BỘ LAO ĐỘNG – THƯƠNG BINH VÀ XÃ HỘI
TỔNG CỤC DẠY NGHỀ
Dự án giáo dục kỹ thuật và dạy nghề (VTEP)

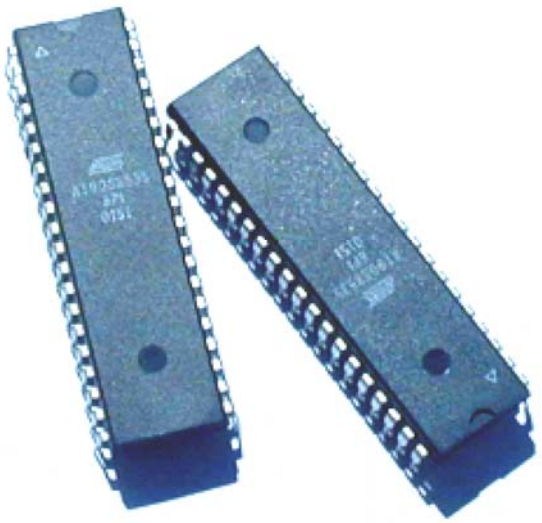
GIÁO TRÌNH

VI ĐIỀU KHIỂN NÂNG CAO

Mả số : CIO 02 27 02

NGHỀ : SỬA CHỮA ĐIỆN TỬ CÔNG NGHIỆP

Trình độ : 3



HÀ NỘI – 2004

Tổng cục dạy nghề cám ơn và hoan nghênh các thông tin giúp cho việc tu sửa và hoàn thiện tốt hơn tài liệu này

Địa chỉ liên hệ
Dự án giáo dục kỹ thuật và nghề nghiệp
Tiểu ban phát triển chương trình học liệu

Mã tài liệu :

Mã quốc tế ISBN :

LỜI TỰA

Tài liệu này là một trong các kết quả của dự án GDKT – DN được tài trợ bởi ngân hàng phát triển Á châu cho các trường kỹ thuật trọng điểm toàn quốc trực thuộc tổng cục dạy nghề.

Tài liệu được soạn là một giáo trình phục vụ cho đối tượng công nhân nghề sửa chửa điện tử công nghiệp. Do đó, trình tự nội dung được sắp xếp từ dể đến khó nhằm giúp người học tiếp thu một cách dể dàng. Đồng thời đi kềm với tài liệu còn có sổ tay hướng dẫn dành riêng cho giáo viên trong đó đề nghị các bước thực hiện quá trình giãng dạy một cách nhất quán từ đó tạo điều kiện cho giáo viên khai thác nội dung giá trình một cách tốt nhất

Đội ngủ biên soạn là nhóm CDC của trường công nhân kỹ thuật cần thơ, nội dung của tài liệu là sự kết hợp giữa yêu cầu đào tạo với tình hình công nghệ hiện tại trong thực tế sản xuất và cũng được tham khảo theo tình hình giảng dạy tại các trường kỹ thuật cũng như các cơ sở đào tạo nghề có liên quan.

Tài liệu này được thiết kế theo từng mô đun/ môn học thuộc hệ thống mô đun/ môn học của một chương trình đào tạo hoàn chỉnh nghề sửa chửa thiết bị điện tử công nghiệp ở cấp trình độ 3 và được dùng làm giáo trình cho học viên trong các khóa đào tạo.

Ngoài ra, tài liệu cũng có thể được sử dụng cho đào tạo ngắn hạn hoặc cho các công nhân kỹ thuật, các nhà quản lý và người sử dụng nhân lực tham khảo.

Đây là tài liệu thử nghiệm sẻ được hoàn chỉnh để trở thành chính thức trong hệ thống dạy nghề.

Hà Nội, ngày tháng năm 2005

Giám đốc Dự án quốc gia

MỤC LỤC

GIỚI THIỆU VỀ MÔ ĐUN

Vị trí, ý nghĩa, vai trò mô đun

- Đây là một mô đun chuyên ngành được học vào năm thứ hai của trình độ 3 sau khi học viên đã hoàn tất các mô đun hỗ trợ trước đó như: Linh kiện điện tử, mạch điện tử, kỹ thuật số.
- Trong các dây chuyền sản xuất, cũng như các thiết bị tự động đơn lẻ và các hệ nhúng hiện nay việc ứng dụng vi điều khiển đặc biệt là các họ vi điều khiển tich hợp nhiều khối ngoại vi trong các lĩnh vực này là rất phổ biến nhằm tăng tính linh hoạt, độ chính xác, giãm giá thành cũng như độ ổn định của hệ thống. Do đó, kiến thức về cấu trúc cũng như kỹ thuật lập trình về các loại vi điều khiển này là rất cần thiết cho công nhân ngành sửa chửa thiết bị điện tử công nghiệp.

Mục tiêu của môdun

Sau khi hoàn tất mô-đun này, học viên có năng lực:

- Hiểu, giải thích được nguyên lý làm việc các hệ điều khiển ứng dụng vi điều khiển AVR
- Cải tiến được chức năng của hệ điều khiển nhúng theo yêu cầu
- Phảt triển được các hệ điều khiển trên cơ sở khối trung tâm là vi điều khiển

Mục tiêu thực hiện của mô đun

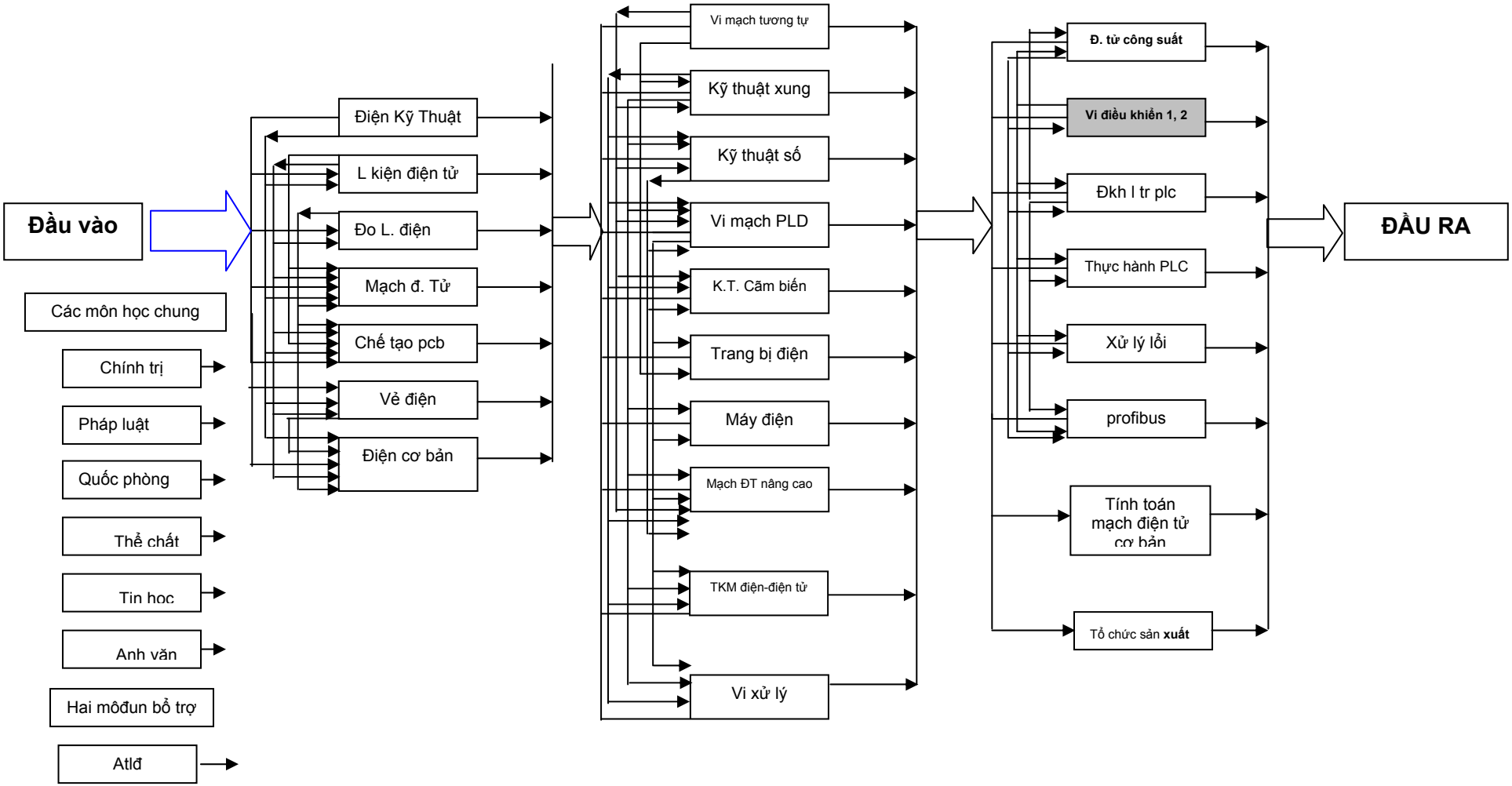
- Vận hành, kiểm tra được các hệ điều khiển ứng dụng AVR
- Sửa chửa được phần mềm và thay thế được linh kiện phần cứng
- Thi công, lắp ráp thiết bị theo sơ đồ có sẳn.

Nội dung chính của mô đun

Mô đun này bao gồm 3 bài học như sau :

1. Ngôn ngữ C51
2. Vi điều khiển AT90S8535
3. Vi điều khiển PIC16F8x

SƠ ĐỒ QUAN HỆ THEO TRÌNH TỰ HỌC NGHỀ



CÁC HÌNH THỨC HỌC TẬP CHÍNH TRONG MÔ ĐUN

Hình thức 1: Học lý thuyết trên lớp
- Tất cả các bài học từ 1 đến 3
- Giải các câu hỏi và bài tập phần lý thuyết
- Viết chương trinh điều khiển
- Phân tích chương trình có sẵn

Hình thức 2: Học thực hành trong xưởng
- Chạy thử các chương trình đã viết trên lớp
- Ráp mô hình đối tượng điều khiển
- Ráp toàn bộ mạch điều khiển kết hợp mô hình đối tượng điều khiển và kiểm tra hoạt động

Hình thức 3: Tự nghiên cứu
- Tự đề ra yêu cầu và thực hiện
- Tham khảo các vấn đề liên quan trên sách báo, internet...
- Tham quan thực tế sản xuất

YÊU CẦU VỀ ĐÁNH GIÁ HOÀN THÀNH MÔ ĐUN

Hiểu và thực hiện được các nội dung sau
- Phân tích được chương trình trên hệ thống thực.
- Phát triển được phần mềm theo yêu cầu
- Lắp ráp, vận hành và sửa chửa được hệ điều khiển dùng vi điều khiển

Về thái độ
- Chuyên cần, sáng tạo
- Luôn kiểm tra kết quả bằng nhiều phương pháp khác nhau để tăng mức độ tin cậy, chính xác.
- Cẩn thận, lưu ý đến các yêu cầu về an toàn

BÀI 1

Tên bài: **Ngôn ngữ lập trình C51** Mã bài: **CIO 02 27 01**

GIỚI THIỆU

Bài này trình bày về công cụ phần mềm hỗ trợ phát triển các ứng dụng trên nền họ 8051 đó là trình dịch C51, trình liên kết L51 và trình quản lý thư viện LIB51. Trình dịch C51 là trình dịch ngôn ngữ ANSI-C mở rộng với họ 8051, cũng giống như các ngôn ngữ lập trình cấp cao khác do đặc tính gần gũi với ngôn ngữ tự nhiên nên giúp đơn giản hóa quá trình viết phần mềm từ đó tạo khả năng thực hiện được những yêu cầu phức tạp nhanh chóng hơn so với hợp ngữ. Đây là một kiến thức bổ sung rất cần thiết cho những người đã từng quen thuộc với việc lập trình vi điều khiển bằng hợp ngữ.

Nội dung bài gồm cả lý thuyết và thực hành trên bộ thực tập UNIKIT, bài tập trong phần này là các chương trình hoàn chỉnh có thể được nạp vào EPROM để thực hiện

MỤC TIÊU THỰC HIỆN

- Hiểu được sự cần thiết và cơ chế hoạt động của trình dịch C51
- Biết được cấu trúc chung của chương trình viết bằng C51
- Sử dụng thành thạo các chỉ dẫn, các điều khiển và tập lệnh của trình dịch C51
- Biết cách tổ chức một chương trình kích thước lớn bằng cách phân chia thành các mô đun chương trình
- Hiểu được cách sử dụng trình liên kết L51 và trình quản lý thư viện LIB51
- Viết được chương trình điều khiển theo yêu cầu

NỘI DUNG CHÍNH

Nội dung bài học tập trung về các chủ đề chính như sau:

- Giới thiệu chung về ngôn ngữ ANSI-C
- Khuôn dạng một chương trình viết bằng C, cấu trúc dòng lệnh
- Các toán tử số học, logic và các toán tử đặc biệt khác
- Các chỉ dẫn và các điều khiển biên dịch của C51
- Các mở rộng phù hợp với họ vi điều khiển 8051
- Quy tắc gọi hàm
- Giao tiếp giữa C51 với hợp ngữ
- Hoạt động liên kết, trình liên kết L51
- Chức năng trình quản lý thư viện LIB51
- Các ví dụ lập trình bằng C51 và tạo chương trình thực thi bằng L51

1. KHÁI NIỆM CƠ BẢN VỀ ANSI-C

1.1 Tập ký tự dùng trong ngôn ngữ C

Mọi ngôn ngữ lập trình đều được xây dựng từ một bộ ký tự nào đó. Các ký tự được nhóm lại theo nhiều cách khác nhau để tạo thành các từ, tiếp theo đó các từ lại được liên kết theo một quy tắc nào đó để tạo thành các câu lệnh. Một chương trình bao gồm nhiều câu lệnh biểu diễn một thuật toán để giải quyết một bài toán xác định. Ngôn ngữ C được xây dựng trên bộ ký tự sau:

- 26 chữ cái hoa: ABC..Z
- 26 chữ cái thường: abc..z
- 10 chữ số: 0..9
- Các ký hiệu toán học: + - * / = ()
- Ký tự gạch nối dưới : _
- Các ký hiệu đặc biệt khác: . , ; : [] {} ? ! \ & | % # $,...

