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.

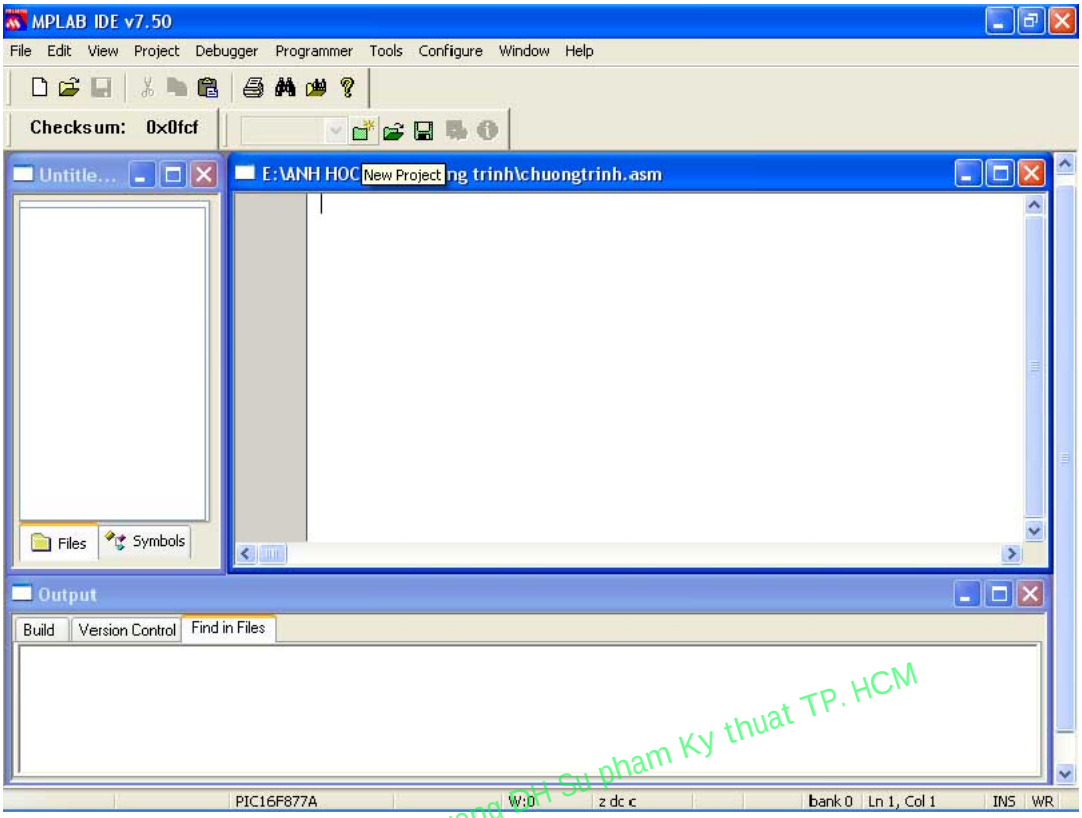
Phần mềm MPLAB IDE tương thích với hệ điều hành:

- ☐ 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.

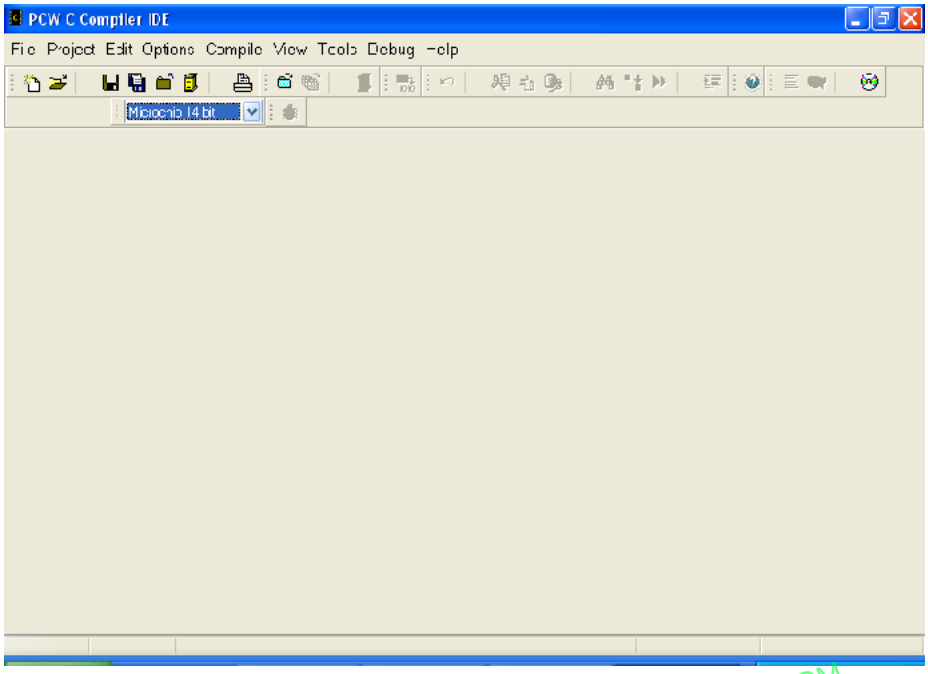
Chương trình là sự tích hợp của 3 trình biên dich riêng biệt cho 3 dòng PIC khác nhau đó là:

- 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.

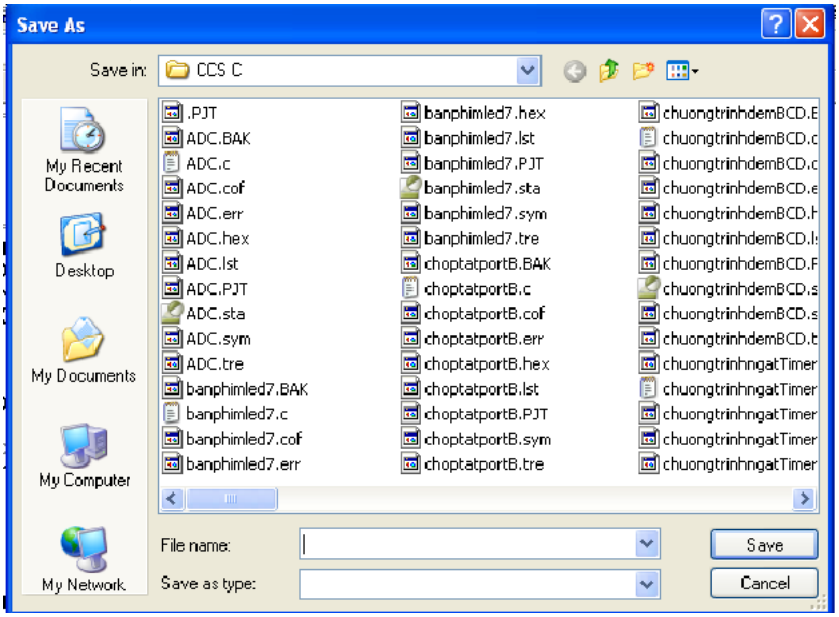
Khi khởi động chương trình CCS thì cửa sổ chương hình như hình dưới:



Hình 3-3. Màn hình khởi động của CCS C.

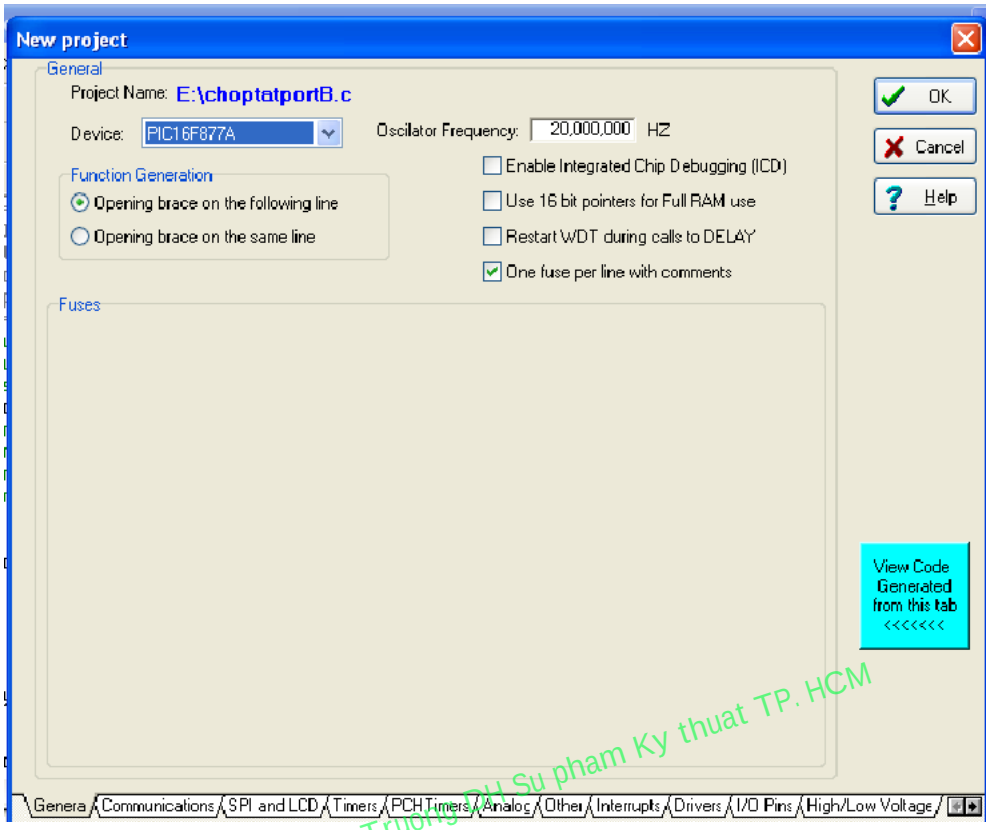
Hướng dẫn tạo một Project mới trong CCS:

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:



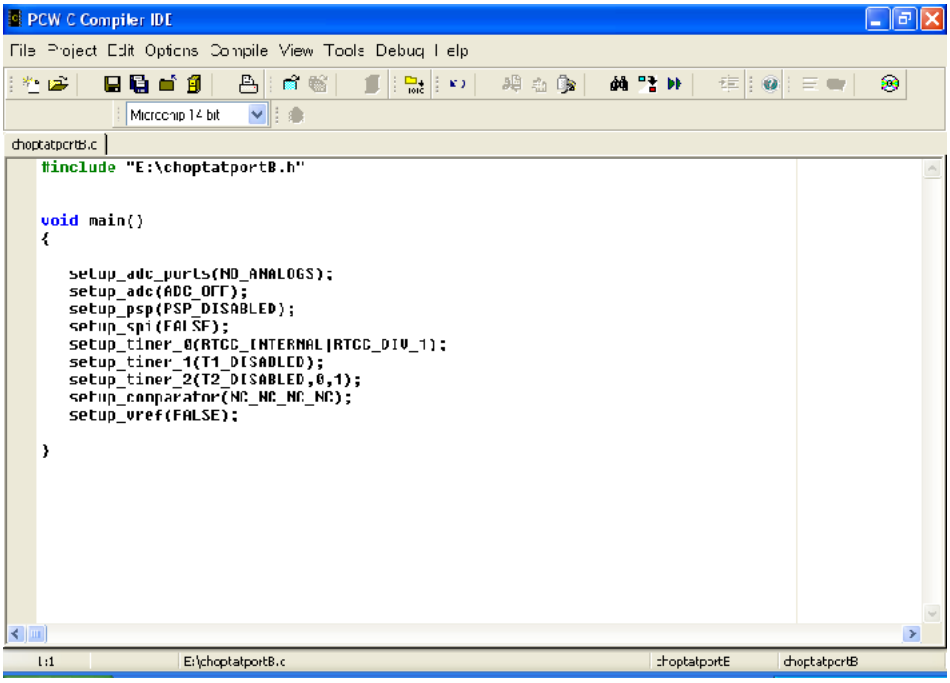
Hình 3-4. Lưu file.

Chọn Save một cửa sổ mới hiện ra như hình sau:



Hình 3-5. Tạo Project mới.

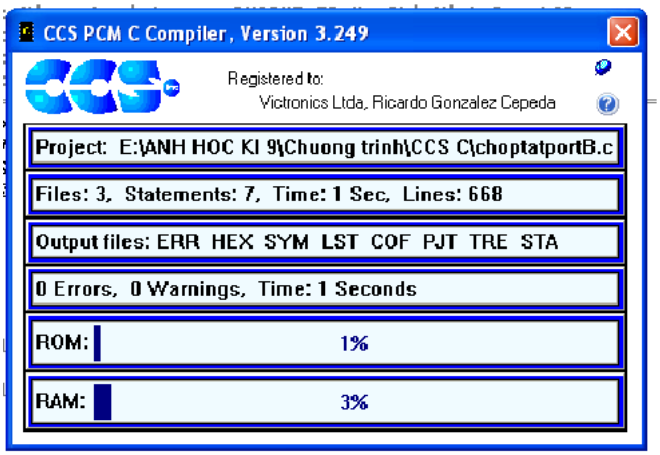
Sau đó nhấp OK là đã tạo được một Project mới và có cửa sổ làm việc mới như hình sau:



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.

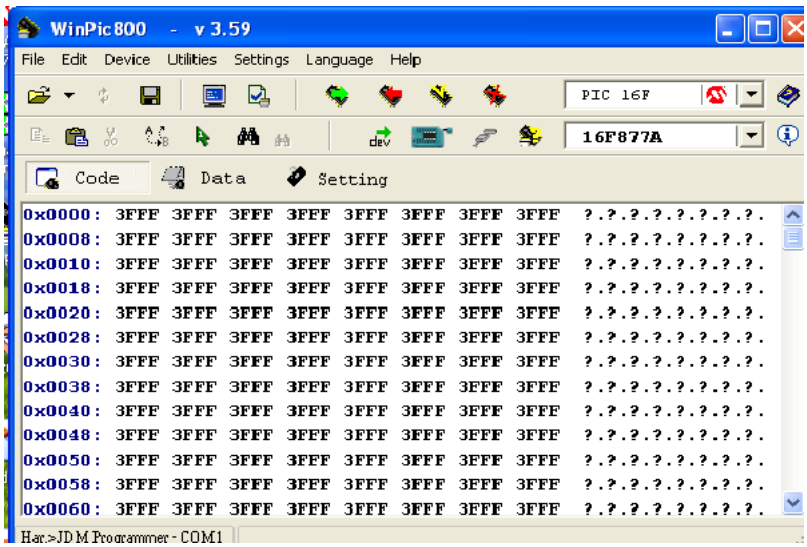
II. CHƯƠNG TRÌNH NẠP CHO PIC:

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:

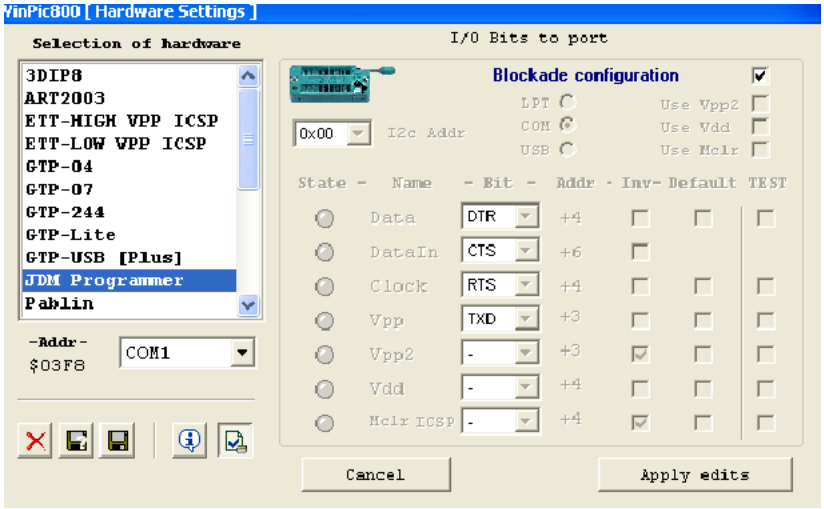
Hướng dẫn cài đặt Winpic800: chạy file WinPic800_V3_59.exe để cài đặt Winpic800, sau đó chọn next để tiến hành cài đặt.

