb1:         lcall   docbyte
                lcall   sendbyte
                inc     dptr
                djnz    1h,ngatb1

                mov     a,dpl           ;chuyen dia chi byte thap vao A
                anl     a,#0            ;xoa 4 bit thap
                swap    a
                inc     a
                swap    a               ;tra lai sau khi tang

                mov     dpl,a           ;chuyen sang ngay tiep theo
                cjne    a,#0ah,ngatb2
;goi ngay 30 den ngay 31

                mov     dptr,#300h      ;dia chi bat dau

ngatc2:         mov     15h,#4          ;so luong byte cua 1 ngay
                mov     a,15h
                lcall   sendbyte


ngatc1:         lcall   docbyte
                lcall   sendbyte
                inc     dptr
                djnz    1h,ngata1

                mov     a,dpl           ;chuyen dia chi byte thap vao A
                anl     a,#0            ;xoa 4 bit thap
                swap    a
                inc     a
                swap    a               ;tra lai sau khi tang

                mov     dpl,a           ;chuyen sang ngay tiep theo
                cjne    a,#02h,ngatc2

;goi byte 00 de bao ket thuc
                mov     a,#0
                lcall   sendbyte
                reti


sendbyte:       jnb     ti,$
                clr     ti
                mov     sbuf,a
```

```
        ret

end
```

BÀI THIẾT KẾ SỐ 2:
"THIẾT KẾ HỆ THỐNG KIỂM TRA IC SỐ".

ĐẶT VẤN ĐỀ

Thiết kế một hệ thống kiểm tra IC số để biết IC đó còn hoạt động được hay không, hoặc có thể nhận dạng ra được loại IC số nếu bị mất số.

GIẢI QUYẾT VẤN ĐỀ

1. PHÂN TÍCH CÁC ĐẶC TÍNH CỦA IC SỐ:

Một trong những đặc tính điện quan trọng nhất của IC số là chỉ có 2 mức logic 0 và 1. Mức logic 0 tương ứng với điện áp 0 volt và mức logic 1 tương ứng với điện áp 5 volt lý tưởng. Với 2 trạng thái làm việc nên ta có thể thực hiện việc kiểm tra tương đối dễ dàng hơn so với IC tương tự.

Các IC số được chia ra làm nhiều loại như sau:

- Các cổng logic như: and, or, not, nand, nor, ex-or, ex-nor, cổng 3 trạng thái.
- Các flip flop: flip JK, flip flop T, Flip flop D.
- Các IC đếm BCD, đếm nhị phân, đếm lên đếm xuống.
- Các thanh ghi dịch: dịch nối tiếp, dịch song song sang nối tiếp.
- Các IC giải mã: m đường sang n đường, giải mã led 7 đoạn.
- Các IC đa hợp, các IC nhớ.

Để kiểm tra một IC số xem chúng có còn tốt hay hỏng và hỏng thành phần nào trong một IC vì một IC có thể chức nhiều thành phần như IC cổng NOT thì có tới 6 cổng NOT.

Đối với từng IC chúng ta phải xác định các thành phần có trong một IC và tiến hành kiểm tra từng thành phần. Để kiểm tra từng thành phần chúng ta phải biết bảng trạng thái hoạt động của chúng ví dụ như:

a. Kiểm tra các cổng logic:

Cổng NOT thì ta cho trạng thái ngõ vào ở mức logic 0 và kiểm tra trạng thái logic của ngõ ra bằng 1 và tiến hành đảo trạng thái ngõ vào là 1 và kiểm tra trạng thái ngõ ra là 0 thì cổng này còn tốt. Còn nếu không đúng trạng thái thì cổng này đã bị hỏng.

Tiến hành kiểm tra các cổng logic con lại theo cách thức tương ứng.

Cổng AND có 2 ngõ vào A, B và một ngõ ra Y thì ta tiến hành kiểm tra theo bảng trạng thái của cổng AND như sau:

Các bước thực hiện	Ngõ vào A	Ngõ vào B	Ngõ ra Y
1	0	0	0
2	0	1	0
3	1	0	0
4	1	1	1

Nếu kết quả kiểm tra đúng như bảng trạng thái trên thì cổng AND này còn tốt và tiến hành kiểm tra các cổng còn lại.

Nếu kết quả không đúng thì cổng đã hỏng.

b. Kiểm tra các IC đếm:

Đối với IC đếm ví dụ như IC7490 thì đó là IC đếm BCD có 2 bộ đếm đọc lập : đếm 2 và đếm 5 thì ta phải tiến hành kiểm tra như sau:

- Kiểm tra trạng thái CLEAR của IC.
- Kiểm tra bộ đếm 2.
- Kiểm tra bộ đếm 5.

c. Kiểm tra các thanh ghi dịch:

Đối với thanh ghi dịch như 74LS164 thì ta phải tiến hành kiểm tra IC theo chức năng dịch chuyển dữ liệu bằng cách tạo ra dữ liệu và tạo xung clock để dịch chuyển dữ liệu. Trình tự kiểm tra như sau:

- Kiểm tra trạng thái clear của IC.
- Kiểm tra trạng thái dịch chuyển dữ liệu mức 1.
- Kiểm tra trạng thái dịch chuyển mức 0.

d. Kiểm tra các IC giải mã:

Đối với IC giải mã thì ta cũng tiến hành tương tự:

- Kiểm tra trạng thái test nếu có của IC.
- Kiểm tra trạng thái giải mã của từng trạng thái ngõ vào.

Nói chung chúng ta phải tìm hiểu hoạt động của từng IC và kiểm tra IC còn tốt hay đã hỏng theo chức năng hoạt động của chúng.

2. THIẾT KẾ SƠ ĐỒ KHỐI VÀ SƠ ĐỒ MẠCH KIỂM TRA IC SỐ:

Phần này sẽ trình bày cách thiết kế mạch điện để kiểm tra IC số – đây chính là vấn đề cốt lõi của yêu cầu thiết kế. Trước khi tiến hành thiết kế chúng ta phải đặt ra các yêu cầu của mạch:

- Mạch kiểm tra IC không được làm hỏng IC trong quá trình kiểm tra.
- Mạch phải kiểm tra được nhiều loại IC trên cùng một đế cài chung.
- Phải đảm bảo kiểm tra chính xác.

Để tiến hành thực hiện việc thiết kế mạch kiểm tra các IC trên cùng một đế cài chung chúng ta phải tìm hiểu đặc tính điện vào ra của từng chân IC.

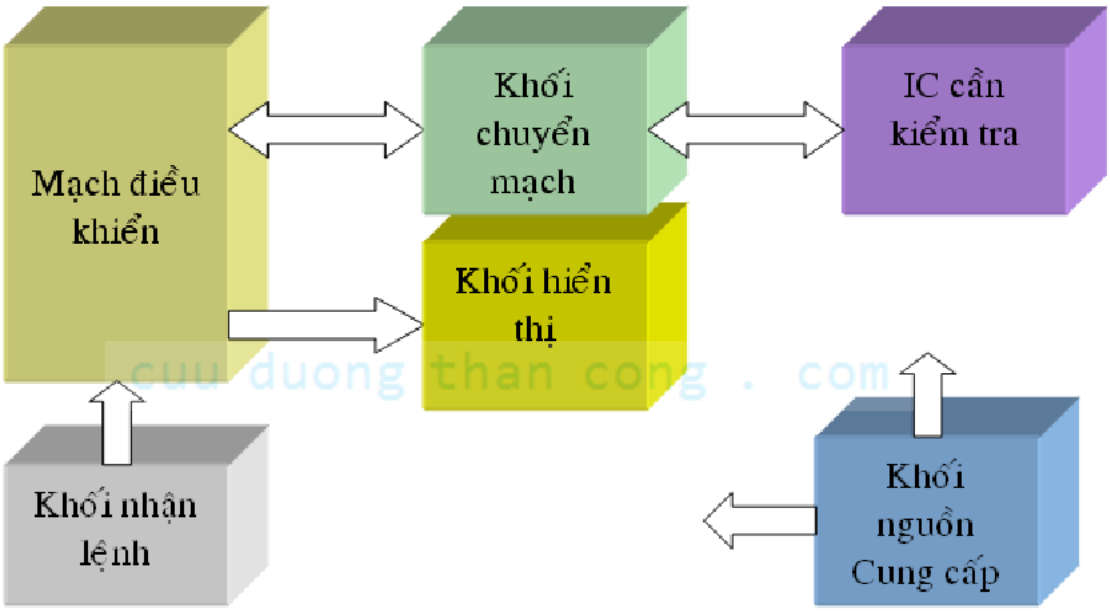
Các IC số được tích hợp theo chức năng và số lượng chân IC tùy thuộc vào từng IC như chúng được phân ra làm nhiều loại như sau:

- IC số 14 chân.
- IC số 16 chân.
- IC số 18 chân.
- IC số 20 chân.
- IC số 24 chân.
- IC số 28 chân.

Một trong các vấn đề khó khăn nhất là các IC với các ngõ vào ra là tuỳ ý: đối với IC này thì 1 số chân là ngõ vào như đối với IC khác thì lại là ngõ ra. Nên mạch điện kiểm tra phải được thiết kế đầy đủ với nhiều chức năng: một chân tín hiệu của đế cài có thể là ngõ xuất tín hiệu, là ngõ nhận tín hiệu, là mass, là nguồn cung cấp 5volt.

Các ngõ tín hiệu phải ở mức logic 0 khi gắn IC vào kiểm tra và lúc lấy IC ra khỏi mạch để đảm bảo không làm hỏng IC.

Từ các yêu cầu thiết kế trên ta tiến hành xây dựng sơ đồ khối của mạch như sau:



Hình 1-3. Sơ đồ khối của hệ thống.

Mạch điều khiển thực hiện quá trình kiểm tra sử dụng vi điều khiển.

Khối chuyển mạch để cung cấp tín hiệu cho IC số sử dụng transistor.

Khối hiển thị cho biết kết quả kiểm tra, vì có thể có nhiều thông tin nên người thiết kế sử dụng LCD để hiển thị kết quả kiểm tra.

Khối nhận lệnh sử dụng bàn phím để nhập vào các thông tin cần thiết cho việc kiểm tra ví dụ như cần kiểm tra IC số cổng nand 74LS00 thì hãy nhập vào các thông tin để hệ thống biết IC đang kiểm tra là cổng NAND. Vì có nhiều IC nên phải xây dựng một bảng mã qui định hay một danh sách lựa chọn.

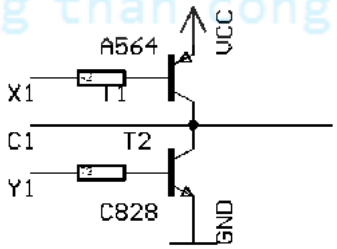
Khối nguồn cung cấp chỉ cần tạo ra nguồn 5 volt để cung cấp năng lược cho toàn mạch hoạt động.

Từ sơ đồ khối ta tiến hành thiết kế sơ đồ nguyên lý cho hệ thống.

Mạch điều khiển sử dụng 89S52 hoặc 89S8252 có bộ nhớ nội 8kbyte để lưu trữ chương trình. Vi điều khiển có 4 port xuất nhập 8 bit tổng cộng 32 đường, trong khi đó mỗi một ngõ tín hiệu có thể kết nối với mass, nối Vcc, hoặc làm đường tín hiệu. Khi đó sơ đồ cho một đường tín hiệu phải sử dụng 3 tín hiệu điều khiển gồm:

- Một đường điều khiển transistor pnp để đóng nguồn 5V cho mạch.
- Một đường điều khiển transistor npn để đóng mass 0V cho mạch.
- Một đường điều khiển nhận tín hiệu hoặc xuất tín hiệu logic.

Sơ đồ mạch của một chân tín hiệu như hình 2.



Hình 1-4. Sơ đồ điều khiển 1 chân của IC.

Từ sơ đồ mạch của một đường tín hiệu thì ta có thể tính được:

- Nếu kết nối để kiểm tra IC số 14 chân thì số đường tín hiệu cần thiết là 42 đường.
- Nếu kết nối để kiểm tra IC số 16 chân thì số đường tín hiệu cần thiết là 48 đường.
- Nếu kết nối để kiểm tra IC số 18 chân thì số đường tín hiệu cần thiết là 54 đường.
- Nếu kết nối để kiểm tra IC số 20 chân thì số đường tín hiệu cần thiết là 60 đường.

Để đơn giản thì ta giới hạn lại yêu cầu và chỉ kiểm tra IC số cho 2 loại 14 chân đến 16 chân và số đường tín hiệu cần thiết tối đa là 48 đường. Để thêm các đường tín hiệu điều khiển ta có thể thực hiện bằng nhiều cách và trong đề tài này ta chọn cách mở rộng là sử dụng IC chốt 74573. Việc kết nối này sẽ làm mất hết 12 đường điều khiển nhưng ta có thêm được 32 ngõ ra.

Số lượng đường điều khiển còn lại của vi điều khiển là 20 đường ta sẽ sử dụng 16 đường để làm các đường điều khiển IO, như vậy còn lại 4 đường không đủ để kết nối với bàn phím và LCD nên người nghiên cứu sử dụng thêm 1 vi điều khiển thứ 2 có chức năng kết nối với LCD và bàn phím và kết nối với vi điều khiển thứ nhất qua truyền dữ liệu nối tiếp.

Vi điều khiển thứ 2 có nhiệm vụ nhận tín hiệu điều khiển từ bàn phím và ra lệnh cho vi điều khiển thứ nhất tiến hành kiểm tra và sau khi kiểm tra xong thì phải gởi lại kết qủa kiểm tra cho vi điều khiển thứ 2 để hiển thị trạng thái kiểm tra trên màn hình.

Sơ đồ nguyên lý mạch như hình 3a, 3b, 3c.

Trình tự thực hiện kiểm tra IC do người nghiên cứu đề ra như sau:

Bước 1: gắn IC vào socket cho đúng chiều qui định.

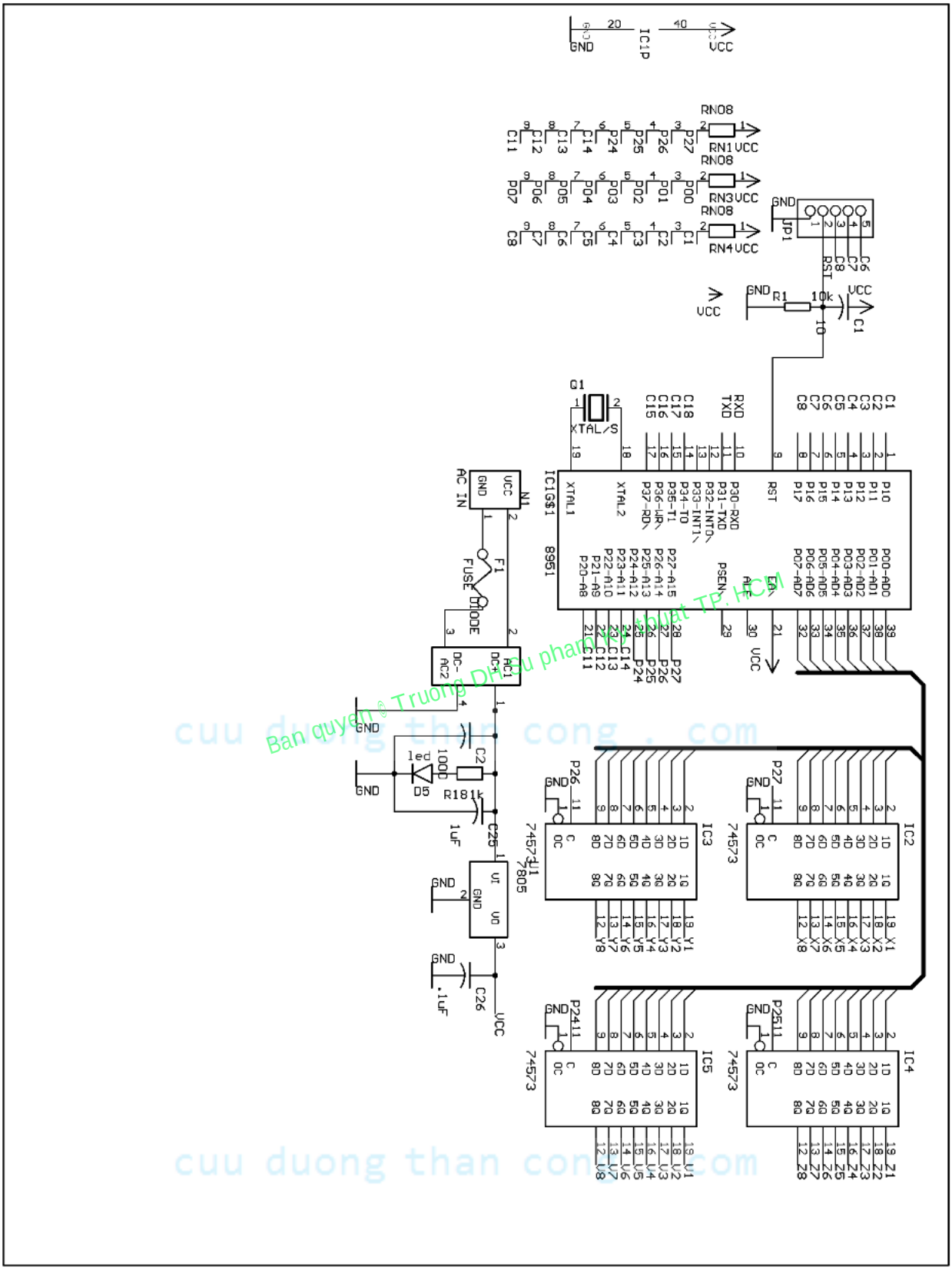
Bước 2: tiến hành nhập mã của IC ví dụ như kiểm tra IC 7400 ta tiến hành nhập các số 7400 rồi nhấn phím test.

Bước 3: vi điều khiển 2 sẽ truyền yêu cầu thông tin đến vi điều khiển 1 và chờ nhận tín hiệu trả lời.

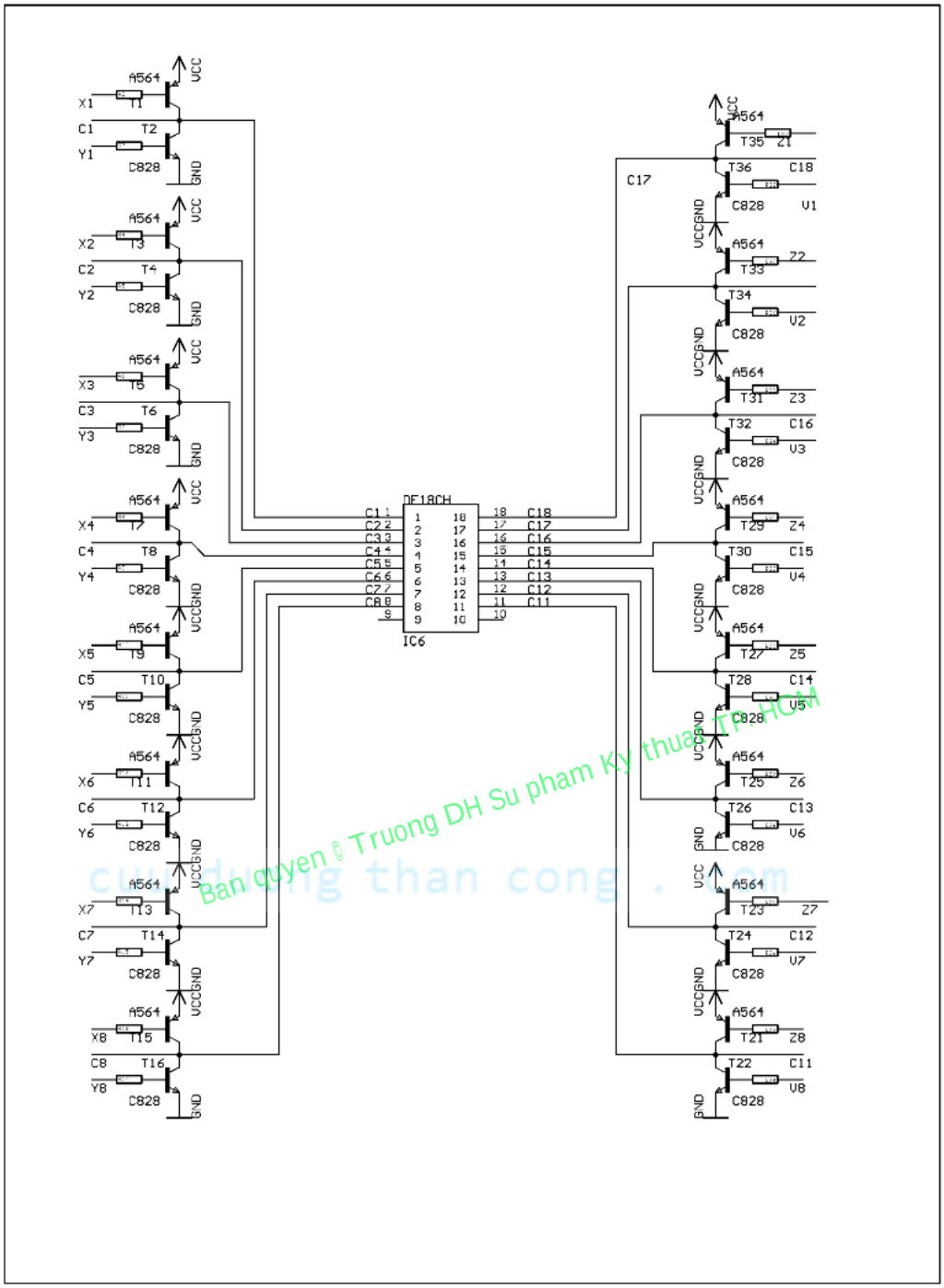
Bước 5: vi điều khiển 1 tiến hành kiểm tra và gởi kết quả kiểm tra trở lại vi điều khiển 2.

Bước 6: vi điều khiển 2 nhận thông tin kết qủa trả lời và hiển thị trên LCD.

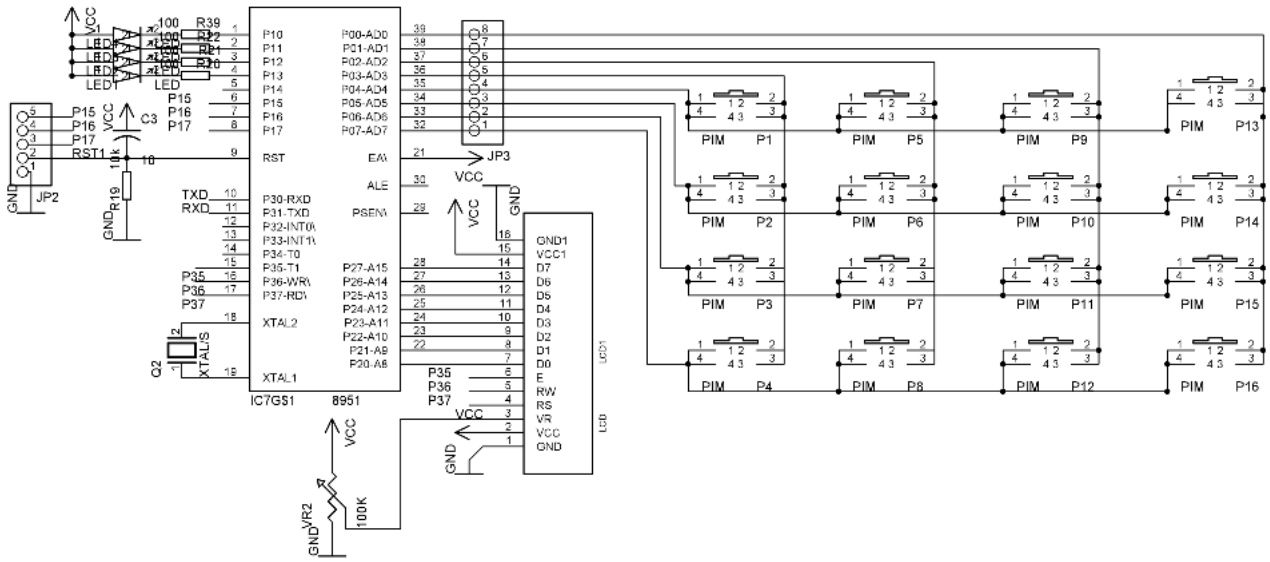
Bước 7: quá trình kiểm tra kết thúc.



Hình 1-5a. Sơ đồ kết nối vi điều khiển 1 với IC chốt.



Hình 1-5b. Sơ đồ kết nối các đường tín hiệu điều khiển với socket 18 chân .



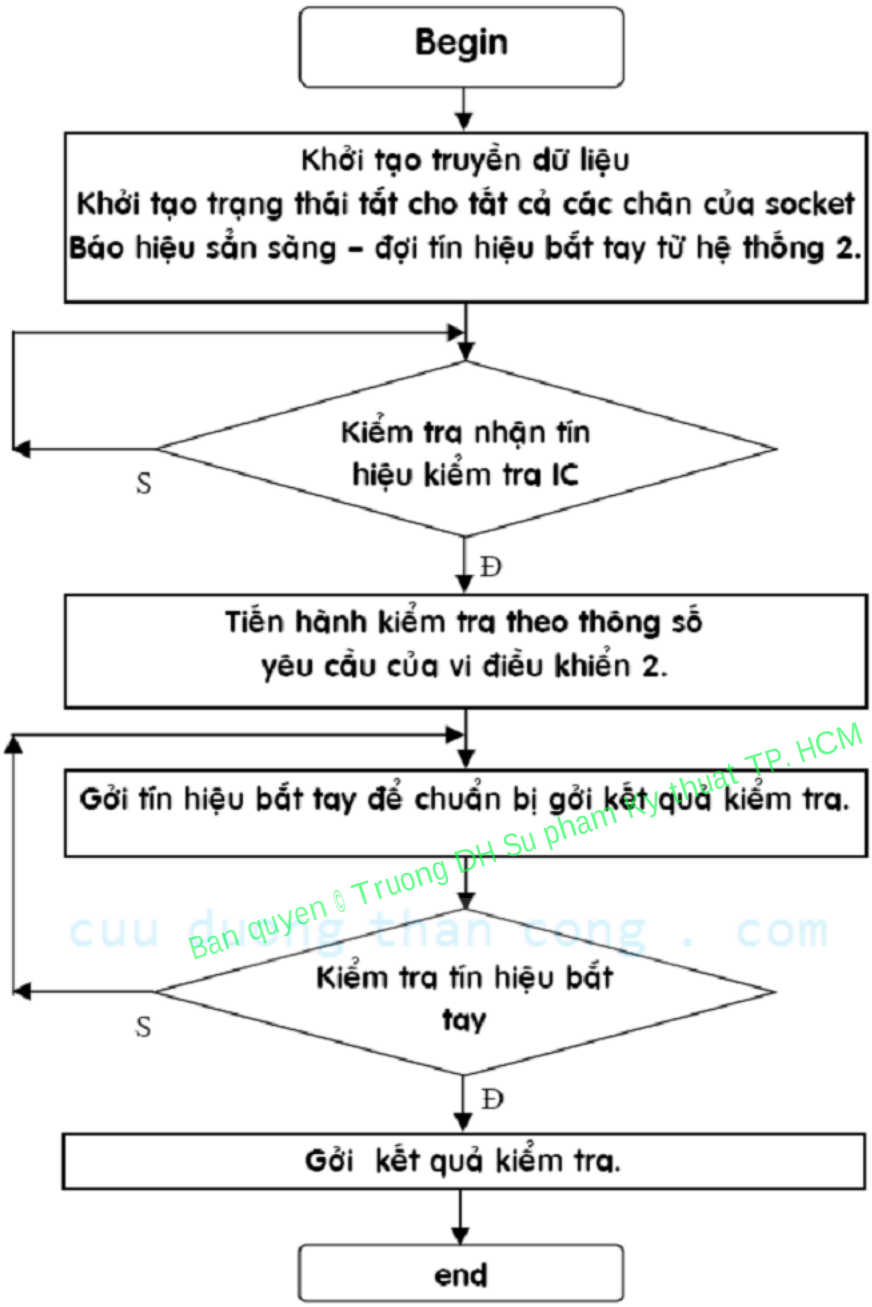
Hình 1-5c. Sơ đồ kết nối vi điều khiển 2 với LCD và bàn phím ma trận.

3. THIẾT KẾ PHẦN MỀM:

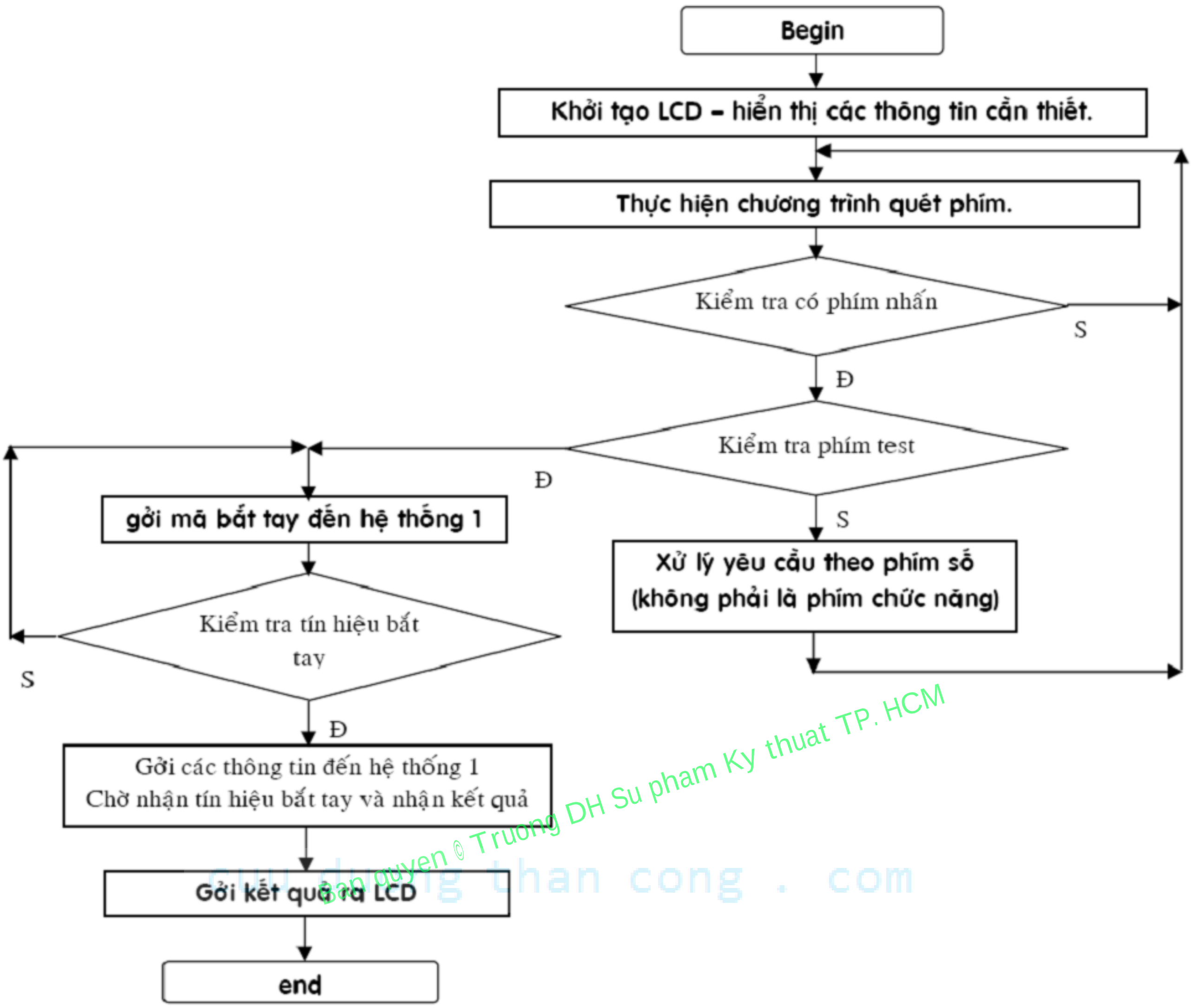
Sau khi đã thiết kế xong các hệ thống điều khiển và hệ thống mở rộng ta tiến hành viết các chương trình cho các hệ thống.

Với trình tự thực hiện ở trên ta tiến hành viết lưu đồ cho 2 hệ thống vi điều khiển 1 và 2.

Lưu đồ và chương trình cho vi điều khiển 1:



Lưu đồ và chương trình cho vi điều khiển 2:



Dựa vào lưu đồ điều khiển ta tiến hành viết chương trình cho từng hệ thống.

```
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;chuong trinh test IC so
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;chuong trinh viet cho vi dieu khien he thong 2:
                enter   equ     0ah     ;dinh nghia ma phim enter
                ptang   equ     0bh     ;dinh nghia ma phim tang
                pgiam   equ     0ch     ;dinh nghia ma phim giam
                pclear  equ     0dh     ;dinh nghia ma phim xoa

                pkey    equ     p0      ;ket noi voi ma tran phim nhan
;ma bat tay goi di la AAH, ma bat tay nhan ve la BBh
;ma bat tay goi di la C0H la bao IC tot
;ma qui dinh goi di la d0H la bao IC xau
;ma qui dinh goi di la CFH la bao he  thong 1 chua xay dung phan kiem tra IC
                pcon    equ     87h
;cac vung nho se su dung de luu thong tin

;               20h,21h,22h de luu ma cua IC = 740xxx
;               23h luu ma so cao de truyen di

;               30h ->  3fh de luu thong tin cho LCD hang thu 1
;               40h ->  4fh de luu thong tin cho LCD hang thu 2
                org 0000h
```

```
                mov     sp,#50h

                mov     22h,#00h        ;mac nhien la khi khoi dong
                mov     21h,#00h        ;mac nhien la khi khoi dong
                mov     20h,#074h       ;mac nhien la khi khoi dong la 74

                lcall   khoitao_trxd    ;goi chtr con khoi tao truyen du lieu giua 2 vdk
                lcall   khoitao_lcd     ;goi chtr khoi tao LCD
                lcall   hienthi_lcd     ;goi chtr con hien thi thong tin ra LCD

main1a:         lcall   quetphim        ;goi chtr con quet phim
                cjne    a,#0ffh,main3   ;co phim nhan thi nhay
                sjmp    main1a          ;neu khong co phim nhan thi tiep tuc
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

main3:          lcall   hienthi_lcd_key ;goi chtr con hien thi nhan ma so IC

main1:          lcall   quetphim        ;goi chtr con quet phim
                cjne    a,#0ffh,main3a  ;co phim nhan thi nhay
                sjmp    main1           ;neu khong co phim nhan thi tiep tuc

main3a:         cjne    a,#10,main4     ;kiem tra phim so
main4:          jnc     main5           ;cac phim chuc nagn con lai
                ljmp    xuly_phimso     ;nhay den chtr xu ly phim so

main5:          cjne    a,#enter,main2  ;kiem tra phim enter
                lcall   hienthi_test
                ljmp    xuly_enter      ;nhay den chtr xu ly phim enter

main2:          cjne    a,#pclear,main3 ;kiem tra phim clear
                ljmp    xuly_clear      ;nhay den chtr xu ly phim clear

                ljmp    main1           ;nhay den chtr xu ly

;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;xu ly cac phim so luu tru vao 2 o nho 21h (byte H) va 22h (byte L)
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

xuly_phimso:    push    acc             ;cat tam noi dung A

                mov     a,22h
                swap    a               ;
                mov     22h,a           ;ket qua tu ZVH thanh VZH

                clr     a               ;
                mov     r0,#22h
                xchd    a,@r0           ;ket qua (A) = YZH
                mov     21h,a           ;ket qua (A) = YZH vao o nho 21h

                pop     acc             ;lay lai A
                orl     22h,a           ;ket qua (22H)=VW: bon 4 thap moi vua vao

xuly_pso1:      lcall   giaima          ;goi chtr con giai ma
                lcall   hienthichung
                ljmp    main1           ;tro ve chtr chinh

;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;chtr con giai ma
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
giaima:         mov     a,21h
                anl     a,#0fh
                add     a,#30h
                mov     4dh,a

                mov     a,22h
```

```
                anl     a,#0f0h
                swap    a
                add     a,#30h
                mov     4eh,a

                mov     a,22h
                anl     a,#0fh
                add     a,#30h
                mov     4fh,a
                ret

;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;xu ly phim clear bang cach xoa 2 o nho 21h (byte H) va 22h (byte L) ve 00
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xuly_clear:     mov     21h,#00h        ;xoa cac thong tin ve 00
                mov     22h,#00h
                ljmp    xuly_pso1

;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;xu ly phim enter bang cach goi noi dung 2 o nho 21h (byte H) va
;22h (byte L) sang vdk 1
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xuly_enter:     mov     a,21h           ;chuyen byte
                mov     dptr,#1000h     ;nap dia chi 1000h
                movc    a,@a+dptr       ;
                cjne    a,#0ffh,xuly_en1

                lcall   hienthi_tbao1   ;hien thi thong bao chua cai dat IC
                ljmp    main1a          ;nhay ve chuong trinh chinh

xuly_en1:       lcall   truyenma        ;goi chtr truyen ma kiem tra ic di
                ljmp    main1a          ;tro ve sau khi xu ly xong

;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;chtr con khoi tao truyen du lieu giua 2 vdk A va B
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
khoitao_trxd:   mov     th1,#0fah
                mov     tl1,#0fah
                anl     tmod,#0fh
                orl     tmod,#20h
                setb    tr1
                mov     scon,#50h
                setb    ti
                orl     pcon,#80h
                ret
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;goi chtr truyen ma kiem tra ic di: goi ma bat tay la AAH, ma nhan ve la BB
; goi ma IC, cho nhan tin hieu tra loi- hien thi ket qua va thoat
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
truyenma:       jnb     ti,$            ;kiem tra co truyen
                clr     ti
                mov     sbuf,#0aah      ;goi ma bat tay di

                mov     7eh,#0          ;kiem tra co truyen co delay de
truyenma3:      mov     7fh,#0          ;thoat khi he thong khong bat tay
truyenma2:      jb      ri,truyenma1
                djnz    7fh,truyenma2
                djnz    7eh,truyenma3
                lcall   hienthi_tbao2   ;goi thong bao chua bat tay ht 1
                ret

truyenma1:      clr     ri
                mov     a,sbuf          ;nhan byte bat tay
                cjne    a,#0bbh,truyenma4
                sjmp    truyenma5
```

```
truyenma4:      lcall   hienthi_tbao3      ;goi thg bao bat tay khong dung
                ret
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;goi ma test cua IC den ht 1
truyenma5:      lcall   hienthi_tbao4      ;goi thong bao bat tay tot

                jnb     ti,$
                clr     ti
                mov     a,21h
                mov     sbuf,a             ;truyen byte ma cao

                jnb     ti,$
                clr     ti
                mov     a,22h              ;truyen byte ma thap
                mov     sbuf,a
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;cho nhan ket qua
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
                mov     7dh,#0             ;kiem tra co truyen co delay de
truyenmaa:      mov     7eh,#0             ;kiem tra co truyen co delay de
truyenma6:      mov     7fh,#0             ;thoat khi he thong khong bat tay
truyenma7:      jb      ri,truyenma8
                djnz    7fh,truyenma7
                djnz    7eh,truyenma6
                djnz    7dh,truyenmaa

                lcall   hienthi_tbao5      ;goi thong bao khong co ket qua hoi am
                ret

;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;cho nhan ket qua
;ma bat tay goi di la C0H la bao IC tot
;ma bat tay goi di la d0H la bao IC xau
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

truyenma8:      clr     ri
                mov     a,sbuf
                cjne    a,#0c0h,truyenma9
                lcall   hienthi_tbao6      ;goi thong bao IC tot
                ret

truyenma9:      cjne    a,#0d0h,truyenma10
                lcall   hienthi_tbao7      ;goi thong bao IC hong
                ret

truyenma10:     cjne    a,#0cfh,truyenma11
truyenma11:     lcall   hienthi_tbao1      ;hien thi thong bao chua cai dat IC ben ht1
                ret

;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;chtr con kiem tra phim nhan: p0 ket noi voi ma tran 16 phim
;10 phim so co ma tu 00 den 09 va sau phim chuc nang co ma tu 0ah den 0fh:
;co phim nhan thi ma dung voi qui dinh - neu khong co thi A = FFH
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
quetphim:
keypres: mov    r3,#10             ;nhap so dem 10 lan
keypres1:       lcall   KEY                ;Neu co phim an thi co c=1
                jc      pn1                ;kiem tra tiep neu c = 1
                ret                        ;Neu khong co phim nhan thi co c=0

pn1:            djnz    r3,keypres1        ;Quay ve lap lai chong nay
                push    acc                ;Cat noi dung ma phim trong A
```

```
keypres2:       mov     r3,#50          ;Nhap so dem 10 lan cho nha phim
keypres3:       lcall   key             ;Co phim nhan hay khong
                jc      keypres2        ;Co thi kiem tra lai
                djnz    r3,keypres3     ;Khong thi lap lai 50 lan va dam bao
                pop     acc             ;Khoi phuc lai gia tri cho A
                ret                     ;ket thuc mot chuong trinh con
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;Chuong trinh con quet phim
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
key:            mov     r7,#0feh        ;bat dau voi cot so 0(feh)
                mov     r6,#4           ;Su dung r6 lam bo dem
                mov     r5,#00

key1:           mov     pkey,r7         ;xuat ma quet ra cot
                mov     a,pkey          ;Doc lai port1 de xu ly tiep theo
                anl     a,#0f0h         ;xoa 4 bit thap la hang
                cjne    a,#0f0h,key2    ;co nhan fim thi nhay

                mov     a,r7
                rl      a               ;xoay de chuyen den cot ke tiep
                mov     r7,a

                mov     a,r5            ;chuyen ma fim sang cot ke
                add     a,#4
                mov     r5,a

                djnz    r6,key1         ;Neu nhu sau moi lan 1 cot ma khong

                clr     c               ;clr c neu nhu khong co phim duoc an
                mov     a,#0ffh         ;thoat voi ma trong a = FFh
                ret

key2:           swap    a
key4:           rrc     a               ;xoay sang phai tim bit 0
                jnc     key3            ;nhay neu (c)=0
                inc     r5              ;tang ma fim len cot ke
                sjmp    key4            ;tiep tuc cho den khi duoc (C)=0

key3:           mov     a,r5
                setb    c
                ret


;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;goi chtr con hien thi nhap ma so IC
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
hienthi_lcd_key: mov    dptr,#tbaoA1_DATA       ;nap dia chi bat dau
                lcall   copydata                ;goi ch tr con copy 32 byte
                lcall   hienthichung
                ret
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;goi chtr con hien thi dang test so IC
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
hienthi_test:   mov     dptr,#tbaob1_DATA       ;nap dia chi bat dau
                lcall   copydata                ;goi ch tr con copy 32 byte
                lcall   hienthichung
                ret

;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;chtr con hien thi thong tin ra LCD MAC DINH KHI KHOI DONG
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
hienthi_lcd:    mov     dptr,#tbao01_DATA       ;nap dia chi bat dau
                lcall   copydata                ;goi ch tr con copy 32 byte
                lcall   hienthichung
                ret
```

```
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;hien thi thong bao chua cai dat IC
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
hienthi_tbao1:  mov     dptr,#tbao11_DATA       ;nap dia chi bat dau
                lcall   copydata                ;goi ch tr con copy 32 byte
                lcall   hienthichung
                ret

;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;hien thi thong bao khong bat tay he thong 1
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
hienthi_tbao2:  mov     dptr,#tbao21_DATA       ;nap dia chi bat dau
                lcall   copydata                ;goi ch tr con copy 32 byte
                lcall   hienthichung
                ret

;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;goi thg bao bat tay khong dung
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
hienthi_tbao3:  mov     dptr,#tbao31_DATA       ;nap dia chi bat dau
                lcall   copydata                ;goi ch tr con copy 32 byte
                lcall   hienthichung
                ret


;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;goi thong bao bat tay tot
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
hienthi_tbao4:  mov     dptr,#tbao41_DATA       ;nap dia chi bat dau
                lcall   copydata                ;goi ch tr con copy 32 byte
                lcall   hienthichung
                ret

;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;goi thong bao khong co ket qua hoi am
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
hienthi_tbao5:  mov     dptr,#tbao51_DATA       ;nap dia chi bat dau
                lcall   copydata                ;goi ch tr con copy 32 byte
                lcall   hienthichung
                ret

;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;goi thong bao IC tot
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
hienthi_tbao6:  mov     dptr,#tbao61_DATA       ;nap dia chi bat dau
                lcall   copydata                ;goi ch tr con copy 32 byte
                lcall   hienthichung
                ret

;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;goi thong bao IC hong
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
hienthi_tbao7:  mov     dptr,#tbao71_DATA       ;nap dia chi bat dau
                lcall   copydata                ;goi ch tr con copy 32 byte
                lcall   hienthichung
                ret

;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;lay du lieu tu bo nho chuong trinh vao bo nho ram
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

copydata:       mov     r0,#30h
                mov     r1,#0
```

```
copydata1:      mov     a,r1
                movc    a,@a + dptr             ;lay data
                mov     @r0,a                   ;cat data
                inc     r1
                inc     r0
                cjne    r1,#33,copydata1
                ret

;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;Chuong trinh con hien thi noi dung  tren LCD cua2 vung nho
;30H->3Fh hang 1; 40H-> 4Fh hang 2;
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
hienthichung:   MOV     A,#080h         ;set DDRAM
                LCALL   KTAO
                mov     r1,#16
                MOV     R0,#30H

fline:          lcall   Write
                djnz    r1,fline

                mov     a,#0c0h         ;set DDRAM
                LCALL   KTAO

                mov     r1,#16
                MOV     R0,#40H

sline:          lcall   Write
                djnz    r1,sline

                ret
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;chtr con goi data hien thi ra LCD
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
write:          MOV     byteout,@R0
                Lcall   data_byte
                inc     r0
                ret


;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;chtr con khoi tao LCD

;chuong trinh dieu khien LCD 16X2 tren kit vi dieu khien LOAI NHO
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
                E               BIT     P3.5
                rw              BIT     P3.6
                rs              BIT     P3.7
                byteout         equ     p2


khoitao_lcd:    mov     0a2h,#0
                LCALL   khtaolcd ;khoi tao lcd
                ret

;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;chtr con khoi tao LCD
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
khtaolcd:       setb    e               ;Enable
                clr     rs              ;RS low
                clr     rw              ;RW low

                MOV     a,#38h          ;tu dieu khien LCD
                LCALL   KTAO
                LCALL   ddelay41        ;delay 4.1 mSec
```

```
                MOV   A,#38h            ;function set
                LCALL KTAO
                LCALL ddelay100         ;delay

                MOV   A,#38h            ;function
                LCALL KTAO

                MOV   A,#0ch            ;tu dieu khien display on
                LCALL KTAO
                MOV   A,#01h            ;tu dieu khien Clear display
                LCALL KTAO

                MOV   A,#06h            ;tu dieu khien entry mode set
                LCALL KTAO

                MOV   A,#80h            ;thiet lap dia chi LCD (set DD RAM)
                LCALL KTAO
                RET

;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;chuong trinh con khoi tao LCD
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
KTAO:           mov     byteout,a
                lcall   command_byte
                RET

;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;Feed command/data to the LCD module
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
command_byte:
                clr   rs                ;RS low for a command byte
                ljmp  bdelay

data_byte:      setb  rs                ;RS high for a data byte
bdelay:         clr   rw                ;R/W low for a write mode
                clr   e
                nop

                setb  e                 ;Enable pulse
                nop
                nop

                mov   byteout,#0ffh     ;configure port1 to input mode

                setb  rw                ;set RW to read
                clr   rs                ;set RS to command
                clr   e                 ;generate enable pulse

                nop
                nop
                setb  e
                lcall   ddelay100
                ret

;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;chuong trinh con delay 4.1 ms
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
ddelay41:       mov     7eh,#90h
del412:         mov     7fh,#200
                djnz    7fh,$
                djnz    7eh,del412
                ret

;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;chuong trinh con delay 255 microgiay
```

```
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
ddelay100:      mov     7fh,#00
                djnz    7fh,$
                ret

;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;       Data bytes
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
FLINE_DATA:   DB      'NGUYEN DINH PHU ';,099h
SLINE_DATA:   DB      'DAI HOC SPKT HCM';,099h

tbaoa1_DATA:  DB      'HAY NHAP MA 3 SO';,099h
tbaoa2_DATA:  DB      'CUOI      74XXX';,099h

tbaob1_DATA:  DB      'TESTING IC - YOU';,099h
tbaob2_DATA:  DB      'GIVE ME CIGARETT';,099h

tbao01_DATA:  DB      'BO TEST IC 74XXX';,099h
tbao02_DATA:  DB      'DESIGN BY MR PHU';,099h


tbao11_DATA:  DB      'IC NAY CHUA CAI ';,099h
tbao12_DATA:  DB      'TRONG HE THONG  ';,099h

tbao21_DATA:  DB      'HE THONG 2 KHONG';,099h
tbao22_DATA:  DB      'BAT TAY HETHONG1';,099h

tbao31_DATA:  DB      'MA BAT TAY TRA  ';,099h
tbao32_DATA:  DB      'KHONG DUNG -*** ';,099h

tbao41_DATA:  DB      'HE THONG BAT TAY';,099h
tbao42_DATA:  DB      'TOT - TIEP TUC  ';,099h

tbao51_DATA:  DB      'KHONG CO KET QUA';,099h
tbao52_DATA:  DB      'TRA LOI -HT2 LOI';,099h

tbao61_DATA:  DB      'IC NAY CON TOT  ';,099h
tbao62_DATA:  DB      'GOODBYE GOODLUCK';,099h

tbao71_DATA:  DB      'IC NAY DA HONG  ';,099h
tbao72_DATA:  DB      'MONEY-MONEY -BAD';,099h

;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;vung nho luu tru 2 so hang tram va hang ngan cua IC so
;khi them IC vao thi vung nho nay phai cap nhat
;hien tai chi xu ly cac IC so ho 7400 den 74199
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
                org 1000h
                db      00h,01h