Dấu cách (space) thực sự là một khoảng trống dùng để tách các từ. VD: HA NOI gồm 6 ký tự trong khi đó HANOI chỉ có 5 ký tự

1.2 Từ khóa

Từ khóa là những từ có một ý nghĩa hoàn toàn xác định, thường được dùng để khai báo các kiểu dữ liệu, để viết các toán tử và các câu lệnh. Sau đây là các từ khóa của C:

asm	break	case	cdecl
char	const	continue	default
do	double	else	enum
extern	far	float	for
goto	huge	if	int
interrupt	long	near	pascal
register	return	short	signed
sizeof	static	struct	switch
typedef	union	unsigned	void
volatile	while		

Ý nghĩa và cách sử dụng chúng sẽ được bàn đến ở các phần sau, có hai điểm cần lưu ý, đó là:

- Không được dùng từ khóa để đặt tên cho các hằng, biến và hàm
- Từ khóa phải được viết bằng chữ thường

1.3 Tên

Tên là một khái niệm rất quan trọng, được dùng để xác định các đại lượng khác nhau trong một chương trình như tên hằng, tên biến, tên hàm…..

Tên được đặt theo cấu trúc như sau:

Tên là một dãy ký tự: Chữ, số và dấu gạch nối. Ký tự đầu của tên phải là chữ hoặc dấu gạch nối, độ dài mặc định của tên là 32, các ví dụ đúng về tên:

A_1 BETA x1 delta_7_x1

Các tên sau là sai

3XYZ_7	(ký tự đầu tiên là số)
r#3	(sử dụng ký tự #)
f(x)	(sử dụng dấu ngoặc tròn)
case	(trùng với từ khóa)
be ta	(sử dụng khoảng trắng)
X-1	(sử dụng dấu gạch ngang)

Trong các tên có phân biệt chữ hoa và chữ thường, do đó tên AB khác tên ab. Trong C thường dùng chữ hoa để đặt tên cho các hằng và chữ thường cho hầu hết các đạl lượng còn lại. Tuy nhiên, điều này không bắt buộc.

2. KIỂU DỮ LIỆU

Trong C sử dụng các kiểu dữ liệu sau: Số nguyên (int), số thực hay số dấu phẩy động (float), số dấu phẩy động có độ chính xác kép (double) và ký tự (char). Hằng chính là một giá trị thông tin cụ thể, biến và mảng là các đạl lượng mang tin. Mỗl loạl biến có thể chứa một dạng thông tin nào đó VD biến kiểu int chứa các số nguyên, biến kiểu float chứa các số thực. Để có thể lưu trữ được thông tin biến phảl được cấp phát bộ nhớ, biến lạl được chia thành biến tĩnh, biến tự động và biến ngoài.

Biến tự động chỉ tồn tạl (được cấp phát bộ nhớ) khi chúng đang được sử dụng, biến ngoài và tĩnh tồn tạl trong suốt thờl gian hoạt động của chương trình. Cách tổ chức như vậy vừa tiết kiệm bộ nhớ vừa cho phép sử dụng cùng một tên cho các đốl tượng khác nhau mà không gây ra nhầm lẫn.

2.1 Kiểu char

Một giá trị kiểu char chiếm một byte bộ nhớ và biểu diễn được một ký tự thông qua bảng mã ASCII. Ví dụ:

Ký tự	**Mã ASCII**
0	048
1	049
2	050
A	065
B	066
a	097
b	098

Có hai kiểu char là: Signed char và unsigned char. Kiểu thứ nhất biểu diễn một số nguyên từ -128 đến +127, kiểu thứ hai có giá trị từ 0 đến 255

Có thể chia 256 ký tự thành 3 nhóm:

- Nhóm thứ nhất là các ký tự điều khiển có mã từ 0 đến 31, các ký tự trong nhóm này không hiển thị ra màn hình.
- Nhóm thứ hai là các klý tự văn bản có mã từ 32 đến 126, các ký tự này có thể đưa ra màn hình và máy in.
- Nhóm ba là các ký tự đồ họa có mã từ 127 đến 255 có thể đưa ra màn hình.

2.2 Kiểu nguyên

Trong C cho phép sử dụng số nguyên (int), số nguyên dài (long) và số nguyên không dấu (unsigned)

Kiểu	*Phạm vi biểu diễn*	*Kích thước*
Int	-32768…+32767	2 byte
Unsigned int	0...65535	2 byte
Long	-2147483648…2147483647	4 byte
Unsigned long	0…4294967295	4 byte

Các kiểu ký tự cũng có thể xem là một dạng của kiểu nguyên

2.3 Kiểu dấu phẩy động

Trong C cho phép sử dụng ba loạl giá trị dấu phẩy động đó là: Float, double và long double được trình bày như sau:

Kiểu	*Phạm vi biểu diễn*	*Kích thước*
Float	3.4E-38…3.4E+38	4 byte
Double	1.7E-308...1.7E+308	8 byte

Long double 3.4E-4932…1.1E+4932 10 byte

2.4 Kiểu ENUM (liệt kê)

Cú pháp khai báo kiểu enum có 4 dạng sau:

Enum type_name {e1, e2,…} var_name1, var_name2,…;
Enum type_name {e1, e2,…} ;
Enum {e1, e2,…} var_name1, var_name2,…;
Enum {e1, e2,…} ;

Trong đó: type_name là tên kiểu dữ liệu enume
E1, e2,… là tên các phần tử
Var_name là tên các biến kiểu enume

Ý nghĩa:

Dạng thứ nhất có 3 chức năng sau

1. Định nghĩa các macro (giống như #define) e1, e2,…với các giá trị nguyên liên tiếp tính từ 0 (e1 = 0, e2 = 1), chức năng này tương đương các câu lệnh: #define e1 0 #define e2 1
2. Định nghĩa kiểu enume có tên là type_name
3. Khai báo các biến kiểu enume có tên là var_name1, var_name2

Dạng thứ hai có chức năng 1 và 2, dạng thứ ba có chức năng 1 và 3 và dạng thứ tư chỉ có chức năng 1

Biến kiểu enum thực chất là biến nguyên, nó được cấp phát 2 byte bộ nhớ và có thể nhận một giá trị nguyên bất kỳ. Mục đích của enum là tạo ra macro, các kiểu và các biến gợi nhớ. VD các ngày trong tuần có thể định nghĩa thông qua kiểu week_day và biến day như sau:

Enum week_day {Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday} day;

3. HẰNG

Hằng là các đạl lượng mà giá trị của nó không thay đổl trong quá trình tính toán, các loạl hằng sử dụng trong C gồm có:

3.1 Hằng dấu phẩy động

Được viết theo 2 dạng:

Dang thập phân: Gồm phần nguyên, dấu chấm thập phân và phần thập phân. VD:

214.35 -4563.48 234.0

Chú ý: Phần nguyên và phần thập phân có thể không có nhưng dấu chấm không thể thiếu. VD:

.34 15.

Dạng khoa học: Số được tách thành hai phần là phần định trị và phần bậc. Phần định trị là một số nguyên hoặc một số thực dạng thập phân, phần bậc là một số nguyên. Hai phần này cách nhau bởl ký tự e hoặc E. VD:

123.456E-4 biểu diễn giá trị 0.0123456
0.12E3 biểu diễn giá trị 120.0
-49.5e-2 biểu diễn giá trị -0.495
1E8 biểu diễn giá trị 100000000.0

3.2 Hằng số nguyên

Là số nguyên có giá trị trong khoãng từ -32768…+32767. VD:

-45 4007 635…

Chú ý phân biệt 125 và 125.0. Giá trị 125 là hằng số nguyên còn 125.0 là hằng số thực

Hằng long được viết như sau:

-4893L hoặc -4893l

Thêm L hoặc l vào sau đuôi. Một số nguyên vượt ra ngoài phạm vi của int cũng được xem là hằng long VD:

4563946L và 4563946
là hai hằng long có cùng giá trị

Hằng nguyên hệ 16 sử dụng 16 ký tự: 0...9 và A...F hoặc a...f, cách viết như sau:

0xc1c2c3… hoặc 0Xc1c2c3…
Trong đó ci là một chữ số hệ 16. VD:

0xa9, 0Xa9, 0xA9, 0XA9 là các hằng nguyên hệ 16 có cùng giá trị thập phân là 169

3.3 Hằng ký tự

Là một ký tự riêng biệt được viết trong hai dấu nháy đơn. VD: 'a', giá trị của 'a' chính là mã ASCII của chữ a. Như vậy, giá trị của 'a' là 97. Hằng ký tự có thể tham gia vào các phép toán như mọI số nguyên khác.VD:

'9' – '0' = 57 – 48 = 9
ĐốI vớI các hằng ký tự đặc biệt phảI dùng cách viết sau:

Cách viết	**Ký tự**
'\''	'
'\" '	"
'\\'	\
'\n'	xuống dòng
'\0'	\0 (null)
'\t'	tab
'\b'	backspace
'\r'	CR (về đầu dòng)
'\f'	LF (sang trang)

- Cần phân biệt hằng '0' và '\0', hằng thứ nhất là số 0 có mã là 48 còn hằng thứ hai ứng với ký tự \0 (thường gọi là ký tự null) có mã là 0.
- Hằng ký tự thực sự là một số nguyên nên có thể dùng các số nguyên hệ 10 để biểu diễn ký tự

3.4 Hằng xâu ký tự

Là một dãy ký tự bất kỳ đặt trong hai dấu nháy kép. Ví dụ:

"Hà Nội"
"Hải Phòng"
"" (xâu rỗng)

Xâu ký tự được lưu trữ trong máy dưới dạng một mảng có các phần tử là các ký tự riêng biệt. Trình biên dịch tự động thêm ký tự null vào cuối mỗi xâu để báo kết thúc xâu.

Lưu ý 'a' là hằng ký tự được lưu trữ trong 1 byte còn "a" là hằng xâu ký tự được lưu trữ trong 1 mảng gồm hai phần tử: Phần tử thứ nhất chứa chữ a và phần tử thứ hai chứa \0

3.5 Tên hằng

Tác dụng của toán tử

```
#define MAX 1000
```

là: tất cả các tên MAX trong chương trình xuất hiện sau toán tử này đều được thay bằng 1000 vì vậy MAX thường được gọi là tên hằng hay macro nó biểu diễn số 1000, một ví dụ khác

```
#define PI 3.14
```

Đặt tên cho hằng số π là PI

Các hằng kiểu int, long, float, double thường dùng trong tính toán, còn các hằng ký tự và xâu ký tự thường dùng trong in ấn để đưa ra các dòng thông tin, văn bản, ghi chú…

4. BIẾN

Mọi biến cần phải được khai báo trước khi sử dụng, mẫu khai báo như sau

```
Type      var_name;
```

Ví dụ:

```
Main ( )
{
  int       a, b, c; /* 3 biến kiểu integer */
  char      d, e, f; /* 3 biến kiểu char */
  float     x, y, z; /* 3 biến kiểu float */

…….

}
```

Biến kiểu int chỉ nhận các giá trị int, các biến kiểu khác cung có đặc tính tương tự, ví dụ biến kiểu char chỉ chứa một ký tự, để lưu trữ một xâu ký tự cần phải dùng một mảng kiểu char

Các khai báo biến phải đượ đặt ngay sau dấu { đầu tiên của thân hàm và phải đứng trước mọi câu lệnh khác.

Các biến có thể được khởi gán ngay khi khai báo bằng cách đặt thêm dấu = kèm theo một giá trị Ví dụ:

```
Int       a, b = 20, c, d = 40;
```

Kết quả này cũng tương đương bằng các câu lệnh gán sau:

```
Int       a, b, c, d;
          b = 20;
          d = 40;
```

Mỗi biến được cung cấp một vùng nhớ gồm nhiều byte liên tiếp, địa chỉ byte đầu tiên chính là địa chỉ của biến, địa chỉ biến nhận đươc qua phép toán

```
& var_name
```

5. MẢNG

Mỗi biến chỉ có thể biểu diễn được một giá trị. Để biểu diễn một dãy số hoặc một bảng số có thể dùng nhiều biến nhưng cách này không thuận tiện khi số giá trị quá nhiều, cách tốt nhất là sử dụng mảng trong những trường hợp như thế này.