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.

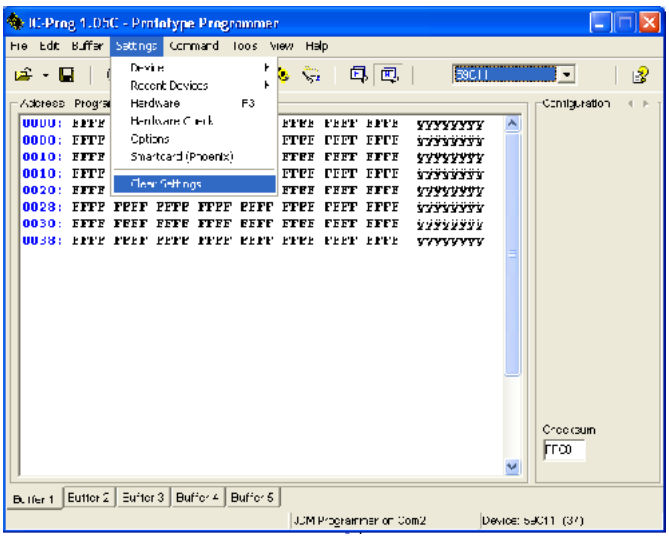
Sau đó chọn họ PIC và tên PIC muốn nạp chương trình. Ví dụ như muốn nạp cho PIC16F877A thì chọn họ 16F tên PIC là 16F877A.

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.

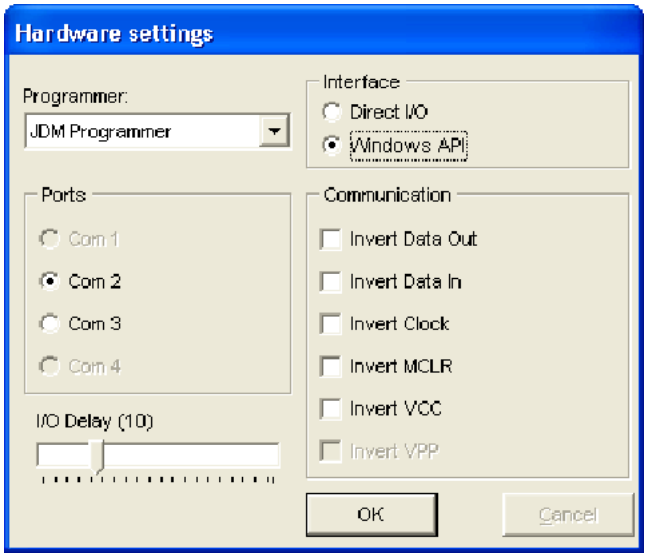
2. CHƯƠNG TRÌNH NẠP IC-PRO:

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.

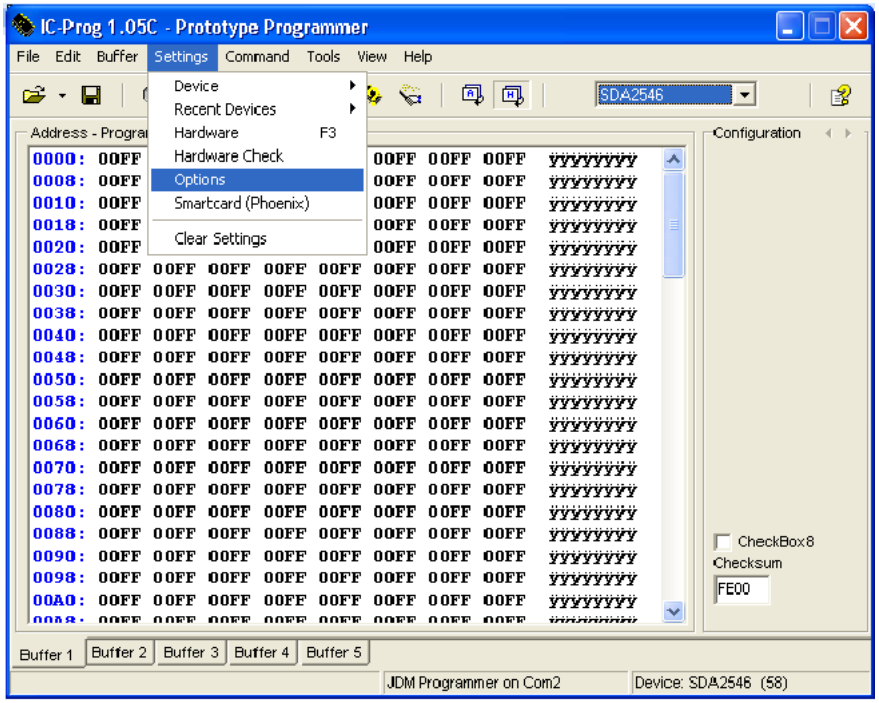
Sau khi nhấn Yes liên tục, một màn hình Hardware settings sẽ hiện ra như sau:



Hình 3-11. Cửa sổ Hardware Setting.

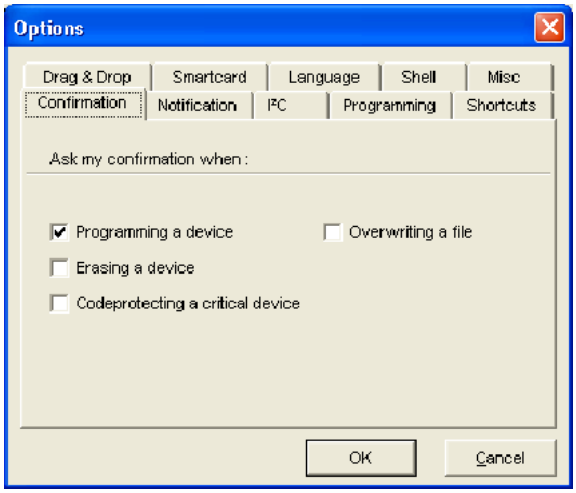
Do chúng ta chọn dùng bộ nạp PG1A là một bộ nạp được phát triển của JDM, cho nên phần Programmer chúng ta sẽ chọn JDM Programmer. Phần cổng, chúng ta sẽ chọn COM1, COM2 hoặc COM3 tùy theo máy tính. Phần Interface, các bạn chọn Windows API và phần Communication không đánh dấu gì cả, sau đó chọn OK. Khi 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.

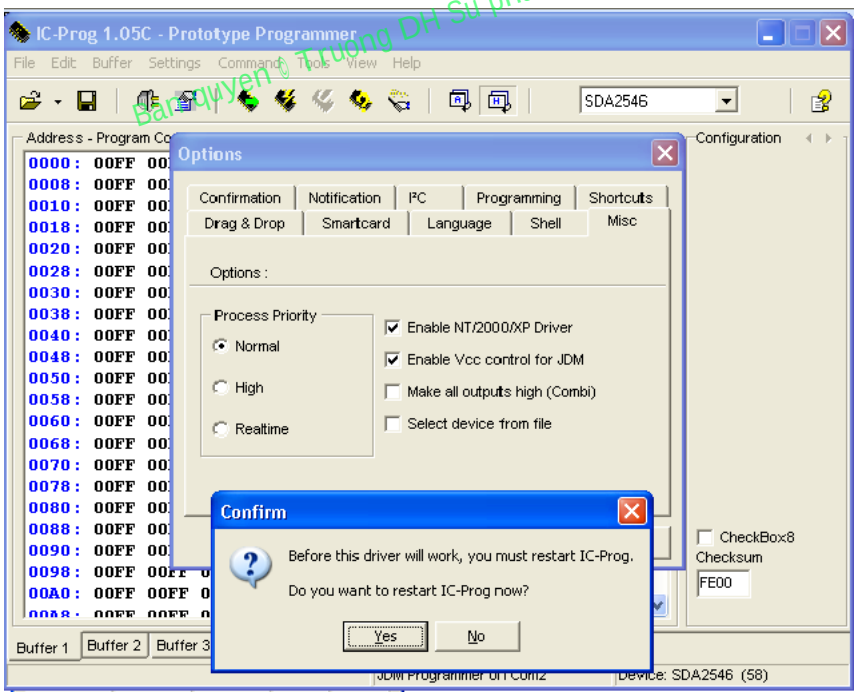
Màn hình Options sẽ hiện ra. Chỉ quan tâm tới phần Misc, còn các phần khác không cần quan tâm. Cứ để mặc định như chương trình ban đầu đã có.



Hình 3-13. Cửa sổ lựa chọn.

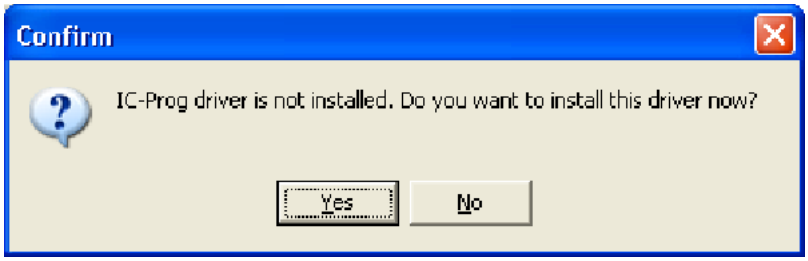
Chọn Enable Vcc control for JDM, sau đó mới chọn tiếp Enable NT/2000/XP Driver. Khi chọn Enable Driver xong, ngay lập tức sẽ có một màn hình Confirm hiện lên như trong hình bên dưới nhấn Yes để cài đặt.

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.

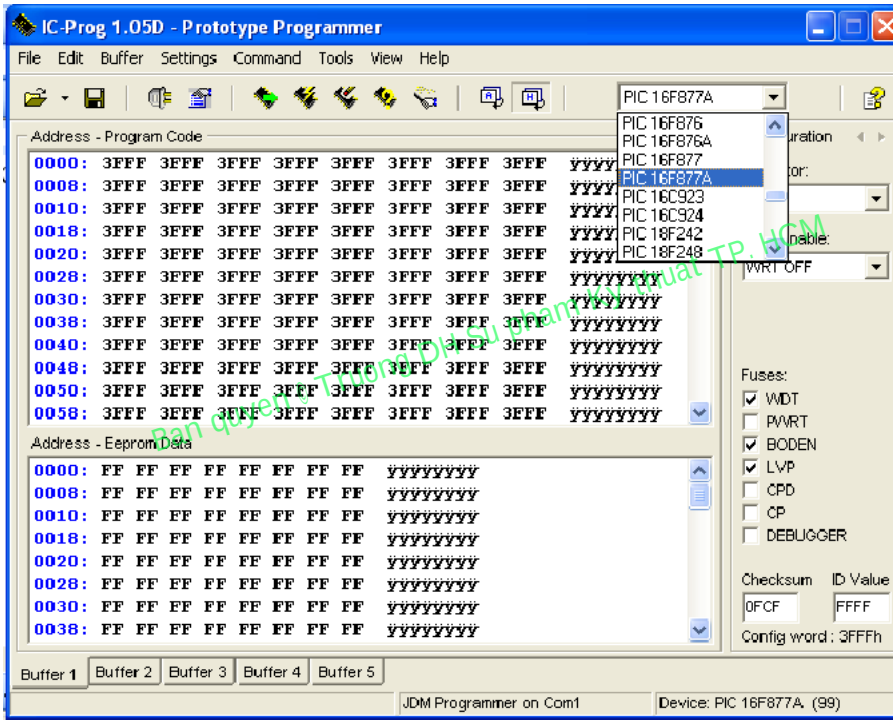


Hình 3-15. Cài đặt Driver.

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.

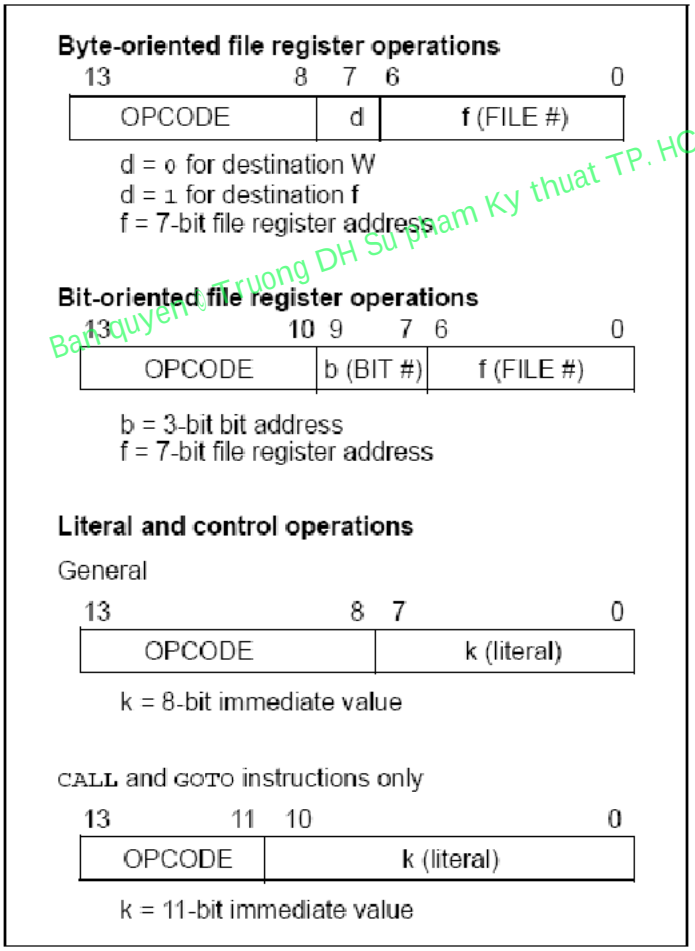
c. Quy ước kí hiệu trong MPLAB:

Bảng 3-1: Kí hiệu quy ước cấu trúc lệnh

Kí hiệu	Chức năng
f	Địa chỉ của file thanh ghi từ 0×00 đến 0×7F
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.

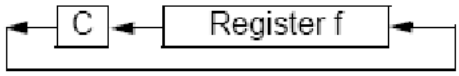
y. Lệnh: **RETLW** lệnh trở về từ chương trình con phục vụ ngắt.

- 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.

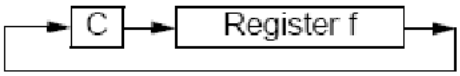
Sau khi thực hiện lệnh thì: FSR=0×C2, nội dung của địa chỉ (FSR) = 0111 0101 và C = 0.

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.

