Hướng dẫn lập trình Assembly
Cho AVR
sử dụng AVRstudio 4.2

Hướng dẫn lập trình AVR với nội dung hướng dẫn các bạn mới làm quen với vi điều khiển AVR và phần mềm AVRstudio4.2 . Nội dung chính của tài liệu này là . Hướng dẫn các bạn cách để viết một chương trình assembly và điều khiển vào ra dữ liệu.

Tài liệu này được chia làm 3 phần:
Phần 1: Các chỉ thị hợp dịch trong ASSEMBLY.
Phần 2:Viết mã lệnh cho một chương trình ASSEMBLY.
Phần3: Điều khiển vào ra dữ liệu và các thiết bị tích hợp trong AVR.

Phần 1: Các chỉ thị hợp dịch.

Chương trình dịch Assembly làm việc trên file chương trình nguồn và một file nguồn bao gồm : các lệnh , các nhãn và các chỉ dẫn.Chúng được xếp tuần tự trong file nguồn.

Một dòng lệnh có chiều dài cực đại là :120 kí tự.
Mọi dòng lệnh đều có thể đặt trước bởi một nhãn,nó là một chuỗi kí tự và kết thúc bằng dấu 2 chấm.Nhãn được sẻ dụng như là đích cho các lệnh nhảy, Và các chỉ thị rẽ nhánh.Và còn được sử dụng như là tên biến trong bộ nhớ chương trình và bộ nhớ dữ liệu.

Một dòng lệnh có thể là một trong bốn dạng sau:

1. [nhan:] chỉ_thị [toán_hạng] [;lời chú thích]
2. [nhan:] lệnh [toán_hạng] [;lời chú thích]
3. ;chú thích
4. dòng trống (không chứa kí tự nào)

Một lời chú thích luôn đi sau dấu chấm phảy(“;”)và nó không được dịch sang mã máy chỉ có tác dụng cho người đọc chương trình dễ hiểu.

Chương trình Assembly hỗ trợ một số các chỉ thị.Các chỉ thị này không được dịch ra mã nhị phân (mã máy).Và nó được sử dụng để điwuf khiển quá trình dịch và cụ thể là : điều khiểu ghi lệnh vào bộ nhớ chương trình, định nghĩa các biến …

Dưới đây là bảng các chỉ thị :

Chỉ thị	Mô tả
BYTE	Định nghĩa một biến kiểu byte
CSEG	Đoạn mã chương trình
DB	Định nghĩa một hằng số kiểu byte
DEF	Định nghĩa một tên gợi nhớ cho một thanh ghi
DEVICE	Định nghĩa loại VĐK cho chương trình
DSEG	Đoạn dữ liệu
DW	Định nghĩa một hằng số kiểu 2 byte (word)
ENDMACRO	Kêt thúc của một macro
EQU	Thay một biểu thức bằng một kí tự.
ESEG	Đoạn EEPROM
EXIT	Thoát ra từ một file
INCLUDE	Sử dụng mã nguồn từ một file khác
LIST	Cho phép tạo ra trong file list
LISTMAC	Cho phép thêm macro vào list khi được gọi
MACRO	Bắt đầu macro
NOLIST	Không cho phép tạo ra trong file list
ORG	Thiết lập mốc của chương trình
SET	Gán một nhãn cho một giá trị

Tất cả các chỉ thị đều đặt sau dấu chấm (“.”).

1.1.BYTE :

Chỉ thị này giành trước tài nguyên bộ nhớ trong SRAM.Chỉ thị này phải đi sau một nhãn và có một tham số,nó chỉ ra số byte được giành trước.Chỉ thị này chỉ dùng trong đoạn dữ liệu.

Cú pháp :

```
LABEL: .BYTE expression
```

Ví dụ:

```
.DSEG
        var1:   .BYTE 1 ;
        var2 :  .BYTE 10;
.CSEG
        ldi r30,low(var1);
```

Nếu như bạn nào đã học qua một ngôn ngữ cấp cao nào đó thì thực ra vùng nhớ này cũng như là một biến.Dữ liệu sẽ không tự động được ghi vào và chỉ khi bạn dùng các lệnh tác động đến nó mà thôi.Nhãn chính là địa chỉ đầu của đoạn bộ nhớ được giành trước .

1.2.Chỉ thị CSEG:
Chỉ thị này định nghĩa điểm bắt đầu của đoạn mã chương trình.Một file nguồn assembly có thể chứa nhiều đoạn mã chương trình ,và chúng lại được liên kết thành một đoạn mã lệnh khi dịch.Chỉ thị BYTE không được sử dụng trong đoạn này.Một đoạn chương trình nếu không được định nghĩa là mã lệnh hay dữ liệu thì đều được mặc định là đoạn mã lệnh.Mỗi đoạn mã lệinh thì có một địa chỉ riêng 16 bit (hay là một từ).Chỉ thị ORG có thể được sử dụng để đặt vị trí của các đoạn mã lệnh và hằng số trong bộ nhớ chương trình .Chỉ thị này không kèm theo bất kì một tham số nào.
Cú pháp:

```
.CSEG
```

Ví dụ:

```
.DSEG
        var1: .BYTE  1
.CSEG
        CONST: .DW  2
        MOV R1,R0
```

1.3.DB:
Định nghĩa các hằng số kiểu byte được lưu trong bộ nhớ chương trình họac bộ nhớ EEPROM.Và chỉ thị này luôn theo sau một nhãn .Chỉ thị này thường được sử dụng trong việc lưu giữ các bảng và các biểu thức (nhưng có thể tính ra giá trị cuối cùng) .Các nhãn chính là địa chỉ khởi đầu cho giá trị ban đầu của bảng.Chỉ dẫn này chỉ có thể đặt được trong đoạn mã hoặc đoạn bộ nhớ EEPROM.
Các phần tử trong bảng được phân biệt bằng dấu phảy.
Cú pháp:

```
Label:      .DB      danh_sach_biểu_thức
```

Ví dụ:

```
.CSEG
Sin: .DB      0,1,2,3,4,6,7
.ESEG
const: .DB 1,2,3
```

Chú ý: Một số hay một biểu thức (phải có kết quả) nằm trong khoảng –128 đến 255.Nếu số đó là só âm thì sẽ được lưu dưới dạng 8bit mã bù 2.
4.DEF:

Chỉ thị này có tác dụng cho phép lập trình viên đặt tên cho một thanh ghi.Thay bằng nhớ thanh ghi đó lập trình viên có thể đặt tên cho nó với cái tên gợi nhớ hơn .
Cú pháp:

```
        .DEF tên_gợi_nhớ=thanh_ghi
```

Ví dụ:

```
        .DEF xh=R28
        .DEF xl=R29
```

Chú ý: Một thanh ghi có thể có rất nhiều tên gợi nhó gán cho nó nhưng điều đó sẽ rất nguy hiểm có thể vô tình bạn làm mất dữ liệu trong thanh ghi đó mà bạn không mong muốn.
1.5.DEVICE:
Chỉ thị này chỉ cho chương trình dịch biết loại vi điều khiển mà ta đang viết chương trình.
Cú pháp:

```
                .DEVICE     Loại_vi_điều_khiển
```

Ví dụ:

```
                .DEVICE     AT90S8535
```

Chỉ thị này sẽ báo cho chúng ta những lỗi sinh ra khi mà chương trình dịch tìm thấy những lệnh cũng như những thiếtbị ngoại vi không được hỗ trợ trong loại vi điều khiển này.
1.6.DSEG:
Chỉ thị này định nghĩa điểm bắt đầu của đoạn dữ liệu.Một file nguồn có thể có nhiều đoạn dữ liệu nhưng khi dịch chúng thì chúng được gộp liên kết vào một đoạn.Một đoạn dữ liệu bình thường chỉ chứa duy nhất chỉ thị BYTE.Mỗi đoạn dữ liệu đều có một con trỏ vị trí riêng và đó là con trỏ 8 bit (vì chúng trong bộ nhớ RAM).Chỉ thị ORG có thể được sử dụng để đặt các biến tại các vị trí xác định trong RAM (chỉ thị này sẽ được nói ở sau).
Cú pháp:

```
                .DSEG
        var1:   .BYTE       1
        table:  .BYTE       table_size
```

1.7.DW:
Đây là chỉ thị cho phép người sử dụng định nghĩa các hằng số ở dạng 2 byte trong bộ nhớ chương trình hoặc bộn nhớ EEPROM nó hoàn toàn tương tự như chỉ thị DB.
Cú pháp:

```
                Label:  .DW         danh_sách_biểu_thức
```

Ví dụ:

```
                Var:            .DW   12,354,3434,31345
```

1.8.ENDMACRO:
Kết thúc của một macro.Chỉ thị này không đi kèm một tham số nào cả.
Cú pháp:

```
    .Endmacro
```

Ví dụ:

```
        .Macro  subi16
            subi r16,low(@0)
            subi r17,high(@0)
        .Endmacro
```

1.9.EQU:
Chỉ thị EQU gán giá trị của một biểu thức cho một nhãn.Nhãn sau khi gán giá trị trở thành một hằng số và không được định lại giá trị.
Cú pháp:

```
        .EQU  const=expression
```

Ví dụ:

```
        .EQU io=0x23
        .EQU ios=io-10
```

1.10. ESEG:
Hoàn toàn giống với CSEG
1.11. EXIT:
Chỉ thị EXIT báo cho chương trình dịch biết dừng việc đọc file lại.Bình thường thì chương trình dịch sẽ chạy cho tới khi hết file thì kết thúc.Nhưng nếu như trong file có chứa chỉ thị này thì khi nào chương trình dịch gặp chỉ thị này thì sẽ kết thúc qúa trình đọc.
Cú pháp:

```
        . EXIT
```

1.12.INCLUDE
Chỉ thị này báo cho chương trình dịch biết bắt đầu đọc từ một file xác định cho tới khi hết file đó hoặc một chỉ thị ngừng đọc (EXIT).
Cú pháp:

```
        .Include “tên_file” ; Đôi khi cả tên file và đường dẫn.
```

Ví dụ:

```
            ;Nội dung của file iodef.asm”
            .EQU sreg = 0x3f
            .EQU sphigh=0x3e
            .EQU splow=0x3d
            ;Trong chương trình
            .INCLUDE “iodef.asm”
                in r0,sreg ; đọc thanh ghi trang thái.
```

1.13.LIST:

Cho phép chương trình dịch tạo ra file list.
Cú pháp:

.LIST

Chú ý:Mặc định của chương trình dịch là cho phép tạo ra file list và chỉ thị này luôn đi kèm với chỉ thị NOLIST.

1.14.LISTMAC:

Hoàn toàn giống với LIST nhưng với macro.

1.15.MACRO:

Chỉ dẫn khai báo macro.Vậy macro là gì? Macro thực ra là một đoạn chương trình .Khi mà macro được gọi thì đoạn chương trình đó sẽ được dán vào vị trí gọi macro.

Tham số đi theo ngay sau chỉ thị này là tên của macro.Một macro có thể có tới 10 tham số.

Cú pháp:

```
.Macro        macro_name
```

Ví dụ:

```
.Macro        sub16
………..;lệnh gì đó
……….;lênh nào đó
.Endmacro
.CSEG
sub16 ;goi macro
```

1.16.NOLIST:

1.17. ORG:

Chỉ thị ORG thiết lập một con trỏ tuyệt đối .Giá trị được thiết lập chính là tham số cho chỉ thị này.Nếu như chỉ thị này nằm trong đoạn dữ liệu thì vị trí được thiết lập chính là một vị trí trong SRAM và cụ thể đó là vị trí bắt đầu của biến đước khai báo sau chỉ thị BYTE.

Còn khi chỉ thị này được khai báo trong đoạn chương trình thì vị trí tuyệt đối đó nằm trong bộ nhớ chương trình và đoạn mã lệnh theo sau nó sẽ được ghi vào bôin nhớ chương trình từ con trỏ đó.Và đối với đoạn ESEG cũng tương tự.Nếu như chỉ thị này đi sau một nhãn thì nhãn đó có giá trị chính bằng tham số của chỉ thị này.

Cú pháp:

```
    . org tham_só
hoặc   Label: . org  tham_số
```

Ví dụ:

```
.DSEG
    . org  0x60
    var1: .BYTE 2
```

```
. ESEG
        . org  0x20
        evar:  .DB 0xff
.CSEG
        . org 0x10
        mov r0,r1
```

1.18.SET:
Gán một giá trị cho một nhãn.Nhãn này có thể sử dụng thay cho giá trị đó và nó hoàn toàn có thể bị thay đổi phụ thuộc vào chương trình.(Đây là điểm khác biệt của nó so với chỉ thị EQU).