Mảng là một tập hợp gồm nhiều phần tử có cùng kiểu dữ liệu và có chung một tên, mỗi phần tử mảng biểu diễn được một giá trị. Có bao nhiêu kiểu dữ liệu thì cũng có bấy nhiêu kiểu mảng, mảng cần được khai báo về:

- Kiểu mảng (int, char,…)
- Tên mảng
- Số chiều và kích thước mỗi chiều

Kiểu mảng và tên mảng cũng giống như kiểu biến và tên biến. Ví dụ các khai báo sau đây:

```
Int     a[10], b[4][2];
Float   x[5], y[3][3];
```

Sẻ xác định 4 mảng: a, b, x và y như sau:

- Mảng thứ nhất có kiểu là int, tên là a, số chiều là 1, kích thước là 10. Mảng gồm 10 phần tử có số thứ tự là: a[0], a[1], a[2],…,a[9]. Mỗi một phần tử a[i] chứa một giá trị int và mảng này biểu diễn một dãy gồm 10 số nguyên.
- Mảng thứ hai có kiểu là int, tên là b, số chiều là 2, kích thước các chiều là 4 và 2. Mảng gồm 8 phần tử được đánh số và sắp xếp như sau:

b[0][0]	b[0][1]
b[1][0]	b[1][1]
b[2][0]	b[2][1]
b[3][0]	b[3][1]

Mỗi phần tử b[i][j] chứa được một giá trị kiểu int, và mảng này biểu diễn được một bảng số nguyên gồm 4 hàng và 2 cột

- Mảng thứ ba có kiểu là float, tên là x, số chiều là 1, kích thước là 5. Mảng gồm 5 phần tử được đánh số như sau: x[0], x[1], x[2], x[3], x[4]. Mỗi phần tử x[i] chứa được một giá trị kiểu float và mảng này biểu diễn được một dãy gồm 5 số thực
- Mảng thứ tư có kiểu là float, tên là y, số chiều là 2, kích thước các chiều là 3, mảng gồm 9 phần tử được đánh số và sắp xếp như sau:

y[0][0]	y[0][1]	y[0][2]
y[1][0]	y[1][1]	y[1][2]
y[2][0]	y[2][1]	y[2][2]

Mỗi phần tử y[i][j] chứa được một giá trị kiểu float và mảng y biểu diễn một bảng số thực 3 hàng 3 cột

Chú ý:

- Các phần tử của mảng được cấp phát các khoảng nhớ liên tiếp nhau có nghĩa là địa chỉ của chúng cũng liên tiếp nhau
- Trong mảng hai chiều các phần tử của mảng được sắp xếp theo hàng.

Một phần tử của mảng được xác điịnh nhờ chỉ số của nó, chỉ số của mảng phải có giá trị int và không vượt quá kích thước của chiều tương ứng. Số chỉ số phải bằng số chiều của mảng.

Giả sử a, b, x, y đã được khai báo như trên và giả sử I, j là 2 biến nguyên. Trong đó I = 2; j = 1 thì

a[j+i-1]	là a[2]
b[j+i][2-i]	là b[3][0]
x[j/i]	là x[0]
y[i][j]	là y[2][1]

Và các cách viết sau đây là sai

y[j] vì y là mảng 2 chiều cần có 2 chỉ số
b[i][j][1] vì b cũng là mảng 2 chiều chỉ cần 2 chỉ số

Tuy nhiên, biểu thức dùng làm chỉ số có thể là số thực, khi đó phần nguyên sẻ là chỉ số mảng. VD: a[2.4] là a[2], a[1.9] là a[1]. Khi chỉ số vượt ra ngoài kích thước mảng máy vẫn không báo lỗi nhưng sẻ truy cập đến vùng nhớ ngoài mảng có dữ liệu không biết trước.

Có thể lấy địa chỉ của phần tử mảng một chiều và thường không lấy được địa chỉ của phần tử mảng nhiều chiều, phép tính sau lấy địa chỉ phần tử mảng 1 chiều

```
& a[i]
```

nhưng máy không chấp nhận phép tính

```
& y[i][j]
```

Một lưu ý quan trọng là tên mảng chính là địa chỉ đầu của mảng

```
a = & a[0]
```

6. TYPEDEF

Từ khóa typedef được dùng để đặt tên cho một kiểu dữ liệu, tên kiểu lại được dùng để khai báo kiểu dữ liệu sau này. Nên chọn tên ngắn gọn và gợi nhớ

Đặt typedef trước một khai báo thông thường, khi đó tên dữ liệu sẻ trở thành tên kiểu. VD:

```
Typedef int nguyen;
```

Sẻ đặt tên kiêu dữ liệu int là nguyen, từ lúc này trở đi có thể dùng int hoặc nguyen để khai báo biến như sau:

```
Nguyen  x, y, a[10], b[20][30];
```

Các ví dụ tương tự

typedef float mt[50]; // đặt tên mảng thực 1 chiều 50 phần tử là m50

typedef int m_20_30[20][30]; // mảng nguyên 2 chiều 20 hàng 30 cột tên là m_20_30

typedef enum {T1, T2, T3} T; // kiểu enum có 3 phần tử {T1, T2, T3} có tên là T

Sau đó, có thể dùng các tên này để khai báo

mt50 x, y; // các mảng thực 1 chiều có 50 phần tử

m_20_30 a, b; // các mảng nguyên 2 chiều 20x30

T t; // t là biến enum

7. CÁC LOẠI BIẾN

Biến là một đại lượng chứa dữ liệu và dữ liệu này bị thay đổi trong suốt thời gian chạy chương trình, các biến được nhận dạng thông qua tên biến, biến phải được khai báo trước khi đem ra sử dụng, mỗi biến phải thuộc về một kiểu dữ liệu.

7.1 Biến tự động

Các biến khai báo bên trong các hàm là các biến tự động hay biến cục bộ. Đối của hàm cũng được xem là biến tự động.

- Các biến tự động chỉ có tác dụng bên trong thân của hàm mà tại đó chúng được khai báo.
- Các biến tự động của hàm chỉ tồn tại trong khoãng thời gian kể từ khi máy bắt đầu làm việc với hàm cho đến khi thoát khỏi hàm.
- Do chương trình bắt đầu làm việc từ câu lệnh đầu tiên của hàm main ( ) và khi ra khỏi hàm main ( ) thì chương trình kết thúc nên các biến khai báo trong hàm main ( ) sẻ tồn tại trong suốt thời gian làm việc của chương trình.

7.2 Biến ngoài

Biến được khai báo bên ngoài các hàm gọi là biến ngoài

- Biến ngoài tồn tại (được cấp phát bộ nhớ) trong suốt thời gian làm việc của chương trình.
- Phạm vi hoạt động của biến ngoài là từ vị trí khai báo đến cuối file chương trình. Như vậy nếu một biến ngoài được khai báo ở đầu chương trình (trước tất cả các hàm) thì nó có thể được sử dụng trong bất kỳ hàm nào miễn là hàm đó không có biến cục bộ nào trùng tên với biến ngoài này.
- Nếu chương trình được viết trên nhiều file và các file được dịch độc lập thì phạm vi sử dụng của biến ngoài có thể mở rộng từ file này sang file khác bằng từ khóa extern

Các quy tắc khởi đầu biến ngoài:

1. Các biên ngoài có thể được khởi đầu vào lúc dịch chương trình bằng cách dùng các biểu thức hằng. Nếu không được khởi đầu máy sẻ gán cho chúng giá trị là 0.

Ví dụ:

```
char sao = '*' ;
int a = 6*365 ;
float x =32.5 ;
```

2. Khi khởi đầu mảng ngoài có thể không cần chỉ ra kích thước (số phần tử) của mảng. Khi đó, máy sẻ tự động dành cho mảng một khoảng nhớ đủ để chứa danh sách giá trị khởi đầu.

```
float a[ ] = {2.6, 3, 15} ;
int [ ] [ ] = {
                {25, 31},
                    {46, 34},
                    {93, 81}
                  } ;
```

3. Khi chỉ ra kích thước của mảng thì kích thước này không được nhỏ hơn bộ giá trị khởi đầu.

Ví dụ:

```
float x[8] = {6.3, 12.5} ;
```

4. Đối với mảng 2 chiều có thể khởi đầu theo các cách sau (số khởi đầu trên mỗi hàng có thể khác nhau)

Ví dụ:

```
float a[ ][3] = {
                    {0},
                    {2.5, 3.6, 8},
                    {-6.3}
                    } ;
```

5. Bộ khởi đầu của 1 mảng char có thể
   - Hoặc là danh sách các hằng ký tự
   - Hoặc là một hằng xâu ký tự

Ví dụ:

```
char name[ ] = {'h', 'a', 'n', 'g', '\0'} ;
char name[ ] = "hang" ;
```

7.3 Biến tĩnh

Biến tĩnh được khai báo bằng từ khóa static đặt phía trước

```
static int a, b, c[10] ;
static float x, y[10][6] ;
```

Các khai báo trên có thể đặt bên trong hoặc bên ngoài các hàm. Nếu đặt bên trong là biến tĩnh trong và ngược lại là biến tĩnh ngoài. Các biến tĩnh trong và ngoài có điểm giống nhau là:

- Chúng được cấp phát bộ nhớ trong suốt thời gian hoạt động của chương trình. Do đó giá trị của chúng được lưu trữ từ đầu đến cuối chương trình
- Có thể được khởi đầu một lần khi dịch chương trình nhờ các biểu thức hằng

Các biến tĩnh trong và ngoài chỉ khác nhau ở phạm vi hoạt động.

- Các biến tĩnh trong chỉ hoạt động được bên trong thân của hàm mà tại đó chúng được khai báo.
- Phạm vi hoạt động của các biến tĩnh ngoài tính từ khi chúng được khai báo cho đến cuối file chương trình chứa chúng

8.TOÁN TỬ

Toán hạng (operand) có thể xem là một đại lượng có một giá trị nào đó ví dụ: Hằng, biến, mảng và hàm. Toán là là các phép toán tác động lên các toán hạng. Trong C toán hạng được chia thành các nhóm chính như sau:

- Toán tử số học (arithmatic operator)
- Toán tử quan hệ (relative operator)
- Toán tử logic (logical operator)
- Toán tử xử lý bít (bitwise operator)

Ngoài ra, C còn giới thiệu thêm 2 toán tử mối đó là:

- Toán tử tăng (incrementing operator)
- Toán tử giãm (decrementing operator)

8.1 Toán tử gán

Theo C ký hiệu = không có nghĩa là bằng, ký hiệu này được gọi là toán tử gán (assignment operator), toán tử gán được gọi là gán đơn (single assignment) để phân biệt với gán phức hợp (compound assignment)

```
lulu = 2002 ;
```

Gán giá trị nguyên 2002 cho biến tên là lulu, toán tử gán cho phép gán một trị mới cho một biên. Dạng đơn giản nhất là:

```
biến = biểu thức ;
```

Nghĩa là biểu thức ở bên phải dấu = sẻ được định trị và kết quả định trị sẻ được gán cho biến ở phía trái dấu =. Nói cách khác ở bên trái dấu = phải là biến chứ không được là hằng còn ở bên phải là trị của biến, xem câu lệnh sau

```
I = I + 1 ;
```

Về mặt biểu diễn toán học thì không đúng, nhưng đối với máy tính thì hoàn toàn hợp lệ, giá trị của biến I được cộng với 1 rồi đem kết quả gán trở lại vào I, nhưng câu lệnh sau

```
2002 = lulu ;
```

Thì hoàn toàn sai vì 2002 là 1 hằng, không thể lấy giá trị biến lulu gán lên 1 hằng số

Ví dụ:

```
toc_do = khoang_cach / thoi_gian ;
gia_ban = gia_mua + chi_phi + lai ;
```

8.2 Toán tử cộng

Toán tử cộng (+) sẻ cộng hai trị ở hai bên dấu + lại với nhau

```
thu_nhap = luong + tien_tham_nhung ;
```

Máy sẻ cộng giá trị hai bên dấu + và sau đó gán vào biến thu_nhap, tón tử + là loại toán tử 2 ngôi (binary operator) vì nó cần đến 2 toán hạng)

8.3 Toán tử trừ

toán tử trừ lấy giá trị trước dấu – trừ đi giá trị sau dấu trừ

```
tien_dem_ve = thu_nhap – tien_hoi_lo
```

Dấu trừ còn được dùng để thay dấu đại số của một giá trị. Ví dụ:

```
toto = -12 ;
lili = - toto ;
```

Dấu trừ được dùng theo cách này được gọi là toán tử một ngôi vì nó chỉ cần một toán hạng

8.4 Toán tử nhân

Toán tử nhân dùng dấu *, ví dụ:

```
luong_thang = luong_ngay * 30 ;
```

8.5 Toán tử chia

C dùng ký hiệu / làm toán tử chia. Số bị chia là trị ở bên trái và số chia là trị ở bên phải

```
Ngay_trong_tuan = 30 / 4 ;
```

Cách chia số nguyên khác với cách chia số thực, chia theo số thực sẻ cho ra kết quả là một số thực. Khi chia hai số nguyên với nhau kết quả sẻ là phần nguyên còn phần thập phân sẻ bị cắt bỏ. Nếu trong phép chia có số nguyên và số thực lẫn lộn thì kết quả sẻ là số thực vì số nguyên sẻ được chuyển thành số thực trước khi chia.