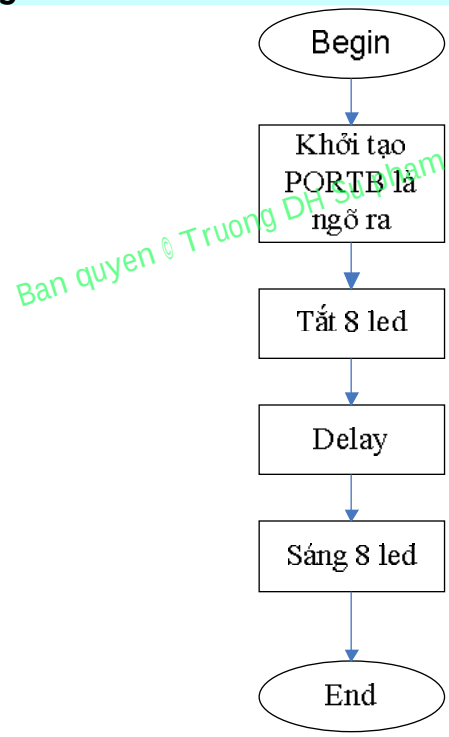
Lưu đồ chớp tắt PORTB (led đơn)

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
*=================================================*/
```

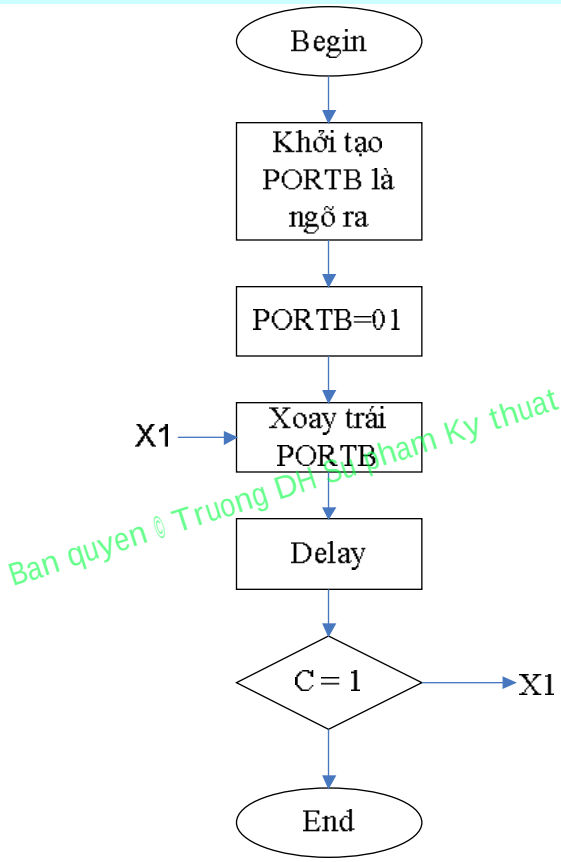
```
#include<16F877A.h>
#include<def_16f877a.h>
#fuses NOWDT,PUT,HS,NOPROTECT,NOLVP
#use delay(clock=20000000)
#use fast_io(b)
main()
  {
    trisb=0;
    while(true)
      {
                portb=0xff
                delay_ms(500);
                portb=0;
                delay_ms(500);
      }
  }
```

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:

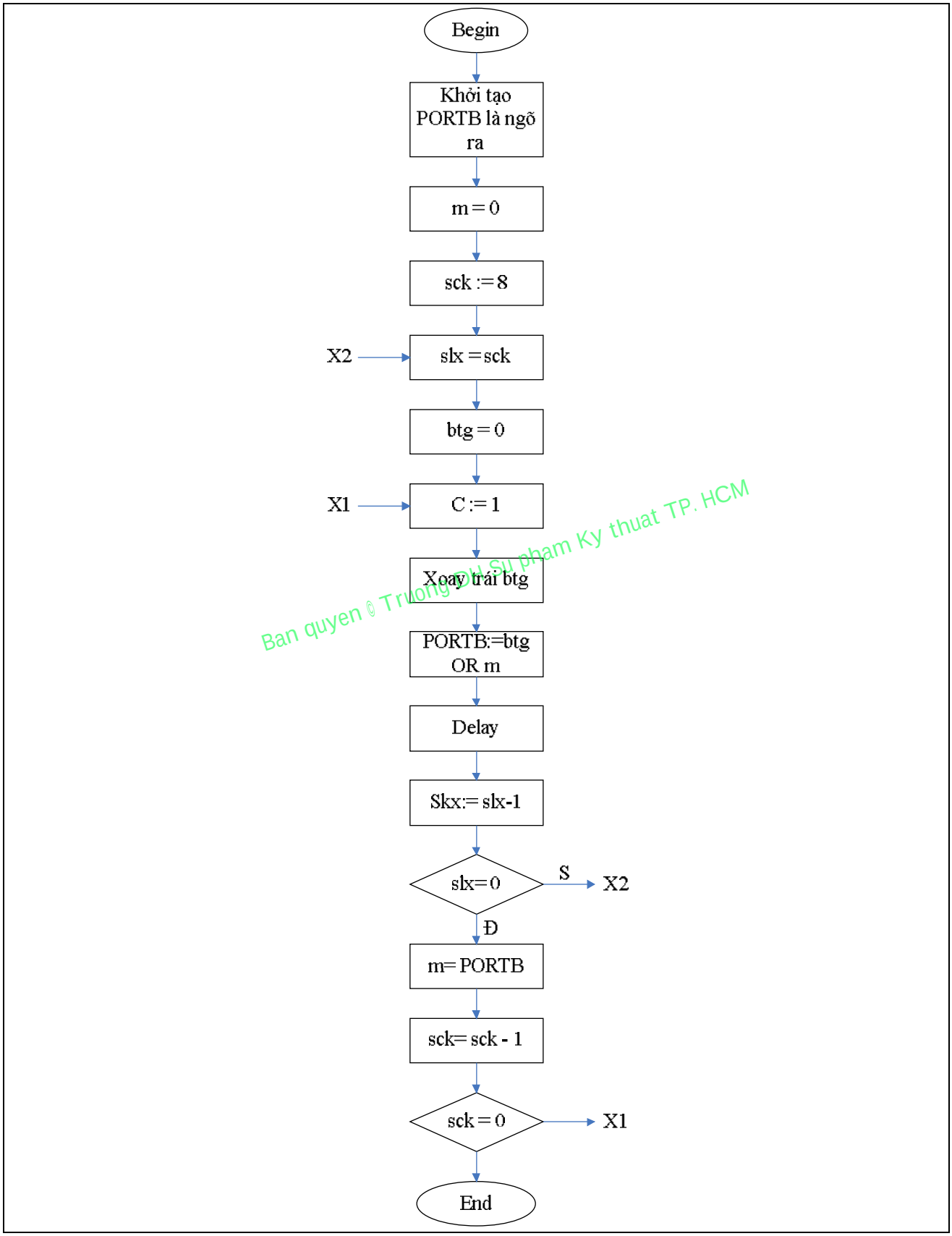

```
title       "sang don_Port_B.asm"
processor  p16f877a
include     <P16f877a.inc>
__CONFIG            _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_ON &  _XT_OSC
& _WRT_OFF & _LVP_OFF & _CPD_OFF
;-----------------------------------------------------------------
;Dat bien
;-----------------------------------------------------------------
            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:

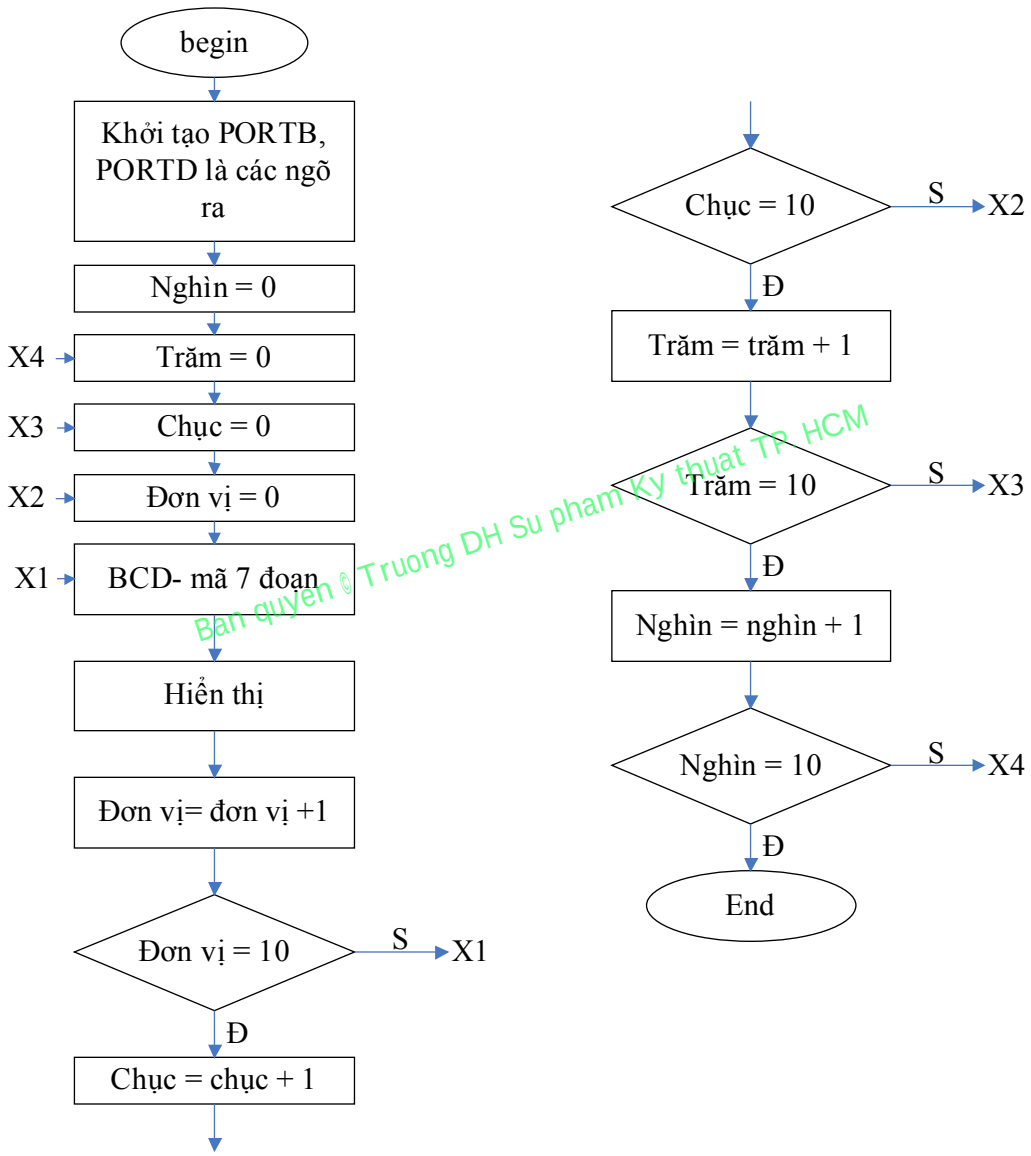
```
/*================================================
* Title          : Chuong trinh sang don portB
* Writer         :
* Processer      : PIC16F877A
* Compiler       : CCS C
*================================================*/
#include<16F877A.h>
#include<def_16f877a.h>
#fuses NOWDT,PUT,HS,NOPROTECT,NOLVP
#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;
```

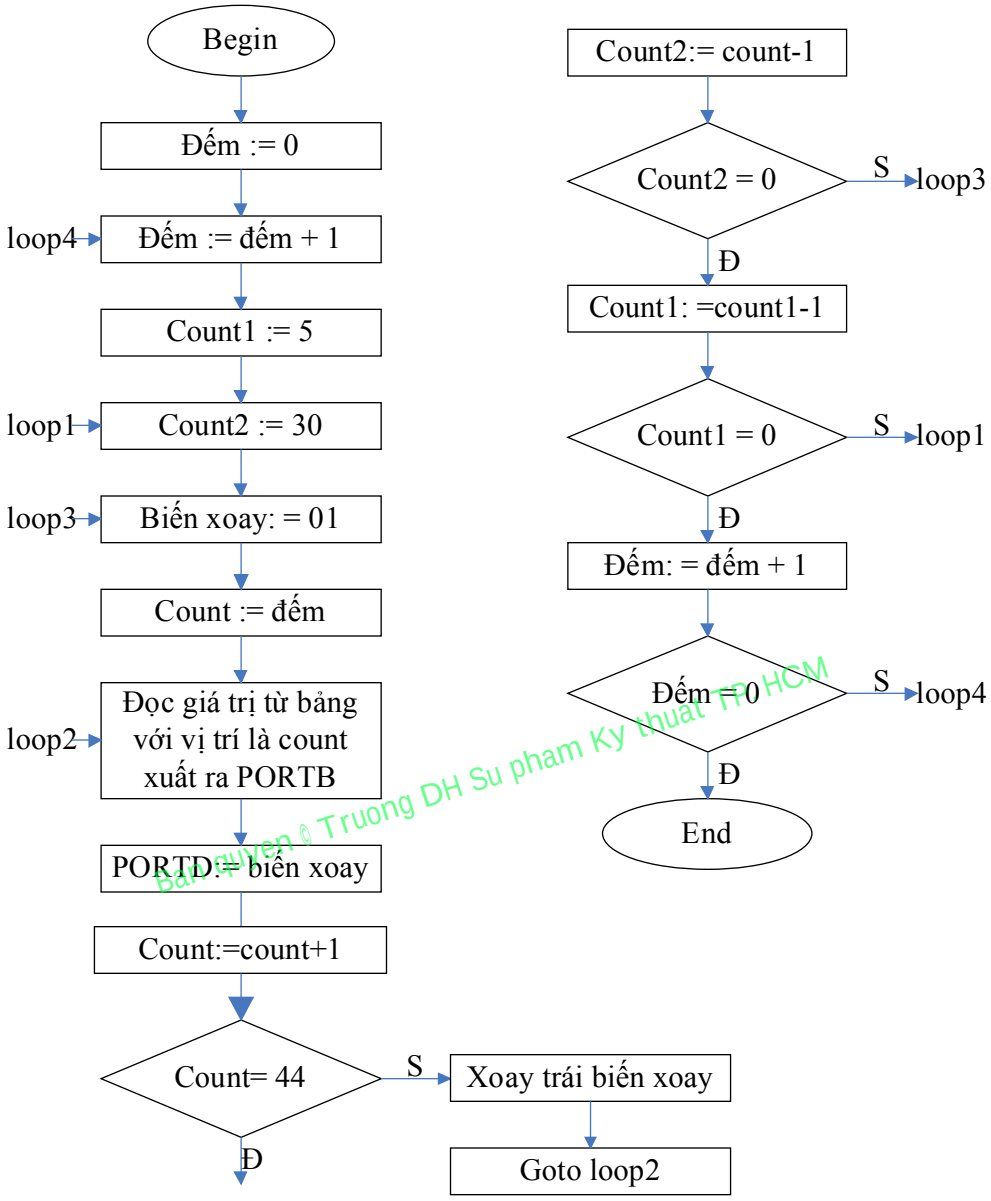
```
                delay_us(100);
                portb=dig[tramk];
                portd=0xdf;
                delay_us(100);
                i++;
        }
}
void main()
 {
                trisb=0x0;
                trisd=0x0;
                bien=0;
                while(1)
                {
                        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>
```