Một số chỉ thị khác :

```
.IFDEF <symbol>
.IFNDEF <symbol>
.IF <expression>

.IFDEF <symbol> |.IFNDEF <symbol>
...
.ELSE | .ELIF<expression>
...
.ENDIF
```

Ví dụ:

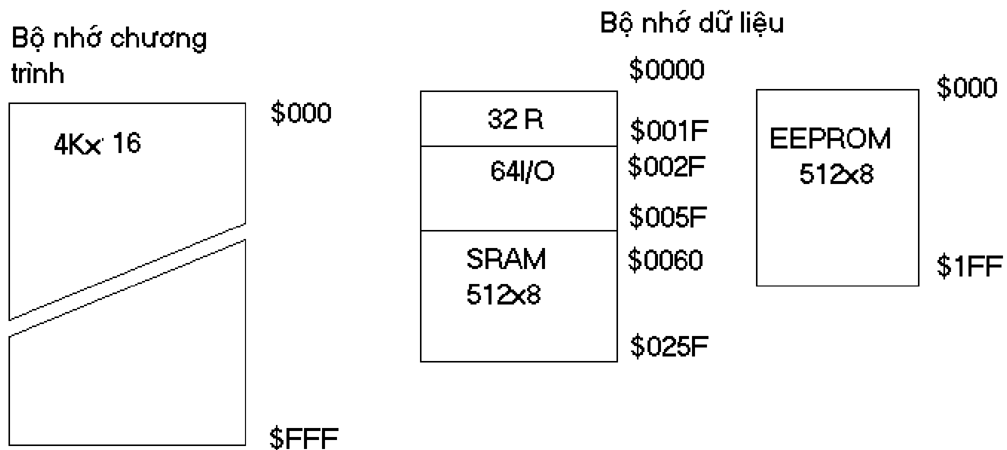
```
.MACRO SET_BAT
.IF @0>0x3F
.MESSAGE "Address larger than 0x3f"
lds @2, @0
sbr @2, (1<<@1)
sts @0, @2
.ELSE
.MESSAGE "Address less or equal 0x3f"
.ENDIF
.ENDMACRO
```

Phần 2: Viết lệnh cho VĐK

Trước khi chúng ta viết lệnh cho VĐK thì cần nắm được các vấn đề sau:
1.Cấu trúc bộ nhớ chương trình và bộ nhớ dữ liệu.
2. Các cách định địa chỉ.
3.Các thanh ghi chức năng đặc biệt.
4.Các lệnh cụ thể
5.Một chương trình mẫu.
6.Lập trình cấu trúc.
7.Chương trình con và Macro.
Ta lần lượt tìm hiểu từng nội dung.

2.1.Cấu trúc bộ nhớ:

Cũng như mọi vi điều khiển khác AVR có cấu trúc Harvard tức là có bộ nhớ và đường bus riêng cho bộ nhớ chương trình và bộ nhớ dữ liệu.
Sơ đồ bộ nhớ:



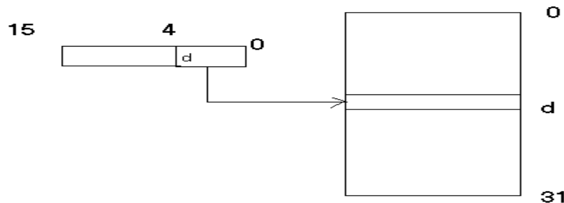
Ta thấy không gian bộ nhớ của bộ nhớ chương trình gồm 4Kx8 và có địa chỉ đánh từ 0000H tới FFFH.
Bộ nhớ dữ liệu gồm hai phần :bộ nhớ RAM và bộ nhớ EEPROM trong đó không gian bộ nhớ RAM lại chia làm 3 phần :Các thanh ghi chức năng chung,các thanh ghi vào ra và cuối cùng là 512 byte bộ nhớ SRAM . Bộ nhớ EEPROM mặc dù cùng là một phần của bộ nhớ dữ liệu nhưng lại hoàn toàn đứng độc lập như một bộ nhớ độc lập và cũng được đánh địa chỉ riêng.
2.1.1.Bộ nhớ dữ liệu:
Các thanh ghi chức năng chung:AVR có 32 thanh ghi chức năng chung và chúng được liên kết trực tiếp với ALU đây là điểm khác biệt của AVR và tạo cho nó một tốc độ xử lý cực cao.Các thanh ghi được đặt tên từ R0 tới R31.Và đặc biệt cặp 6 thanh ghi cuối (từ R6 tới R31) từng đôi một tao thành các thanh ghi 16 bit sử dụng làm con trỏ trỏ tới bộ nhớ chương trình và dữ liệu. Chúng lần lượt có tên là X,Y,Z (và sẽ tìm hiểu kĩ hơn ở phần sau).
Không gian các thanh ghi cổng vào ra bao gồm cá thanh ghi dữ liệu và thanh ghi điều khiển cho cổng vào ra.(Phần này sẽ được nói tới trong phần lập trình cho các thiết bị ngoại vi).
Cuối cùng là bộ nhớ SRAM.
2.1.2.Bộ nhớ chương trình:
Với bộ nhớ chương trình có địa chỉ từ 0000H tới 0010H được giành cho bảng véc tơ ngắt.
cụ thể:

Địa chỉ bộ nhớ	Nguồn báo ngắt	Ngắt
$0000	RESET	Khởi động lại hệ thống
$001	INT0	Ngắt ngoài 0
$002	INT1	Ngắt ngoài 1
$003	TIMER2 COMP	bộ so sánh timer2
$004	TIMER2 over	Tràn bộ đếm/định thời 2
$005	TIMER1 CAPT	
$006		
$007		
$008		
$009		
$00A		
$00B		
$00C		
$00D		
$00E		
$00F		

$010		
$011	Hết véc tơ ngắt.	

2.2.Các chế độ chuy nhập địa chỉ của AVR:

2.2.1. Địa chỉ thanh ghi đơn trực tiếp:



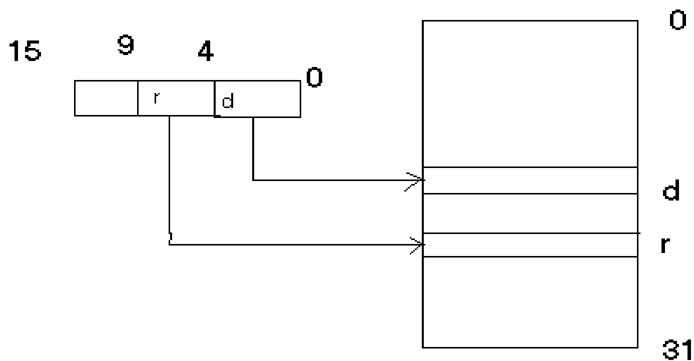
Ở chế dộ này dịa chỉ của thanh ghi được lấy trực tiếp từ vùng các thanh ghi (từ 0 tới 31)

Ví dụ:

COM Rd
NEG Rd
…

2.2.2. Địa chỉ hai thanh ghi trực tiếp:

Đây là chế độ mà trong một lênh ALU truy nhập trực tiếp vào hai thanh ghi.



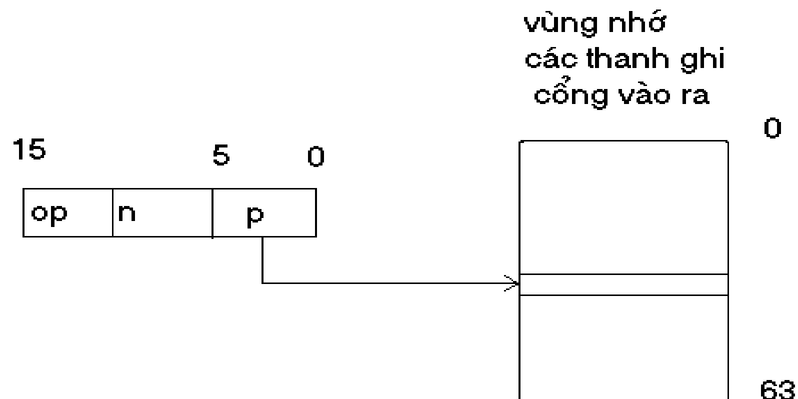
Chế độ này hoàn toàn tương tương như chế độ trên.

Ví dụ:

ADD Rd,Rr

…

2.2.3. Địa chỉ trực tiếp cổng vào ra:



Trong đó địa chỉ của toán hạng được chứa trong 6 bit của một từ lệnh .n là địa chỉ của thanh ghi nguồn hoặc đích.
Ví dụ:

```
Out DDRB,R16
In    R12, DDRB
```

2.2.4.Trực tiếp dữ liệu:
Địa chỉ của dữ liệu trong RAM được đưa trực tiếp vào lệnh
Ví dụ:

```
LDS R12,0x0fff
STS  0x0fff,R11
```

2.2.5. Địa chỉ dữ liệu dán tiếp cùng với dịch chuyển:
Ví dụ:

```
LDD        R11,Y+10
```

Địa chỉ của toán hạng nguồn họac đích được trỏ bởi thanh ghi Y hoặc Z công thêm một chỉ số nào đó
2.2.6. Địa chỉ gián tiếp dữ liệu:
Đây là cách mà CPU truy nhập tới dữ liệu trong RAM thông qua thanh ghi X,Y,Z địa chỉ của dữ liệu được lưu trong thanh ghi này.
Ví dụ:

```
ST     X,R11
LD     R13,Y
```

2.2.7. Địa chỉ dữ liệu dán tiếp cùng với tăng hoặc giảm con trỏ:
Ví dụ:

```
LD         R17,X+
LD         -Y,R14
```

….

2.2.8. Địa chỉ của hằng số trong bộ nhớ chương trình.
Cách này chỉ sử dụng cho lệnh LPM
Địa chỉ của hằng số được lưu trong thanh ghi Z
Ví dụ:

```
    LDI   R30,0x07;dia chi truc tiep du lieu 0x07
    LDI   R31,0xFF
    LPM          ; đưa nội dung của ô nhớ có địa chỉ trong Z (0x07FF )về
```

thanh ghi R0.
2.2.9. Địa chỉ bộ nhớ chương trình gian tiếp:
Địa chỉ đoạn mã được trỏ bởi thanh ghi Z sử dụng trong các lệnh IJMP và ICALL.
Ví dụ:

```
    Label:
        LDI R29,High(Label)
        LDI R28,Low(Lebel)
    ICALL
```

2.2.10. Địa chỉ tương dối của bộ nhớ chương trình.
Các định địa chỉ này dùng cho các lệnh RJMPvà RCALL khi đó CPU sẽ có giá trị PC+k+1
Ví dụ:

```
        Label:
        LDI R29,High(Label)
        LDI R28,Low(Lebel)
    RCALL  Label
```

2.3.Các thanh ghi chức năng đặc biệt:

Bao gồm các thanh ghi dữ liệu và các thanh ghi điều khiển các cổng vào ra.Chúng có thể truy nhập được bằng 2 cách: Bằng địa chỉ trực tiếp
Ví dụ như: STR $3F,R11
hoặc: STR   SREG.R11
Ta có thể truy nhập gián tiếp chúng thông qua thanh ghi X,Y,Z
Ví dụ :

```
    LDI   R28,0x00
    LDI   R27,0x5F
    STD   X,R11
```

Ở hai ví dụ này hoàn toàn tương đương. Đều ghi dữ liệu vào thanh ghi SREG.
Cụ thể từng thanh ghi và chức năng của chúng sẽ được nói tới trong phần lập trình cho các thiết bị ngoại vi và vào ra dữ liệu.

Ở đây chỉ giới thiệu về thanh ghi rất đặc biệt mà thôi (CÁc thanh ghi khác sẽ tìm hiểu ở các phần lập trình vào ra dữ liệu và điều khiển các thiết bị ngoại vi).

Status Register (SREG)

Đây là thanh ghi trạng thái có 8 bit lưu trữ trạng thái của ALU sau các phép tính số học và logic.

C: Carry Flag ;cờ nhớ (Nếu phép toán có nhớ cờ sẽ được thiết lập)

Z: Zero Flag ;Cờ zero (Nếu kết quả phép toán bằng 0)

N: Negative Flag (Nếu kết quả của phép toán là âm)

V: Two's complement overflow indicator (Cờ này được thiết lập khi tràn số bù 2)

S: Ṇ V, For signed tests (S=N XOR V)

H: Half Carry Flag (Được sử dụng trong một số toán hạng sẽ được chỉ rõ sau)

T: Transfer bit used by BLD and BST instructions(Được sử dụng làm nơi chung gian trong các lệnh BLD,BST).

I: Global Interrupt Enable/Disable Flag (Đây là bit cho phép toàn cục ngắt.Nếu bit này ở trang thái logic 0 thì không có một ngắt nào được phục vụ.)

Registers and Operands(kí hiệu các thanh ghi và các toán hạng)

Rd: Thanh ghi đích (một trong 32 các thanh ghi chức năng chung)

Rr: Thanh ghi nguồn (Một trong 32 thanh ghi chức năng chung)

R: Kết quả sau khi lệnh chạy.

K: Hằng số dữ liệu

k: Hằng số địa chỉ (Có thể là một nhãn hoặc một địa chỉ cụ thể)

b: Bit trong thanh ghi chức năng chung hoặc trong thanh ghi chức năng đặc biệt (0-7).

s: Bit trong thanh ghi trạng thái (0-7).

X,Y,Z: Thanh ghi địa chỉ (Để trỏ tới địa chỉ trong RAM ,hoặc Z có thể trỏ tới địa chỉ trong ROM).

(X=R27:R26, Y=R29:R28 and Z=R31:R30)

A: I/O location address

q:Chỉ số cho các địa chỉ trực tiếp (0-63).

Stack :Ngăn xếp.