end
```


BÀI TẬP:

Bài Số 1: Hãy đọc datasheet của bộ nhớ họ 27Cxx và thiết kế mạch nạp chương trình cho bộ loại 27C64 và sau đó mở rộng thiết kế cho cả họ từ 27C16 đến họ 27C512.

Bài Số 2: Hãy đọc datasheet của bộ nhớ họ 27Cxx và thiết kế mạch nạp chương trình cho bộ loại 27C64 và sau đó mở rộng thiết kế cho cả họ từ 27C16 đến họ 27C512.

Bài Số 3: Hãy đọc datasheet của vi điều khiển 89C51 và thiết kế mạch nạp chương trình cho vi điều khiển.

Bài Số 4: Hãy đọc datasheet của vi điều khiển 89S51 và thiết kế mạch nạp chương trình cho vi điều khiển dạng nối tiếp.

the end

return

Chương 2

VI ĐIỀU KHIỂN PIC 16F877A

Bảng và hình:

I. TỔNG QUAN VỀ VI ĐIỀU KHIỂN PIC

PIC là một họ vi điều khiển RISC được sản xuất bởi công ty Microchip Technology. Thế hệ PIC đầu tiên là PIC1650 được phát triển bởi Microelectronics Division thuộc General – Instrument.

PIC là viết tắt của "Programmable Intelligent Computer" là một sản phẩm của hãng General Instruments đặt cho dòng sản phẩm đầu tiên là PIC1650. Tại thời điểm đó PIC1650 được dùng để giao tiếp với các thiết bị ngoại vi cho máy chủ 16 bit CP1600, vì vậy, người ta cũng gọi PIC với cái tên "Peripheral Interface Controller" – bộ điều khiển giao tiếp ngoại vi.

CP1600 là một CPU mạnh nhưng lại yếu về các hoạt động xuất nhập vì vậy PIC 8-bit được phát triển vào khoảng năm 1975 để hỗ trợ cho hoạt động xuất nhập của CP1600.

PIC ROM để chứa mã, mặc dù khái niệm RISC chưa được sử dụng thời bấy giờ, nhưng PIC thực sự là một vi điều khiển với kiến trúc RISC, chạy một lệnh với một chu kỳ máy – gồm 4 chu kỳ của bộ dao động.

Năm 1985 General Instruments bán công nghệ các vi điện tử của họ, và chủ sở hữu mới hủy bỏ hầu hết các dự án - lúc đó đã quá lỗi thời. Tuy nhiên PIC được bổ sung EEPROM để tạo thành 1 bộ điều khiển vào ra lập trình.

Ngày nay rất nhiều dòng PIC được xuất xưởng với hàng loạt các module ngoại vi tích hợp sẵn (như USART, PWM, ADC...), với bộ nhớ chương trình từ 512 Word đến 32K Word.

II. MỘT SỐ ĐẶC TÍNH CỦA VI ĐIỀU KHIỂN PIC

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

- Là CPU 8/16 bit, xây dựng theo kiến trúc Harvard có sửa đổi.
- Có bộ nhớ Flash và ROM có thể tuỳ chọn từ 256 byte đến 256 Kbyte .
- Có các cổng xuất – nhập (I/O ports).
- Có timer 8/16 bit.
- Có các chuẩn giao tiếp nối tiếp đồng bộ/không đồng bộ USART.
- Có các bộ chuyển đổi ADC 10/12 bit.
- Có các bộ so sánh điện áp (Voltage Comparators).
- Có các khối Capture/Compare/PWM.
- Có hỗ trợ giao tiếp LCD.
- Có MSSP Peripheral dùng cho các giao tiếp I²C, SPI, và I²S.
- Có bộ nhớ nội EEPROM - có thể ghi/xoá lên tới 1 triệu lần.
- Có khối Điều khiển động cơ, đọc encoder.
- Có hỗ trợ giao tiếp USB.
- Có hỗ trợ điều khiển Ethernet.
- Có hỗ trợ giao tiếp CAN.

Đặc điểm thực thi tốc độ cao CPU RISC của họ vi điều khiển PIC16F87XA là:

- Chỉ gồm 35 lệnh đơn.
- Tất cả các lệnh là 1 chu kì ngoại trừ chương trình con là 2 chu kì.
- Tốc độ hoạt động:

  * DC- 20MHz ngõ vào xung clock.
  * DC- 200ns chu kì lệnh.
- Dung lượng của bộ nhớ chương trình Flash là 8K×14words.
- Dung lượng của bộ nhớ dữ liệu RAM là 368×8bytes.
- Dung lượng của bộ nhớ dữ liệu EEPROM là 256×8 bytes.

a. Các đặc tính ngoại vi

- Timer0: là bộ định thời timer/counter 8 bit có bộ chia trước.
- Timer1: là bộ định thời timer/counter 16 bit có bộ chia trước, có thể đếm khi CPU đang ở trong chế độ ngủ với nguồn xung từ tụ thạch anh hoặc nguồn xung bên ngoài.
- Timer2: bộ định thời timer/counter 8 bit với thanh ghi 8-bit, chia trước và postscaler.
- Hai khối Capture, Compare, PWM.
  - Capture có độ rộng 16-bit, độ phân giải 12.5ns
  - Compare có độ rộng 16-bit, độ phân giải 200ns
  - Độ phân giải lớn nhất của PWM là 10-bit.

b. Các đặc tính về tương tự

- Có 8 kênh chuyển đổi tín hiệu tương tự thành tín hiệu số ADC 10-bit.
- Có reset BOR (Brown- Out Reset).
- Khối so sánh điện áp tương tự:
  - Hai bộ so sánh tương tự.
  - Khối tạo điện áp chuẩn V_{REF} tích hợp bên trong có thể lập trình.
  - Đa hợp ngõ vào lập trình từ ngõ vào của CPU với điện áp chuẩn bên trong.
  - Các ngõ ra của bộ so sánh có thể truy xuất từ bên ngoài.

c. Các đặc tính đặc biệt của vi điều khiển:

- Bộ nhớ chương trình Enhanced Flash cho phép xóa và ghi 100000 lần.
- Bộ nhớ dữ liệu EEPROM cho phép xóa và ghi 1000000 lần.
- Bộ nhớ EEPROM có thể lưu giữ dữ liệu hơn 40 năm và có thể tự lập trình lại dưới sự điều khiển của phần mềm.

- Mạch lập trình nối tiếp ICSP thông qua 2 chân (In-Circuit Serial Programming).
- Nguồn sử dụng là nguồn đơn 5V cấp cho mạch lập trình nối tiếp.
- Có Watchdog Timer (WDT) với bộ dao động RC tích hợp sẵn trên Chip.
- Có thể lập trình mã bảo mật.
- Có thể hoạt động ở chế độ Sleep để tiết kiệm năng lượng.
- Có thể lựa chọn bộ dao động.
- Có mạch điện gỡ rối ICD (In-Circuit Debug) thông qua 2 chân.

d. Công nghệ CMOS:

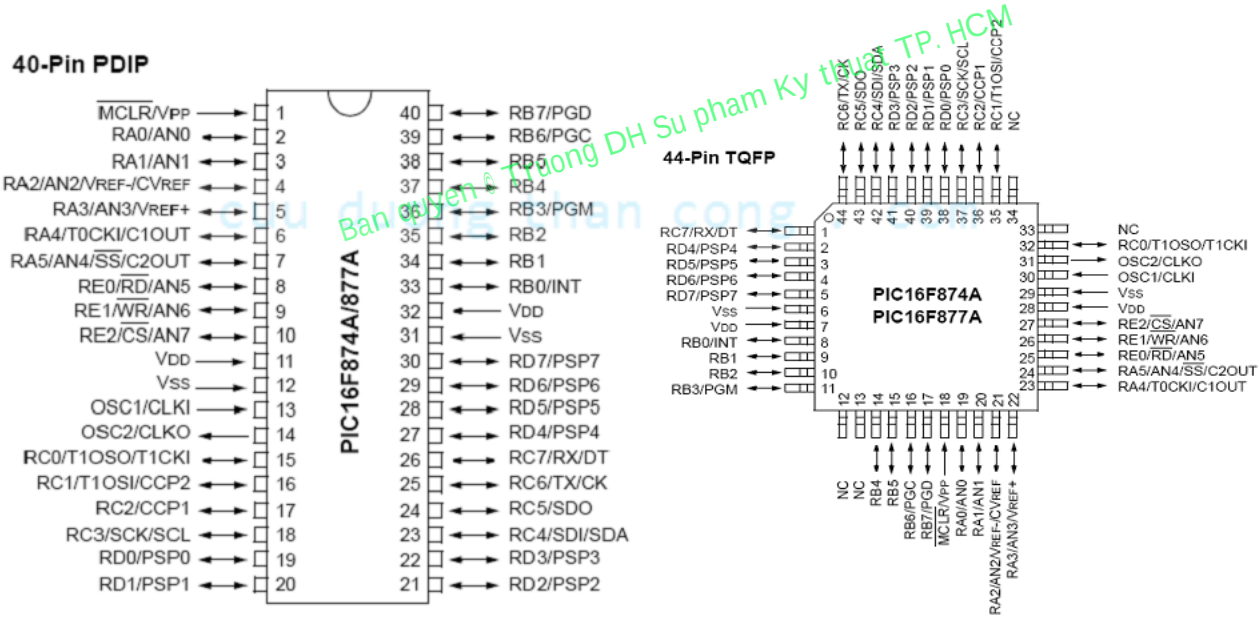
Công nghệ CMOS có các đặc tính: công suất thấp, công nghệ bộ nhớ Flash/EEPROM tốc độ cao. Điện áp hoạt động từ 2V đến 5,5V và tiêu tốn năng lượng thấp. Phù hợp với nhiệt độ làm việc trong công nghiệp và trong thương mại.

Bảng 2-1 trình bày tóm tắt cấu trúc của 4 loại PIC16F87X.

Device	Program Memory		Data SRAM (Bytes)	EEPROM (Bytes)	I/O	10-bit A/D (ch)	CCP (PWM)	MSSP		USART	Timers 8/16-bit	Comparators
	Bytes	# Single Word Instructions						SPI	Master I^2C			
PIC16F873A	7.2K	4096	192	128	22	5	2	Yes	Yes	Yes	2/1	2
PIC16F874A	7.2K	4096	192	128	33	8	2	Yes	Yes	Yes	2/1	2
PIC16F876A	14.3K	8192	368	256	22	5	2	Yes	Yes	Yes	2/1	2
PIC16F877A	14.3K	8192	368	256	33	8	2	Yes	Yes	Yes	2/1	2

Bảng 2-1. Các vi điều khiển họ PIC16F87X.

Hình 2-1 trình bày sơ đồ chân của các loại PIC16F87XA.



Hình 2-1. Sơ đồ chân họ PIC16F87XA.

III. VI ĐIỀU KHIỂN PIC 16F877A

1. TỔNG QUÁT VỀ PIC16F877A:

a. Giới thiệu:

PIC16F877A có 40/44 chân với cấu trúc như sau:

- Có 5 port xuất/ nhập.
- Có 8 kênh chuyển đổi A/D.
- Được bổ sung các port tử song song.
- Có bộ nhớ gap nối so với PIC16F873A/PIC16F874A.

Bảng 2-2 sẽ tóm tắt đặc điểm PIC16F877A:

Đặc điểm	PIC16F877A
Tan số hoạt động	DC- 20MHz
Reset (và Delay)	POR, BOR (PWRT, OST)
Bộ nhớ chương trình Flash (14-bit word)	8K
Bộ nhớ dữ liệu (byte)	368
Bộ nhớ dữ liệu EEPROM (byte)	256
Các nguồn ngắt	15
Các port xuất nhập	Các port A, B, C, D, E
Timer	3
Các module capture/compare/PWM	2
Giao tiếp nối tiếp	MSSP, USART
Giao tiếp song song	PSP
Module A/D 10bit	8 kênh ngõ vào
Bộ so sánh tương tự	2
Tập lệnh	35 lệnh
Số chân	40 chân PDIP 44 chân PLCC 44 chân TQFP 44 chân QFN

Bảng 2-2. Tóm tắt đặc điểm PIC16F877A

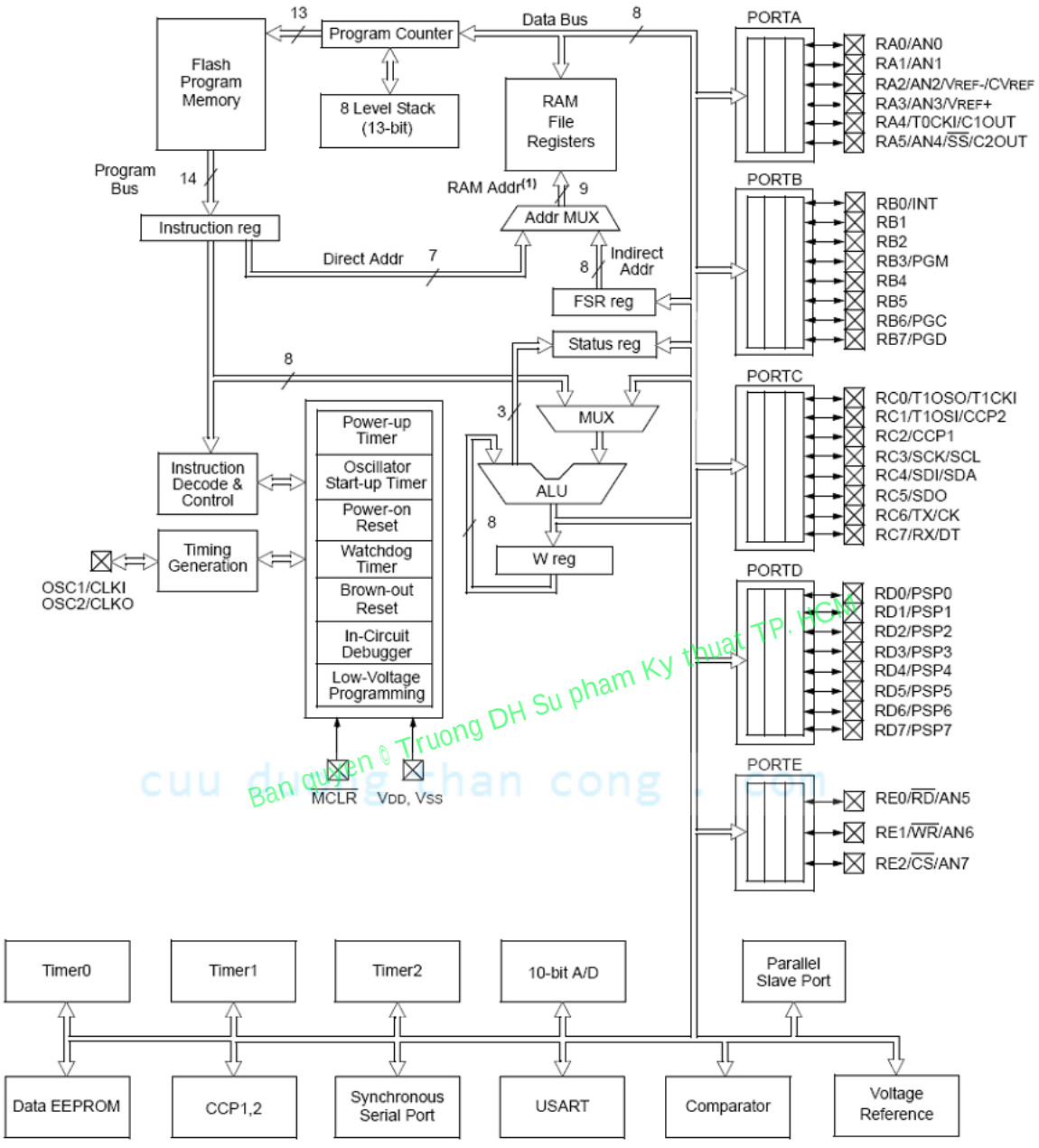
b. Sơ đồ khối:

Hình 2-2 trình bày sơ đồ khối của PIC16F877A, sơ đồ khối của PIC gồm các khối:

- Khối ALU – Arithmetic Logic Unit.
- Khối bộ nhớ chứa chương trình – Flash Program Memory.
- Khối bộ nhớ chứa dữ liệu EEPROM – Data EPROM.
- Khối bộ nhớ file thanh ghi RAM – RAM file Register.
- Khối giải mã lệnh và điều khiển – Instruction Decode Control.
- Khối thanh ghi đặc biệt.
- Khối ngoại vi timer.
- Khối giao tiếp nối tiếp.
- Khối chuyển đổi tín hiệu tương tự sang số –ADC.
- Khối các port xuất nhập.

c. Sơ đồ chân và chức năng các chân:

Sơ đồ chân của PIC gồm nhiều loại nhưng ở đây khảo sát loại PIC 40 chân như hình 2-3.



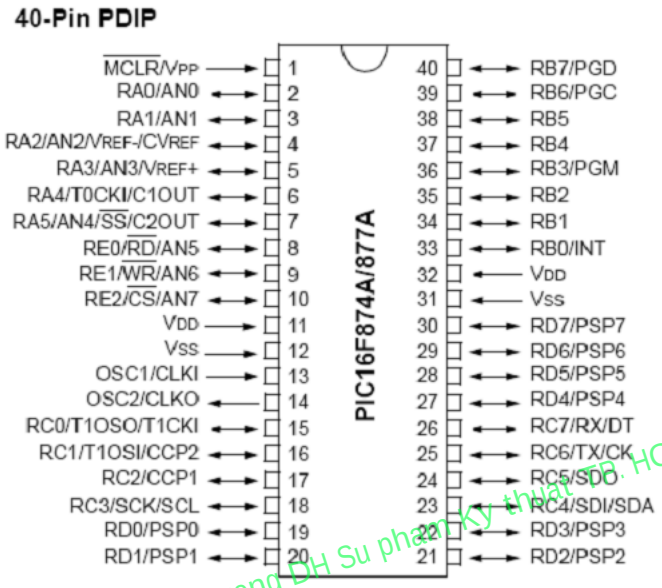
Device	Program Flash	Data Memory	Data EEPROM
PIC16F874A	4K words	192 Bytes	128 Bytes
PIC16F877A	8K words	368 Bytes	256 Bytes

Hình 2-2. Sơ đồ khối PIC16F87XA.

Chức năng của các chân như sau:

- Chân OSC1/CLKI (13): là ngõ vào kết nối với dao động thạch anh hoặc ngõ vào nhận xung clock bên ngoài.
  - OSC1: ngõ vào dao động thạch anh hoặc ngõ vào nguồn xung ở bên ngoài. Ngõ vào có mạch Schmitt Trigger nếu sử dụng dao động RC.
  - CLKI: ngõ vào nguồn xung bên ngoài.

- Chân OSC2/CLKO (14): ngõ ra dao động thạch anh hoặc ngõ ra cấp xung clock.
  - OSC2: ngõ ra dao động thạch anh. Kết nối đến thạch anh hoặc bộ cộng hưởng.
  - CLKO: ở chế độ RC, ngõ ra của OSC2, bằng ¼ tần số của OSC1 và chính là tốc độ của chu kì lệnh.



Hình 2-3. Sơ đồ chân

- Chân $\overline{MCLR}/V_{PP}$ (1): có 2 chức năng:
  - $\overline{MCLR}$: là ngõ vào reset tích cực mức thấp.
  - V_{PP}: khi lập trình cho PIC thì đóng vai trò là ngõ vào nhận điện áp lập trình.
- Chân RA0/AN0 (2): có 2 chức năng:
  - RA0: xuất/ nhập số.
  - AN0: ngõ vào tương tự của kênh thứ 0.
- Chân RA1/AN1(3):
  - RA0: xuất/nhập số.
  - AN1: ngõ vào tương tự của kênh thứ 1.
- Chân RA2/AN2/VREF-/CVREF (4):
  - RA2: xuất/nhập số.
  - AN2: ngõ vào tương tự của kênh thứ 2.
  - VREF-: ngõ vào điện áp chuẩn (thấp) của bộ A/D
  - CVREF: điện áp tham chiếu VREF ngõ ra bộ so sánh
- Chân RA3/AN3/VREF+ (5):
  - RA3: xuất/nhập số.
  - AN3: ngõ vào tương tự kênh thứ 3.
  - VREF+: ngõ vào điện áp chuẩn (cao) của bộ A/D.
- Chân RA4/TOCKI/C1OUT (6):

- RA4: xuất/nhập số – mở khi được cấu tạo là ngõ ra.
- TOCKI: ngõ vào xung clock bên ngoài cho Timer 0.
- C1OUT: ngõ ra bộ so sánh 1.

- Chân RA5/AN4/$\overline{SS}$/C2OUT (7):
  - RA5: xuất/nhập số.
  - AN4: ngõ vào tương tự kênh thứ 4.
  - $\overline{SS}$: ngõ vào chọn lựa SPI phụ.
  - C2OUT: ngõ ra bộ so sánh 2.
- Chân RB0/INT (33):
  - RB0: xuất/nhập số.
  - INT: ngõ vào nhận tín hiệu ngắt ngoài.
- Chân RB1 (34): xuất/nhập số.
- Chân RB2 (35): xuất/nhập số.
- Chân RB3/PGC:
  - RB3: xuất/nhập số.
  - Chân cho phép lập trình điện áp thấp ICSP.
- Chân RB4 (37), RB5 (38): xuất/nhập số.
- Chân RB6/PGC (39):
  - RB6: xuất/nhập số.
  - PGC: mạch gỡ rối và xung clock lập trình ICSP.
- Chân RB7/PGD (40):
  - RB7: xuất/nhập số.
  - PGD: mạch gỡ rối và dữ liệu lập trình ICSP.
- Chân RC0/T1OCO/T1CKI (15):
  - RC0: xuất/nhập số.
  - T1OCO: ngõ vào bộ dao động Timer1.
  - T1CKI: ngõ vào xung clock bên ngoài Timer1.
- Chân RC1/T1OSI/CCP2 (16):
  - RC1: xuất/nhập số.
  - T1OSI: ngõ vào bộ dao động Timer1.
  - CCP2: ngõ vào Capture2, ngõ ra compare2, ngõ ra PWM2.
- Chân RC2/CCP1 (17):
  - RC2: xuất/nhập số
  - CCP1: ngõ vào Capture1, ngõ ra compare1, ngõ ra PWM1
- Chân RC3/SCK/SCL (18):
  - RC3: xuất/nhập số.
  - SCK: ngõ vào xung clock nối tiếp đồng bộ/ngõ ra của chế độ SPI.

- SCL: ngõ vào xung clock nối tiếp đồng bộ/ngõ ra của chế độ I_2C.

- Chân RC4/SDI/SDA (23):
  - RC4: xuất/nhập số.
  - SDI: dữ liệu vào SPI.
  - SDA: xuất/nhập dữ liệu I^2C.
- Chân RC5/SDO (24):
  - RC5: xuất/nhập số.
  - SDO: dữ liệu ra SPI.
- Chân RC6/TX/CK (25):
  - RC6: xuất/nhập số.
  - TX: truyền bất đồng bộ USART.
  - CK: xung đồng bộ USART.
- Chân RC7/RX/DT (26):
  - RC7: xuất/nhập số.
  - RX: nhận bất đồng USART.
  - DT: dữ liệu đồng bộ USART.
- Chân RD0/PSP0 (19):
  - RD0: xuất/nhập số.
  - PSP0: dữ liệu port tớ song song.
- Chân RD1/PSP1 (20):
  - RD1: xuất/nhập số.
  - PSP1: dữ liệu port tớ song song.
- Các chân RD2/PSP2 (21), RD3/PSP3 (22), RD4/PSP4 (27), RD5/PSP5 (28), RD6/PSP6 (29), RD7/PSP7 (30) tương tự chân 19, 20.
- Chân RE0/$\overline{RD}$/AN5 (8):
  - RE0: xuất/nhập số.
  - $\overline{RD}$: điều khiển đọc port tớ song song.
  - AN5: ngõ vào tương tự 5.
- Chân RE1/$\overline{WR}$/AN6 (9):
  - RE1: xuất/nhập số.
  - $\overline{WR}$: điều khiển ghi port tớ song song.
  - AN6: ngõ vào tương tự kênh thứ 6.
- Chân RE2/$\overline{CS}$/AN7 (10):
  - RE2: xuất/nhập số.
  - $\overline{CS}$: Chip chọn lựa điều khiển port tớ song song.
  - AN7: ngõ vào tương tự kênh thứ 7.
- Chân VDD (11,32) và VSS (12, 31): là các chân nguồn của PIC.

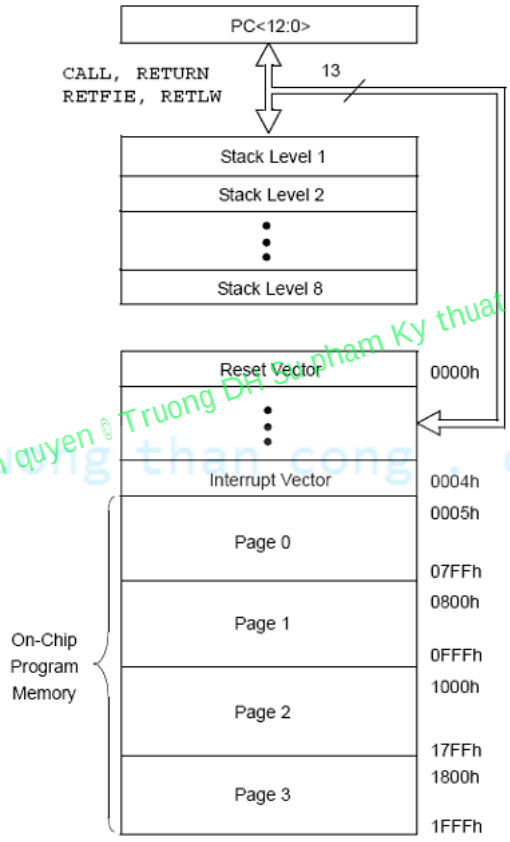
2. TỔ CHỨC BỘ NHỚ:

Có 3 khối bộ nhớ trong PIC16F877A. Bộ nhớ chương trình và bộ nhớ dữ liệu được mô tả chi tiết trong phần này. Khối bộ nhớ dữ liệu EEPROM được mô tả chi tiết ở phần sau.

a. Cấu trúc bộ nhớ chương trình:

PIC16F877A có bộ đếm chương trình 13 bit có thể quản lý bộ nhớ chương trình có dung lượng là 8Kword×14bit (1KWord = 14bit).

Khi PIC bị reset thì thanh ghi PC có giá trị là 0000h hay còn vector ngắt có địa chỉ 0004H.



Hình 2-4. Sơ đồ bộ nhớ chương trình và ngăn xếp.

b. Cấu trúc bộ nhớ dữ liệu:

Bộ nhớ dữ liệu được phân chia thành nhiều Bank và những thanh ghi chức năng đặc biệt. Hai bit RP_1RP_0 – bit trạng thái thứ 6 và thứ 5 được dùng để chọn bank như bảng 2-3.

Mỗi bank có thể mở rộng lên đến địa chỉ 7Fh (tương đương với 128byte). Các ô nhớ có địa chỉ thấp của mỗi bank được dành cho các thanh ghi chức năng đặc biệt. Trên các thanh ghi chức năng đặc biệt là các thanh ghi đa dụng – xem như bộ nhớ RAM. Tất cả các bank thanh ghi đều chứa những thanh ghi đặc biệt.

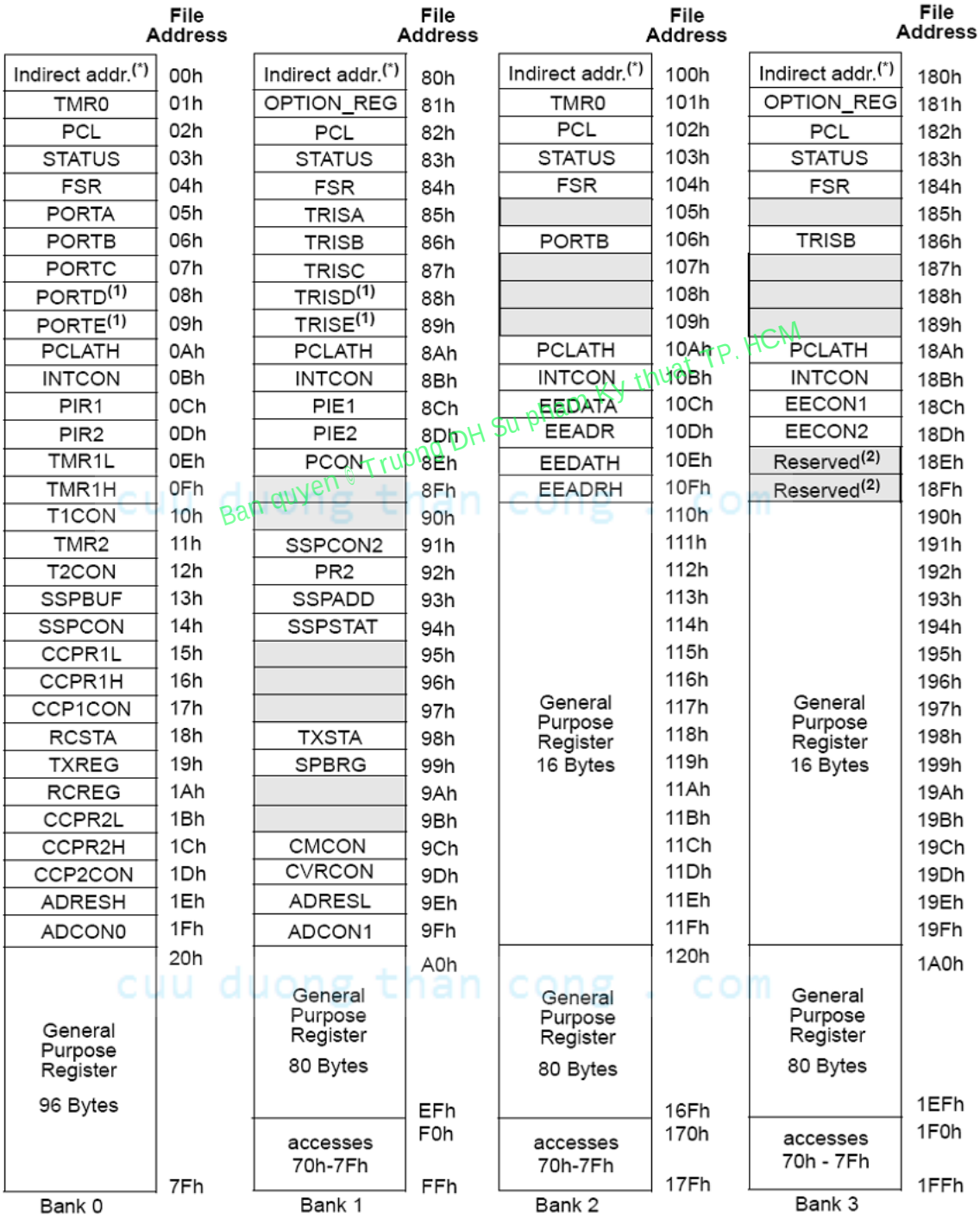
RP1:RP0	Bank
00	0
01	1

10	2
11	3

Bảng 2-3. Lựa chọn bank thanh ghi.

c. File thanh ghi kết quả tổng quát:

File thanh ghi có thể được truy xuất trực tiếp hoặc gián tiếp thông qua File thanh ghi đặc biệt.



Hình 2-5. Sơ đồ File thanh ghi.

Quy ước: (1) Các ô nhớ tô màu xám là chưa thiết kế nếu đọc sẽ có giá trị là 0.

Quy ước: (2) Các dấu (*) không phải là thanh ghi vật lý.

Chú ý: (1) Những thanh ghi này không có trong PIC 16F876A.

Chú ý: (2) Những thanh ghi này được bảo vệ.

Bảng liệt kê những thanh ghi nằm trong bank thứ 0 được trình bày trong bảng 2-4.

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Details on page:
Bank 0											
00h(3)	INDF	Addressing this location uses contents of FSR to address data memory (not a physical register)								0000 0000	31, 150
01h	TMR0	Timer0 Module Register								xxxx xxxx	55, 150
02h(3)	PCL	Program Counter (PC) Least Significant Byte								0000 0000	30, 150
03h(3)	STATUS	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	22, 150
04h(3)	FSR	Indirect Data Memory Address Pointer								xxxx xxxx	31, 150
05h	PORTA	—	—	PORTA Data Latch when written: PORTA pins when read						--0x 0000	43, 150
06h	PORTB	PORTB Data Latch when written: PORTB pins when read								xxxx xxxx	45, 150
07h	PORTC	PORTC Data Latch when written: PORTC pins when read								xxxx xxxx	47, 150
08h(4)	PORTD	PORTD Data Latch when written: PORTD pins when read								xxxx xxxx	48, 150
09h(4)	PORTE	—	—	—	—	—	RE2	RE1	RE0	---- -xxx	49, 150
0Ah(1,3)	PCLATH	—	—	—	Write Buffer for the upper 5 bits of the Program Counter					---0 0000	30, 150
0Bh(3)	INTCON	GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF	0000 000x	24, 150
0Ch	PIR1	PSPIF(3)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	26, 150
0Dh	PIR2	—	CMIF	—	EEIF	BCLIF	—	—	CCP2IF	-0-0 0--0	28, 150
0Eh	TMR1L	Holding Register for the Least Significant Byte of the 16-bit TMR1 Register								xxxx xxxx	60, 150
0Fh	TMR1H	Holding Register for the Most Significant Byte of the 16-bit TMR1 Register								xxxx xxxx	60, 150
10h	T1CON	—	—	T1CKPS1	T1CKPS0	T1OSCEN	$\overline{T1SYNC}$	TMR1CS	TMR1ON	--00 0000	57, 150
11h	TMR2	Timer2 Module Register								0000 0000	62, 150
12h	T2CON	—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0	-000 0000	61, 150
13h	SSPBUF	Synchronous Serial Port Receive Buffer/Transmit Register								xxxx xxxx	79, 150
14h	SSPCON	WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0	0000 0000	82, 82, 150
15h	CCPR1L	Capture/Compare/PWM Register 1 (LSB)								xxxx xxxx	63, 150
16h	CCPR1H	Capture/Compare/PWM Register 1 (MSB)								xxxx xxxx	63, 150
17h	CCP1CON	—	—	CCP1X	CCP1Y	CCP1M3	CCP1M2	CCP1M1	CCP1M0	--00 0000	64, 150
18h	RCSTA	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D	0000 000x	112, 150
19h	TXREG	USART Transmit Data Register								0000 0000	118, 150
1Ah	RCREG	USART Receive Data Register								0000 0000	118, 150
1Bh	CCPR2L	Capture/Compare/PWM Register 2 (LSB)								xxxx xxxx	63, 150
1Ch	CCPR2H	Capture/Compare/PWM Register 2 (MSB)								xxxx xxxx	63, 150
1Dh	CCP2CON	—	—	CCP2X	CCP2Y	CCP2M3	CCP2M2	CCP2M1	CCP2M0	--00 0000	64, 150
1Eh	ADRESH	A/D Result Register High Byte								xxxx xxxx	133, 150
1Fh	ADCON0	ADCS1	ADCS0	CHS2	CHS1	CHS0	$\overline{GO/DONE}$	—	ADON	0000 00-0	127, 150

Bảng 2-4. Tóm tắt các thanh ghi đặc biệt.

Quy ước: × = không xác định, u = không thay đổi, q= giá trị tuỳ thuộc vào điều kiện,- =chưa sử dụng nếu đọc sẽ có giá trị 0, r = dự trữ.

Chú ý: (1) Byte cao của thanh ghi PC không thể truy xuất trực tiếp. PCLATH chứa các bit PC<12:8>, nội dung của thanh ghi này sẽ chuyển cho byte cao của thanh ghi PC.

Chú ý: (2) Các bit PSPIE và PSPIF được dữ trữ cho PIC 16F873A/876A.

Chú ý: (3) Các thanh ghi có thể địa chỉ hoá từ bất kỳ bank nào.

Chú ý: (4) PORTD, PORTE, TRISD và TRISE không có trong PIC 16F873A/876A nếu đọc sẽ có giá trị 0.

Chú ý: (5) Bit thứ 4 của thanh ghi EEADRH chỉ được dùng cho PIC 16F876A/877A.

Bảng liệt kê những thanh ghi nằm trong bank thứ 1 được trình bày trong bảng 2-5.

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Details on page:
Bank 1											
80h(3)	INDF	Addressing this location uses contents of FSR to address data memory (not a physical register)								0000 0000	31, 150
81h	OPTION_REG	$\overline{RBPU}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	23, 150
82h(3)	PCL	Program Counter (PC) Least Significant Byte								0000 0000	30, 150
83h(3)	STATUS	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	22, 150
84h(3)	FSR	Indirect Data Memory Address Pointer								xxxx xxxx	31, 150
85h	TRISA	—	—	PORTA Data Direction Register						--11 1111	43, 150
86h	TRISB	PORTB Data Direction Register								1111 1111	45, 150
87h	TRISC	PORTC Data Direction Register								1111 1111	47, 150
88h(4)	TRISD	PORTD Data Direction Register								1111 1111	48, 151
89h(4)	TRISE	IBF	OBF	IBOV	PSPMODE	—	PORTE Data Direction bits			0000 -111	50, 151
8Ah(1,3)	PCLATH	—	—	—	Write Buffer for the upper 5 bits of the Program Counter					---0 0000	30, 150
8Bh(3)	INTCON	GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF	0000 000x	24, 150
8Ch	PIE1	PSPIE(2)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	25, 151
8Dh	PIE2	—	CMIE	—	EEIE	BCLIE	—	—	CCP2IE	-0-0 0--0	27, 151
8Eh	PCON	—	—	—	—	—	—	$\overline{POR}$	$\overline{BOR}$	---- --qq	29, 151
8Fh	—	Unimplemented								—	—
90h	—	Unimplemented								—	—
91h	SSPCON2	GCEN	ACKSTAT	ACKDT	ACKEN	RCEN	PEN	RSEN	SEN	0000 0000	83, 151
92h	PR2	Timer2 Period Register								1111 1111	62, 151
93h	SSPADD	Synchronous Serial Port (I^2C mode) Address Register								0000 0000	79, 151
94h	SSPSTAT	SMP	CKE	D/$\overline{A}$	P	S	R/$\overline{W}$	UA	BF	0000 0000	79, 151
95h	—	Unimplemented								—	—
96h	—	Unimplemented								—	—
97h	—	Unimplemented								—	—
98h	TXSTA	CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D	0000 -010	111, 151
99h	SPBRG	Baud Rate Generator Register								0000 0000	113, 151
9Ah	—	Unimplemented								—	—
9Bh	—	Unimplemented								—	—
9Ch	CMCON	C2OUT	C1OUT	C2INV	C1INV	CIS	CM2	CM1	CM0	0000 0111	135, 151
9Dh	CVRCON	CVREN	CVROE	CVRR	—	CVR3	CVR2	CVR1	CVR0	000- 0000	141, 151
9Eh	ADRESL	A/D Result Register Low Byte								xxxx xxxx	133, 151
9Fh	ADCON1	ADFM	ADCS2	—	—	PCFG3	PCFG2	PCFG1	PCFG0	00-- 0000	128, 151

Bảng 2-5. Tóm tắt các thanh ghi đặc biệt.

Bảng liệt kê những thanh ghi nằm trong bank thứ 2 được trình bày trong bảng 2-6.

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Details on page:
Bank 2											
100h(3)	INDF	Addressing this location uses contents of FSR to address data memory (not a physical register)								0000 0000	31, 150
101h	TMR0	Timer0 Module Register								xxxx xxxx	55, 150
102h(3)	PCL	Program Counter's (PC) Least Significant Byte								0000 0000	30, 150
103h(3)	STATUS	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	22, 150
104h(3)	FSR	Indirect Data Memory Address Pointer								xxxx xxxx	31, 150
105h	—	Unimplemented								—	—
106h	PORTB	PORTB Data Latch when written: PORTB pins when read								xxxx xxxx	45, 150
107h	—	Unimplemented								—	—
108h	—	Unimplemented								—	—
109h	—	Unimplemented								—	—
10Ah(1,3)	PCLATH	—	—	—	Write Buffer for the upper 5 bits of the Program Counter					---0 0000	30, 150
10Bh(3)	INTCON	GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF	0000 000x	24, 150
10Ch	EEDATA	EEPROM Data Register Low Byte								xxxx xxxx	39, 151
10Dh	EEADR	EEPROM Address Register Low Byte								xxxx xxxx	39, 151
10Eh	EEDATH	—	—	EEPROM Data Register High Byte						--xx xxxx	39, 151
10Fh	EEADRH	—	—	—	—(5)	EEPROM Address Register High Byte				---- xxxx	39, 151
Bank 3											
180h(3)	INDF	Addressing this location uses contents of FSR to address data memory (not a physical register)								0000 0000	31, 150
181h	OPTION_REG	$\overline{RBPU}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	23, 150
182h(3)	PCL	Program Counter (PC) Least Significant Byte								0000 0000	30, 150
183h(3)	STATUS	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	22, 150
184h(3)	FSR	Indirect Data Memory Address Pointer								xxxx xxxx	31, 150
185h	—	Unimplemented								—	—
186h	TRISB	PORTB Data Direction Register								1111 1111	45, 150
187h	—	Unimplemented								—	—
188h	—	Unimplemented								—	—
189h	—	Unimplemented								—	—
18Ah(1,3)	PCLATH	—	—	—	Write Buffer for the upper 5 bits of the Program Counter					---0 0000	30, 150
18Bh(3)	INTCON	GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF	0000 000x	24, 150
18Ch	EECON1	EEPGD	—	—	—	WRERR	WREN	WR	RD	x--- x000	34, 151
18Dh	EECON2	EEPROM Control Register 2 (not a physical register)								---- ----	39, 151
18Eh	—	Reserved; maintain clear								0000 0000	—
18Fh	—	Reserved; maintain clear								0000 0000	—

Bảng 2-6. Tóm tắt các thanh ghi đặc biệt.

d. Các thanh ghi có chức năng đặc biệt:

Những thanh ghi chức năng đặc biệt là những thanh ghi được sử dụng bởi CPU và những khối ngoại vi để điều khiển hoạt động theo yêu cầu của CPU. Những thanh ghi này xem như RAM tĩnh.

Thanh ghi trạng thái – STATUS

TGTT chứa trạng thái của khối ALU, trạng thái Reset và các bit chọn bank bộ nhớ dữ liệu.

STATUS REGISTER (ADDRESS 03h, 83h, 103h, 183h)

R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x
IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C
bit 7							bit 0

Chức năng của các bit trong thanh ghi trạng thái:

Bit 7	**IRP**: bit lựa chọn thanh ghi (dùng địa chỉ gián tiếp).
	1 = bank 2, 3 (100h-1FFh)
	0 = bank 0, 1 (00h- FFh)
Bit 6-5	**RP1:RP0**: các bit lựa chọn thanh ghi (dùng địa chỉ trực tiếp)
	11 = bank 3 (180h-1FFh)
	10 = bank 2 (100h- 17Fh)
	01 = bank 1 (80h- FFh)
	00 = bank 0 (00h- 7Fh)
	Mỗi bank là 128 byte.
Bit 4	$\overline{TO}$: Time-out bit (Bit thời gian chờ)
	1 = sau khi mở nguồn, lệnh CLRWDT hoặc SLEEP
	0 = thời gian chờ của WDT được thực hiện
Bit 3	$\overline{PD}$: Power-down bit (bit tắt nguồn)
	1= sau khi mở nguồn hoặc bằng lệnh CLRWDT
	0= thực thi lệnh SLEEP
Bit 2	**Z**: Zero bit (bit 0)
	1 = khi kết quả bằng 0.
	0 = khi kết quả khác 0.
Bit 1	**DC**: Digit carry/ $\overline{borrow}$ bit (các lệnh ADDWF, ADDLW, SUBLW, SUBWF) (bit tràn / mượn)
	1 = khi cộng 4 bit thấp bị tràn.
	0 = khi cộng 4 bit thấp không bị tràn.
Bit 0	**C**: Carry/ $\overline{borrow}$ bit (các lệnh ADDWF, ADDLW, SUBLW, SUBWF)
	1 = khi kết quả phép toán có tràn.
	0 = khi kết quả phép toán không bị tràn.

Chú ý: Nếu phép toán trừ thì trạng thái của cờ DC và C thì ngược lại và cụ thể như sau: nếu phép trừ lớn hơn 0 thì cờ C bằng 1, nếu kết quả trừ nhỏ hơn 0 thì cờ C bằng 0.

Quy ước: R = bit có thể đọc, W = bit có thể ghi, U = bit chưa sử dụng đọc là 0,

Quy ước: -n= giá trị tuỳ thuộc POR, '1'= bit bị SET, '0'= bit bị xoá, x= bit không xác định.

Thanh ghi OPTION_REG

Là thanh ghi có thể đọc/ghi, thanh ghi này có những bit điều khiển khác nhau để thiết lập bộ chia trước cho Timer0/WDT, ngắt INT bên ngoài, Timer0 và treo PORTB.

OPTION_REG REGISTER (ADDRESS 81h, 181h)

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

Bit 7	$\overline{RBPU}$: PORTB Pull-up Enable bit (bit cho phép treo PORTB)

1 = không cho phép treo PORTB.

0 = cho phép treo PORTB.

Bit 6 **INTEDG**: Interrupt Edge Select bit (bit lựa chọn cạnh ngắt)

1 = cho phép chân ngắt RB0/INT tích cực cạnh lên.

0 = cho phép chân ngắt RB0/INT tích cực cạnh xuống.

Bit 5 **T0CS**: TMR0 Clock Source Select bit (bit lựa chọn nguồn xung clock TMR0)

1 = cho phép nhận xung ngõ vào ở chân RA4/T0CKI.

0 = cho phép nhận xung nội bên trong.

Bit 4 **T0SE**: TMR0 Source Edge Select bit (bit lựa chọn kiểu tác động cho TMR0)

1 = cho phép xung vào chân RA4/T0CKI tích cực cạnh lên.

0 = cho phép xung vào chân RA4/T0CKI tích cực cạnh xuống.

Bit 3 **PSA**: Prescaler Assignment bit (bit gán bộ chia)

1 = bộ chia được gán cho WDT.

0 = bộ chia được gán cho Timer0.

Bit 2-0 **PS2:PS0**: Prescaler Rate Select bits (bit lựa chọn hệ số chia trước)

Giá trị bit	Tỉ lệ TMR0	Tỉ lệ WDT
000	1:2	1:1
001	1:4	1:2
010	1:8	1:4
011	1:16	1:8
100	1:32	1:16
101	1:64	1:32
110	1:128	1:64
111	1:256	1:128

Bảng 2-7. Các bit lựa chọn hệ số chia trước.

Chú ý: Khi sử dụng lập trình điện áp thấp ICSP (LVP) và treo PORTB được cho phép thì phải xoá bit 3 trong thanh ghi TRISB để không cho phép treo ở chân RB3 và đảm bảo cho hoạt động riêng của PIC.

Thanh ghi INTCON

Là thanh ghi có thể đọc và ghi, chứa những bit cờ và bit cho phép các ngắt khác nhau như ngắt khi TMR0 tràn, ngắt khi có thay đổi ở PORTB và ngắt ngoài ở chân RB0/INT.

INTCON REGISTER (ADDRESS 0Bh, 8Bh, 10Bh, 18Bh)

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

Bit 7 **GIE**: bit cho phép ngắt toàn cục

1= cho phép tất cả các nguồn ngắt.

0= không cho phép tất cả các nguồn ngắt.

Bit 6 **PEIE**: bit cho phép ngắt ngoại vi

1= cho phép ngắt.

0= không cho phép ngắt.

Bit 5 **TMR0IE**: bit cho phép ngắt TMR0

1= cho phép ngắt.

0= không cho phép ngắt.

Bit 4 **INTE**: bit cho phép ngắt ngoài ở chân RB0/INT

1= cho phép ngắt.

0= không cho phép ngắt.

Bit 3 **RBIE**: bit cho phép ngắt thay đổi PORTB

1= cho phép ngắt khi PORTB thay đổi

0= không cho phép ngắt khi PORTB thay đổi

Bit 2 **TMR0IF**: cờ tràn TMR0

1= thanh ghi TMR0 tràn (xóa bằng phần mềm).

0= thanh ghi TMR0 không tràn hay chưa tràn.

Bit 1 **INTF**: cờ báo ngắt ngoài RB0/INT.

1= ngắt ngoài ở chân RB0/INT đã xảy ra (xóa bằng phần mềm)

0= ngắt ngoài ở chân RB0/INT không xảy ra

Bit 0 **RBIF**: cờ báo khi PORTB có thay đổi

1= có ít nhất các chân RB7:RB4 thay đổi trạng thái; điều kiện không tương thích sẽ tiếp tục làm bit này bằng 1. Khi đọc PORTB sẽ chấm dứt điều kiện không tương thích và cho phép xóa cờ báo này bằng phần mềm.

0= các chân RB7:RB4 không có sự thay đổi trạng thái.

Thanh ghi PIE1

Là thanh ghi chứa các bit cho phép ngắt độc lập cho các ngắt ngoại vi.

PIE1 REGISTER (ADDRESS 8Ch)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE
bit 7							bit 0

Bit 7 **PSPIE**: bit cho phép ngắt đọc/ghi ở port nhánh song song

1= cho phép ngắt PSP đọc/ghi

0= không cho phép ngắt PSP đọc/ghi

Bit 6 **ADIE**: bit cho phép ngắt bộ chuyển đổi A/D

1= cho phép ngắt.

0= không cho phép ngắt.

Bit 5 **RCIE**: bit cho phép ngắt nhận dữ liệu USART

1= cho phép ngắt.

0= không cho phép ngắt.

Bit 4 **TXIE**: bit cho phép ngắt phát dữ liệu USART

1= cho phép ngắt.

0= không cho phép ngắt.

Bit 3 **SSPIE**: bit cho phép ngắt port nối tiếp đồng bộ

1= cho phép ngắt SSP.

0= không cho phép ngắt SSP.

Bit 2 **CCP1IE**: bit cho phép ngắt CCP1

1= cho phép ngắt CCP1.

0= không cho phép ngắt CCP1.

Bit 1 **TMR2IE**: bit cho phép ngắt tương thích ứng TMR2 với PR2

1= cho phép ngắt tương thích TMR2 với PR2.

0= không cho phép ngắt tương thích TMR2 với PR2.

Bit 0 **TNR1IE:** bit cho phép ngắt tràn TMR1

1= cho phép ngắt TMR1 tràn.

0= không cho phép ngắt TMR1 tràn.

Thanh ghi PIR1

Là thanh ghi chứa các bit cờ cho các ngắt ngoài.

Chú ý: những bit cờ ngắt được Set khi điều kiện ngắt xảy ra bất chấp trạng thái của bit cho phép hoặc bit cho phép ngắt toàn cục GIE (INTCON<7>). Phần mềm của người dùng phải đảm bảo những bit ngắt tương ứng phải bị xóa trước khi cho phép ngắt.

PIR1 REGISTER (ADDRESS 0Ch)

R/W-0	R/W-0	R-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0
PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
bit 7							bit 0

Bit 7 **PSPIF**: cờ báo ngắt port nhánh song song đọc/ghi

1= hoạt động đọc hoặc ghi được thực thi.

0= hoạt động đọc hoặc ghi không xảy ra.

Bit 6 **ADIF**: cờ báo ngắt bộ chuyển đổi A/D

1= chuyển đổi A/D đã được hoàn thành.

0= chuyển đổi A/D chưa được hoàn thành.

Bit 5 **RCIF**: cờ báo ngắt nhận USART

1= USART nhận xong.

0= USART nhận chưa xong.

Bit 4 **TXIF**: cờ báo ngắt truyền USART

1= truyền đệm của USART còn trống.

0= truyền đệm của USART đầy.

Bit 3 **SSPIF:** cờ báo ngắt port nối tiếp đồng bộ (SSP)

1= điều kiện ngắt SSP đã xảy ra và phải xóa bằng phần mềm trước khi quay trở về từ chương trình con phục vụ ngắt (Interrupt Service Routine). Điều kiện để bit trạng thái này lên 1 là:

- SPI- truyền/nhận đã được thực thi.
- I^2C Slave: truyền/nhận đã được thực thi.
- I^2C Master:
  - Truyền/nhận đã được thực thi.
  - Điều kiện Start khởi động đã được hoàn thành bởi khối SSP.
  - Điều kiện Stop khởi động đã được hoàn thành bởi khối SSP.
  - Điều kiện Restart khởi động đã được hoàn thành bởi khối SSP.
  - Điều kiện bắt tay đã được hoàn thành bởi khối SSP.
  - Điều kiện Start đã xảy ra khi khối SSP đang ở trạng thái rỗi (multi-master system: hệ thống nhiều chủ).
  - Điều kiện Stop đã xảy ra khi khối SSP đang ở trạng thái rỗi.

0= không có điều kiện ngắt SSP nào xảy ra.

Bit 2 **CCP1IF:** cờ báo ngắt CCP1

Chế độ Capture:

1= thanh ghi bắt nhịp TMR1 có xảy ra (xóa bằng phần mềm).

0= thanh ghi bắt nhịp TMR1 không xảy ra.

Chế độ so sánh:

1= thích ứng so sánh thanh ghi TMR1 có xảy ra.

0= thích ứng so sánh thanh ghi TMR1 không xảy ra.

Bit 1 **TMR2IF**: cờ báo ngắt tương thích TMR2 với PR2

1= TMR2 tương thích với PR2 (xóa bằng phần mềm).

0= TMR2 không tương thích với PR2.

Bit 0 **TMR1IF**: cờ báo ngắt tràn TMR1

1= thanh ghi TMR1 đã tràn.

0= thanh ghi TMR1 không tràn.

"1"= bit được set

Thanh ghi PIE2

Là thanh ghi chứa các bit cho phép ngắt ngoại vi CCP2, ngắt xung đột đường truyền SSP, ngắt hoạt động ghi của EEPROM và ngắt của bộ so sánh.

PIE2 REGISTER (ADDRESS 8Dh)

U-0	R/W-0	U-0	R/W-0	R/W-0	U-0	U-0	R/W-0
—	CMIE	—	EEIE	BCLIE	—	—	CCP2IE
bit 7							bit 0

Bit 7	Chưa dùng: đọc là '0'
Bit 6	**CMIE**: bit cho phép ngắt bộ so sánh
	1= cho phép ngắt.
	0= không cho phép ngắt.
Bit 5	Chưa dùng: đọc là '0'
Bit 4	**EEIE**: bit cho phép ngắt hoạt động ghi của EEPROM
	1= cho phép ngắt.
	0= không cho phép ngắt.
Bit 3	**BCLIE**: bit cho phép ngắt sự xung đột đường dẫn
	1= cho phép ngắt.
	0= không cho phép ngắt.
Bit 2-1	Chưa dùng: đọc là '0'
Bit 0	**CCP2IE**: bit cho phép ngắt CCP2
	1= cho phép ngắt.
	0= không cho phép ngắt.

Chú ý: bit PEIE (INTCON<6>) phải được set để cho phép bất kì sự ngắt ngoài nào.

Thanh ghi PIR2

Là thanh ghi chứa các bit cờ báo ngắt CCP2, ngắt xung đột đường dẫn SSP, ngắt hoạt động ghi của EEPROM và ngắt bộ so sánh.

PIR2 REGISTER (ADDRESS 0Dh)

U-0	R/W-0	U-0	R/W-0	R/W-0	U-0	U-0	R/W-0
—	CMIF	—	EEIF	BCLIF	—	—	CCP2IF
bit 7							bit 0

Bit 7	Chưa dùng: đọc là '0'.
Bit 6	**CMIF**: cờ báo ngắt bộ so sánh
	1= ngõ vào bộ so sánh đã thay đổi (xóa bằng phần mềm).
	0= ngõ vào bộ so sánh không thay đổi.
Bit 5	Chưa dùng: đọc là '0'.
Bit 4	**EEIF**: cờ báo ngắt hoạt động ghi của EEPROM
	1= hoạt động ghi được hoàn thành (xóa bằng phần mềm).
	0= hoạt động ghi chưa hoàn thành hoặc chưa khởi động.
Bit 3	**BCLIF**: cờ báo ngắt xung đột đường dẫn

1= xung đột đường dẫn đã xảy ra trong SSP khi được thiết lập cấu hình ở chế độ I^2C chủ.

0= đường dẫn không xảy ra xung đột.

Bit 2-1 Chưa dùng: đọc là '0'

Bit 0 **CCP2IF**: bit cờ ngắt CCP2

Chế độ Capture:

1= thanh ghi bắt nhịp TMR1 xảy ra (xóa bằng phần mềm).

0= thanh ghi bắt nhịp TMR1 chưa xảy ra.

Chế độ so sánh:

1= tương thích so sánh thanh ghi TMR1 xảy ra (xóa bằng phần mềm).

0= tương thích so sánh thanh ghi TMR1 chưa xảy ra.

Chế độ PWM:

Không được sử dụng

Chú ý: cờ báo ngắt được Set khi ngắt xảy ra với điều kiện bit cho phép tương ứng hoặc toàn bộ bit được phép, GIE (INTCON<7>). Người dùng phải đảm bảo sự phù hợp của những bit ngắt được xóa sớm hơn để cho phép ngắt.

Thanh ghi PCON

Thanh ghi PCON (Power Control) chứa các cờ để cho phép phân biệt sự khác nhau của các trạng thái reset: khi mở điện – Power-on Reset (POR), Brown-out Reset (BOR), Watchdog Reset (WDT) và $\overline{MCLR}$ Reset.

PCON REGISTER (ADDRESS 8Eh)

U-0	U-0	U-0	U-0	U-0	U-0	R/W-0	R/W-1
—	—	—	—	—	—	$\overline{POR}$	$\overline{BOR}$
bit 7							bit 0

Bit 7-2 Chưa dùng: đọc là '0'

Bit 1 $\overline{POR}$: bit trạng thái Power-on Reset

1= reset khi mở điện không xảy ra.

0= reset khi mở điện đã xảy ra (phải Set bằng phần mềm sau khi Power-on Reset xảy ra).

Bit 0 $\overline{BOR}$: bit trạng thái Brown-out Reset

1= Brown-out Reset không xảy ra

0= Brown-out Reset xảy ra (phải Set bằng phần mềm sau khi Brown-out Reset xảy ra).

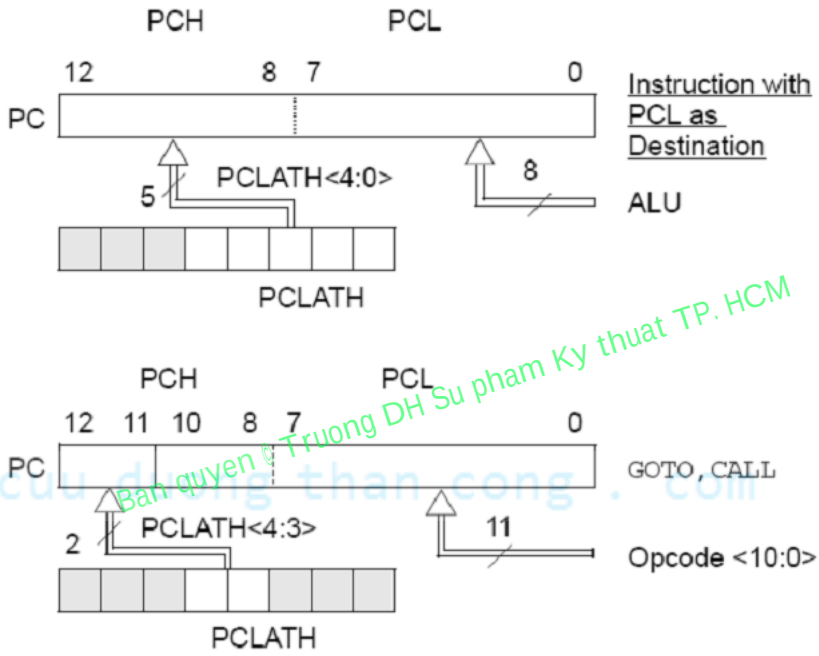
Thanh ghi PC và PCLATH

Thanh ghi bộ đếm chương trình (PC) có độ rộng 13 bit, 8 bit thấp nằm ở thanh ghi PCL, các bit này có thể đọc và ghi. Các bit cao còn lại (8:12) thì không thể đọc nhưng có thể ghi gián tiếp thông qua thanh ghi PCLATH. Khi bất kỳ Reset nào xảy ra thì các bit cao của thanh ghi PC sẽ bị

xóa. Hình 2-6 trình bày các tình huống thanh ghi PC được nạp giá trị. Các ví dụ sẽ minh hoạ cách thức PC được nạp giá trị thông qua lệnh CALL hoặc lệnh GOTO.

Tính toán GOTO

Tính toán GOTO được thực hiện bằng cách cộng giá trị lệnh (offset) với bộ đếm chương trình PC (bằng lệnh ADDWF PCL). Khi thực hiện đọc bảng dùng phương pháp tính toán GOTO, nên quan tâm đến giới hạn của bảng nếu vị trí bảng vượt quá giới hạn bộ nhớ PCL (mỗi khối bộ nhớ gồm 256 byte).



Hình 2-6. Các trường hợp nạp giá trị cho PC.

Ngăn xếp

PIC16F877A có 8 ngăn xếp, độ rộng là 13 bit, bộ nhớ ngăn xếp không lấy từ bộ nhớ chương trình hay dữ liệu, con trỏ ngăn xếp không thể đọc hoặc ghi.

Địa chỉ của thanh ghi PC được cất vào ngăn xếp khi thực hiện lệnh CALL hoặc ngắt xảy ra. Dữ liệu trong ngăn xếp được lấy ra khi thực hiện các lệnh RETURE, RETLW hoặc RETFIE. Thanh ghi PCLATH không bị ảnh hưởng bởi việc cất và lấy.

Ngăn xếp hoạt động như là một vòng kín. Điều đó có nghĩa sau khi ngăn xếp được cất vào 8 lần, thì lần thứ 9 sẽ ghi chồng lên giá trị đã được lưu từ lần cất vào đầu tiên. Lần cất thứ 10 sẽ ghi chồng lên giá trị đã được lưu ở lần cất thứ 2.

Chú ý: Không có bit trạng thái để báo ngăn xếp bị tràn hoặc chưa tràn. Không có các lệnh gọi là lệnh cất (PUSH) hoặc lấy (POP). Những lệnh liên quan đến ngăn xếp là lệnh CALL, RETURN, RETLW và RETFIE hoặc ngắt.

e. Phân trang bộ nhớ chương trình:

PIC16F877A có bộ nhớ chương trình là 8K word. Những lệnh CALL và GOTO chỉ cung cấp địa chỉ 11bit để cho phép chương trình con nằm trong phạm vi 2K word trang bộ nhớ.

Khi thực hiện lệnh CALL hoặc GOTO thì 2 bit cao của địa chỉ được lấy từ thanh ghi PCLATH (4:3). Khi thực hiện lệnh CALL hoặc GOTO, người dùng phải đảm bảo rằng bit lựa chọn trang đã được lập trình để xác định đúng địa chỉ trang bộ nhớ chương trình. Nếu thực hiện lệnh trở về thì toàn bộ 13 bit được lấy từ ngăn xếp trao cho PC. Vì thế các bit của thanh ghi PCLATH(4:3) không cần khi thực hiện lệnh trở về.

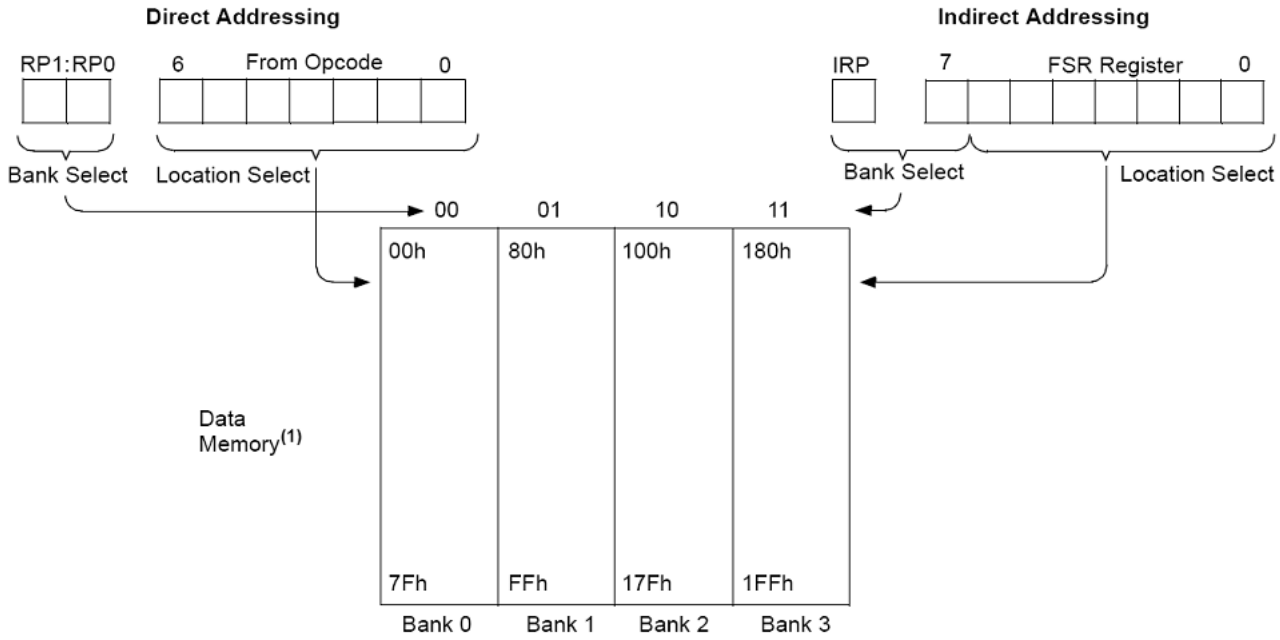
Chú ý: các thành phần của thanh ghi PCLATH không bị thay đổi sau khi thực hiện lệnh RETURN hoặc RETFIL. Người sử dụng phải nạp lại nội dung cho thanh ghi PCLATH khi gọi chương trình con tiếp theo hoặc cho lệnh GOTO.

Ví dụ 2-1 minh hoạ cho lệnh gọi chương trình con nằm trong trang 1 của bộ nhớ chương trình. Ví dụ này giả sử rằng thanh ghi PCLATH được cất và được khôi phục bởi chương trình con phục vụ ngắt.

```
                ORG 0x500
                BCF PCLATH,4
                BSF PCLATH,3    ;Select page 1
                                ;(800h-FFFh)
                CALL SUB1_P1    ;Call subroutine in
                :               ;page 1 (800h-FFFh)
                :
                ORG 0x900       ;page 1 (800h-FFFh)
SUB1_P1
                :               ;called subroutine
                                ;page 1 (800h-FFFh)
                :
                RETURN          ;return to
                                ;Call subroutine
                                ;in page 0
                                ;(000h-7FFh)
```


f. Các thanh ghi địa chỉ gián tiếp, thanh ghi INDF và FSR:

Thanh ghi INDF không phải là thanh ghi vật lí. Địa chỉ hoá thanh ghi INDF sẽ tạo ra địa chỉ gián tiếp. Địa chỉ gián tiếp thì có thể thực hiện được bằng cách dùng thanh ghi INDF. Bất kì lệnh nào sử dụng thanh ghi INDF đều truy xuất thanh ghi chỉ định bởi thanh ghi lựa chọn file (File Select Register –FSR). Khi đọc gián tiếp chính thanh ghi INDF này sẽ đọc là 00h. Khi ghi gián tiếp vào thanh ghi INDF, kết quả sẽ không được gì (mặc dù các bit của thanh ghi trạng thái có thể bị ảnh hưởng). Một chương trình đơn để xóa RAM ở vị trí 20h-2Fh sử dụng địa chỉ gián tiếp được trình bày trong ví dụ 2-2:



Hình 2-7. Địa chỉ trực tiếp/gián tiếp.

Ví dụ 2-2:

```
                MOWLW   0x20    ;đưa 20h vào thanh ghi W
                MOVWF   FSR     ;đưa nội dung của W vào FSR
NEXT            CLRF    INDF    ;xóa INDF
                INCF    FSR,F   ;tăng FSR
                BTFSSFSR,4      ;kiểm tra bit 4 trong FSR
                GOTO NEXT       ;nhảy đến NEXT
CONTINUE
                :               ;tiếp tục
```


3. DỮ LIỆU EEPROM VÀ BỘ NHỚ CHƯƠNG TRÌNH FLASH:

Dữ liệu EEPROM và bộ nhớ chương trình Flash có thể đọc và ghi trong suốt quá trình hoạt động bình thường. Bộ nhớ này không được thiết lập trực tiếp trong không gian file thanh ghi. Chúng được định địa chỉ gián tiếp thông qua các thanh ghi đặc biệt. Có 6 thanh ghi FSR được sử dụng để đọc và ghi bộ nhớ này:

- ✓ EECON1
- ✓ EECON2
- ✓ EEDATA
- ✓ EEDATH
- ✓ EEADR
- ✓ EEADRH

Khi giao tiếp với khối bộ nhớ dữ liệu, thanh ghi EEDATA chứa 8bit dữ liệu cho việc đọc/ghi và thanh ghi EEADR chứa địa chỉ ô nhớ của EEPROM đang được truy xuất. Nếu PIC có bộ nhớ

EEPROM là 128 byte thì địa chỉ nằm trong khoảng từ 80H đến FFH, nếu PIC có bộ nhớ EEPROM là 256 byte thì địa chỉ nằm trong khoảng từ 00H đến FFH.

Khi giao tiếp với khối bộ nhớ chương trình thì hai thanh ghi EEDATA và EEDATH kết hợp lại với nhau thành thanh ghi 16 bit để lưu trữ dữ liệu 14 bit cho lệnh đọc/ghi và hai thanh ghi EEADR và EEADRH kết hợp lại với nhau thành thanh ghi 16 bit để lưu trữ địa chỉ 13 bit của ô nhớ đang truy xuất. Với PIC có dung lượng bộ nhớ chương trình là 4Kword thì địa chỉ nằm trong khoảng từ 0000H đến 0FFFH, với PIC có dung lượng bộ nhớ chương trình là 8Kword thì địa chỉ nằm trong khoảng từ 0000H đến 1FFFH. Nếu truy xuất ô nhớ có địa chỉ lớn hơn thì sẽ bị cuộn nằm trong vùng nhớ thực.

Bộ nhớ dữ liệu EEPROM cho phép đọc và ghi 1byte. Bộ nhớ chương trình Flash cho phép đọc 1 word và ghi khối 4 word. Hoạt động ghi của bộ nhớ chương trình sẽ tự động thực hiện xóa trước khi ghi vào khối 4 word. Một byte ghi vào bộ nhớ dữ liệu EEPROM sẽ tự động xóa ô nhớ rồi mới ghi dữ liệu mới – xoá trước khi ghi.

Khi chip có mã bảo vệ thì CPU có thể đọc và ghi dữ liệu bộ nhớ EEPROM. Tuỳ thuộc vào cách thiết lập các bit bảo vệ chống ghi, PIC có thể cho hoặc không cho ghi dữ liệu vào một vài khối bộ nhớ chương trình; tuy nhiên cho phép đọc bộ nhớ chương trình. Khi PIC có mã bảo vệ, thì người dùng không còn được phép truy xuất bộ nhớ dữ liệu hoặc bộ nhớ chương trình.

a. Thanh ghi EEADR và EEADRH:

Cặp thanh ghi EEADRH:EEADR có thể định địa chỉ tối đa 256 byte của bộ nhớ dữ liệu EEPROM hoặc tối đa 8k word của bộ nhớ chương trình EEPROM.

Khi truy xuất bộ nhớ dữ liệu thì chỉ dùng thanh ghi EEADR để lưu byte địa chỉ thấp.

Khi truy xuất bộ nhớ chương trình thì dùng thanh ghi EEADR để lưu byte địa chỉ thấp và thanh ghi EEADRH lưu địa chỉ byte cao.

b. Thanh ghi EECON1 và EECON2:

EECON1 là thanh ghi điều khiển để truy xuất bộ nhớ. Bit điều khiển EEPGD dùng để xác định truy xuất bộ nhớ chương trình hoặc bộ nhớ dữ liệu. Khi reset hoặc khi xoá bit EEPGD sẽ cho phép truy xuất bộ nhớ dữ liệu. Khi bit EEPGD bằng 1 thì truy xuất bộ nhớ chương trình.

EECON1 REGISTER (ADDRESS 18Ch)

R/W-x	U-0	U-0	U-0	R/W-x	R/W-0	R/S-0	R/S-0
EEPGD	—	—	—	WRERR	WREN	WR	RD
bit 7							bit 0

Bit 7 **EEPGD**: bit lựa chọn bộ nhớ dữ liệu/chương trình EEPROM

1= truy xuất bộ nhớ chương trình

0= truy xuất bộ nhớ dữ liệu

Đọc ‘0’ sau khi POR, bit này không thay đổi trong khi hoạt động ghi đang diễn ra.

Bit 6-4 Chưa sử dụng: đọc là ‘0’

Bit 3 **WRERR**: bit cờ lỗi EEPROM

1= việc ghi thực hiện xong sớm.

0= việc ghi đã được hoàn thành.

Bit 2 **WREN**: bit cho phép ghi EEPROM

1= cho phép ghi.

0= không cho ghi.

Bit 1 **WR**: bit điều khiển ghi

1= Bắt đầu chu kỳ ghi. Bit **WR** được xóa bằng phần cứng sau mỗi lần ghi xong.

0= quá trình ghi vào EEPROM đã hoàn thành.

Bit 0 **RD:** bit điều khiển đọc

1= bắt đầu chu kỳ đọc EEPROM. Bit **RD** được xoá bằng phần cứng. Bit **RD** chỉ có thể được Set trong phần mềm.

0= không khởi động chu kỳ đọc EEPROM.

Các bit điều khiển, RD và WR khởi động đọc và ghi hoặc xoá theo thứ tự. Các bit này không thể xoá mà chỉ set bằng phần mềm. Các bit này được xoá bằng phần cứng ngay sau khi các hoạt động đọc ghi thực hiện xong.

Khi set bit WREN sẽ cho phép ghi hoặc xóa. Khi mở nguồn bit WREN bị xóa. Bit WREN được set khi hoạt động ghi hoặc xóa bị ngắt bởi $\overline{MCLR}$ hoặc WDT Time-out Reset trong quá trình hoạt động bình thường. Khi rơi vào tình huống này thì người dùng có thể kiểm tra bit WRERR và có thể ghi lại. Dữ liệu và địa chỉ trong các thanh ghi EEDATA và EEADR không bị thay đổi.

Bit cờ ngắt EEIF trong thanh ghi PIR2 được set khi quá trình ghi hoàn tất và phải xóa cờ ngắt này bằng phần mềm.

Thanh ghi EECON2 không là thanh ghi vật lí. Khi đọc EECON2 sẽ có giá trị là ‘0’. EECON2 được sử dụng riêng cho hoạt động ghi dữ liệu vào EEPROM.

c. Đọc dữ liệu từ bộ nhớ EEPROM:

Để đọc dữ liệu của 1 ô nhớ thì người sử dụng phải ghi địa chỉ vào thanh ghi EEADR, xóa bit điều khiển EEPGD (EECON1<7>) và sau đó Set bit điều khiển RD (EECON1<1>). Dữ liệu sẽ xuất hiện trong thanh ghi EEDATA ở chu kì kế. EEDATA sẽ lưu trữ giá trị này cho đến khi xuất hiện lần đọc kế hoặc bị thay đổi bởi người sử dụng.

Các bước để đọc bộ nhớ dữ liệu EEPROM:

- Ghi địa chỉ vào EEADR địa chỉ không được lớn hơn dung lượng bộ nhớ.
- Xóa bit EEPGD chỉ hướng vào bộ nhớ dữ liệu EEPROM
- Set bit RD để bắt đầu hoạt động đọc.
- Đọc dữ liệu từ thanh ghi EEPROM.

Ví dụ 2-3: đọc dữ liệu EEPROM

```
BSF     STATUS, RP1         ;
BCF     STATUS, RP0         ; bank 2

MOVF    DATA_EE_ADDR, W     ; data memory
```

```
MOVWF   EEADR           ; address to read
BSF     STATUS, RP0     ; bank 3
BCF     EECON1, EEPGD   ; point to data
                        ;  memory
BSF     EECON1, RD      ; EE read
BCF     STATUS,RP0      ; bank 2

MOVF    EEDATA, W       ; w=EEDATA
```

d. Ghi dữ liệu vào bộ nhớ EEPROM:

Để ghi dữ liệu vào EEPROM thì người sử dụng phải ghi địa chỉ vào thanh ghi EEADR và dữ liệu vào thanh ghi EEDATA. Sau đó phải thực hiện ghi theo trình tự chỉ định để bắt đầu ghi cho mỗi byte.

Quá trình ghi sẽ không được khởi động nếu thứ tự ghi không được thực hiện chính xác (ghi 55h vào EECON2, ghi AAh vào EECON2, sau đó set bit WR) cho mỗi byte. Phải cấm tất cả các yêu cầu ngắt khi thực hiện quá trình ghi này.

Ngoài ra, bit WREN trong thanh ghi EECON2 phải được Set để cho phép ghi. Cơ chế này ngăn chặn các hoạt động ghi ngẫu nhiên vào EEPROM liên quan đến sai sót mã bảo vệ. Người sử dụng nên giữ bit WREN ở trạng thái Clear, ngoại trừ khi cập nhập dữ liệu vào bộ nhớ dữ liệu EEPROM. Bit WREN không xóa được bằng phần cứng.

Sau khi trình tự ghi đã được khởi động thì nếu ta xóa bit WREN sẽ không ảnh hưởng đến chu kì ghi này. Bit WR sẽ bị chặn không cho lên 1 trừ khi bit WREN được Set.

Khi hoàn tất chu kì ghi, bit WR được xóa bởi phần cứng và bit cờ báo ngắt hoàn thành xong quá trình ghi EEIF được Set. Người dùng có thể cho phép sự ngắt hoặc kiểm tra bit này để biết quá trình ghi kết thúc. Bit EEIF phải được xóa bằng phần mềm.

Các bước để ghi vào bộ nhớ dữ liệu EEPROM:

Bước 1: Nếu bước thứ 10 không hoàn thành thì kiểm tra bit WR để xem có còn trong tiến trình ghi hay không.

Bước 2: Ghi địa chỉ vào EEADR và địa chỉ không lớn hơn dung lượng bộ nhớ.

Bước 3: Ghi dữ liệu 8bit vào thanh ghi EEDATA.

Bước 4: Xóa bit EEPGD để chọn bộ nhớ dữ liệu EEPROM.

Bước 5: Set bit WREN để cho phép hoạt động ghi.

Bước 6: Cấm tất cả ngắt (nếu đã cho phép ngắt trước đó).

Bước 7: Thực hiện tuần tự 5 lệnh đặc biệt:

- Ghi 55H vào thanh ghi EECON2 được chia thành 2 bước: bước thứ nhất vào thanh ghi W, bước thứ 2 vào thanh ghi EECON2.
- Ghi AAH vào thanh ghi EECON2 được chia thành 2 bước: bước thứ nhất vào thanh ghi W, bước thứ 2 vào thanh ghi EECON2.
- Set bit WR.

Bước 8: Cho phép ngắt trở lại nếu có sử dụng ngắt.

Bước 9: Xóa bit WREN để không cho phép hoạt động ghi.

Bước 10: Khi hoàn thành chu kì ghi, bit WR được xóa và bit cờ báo ngắt EEIF được set. (EEIF phải được xóa). Nếu bước 1 không hoàn tất thì phần mềm sẽ kiểm tra EEIF để set, hoặc WR để xóa, để xác định chu kỳ ghi.

```
BSF     STATUS,RP1      ;
BSF     STATUS,RP0
BTFSC   EECON1,WR       ;Wait for write
GOTO    $-1             ;to complete
BCF     STATUS,  RP0    ;Bank 2
MOVF    DATA_EE_ADDR,W  ;Data Memory
MOVWF   EEADR           ;Address to write
MOVF    DATA_EE_DATA,W  ;Data Memory Value
MOVWF   EEDATA          ;to write
BSF     STATUS,RP0      ;Bank 3
BCF     EECON1,EEPGD    ;Point to DATA
                        ;memory
BSF     EECON1,WREN     ;Enable writes

BCF     INTCON,GIE      ;Disable INTs.
MOVLW   55h             ;
MOVWF   EECON2          ;Write 55h
MOVLW   AAh             ;
MOVWF   EECON2          ;Write AAh
BSF     EECON1,WR       ;Set WR bit to
                        ;begin write
BSF     INTCON,GIE      ;Enable INTs.
BCF     EECON1,WREN     ;Disable writes
```

Required Sequence

e. Đọc dữ liệu từ bộ nhớ chương trình Flash:

Để đọc dữ liệu của bộ nhớ dữ liệu thì phải nạp địa chỉ 2 byte vào hai thanh ghi EEADR và EEADRH, set bit điều khiển (EECON1<7>) và sau đó set bit điều khiển RD (EECON1<0>). Mỗi lần set bit điều khiển đọc thì bộ điều khiển bộ nhớ chương trình Flash sẽ dùng hai chu kì lệnh tiếp theo để đọc dữ liệu. Điều này làm cho hai lệnh sau "BSF EECON1, RD" bị bỏ qua.

Dữ liệu sẽ có hiệu lực trong hai thanh ghi EEDATA và EEDATH ở chu kì kế tiếp, vì vậy có thể đọc 2 byte ngay sau lệnh tiếp theo. Hai thanh ghi EEDATA và EEDATH sẽ lưu dữ liệu cho đến khi thực hiện lần đọc tiếp theo hoặc do ghi bởi người sử dụng.

Ví dụ 2-4: đọc bộ nhớ chương trình Flash

```
BSF     STATUS, RP1             ;
BCF     STATUS, RP0             ; bank 2
MOVLW   MS_PROG_EE_ADDR         ;
MOVWF   EEADRH                  ; MS byte of Program Address to read
MOVLW   LS_PROG_EE_ADDR         ;
MOVWF   EEADR                   ; LS byte of Program Address to read
BSF     STATUS, RP0             ; bank 3
BSF     EECON1, EEPGD   ; point to PROGRAM memory
```

```
BSF     EECON1, RD      ;  EE Read
  ;
NOP
NOP                     ; Any instruction here are ignored as program
                        ; Memory is read in second cycle after BSF EECON1, RD
  ;
BCF     STATUS, RP0     ; bank 2
MOVF    EEDATA, W       ; W= LS byte of program EEDATA
MOVWF   DATAL           ;
MOVW    EEDATH, W       ; W= MS byte of program EEDATA
MOWF    DATAH           ;
```

f. Ghi dữ liệu vào bộ nhớ chương trình Flash:

Bộ nhớ chương trình Flash chỉ cho phép ghi nếu ô nhớ không có bảo vệ chống ghi, khi được xác định ở các bit WRT1:WRT0 của từ định cấu hình của thiết bị. Bộ nhớ chương trình Flash phải được ghi mỗi lần 1 khối gồm có 4 word. Một khối gồm 4 word có địa chỉ liên tục và dùng địa chỉ thấp nhất làm địa chỉ cho cả khối, các bit EEADR<1:0> = 00. Cùng lúc đó, tất cả các khối ghi vào bộ nhớ chương trình được thực hiện như các hoạt động ghi và xóa.

Để ghi dữ liệu vào bộ nhớ chương trình thì trước tiên ta phải nạp dữ liệu vào các thanh ghi đệm như hình. Điều này được thực hiện hoàn tất chỉ khi địa chỉ vào hai thanh ghi EEADR và EEADRH trước và sau đó mới ghi dữ liệu vào EEDATA và EEDATH. Sau khi địa chỉ và dữ liệu được thiết lập thì trình tự ghi được thực hiện theo thứ tự như sau:

- Set bit điều khiển EEPGD (EECON1<7>).
- Ghi 55H, sau đó AAH vào thanh ghi EECON2.
- Set bit điều khiển WR (EECON1<1>).

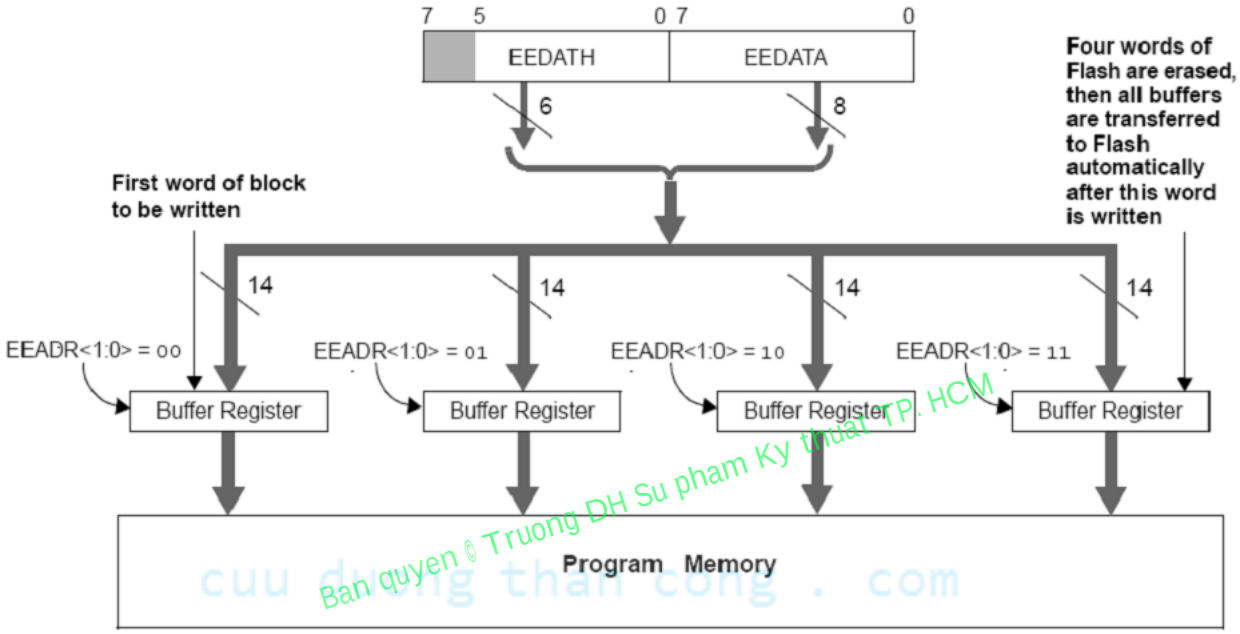
Tất cả bốn thanh ghi đệm PHẢI được ghi đúng dữ liệu. Nếu chỉ 1, 2 hoặc 3 word và giữ nguyên các word còn lại trong bộ nhớ thì ta phải thực hiện đọc nội dung các ô nhớ muốn giữ nguyên dữ liệu ra và lưu vào trong hai thanh ghi EEDATA và EEDATH. Tiếp theo tải dữ liệu vào các thanh ghi đệm và thực hiện giống như ghi đầy đủ 4 word.

Để chuyển dữ liệu từ các thanh ghi đệm vào bộ nhớ chương trình thì thanh ghi EEADR và EEADRH trỏ đến vị trí cuối cùng trong khối 4 word (EEADR<1:0> = 11). Sau đó phải thực hiện các bước sau một cách tuần tự:

- Set bit điều khiển EEPGD (EECON1<7>).
- Ghi 55h, sau đó AAh vào EECON2.
- Set bit điều khiển WR (EECON1<1>).

Người sử dụng phải thực hiện theo đúng các trình tự để khởi động quá trình ghi cho mỗi word trong khối bộ nhớ chương trình, ghi từng word theo tuần tự (00,01,10,11). Khi ghi word cuối cùng (EEADR<1:0> = 11) thì khối dữ liệu 4 word sẽ tự động bị xóa và nội dung của thanh ghi đệm đã được ghi vào bộ nhớ chương trình.

Sau khi thực hiện lệnh "*BSF EECON1, WR*" thì bộ xử lí cần hai chu kì để thiết lập hoạt động xóa/ghi. Người sử dụng phải đặt hai lệnh NOP sau khi bit WR được set. Khi dữ liệu đang ghi vào các thanh ghi đệm thì quá trình ghi 3 word đầu tiên của khối thực thi ngay lập tức. Bộ xử lí sẽ tạm ngừng các hoạt động bên trong khoảng 4ms, chỉ xảy ra trong chu kì xóa. Đây không phải là chế độ SLEEP cũng như xung đồng hồ và thiết bị ngoại vi vẫn tiếp tục hoạt động. Sau chu kì ghi, bộ xử lí sẽ hồi phục lại hoạt động ngay lệnh thứ 3 nằm sau lệnh ghi EECON1. Nếu trình tự thực hiện cho bất kì ô nhớ khác thì hoạt động sẽ bị hủy.



Hình 2-8. Ghi dữ liệu khối vào bộ nhớ chương trình flash.

Ví dụ 2-5 sẽ trình bày trình tự ghi dữ liệu 4 word vào bộ nhớ. Địa chỉ bắt đầu được nạp vào cặp thanh ghi EEADRH:EEADR, 4 word dữ liệu được nạp dùng kiểu truy xuất địa chỉ gián tiếp.

```
; This write routine assumes the following:
;
; 1. A valid starting address (the least significant bits = '00')is loaded in ADDRH:ADDRL
; 2. The 8 bytes of data are loaded, starting at the address in DATADDR
; 3. ADDRH, ADDRL and DATADDR are all located in shared data memory 0x70 - 0x7f
;
        BSF     STATUS,RP1          ;
        BCF     STATUS,RP0          ; Bank 2
        MOVF    ADDRH,W             ; Load initial address
        MOVWF   EEADRH              ;
        MOVF    ADDRL,W             ;
        MOVWF   EEADR               ;
        MOVF    DATAADDR,W          ; Load initial data address
        MOVWF   FSR                 ;
LOOP    MOVF    INDF,W              ; Load first data byte into lower
        MOVWF   EEDATA              ;
        INCF    FSR,F               ; Next byte
        MOVF    INDF,W              ; Load second data byte into upper
        MOVWF   EEDATH              ;
        INCF    FSR,F               ;
        BSF     STATUS,RP0          ; Bank 3
        BSF     EECON1,EEPGD        ; Point to program memory
        BSF     EECON1,WREN         ; Enable writes
        BCF     INTCON,GIE          ; Disable interrupts (if using)
```