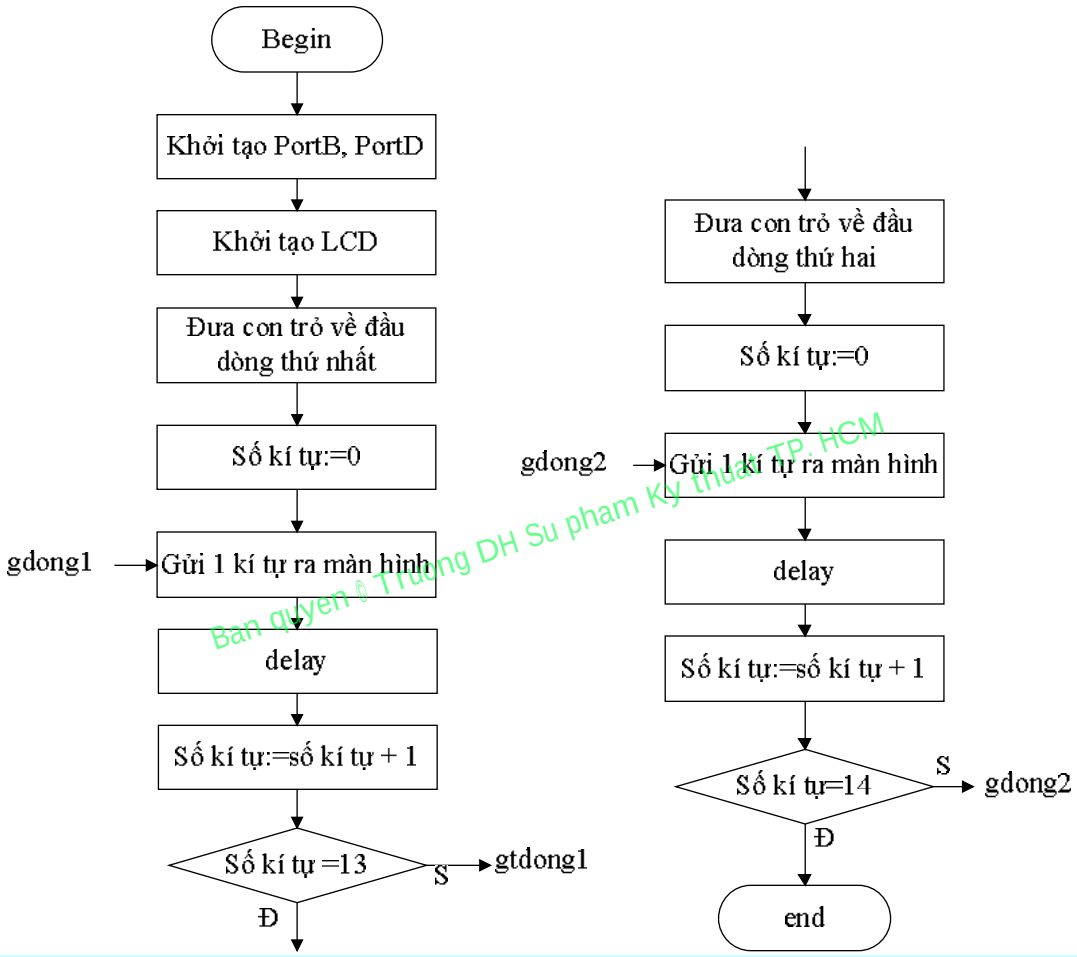
```
#fuses NOWDT,PUT,HS,NOPROTECT,NOLVP
#use delay(clock=20000000)
#use fast_io(b)
#use fast_io(d)
int8 dem,a,i,j;
int16 b;
const unsigned char dig[]={0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,\
0xcd,0xb6,0xb6,0xd9,0xff,0x80,0xb7,0xb7,0xcf,0xff,0x80,0xeb,0xdd,0xbe,\
0xff,0xbf,0xbf,0x80,0xbf,0xbf,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff};
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
;=================================================
; 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  Ban quyen © Truong DH Su pham Ky thuat TP. HCM"  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';
```