STACK: Là nơi lưu giữ con trỏ PC cho các chương trình con và các trình phục vụ ngắt .(Cụ thết là một vùng nhớ có tính năng "vào sau ra trước")

SP: Là một thanh ghi 16 bit nhưng cũng có thể được xem như hai thanh ghi chức năng đặc biệt 8 bit.Có địa chỉ trong các thanh ghi chức năng đặc biệt là

$3E(Trong bộ nhớ RAM là $5E).Có nhiệm vụ trỏ tới vùng nhớ trong RAM chứa ngăn xếp. Khi chương trình phục vu ngắt hoặc chương trình con thì con trỏ PC được lưu vào ngăn xếp trong khi con trỏ ngăng xếp giảm hai vị trí.Và con trỏ ngăn xếp sẽ giảm 1 khi thực hiện lệnh push.Ngược lại khi thực hiện lệnh POP thì con trỏ ngăn xếp sẽ tăng 1 và khi thực hiện lệnh RET hoặc RETI thì con trỏ ngăn xếp sẽ tăng 2.Như vậy con trỏ ngăn xếp cần được chương trình đặt trước giá trị khởi tạo ngăn xếp trước khi một chương trình con được gọi hoặc các ngắt được cho phép phục vụ.Và giá trị ngăn xếp ít nhất cũng phải lơn hơn học bằng 60H (0x60) vì 5FH trở lại là vùng các thanh ghi.

2.4.Các lệnh cụ thể:

Có lẽ phần lệnh này các bạn tham khảo cuốn AVR assembly user guide
Rất đầy đủ và chi tiết.Tôi chỉ có thể hướng dẫn các bạn cách đọc một lệnh trong đó mà thôi.
Ví dụ:

LDI – Load Immediate

1.Description:

2. Loads an 8 bit constant directly to register 16 to 31.

3.Operation:

4.(i) Rd ← K

5.Syntax:	**Operands:**	**Program Counter:**
6.(i) LDI Rd,K	$16 \le d \le 31$, $0 \le K \le 255$	PC ← PC + 1

7.16-bit Opcode:

8.Status Register (SREG) and Boolean Formula:

9.Example:

```
    clr r31 ; Clear Z high byte
    ldi r30,$F0 ; Set Z low byte to $F0
    lpm ; Load constant from Program
; memory pointed to by Z
```

Words: 1 (2 bytes)
Cycles: 1
1110 KKKK dddd KKKK
I T H S V N Z C
– – – – – – – –

Đoạn trên mô tả về lệnh lấy dữ liệu trực tiếp .
Muc description mô tả toàn bộ về lệnh này.Về tính năng và các thanh ghi được sử dụng trong lệnh.

Các bạn chú ý tới dòng thứ 5:Dòng này chia làm 3 cột cột 1 :syntax là cú pháp của câu lệnh.Cột 2: Operands là các toán hạng được sử dụng trong lệnh.Các bạn thấy ở cột đó viết: 16 ≤ d ≤ 31, 0 ≤ K ≤ 255. Ở đây d tức là Rd hay cụ thể hơn là toán hạng này chỉ có thể là các thanh ghi từ R16 tới R31.Còn K là một số nguyên có giá trị từ 0 tới 255. Ở cột thứ ba các bạn thấy program counter : để chỉ sự thay đổi con trỏ PC khi thực hiện lệnh này. Dòng số 8: là các thanh ghi bị tác động sau lệnh này.Và dòng thứ 9 là mọt ví dụ minh họa.

Kinh nghiệm để học phần này : Các bạn muốn học tốt phần này thì cần phải chiu khó viết các lệnh càng nhiều càng tốt. Đặc biệt khi viết một lệnh các bạn nên viết cả phần giải thích bằng tiến anh của nó nữa .Ví dụ:

ADIW Rd,K add immediate to word

Và các bạn nhớ xác định lệnh này dùng kiểu định địa chỉ gì.Ví như lệnh trên thì Rd là thanh ghi đích và đây là kiểu đánh địa chỉ trực tiếp thanh ghi.Còn K là một hằng số từ 0 đến 255 (8bit).Và đây là cách định địa chỉ dữ liệu trực tiếp.

Để học được nhiều lệnh thì các bạn nên học theo từng nhóm lệnh ví dụ như lệnh về chuyển dữ liệu:

Các bạn có lệnh Load như vậy các bạn hãy xem lệnh này có những kiểu định địa chỉ thế nào và bạn lần lượt viết từng lệnh và chắc chắn rằng sau một lần viết thì bạn đã nhớ gần như toàn bộ các lệnh đó.

Sau khi đã nắm gần như toàn bộ các lệnh dó thì các bạn chú ý đến các cờ và sự tác động của chúng đến từng cờ một.

Đặc biệt chú ý : Một số các lệnh chỉ thực hiện với nửa số thanh ghi từ 16 trở đi như là LDI …

2.5.Một chương trình mẫu

```
;chuong trinh dau tien
;khia bao thiet bi
.DEVICE         AT90S8535;khai bao thiết bị
.DSEG           ;khai bao doan du lieu
        var1:       .BYTE           2
.CSEG           ;khai bao doan chuong trinh

    .def    tam=R16     ;dinh nghia mot ten moi cho thanh ghi R16
```

```
        .Macro          sub16 ;khai bao macro
        ;Macro chu hai byte 16bit
;bien vao :xh,xl
;           yh,yl
;bien ra:xh,xl va co C
        sub     xl,yl
        sbc     xh,yh
        .endmacro
.include "8535def.inc"    ;mo va doc tep nay (copy noi dung cua tep nay vao
chuong trinh)
        .org    0x0000
                rjmp   start
        .org    0x0001
                rjmp   int_0
        .org    0x0011
                start:
                ldi             xh,0x0
                ldi             xl,0xa
                ldi             yh,0x0
                ldi             yl,0x5
                sub16
                rcall add16
        here:
                rjmp   here
        int_0:
        ;chuong trinh phuc vu ngat int0
        nop
        reti;tro lai chuong trinh chinh tu ngat
        add16:
        ;chuong trinh con cong hai so 16 bit
        ;bien vao:xh,xl
        ;               yh,yl
        ;bien ra: :xh,xl va co C
        add     xl,yl
        adc     xh,yh
        ret
```

Đây là một chương trình đầu tiên và là một chương trình rất cơ bản đối với các bạn.

3.2.6. Lập trình cấu trúc trong Assembly:

Mọi người thường nói đây là điểm mạnh cảu ngôn ngữ bậc cao ! Vâng đúng vậy.Nhưng điều đó không có nghĩa là ngôn ngữ cấp thấp như assembly lại không làm được.

Để viết được các dạng cấu trúc như trong các ngôn ngữ bậc cao đòi hỏi các bạn phải viết nhiều và rất nhiều.Sau đây tôi sẽ viết một vài ví dụ để các bạn tham khảo.

1.

```
if  (điều kiện)
{
khối lệnh
}
else
{
khối lệnh
}
```

và cụ thể như sau:

Nếu R10 = 0xff thì copy thanh ghi R10 vào thanh ghi R11 và nếu không bằng thì đưa giá trị 0 vào thanh ghi R11

```
;doan chuong trinh viet theo cach 1:
ldi     R0,0xff
cp      R10,R0
brne    else
mov     R11,R10
rjmp    endif
else:
ldi     R11,0x00
endif:
;doan chuong trinh viet bang cah 2:
ldi     R0,0xff
cp      R10,R0
breq    then
ldi     R11,0x00
rjmp    endif
then:
mov     R11,R10
endif:
```

2.

For (khởi tạo biến,điều kiện,toán hạng)
{
khối lệnh.
}
Viết đoạn chương trình đọc 10 byte từ bộ nhớ SRAM bắt đầu từ địa chỉ 0x60 ra 10 thanh ghi đầu tiên.chương trình sử dụng thanh ghi R16 làm biến con chạy và thanh ghi X để trỏ tới địa chỉ của 10 thanh ghi và thanh ghi Y để trỏ tới bộ nhớ SRAM
Chương trình viết dưới dạng cấu trúc như sau:

```
For (X=0x00;Y=0x60;R10=10,R10<=0,++X;++Y;--R10)
{
Đọc nội dung từ [Y] đưa [X]

}
```

Chương trình assembly như sau:

```
ldi    xl,0x00       ;khởi tạo các biến con chạy
ldi    xh,0x00
ldi    yl,0x60
ldi    yh,0x00
ldi    R16,0xa
ldi    R17,0x00
loop:
cp     R16,R17       ;kiểm tra R16 =0?
breq   exit          ;Nếu R16=0 thì thoát khỏi còng lặp.
ld     R11,Y+        ;lấy dữ liệu từ [Y]  và tăng Y lên 1
ST     X+,R11        ;Lưu vào [X] và tăng X lên 1
dec    R16
rjmp   loop
exit:

;như vậy khi thoat khỏi vòng lắp thì 10 byte đã được lấy.
```

3.
Do
{
Khối lệnh.
}
While (Điều khiện)

Ví dụ 1:
ta có thể viết lại chương trình ở phần 2 như sau:

Chương trình assembly như sau:

```
ldi     xl,0x00       ;khởi tạo các biến con chạy
ldi     xh,0x00
ldi     yl,0x60
ldi     yh,0x00
ldi     R16,0xa
loop:
ld      R11,Y+        ;lấy dữ liệu từ [Y]  và tăng Y lên 1
ST      X+,R11        ;Lưu vào [X] và tăng X lên 1
dec     R16
brne    loop          ;kiểm tra R16 nếu như >0 tiếp tục vòng lặp
```

Ví dụ 2:

Lấy dữ liệu từ cổng PA vào và khi nào lấy được một số âm thì dừng lại và ghi số đó vào vùng nhớ 0x064 trong SRAM

Đây là một vòng lặp mà bạn chưa hề biết trước số lần lặp:

```
;chương trình được viết như sau:
ldi     R16,0x00
sts     DDA  ,R16     ;định nghĩa cổng PA là cổng vào.
ldi     R16.0xff
sts     PORA,R16      ; lối vào được đưa lên pull-up
do:
in      R15,$19       ;lấy dữ liệu từ cổng PORA
mov     R16,R15       ;copy một bản của dữ liệu
lsl     R16           ;dịch trái có cờ nhớ để lấy bit MSB
brcc    do            ;kiểm tra bit đó nếu bằng 0 thì lặp lại quá trình lấy mẫu
;đã tìm được số âm
ldi     xl,0x64       ;đặt địa chỉ cho con trỏ SRAM
ldi     xh,0x00
st      x,R15         ;Lưu dữ liệu vào SRAM
;Đoạn chương trình đã tìm được số âm từ cổng PA
```

Các cấu trúc còn lại các bạn có thể tự khám phá.Và kinh nghiệm của tôi khi viết về phần này là “viết và viết thật nhiều”thì bạn có thể sử dụng được thành thạo mọi cấu trúc.

Như vậy về phần ngôn ngữ lập trình cho tới giờ các bạn đã thấy Assembly cũng không phải là một ngôn ngữ quá phức tạp và không xây dựng được các ứng dụng lớn.Nó hoàn toàn có thể modul hóa các dự án lớn và có thể thực hiện mọi cấu trúc của bất kì ngôn ngữ cấp cao nào.

32.7.Chương trình con và Macro
Có lẽ khi nói tới chương trình con thì ai cũng đã biết. Đối với assembly thì chương trình con hết sức đơn giản.
Ví dụ:

```
        Sub16:
        ;chương trình con cộng hai số 16bit
        ;inputs:        ah=R20,al=R19
        ;               bh=R22,bl=R21
        ;outputs        ah,al,co c
        .def    ah=R20
        .def    al=R19
        .def    bh=R22
        .def    bl=R21
                add     al,bl
                adc     ah,bh
        ret
```

Như các bạn thấy đấy một chương trình con r ất đơn giản.Tên chương trình con là một nhãn và khi kết thúc chương trình con bằng lệnh RET.Hoạt động của chương trình con:
Ví dụ:

```
        .CSEG
.include "8535def.inc"
org     0x0000
        rjmp    start
org     0x0011
        start:
ldi     R20,10
ldi     R21,0
ldi     R19,0
ldi     R22,0x1f
rcall   sub16                   ;goi chuong trinh con
here:
rjmp    here
Sub16:                          ;khai bao chuong trinh con
        ;chương trình con cộng hai số 16bit
        ;inputs:        ah=R20,al=R19
        ;               bh=R22,bl=R21
        ;outputs        ah,al,co c
        .def    ah=R20
        .def    al=R19
```

```
.def    bh=R22
.def    bl=R21
        add     al,bl
        adc     ah,bh
ret                     ;ket thuc chuong trinh con.
```

Khi chương trình chính chạy tới lệnh gọi chương trình con (rcall sub16 thì con trỏ PC sẽ trỏ tới nơi luugwx chương trình con và cụ thể là nhãn sub16.Thực hiện hết các dòng lệnh cho tới khi gặp lệnh RET thì con trỏ PC lại trỏ tới lệnh ngay sau lệnh rcall.Quá trình cất PC và khôi phục PC thì CPU sử dụng ngăn xếp.(Sẽ được nói sau)

Macro:

Để định nghĩa Macro trước hết ta hãy xét một ví dụ về Macro:

```
.Macro          sub16 ;khai bao macro
        ;Macro chu hai byte 16bit
        ;bien vao :xh,xl
        ;           yh,yl
        ;bien ra:xh,xl va co C
        sub     xl,yl
        sbc     xh,yh
        .endmacro               ;ket thuc macro
```

Từ ví dụ các bạn có thể thấy một macro được khai báo bằng chỉ thị macro (đã nói ở trước) (Tại sao lại là chỉ thị?).Tôi xin được nhắc lại một chút đó là :Chỉ thị là những chỉ dẫn giúp cho chương trình dich dịch các lệnh trong file nguồn mà thôi nó không phải là một lệnh của vi diều khiển (chúng ta sẽ lại trở lại vấn đề này sau)

Như vây để viết một macro các bạn dùng chỉ dẫn MACRO để khai báo.Tham số đi ngay theo sau chỉ dẫn này chính là tên của macro (Nó có ý nghĩa gì ?)và theo sau tên có thể là các tham số hoặc không (chúng được cách nhau bởi dấu phảy)

Sau khi khai báo macro là khối lệnh mà các bạn muốn thực hiện.Để kết thúc macro thì các bạn dùng chỉ dẫn .endmacro

Macro sẽ làm việc như thế nào ?Ta sẽ tìm hiểu qua ví dụ sau:

```
;chuong trinh su dung macro
;khia bao thiet bi
                                .DEVICE         AT90S8535
.DSEG           ;khai bao doan du lieu
        var1:           .BYTE           2
.CSEG           ;khai bao doan chuong trinh

        .Macro          sub16 ;khai bao macro
```

```
        ;Macro chu hai byte 16bit
;bien vao :xh,xl
;           yh,yl
;bien ra:xh,xl va co C
        sub     xl,yl
        sbc     xh,yh
        .endmacro

.include "8535def.inc"     ;mo va doc tep nay (copy noi dung cua tep nay vao chuong trinh)
        .org    0x0000      ;bat dau chuong trinh chinh
                rjmp    start
        .org    0x0011
                start:
                ldi         xh,0x0      ;khoi tao cac thanh ghi
                ldi         xl,0xa
                ldi         yh,0x0
                ldi         yl,0x5
                sub16                   ;gọi macro
```

Các bạn nhận thấy đấy :Macro được sử dụng để trừ hai số 16 bit với biến đầu vào là thanh ghi x và thanh ghi y.Kết quả được lưu vào thanh ghi x. Khi dịch chương trình này ra mã máy thì khi gặp lệnh gọi macro (lệnh sub16)thì chương trình dịch sẽ copy và dán toàn bộ nội dung bên trong của hai chỉ dẫn .macro và .endmacro vào vị trí có lệnh gọi.

Cụ thể chương trình trên tương đương với chương trình sau:

```
;chuong trinh dau tien
;khia bao thiet bi
                                        .DEVICE         AT90S8535
.DSEG               ;khai bao doan du lieu
            var1:       .BYTE           2
.CSEG               ;khai bao doan chuong trinh
.include "8535def.inc"     ;mo va doc tep nay (copy noi dung cua tep nay vao chuong trinh)
        .org    0x0000
                rjmp    start
        .org    0x0011
                start:
                ldi         xh,0x0
                ldi         xl,0xa
                ldi         yh,0x0
```

```
ldi     yl,0x5
sub     xl,yl
sbc     xh,yh
```

.Như vậy đến bây giời có lẽ các bạn đã biết được thế nào là một macro!.Vậy macro là một đoạn chương trình mà khi có lệnh gọi nó chương trình dịch sẽ dán nội dung rong macro vào vị trí gọi nó.Và đây là lý do và người ta gọi là chỉ thị macro

!!!Chú ý khi sử dụng macro:Do macro được thiết kế với mục đích modul hóa các đoạn chương trình và để có thể dùng lai nhiều lần (Ví dụ bạn viết một file lưu giữ từng macro và khi cần thì bạn chỉ cần include chúng vào file dự án của bạn là được (Phần này thì có lẽ các bạn mới học thì hơi khó nắm bắt tối sẽ nói đến trong phần modul hóa chương trình lớn)).Và vì một mục đích đơn giản nữa là các bạn không phải viết đi viết lại nhiều lần một đoạn chương trình.Vì vậy phần giới thiệu về macro là phần rất quan trọng.Mặc dù chúng không được dịch ra mã máy (chúng chỉ là những chỉ dẫn giúp cho người đọc chương trình dễ hiểu)nhu nó sẽ làm cho những người sử dụng nó dễ hiểu và có thể sử dụng được chúng.

Sự khác nhau giữa chương trình con và Macro là gì ?

Câu hỏi này có lẽ rất nhiều bạn thắc mắc ?nhưng sau khi đã tìm hiểu về chương trình con và macro có lẽ các bạn cũng tự trả lời được.Nhưng tôi cúng xin tóm lại chút ít như sau:

Thú nhất:Chương trình con là một cơ chế mà CPU cho phép thay đổi vị trí truy nhập bộ nhớ chương trình(khi có lệnh gọi chương trình con thì CPU truy nhập tới nơi lưu chuong trình con và khi gặp lệnh RET thì CPU trở lại chương trình chính) còn Macro chỉ là chỉ thị để chương trình dịch biết cách copy những đoạn chương trình giống nhau vào những vị trí xác định (vị trí gọi macro).

Thứ hai: Về mặt sử dụng tùy từng ứng dụng mà ta sử dụng chương trình con hay Macro.Nếu như bạn muốn tiết kiệm bộ nhớ tức là nhiều đoạn giống nhau thì ta chỉ cần viết một đoạn và đặt một nơi trong bộ nhớ khi cần dùng đến nó thì cho con trỏ PC trở tới nó sau khi đã dùng xong thì trở lại chuơng trình chính (chương trình con).Nhưng mỗi lần trỏ tới nó như thế thì bạn sẽ rất mất thời gian (vì các lệnh gọi chương trình và trở về chương trình chính sẽ ngún đi của bạn chừng 3 chu kỳ máy)như vậy thì nếu bạn đặt luôn đoạn chương trình cần chạy ở ngay đó thì tiết kiệm được 3 chu kì máy (Macro).Như vậy cả hai phương pháp này đều có điểm manh và điểm yếu và chúng trái ngược nhau.Tùy mục đích sử dụng mà các bạn nên chọn loại nào.

Phương pháp modul hóa chương trình:

Như bao người vẫn quan niệm.Ngôn ngữ assembly chỉ dùng để học về vi điều khiển chứ không thể sử dụng với các ứng dụng lớn được.Và mọi người

thường nói C hay Pascal hoặc Basic mới thích hợp với điều đó.Đây thực sự là một quan niệm rất sai lầm !!!Tôi sẽ giới thiệu cho các bạn một phương pháp để modul hóa (chia nhỏ)một dự án lớn thành các chương trình nhỏ.Đối với AVR studio việc này là hoàn toàn đơn giản.
Ta có thể làm một ví dụ:
Giả sử bạn tao một dự án mới có tên là "thu" và tất nhiên file asembly do chương trình tự tao ra cũng có tên là thu.asm và có nội dung như sau:

```
;chuong trinh dau tien
;khia bao thiet bi
.DEVICE         AT90S8535;khai bao thiết bị
.DSEG           ;khai bao doan du lieu
        var1:       .BYTE       2
.CSEG           ;khai bao doan chuong trinh

    .def    tam=R16     ;dinh nghia mot ten moi cho thanh ghi R16

    .Macro      sub16 ;khai bao macro
    ;Macro chu hai byte 16bit
;bien vao :xh,xl
;       yh,yl
;bien ra:xh,xl va co C
    sub     xl,yl
    sbc     xh,yh
    .endmacro
.include "8535def.inc"      ;mo va doc tep nay (copy noi dung cua tep nay vao
chuong trinh)
    .org    0x0000
        rjmp    start
    .org    0x0001
        rjmp    int_0
    .org    0x0011
        start:
        ldi         xh,0x0
        ldi         xl,0xa
        ldi         yh,0x0
        ldi         yl,0x5
        sub16
        rcall add16
    here:
        rjmp    here
```

```
int_0:
;chuong trinh phuc vu ngat int0
nop
reti;tro lai chuong trinh chinh tu ngat
add16:
;chuong trinh con cong hai so 16 bit
;bien vao:xh,xl
;                yh,yl
;bien ra: :xh,xl va co C
add    xl,yl
adc    xh,yh
ret
```

Như vậy trong chương trình của tôi có 1 macro.Tôi sẽ nhóm tất cả các macro và đưa vào một file khác.Để làm điều này tôi vào project chọn create new file sau đó đánh tên file là macro.asm(hoặc macro.inc hoặc tên bất kì như có đuôi là .asm hoặc .inc)và ghi vào thư mục có chứa dự án.
với nội dung file như sau:

```
        .Macro        sub16 ;khai bao macro
        ;Macro chu hai byte 16bit
;bien vao :xh,xl
;           yh,yl
;bien ra:xh,xl va co C
        sub    xl,yl
        sbc    xh,yh
        .endmacro
```

Và chương trình của tôi được viết lại như sau:

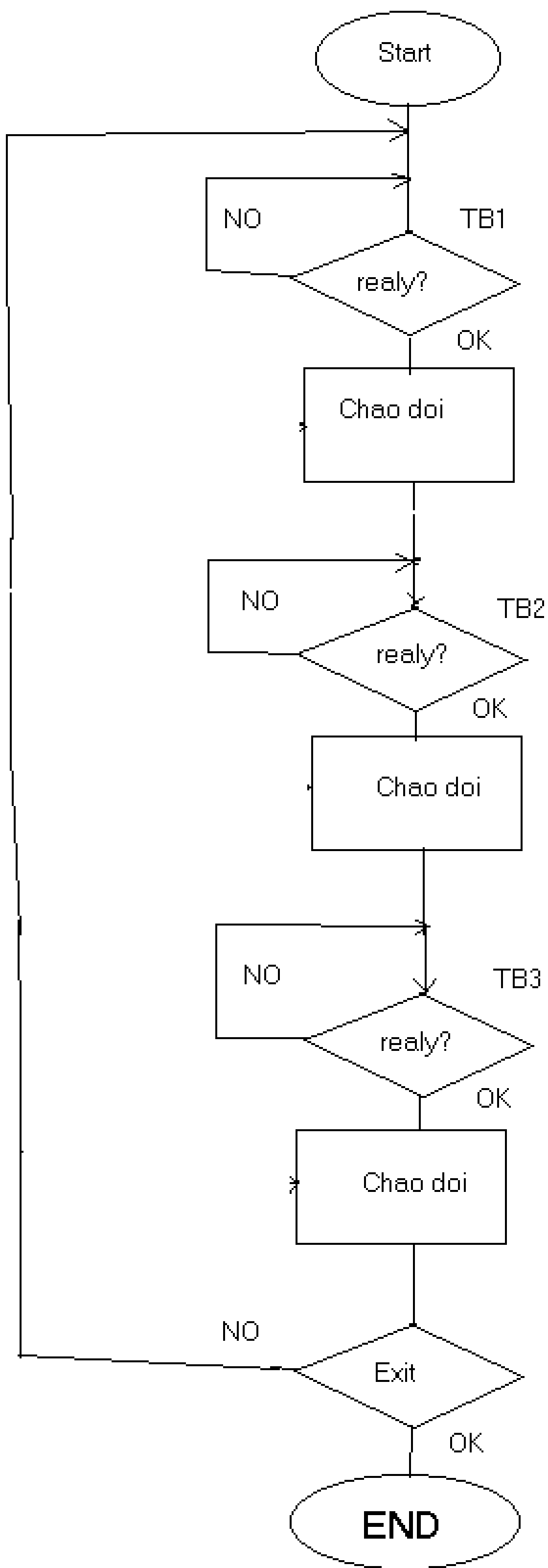
```
;chuong trinh duoc viet lai
;khia bao thiet bi
                                        .DEVICE            AT90S8535
.DSEG               ;khai bao doan du lieu
        var1:           .BYTE               2
.CSEG               ;khai bao doan chuong trinh

        .def    tam=R16     ;dinh nghia mot ten moi cho thanh ghi R16

.include "macro.asm"        ;mo file chua cac modul va doc
.include "8535def.inc"      ;mo va doc tep nay (copy noi dung cua tep nay vao
chuong trinh)
        .org    0x0000
                rjmp    start
```

```
.org    0x0001
        rjmp   int_0
.org    0x0011
        start:
        ldi             xh,0x0
        ldi             xl,0xa
        ldi             yh,0x0
        ldi             yl,0x5
        sub16
        rcall add16
here:
        rjmp   here
int_0:
;chuong trinh phuc vu ngat int0
nop
reti;tro lai chuong trinh chinh tu ngat
add16:
;chuong trinh con cong hai so 16 bit
;bien vao:xh,xl
;               yh,yl
;bien ra: :xh,xl va co C
add     xl,yl
adc     xh,yh
ret
```