8.6 Toán tử modulus

Toán tử modulus ký hiệu là % sẻ cho ra số dư của phép chia, ví dụ:

```
13 % 5
```

Đọc là 13 modulus 5 sẻ cho ra kết quả là 3, đây chính là số dư của phép chia 13 cho 5. Toán tử này chỉ làm việc với số nguyên

8.7 Toán tử tăng / giãm

Toán tử tăng (++) tăng giá trị của toán hạng lên 1 còn toán tử giãm (--) thì làm giãm giá trị của toán hạng xuống 1

```
toto++ ;
kiki -- ;
```

Tương đương với hai lệnh sau

```
toto = toto + 1 ;
kiki = kiki – 1 ;
```

Các toán tử này có thể viết theo 2 cách prefix mode hoặc postfix mode có nghĩa là có thể viết phía trước hay sau toán hạng

```
++toto   hoặc   toto++
--kiki   hoặc   kiki—
```

Nếu viét ++ trước thì giá trị của toán hạng sẻ được tăng 1 trước (preincrement) sau đó kết quả này mới được dùng cho các thao tác khác, còn nếu viết sau thì giá trị của toán hạng sẻ được dùng trước rồi mới tăng 1 sau (postincrement), tương tự đối với toán tử giãm –

8.8 Toán tử gán phức hợp

C còn cung cấp nhiều phép gán phức hợp để rút ngắn cách viết các biểu thức. Ví dụ:

```
ctr = ctr + 3 ;
ctr = ctr * 3 ;
ctr = ctr – 3 ;
ctr = ctr / 3 ;
ctr = ctr % 3 ;
```

Nhận xét là ở hai bên dấu gán đều có cùng một tên biến, để rút ngắn có thể viết như sau:

```
ctr += 3 ;
ctr *= 3 ;
ctr -= 3 ;
ctr /= 3 ;
ctr %= 3 ;
```

8.9 Toán tử sizeof

Toán tử sizeof cho biết kích thước (tính theo byte) của một kiểu dữ liệu cũng như một đối tượng dữ liệu, cú pháp như sau:

```
sizeof (kiểu dữ liệu)
sizeof đối tượng dữ liệu
```

Kiểu dữ liệu có thể là kiểu chuẩn như: int, float và kiểu được đinh nghĩa bằng typedef, enum, struct, union.

Đối tượng dữ liệu bao gồm biến, mảng, cấu trúc…(tên của vùng nhớ dữ liệu)

Toán tử này thường dùng để xác định số phần tử của một mảng, lưu ý nếu là kiểu dữ liệu thì phái đặt trong dấu ngoặc đơn.

8.10 Toán tử ép kiểu

Trong các biểu thức hổn hợp có chứa nhiều kiểu dữ liệu thì việc chuyển đổi được thực hiện một cách tự động. Tuy nhiên, lập trình viên cung có thể quyết định việc chuyển

kiểu thông qua toán tử ép kiểu bằng cách đặt trước toán hạng cần chuyển kiểu dữ liệu muốn chuyển trong ngặc đơn., cú pháp chung như sau:

(kiểu dữ liệu) biểu thức

x = (float) I / j ; dữ liệu biến I được chuyển thành float
a = 1.5 + 1.5 kết quả a là 3.0
a = (int) 1.5 + (int) 1.5 kết quả a là 2

9. BIỂU THỨC VÀ CÂU LỆNH

Biểu thức (expression) là sự kết hợp giữa các toán tử với các toán hạng và câu lệnh (statement) thì lại được tạo nên từ các biểu thức

9.1 Biểu thức

Một vài ví dụ về biểu thức

4
-6
4 + 21
a* (b + c/d) /20
q = 5*2
x = ++q %3
q > 3
6 + (c = 3+8)
x = y/3 + power(i, 2)

Từ các ví dụ trên cho thấy các toán hạng trong các biểu thức có thể là hằng, biến hoặc hàm (hàm power). Nói chung, biểu thức là một sự kết hợp hợp lệ của các phép toán được thực hiện trên trị của các biến, trị hằng hay trị do các hàm trả về. Một vài biểu thức lại là tổ hợp của nhiều biểu thức nhỏ.

Đặc tính quan trọng của C là mỗi biểu thức đều có một trị, muốn xác định trị của biểu thức phải dựa trên thứ tự nưu tiên của các toán tử (operator precedence) để thực hiện các phép toán.

9.2 Biểu thức hằng

Một biểu thức toán học được xem là một biểu thức hằng nếu trong biểu thức đó các toán hạng đều là các hằng số hoặc hằng ký tự nghĩa là trong biểu thức hằng không có chứa biến. Khi đó, trình biên dịch sẻ định trị trước biểu thức hằng và đem đi lưu trữ. Biểu thức hằng thường được dùng nhiều trong các cấu trúc vòng lặp hoặc cấu trúc chọn lựa.

8* 20 -13 kết quả là 147
'a' – 'A' kết quả là 32

Đây là những biểu thức hằng

Khả năng tính trước này làm giãm thời gian cần thiết để chạy chương trình vì máy không cần tính lại những biểu thức này nữa

9.3 Các câu lệnh

Chương trình là một tập hợp các câu lệnh, mỗi câu lệnh được kết thúc bởi dấu chấm phẩy. Đối với máy câu lệnh là một chỉ thị đầy đủ

tay = 2

là một biểu thức hoặc có thể là thành phần của một biểu thức lớn hơn, nhưng

tay =2 ;

lại là một câu lệnh vì được kết thúc bằng dấu chấm phẩy. Biểu thức sau đây:

```
2 + 2
```

Không phải là một chỉ thị đầy đủ vì máy sẻ không biết phải làm gì nữa sau khi cộng xong. Tuy nhiên, nếu ra lệnh

```
toto = 2 + 2 ;
```

có nghĩa là sau khi cộng xong đem kết quả vào trong biến toto và máy có thể thực hiện thao tác kế tiếp

9.4 Câu lệnh phức hợp (khối lệnh)

Câu lệnh phức hợp (compound statement) bao gồm 2 hoặc nhiều câu lệnh đơn gộp chung lại đặt trong một cặp dấu ngoặc nghoéo { } và còn được gọi là khối lệnh. Ví dụ:

```
/* trường hợp 1 */
index = 0 ;
while (index++ < 10)
  toto = 10* index + 2 ;
  printf (“toto = %d\n”, toto) ;

/* trường hợp 2 */
index = 0 ;
while (index++ < 10)
  {
    toto = 10* index + 2 ;
    printf (“toto = %d\n”, toto) ;
  }
```

Trong trường hợp 1 chỉ có câu lệnh gán toto = 10* index + 2 là nằm trong vòng lặp while. Do đó, lệnh in printf ( ) chỉ được thực hiện một lần sau khi kết thúc vòng lặp.

Trong trường hợp 2 vì có cặp dấu ngoặc { } nên câu lênh gán và lệnh in đều nằm trong vòng lặp và lệnh in sẻ được thực hiện mỗi vòng lặp một lần, lúc này câu lênh phức hợp được xem như một lệnh đơn.

10. HÀM TRONG C

Một chương trình C gồm một hoặc nhiều đơn thể chương trình được gọi là hàm (function) ghép lại, thường hàm trả về cho chương trình một trị sau khi được thực hiện.

Hàm cho phép chia một chương trình lớn thành những chương trình nhỏ nhằm dể quản lý và do đó làm đơn giản hóa công việc lập trình. Ngoài ra, một hàm hoạt động tốt ở chương trình này có thể được ghép vào một chương trình khác mà không cần phải viết lại từ đầu nên giúp giãm thời gian phát triển chương trình. Hàm được chia làm 2 loại:

- Hàm do người dùng viết gọi là user defined function (UDF). Đây là những hàm thực hiện các công việc cụ thể nào đó tại nhiều nơi trong chương trình
- Hàm thư viện chuẩn của C như hàm printf ( ). C cung cấp một số hàm thư viện chuẩn khá phong phú để tính toán, xử lý chuỗi, xử lý bít, xuất nhập dữ liệu…

10.1 Cấu trúc của hàm

Đặc điểm chung của hàm:

1. Được đặt chung trong cùng một file chương trình nguồn hoặc ghép từ một file nguồn khác nhờ chỉ thị #include hoặc được biên dịch riêng rẻ sau đó liên kết lại để tạo thành file thực thi được.
2. Được gọi từ chương trình chính main ( ) hoặc từ một hàm khác hoặc từ chính nó (đệ quy)

3. Có hoặc không có đối muc (argument)
4. Có hoặc không có trị trả về (return value)
5. Mỗi hàm chỉ có một điểm nhập chính là lệnh đầu tiên của hàm.
6. Một hàm có thể có nhiều điểm thoát thông qua lệnh return hoặc khi gặp lệnh cuối cùng của hàm

Một số lưu ý đối với hàm trong C:

1. Không cho phép các hàm lồng nhau có nghĩa là không được định nghĩa một hàm trong một hàm khác
2. Các đối mục thực sự của hàm chỉ có thể chuyển bằng trị có nghĩa là trị của đối mục được sao chép vào một biến tạm được gọi là đối mục hình thức và hàm thao tác trên biến tạm đó

Dạng tổng quát của một hàm như sau:

```
Kiêu      Tên_hàm (các đối mục)
{
          thân hàm
}
```

- ***Kiểu***: Là kiểu của trị mà hàm sẻ trả về thông qua lệnh return. Trị mà hàm trả về phải có cùng kiểu với kiểu đặt trước tên hàm, nếu không xác định trước kiểu trả về thì máy sẻ mặc định là kiểu int.
- ***Tên_hàm***: Do người dùng tự đặt để nhận diện, không được dùng cùng một tên cho 2 hàm khác nhau
- ***Các đối mục***: Là danh sách các biến được phân cách bởi dấu phẩy. Nếu hàm khồng có đối mục thì danh sách đối mục là rổng nhưng vẫn phải có 2 dấu móc đơn. VD hàm main ( )
- ***Thân hàm***: Là phần nằm trong cặp dấu ngoặc nhọn { } gồm một loạt các khai báo biến và các câu lệnh kết thúc bởi dấu chấm phẩy. Phần thân hàm có thể rổng

Ví dụ hàm power ( ) được định nghĩa như sau:

```
Int  power (int x, n)
{
  int  j, p ;
  p = 1 ;
  for (j = 1; j <= n ; j++)
  p = p*x ;
  return (p) ;
}
```

Dòng đầu tiên mô tả tên hàm (power) và danh sách các đối mục (x và n). Các đối mục này gọi là các đối mục hình thức (formal argument hoặc formal parameter). Đây là các biến trung gian tiếp nhận trị của các thông số cung cấp từ chương trình triệu gọi hàm. VD trong main ( )

```
main ( )
{
  power (2, i) ;
}
```

Khi gọi hàm power (2, i) để tính 2^i thì 2 và I gọi là các đối mục thực sự (actual argument). Như vậy khi gọi hàm power ( ) với các thông số 2 và I thì máy tính sẻ biến x

thành 2 và n thành i. Quá trình sao chép các trị từ các đối mục thực sự của chương trình triệu gọi sang các đối mục hình thức của hàm được gọi là truyền đối mục (parameter passing).

Vì hàm có thể trả về một trị cho chương trình triệu gọi như là kết quả thực hiện của hàm. Do đó kiểu của trị trả về phải được chỉ rỏ ở trước tên hàm, nếu không chỉ rỏ thì kiểu trả về mặc nhiên xem như là int.

Các đối mục hình thức phải cùng kiểu với các đối mục thực sự khi gọi hàm. Các đối mục hình thức và các biến khai báo bên trong hàm được xem như là các biến cục bộ nghĩa là chúng chỉ tồn tại và có ý nghĩa bên trong thân hàm.

Câu lệnh return trả về trị do power ( ) tính được cho main ( ). Bất cứ trị nào của một biến, hằng hay một biểu thức đều có thể xuất hiện bên trong dấu ngoặc đơn của return với điều kiện là kiểu của đại lượng đó phải thích hợp với kiểu của trị trả về đã được khai báo trên dòng mô tả tên hàm.

Không nhất thiết hàm nào cũng phải trả về một trị, câu lệnh return không có đối mục trong dấu ngoặc sẻ không trả về trị nào cả mà chỉ đơn thuần là trả quyền điều khiển về cho chương trình triệu gọi để chương trình này tiếp tục thực hiện các công việc khác.