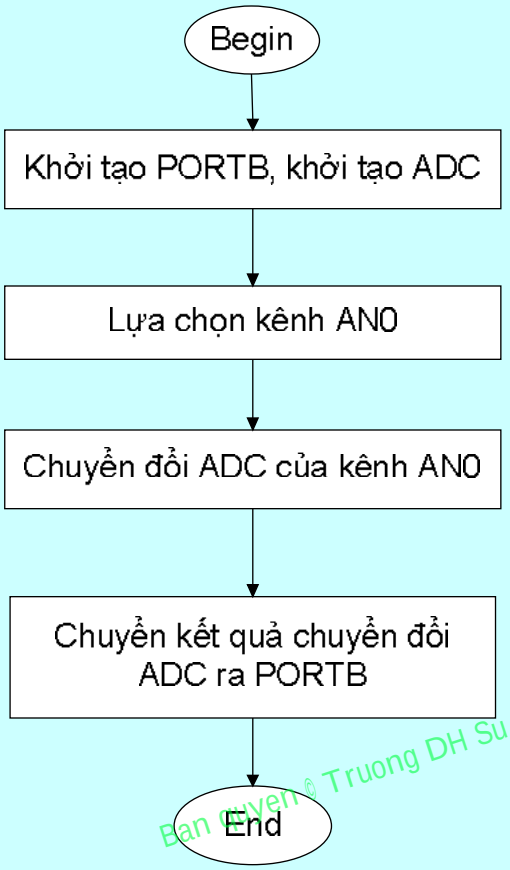
```
datawrt();
LCD = 'u';
datawrt();
LCD = 'o';
datawrt();
LCD = 'n';
datawrt();
LCD = 'g';
datawrt();
LCD = ' ';
datawrt();
LCD = 'D';
datawrt();
LCD = 'H';
datawrt();
LCD = 'S';
datawrt();
LCD = 'P';
datawrt();
LCD = 'K';
datawrt();
LCD = 'T';
datawrt();
LCD = 0xC0;             //Vi tri hang 2,cot 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).

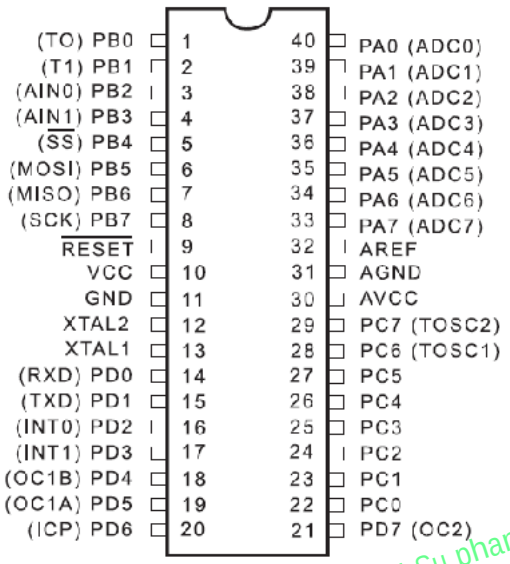
Công suất tiêu tán của chip AVR AT90S8535 được đo tại tần số hoạt động 4MHz, nguồn cung cấp 3V và nhiệt độ môi trường là 20 C:

- 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:

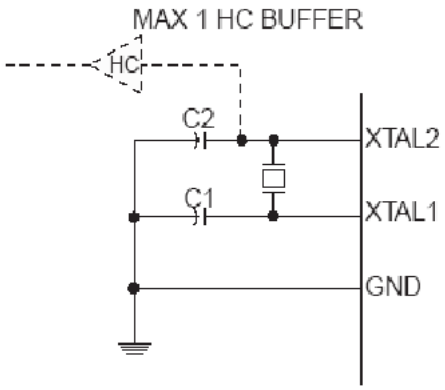
Là chân ngõ vào cung cấp điện áp tham chiếu (điện áp chuẩn) cho bộ chuyển đổi AD. Đối với hoạt động của ADC thì phải cung cấp một điện áp nằm trong khoảng từ AGND đến Avcc đến ngõ vào chân AREF.

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).

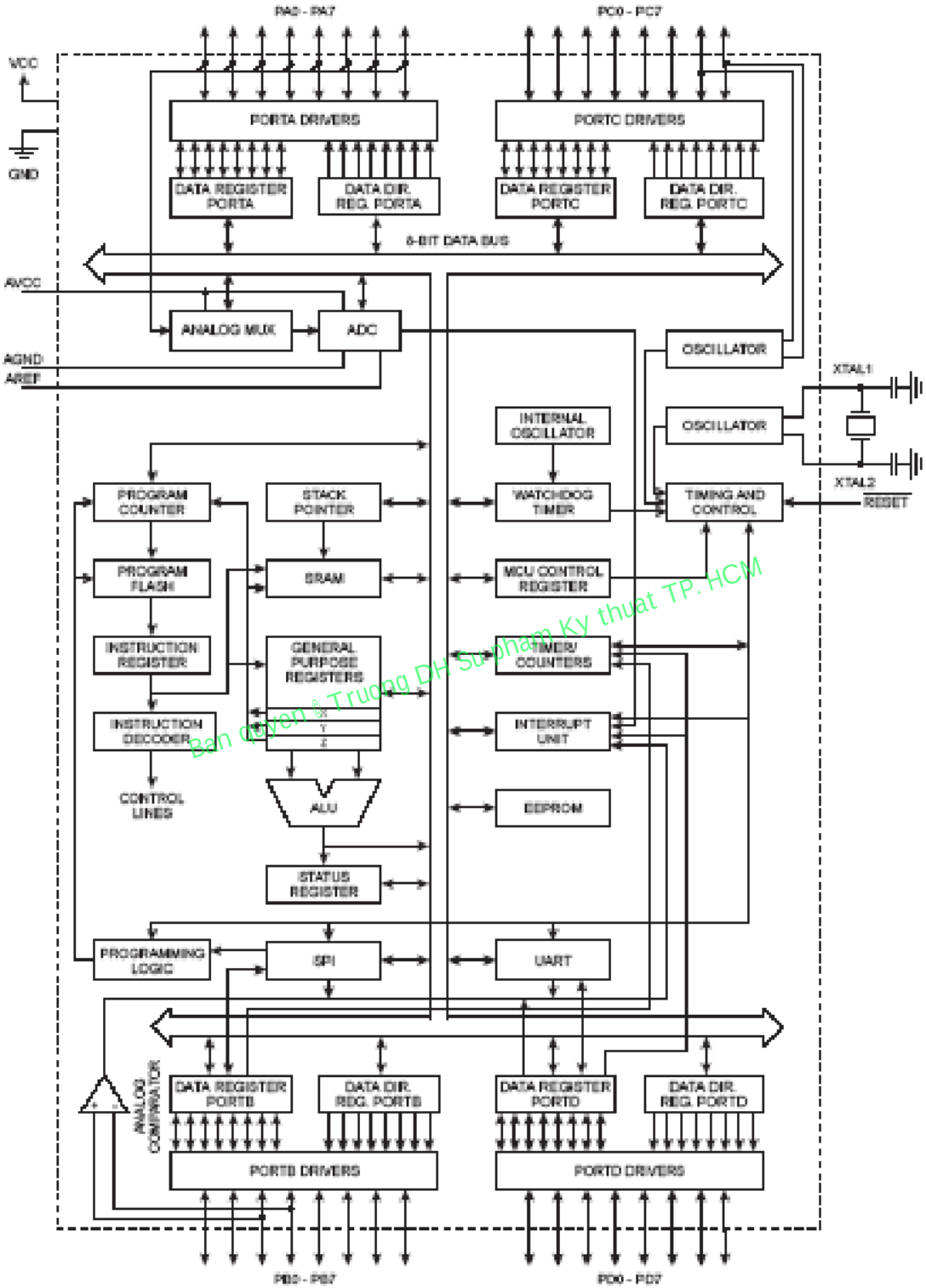
- Khối điều khiển lập trình (programming control): đây là khối quan trọng nhất của toàn bộ hệ thống.

- 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.

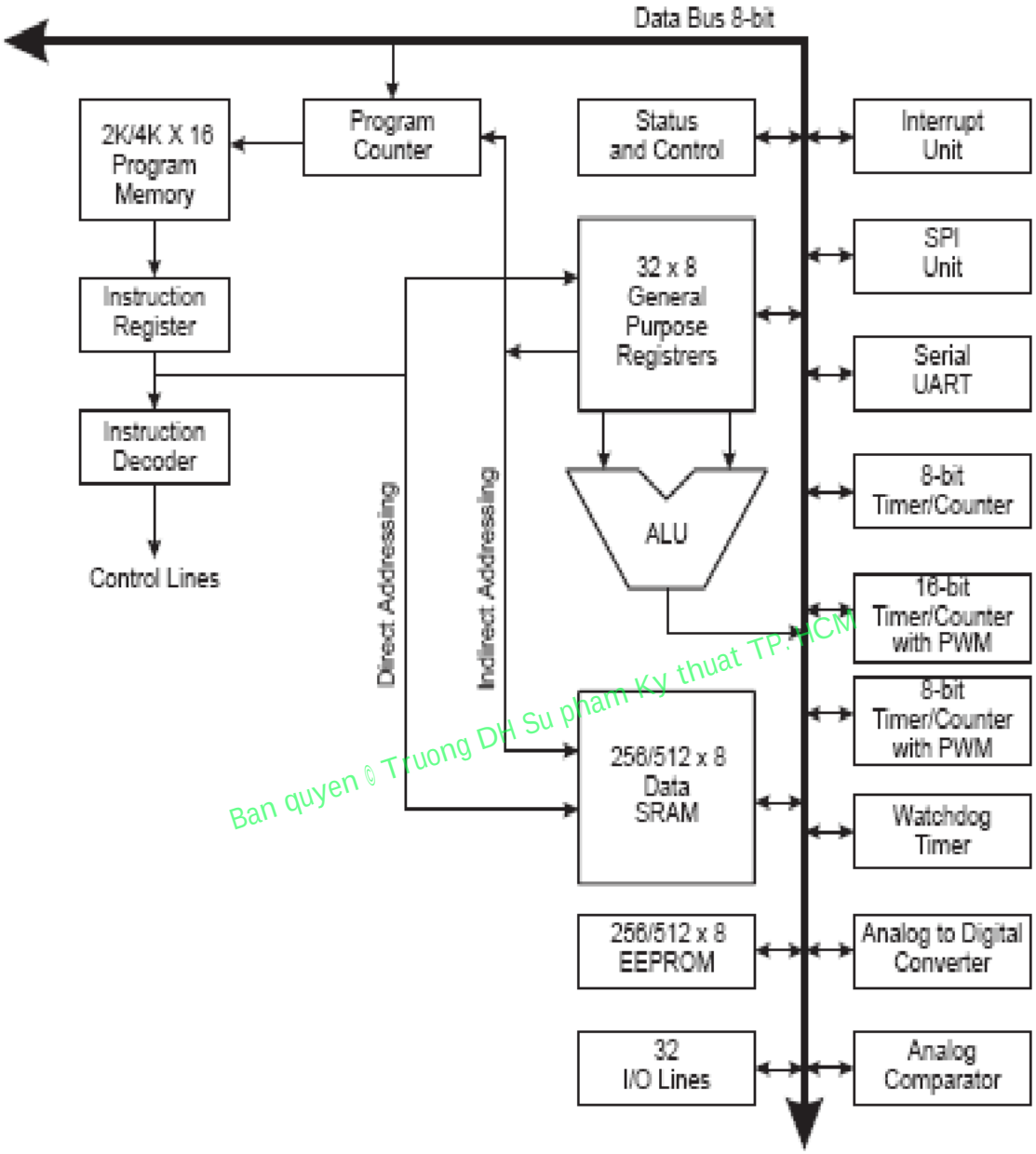
4. Tổng quan về cấu trúc AVR

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.

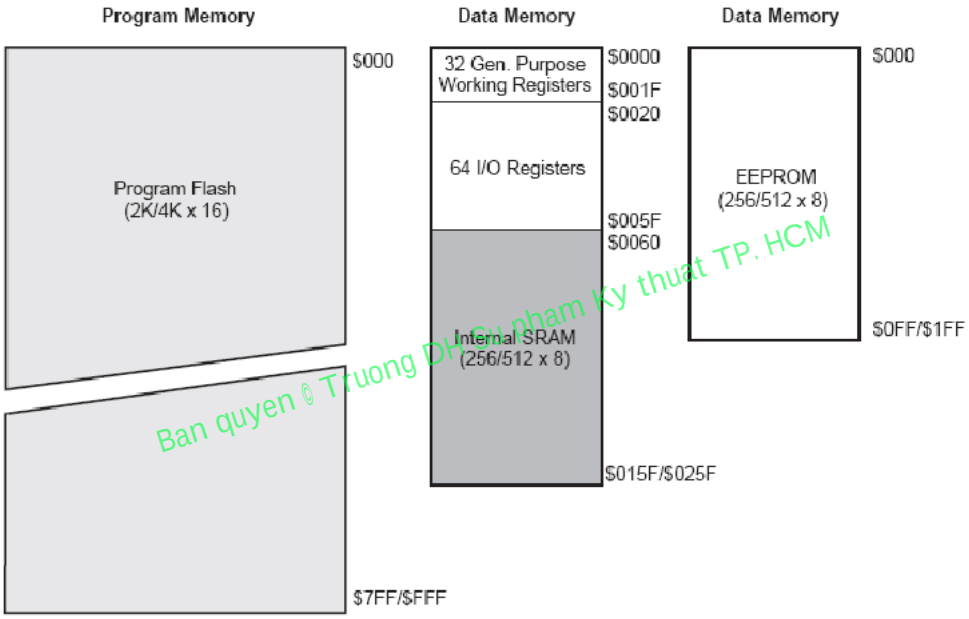
AVR dùng ý tưởng cấu trúc Harvard – với việc chia các vùng nhớ và bus cho bộ nhớ chương trình và bộ nhớ dữ liệu. Bộ nhớ chương trình được thực hiện với 2 tầng đường ống (two stage pipeline). Trong khi 1 lệnh được thực hiện thì lệnh tiếp theo sẽ được đón từ bộ nhớ chương trình. Ý tưởng này cho phép các 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.

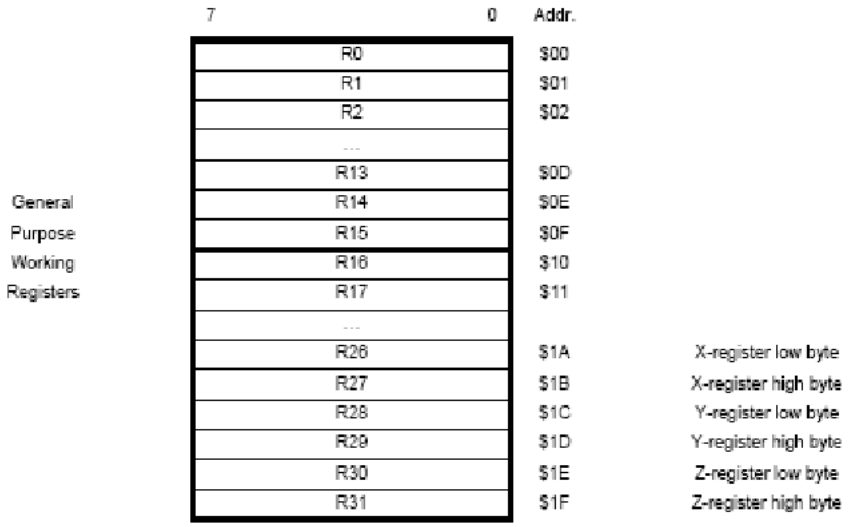
Bộ nhớ chương trình của AT90S8535 có dung lượng là 8Kbyte, mỗi một ô nhớ là 16 bit, còn các loại bộ nhớ dữ liệu thì mỗi một ô nhớ là 8 bit. Bảng đồ nhớ ở trên có 2 thông số thì thông số đứng trước là của AT90S4434, còn thông số đứng sau là của AT90S8535.

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.

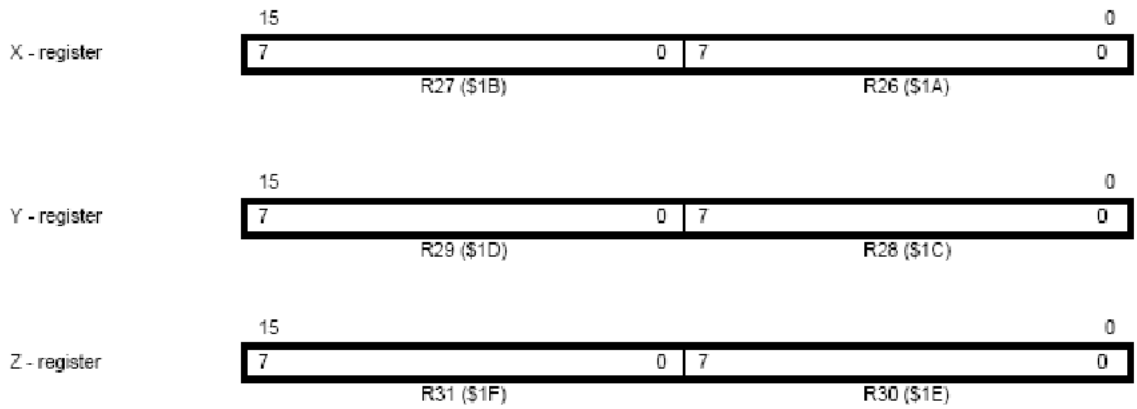
Tất cả các lệnh xử lý trong tập lệnh đều có thể truy xuất trực tiếp trên các thanh ghi này và chỉ thực hiện trong khoảng thời gian 1 chu kỳ máy - ngoại trừ 5 lệnh thực hiện các phép toán đại số và logic: SBCI, SUBI, CPI, ANDI, ORI xảy ra giữa hằng số và thanh ghi và lệnh LDI – nạp dữ liệu hằng số trực 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:

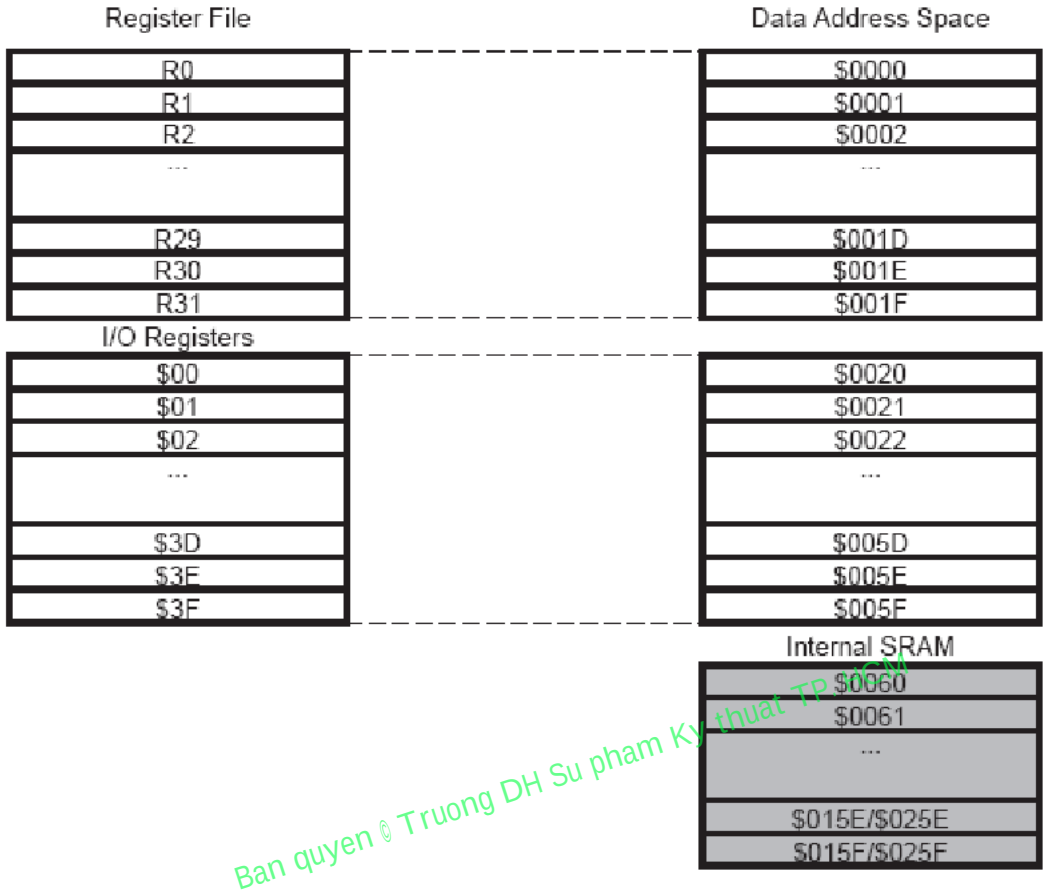
Khối ALU trong AVR thực hiện kết nối trực tiếp với tất cả 32 thanh ghi hoạt động đa năng tổng quát. Trong một chu kỳ duy nhất của xung clock, các phép toán của khối ALU xảy ra giữa các thanh ghi trong file thanh ghi được thực hiện. Các phép toán của khối ALU được chia ra làm 3 loại: phép toán số học, 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.

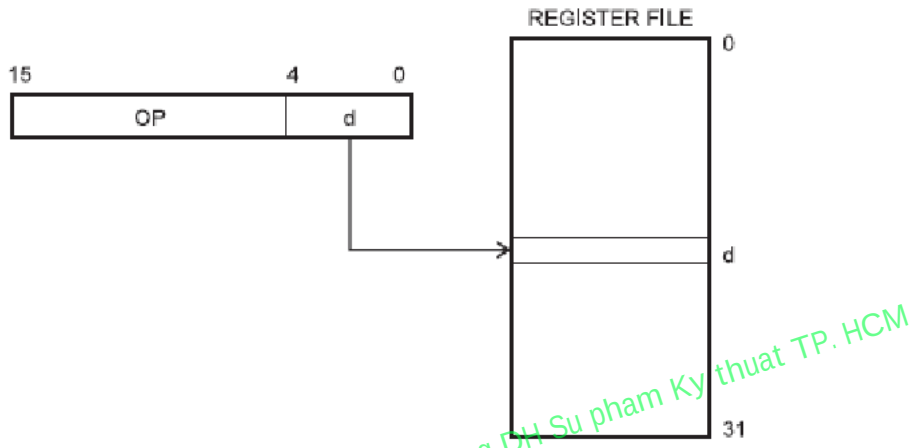
Các kiểu truy xuất bộ nhớ dữ liệu và bộ nhớ chương trình:

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.

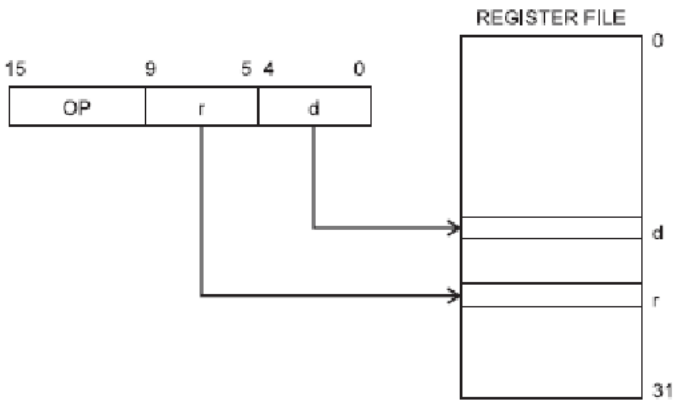


Hình 9. Kiểu định địa chỉ trực tiếp dùng thanh ghi và 2 thanh ghi.

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.

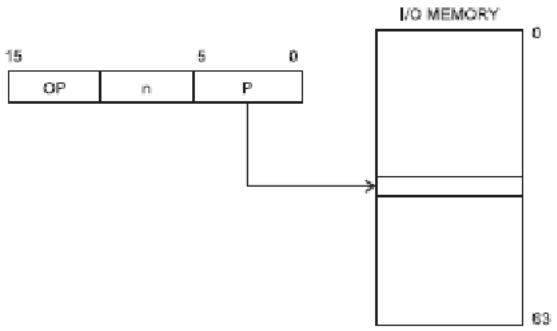


Hình 10. Kiểu định địa chỉ trực tiếp dùng thanh ghi và 2 thanh ghi.

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.

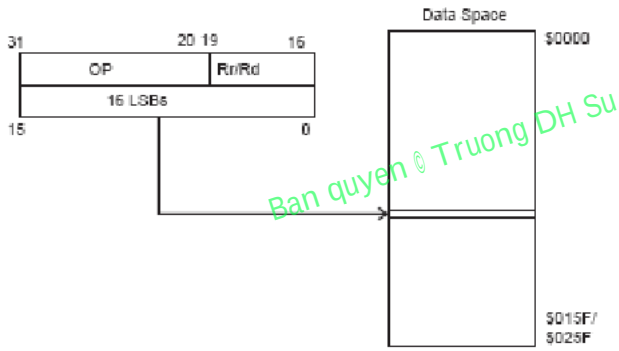


Hình 11. Kiểu định địa chỉ trực tiếp IO.

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.

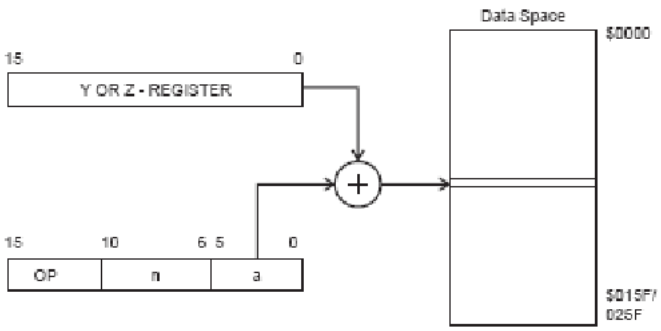


Hình 12. Kiểu định địa chỉ trực tiếp dữ liệu.

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.

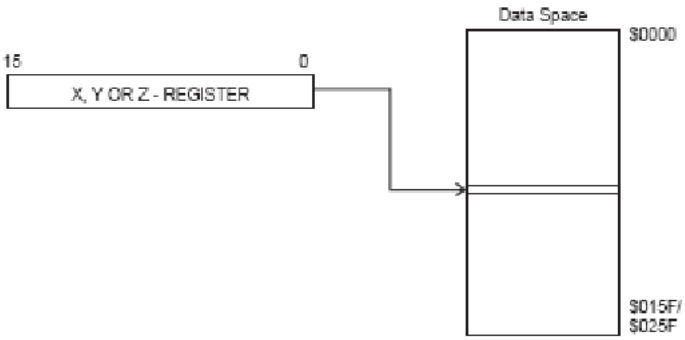


Hình 13. Kiểu định địa chỉ gián tiếp dữ liệu với displacement.

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.

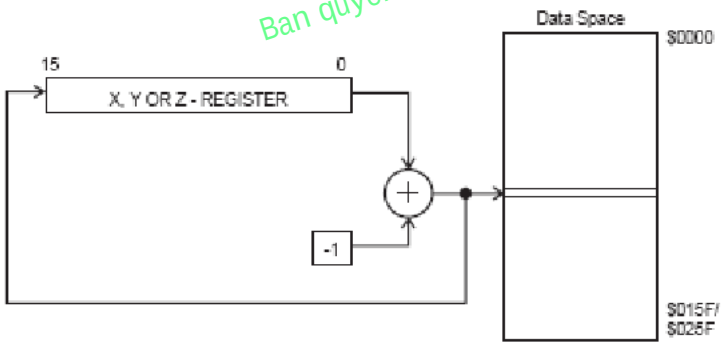


Hình 14. Kiểu định địa chỉ gián tiếp dữ liệu.

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.



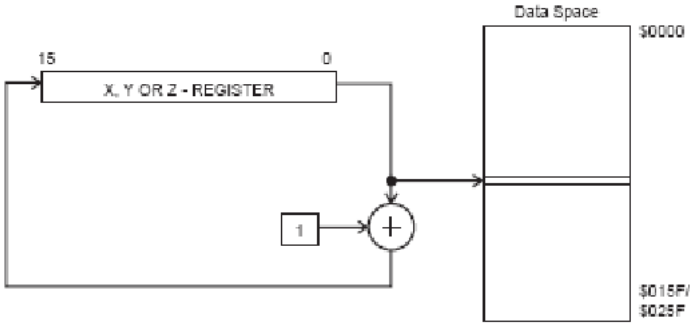
Hình 15. Kiểu định địa chỉ gián tiếp dữ liệu với pre - displacement.

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.



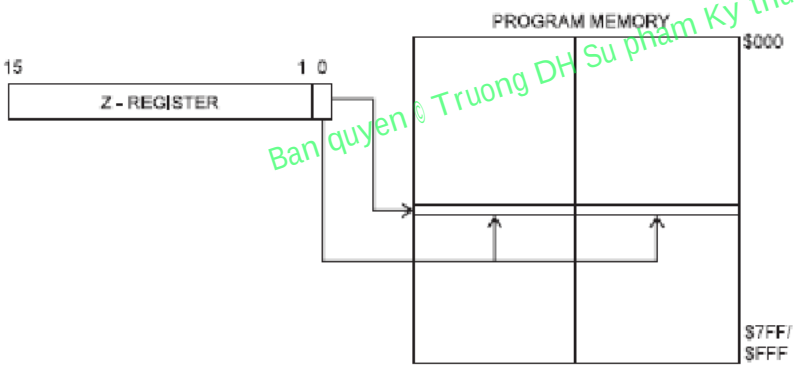
Hình 16. Kiểu định địa chỉ gián tiếp dữ liệu với Post - Increment.

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.

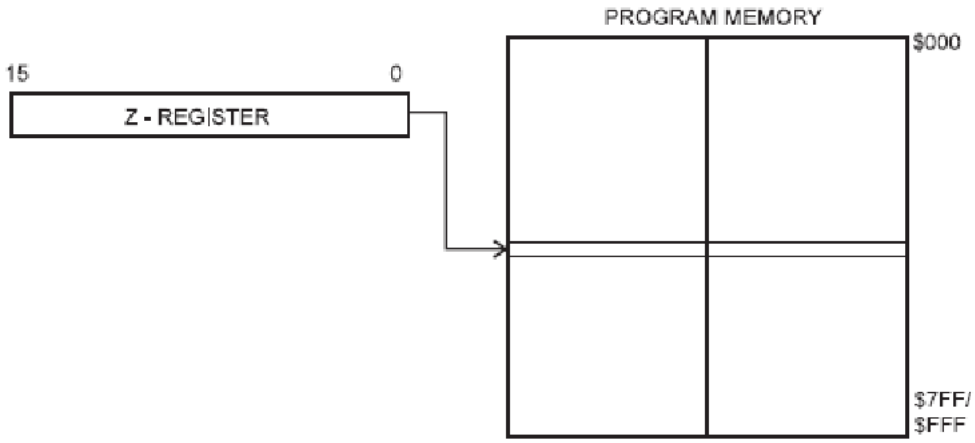


Hình 17. Kiểu định địa chỉ dùng lệnh LPM.

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.

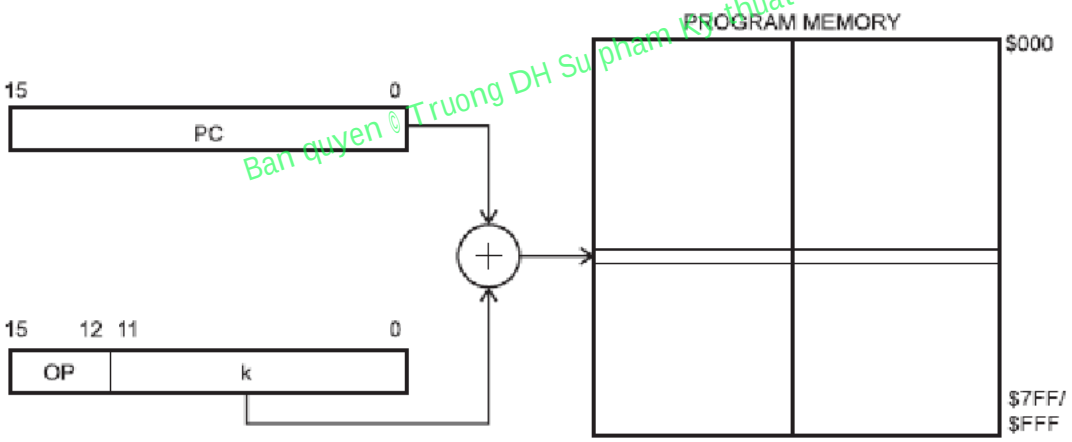


Hình 18. Kiểu định địa chỉ gián tiếp bộ nhớ chương trình.

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:

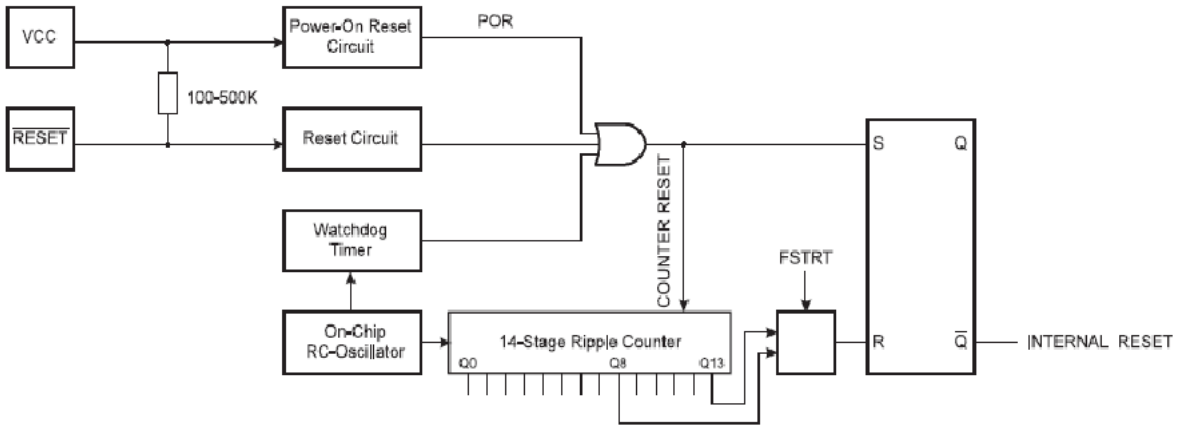
```
Address          Labels     Code          Comments
$000                        rjmp   RESET      ; Reset Handler
$001                        rjmp   EXT_INT0   ; IRQ0 Handler
$002                        rjmp   EXT_INT1   ; IRQ1 Handler
$003                        rjmp   TIM2_COMP  ; Timer2 Compare Handler
$004                        rjmp   TIM2_OVF   ; 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_OVF   ; Timer1 Overflow Handler
$009                        rjmp   TIM0_OVF   ; 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}	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