Như vậy thay vào vị trí đặt macro tôi đã thay vào bằng chỉ thị include :mở file macro.asm và đọc (khi biên dịch).
Vị trí đặt của chỉ thị này không nhất thiết nằm ở vị trí khia báo macro mà chỉ cần nó đặt ở vị trí trước khi gọi macro là được.
Một file có thể lưu rất nhiều macro mà dự án của bạn cần.
Ở trên các bạn đã được tôi phân tích cách làm thông qua một ví dụ cụ thể.Đến lượt các bạn,nếu như các bạn muốn làm một dự án lớn có nhiều lập trình viên thì cần phải có người quản trị dự án đây là người có vai trò tách một chương trình lớn thành các chương trình nhỏ và giao nhiệm vụ cho từng lập trình viên .Cuối cùng người đó còn có nhiệm vụ ghép các modul nhỏ đó lại thành một chương trình hoàn chỉnh.
Chú ý:không chỉ có macro mà bạn có thể tách từng khối nào đó của chương trình vào một file khác và liên kết nó bằng chỉ thị include nhưng chỉ thị này cần đặt đúng vị trí mà khối lệnh đó được đặt ở chương trình chính

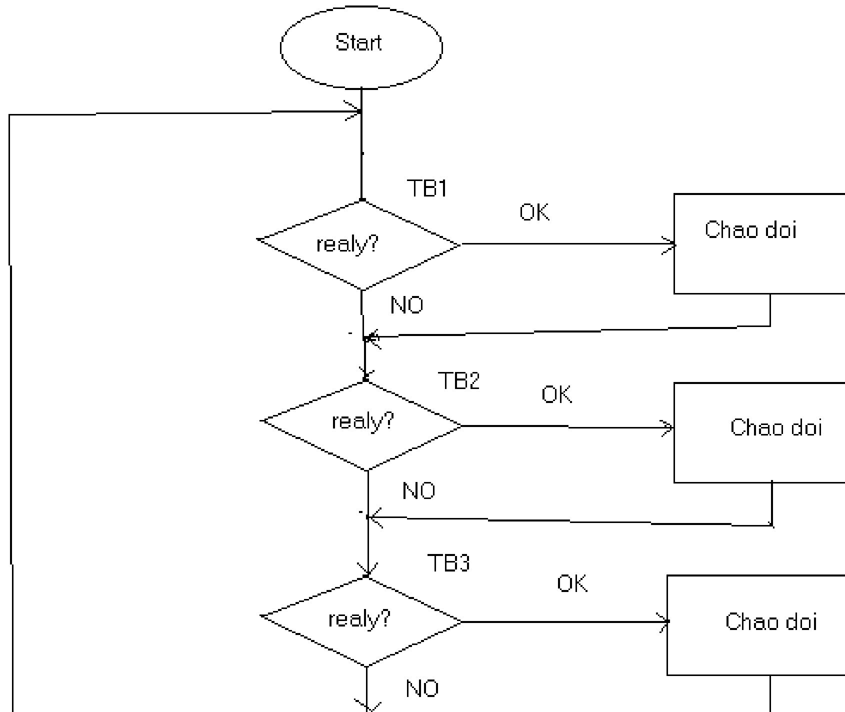
Phần 3:Vào ra dữ liệu và lập trình cho các thiết bị ngoại vi.

Một vi điều khiển thì như các bạn đã biết nó bao gồm CPU là bộ não trung tâm của nó.Nhưng nó không thể đứng độc lập được.và để xử lý được dữ liệu thì tất nhiên nó phải lấy dữ liệu từ một nguồn dữ liệ nào đó.Các thiết bị ngoại vi trên nó và các port chính là các thiết bị trung gian đưa dữ liệu vào cho CPU và chuyển dữ liệu đã xử lý ra các cơ cấu chấp hành và lên mạng thông tịn…Quá trình vào ra dữ liệu là quá trình điều khiển các port các thiết bị ngoại vi sao cho CPU có thể nhập hợc xuất dữ liệu một cách đồng bộ
Có 3 phương pháp vào ra dữ liệu đó là:vào ra dữ liệu bàng chương trình,vào ra dữ liệu bằng ngắt và vào ra dẽ liệu bằng DMA.
Ở đây vi điều khiển AVR chỉ có hai phương pháp trên còn DMA các bạn muốn tìm hiểu thêm có thể xem tào liệu về 8086 hợc Pentum.
1.Vào ra dữ liệu bằng chương trình :
Đây là phương pháp mà CPU hỏi thiết bị ngoại vi về khả năng sẵ sàng chao đổi dữ liệu.Phương pháp này có hai chế độ đó là chế độ ưu tiên và chế độ bình đẳn.Chế độ ưu tiên là chế độ :


Start
NO
TB1
realy?
OK
Chao doi
NO
TB2
realy?
OK
Chao doi
NO
TB3
realy?
OK
Chao doi
NO
Exit
OK
END

Ở chế độ này thì thiết bị được quyền ưu tiên sẽ chao đổi dữ liệu xong mới tới thiết bị có mức ưu tiên thấp hơn.

Chế độ bình đẳng:



Và từng thiết bị một sẽ được hỏi nếu như không có nhu cầu chao đổi dữ liệu thì CPU sẽ hỏi thiết bị khác.

Ưu điểm của phương pháp này là: CPU chủ động và phân được quyền ưu tiên.Nhưng nhược điểm của phương oháp này là :Không đáp ứng được các sự kiện tức thời xảy ra (Ví như có một thiết bị có yêu cầu chao đổi dữ liệu khẩn cấp mà vẫn chư tới lượt được hỏi),và trong thời gian hỏi trạng thái của các thiết bị thì cpu không thể làm việc khác, điều này làm lãng phí tài nguyên xử lý của CPU.

2.Phương pháp vào ra bằng ngăt.

Ngắt là nguyên tắc cho phép thiết bị ngoại vi báo cho CPU biết về khả năng sẵn sàng chao đổi dữ liệu của mình.

Các bạn có thể tự tìm thấy ưu và nhược điểm của phương pháp này.

Trên VĐK có 32 đường vào ra gồm nhiều chức năng có thể sử dụng làm cổng vào ra số ,cho các thiết bị ngoại vi tương tự.Sau đây ta sẽ tìm hiểu từng thiết bị ngoại vi và các cổng vào ra.(mục đích của phần này là hiểu được cấu tạo và có thể điều khiển vào ra dữ liệu được chúng).

3.1.Cổng vào ra:
Vi điều khiển AT90S8535 có 32 đường vào ra chia làm bốn nhóm 8bit một.Các đường vào ra này có rất nhiều tính năng và có thể lập trình được. Ở đây chúng ta sẽ xet chúng là các cổng vào ra số.Nếu xét trên mặt này thì các cổng vào ra này là cổng vào ra hai chiều có thể định hướng theo từng bit.Và chứa cả điện trở pull-up (có thể lập trình được).Mặc dù mỗi port có các đặc điểm riêng nhưng khi xet chúng là các cổng vào ra số thì dường như điều khiển vào ra dữ liệu thì hoàn toàn như nhau.Chúng ta có thanh ghi và một địa chỉ cổng đối với mỗi cổng , đó là : thanh ghi dữ liệu cổng (PORTA,PORTB,PORTC),thanh ghi dữ liệu điều khiển cổng (DDRA, DDRB, DDRC)và cuối cùng là địa chỉ chân vào của cổng(PINA,PINB,PINC)

1.Thanh ghi DDRA:

Đây là thanh ghi 8 bit (các bạn có thể đọc và ghi các bit ở thanh ghi này) và có tác dụng điều khiển hướng cổng PA(tức là cổng ra hay cổng vào).Nếu như một bit trong thanh ghi này được set thì bit tương ứng đó trên PA được định nghĩa như một cổng ra.Ngược lại nếu như bit đó không được set thì bit tương ứng trên PA được định nghĩa là cổng vào.

2.Thanh ghi PORTA:

Đây cũng là thanh ghi 8 bit (các bit có thể đọc và ghi được) nó là thanh ghi dữ liệu của cổng PA và trong trường hợp nếu cổng được định nghĩa là cổng ra thì khi ta ghi một bit lên thanh ghi này thì chân tương ứng trên port đó cũng có cùng mức logic. Trong trường hợp mà cổng được định nghĩa là cổng vào thì thanh ghi này lại mang dữ liệu điều khiển cổng.Cụ thể nếu bit nào đó của thanh ghi này được set (đưa lên mức 1) thì điện trở cheo (pull-up)của chân tương ứng của port đó sẽ được kích hoạt.Ngược lại nó sẽ ở trạng thái cao trở.
Thanh ghi này sau khi khởi động VĐK sẽ có giá trị là 0x00.

3. Địa chỉ chân vào PINA:

Đây là địa chỉ cho phép truy nhập trức tiếp ra các chân vật lý của vi điều khiển.Tất nhiên vì thế mà với địa chỉ này bạn chỉ có thể đọc mà thôi(Không thể ghi được!!!)
Ta có bảng tóm tắt sau:

DDAn	PORTAn	I/O	Pull-up	Chú thích
0	0	cổng vào	Không	Cao trở
0	1	cổng vào	Có	
1	0	cổng ra	Không	đầu ra đẩy kéo
1	1	cổng ra	Không	đầu ra đẩy kéo

Trong đó: DDAn là bit thứ n trong thanh ghi DDA
Bảng các thanh ghi và địa chỉ của chúng cho từng PORT

PORT	Tên	Địa chỉ thanh ghi/Địa chỉ trong SRAM
A	PORTA	0x1B/0x3B
A	DDRA	0x1A/0x3A
A	PINA	0x19/0x39
B	PORTB	0x18/0x38
B	DDRB	0x17/0x37
B	PINB	0x16/0x36
C	PORTC	0x15/0x35
C	DDRC	0x14/0x34
C	PINC	0x13/0x33
D	PORTD	0x12/0x32
D	DDRD	0x11/0x31
D	PIND	0x10/0x30

Tóm lại:

1. Để đọc dữ liệu từ ngoài thì ta phải thực hiện các bước sau:

Dưa dữ liệu ra thanh ghi điều khiển DDRxn để đặt cho PORT (hoặc bit trong port) đó là đầu vào (xóa thanh ghi ddr hoặc bit)

Sau đó kích hoạt điện trở pull-up bằng cách set thanh ghi PORT(bit)

Cuối cùng đọc dữ liệu từ địa chỉ PINxn (trong đó x: là cổng và n là bit)

Sau đây là một ví dụ:

Ví dụ 1:

Đọc cổng PA vào thanh ghi R16

```
;chương trình bắt đầu
ldi     R17,0x00
sts     $3a,R17         ; Định nghĩa port A là cổng vào
ser     R17             ;set thanh ghi R17
sts     $3b,R17     ;Kích hoạt điện trở pull-up
in      R16,PINA
```

Ví dụ 2 : Đặt cổng PA thanh hai nửa byte.Một nửa thấp là cổng ra còn nửa cao là cổng vào.

```
Ldi     R17,0x0f
Out     DDRA,R17  ; Định nghĩa
SER     R17
OUT     PORTA,R17
IN      R16,PINA
```

Như các bạn thấy đấy ! ở hai đoạn chương trình trên tôi chỉ thay dữ liệu mỗi của thanh ghi DDRA.Hai đoạn lệnh này tôi sử dụng các kiểu định dịa chỉ khác nhau do đó các bạn cảm tưởng như hoàn toàn khác nha.
Thực ra thì lệnh:

```
OUT      DDRA,R17
```

Hoàn toàn tương đương với:

```
STS      0x3a,R17
```

.
Ví dụ 3: Đọc dữ liệu từ bit 4 của cổng PA
Các bước tiến hành:
Đặt bit 4 của công PA là đầu vào .Sau đó kích hoạt điện trở pull-up và cuối cùng là lấy dữ liệu.

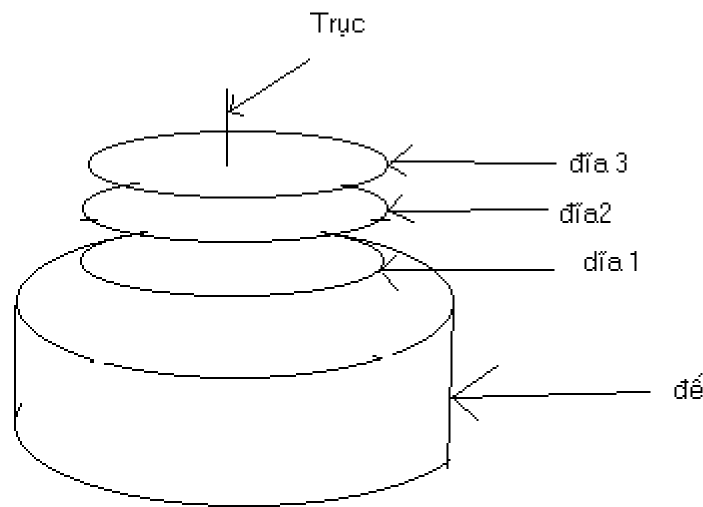
```
;chuong trinh duoc viet nhu sau:
.def     tam=R17      ;dinh nghia ten moi cho R17 de se su dung
lid      tam,0b11110111
OUT      DDRA,tam
SER      tam          ;dua thanh ghi nay len 0xff
OUT      PORTA,tam
INT      tam,PINA
BST      tam,0x4      ;bit nay duoc luu vao co T
;het
```

2. Để đưa dữ liệu từ VĐK ra các port cũng có các bước hoàn toàn tương tự.Ban đầu các bạn cũng phải định nghĩa đó là cổng ra bằng cách set bit tương ứng của cổng đó….và sau đó là ghi dữ liệu ra bit tương ứng của thanh ghi PORTx.