```
Required Sequence
        MOVLW   55h             ; Start of required write sequence:
        MOVWF   EECON2          ; Write 55h
        MOVLW   AAh             ;
        MOVWF   EECON2          ; Write AAh
        BSF     EECON1,WR       ; Set WR bit to begin write
        NOP                     ; Any instructions here are ignored as processor
                                ; halts to begin write sequence
        NOP                     ; processor will stop here and wait for write complete
                                ; after write processor continues with 3rd instruction
        BCF     EECON1,WREN     ; Disable writes
        BSF     INTCON,GIE      ; Enable interrupts (if using)
        BCF     STATUS,RP0      ; Bank 2
        INCF    EEADR,F         ; Increment address
        MOVF    EEADR,W         ; Check if lower two bits of address are '00'
        ANDLW   0x03            ; Indicates when four words have been programmed
        XORLW   0x03            ;
        BTFSC   STATUS,Z        ; Exit if more than four words,
        GOTO    LOOP            ; Continue if less than four words
```

g. Bảo vệ chống ghi nhầm:

Có những trường hợp vi điều khiển không cho ghi dữ liệu vào bộ nhớ EEPROM hoặc bộ nhớ chương trình Flash. Để bảo vệ việc chống ghi nhầm thì có nhiều kĩ thuật khác nhau được thiết kế. Khi mở nguồn thì bit WREN bị xóa. Do đó bộ định thời khi mở điện sau khoảng 72ms sẽ ngăn chặn ghi dữ liệu vào EEPROM.

Trình tự khởi động ghi và bit WREN cùng hỗ trợ để bảo vệ ghi ngẫu nhiên trong suốt khoảng thời gian nguồn điện bị sụt áp do quá tải bất thường, nguồn xung tạp nhiễu hoặc sự cố phần mềm.

h. Hoạt động trong lúc bảo vệ chống ghi:

Khi bộ nhớ dữ liệu EEPROM có mã bảo vệ thì vi điều khiển có thể đọc và ghi vào EEPROM một cách bình thường. Tuy nhiên tất cả các truy xuất từ bên ngoài vào bộ nhớ EEPROM thì không được cho phép.

Khi bộ nhớ chương trình EEPROM có mã bảo vệ, vi điều khiển có thể đọc và ghi vào bộ nhớ chương trình một cách bình thường cũng như thực hiện các lệnh. Tuy nhiên tất cả các truy xuất từ bên ngoài vào bộ nhớ EEPROM thì không được cho phép.

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on Power-on Reset	Value on all other Resets
10Ch	EEDATA	EEPROM/Flash Data Register Low Byte								xxxx xxxx	uuuu uuuu
10Dh	EEADR	EEPROM/Flash Address Register Low Byte								xxxx xxxx	uuuu uuuu
10Eh	EEDATH	—	—	EEPROM/Flash Data Register High Byte						xxxx xxxx	---0 q000
10Fh	EEADRH	—	—	—	EEPROM/Flash Address Register High Byte					xxxx xxxx	---- ----
18Ch	EECON1	EEPGD	—	—	—	WRERR	WREN	WR	RD	x--- x000	---0 q000
18Dh	EECON2	EEPROM Control Register 2 (not a physical register)								---- ----	---- ----
0Dh	PIR2	—	CMIF	—	EEIF	BCLIF	—	—	CCP2IF	-0-0 0--0	-0-0 0--0
8Dh	PIE2	—	CMIE	—	EEIE	BCLIE	—	—	CCP2IE	-0-0 0--0	-0-0 0--0

Legend: x = unknown, u = unchanged, - = unimplemented, read as '0', q = value depends upon condition.
Shaded cells are not used by data EEPROM or Flash program memory.

Bảng 2-8. Các thanh ghi sử dụng cho bộ nhớ EEPROM.

4. CÁC PORT XUẤT NHẬP (IO):

a. PORTA và thanh ghi TRISA:

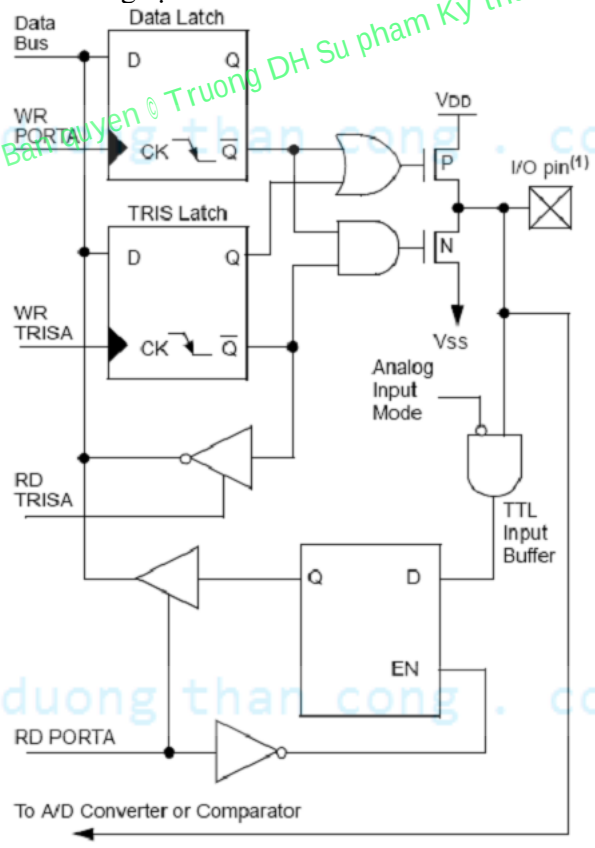
PORTA là port hai chiều chỉ có 6 bit, thanh ghi định hướng dữ liệu tương ứng là TRISA. Khi bit TRISA bằng 1 thì PORTA là port nhập và khi bit TRISA bằng 0 thì PORTA là port xuất dữ liệu.

Đọc thanh ghi PORTA là đọc trạng thái ở các chân, nhưng ngược lại khi ghi thì dữ liệu sẽ vào mạch chốt port. Tất cả hoạt động ghi gồm 3 giai đoạn: đọc – hiệu chỉnh – ghi. Do đó ghi dữ liệu vào 1 Port được hiểu ngầm là đọc dữ liệu từ port rồi hiệu chỉnh và sau cùng là ghi dữ liệu vào mạch chốt dữ liệu.

Chân RA4 được đa hợp với ngõ vào xung clock của module Timer0 có tên là RA4/T0CKI – có cấu hình Schmitt trigger và cực máng để hở. Tất cả các chân còn lại của PORTA ở chuẩn TTL khi nó là ngõ vào và khi xuất dữ liệu thì theo chuẩn CMOS.

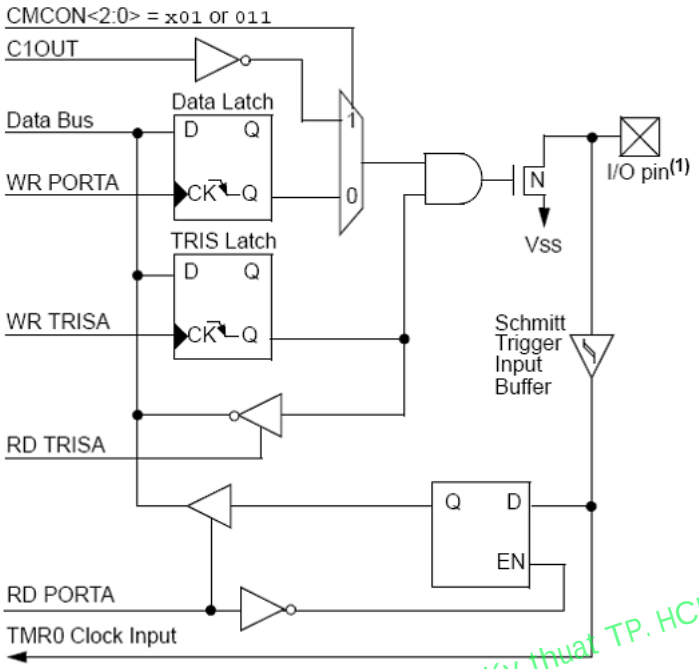
Những chân khác của PORTA được đa hợp với các ngõ vào tương tự và ngõ vào tương tự V_{REF} cho các bộ chuyển đổi A/D và các bộ so sánh. Hoạt động của mỗi chân được lựa chọn bằng cách xoá/lập các bit điều khiển cho phù hợp trong thanh ghi ADCON1 và/hoặc thanh ghi CMCON.

Thanh ghi TRISA điều khiển hướng các chân của Port ngay cả khi chúng được sử dụng như là ngõ vào tương tự. Người sử dụng phải đảm bảo rằng các bit ở thanh ghi TRISA được duy trì ở mức 1 khi sử dụng chúng là ngõ vào tương tự.



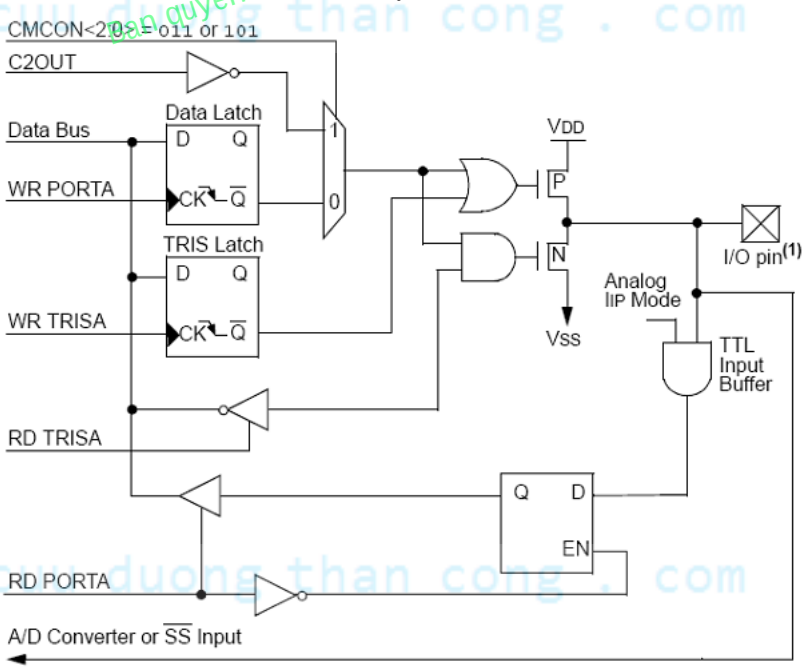
Note 1: I/O pins have protection diodes to VDD and VSS.

Hình 2-9. Sơ đồ mạch chân RA3:RA0.



Note 1: I/O pin has protection diodes to Vss only.

Hình 2-10. Sơ đồ mạch chân RA4/T0CKI.



Hình 2-11. Sơ đồ mạch chân RA5.

TÊN	BIT#	KIỂU ĐỆM	CHỨC NĂNG
RA0/AN0	Bit 0	TTL	I/O
RA1/AN1	Bit 1	TTL	I/O

RA2/AN2/V_{REF-}/CV_{REF}	Bit 2	TTL	I/O hoặc V_{REF-} hoặc VC_{REF}
RA3/AN3/V_{REF+}	Bit 3	TTL	I/O hoặc V_{REF+}
RA4/TOCKI/C1OUT	Bit 4	TTL	I/O hoặc ngõ vào xung clock cho Timer0 hoặc ngõ ra bộ so sánh
RA5/AN4/ $\overline{SS}$ /C2OUT	Bit 5	TTL	I/O hoặc ngõ vào tương tự

Bảng 2-9. Các chức năng của PORTA.

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other Resets
05h	PORTA	—	—	RA5	RA4	RA3	RA2	RA1	RA0	--0x 0000	--0u 0000
85h	TRISA	—	—	PORTA Data Direction Register						--11 1111	--11 1111
9Ch	CMCON	C2OUT	C1OUT	C2INV	C1INV	CIS	CM2	CM1	CM0	0000 0111	0000 0111
9Dh	CVRCON	CVREN	CVROE	CVRR	—	CVR3	CVR2	CVR1	CVR0	000- 0000	000- 0000
9Fh	ADCON1	ADFM	ADCS2	—	—	PCFG3	PCFG2	PCFG1	PCFG0	00-- 0000	00-- 0000

Bảng 2-10. Tóm tắt các thanh ghi liên kết với PORTA.

b. PORTB và thanh ghi TRISB:

PORTB là port hai chiều 8 bit. Thanh ghi định hướng là TRISB. Khi bit TRISB = 1 thì PORTB là port nhập, khi TRISB = 0 thì PORTB là port xuất.

Ba chân của PORTB được đa hợp với mạch điện gỡ rối bên trong và chức năng lập trình điện áp thấp: RB3/PGM, RB6/PGC và RB7/PGD.

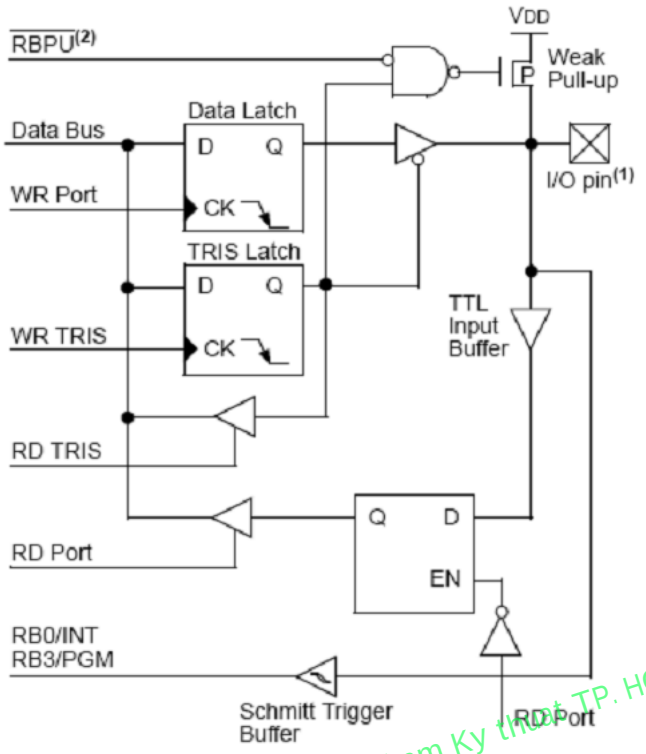
Mỗi chân của PORTB có điện trở kéo lên. Bit điều khiển $\overline{RBPU}$ (OPTION_REG<7>) = 0 thì có thể mở tất cả các điện trở kéo lên. Khi PORTB được thiết lập là các ngõ ra thì sẽ tự động tắt chức năng điện trở kéo lên cũng tương tự khi CPU bị reset lúc mới cấp điện.

Bốn chân của PORTB RB4:RB7 có cấu trúc ngắt thay đổi. Chỉ có những chân được thiết lập ở cấu hình là ngõ vào thì mới có chức năng ngắt. Các chân ngõ vào (RB7:RB4) được so sánh với giá trị cũ đã được chốt trong lần đọc trước của PORTB. Các ngõ ra không trùng nhau của các chân RB4:RB7 được OR lại với nhau để tạo ngắt ở PORTB với bit cờ báo ngắt RBIF (INTCON<0>).

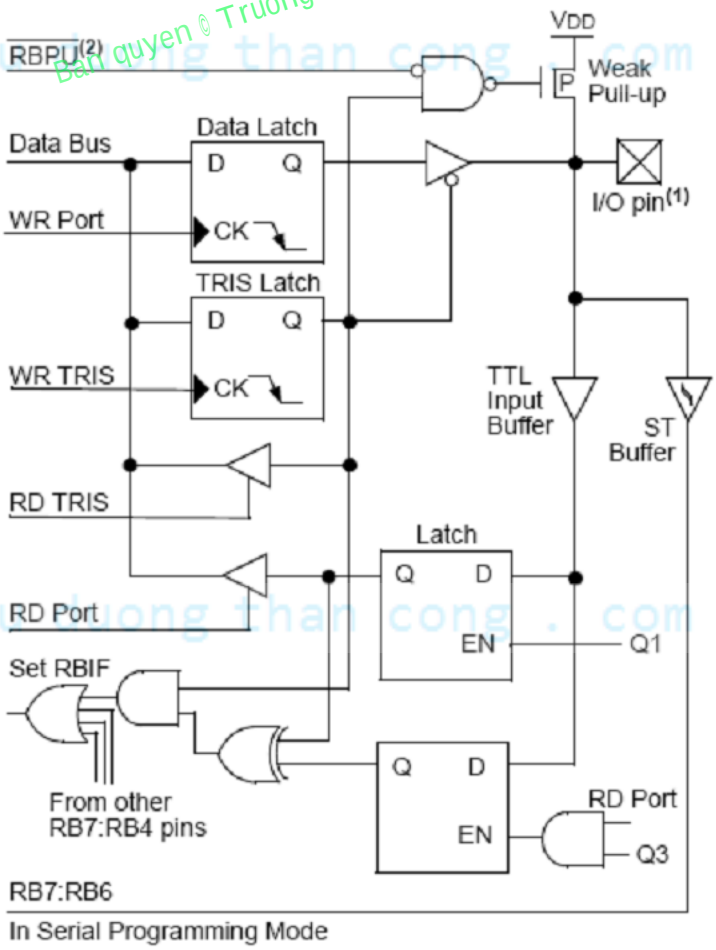
Ngắt này có thể kích hoạt vi điều khiển trở lại trạng thái hoạt động khi nó đang ở chế độ SLEEP. Trong chương trình phục vụ ngắt thì người dùng có thể xóa ngắt bằng các cách sau:

- Bất kỳ lệnh đọc hay ghi PORTB sẽ kết thúc điều kiện không thích ứng.
- Xóa bit cờ RBIF.

Điều kiện không tương thích sẽ tiếp tục làm cờ báo ngắt RBIF bằng 1. Khi đọc PORTB sẽ chấm dứt điều kiện không tương thích và cho phép xóa bit cờ báo ngắt RBIF.



Hình 2-12. Sơ đồ mạch các chân RB3:RB0.



Hình 2-13. Sơ đồ mạch các chân RB7:RB4.

Cấu trúc ngắt thay đổi dùng để thoát khỏi chế độ nghỉ khi có nhấn phím và các hoạt động mà PORTB chỉ được dùng cho cấu trúc thay đổi ngắt.

Cấu trúc ngắt không tương thích kết hợp với 4 chân có cấu hình điện trở kéo lên bằng phần mềm dễ dàng cho phép giao tiếp với bàn phím.

Tên	Bit#	Kiểu đệm	Chức năng
RB0/INT	Bit 0	TTL/ST	I/O hoặc ngõ vào ngắt. Có lập trình điện trở kéo lên.
RB1	Bit 1	TTL	I/O. Có lập trình điện trở kéo lên.
RB2	Bit 2	TTL	I/O. Có lập trình điện trở kéo lên.
RB3/PGM	Bit 3	TTL	I/O hoặc lập trình ở chế độ LVP. Có lập trình điện trở kéo lên.
RB4	Bit 4	TTL	I/O (ngắt khi có thay đổi). Có lập trình điện trở kéo lên.
RB5	Bit 5	TTL	I/O (ngắt khi có thay đổi). Có lập trình điện trở kéo lên.
RB6/PGC	Bit 5	TTL/ST	I/O (ngắt khi có thay đổi) hoặc chân mạch gỡ rối. Có lập trình điện trở kéo lên. Xung lập trình nối tiếp.
RB7/PGD	Bit 5	TTL/ST	I/O (ngắt khi có thay đổi) hoặc chân mạch gỡ rối. Có lập trình điện trở kéo lên. Dữ liệu lập trình nối tiếp.

Bảng 2-11. Các chức năng của PORTB.

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other Resets
06h, 106h	PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0	xxxx xxxx	uuuu uuuu
86h, 186h	TRISB	PORTB Data Direction Register								1111 1111	1111 1111
81h, 181h	OPTION_REG	RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	1111 1111

Bảng 2-12. Các thanh ghi kết nối với PORTB.

c. PORTC và thanh ghi TRISC:

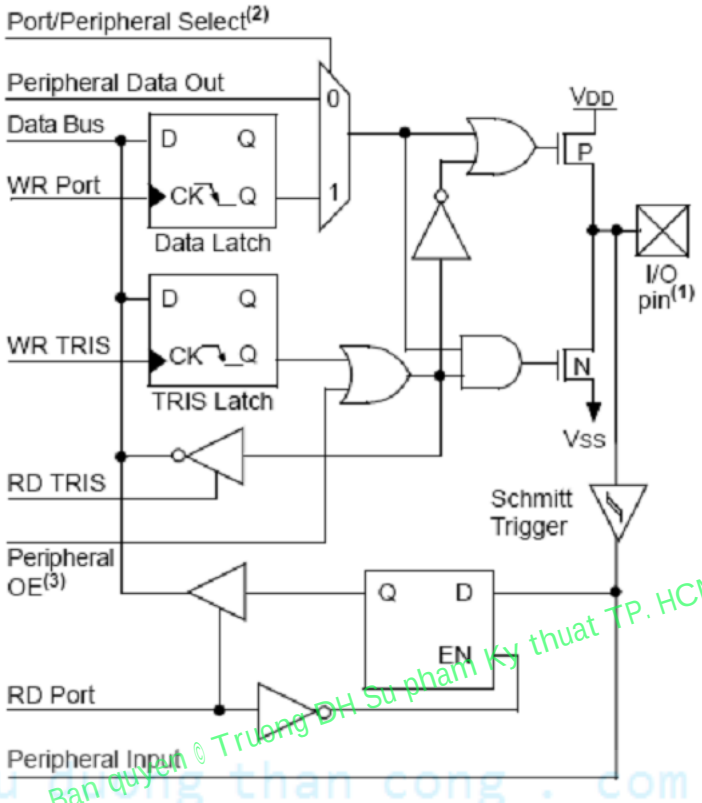
PORTC là port hai chiều 8 bit. Thanh ghi định hướng là TRISC. Khi bit TRISC =1 thì PORTC là port nhập, khi bit TRISC= 0 thì PORTC là port xuất.

PORC là được đa hợp với vài chức năng ngoại vi. Các chân của PORTC có mạch đệm Schmitt Trigger ở ngõ vào.

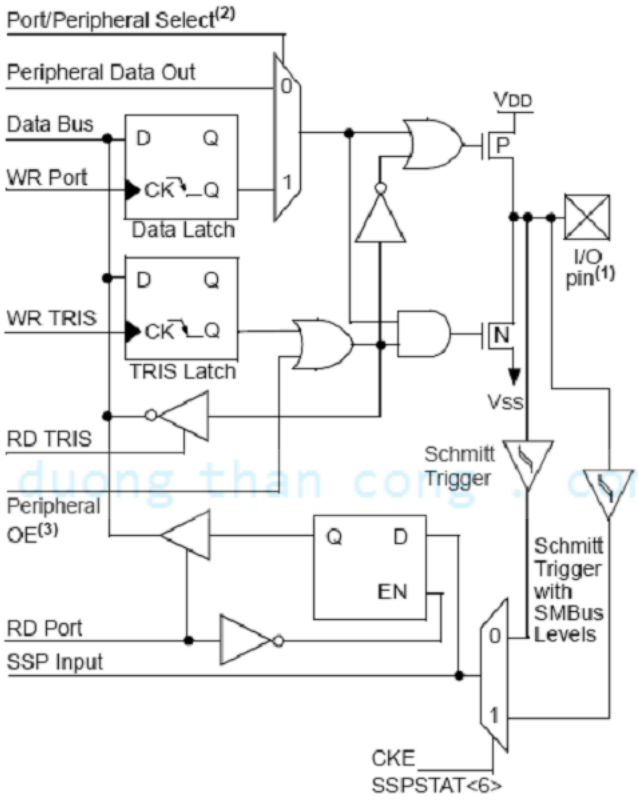
Khi khối I^2C được cho phép thì các chân PORTC<4:3> có thể được định cấu hình ở các mức I^2C hoặc mức SMBUS bằng cách sử dụng bit CKE (SSPSTAT<6>).

Khi cho phép các chức năng ngoại VI, nên chủ ý đến các bit TRIS cho mỗi chân của PORTC. Một vài thiết bị ngoại vi ghi neon lên bit TRIS để làm một chân như là 1 ngõ ra, trong khi đó các thiết bị ngoại vi khác ghi neon lên bit TRISB để làm một chân như là 1 ngõ vào. Khi ghi đè bit TRIS thì không ảnh hưởng đến các thiết bị đã cho phép, các lệnh đọc – hiệu chỉnh – ghi (BSF,

BCF, XORWF) với TRISC là đích đến phải tránh dùng. Người sử dụng tham chiếu tới phần thiết bị ngoại vi tương ứng để thiết lập cho đúng bit TRIS.



Hình 2-14. Sơ đồ mạch các chân RC7:RB5 và RC2:RB0.



Hình 2-15. Sơ đồ mạch các chân RC4:RB3.

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other Resets
07h	PORTC	RC7	RC6	RC5	RC4	RC3	RC2	RC1	RC0	xxxx xxxx	uuuu uuuu
87h	TRISC	PORTC Data Direction Register								1111 1111	1111 1111

Legend: x = unknown, u = unchanged

Bảng 2-13. Các thanh ghi kết nối với PORTB.

TÊN	BIT#	KIỂU ĐỆM	CHỨC NĂNG
RC0/T1OSO/T1CKI	Bit 0	ST	I/O hoặc ngõ ra bộ dao động Timer1/ngõ vào xung của Timer1.
RC1/T1OSI/CCP2	Bit 1	ST	I/O hoặc ngõ vào bộ dao động Timer1/ngõ vào Capture, ngõ ra compare2/ngõ ra PWM.
RC2/CCP1	Bit 2	ST	I/O hoặc ngõ vào Capture1/ngõ ra Compare1/ngõ ra PWM.
RC3/SCK/SCL	Bit 3	ST	RC3 cũng có thể là xung clock nối tiếp đồng bộ cho chế độ SPI và I^2C.
RC4/SDI/SDA	Bit 4	ST	RC4 cũng có thể là dữ liệu SPI hoặc dữ liệu xuất/nhập (chế độ I^2C).
RC5/SDO	Bit 5	ST	I/O hoặc ngõ ra dữ liệu port nối tiếp đồng bộ.
RC6/TX/CK	Bit 6	ST	I/O hoặc truyền bất đồng bộ USART hoặc xung đồng bộ.
RC7/RX/DT	Bit 7	ST	I/O hoặc nhận bất đồng bộ USART hoặc dữ liệu đồng bộ.

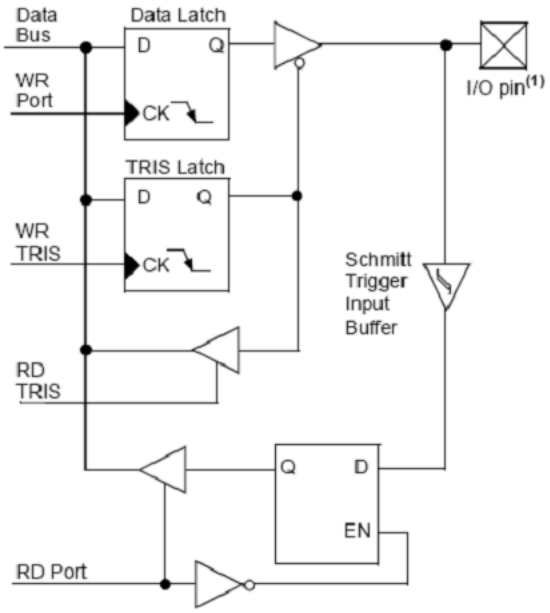
Bảng 2-14. Các chức năng của PORTC.

d. PORTD và thanh ghi TRISD:

PORTD là port 8 bit với ngõ vào có mạch Schmitt Trigger. Mỗi chân có thể được cấu hình độc lập là ngõ vào hoặc ngõ ra.

PORTD có thể định cấu hình như port của vi xử lý 8 bit bằng cách thiết lập bit điều khiển PSPMODE (TRISE<4>). Trong mode này thì các bộ đệm ngõ vào dạng TTL.

Chú ý: PORTD và TRISD không được xây dựng cho chip PIC 28 chân.



Hình 2-16. Sơ đồ mạch các chân PORTD.

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other Resets
08h	PORTD	RD7	RD6	RD5	RD4	RD3	RD2	RD1	RD0	xxxx xxxx	uuuu uuuu
88h	TRISD	PORTD Data Direction Register								1111 1111	1111 1111
89h	TRISE	IBF	OBF	IBOV	PSPMODE	—	PORTE Data Direction Bits			0000 -111	0000 -111

Bảng 2-15. Các thanh ghi kết nối với PORTD.

TÊN	BIT#	KIỂU ĐỆM	CHỨC NĂNG
RD0/PSP0	Bit 0	ST/TTL	Port I/O hoặc bit 0 của port tớ song song
RD1/PSP1	Bit 1	ST/TTL	Port I/O hoặc bit 1 của port tớ song song
RD2/PSP2	Bit 2	ST/TTL	Port I/O hoặc bit 2 của port tớ song song
RD3/PSP3	Bit 3	ST/TTL	Port I/O hoặc bit 3 của port tớ song song
RD4/PSP4	Bit 4	ST/TTL	Port I/O hoặc bit 4 của port tớ song song
RD5/PSP5	Bit 5	ST/TTL	Port I/O hoặc bit 5 của port tớ song song
RD6/PSP6	Bit 6	ST/TTL	Port I/O hoặc bit 6 của port tớ song song
RD7/PSP7	Bit 7	ST/TTL	Port I/O hoặc bit 7 của port tớ song song

Bảng 2-16. Các chức năng của PORTD.

e. PORTE và thanh ghi TRISE:

PORTE có 3 chân: RE0/$\overline{RD}$/AN5, RE1/$\overline{WR}$/AN6 và RE2/$\overline{CS}$/AN7 có cấu hình độc lập để thiết lập ngõ vào hoặc ngõ ra. Những chân này có mạch điện Schmitt Trigger ở ngõ vào.

PORTE trở thành các ngõ vào điều khiển xuất/nhập của vi xử lí khi bit PSPMODE (TRISE<4>) bằng 1. Trong chế độ này người sử dụng phải chắc chắn rằng các bit TRISE<0:2> phải bằng 11 và chắc chắn rằng các chân đó được thiết lập là các ngõ vào số. Cũng đảm bảo rằng

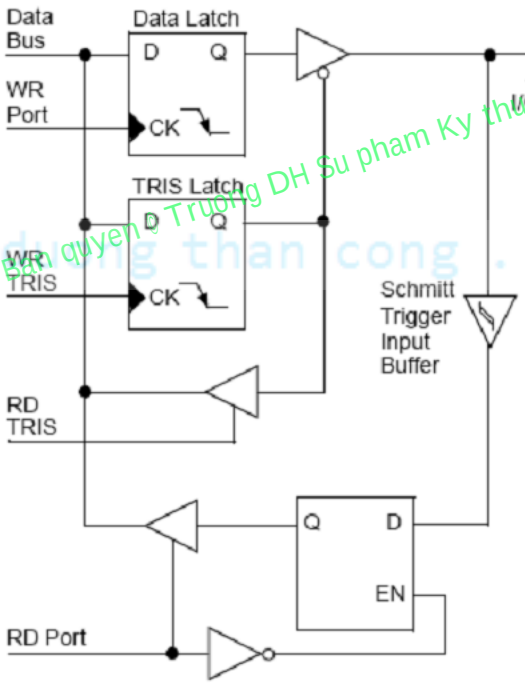
ADCON1 được cấu hình như những ngõ xuất/nhập số. Trong chế độ này, bộ đệm ngõ vào dạng TTL.

Các chân ở PORTE cũng được đa hợp với các ngõ vào tương tự. Khi được chọn là ngõ vào tương tự thì khi đọc các chân này sẽ có giá trị là '0'.

TRISE điều khiển định hướng các chân RE, ngay cả khi chúng được dùng như những ngõ vào tương tự. Người dùng phải chắc chắn các chân được định cấu hình là những ngõ vào khi dùng chúng là những ngõ vào tương tự.

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other Resets
09h	PORTE	—	—	—	—	—	RE2	RE1	RE0	---- -xxx	---- -uuu
89h	TRISE	IBF	OBF	IBOV	PSPMODE	—	PORTE Data Direction bits			0000 -111	0000 -111
9Fh	ADCON1	ADFM	ADCS2	—	—	PCFG3	PCFG2	PCFG1	PCFG0	00-- 0000	00-- 0000

Bảng 2-17. Các thanh ghi kết nối với PORTE.



Hình 2-17. Sơ đồ mạch các chân PORTE.

TÊN	BIT#	KIỂU ĐỆM	CHỨC NĂNG
RE0/$\overline{RD}$/AN5	Bit 0	ST/TTL	I/O hoặc ngõ vào điều khiển đọc trong chế độ port tớ song song hoặc ngõ vào tương tự: $\overline{RD}$ 1= bình thường 0= điều khiển đọc.
RE1/$\overline{WR}$/AN6	Bit 1	ST/TTL	I/O hoặc ngõ vào điều khiển ghi trong chế độ port tớ song song hoặc ngõ vào tương tự: $\overline{WR}$ 1= bình thường.

			0= điều khiển ghi.
RE2/$\overline{CS}$ /AN7	Bit 2	ST/TTL	I/O hoặc ngõ vào điều khiển chọn lựa chip trong chế độ port tớ song song hoặc ngõ tương tự: $\overline{CS}$ 1= VĐK không được chọn. 0= VĐK được chọn.

Bảng 2-18. Các chức năng của PORTE.

TRISE REGISTER (ADDRESS 89h)

R-0	R-0	R/W-0	R/W-0	U-0	R/W-1	R/W-1	R/W-1
IBF	OBF	IBOV	PSPMODE	—	Bit 2	Bit 1	Bit 0
bit 7							bit 0

Trạng thái port tớ song song/các bit điều khiển:

Bit 7 **IBF**: bit báo trạng thái bộ đệm ngõ vào đầy (Input Buffer Full Status bit):

1= một word đã nhận và đang chờ CPU đọc.

0= không có word nào được nhận.

Bit 6 **OBF**: bit báo trạng thái bộ đệm ngõ ra đầy (Output Buffer Full Status bit):

1= bộ đệm ngõ ra vẫn còn giữ word đã ghi trước đó.

0= bộ đệm ngõ ra đã được đọc.

Bit 5 **IBOV**: bit phát hiện tràn bộ đệm ngõ vào (Input Buffer Overflow Detect bit):

1= quá trình ghi xảy khi word ngõ vào trước đó chưa được đọc.

0= không xảy ra tràn.

Bit 4 **PSPMODE**: Bit chọn lựa chế độ port tớ song song

1= PORTD được định ở chế độ là port tớ song song.

0= PORTD được định ở chế độ là port xuất nhập.

Bit 3 Chưa dùng: đọc là '0'

Các bit ở PORTE là các bit dữ liệu trực tiếp:

Bit 2 Bit 2: bit điều khiển hướng cho chân RE2/$\overline{CS}$ /AN7

1= ngõ vào.

0= ngõ ra.

Bit 1 Bit 1: bit điều khiển hướng cho chân RE1/$\overline{WR}$ /AN6

1= ngõ vào.

0= ngõ ra.

Bit 0 Bit 0: bit điều khiển hướng cho chân RE0/$\overline{RD}$ /AN5

1= ngõ vào.

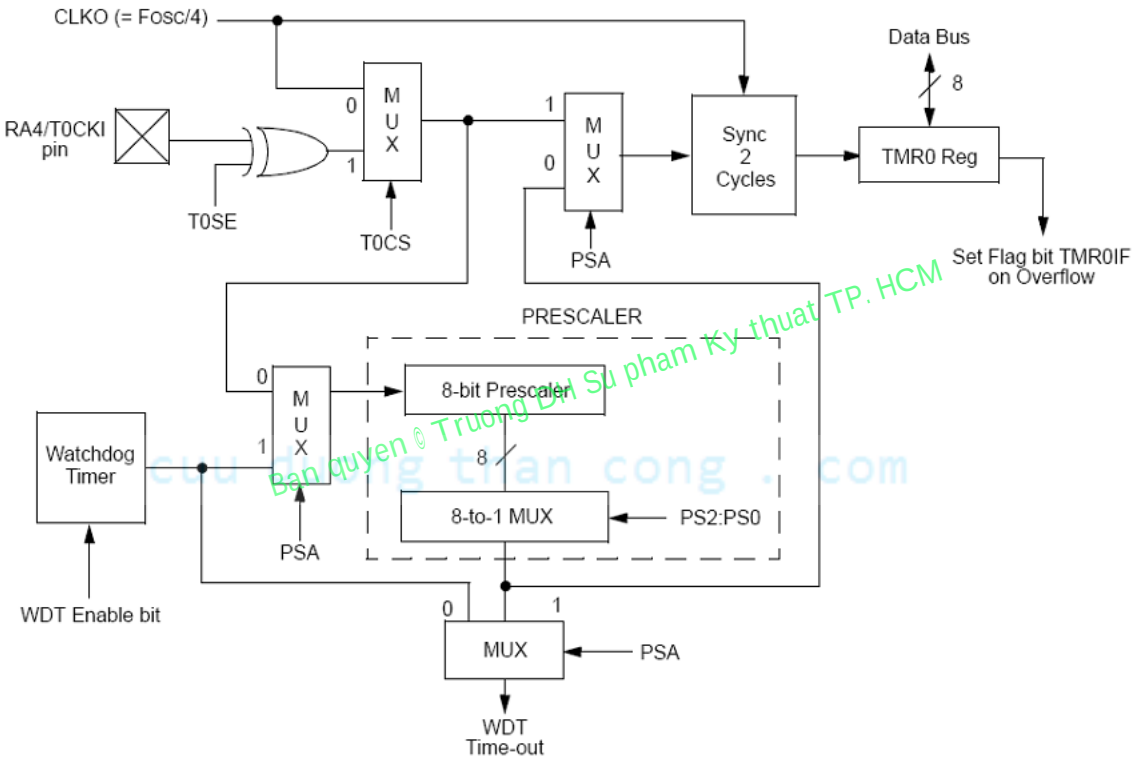
0= ngõ ra.

5. BỘ ĐỊNH THỜI TIMER0:

Bộ timer/counter của module Timer có những đặc điểm sau:

- Là timer/counter 8 bit.
- Có thể đọc và ghi.
- Có bộ chia trước 8 bit có thể lập trình bằng phần mềm.
- Cho phép lựa chọn nguồn xung clock bên trong hoặc bên ngoài.
- Phát sinh ngắt khi bị tràn từ FFH đến 00H.
- Cho phép lựa chọn tác động cạnh cho xung clock bên ngoài.

Sơ đồ khối của timer0 và bộ chia trước với WDT như hình 2-18:



Hình 2-18. Sơ đồ khối của timer0 và bộ chia trước với WDT.

Chế độ định thời được chọn bởi việc xóa bit TOCS (OPTION_REG<5>). Trong chế độ định thời Timer thì Timer0 sẽ tăng giá trị sau mỗi chu kì lệnh (không có bộ chia). Nếu thanh ghi TMR0 bị ghi giá trị mới thì quá trình tăng sẽ bị cấm trong 2 chu kì theo sau. Người sử dụng có thể làm tròn giá trị này bằng cách ghi giá trị có điều chỉnh vào thanh ghi TMR0.

Nếu bit TOCS (OPTION_REG<5>) bằng 1 thì chọn chế độ đếm Counter. Trong chế độ đếm Counter thì Timer0 sẽ tăng giá trị đếm mỗi khi có xung tác động cạnh lên hoặc cạnh xuống ở chân RA4/TOCKI. Cạnh tác động của xung được chọn lựa bởi bit T0SE (OPTION_REG<4>): T0SE = 0 thì chọn cạnh lên, ngược lại thì chọn cạnh xuống.

Bộ chia trước không thể đọc/ghi và có mối quan hệ với Timer0 và Watchdog Timer.

a. Ngắt của Timer0:

Ngắt TMR0 được kích hoạt khi thanh ghi TMR0 tràn từ 00h đến FFh. Khi tràn xảy ra sẽ làm bit TMR0IF (INTCON<2>) lên mức 1. Ngắt có thể che (cấm) bằng cách xóa bit TMR0IE

(INTCON<5>). Bit TMR0IF phải được xóa bằng phần mềm bởi chương trình con phục vụ ngắt của Timer0 trước khi cho phép ngắt trở lại. Ngắt TMR0 không thể kích vi xử lí thoát khỏi chế độ SLEEP vì bộ định thời sẽ ngừng khi vi xử lý ở chế độ SLEEP.

b. Timer0 với nguồn xung đếm từ bên ngoài:

Khi không sử dụng bộ chia trước thì ngõ vào xung clock bên ngoài giống như ngõ ra bộ chia trước. Sự đồng bộ hóa của TOCKI với các xung clock bên trong được thực hiện bằng cách lấy mẫu ngõ ra bộ chia ở những chu kì Q2 và Q4 của xung clock bên trong. Do đó, nó rất cần thiết cho T0CKI ở trạng thái mức cao ít nhất 2 T_{OSC} và ở trạng thái mức thấp ít nhất 2 T_{OSC}.

c. Bộ chia trước:

Chỉ có một bộ chia có tác dụng mà nó có quan hệ với Timer0 và WDT. Khi gán bộ chia trước cho Timer0 thì sẽ không có bộ chia cho Watchdog Timer và ngược lại. Bộ chia trước thì không thể đọc hoặc ghi. Các bit PSA và PS2:PS0 (OPTION_REG<3:0>) xác định gán của bộ chia và tỉ lệ chia.

Khi được gán cho Timer0 thì tất cả các lệnh ghi cho thanh ghi TMR0 (ví dụ CLRF 1, MOVWF 1, BSF 1, ...) sẽ xoá bộ chia trước.

Khi được gán cho WDT thì lệnh CLRWDT sẽ xoá bộ chia trước cùng với Watchdog Timer. Bộ chia trước cũng không thể đọc hoặc ghi.

OPTION_REG REGISTER

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

Bit 7	*$\overline{RBPU}$*
Bit 6	**INTEDG**
Bit 5	**T0CS**: bit lựa chọn nguồn xung cho TMR0 (TMR0 Clock Source Select bit)
	1= xung đưa đến chân T0CKI.
	0= xung clock bên trong.
Bit 4	**T0SE**: bit lựa chọn cạnh tích cực T0SE (TMR0 Source Edge Select bit)
	1= tích cực cạnh xuống ở chân T0CKI.
	0= tích cực cạnh lên ở chân T0CKI.
Bit 3	**PSA**: bit gán bộ chia trước
	1= gán bộ chia cho WDT.
	0= gán bộ chia Timer0.
Bit 2-0	**PS2:PS0**: các bit lựa chọn tỉ lệ bộ chia trước

Bit lựa chọn	Tỉ lệ TMR0	Tỉ lệ WDT
000	1:2	1:1
001	1:4	1:2

010	1:8	1:4
011	1:16	1:8
100	1:32	1:16
101	1:64	1:32
110	1:128	1:64
111	1:256	1:128

Bảng 2-19. Các bit lựa chọn tỉ lệ bộ chia trước.

6. BỘ ĐỊNH THỜI TIMER1

Timer1 là định thời/đếm 16 bit gồm 2 thanh ghi 8 bit (TMR1H và TMR1L) – có thể đọc và ghi. Hai thanh ghi này tăng từ 0000h đến FFFFh và quay trở lại 0000h. Ngắt của Timer1 nếu được cho phép sẽ phát sinh ngắt khi tràn và ngắt được chốt vào bit cờ ngắt TMR1IF (PIR1<0>). Ngắt này có thể cho phép/cấm khi bit TMRIE (PIE1<1>) có giá trị 1/0 tương ứng.

Timer1 có thể hoạt động ở 1 trong 2 chế độ được lựa chọn bởi bit TMR1CS (T1CON<1>):

- ☐ Bộ định thời – timer.
- ☐ Bộ đếm – counter.

Trong chế độ định thời timer, Timer1 tăng giá trị ở mỗi chu kì lệnh. Trong chế độ đếm thì bộ đếm tăng giá trị mỗi khi có cạnh của xung clock ngõ vào từ bên ngoài.

Timer1 có bit điều khiển cho phép/cấm đếm TMR1ON – bằng 1 thì cho, bằng 0 thì cấm.

Timer1 cũng có ngõ vào reset. Ngõ vào reset có thể được tạo ra bởi cả 2 khối CCP.

T1CON: TIMER1 CONTROL REGISTER (ADDRESS 10h)

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	T1CKPS1	T1CKPS0	T1OSCEN	$\overline{\text{T1SYNC}}$	TMR1CS	TMR1ON
bit 7							bit 0

Bit 7-6	Chưa sử dụng nếu đọc sẽ có giá trị '0'.
Bit 5-4	**T1CKPS1:T1CKPS1:** các bit lựa chọn bộ chia
	11=1:8 giá trị chia.
	10=1:4 giá trị chia.
	01=1:2 giá trị chia.
	00=1:1 giá trị chia.
Bit 3	**T10SCEN**: bit điều khiển cho phép bộ dao động Timer1
	1= bộ dao động được phép.
	0= Tắt bộ dao động.
Bit 2	**T1SYNC**: bit điều khiển đồng bộ ngõ vào xung clock bên ngoài của timer1
	Khi **TMR1CS = 1**
	1= không thể đồng bộ ngõ vào clock ở từ bên ngoài.
	0= đồng bộ ngõ vào clock ở từ bên ngoài.

Khi **TMR1CS = 0**

Bit này bị bỏ qua. Timer1 dùng xung clock bên trong khi **TMR1CS = 0.**

Bit 1 **TMR1CS**: bit lựa chọn nguồn xung clock của timer1

1= chọn nguồn xung clock từ bên ngoài ở chân RC0/T1OSO/T1CKI (cạnh lên).

0= Chọn nguồn xung clock bên trong ($F_{OSC}/4$).

Bit 0 **TMR1ON**: bit điều khiển Timer1

1= Cho phép Timer1 đếm.

0= Timer1 ngừng đếm.

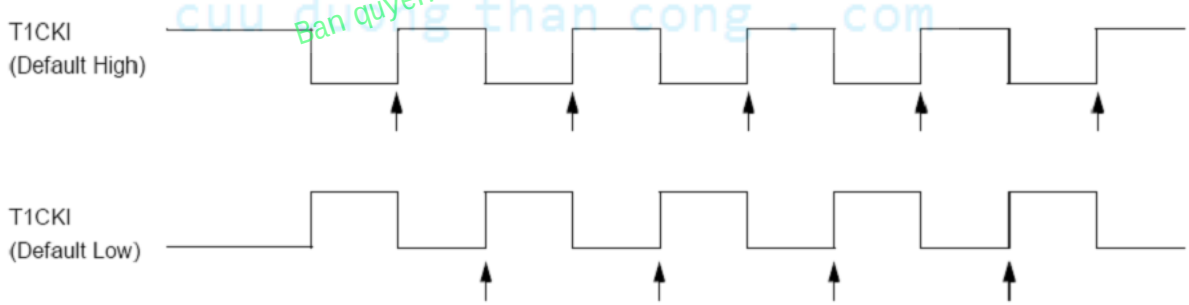
a. Hoạt động của Timer1 ở chế độ định thời:

Nếu bit TMR1CS (T1CON<1>) bằng 0 thì Timer1 sẽ làm việc ở chế độ định thời và tần số xung đồng hồ đưa đến bộ đếm bằng $F_{OSC}/4$. Bit điều khiển đồng bộ $\overline{T1SYNC}$ (T1CON<2>) không bị ảnh hưởng do xung clock bên trong luôn đồng bộ.

b. Hoạt động của Timer1 ở chế độ Counter:

Timer1 có thể hoạt động ở chế độ đồng bộ hoặc bất đồng bộ tuỳ thuộc vào bit TMR1CS.

Timer1 tăng lên 1 khi có cạnh lên của xung bên ngoài. Sau khi Timer1 được phép bắt đầu hoạt động ở chế độ Counter thì Counter phải nhận 3 xung cạnh xuống trước khi có xung đếm được minh hoạ hình 2-19.



Hình 2-19. Giản đồ thời gian xung đếm của Counter1.

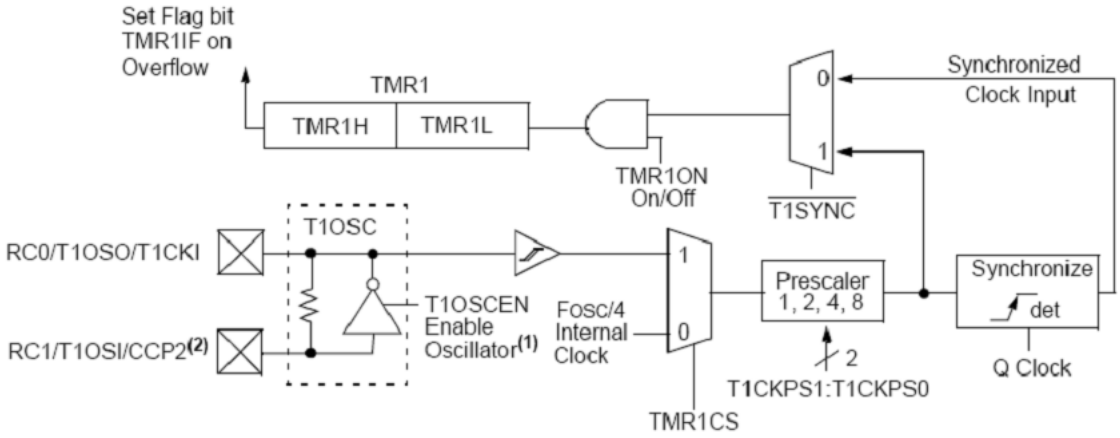
c. Hoạt động của Timer1 ở chế độ Counter đồng bộ:

Khi bit TMR1CS bằng 1 thì Timer1 hoạt động ở chế độ Counter và sẽ tăng giá trị nếu:

- Có xung cạnh lên đưa đến ngõ vào ở chân RC1/T1OSI/CCP2 và bit T1OSCEN bằng 1.
- Có xung cạnh lên đưa đến ngõ vào ở chân RC0/T1OSO/T1CKI và bit T1OSCEN bằng 0.

Nếu bit $\overline{T1SYNC}$ bị xóa thì ngõ vào xung clock từ bên ngoài được đồng bộ với xung clock bên trong. Sự đồng bộ được thực hiện sau tầng chia trước. Tầng chia trước là bộ đếm bất đồng bộ.

Ở cấu hình này, trong suốt chế độ SLEEP, Timer1 sẽ không tăng giá trị đếm khi có xung từ bên ngoài xuất hiện do mạch đồng bộ ngừng hoạt động. Tuy nhiên bộ chia vẫn tiếp tục tăng.



Hình 2-20. Sơ đồ khối của Timer1.

d. Hoạt động của Timer1 ở chế độ Counter bất đồng bộ:

Nếu bit $\overline{T1SYNC}$ (T1CON<2>) bằng 1 thì xung ngõ vào từ bên ngoài không được đồng bộ. Bộ đếm tiếp tục tăng bất đồng bộ với xung bên trong. Bộ đếm sẽ tiếp tục đếm khi ở trong chế độ SLEEP và có thể phát sinh ngắt tràn để khởi động lại bộ xử lí. Tuy nhiên, các phòng ngừa đặc biệt trong phần mềm thì cần phải đọc/ghi bộ định thời.

Ở chế độ bộ đếm không đồng bộ thì Timer1 không thể dùng bộ tạo thời gian chuẩn để bắt kịp hoặc các hoạt động so sánh.

e. Đọc và ghi Timer1 trong chế độ đếm không đồng bộ:

Đọc các thanh ghi TMR1H hoặc TMR1L trong khi timer đang hoạt động đếm xung bất đồng bộ bên ngoài thì giá trị đọc có hiệu lực.

Khi ghi thì người dùng nên ngừng timer lại rồi mới ghi giá trị mong muốn vào các thanh ghi. Nội dung ghi vào timer có thể thực hiện khi timer đang đếm nhưng có thể tạo ra một giá trị đếm không dự đoán được ở các thanh ghi của timer.

f. Bộ dao động của Timer1:

Mạch dao động thạch anh được thiết kế và tích hợp bên trong giữa các chân T1OSI (ngõ vào) và T1OSO (ngõ ra có khuếch đại). Bộ dao động được phép hoạt động bằng cách làm bit điều khiển T1OSCEN (T1CON<3>) lên mức 1. Bộ dao động là dao động công suất thấp, tốc độ 200 kHz. Bộ dao động vẫn tiếp tục chạy khi ở trong chế độ SLEEP. Bộ dao động dự định chủ yếu dùng với thạch anh 32 kHz. Bảng 2-20 trình bày cách lựa chọn tụ cho bộ dao động Timer1.

Osc Type	Freq.	C1	C2
LP	32 kHz	33 pF	33 pF
	100 kHz	15 pF	15 pF
	200 kHz	15 pF	15 pF

Bảng 2-20. Lựa chọn tụ cho bộ dao động.

g. Reset Timer1 sử dụng ngõ ra CCP Trigger:

Nếu khối CCP1 và CCP2 được định cấu hình ở chế độ so sánh để tạo ra "xung kích sự kiện đặc biệt" (CCP1M3:CCP1M0 = 1011), tín hiệu này sẽ reset Timer1.

Chú ý: "xung kích sự kiện đặc biệt" từ khối CCP1 và CCP2 sẽ không làm bit cờ ngắt TMR1IF (PIR1<0>) bằng 1.

Timer1 phải định cấu hình ở chế độ định thời hoặc bộ đếm đồng bộ để tạo tiện ích cho cấu trúc này. Nếu Timer1 đang hoạt động ở chế độ đếm bất đồng bộ thì hoạt động Reset không thể thực hiện được.

h. Reset cặp thanh ghi TMR1H, TMR1L của Timer1:

Hai thanh ghi TMR1H và TMR1L không thể về 00h khi reset lúc cấp nguồn POR (Power On Reset) hoặc bất kì reset nào khác, ngoại trừ "xung kích sự kiện đặc biệt" của CCP1 và CCP2.

Thanh ghi T1CON được reset về 00h khi hệ thống bị reset lúc cấp nguồn POR hoặc khi Brown-out Reset, reset này sẽ tắt timer và hệ số chia trước có giá trị 1:1. Ở tất cả các reset khác còn lại thì thanh ghi không bị ảnh hưởng.

Các thanh ghi của Timer1 như bảng 2-21:

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other Resets
0Bh,8Bh, 10Bh, 18Bh	INTCON	GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF	0000 000x	0000 000u
0Ch	PIR1	PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	0000 0000
8Ch	PIE1	PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	0000 0000
0Eh	TMR1L	Holding Register for the Least Significant Byte of the 16-bit TMR1 Register								xxxx xxxx	uuuu uuuu
0Fh	TMR1H	Holding Register for the Most Significant Byte of the 16-bit TMR1 Register								xxxx xxxx	uuuu uuuu
10h	T1CON	—	—	T1CKPS1	T1CKPS0	T1OSCEN	$\overline{\text{T1SYNC}}$	TMR1CS	TMR1ON	--00 0000	--uu uuuu

Bảng 2-21. Các thanh ghi của Timer1.

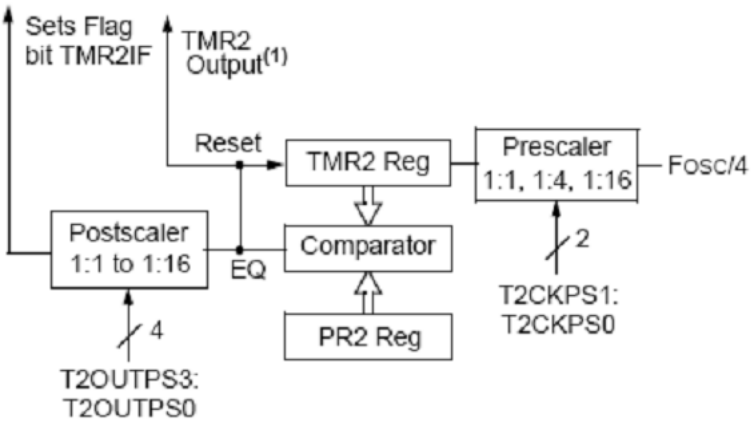
7. BỘ ĐỊNH THỜI TIMER2:

Timer2 là timer 8 bit có bộ chia trước và có postscaler. Timer2 có thể được sử dụng như bộ tạo xung có PWM cho chế độ hoạt động PWM của khối CCP. Thanh ghi TMR2 có thể đọc và ghi và có thể xoá khi bị reset.

Ngõ vào xung clock ($F_{OSC}/4$) có tùy chọn hệ số chia trước là: "1:1", "1:4" hoặc "1:16" được lựa chọn bằng các bit điều khiển T2CKPS1:T2CKPS0 (T2CON<1:0>).

Timer2 có 1 thanh ghi chu kỳ 8 bit PR2. Timer2 tăng giá trị từ 00h cho đến khi bằng PR2 và sau đó reset về 00h ở chu kì kế tiếp. PR2 là thanh ghi có thể đọc và ghi. Khi hệ thống bị reset thì thanh ghi PR được khởi tạo giá trị FFH.

Ngõ ra của TMR2 đi qua postscaler 4 bit để tạo ra yêu cầu ngắt TMR2 được chốt trong bit cờ TMR2IF (PIR1<1>). Có thể tắt Timer2 bằng cách xoá bit điều khiển TMR2ON (T2CON<2>), để giảm công suất tiêu thụ.



Hình 2-21. Sơ đồ khối của Timer2.

Thanh ghi điều khiển timer2:

T2CON: TIMER2 CONTROL REGISTER (ADDRESS 12h)

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0
bit 7							bit 0

Bit 7 Chưa sử dụng nếu đọc sẽ có giá trị '0'.

Bit 6-3 **TOUTPS3:TOUTPS0:** các bit lựa chọn ngõ ra Postscaler của Timer2

0000=1:1 postscaler.

0001=1:2 postscaler.

0010=1:3 postscaler.

.

.

1111=1:16 postscaler.

Bit 2 **TMR2ON**: Bit điều khiển cho phép/cấm Timer2

1= cho phép timer2 đếm.

0= Timer2 ngừng đếm.

Bit 1-0 **T2CKPS1:T2CKPS0**: bit lựa chọn hệ số chia trước cho nguồn xung clock của timer2

00= hệ số chia là 1.

01= hệ số chia là 4.

1x= hệ số chia là 6.

a. Bộ chia trước và postscaler của Timer2:

Bộ đếm chia trước và bộ đếm postscaler sẽ bị xóa khi xảy ra một trong các sự kiện sau:

- ☐ Thực hiện ghi dữ liệu vào thanh ghi TMR2.
- ☐ Ghi vào thanh ghi T2CON.
- ☐ Bất kì sự Reset nào của linh kiện.

TMR2 không bị xóa khi ghi dữ liệu vào thanh ghi T2CON.

b. Ngõ ra của TMR2:

Ngõ ra của TMR2 được nối tới khối SSP – khối này có thể tùy chọn để tạo ra xung nhịp.

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other Resets
0Bh, 8Bh, 10Bh, 18Bh	INTCON	GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF	0000 000x	0000 000u
0Ch	PIR1	PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	0000 0000
8Ch	PIE1	PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	0000 0000
11h	TMR2	Timer2 Module's Register								0000 0000	0000 0000
12h	T2CON	—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0	-000 0000	-000 0000
92h	PR2	Timer2 Period Register								1111 1111	1111 1111

Bảng 2-22. Các thanh ghi của Timer2.

8. KHỐI CHUYỂN ĐỔI TƯƠNG TỰ SANG SỐ ADC:

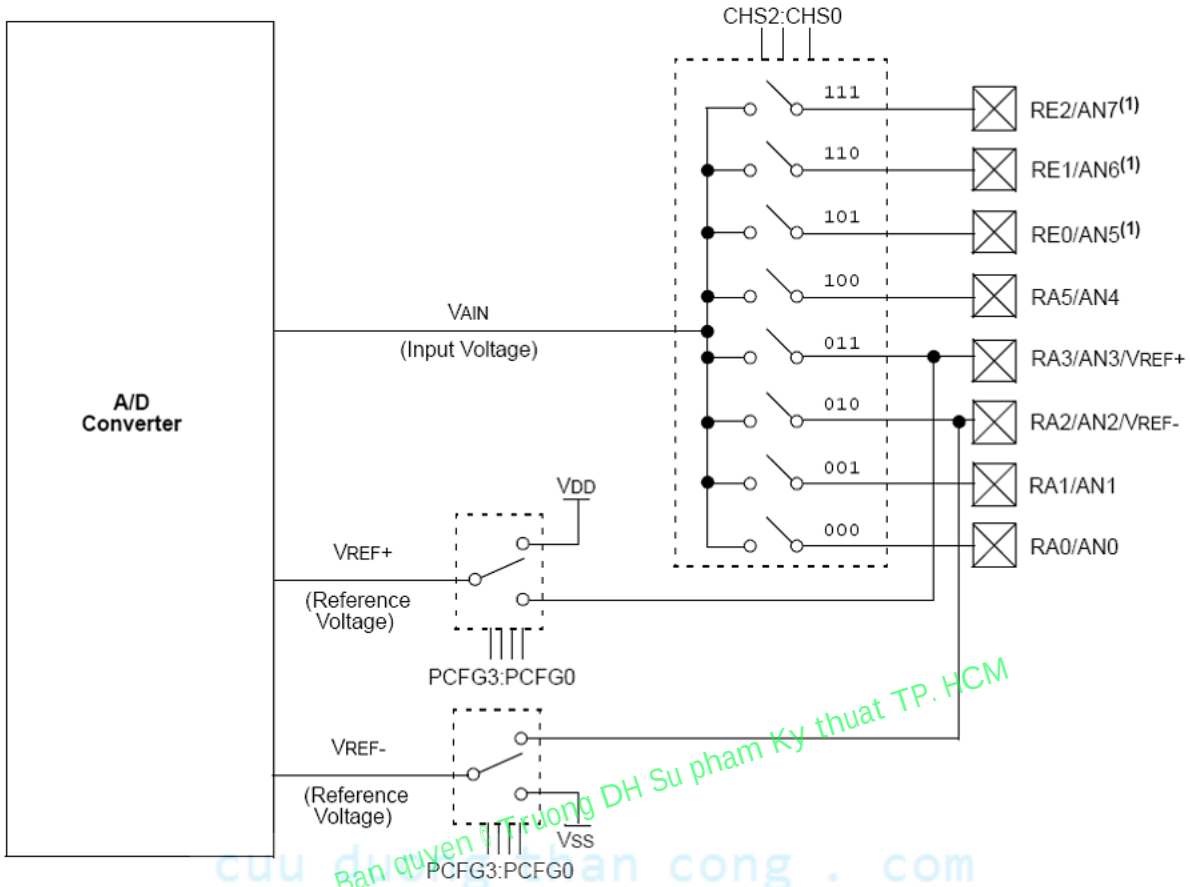
Khối chuyển đổi ADC 8 kênh và mỗi kênh ADC là 10 bit với chip 40 chân. Khối ADC có các ngõ vào điện áp chuẩn thấp và cao và các chân này có thể bằng phần mềm để tạo tổ hợp V_{DD}, V_{SS}, RA2 hoặc RA3.

Bộ chuyển đổi ADC có cấu trúc độc lập để có thể hoạt động trong khi vi điều khiển đang ở chế độ SLEEP, xung cung cấp cho ADC được lấy từ dao động RC bên trong của khối ADC.

Khối ADC có 4 thanh ghi:

- ☐ ADRESH (A/D Result High Register)
- ☐ ADRESL (A/D Result Low Register)
- ☐ ADCON0 (A/D Control Register 0)
- ☐ ADCON1 (A/D Control Register 1)

Thanh ghi ADCON0 có chức năng điều khiển hoạt động của khối ADC được trình bày ở hình 2-22. Thanh ghi ADCON1 thiết lập chức năng cho các chân của port là các ngõ vào nhận tương tự hoặc chân xuất nhập IO.



Hình 2-22. Sơ đồ khối của Timer2.

Thanh ghi ADCON0

ADCON0 REGISTER (ADDRESS 1Fh)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0
ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/$\overline{\text{DONE}}$	—	ADON
bit 7							bit 0

Bit 7 **ADCS0:ADCS:** các bit lựa chọn xung chuyển đổi AD

ADCON1 <ADCS2>	ADCON0 <ADCS1:ADCS0>	Clock Conversion
0	00	Fosc/2
0	01	Fosc/8
0	10	Fosc/32
0	11	FRC (clock derived from the internal A/D RC oscillator)
1	00	Fosc/4
1	01	Fosc/16
1	10	Fosc/64
1	11	FRC (clock derived from the internal A/D RC oscillator)

Bảng 2-23. Các bit lựa chọn xung chuyển đổi ADC.

Bit 5-3 **CHS2:CHS0**: các bit lựa chọn kênh tương tự

000 = kênh 0 (AN0)

001 = kênh 1 (AN1)

010 = kênh 2 (AN2)

011 = kênh 3 (AN3)

100 = kênh 4 (AN4)

101 = kênh 5 (AN5)

110 = kênh 6 (AN6)

111 = kênh 7 (AN7)

Bit 2 GO/ $\overline{DONE}$: bit báo trạng thái chuyển đổi ADC

Khi ADON = 1:

1= chuyển đổi ADC đang diễn ra (bằng 1 khi bắt đầu quá trình chuyển đổi và sẽ bị xoá về 0 khi quá trình chuyển đổi kết thúc).

0= chuyển đổi ADC không diễn ra.

Bit 1 Chưa dùng nếu đọc là '0'

Bit 0 **ADON**: bit mở nguồn cho ADC hoạt động:

1= khối chuyển đổi ADC được mở nguồn.

0= khối chuyển đổi ADC bị tắt nguồn để giảm công suất tiêu thụ.

Thanh ghi ADCON1

ADCON1 REGISTER (ADDRESS 9Fh)

R/W-0	R/W-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
ADFM	ADCS2	—	—	PCFG3	PCFG2	PCFG1	PCFG0
bit 7							bit 0

Bit 7 **ADFM**: bit lựa chọn định dạng kết quả ADC:

1= canh lề phải, 6 bit MSB của ADRESH có giá trị là '0'.

0= canh lề trái, 6 bit LSB của ADRESL có giá trị là '0'.

Bit 6 **ADCS2**: bit lựa chọn xung clock cho chuyển đổi ADC:

Bit 5-4: chưa dùng nếu đọc sẽ có giá trị là '0'

Bit 3-0: **PCFG3:PCFG0**: bit điều khiển ADC

PCFG <3:0>	AN7	AN6	AN5	AN4	AN3	AN2	AN1	AN0	VREF+	VREF-	C/R
0000	A	A	A	A	A	A	A	A	VDD	Vss	8/0
0001	A	A	A	A	VREF+	A	A	A	AN3	Vss	7/1
0010	D	D	D	A	A	A	A	A	VDD	Vss	5/0
0011	D	D	D	A	VREF+	A	A	A	AN3	Vss	4/1
0100	D	D	D	D	A	D	A	A	VDD	Vss	3/0
0101	D	D	D	D	VREF+	D	A	A	AN3	Vss	2/1
011x	D	D	D	D	D	D	D	D	—	—	0/0
1000	A	A	A	A	VREF+	VREF-	A	A	AN3	AN2	6/2
1001	D	D	A	A	A	A	A	A	VDD	Vss	6/0
1010	D	D	A	A	VREF+	A	A	A	AN3	Vss	5/1
1011	D	D	A	A	VREF+	VREF-	A	A	AN3	AN2	4/2
1100	D	D	D	A	VREF+	VREF-	A	A	AN3	AN2	3/2
1101	D	D	D	D	VREF+	VREF-	A	A	AN3	AN2	2/2
1110	D	D	D	D	D	D	D	A	VDD	Vss	1/0
1111	D	D	D	D	VREF+	VREF-	D	A	AN3	AN2	1/2

A = Analog input D = Digital I/O
C/R = # of analog input channels/# of A/D voltage references

Bảng 2-24. Các bit điều khiển ADC.

Các thanh ghi ADRESH:ADRESL chứa kết quả 10 bit của chuyển đổi. Khi chuyển đổi ADC được hoàn thành, kết quả được nạp vào cặp thanh ghi kết quả, bit GO/ $\overline{DONE}$ bị xóa và bit cờ báo ngắt ADC là ADIF lên 1. Sơ đồ khối của khối ADC được trình bày trong hình 2-22.

Sau khi khối ADC đã được định cấu hình theo yêu cầu, thì phải thực hiện chọn kênh trước khi bắt đầu quá trình chuyển đổi. Các kênh ngõ vào tương tự phải có các bit TRIS tương ứng được chọn như những ngõ vào.

Bước 1: Để thực hiện chuyển đổi ADC thì phải thực hiện các bước sau:

Bước 2: Thiết lập cấu hình ADC:

- ☐ Định cấu hình cho các chân tương tự/điện áp chuẩn và xuất/nhập số (ADCON1).
- ☐ Chọn lựa kênh ngõ vào ADC (ADCON0).
- ☐ Chọn lựa xung clock cho chuyển đổi ADC (ADCON0).
- ☐ Mở điện cho ADC (ADCON0)

Bước 3: Thiết lập cấu hình ngắt ADC (nếu được yêu cầu):

- ☐ Xóa bit ASDIF.
- ☐ Set bit ADIF.
- ☐ Set bit PEIE.
- ☐ Set bit GIE

Bước 3: Chờ hết thời gian theo yêu cầu:

Bước 4: Bắt đầu chuyển đổi: set bit GO/ $\overline{DONE}$

Bước 5: Chờ chuyển đổi ADC hoàn thành bằng cách:

- ☐ Kiểm tra liên tục bit GO/ $\overline{DONE}$ về 0 hay chưa (nếu không dùng ngắt).
- ☐ Chờ ngắt ADC xảy ra.

Bước 6: Đọc cặp thanh ghi kết quả (ADRESH:ADRSL), xóa bit ADIF nếu được yêu cầu

Bước 7: Thực hiện chuyển đổi kế tiếp. Thời gian chuyển đổi cho 1 bit là TAD.

a. Ngõ ra của TMR2:

Ngõ ra của TMR2 được nối tới khối SSP – khối này có thể tùy chọn để tạo ra xung nhịp.

b. Các yêu cầu nhận dữ liệu ADC:

Đối với các bộ chuyển đổi A/D để đảm bảo chuyển đổi chính xác theo thông số chỉ định thì các tụ giữ điện áp nạp phải nạp đầy đúng bằng mức điện áp của kênh ngõ vào. Sơ đồ ngõ vào tương tự được trình bày trong hình 2-23. Trở kháng nguồn (R_S) và trở kháng của chuyển mạch lấy mẫu bên trong (R_{SS}) ảnh hưởng trực tiếp đến thời gian nạp của tụ C_{HOLD}. Trở kháng của chuyển mạch lấy mẫu thay đổi theo điện áp của vi mạch như hình 2-23. Trở kháng tối đa cho nguồn tín hiệu tương tự được đề nghị là 2,5Ω. Khi trở kháng giảm thì thời gian thu nhận có thể giảm. Sau khi kênh ngõ vào được chọn thì thu nhận dữ liệu mới được thực hiện trước khi thực hiện chuyển đổi.

Để tính toán thời gian thu nhận nhỏ nhất thì ta có thể sử dụng phương trình 2-1. Giả sử lỗi của phương trình này là ½ LSB với ADC 10 bit có 1024 bước.

Phương trình 2-1: thời gian nhận

T_{ACQ} = Amplifier Settling Timer + Hold Capacitor Charging Timer + Temperature Coefficient

$= T_{AMP} + T_C + T_{COFF}$

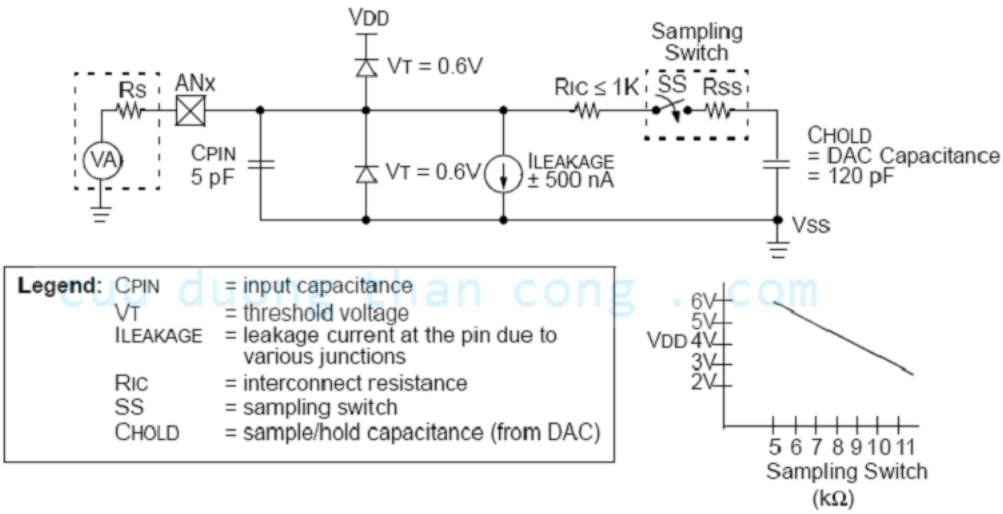
$T_C = C_{HOLD}(R_{IC} + R_{SS} + R_S)\ \text{In}(1/2047)$

$= -120\text{pF}(1\text{k}\Omega + 7\text{k}\Omega + 10\text{k}\Omega)\ \text{In}(0{,}0004885)$

$= 16{,}47\mu\text{s}$

$T_{ACQ} = 2\mu\text{s} + 16{,}47\mu\text{s} + [(50^{o}\text{C} - 25^{o}\text{C})(0{,}05\mu\text{s}/^{o}\text{C})$

$= 19{,}72\mu\text{s}$



Hình 2-23. Sơ đồ mạch của ngõ vào ADC.

c. Lựa chọn xung clock cho ADC:

Thời gian chuyển đổi ADC cho mỗi bit được xác định là T_{AD}. Chuyển đổi ADC sẽ dùng tối thiểu lượng thời gian 12 T_{AD} để chuyển đổi cho 10 bit. Nguồn xung đồng hồ cho ADC được lựa chọn bằng phần mềm. Bảy khả năng lựa chọn cho T_{AD} là:

- ☐ 2 T_{OSC}
- ☐ 4 T_{OSC}
- ☐ 8 T_{OSC}
- ☐ 16 T_{OSC}
- ☐ 32 T_{OSC}
- ☐ 64 T_{OSC}
- ☐ Bộ dao động RC bên trong module ADC.

Để chuyển đổi ADC là chính xác thì xung đồng hồ phải được chọn thời gian nhỏ nhất T_{AD} = 1,6µs. Bảng 2-25 trình bày thời gian TAD được tính toán từ các tần số hoạt động của vi điều khiển PIC và lựa chọn nguồn xung clock.

AD Clock Source (TAD)		Maximum Device Frequency
Operation	ADCS2:ADCS1:ADCS0	
2 Tosc	000	1.25 MHz
4 Tosc	100	2.5 MHz
8 Tosc	001	5 MHz
16 Tosc	101	10 MHz
32 Tosc	010	20 MHz
64 Tosc	110	20 MHz
RC(1, 2, 3)	x11	(Note 1)

Bảng 2-25. Các bit lựa chọn xung chuyển đổi ADC.

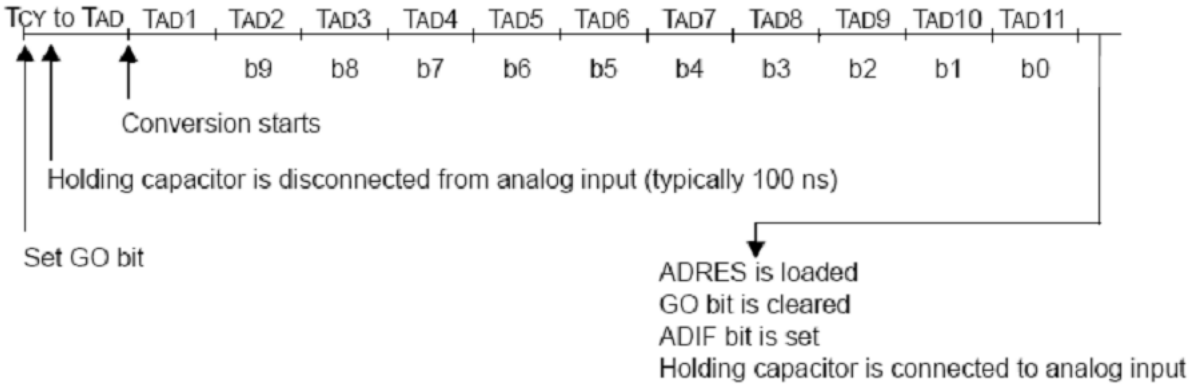
d. Định cấu hình cho các ngõ vào tương tự của ADC:

Hai thanh ghi ADCON1 và TRIS điều khiển hoạt động các chân port ADC. Các chân của port dùng làm các ngõ vào tương tự thì các bit TRIS tương ứng phải ở mức 1(ngõ vào). Nếu bit TRIS bằng 0 (ngõ ra) thì sẽ chuyển thành ngõ ra số. Hoạt động chuyển đổi ADC độc lập với trạng thái của các bit CHS0:CHS2 và các bit TRIS.

e. Chuyển đổi ADC:

Xóa bit GO/$\overline{DONE}$ trong thời gian chuyển đổi sẽ huỷ bỏ quá trình đang chuyển đổi. Cặp thanh ghi lưu kết quả chuyển đổi ADRESH:ADRESL sẽ không được cập nhập chuyển đổi và tiếp tục chứa các giá trị đã chuyển đổi của lần trước. Sau khi huỷ chuyển đổi ADC thì quá trình chuyển đổi tiếp theo của kênh đã chọn được bắt đầu một cách tự động. Làm bit GO/$\overline{DONE}$ lên 1 để bắt đầu quá trình chuyển đổi.

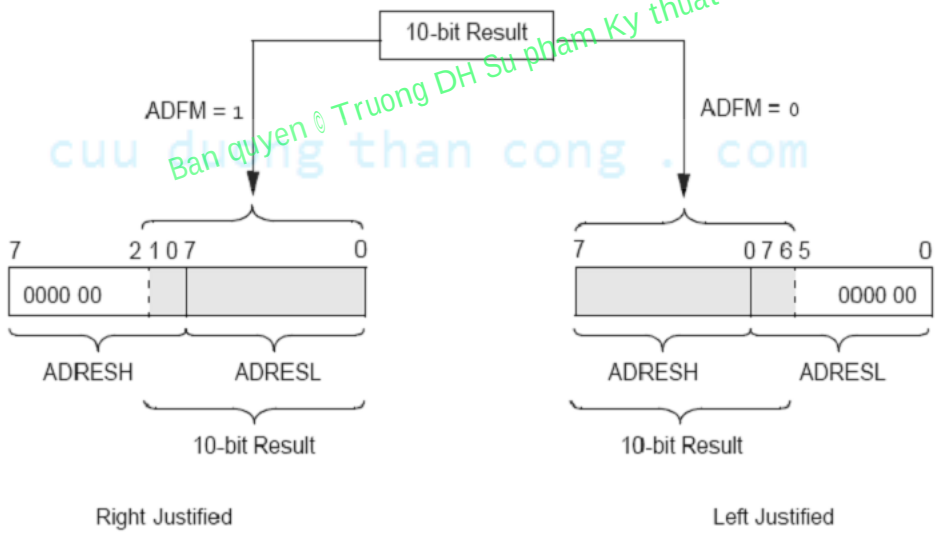
Trong hình 2-24 trình bày các chu kỳ chuyển đổi từ khi bit GO lên 1 cho đến lúc bắt đầu chuyển đổi và cho đến lúc kết thúc.



Hình 2-24. Chu kỳ chuyển đổi ADC.

f. Các thanh ghi lưu kết quả của ADC:

Cặp thanh ghi 16 bit ADRESH:ADRESL dùng để lưu kết quả chuyển đổi 10 bit của ADC sau khi chuyển đổi xong. Do kết quả chỉ có 10 bit nhưng lưu trong cặp thanh ghi 16 bit nên có các kiểu định dạng sau tuỳ thuộc vào trạng thái bit ADFM (ADC Format). Hình 2-25 trình bày 2 kiểu định dạng của cặp thanh ghi kết quả:



Hình 2-25. Cặp thanh ghi kết quả hiệu chỉnh phải và trái.

g. Hoạt động chuyển đổi ADC trong chế độ Sleep:

Khối ADC có thể hoạt động trong chế độ Sleep và nguồn xung clock được thiết lập cho RC (ADCS1:ADCS0 = 11). Khi chọn nguồn xung clock RC thì khối ADC chờ thêm một chu kỳ lệnh trước khi quá trình chuyển đổi bắt đầu. Việc chờ thêm một chu kỳ lệnh cho phép thực hiện lệnh SLEEP để vi điều khiển PIC vào chế độ nhằm loại trừ tất cả nhiễu chuyển mạch số. Khi chuyển đổi kết thúc thì bit GO/$\overline{DONE}$ bị xóa và kết quả lưu vào thanh ghi ADRES. Nếu ngắt ADC được cho phép, vi điều khiển PIC sẽ thoát khỏi chế độ SLEEP và hoạt động bình thường trở lại. Nếu ngắt ADC không được cho phép, khối ADC sẽ tắt sau đó, mặc dù bit ADON vẫn duy trì ở mức 1.

h. Ảnh hưởng của reset:

Reset PIC sẽ buộc các thanh ghi ở trạng thái reset. Khi reset sẽ làm khối ADC tắt và huỷ bỏ luôn quá trình ADC đang chuyển đổi. Tất cả các chân ngõ vào ADC được định cấu hình như những ngõ vào tương tự.

Giá trị trong hai thanh ghi ADRESH:ADRESL là không được hiệu chỉnh khi reset lúc cấp điện. Hai thanh ghi ADRESH:ADRESL chứa dữ liệu không xác định sau khi reset lúc cấp điện.

Các thanh ghi sử dụng cho khối ADC:

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Value on MCLR, WDT
0Bh,8Bh, 10Bh,18Bh	INTCON	GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF	0000 000x	0000 000u
0Ch	PIR1	PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	0000 0000
8Ch	PIE1	PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	0000 0000
1Eh	ADRESH	A/D Result Register High Byte								xxxx xxxx	uuuu uuuu
9Eh	ADRESL	A/D Result Register Low Byte								xxxx xxxx	uuuu uuuu
1Fh	ADCON0	ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/$\overline{\text{DONE}}$	—	ADON	0000 00-0	0000 00-0
9Fh	ADCON1	ADFM	ADCS2	—	—	PCFG3	PCFG2	PCFG1	PCFG0	00-- 0000	00-- 0000
85h	TRISA	—	—	PORTA Data Direction Register						--11 1111	--11 1111
05h	PORTA	—	—	PORTA Data Latch when written: PORTA pins when read						--0x 0000	--0u 0000
89h(1)	TRISE	IBF	OBF	IBOV	PSPMODE	—	PORTE Data Direction bits			0000 -111	0000 -111
09h(1)	PORTE	—	—	—	—	—	RE2	RE1	RE0	---- -xxx	---- -uuu

Bảng 2-26. Các thanh ghi dùng cho chuyển đổi ADC.

9. KHỐI SO SÁNH

Khối so sánh chứa hai bộ so sánh tương tự. Ngõ vào bộ so sánh đa hợp với các chân I/O từ RA0 đến RA3 và các ngõ ra đa hợp với các chân từ chân RA4 và RA5. Điện áp chuẩn trên IC cũng có thể là một ngõ vào của bộ so sánh.

Thanh ghi CMCON điều khiển bộ đa hợp ngõ vào và ngõ ra của bộ so sánh. Sơ đồ khối các mô hình khác nhau của bộ so sánh được trình bày trong hình 2-26.

Thanh ghi CMCON

CMCON REGISTER

R-0	R-0	R/W-0	R/W-0	R/W-0	R/W-1	R/W-1	R/W-1
C2OUT	C1OUT	C2INV	C1INV	CIS	CM2	CM1	CM0
bit 7							bit 0

Bit 7 **COUT**: bit ngõ ra bộ so sánh 2

Khi C2INV= 0:

1= C2 V_{IN+} > C2 V_{IN-}

0= C2 V_{IN+} < C2 V_{IN-}

Khi CINV= 1:

1= C2 V_{IN+}< C2 V_{IN-}

0= C2 V_{IN+} > C2 V_{IN-}

Bit 6 **C1OUT**: ngõ ra bộ so sánh 1

Khi C1INV= 0:

1= C V_{IN+} > C1 V_{IN-}

0= C1 V_{IN+} < C1 V_{IN-}

Khi CINV= 1:

1= C1 V_{IN+}< C1 V_{IN-}

0= C1 V_{IN+} > C1 V_{IN-}

Bit 5 **C2INV**: bit đảo ngõ ra bộ so sánh 2

1= ngõ ra C2 được đảo

0= ngõ ra C2 không được đảo

Bit 4 **C1INV**: bit đảo ngõ ra bộ so sánh 1

1= ngõ ra C1 được đảo

0= ngõ ra C1 không được đảo

Bit 3 **CIS**: bit chuyển đổi ngõ vào bộ so sánh

Khi CM2:CM0 = 110:

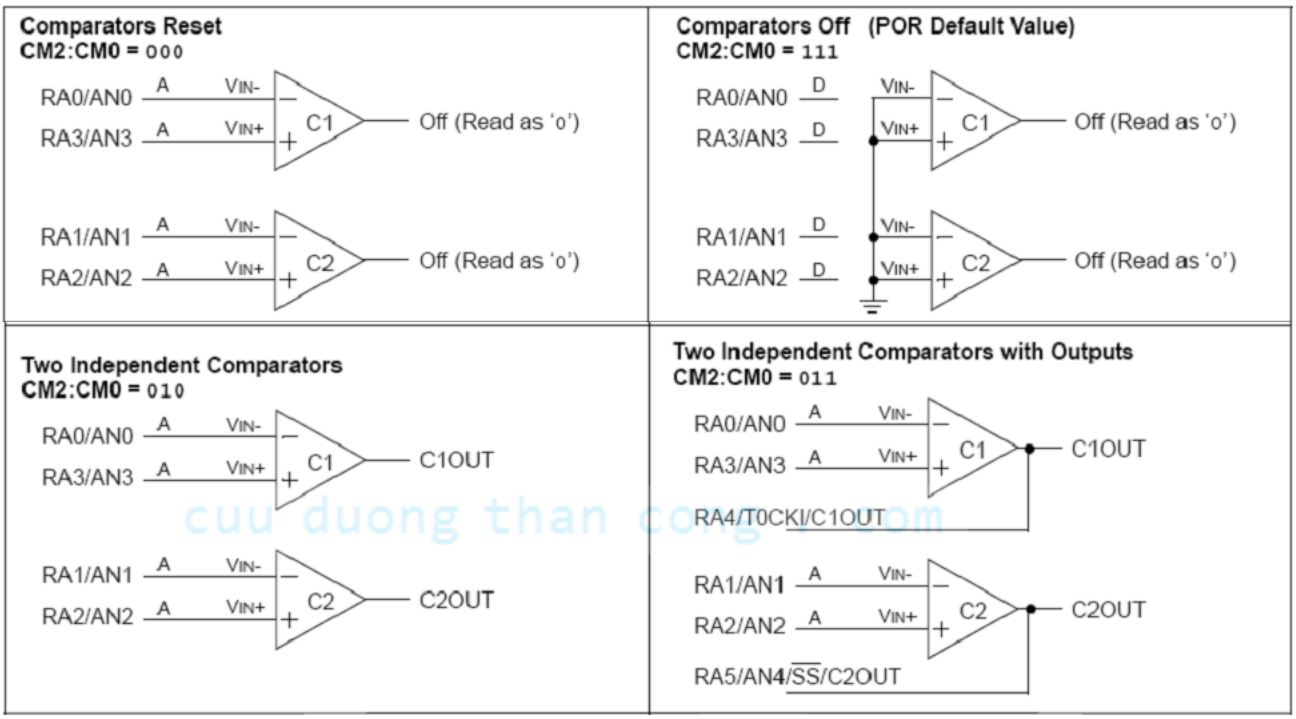
1= C1 V_{IN-} kết nối với RA3/AN3

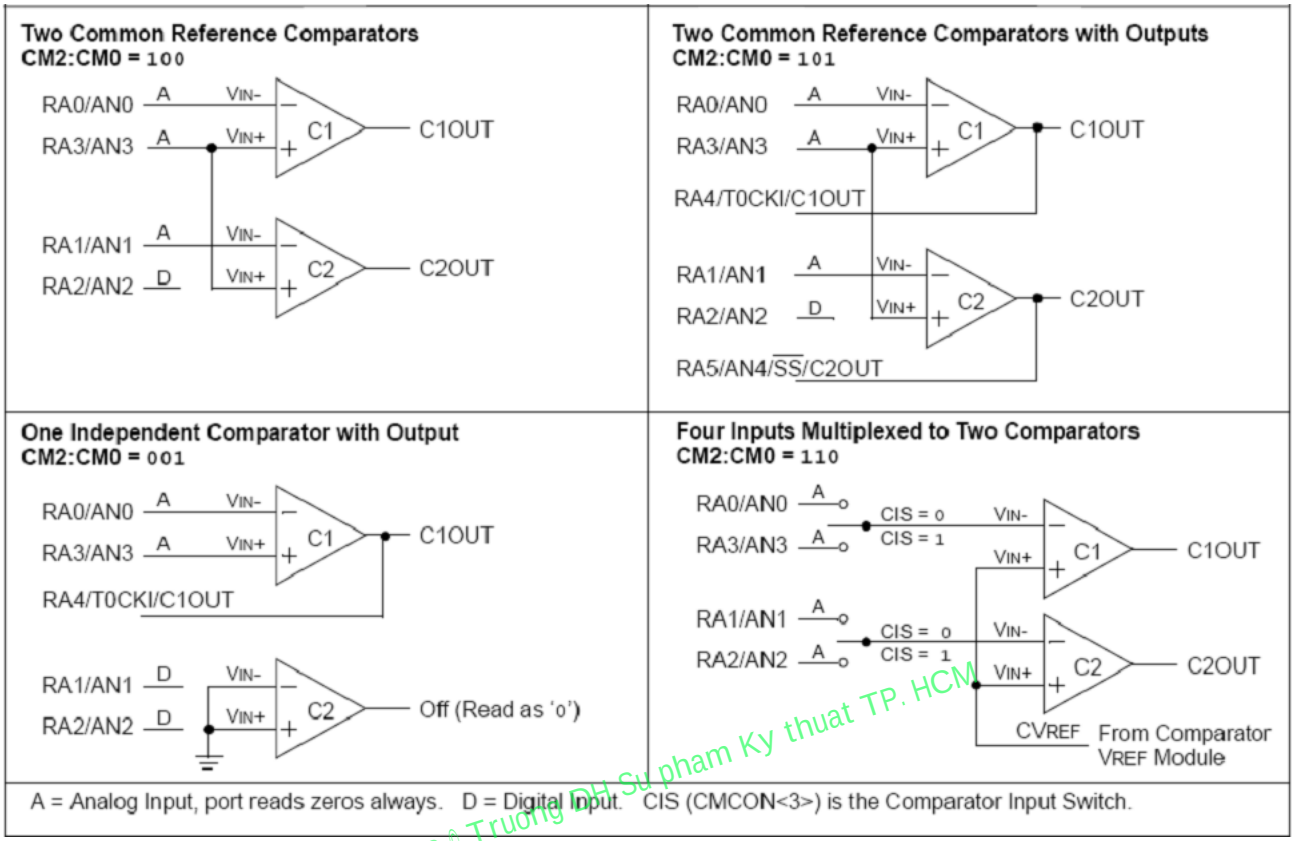
C2 V_{IN-} kết nối với RA2/AN2

0= C1 V_{IN-} kết nối với RA0/AN0

C2 V_{IN-} kết nối với RA1/AN1

Bit 2-0 **CM2:CM0**: các bit chọn kiểu so sánh





Hình 2-26. Các kiểu hoạt động của bộ so sánh.

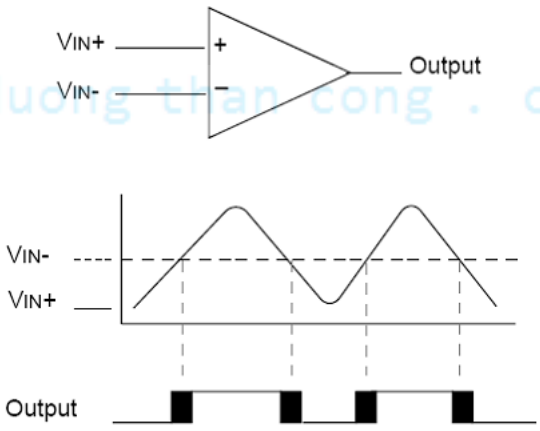
Có 8 kiểu hoạt động của bộ so sánh. Thanh ghi CMCON được sử dụng để lựa chọn các kiểu này. Thanh ghi TRISA điều khiển dữ liệu trực tiếp các chân của bộ so sánh ở mỗi kiểu.

a. Hoạt động so sánh:

Bộ so sánh đơn trình bày trong hình 2-27 cùng với mối quan hệ giữa các mức ngõ vào tương tự và ngõ ra số.

- Khi ngõ vào V_{IN+} nhỏ hơn ngõ vào V_{IN-} thì ngõ ra số của bộ so sánh ở mức thấp (0).
- Khi ngõ vào V_{IN+} lớn hơn ngõ vào V_{IN-} thì ngõ ra số của bộ so sánh ở mức cao (1).

Hình 2-27 trình bày mạch so sánh, dạng sóng của các tín hiệu vào so sánh và tín hiệu ra: ngõ ra số có thể bị lệch nhanh hay chậm là do điện áp offset và thời gian đáp ứng.



Hình 2-27. Các kiểu hoạt động của bộ so sánh.

b. Điện áp so sánh:

Tín hiệu chuẩn bên ngoài hoặc bên trong có thể được sử dụng tuỳ thuộc vào kiểu hoạt động của bộ so sánh. Tín hiệu tương tự V_{IN-} được so sánh với tín hiệu V_{IN+} và ngõ ra số được điều chỉnh sao cho phù hợp.

Tín hiệu chuẩn bên ngoài:

Khi sử dụng điện áp chuẩn bên ngoài thì bộ so sánh có thể được định hình để có các hoạt động so sánh với nguồn điện áp chuẩn giống nhau hoặc khác nhau. Tuy nhiên những ứng dụng tách sóng theo ngưỡng thì cần điện áp chuẩn giống nhau. Tín hiệu chuẩn nằm trong giới hạn từ V_{SS} đến V_{DD} và có thể cấp đến các chân ngõ vào của bộ so sánh.

Tín hiệu chuẩn bên trong:

Bộ so sánh cũng cho phép lựa chọn nguồn điện áp chuẩn bên trong cho bộ so sánh. Tín hiệu chuẩn bên trong được sử dụng khi các bộ so sánh hoạt động ở kiểu CM<2:0> = 110 như hình 2-24 ở trên. Ở kiểu này thì điện áp chuẩn bên trong được đưa đến chân V_{IN+} của các bộ so sánh.

c. Thời gian đáp ứng:

Thời gian đáp ứng là thời gian nhỏ nhất sau khi lựa chọn một điện áp chuẩn mới hoặc nguồn ngõ vào, trước khi ngõ ra bộ so sánh ở mức hợp lệ. Nếu điện áp chuẩn bên trong bị thay đổi, thời gian trì hoãn lớn nhất của điện áp chuẩn bên trong phải được xem xét khi sử dụng các ngõ ra của bộ so sánh. Tra bảng thông số đặc tính để biết thời gian đáp ứng.

d. Ngõ ra bộ so sánh:

Ngõ ra bộ so sánh được đọc thông qua thanh ghi chỉ đọc CMCON. Các ngõ ra của bộ so sánh cũng có thể là các ngõ ra trực tiếp ở các chân xuất/nhập RA4 và RA5. Khi được cho phép, bộ đa hợp các chân RA4 và RA5 sẽ chuyển mạch và ngõ ra của mỗi chân sẽ không đồng bộ với ngõ ra của bộ so sánh. Hình 2-28 trình bày sơ đồ mạch của bộ so sánh.

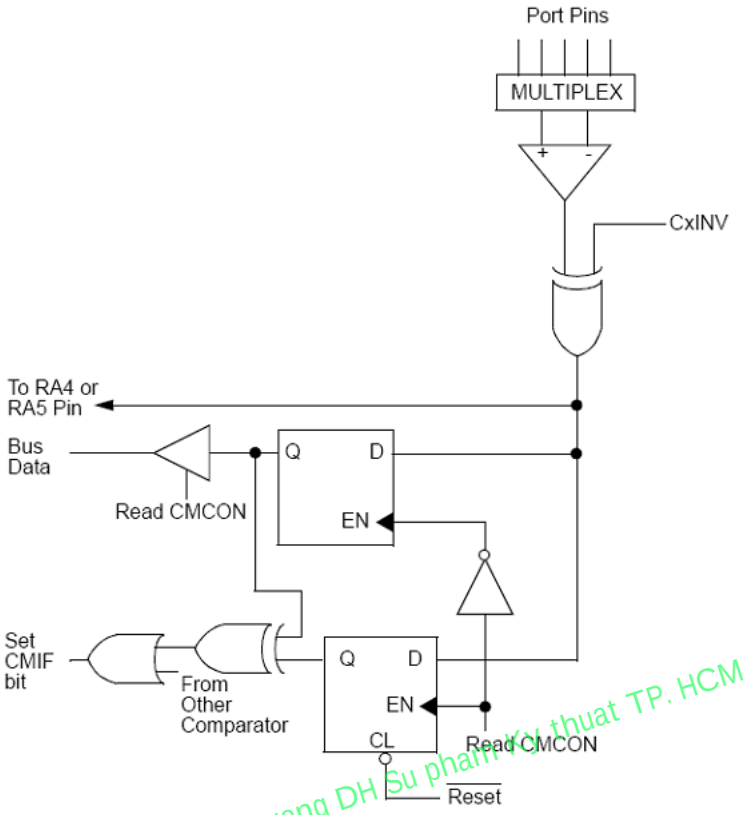
Các bit TRISA vẫn còn chức năng cho phép/cấm đối với các chân RA4 và RA5.

Cực tính ngõ ra của bộ so sánh có thể được chuyển đổi sử dụng 2bit C2IHV và C1INV (CMCON<4:5>)

Chú ý: khi đọc thanh ghi port, tất cả các chân được dùng là ngõ vào tương tự sẽ đọc là '0'. Các ngõ vào được định cấu hình như ngõ vào số sẽ chuyển đổi thành ngõ vào tương tự tùy vào yêu cầu kĩ thuật ngõ vào Schmitt Trigger.

Các mức điện áp tương tự trên bất kỳ chân nào được định nghĩa như ngõ vào số có thể trở thành bộ đệm để tăng thêm dòng.

RA4 là chân IO dạng cực thu để hở. Khi được dùng làm ngõ ra thì phải có điện trở kéo lên.



Hình 2-28. Sơ đồ mạch của bộ so sánh.

e. Ngắt của bộ so sánh:

Cờ ngắt bộ so sánh lên mức 1 bất kỳ lúc nào có sự thay đổi giá trị ngõ ra bộ so sánh. Phần mềm duy trì kiểm tra trạng thái các bit ngõ ra bằng cách đọc các bit CMCON<7:6>, để xác định chuyển đổi thực đã xảy ra hay chưa. Bit CMIF (thanh ghi PIR) là cờ báo ngắt bộ so sánh. Bit CMIF phải được reset bằng cách xóa nó. Do cũng có thể ghi mức 1 vào thanh ghi này, khi đó ngắt mô phỏng được bắt đầu.

Bit CMIE (thanh ghi PIE) và PEIE (thanh ghi INTCON) phải được set ở mức 1 để cho phép ngắt. Thêm vào đó, bit GIE cũng phải được set ở mức 1. Nếu bất kì một trong những bit này bị xóa thì ngắt sẽ không được cho phép, mặc dù bit CMIF vẫn còn ở mức 1 nếu điều kiện ngắt xảy ra.

Trong chương trình phục vụ ngắt, người sử dụng có thể xóa sự ngắt theo những cách sau:

- ☐ Thực hiện bất kỳ lệnh đọc/ghi CMCON sẽ kết thúc điều kiện không thích ứng.
- ☐ Xóa bit cờ CMIF.

f. Hoạt động của bộ so sánh ở chế độ Sleep:

Khi bộ so sánh hoạt động và vi điều khiển ở trong chế độ Sleep, bộ so sánh vẫn duy trì hoạt động và sẽ báo ngắt nếu được cho phép. Ngắt do bộ so sánh tạo ra sẽ làm vi điều khiển thoát khỏi chế độ Sleep. Để giảm tiêu tốn năng lượng trong chế độ Sleep thì nên tắt bộ so sánh (CM<2:0>=111) trước khi thực hiện Sleep. Nếu vi điều khiển thoát khỏi chế độ Sleep thì nội dung của thanh ghi CMCON không bị ảnh hưởng.

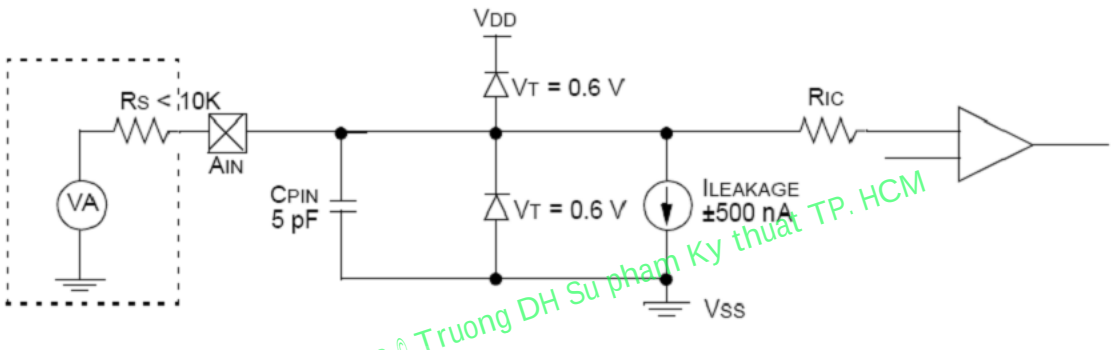
g. Ảnh hưởng của reset:

Khi Reset PIC sẽ ảnh hưởng đến trạng thái reset của thanh ghi CMCON làm cho khối so sánh ở trong chế độ tắt, CM<2:0> = 111.

h. Kết nối các ngõ vào tương tự:

Mạch điện đơn giản cho ngõ vào tương tự được trình bày trong hình 2-29. Do các chân tương tự kết nối đến các ngõ ra số nên chúng có 2 diode phân cực ngược đối với V_{DD} và V_{SS}. Do đó, ngõ vào tương tự phải nằm trong giới hạn điện áp từ V_{DD} đến V_{SS}.

Nếu điện áp ngõ vào lệch khỏi giới hạn trên khoảng 0,6V theo chiều tăng hoặc giảm thì một trong các diode sẽ phân cực thuận và xảy ra hiện tượng ghi áp. Trở kháng nguồn lớn nhất nên dùng là 10kΩ cho nguồn tương tự. Bất kì thành phần nào bên ngoài kết nối đến chân ngõ vào tương tự như là tụ hoặc diode Zener sẽ có một dòng rò có giá trị nhỏ.



Legend:
CPIN = Input Capacitance
VT = Threshold Voltage
ILEAKAGE = Leakage Current at the pin due to various junctions
RIC = Interconnect Resistance
RS = Source Impedance
VA = Analog Voltage

Hình 2-29. Sơ đồ mạch ngõ vào tương tự.

Các thanh ghi kết hợp của khối so sánh như bảng sau:

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR	Value on all other Resets
9Ch	CMCON	C2OUT	C1OUT	C2INV	C1INV	CIS	CM2	CM1	CM0	0000 0111	0000 0111
9Dh	CVRCON	CVREN	CVROE	CVRR	—	CVR3	CVR2	CVR1	CVR0	000- 0000	000- 0000
0Bh, 8Bh, 10Bh,18Bh	INTCON	GIE/ GIEH	PEIE/ GIEL	TMR0IE	INTIE	RBIE	TMR0IF	INTIF	RBIF	0000 000x	0000 000u
0Dh	PIR2	—	CMIF	—	—	BCLIF	LVDIF	TMR3IF	CCP2IF	-0-- 0000	-0-- 0000
8Dh	PIE2	—	CMIE	—	—	BCLIE	LVDIE	TMR3IE	CCP2IE	-0-- 0000	-0-- 0000
05h	PORTA	—	—	RA5	RA4	RA3	RA2	RA1	RA0	--0x 0000	--0u 0000
85h	TRISA	—	—	PORTA Data Direction Register						--11 1111	--11 1111

Bảng 2-27. Các thanh ghi dùng cho bộ so sánh.

Bộ tạo điện áp chuẩn của mạch so sánh là một mạng điện trở bậc thang 16 cấp nhằm tạo ra điện áp chuẩn cố định khi bộ so sánh làm việc ở kiểu '110'. Thanh ghi lập trình điều khiển chức năng của bộ tạo điện áp chuẩn là CVRCON.

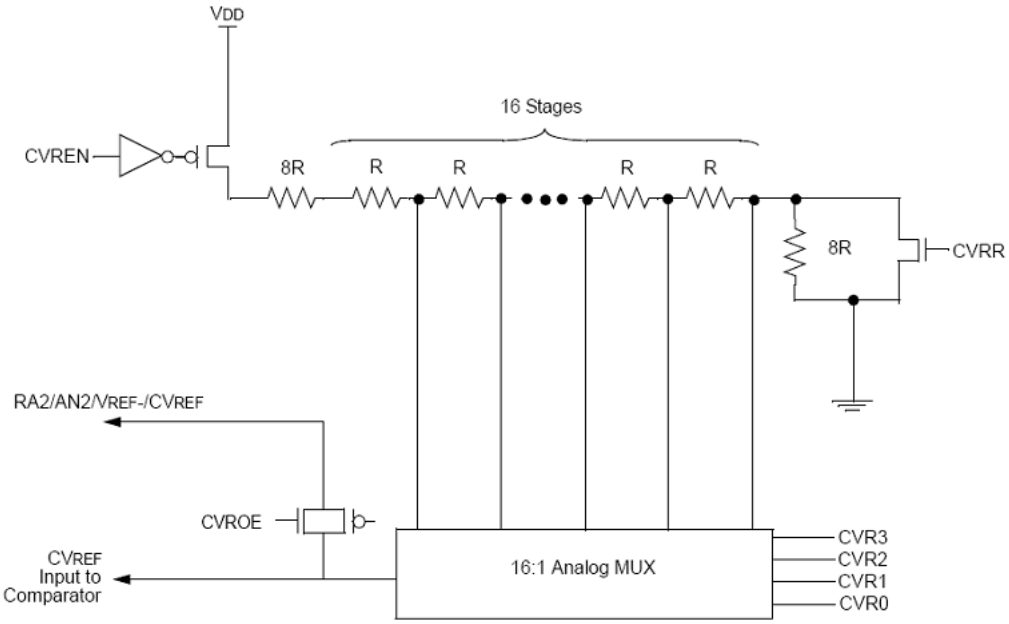
Hình 2-30 là sơ đồ bậc thang điện trở được phân đoạn để tạo ra hai dãy giá trị của CV_{REF} và chức năng giảm công suất để giảm công suất tiêu tán khi nguồn điện áp chuẩn không sử dụng. Điện áp cung cấp cho mạch tạo điện áp chuẩn lấy từ nguồn V_{DD}.

Ngõ ra của bộ tạo điện áp chuẩn có thể nối với chân RA2/AN2/Vref-/CV_{REF}. Chân này có thể dùng như chân ngõ ra của bộ chuyển đổi DAC đơn giản nhưng chức năng chính của việc đưa điện áp chuẩn ra chân đó là nhằm kiểm tra xem nguồn điện áp chuẩn có chính xác hay không.

CVRCON CONTROL REGISTER (ADDRESS 9Dh)

R/W-0	R/W-0	R/W-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
CVREN	CVROE	CVRR	—	CVR3	CVR2	CVR1	CVR0
bit 7							bit 0

Bit 7	**CVREN**: bit cho phép mạch tạo điện áp chuẩn cho bộ so sánh
	1= cho phép mạch hoạt động.
	0= không cho phép mạch hoạt động.
Bit 6	**CVROE**: bit cho phép ngõ ra bộ so sánh V_{REF}
	1= mức điện áp VCREF được đưa đến chân RA2/AN2/Vref-/CV_{REF}.
	0= mức điện áp VCREF không được đưa đến chân RA2/AN2/Vref-/CV_{REF}.
Bit 5	**CVRR**: bit lựa chọn dãy điện áp V_{REF} của bộ so sánh 2
	1= từ 0 đến 0.75 CV_{RSRC} với độ phân giải của bước là CV_{RSRC} /24.
	0= từ 0.25 đến 0.75 CV_{RSRC} với độ phân giải của bước là CV_{RSRC} /32.
Bit 4	**Chưa dùng nếu đọc sẽ có giá trị '0'**
	1= ngõ ra C1 được đảo
	0= ngõ ra C1 không được đảo
Bit 3-0	**CVR3:CVR0**: các bit lựa chọn giá trị VREF của bộ so sánh từ 0 đến 15
	Khi CVRR=1:
	CV_{REF} =(VR<3:0>/24)*CV_{RSRC}.
	Khi CVRR=0:
	CV_{REF} =1/4 * CV_{RSRC} +(VR<3:0>/32)*CV_{RSRC}.



Hình 2-30. Sơ đồ khối mạch tạo điện áp chuẩn cho bộ so sánh.

Các thanh ghi kết hợp của khối so sánh như bảng sau:

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR	Value on all other Resets
9Dh	CVRCON	CVREN	CVROE	CVRR	—	CVR3	CVR2	CVR1	CVR0	000- 0000	000- 0000
9Ch	CMCON	C2OUT	C1OUT	C2INV	C1INV	CIS	CM2	CM1	CM0	0000 0111	0000 0111

Bảng 2-28. Các thanh ghi dùng cho bộ tạo điện áp chuẩn.

10. CÁC CẤU TRÚC ĐẶC BIỆT CỦA CPU:

PIC16F87XA có một số đặc điểm làm tăng độ tin cậy, giảm giá thành đến mức tối thiểu thông qua việc loại bỏ các bộ phận bên ngoài, cung cấp các chế độ hoạt động tiết kiệm năng lượng và cung cấp mã bảo vệ. Đó là:

- ☐ Sự lựa chọn bộ dao động (OSC).
- ☐ Reset:
  - Power-on Reset (POR).
  - Power-up Timer (PWRT).
  - Bộ dao động Start-up Timer (OSC).
  - Brown-out Reset (BOR).
- ☐ Các ngắt.
- ☐ Watchdog Timer (WDT).
- ☐ Sleep.
- ☐ Mã bảo vệ.
- ☐ Nhận dạng ID.
- ☐ Lập trình tuần tự trong mạch.

- ☐ Lập trình tuần tự trong mạch điện áp thấp.
- ☐ Bộ gỡ rối.

PIC16F87XA có WDT có thể dừng thông qua các bit định cấu hình. WDT có bộ dao động RC hoạt động riêng để tăng độ tin cậy.

Có hai bộ định thời cung cấp thời gian trì hoãn cần thiết khi cấp điện. Một là OST (Oscillator Start-up Timer) có chức năng giữ IC ở trạng thái Reset cho đến khi bộ dao động thạch anh hoạt động ổn định. Hai là PWRT (Power-up timer) cung cấp thời gian trì hoãn cố định khoảng 72ms chỉ khi mới cấp điện. Nó được thiết kế để giữ thiết bị ở trạng thái Reset chờ nguồn cung cấp ổn định. Với hai bộ định thời, hầu hết các ứng dụng không cần mạch Reset ngoài.

Chế độ Sleep được thiết kế để tiêu thụ dòng thấp khi ở chế độ power-down. Người sử dụng có thể khởi động vi điều khiển khỏi chế độ Sleep thông qua Reset ngoài, WDT hoặc thông qua ngắt.

Một vài bộ dao động tùy chọn cũng được thiết kế để cho phép một số bộ phận hoạt động phù hợp với ứng dụng. Bộ dao động tùy chọn RC tiết kiệm chi phí trong khi bộ dao động thạch anh LP tiết kiệm năng lượng. Một tập hợp các bit định cấu hình được sử dụng cho những chọn lựa khác.

Các bit định cấu hình:

Các bit định cấu hình có thể lập trình được (khi đọc có giá trị '0') hoặc không lập trình được (khi đọc có giá trị '1') để lựa chọn các cấu hình khác nhau cho vi điều khiển. Giá trị xóa hoặc không lập trình được của thanh ghi định cấu hình (Configuration Word register) là 3FFFh. Những bit này nằm trong bộ nhớ chương trình tại địa chỉ 2007h. Đặc biệt chú ý là địa chỉ này nằm ngoài giới hạn của bộ nhớ chương trình.

Thanh ghi Configuration Word

R/P-1	U-0	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1	U-0	U-0	R/P-1	R/P-1	R/P-1	R/P-1
CP	—	DEBUG	WRT1	WRT0	CPD	LVP	BOREN	—	—	$\overline{\text{PWRTEN}}$	WDTEN	Fosc1	Fosc0
bit 13													bit0

Bit 13 **CP**: bit mã bảo vệ bộ nhớ chương trình Flash

1= mã bảo vệ tắt.

0= tất cả bộ nhớ chương trình có mã bảo vệ.

Bit 12 chưa sử dụng: đọc là '1'

Bit 11 **DEBUG**: bit thiết lập chế độ gỡ rối

1= mạch gỡ rối không được cho phép, RB6 và RB7 là các chân IO.

0= mạch gỡ rối được phép, chân RB6 và RB7 dành cho việc gỡ rối.

Bit 10-9 **WRT1-WRT0**: các bit cho phép ghi bộ nhớ chương trình Flash

11= tắt bảo vệ ghi; có thể ghi dữ liệu vào bộ nhớ chương trình bằng cách điều khiển thanh ghi EECON.

10= các ô nhớ có địa chỉ từ 0000h đến 00FFh bảo vệ chống ghi, từ 0100h đến 1FFFh cho phép ghi bằng cách điều khiển thanh ghi EECON.

01= các ô nhớ có địa chỉ từ 0000h đến 07FFh bảo vệ chống ghi, từ 0800h đến 1FFFh cho phép bằng cách điều khiển thanh ghi EECON.

00= các ô nhớ có địa chỉ từ 0000h đến 0FFFh bảo vệ chống ghi, từ 1000h đến 1FFFh cho phép ghi bằng cách điều khiển thanh ghi EECON.

Bit 8	**CPD**: bit mã bảo vệ bộ nhớ dữ liệu EEPROM
	1= tắt mã bảo vệ bộ nhớ dữ liệu EEPROM.
	0= bộ nhớ dữ liệu EEPROM có mã bảo vệ chống ghi.
Bit 7	**LVP**: bit cho phép mạch lập trình tuần tự điện áp thấp.
	1= chân RB3/PGM có chức năng PGM – cho phép lập trình điện áp thấp.
	0= chân RB3 là IO số, chân HV ở $\overline{MCLR}$ dùng để lập trình.
Bit 6	**BOREN**: bit cho phép Reset Brown-out
	1= cho phép BOR
	0= không cho phép BOR
Bit 5-4	Không được hỗ trợ: đọc là '1'
Bit 3	$\overline{PWRTEN}$: bit cho phép Timer hoạt động khi reset lúc mở điện:
	1= không cho phép PWRT.
	0= cho phép PWRT.
Bit 2	**WDTEN**: bit cho phép WDT
	1= cho phép WDT
	0= không cho phép WDT
Bit 1-0	**F_{OSC}1:F_{OSC}0**: các bit lựa chọn bộ dao động
	11= bộ dao động RC
	10= bộ dao động HS
	01= bộ dao động XT
	00= bộ dao động LP

11. CẤU HÌNH BỘ DAO ĐỘNG:

a. Các loại mạch dao động:

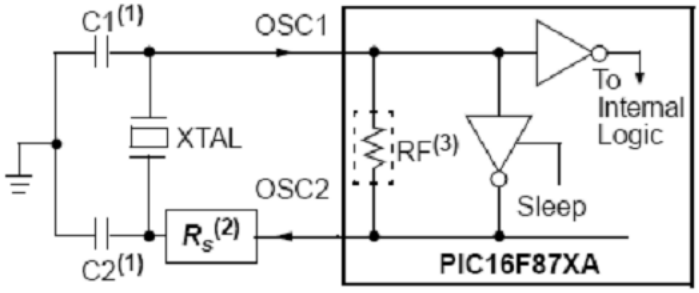
PIC16F877A có thể hoạt động ở 4 kiểu dao động khác nhau. Người dùng có thể lập trình cho hai bit định cấu hình để chọn 1 trong 4 chế độ:

- ☐ LP: Low-power Crystal (thạch anh năng lượng thấp)
- ☐ XT: Crystal/Resonator (thạch anh/cộng hưởng)
- ☐ HS: High-speed Crystal/Resonator (thạch anh/ cộng hưởng tốc độ cao)
- ☐ RC: Resistor/Capacitor (điện trở/tụ)

b. Dao động thạch anh/tụ Ceramic:

Trong các kiểu dao động XT, LP hoặc HS, thạch anh hoặc tụ Ceramic được nối đến các chân OSC1/CLKI và OSC2/CLKO để tạo dao động như hình 2-31. Thiết kế bộ dao động PIC16F877A yêu cầu sử dụng kiểu cắt thạch anh song song. Sử dụng thạch anh kiểu cắt nối tiếp có thể khác với

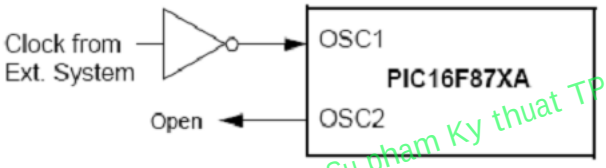
tần số yêu cầu của nhà sản xuất. Khi hoạt động ở kiểu XT, LP hoặc HS thì PIC có thể sử dụng nguồn xung clock từ bên ngoài đưa đến chân OSC1/CLKI như hình 2-32.



Hình 2-31. Dao động dùng thạch anh/tụ cộng hưởng cấu hình XT, LP hoặc HS.

Chú ý: (1) Xem bảng 2-22 để có các giá trị của tụ C1 và C2.

Chú ý: (2) Điện trở nối tiếp R_S có thể được yêu cầu đối với thạch anh.



Hình 2-32. Ngõ vào nhận xung từ bên ngoài cấu hình XT, LP hoặc HS.

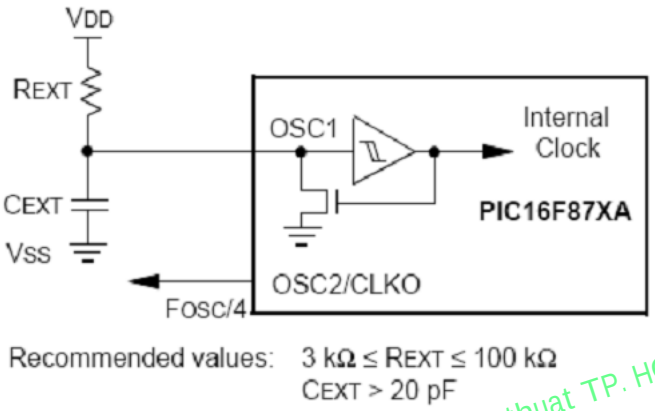
Bảng 2-29 liệt kê tụ C1 và C2:

Osc Type	Crystal Freq.	Cap. Range C1	Cap. Range C2
LP	32 kHz	33 pF	33 pF
	200 kHz	15 pF	15 pF
XT	200 kHz	47-68 pF	47-68 pF
	1 MHz	15 pF	15 pF
	4 MHz	15 pF	15 pF
HS	4 MHz	15 pF	15 pF
	8 MHz	15-33 pF	15-33 pF
	20 MHz	15-33 pF	15-33 pF
These values are for design guidance only. See notes following this table.			
Crystals Used			
32 kHz	Epson C-001R32.768K-A	± 20 PPM	
200 kHz	STD XTL 200.000KHz	± 20 PPM	
1 MHz	ECS ECS-10-13-1	± 50 PPM	
4 MHz	ECS ECS-40-20-1	± 50 PPM	
8 MHz	EPSON CA-301 8.000M-C	± 30 PPM	
20 MHz	EPSON CA-301 20.000M-C	± 30 PPM	

Bảng 2-29. Chọn các thạch anh và tụ.

c. Bộ dao động RC:

Trong những ứng dụng đòi hỏi về tốc độ, chọn bộ dao động RC để tiết kiệm chi phí. Tần số của bộ dao động RC là hàm của điện áp nguồn cung cấp, giá trị của điện trở (R_{EXT}) và tụ (C_{EXT}) và nhiệt độ hoạt động. Bên cạnh đó, tần số của bộ dao động sẽ thay đổi từ giá trị này đến giá trị khác liên quan đến các biến tham số của tiến trình. Hơn nữa, sự khác nhau về giá trị điện dung giữa các kiểu đóng gói sẽ ảnh hưởng đến tần số dao động, đặc biệt với giá trị tụ C_{EXT} thấp. Hình 2-33 trình bày tổ hợp RC nối với PIC 16F87XA



Hình 2-33. Bộ dao động RC.

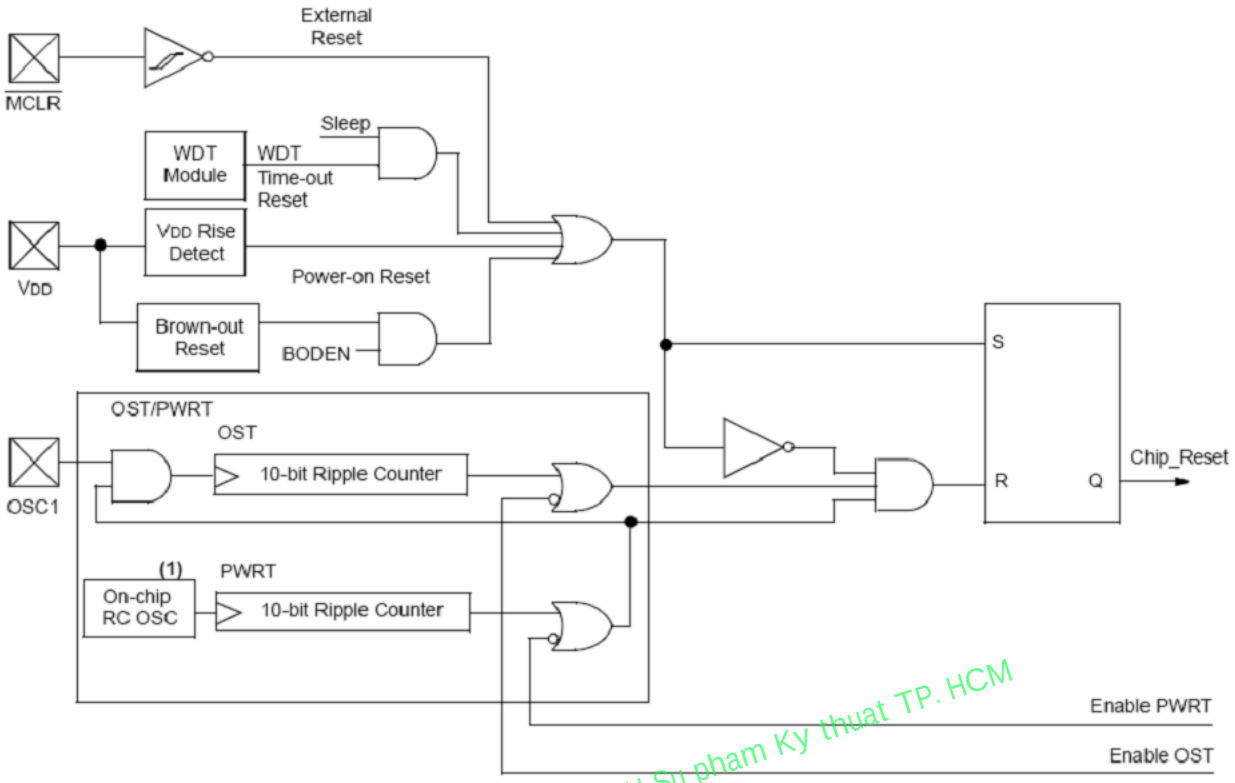
12. MẠCH RESET CPU

PIC16F877A có 6 loại Reset khác nhau:

- Power-on Reset (POR).
- Reset $\overline{MCLR}$ trong lúc hoạt động bình thường.
- Reset $\overline{MCLR}$ trong chế độ Sleep.
- Reset WDT trong chế độ Sleep.
- Khởi động WDT (trong chế độ hoạt động bình thường).
- Brown-out Reset (BOR).

Có một vài thanh ghi không bị ảnh hưởng với bất kì hoạt động reset nào. Trạng thái của chúng là không xác định khi POR và không thay đổi các kiểu reset còn lại. Hầu hết các thanh ghi khác còn lại đều ở Reset đối với reset POR, reset $\overline{MCLR}$ và reset WDT, reset $\overline{MCLR}$ trong chế độ Sleep và reset BOR. Các thanh ghi không ảnh hưởng WDT để đánh thức CPU khỏi chế độ ngủ vào trạng thái hoạt động bình thường trở lại. Các bit $\overline{TO}$ và $\overline{PD}$ được set hoặc xóa tùy thuộc vào tình huống Reset khác nhau. Bảng 2-22 diễn tả các trạng thái reset của tất cả các thanh ghi.

Sơ đồ khối đơn giản của mạch reset trên chip như hình 2-34.



Note 1: This is a separate oscillator from the RC oscillator of the CLKI pin.

Hình 2-34. Sơ đồ mạch reset trong chip.

$\overline{POR}$	$\overline{BOR}$	$\overline{TO}$	$\overline{PD}$	Condition
0	x	1	1	Power-on Reset
0	x	0	x	Illegal, $\overline{TO}$ is set on $\overline{POR}$
0	x	x	0	Illegal, $\overline{PD}$ is set on $\overline{POR}$
1	0	1	1	Brown-out Reset
1	1	0	1	WDT Reset
1	1	0	0	WDT Wake-up
1	1	u	u	$\overline{MCLR}$ Reset during normal operation
1	1	1	0	$\overline{MCLR}$ Reset during Sleep or Interrupt Wake-up from Sleep

Bảng 2-30. Chọn các thạch anh và tụ.

Condition	Program Counter	Status Register	PCON Register
Power-on Reset	000h	0001 1xxx	---- --0x
$\overline{MCLR}$ Reset during normal operation	000h	000u uuuu	---- --uu
$\overline{MCLR}$ Reset during Sleep	000h	0001 0uuu	---- --uu
WDT Reset	000h	0000 1uuu	---- --uu
WDT Wake-up	PC + 1	uuu0 0uuu	---- --uu
Brown-out Reset	000h	0001 1uuu	---- --u0
Interrupt Wake-up from Sleep	PC + 1(1)	uuu1 0uuu	---- --uu

Bảng 2-31. Chọn các thạch anh và tụ.

Chú ý: (1) khi thoát khỏi chế độ ngủ tuỳ thuộc vào ngắt xảy ra và bit GIE bị set lên 1, nội dung thanh ghi PC được nạp vector địa chỉ ngắt là 0004h.

Giá trị của các thanh ghi khi bị reset:

Register	Devices				Power-on Reset, Brown-out Reset	$\overline{\text{MCLR}}$ Resets, WDT Reset	Wake-up via WDT or Interrupt
W	73A	74A	76A	77A	xxxx xxxx	uuuu uuuu	uuuu uuuu
INDF	73A	74A	76A	77A	N/A	N/A	N/A
TMR0	73A	74A	76A	77A	xxxx xxxx	uuuu uuuu	uuuu uuuu
PCL	73A	74A	76A	77A	0000 0000	0000 0000	PC + 1(2)
STATUS	73A	74A	76A	77A	0001 1xxx	000q quuu(3)	uuuq quuu(3)
FSR	73A	74A	76A	77A	xxxx xxxx	uuuu uuuu	uuuu uuuu
PORTA	73A	74A	76A	77A	--0x 0000	--0u 0000	--uu uuuu
PORTB	73A	74A	76A	77A	xxxx xxxx	uuuu uuuu	uuuu uuuu
PORTC	73A	74A	76A	77A	xxxx xxxx	uuuu uuuu	uuuu uuuu
PORTD	73A	74A	76A	77A	xxxx xxxx	uuuu uuuu	uuuu uuuu
PORTE	73A	74A	76A	77A	---- -xxx	---- -uuu	---- -uuu
PCLATH	73A	74A	76A	77A	---0 0000	---0 0000	---u uuuu
INTCON	73A	74A	76A	77A	0000 000x	0000 000u	uuuu uuuu(1)
PIR1	73A	74A	76A	77A	r000 0000	r000 0000	ruuu uuuu(1)
	73A	74A	76A	77A	0000 0000	0000 0000	uuuu uuuu(1)
PIR2	73A	74A	76A	77A	-0-0 0--0	-0-0 0--0	-u-u u--u(1)
TMR1L	73A	74A	76A	77A	xxxx xxxx	uuuu uuuu	uuuu uuuu
TMR1H	73A	74A	76A	77A	xxxx xxxx	uuuu uuuu	uuuu uuuu
T1CON	73A	74A	76A	77A	--00 0000	--uu uuuu	--uu uuuu
TMR2	73A	74A	76A	77A	0000 0000	0000 0000	uuuu uuuu
T2CON	73A	74A	76A	77A	-000 0000	-000 0000	-uuu uuuu
SSPBUF	73A	74A	76A	77A	xxxx xxxx	uuuu uuuu	uuuu uuuu
SSPCON	73A	74A	76A	77A	0000 0000	0000 0000	uuuu uuuu
CCPR1L	73A	74A	76A	77A	xxxx xxxx	uuuu uuuu	uuuu uuuu
CCPR1H	73A	74A	76A	77A	xxxx xxxx	uuuu uuuu	uuuu uuuu
CCP1CON	73A	74A	76A	77A	--00 0000	--00 0000	--uu uuuu
RCSTA	73A	74A	76A	77A	0000 000x	0000 000x	uuuu uuuu
TXREG	73A	74A	76A	77A	0000 0000	0000 0000	uuuu uuuu
RCREG	73A	74A	76A	77A	0000 0000	0000 0000	uuuu uuuu
CCPR2L	73A	74A	76A	77A	xxxx xxxx	uuuu uuuu	uuuu uuuu
CCPR2H	73A	74A	76A	77A	xxxx xxxx	uuuu uuuu	uuuu uuuu
CCP2CON	73A	74A	76A	77A	0000 0000	0000 0000	uuuu uuuu
ADRESH	73A	74A	76A	77A	xxxx xxxx	uuuu uuuu	uuuu uuuu
ADCON0	73A	74A	76A	77A	0000 00-0	0000 00-0	uuuu uu-u
OPTION_REG	73A	74A	76A	77A	1111 1111	1111 1111	uuuu uuuu
TRISA	73A	74A	76A	77A	--11 1111	--11 1111	--uu uuuu
TRISB	73A	74A	76A	77A	1111 1111	1111 1111	uuuu uuuu
TRISC	73A	74A	76A	77A	1111 1111	1111 1111	uuuu uuuu

Bảng 2-32. Giá trị của các thanh ghi khi bị reset.

Chú ý: (1) Một hoặc nhiều bit trong INTCON, PIR1 và/hoặc PIR2 sẽ bị ảnh hưởng.

Chú ý: (2) khi thoát khỏi chế độ ngủ tuỳ thuộc vào ngắt xảy ra và bit GIE bị set lên 1, nội dung thanh ghi PC được nạp vector địa chỉ ngắt là 0004h.

Chú ý: (3) Xem bảng 2-31 để có các giá trị reset với điều kiện chỉ định.

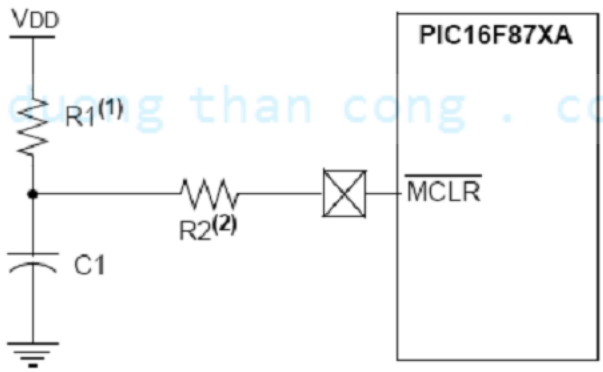
Register	Devices				Power-on Reset, Brown-out Reset	$\overline{MCLR}$ Resets, WDT Reset	Wake-up via WDT or Interrupt
TRISD	73A	74A	76A	77A	1111 1111	1111 1111	uuuu uuuu
TRISE	73A	74A	76A	77A	0000 -111	0000 -111	uuuu -uuu
PIE1	73A	74A	76A	77A	r000 0000	r000 0000	ruuu uuuu
	73A	74A	76A	77A	0000 0000	0000 0000	uuuu uuuu
PIE2	73A	74A	76A	77A	-0-0 0--0	-0-0 0--0	-u-u u--u
PCON	73A	74A	76A	77A	---- --qq	---- --uu	---- --uu
SSPCON2	73A	74A	76A	77A	0000 0000	0000 0000	uuuu uuuu
PR2	73A	74A	76A	77A	1111 1111	1111 1111	1111 1111
SSPADD	73A	74A	76A	77A	0000 0000	0000 0000	uuuu uuuu
SSPSTAT	73A	74A	76A	77A	--00 0000	--00 0000	--uu uuuu
TXSTA	73A	74A	76A	77A	0000 -010	0000 -010	uuuu -uuu
SPBRG	73A	74A	76A	77A	0000 0000	0000 0000	uuuu uuuu
CMCON	73A	974	76A	77A	0000 0111	0000 0111	uuuu uuuu
CVRCON	73A	74A	76A	77A	000- 0000	000- 0000	uuu- uuuu
ADRESL	73A	74A	76A	77A	xxxx xxxx	uuuu uuuu	uuuu uuuu
ADCON1	73A	74A	76A	77A	00-- 0000	00-- 0000	uu-- uuuu
EEDATA	73A	74A	76A	77A	xxxx xxxx	uuuu uuuu	uuuu uuuu
EEADR	73A	74A	76A	77A	xxxx xxxx	uuuu uuuu	uuuu uuuu
EEDATH	73A	74A	76A	77A	xxxx xxxx	uuuu uuuu	uuuu uuuu
EEADRH	73A	74A	76A	77A	xxxx xxxx	uuuu uuuu	uuuu uuuu
EECON1	73A	74A	76A	77A	x--- x000	u--- u000	u--- uuuu
EECON2	73A	74A	76A	77A	---- ----	---- ----	---- ----

Bảng 2-33. Giá trị của các thanh ghi khi bị reset (tiếp tục).

a. Reset $\overline{MCLR}$:

PIC16F877A có bộ lọc nhiễu cho ngõ vào của chân reset $\overline{MCLR}$. Bộ lọc nhiễu sẽ tách và hủy các xung nhỏ.

Nên chú ý rằng reset WDT không điều khiển chân $\overline{MCLR}$ xuống mức thấp.



Hình 2-34. Mạch reset.

Chú ý: (1) Giá trị được đề nghị cho điện trở là R1< 40kΩ để đảm bảo rằng điện áp rơi trên điện trở không vượt quá các thông số chỉ định.

Chú ý: (2) Giá trị yêu cầu cho điện trở là R2 > 1kΩ để hạn chế dòng điện chạy vào chân $\overline{MCLR}$ từ tụ C bên ngoài, trong trường hợp chân $\overline{MCLR}/V_{PP}$ sụt áp liên quan đến phóng tĩnh điện.

b. Reset khi mới cấp điện POR:

Một xung reset khi mới cấp điện được tạo ra trên chip khi phát hiện nguồn V_{DD} tăng giá trị nằm trong khoảng từ 1,2V đến 1,7V. Để tạo thuận lợi cho reset POR, ta nối chân $\overline{MCLR}$ đến V_{DD} thông qua mạch RC như đã trình bày ở trên.

Khi chip bắt đầu làm việc bình thường với các thông số làm việc như điện áp, tần số, nhiệt độ, ... phải đúng để đảm bảo cho hoạt động của mạch. Nếu những điều kiện này không đúng thì chip bị giữ ở trạng thái Reset cho đến khi đúng. Reset Brown-out có thể được dùng để làm đúng các điều kiện khởi động.

c. Timer reset khi mới cấp điện (PWRT):

PWRT tạo ra thời gian 72ms chờ mở nguồn từ POR. PWRT hoạt động dựa vào bộ dao động RC bên trong. Chip được giữ ở trạng thái Reset trong suốt khoảng thời gian PWRT hoạt động. Thời gian trì hoãn của PWRT cho phép nguồn V_{DD} tăng đến giá trị có thể chấp nhận được. Một bit định cấu hình sẽ cho phép hoặc không cho phép PWRT.

Thời gian trì hoãn mở nguồn sẽ thay đổi đối với các chip khác nhau liên quan đến V_{DD}, nhiệt độ và các biến xử lý.

d. Bộ dao động Start-up (OST):

Bộ dao động OST cho phép trể 1024s chu kì dao động (từ ngõ vào OSC1) sau khi delay PWRT đã kết thúc (nếu PWRT được phép). Thời gian trể này giúp cho dao động thạch anh mạch cộng hưởng được bắt đầu hoạt động và ổn định.

Bộ dao động OST chỉ dùng ở các kiểu reset XT, LP và HS và chỉ dùng cho rest POR hoặc đánh thức CPU khỏi chế độ ngủ.

e. Reset Brown-out (BOR):

Bit định cấu hình BODEN có thể cho phép hoặc không cho phép mạch reset BOR. Nếu điện áp V_{DD} giảm xuống dưới mức V_{BOR} (khoảng 4V) trong khoảng thời gian dài hơn T_{BOR} (khoảng 100µs) thì xảy ra reset Brown-out sẽ reset chip. Nếu V_{DD} giảm xuống dưới mức V_{BOR} trong khoảng thời gian ngắn hơn T_{BO} thì reset không thể xảy ra.

Khi Brown-out xảy ra, chip sẽ duy trì ở trạng thái Brown-out Reset cho đến khi điện áp V_{DD} tăng lớn hơn V_{BOR}. Sau đó Power-up Timer giữ chip ở trạng thái Reset trong thời gian T_{PWRT} (72ms). Nếu V_{DD} giảm xuống dưới mức V_{BOR} trong thời gian T_{PWRT} thì tiến trình Brown-out Reset sẽ khởi động trở lại khi V_{DD} tăng lên lớn hơn điện áp V_{BOR} với reset Power-up Timer. Power-up Timer luôn được phép khi mạch điện Brown-out Reset được phép bất chấp trạng thái của bit định cấu hình PWRT.

f. Trình tự thời gian:

Khi mở nguồn, trình tự thời gian chờ là như sau: thời gian trì hoãn PWRT bắt đầu khi reset POR xảy ra. Sau đó, OST bắt đầu đếm 1024 chu kỳ dao động khi PWRT kết thúc. Khi OST kết thúc thì chip mới thoát khỏi trạng thái Reset.

Nếu chân $\overline{MCLR}$ được giữ ở mức thấp trong thời gian đủ dài thì thời gian nghỉ sẽ kết thúc. Sau đó, khi chân $\overline{MCLR}$ ở mức cao thì chip bắt đầu thực hiện ngay lập tức. Điều này hữu dụng hơn cho các chức năng kiểm tra hoặc để đồng bộ nhiều PIC16F877A hoạt động song song.

g. Thanh ghi trạng thái/thanh ghi công suất:

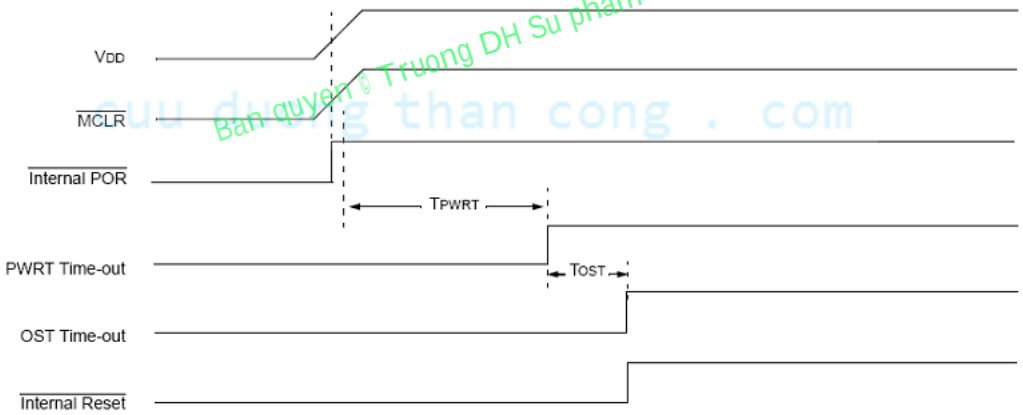
Thanh ghi công suất/ trạng thái PCON có 2 bit phụ thuộc vào chip.

Bit thứ 0 là bit trạng thái Brown-out Reset $\overline{BOR}$ có giá trị không xác định khi reset POR. Sau đó người dùng phải set bit này lên 1 và kiểm tra các trạng thái Reset xảy ra sau đó để xem bit có bị xóa, để xác định rằng reset BOR đã xảy ra.

Khi Brown-out Reset không được cho phép thì trạng thái của bit $\overline{BOR}$ là không xác định.

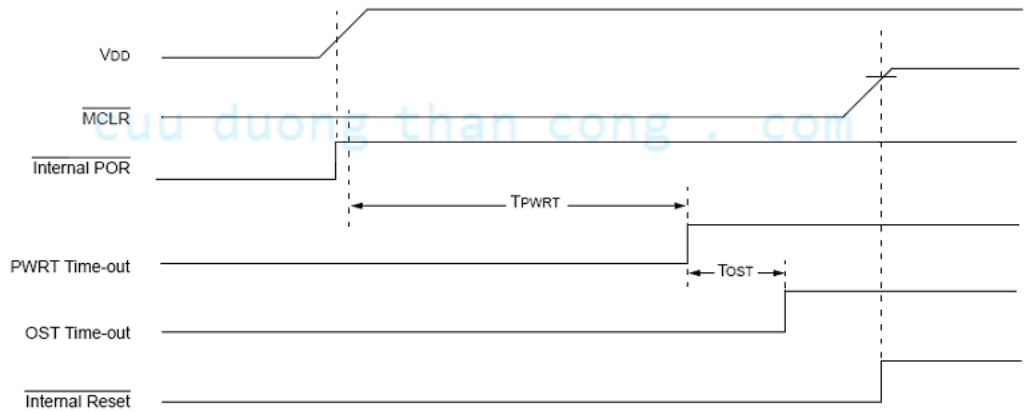
Bit thứ 1 là bit trạng thái $\overline{POR}$ của reset POR. Nó bị xóa khi reset POR và không ảnh hưởng của các kiểu reset khác. Người sử dụng phải set bit này sau khi reset POR.

Trình tự thời gian khi cấp điện như hình 2-35 với ngõ vào $\overline{MCLR}$ nối với V_{DD} qua mạch RC:



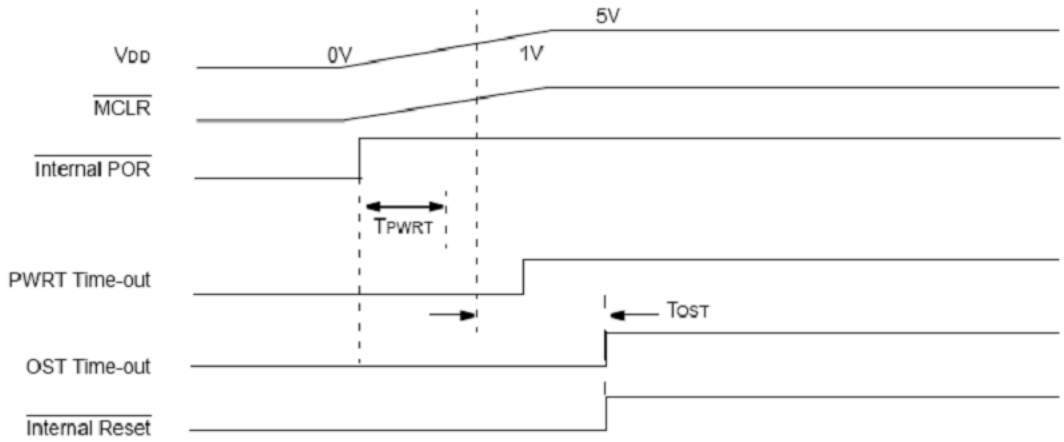
Hình 2-35. Trình tự thời gian khi reset POR có nối $\overline{MCLR}$.

Trình tự thời gian khi cấp điện như hình 2-36 với ngõ vào $\overline{MCLR}$ không nối với V_{DD}:



Hình 2-36. Trình tự thời gian khi reset POR không nối $\overline{MCLR}$.

Trình tự thời gian khi nguồn tăng chậm như hình 2-37 với ngõ vào $\overline{MCLR}$ nối với V_{DD} qua mạch RC:



Hình 2-37. Trình tự thời gian khi reset POR không nối $\overline{MCLR}$.

13. HOẠT ĐỘNG NGẮT:

PIC16F87XA có 15 nguồn ngắt. Thanh ghi điều khiển ngắt (INTCON) ghi nhận những yêu cầu ngắt độc lập ở các bit cờ và chứa các bit cho phép ngắt riêng và ngắt toàn cục.

Chú ý: Các cờ báo ngắt được set lên 1 bất chấp trạng thái của ngắt tương ứng cho hay không cho phép hoặc bit ngắt toàn cục GIE.

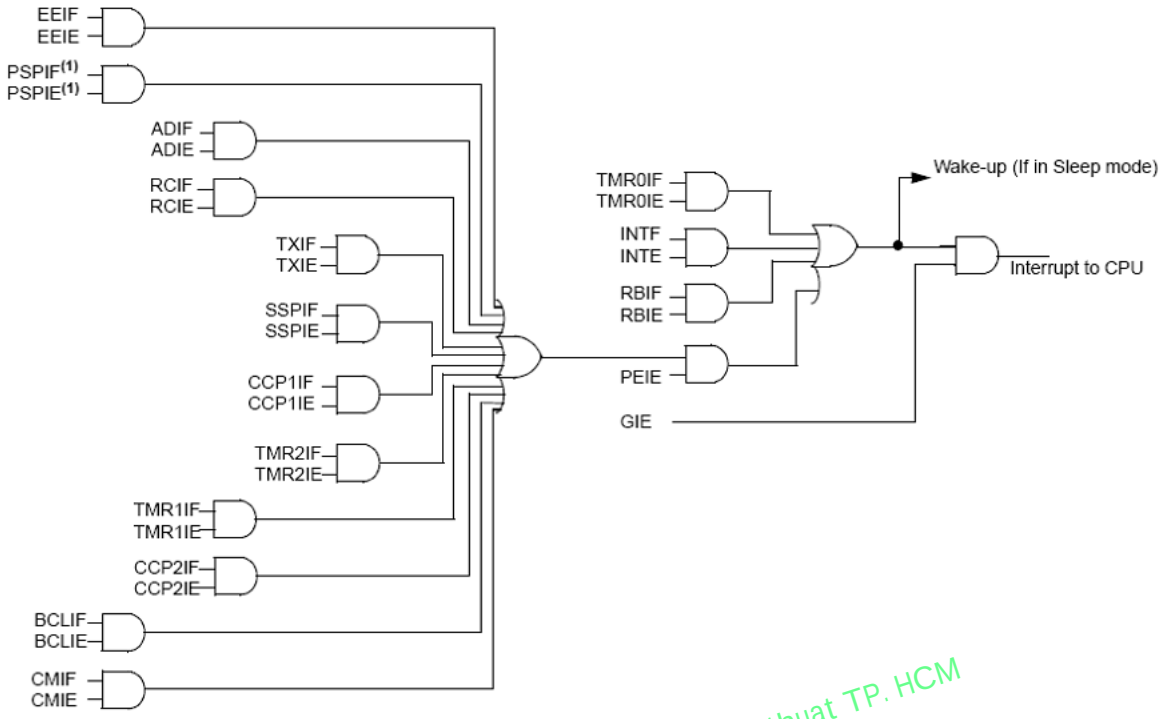
Bit cho phép ngắt toàn cục GIE (INTCON<7>) nếu bằng 1 thì cho phép tất cả các nguồn ngắt và nếu bằng 0 thì không cho phép tất cả các ngắt. Khi bit GIE được phép và nếu bit cờ ngắt và bit cho phép ngắt độc lập lên 1 thì ngắt sẽ xảy ra ngay lập tức. Các nguồn ngắt độc lập có thể cho phép hoặc cấm bởi các bit cho phép ngắt riêng tương ứng. Các bit ngắt riêng được set bất chấp trạng thái bit GIE. Bit GIE bị xóa khi reset.

Lệnh RETFIE là lệnh kết thúc chương trình con phục vụ ngắt trở về chương trình chính, và set bit GIE để cho phép ngắt trở lại.

Ngắt ở chân RB0/INT, ngắt khi có thay đổi PORTB và ngắt cờ tràn của TMR0 được chứa trong thanh ghi INTCON.

Các cờ ngắt ngoại vi được chứa trong thanh ghi đặc biệt PIR1 và PIR2. Các bit cho phép ngắt tương ứng chứa trong các thanh ghi đặc biệt PIE1 và PIE2, bit cho phép ngắt ngoại vi chứa trong thanh ghi đặc biệt INTCON.

Khi một ngắt được đáp ứng thì bit GIE bị xóa để không cho phép bất kì hoạt động ngắt nào xảy ra nữa, địa chỉ trở về được cất vào trong ngăn xếp và PC được nạp địa chỉ 0004h. Trong mỗi chương trình con phục vụ ngắt, nguồn ngắt có thể được xác định bằng cách kiểm tra các bit cờ báo ngắt. Bit cờ ngắt phải xóa trong phần mềm trước khi cho phép ngắt trở lại để tránh gọi lại ngắt đã thực hiện.



Hình 2-38. Sơ đồ logic của các ngắt.

a. Ngắt ngoài INT:

Ngắt ngoài trên chân RB0/INT kích bằng cạnh lên nếu bit INTEDG (OPTION_REG<6>) bằng 1 và kích bằng cạnh xuống nếu bit INTEDG bằng 0. Khi có cạnh tích cực xuất hiện trên chân RB0/INT thì bit cờ INTF (INTCON<1>) bị set lên mức 1. Ngắt RB0/INT này có thể cấm bằng cách xóa bit cho phép INTE (INTCON<4>). Bit cờ báo ngắt INTF phải được xóa trong phần mềm trước khi cho phép ngắt trở lại. Ngắt ở chân INT có thể đánh thức vi xử lý khỏi chế độ Sleep nếu bit INTE đã được set trước khi đi vào chế độ Sleep. Trạng thái của bit cho phép ngắt toàn cục GIE quyết định bộ vi xử lí có phân nhánh hay không để cho vector ngắt sau khi bị đánh thức.

b. Ngắt TMR0:

Khi giá trị trong thanh ghi TMR0 tràn từ FFh sang 00h sẽ set bit cờ TMR0IF (INTCON<2>). Ngắt này có thể cho phép/cấm bằng cách set/clear bit cho phép TMR0IE (INTCON<5>).

c. Ngắt PORTB thay đổi:

Khi có thay đổi ở các bit PORTB<4:7> sẽ làm bit cờ RBIF (INTCON<0>) lên 1. Ngắt có thể cho phép/cấm bằng cách set/clear bit cho phép RBIE (INTCON<5>).

d. Lưu dữ liệu khi xảy ra ngắt:

Khi thực hiện ngắt thì chỉ có giá trị trở về của thanh ghi PC được lưu vào bộ nhớ ngăn xếp. Thường thì người sử dụng muốn lưu các thanh ghi quan trọng khi xảy ra ngắt như thanh ghi W và thanh ghi trạng thái. Điều này chỉ được thực hiện trong phần mềm.

Do 16 byte cao nằm trên mỗi bank có trong PIC16F877A chứa tạm thời các thanh ghi W_TEMP, STATUS_TEMP và PCLATH_TEMP, nên phải đặt ở vị trí này. Vị trí 16 byte này không nằm trong bank và vì vậy dễ dàng lưu trữ và phục hồi.

14. HOẠT ĐỘNG CỦA WATCHDOG TIMER WDT:

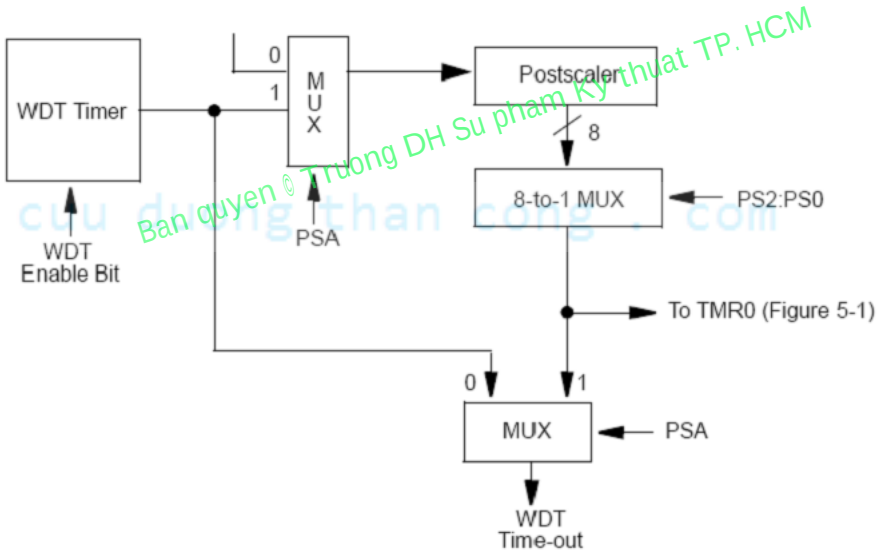
WDT là bộ chạy tự do dựa vào bộ dao động RC của chip mà không yêu cầu bất cứ thành phần nào ở bên ngoài. Bộ dao động RC này được tách từ bộ dao động RC của chân OSC1/CLKI. Điều này có nghĩa là WDT sẽ chạy dù cho xung clock trên các chân OSC1/CLKI và OSC2/CLKO của PIC bị dừng lại, ví dụ khi thực thi lệnh Sleep.

Khi hoạt động việc bình thường, sau khi hết một khoảng thời gian WDT sẽ reset cpu. Nếu cpu đang ở chế độ Sleep, thì sau khi hết thời gian của WDT sẽ đánh thức cpu để trở lại hoạt động bình thương. Bit $\overline{TO}$ trong thanh ghi trạng thái sẽ bị xóa khi WDT đếm đủ thời gian.

WDT có thể bị cấm thường xuyên bằng cách xóa bit định cấu hình WDTE.

Chú ý: (1) Các lệnh CLRWDT và SLEEP xoá WDT và postscaler nếu được gán cho WDT và ngăn chặn WDT hết thời gian để khỏi xảy ra điều kiện reset cpu.

Chú ý: (2) Khi lệnh CLRWDT được thực hiện và bộ chia trước được gán cho WDT, bộ đếm chia trước sẽ bị xoá nhưng giá trị gán cho bộ chia trước vẫn không đổi.



Note: PSA and PS2:PS0 are bits in the OPTION_REG register.

Hình 2-39. Sơ đồ khối của WDT.

Tóm tắt các thanh ghi của WDT

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
2007h	Config. bits	(1)	BODEN(1)	CP1	CP0	$\overline{PWRTE}$(1)	WDTE	Fosc1	Fosc0
81h, 181h	OPTION_REG	RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0

Bảng 2-34. Các thanh ghi của WDT.

15. HOẠT ĐỘNG CỦA CPU Ở CHẾ ĐỘ NGỦ SLEEP:

Chế độ Power-down được nhập vào bởi việc thực thi lệnh Sleep. Nếu được cho phép bộ định thời WDT sẽ bị xóa nhưng vẫn chạy, bit $\overline{PD}$ (Status<3>) bị xóa, bit $\overline{TO}$ (Status<4>) bị set lên mức 1 và bộ điều khiển dao động bị tắt đi. Những port IO vẫn duy trì trạng thái đã xác định trước khi thực hiện lệnh Sleep (ở mức cao, thấp hoặc trở kháng cao).

Để dòng tiêu tốn trong chế độ này là thấp nhất thì phải đặt tất cả các chân IO ở tại điện áp V_{DD} hoặc V_{SS} để bảo đảm không có mạch điện nào ở bên ngoài lấy dòng từ các chân IO, tắt nguồn cung cấp cho khối ADC và không cho phép các nguồn xung clock ở bên ngoài. Kéo tất cả các chân IO mà chúng có ngõ vào trở kháng cao ở mức cao hoặc thấp để tránh những dòng chuyển mạch gây ra bởi ngõ vào thả nỗi. Ngõ vào chân T0CKI cũng nên đặt ở V_{DD} hoặc V_{SS} để cho dòng tiêu tốn là thấp nhất.

Chân $\overline{MCLR}$ phải ở mức logic cao (V_{IHMC}).

a. Đánh thức cpu khỏi chế độ ngủ:

PIC có thể đánh thức cpu khỏi chế độ Sleep thông qua một trong những trường hợp sau:

- ☐ Ngõ vào Reset bên ngoài trên chân $\overline{MCLR}$
- ☐ Sử dụng WDT (nếu WDT được cho phép)
- ☐ Ngắt từ chân INT, ngắt PORT thay đổi hoặc ngắt ngoại vi.

Reset ngoài $\overline{MCLR}$ sẽ làm cho cpu bị reset bắt đầu thực hiện chương trình từ đầu. Những trường hợp còn lại sẽ đánh thức cpu tiếp tục thực hiện chương trình. Các bit $\overline{TO}$ và $\overline{PD}$ trong thanh ghi trạng thái có thể được dùng để xác định nguyên nhân Reset của thiết bị. Bit $\overline{PD}$ được set khi mới cấp nguồn, sẽ bị xóa khi vào chế độ Sleep. Bit $\overline{TO}$ bị xóa nếu thời gian của WDT xảy ra và đánh thức cpu khỏi chế độ ngủ.

Những ngắt ngoại vi theo sau có thể đánh thức cpu khỏi chế độ ngủ:

- Đọc hoặc ghi PSP chỉ có ở PIC 16F874/877.
- Ngắt của TMR1. TMR1 phải làm việc như bộ đếm bất đồng bộ.
- Ngắt kiểu Capture CCP.
- Bộ kích sự kiện đặc biệt (TMR1 trong kiểu bất đồng bộ dùng xung clock bên ngoài).
- Ngắt phát hiện bit SSP (khởi động/dừng).
- Truyền hoặc nhận SSP ở chế độ tớ (SPI/I^2C).
- Truyền dữ liệu bất đồng bộ USART Tx hoặc Rx.
- Chuyển đổi ADC (khi nguồn xung clock của ADC là RC).
- Hoàn thành hoạt động ghi dữ liệu vào bộ nhớ EEPROM.
- Bộ so sánh thay đổi trạng thái ngõ ra.

Các thiết bị ngoại vi khác không thể tạo ra ngắt khi cpu ở trong chế độ ngủ, không có nguồn xung clock. Khi lệnh SLEEP đang được thực thi, thì lệnh kế tiếp (PC+1) được đoán về. Đối với PIC để đánh thức thông qua một sự kiện ngắt thì bit cho phép ngắt tương ứng phải được set lên mức 1. Đánh thức bất chấp trạng thái của bit cho phép ngắt toàn cục GIE.

Nếu bit GIE bị xóa (không cho phép) thì cpu tiếp tục thực thi lệnh sau lệnh Sleep.

Nếu bit GIE được set (cho phép) thì cpu thực thi lệnh đứng sau lệnh Sleep và sau đó rẽ nhánh nhảy đến địa chỉ ngắt (0004h).

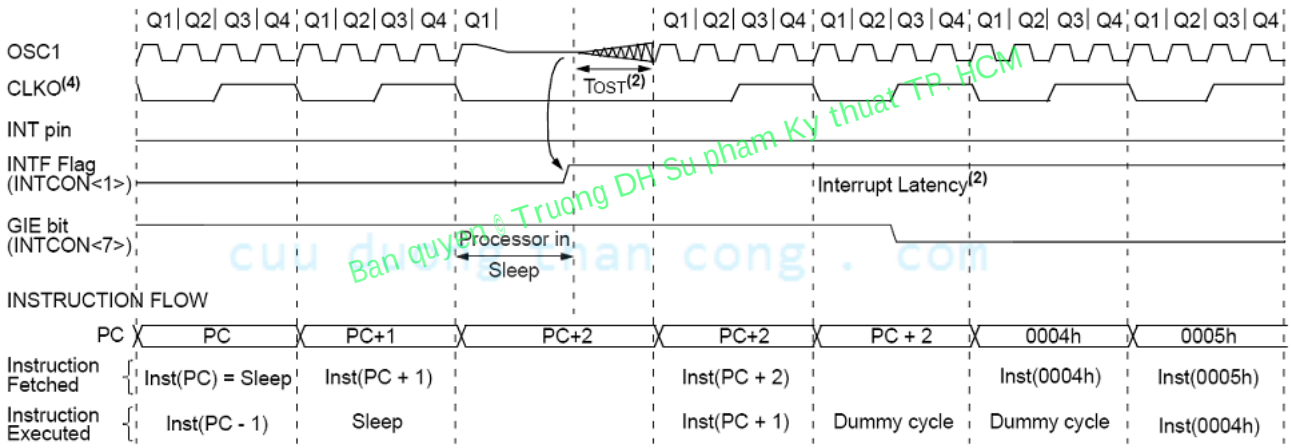
b. Đánh thức cpu dùng các ngắt:

Khi bit ngắt toàn cục là không được cho phép (bit GIE bị xóa) và bất kì nguồn ngắt nào có bit cho phép ngắt và bit cờ ngắt lên 1 thì một trong những sự kiện sau sẽ xảy ra:

- Nếu ngắt xảy ra trước lệnh SLEEP thì lệnh SLEEP sẽ hoàn thành như là lệnh NOP. Vì vậy, WDT và postscaler WDT sẽ không bị xóa, bit $\overline{TO}$ sẽ không được set lên 1 và bit $\overline{PD}$ sẽ không bị xóa.
- Nếu ngắt xảy ra trong lúc và sau khi thực hiện lệnh Sleep, ngay lập tức cpu sẽ bị đánh thức khỏi chế độ ngủ. Lệnh SLEEP sẽ được thực hiện xong trước khi đánh thức cpu. Vì vậy, WDT và postscaler WDT sẽ bị xóa, bit $\overline{TO}$ sẽ được set và bit $\overline{PD}$ sẽ bị xóa.

Thậm chí nếu bit cờ đã được kiểm tra trước khi thực hiện lệnh SLEEP, có khả năng các bit cờ được set trước khi lệnh Sleep hoàn thành. Để xác định lệnh Sleep có thực hiện hay không phải kiểm tra bit $\overline{PD}$. Nếu bit $\overline{PD}$ được set thì lệnh SLEEP được thực hiện như lệnh NOP.

Để chắc chắn WDT bị xóa, lệnh CLRWDT nên thực hiện trước khi lệnh thực hiện SLEEP.



Hình 2-40. Đánh thức cpu bằng cách dùng ngắt.

Chú ý: (1) Sử dụng bộ dao động XT, HS hoặc LP.

Chú ý: (2) TOST = 1024 TOSC.

Chú ý: (3) Nếu bit GIE = 1. Trong trường hợp này sau khi đánh thức thì cpu nhảy đến thực hiện chương trình con phục vụ ngắt. Nếu bit GIE = 0 thì thực hiện tiếp lệnh trong chương trình.

Chú ý: (4) CLKO không có tác dụng trong các kiểu dao động trên nhưng được trình bày để tham khảo.

16. MẠCH GỠ RỐI

Khi bit gỡ rối nằm từ thiết lập cấu hình được lập trình ở mức 0 thì chức năng của mạch gỡ rối được phép. Chức năng này cho phép chức năng gỡ rối khi sử dụng với chương trình MPLAB ICD. Khi vi điều khiển có chức năng này được phép thì một tài nguyên sẽ trở nên mất tác dụng. Bảng sau liệt kê các cấu trúc bị bỏ khi làm việc ở chế độ gỡ rối.

I/O pins	RB6, RB7
Stack	1 level
Program Memory	Address 0000h must be NOP
	Last 100h words
Data Memory	0x070 (0x0F0, 0x170, 0x1F0) 0x1EB-0x1EF

Bảng 2-35. Các tài nguyên của mạch gỡ rối.

Để sử dụng chức năng gỡ rối của vi điều khiển thì phải thiết kế giao tiếp lập trình nối tiếp với các chân $\overline{MCLR}/V_{PP}$, V_{DD}, GND, RB7 và RB6. Kết nối này sẽ giao tiếp với mạch gỡ rối được xây dựng Microchip.

17. KIỂM TRA CHƯƠNG TRÌNH/ BẢO VỆ BẰNG MÃ:

Nếu bit mã bảo vệ không được lập trình thì bộ nhớ chương trình trên chip có thể đọc ra cho mục đích kiểm tra.

18. MÃ NHẬN DẠNG:

Bốn vị trí bộ nhớ có địa chỉ từ 2000h đến 2003h được thiết kế để chứa mã nhận dạng, nơi mà người sử dụng có thể lưu trữ tổng kiểm tra (checksum) hoặc những con số nhận dạng mật mã khác. Những mã này là có thể không cần thiết truy xuất khi cpu hoạt động bình thường nhưng có thể đọc và ghi trong quá trình lập trình/kiểm tra. Chỉ sử dụng 4 bit thấp LSB của các ô nhớ định vị ID.

19. LẬP TRÌNH TUẦN TỰ CỦA MẠCH TÍCH HỢP BÊN TRONG ICSP (In-Circuit Serial Programming):

Vi điều khiển PIC16F87XA có thể lập trình tuần tự khi chúng nằm trong mạch ứng dụng.

Mạch giao tiếp rất đơn giản gồm 5 đường dây: một cho đường xung clock, một cho đường dữ liệu, một đường nguồn, một đường mass và một đường cấp điện áp lập trình. Điều này cho phép khách hàng sản xuất những bo mạch sử dụng cpu chưa được lập trình và sau đó lập trình cho vi điều khiển trước khi giao hàng.

Khi sử dụng lập trình nối tiếp ICSP thì mạch nạp phải cấp nguồn từ 4,5V đến 5,5V nếu cần thực hiện xóa bộ nhớ. Điện áp này dùng để lập trình lại mã bảo vệ, cả trạng thái mở đến trạng thái tắt. Trong tất cả các trường hợp khác còn lại của ICSP thì mạch nạp sử dụng nguồn điện áp bình thường thiết bị có thể được lập trình ở điện áp làm việc bình thường.

20. LẬP TRÌNH ĐIỆN ÁP THẤP ICSP (NGUỒN ĐƠN):

Bit LVP nằm ở từ định cấu hình cho phép lập trình điện áp thấp ICSP. Kiểu lập trình này cho phép vi điều khiển lập trình qua ICSP sử dụng điện áp nguồn V_{DD} nằm giới hạn điện áp làm việc. Trong kiểu lập trình này, chân RB3/PGM được dùng cho chức năng lập trình và không thể làm các chân I/O. Trong quá trình lập trình nguồn V_{DD} được cấp đến chân $\overline{MCLR}$. Để đi vào chế độ lập trình thì nguồn V_{DD} được cấp cho chân RB3/PGM làm cho bit LVP được set lên mức a. Bit LVP mặc định là 1 từ nhà máy chế tạo.

Chú ý: (1) Chế độ lập trình điện áp cao luôn có hiệu lực bất chấp trạng thái của bit LVP bằng cách cấp điện áp V_{IHH} tới chân $\overline{MCLR}$.

Chú ý: (2) Khi ở kiểu lập trình điện áp thấp thì chân RB3 không được sử dụng là I/O.

Chú ý: (3) Khi sử dụng lập trình điện áp thấp ICSP và chức năng kéo PORTB là được phép, bit thứ 3 trong thanh ghi TRISB phải được xóa để không cho phép kéo chân RB3 và đảm bảo hoạt phù hợp cho chip.

Chú ý: (4) Chân RB3 không được thả nỗi nếu LVP được phép. Thiết bị lập trình bên ngoài mặc nhiên phải đặt CPU ở trạng thái hoạt động bình thường. Khi điều khiển RB3 lên mức 1 thì điều khiển CPU chuyển sang chế độ lập trình.

Nếu chế độ lập trình điện áp thấp không được sử dụng, bit LVP có thể được lập trình về '0' và chân RB3/PGM trở thành chân IO. Tuy nhiên bit LVP chỉ có thể được chuyển sang lập trình khi điện áp V_{IHH} xuất hiện trên chân $\overline{MCLR}$. Bit LVP chỉ có thể thay đổi khi sử dụng điện áp cao ở chân $\overline{MCLR}$.

21. SƠ ĐỒ NGUYÊN LÝ GIAO TIẾP GIỮA MÁY TÍNH VÀ PIC 16F877A:

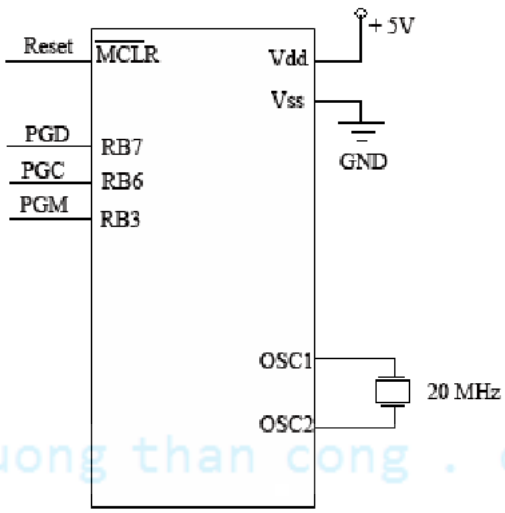
PIC16F877A cho phép nạp chương trình dạng nối tiếp qua cổng LPT và cổng COM.

a. Mạch nạp PIC trực tiếp từ cổng COM:

Khi nạp nối tiếp qua cổng COM cho VĐK PIC16F877A thì cần dùng 4 đường điều khiển:

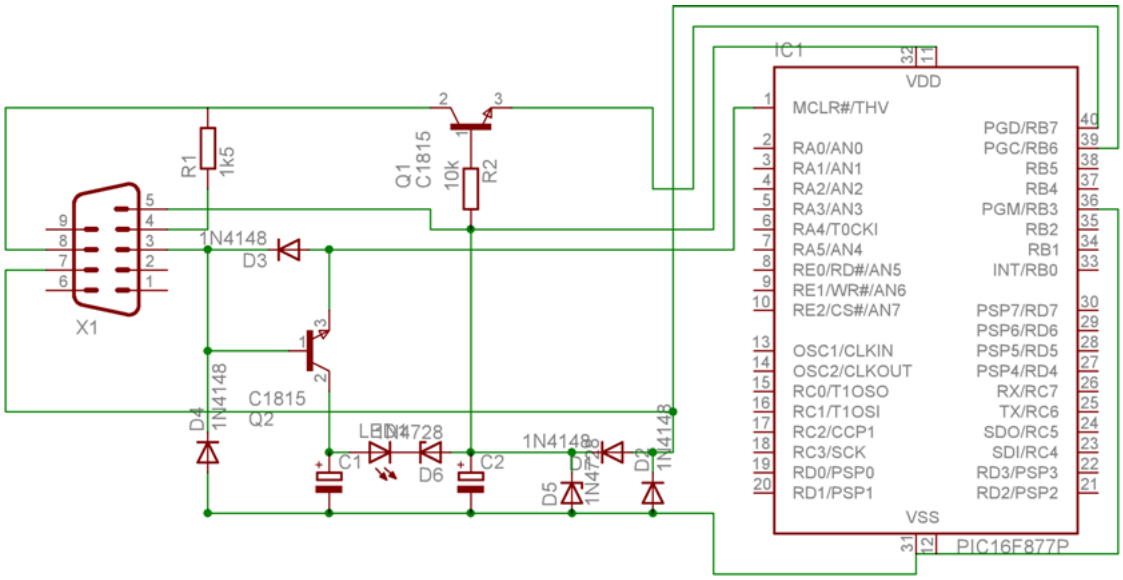
- Chân $\overline{MCLK}$ là đường reset để nạp chương trình cho PIC.
- Chân RB7/PGD.
- Chân RP6/RGC.
- Chân RB3/PGM.

Sơ đồ trình bày các đường tín hiệu điều khiển nạp:



Hình 2-41. Các đường giao tiếp với mạch nạp nối tiếp.

Sơ đồ nguyên lí của mạch nạp trực tiếp qua cổng COM:



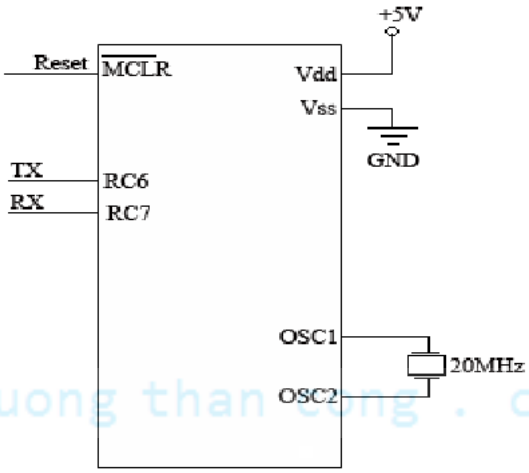
Hình 2-42. Sơ đồ nguyên lý mạch nạp trực tiếp từ cổng COM.

b. Mạch nạp PIC gián tiếp từ cổng COM qua ic max232:

Nạp chương trình cho PIC16F877A thông qua IC MAX232 (hoặc RS232) thì cần có dao động xung clock và kết nối các đường TX và RX của RS-232 với các đường TX và RX vi điều khiển PIC16F877A:

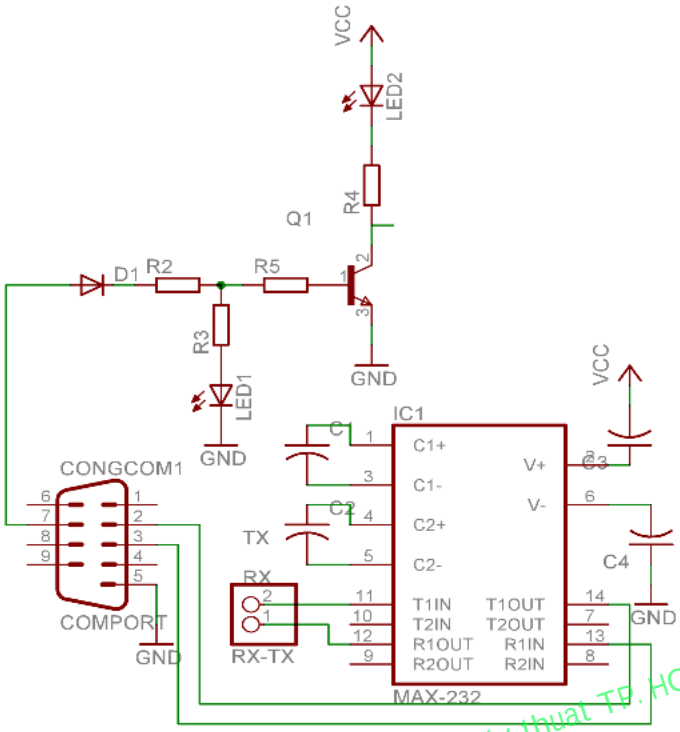
- Chân RX: chân nhận dữ liệu từ máy tính.
- Chân TX: chân truyền dữ liệu về máy tính.

Sơ đồ trình bày các đường tín hiệu nạp cho PIC16F877A qua RS-232:



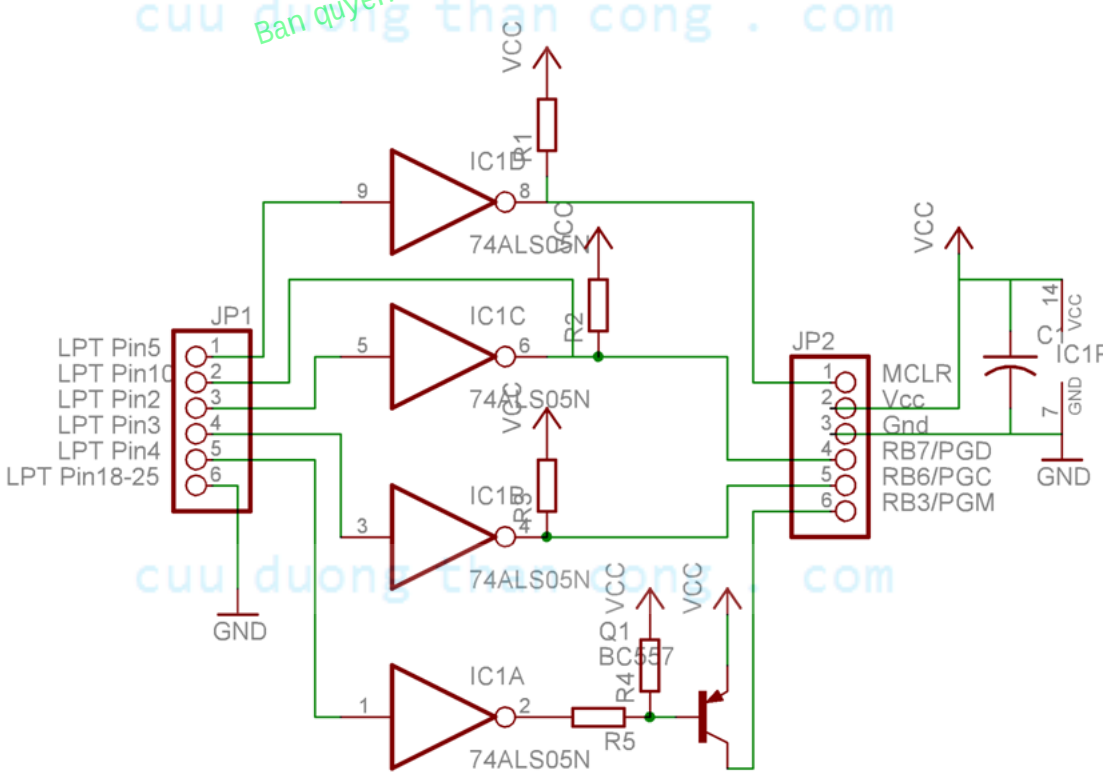
Hình 2-43. Các đường giao tiếp với mạch nạp nối tiếp qua IC chuyển đổi.

Sơ đồ nguyên lí của mạch nạp qua RS-232:



Hình 2-44. Sơ đồ nguyên lý mạch nạp nối tiếp từ cổng COM qua IC chuyển đổi.

c. Mạch nạp PIC qua cổng LPT:



Hình 2-45. Sơ đồ nguyên lý mạch nạp dùng cổng LPT.

the end

return

Chương 3

CHƯƠNG TRÌNH BIÊN DỊCH VÀ NẠP CHO VI ĐIỀU KHIỂN PIC16F877A

Hình và bảng

I. CHƯƠNG TRÌNH BIÊN DỊCH:

Hiện nay có rất nhiều chương trình biên dịch cho PIC viết trên nhiều ngôn ngữ khác nhau như ASM, BASIC, C,... hai phần mềm MPLAB của hãng Microchip và phần mềm CCS C. Ngoài ra còn có các phần mềm biên dịch khác như: Mikro BASIC, Mikro C, HI-TECH, ...

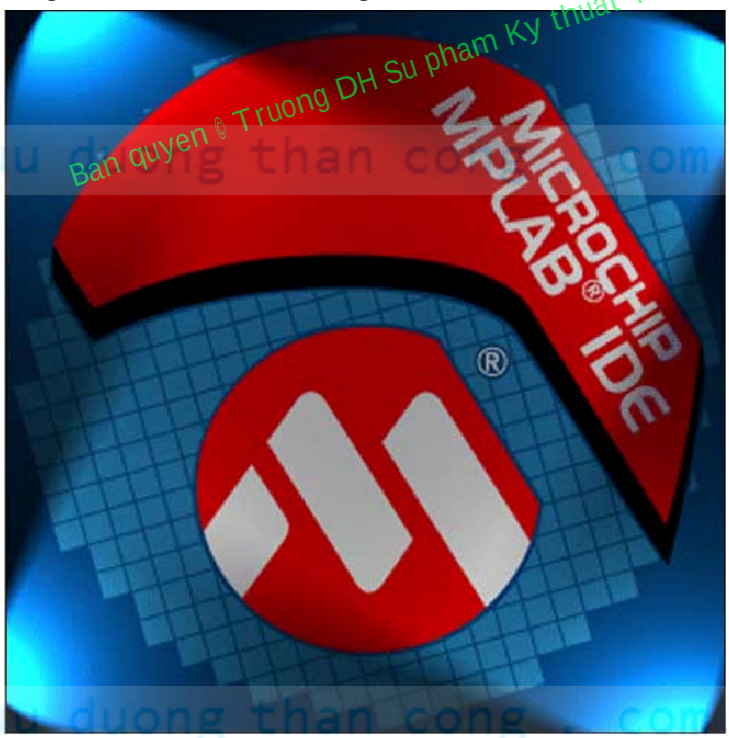
1. CHƯƠNG TRÌNH BIÊN DỊCH MPLAB IDE:

Chương trình biên dịch MPLAB IDE của hãng Microchip cho miễn phí tại website http://www.microchip.com.

Phần mềm MPLAB IDE tương thích với hệ điều hành:

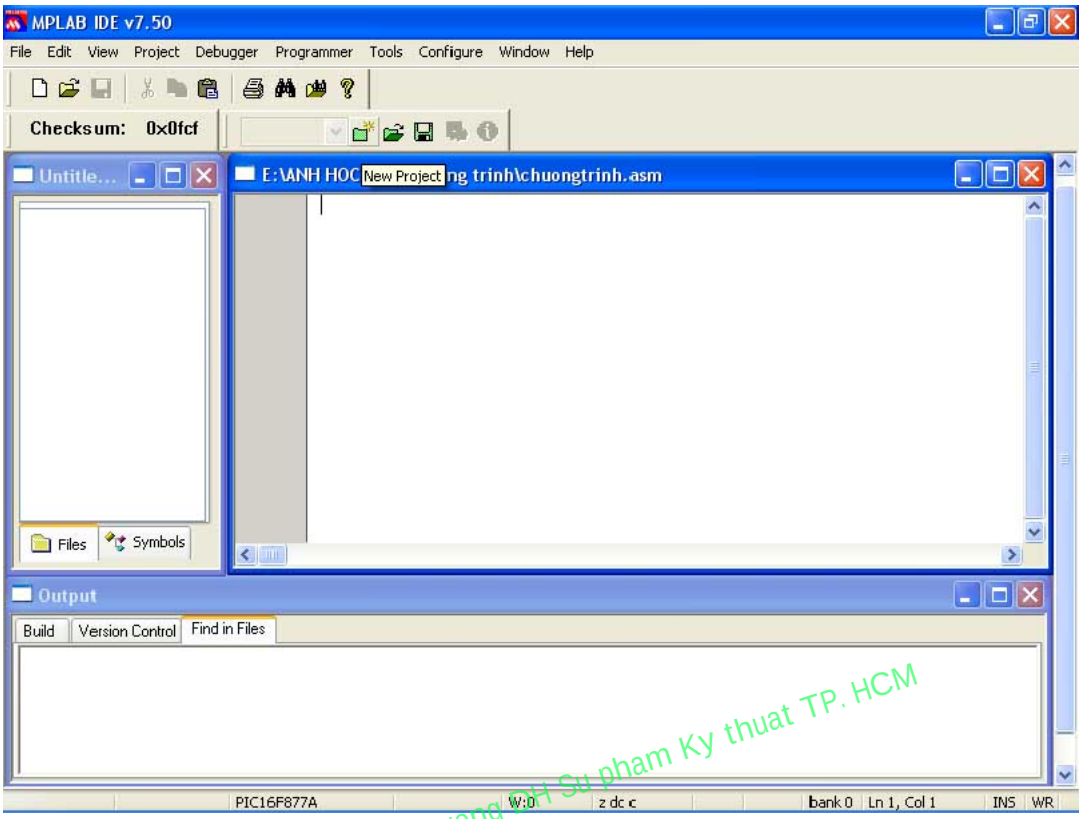
- ☐ Windows 98 SE
- ☐ Windows ME
- ☐ Windows NT 4.0 SP6a Workstations (NOT Servers)
- ☐ Windows 2000 SP2
- ☐ Windows XP Home and Professional

Sau khi cài đặt xong thì click vào biểu tượng  thì màn hình sẽ xuất hiện biểu tượng:



Hình 3-1. Cửa sổ khởi động.

Sau đó có màn hình soạn thảo như sau:



Hình 3-2. Cửa sổ làm việc của MPLAB.

Khi muốn biên dịch từ file .ASM sang file .HEX vào menu Project rồi chọn Build all hoặc QuickBuild để biên dịch.

Nếu chương trình viết bị lỗi thì tại cửa sổ Output sẽ xuất hiện một thông báo là biên dịch thất bại (BUILD FAILED) và số lỗi của chương trình với vị trí của từng lỗi nằm trong chương trình.

Khi dùng MPASM, các số có thể được biên dịch một trong các hệ thống số cơ bản. Mặc định cho file nguồn có thể được thiết lập bằng chỉ dẫn Radix:

```
Radix    dec
```

Bên trong file nguồn, giá trị mã có thể nhập vào các cơ số khác nhau sử dụng cấu trúc sau:

```
D‘123’  .123              ;       thập phân
H‘1AF’ 0x1F               ;       thập lục phân
O‘777’                    ;       bát phân
B ‘00111001’              ;       nhị phân
0B00111001                ;       nhị phân
‘A’              ‘C’      ;       7-bit ASCII
dt ‘This is a string’     ;       dãy ASCII
```

Cấu trúc một chương trình ASM trong MPLAB như sau:

```
Title “tên gọi của chương trình”
Include <p16f877a.inc>; Tên PIC cần viết chương trình
_CONFIG CP_OFF &.....              ; khai báo cho PIC
;-------- Khai báo biến----------------------
temp    EQU             0x20       ; đặt biến có tên temp có địa chỉ là ô nhớ 0x20
```

```
;-------------------------------------------------
ORG                     0x0000    ;vector reset
GOTO                    START
 ;---------------Chương trình ngắt---------
 ORG    0x0004                    ;vector interrupt
                                  ; ....mã ngắt ở đây.
 RETFIE                           ; thoát khỏi chương trình ngắt
 ;----------------Kết thúc chương trình ngắt-----
 ;==========Chương trình chính=========
 Start
        mã chương trình chính ở đây

 END                              ;kết thúc chương trình chính
 ;================================
```

Công cụ MPLAB SIM trong MPLAB IDE (công cụ mô phỏng cho chương trình):

Chọn Debugger → Select Tool để chọn công cụ, sau đó chọn MPLAB SIM. Đây là công cụ mô phỏng dùng để giả lập tín hiệu điện của các chân và trạng thái các thanh ghi của con chip được dùng. Có hai loại: đồng bộ và không đồng bộ.

Đồng bộ: tín hiệu được giả lập đồng bộ với những vòng lệnh của chip.

Không Đồng bộ: tín hiệu được áp đặt bởi người dùng trong thời gian thực (real time) khi MPLAB SIM đang chạy.

2. CHƯƠNG TRÌNH BIÊN DỊCH CCS C:

Chương trình biên dịch CCS C được cung cấp tại địa chỉ:
http://www.ccsinfo.com/download.shtml.

Vì là trình biên dịch có thu phí nên phiên bản demo có một số hạn chế so với phiên bản có thu phí.

CCS là trình biên dịch lập trình ngôn ngữ C cho Vi điều khiển PIC của hãng Microchip.

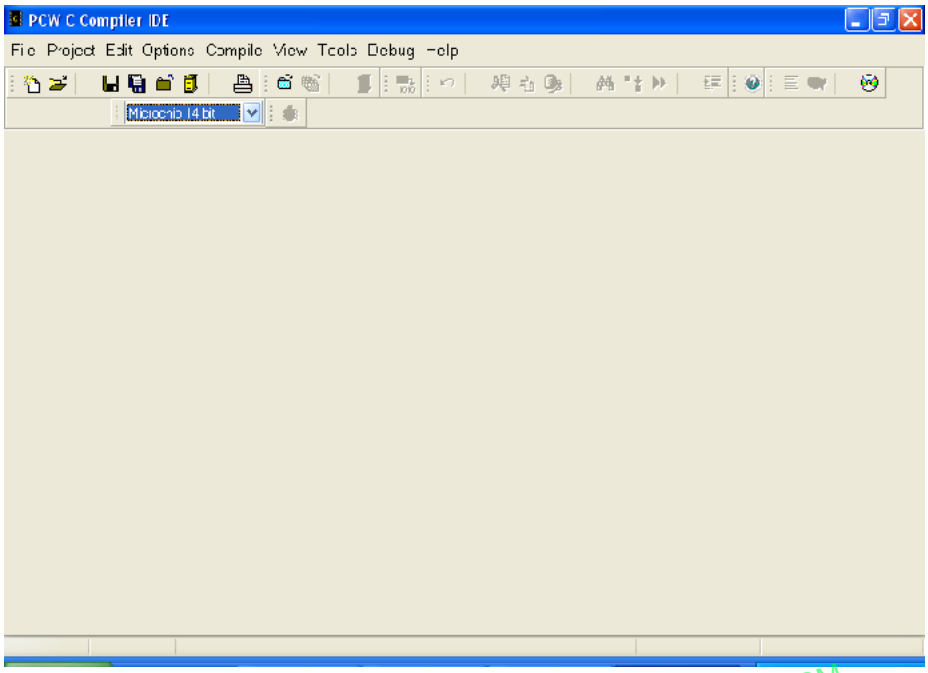
Chương trình là sự tích hợp của 3 trình biên dich riêng biệt cho 3 dòng PIC khác nhau đó là:

- PCB cho dòng PIC 12bit opcodes
- PCM cho dòng PIC 14bit opcodes
- PCH cho dòng PIC 16 và 18bit

Tất cả 3 trình biên dịch này được tích hợp lại vào trong một chương trình bao gồm cả trình soạn thảo và biên dịch là CCS.

Giống như nhiều trình biên dịch C khác cho PIC, CCS giúp cho người sử dụng nắm bắt nhanh được vi điều khiển PIC và sử dụng PIC trong các dự án. Các chương trình điều khiển sẽ được thực hiện nhanh chóng và đạt hiệu quả cao thông qua việc sử dụng ngôn ngữ lập trình cấp cao – ngôn ngữ C.

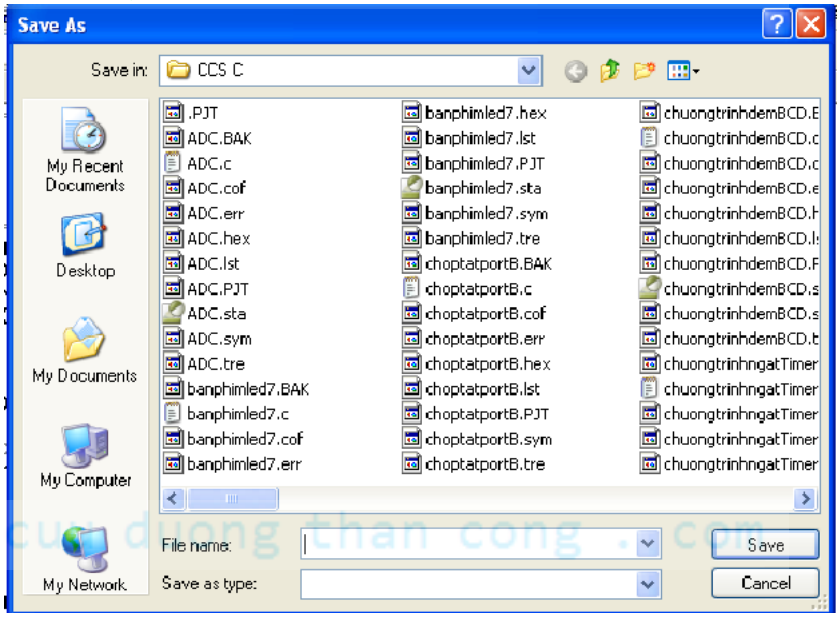
Khi khởi động chương trình CCS thì cửa sổ chương hình như hình dưới:



Hình 3-3. Màn hình khởi động của CCS C.

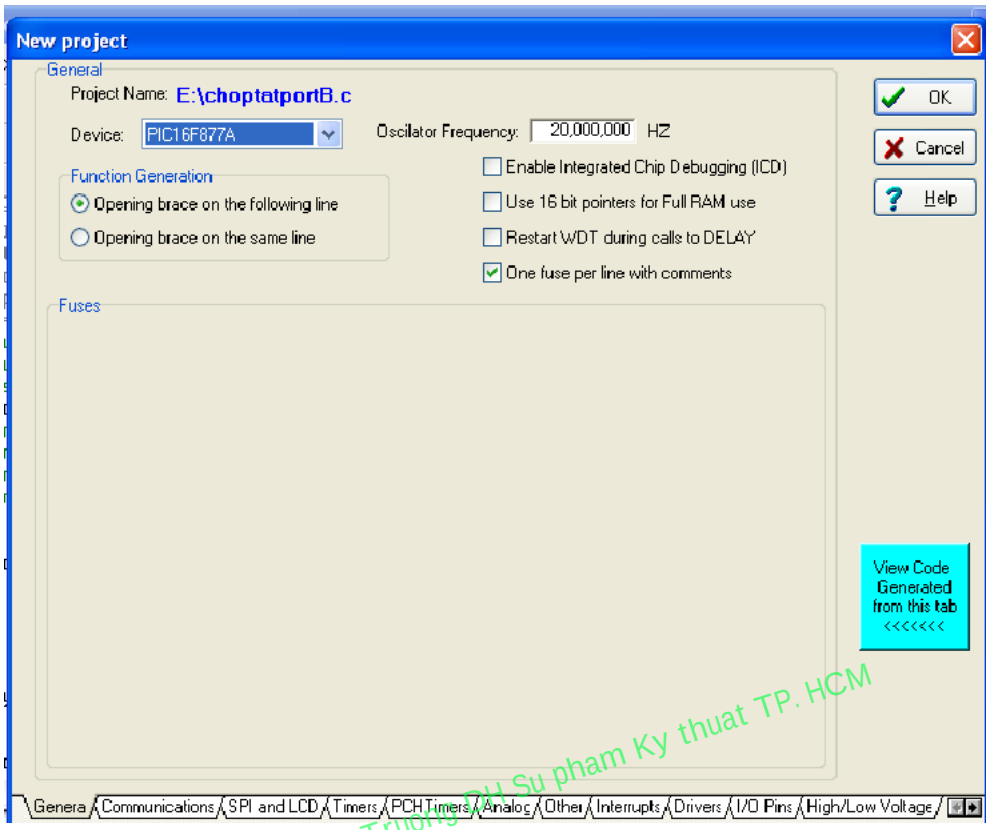
Hướng dẫn tạo một Project mới trong CCS:

Để tạo một Project trong CCS có nhiều cách, có thể dùng Project Wizard, Manual Create, hay là tạo một Files mới và thêm vào đó các khai báo ban đầu cần thiết. Vào Project →chọn PIC Wizard sau khi chọn một cửa sổ hiện ra yêu cầu nhập tên file cần tạo như hình sau:



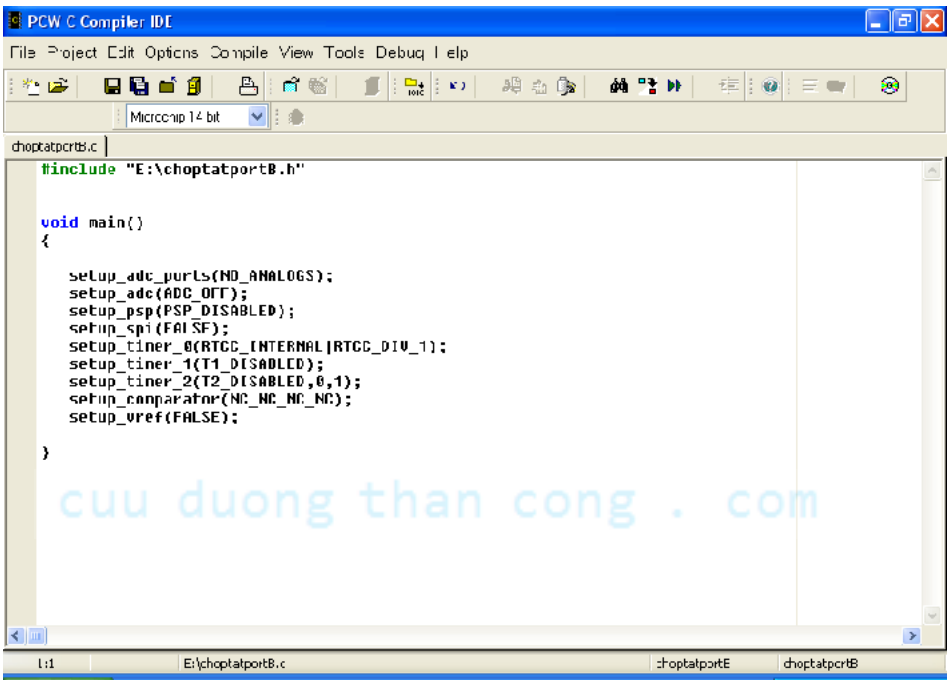
Hình 3-4. Lưu file.

Chọn Save một cửa sổ mới hiện ra như hình sau:



Hình 3-5. Tạo Project mới.

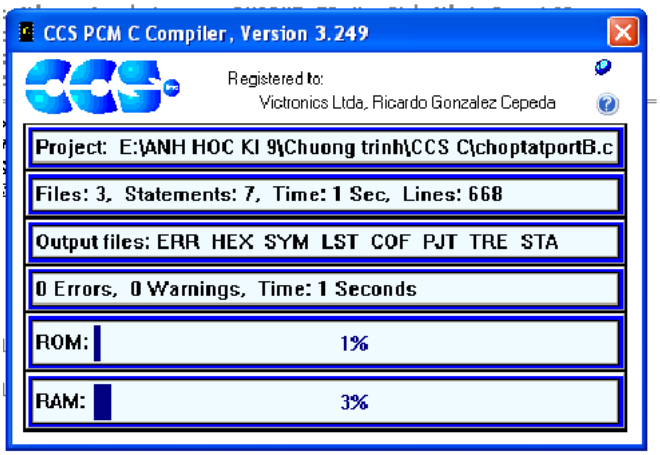
Sau đó nhấp OK là đã tạo được một Project mới và có cửa sổ làm việc mới như hình sau:



Hình 3-6. Cửa sổ làm việc của CCSC.

Như vậy, chúng ta đã tạo được một Project mới và tiến hành viết chương trình cho PIC.

Khi muốn biên dịch từ file *.c sang file *.Hex thì vào Compile → chọn Compile hoặc bấm F9 thì CCS sẽ tiến hành biên dịch file *.c sang file *.Hex để nạp cho PIC. Khi biên dịch thì trình biên dịch sẽ xuất hiện cửa sổ như hình sau là chương trình biên dịch thành công (chương trình không có lỗi về cấu trúc lệnh).



Hình 3-7. Thông báo sau khi biên dịch.

Nếu chương trình viết có lỗi thì khi biên dịch sẽ báo lỗi tại vị trí con trỏ ở trong chương trình.

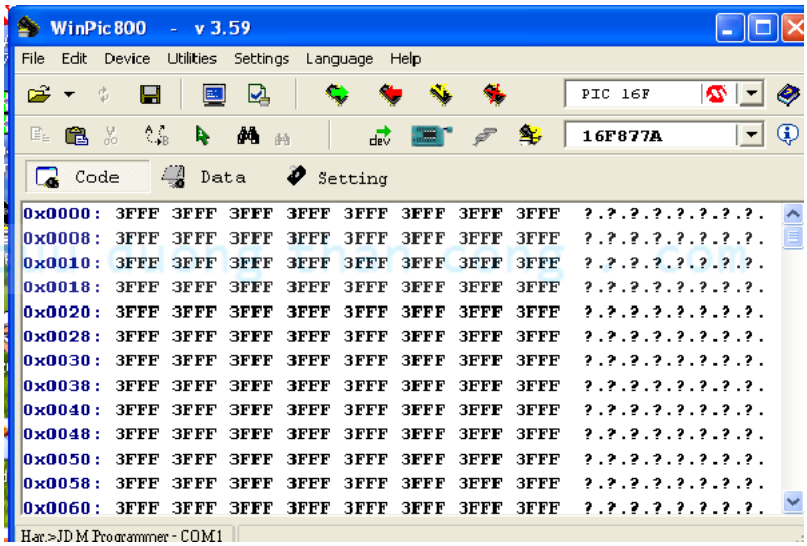
II. CHƯƠNG TRÌNH NẠP CHO PIC:

Hiện nay có rất nhiều phần mềm nạp khác nhau cho PIC như phần mềm nạp Winpic800 và IC-Pro để giới thiệu vì hai phần mềm này được sử dụng nhiều và được cộng đồng sử dụng PIC đánh giá tốt.

1. CHƯƠNG TRÌNH NẠP WINPIC800:

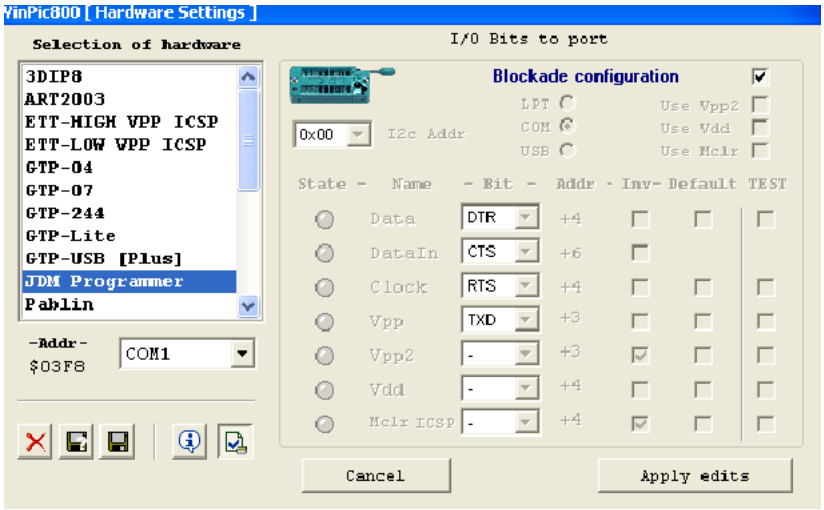
Hướng dẫn cài đặt Winpic800: chạy file WinPic800_V3_59.exe để cài đặt Winpic800, sau đó chọn next để tiến hành cài đặt.

Khi cài đặt xong thì trên màn hình desktop xuất hiện biểu tượng Winpic800, click vào biểu tượng Winpic800 để chạy chương trình nạp, cửa sổ của Winpic800 như hình sau:



Hình 3-8. Cửa sổ của WINPIC800.

Sau đó vào Settings → chọn Hardware để tiến hành cài đặt phần cứng cho chương trình nạp, màn hình hardware settings xuất hiện như sau:



Hình 3-9. Cửa sổ Hardware Setting.

Chọn hardware là JMD Programmer, chọn Apply Edits để chấp nhận.

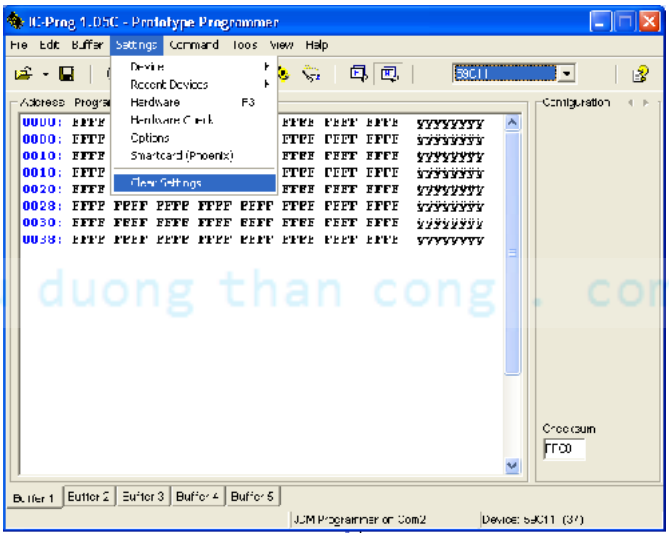
Sau đó chọn họ PIC và tên PIC muốn nạp chương trình. Ví dụ như muốn nạp cho PIC16F877A thì chọn họ 16F tên PIC là 16F877A.

Hướng dẫn nạp chương trình cho PIC16F877A bằng Winpic800:

Chọn File → Open hoặc chọn  để chọn file *.Hex cần nạp. Sau đó chọn Device → Program All (Ctrl+P) hoặc chọn  để nạp chương trình.

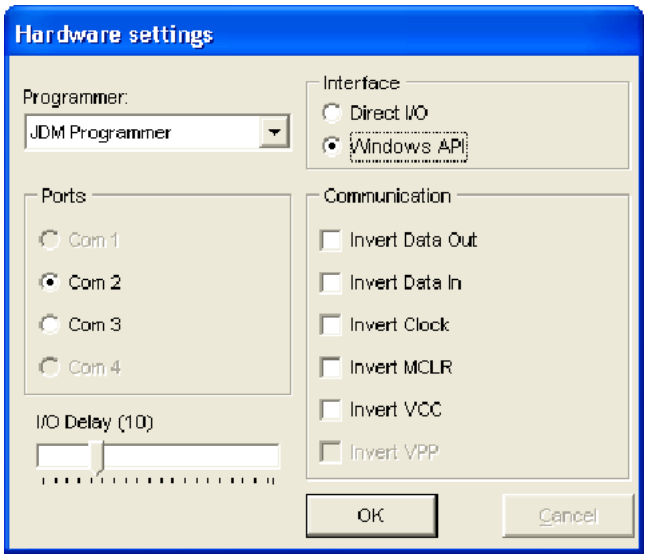
2. CHƯƠNG TRÌNH NẠP IC-PRO:

Hướng dẫn cài đặt IC-Pro: giải nén file IC-pro vào thư mục bất kì như IC-Pro sau đó chạy file ICProg.exe bỏ qua tất cả các lỗi để mở chương trình ra. Sau đó chọn Settings >> Clear Settings như hình sau:



Hình 3-10. Màn hình của IC-Pro.

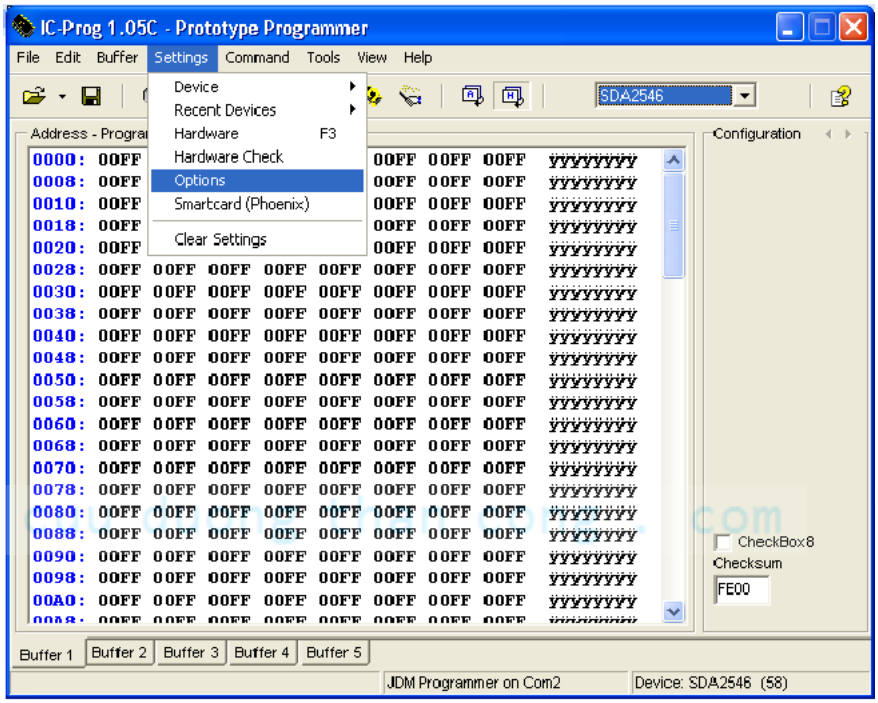
Sau khi nhấn Yes liên tục, một màn hình Hardware settings sẽ hiện ra như sau:



Hình 3-11. Cửa sổ Hardware Setting.

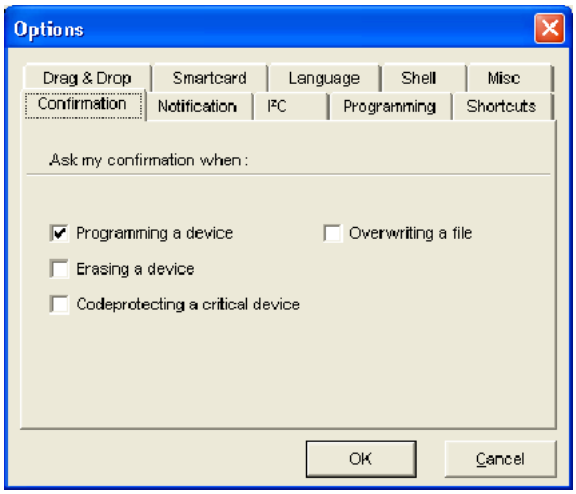
Do chúng ta chọn dùng bộ nạp PG1A là một bộ nạp được phát triển của JDM, cho nên phần Programmer chúng ta sẽ chọn JDM Programmer. Phần cổng, chúng ta sẽ chọn COM1, COM2 hoặc COM3 tùy theo máy tính. Phần Interface, các bạn chọn Windows API và phần Communication không đánh dấu gì cả, sau đó chọn OK. Khi sử dụng Windows API, không cần quan tâm đến phần I/O Delay.

Màn hình ban đầu sau khi khởi động lại IC-Prog hiện ra như hình dưới. Chúng ta sẽ chọn Settings →Options để tiếp tục cài đặt cho IC-Prog.



Hình 3-12. Cửa sổ Setting.

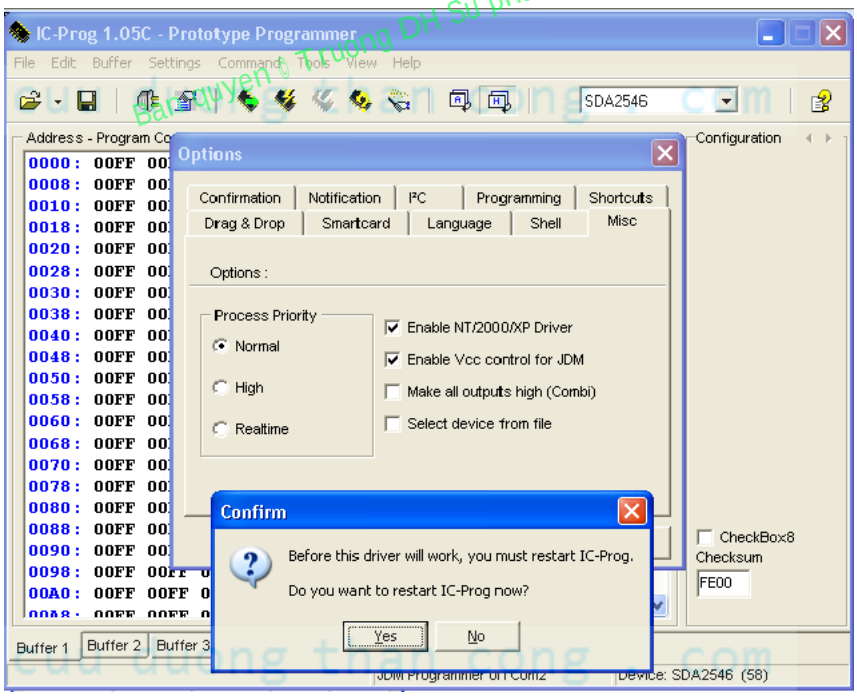
Màn hình Options sẽ hiện ra. Chỉ quan tâm tới phần Misc, còn các phần khác không cần quan tâm. Cứ để mặc định như chương trình ban đầu đã có.



Hình 3-13. Cửa sổ lựa chọn.

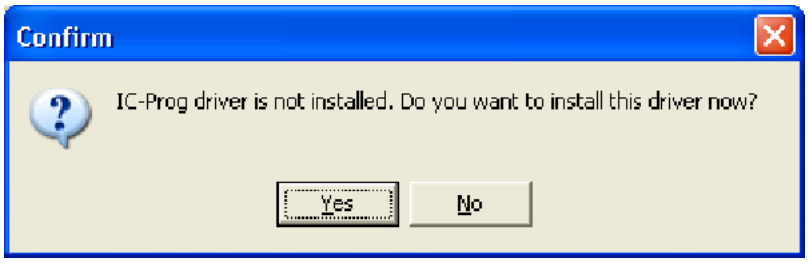
Chọn Enable Vcc control for JDM, sau đó mới chọn tiếp Enable NT/2000/XP Driver. Khi chọn Enable Driver xong, ngay lập tức sẽ có một màn hình Confirm hiện lên như trong hình bên dưới nhấn Yes để cài đặt.

Lưu ý rằng, driver đã nằm sẵn trong thư mục ICProg. Do vậy, ICProg sẽ tự động nhận ra và khởi động lại ICProg.



Hình 3-14. Cửa sổ lựa chọn.

Một màn hình Confirm khác sẽ hiện ra để yêu cầu xác nhận việc cài đặt driver cho Windows NT/2000/XP,chọn Yes.

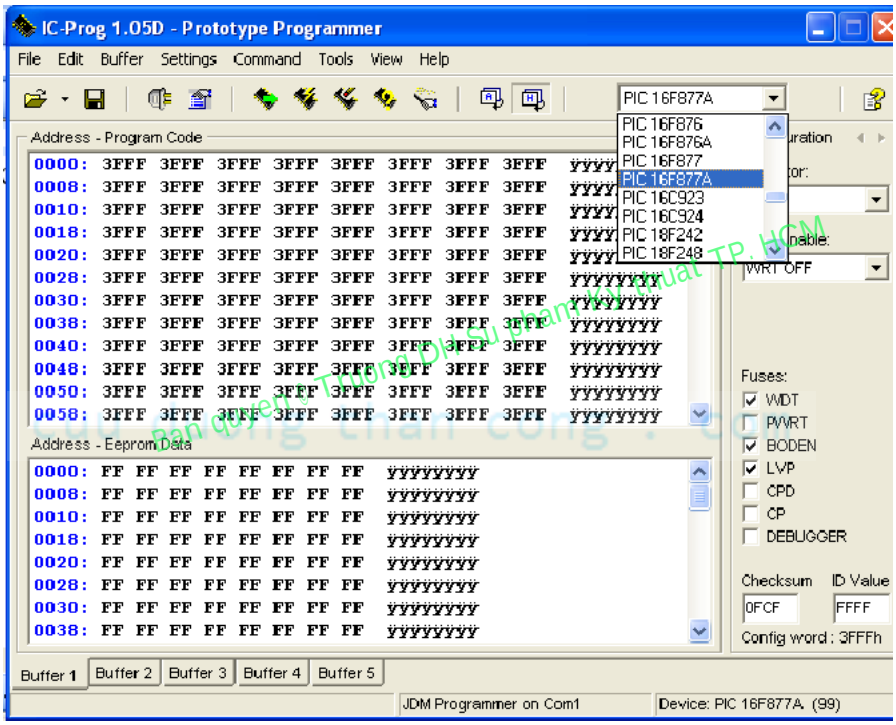


Hình 3-15. Cài đặt Driver.

Như vậy công việc cài đặt đã hoàn tất.

Hướng dẫn nạp cho PIC16F877A bằng mạch nạp PG1A:

Khởi động chương trình nạp IC-Pro sau đó chọn tên PIC cần nạp như hình sau:



Hình 3-16. Chọn PIC cần nạp.

Ví dụ như chọn PIC16F877A, sau đó vào File -> chọn Open file hoặc chọn để chọn file .HEX cần nạp. Sau đó chọn Command ->Program All (F5) hoặc chọn để nạp chương trình cho PIC16F877A.

III. NGÔN NGỮ LẬP TRÌNH ASM CỦA MPLAB:

1. CÁC QUY ƯỚC CỦA NGÔN NGỮ MPLAB:

[nhãn] LỆNH tham số 1, tham số 2

Một dòng như trên gọi là một dòng lệnh. Chương trình MPLAB được chia làm 4 cột rõ ràng:

- Cột thứ nhất để viết nhãn.
- Cột thứ hai để viết tên lệnh muốn thực hiện.
- Cột thứ 3 là tham số thứ nhất của lệnh.
- Cột thứ tư là tham số thứ hai của lệnh.

Giữa tham số thứ nhất và tham số thứ 2 luôn cách nhau một dấu phẩy (,). Các cột được cách nhau bằng ít nhất một ký tự TAB (khoảng trắng rộng) hay một kí tự trắng.

a. [nhãn]:

[nhãn] là một chuỗi ký tự để đánh dấu một điểm nào đó trong chương trình, thay vì phải ghi địa chỉ bộ nhớ thì chúng ta thay địa chỉ đó bằng một cái **[nhãn]**. **[nhãn]** này thường được gọi lại bằng lệnh **GOTO** hoặc **CALL**.

Mỗi câu lệnh, có thể có hoặc không có **[nhãn]**. Tuy nhiên, nên viết sao cho số **[nhãn]** là ít nhất để tránh sự lầm lẫn và rối mắt khi lập trình.

[nhãn] được viết trong cột thứ nhất của dòng lệnh. **[nhãn]** không được bắt đầu bằng các ký tự đặc biệt như: *,&, khoảng trắng, các con số (0,1,2...)... Giữa các ký tự của nhãn cũng không được có các ký tự đặc biệt *,^,...

Độ dài của một **[nhãn]** không giới hạn, tuy nhiên, chúng ta phải viết sao cho [nhãn] luôn nằm trong cột thứ nhất của dòng lệnh, độ dài nhãn vừa phải để dễ quan sát, đủ thông tin gợi nhớ và thuận tiện khi lập trình.

Chúng ta hoàn toàn có thể ký hiệu các **[nhãn]** là **NHAN_1**, **NHAN_2**... nhưng nội dung thông tin của nhãn không đủ để thể hiện công việc sẽ được thực hiện, như vậy sẽ rất khó nhớ khi lập trình, nhất là khi chương trình viết dài và có đến hàng chục hàng trăm nhãn trong chương trình.

Ví dụ:

Nhãn đúng:

Good_bye
Exit
KHOIDONG
Lap_1

Nhãn sai:

1Exit
Good^bye
Khoi dong

b. Lệnh và các tham số:

LỆNH là tên của các lệnh gợi nhớ được liệt kê theo bảng bên dưới. LỆNH được viết vào cột thứ hai, mỗi dòng lệnh phải có tên LỆNH, nếu không có thì sẽ không biết dòng lệnh đó làm việc gì. LỆNH thể hiện công việc phải làm của dòng lệnh.

Tùy theo LỆNH mà có thể có tham số 1 và tham số 2, hoặc chỉ có tham số 1, hoặc không có tham số nào hết. Trong một dòng lệnh, phải viết đủ tham số của LỆNH đó.

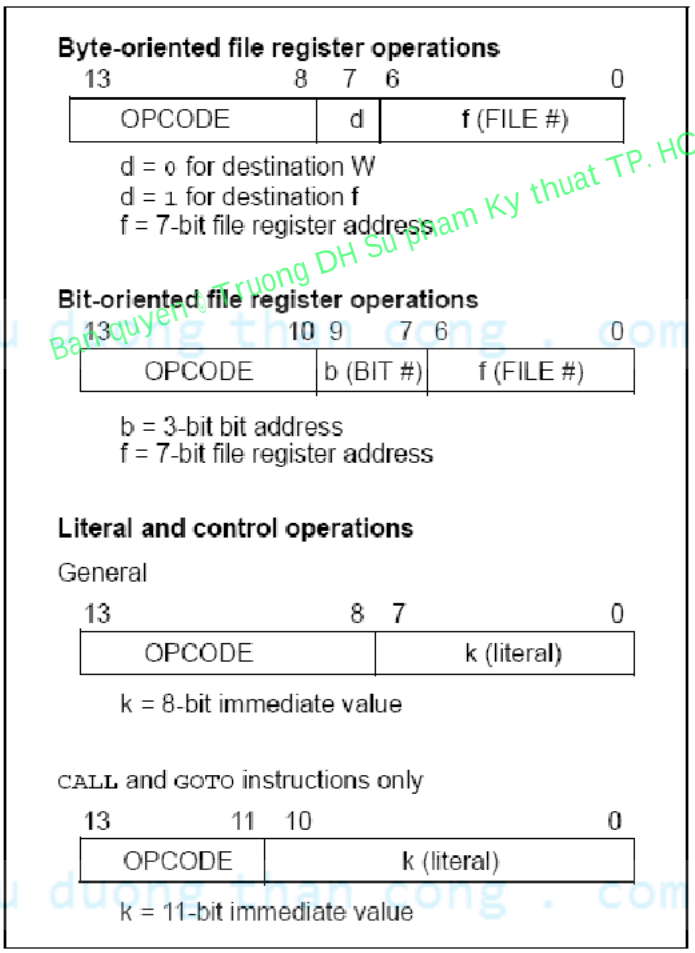
c. Quy ước kí hiệu trong MPLAB:

Bảng 3-1: Kí hiệu quy ước cấu trúc lệnh

Kí hiệu	Chức năng
f	Địa chỉ của file thanh ghi từ 0x00 đến 0x7F
w	Thanh ghi W - Working register (accumulator)

b	Là địa chỉ nằm trong file thanh ghi 8 bit
k	Hằng số hoặc nhãn
x	Không quan tâm là 0 hay 1
d	Lực chọn nơi nhận dữ liệu d =0 lưu kết quả vào thanh ghi W d =1 lưu kết quả vào trong thanh ghi f Mặc định d = 1
PC	Bộ đếm chương trình
TO	Bit Time-out
PD	Bit Power-down

Bảng 3-1. Kí hiệu các thanh ghi trong MPLAB.



Hình 3-17. Khuôn khổ chung cho một số lệnh của PIC 16F877A.

Bảng 3-2: Tập lệnh của PIC16F877A:

Mnemonic, Operands		Description	Cycles	14-Bit Opcode MSb LSb	Status Affected	Notes
BYTE-ORIENTED FILE REGISTER OPERATIONS						
ADDWF	f, d	Add W and f	1	00 0111 dfff ffff	C,DC,Z	1,2
ANDWF	f, d	AND W with f	1	00 0101 dfff ffff	Z	1,2
CLRF	f	Clear f	1	00 0001 lfff ffff	Z	2
CLRW	-	Clear W	1	00 0001 0xxx xxxx	Z	
COMF	f, d	Complement f	1	00 1001 dfff ffff	Z	1,2
DECF	f, d	Decrement f	1	00 0011 dfff ffff	Z	1,2
DECFSZ	f, d	Decrement f, Skip if 0	1(2)	00 1011 dfff ffff		1,2,3
INCF	f, d	Increment f	1	00 1010 dfff ffff	Z	1,2
INCFSZ	f, d	Increment f, Skip if 0	1(2)	00 1111 dfff ffff		1,2,3
IORWF	f, d	Inclusive OR W with f	1	00 0100 dfff ffff	Z	1,2
MOVF	f, d	Move f	1	00 1000 dfff ffff	Z	1,2
MOVWF	f	Move W to f	1	00 0000 lfff ffff		
NOP	-	No Operation	1	00 0000 0xx0 0000		
RLF	f, d	Rotate Left f through Carry	1	00 1101 dfff ffff	C	1,2
RRF	f, d	Rotate Right f through Carry	1	00 1100 dfff ffff	C	1,2
SUBWF	f, d	Subtract W from f	1	00 0010 dfff ffff	C,DC,Z	1,2
SWAPF	f, d	Swap nibbles in f	1	00 1110 dfff ffff		1,2
XORWF	f, d	Exclusive OR W with f	1	00 0110 dfff ffff	Z	1,2
BIT-ORIENTED FILE REGISTER OPERATIONS						
BCF	f, b	Bit Clear f	1	01 00bb bfff ffff		1,2
BSF	f, b	Bit Set f	1	01 01bb bfff ffff		1,2
BTFSC	f, b	Bit Test f, Skip if Clear	1 (2)	01 10bb bfff ffff		3
BTFSS	f, b	Bit Test f, Skip if Set	1 (2)	01 11bb bfff ffff		3
LITERAL AND CONTROL OPERATIONS						
ADDLW	k	Add Literal and W	1	11 111x kkkk kkkk	C,DC,Z	
ANDLW	k	AND Literal with W	1	11 1001 kkkk kkkk	Z	
CALL	k	Call Subroutine	2	10 0kkk kkkk kkkk		
CLRWDT	-	Clear Watchdog Timer	1	00 0000 0110 0100	$\overline{TO}$,$\overline{PD}$	
GOTO	k	Go to Address	2	10 1kkk kkkk kkkk		
IORLW	k	Inclusive OR Literal with W	1	11 1000 kkkk kkkk	Z	
MOVLW	k	Move Literal to W	1	11 00xx kkkk kkkk		
RETFIE	-	Return from Interrupt	2	00 0000 0000 1001		
RETLW	k	Return with Literal in W	2	11 01xx kkkk kkkk		
RETURN	-	Return from Subroutine	2	00 0000 0000 1000		
SLEEP	-	Go into Standby mode	1	00 0000 0110 0011	$\overline{TO}$,$\overline{PD}$	
SUBLW	k	Subtract W from Literal	1	11 110x kkkk kkkk	C,DC,Z	
XORLW	k	Exclusive OR Literal with W	1	11 1010 kkkk kkkk	Z	

Bảng 3-2. Tóm tắt tập lệnh.

***Chú ý* (1):** khi thanh ghi IO bị thay đổi (ví dụ như lệnh MOVF PORT, 1) thì giá trị dùng trong lệnh là giá trị xuất hiện ở ngõ ra. Ví dụ thanh ghi chốt dữ liệu là '1' để định cấu hình là ngõ ra và được điều khiển xuống mức thấp bởi thiết bị bên ngoài thì dữ liệu đọc vào là mức.

***Chú ý* (2):** nếu lệnh này được thực hiện cho thanh ghi TMR0 thì bộ chia trước sẽ bị xoá nếu gán cho khối Timer0.

***Chú ý* (3):** nếu thanh ghi PC bị thay đổi thì cần 2 chu kỳ, chu kỳ thứ 2 thực hiện lệnh NOP.

2. DIỄN TẢ CÁC LỆNH:

a. Lệnh: **ADDLW** Cộng hằng số k vào W

- Cú pháp: ADDLW k
- Tác tố: $0 \leq k \leq 255$

- Thực thi: (W) + k → (W)
- Cờ ảnh hưởng: C,DC,Z
- Chức năng: cộng nội dung thanh ghi W với hằng số k 8 bit và kết quả lưu vào W.
- Chu kỳ thực hiện: 1.

b. Lệnh: **ADDWF** Cộng W với f

- Cú pháp: ADDWF f,d
- Tác tố: $0 \le f \le 127, d \in [0,1]$
- Thực thi: (W) + (f) → (dest)
- Cờ ảnh hưởng: C,DC,Z
- Chức năng: cộng nội dung thanh ghi W với thanh ghi f. Nếu d= 0 thì lưu kết quả vào thanh ghi W, còn d=1 thì lưu vào thanh ghi f.
- Chu kỳ thực hiện: 1.

c. Lệnh: **ANDLW** Anal hằng số với W

- Cú pháp: ADDLW k
- Tác tố: $0 \le k \le 255$
- Thực thi: (W) AND (k)→(W)
- Cờ ảnh hưởng: Z
- Chức năng: nội dung thanh ghi W được AND với hằng số k 8 bit, kết quả lưu vào thanh ghi W
- Chu kỳ thực hiện: 1.

d. Lệnh: **ANDWF** Anal W với F

- Cú pháp: ANDWF f,d
- Tác tố: $0 \le f \le 127, d \in [0,1]$
- Thực thi: (W) AND (f) → (dest)
- Cờ ảnh hưởng: Z
- Chức năng: AND thanh ghi W với thanh ghi f. Nếu d = 0 thì kết quả lưu vào thanh ghi W, nếu d=1 thì kết quả lưu vào thanh ghi f.
- Chu kỳ thực hiện: 1.

e. Lệnh: **BCF** xoá bit trong thanh ghi F

- Cú pháp: BCF f,d
- Tác tố: $0 \le f \le 127, 0 \le b < 7$
- Thực thi: 0 → (f<b>)
- Cờ ảnh hưởng: không
- Chức năng: bit b trong thanh ghi f bị xóa
- Chu kỳ thực hiện: 1.

f. Lệnh: **BSF** set bit trong thanh ghi F

- Cú pháp: BSF f,d
- Tác tố: $0 \le f \le 127, 0 \le b < 7$
- Thực thi: 1 → (f<b>)

- Cờ ảnh hưởng: không
- Chức năng: bit b trong thanh ghi f được set lên 1.
- Chu kỳ thực hiện: 1.

g. Lệnh: **BTFSS** kiểm tra 1 bit trong thanh ghi F và nhảy nếu bằng 1

- Cú pháp: BTFSS f,d
- Tác tố: $0 \le f \le 127, 0 \le b < 7$
- Thực thi: nhảy nếu f<b>=1
- Cờ ảnh hưởng: không
- Chức năng: nếu bit b trong thanh ghi f bằng 1 thì lệnh kế bị bỏ qua và thay bằng lệnh NOP.
- Chu kỳ thực hiện: 1(2).

h. Lệnh: **BTFSC** kiểm tra 1 bit trong thanh ghi F và nhảy nếu bằng 0

- Cú pháp: BTFSC f,d
- Tác tố: $0 \le f \le 127, 0 \le b < 7$
- Thực thi: nhảy nếu f<b>=0
- Cờ ảnh hưởng: không
- Chức năng: nếu bit b trong thanh ghi f=0 thì lệnh kế bị bỏ qua và thay bằng lệnh NOP.
- Chu kỳ thực hiện: 1(2).

i. Lệnh: **CALL** gọi chương trình con

- Cú pháp: CALL k
- Tác tố: $0 \le k \le 2047$
- Thực thi: (PC) + 1→ TOS; k→PC<10:0>; (PCLATH<4:3>)→(PC<12:11>)
- Cờ ảnh hưởng: không
- Chức năng: gọi chương trình con. 13 bit địa chỉ trở về (PC+1) được cất vào ngăn xếp. Tiếp Theo 11bit địa chỉ <10:0> được tải vào PC. Hai bit cao của PC được nạp từ PCLATH<4:3>.
- Chu kỳ thực hiện: 2.

j. Lệnh: **CLRF** xoá thanh ghi f

- Cú pháp: CLRF f
- Tác tố: $0 \le f \le 127$
- Thực thi: 00h →(f); 1 → Z
- Trạng thái ảnh hưởng: Z
- Chức năng: Xoá thanh ghi f và bit Z được set.
- Chu kỳ thực hiện: 1.

k. Lệnh: **CLRW** xoá thanh ghi W

- Cú pháp: CLRW
- Tác tố: không
- Thực thi: 00h →(W), 1 → Z

- Cờ ảnh hưởng: Z
- Chức năng: xoá thanh ghi W và bit Z lên 1.
- Chu kỳ thực hiện: 1.

l. Lệnh: **CLRWDT** xoá WDT

- Cú pháp: CLRWDT
- Tác tố: không
- Thực thi: 00 → WDT; 0 → Bộ đếm chia trước của WDT; 1 → $\overline{TO}$; 1 → $\overline{PD}$
- Cờ ảnh hưởng: $\overline{TO}$, $\overline{PD}$
- Chức năng: lệnh CLRWDT sẽ xoá bộ định thời WDT và xoá luôn bộ đếm chia trước của WDT. Các bit $\overline{PD}$, $\overline{TO}$ được set lên 1.
- Chu kỳ thực hiện: 1.

m. Lệnh: **COMF** bù thanh ghi f

- Cú pháp: COMF f,d
- Tác tố: $0 \le f \le 127, d \in [0,1]$
- Thực thi: $(\overline{f})$ → (dest)
- Cờ ảnh hưởng: Z
- Chức năng: bù 1 nội dung thanh ghi f. Nếu d=0 thì kết quả lưu vào thanh ghi W. Nếu d=1 thì kết quả lưu vào thanh ghi f.
- Chu kỳ thực hiện: 1.

n. Lệnh: **DECF** giảm nội dung thanh ghi f

- Cú pháp: DECF f,d
- Tác tố: $0 \le f \le 127, d \in [0,1]$
- Thực thi: (f) – 1 → (dest)
- Cờ ảnh hưởng: Z
- Chức năng: giảm nội dung thanh ghi f đi 1. Nếu d= 0 thì kết quả lưu vào thanh ghi W. Nếu d= 1 thì kết quả lưu vào thanh ghi f.
- Chu kỳ thực hiện: 1.

o. Lệnh: **DECFSZ** giảm nội dung thanh ghi f và nhảy nếu bằng 0

- Cú pháp: DECFSZ f,d
- Tác tố: $0 \le f \le 127, d \in [0,1]$
- Thực thi: (f) – 1 → (dest); Nhảy nếu kết quả = 0
- Cờ ảnh hưởng: không
- Chức năng: nội dung thanh ghi f giảm đi 1. Nếu d = 0 thì kết quả lưu vào thanh ghi f. Nếu d = 1 thì kết quả lưu vào thanh ghi W. Nếu kết quả bằng 0 thì bỏ qua lệnh kế và thay bằng lệnh NOP (do mã đón về trong lúc lệnh đang thực hiện).
- Chu kỳ thực hiện: 1(2).

p. Lệnh: **GOTO** lệnh rẽ nhánh không điều kiện

- Cú pháp: GOTO k
- Tác tố: $0 \le k \le 2047$

- Thực thi: k → PC<10:0>; PCLATH<4:3> → PC<12:11>
- Cờ ảnh hưởng: không
- Chức năng: GOTO là lệnh nhảy không điều kiện. Giá trị của 11bit <10:0> được tải vào PC. Các bit cao của PC được tải từ PCLATH<4:3>.
- Chu kỳ thực hiện: 2.

q. Lệnh: **INCF** lệnh tăng nội dung thanh ghi f

- Cú pháp: INCF f,d
- Tác tố: $0 \le f \le 127$, $d \in [0,1]$
- Thực thi: (f) + 1 → (dest)
- Cờ ảnh hưởng: Z
- Chức năng: nội dung của thanh ghi f tăng lên 1. Nếu d = 0 thì kết quả lưu vào thanh ghi W. Nếu d = 1 thì kết quả lưu trở lại vào thanh ghi f.
- Chu kỳ thực hiện: 1.

r. Lệnh: **INCFSZ** lệnh tăng nội dung thanh ghi f và nhảy nếu bằng 0

- Cú pháp: INCFSZ f,d
- Tác tố: $0 \le f \le 127$, $d \in [0,1]$
- Thực thi: (f) + 1→ (dest)
- Cờ ảnh hưởng: không.
- Chức năng: nội dung của thanh ghi f tăng. Nếu d= 0 thì kết quả lưu vào thanh ghi W. Nếu d= 1thì kết quả lưu vào thanh ghi f. Nếu kết quả là bằng 0 thì bỏ qua lệnh kế và được thay bằng lệnh NOP.
- Chu kỳ thực hiện: 1(2).

Ví dụ:

```
HERE    INCFSZ  CNT,1
        GOTO    LOOP
CONTI   ...
        ...
```

Trường hợp 1: Trước khi thực hiện lệnh thì **PC=địa chỉ HERE**, **CNT = 0×FF.**

Sau khi thực hiện lệnh thì **PC=địa chỉ CONTI**, **CNT = 0×00**. Bỏ qua lệnh **GOTO**

Trường hợp 2: Trước khi thực hiện lệnh thì **PC=địa chỉ HERE**, **CNT = 0×00.**

Sau khi thực hiện lệnh thì **PC=địa chỉ HERE+1**, **CNT = 0×01**. Lệnh **GOTO** được thực hiện.

s. Lệnh: **IORLW** lệnh OR hằng số với W

- Cú pháp: IORLW k
- Tác tố: $0 \le k \le 255$
- Thực thi: (W) OR k → W
- Cờ ảnh hưởng: Z
- Chức năng: OR hằng số k 8 bit với W. Nếu d= 0 thì kết quả được lưu vào thanh ghi W. Nếu d= 1 thì kết quả lưu vào thanh ghi f.
- Chu kỳ thực hiện: 1.

t. Lệnh: **IORWF** lệnh OR W với f

- Cú pháp: IORWF f,d
- Tác tố: $0 \le f \le 127$
- Thực thi: (W) OR k → (dest)
- Cờ ảnh hưởng: Z
- Chức năng: nội dung thanh ghi W được OR với nội dung thanh ghi W. Nếu d= 0 thì kết quả lưu vào thanh ghi W. Nếu d= 1 thì kết quả lưu vào thanh ghi f.
- Chu kỳ thực hiện: 1.

Ví dụ: **IORWF RESULT,0**

Trước khi thực hiện lệnh thì W=0×91 và RESULT =0×13.

Sau khi thực hiện lệnh thì W=0×93 và RESULT =0×13 và Z=0.

u. Lệnh: **MOVLW** lệnh copy dữ liệu

- Cú pháp: MOVLW k
- Tác tố: $0 \le k \le 255$
- Thực thi: k → W
- Cờ ảnh hưởng: không.
- Chức năng: dữ liệu 8 bit k nạp vào thanh ghi W.
- Chu kỳ thực hiện: 1.

Ví dụ1: **MOVLW 0×5A**

Sau khi thực hiện lệnh thì W=0×5A

Ví dụ2: **MOVLW MYREG**

Trước khi thực hiện lệnh thì W=0×01.

Kí hiệu MYREG là dữ liệu của ô nhớ là 0×37.

Sau khi thực hiện lệnh thì W=0×37.

Ví dụ3: **MOVLW HIGH(LU_TABLE)**

Trước khi thực hiện lệnh thì W=0×01.

LU_TABLE là nhãn của ô nhớ có địa chỉ là 0×9375.

Sau khi thực hiện lệnh thì W=0×93.

v. Lệnh: **MOVF** lệnh copy dữ liệu

- Cú pháp: MOVF f,d
- Tác tố: $0 \le f \le 127$, $d \in [0,1]$
- Thực thi: (f) → W
- Cờ ảnh hưởng: Z.
- Chức năng: nội dung thanh ghi 'f' được copy sang nơi đến tuỳ thuộc vào giá trị của 'd'. Nếu 'd' = 0 thì nơi đến là thanh ghi W. Nếu 'd'=1 thì nơi đến chính là thanh ghi 'f'. Trường hợp 'd'=1 rất tiện lợi để kiểm tra thanh ghi file vì trạng thái của cờ Z bị ảnh hưởng.
- Chu kỳ thực hiện: 1.

Ví dụ1: **MOVF FSR,0**

Trước khi thực hiện lệnh thì: W=0×01, FSR=0×C2.

Sau khi thực hiện lệnh thì W=0×C2 và cờ Z=0.

Ví dụ2: **MOVLW FSR,1**

Trường hợp 1: Trước khi thực hiện lệnh thì FSR=0×C2.

Sau khi thực hiện lệnh thì FSR=0×C2 và Z =0.

Trường hợp 2: Trước khi thực hiện lệnh thì FSR=0×00.

Sau khi thực hiện lệnh thì FSR=0×00 và Z =1.

w. Lệnh: **MOVWF** lệnh copy dữ liệu

- Cú pháp: MOVWF f
- Tác tố: $0 \le f \le 127$, $d \in [0,1]$
- Thực thi: (W) → f
- Trạng thái ảnh hưởng: không.
- Chức năng: nội dung thanh ghi W được copy sang thanh ghi 'f'.
- Chu kỳ thực hiện: 1.

Ví dụ1: **MOVWF OPTION_REG**

Trước khi thực hiện lệnh thì: OPTION_REG=0×FF, W=0×4F.

Sau khi thực hiện lệnh thì OPTION_REG=0×4F, W=0×4F.

Ví dụ2: **MOVLW INDF**

Trước khi thực hiện lệnh thì W=0×17, FSR=0×C2 và nội dung của địa chỉ (FSR)=0×00.

Sau khi thực hiện lệnh thì W=0×17, FSR=0×C2 và nội dung của địa chỉ (FSR)=0×17.

x. Lệnh: **RETFIE** lệnh trở về từ chương trình con phục vụ ngắt.

- Cú pháp: RETFIE
- Tác tố: không có.
- Thực thi: TOS → PC, 1 → GIE.
- Cờ ảnh hưởng: không.
- Chức năng: trở về từ chương trình phục vụ ngắt. 13 bit địa chỉ ở đỉnh ngăn xếp (TOS) được nạp cho thanh ghi PC. Bit cho phép ngắt toàn cục tự động được set lên mức 1 để cho phép ngắt.
- Chu kỳ thực hiện: 2.

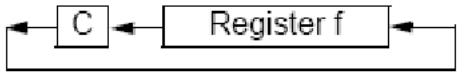
y. Lệnh: **RETLW** lệnh trở về từ chương trình con phục vụ ngắt.

- Cú pháp: RETLW k
- Tác tố: $0 \le k \le 255$
- Thực thi: k → W, TOS → PC.
- Trạng thái ảnh hưởng: không.

- Chức năng: thanh ghi W được nạp giá trị 8 bit 'k'. 13 bit địa chỉ ở đỉnh ngăn xếp (địa chỉ trở về) được nạp cho thanh ghi PC.
- Chu kỳ thực hiện: 2.

z. Lệnh: **RLF** lệnh xoay trái qua cờ C

- Cú pháp: RLF f,d
- Tác tố: $0 \le f \le 127$, $d \in [0,1]$
- Thực thi:



- Trạng thái ảnh hưởng: C
- Chức năng: nội dung của thanh ghi f được xoay sang trái một bit qua cờ C. Nếu d= 0 thì kết quả được lưu vào thanh ghi W. Nếu d= 1 thì kết quả lưu vào thanh ghi f.
- Chu kỳ thực hiện: 1.

Ví dụ1: **RLF REG1,0**

Trước khi thực hiện lệnh thì: **REG1**=1110 0110 và C =0.

Sau khi thực hiện lệnh thì **REG1**=1110 0110, **W**=1100 1100 và C =1.

Ví dụ2: **RLF INDF,1**

Trường hợp 1: Trước khi thực hiện lệnh thì: FSR=0×C2, nội dung của địa chỉ (FSR) = 0011 1010 và C = 1.

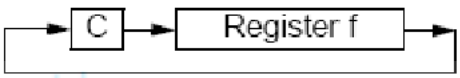
Sau khi thực hiện lệnh thì: FSR=0×C2, nội dung của địa chỉ (FSR) = 0111 0101 và C = 0.

Trường hợp 2: Trước khi thực hiện lệnh thì: FSR=0×C2, nội dung của địa chỉ (FSR) = 1011 1001 và C = 0

Sau khi thực hiện lệnh thì: FSR=0×C2, nội dung của địa chỉ (FSR) = 0111 0010 và C = 1.

aa. Lệnh: **RRL** lệnh xoay phải qua cờ C

- Cú pháp: RRF f,d
- Tác tố: $0 \le f \le 127$, $d \in [0,1]$
- Thực thi:



- Trạng thái ảnh hưởng: C
- Chức năng: nội dung của thanh ghi f được xoay sang phải một bit qua cờ C. Nếu d= 0 thì kết quả được lưu vào thanh ghi W. Nếu d= 1 thì kết quả lưu vào thanh ghi f.
- Chu kỳ thực hiện: 1.

Ví dụ1: **RRF REG1,0**

Trước khi thực hiện lệnh thì: **REG1**=1110 0110, **W**=×××× ×××× và C =0.

Sau khi thực hiện lệnh thì **REG1**=1110 0110, **W**=0111 0011 và C =0.

Ví dụ2: **RRF INDF,1**

Trường hợp 1: Trước khi thực hiện lệnh thì: FSR=0×C2, nội dung của địa chỉ (FSR) = 0011 1010 và C = 1.

Sau khi thực hiện lệnh thì: FSR=0×C2, nội dung của địa chỉ (FSR) = 1001 1101 và C = 0.

Trường hợp 2: Trước khi thực hiện lệnh thì: FSR=0×C2, nội dung của địa chỉ (FSR) = 0011 1001 và C = 0

Sau khi thực hiện lệnh thì: FSR=0×C2, nội dung của địa chỉ (FSR) = 0011 1100 và C = 1.

bb. Lệnh: **RETURN** lệnh kết thúc chương trình con

- Cú pháp: RETURN
- Tác tố: không
- Thực thi: TOS → PC
- Trạng thái ảnh hưởng: không
- Chức năng: lệnh trở về từ chương trình con. Nội dung đỉnh ngăn xếp trả cho PC. Lệnh này thực hiện trong 2 chu kì lệnh.
- Chu kỳ thực hiện: 2.

cc. Lệnh: **SLEEP** lệnh ngủ

- Cú pháp: SLEEP
- Tác tố: không
- Thực thi: 00h → WDT; 0 → bộ đếm chia trước của WDT; 1 → $\overline{TO}$; 0 → $\overline{PD}$
- Cờ ảnh hưởng: $\overline{TO}$, $\overline{PD}$
- Chức năng: bit trạng thái giảm nguồn $\overline{PD}$ (Power Down Status bit) bị xóa. Bit trạng thái tạm nghỉ $\overline{TO}$ (Time-Out) được set. Bộ định thời WDT và bộ chia trước bị xóa. Vi xử lí bước vào chế độ ngũ (SLEEP) và bộ dao động ngừng hoạt động.
- Chu kỳ thực hiện: 1.

dd. Lệnh: **SUBLW** lệnh trừ hằng số cho thanh ghi W

- Cú pháp: SUBLW k
- Tác tố: $0 \le k \le 255$
- Thực thi: k – (W)→ (W)
- Cờ ảnh hưởng: C, DC, Z
- Chức năng: hằng số k 8 bit trừ cho nội dung thanh ghi W và kết quả được lưu vào thanh ghi W.
- Chu kỳ thực hiện: 1.

Ví dụ1: **SUBLW 0x02**

Trường hợp 1: Trước khi thực hiện lệnh thì: W=0×01, cờ C = × và Z = ×.

Sau khi thực hiện lệnh thì: W=0×01, cờ C = 1 (kết quả dương) và Z = 0.

Trường hợp 2: Trước khi thực hiện lệnh thì: W=0×02, cờ C = × và Z = ×.

Sau khi thực hiện lệnh thì: W=0×00, cờ C = 1 (kết quả bằng 0) và Z = 1.

Trường hợp 3: Trước khi thực hiện lệnh thì: W=0×03, cờ C = × và Z = ×.

Sau khi thực hiện lệnh thì: W=0×FF, cờ C = 1 (kết quả âm) và Z = 0.

Ví dụ2: **SUBLW MYREG**

Trước khi thực hiện lệnh thì: W=0×10, kí hiệu MYREG là nội dung ô nhớ có giá trị 0×37.

Sau khi thực hiện lệnh thì: W=0×27, cờ C = 1 (kết quả dương) và Z = 0.

ee. Lệnh: **SUBWF** lệnh trừ thanh ghi f cho thanh ghi W

- Cú pháp: SUBLW f,d
- Tác tố: $0 \le f \le 127$, $d \in [0,1]$
- Thực thi: (f) – (W)→ (dest)
- Cờ ảnh hưởng: C, DC, Z
- Chức năng: nội dung thanh ghi f trừ cho nội dung thanh ghi W. Nếu d= 0 thì kết quả lưu vào thanh ghi W. Nếu d= 1 thì kết quả lưu vào thanh ghi f.
- Chu kỳ thực hiện: 1.

Ví dụ1: **SUBWF REG1,1**

Trường hợp 1: Trước khi thực hiện lệnh thì: REG1=0×03, W = 0×02, cờ C = × và Z = ×.

Sau khi thực hiện lệnh thì: REG1=0×01, W=0×02, cờ C = 1 (kết quả dương) và Z = 0.

Trường hợp 2: Trước khi thực hiện lệnh thì: REG1=0×02, W = 0×02, cờ C = × và Z = ×.

Sau khi thực hiện lệnh thì: REG1=0×00, W=0×02, cờ C = 1 (kết quả zero) và Z = 1.

Trường hợp 3: Trước khi thực hiện lệnh thì: REG1=0×01, W = 0×02, cờ C = × và Z = ×.

Sau khi thực hiện lệnh thì: REG1=0×FF, W=0×02, cờ C = 1 (kết quả âm) và Z = 0.

ff. Lệnh: **SWAPF** lệnh hoán chuyển 4 bit của thanh ghi f

- Cú pháp: SWAPF f,d
- Tác tố: $0 \le f \le 127$, $d \in [0,1]$
- Thực thi: (f<3:0>) → (dest<7:4>); (f<7:4>) → (dest<3:0>)
- Cờ ảnh hưởng: không
- Chức năng: 4 bit cao và 4 bit thấp của thanh ghi f được đổi với nhau. Nếu d= 0 thì kết quả lưu vào thanh ghi W. Nếu d= 1 thì kết quả lưu vào thanh ghi f.
- Chu kỳ thực hiện: 1.

Ví dụ1: **SWAPF REG1,0**

Trước khi thực hiện lệnh thì: REG1=0×A5.

Sau khi thực hiện lệnh thì: REG1=0×A5, W=0×5A.

Ví dụ2: **MOVLW FSR,1**

Trước khi thực hiện lệnh thì FSR=0×C2, nội dung của địa chỉ (FSR) = 0×20.

Sau khi thực hiện lệnh thì FSR=0×C2 và nội dung của địa chỉ (FSR) = 0×02.

gg. Lệnh: **XORLW** lệnh XOR hằng số với W

- Cú pháp: XORLW k
- Tác tố: $0 \le k \le 255$

- Thực thi: (W) XOR k → (W)
- Cờ ảnh hưởng: Z
- Chức năng: nội dung thanh ghi W được XOR với hằng số k 8 bit và kết quả lưu vào thanh ghi W.
- Chu kỳ thực hiện: 1.

Ví dụ: **XORLW 0×AF**

Trước khi thực hiện lệnh thì: W=0×B5.

Sau khi thực hiện lệnh thì: W=0×1A, Z=0.

hh. Lệnh: **XORWF** lệnh XOR W với f

- Cú pháp: XORLW f,d
- Tác tố: $0 \le f \le 127$, $d \in [0,1]$
- Thực thi: (W) XOR (f) → (dest)
- Cờ ảnh hưởng: Z
- Chức năng: nội dung thanh ghi W được XOR với nội dung thanh ghi f. Nếu d= 0 thì kết quả được lưu vào thanh ghi W. Nếu d= 1 thì kết quả lưu vào thanh ghi f.

IV. NGÔN NGỮ LẬP TRÌNH C CỦA CCS C:

1. GIỚI THIỆU CCS C:

CCS C là trình biên dịch dùng ngôn ngữ C cho vi điều khiển PIC đây là ngôn ngữ cấp cao giúp viết chương trình dễ dàng hơn ngôn ngữ Assembly.

Mã lệnh được tối ưu khi biên dịch.

CCS chứa nhiều hàm phục vụ cho mọi mục đích và có nhiều cách lập trình cho một vấn đề dẫn đến khác nhau về tốc độ thực thi mã và bộ nhớ chương trình, sự tối ưu của chương trình là do người lập trình quyết định.

2. NGÔN NGỮ LẬP TRÌNH C TRÊN CCS C:

Để viết một chương trình C mới chạy CCS vào New tạo một Project mới.

Trên thanh toolbar: Chọn “Microchip 12 bit” để viết chương trình cho PIC 12 bit. “Microchip 14 bit” để viết chương trình cho PIC 14 bit. “Microchip PIC18” để viết chương trình cho PIC18. Chọn “Compiler” để biên dịch chương trình.

Cấu trúc một chương trình C viết trong CCS:

```
#include <16F877A.h> // khai báo PIC sử dụng của chương trình
#device RS232                    // Khai báo thiết bị cần sử dụng
#use delay(clock=20000000)       // khai báo hàm delay
. . . .
Int16 a,b;                       // khai báo biến
. . . .
Void xu_ly_ADC ( )               //chương trình con
{ . . .
. . .
}
```

```
#INT_TIMER1                    // khai báo ngắt
Void xu_ly_ngat_timer ( )      //chương  trình xử lí ngắt
{ . . .
. . .
}
Main ( )                       // chương trình chính
{ . . .
. . .
}
```

3. KHAI BÁO VÀ SỬ DỤNG BIẾN, HẰNG, MẢNG:

a. Khai báo biến, hằng, mảng:

Các loại biến sau được hỗ trợ:

int1 : số 1 bit = true hay false (0 hay1)
int8 : số nguyên 1 byte (8 bit)
int16 : số nguyên 16 bit
int32 : số nguyên 32 bit
char : ký tự 8 bit
float : số thực 32 bit
short : mặc định như kiểu int1
byte : mặc định như kiểu int8
int : mặc định như kiểu int8
long : mặc định như kiểu int16

Thêm signed hoặc unsigned phía trước để chỉ đó là số có dấu hay không dấu.

- Khai báo hằng:

  Ví dụ: int8 const a=12;// a là hằng số có giá trị là 12

- Khai báo 1 mảng hằng số:

Ví dụ: int8 const a[2]={1,2,0};// mảng có 3 phần tử và chỉ số mảng đầu tiên là 0 với a[0]=1 và độ lớn của một phần tử là 1byte (hay 8bit).

Đối với vi điều khiển PIC16F877A thì chỉ số mảng có kích thước tối đa là 256 byte.

b. Cách sử dụng biến:

Khi sử dụng các phép toán cần lưu ý: tràn số, tính toán với số âm, đổi kiểu và ép kiểu.

Giống như C trong lập trình C cho máy tính. Biến có thể được khai báo như toàn cục hay cục bộ. Biến khai báo trong hàm sẽ là cục bộ và chỉ dùng được trong hàm đó, kể cả trong hàm main(). Ngoài ra còn có thể khai báo ngay trong 1 khối lệnh, và cũng chỉ tồn tại trong khối lệnh đó.

4. CÁC CẤU TRÚC LỆNH:

Gồm các lệnh như while . . do, case, . . .

- while (expr) stmt: xét điều kiện trước rồi thực thi biểu thức sau.
- do stmt while (expr): thực thi biểu thức rồi mới xét điều kiện sau.
- Return: dùng cho hàm có trả về trị, hoặc không trả về trị cũng được, khi đó chỉ cần dùng: return (nghĩa là thoát khỏi hàm tại đó).
- Break: ngắt ngang (thoát khỏi) vòng lặp while. Continue: quay trở về đầu vòng lặp while.

Bảng 3-3: Tập lệnh của ngôn ngữ C

STATEMENT	EXAMPLE

if (expr) **stmt;** [else **stmt;**]	if (x==25)
	x=1;
	else
	x=x+1;

while **(expr) stmt;**	while (get_rtcc()!=0) putc('n');

do **stmt** while **(expr);**	do {
	putc(c=getc());
	} while (c!=0);

for **(expr1;expr2;expr3) stmt;**	for (i=1;i<=10;++i) printf("%u\r\n",i);

switch (expr) {
case **cexpr: stmt; //one or more case**
[default**:stmt]**
... }
	switch (cmd) {
	case 0: printf("cmd 0");
	break;
	case 1: printf("cmd 1");
	break;
	default: printf("bad cmd");
	break; }

return **[expr];**	return (5);
goto **label;**	goto loop;
label: stmt;	loop: I++;
break;	break;
continue;	continue;
expr;	i=1;
;	;
{[stmt]} **Zero or more**	{a=1; b=1;}

Bảng 3-3. Tập lệnh ngôn ngữ C.

Các mục trong [ ] là có thể có hoặc không.

5. CHỈ THỊ TIỀN XỬ LÝ:

a. #ASM và #ENDASM:

Cho phép đặt 1 đoạn mã ASM giữa 2 chỉ thị này, chỉ đặt trong hàm. CCS định nghĩa sẵn 1 biến 8 bit RETURN để gán giá trị trả về cho hàm từ đoạn mã Assembly.

Khi sử dụng các biến không ở bank hiện tại, CCS sinh thêm mã chuyển bank tự động cho các biến đó. Nếu sử dụng #ASM ASIS thì CCS không sinh thêm mã chuyển bank tự động. Mã assembly đúng theo mã tập lệnh của vi điều khiển, không phải là mã lệnh của MPLAB.

```
Ví dụ:
int find_parity (int data) {
int count;
#asm
        movlw 0x8
        movwf count
        movlw 0
loop:
        xorwf data,w
        rrf data,f
        decfsz count,f
        goto loop
        movwf _return_
#endasm
}
```

b. #INCLUDE:

Cú pháp: #include <filename> hay #include " filename"

Filename: tên file cho thiết bị có thể *.h hay *.c. Nếu chỉ định file ở đường dẫn khác thì thêm đường dẫn vào và luôn có để khai báo chương trình viết cho vi điều khiển nào và đặt ở dòng đầu tiên.

```
Ví dụ: #include <P16F877A.h> //khai báo chương trình viết cho PIC16F877A
#include <lcd.c>  // khai báo các hàm hay chương trình con cho LCD
```

c. #BIT, #BYTE, #LOCATE và #DIFINE:

#BIT id = x.y với id là tên biến, x là biến (8,16,32bit,....) hay hằng số địa chỉ thanh ghi, y là vị trí của bit trong biến x.

```
Ví dụ: #bit TMR1Flag = 0xb.2 //bit cờ ngắt timer1 ở địa chỉ 0xb.2 (PIC16F877A)
Khi đó TMR1Flag = 0 // xoá cờ ngắt Timer1
```

#BYTE id = x. Trong đó id là tên biến, x: địa chỉ thanh ghi.

```
Ví dụ: #BYTE portB=0xC6; // Thanh ghi PortB có giá trị là 0xC6
  Khi muốn xuất ra PortB giá trị 120 thì ta dùng lệnh portB=120;
```

LOCATE id = x giống như #byte id=x nhưng có thêm chức năng bảo vệ không cho CCS sử dụng địa chỉ đó vào mục đích khác.

DEFINE id text với text là chuỗi hay số id là tên biến.

d. #DEVICE:

Cú pháp # DEVICE chip option

chip: tên vi điều khiển sử dụng, không dùng tham số này nếu đã khai báo tên chip ở # include.

Option: toán tử tiêu chuẩn theo từng chip:

* = 5 dùng pointer 5 bit (tất cả PIC)
* = 8 dùng pointer 8 bit (PIC14 và PIC18)
* = 16 dùng pointer 16 bit (PIC14, PIC 18)

ADC=x sử dụng ADC x bit (8, 10, . . . bit tuỳ chip), khi dùng hàm read_adc( ), sẽ trả về giá trị x bit.

e. #ORG:

org **start, end**

org **segment**

#org **start, end *{ }***

Start, end: bắt đầu và kết thúc vùng ROM dành riêng cho hàm theo sau, hoặc để riêng không dùng.

Ví dụ:

```
    Org 0x30, 0x1F
Void xu_ly( )
{

} // hàm này bắt đầu ở địa chỉ 0x30

Org 0x30, 0x1F { }
// không có gì cả đặt trong vùng ROM này
-Thường thì không dùng ORG.
```


f. #USE:

#USE delay(clock=speed)//khai báo hàm delay cho vi điều khiển

Với speed là tốc độ dao động đang dùng.

Ví dụ: như dùng thạch anh 20MHz thì khai báo là: #USE delay(clock=20000000)
Khi sử dụng chỉ thị #USE delay(clock=20000000) thì gọi hàm delay như sau:
Delay_ms(100); // lệnh này để thực hiện delay 100ms.

#USE fast_io(port)

Port: là tên port: từ A-E (đối với PIC16F877A)

Khi dùng chỉ thị này thì trong chương trình nếu dùng các lệnh I/O như output_low(), . . . thì nó sẽ set chỉ với 1 lệnh, nhanh hơn so với khi không dùng chỉ thị này.

Trong hàm main( ) phải dùng hàm set_tris_x( ) để chỉ rõ chân vào ra thì chỉ thị trên mới có hiệu lực, không thì chương trình sẽ chạy sai.

Ví dụ: # use fast_io(A)

#USE I2C (options)

Thiết lập giao tiếp I2C.

Option bao gồm các thông số sau, cách nhau bởi dấu phẩy:

- Master: chip ở chế độ master
- Slave: chip ở chế độ slave
- SCL=pin : chỉ định chân SCL
- SDA=pin : chỉ định chân SDA
- ADDRESS=x : chỉ định địa chỉ chế độ slave
- FAST: chỉ định FAST I2C
- SLOW: chỉ định SLOW I2C
- RESTART_WDT: restart WDT trong khi chờ I2C_READ()

Ví dụ:

```
#use I2C(master, sda=pin_B0, scl = pin_B1)
#use I2C (slave, sda= pin_C4, scl= pin_C3, address = 0xa00, FORCE_HW)
```

#USE RS232 (options)

Thiết lập giao tiếp RS232 cho chip (có hiệu lực sau khi nạp chương trình cho chip, không phải giao tiếp RS232 đang sử dụng để nạp chip).

Option bao gồm:

- BAUD=x: thiết lập tốc độ baud rate: 19200, 38400, 9600, . . .
- PARITY=x: x= N,E hay O, với N: không dùng bit chẵn lẻ.
- XMIT=pin: set chân transmit (chuyển data)
- RCV=pin : set chân receive (nhận data)
- Các thông số trên hay dùng nhất, các tham số khác sẽ bổ sung sau.

Ví dụ:

```
#use rs232(baud=19200,parity=n,xmit=pin_C6,rcv=pin_C7)
```

g. Một số chỉ thị tiền xử lý khác:

#CASE: cho phép phân biệt chữ hoa/thường của tên biến, dành cho người quen lập trình C.
#OPT n: với n=0-9: chỉ định cấp độ tối ưu mã.

#PRIORITY ints: với ints là danh sách các ngắt theo thứ tự ưu tiên thực hiện khi có nhiều ngắt xảy ra đồng thời, ngắt đứng đầu sẽ là ngắt ưu tiên nhất.

6. CÁC HÀM XỬ LÝ SỐ, XỬ LÝ BIT, DELAY:

a. Các hàm xử lý số:

Bao gồm các hàm:

- Sin(), cos(), tan(), asin(), acos(), atan()
- Asin(), acos(), atan(): là các hàm arcsin, arccos, arctan
- Abs() : lấy giá trị tuyệt đối
- Exp() :là hàm mũ ex
- Log() : hàm logarit
- Log10(): hàm logarit cơ số 10
- Pow() : hàm tính lũy thừa
- Sqrt() : hàm tính căn thức

Các hàm này có thể làm cho chương trình chạy chậm vì trên vi điều khiển không có bộ nhân và chia phần cứng, do đó nếu không đòi hỏi tốc độ thì dùng các hàm này cho đơn giản.

b. Các hàm xử lý bit và các phép toán:

Bao gồm các hàm sau:

- Shift_right()
- Shift_left()
- Bit_clear()

- Bit_set()
- Bit_test()
- Swap()
- Make8()
- Make16()
- Make32()

Hàm Shift_right(address,byte,value)
Hàm Shift_left(address,byte,value)

- Dịch phải (trái) 1 bit vào một mảng hay một cấu trúc.
- Địa chỉ có thể là địa chỉ mảng hay địa chỉ trỏ tới cấu trúc.

Hàm Bit_clear(var,bit)
Hàm Bit_set(var,bit)

- Bit_clear dùng để xóa bit được định bởi vị trí bit trong biến var.
- Bit_set dùng để set =1 bit được định bởi vị trí trong biến var.
- Var: biến 8,16,32 bit bất kì.
- Bit: vị trí clear (set): từ 0-7 (biến 8 bit), từ 0-16 (biến 16bit), từ 0-32 (biến 32bit).

Ví dụ:

```
Int8 x;
x=9;                //x=0b1001
bit_clear(x,0); //x=0b1000
```


Hàm Bit_test(var,bit)

- Dùng để kiểm tra vị trí bit trong biến var, hàm trả về 0 hay 1 là giá trị bit trong biến var.
- var là biến 8, 16 hay 32 bit
- bit là vị trí bit trong biến var

Ví dụ: nếu muốn kiểm tra x=256 chưa thì có thể sử dụng
If (x >=256) thì mất gần 5us
If (bit_test(x,9)) thì chỉ mất 0.4us (đối với thạch anh 20MHz)

Hàm Swap(var)

- Var là biến một byte
- Hàm này đảo vị trí của 4 bit cao cho 4 bit thấp trong một byte. Hàm này không trả về giá trị.


```
Ví dụ:
Int8 x;
X=0b00000101;       //x=0b00000101
Swap(x);            //x=0b01010000
```

Hàm Make8(var,offset)

- Hàm này trích 1 byte từ biến var
- Var là biến 8,16,32 bit
- Offset là vị trí của byte cần trích (0,1,2,3)

Ví dụ:

```
X=0x12A4;
Y=Make8(x,2); // y=02h
```

Hàm Make16(varhigh,varlow)

- Trả về giá trị 16 bit kết hợp từ hai biến 8bit là varhigh và varlow, byte cao là varhigh byte thấp là varlow.

```
Ví dụ:
        X=0x02;
        Y=0xAF;
Z=make16(x,y); // z=0x02AF
```

Hàm Make32(var1,var2,var3,var4)

- Hàm trả về giá trị 32 bit kết hợp giá trị 8 bit hay 16 bit từ var1 đến var4 trong đó từ var2 đến var4 có thể có hoặc không. Giá trị var1 sẽ là MSB, kế tiếp là var2,... Nếu tổng số bit kết hợp nhỏ hơn 32 bit thì 0 được thêm vào MSB để đủ 32 bit.

```
Ví dụ:
                Int a=0x01, b=0x02, c=0x03, d=0x04; // các giá trị hex
Int32 e ;
e = make32 (a,b,c,d);          // e = 0x01020304
e = make32 (a,b,c,5) ;         // e = 0x01020305
e = make32 (a,b,8);            // e = 0x00010208
e = make32 (a,0x1237) ;        // e = 0x00011237
```

c. Các hàm xử lý bit và các phép toán:

Để sử dụng các hàm delay thì cần phải có khai báo tiền xử lí ở đầu file.

Như thạch anh 20MHz thì khai báo là #USE delay(clock=20000000)

Có ba hàm phục vụ delay:

Hàm delay_cycles(count):

- Count : là hằng số từ 0-255 là số chu kì lệnh, 1 chu kì lệnh bằng 4 chu kì máy.

```
Ví dụ: delay_cycles(10);//delay 10 chu kì lệnh
```

Hàm delay_us(time) : hàm delay micro giây

- Time: là biến số thì có giá trị từ 0-255, là hằng số thì có giá trị từ 0-65355
- Hàm này không trả về giá trị.

```
Ví dụ: delay_us(1); // delay 1 micro giây
```

Hàm delay_ms(time) : hàm delay mili giây

- Time: có giá trị 0-255 nếu là biến, có giá trị từ 0-65355 nếu là hằng số.
- Hàm không trả về giá trị.

```
Ví dụ: delay_ms(1000); // hàm delay 1 giây
```

7. XỬ LÝ ADC VÀ CÁC HÀM IO TRONG C:

a. Các hàm xử lý ADC:

Hàm Setup_ADC(mode)

Hàm không trả về giá trị, dùng xác định cách thức hoạt động bộ biến đổi ADC. Tham số mode tuỳ thuộc file thiết bị *.h có tên tương ứng tên chip đang dùng, nằm trong thư mục DEVICES của CCS. Muốn biết có bao nhiêu tham số có thể dùng cho chip đó.

Hàm ADC_OFF: tắt hoạt động ADC (tiết kiệm điện, dành chân cho hoạt động khác).

Hàm ADC_CLOCK_INTERNAL: thời gian lấy mẫu bằng xung clock IC (mất 2-6 us) thường là chung cho các chip.

Hàm ADC_CLOCK_DIV_2: thời gian lấy mẫu bằng xung clock / 2 (mất 0.4 us trên thạch anh 20MHz)

Hàm ADC_CLOCK_DIV_8: thời gian lấy mẫu bằng xung clock / 8 (1.6 us)

Hàm ADC_CLOCK_DIV_32: thời gian lấy mẫu bằng xung clock / 32 (6.4 us)

b. SETUP_ADC_port (value):

Xác định chân lấy tín hiệu analog và điện thế chuẩn sử dụng. Tùy thuộc bố trí chân trên chip, số chân và chân nào dùng cho ADC và số chức năng ADC mỗi chip mà value có thể có những giá trị khác nhau, với Vref: áp chuẩn, Vdd: áp nguồn

Sau đây là các hàm khai báo ADC của 16F877A:

Hàm ALL_ANALOGS: dùng tất cả chân sau làm analog : A0 A1 A2 A3 A5 E0 E1 E2 (Vref=Vdd)

Hàm NO_ANALOG: không dùng analog, các chân đó sẽ là chân I /O.

Hàm AN0_AN1_AN3: A0 A1 A3, Vref = Vdd

Hàm AN0_AN1_VSS_VREF: A0 A1 VRefh = A3

Hàm AN0_AN1_AN4_AN5_AN6_AN7_VREF_VREF: A0 A1 A5 E0 E1 E2 VRefh=A3, VRefl=A2.

Hàm AN0_AN1_AN2_AN4_AN5_VSS_VREF: A0 A1 A2 A5 E0 VRefh=A3

Hàm AN0_AN1_AN4_AN5_VREF_VREF: A0 A1 A5 E0 VRefh=A3 VRefl=A2

Hàm AN0_AN1_AN4_VREF_VREF: A0 A1 A5 VRefh=A3 VRefl=A2

Hàm AN0_VREF_VREF: A0 VRefh=A3 VRefl=A2

Ví dụ: setup_adc_ports (AN0_AN1_AN3) ; // A0, A1, A3 nhận analog, áp nguồn +5V cấp cho IC sẽ là điện áp chuẩn

c. SETUP_ADC_channel (channel):

Chọn chân để đọc tín hiệu analog bằng lệnh Read_ADC( ). Giá trị channel tuỳ số chân chức năng ADC mỗi chip. Với 16F877A, channel có giá trị từ 0 -7: 0-chân A0, 1-chân A1, 2-chân A2, 3-chân A3, 4-chân A5, 5-chân E0, 6-chân E1, 7-chân E2.

Hàm không trả về trị. Nên delay 10 µs sau hàm này rồi mới dùng hàm read_ADC ( ) để bảo đảm kết quả đúng. Hàm chỉ hoạt động với A/D phần cứng trên chip.

d. Read_ADC(mode):

Dùng đọc giá trị ADC từ thanh ghi chứa kết quả biến đổi ADC.

Nếu giá trị ADC là 8 bit như khai báo trong chỉ thị #DEVICE, giá trị trả về của hàm là 8 bit, ngược lại là 16 bit nếu khai báo #DEVICE sử dụng ADC 10 bit trở lên.

Khi dùng hàm này thì sẽ chuyển đổi ADC của ngõ vào đã chọn trong hàm Set_ADC_channel( ) trước đó. Nghĩa là mỗi lần chỉ đọc 1 kênh muốn đổi sang đọc chân nào, dùng hàm set_ADC_channel( ) cho chân đó. Nếu không có đổi chân, dùng read_ADC( ) bao nhiêu lần cũng được.

Mode có thể có hoặc không, gồm có:

- ADC_START_AND_READ : giá trị mặc định.
- ADC_START_ONLY : bắt đầu chuyển đổi và trả về.

- ADC_READ_ONLY : đọc kết quả chuyển đổi lần cuối.

#DEVCE	8 bit	10 bit	11 bit	16 bit
ADC=8	0-255	0-255	00-255	00-255
ADC=10	x	0-1023	x	x
ADC=11	x	x	0-2047	x
ADC=16	0-65280	0-65472	0-65504	0-65535

Bảng 3-4. Kết quả đọc ADC.

PIC16F877A chỉ hỗ trợ ADC 8 và 10 bit.

e. Các hàm IO trong C:

Bao gồm các hàm sau:

- Output_low()
- Output_high()
- Output_bit()
- Input()
- Ouput_X()
- Input_X()
- port_b_pullups()
- Set_tris_X()

Hàm Output_low (pin), Output_high (pin)

- Dùng thiết lập mức 0 (low, 0V) hay mức 1 (high,5V) cho chân IC, pin chỉ vị trí chân.
- Hàm này sẽ đặt pin làm ngõ ra, xem mã asm để biết cụ thể.
- Hàm này thực hiện mất 2-4 chu kỳ máy. Cũng có thể xuất xung dùng set_tris_X() và #use fast_io.

Ví dụ : chương trình sau xuất xung vuông chu kỳ 500ms, duty =50% ra chân B0, nối B0 với 1 led sẽ làm nhấp nháy led.

```
#include <16F877A.h>
#use delay( clock=20000000)
Main()
{
        while(1)
{
        output_high(pin_B0) ;
        Delay_ms(250) ; // delay 250ms
        Output_low (pin_B0);
        Delay_ms (250 );
}
```

```
}
```

Hàm Output_bit (pin,value)

- pin: tên chân value: giá trị 0 hay 1
- Hàm này cũng xuất giá trị 0 / 1 trên pin, tương tự 2 hàm trên. Thường dùng nó khi giá trị ra tuỳ thuộc giá trị biến 1 bit nào đó, hay muốn xuất đảo của giá trị ngõ ra trước đó.

Ví dụ:
Khai báo int1 x; // x mặc định = 0
Trong hàm main:

```
Main()
{
        while (1 )
{
        output_bit( pin_B0, !x ) ;
        Delay_ms(250 );
}
}
```

Chương trình trên cũng xuất xung vuông chu kỳ 500ms,duty =50%

Hàm Input_bit (pin)

- Hàm này trả về giá trị 0 hay 1 là trạng thái của chân IC. Giá trị là 1 bit

Hàm Output_X(value)

- X là tên port có trên chip. Value là giá trị 1 byte.
- Hàm này xuất giá trị 1 byte ra port. Tất cả chân của port đó đều là ngõ ra.

Ví dụ :

```
Output_B ( 255 ); // xuất giá trị 11111111 ra port B
```

Hàm Input_X( )

- X: là tên port (A,B,C,D,E).
- Hàm này trả về giá trị 8 bit là giá trị đang hiện hữu của port đó.

Ví dụ: m=input_E();

Hàm Port_B_pullups(value)

- Hàm này thiết lập ngõ vào port B pullup (điện trở kéo lên). Value =1 sẽ kích hoạt tính năng này và value =0 sẽ ngừng.
- Chỉ các chip có port B có tính năng này mới dùng hàm này.

Hàm Set_tris_X(value)

- Hàm này định nghĩa chân IO cho 1 port là ngõ vào hay ngõ ra. Chỉ được dùng với #use fast_IO. Sử dụng #byte để tạo biến chỉ đến port và thao tác trên biến này chính là thao tác trên port.
- Value là giá trị 8 bit. Mỗi bit đại diện 1 chân và bit=0 sẽ set chân đó là ngõ vào, bit= 1 set chân đó là ngõ ra.

Ví dụ: chương trình thao tác trên portB

```
#include < 16F877A.h > //chip sử dụng là PIC16F877A
#use delay(clock=20000000) // dùng thạch anh 20MHz
#use Fast_IO(B)
#byte portB = 0x6 // 16F877 có port b ở địa chỉ 6h
#bit B0 = portB. 0 // biến B0 chỉ đến chân B0
#bit B1=portB.1 // biến B1 chỉ đến chân B1
#bit B2=portB.2 // biến B2 chỉ đến chân B2
#bit B3=portB.3 // biến B3 chỉ đến chân B3
#bit B4=portB.4 // biến B4 chỉ đến chân B4
#bit B5=portB.5 // biến B5 chỉ đến chân B5
#bit B6=portB.6 // biến B6 chỉ đến chân B6
#bit B7=portB.7 // biến B7 chỉ đến chân B7
Main()
{
      set_tris_B (80) ; //B0-B6 ngõ ra, B7 ngõ vào
      if (B7) //nếu ngõ vào chân B7 là 1 thì xuất ra B0-B6 giá trị là 1
{
      B1 = 1;
      B2 = 1;
      B3 = 1;
      B4 = 1;
      B5 = 1;
      B6 = 1;
}
      Else B1=B2=B3=B4=B5=B6= 0;
}
```


8. KHAI BÁO NGẮT VÀ CÁC HÀM THIẾT LẬP HOẠT ĐỘNG NGẮT :

a. Khai báo ngắt:

Mỗi họ PIC có số lượng nguồn ngắt khác nhau: PIC 14 có 14 ngắt, PIC 18 có 35 ngắt.

Danh sách các ngắt với chức năng tương ứng:

#INT_GLOBAL: ngắt toàn cục
#INT_AD: ngắt khi chuyển đổi A /D đã hoàn tất
#INT_CCP1: ngắt khi có Capture hay compare trên CCP1
#INT_CCP2: ngắt khi có Capture hay compare trên CCP2
#INT_COMP: kiểm tra bằng nhau trên Comparator
#INT_EEPROM: hoàn thành ghi EEPROM
#INT_EXT: ngắt ngoài
#INT_EXT1: ngắt ngoài 1
#INT_EXT2: ngắt ngoài 2
#INT_I2C: có hoạt động I2C
#INT_LOWVOLT: phát hiện áp thấp
#INT_PSP: có data vào cổng Parallel slave
#INT_RB: bất kỳ thay đổi nào trên chân B4 đến B7

#INT_RDA: data nhận từ RS 232 sẵn sàng
#INT_RTCC: tràn Timer 0
#INT_SSP: có hoạt động SPI hay I2C
#INT_TBE: bộ đệm chuyển RS 232 trống
#INT_TIMER0: một tên khác của #INT_RTCC
#INT_TIMER1: tràn Timer 1
#INT_TIMER2 : tràn Timer 2
#INT_TIMER3: tràn Timer 3
#INT_TIMER5 : tràn Timer 5
#INT_OSCF: lỗi OSC

b. Các hàm thiết lập hoạt động ngắt:

Hàm enable_interrupts(level)

- level là tên các ngắt đã cho ở trên hay là GLOBAL để cho phép ngắt ở cấp toàn cục.
- Mọi ngắt của vi điều khiển đều có 1 bit cờ ngắt, 1 bit cho phép ngắt. Khi có ngắt thì bit cờ ngắt lên mức 1, nhưng ngắt có hoạt động được hay không tuỳ thuộc bit cho phép ngắt. Hàm enable_interrupts (int_xxx) sẽ cho phép ngắt. Nhưng tất cả các ngắt đều không thể thực thi nếu bit cho phép ngắt toàn cục bằng 0, hàm enable_interrupts(global) sẽ cho phép ngắt toàn cục.

Ví dụ: để cho phép ngắt timer0 và timer1 hoạt động:

```
enable_interrupts (int_timer0);
enable_interrupts (int_timer1);
enable_interrupts (global);
```


Hàm disable_interrupts (level)

- level giống như trên.
- Hàm này vô hiệu 1 ngắt bằng cách set bit cho phép ngắt = 0.
- disable_interrupts(global) set bit cho phép ngắt toàn cục =0, cấm tất cả các ngắt.
- Không dùng hàm này trong hàm phục vụ ngắt vì không có tác dụng, cờ ngắt luôn bị xoá tự động.

Hàm clear_interupt(level)

- level không có GLOBAL.
- Hàm này xoá cờ ngắt của ngắt được chỉ định bởi level.

Hàm ext_int_edge (source, edge)

- Hàm này thiết lập nguồn ngắt ngoài EXTx là cạnh lên hay cạnh xuống.
- source: nguồn ngắt. Trên PIC 18 có 3 nguồn ngắt trên 3 chân EXT0, EXT1, EXT2 ứng với source = 0,1, 2. Các PIC khác chỉ có 1 nguồn EXT nên source = 0.
- edge: chọn cạnh kích ngắt, edge = L_TO_H nếu chọn cạnh lên (từ mức thấp chuyển lên mức cao) hay H_TO_L nếu chọn cạnh xuống.

c. Các hàm giao tiếp với máy tính qua cổng COM:

Để sử dụng chức năng này cần phải có hai khai báo: khai báo sử dụng RS232 và khai báo dao động của chip.

Ví dụ:

```
#use rs232 (baud=9600, parity=n, xmit=pin_C6, rcv=pin_C7)//tốc độ baud là 9600. không chẵn lẻ, chân truyền C6, chân nhận C7
```

```
#use delay (clock = 20000000 ) // nếu chip đang dùng OSC 20Mhz
```

Hàm xử lí liên quan:

Hàm printf (string)

Hàm Printf (cstring,values . . .)

- Dùng xuất chuỗi theo chuẩn RS232 ra PC.
- string là 1 chuỗi hằng hay 1 mảng ký tự (kết thúc bởi ký tự null).
- value là danh sách các biến, cách nhau bởi dấu phẩy.

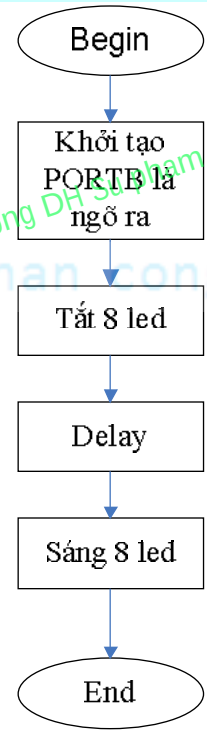
Lưu đồ chớp tắt PORTB (led đơn)

V. CÁC CHƯƠNG TRÌNH VÍ DỤ:

1 CHƯƠNG TRÌNH ĐIỀU KHIỂN 8 LED ĐƠN CHỚP TẮT:

Kết nối portB với 8 led đơn.

Lưu đồ của chương trình .



Chương trình điều khiển 8 led chớp tắt viết bằng ASM:


```
title           "choptat_Port_B.asm"
processor       p16f877a
include                 <P16f877a.inc>
__CONFIG        _CP_OFF & _PWRTE_ON & _WDT_OFF & _HS_OSC &_LVP_OFF
;=================================================
; Chuong trinh chinh
;=================================================
        count_1 equ     0x20
        count_2 equ     0x21
```

```
        org     0x000
        GOTO    start

        ;--------------------------------------------------------
        ;khoi tao Port B
        ;--------------------------------------------------------
start   org     0x0005
        banksel TRISB
        clrf    TRISB
        banksel PORTB
        ;--------------------------------------------------------
        ; vong lap chuong trinh chinh
        ;--------------------------------------------------------
        loop    clrf    PORTB
                call    delay
                movlw   h'ff'
                movwf   PORTB
                call    delay
                goto    loop
;=================================================
; CHUONG TRINH CON
;=================================================
;--------------------------------------------------------
; chuong trinh delay
;--------------------------------------------------------
        delay   clrf    count_1
        d2      clrf    count_2
        d1      decfsz  count_2
                goto    d1
                decfsz  count_1
                GOTO    d2
                return
;--------------------------------------------------------

                end
        ;=================================================
```


Chương trình điều khiển 8 led chớp tắt viết bằng C:

```
/*=================================================
* Title                 : Chuong trinh chop tat PortB
* Writer                :
* Hardware              : PIC16F877A
* Compiler              : CCS C
*=================================================*/
```

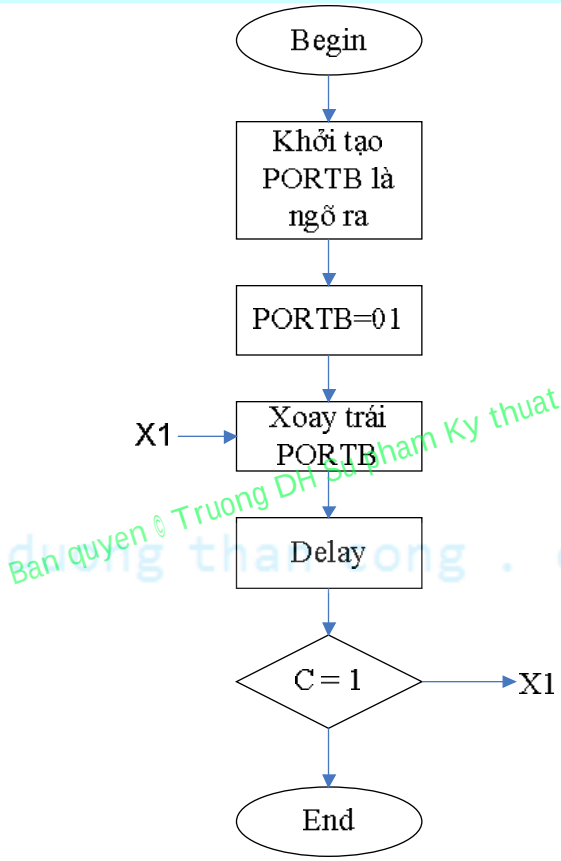
```
#include<16F877A.h>
#include<def_16f877a.h>
#fuses NOWDT,PUT,HS,NOPROTECT,NOLVP
#use delay(clock=20000000)
#use fast_io(b)
main()
  {
    trisb=0;
    while(true)
      {
                portb=0xff
                delay_ms(500);
                portb=0;
                delay_ms(500);
      }
  }
```

Lưu đồ chương trình Led sáng 1 điểm từ trái qua phải

2. CHƯƠNG TRÌNH ĐIỀU KHIỂN 1 ĐIỂM SÁNG DI CHUYỂN TỪ TRÁI SANG PHẢI:

Kết nối portB với 8 led đơn.

Lưu đồ điều khiển:



Chương trình 1 điểm sáng di chuyển từ trái sang phải viết bằng ASM:

```
        title           "chuongtrinhdich_led.asm"
        processor   p16f877a
        include     <p16f877a.inc>
    __CONFIG  _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_ON & _HS_OSC &
_LVP_OFF & _CPD_OFF
    ;-----------------------------------------------------
    ; Khoi tao cac bien
    ;-----------------------------------------------------
        count1      EQU     0x20 ; cac bien dung cho doan chuong trinh delay
        count2      EQU     0x21
    ;=====================================================
    ; CHUONG TRINH CHINH
    ;=====================================================
```

```
                ORG     0x000
                GOTO    start
start
;-----------------------------------------------------
; khoi tao PORT B
;-----------------------------------------------------
                BCF     STATUS,RP1
                BSF     STATUS,RP0
                CLRF    TRISB
                BCF     STATUS,RP0
        ;-----------------------------------------------
        ; vong lap chuong trinh chinh
        ;-----------------------------------------------
        main    MOVLW   b'00000001'
                MOVWF   PORTB
                clrc
        loop    CALL    delay
                RLF     PORTB,1
                BTFSS   STATUS,0
                GOTO    loop
                GOTO    main
;=====================================================
; CHUONG TRINH CON
;=====================================================
delay           clrf    count_1
d2              clrf    count_2
d1              decfsz  count_2
                goto    d1
                decfsz  count_1
                GOTO    d2
                return
                END
;=====================================================
```

Chương trình 1 điểm sáng di chuyển từ trái sang phải viết bằng C:

```
#include<16F877A.h>
#include<def_16f877a.h>
#fuses NOWDT,PUT,HS,NOPROTECT,NOLVP
#use delay(clock=20000000)
#use fast_io(b)
int8 a;
main()
  {
    trisb=0;
    while(true)
```

```
        {
                a=a<<1; // dich trai a 1bit
                if(a==256)
                {
                        a=1;
                }
                portb=~a;
                delay_ms(100);
        }
    }
```

3. CHƯƠNG TRÌNH ĐIỀU KHIỂN 8 LED SÁNG DỒN

Kết nối portB với 8 led đơn.

Lưu đồ điều khiển: bên dưới

Chương trình viết bằng ngôn ngữ ASM:

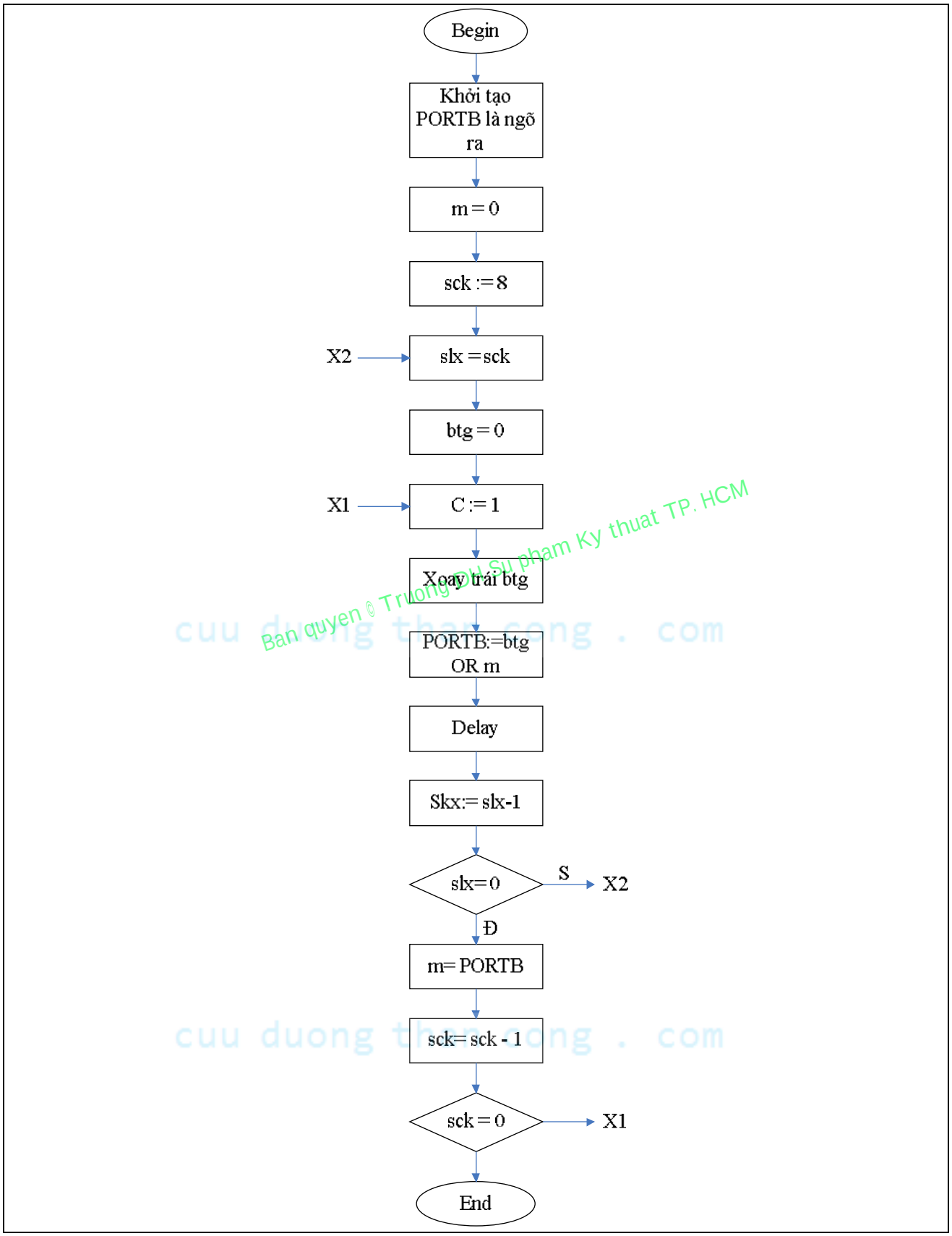

```
title       "sang don_Port_B.asm"
processor  p16f877a
include     <P16f877a.inc>
__CONFIG            _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_ON &  _XT_OSC
& _WRT_OFF & _LVP_OFF & _CPD_OFF
;-----------------------------------------------------------------
;Dat bien
;-----------------------------------------------------------------
            cblock      0x020
                        count1
                        counta
                        countb
                        m
                        sck
                        slx
                        btg
                        endc
;-----------------------------------------------------------------
;=================================================
; Chuong trinh chinh
;=================================================
                        org         0x0000
;-----------------------------------------------------------------
;khoi tao PortB
;-----------------------------------------------------------------
                        banksel     TRISB
```

```
                clrf    TRISB
                banksel PORTB
;-----------------------------------------------------------------
main2           clrf    PORTB
                call    delay
                clrf    m
                movlw   d'8'
                movwf   sck
        x21     movf    sck,0
                MOVWF   slx
                movlw   h'00'
                movwf   btg
                setc
        x11     rlf     btg
                clrc
                movf    btg,0
                iorwf   m,0
                movwf   PORTB
                call    delay
                decfsz  slx
                goto    x11
                movf    PORTB,0
                movwf   m
                decfsz  sck
                goto    x21
                goto    main2
        ;---------------------------------------------------------
        ;chuong trinh con delay
        ;---------------------------------------------------------
        delay   movlw   d'255'
                movwf   count1
        d1
                movlw   0xC7
                movlw   counta
                movlw   0x01
                movlw   countb
        delay_0 decfsz  counta,1
                goto    $+2
                decfsz  countb,1
                goto    delay_0
                decfsz  count1,1
                goto    d1
                return
                end
```

;===

Chương trình viết bằng ngôn ngữ C:

```
/*=============================================
* Title         : Chuong trinh sang don portB
* Writer        :
* Processer     : PIC16F877A
* Compiler      : CCS C
*=============================================*/
#include<16F877A.h>
#include<def_16f877a.h>
#fuses NOWDT,PUT,HS,NOPROTECT,NOLVP
#use delay(clock=20000000)
#use fast_io(b)
int8 sck,slx,bienxoay,bienluu,giatri;
main()
  {
    trisb=0;
    while(true)
    {
            sck=8;
            bienluu=0;
            portb=0;
            delay_ms(100);
            while(sck>0)
            {
                    bienxoay=1;
                    slx=sck;
                    while(slx>0)
                    {
                            giatri=bienluu|bienxoay;
                            portb=giatri;
                            delay_ms(100);
                            bienxoay=bienxoay<<1;
                            slx--;
                    }
                    bienluu=giatri;
                     sck--;
            }
    }
  }
```

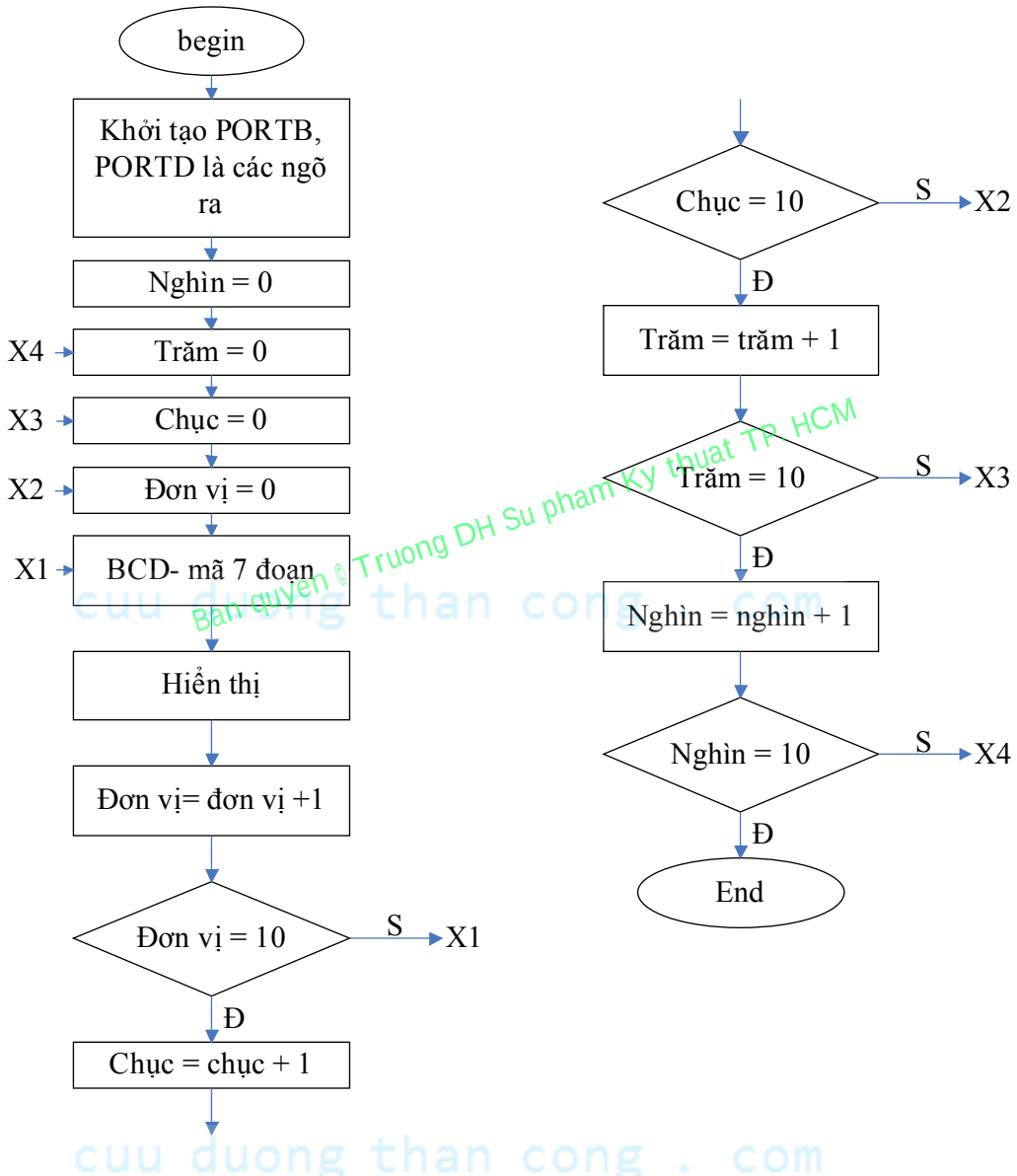


4. CHƯƠNG TRÌNH ĐIỀU KHIỂN ĐẾM TỪ 0 ĐẾN 9999 TRÊN LED 7 ĐOẠN:

Kết nối portB với 7 đoạn và dấu chấm thập phân của led 7 đoạn.

Kết nối portD với 8 transistor điều khiển quét 8 led 7 đoạn.

Lưu đồ điều khiển:



Chương trình viết bằng ngôn ngữ ASM:

```
        title       "dem BCD 0-9999 hien tren led 7 thanh.asm"
        processor   p16f877a
        include     <P16f877a.inc>
      __CONFIG      _CP_OFF & _PWRTE_ON & _WDT_OFF & _HS_OSC &_LVP_OFF
;================================================
; Chuong trinh chinh
;================================================
;---------- cac bien ------------------------------
            cblock  0x020
            count1
            count2
            count3
            nghin
            tram
            chuc
            dvi
            bien1
            bien2
            bien3
            bien4
            endc
;-------------------------------------------------
                    org     0x000
                    clrf    STATUS
                    movlw   0x00
                    movwf   PCLATH
                    goto    start
;-------------------------------------------------------
;khoi tao Port B
;-------------------------------------------------------
      start         org     0x005
                    BCF     STATUS,RP1
                    BSF     STATUS,RP0      ;chon BANK1
                    CLRF    TRISB           ;PORTB <- outputs
                    CLRF    TRISD
                    BCF     STATUS,RP0      ;chon BANK0
;-------------------------------------------------------
; vong lap chuong trinh chinh
;-------------------------------------------------------
      x5            movlw   0x00
                    movwf   nghin
      x4            movlw   0x00
                    movwf   tram
```

```
x3          movlw   0x00
            movwf   chuc
x1          movlw   0x00
            movwf   dvi
x2          call    bcd_7doan
            call    delayhthi
            incf    dvi
            movf    dvi,0
            xorlw   d'10'
            btfss   STATUS,Z
            goto    x2
            incf    chuc
            movf    chuc,0
            xorlw   d'10'
            btfss   STATUS,Z
            goto    x1
            incf    tram
            movf    tram,0
            xorlw   d'10'
            btfss   STATUS,Z
            goto    x3
            incf    nghin
            movf    nghin,0
            xorlw   d'10'
            btfss   STATUS,Z
            goto    x4
            goto    x5
bcd_7doan
            movf    dvi,0
            call    table
            movwf   bien1
            movf    chuc,0
            call    table
            movwf   bien2
            movf    tram,0
            call    table
            movwf   bien3
            movf    nghin,0
            call    table
            movwf   bien4
            return
delayhthi
            movlw   0x04
            movwf   count1
```

```
        del1    movlw   0x100
                movwf   count2
        del2    call    hienthi
                decfsz  count2
                goto    del2
                decfsz  count1
                goto    del1
                return
hienthi
                movf    bien1,0
                movwf   PORTB
                movlw   0xfe
                movwf   PORTD
                call    delay
                movf    bien2,0
                movwf   PORTB
                movlw   0xfd
                movwf   PORTD
                call    delay
                movf    bien3,0
                movwf   PORTB
                movlw   0xfb
                movwf   PORTD
                call    delay
                movf    bien4,0
                movwf   PORTB
                movlw   0xf7
                movwf   PORTD
                call    delay
                return

delay           movlw   0xff
                movwf   count3
dela1           decfsz  count3
                goto    dela1
                return
table           addwf   PCL,1
                DT 0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90
                End
;================================================
```

Chương trình viết bằng ngôn ngữ C:

-Chương trình đếm lên từ 0 đến 9999 trên led 7 đoạn viết bằng C:

```
/*=========================================
```

```
* Title          : Chuong trinh dem BCD 0-999999
* Writer         :
* Hardware       : PIC16F877A
* Compiler  : CCS C
*=============================================*/
#include<16F877A.h>
#include<def_16f877a.h>
#fuses NOWDT,PUT,HS,NOPROTECT,NOLVP
#use delay(clock=20000000)
#use fast_io(b)
#use fast_io(d)
int8 i,chuck,nghin,tram,chuc,dvi;
int32 a,bien;
const unsigned char dig[]={0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90};
void hex_bcd()
 {
      chuck=a/10000;
      a=a%10000;
      nghin=a/1000;
      a=a%1000;
      tram=a/100;
      a=a%100;
      chuc=a/10;
      dvi=a%10;
   }
void hienthi()
{
      i=0;
      while (i<200)
      {
            portb=dig[dvi];
            portd=0xfe;
            delay_us(100);
            portb=dig[chuc];
            portd=0xfd;
            delay_us(100);
            portb=dig[tram];
            portd=0xfb;
            delay_us(100);
            portb=dig[nghin];
            portd=0xf7;
            delay_us(100);
            portb=dig[chuck];
            portd=0xef;
```

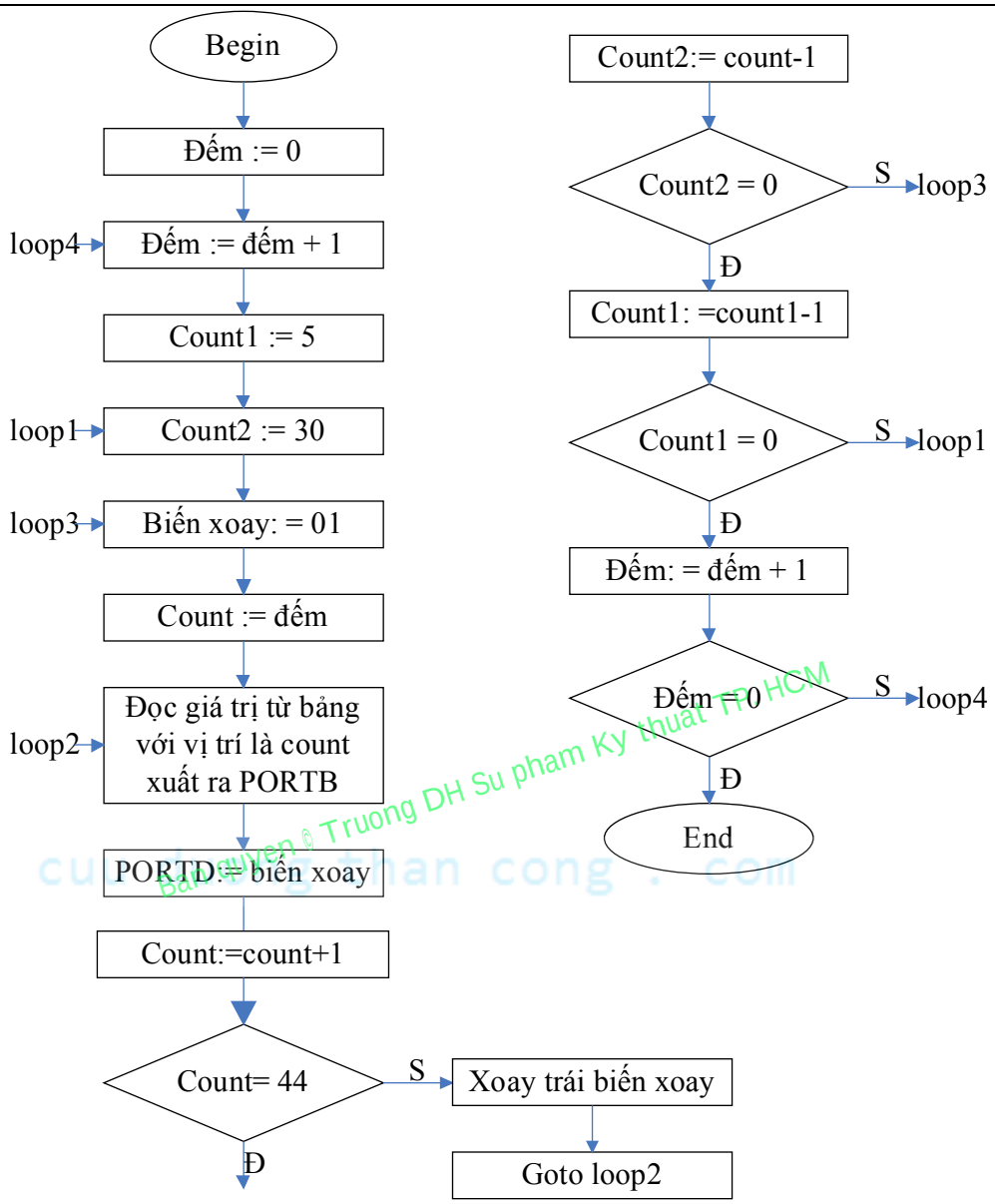
```
                delay_us(100);
                portb=dig[tramk];
                portd=0xdf;
                delay_us(100);
                i++;
        }
}
void main()
 {
                trisb=0x0;
                trisd=0x0;
                bien=0;
                while(1)
                {
                        bien=bien+1;
                         if(bien==100000)
                        {
                                bien=0;
                        }
                        a=bien;
                        hex_bcd();
                        hienthi();
                }
  }
```


5. CHƯƠNG TRÌNH ĐIỀU KHIỂN LED MA TRẬN HIỂN THỊ CHUỖI "SPKT":

Kết nối portB với hàng của led ma trận.

Kết nối portD với 8 transistor điều khiển cột.

Lưu đồ điều khiển:



Chương trình viết bằng ASM:

```
        title "chuongtrinhhienthi SPKT.asm"
        processor p16f877a
        include <p16f877a.inc>
        __CONFIG _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_ON  &_XT_OSC &
_WRT_OFF & _LVP_OFF & _CPD_OFF

;-------------------------------------------------
; Khoi tao cac bien
;-------------------------------------------------
                count1      EQU         0x20
                Bxoay       EQU         0x21
                count2      EQU         0x22
                count       EQU         0x23
```

```
            a           EQU         0x24
            dem         equ         0x25

;================================================
;CHUONG TRINH CHINH
;================================================
                        ORG         0x000
                        GOTO        start
        start
;------------------------------------------------
; Khoi tao PortB
;------------------------------------------------
                        BCF         STATUS,RP1
                        BSF         STATUS,RP0
                        CLRF        TRISB
                        CLRF        TRISD
                        BCF         STATUS,RP0
            ;-------------------------------------------------
            ; vong lap chuong trinh chinh
            ;-------------------------------------------------
        main            movlw       0x00
                        movwf       dem
        loop4           incf        dem,1
                        movlw       0x05
                        movwf       count1
        Loop1           movlw       0x30
                        movwf       count2
        loop3           movlw       0x01
                        movwf       bxoay
                        movf        dem,0
                        movwf       count
        Loop2           MOVF        count, 0
                        CALL        table
                        MOVWF       PORTB
                        movf        bxoay,0
                        movwf       PORTD
                        CALL        delay
                        movlw       0x00
                        MOVWF       PORTD
                        INCF        count, 0
                        XORLW       d'44'
                        BTFSC       STATUS,Z
                        GOTO        loop11
                        INCF        count, 1
```

```
                    rlf     bxoay
                    GOTO    Loop2
        loop11      decfsz  count2
                    goto    loop3
                    decfsz  count1
                    goto    Loop1
                    movf    dem,0
                    xorlw   d'43'
                    BTFSC   STATUS,Z
                    goto    main
                    goto    loop4
;================================================
; Cac chuong trinh con
;================================================
;------------------------------------------------
; Chuong trinh con cho ki thuat bang
;------------------------------------------------
        table
            ADDWF PCL,1
            DT 0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff
            DT 0xff,0xff,0xcd,0xb6,0xb6,0xd9        ;S
            DT 0xff,0xff,0x80,0xb7,0xb7,0xcf        ;P
            DT 0xff,0xff,0x80,0xeb,0xdd,0xbe        ;K
            DT 0xff,0xbf,0xbf,0x80,0xbf,0xbf        ;T
            DT 0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff
;------------------------------------------------
; Chuong trình con delay
;------------------------------------------------
            delay   movlw   0xff
                    movwf   a
            del1    decfsz  a
                    goto    del1
                    return
                    END
        ;=======================================
```

Chương trình viết bằng C:

```
        /*================================================
        * Title         : Chuong trinh hien thi chu SPKT
        * Writer        :
        * Hardware      : PIC16F877A
        * Compiler      : CCS C
        *================================================*/
        #include<16F877A.h>
        #include<def_16f877a.h>
```

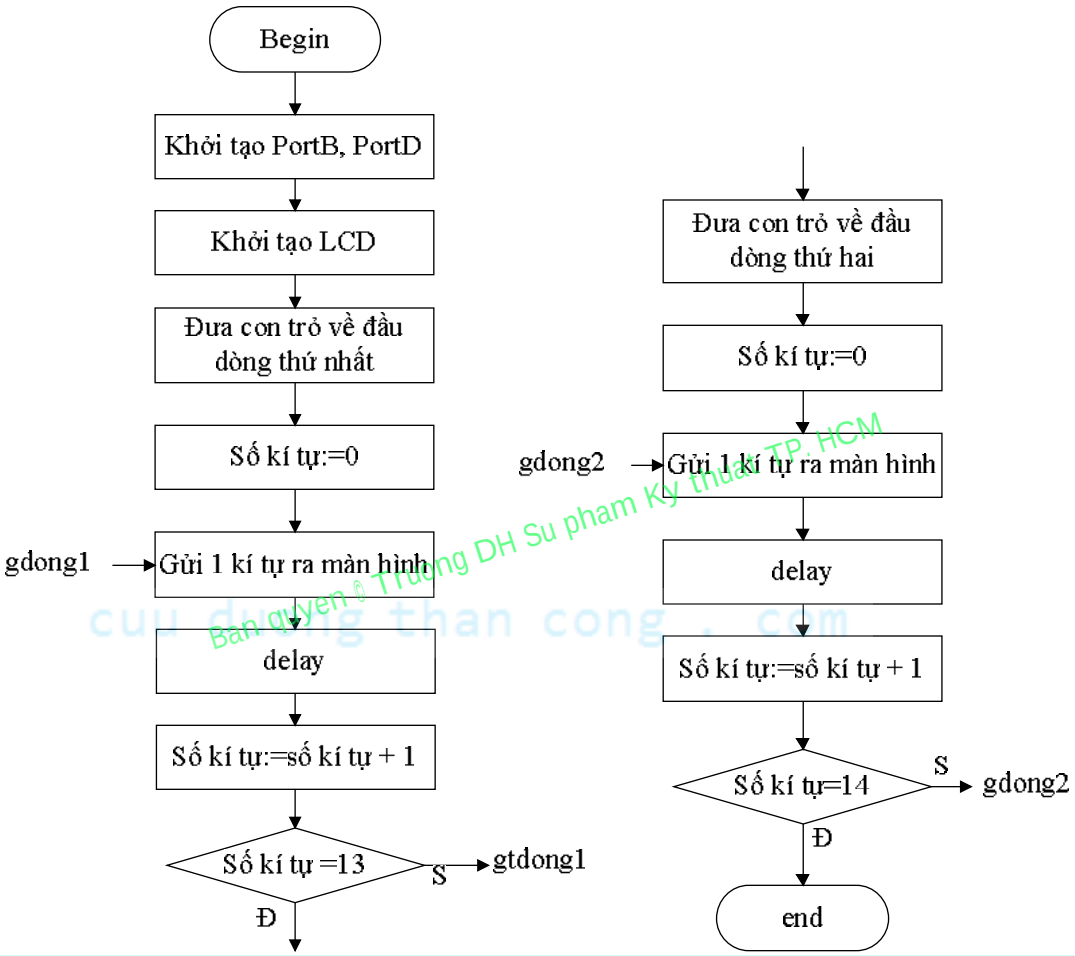
```
#fuses NOWDT,PUT,HS,NOPROTECT,NOLVP
#use delay(clock=20000000)
#use fast_io(b)
#use fast_io(d)
int8 dem,a,i,j;
int16 b;
const unsigned char dig[]={0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,\
0xcd,0xb6,0xb6,0xd9,0xff,0x80,0xb7,0xb7,0xcf,0xff,0x80,0xeb,0xdd,0xbe,\
0xff,0xbf,0xbf,0x80,0xbf,0xbf,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff};
void main()
  {
    trisb=0;
    trisd=0;
    while(1)
     {
                dem=0;
                while(dem<38)
                {
                        i=dem;
                        for(b=0;b<10000;b++)
                        {
                                a=a<<1;
                                if(a==256)
                                {
                                        a=01;
                                        i=dem;
                                }
                                portb=dig[i];
                                portd=a;
                                delay_us(100);
                                i++;

                        }
                        dem++;
                }
     }
  }
```

6. CHƯƠNG TRÌNH ĐIỀU KHIỂN LCD:

Kết nối portB với 8 đường dữ liệu của LCD.
Kết nối portD với 3 bit điều khiển của LCD.

Lưu đồ điều khiển:



Chương trình viết bằng ASM:

```
        title "hienthichu Truong DHSPKT TP Ho Chi Minh.asm"
        processor p16f877a
        include <p16f877a.inc>
        __CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON &_HS_OSC
;=================cuu duong than cong . com=================
; Khoi tao cac bien
;=============================================
        count1      EQU     0x20
        count2      EQU     0x21
        count3      EQU     0x22
        count       EQU     0x23
        a           EQU     0x24
;=============================================
;CHUONG TRINH CHINH
```

```
;================================================
                    ORG     0x000
                    clrf    STATUS
                    movlw   0x00
                    movwf   PCLATH
                    GOTO    start
      Start
;-----------------------------------------------
; Khoi tao PortB,PortD
;-----------------------------------------------
                    BCF     STATUS,RP1
                    BSF     STATUS,RP0 ;
                    CLRF    TRISB ;
                    clrf    TRISD
                    BCF     STATUS,RP0 ;
      ;-----------------------------------------------
      ; vong lap chuong trinh chinh
      ;-----------------------------------------------
      main          call    khoitaolcd
                    call    delay40ms
                    call    dong1
                    call    dong2
                    call    delay40ms
                    goto    $
      khoitaolcd
                    movlw   0x01
                    movwf   a
                    call    ghimadkhien
                    call    delay40ms
                    movlw   0x38
                    movwf   a
                    call    ghimadkhien
                    call    delay40ms
                    movwf   a
                    call    ghimadkhien
                    call    delay40ms
                    movlw   0x0C
                    movwf   a
                    call    ghimadkhien
                    call    delay40ms
                    movlw   0x01
                    movwf   a
                    call    ghimadkhien
                    call    delay40ms
```

```
                return
ghimadkhien
                movf    a,0
                movwf   PORTB
                bcf     PORTD,0
                bcf     PORTD,1
                BSF      PORTD,2
                BCF     PORTD,2
                Return

delay40ms       movlw   d'255'
                movlw   count1
 del1           movlw   0xff
                movwf   count2
 del2           decfsz  count2
                goto    del2
                decfsz  count1
                goto    del1
                return
dong1
                movlw   0x80
                movwf   a
                call    ghimadkhien
                call    delay
                clrf    count
loop2           movf    count,0
                call    table
                movwf   a
                call    ghidata
                call    delay
                incf    count,0
                xorlw   d'15'
                btfsc   STATUS,Z
                goto    exit
                incf    count,1
                goto    loop2
     exit       return

     dong2      movlw   0xc1
                movwf   a
                call    ghimadkhien
                call    delay
                clrf    count
     loop4      movf    count,0
```