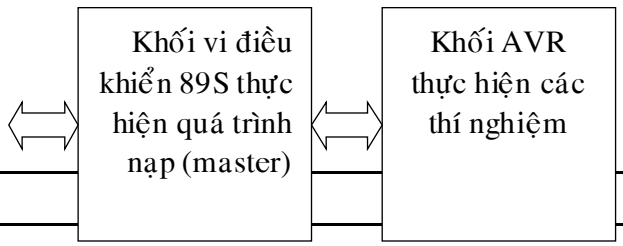
Bảng 5. Tóm tắt tập lệnh (tiếp theo).

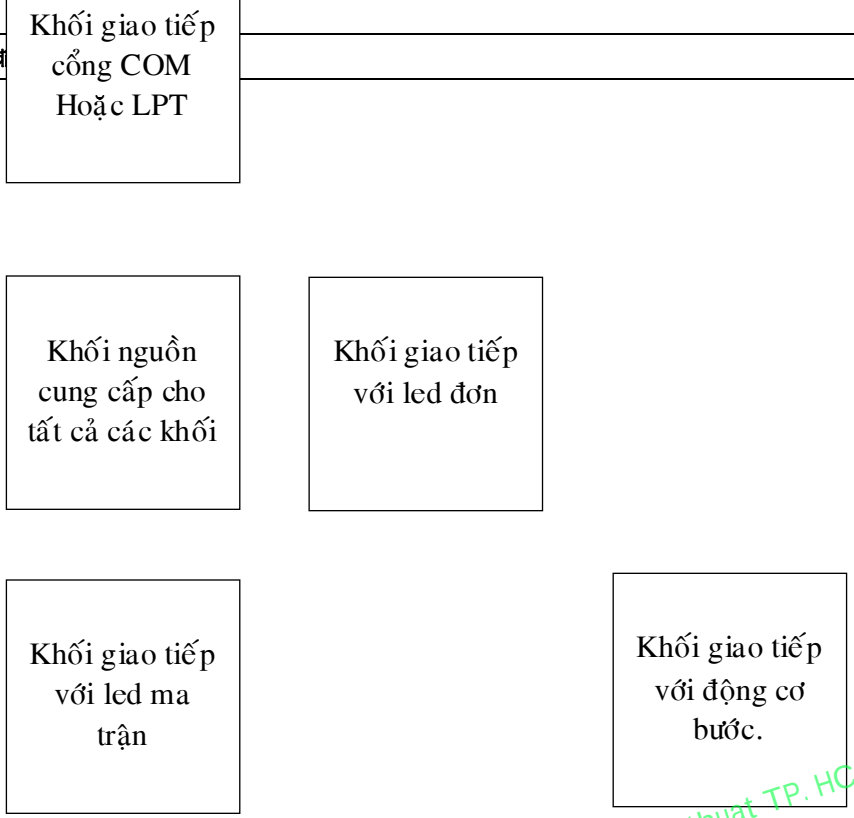
6. Thiết kế phần cứng hệ thống:

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.

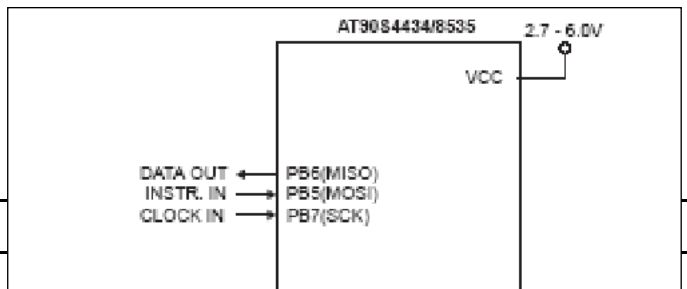
Chân PB.5 có tên là MOSI: là đường nhập dữ liệu vào nối tiếp của vi điều khiển nạp.

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.



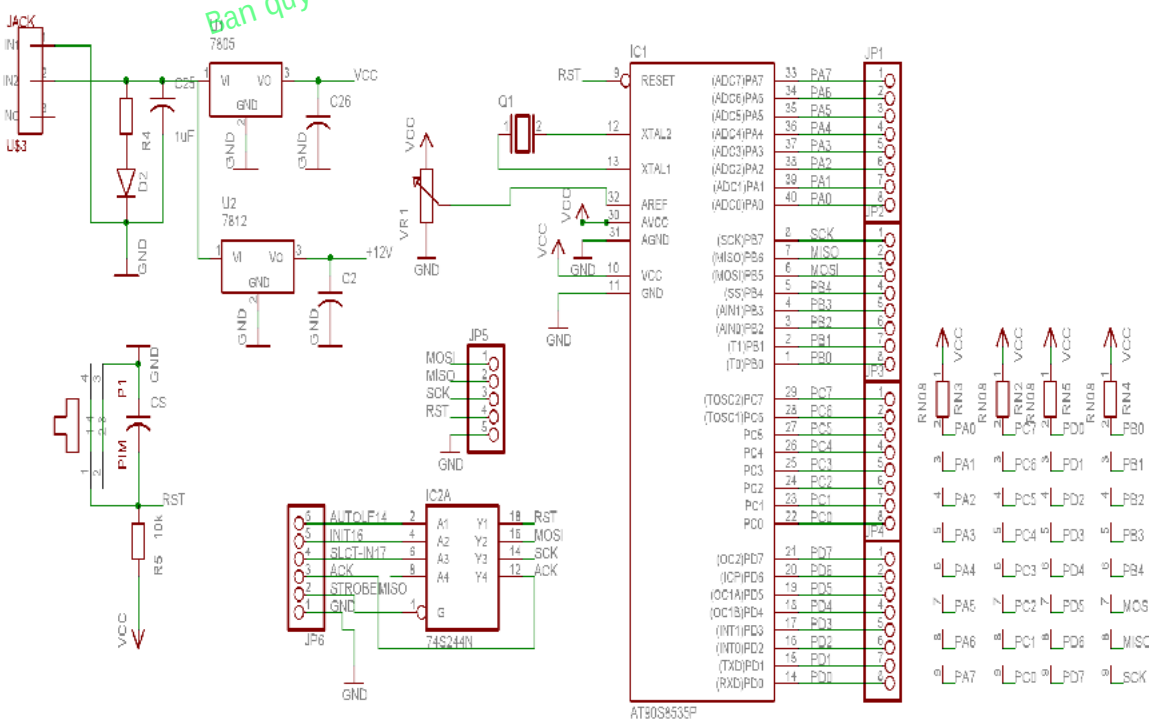
Hình 22. Sơ đồ giao tiếp mạch nạp.

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	*12*xx xxxP	Read Lock and Fuse bits. '0' = programmed, '1' = unprogrammed.
Write Lock Bits	1010 1100	1111 1*21*1	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 111F	xxxx xxxx	xxxx xxxx	Write FSTRT fuse. Set bit F = '0' to program, '1' to unprogram.

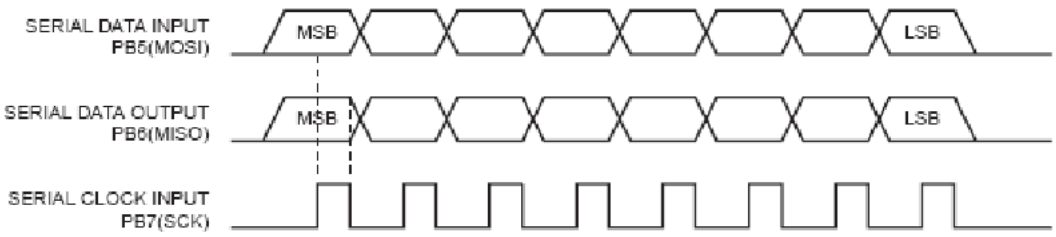
Note: a = address high bits
b = address low bits
H = 0 - Low byte, 1 - High Byte
o = data out
i = data in
x = don't care
1 = Lock Bit 1
2 = Lock Bit 2
F = FSTRT Fuse
S = SPIEN Fuse

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.