10.2 Chương trình gồm nhiều hàm

Như đã nói ở trên chương trình C gồm một hoặc nhiều hàm, và đương nhiên một trong các hàm đó phải là hàm main ( ). Hàm cung cấp một phương tiện hữu hiệu để đóng gói một công việc cụ thể nào đó. Khi hàm được triệu gọi thì công việc này sẻ được thi hành. Nói chung, có thể xem hàm như một cái hộp đen mang một cái tên và có thể thực hiện một công việc cụ thể nào đó

Trong chương trình C chỉ có một hàm duy nhất tên là main ( ) và không có đối mục. Chương trình được bắt đầu thi hành từ hàm main ( ) trở đi, hàm có thể triệu gọi các hàm khác tạo thành lớp hàm phân thành đẳng cấp hàm nối kết nhau

10.3 Hàm có kiểu void

Trong các phiên bản về sau, C có đưa ra một loại hàm đặc biệt không có trị trả về được gọi là void, void có nghĩa là không có hiệu lực. VD:

```
void  tên_hàm ( )
{
}
```

Trong C thứ tự xuất hiện các hàm là không quan trọng

11. CẤU TRÚC ĐIỀU KHIỂN CHƯƠNG TRÌNH

Thông thường chương trình được thi hành một cách tuần tự có nghĩa là lệnh này được thi hành xong thì đến lệnh kế tiếp. Tuy nhiên, luồng điều khiển thi hành các lệnh trong chương trình có thể bị chuyển hướng tùy theo tình huống, việc chuyển hướng này được thực hiện bởi các cấu trúc điều khiển. Giống các ngôn ngữ khác C cũng gồm có 3 nhóm cấy trúc điều khiển chương trình:

1. Cấu trúc tuần tự
2. Cấu trúc chọn lựa
3. Cấu trúc lặp

Cấu trúc chọn lựa và lặp được thực hiện dựa trên các điều kiện, một điều kiện sẻ xác định một biểu thức là đúng (true) hoặc sai (false)

11.1 Toán tử quan hệ

Các phép tính này sẻ cho kết quả là true hoặc false

<	nhỏ hơn
>	lớn hơn
<=	nhỏ hơn hoặc bằng
>=	lớn hơn hoặc bằng
==	bằng

!= không bằng

Trong C toán tử bằng dùng ký hiệu == để phân biệt với ký hiệu gán =. Ngoài ra, ký hiệu <=, >=, != gồm hai ký hiệu liên tiếp nhau nên không được có khoảng trắng ở giữa.

Các toán tử quan hệ trên là toán tử hai ngôi nghĩa là hoạt động dựa trên hai biểu thức ở hai bên toán tử và dẫn đến kết quả logic là true hoặc false

11.2 Toán tử logic và biểu thức

Trong C gồm có các toán tử logic như sau:

```
&&      AND
||      OR
!       NOT
```

Giả sử muốn chương trình chuyển hướng khi thỏa mãn cùng lúc hai điều kiện thì phải dùng toán tử AND

```
If (phai == 1 && tuoi >= 65)
   Phu_nu_gia++ ;
```

Lệnh if ở trên có 2 điều kiện: Điều kiện giới tính phải là nử (phai == 1), điều kiện thứ hai là tuổi phải lớn hơn hoặc bằng 65 (tuoi >= 65). Hai biểu thức này phải được định trị trước vì toán tử == và >= có mức ưu tiên cao hơn && và sau đó mới phối hợp với toán tử && để cho ra trị cuối cùng nếu kết quả là true thì tăng biến phu_nu_gia lên 1

Toán tử && và || thuộc loại toán tử hai ngôi, nghĩa là hoạt động trên hai biểu thức đưa đến một trị hoặc true hoặc false. Cả hai toán tử này xem các toán hạng như là trị logic.

Toán tử phủ định (!) là toán tử một ngôi. Một số biểu thức dùng toán tử ! như sau:

```
!5      cho ra trị 0 vì 5 xem như là true
!0      cho ra trị 1
!'t'    cho ra trị 0 vì 't' coa mã ASCII là 16 xem như là true
3 + !x  cho ra trị là 3 nếu x khác 0 hoặc là 4 nếu x bằng 0
!!5     cho ra trị 1
!!0     cho ra trị 0
```

11.3 Toán tử điều kiện

Toán tử điều kiện gồm 3 biểu thức, biểu thức thứ nhất và thứ hai hai cách nhau bởi một dấu hỏi và biểu thức thứ hai cách biểu thức thứ ba bởi dấu hai chấm. Cú pháp như sau:

```
Biểu thức 1 ? biểu thức 2 : biểu thức 3
```

Biểu thức 1 được dùng để quyết định xem một trong hai biểu thức nào theo sau sẻ được định trị. Nếu biểu thức 1 cho ra trị true thì biểu thức 2 được định trị còn ngược lại nếu biểu thức 1 cho ra trị false thì biểu thức 3 được định trị. Ví dụ:

```
X = (y < 0)? –y : y ;  ý nghĩa câu lệnh này như sau:

If (y < 0)
  X = -y ;
else
  X = y ;
```

11.4 Cấu trúc lặp

Đa số chương trình máy tính đều liên quan đến việc thi hành lặp đi lặp lại một nhóm lệnh cho đến khi một điều kiện chấm dứt nào đó được thỏa mản. Có hai cách để điều khiển vòng lặp.

1. Bằng cách dùng một cái đếm (counter) gọi là counter controlled repetition nghĩa là vòng lặp được điều khiển bởi một cái đếm.
2. Bằng cách sử dụng một trị canh chừng (sentinel)

Theo cách thứ nhất, số lần thực hiện vòng lặp sẻ được biết trước còn theo cách thứ hai thì số lần lặp sẻ không biết trước và có thể là vô hạn.

Với phương pháp thứ nhất cần phải có một biến điều khiển dùng như một cái đếm để đếm số vòng lặp đã thực hiện, biến này thường được tăng 1 sau mỗi vòng lặp, khi trị của biến đạt trị đặt trước thì vòng lặp ngừng lại, quyền điều khiển sẻ được trao cho lệnh kế tiếp ngay sau vòng lặp.

Phương pháp thứ hai chỉ được sử dụng khi:

- Số lần lặp không biết trước.
- Trong thân vòng lặp có lệnh đọc trị canh chừng mỗi lần vòng lặp được thi hành

11.4.1 Vòng lặp điều khiển bằng biến đếm

Vòng lặp này đòi hỏi:

1. Tên của biến điều khiển.
2. Một trị khởi đầu gán cho biến đếm.
3. Nấc tăng mà biến đếm sẻ bị thay đổi mỗi khi vòng lặp được thi hành. Nấc tăng mặc định là 1.
4. Điều kiện trắc nghiệm trị cuối cùng của biến đếm để quyết định tiếp tục hoặc ngưng vòng lặp.

Ví dụ:

```
(1)     #include  <stdio.h>
(2)     main ( )
        {
(3)       int cai_dem = 1;
(4)       while (cai_dem <= 10) {
(5)         printf (“%d\n”, cai_dem) ;
(6)         cai_dem++ ;
          }
        }
```

Lệnh số 3 khai báo cai_dem là số nguyên với trị ban đầu là 1, điều kiện tiếp tục vòng lặp trong lệnh 4 sẻ trắc nghiệm xem trị của biến đếm <= 10, vòng lặp sẻ chấm dứt khi cai_dem = 11. Chương trình trên có thể viết ngắn gọn hơn

```
while (++cai_dem <= 10)
  Prinf (“%d\n”, cai_dem) ;
```

Cấu trúc for

Cấu trúc for sẻ tự động lo mọi việc điều khiển vòng lặp dựa trên phương pháp biến đếm. Cú pháp như sau:

```
for (biểu thức 1; biểu thức 2 ; biểu thức 3)
   <câu lệnh> ;
```

Trong đó:

for	: Là từ khóa
biểu thức 1	: Là biểu thưc gán trị khởi đầu cho biến đếm.
biểu thức 2	: Biểu thức điều kiện cho ra trị true hoặc false.
Biểu thức 3	: Là nấc tăng của biến điề khiển.
<câu lệnh>	: Thân vòng lặp.

Trong thân vòng lặp có thể chứa một cấu trúc for khác dẫn đến các vòng lặp lồng nhau. Ví dụ:

```
for (k = 1 ; k <= 10 ; k++)
        printf (“binh phuong cua %d la %d\n”, k, k*k) ;
```

Chương trình sẻ in ra bình phương của 10 số nguyên dương đầu tiên

Lưu ý:

- Trị khởi đầu, trị điều kiện và trị tăng nấc có thể là những biểu thức. Giả sử có x = 2 và y = 10 thì câu lệnh sau sẻ tương đương

```
for (j = x ; j <= 4*x*y ; j += y/x)
for (j = 2 ; j <= 80 ; j += 5)
```

- Trị tăng nấc có thể là âm, trong trường hợp này vòng lặp sẻ đếm thụt lùi.
- Nếu bắt đầu biểu thức điều kiện có giá trị là false thì thân vòng lặp sẻ không được thực hiện

Việc sử dụng toán tử dấu phẩy cũng có thể áp dụng cho cả 3 biểu thức và rất thường xuyên được áp dụng cho biểu thức 1 và biểu thức 3, lý do là lập trình viên có thể dùng nhiều lệnh khởi gán cũng như nhiều lệnh tăng nấc cùng lúc. Ví dụ trong một cấu trúc for có thể có hai biến điều khiển cần khởi gán và tăng nấc, các biểu thức khởi gán và tăng nấc được cách nhau bởi dấu phẩy.

Các biểu thức trong cấu trúc for có thể vắng mặt nhưng dấu chấm phẩy phân cách phải có mặt. Ví dụ:

```
t_cong = 0 ;
I = 1;
for (; I <= m ; i++) ;
  t_cong += I ;
```

Nếu cho biểu thức 3 nằm lẫn vào lệnh gán thì cung không cần viết ra biểu thức này

```
t_cong = 0 ;
I = 1;
for (; I <= m ; i++) ;
  t_cong += I++ ;
```

Nếu biểu thức 2 cung biến luôn thì điều kiện bao giờ cung có trị là true và vòng lặp trở thành vô tận

```
for ( ; ; ) {
}
```

11.4.2 Vòng lặp dùng trị canh chừng

Lệnh while là lệnh cơ bản để tạo cấu trúc vòng lặp, cú pháp như sau:

```
while (biểu thức điều kiện)
        <lệnh> ;
```

Lệnh while được thi hành bằng cách trước tiên định trị biểu thức điều kiện nằm trong dấu ngoặc. Nếu kết quả khác 0 nghĩa là điều kiện trở thành true thì <lênhj> được thi hành, tiến trình này lại lặp lại lần nữa bằng cách định trị (biểu thức điều kiện). Vòng lặp tiếp tục cho đến khi (biểu thức điều kiện) có trị là 0

Ví dụ: Đọc vào một loạt số thực dương rồi cộng dồn . Chấm dứt chương trình khi gặp số âm đầu tiên

```
#include  <stdio.h>
#define  zero 0.0
main ( )
{
   float so, t_cong = zero ;
   while (scanf (“%f”, &so), so >= zero)
      t_cong += so ;
  printf (“Tong cong la %f\n”, t_cong) ;
```

11.5 Cấu trúc chọn lựa

Cấu trúc lựa chọn được đại diện bởi lệnh if và switch

11.5.1 Lệnh if

Cú pháp như sau:

```
If (biểu thức điều kiện)
   Lệnh 1 ;
elsse
   Lệnh 2 ;
```

Nếu (biểu thức điều kiện) cho ra một trị khác không (true) thì lệnh 1 sẻ được thực hiện và chương trình sẻ tiếp tục với lệnh theo sau lệnh if, ngược lại nếu (biểu thức điều kiện) cho ra trị là 0 (false) thì lệnh 2 được thực hiện.

Ví dụ: Tráo biến

```
if (a < b)
   printf (“nguyên thủy a nhỏ hơn b\n”) ;
else {
   tam = a ;
   a = b ;
   a = tam ;
   printf (“tráo a và b\n”) ;
```

Cấu trúc lồng nhau của lệnh if. Ví dụ: chương trình phân loại điểm như sau

Điểm	Loại
80-100	A
70-79	B
60-69	C
50-59	D
40-49	E
0-39	F

```
if (điểm >= 80)
   loai = ‘A’ ;
else if (điểm >= 70)
   loai = ‘B’ ;
else if (điểm >= 60)
   loai = ‘C’ ;
else if (điểm >= 50)
   loai = ‘D’ ;
else if ( điểm >= 40)
   loại = ‘E’ ;
else
```

```
    loại = 'F' ;
```

11.5.2 Lệnh switch

Như trong ví dụ trên cho thấy việc só sánh trị của một biến với các trị khác nhau được xảy ra nhiều lần, để thuận tiện C đưa ra một lệnh đặc biệt cho mục đích này đó là lệnh switch, switch có nghĩa là bật chuyển qua lại, switch được hình dung như là một công tắc xoay nhiều vị trí. Cú pháp như sau:

```
switch (biểu thức)
{
  case biểu_thức_hằng_1:
    lệnh 1a ;
    lệnh 1b ;
  case biểu_thức_hằng_2:
    lệnh 2a ;
    lệnh 2b ;
……
……
  case biểu_thức_hằng_n:
    lệnh na ;
    lệnh nb ;
  default:
    lệnh da ;
    lệnh db ;
}
```

Biểu thức điều kiện trong dấu ngoặc sẻ được định trị, kết quả định trị phải là kiểu nguyên kể cả kiểu char. Các biểu thức hằng ở từng trường hợp sau từ khóa case (thường được gọi là case labels) cũng phải là một số nguyên và không thể có hai số nguyên trùng nhau.