3.2.Ngắt và lập trình ngắt.

Các kiến thức về ngắt có lẽ các bạn đã nắm được trong môn vi xử lý 1 tôi chỉ nhắc lại một số kái niệm cơ bản:Ngắt là một cơ chế cho phép thiết bị ngoại vi báo cho CPU biết về tình trạng sẵn xàng cho đổi dữ liệu của mình.Ví dụ:Khi bộ truyền nhận UART nhận được một byte nó sẽ báo cho CPU biết thông qua cờ RXC,hợc khi nó đã truyền được một byte thì cờ TX được thiết lập…
Phục vị ngắt:
Nếu như ngắt đó được cho phép thực hiện thì:
Khi có tín hiệu báo ngắt.CPU sẽ tạm dừng công việc đạng thực hiện lại và lưu vị trí đang thực hiện chương trình (con trỏ PC) vào ngăn xếp sau đó trỏ tới vector phuc vụ ngắt và thức hiện chương trình phục vụ ngắt đó chơ tới khi gặp lệnh RETI (return from interrup) thì CPU lại lấy PC từ ngăn xếp ra và tiếp tục thực hiện chương trình mà trước khi có ngăt nó đang thực hiện. Trong trường hợp mà có nhiều ngắt yêu cầu cùng một lúc thì CPU sẽ lưu

các cờ báo ngắt đó lại và thực hiện lần lượt các ngắt theo mức ưu tiên.Trong khi đang thực hiện ngắt mà xuất hiện ngắt mới thì sẽ xảy ra hai trường hợp .Trường hớp ngắt này có mức ưu tiên cao hơn thì nó sẽ được phục vụ.Còn nó mà có mức ưu tiên thấp hơn thì nó sẽ bị bỏ qua.
Để khôi phục lại vị trí thực hiện chương trình chính thì CPU đã sử dụng đến ngăn xếp: Vậy ngăn xếp là gì ?và nó lằm ở đâu ?.Vâng xin trả lời là là ngăn xếp là một vùng nhớ mà có cách truy nhập hơi đặc biệt đó là "vào sau ra trước"và bạn hãy tưởng tượng như trồng đĩa CD của bạn cũng vậy:



Bạn thấy đấy bạn để đĩa vào từ cái một mới tới cái thứ 3 và muốn lấy cái thứ 1 ra thì bạn không thể tháo đế ra để lấy mà phải lấy cái thứ 3 rồi đến thứ 2 rồi mới lấy được cái thứ nhất (Tạm hiểu như thế).Nhưng trong tài liệu của hãng sản xuất không thấy nói tới bộ nhớ ngăn xếp?vâng nó là vùng bất kì trong SRAM từ địa chỉ 0x60 trở lên. Để truy nhập vào SRAM thông thường thì bạn dùng con trỏ X,Y,Z và để truy nhập vào SRAM theo kiểu ngăn xếp thì bạn dùng con trỏ SP.Con trỏ này là một thanh ghi 16 bit và được truy nhập như hai thanh ghi 8 bit chung có địa chỉ :SPL :0x3D/0x5D(IO/SRAM) Và SPH:0x3E/0x5E.
Khi chương trình phục vu ngắt hoặc chương trình con thì con trỏ PC được lưu vào ngăn xếp trong khi con trỏ ngăng xếp giảm hai vị trí.Và con trỏ ngăn xếp sẽ giảm 1 khi thực hiện lệnh push.Ngược lại khi thực hiện lệnh POP thì con trỏ ngăn xếp sẽ tăng 1 và khi thực hiện lệnh RET hoặc RETI thì con trỏ ngăn xếp sẽ tăng 2.Như vậy con trỏ ngăn xếp cần được chương trình đặt trước giá trị khởi tạo ngăn xếp trước khi một chương trình con được gọi

hoặc các ngắt được cho phép phục vụ.Và giá trị ngăn xếp ít nhất cũng phải lơn hơn 60H (0x60) vì 5FH trỏ lại là vùng các thanh ghi.
Để thau đổ giá trị của SP bạn có hai cách:

```
;Su dung dia chi truc tiep trong cac thanh I/O
Ldi         R17,0x00
OUT         SPH,R17     ;nap gia tri 0x00 vao thanh ghi SPH
Ldi         R17,0xff
OUT         SPL         ;nap gia tri 0xff vao thanh ghi SPL
;end
;su dung dia chi cua chung gian tiep qua SRAM
Ldi         R17,0x00
STS         0x5E,R17
Ldi         R17,0xFF
STS         0x5D,R17
;end
```

3.3.SPI

Serial peripheral interface

Giao diện nối tiếp thiết bị ngoại vi:

SPI là một giao diện thực hiện việc chao đổi dữ liệu giữa các thiết bị tương thích với khung giữ liệu 8bit và được truyền đồng bộ (cùng xung nhịp đồng hồ)

SPI cho phép truyền dữ liệu nối tiếp đồng bộ giữa thiết bị ngoại vi và vi điều khiển AVR hoặc giữa các vi điều khiển AVR . SPI của AT90S8535 có các đặc điểm đặc biệt sau:

-Chế độ song công,truyền dữ liệu đồng bộ 3 dây.

-Có thể giữ vai trò Master hoặc Slave.

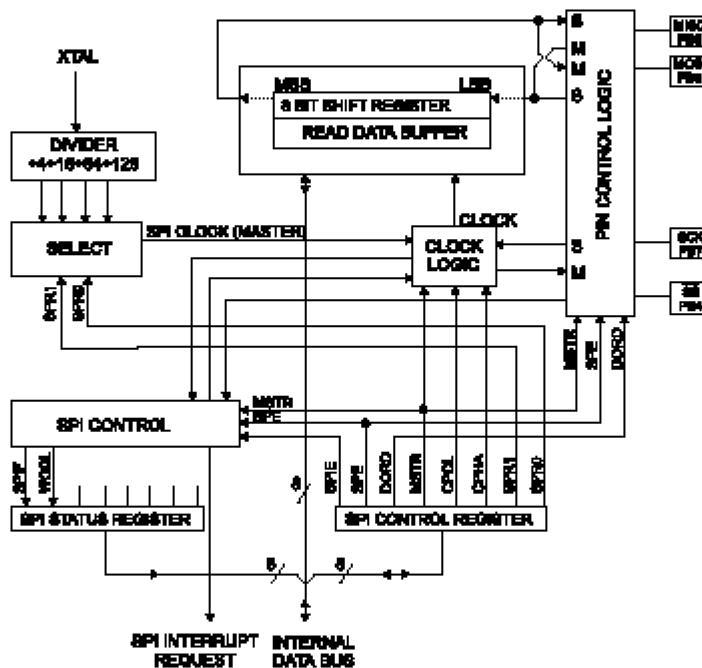
-Bit MSB hoặc LSB có thể được truyền trước tùy vào người lập trình.

-Bốn tốc độ truyền có thể lập trình thông qua hai bit

-Cờ ngắt báo kết thúc truyền

-Vận hành từ trạng thái ngủ (Được đánh thức từ trạng thái ngủ).

Sơ đồ cấu trúc:



Để điều khiển khối giao tiếp SPI thì chúng ta có 3 thanh ghi.Đó là 1 thanh ghi điều khiển:SPCR(SPI control Register).Một thanh ghi trang thái SPSR (SPI status Register) Và cuối cùng là thanh ghi dữ liệu SPDR(SPI Dât Register).

1.Thanh ghi SPCR:
Đây là thanh ghi 8 bit có địa chỉ trong các thanh I/O là 0x0D và trong SRAM là 0x2D.các bit trong thanh ghi này đều có thể đọc hoặc ghi.

Bit 7-SPIE: SPI interrupt enable
Bit này cho phép ngắt của bộ truyền tin SPI (nếu ngắt toàn cục và ngắt này được cho phép thì nếu cờ SPIF được bật thì ngắt đó sẽ được phục vụ.)
Bit 6-SPE: SPI Enable
Nếu bit này được set thì khối SPI sẽ được hoạt động và nó phải được set trong suốt quá trình SPI hoạt động.
Bit 5-DORD: Data order
Khi mà DORD được set thì LSB của byte dữ liệu sẽ được truyền trước và ngược lại.
Bit 4-MSTR: Master/Slave select
Đây là bit dùng để lựa chọn chế độ master hay slave.Nếu bit này được set thì bộ SPI này có vai trò là Master và ngược lại.Nếu như SS được cấu hình là lối vào và được đặt xuống mức thấp thì MSTR bị xóa về 0và SPIF và SPSR bị đặt lên 1 khi đó bạn sẽ phải đặt lại MSTR về 1.
Bit 3-CPOL: Clock polarity
Khi bit này được set thì SCK ở mức cao trong trang thái ngủ và ngược lai .
Bit 2-CPHA:Clock Phase
Quy định pha kích hoạt của xung nhip.
Bit 1,0-SPR1,SPR0 :Clock rate select:
Đây là hai bit điều khiển tốc độ xung nhịp truyền của kết nối và được thiết lập trên Master.Nó không có tác dụng gì nếu như ta thiết lập trên slave.
Và giá trị của chúng ứng theo tổ hợp các bit như sau:

SPR1	SPR0	Tần số SCK
0	0	Fcl/4
0	1	Fcl/16
1	0	Fcl/64
1	1	Fcl/128

Như vậy đây là thanh ghi điều khiển toàn bộ khối SPI từ vai trò (Master/slave đến tốc độ truyền,cho phép ngắt,cho phép hoạt động,mức logic trong trang thái ngủ và pha kích hoạt xung nhịp.

Ví dụ 1:

Thiết lập giao diện SPI với vai trò Master tốc độ truyền là Fcl/128 pha kích hoạt xung nhịp thấp không cho phép ngắt và chưa cho phép hoạt động.

Để là được điều đó trước hết các bạn cần phải thiết lập các chân cho SPI .Cụ thể SCK là chân PB7 là output đồng nghĩa với DDB7 được set.MISO(PB6) là chân vào vì vậy DDB6 xóa và để kích hoạt điện trở kéo thì PORTB6 phải được set lên 1 .MOSI(PB5) là chân ra do đó DDB5 cần phải set lên 1.SS(PB4) đây là chân lựa chon thiết bị Slave vì vậy được định nghĩa là chân ra và ở mức tích cực thấp (xóa DDB4 và PORTB4).

```
;Đoạn chương trình như sau:
sbi    0x17,7        ;set bit DDBR7 - đặt SCK là chân ra.
cbi    0x17,6        ;xóa bit DDBR6-đặt PB6 là cổng vào.
sbi    0x18,6        ;set bit PORTB6-Kích hoạt điện trở kéo.
sbi    0x17,5        ;Đặt PB5 là chân ra.
sbi    0x17,4
cbi    0x18,4
;Đặt xong cấu hình các chân
;Tác dụng lên từng bit để không ảnh hưởng đến các chân khác.
;Bây giờ tiếp tục đặt cấu hình cho SPI
OUT  0x0d,0b00010111
;hết.
```

Để thiết lập cho nó lá Slave thì hoàn toàn tương tự.

2.Thanh ghi SPSR:

Đây là thanh ghi 8 bit (có địa chỉ 0x0e/0x2e)lưu giữ trạng thái của bộ truyền nhận SPI.Nhưng nó chỉ có hai bt được định nghĩa có khả năng đọc và ghi.Các bit còn lại không được định nghĩa và khi đọc chúng thì có giá trị zero.

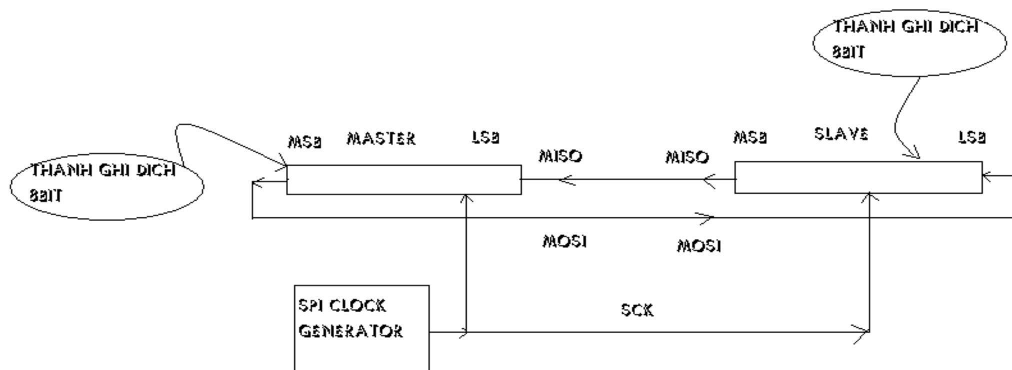
Bit 7-SPIF: SPI interrupt Flag

Khi truyền xong một byte dữ liệu thì bit này được set và một ngắt được tạo ra.Nếu bit cho phép ngắt SPIE trong thanh ghi SPCR được set và ngắt toàn cục được cho phép thì ngắt đó được thi hành.Nếu không nó sẽ bị bỏ qua.Khi mà chân ss của Master được định nghĩa là cổng vào lài được thiết lập mức thấp thì cờ này cũng được set.Nó được xóa bởi phần cứng hi ngắt được phục vụ.