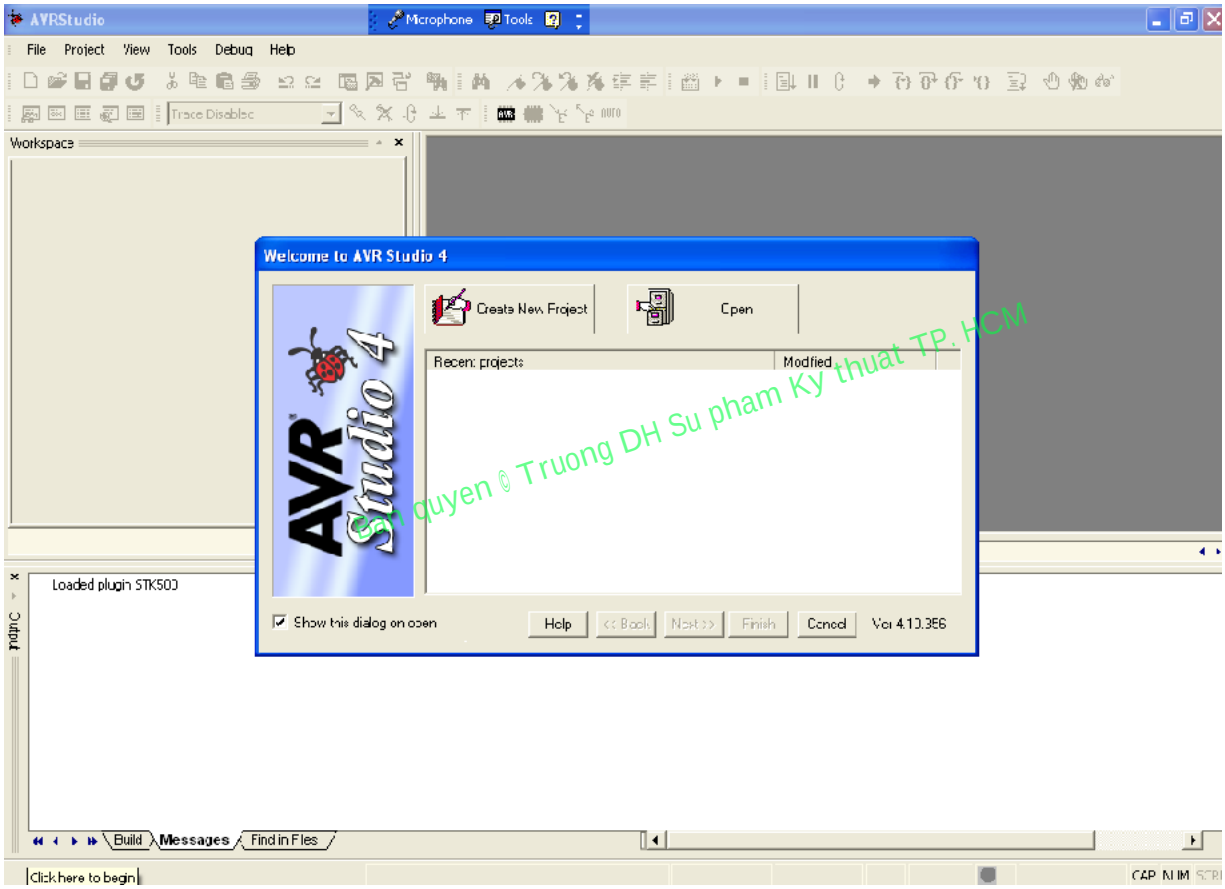
7. Hướng dẫn sử dụng phần mềm:

- 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:

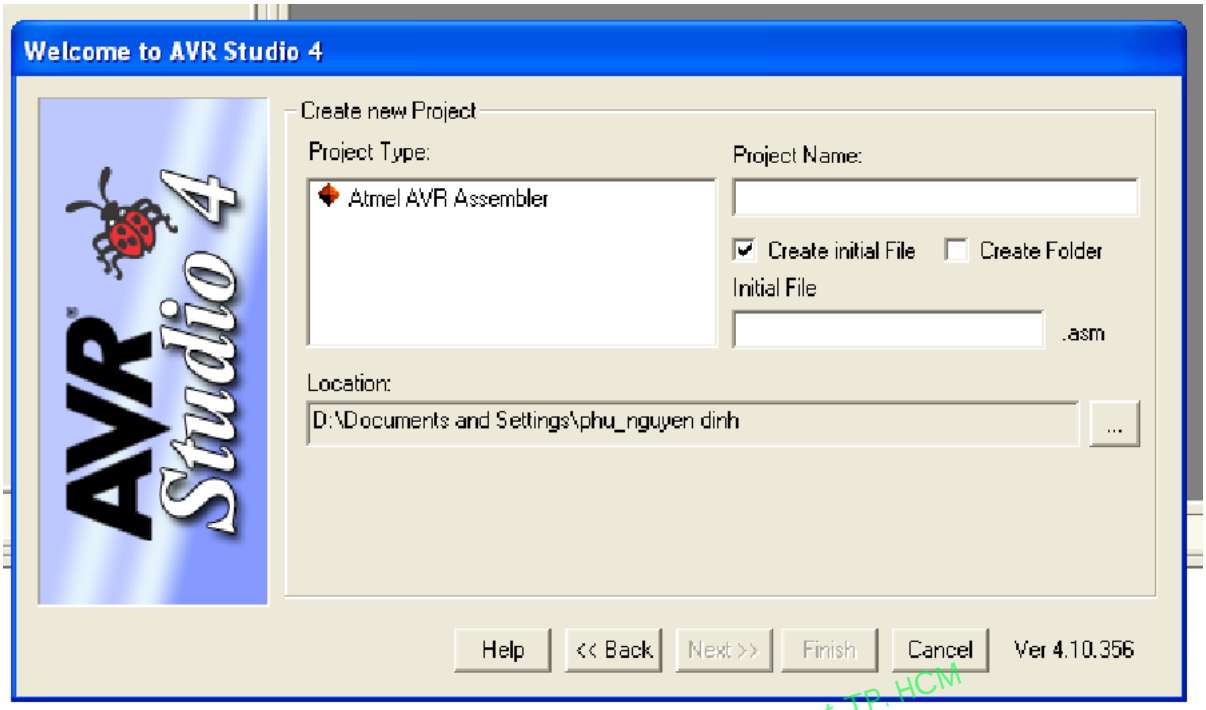


Hình 33. Màn hình soạn thảo của chương trình AVRStudio.

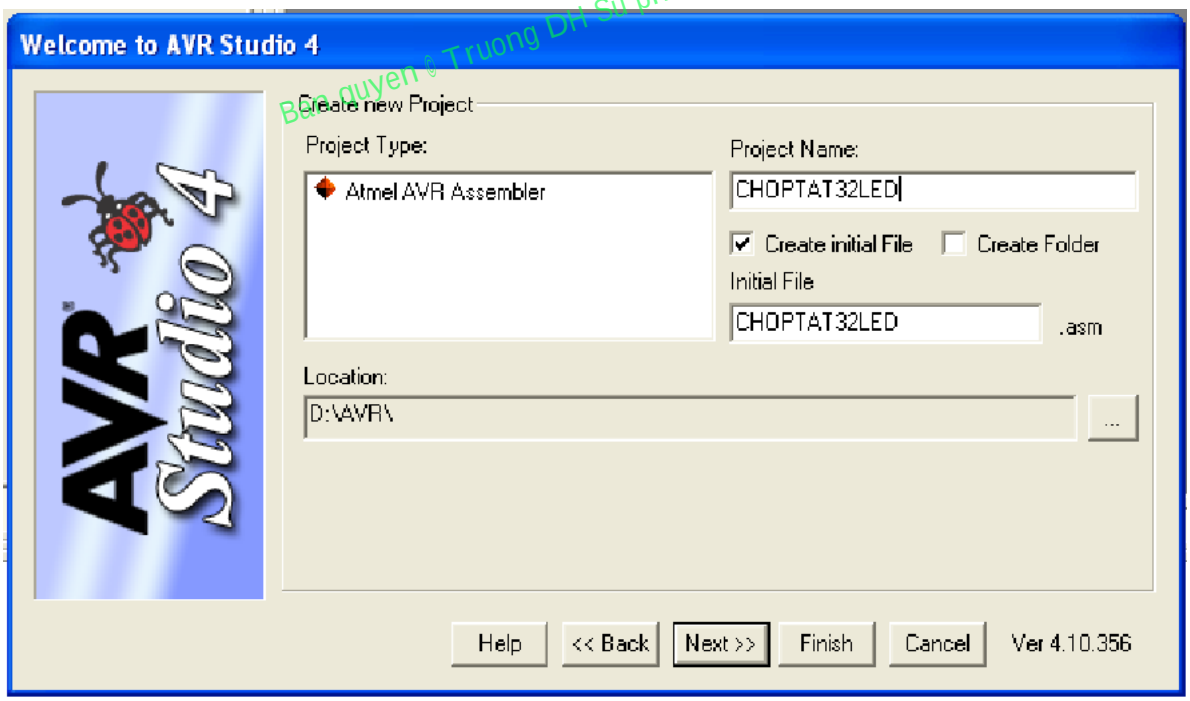
Một của sổ menu nhỏ xuất hiện cho phép bạn chọn project mới hay mở một project có sẵn.

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.

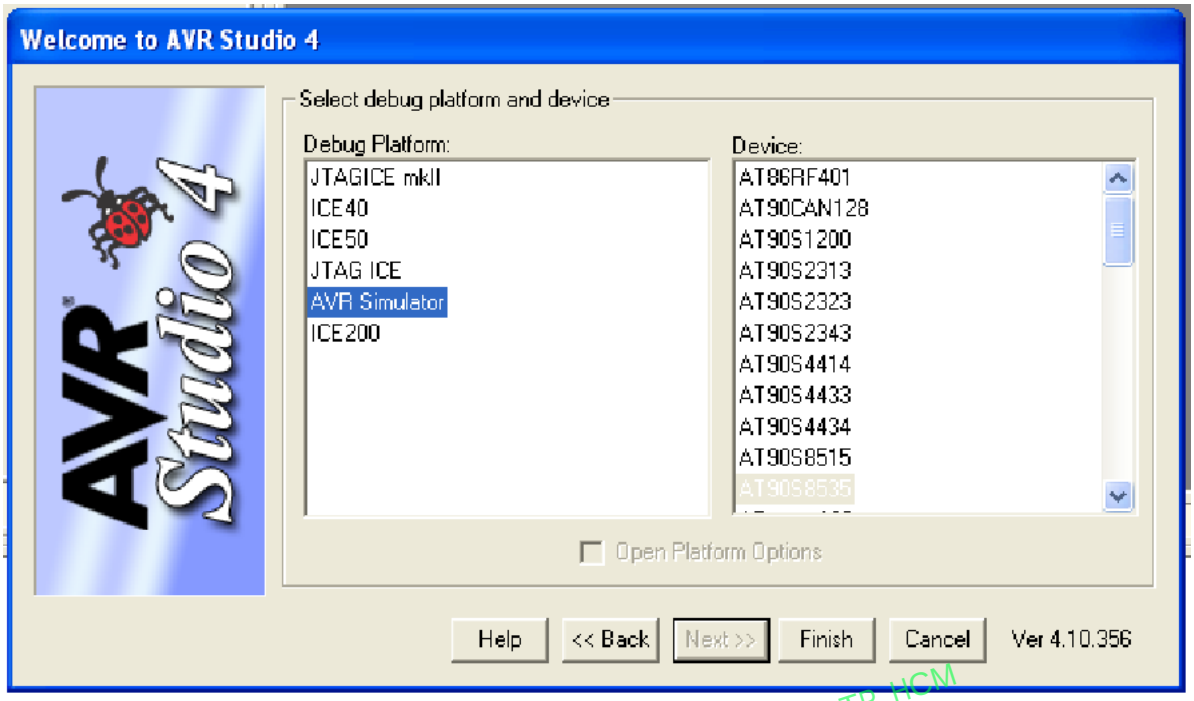


Hình 34. Màn hình soạn thảo project mới của chương trình AVRStudio.

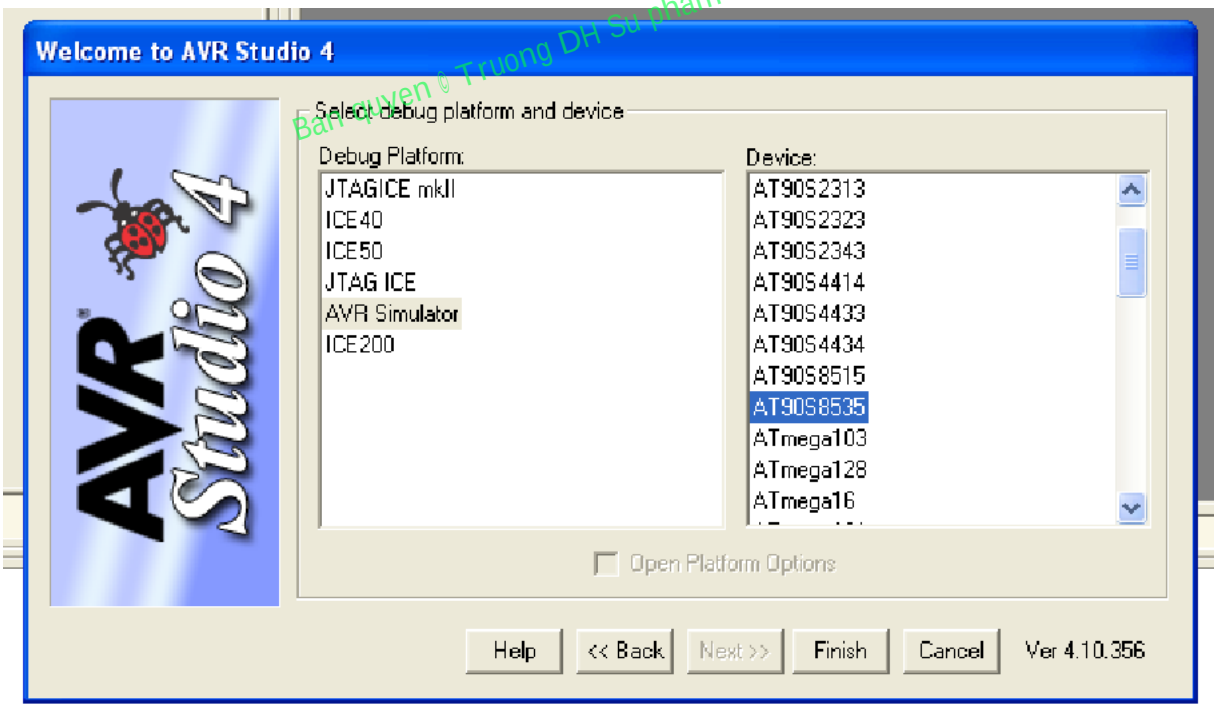


Hình 35. Màn hình nhập tên và thư mục của project mới.

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".

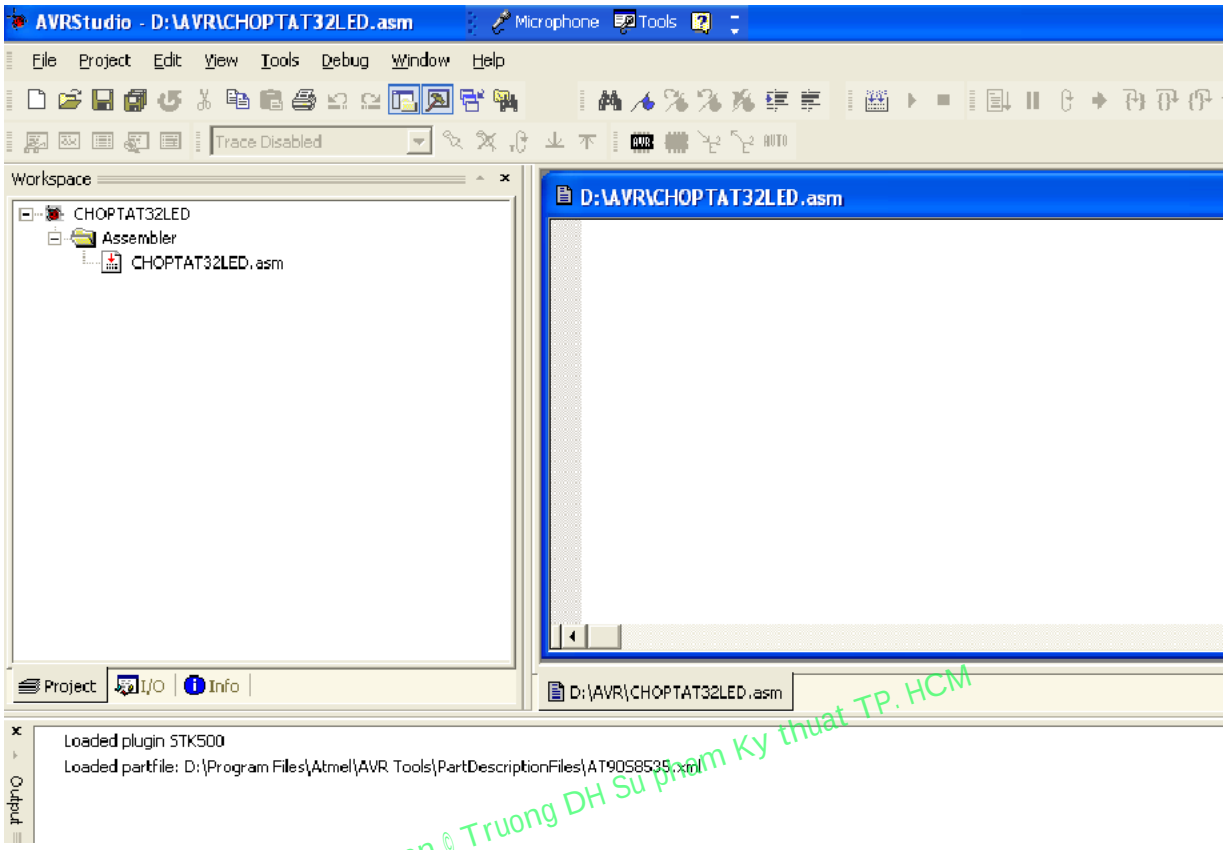


Hình 36. Màn hình chọn mô phỏng của project mới.