Khi thực hiện lệnh switch thì trước tiên biểu thức nằm sau từ switch được định trị. Trị kết quả sau đó được đem so sánh với từng case label, nếu bằng nhau thì quyền điều khiển sẻ được trao cho lệnh đầu tiên trong nhóm case label này và tất cả các lệnh kể từ đây cho đến cuối switch sẻ được thi hành. Ví dụ:

```
n = 2 ;
switch (n)
{
  case 1: printf ("một\n") ;
  case 2: printf ("hai\n") ;
  case 3: printf ("ba\n") ;
  case 4: printf ("bốn\n") ;
  default: printf ("default\n") ;
}
printf ("cuối switch\n") ;
```

Biểu thức điều kiện ở đây là trị của n = 2. Kết quả chương trình như sau:

```
hai
ba
bốn
default
cuối switch
```

Lệnh trong case label có thể là một lệnh rỗng. Nếu không có biểu thức hằng của case label nào bằng với trị của biểu thức điều kiện thì các lệnh của default sẻ được thi hành. Trong ví dụ trên nếu n = 7 thì kết quả sẻ như sau

default
cuối switch

Từ default và các lệnh liên quan là tùy chọn có nghĩa là không có cũng được. Nếu trị của biểu thức điều kiện không bằng với bất cúa biểu thức hằng nào và trong trường hợp này cũng không có default thì các lệnh trong switch sẻ không được thi hành.

11.5.3 Lệnh break

Như đã trình bày ở trên khi quyền điều khiển trao về cho trường hợp case label bằng với trị của biểu thức điều khiển, thì việc thi hành sẻ bắt đầu từ đó cho đến cuối switch. Tuy nhiên, thông thường lệnh switch được sử dụng như là một phương thức lựa chọn nhiều chiều. Một khi quyền điều khiển được trao cho một case label thì chỉ những lệnh thuộc case label này được thi hành rồi nhảy về cuối switch bỏ qua những lệnh của các case label khác. Việc này được thực hiện bằng cách đặt tiếp theo sau lệnh cuối trong case label một lệnh break. Ví dụ:

```
n = 3 ;
switch (n)
{
  case 1: printf (“một\n”) ; break ;
  case 2: ;
  case 3: printf (“hai hoặc ba\n”) ; break ;
  case 4: printf (“bốn\n”) ; break ;
}
  printf (“cuối switch\n”) ;
```

sẻ cho kết quả như sau:

hai hoặc ba
cuối switch

Thực ra trong case 4 không cần lệnh break

12. TRÌNH DỊCH C51

Trình dịch C51 được tối ưu hóa để lập trình cho họ vi điều khiển 8051, chương trình làm việc theo chuẩn ngôn ngữ ANSI-C và được mở rộng để thích hợp với cấu trúc đặc biệt của 8051. Trình dịch C51 tạo ra file đối tượng, file này được xử lý tiếp bởi trình liên kết L51. Đồng thời file liệt kê cũng được tạo ra sau khi biên dịch gồm một bảng liệt kê đươc định dạng kèm với file nguồn là danh sách các ký hiệu, các tham chiếu chéo và các thông báo lỗi, file liệt kê dùng để in và định vị lỗi có trong file nguồn

Có thể gọi C51 theo dòng lệnh sau đây :

C51 [đường dẫn] file_name [ddieeuf khiển biên dịch]

- Đường dẫn : Có thể được dùng khi file nguồn không nằm trong thư mục hiện hành
- File_name : Tên file nguồn cần biên dịch. Phải có phần mở rộng VD : PROG1.C51
- Điều khiển biên dịch : Hướng dẫn hoạt động của trình biên dịch. Các điều khiển biên dịch cũng có thể được đặt trong file nguồn

12.1 Điều khiển biên dịch

Các điều khiển này chỉ ảnh hưởng đến cách làm việc của trình dịch C, chúng có thể được gọi trực tiếp từ dòng lệnh hoặc được đặt trong file nguồn. Các điều khiển này được chia thành 2 nhóm:

- Điều khiển sơ cấp: Được gọi trực tiếp từ dòng lệnh hoặc trong file nguồn nhưng chỉ cho phép đặt tên một lần trong mỗi file nguồn và chúng sẻ ảnh hưởng đến toàn bộ quá trình dịch.
- Điều khiển thứ cấp: Chỉ được đặt trong file nguồn nhưng lại cho phép áp dụng nhiều lần.

Cú pháp các điều khiển đặt trong file nguồn có dạng như sau:

#pragma điều_khiển

Ví dụ:

```
#pragma  large debug code objectextend
#pragma optimize (4, size)
```

12.1.1 Các điều khiển sơ cấp

LISTINCLUDE

Ghi tên các file include vào một danh sách

- Mặc định: NO LISTINCLUDE
- Viết tắt: LC, NOLC

Ví dụ:

C51 VIDU.C51 LC

#pragma LISTINCLUDE

BẢNG 1.1 Các điều khiển biên dịch

Tên	Viết tắt	Mặc định	File nguồn	Dòng lệnh	Loại
[NO]LISTINCLUDE	[NO]LC	NOLC	YES	YES	Sơ cấp
[NO]SYMBOLS	[NO]SB	NOSB	YES	YES	Sơ cấp
[NO]PREPRINT	[NO]PP	NOPP	NO	YES	Sơ cấp
[NO]CODE	[NO]CD	NOCD	YES	YES	Sơ cấp
[NO]PRINT	[NO]PR	PR(file name)LST	YES	YES	Sơ cấp
[NO]COND	[NO]CO	CO	YES	YES	Sơ cấp
PAGELENGHT	PL	PL(69)	YES	YES	Sơ cấp
PAGEWIDTH	PW	PW(80)	YES	YES	Sơ cấp
[NO]DEBUG	[NO]DB	NODB	YES	YES	Sơ cấp
SMALL	SM	SM	YES	YES	Sơ cấp
COMPACT	CP	‘—‘	YES	YES	Sơ cấp
LARGE	LA	‘—‘	YES	YES	Sơ cấp
[NO]INTVECTOR	[NO]IV	IV	YES	YES	Sơ cấp
[NO]OBJECT	[NO]OJ	OJ(file name.OBJ)	YES	YES	Sơ cấp
OBJECTEXTEND	OE	‘—‘	YES	YES	Sơ cấp
ROM	‘—‘	ROM(LARGE)	YES	YES	Sơ cấp
[NO]EXTEND	‘—‘	EXTEND	YES	YES	Sơ cấp
SAVE	‘—‘	‘—‘	YES	NO	Thứ cấp
RESTORE	‘—‘	‘—‘	YES	NO	Thứ cấp
OPTIMIZE	OT	OT(5,speed)	YES	YES	Thứ cấp

EJECT	EJ	'—'	YES	NO	Thứ cấp
[NO]REGPARMS	[NO]RP	RP	YES	YES	Thứ cấp
REGISTERBANK	RB	RB(0)	YES	NO	Thứ cấp
[NO]AREGS	[NO]AR	AR	YES	YES	Thứ cấp
DISABLE	'—'	'—'	YES	NO	Thứ cấp
DEFINE	DF	'—'	YES	YES	Thứ cấp

SYMBOLS

Hướng dẫn này báo cho trình dịch C đưa vào một danh sách tất cả các ký hiệu đã sử dụng.

- Mặc định: NOSYMBOLS
- Viết tắt: SB, NOSB

Ví dụ:

C51 VD.C51 SB

#pragma SYMBOLS

PREPRINT

Ra lệnh cho C tạo một file liệt kê có tên giống file nguồn với phần mở rộng là .LI. Trong đó không có phần ghi chú và các chỉ thị tiền xử lý, có thể thay đổi tên file liệt kê bằng cú pháp PREPRINT (name). Thông số này chỉ được gọi trong dòng lệnh.

- Mặc định: NỎPEPRINT.
- Viết tắt: PP, NOPP.

Ví dụ:

C51.VD.C51 PP (test.LI)

CODE

Dịch ra mã assemler của từng lệnh C tương ứng trong file liệt kê

- Mặc định: NOCODE.
- Viết tắt: CD, NOCD.

Ví dụ:

C51 VD.C51 CD

#pragma CODE

PRINT

Thông thường trình dịch C tạo ra một danh sách và đặt trong file "source.LST" với thông số PRINT (name.ext) có thể tạo danh sách này trong một file bất kỳ.

- Mặc định: PRINT (source.lst)
- Viết tắt: PR, NOPR

Ví dụ:

C51 VD.C51 NOPR

#pragma PRINT (listing.prn)

COND

Thông số này xác định đoạn chương trình đã dịch có được nhập vào file list hay không. Với COND có nghĩa là đoạn chương trình không được ghi vào file list. Nhưng để có được sự trình bày tổng quát tốt hơn thì không cần đánh số dòng và mức độ lồng nhau.

- Mặc định: COND
- Viết tắt: CO, NOCO

Ví dụ:

C51 VD.C51 CO

#pragma NOCOND

PAGELENGHT

Đối số này thiết lập số dòng trong một trang bao gồm cả các dòng trống và dòng tiêu đề trong file list.

- Mặc định: PAGELENGHT (69)
- Viết tắt: PL

Ví dụ:

C51 VD.C51 PL (65)

#pragma PL (70)

PAGEWIDTH

Thông số này xác định số lượng ký tự trên một dòng. Nếu dòng quá dài thì sẻ bị ngắt

- Mặc định: PAGEWIDTH (132)
- Viết tắt: PW

Ví dụ:

C51 VD.C51 PW (80)

#pragma PW (75)

DEBUG

Trình dịch C sẻ thêm các thông tin gở rối vào file đối tượng như: Số thứ tự dòng và các ký hiệu để phục vụ cho quá trình tìm lỗi chương trình về sau

- Mặc định: NODEBUG
- Viết tắt: DB, NODB

Ví dụ:

C51 VD.C51 DB

#pragma DEBUG

SMALL / COMPACT / LARGE

Các hướng dẫn điều khiển này thiết lập kiểu bộ nhớ cho vùng nhớ dữ liệu

- SMALL: Đặt tất cả các biến trong vùng nhớ nội
- COMPACT: Tất cả các biến được đặt trong bộ nhớ ngoài và áp dụng cách định địa chỉ ngắn để truy cập (MOVX @R0 / R1), độ lớn vùng dữ liệu được giới hạn trong phạm vi 256 byte (1 trang)

- LARGE: Toàn bộ các biến được đặt trong bộ nhớ ngoài và áp dụng cách định địa chỉ dài (DPTR), vùng nhớ dữ liệu có độ lớn đến 64 KB.

- Mặc định: SMALL

- Viết tắt: SM, CA, LA

Ví dụ:

C51 VD.C51 LA

#pragma COMPACT

INTVECTOR
Trình dịch C tạo ra vectơ ngắt cho mỗi phục vụ ngắt tại vị trí tương ứng
- Mặc định: INTVECTOR
- Viết tắt: IV, NOIV

Ví dụ:

C51 VD.C51 NOIV

#pragma NOINTVECTOR

OBJECT
Điều khiển OBJECT ("file name") báo cho C tạo ra file đối tượng có tên mong muốn. Giá trị mặc định là tên và đường dẫn của file nguồn với phần mở rộng là .OBJ
- Mặc định: OBJECT (source.obj)
- Viết tắt: OJ, NOOJ

Ví dụ:

C51 VD.C51 OJ (test.obj)

#pragma NOOJ

OBJECTEXTEND
Thêm vào thông tin về kiểu của các ký hiệu trong file đối tượng (định dạng INTEL-OMF), thông tin này giúp cho việc hiển thị các biến được rỏ ràng hơn trong trình gở rối
- Mặc định: Không có thông tin mở rộng
- Viết tắt: OE

Ví dụ:

C51 VD.C51 OE

#pragma OBJECTEXTEND

EXTEND
Cho phép trình dịch C mở rộng các lệnh đặc biệt C51 trong môi trường ANSI – C
- Mặc định: EXTEND
- Viết tắt: Không có.

Ví dụ:

C51 VD.C51 EXTEND

#pragma NOEXTEND

12.1.2 Các điều khiển thứ cấp

SAVE và RESTORE

Hướng dẫn SAVE lưu trữ các cài đặt của các điều khiển OPTIMIZE, AREGS và REGPARMS, hướng dẫn RESTORE phục hồi các cài đặt đã lưu trữ.

Ví dụ: Khi cần thay đổi các cài đặt trong file head

- Mặc định: không có.
- Viết tắt: Không có

Ví dụ:

```
/** file: Wait.h **/
#pragma ot (5, speed)
void wait_5ms (void)
{
  unsigned int counter;
  for (counter = 0; counter < 5000; counter++)
    ; /** lệnh rỗng **/
}
/** kết thúc file wait.h **/
/** VD1.C51 **/

#pragma ot (4, size)
#pragma SAVE
#include <wait.h>
#pragma RESTORE

void main (void)
{
  /** chương trình chính **/
}
```

OPTIMIZE

Thiết lập mức tối ưu hóa, mức cao luôn tối ưu hơn mức thấp

- Mặc định: OPTIMIZE (5, size)
- Viết tắt: OT

Ví dụ:

```
C51 VD.C51 OT (4, speed)

#pragma OT (2, size)
```

EJECT

Tạo một ngắt trang trong file liệt kê, chỉ khai báo trong file nguồn.