Bit 6-WCOL: wite collision flag

Cờ báo xung đột khi ghi:Cờ này được set lên 1 nếu như dữ liệu được ghi lên thanh ghi dữ liệu SPI khi đang diễn ra một cuộc truyền.VÀ nó được xóa cùng với cờ SPIF khi đọc thanh ghi trạng thái và truy nhập vào thanh ghi dữ liệu.

Để bắt đàu một cuộc truyền thì các bạn cần cho phép bộ truyền nhận hoạt động.Khi truyền bạn chỉ cần ghi byte dữ liệu cần truyền lên thanh ghi dữ liệu và đợi cho tới khi có cờ SPIF bật lên rồi tiếp tục truyền byte mới.
Để bắt đầu nhận dữ liệu cũng vậy.SPI đã được khởi động,chờ khi nào cờ SPIF bật lên thì ta đọc dữ liệu (cờ tự xóa khi ta đọc thanh ghi trạng thái).
3.Thanh ghi SPDR:
Đây cũng là thanh ghi 8 bit (0x0f/0x2f) có thể đọc và ghi được.Nó được sử dụng để truyền dữ liệu giữa hai bộ truyền nhận SPI.Ghi dữ liệu vào thanh ghi này có nghĩa là ta bắt đầu cuộc truyền.Và đọc dữ liệu từ thanh ghi này là đọc dữ liệu được nhận.
4.Nguyên lý hoạt động:
xét hình sau:



Đây là sự ghép nối giữa hai bộ SPI song công (như của 2 vi điều khiển AVR).
Đối với VĐK AVR thì các chân SCK (Serial clock) là chân PB7,đây là chân xung nhịp ra trong trường hợp nó là Master và là chân xung nhịp vào nếu nó là Slave.khi ghi dữ liệu lên thanh ghi dữ liệu SPDR của khối Master sẽ khởi động bộ tạo xung và dữ liệu được dịch và đưa ra chân MOSI (PB5) và vào chân MOSI của slave (PB5 đối với AVR).Sau khi dịch hết một byte bộ tạo xung ngừng hoạt động,và cờ SPIF được phất báo kết thúc truyền.Nếu như ngắt này được phép thì chương trình phục vu ngắt sẽ được phục vụ và khi đó cờ sẽ bị xóa.Đầu vào lựa chọn slave (SS và là chân PB4) được set mức tích cực thấp để lựa chọn thiết bị SPI slave và được dùng cho việc ghép nối nhiều VĐK.Hai thanh ghi dịch của hai bộ truyền và nhận (Master và slave) được xem như là một thanh ghi dịch vòng 16 bit.Và trong một lần chao đổi dữ liệu thì dữ liệu ở thanh ghi của Master và slave đã chao đổi cho nhau.Một “bộ”SIP làm đồng thời cả hai nhiệm vụ truyền và nhận nhưng chúng lại chỉ có một bộ đệm khi truyền à có hai bộ đệm khi nhận.Như vậy có nghĩa là dữ liệu truyền đi sẽ không được ghi lên thanh ghi dữ liệu truyền nếu như byte

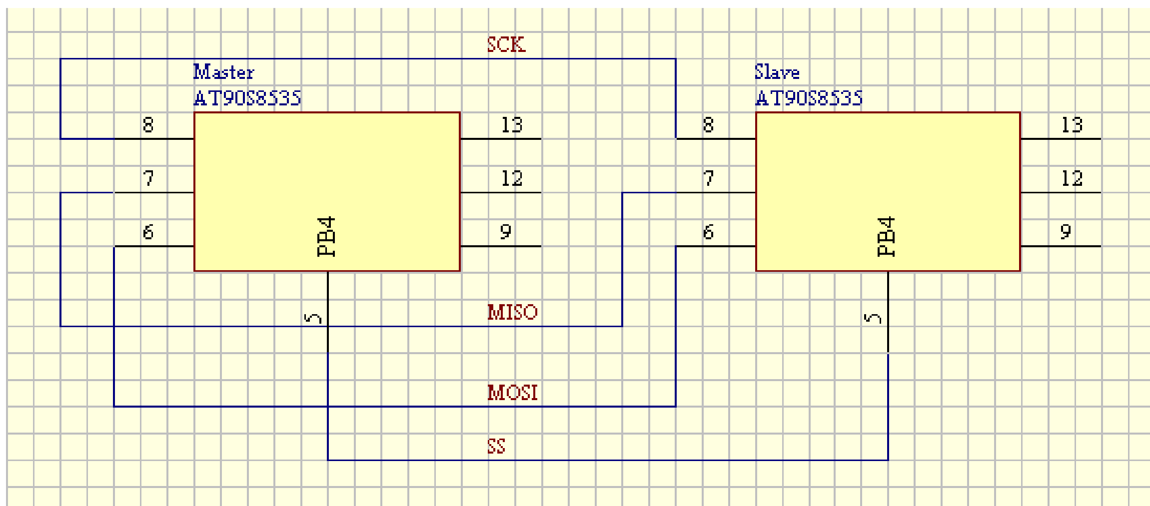
trước đó chưa được truyền xong (hay cờ SPIF chưa được bật).Và khi nhận dữ liệu cũng vậy dữ liệu cần phải được đọc trước khi dữ liệu mới được nhận xong.

Bảng cấu hình chân:

Chân	Trực tiếp,Master	Trực tiếp,Slave
MOSI	Người dùng định nghĩa	Chân vào
MISO	Chân vào	Người dùng định nghĩa
SCK	Người dùng định nghĩa	Chân vào
SS	Người dùng định nghĩa	Chân vào

Vậy như ví dụ 1 các bạn đã biết làm thế nào để cấu hình cho giao diện SPI

Sau đây là một ví dụ cụ thể:



Hai VĐK AVR được ghép với nhau theo giao diện SPI .Viết chương trình con lập một VĐK là master và cái còn lại lá Slave.Lấy 10 byte trong bộ nhớ SRAM kể từ vị trí 0xff gửi sang vi điều khiển thứ hai. Ở vi điều khiển thứ hai nhận 10 byte này và ghi vào SRAM kêt từ vị trí oxff.Biết chúng hoạt động cung xung nhip.

Chương trình với master:

```
; đoạn khởi tạo cổng
        sbi     0x17,7      ;set bit DDBR7 - đặt SCK là chân ra.
        cbi     0x17,6      ;xóa bit DDBR6-đặt PB6 là cổng vào.
        sbi     0x18,6      ;set bit PORTB6-Kích hoạt điện trở kéo.
        sbi     0x17,5      ;Đặt PB5 là chân ra.
        sbi     0x17,4
        cbi     0x18,4
        ;Đặt xong cấu hình các chân
        ;Tác dụng lên từng bit để không ảnh hưởng đến các chân khác.
        ;Bây giờ tiếp tục đặt cấu hình cho SPI
```

```
        LDI   R17, 0b01010111
        OUT   0x0d,R17      ;băt đầu hoạt động với xung nhịp Fcl/128
SENDFUNTION:
              ldi     R17,0x0a      ;khởi tạo số đếm
              ldi     xh,0x00       ;khởi tạo con trỏ
              ldi     xl,0xff       ;khởi tạo con trỏ
        SEND:

              Ld      R16,X+        ;lấy dữ liệu trong RAM
              Out     SPDR,R16
              Dec     R17
              Breq    exit          ;thoát nếu R17=0
              In      R16,SPSR      ;Lấy thanh ghi trang thái
              Bst     R16,7         ;lấy bit SPIF
              Brts    SEND          ;tiếp tục gửi khi byte trước đó được gửi.
        Exit:
              RET
```

Chương trình với Slave

Các bạn tự viết.

3.4. UART:

3.5.Watchdog timer:

3.6. Bộ nhớ ROM có thể được xóa bằng điện:

EEPROM

Bộ nhớ EEPROM là một bộ nhớ không bị mất dữ liệu khi nguồn điện cung cấp bị ngắt.Dữ liệu trong nó có thể được ghi và xóa bằng điện và vì vậy việc ghi và đọc bộ nhớ này VĐK có thể làm trực tiếp.

Bộ nhớ này được xem như một bộ nhớ dữ liệu nhưng chúng không được truy nhập như một bộ nhớ SRAM mà được truy nhập như một thiết bị ngoại vi.Thời gian truy cập để viết mất khoảng 2.5 đến 4 ms,và phụ thuộc vào nguồn điện cung cấp cho vi điều khiển (Vcc).

Để điều khiển vào ra dữ liệu với EEPROM chúng ta có thể sử dụng 3 thanh ghi đó là:EEPROM address ,EEDR và EECR.

1.EEPROM address :
Đây là thanh ghi 16 bit lưu địa chỉ của các ô nhớ của EEPROM (từ 0 đến 511). Nó được truy nhập như hai thanh ghi 8 bit độc lập EEARH và EEARL :

Bit	15	14	13	12	11	10	9	8	
$1F ($3F)	–	–	–	–	–	–	–	EEAR8	EEARH
$1E ($3E)	EEAR7	EEAR6	EEAR5	EEAR4	EEAR3	EEAR2	EEAR1	EEAR0	EEARL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	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	X	
	X	X	X	X	X	X	X	X	

2.EECR :EEPROM control register
Đây là thanh ghi điều khiển EEPROM

Bit	7	6	5	4	3	2	1	0	
$1C ($3C)	–	–	–	–	EERIE	EEMWE	EEWE	EERE	EECR
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Là thanh ghi 8 bit trong đó có 4 bit được định nghĩa để điều khiển hoạt động ghi đọc dữ liệu ở bộ nhớ EEPROM.
-Bit 3-EERIE :EEPROM ready interrupt enable
Bit này cho phép ngắt haọt động (thông báo cho CPU biết khả năng chao đổi dữ liệu với CPU.).Nếu bit này được set 1 thì nó được phép hoạt động.Và ngược lại.
-Bit 2-EEMWE: EEPROM master write enable.
Bit này khi được set 1 sẽ ghi dữ liệu từ thanh ghi EEDR vào ô nhớ có địa chỉ lưu trong thanh ghi EEAR của EEPROM. Bit này được set bằng phần mềm và được xóa bằng phần cứng sau bốn chu khi máy.
-Bit 1-EEWE: EEPROM write enable
Đây là bit cho phép ghi dữ liệu vào EEPROM để tránh trường hợp ta ghi dữ liệu khi mà một dữ liệu trước đó chưa được ghi xong.Nó được xáo bằng phần cứng khi mà sữ liệu đã được ghi xong vào EEPROM.
-Bit 0-EERE : EEPROM read enable
Bit này ra lệnh cho CPU đọc dữ liệu từ bộ nhớ này ra thanh ghi dữ liệu với địa chỉ đã lưu bên trong thanh ghi địa chỉ.Và nó được xóa bằng phần cừng khi mà dữ liệu đã được đọc ra thanh ghi dữ liệu.

Vậy để ghi dữ liệu vào EEPROM ta làm các bước sau:
Bước1:chờ đợi bit EEWE đã bị xóa chưa ?
Bước 2:Ghi dữ liệu mới vào thanh ghi dữ liệu (EEDR)
Bước3:Set bit EEWE rồi đến bit EEMWE để bắt đầu ghi dữ liệu.

Chý ý:Nếu như dang ghi dữ liệu ở EEPROM mà xuất hiện ngắt thì dữ liệu đó sẽ không được ghi một cách an toàn vào EEPOM.
Sau đây là một ví dụ:

Ghi 1 byte vào bộ nhớ EEPROM địa chỉ lưu trong thanh ghi Y ,dữ liệu trong R17.

```
Write:
        Sbic   EECR,1        ;nhảy qua nếu EEWE bị xóa.
        Rjmp   Write
        Write_start:
        Out    EEAL,YL
        Out    EEAH,YH
        Out    EEDR,R17      ;Nap dữ liệu vào thanh ghi dữ liệu.
        Cli                  ;cam tat ca ca ngat.
        Sbi    EECR,1
        Sbi    EECR,2
        Sei                  ;cho phép ngắt.
```

Để đọc dữ liệu vào EEPROM thì dơn giản hơn.

Bước1:kiểm tra bit EEWE nếu như có quá trình ghi EEPROM thì chờ đợi.

Bước2: Đưa địa chỉ cần đọc vào thanh ghi địa chỉ EEAR

Bươc3:Set bit EERE lên 1 bắt đầu quá trình đọc.

Bước4:Chờ đợi đoc xong bằng cách kiểm tra bit EERE nếu đã được xóa thì có dẽ liệu ở thanh ghi dữ liệu.Sau đó đọc dữ liệu.