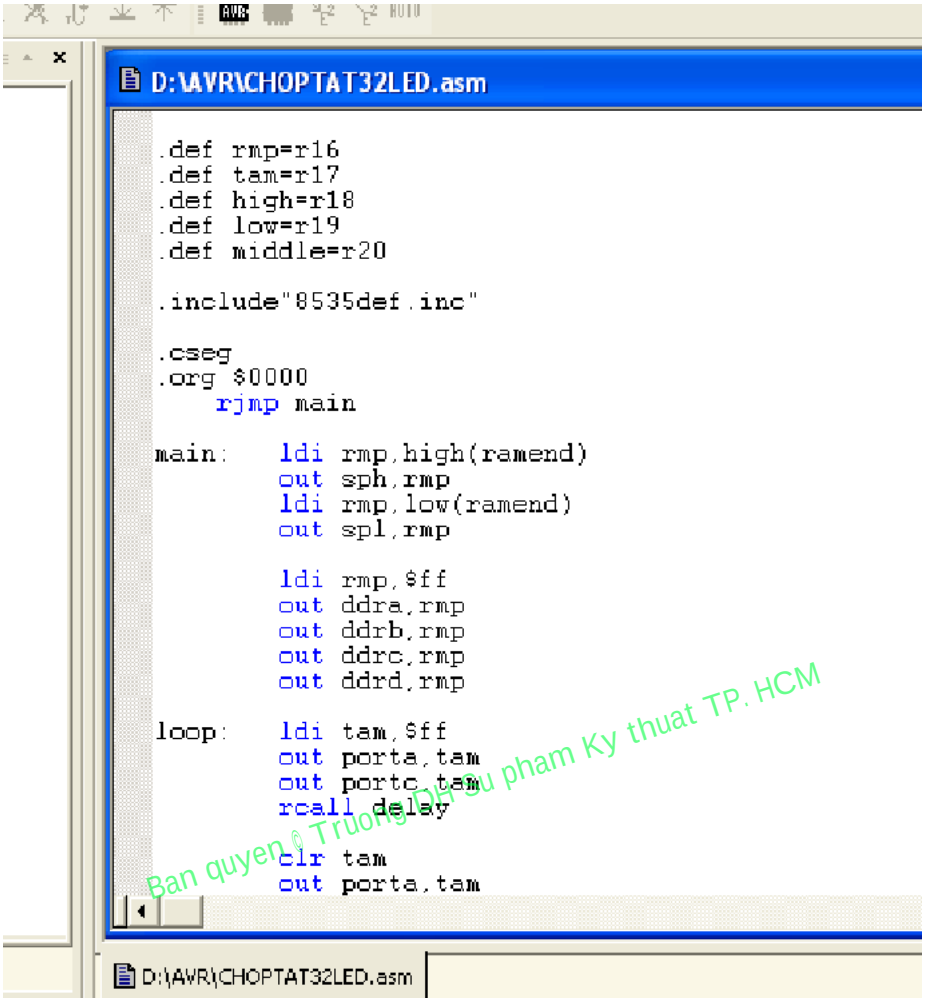
Hình 37. Màn hình chọn IC mô phỏng của project mới.

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.

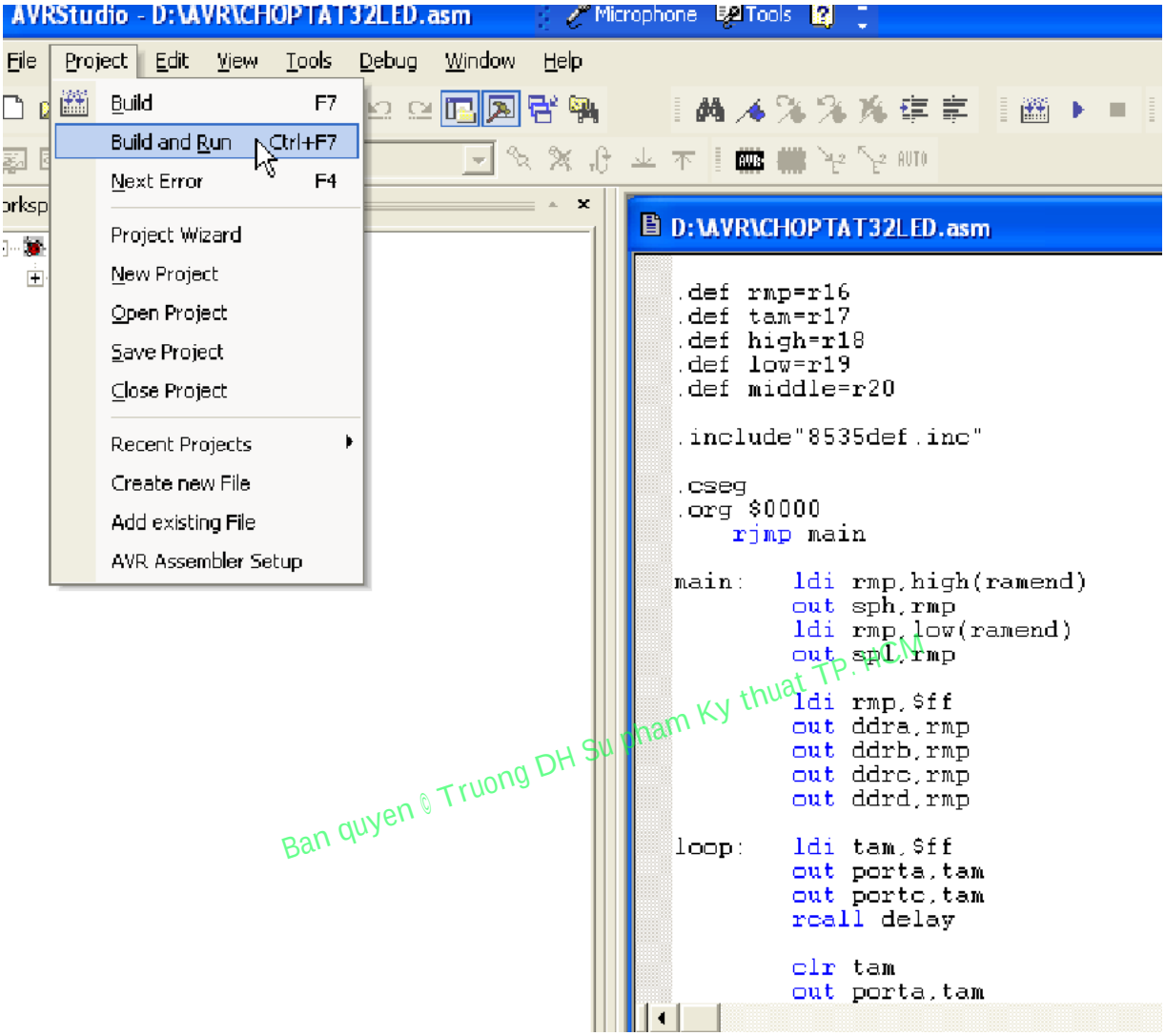


Hình 39. Màn hình soạn thảo chương trình chớp tắt 32 led.

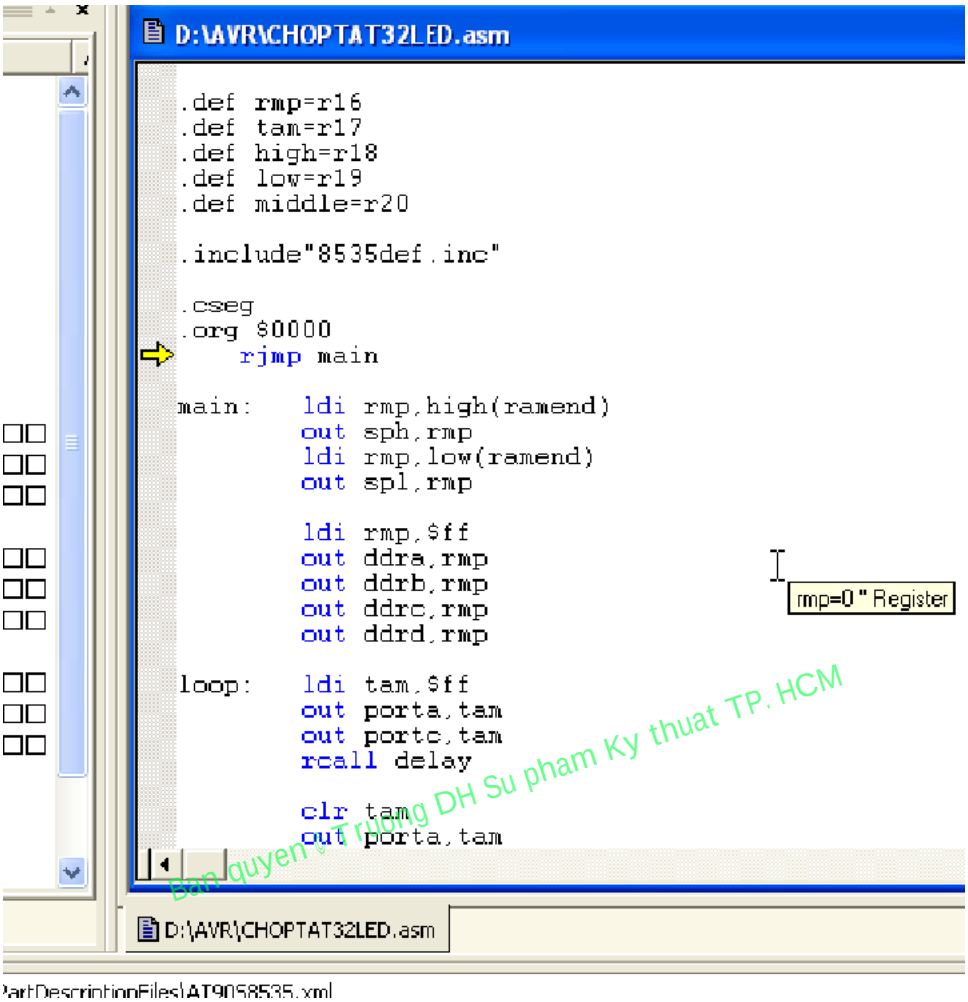
Tiến hành biên dịch bằng cách vào menu lệnh project và chọn vào mục như hình 40 thì khi đó chương trình sẽ được biên dịch.

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.



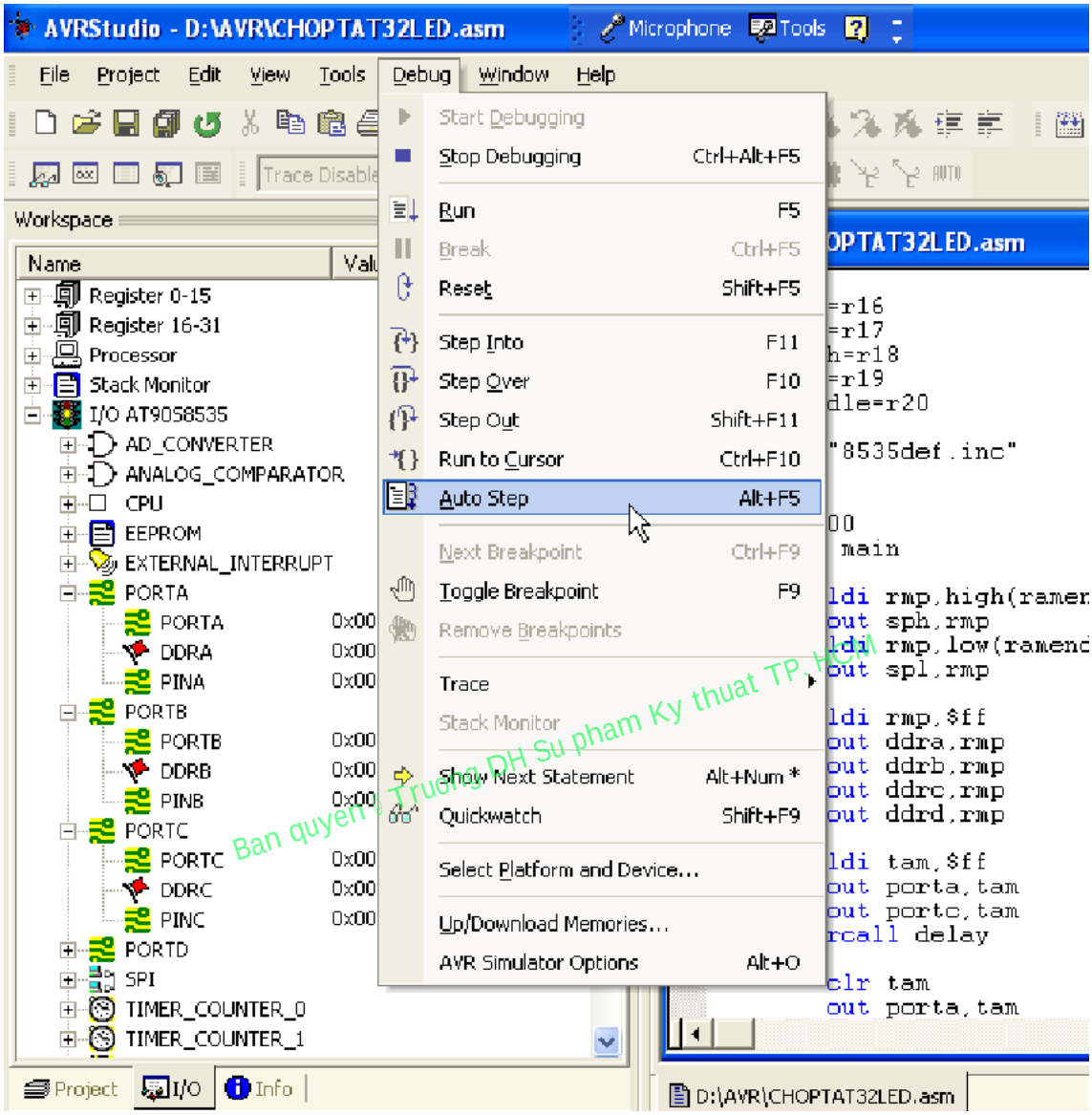
Hình 40. Menu lệnh biên dịch chương trình chóp tắt 32 led.



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**