- Mặc định: Không có
- Viết tắt: EJ

Ví dụ:

```
Void wait_5ms (void)
{
  unsigned int counter;
```

```
  for (counter = 0; counter < 5000; counter++)
    ;
}

#pragma EJ

void main (void)
{
  /** chương trình chính **/
}
```

REGPARMS

Với hướng dẫn này trình dịch C sẻ chuyển từ các lời gọi hàm đến 3 biến trong các thanh ghi để có thể tiếp tục sử dụng các hàm trước đó. Có thể vô hiệu hóa tùy chọn này bằng cú pháp NOREGPARMS

- Mặc định: REGPARMS
- Viết tắt: RP, NORP

Ví dụ :

```
Void funktion1 (int zahl) ;
Int funktion2 (char z, int a) ;
#pragma NORP
char funktion_alt (char a, char b, char c) ;
#pragma RP
```

REGISTERBANK

Chỉ dẫn này là cơ sở tính toán địa chỉ tuyệt đối của thanh ghi bằng cách định địa chỉ trực tiếp, có thể được dùng nhiều lần trong chương trình nguồn. Tuy nhiên, phải đặt bên ngoài lệnh gọi hàm. Nên nhớ rằng các hàm được gọi tự sử dụng các dãy thanh ghi nếu không giá trị trả về và giá trị chuyển giao sẻ ghi sai dãy thanh ghi. Khác với USING n, điều khiển REGISTERBANK không tự động thay đổi dãy thanh ghi

- Mặc định : REGISTERBANK(0)
- Viết tắt : RB

Ví dụ :

```
#pragma RB(1)
void wait_5mS(void)
{
  unsigned int zaehler ;

  for (zaehler = 0 ; zaehler < 500 ; zaehler++)
  ; /* trống */
}

void main(void)
{
/* chương trình chính */
  rb0 = 1 ; /* chuyển sang dãy thanh ghi 1 */
  rb1 = 0 ;
  wait_5mS( ) ;
/*…..*/
}
```

AREGS

Điều khiển này yêu cầu trình dịch tối ưu hóa mã lệnh tạo ra khi định địa chỉ trực tiếp các thanh ghi. VD : PUSH và POP

- Mặc định : AREGS

- Viết tắt : AR, NOAR

Ví dụ :
```
#pragma AREGS

; Lệnh hợp ngữ
        PUSH  AR6
        PUSH  AR7

#pragma NOAREGS

; Lệnh hợp ngữ
        MOV   A, R6
        PUSH  ACC
        MOV   A, R7
        PUSH  ACC
```

DISABLE

Điều khiển DISABLE phải được đặt trước một hàm để cấm tất cả các ngắt trong thời gian thực hiện hàm. Hàm có thể không trả về giá trị bít do PSW được cất khi gọi hàm

- Mặc định : Không có
- Viết tắt : Không có

Ví dụ :
```
#pragma  DISABLE
/* cấm tất cả các ngắt */
void wait_5mS(void)
{
/* cất PSW */
  unsigned  int  zaehler ;
  for (zaehler = 0 ; zaehler < 5000 ; zaehler++)
  ; /* trống */
/* phục hồi PSW */
}

void  main(void)
{
  /* chương trình chính */
   wait_5mS( ) ;
   /*…..*/
}
```

DEFINE

Quá trình tiền xử lý có thể được điều khiển bởi DEFINE. Chỉ dẫn này được thực hiện trước khi biên dịch. Quá trình tiền xử lý có thể xử lý các lệnh tiếp theo sau

- #define ĐỊNH DANH BIỂU THỨC định nghĩa một tên cho biểu thức sau đó tên này khi được trong file nguồn sẻ được thay bằng biểu thức đã đingh nghĩa
- #undef ĐỊNH DANH xóa định danh
- defined (ĐỊNH DANH) tạo ra nếu #if- hoặc #elif- lệnh ĐÚNG hoặc không tạo ra nếu SAI
- #if BIỂU THỨC. Nếu thỏa mản điều kiện thì dịch đoạn chương trình theo sau
- #elif BIỂU THỨC nếu điều kiện trước #if-lệnh SAI và điều kiện của #elif-lệnh được thỏa thì dịch đoạn chương trình theo sau

- #else nếu điều kiện trước #if-lệnh hoặc #elif-lệnh SAI thì biên dịch đoạn chương trình theo sau
- #ifdef ĐỊNH DANH nếu định danh đã định nghĩa thì dịch đoạn chương trình theo sau
- #ifndef ĐỊNH DANH nếu định danh không được định nghĩa thì dịch đoạn chương trình theo sau
- #endif phải được đặt sau mỗi phát biểu #if. Nên lưu ý đến ký tự hoa hoặc thường khi viết ĐINH DANH và BIỂU THỨC
- Mặc đinh : Không có
- Viết tắt : DF

Ví dụ :

```
#define ENDLOS  while(1) ; /* vòng lặp ENDLOS */

  ENDLOS        /* ứng dụng định danh */

#define DEMO
#ifdef DEMO
        #include <demo.h>
#else
        #include <profi.h>
#endif
```

12.2 Các kiểu dữ liệu đặc biệt trong 8051

Trình dịch C51 nhận biết tất cả các kiểu dữ liệu được xác định thoe chuẩn ANSI-C. Để có thể chuyển các phép tính sang cấu trúc đặc biệt của 8051 cần phải thêm các mở rộng về định nghĩa biến

- Bằng cách nhập kiểu bộ nhớ, mỗi biến có thể được đặt vào các vùng địa chỉ khác nhau của 8051 không phụ thuộc vào cấu hình bộ nhớ đa chọn.
- Kiểu sfr hoặc sfr16 được dùng để mô tả thanh ghi đặc biệt của 8051
- Kiểu sbit được dùng để định nghĩa các bít tuyệt đối trong vùng thanh ghi chức năng đặc biệt.
- Dãy thanh ghi sử dụng được cho biết trước đối với mỗi hàm.

Kiểu bộ nhớ

Nếu không cho biết kiểu bộ nhớ. Các biến tự động được đặt vào vùng địa chỉ cho trước trong cấu hình bộ nhớ. Nhằm mục đích tối ưu hóa chương trình, các biến xác định được đặt trong một không gian địa chỉ xác định. Qua đó có những khả năng như sau :

- Mô tả cấp bộ nhớ **kiểu bộ nhớ**
- Mô tả cấp bộ nhớ
- Mô tả **kiểu bộ nhớ**
- Mô tả

Các cấp bộ nhớ cho phép

- Auto
- Static
- Register
- Extern

Ví dụ :

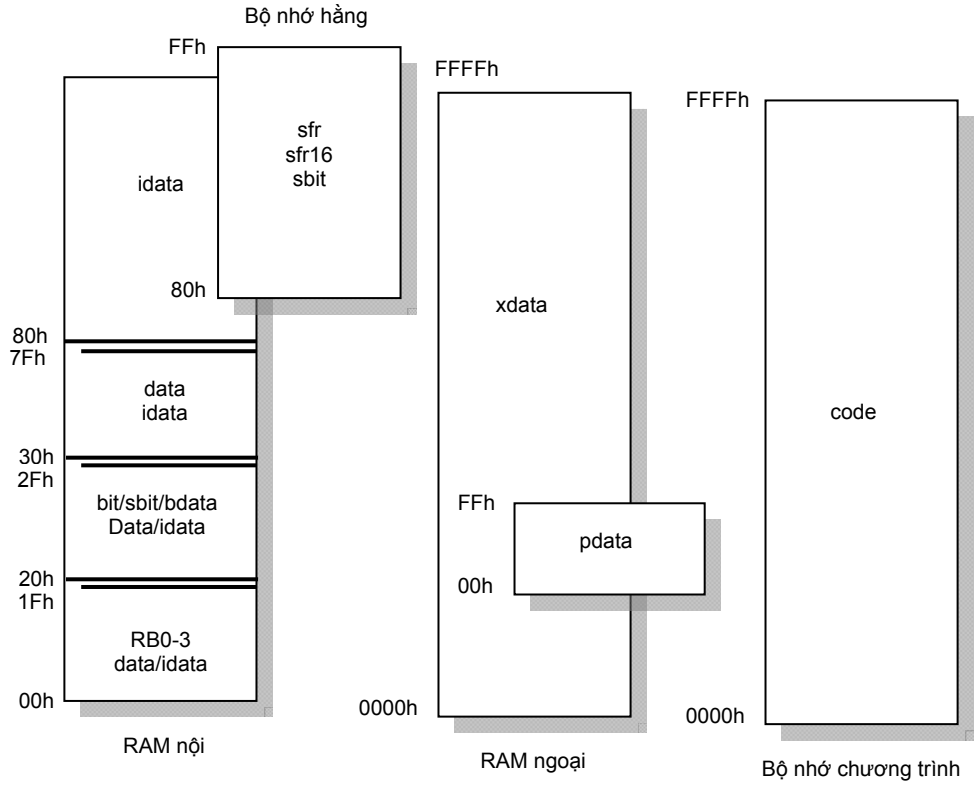
```
Static char counter ;
```

Extern xdata long ket_qua ;
Register alarm_value ;

Các kiểu bộ nhớ cho phép

- Code
- Bit
- Data
- Bdata
- Idata
- Pdata
- Xdata

Data : 00h – 7Fh, bộ nhớ trong địa chỉ trực tiếp
Idata : 00h – 7Fh (FFh), bộ nhớ trong địa chỉ gián tiếp
Bít : 00h – 7Fh, bộ nhớ trong địa chỉ bít
Sbit : 00h – FFh, bộ nhớ trong địa chỉ bít và vùng SFR
Bdata : 20h – 2Fh, địa chỉ byte của bộ nhớ trong địa chỉ bít
Sfr : 80h – FFh, vùng nhớ sfr địa chỉ trực tiếp
Sfr16 : 80h – FFh, 16 thanh ghi sfr địa chỉ trực tiếp
Pdata : 00h – FFh, bộ nhớ ngoài địa chỉ gián tiếp (trang)
Xdata : 0000h – FFFFh, bộ nhớ ngoài địa chỉ gián tiếp
Code : 0000h – FFFFh, bộ nhớ chương trình địa chỉ gián tiếp



Hình 1.1 Kiểu bộ nhớ và địa chỉ

Ví dụ :

```
Static data char counter ;
Extern xdata int loop_count ;
Idata char string1[20] ;
Bit alarm ;
```

CODE

Các biến đã được định nghĩa với kiểu bộ nhớ CODE chỉ cho phép đọc, cung x như các hằng số, chúng được đặt trong bộ nhớ chương trình của vi điều khiển.

Ví dụ :

```
Static code char error[ ] = { “phim khong cho phep” } ;
Code int MAX_COUNT = 10 ;
```

BIT

Các biến kiểu BIT được trình dịch đưa vào bộ nhớ trong từ địa chỉ 20h đến 2Fh, chúng chỉ thích hợp với các kiểu bộ nhớ DATA, BDATA và IDATA, không cho phép định nghĩa biến bít cho con trỏ (bit *bit_ptr ;) cũng không được định nghĩa mảng con trỏ (bit bit_array[4] ;)

Ví dụ :

```
Extern bit Y_N ;
Static data bit back_light ;
```

DATA

Các biến có kiểu bộ nhớ DATA được đặt trong bộ nhớ dữ liệu nội từ địa chỉ 00h đến 7Fh, vì vùng nhớ này rất hẹp và đối với hầu hết các họ vi điều khiển 8051 không tồn tại vùng nhớ bắt đầu từ địa chỉ 80h, nên ngăn xếp cũng được đặt trong vùng nhớ này nên ở đây chỉ thường được dùng cho các biến Ví dụ : Bộ đếm và con trỏ.

Ví dụ :

```
Data char *string_ptr ;
Data char counter ;
```

BDATA

Với các biến được định nghĩa kiểu BDATA, chúng được trình dịch đặt vào bộ nhớ nội từ địa chỉ 20h đến 2Fh, kiểu bộ nhớ này không được dùng trong các cấu trúc.

Ví dụ :

```
Bdata char bitfield ;
Sbit bit0 = bitfield^0 ;
Sbit bit1 = bitfield^1 ;
Sbit bit2 = bitfield^2 ;
Sbit bit3 = bitfield^3 ;
```

IDATA

Các biến kiểu IDATA được đặt ở bộ nhớ dữ liệu trong từ địa chỉ từ 00h đến 7Fh. Với các vi điều khiển có bộ nhớ dữ liệu trong là 256 byte thì vùng địa chỉ từ 80h đến FFh cũng được dùng cho IDATA. Các biến IDATA được truy cập gián tiếp qua thanh ghi R0 và R1

Ví dụ :

```
Idata char array [5] ;
```

PDATA

Vùng nhớ PDATA nằm trong bộ nhớ dữ liệu ngoài và có độ lớn 256 byte. Các biến kiểu này được định địa chỉ thông qua thanh ghi R0 và R1. các trang trong bộ nhớ này được cố định trong file STARTUP.A51. Trang có thể được thay đổi bởi SFR P2. Tuy nhiên, sau đó nó phải được sắp xếp để luôn phù hợp với biến đã đặt.