```
                call    table1
                movwf   a
                call    ghidata
                call    delay
                incf    count,0
                xorlw   d'14'
                btfsc   STATUS,Z
                goto    exit1
                incf    count,1
                goto    loop4
        exit1   return

        ghidata movf    a,0
                movwf   PORTB
                bsf     PORTD,0
                bcf     PORTD,1
                bsf     PORTD,2
                bcf      PORTD,2
                return

    table       addwf PCL,1
                DT      "  Truong DHSPKT"
    table1      addwf PCL,1
                DT      "TP Ho Chi Minh"
;-------------------------------------------------------------------
        delay   movlw   d'255'
                movwf    count3
        dela1   decfsz  count3
                goto     dela1
                return
                end
;=======================================
;Ket thuc chuong trinh
;=======================================
```


Chương trình viết bằng C:

```
/*=============================================
* Title         : Hien chuoi truong DHSPKT TP Ho Chi Minh len LCD
* Writer        :
* Hardware      : PIC16F877A
* Compiler      : CCS C
*=============================================*/
#include <16F877A.h>
#include <DEF_16F877A.h>
```

```
#fuses HS,NOWDT,NOPROTECT,NOLVP
#use delay(clock=20000000)
#define RS  RD0
#define RW  RD1
#define E   RD2
#define LCD PORTB

/*Ham yeu cau goi lenh dieu khien LCD*/
void comnwrt()
{
      RS = 0;
      RW = 0;
      E  = 1;
      E  = 0;
      delay_ms(1);
}
/*Ham yeu cau goi du lieu hien thi len LCD*/
void datawrt()
{
      RS = 1;
      RW = 0;
      E  = 1;
      E  = 0;
      delay_ms(1);
}
/*Ham main*/
void main()
{
      trisb=0;
      trisd=0;
      delay_ms(100);
      LCD = 0x01;
      comnwrt();
      LCD = 0x01;
      comnwrt();
      LCD = 0x38;
      comnwrt();
      LCD = 0x0C;
      comnwrt();
      LCD = 0x81;
      comnwrt();
      LCD = 'T';                  // Xuat dong chu "Truong DHSPKT" ra dong 1 LCD
       datawrt();
      LCD = 'r';
```

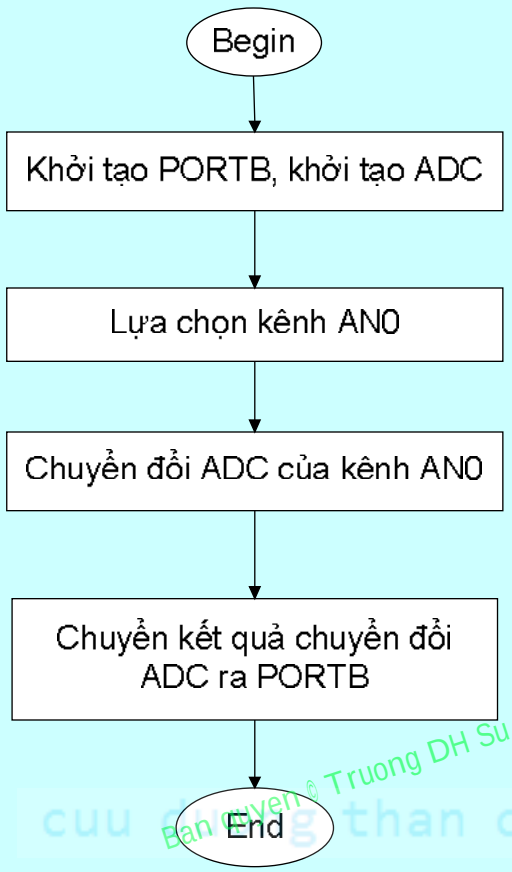
```
datawrt();
LCD = 'u';
datawrt();
LCD = 'o';
datawrt();
LCD = 'n';
datawrt();
LCD = 'g';
datawrt();
LCD = ' ';
datawrt();
LCD = 'D';
datawrt();
LCD = 'H';
datawrt();
LCD = 'S';
datawrt();
LCD = 'P';
datawrt();
LCD = 'K';
datawrt();
LCD = 'T';
datawrt();
LCD = 0xC0;              //Vi tri hang 2,cot 2
comnwrt();
LCD = 'T';               //Xuat dong chu "TP Ho Chi Minh" ra dong 2 LCD
datawrt();
LCD = 'P';
datawrt();
LCD = ' ';
datawrt();
LCD = 'H';
datawrt();
LCD = 'o';
datawrt();
LCD = ' ';
datawrt();
LCD = 'C';
datawrt();
LCD = 'h';
datawrt();
LCD = 'i';
datawrt();
LCD = ' ';
```

```
        datawrt();
        LCD = 'M';
        datawrt();
        LCD = 'i';
        datawrt();
        LCD = 'n';
        datawrt();
        LCD = 'h';
        datawrt();
    }
```

7. CHƯƠNG TRÌNH ĐIỀU KHIỂN ADC:

Kết nối portB với 8 đường dữ liệu của LCD.
Kết nối portD với 3 bit điều khiển của LCD.
Sử dụng kênh AN0 và hiển thị kết quả nhị phân trên 8 led hiển thị ở portB

Lưu đồ điều khiển:



Chương trình viết bằng ASM:

```
        title           "Chuyen doi ADC kenh AN0.asm"
        processor       p16f877a
        include  <P16f877a.inc>
        __CONFIG        _CP_OFF & _PWRTE_ON & _WDT_OFF & _HS_OSC & _LVP_OFF
;=============================================
; Chuong trinh chinh
;=============================================
                count_1     equ     0x20
                count_2     equ     0x21
                DLY12       equ     0x22
;-------------dat bien ADC---------------------
                            ORG     0x030
                REGAD1      RES     1
;-----------------------------------------
                            org     0x000
                            goto    start
;-------------------------------------------------------
;khoi tao Port B
;-------------------------------------------------------
                start       org     0x0005
                            banksel TRISB
                            clrf    TRISB
                            banksel PORTB
;---------------------
;Khoi tao cac ngo vao
;---------------------
                ADC         movlw   B'00000000'     ;Tat ca portA la ngo vao ADC
                            movwf   ADCON1          ;chon Vref = VDD
;-------------------------------------------------------
; vong lap chuong trinh chinh
;-------------------------------------------------------
;=============================
; Doc ADC
; Su dung bit GO/DONE
;=============================

                main        movlw   B'00000001'
                            call    docADC
                            movwf   REGAD1
                            movf    REGAD1,0
                            movwf   PORTB
                            goto    main
;=====================================
```

```
; CHUONG TRINH CON
;=====================================
        docADC    movwf    ADCON0
        DELAY12   decfsz   DLY12,F
                  goto     DELAY12
                  bsf      ADCON0,2
        GODONE    btfsc    ADCON0,2
                  goto     GODONE          ;Cho den khi convert xong
                  movf     ADRESL,W
                  return
        end
 ;=========================================
```

Chương trình viết bằng C:

```
#include <16F877.h>
#include <def_16f877a.h>
#fuses HS,NOWDT,NOPROTECT,NOLVP
#device 16F877*=16 ADC=8
#use delay(clock=20000000)
Int8 adc;
main()
{
    setup_adc(adc_clock_internal);
    setup_adc_ports(AN0);
    set_adc_channel(0);
    delay_ms(10);
    while(true)
    {
        adc=read_adc();
        output_B(adc);
    }
}
```


the end

return

Chương 4

VI ĐIỀU KHIỂN AVR

Chip AVR rất đa dạng ở đây chỉ trình bày chip AT90S8535.

1. Các chức năng của chip AVR AT90S8535 được tóm tắt như sau:

Có cấu trúc RISC công suất tiêu thụ thấp và thực hiện cao.

- Có 118 lệnh – thời gian thực hiện mỗi lệnh là 1 chu kỳ máy.
- Có 32 thanh ghi đa năng 8 bit.
- Có thể thực hiện 8 triệu lệnh trên 1 giây nếu hoạt động với tần số thạch anh là 8MHz.
- Dữ liệu và chương trình được lưu trong bộ nhớ không bay hơi (không mất dữ liệu khi mất điện).
- Có 8Kbyte bộ nhớ chương trình cho phép nạp nối tiếp trên hệ thống đang hoạt động ISP (In – System Programmable Flash) và cho phép nạp xoá chương trình 1000 lần.
- Có 512 byte EEPROM và đảm bảo cho phép nạp xóa 100000 lần.
- Có 512 byte SRAM nội bên trong.
- Có chức năng lập trình bảo mật chương trình.

Cấu trúc ngoại vi:

- Có 8 kênh ADC 10 bit.
- Cho phép lập trình UART.
- Có 2 timer/ counter 8 bit có bộ chia trước và có mode hoạt động so sánh.
- Có 1 timer/counter 16 bit có bộ chia trước, có so sánh và có chức năng bắt kịp và có chức năng điều chế độ rộng xung.
- Có một watchdog timer với tần số hoạt động trên chip.
- Có mạch so sánh điện áp tương tự trên chip.

Các cấu trúc đặc biệt của vi điều khiển:

- Có mạch điện reset khi cấp điện.
- Có xung đồng hồ thực – Real Time Clock (RTC) với mạch dao động chia sẵn và mode hoạt động counter.
- Có nhiều tín hiệu báo ngắt bên ngoài và cả bên trong.
- Có 3 chế độ hoạt giảm công suất: mode nghỉ (idle), mode tiết kiệm công suất (power save) và mode giảm công suất (power down).

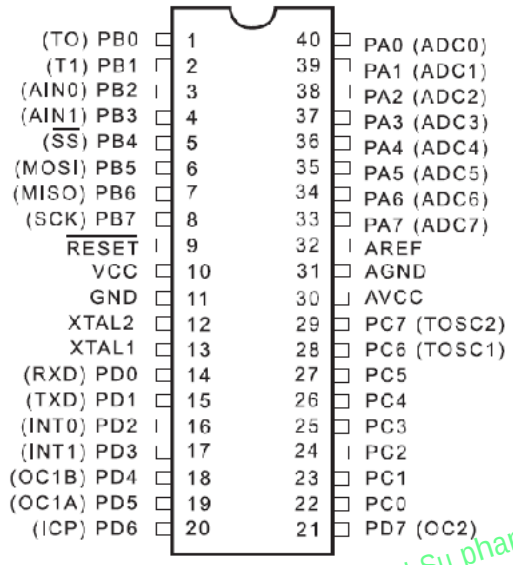
Công suất tiêu tán của chip AVR AT90S8535 được đo tại tần số hoạt động 4MHz, nguồn cung cấp 3V và nhiệt độ môi trường là 20 C:

- Chế độ hoạt động là 6,4 mA.
- Chế độ nghỉ (idle) là 1,9A.
- Chế độ power down là < 1µA.

Tần số làm việc là:

- Từ 0 đến 8 MHz đối với AT90S8535.
- Từ 0 đến 4 MHz đối với AT90LS8535.

2. Cấu trúc chân của AT90S8535: hình 1 trình bày sơ đồ chân của AVR AT90S8535.



Hình 1. Sơ đồ chân chip AVR AT90S8535.

Chip AVR AT90S8535 có 40 chân giống như họ 89 nhưng chỉ khác là chân cấp nguồn khác vị trí. Tín hiệu reset tích cực mức thấp. Có 4 port nhưng với tên thay đổi là portA, portB, portC và portD.

Chức năng các chân tín hiệu:

- Vcc : chân cấp nguồn.
- GND: chân nối mass 0V.
- Port A: (PA7.. PA0)

Port A là port xuất nhập (IO) 8 bit 2 chiều. Các chân của port A có thể tạo ra các điện trở kéo lên bên trong (được lựa chọn cho từng bit). Bộ đệm của portA có thể nhận dòng khoảng 20mA và có thể điều khiển thúc trực tiếp các led hiển thị. Khi các chân từ PA0 đến PA7 được dùng như là các ngõ vào và kéo xuống mức thấp ở bên ngoài thì chúng sẽ cung cấp dòng ra nếu các điện trở kéo lên bên trong được tác động.

Port A còn đóng vai trò là các ngõ vào của các bộ chuyển đổi ADC.

Các chân của port A sẽ ở trạng thái tổng trở cao khi reset bị tác động và ngay cả khi mạch dao động không hoạt động.

Port B: (PB7.. PB0)

Port B là port xuất nhập (IO) 8 bit 2 chiều với các điện trở kéo lên bên trong. Bộ đệm của portB có thể nhận dòng khoảng 20mA. Khi được dùng như là các ngõ vào và kéo xuống mức thấp ở bên ngoài thì chúng sẽ cung cấp dòng ra nếu các điện trở kéo lên bên trong được tác động.

Port B còn phục vụ nhiều chức năng với nhiều cấu trúc đặc biệt khác nhau được trình bày ở phần sau.

Các chân của port B sẽ ở trạng thái tổng trở cao khi reset bị tác động và ngay cả khi mạch dao động không hoạt động.

Port C: (PC7.. PC0)

Port C là port xuất nhập (IO) 8 bit 2 chiều với các điện trở kéo lên bên trong. Bộ đệm của port C có thể nhận dòng khỏang 20mA. Khi được dùng như là các ngõ vào và kéo xuống mức thấp ở bên ngoài thì chúng sẽ cung cấp dòng ra nếu các điện trở kéo lên bên trong được tác động.

Hai chân của Port C có thể được sử dụng như là ngõ vào của timer/counter2.

Các chân của port C sẽ ở trạng thái tổng trở cao khi reset bị tác động và ngay cả khi mạch dao động không hoạt động.

Port D: (PD7.. PD0)

Port D là port xuất nhập (IO) 8 bit 2 chiều với các điện trở kéo lên bên trong. Bộ đệm của port D có thể nhận dòng khỏang 20mA. Khi được dùng như là các ngõ vào và kéo xuống mức thấp ở bên ngoài thì chúng sẽ cung cấp dòng ra nếu các điện trở kéo lên bên trong được tác động.

Port D còn phục vụ nhiều chức năng với nhiều cấu trúc đặc biệt khác nhau được trình bày ở phần sau.

Các chân của port D sẽ ở trạng thái tổng trở cao khi reset bị tác động và ngay cả khi mạch dao động không hoạt động.

RESET:

Là ngõ vào tác động mức thấp. Xung reset dài hơn 50ns sẽ reset AVR ngay cả khi mạch dao động không hoạt động.

XTAL1:

Là ngõ vào của mạch khuếch đại dao động đảo và ngõ vào của mạch điện tạo dao động bên trong.

XTAL2:

Là ngõ ra của mạch khuếch đại dao động đảo.

AVCC:

Là chân cung cấp nguồn điện áp cho bộ chuyển đổi AD. Chân AVCC này sẽ được kết nối với Vcc thông qua một mạch lọc thông thấp. Phần sau sẽ trình bày hoạt động của mạch.

AREF:

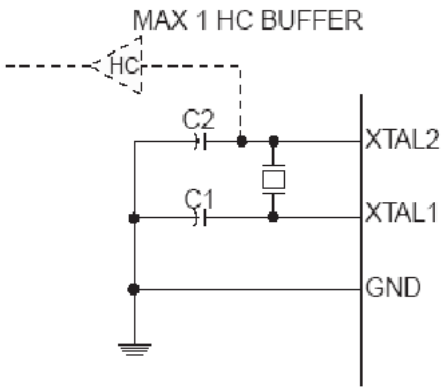
Là chân ngõ vào cung cấp điện áp tham chiếu (điện áp chuẩn) cho bộ chuyển đổi AD. Đối với hoạt động của ADC thì phải cung cấp một điện áp nằm trong khoảng từ AGND đến Avcc đến ngõ vào chân AREF.

AGND:

Là chân nối mass của tín hiệu tương tự. Nếu bo mạch có mass analog thì kết nối chân này đến mass analog, còn nếu không có thì nối chung với GND.

Sử dụng dao động thạch anh:

XTAL1 và XTAL2 theo thứ tự là ngõ vào và ngõ ra của mạch khuếch đại đảo được thiết kế sử dụng như mạch dao động nội được trình bày ở hình 2. Thạch anh hoặc mạch cộng hưởng ceramic đều có thể sử dụng để tạo dao động.



Hình 3. Kết nối thạch anh với 2 ngõ vào XTAL1 và XTAL2.

3. Cấu trúc phần cứng:

Cấu trúc phần cứng của AVR AT90S8535 được trình bày ở hình 3.

Các khối bên trong bao gồm:

- Khối khuếch đại thúc cho các port A, B, C, D (port driver), khối thanh ghi dữ liệu port (Data Register Port - DRP) và khối điều khiển hướng cho các port (Data Direction Register port - DDRP).

- Khối chuyển đổi tín hiệu tương tự sang số (ADC) và khối đa hợp chọn kênh ngõ vào (analog mul).

- Khối dao động nối với tụ thạch anh bên ngoài (oscillator) và khối dao động nhận tín hiệu dao động từ bên ngoài.

- Khối thời gian và điều khiển (timing and control).

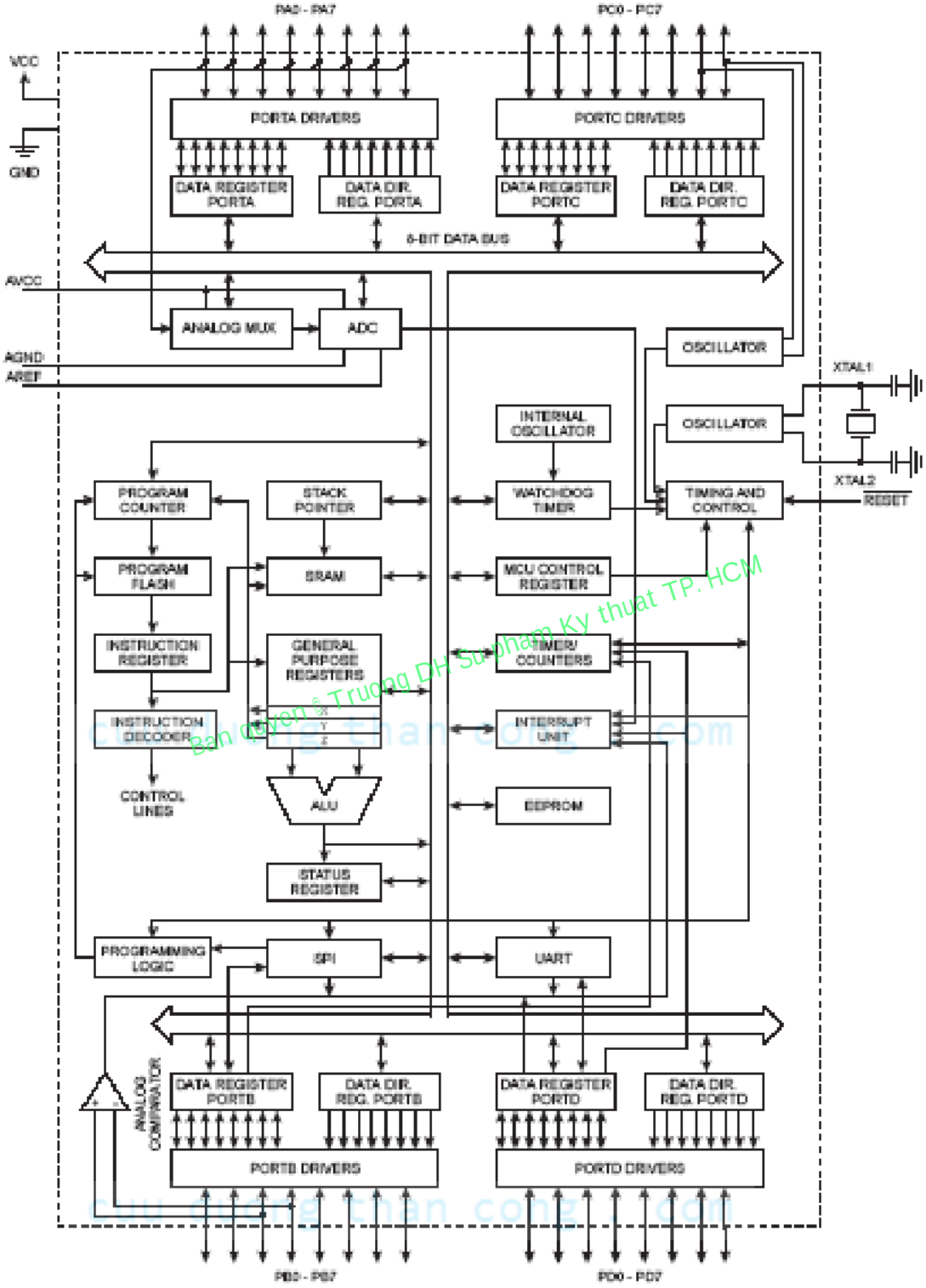
- Khối điều khiển lập trình (programming control): đây là khối quan trọng nhất của toàn bộ hệ thống.

- Khối thanh ghi con trỏ ngăn xếp Stack pointer (Sp): có chức năng quản lý bộ nhớ ngăn xếp (dùng bộ SRAM bên trong) để lưu trữ các dữ liệu tạm thời trong quá trình xử lý dữ liệu.

- Bộ nhớ SRAM dùng để lưu trữ các dữ liệu phục vụ cho chương trình và dùng làm bộ nhớ ngăn xếp để lưu trữ các dữ liệu tạm thời và lưu trữ địa chỉ khi thực hiện các chương trình con hay các chương trình con phục vụ ngắt.

- Khối thanh ghi bộ đếm chương trình Program Counter (PC): có chức năng quản lý bộ nhớ chương trình hay chính xác hơn là quản lý lệnh. Khi thanh ghi PC trỏ đến lệnh nào thì lệnh đó được thực hiện.

- Khối bộ nhớ chương trình Program Flash: dùng để lưu trữ mã lệnh của chương trình. Thanh ghi PC sẽ quản lý bộ nhớ này.



Hình 3. Sơ đồ cấu trúc bên trong của AVR AT90S8535.

- Thanh ghi lệnh (Instruction Register) có chức năng lưu trữ mã lệnh.

- Khối giải mã lệnh (Instruction Decoder) có chức năng giải mã lệnh để cho khối điều khiển chương trình biết lệnh yêu cầu thực hiện công việc xử lý gì. Sau khi giải mã xong các yêu cầu thực

hiện của lệnh sẽ chuyển sang cho khối điều khiển chương trình thực hiện. Rồi tiếp tục thực hiện việc giải mã lệnh tiếp theo.

- Khối thanh ghi đa năng và các thanh ghi x, y, z: có chức năng phục vụ cho việc lưu trữ dữ liệu để chương trình xử lý.

- Khối ALU có chức năng thực hiện các lệnh xử lý dữ liệu như cộng, trừ, nhân, chia, tăng, giảm, and, or, exor, so sánh, ... Khối này sẽ thực hiện các phép toán với các dữ liệu chứa trong các thanh ghi trên.

- Thanh ghi trạng thái – Status Register có chức năng cho biết trạng thái của dữ liệu sau khi xử lý dữ liệu bởi khối ALU.

- Khối dao động bên trong để tạo ra nhiều cấp tần số khác nhau để phục vụ cho các ứng dụng khác như truyền dữ liệu.

- Khối Watch Dog timer được tích hợp để phục vụ cho việc định thời thoát khỏi các vòng lặp vô hạn.

- Khối timer/counter dùng để phục vụ cho các bộ định thời cho các ứng dụng điều khiển và các ứng dụng nhận xung đếm từ bên ngoài.

- Khối điều khiển ngắt (interrupt): bao gồm các chức năng nhận tín hiệu yêu cầu ngắt từ bên ngoài, xử lý ngắt, ưu tiên ngắt, điều khiển cho phép hay không cho phép ngắt.

- Khối bộ nhớ EEPROM cho phép lưu trữ các dữ liệu mà khi mất điện thì dữ liệu vẫn còn, số lần cho phép ghi xoá lên đến 100 000 lần.

- Khối truyền dữ liệu bất đồng bộ UART: cho phép AVR truyền dữ liệu với các đối tượng khác.

- Bus dữ liệu bên trong dùng để kết nối tất cả các khối với nhau để trao đổi dữ liệu.

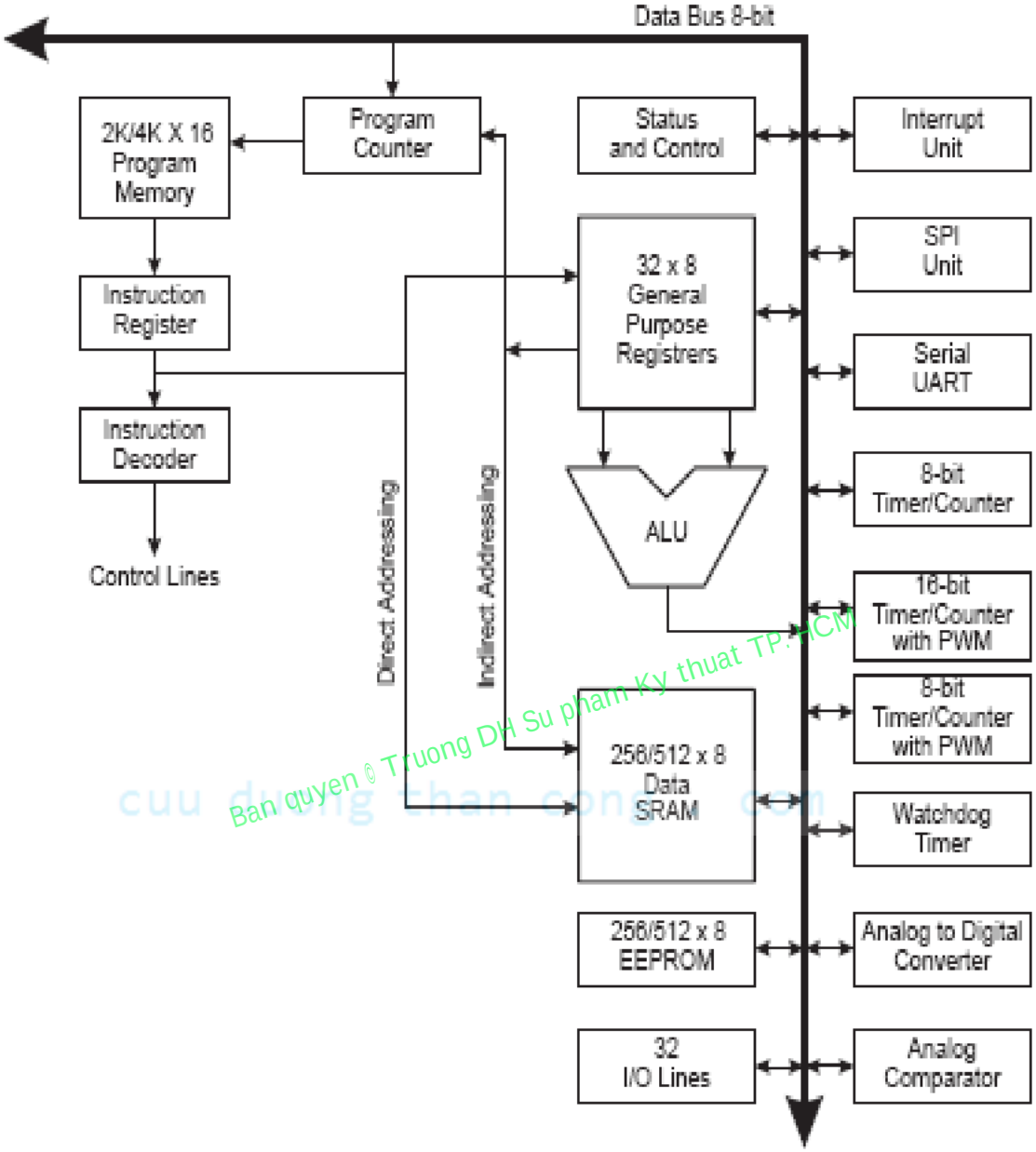
4. Tổng quan về cấu trúc AVR

Khái niệm file thanh ghi truy xuất nhanh chứa 32 thanh ghi 8 bit hoạt động đa năng tổng quát với thời gian truy xuất trong một chu kỳ xung clock duy nhất. Điều này có nghĩa là trong mỗi 1 chu kỳ xung clock – một phép toán trong khối ALU được thực hiện. Hai toán tử được xuất ra từ file thanh ghi, phép toán được thực hiện và kết quả được lưu trở lại trong file thanh ghi chỉ trong 1 chu kỳ xung clock.

Sáu trong 32 thanh ghi có thể được dùng như là các con trỏ thanh ghi địa chỉ gián tiếp 16 bit để định địa chỉ vùng nhớ dữ liệu (Data Space) – cho phép tính toán địa chỉ một cách hiệu quả. Một trong 3 con trỏ địa chỉ cũng được sử dụng như là con trỏ địa chỉ cho chức năng tìm kiếm hằng số bảng (for the constanr table look up function). Các thanh ghi chức năng này là X register, Y regiester và Z register.

Khối ALU thực hiện các phép toán đại số và các phép toán logic giữa các thanh ghi hoặc giữa hằng số và thanh ghi. Các phép toán chỉ xảy ra trên 1 thanh ghi cũng được thực hiện bởi khối ALU. Hình 4 trình bày cấu trúc RISC của vi điều khiển AVR AT90S8535.

Thêm vào các hoạt động của thanh ghi thì các kiểu định địa chỉ bộ nhớ theo qui ước cũng có thể được sử dụng trên file thanh ghi. Điều này được cho phép bởi file thanh ghi được gán bởi 32 địa chỉ vùng nhớ dữ liệu thấp nhất ($00 – $1F) cho phép chúng được truy xuất như là những ô nhớ bình thường.



Hình 4. Cấu trúc RISC của AVR AT90S8535.

Vùng nhớ IO có 64 địa chỉ phục vụ cho khối ngoại vi của CPU như các thanh ghi điều khiển (control registers), timer/counters, A/D converters và các chức năng IO khác. Vùng nhớ IO có thể truy xuất trực tiếp hoặc được xem như là một vùng nhớ dữ liệu tiếp theo sau file thanh ghi từ $20 đến $5F.

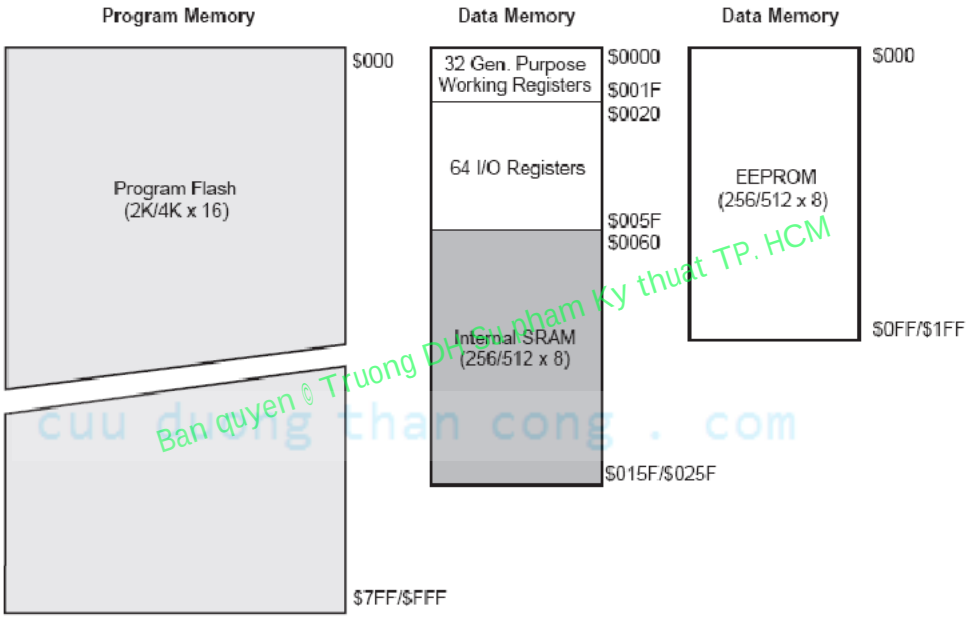
AVR dùng ý tưởng cấu trúc Harvard – với việc chia các vùng nhớ và bus cho bộ nhớ chương trình và bộ nhớ dữ liệu. Bộ nhớ chương trình được thực hiện với 2 tầng đường ống (two stage pipeline). Trong khi 1 lệnh được thực hiện thì lệnh tiếp theo sẽ được đón từ bộ nhớ chương trình. Ý tưởng này cho phép các lệnh được thực hiện trong mỗi chu kỳ xung clock. Bộ nhớ chương trình được tích hợp trong hệ thống bộ nhớ Flash.

Với các lệnh nhảy và lệnh gọi tương đối trong phạm vi 4k không gian bộ nhớ thì được truy xuất trực tiếp. Hầu hết các lệnh của AVR đều có từ mã lệnh 16 bit. Mỗi một địa chỉ ô nhớ chương trình chứa lệnh 16 bit hoặc 32 bit.

Khi thực hiện ngắt và lệnh gọi các chương trình con thì địa chỉ trở về chương trình chính lưu thanh ghi PC được lưu trong ngăn xếp. Bộ nhớ ngăn xếp được cấp phát trong vùng nhớ SRAM đa dụng và do đó kích thước bộ nhớ ngăn xếp sẽ bị giới hạn bởi kích thước toàn bộ bộ nhớ SRAM và việc sử dụng bộ nhớ SRAM. Tất cả các chương trình của người dùng phải được khởi tạo với SP trong thủ tục reset (trước khi chương trình con hoặc ngắt được thực hiện). Con trỏ SP quản lý bộ nhớ ngăn xếp có chiều dài 10 bit có thể được truy xuất đọc/ghi trong vùng nhớ IO.

Vùng nhớ dữ liệu SRAM 512 byte có thể dễ dàng được truy xuất thông qua 5 kiểu định địa chỉ khác nhau được xây dựng trong cấu trúc của AVR.

Các vùng nhớ trong cấu trúc của AVR là các bảng đồ nhớ tuyến tính và quy tắc. Cấu trúc bộ nhớ bên trong của AVR như hình 5:



Hình 5. Bảng đồ của các bộ nhớ.

Trong bảng đồ nhớ ở trên chúng ta thấy có 3 loại bộ nhớ khác nhau tích hợp trong AVR gồm bộ nhớ chương trình (Program Memory) và hai loại bộ nhớ dữ liệu (Data Memory): gồm bộ nhớ SRAM và bộ nhớ EEPROM.

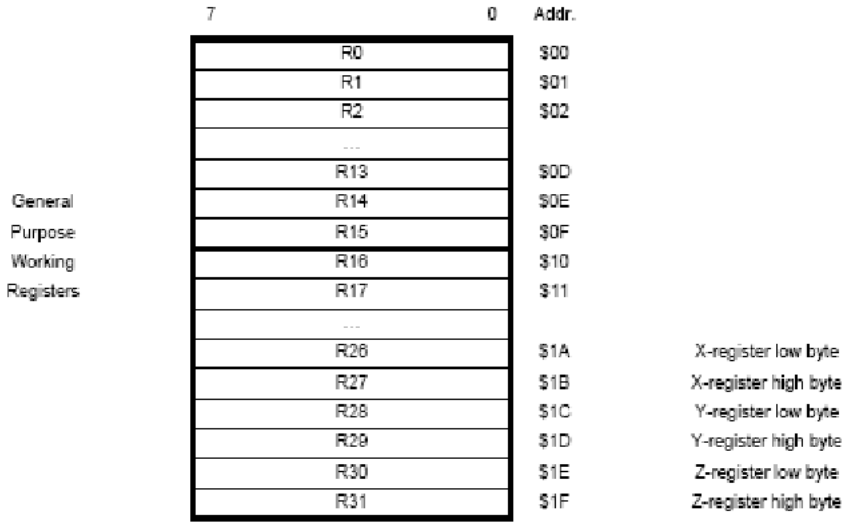
Bộ nhớ chương trình của AT90S8535 có dung lượng là 8Kbyte, mỗi một ô nhớ là 16 bit, còn các loại bộ nhớ dữ liệu thì mỗi một ô nhớ là 8 bit. Bảng đồ nhớ ở trên có 2 thông số thì thông số đứng trước là của AT90S4434, còn thông số đứng sau là của AT90S8535.

Bộ nhớ dữ liệu bên trong gồm có 3 thành phần: thứ nhất là 32 thanh ghi hoạt động đa năng có địa chỉ từ $000 đến $01fh, thành phần thứ 2 là 64 ô nhớ của các thiết bị ngoại vi IO và 512 byte SRAM.

Bộ nhớ dữ liệu EEPROM có 512 ô nhớ dùng để lưu trữ dữ liệu và không có gì đặc biệt.

File thanh ghi hoạt động đa năng:

Hình 6 trình bày cấu trúc của 32 thanh ghi hoạt động đa năng trong CPU.



Hình 6. Các thanh ghi hoạt động đa năng của CPU AVR.

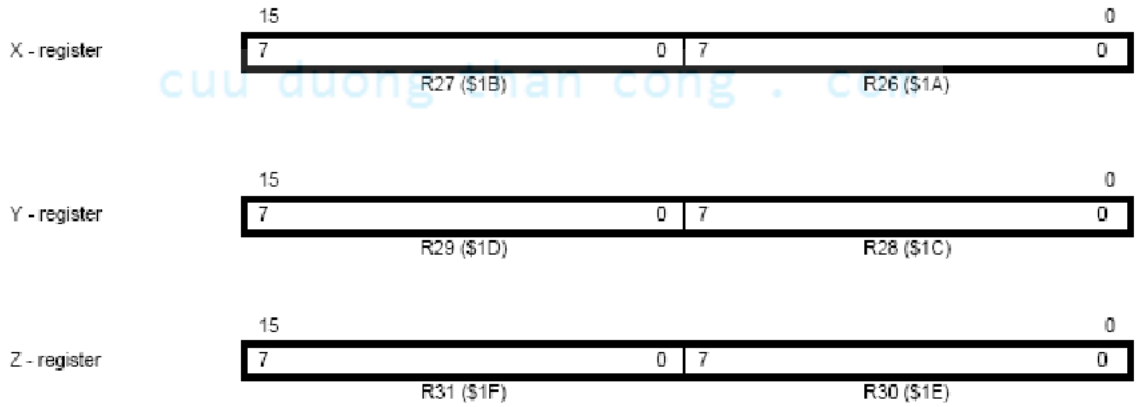
Tất cả các lệnh xử lý trong tập lệnh đều có thể truy xuất trực tiếp trên các thanh ghi này và chỉ thực hiện trong khoảng thời gian 1 chu kỳ máy - ngoại trừ 5 lệnh thực hiện các phép toán đại số và logic: SBCI, SUBI, CPI, ANDI, ORI xảy ra giữa hằng số và thanh ghi và lệnh LDI – nạp dữ liệu hằng số trực tiếp. Các lệnh này chỉ áp dụng cho phân nữa các thanh ghi còn lại trong file thanh ghi từ R16 đến R31. Các lệnh tổng quát SBC, SUB, CP, AND và tất cả các lệnh khác xảy ra giữa 2 thanh ghi và trên 1 thanh ghi đều có thể áp dụng cho toàn bộ file thanh ghi.

Như đã trình bày trong hình 6, mỗi thanh ghi cũng có thể được gán một địa chỉ bộ nhớ dữ liệu, định vị chúng trực tiếp trong 32 ô nhớ đầu tiên của vùng nhớ dữ liệu của người sử dụng. Mặc dù không sử dụng địa chỉ vật lý như các ô nhớ của SRAM nhưng vùng nhớ này được tổ chức cung cấp một sự linh hoạt rất lớn trong việc truy xuất của các thanh ghi như thanh ghi X, Y và Z có thể được thiết lập để chỉ tới bất kỳ thanh ghi nào trong file thanh ghi.

Thanh ghi X, Y và Z:

Các thanh ghi từ R26 đến R31 được kết hợp lại tạo ra 3 thanh ghi X, Y và Z để sử dụng cho các mục đích đa năng khác. Các thanh ghi này là các con trỏ địa chỉ để định địa chỉ gián tiếp các ô nhớ trong vùng nhớ dữ liệu.

Hình 7. Trình bày các thanh ghi X, Y và Z.



Hình 7. Trình bày các thanh ghi X, Y và Z.

Chú ý thứ tự các thanh ghi khi kết hợp.

Trong các kiểu định địa chỉ khác nhau, các thanh ghi này có chức năng lưu trữ địa chỉ cố định, địa chỉ tăng lên để truy xuất đến ô nhớ tiếp theo sau khi thực hiện xong lệnh, địa chỉ giảm xuống để truy xuất ô nhớ kế sau khi thực hiện xong lệnh.

Khối ALU:

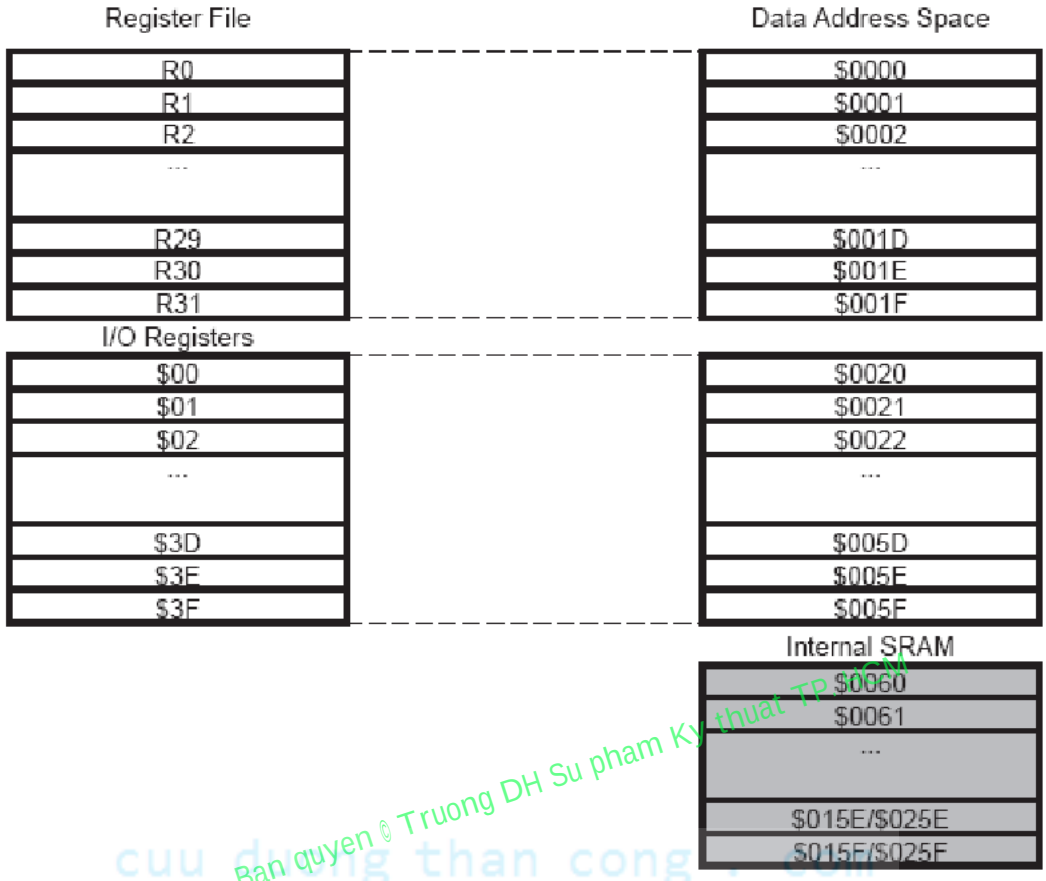
Khối ALU trong AVR thực hiện kết nối trực tiếp với tất cả 32 thanh ghi hoạt động đa năng tổng quát. Trong một chu kỳ duy nhất của xung clock, các phép toán của khối ALU xảy ra giữa các thanh ghi trong file thanh ghi được thực hiện. Các phép toán của khối ALU được chia ra làm 3 loại: phép toán số học, phép toán logic và các phép toán xử lý bit.

Bộ nhớ chương trình flash có thể lập trình trong hệ thống:

AT90S8535 có 8K byte bộ nhớ chương trình flash có thể lập trình trong hệ thống để lưu trữ chương trình. Do tất cả các lệnh đều có chiều dài là 16 bit nên bộ nhớ FLASH tổ chức theo 4K x 16. Bộ nhớ flash có thể đảm bảo được cho 1000 chu kỳ nạp xóa. Thanh ghi PC của AT90S8535 có chiều dài 12 bit do đó có thể truy xuất 4096 địa chỉ của bộ nhớ chương trình.

Bộ nhớ dữ liệu SRAM:

Hình 8 sẽ trình bày cách thức tổ chức của bộ nhớ SRAM của AT90S8535:



Hình 8. Trình bày cấu trúc bộ nhớ SRAM.

Vùng nhớ dữ liệu thấp có 608 địa chỉ dùng để định chỉ cho: file thanh ghi, cho bộ nhớ IO và cho bộ nhớ SRAM. 96 ô nhớ đầu tiên là địa chỉ của file thanh ghi và của vùng nhớ IO. 512 ô nhớ tiếp theo là địa chỉ của vùng nhớ dữ liệu SRAM bên trong.

Năm kiểu định địa chỉ khác nhau cho vùng nhớ dữ liệu bao gồm: định địa trực tiếp (direct), định địa chỉ gián tiếp (indirect with displacement) và gián tiếp với tăng địa chỉ. Trong file thanh ghi, thanh ghi từ R26 đến R31 tổ chức thành thanh ghi con trỏ địa chỉ có thể định địa chỉ gián tiếp.

Địa chỉ hóa trực tiếp toàn bộ vùng dữ liệu.

Trong địa chỉ hóa gián tiếp với cấu trúc kiểu thì 63 ô nhớ có thể truy xuất từ địa chỉ nền được cho bởi thanh ghi Y hoặc thanh ghi Z.

Khi dùng các kiểu định địa chỉ gián tiếp dùng thanh ghi với địa chỉ tự động tăng hoặc giảm, thanh ghi địa chỉ X, Y và Z tăng hoặc giảm.

32 thanh ghi hoạt động đa năng tổng quát, 64 thanh ghi IO và 512 byte của bộ nhớ dữ liệu SRAM bên trong AT90S8535 có thể được truy xuất thông qua tất cả các kiểu định địa chỉ này.

Hãy xem phần tiếp theo sẽ trình bày chi tiết các kiểu định địa chỉ khác nhau.

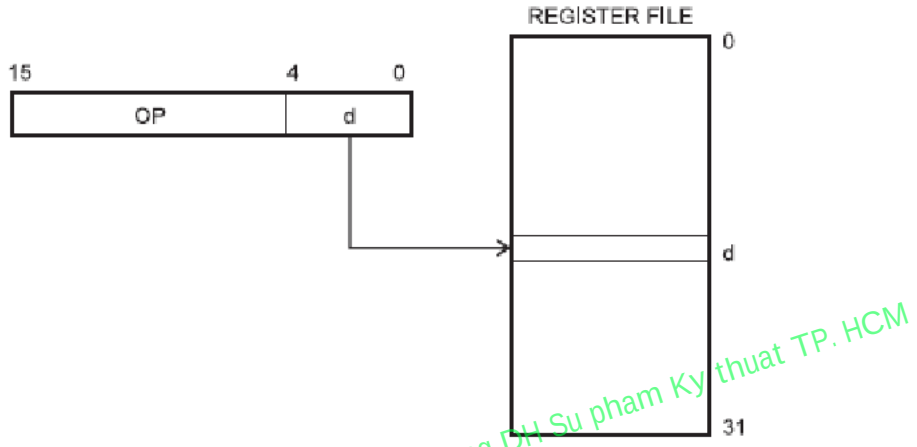
Các kiểu truy xuất bộ nhớ dữ liệu và bộ nhớ chương trình:

Vi điều khiển AVR AT90S8535 cung cấp nhiều kiểu định địa chỉ mạnh và hiệu quả để truy xuất bộ nhớ chương trình (flash) và bộ nhớ dữ liệu (SRAM, file thanh ghi và vùng nhớ IO). Vai trò của nhiều kiểu định địa chỉ khác nhau được cung cấp bởi cấu trúc của AVR. Trong các hình kí hiệu OP

(operation code) có nghĩa là phần mã tác tố của từ lệnh. Để đơn giản, không phải hình nào cũng trình bày vị trí chính xác của các bit định địa chỉ.

a. Kiểu định địa chỉ trực tiếp, thanh ghi đơn Rd:

Kiểu định địa chỉ trực tiếp dùng thanh ghi và 2 thanh ghi như hình 9.

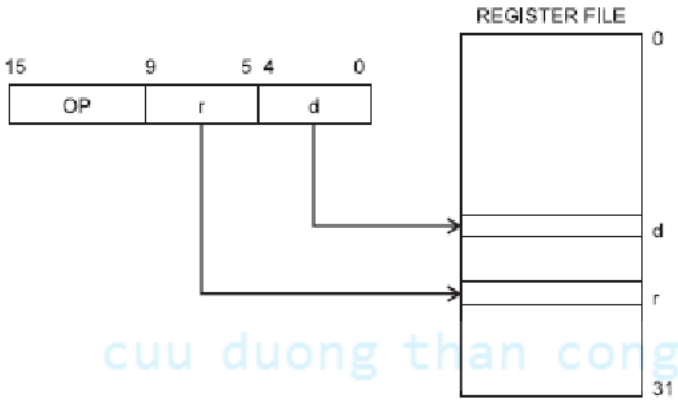


Hình 9. Kiểu định địa chỉ trực tiếp dùng thanh ghi và 2 thanh ghi.

Tác tố được chứa trong thanh ghi d (Rd).

b. Kiểu định địa chỉ trực tiếp, dùng 2 thanh ghi Rd và Rr:

Kiểu định địa chỉ trực tiếp dùng thanh ghi và 2 thanh ghi như hình 10.

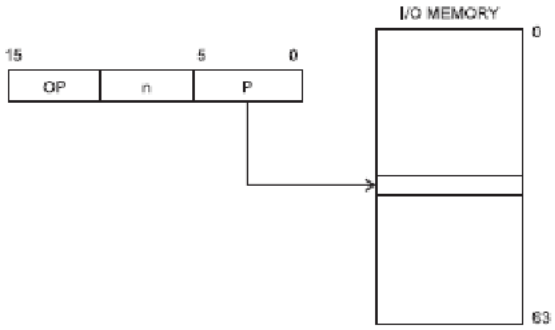


Hình 10. Kiểu định địa chỉ trực tiếp dùng thanh ghi và 2 thanh ghi.

Các tác tố được chứa trong thanh ghi r (Rr) và d (Rd). Kết quả được lưu trong thanh ghi d (Rd).

c. Kiểu định địa chỉ trực tiếp IO:

Kiểu định địa chỉ trực tiếp IO như hình 11.

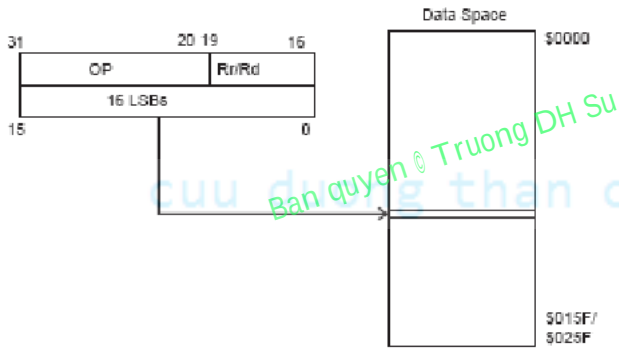


Hình 11. Kiểu định địa chỉ trực tiếp IO.

Địa chỉ của tác tố được chứa trong 6 bit của từ mã lệnh: n là địa chỉ của thanh ghi đến (destination) hoặc thanh ghi nguồn (suorce).

d. Kiểu định địa chỉ trực tiếp dữ liệu:

Kiểu định địa chỉ trực tiếp dữ liệu như hình 12.

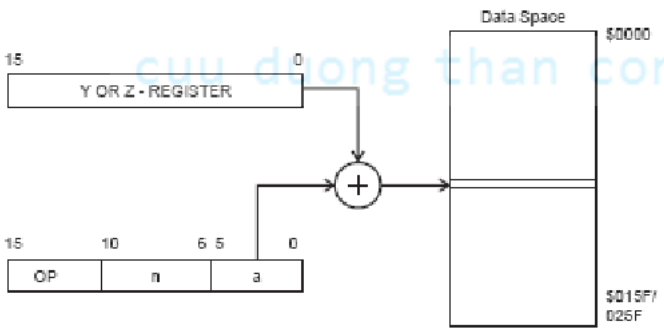


Hình 12. Kiểu định địa chỉ trực tiếp dữ liệu.

Địa chỉ 16 bit của dữ liệu trong từ mã lệnh thứ 2 của từ mã lệnh 2 word. Rd/Rr chỉ định thanh ghi đến hoặc thanh ghi nguồn.

e. Kiểu định địa chỉ gián tiếp dữ liệu với displacement:

Kiểu định địa chỉ gián tiếp dữ liệu với displacement như hình 13.

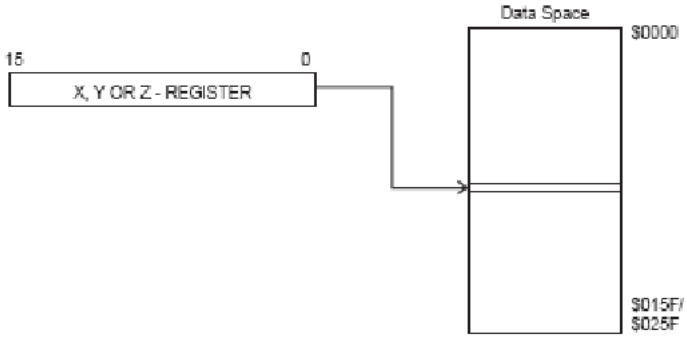


Hình 13. Kiểu định địa chỉ gián tiếp dữ liệu với displacement.

Địa chỉ của tác tố là kết quả của nội dung lưu trong thanh ghi Y hoặc Z cộng với địa chỉ 6 bit a có trong từ mã lệnh.

f. Kiểu định địa chỉ gián tiếp dữ liệu:

Kiểu định địa chỉ gián tiếp dữ liệu như hình 14.

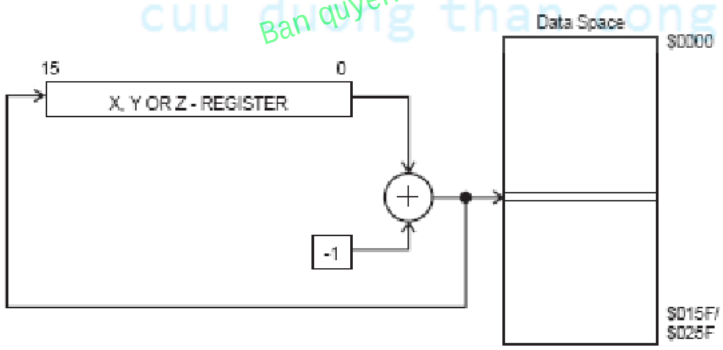


Hình 14. Kiểu định địa chỉ gián tiếp dữ liệu.

Địa chỉ của tác tố lưu trong thanh ghi X hoặc Y hoặc Z.

g. Kiểu định địa chỉ gián tiếp dữ liệu với pre - displacement:

Kiểu định địa chỉ gián tiếp dữ liệu với pre – displacement như hình 15.



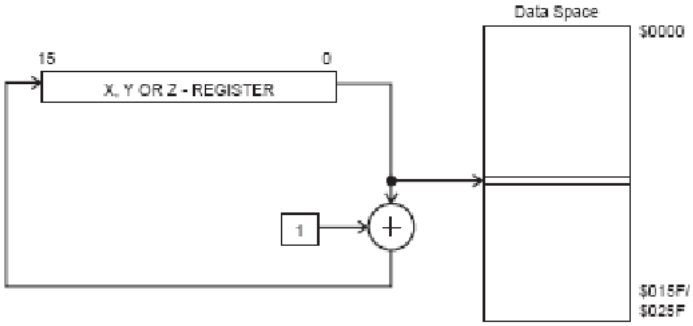
Hình 15. Kiểu định địa chỉ gián tiếp dữ liệu với pre - displacement.

Nội dung của thanh ghi X, Y hoặc Z giảm đi 1 trước khi thực hiện lệnh. Địa chỉ của tác tố lưu trong thanh ghi X, Y hoặc Z sau khi đã giảm đi 1.

Nhìn vào hình 15 cho chúng ta thấy được nội dung của X, Y hoặc Z cộng với số -1 và tạo ra địa chỉ mới được cập nhật trở lại thanh ghi X, Y hoặc Z và đó cũng chính là địa chỉ của ô nhớ cần truy xuất dữ liệu.

h. Kiểu định địa chỉ gián tiếp dữ liệu với Post - Increment:

Kiểu định địa chỉ gián tiếp dữ liệu với Post – Increment như hình 16.



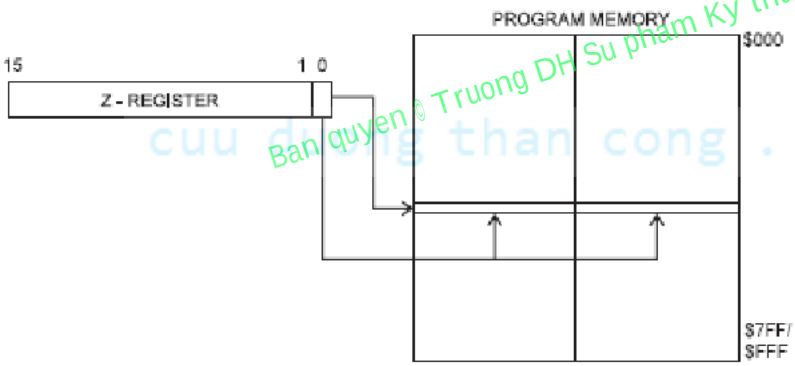
Hình 16. Kiểu định địa chỉ gián tiếp dữ liệu với Post - Increment.

Địa chỉ của tác tố lưu trong thanh ghi X, Y hoặc Z . Sau khi thực hiện xong việc truy xuất dữ liệu thì nội dung thanh ghi X, Y hoặc Z tăng lên 1.

Nhìn vào hình 16 cho chúng ta thấy được nội dung của X, Y hoặc Z là địa chỉ cần try xuất dữ liệu sau đó nội dung thanh ghi X, Y hoặc Z tăng lên 1 và cập nhật trở lại thanh ghi X, Y hoặc Z .

i. Kiểu định địa chỉ dùng lệnh LPM:

Kiểu định địa chỉ dùng lệnh LPM như hình 17.

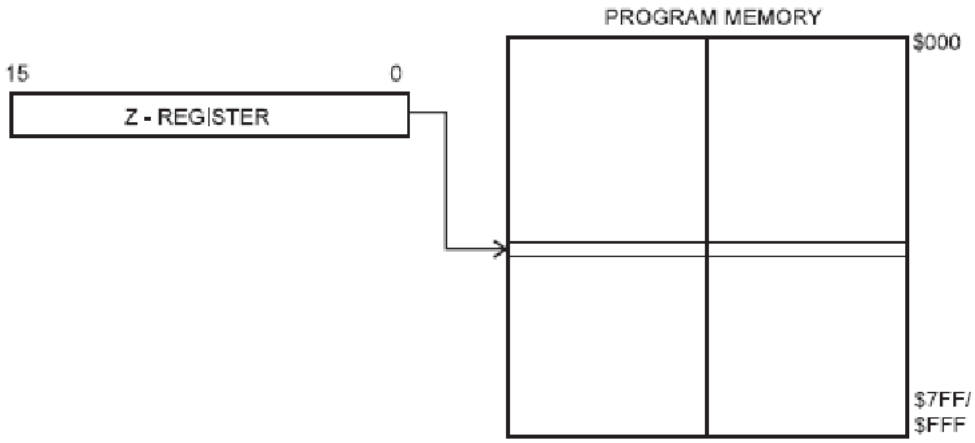


Hình 17. Kiểu định địa chỉ dùng lệnh LPM.

Địa chỉ của byte dữ liệu được chỉ định bởi nội dung thanh ghi Z. 15 bit cao lựa chọn địa chỉ (trong phạm vi 0 – 2K/4K), bit thấp nhất trong thanh ghi Z nếu bằng 0 thì lệnh sẽ truy xuất byte dữ liệu thấp, nếu bằng 1 thì sẽ truy xuất byte cao trong vùng nhớ 2 byte.

j. Kiểu định địa chỉ gián tiếp bộ nhớ chương trình, IJMP và ICALL:

Kiểu định địa chỉ gián tiếp bộ nhớ chương trình như hình 18.

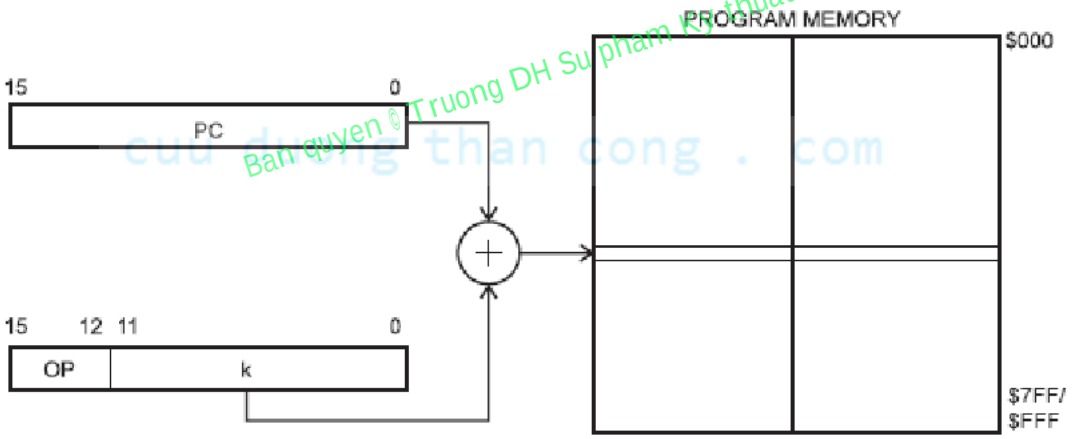


Hình 18. Kiểu định địa chỉ gián tiếp bộ nhớ chương trình.

Việc thực hiện chương trình tiếp tục tại địa chỉ lưu trong thanh ghi Z.

k. Kiểu định địa chỉ tương đối bộ nhớ chương trình, RJMP và RCALL:

Kiểu định địa chỉ gián tiếp bộ nhớ chương trình như hình 19.



Hình 19. Kiểu định địa chỉ gián tiếp bộ nhớ chương trình.

Việc thực hiện chương trình tiếp tục tại địa chỉ lưu mới bằng PC + k + 1. Địa chỉ tương đối k nằm trong phạm vi từ -2048 đến 2047.

Bộ nhớ IO:

Vùng địa chỉ không gian bộ nhớ IO của At90S8535 được trình bày ở bảng 1:

I/O Address (SRAM Address)	Name	Function
$3F ($5F)	SREG	Status REGister
$3E ($5E)	SPH	Stack Pointer High
$3D ($5D)	SPL	Stack Pointer Low
$3B ($5B)	GIMSK	General Interrupt MaSK register
$3A ($5A)	GIFR	General Interrupt Flag Register
$39 ($59)	TIMSK	Timer/Counter Interrupt MaSK register
$38 ($58)	TIFR	Timer/Counter Interrupt Flag register
$35 ($55)	MCUCR	MCU general Control Register
$34 ($45)	MCUSR	MCU general Status Register
$33 ($53)	TCCR0	Timer/Counter0 Control Register
$32 ($52)	TCNT0	Timer/Counter0 (8-bit)
$2F ($4F)	TCCR1A	Timer/Counter1 Control Register A
$2E ($4E)	TCCR1B	Timer/Counter1 Control Register B
$2D ($4D)	TCNT1H	Timer/Counter1 High Byte
$2C ($4C)	TCNT1L	Timer/Counter1 Low Byte
$2B ($4B)	OCR1AH	Timer/Counter1 Output Compare Register A High Byte
$2A ($4A)	OCR1AL	Timer/Counter1 Output Compare Register A Low Byte
$29 ($49)	OCR1BH	Timer/Counter1 Output Compare Register B High Byte
$28 ($48)	OCR1BL	Timer/Counter1 Output Compare Register B Low Byte
$27 ($47)	ICR1H	T/C 1 Input Capture Register High Byte
$26 ($46)	ICR1L	T/C 1 Input Capture Register Low Byte
$25 ($45)	TCCR2	Timer/Counter2 Control Register

I/O Address (SRAM Address)	Name	Function
$24 ($44)	TCNT2	Timer/Counter2 (8-bit)
$23 ($43)	OCR2	Timer/Counter2 Output Compare Register
$22 ($42)	ASSR	Asynchronous Mode Status Register
$21 ($41)	WDTCR	Watchdog Timer Control Register
$1F ($3E)	EEARH	EEPROM Address Register High Byte
$1E ($3E)	EEARL	EEPROM Address Register Low Byte
$1D ($3D)	EEDR	EEPROM Data Register
$1C ($3C)	EECR	EEPROM Control Register
$1B ($3B)	PORTA	Data Register, Port A
$1A ($3A)	DDRA	Data Direction Register, Port A
$19 ($39)	PINA	Input Pins, Port A
$18 ($38)	PORTB	Data Register, Port B
$17 ($37)	DDRB	Data Direction Register, Port B
$16 ($36)	PINB	Input Pins, Port B
$15 ($35)	PORTC	Data Register, Port C
$14 ($34)	DDRC	Data Direction Register, Port C
$13 ($33)	PINC	Input Pins, Port C
$12 ($32)	PORTD	Data Register, Port D
$11 ($31)	DDRD	Data Direction Register, Port D
$10 ($30)	PIND	Input Pins, Port D
$0F ($2F)	SPDR	SPI I/O Data Register
$0E ($2E)	SPSR	SPI Status Register
$0D ($2D)	SPCR	SPI Control Register
$0C ($2C)	UDR	UART I/O Data Register
$0B ($2B)	USR	UART Status Register
$0A ($2A)	UCR	UART Control Register
$09 ($29)	UBRR	UART Baud Rate Register
$08 ($28)	ACSR	Analog Comparator Control and Status Register
$07 ($27)	ADMUX	ADC Multiplexer Select Register
$06 ($26)	ADCSR	ADC Control and Status Register
$05 ($25)	ADCH	ADC Data Register High
$04 ($24)	ADCL	ADC Data Register Low

Bảng 1. Bộ nhớ IO.

Tất cả IO và các ngoại vi của AT90S8535 được đặt trong vùng nhớ IO. Các ô nhớ IO được truy xuất bởi lệnh IN và OUT để truyền dữ liệu giữa 32 thanh ghi hoạt động đa chức năng và vùng nhớ IO. Các thanh ghi IO nằm trong vùng địa chỉ từ $00 đến $1F cho phép truy xuất bit dùng các lệnh SBI và CBI. Trong các thanh ghi này, giá trị của các bit đơn có thể được kiểm tra bằng cách dùng các lệnh SBIS và SBIC. Hãy tham khảo tập lệnh để có thêm chi tiết.

Khi sử dụng các lệnh IN, OUT để định địa các IO thì các địa chỉ truy xuất các IO nằm trong khoảng từ $00 đến $3F. Khi xem các thanh ghi IO như là một phần của bộ nhớ SRAM thì địa chỉ của chúng phải được cộng thêm $20 (hãy xem trong bảng đồ nhớ SRAM). Tất cả địa chỉ của thanh ghi IO được trình bày trong suốt tài liệu này được trình bày với địa chỉ SRAM nằm trong dấu ngoặc đơn.

Để tương thích với các thiết bị sẽ được xây dựng trong tương lai thì các bit chưa sử dụng sẽ được ghi số 0 nếu chúng ta truy xuất. Các địa chỉ của vùng nhớ IO chưa sử dụng sẽ không bao giờ được ghi dữ liệu.

Nhiều cờ trạng thái được xóa bằng cách ghi logic đến chúng (a logical one to them). Chú ý rằng các lệnh CBI và SBI sẽ hoạt động trên tất cả các bit trong thanh ghi IO. Các lệnh CBI và SBI sẽ hoạt động với các thanh ghi chỉ nằm trong khoảng từ $00 đến $1F.

Các thanh ghi IO và các thanh ghi điều khiển ngoại vi được giải thích ở phần tiếp theo sau.

Thanh ghi trạng thái – status register - SREG:

Thanh ghi trạng thái SREG của AVR có địa chỉ trong vùng nhớ IO là $3F ($5F) được xác định như sau:

Bit	7	6	5	4	3	2	1	0	
$3F ($5F)	I	T	H	S	V	N	Z	C	SREG
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial value	0	0	0	0	0	0	0	0	

Chức năng của các bit:

Bit 7: Global Interrupt Enable - bit I: bit cho phép ngắt toàn cục:

Bit I phải được thiết lập ở mức logic 1 để cho phép ngắt.

Sau đó từng bit điều khiển ngắt độc lập được thực hiện trong thanh ghi điều khiển riêng. Nếu Bit I ở mức 0 thì không cho bất kỳ ngắt nào xảy ra cho dù từng bit điều khiển ngắt ở trạng thái cho phép.

Bit I sẽ bị xóa bởi phần cứng sau khi ngắt xảy ra và sẽ trở lại mức 1 để cho phép ngắt sau khi thực hiện lệnh trở về từ chương trình con phục vụ ngắt RETI

Bit 6: Bit Copy Storage - bit T: bit copy và lưu trữ :

Các lệnh copy bit BLD (Bit Load) và BST (Bit Store) dùng bit T như là bit source và bit destination cho các hoạt động bit. Một bit từ 1 thanh ghi trong file thanh ghi có thể copy vào bit T bằng lệnh BST và bit T có thể được copy vào một bit trong thanh ghi nằm trong file thanh ghi bằng lệnh BLD.

Bit 5: Half Carry flag - bit H: bit cờ tràn phụ:

Bit cờ tràn phụ lưu trạng thái tràn phụ trong 1 số các phép toán. Hãy xem chi tiết ở phần lệnh.

Bit 4: Sign bit - bit S – S = N (+) V: bit dấu :

Bit dấu S thường là kết quả của phép toán ex-or giữa bit N (bit Negative) và bit V (over flow). Hãy xem chi tiết ở phần lệnh.

Bit 3: Bit Two's Complement Overflow Flag – bit V:

Cờ tràn bù 2 V được xây dựng để thực hiện các phép toán bù hai.

Bit 2: Bit Negative Flag – bit N:

Cờ số âm N xác định kết qủa phép toán là số âm.

Bit 1: Zero Flag – bit Z:

Cờ zero xác định kết qủa phép toán bằng 0 hay khác 0.

Bit 0: Carry flag – bit C:

Cờ tràn xác định kết qủa phép toán có bị tràn hay không.

Chú ý: thanh ghi trạng thái sẽ không tự động lưu trữ khi thực hiện chương trình con phục vụ ngắt và sẽ không khôi phục lại khi trở về chương trình chính. Chúng ta phải tự lưu trữ bằng phần mềm nếu cần.

Thanh ghi con trỏ ngăn xếp – stack pointer register - SP:

Thanh ghi con trỏ ngăn xếp của AT90S8535 được thiết kế như là 2 thanh ghi 8 bit nằm tại địa chỉ $3E ($5E) và $3D ($5D) trong vùng nhớ IO. Do vùng nhớ dữ liệu chỉ có $25F ô nhớ nên thanh ghi SP chỉ có 10 bit được sử dụng.

Cấu trúc của thanh ghi Sp như hình sau:

Bit	15	14	13	12	11	10	9	8	
$3E ($5E)	-	-	-	-	-	-	SP9	SP8	SPH
$3D ($5D)	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	SPL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

Thanh ghi con trỏ quản lý vùng nhớ dữ liệu ngăn xếp của bộ nhớ SRAM, tại vùng nhớ ngăn xếp dùng để lưu các dữ liệu của chương trình con và chương trình con phục vụ ngắt. Vùng nhớ ngăn xếp trong vùng nhớ SRAM phải được xác định bởi chương trình trước khi có bất kỳ chương trình con nào được thực hiện hoặc các ngắt được phép.

Con trỏ ngăn xếp phải được thiết lập tại địa chỉ trên $60.

Con trỏ ngăn xếp giảm đi 1 khi dữ liệu được cất vào bộ nhớ ngăn xếp bởi lệnh PUSH, con trỏ ngăn xếp giảm đi 2 khi địa chỉ được cất vào ngăn xếp khi thực hiện lệnh gọi chương trình con hoặc khi chương trình con phục vụ ngắt được thực hiện.

Con trỏ ngăn xếp tăng lên 1 khi dữ liệu được lấy ra từ ngăn xếp bằng lệnh POP, và con trỏ sẽ tăng lên 2 khi địa chỉ được lấy ra khỏi ngăn xếp khi thực hiện lệnh kết thúc chương trình con RET trở về chương trình chính hoặc lệnh kết thúc chương trình con phục vụ ngắt RETI để trở về chương trình chính.

Điều khiển RESET và ngắt:

AT90S8535 cung cấp 16 nguồn tín hiệu ngắt khác nhau. Các ngắt này và các vector ngắt đều có 1 vector chương trình riêng trong vùng nhớ chương trình. Tất cả các ngắt được gán với các bit cho phép ngắt độc lập – bit cho phép ngắt phải được thiết lập ở mức 1 cùng với bit I nằm trong thanh ghi trạng thái để cho phép ngắt xảy ra.

Các địa chỉ thấp trong vùng nhớ chương trình được xác định khi reset và các vector ngắt hoàn toàn tự động. Danh sách liệt kê đầy đủ các vector được trình bày ở bảng 2. Danh sách này cũng xác định các mức độ ưu tiên của nhiều ngắt khác nhau. Địa chỉ càng thấp thì mức độ ưu tiên ngắt càng cao. RESET có mức độ ưu tiên ngắt cao nhất, tiếp theo là INTO – External interrupt request 0.

Vector No.	Program Address	Source	Interrupt Definition
1	$000	RESET	Hardware Pin, Power-On Reset and Watchdog Reset
2	$001	INT0	External Interrupt Request 0
3	$002	INT1	External Interrupt Request 1
4	$003	TIMER2 COMP	Timer/Counter2 Compare Match
5	$004	TIMER2 OVF	Timer/Counter2 Overflow
6	$005	TIMER1 CAPT	Timer/Counter1 Capture Event
7	$006	TIMER1 COMPA	Timer/Counter1 Compare Match A
8	$007	TIMER1 COMPB	Timer/Counter1 Compare Match B
9	$008	TIMER1 OVF	Timer/Counter1 Overflow
10	$009	TIMER0 OVF	Timer/Counter0 Overflow
11	$00A	SPI, STC	SPI Serial Transfer Complete
12	$00B	UART, RX	UART, Rx Complete
13	$00C	UART, UDRE	UART Data Register Empty
14	$00D	UART, TX	UART, Tx Complete
15	$00E	ADC	ADC Conversion Complete
16	$00F	EE_RDY	EEPROM Ready
17	$010	ANA_COMP	Analog Comparator

Bảng 2. Reset và bảng vector ngắt.

Từ bảng vector ngắt nên một chương trình thường được bắt đầu như sau để tránh các vùng địa chỉ ngắt như chương trình sau:

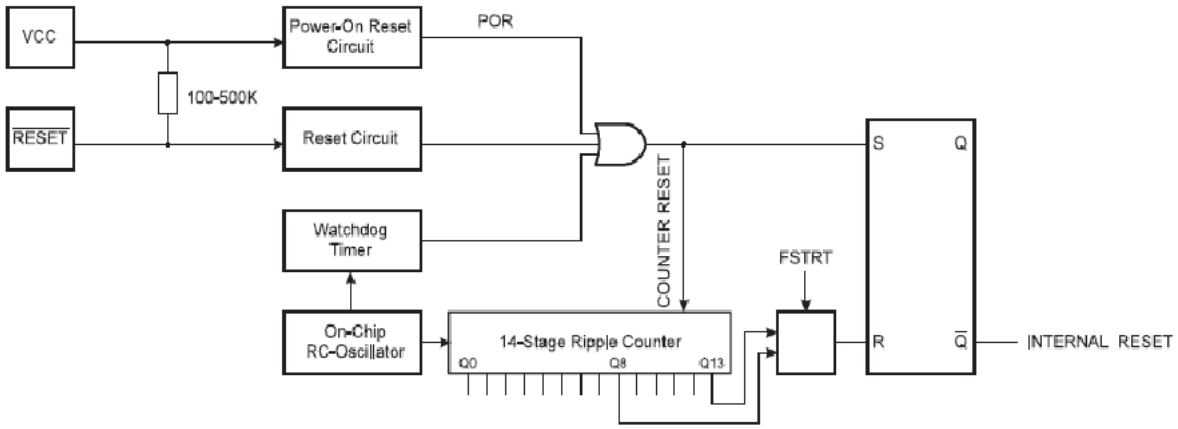
```
Address          Labels     Code            Comments
$000                        rjmp    RESET       ; Reset Handler
$001                        rjmp    EXT_INT0    ; IRQ0 Handler
$002                        rjmp    EXT_INT1    ; IRQ1 Handler
$003                        rjmp    TIM2_COMP   ; Timer2 Compare Handler
$004                        rjmp    TIM2_OVP    ; Timer2 Overflow Handler
$005                        rjmp    TIM1_CAPT   ; Timer1 Capture Handler
$006                        rjmp    TIM1_COMPA  ; Timer1 CompareA Handler
$007                        rjmp    TIM1_COMPB  ; Timer1 CompareB Handler
$008                        rjmp    TIM1_OVP    ; Timer1 Overflow Handler
$009                        rjmp    TIM0_OVP    ; Timer0 Overflow Handler
$00a                        rjmp    SPI_STC;    ; SPI Transfer Complete Handler
$00b                        rjmp    UART_RXC    ; UART RX Complete Handler
$00c                        rjmp    UART_DRE    ; UDR Empty Handler
$00d                        rjmp    UART_TXC    ; UART TX Complete Handler
$00e                        rjmp    ADC         ; ADC Conversion Complete Interrupt Handler
$00f                        rjmp    EE_RDY      ; EEPROM Ready Handler
$010                        rjmp    ANA_COMP    ; Analog Comparator Handler
$011             MAIN:      ldi     r16, high(RAMEND); Main program start
$012                        out     SPH,r16
$013                        ldi     r16, low(RAMEND);
$014                        out     SPL,r16
$015                    <instr> xxx
...              ...        ...     ...
```


Nguồn RESET:

AT90S8535 có 3 nguồn reset:

- Reset khi cấp điện: MCU sẽ bị reset khi có điện áp cung cấp thấp hơn điện áp ngưỡng reset khi mở điện (power – on reset threshold Vpot).
- Ngắt ngoài: MCU bị reset khi có mức thấp được xuất hiện ở ngõ vào chân RESET\ kéo dài hơn 50ns.
- Watchdog reset: MCU bị reset khi chu kì thời gian watchdog hết hiệu lực và watchdog được phép.

Trong quá trình reset tất cả các thanh ghi IO được khởi tạo các giá trị bắt đầu và chương trình được bắt đầu tại địa chỉ $000. Lệnh đặt tại địa chỉ $000 phải là lệnh RJMP – lệnh nhảy tương đối – để nhảy đến một nơi khác tránh vùng nhớ của các vector ngắt. Nếu chương trình không bao giờ sử dụng ngắt, các vector ngắt không được sử dụng thì mã lệnh của chương trình có thể viết bắt đầu tại địa chỉ $000 mà không cần phải nhảy. Mạch điện ở hình 20 trình bày mạch reset. Bảng 3 xác định thời gian và các thông số điện của mạch điện reset.



Hình 20. Mạch điện reset.

Symbol	Parameter	Min	Typ	Max	Units
V_{POT} (1)	Power-On Reset Threshold (rising)	1.0	1.4	1.8	V
	Power-On Reset Threshold (falling)	0.4	0.6	0.8	V
V_{RST}	$\overline{RESET}$ Pin Threshold Voltage		$0.6V_{CC}$		V
t_{TOUT}	Reset Delay Time-Out Period FSTRT Unprogrammed	11	16	21	ms
t_{TOUT}	Reset Delay Time-Out Period FSTRT Programmed	1.0	1.1	1.2	ms

Bảng 3. Các thông số reset với nguồn Vcc = 5V.

Chú ý: reset khi cấp nguồn sẽ không hoạt động trừ khi nguồn cung cấp có điện áp dưới Vpot.

FSTRT	Time-out at V_{CC} = 5V	Number of WDT cycles
Programmed	1.1ms	1K
Unprogrammed	16.0ms	16K

Bảng 4. Số chu kỳ dao động của Watchdog timer.

5. Tập lệnh của AVR:

Thuật ngữ của tập lệnh:

SREG: status register – thanh ghi trạng thái.

C: Carry flag in status register
Z: Zero flag in status register
N: Negative flag in status register
V: Two's complement overflow indicator
S: N (+) V for signed test
H: Half carry flag in status register
T: Transfer bit used by BLD and BSt instructions.
I: Global interrupt enable/ disable flag.

Thanh ghi và tác tố

Rd: Destination (and source) register trong file thanh ghi.
Rr: Source register trong file thanh ghi.

R:	Result after instruction is executed – kết quả sau khi lệnh thực hiện xong
K:	constant data – thông số dữ liệu
k:	constant address – thông số địa chỉ
b:	Bit trong file thanh ghi hoặc thanh ghi IO (3bit)
s:	bit trong thanh ghi trạng thái (3 bit)
X, Y, Z:	thanh ghi địa chỉ gián tiếp. (X = R26:R27, Y = R29:R28, Z = R31:R30).
A:	IO location address – địa chỉ của IO
q:	displacement for direct addressing (6 bit)

IO REGISTER:

RAMPX, RAMPY, RAMPZ

Các thanh ghi được kết nối với X, Y và Z cho phép định địa chỉ gián tiếp của toàn bộ không gian bộ nhớ dữ liệu trong MCU với dung lượng bộ nhớ hơn 64k byte và đón dữ liệu trong MCU với dung lượng bộ nhớ chương trình hơn 64K byte.

RAMPD

Thanh ghi được kết nối với thanh ghi Z cho phép định địa chỉ trực tiếp toàn bộ không gian bộ nhớ dữ liệu trong MCU với dung lượng lớn hơn 64K byte.

EIND

Thanh ghi được kết nối với từ mã lệnh cho phép nhảy gián tiếp và gọi bất kỳ nơi nào trong bộ nhớ chương trình trên MCU với dung lượng không gian bộ nhớ lớn hơn 64K byte.

STACK

Stack:	dùng để lưu các địa chỉ trở về và nội dung các thanh ghi cất tạm thời.
SP:	con trỏ quản lý bộ nhớ ngăn xếp.

FLAG

Kí hiệu cho biết sự ảnh hưởng của lệnh đến các bit trạng thái trong thanh ghi trạng thái.

Flags

⇔:	Flag affected by instruction
0:	Flag cleared by instruction
1:	Flag set by instruction
-:	Flag not affected by instruction

Tập lệnh được tóm tắt ở bảng 5:

Mnemonics	Operands	Description	Operation	Flags	#Clock Note
Arithmetic and Logic Instructions					
ADD	Rd, Rr	Add without Carry	Rd ← Rd + Rr	Z,C,N,V,S,H	1
ADC	Rd, Rr	Add with Carry	Rd ← Rd + Rr + C	Z,C,N,V,S,H	1
ADIW	Rd, K	Add Immediate to Word	Rd+1:Rd ← Rd+1:Rd + K	Z,C,N,V,S	2
SUB	Rd, Rr	Subtract without Carry	Rd ← Rd - Rr	Z,C,N,V,S,H	1
SUBI	Rd, K	Subtract Immediate	Rd ← Rd - K	Z,C,N,V,S,H	1
SBC	Rd, Rr	Subtract with Carry	Rd ← Rd - Rr - C	Z,C,N,V,S,H	1
SBCI	Rd, K	Subtract Immediate with Carry	Rd ← Rd - K - C	Z,C,N,V,S,H	1
SBIW	Rd, K	Subtract Immediate from Word	Rd+1:Rd ← Rd+1:Rd - K	Z,C,N,V,S	2
AND	Rd, Rr	Logical AND	Rd ← Rd • Rr	Z,N,V,S	1
ANDI	Rd, K	Logical AND with Immediate	Rd ← Rd • K	Z,N,V,S	1
OR	Rd, Rr	Logical OR	Rd ← Rd v Rr	Z,N,V,S	1
ORI	Rd, K	Logical OR with Immediate	Rd ← Rd v K	Z,N,V,S	1
EOR	Rd, Rr	Exclusive OR	Rd ← Rd ⊕ Rr	Z,N,V,S	1
COM	Rd	One's Complement	Rd ← $FF - Rd	Z,C,N,V,S	1
NEG	Rd	Two's Complement	Rd ← $00 - Rd	Z,C,N,V,S,H	1
SBR	Rd,K	Set Bit(s) in Register	Rd ← Rd v K	Z,N,V,S	1
CBR	Rd,K	Clear Bit(s) in Register	Rd ← Rd • ($FFh - K)	Z,N,V,S	1
INC	Rd	Increment	Rd ← Rd + 1	Z,N,V,S	1
DEC	Rd	Decrement	Rd ← Rd - 1	Z,N,V,S	1
TST	Rd	Test for Zero or Minus	Rd ← Rd • Rd	Z,N,V,S	1
CLR	Rd	Clear Register	Rd ← Rd ⊕ Rd	Z,N,V,S	1
SER	Rd	Set Register	Rd ← $FF	None	1
MUL	Rd,Rr	Multiply Unsigned	R1:R0 ← Rd × Rr (UU)	Z,C	2
MULS	Rd,Rr	Multiply Signed	R1:R0 ← Rd × Rr (SS)	Z,C	2
MULSU	Rd,Rr	Multiply Signed with Unsigned	R1:R0 ← Rd × Rr (SU)	Z,C	2
FMUL	Rd,Rr	Fractional Multiply Unsigned	R1:R0 ← (Rd × Rr)<<1 (UU)	Z,C	2
FMULS	Rd,Rr	Fractional Multiply Signed	R1:R0 ← (Rd × Rr)<<1 (SS)	Z,C	2
FMULSU	Rd,Rr	Fractional Multiply Signed with Unsigned	R1:R0 ← (Rd × Rr)<<1 (SU)	Z,C	2

Bảng 5. Tóm tắt tập lệnh.

Mnemonics	Operands	Description	Operation	Flags	#Clock Note
Branch Instructions					
RJMP	k	Relative Jump	PC ← PC + k + 1	None	2
IJMP		Indirect Jump to (Z)	PC(15:0) ← Z, PC(21:16) ← 0	None	2
EIJMP		Extended Indirect Jump to (Z)	PC(15:0) ← Z, PC(21:16) ← EIND	None	2
JMP	k	Jump	PC ← k	None	3
RCALL	k	Relative Call Subroutine	PC ← PC + k + 1	None	3 / 4
ICALL		Indirect Call to (Z)	PC(15:0) ← Z, PC(21:16) ← 0	None	3 / 4
EICALL		Extended Indirect Call to (Z)	PC(15:0) ← Z, PC(21:16) ← EIND	None	4
CALL	k	Call Subroutine	PC ← k	None	4 / 5
RET		Subroutine Return	PC ← STACK	None	4 / 5
RETI		Interrupt Return	PC ← STACK	I	4 / 5
CPSE	Rd,Rr	Compare, Skip If Equal	If (Rd = Rr) PC ← PC + 2 or 3	None	1 / 2 / 3
CP	Rd,Rr	Compare	Rd - Rr	Z,C,N,V,S,H	1
CPC	Rd,Rr	Compare with Carry	Rd - Rr - C	Z,C,N,V,S,H	1
CPI	Rd,K	Compare with Immediate	Rd - K	Z,C,N,V,S,H	1
SBRC	Rr, b	Skip if Bit in Register Cleared	If (Rr(b)=0) PC ← PC + 2 or 3	None	1 / 2 / 3
SBRS	Rr, b	Skip if Bit in Register Set	If (Rr(b)=1) PC ← PC + 2 or 3	None	1 / 2 / 3
SBIC	A, b	Skip if Bit in I/O Register Cleared	If(I/O(A,b)=0) PC ← PC + 2 or 3	None	1 / 2 / 3
SBIS	A, b	Skip if Bit in I/O Register Set	If(I/O(A,b)=1) PC ← PC + 2 or 3	None	1 / 2 / 3
BRBS	s, k	Branch if Status Flag Set	If (SREG(s) = 1) then PC←PC+k + 1	None	1 / 2
BRBC	s, k	Branch if Status Flag Cleared	If (SREG(s) = 0) then PC←PC+k + 1	None	1 / 2
BREQ	k	Branch if Equal	If (Z = 1) then PC ← PC + k + 1	None	1 / 2
BRNE	k	Branch if Not Equal	If (Z = 0) then PC ← PC + k + 1	None	1 / 2
BRCS	k	Branch if Carry Set	If (C = 1) then PC ← PC + k + 1	None	1 / 2
BRCC	k	Branch if Carry Cleared	If (C = 0) then PC ← PC + k + 1	None	1 / 2
BRSH	k	Branch if Same or Higher	If (C = 0) then PC ← PC + k + 1	None	1 / 2
BRLO	k	Branch if Lower	If (C = 1) then PC ← PC + k + 1	None	1 / 2
BRMI	k	Branch if Minus	If (N = 1) then PC ← PC + k + 1	None	1 / 2
BRPL	k	Branch if Plus	If (N = 0) then PC ← PC + k + 1	None	1 / 2
BRGE	k	Branch if Greater or Equal, Signed	If (N ⊕ V= 0) then PC ← PC + k + 1	None	1 / 2
BRLT	k	Branch if Less Than, Signed	If (N ⊕ V= 1) then PC ← PC + k + 1	None	1 / 2
BRHS	k	Branch if Half Carry Flag Set	If (H = 1) then PC ← PC + k + 1	None	1 / 2
BRHC	k	Branch if Half Carry Flag Cleared	If (H = 0) then PC ← PC + k + 1	None	1 / 2
BRTS	k	Branch if T Flag Set	If (T = 1) then PC ← PC + k + 1	None	1 / 2
BRTC	k	Branch if T Flag Cleared	If (T = 0) then PC ← PC + k + 1	None	1 / 2

Bảng 5. Tóm tắt tập lệnh (tiếp theo).

Mnemonics	Operands	Description	Operation	Flags	#Clock Note
BRVS	k	Branch if Overflow Flag is Set	If (V = 1) then PC ← PC + k + 1	None	1 / 2
BRVC	k	Branch if Overflow Flag is Cleared	If (V = 0) then PC ← PC + k + 1	None	1 / 2
BRIE	k	Branch if Interrupt Enabled	If (I = 1) then PC ← PC + k + 1	None	1 / 2
BRID	k	Branch if Interrupt Disabled	If (I = 0) then PC ← PC + k + 1	None	1 / 2
Data Transfer Instructions					
MOV	Rd, Rr	Copy Register	Rd ← Rr	None	1
MOVW	Rd, Rr	Copy Register Pair	Rd+1:Rd ← Rr+1:Rr	None	1
LDI	Rd, K	Load Immediate	Rd ← K	None	1
LDS	Rd, k	Load Direct from data space	Rd ← (k)	None	2
LD	Rd, X	Load Indirect	Rd ← (X)	None	2
LD	Rd, X+	Load Indirect and Post-Increment	Rd ← (X), X ← X + 1	None	2
LD	Rd, -X	Load Indirect and Pre-Decrement	X ← X - 1, Rd ← (X)	None	2
LD	Rd, Y	Load Indirect	Rd ← (Y)	None	2
LD	Rd, Y+	Load Indirect and Post-Increment	Rd ← (Y), Y ← Y + 1	None	2
LD	Rd, -Y	Load Indirect and Pre-Decrement	Y ← Y - 1, Rd ← (Y)	None	2
LDD	Rd,Y+q	Load Indirect with Displacement	Rd ← (Y + q)	None	2
LD	Rd, Z	Load Indirect	Rd ← (Z)	None	2
LD	Rd, Z+	Load Indirect and Post-Increment	Rd ← (Z), Z ← Z+1	None	2
LD	Rd, -Z	Load Indirect and Pre-Decrement	Z ← Z - 1, Rd ← (Z)	None	2
LDD	Rd, Z+q	Load Indirect with Displacement	Rd ← (Z + q)	None	2
STS	k, Rr	Store Direct to data space	Rd ← (k)	None	2
ST	X, Rr	Store Indirect	(X) ← Rr	None	2
ST	X+, Rr	Store Indirect and Post-Increment	(X) ← Rr, X ← X + 1	None	2
ST	-X, Rr	Store Indirect and Pre-Decrement	X ← X - 1, (X) ← Rr	None	2
ST	Y, Rr	Store Indirect	(Y) ← Rr	None	2
ST	Y+, Rr	Store Indirect and Post-Increment	(Y) ← Rr, Y ← Y + 1	None	2
ST	-Y, Rr	Store Indirect and Pre-Decrement	Y ← Y - 1, (Y) ← Rr	None	2
STD	Y+q,Rr	Store Indirect with Displacement	(Y + q) ← Rr	None	2
ST	Z, Rr	Store Indirect	(Z) ← Rr	None	2
ST	Z+, Rr	Store Indirect and Post-Increment	(Z) ← Rr, Z ← Z + 1	None	2

Bảng 5. Tóm tắt tập lệnh (tiếp theo).

Mnemonics	Operands	Description	Operation	Flags	#Clock Note
ST	-Z, Rr	Store Indirect and Pre-Decrement	Z ← Z - 1, (Z) ← Rr	None	2
STD	Z+q,Rr	Store Indirect with Displacement	(Z + q) ← Rr	None	2
LPM		Load Program Memory	R0 ← (Z)	None	3
LPM	Rd, Z	Load Program Memory	Rd ← (Z)	None	3
LPM	Rd, Z+	Load Program Memory and Post-Increment	Rd ← (Z), Z ← Z + 1	None	3
ELPM		Extended Load Program Memory	R0 ← (RAMPZ:Z)	None	3
ELPM	Rd, Z	Extended Load Program Memory	Rd ← (RAMPZ:Z)	None	3
ELPM	Rd, Z+	Extended Load Program Memory and Post-Increment	Rd ← (RAMPZ:Z), Z ← Z + 1	None	3
SPM		Store Program Memory	(Z) ← R1:R0	None	-
ESPM		Extended Store Program Memory	(RAMPZ:Z) ← R1:R0	None	-
IN	Rd, A	In From I/O Location	Rd ← I/O(A)	None	1
OUT	A, Rr	Out To I/O Location	I/O(A) ← Rr	None	1
PUSH	Rr	Push Register on Stack	STACK ← Rr	None	2
POP	Rd	Pop Register from Stack	Rd ← STACK	None	2
Bit and Bit-test Instructions					
LSL	Rd	Logical Shift Left	Rd(n+1) ← Rd(n),Rd(0) ← 0,C ← Rd(7)	Z,C,N,V,H	1
LSR	Rd	Logical Shift Right	Rd(n) ← Rd(n+1),Rd(7) ← 0,C ← Rd(0)	Z,C,N,V	1
ROL	Rd	Rotate Left Through Carry	Rd(0) ← C,Rd(n+1) ← Rd(n),C ← Rd(7)	Z,C,N,V,H	1
ROR	Rd	Rotate Right Through Carry	Rd(7) ← C,Rd(n) ← Rd(n+1),C ← Rd(0)	Z,C,N,V	1
ASR	Rd	Arithmetic Shift Right	Rd(n) ← Rd(n+1), n=0..6	Z,C,N,V	1
SWAP	Rd	Swap Nibbles	Rd(3..0) ↔ Rd(7..4)	None	1
BSET	s	Flag Set	SREG(s) ← 1	SREG(s)	1
BCLR	s	Flag Clear	SREG(s) ← 0	SREG(s)	1
SBI	A, b	Set Bit in I/O Register	I/O(A, b) ← 1	None	2
CBI	A, b	Clear Bit in I/O Register	I/O(A, b) ← 0	None	2
BST	Rr, b	Bit Store from Register to T	T ← Rr(b)	T	1
BLD	Rd, b	Bit load from T to Register	Rd(b) ← T	None	1
SEC		Set Carry	C ← 1	C	1
CLC		Clear Carry	C ← 0	C	1
SEN		Set Negative Flag	N ← 1	N	1
CLN		Clear Negative Flag	N ← 0	N	1
SEZ		Set Zero Flag	Z ← 1	Z	1
CLZ		Clear Zero Flag	Z ← 0	Z	1

Bảng 5. Tóm tắt tập lệnh (tiếp theo).

Mnemonics	Operands	Description	Operation	Flags	#Clock Note
SEI		Global Interrupt Enable	I ← 1	I	1
CLI		Global Interrupt Disable	I ← 0	I	1
SES		Set Signed Test Flag	S ← 1	S	1
CLS		Clear Signed Test Flag	S ← 0	S	1
SEV		Set Two's Complement Overflow	V ← 1	V	1
CLV		Clear Two's Complement Overflow	V ← 0	V	1
SET		Set T in SREG	T ← 1	T	1
CLT		Clear T in SREG	T ← 0	T	1
SEH		Set Half Carry Flag in SREG	H ← 1	H	1
CLH		Clear Half Carry Flag in SREG	H ← 0	H	1
NOP		No Operation		None	1
SLEEP		Sleep	(see specific descr. for Sleep)	None	1
WDR		Watchdog Reset	(see specific descr. for WDR)	None	1

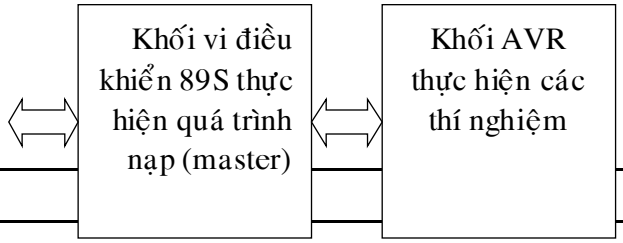
Bảng 5. Tóm tắt tập lệnh (tiếp theo).

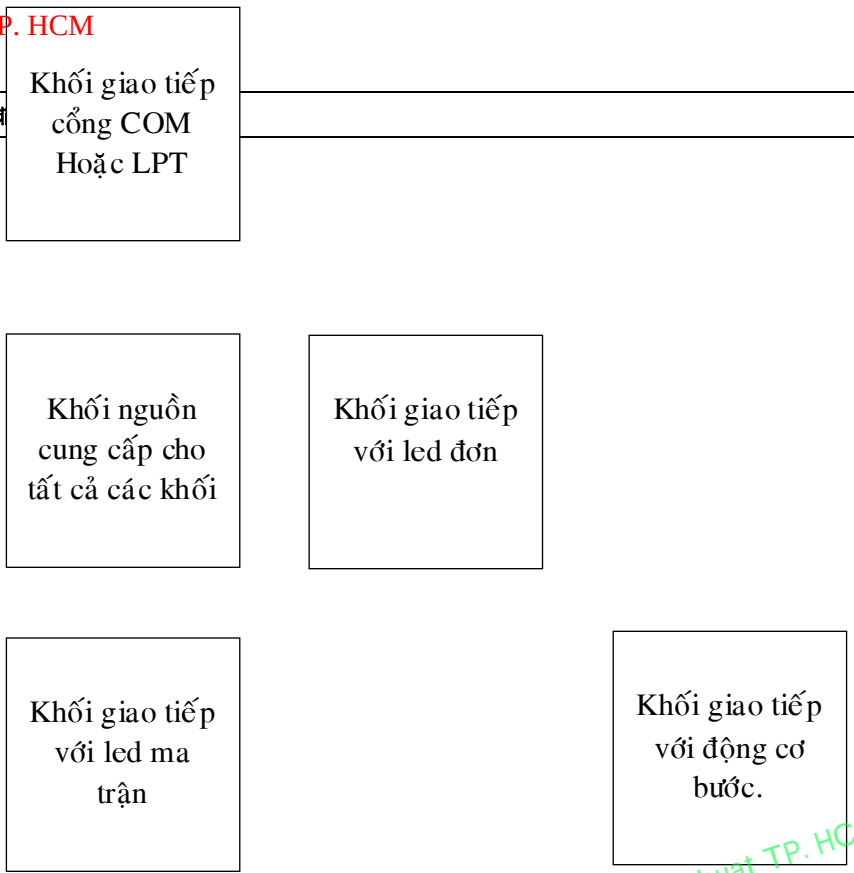
6. Thiết kế phần cứng hệ thống:

Sau khi nắm bắt được cấu trúc, tổ chức và khả năng ứng dụng của chip AVR tác giả phải thực hiện các công việc như sau:

- Phải thiết kế mạch nạp chip AVR dạng nối tiếp và giao tiếp với máy tính bằng cổng COM hoặc cổng LPT.
- Chip AVR thí nghiệm phải đưa ra đầy đủ các chân điều khiển để giao tiếp với các thiết bị ngoại vi.
- Lựa chọn các ứng dụng phổ biến như giao tiếp với led đơn, giao tiếp với led 7 đoạn, giao tiếp với led ma trận, giao tiếp với LCD và giao tiếp với ma trận bàn phím. Các giao tiếp phải có đầy đủ tên các ngõ vào ra, trạng thái điều khiển và thuận tiện cho việc kết nối một cách dễ dàng.
- Thiết kế nguồn cung cấp cho hệ thống.

Sơ đồ kết nối các hệ thống như hình 21:





Hình 21. Sơ đồ khối của hệ thống.

Trong sơ đồ khối ở trên trừ 3 khối đầu tiên liên kết với nhau, còn các khối còn lại không liên kết với nhau nhưng trong các ứng dụng thì chúng sẽ liên kết với nhau bằng các dây bus gắn thêm vào.

Chip AVR thực hiện các thí nghiệm giao tiếp sẽ kết nối với các khối ngoại vi bằng dây bus. Khi thực hiện thí nghiệm giao tiếp với khối nào thì người sử dụng sẽ kết nối với khối đó.

Sơ đồ nguyên lý của các khối như sau:

a. Sơ đồ nguyên lý giao tiếp giữa máy tính và chip AVR AT90S8535:

Chip AVR ngoài chức năng cho phép nạp chương trình dạng song song còn có chức năng cho phép nạp chương trình dạng nối tiếp.

Khi nạp nối tiếp mạch giao tiếp giữa thiết bit nạp với AVR AT90S8535 chỉ cần dùng 3 đường điều khiển của port 1 đó là PB.5, PB.6, PB.7.

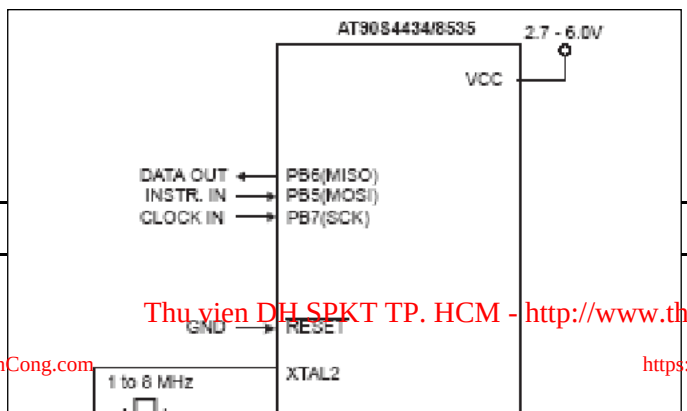
Chân PB.5 có tên là MOSI: là đường nhập dữ liệu vào nối tiếp của vi điều khiển nạp.

Chân PB.6 có tên là MISO: đường xuất dữ liệu vào nối tiếp của vi điều khiển nạp.

Chân PB.7 có tên là SCK: đường cung cấp xung đồng hồ để đồng bộ dữ liệu nối tiếp.

Ngoài 3 chân điều khiển trên thì phải thêm một đường tín hiệu điều khiển chân reset: khi nạp thì chân reset ở mức thấp và sau khi nạp xong thì phải cho chân reset lên mức cao để chip AVR có thể thực hiện chương trình sau khi nạp xong.

Hãy xem sơ đồ trình bày các đường tín hiệu điều khiển nạp như hình 22.



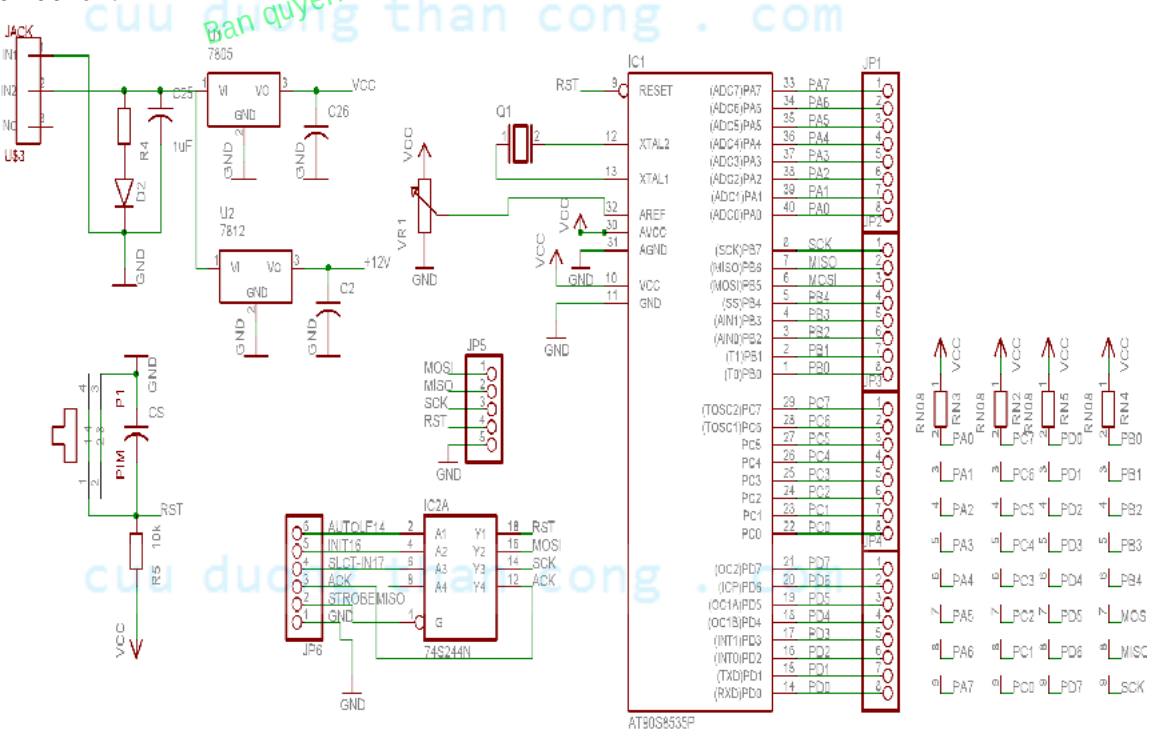
Hình 22. Sơ đồ giao tiếp mạch nạp.

Có rất nhiều đối tượng giao tiếp với vi điều khiển nạp, theo tác giả chọn một vi điều khiển thực hiện quá trình nạp và giao tiếp với máy tính để nhận lệnh và dữ liệu nạp. Nhưng trong quá trình thực hiện thì kết quả là chưa thành công nên tác giả sử dụng mạch nạp dùng cổng LPT của hãng ATMEL và chương trình nạp và biên dịch của chính hãng ATMEL.

Sơ đồ kết nối máy tính dùng cổng LPT và giao tiếp với chip AVR nạp như hình 23.

Trong hệ thống này có luôn cả hệ thống mạch nguồn ổn áp 5 V và 12V cung cấp cho toàn bộ mạch điện nạp và các mạch giao tiếp.

Do bo mạch vừa nạp và thực hiện các thí trên bo nên các port của vi điều khiển thí nghiệm phải sử dụng điện trở kéo lên.



Hình 23 Sơ đồ giao tiếp mạch nạp AVR dùng cổng LPT.

Để nạp dữ liệu cho vi điều khiển thì phải thực hiện theo đúng trình tự yêu cầu của nhà chế tạo. Các quá trình thực hiện được cho ở bảng 6:

Instruction	Instruction Format				Operation
	Byte 1	Byte 2	Byte 3	Byte4	
Programming Enable	1010 1100	0101 0011	xxxx xxxx	xxxx xxxx	Enable Serial Programming while RESET is low.
Chip Erase	1010 1100	100x xxxx	xxxx xxxx	xxxx xxxx	Chip erase Flash and EEPROM memory arrays.
Read Program Memory	0010 H000	xxxx aaaa	bbbb bbbb	oooo oooo	Read H (high or low) data o from Program memory at word address a:b.
Write Program Memory	0100 H000	xxxx aaaa	bbbb bbbb	iiii iiii	Write H (high or low) data i to Program memory at word address a:b.
Read EEPROM Memory	1010 0000	xxxx xxxa	bbbb bbbb	oooo oooo	Read data o from EEPROM memory at address a:b.
Write EEPROM Memory	1100 0000	xxxx xxxa	bbbb bbbb	iiii iiii	Write data i to EEPROM memory at address a:b.
Read Lock and Fuse Bits	0101 1000	xxxx xxxx	xxxx xxxx	12Sx xxxF	Read Lock and Fuse bits. '0' = programmed, '1' = unprogrammed.
Write Lock Bits	1010 1100	1111 12i1	xxxx xxxx	xxxx xxxx	Write Lock bits. Set bits *1,2* = '0' to program Lock bits.
Read Signature Byte	0011 0000	xxxx xxxx	xxxx xxbb	oooo oooo	Read Signature Byte o at address b.(1)
Write FSTRT Fuse	1010 1100	1011 111P	xxxx xxxx	xxxx xxxx	Write FSTRT fuse. Set bit F = '0' to program, '1' to unprogram.

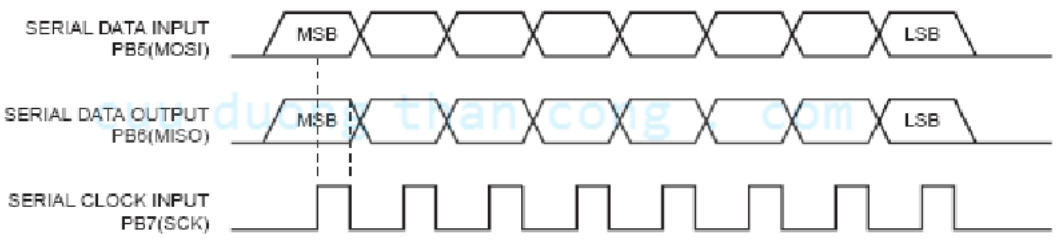
Note: a = address high bits
b = address low bits
H = 0 - Low byte, 1 - High Byte
o = data out
i = data in
x = don't care
1 = Lock Bit 1
2 = Lock Bit 2
F = FSTRT Fuse
S = SPIEN Fuse

Note: 1. The signature bytes are not readable in Lock mode 3, i.e. both Lock bits programmed.

Bảng 6. Các quá trình nạp bộ nhớ flash của AVR AT90S8535.

Trình tự thực hiện dạng sóng của 3 đường tín hiệu điều khiển như hình 24.

Serial Programming Waveforms



Hình 24. Giản đồ thời gian của các đường tín hiệu nạp của AVR AT90S8535.

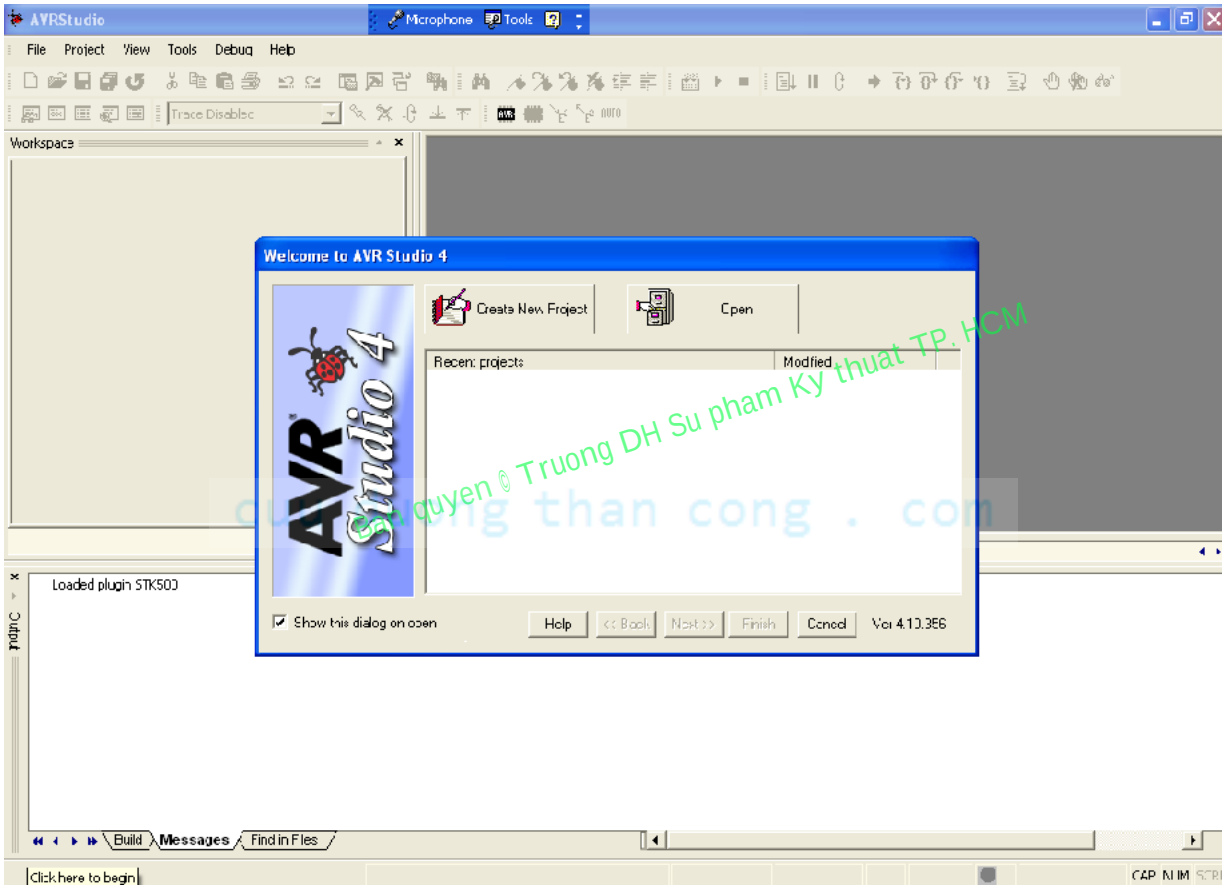
7. Hướng dẫn sử dụng phần mềm:

- Cách sử dụng chương trình trên máy tính để soạn thảo và biên dịch chương trình:

Như đã trình bày ở trên do nghiên cứu chưa thành công mạch nạp dùng vi điều khiển nên tác giả sử dụng mạch nạp và **chương trình biên dịch của hãng** ATMEL**. Chương trình biên dịch có tên là AVRStudio có chức năng soạn thảo chương trình và mô phỏng.**

Cách thức sử dụng chương trình như sau:

Sau khi cài đặt xong chương trình ta tiến hành khởi động chương trình – khi đó màn hình soạn thảo xuất hiện như hình 33:

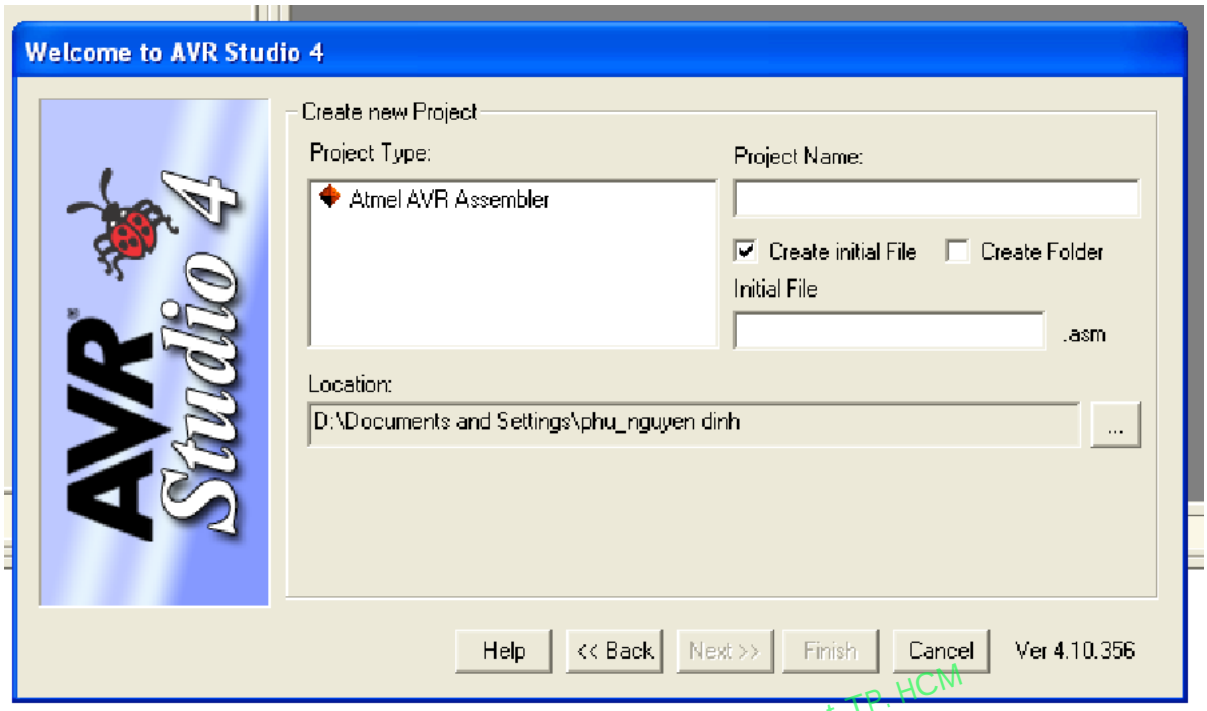


Hình 33. Màn hình soạn thảo của chương trình AVRStudio.

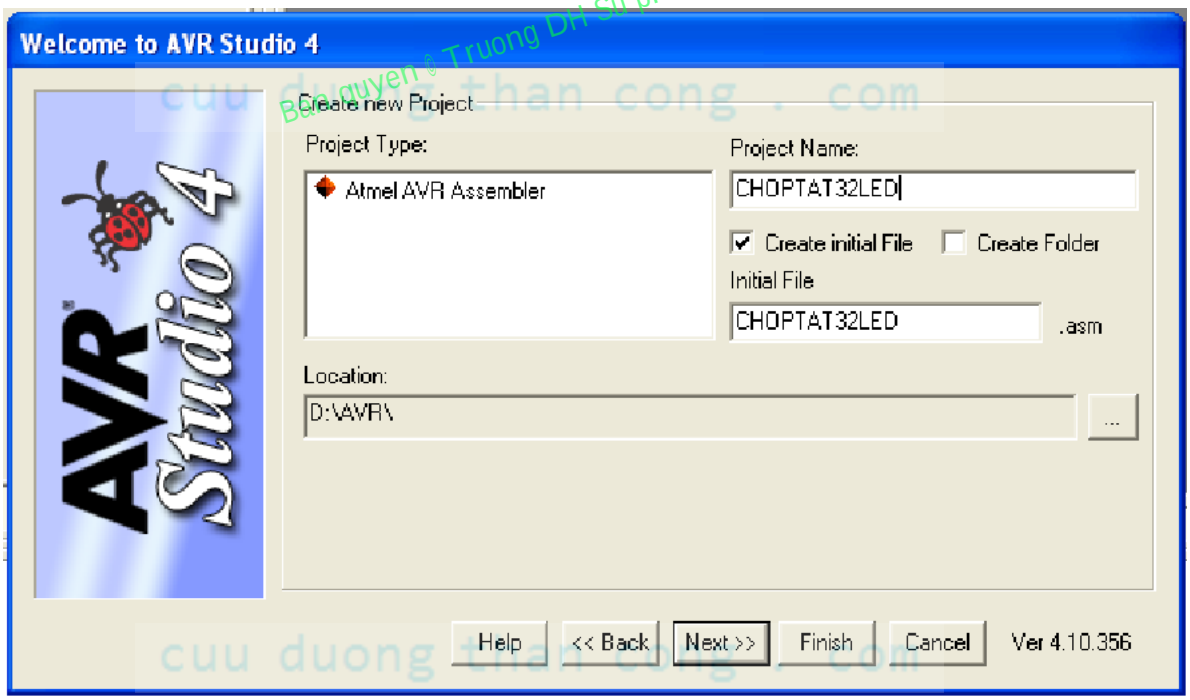
Một của sổ menu nhỏ xuất hiện cho phép bạn chọn project mới hay mở một project có sẵn.

Chọn xây dựng một project mới thì màn hình tiếp theo như hình 34 sẽ xuất hiện.

Người lập trình hãy đánh tên cho project sẽ soạn thảo và mô phỏng vào ô project name và chọn thư mục lưu trữ project này – hãy xem hình 35. Trong hình này tên project mới là "choptat32led". Sau khi nhập tên và lưu chọn thư mục xong ta nhấn nút "next" để chuyển sang lựa chọn IC như hình 36.

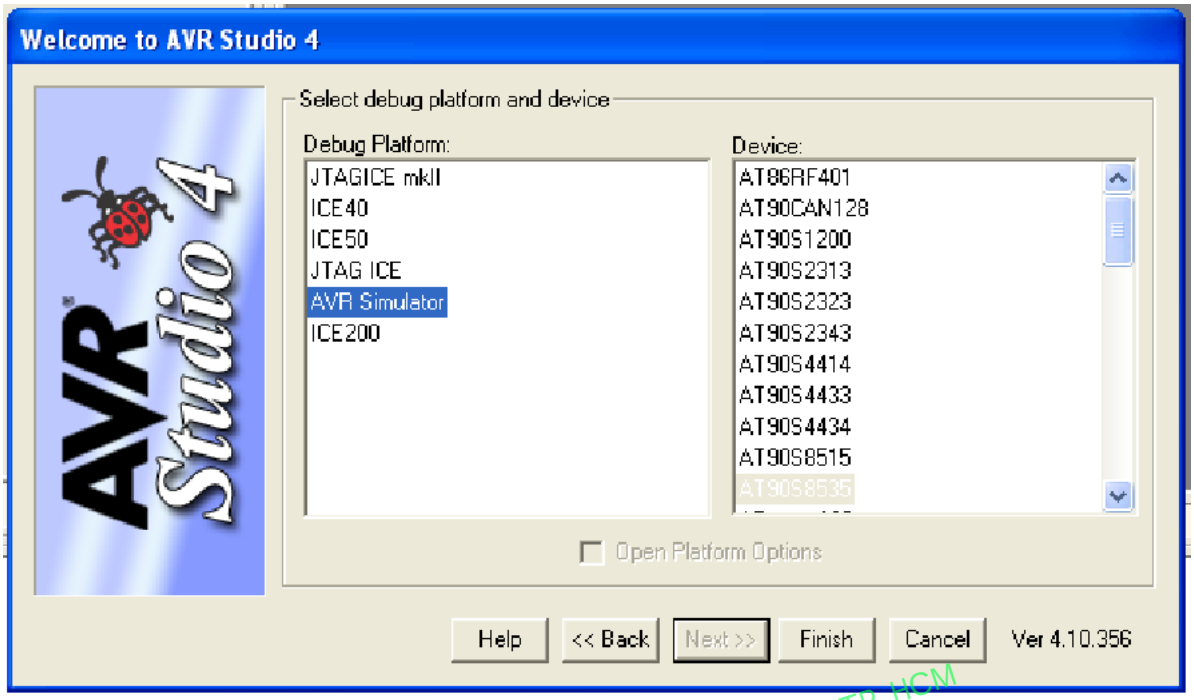


Hình 34. Màn hình soạn thảo project mới của chương trình AVRStudio.

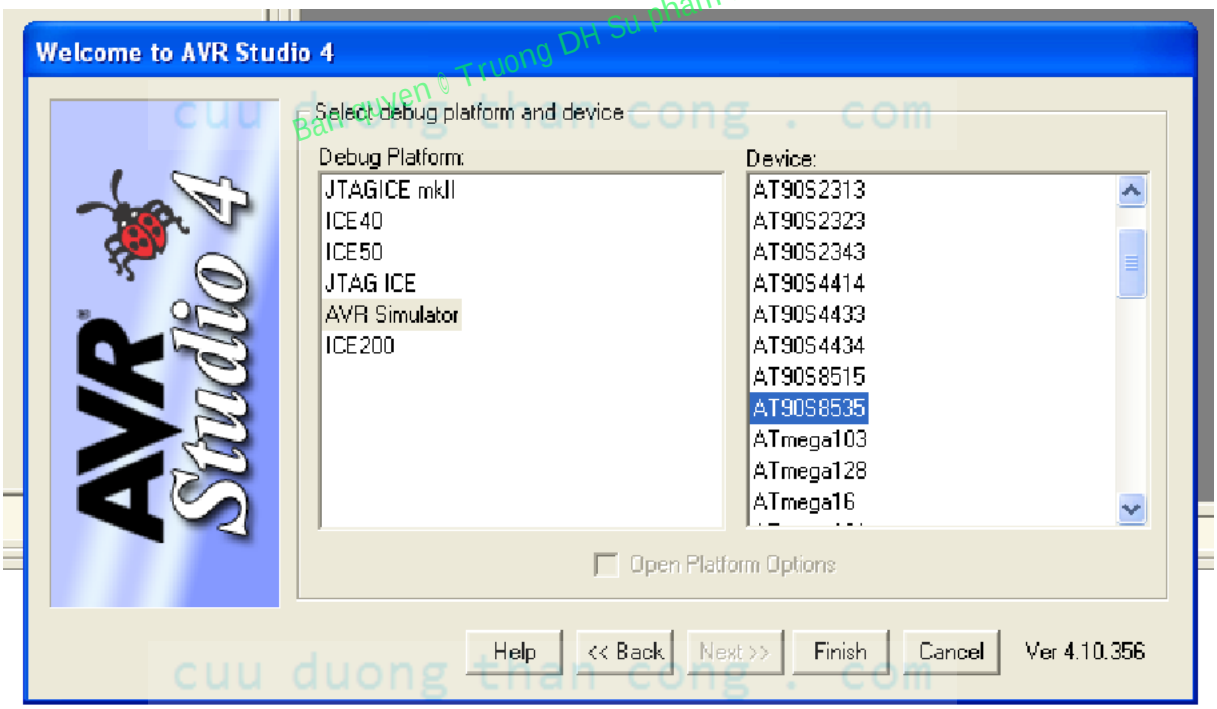


Hình 35. Màn hình nhập tên và thư mục của project mới.

Chọn mục AVR Simulator như trong hình 36 và chọn loại vi điều khiển AT90S8535 như hình 37 rồi nhấn nút lệnh có tên là “Finish”.

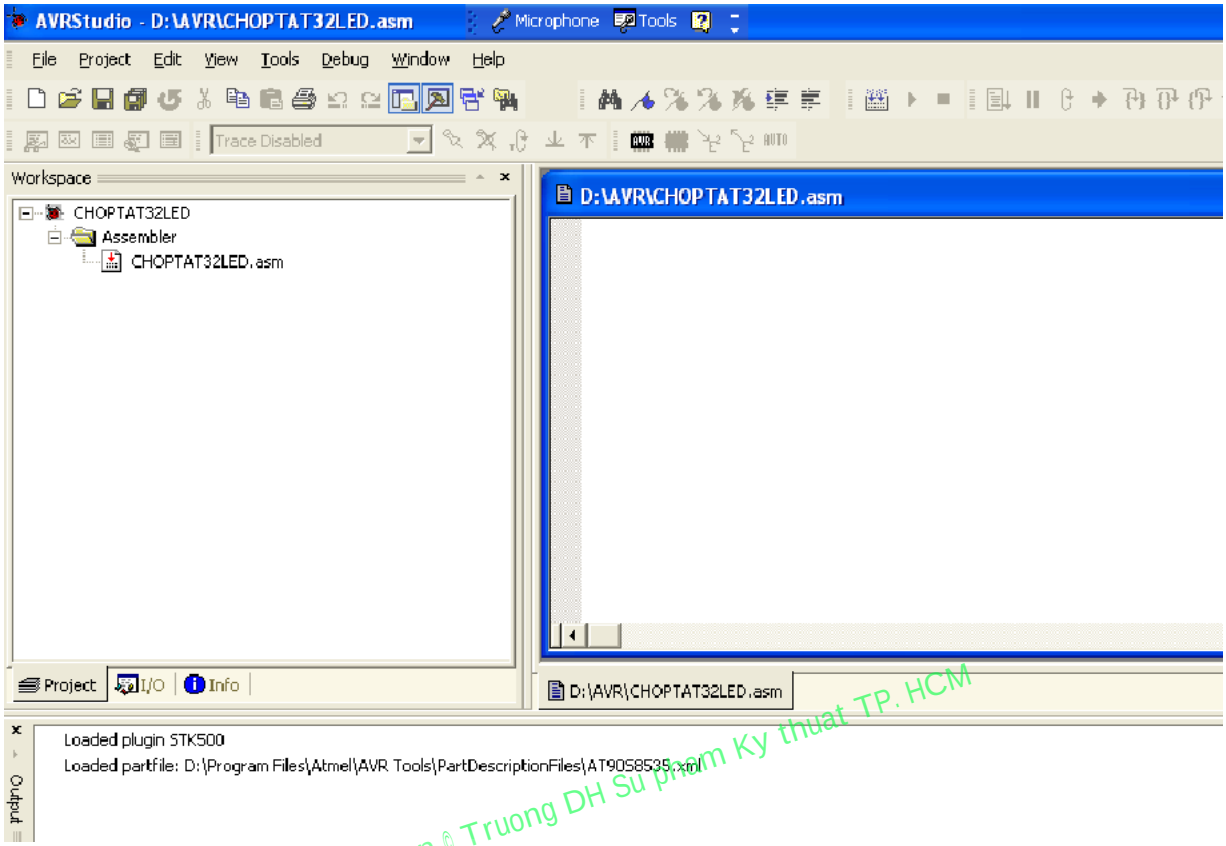


Hình 36. Màn hình chọn mô phỏng của project mới.



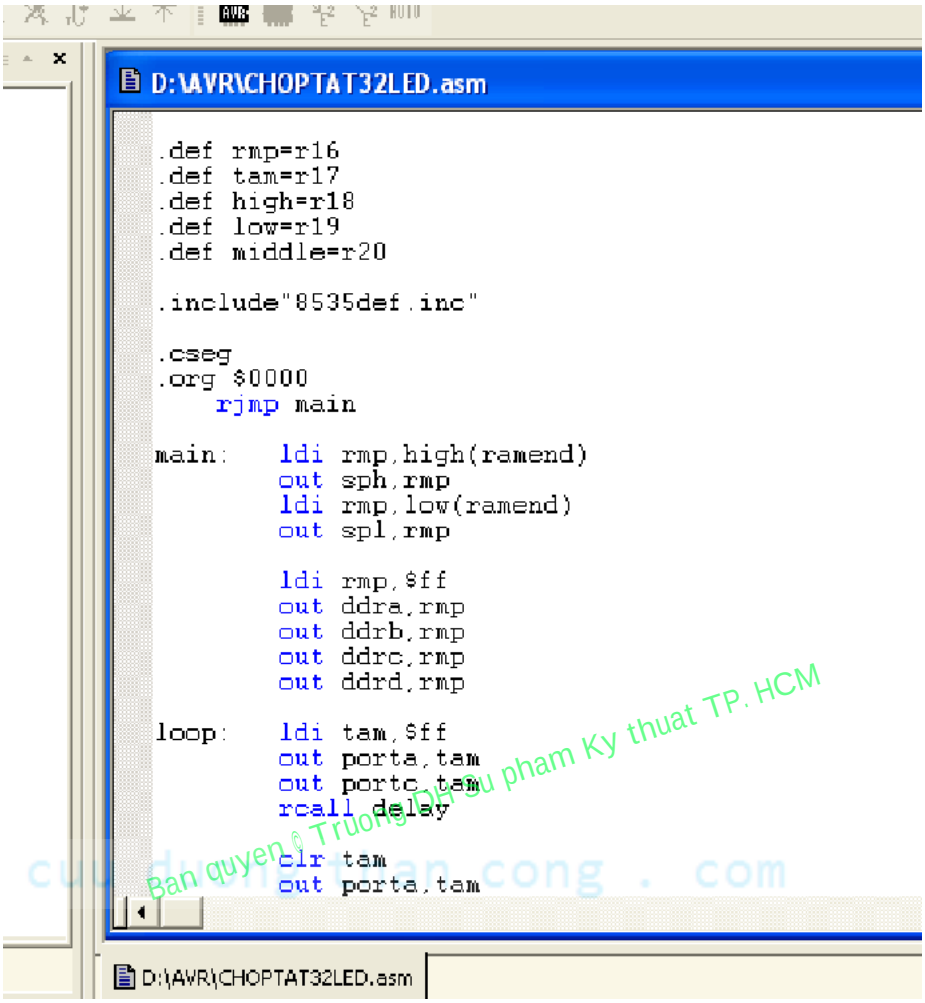
Hình 37. Màn hình chọn IC mô phỏng của project mới.

Kết quả ta được màn hình soạn thảo chương trình như hình 38.



Hình 38. Màn hình soạn thảo chương trình của project mới.

Hãy nhập chương trình chóp tắt 32 led vào như hình 39.

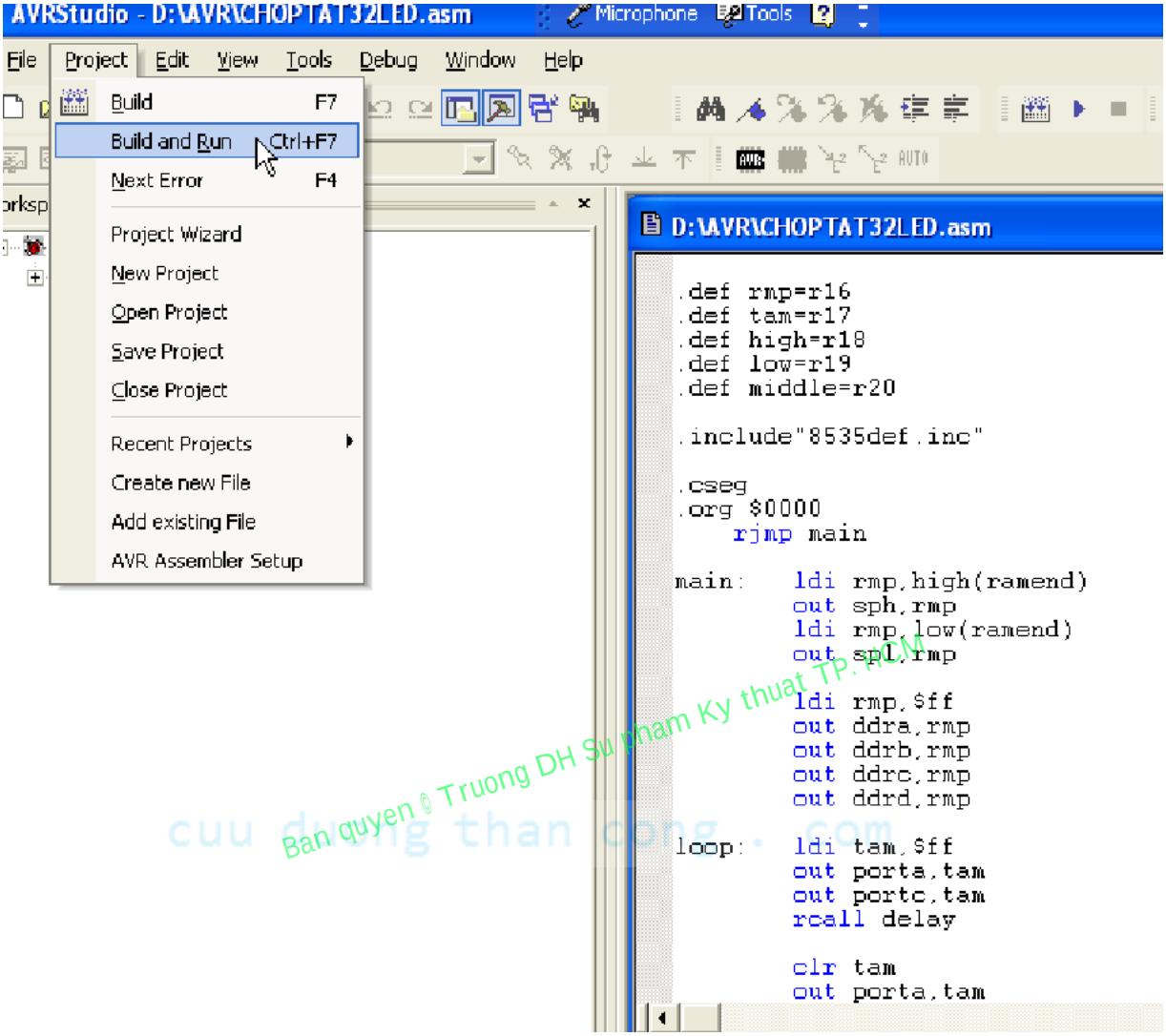


Hình 39. Màn hình soạn thảo chương trình chóp tắt 32 led.

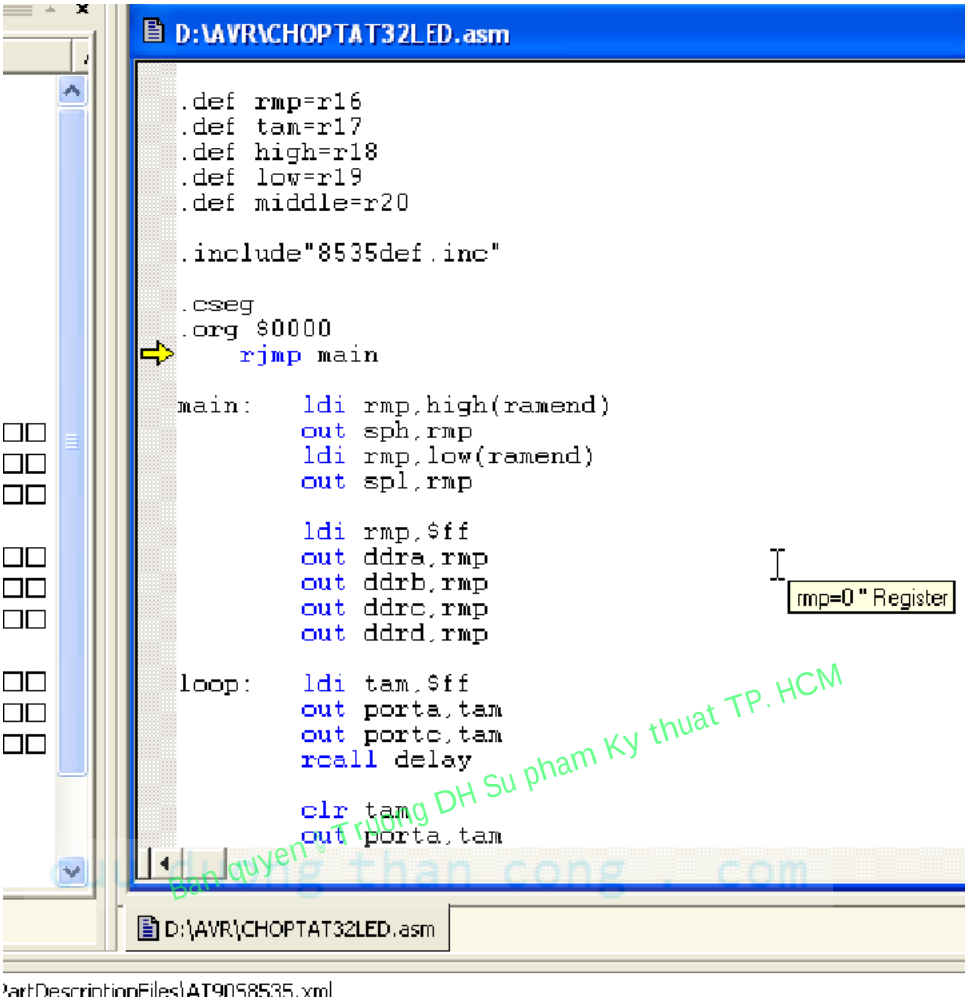
Tiến hành biên dịch bằng cách vào menu lệnh project và chọn vào mục như hình 40 thì khi đó chương trình sẽ được biên dịch.

Nếu chương trình soạn thảo đúng cú pháp thì sẽ thấy xuất hiện thanh trạng thái cho biết quá trình biên dịch đang tiến hành và sau khi biên dịch xong sẽ xuất hiện dấu mũi tên cho phép quá trình mô phỏng sẽ thực hiện – hãy xem hình 41.

Nếu soạn thảo không đúng thì sau khi biên dịch xong sẽ không thấy xuất hiện thanh trạng thái cho biết quá trình biên dịch đang tiến hành và cũng không có dấu mũi tên cho việc mô phỏng sau khi biên dịch xong. Trong trường hợp này chúng ta hãy tiến hành xem lại chương trình xem các lệnh ta viết có đúng cú pháp hay không và lệnh đó có tồn tại hay không. Tiến hành biên dịch lại cho đến khi hết lỗi thì xong.



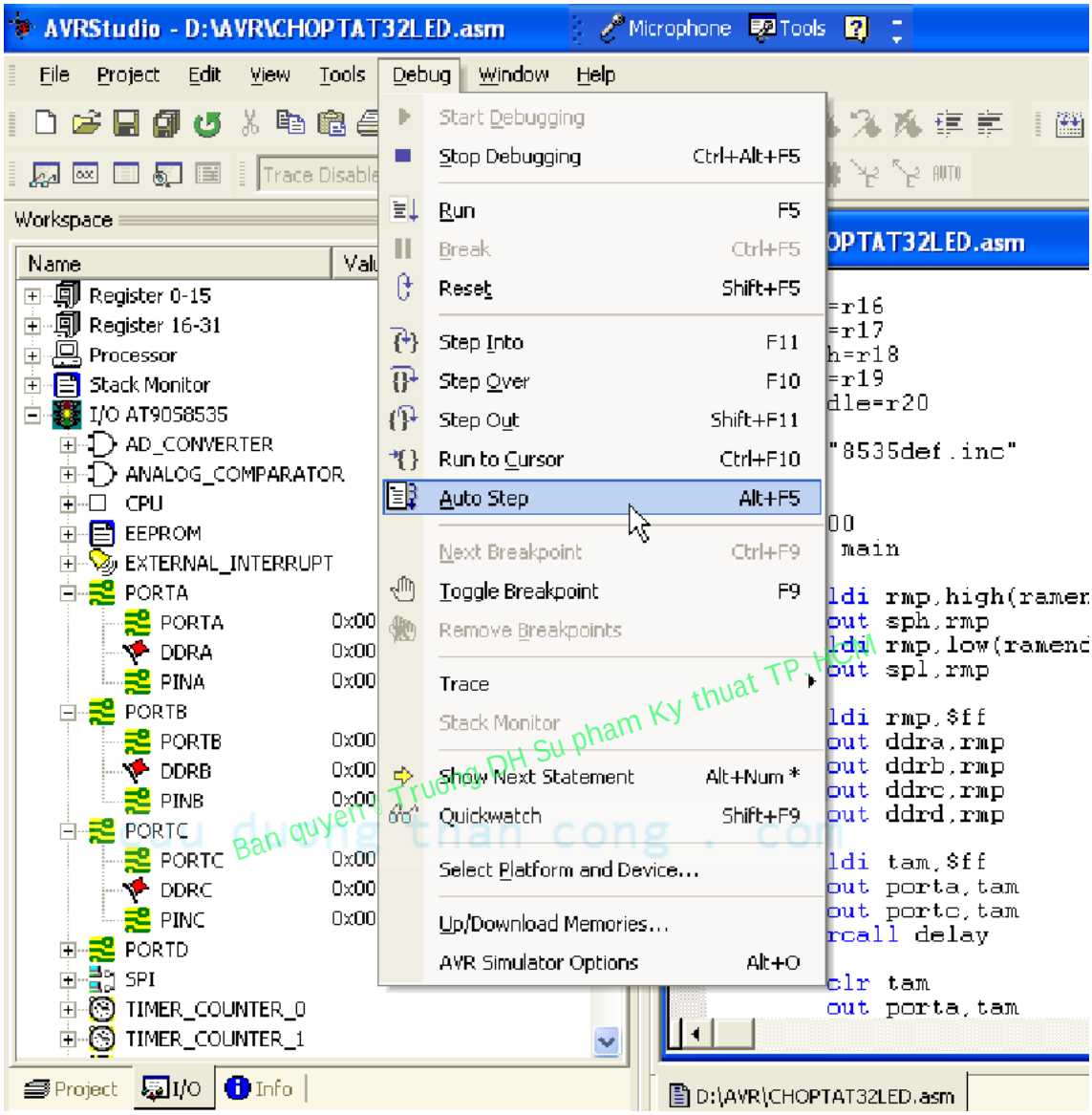
Hình 40. Menu lệnh biên dịch chương trình chớp tắt 32 led.



Hình 41. Dấu mũi tên màu vàng cho biết chương trình biên dịch tốt.

- Tiến hành mô phỏng:

Sau khi chương trình đã biên dịch thành công thì ta tiến hành mô phỏng chương trình bằng cách vào menu **tool** rồi chọn lệnh auto step hay nhấn tổ hợp phím ALT + F5 như hình 42. Khi đó quá trình mô phỏng sẽ được thực hiện. Trong cửa sổ Workspace bạn hãy bấm vào mục I/O AT90S8535 thì cấu hình phần cứng mô phỏng sẽ xuất hiện và bạn ba71, vào các portA, portB, portC và portD thì bạn sẽ thấy kết quả thực hiện chương trình mô phỏng sẽ làm thay đổi nội dung các ô nhớ này.



Hình 42. Chọn lệnh để bắt đầu mô phỏng.

- Viết các bài thí nghiệm:

Các bài thí nghiệm được xây dựng để cho khai thác hết các khả năng của vi điều khiển.

Tất cả các bài thí nghiệm không trình bày trong báo cáo này nhưng có tổ chức thành các thư mục lưu trong đĩa CD kèm theo báo cáo này.

Các bài thực hành giao tiếp với led đơn.

Sử dụng hệ thống vi điều khiển kết nối 4 port với 32 led đơn để viết các chương trình ứng dụng điều khiển led sáng theo các yêu cầu từng bài. Mục đích làm quen với một số lệnh cơ bản và lập trình.

Trình tự thực hiện hãy dùng 4 dây bus 8 sợi kết nối portA, portB, portB và portD đến 32 led theo đúng thứ tự từ bit thấp đến bit cao.

Các bài thí nghiệm giao tiếp với 32 led đơn như sau:

Bài số 11: Viết chương trình điều khiển sáng tắt 32 led .

Bài số 12: Viết chương trình điều khiển 32 led sang dần và tắt dần.

Bài số 13: Viết chương trình điều khiển 32 led sáng dồn.

Bài số 14: Viết chương trình điều khiển 32 led tắt dồn.

Bài số 15: Viết chương trình điều khiển 32 led sáng tổng hợp các chương trình trên.

Các bài thực hành giao tiếp với 8 led 7 đoạn.

Với led 7 đoạn thì có thể cho phép hiển thị chữ và số - khi đó có rất nhiều chương trình ứng dụng có thể thực hiện được trên hệ thống này như chương trình đếm sản phẩm, chương trình đếm tần số, chương trình đồng hồ số, chương trình đồng hồ thể thao ...

Với hệ thống này có thể cho thấy rõ hoạt động của phương pháp quét led hiển thị, việc giải mã led hiển thị bằng chương trình quét hiển thị, nguyên lý làm việc và chương trình quét phím.

Các bài thí nghiệm phục vụ cho việc điều khiển các led gồm các bài cơ bản và rất nhiều bài tập kèm theo.

Khi giao tiếp với 8 led 7 đoạn phải sử dụng 2 port kết nối với led 7 đoạn, trong từng bài có ghi rõ port nào điều khiển transistor quét và port nào điều khiển các đoạn thì phải kết nối đúng port và đúng thứ tự bit.

Các bài thí nghiệm giao tiếp với led 7 đoạn như sau:

Bài số 21: Các chương trình thử 8 led 7 đoạn.

Bài số 22: Chương trình đếm lên 2 số.

Bài số 23: Các chương trình đếm giây.

Bài số 24: Các chương trình đếm phút.

Bài số 25: Chương trình đếm giờ - phút - giây.

Bài số 26: Chương trình điều khiển đèn giao thông.

Bài số 27: Chương trình điều khiển đèn giao thông có hiển thị thời gian đếm xuống.

Bài số 28: Chương trình đếm sản phẩm 1 kênh.

Bài số 29: Chương trình đếm sản phẩm 2 kênh.

Các bài thực hành giao tiếp với led ma trận 8x8 hai màu xanh đỏ.

Với phần cứng đã thiết kế ở trên sử dụng led ma trận 8x8 có 2 màu xanh và đỏ, để điều khiển led ma trận sáng ta tiến hành gởi dữ liệu ra hàng và mã quét ra cột. Trong 4 port ta sử dụng portD làm port điều khiển hàng và portA điều khiển cột xanh và portC điều khiển cột đỏ.

Các chương trình điều khiển led ma trận bao gồm các bài như sau:

Bài số 31: Chương trình hiển thị kí tự A.

Bài số 32: Chương trình chớp tắt kí tự A.

Bài số 33: Chương trình hiển thị chuỗi “SPKT” màu xanh.

Bài số 34: Chương trình hiển thị chuỗi “SPKT” màu đỏ.

Bài số 35: Chương trình hiển thị chuỗi “SPKT” màu cam.

Bài số 36: Chương trình hiển thị chuỗi “SPKT” ba màu xanh đỏ cam.

Bài số 37: Chương trình hiển thị chuỗi "SPKT" hai màu: nữa trên xanh, nữa dưới đỏ và ngược lại.

Bài số 38: Chương trình hiển thị trái tim rơi từ trên xuống và từ dưới lên.

Các bài thực hành giao tiếp với LCD

Như đã trình bày ở trên khi giao tiếp với LCD thì phải dùng 11 đường tín hiệu điều khiển, trong đó có 3 đường điều khiển và 8 đường dữ liệu phải sử dụng nguyên 1 port.

Trong các bài thí nghiệm tác giả sử dụng portA để giao tiếp 8 đường tự liệu (chú ý theo đúng thứ tự bit từ 0 đến 7) và 3 bit 0, 1, 2 của portC làm 3 đường điều khiển.

Các bài thí nghiệm giao tiếp với LCD bao gồm:

Bài số 41: Chương trình hiển thị chuỗi dữ liệu đứng yên.

Bài số 42: Chương trình hiển thị chuỗi dữ liệu dịch chuyển.

Bài số 43: Chương trình hiển thị giờ phút giây.

Bài số 44: Chương trình đếm sản phẩm hiển thị trên LCD.

Các bài thực hành giao tiếp với ma trận phím và 8 led 7 đoạn.

Bàn phím đóng vai trò nhập dữ liệu cho hệ thống điều khiển, để thực hiện giao tiếp với bàn phím thì ngoài giao tiếp chip AVR với bàn phím thì phải có thêm giao tiếp giữa chip AVR với led đơn hoặc led 7 đoạn hoặc LCD thì chúng ta mới biết được quá trình thực hiện các yêu cầu đúng hay sai.

Các bài thí nghiệm giao tiếp với led 7 đoạn như sau:

Bài số 51: Chương trình nhấn phím số nào thì hiển thị trên mà hình đúng số đó.

Bài số 52: Chương trình đếm có các nút điều khiển start, stop.

Bài số 53: Chương trình điều khiển động cơ DC có 2 nút điều khiển Start, Stop.

Ngoài việc khai thác khả năng ứng dụng của timer như đã trình bày ở trên thì các bài thí nghiệm này khai thác khả năng sử dụng ngắt của timer, khai thác khả năng truyền dữ liệu nối tiếp, ngắt truyền dữ liệu.

Ngắt có nhiều ưu điểm trong điều khiển nhưng rất khó điều khiển và phức tạp do đó điều cần phải quan tâm là các bài thí nghiệm và các ứng dụng được thiết kế sao cho dễ hiểu.

Trong các bài thí nghiệm ngắt được dùng để truyền dữ liệu, để định thời, để xử lí nhiều chương trình phân chia theo thời gian – đây là một ứng dụng mạch nhất của ngắt.

the end

return

TÀI LIỆU THAM KHẢO

[1]. **Website của hãng Microchip**
http://www.microchip.com
[2] Datasheet của chip **PIC16F877A**
[3]. **User's Guide**
MPASM™ Assembler,MPLINK™ Object Linker,MPLIB™ Object Librarian
[4]. **CCS C Compiler v4** Help.
[5]. **PICmicro Language Tools and MPLAB IDE**
[6]. **Quick Reference Guide for C language**