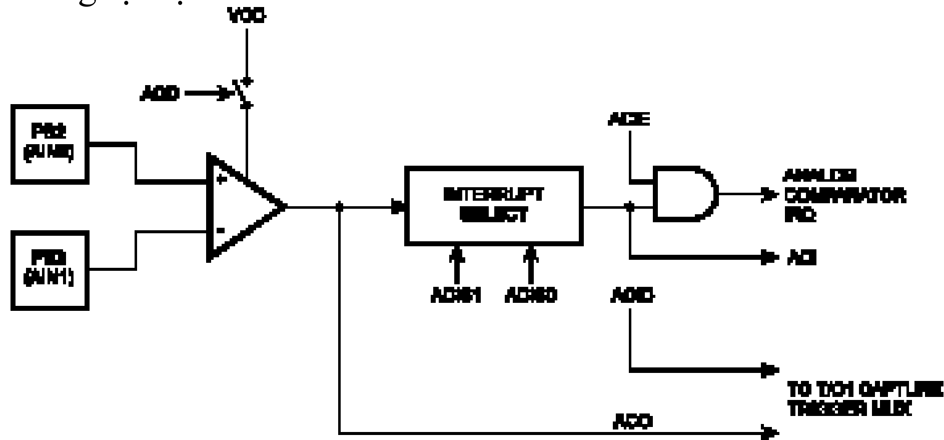
Ví dụ: Đọc một byte dữ liệu từ trong EEPROM bắt từ địa chỉ lưu trong thanh ghi Z và dữ liệu đọc ra thanh ghi R17

```
READ:
        Sbic   EECR,1        ;kiểm tra xem có ghi không?
        Rjmp   READ
    READ_START:              ;Bắt đầu đọc.
        Out    EEAL,ZL
        Out    EEAH,ZH
        Sbi    EECR,0
  Here:
        Sbic   EECR,0        ;kiểm tra EERE đã bị xóa hay chưa?
        Rjmp   here
        In     R17, EEDR
;Hết.
```

3.EEDR:EEPROM data register
Đây là thanh ghi 8 bit lưu dữ liệu lấy ra từ EEPROM hoặc dữ liệu định ghi vào EEPROM.

Bit	7	6	5	4	3	2	1	0	
$1D ($3D)	MSB							LSB	EEDR
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	

3.7. Bộ so sánh tương tự:
Analog comparator
Bộ so sanh tương tự của AVR có đầu vào lài hai chân PB2 và PB3 (như hình vẽ).Với chân PB2 được nối vào cực dương của bộ so sánh và PB3 được nối vào cực âm của bộ so sánh.Nó tạo ra hai mức logic nếu V+>V- thì tín hiệu ra là 1 và ngược lại là 0.



Để điều khiển và qua sát trạng thái của bộ so sánh tương tự ta có một thanh ghi đó là thanh ghi ACSR.Trước khi tìm hiểu về nguyên tắc hoạt động của nó tôi sẽ giới thiệu cho các bạn về thanh ghi này.
Thanh ghi ACSR là một thanh ghi 8 bit có địa chỉ trong các thanh ghi I/O là 0x08 và có địa chỉ trong không gian bộ nhớ SRAM là 0x28.Trong 8 bit thì có 7 bit được định nghĩa và bit 6 không được định nghĩa.Nó chỉ có thể đọc và luôn có giá trị logic là 0.

1.Bit 7-ACD:Analog comparator disable –Đây là bit điều khiển.
Bit này ttrực tiếp điều khiển hoạt động của AC(bộ so sánh tương tự).Nếu như bit này được set lên 1 thì nguồn cung cấp cho AC hoạt

động bị tắt (turn off) và đồng nghĩa với việc nó không hoạt động.Và nếu nó được xóa thì AC được cấp nguồn và hoạt động bình thường.Chú ý :Ta có thể thay đổi giá trị logic của bit này lúc nào cũng được để ngương hoạt động của chúng hoặc cho chúng hoạt đông trở lại nhưng khi thay đổi ghía trị logic của nó thì ngắt (ngắt của AC)cần bị cấm nếu không nó sẽ sinh ra một ngắt (Cụ thể là bit ACIE cần bị xóa).

2.Bit 5-ACO:Analog comparator output –Đây là bit trạng thái.
Bit này được nối trức tiếp với đầu ra của bộ so sánh tương tự.

3.Bit 4-ACI:Analog comparator interrupt flag –Đây là bit trạng thái.
Cờ báo ngắt của bộ so sanh tương tự.Nếu như cời này được set và các ngắt được phép thì một chương trình phục vụ ngắt được gọi và chúng đước xóa bằng phần cứng khi chương trinhg báo ngắt được phục vụ. Các trường hợp làm thay đổi trạng thái cờ ngày ngoài việc thay đổi bit ACD sẽ được nói tới trong các bít o và 1.

4.Bit 3-ACIE:AC interrupt enable –Đây là bit điều khiển.
Nếu bit này được set thì ngắt này được phép và ngược lại.

5.Bit 2ACIC:Analog comparator input Capture Enable –Đây là bit điều khiển.
Khi bit này được set lên 1 thì đầu ra của AC được nối trực tiếp vào đầu vào của chức năng bắt sự kiện của Timer/counter 1.(Đọc thêm timer/counter1).

6.Bit ACIS1 và ACIS0 :Ac interrupt mode select –Đây là hai bit điều khiển.

ACIS1	ACIS0	Chế độ ngắt
0	0	Theo mức
0	1	Dành riêng(chưa dùng đến)
1	0	Sườn suốn
1	1	Sườn lên

Chú ý:Các bit này cũng có thể được thay đổi bất cứ khi nào.Nhưng khi thay đổi thì ngaét của náo phải bị cấm.
Ta có thể sử dụng lệnh SBI hoặc CBIU để thay đổi trạng thái các bit trên thanh ghi này trừ bit ACI.Bi này sau khi được đọc cũng sẽ bị xóa (nếu nó được set).

Thiết port đầu vào cho bộ so sánh tương tự:

Hai chân PB2 và PB3 này cần được thiết lập là đầu vào vào bỏ điện trở treo.

Để lập trình cho AC ta bắt đầu các bước sau:

Bước 1:Thiết lập các chân đầu vào cha AC.
Bước 2:Chọn các chế độ cho AC ví như dùng ngắt …
Bước 3:Khởi động AC bằng cách xóa bit ACD .

Sau đây là một ví dụ:

Bạn có một bài toán đơn giản như sau: Điều khiển nhiệt độ của phòng sao cho nó nhỏ hơn 40độ.Dùng LM335 khi đó đầu vào bạn mắc trực tiếp vào PB2 AVR không thông qua ADC và đầu PB3 mắc vào giá trí điện áp tương ứng với 40 độ của LM335(ví như 3.5v chẳng hạn).Khi đó nếu nhiệt độ lớn hơn 40 độ thì đặt mức logic của PC0 lên 1 cho tới khi nó giảm xuống thì thôi.

Cách 1:không dùng ngắt(sử dụng vào ra bằng chương trình)

```
;Chương trình được viết như sau:
;Thiết lập cổng vào cho AC
cbi    DDBR,2      ;thiết lập chân PB2 là chân vào
cbi    PORTB,2     ;Loại bổ điện trở treo.
Cbi    DDBR,3      ;Thiết lập chân PB3 là chân vào.
Cbi    PORTB,3     ;Loại bỏ điện trở treo .
 ;Thiết lập cho AC
sbi    ACSR,0      ;Tạm ngừng hoạt động của AC
cbi    ACSR,3              ;Cấm ngắt
cbi    ACSR,0      ;Bắt đầu hoạt động
;Theo dõi AC
loop1:
        sbis   ACSR,ACO ;kiểm tra xem nhiệt độ có cao
        ;hơn ngưỡng đặt trước không.
        Rjmp          loop   ;nết không lớn hơn tiếp tục theo dõi.
        ;Nhiết độ cao hơn.
        ; Định nghĩa chân PC0 là chân ra:
        sbi    DDRC,0      ; Định nghĩa là chân ra
        sbi    PORTC,0     ; Đặt lên mức 1
loop2:
        sbic   ACSR,ACO
        Rjmp   Loop2
        Sbc    portc,0         ;xóa nếu nhiệt độ nhỏ hơn
        Rjmp   loop2
;the end.
```

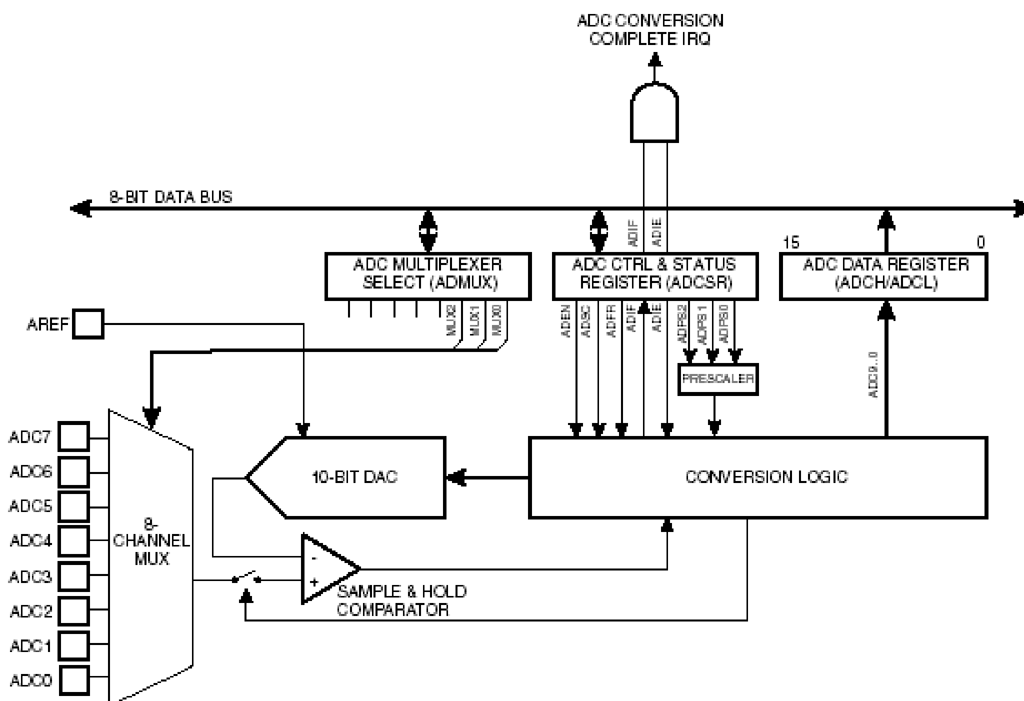
Cách 2: sử dụng ngắt:

3.8. ADC :analog to digital converter

Vi điều khiển AVR 8535 có một bộ biến đổi ADC tích hợp trong chip.
Có các đặc điểm:

Độ phân giải bit.
Sai số tuyến tính:0.5LSB.
Dộ chính xác +/-2LSB.
Thời gian chuyển đổi:65-260µs.
8 Kênh đầu vào có thể được lựa chọn.
Có hai chế độ chuyển đổi.
Có nguồn báo ngắt khi hoàn thành chuyển đổi.
Loại bỏ nhiễu trong chế độ ngủ.

Sơ đồ khối:



Từ sơ đồ khối các bạn có thể thấy:
Tám đầu vào của ADC là tám chân của PORTA và chúng được chọn thông qua một MUX.
Để điều khiển hoạt động vào ra dữ liệu của ADC và CPU chúng ta có 3 thanh ghi:ADMUX đây là thanh ghi điều khiển lựa chọn kênh đầu vào cho ADC . ADCSR Đây là thanh ghi điều khiển và thanh ghi trạng thái của ADC. ADCD :Đây là thanh ghi dữ liệu.
Sau đây là từng thanh ghi:

1.ADMUX: Multiplexer select register
Đây là thanh ghi điều khiển 8 bit:

Bit	7	6	5	4	3	2	1	0	
$07 ($27)	–	–	–	–	–	MUX2	MUX1	MUX0	ADMUX
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Với 3 bit được định nghĩa là MUX2,MUX1,và MUX0.Ứng với các tổ hợp logic các bạn có thể chọn kênh đầu vào.Cụ thể:

MUX2.0	Single-ended Input
000	ADC0
001	ADC1
010	ADC2
011	ADC3
100	ADC4
101	ADC5
110	ADC6
111	ADC7

Chú ý:Nếu như ta thay đổi kênh trong thời điểm mà ADC đanh chuyển đổi thì khi quá trình chuyển đổi đã hoang thiành thì khên vào mới được thay đổi.

2.ADCSR :ADC control and status register

Đây là thanh ghi điều khiển và lưu trạng thái của ADC:

Bit	7	6	5	4	3	2	1	0	
$06 ($26)	ADEN	ADSC	ADFR	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSR
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	

Bit 7-ADEN:ADC enable

Đây là bit điều khiển hoạt động của ADC .Khi bit này được set 1 thì ADC có thể hoạt động và ngược lại.Nếu như ta ngừng hoạt động của ADC trong khi nó đang chuyển đổi thì nó sẽ kết thúc quá trình chuyển đổi.Mặc dù chưa chuyển đổi xong.

Bit 6-ADSC: ADC start conversion

Tài liệu này tôi đang viết dở nhưng các bạn có thể đọc.
(Do thời gian có hạn nên đánh máy sai hơi nhiều mong các bạn cố dịch)
Mấy hôm nữa hoàn thanh hết tôi sẽ up tiếp lên cho các bạn.
Chúc các bạn thành công !
vutricongbka@yahoo.com