Ví dụ :

```
Void test (void)
{
  pdata char c ;
  P2 =1 ; /* đặt trang 1*/
  c = c + 2 ;
  P2 = 0 ; /* đặt trang 0 */
}
```

XDATA

Các biến kiểu XDATA được đặt trong bộ nhớ dữ liệu bên ngoài có độ lớn đến 64 Kbyte, các bién này được truy cập thông qua con trỏ dữ liệu DPTR nên thời gian xử lý chậm

Ví dụ :

```
Xdata char buffer [300] ;
Struct TIME
{
        unsigned char sec ;
        unsigned chả min ;
        unsigned char hour ;
        unsigned char day ;
}

xdata struct TIME times [100] ;
```

Xử lý thanh ghi chức năng đặc biệt (SFR)

Các thanh ghi điều khiển khối ngoại vi tích hợp của 8051 nằm trong vùng thanh ghi chức năng đặc biệt, để truy cập trực tiếp C51 không dùng các định nfghĩa của ANSI-C mà thay vào đó là các từ định sẵn **sfr** và **sfr16**. Phương pháp như sau :

- Sfr name = hằng số ;
- Sfr16 name = hằng số ;

Name là tên tùy chọn. Trong khi hằng số là giá trị cho phép từ 80h đến FFh. Đối với các vi điều khiển thông dụng nhất, các mô tả sfr được chứa trong các file h trong môi trường phát triển của chương trình dịch.

Đặc biệt đối với các họ vi điều khiển 8051 mới hơn các thanh ghi chức năng đặc biệt thường có độ dài 16 bít nên để tăng hiệu quả truy cập thường phải dùng các định nghĩa sfr16. Việc áp dụng sfr16 cũng cho phép khi phần cao của sfr theo sau ngay phần thấp. sau đó, để xác định phải cho biết địa chỉ phần thấp của biến

Ví dụ :

```
Sfr P0 = 0x80 ; /* định nghĩa port 0 */
Sfr16 T2 = 0xCC ; /* thanh ghi Timer2: Phần thấp CCh, phần cao CDh */
```

Mục nhập sau dấu “=” không phải là một chỉ dẫn mà là một mô tả cho biết địa chỉ của các thanh ghi chức năng đặc biệt.

Xử lý các sfr địa chỉ bít

Một số thanh ghi chức năng đặc biệt cho phép truy cập địa chỉ bít nhằm mục đích đơn giản việc truy cập này trình dịch C51 cung cấp một khả năng mô tả bít đặc biệt.

- Sbit sbit_name = sbit_adr ;

Trong đó, sbit_name là bất kỳ, tên biến tùy chọn. sbit_adr được định nghĩa như sau :

- Sfr_name^const : Một biến định nghĩa với sfr.
- Const1^const2 : Một địa chỉ byte.
- Const : Một địa chỉ bít

Ví dụ ứng dụng cách 1 :

```
Sfr PSW = 0xD0 ;
Sbit OV = PSW^2 ;
Sbit CY = PSW^7 ;
```

Trong đó const nhận giá trị từ 0 đến 7

Ví dụ ứng dụng cách 2 :

```
Sbit OV = 0xD0^2 ;
Sbit CY = 0xD0^7 ;
```

Trong đó const1 là địa chỉ trong vùng từ 80h đến FFh và const2 là từ 0 đến 7

Ví dụ ứng dụng cách 3 :

```
Sbit OV = 0xD2 ;
Sbit CY = 0xD7 ;
```

Trong đó const là địa chỉ bít của các bít được địa chỉ hóa. Các bít này phải ở trong vùng từ 80h đến FFh.

Sử dụng con trỏ

Giống như ANSI-C. Trình dịch C51 cũng có thể dùng con trỏ. Để tính toán được cấu trúc đặc biệt của 8051, có hai kiểu con trỏ được cung cấp :

- Con trỏ tổng quát
- Con trỏ xác định bộ nhớ

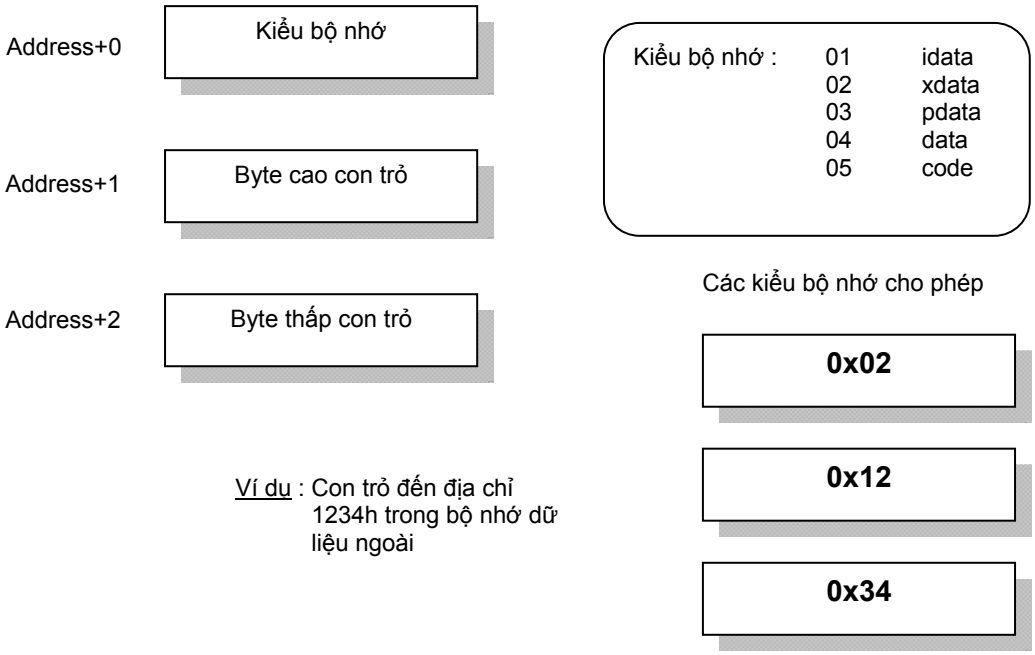
Con trỏ tổng quát

Con trỏ tổng quát cho phép tất cả các vùng bộ nhớ ngoại trù BDATA. Tại vì kiểu bộ nhớ phải được đặt trong con trỏ. Một con trỏ tổng quát được định nghĩa như sau :

- SP_TYP_PTR DATYP *VAR_NAME ;

Trong đó,

- SP_TYP_PTR : Vùng bộ nhớ nơi đặt biến con trỏ. Có thể là các kiểu data, idata, pdata, xdata.
- DATATYP : Kiểu dữ liệu của biến mà con trỏ chỉ đến. Tất cả các kiểu được định nghĩa trong ANSI-C
- VAR_NAME : Tên bất kỳ theo chuẩn ANSI-C



Hình 1.2 Cấu tạo con trỏ tổng quát

Con trỏ tổng quát luôn có độ dài là 3 byte. Các kiểu bộ nhớ chỉ cho phép từ giá trị 1 đến 5, Các giá trị khác sẻ dẫn đến một đặc tính của chương trình chưa định nghĩa.

Con trỏ hằng

Để có thể truy cập một cách đơn giản đến các địa chỉ ngoại vi, con trỏ có thể được định nghĩa bằng địa chỉ tuyệt đối

- #define VAR_NAME ((DATATYP*)0xABBBBL)

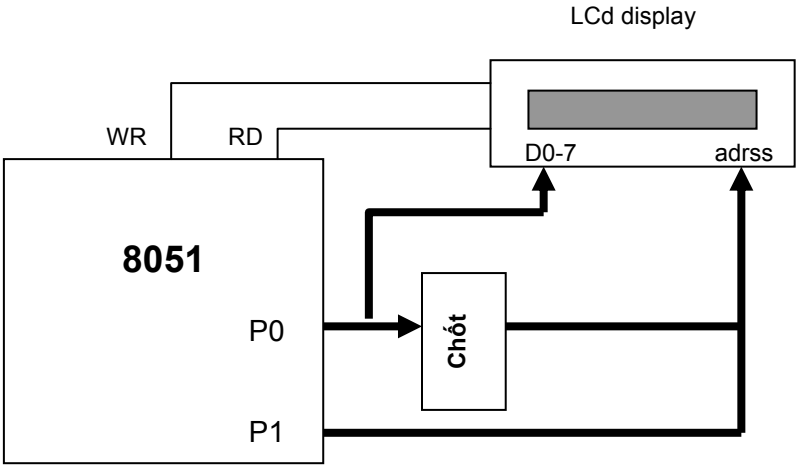
Trong đó :

- DATATYP : Kiểu dữ liệu của biến mà con trỏ chỉ đến. Là tất cả các kiểu được định nghĩa trong ANSI-C
- VAR_NAME : Tên được chọn bất kỳ theo chuẩn ANSI –C
- A : Kiểu bộ nhớ giống như với con trỏ tổng quát (1 = idata, 2 = xdata, 3 = pdata, 4 = data, 5 = code).
- BBBB : Địa chỉ tuyệt đối của khối ngoại vi.
- L : Ký hiệu cho biết địa chỉ được cho là loại **long**

Ví dụ :

```
#include <absacc.h>
#define LCD_DATA_WR ((unsigned char*)0x28001L /* ghi dữ liệu hiển thị LCD */
#define LCD_CONT_RD  XBYTE [0x8000] /* đọc điều khiển hiển thị LCD */
#define busy 0x01 /* trạng thái của bộ hiển thị LCD */
code unsigned char string [ ] = {“TEST TEXT\0”} ;
void main (void)
{
        unsigned char  I ;
        for (I = 0 ; string [i] ! = 0 ; i++)
        {
                LCD_DATA_WR [0] = string [i] ; /* ghi chuỗi ký hiệu vào LCD */
                Do {
```

```
                } uhile (LCD_CONT_LCD == busy) ; /* chờ đến ký hiệu kế tiếp */
            }
}
```



Hình 1.3 Sơ đồ khối bộ điều khiển hiển thị LCD

Một vài macro được định nghĩa trong file h ABSACC.H nằm trong môi trường phát triển của chương trình dịch giúp đơn giản quá trình truy cập đến các vùng nhớ khác nhau dưới dạng byte hoặc word một cách đơn giản, trong ví dụ trên có dùng macro XBYTE được định nghĩa như sau :

```
#define XBYTE ((unsigned char *)0x20000L)
```

Sau đó con trỏ có thể được định nghĩa với một điều khiển biên dịch tiếp theo, (LCD_CONT_RD) có thể được làm việc trực tiếp.

```
#define CONST_PTR   PTR_MACRO [OFFFSET]
```

Trong đó :

- CONST_PTR : Tên bất kỳ theo chuẩn ANSI-C
- PTR_MACRO : Một macro con trỏ được định nghĩa trong ABSACC.h (VD : XBYTE)
- OFFSET : Địa chỉ con trỏ chỉ đến trong vùng nhớ tương ứng.

Ví dụ :

```
#define LCD_CONT_RD XBYTE [0x8000] /* đọc dữ liệu hiển thị LCD tại địa chỉ 8000h */
do {
    } while (LCD_CONT_RD == busy) ; /* chờ đến ký hiệu kế tiếp */
```

Ngược lại, nếu sử dụng một con trỏ tổng quát thì chỉ có thể nhập vào địa chỉ offset

- GEN_PTR [OFFSET] = VALUE

Ví dụ :

```
#define LCD_DATA_WR ((unsigned char *)0x28001L) /* ghi dữ liệu hiển thị LCD vào địa chỉ 8001h */
```

LCD_DATA_WR [0] = string [i] ; /* ghi một ký hiệu vào LCD */

Sử dụng con trỏ xác định bộ nhớ

Với con trỏ xác định bộ nhớ, vùng nhớ của biến nơi mà con trỏ chỉ đến phải được định nghĩa trước trong file nguồn. Qua đó mã lệnh hiệu dụng có thể được tạo ra bởi trình dịch mà không cần gọi thư viện vì mã lệnh trực tiếp cho con trỏ được phát sinh trong vùng nhớ đã định nghĩa.

Con trỏ xác định bộ nhó được định nghĩa theo các cách sau :

- SP_TYP_PTR DATATYP SP_TYP *VAR_NAME ;

Trong đó :

- SP_TYP_PTR : Vùng nhớ trong đó đặt biến con trỏ. Đó là data, idata, pdata, xdata
- DATATYP : Kiểu dữ liệu của biến mà con trỏ chỉ đến là tất cả các kiểu được định nghĩa trong ANSI-C
- SP_TYP : Vùng nhớ con trỏ định địa chỉ. Các kiểu được sử dụng là : data, idata, pdata, xdata, code.
- VAR_NAME : Tên bất kỳ theo chuẩn ANSI-C

Ví dụ :

```
Gen_ptr_test ( )
{
        data char data *ms_ptr ;
        data char var1, var2 ;
        ms_ptr = &var1 ; /* dòng 1 */
        var2 = *ms_ptr ; /* dòng 2 */
}
; mã hợp ngữ của đoạn chương trình trên
; dòng 1
        mov    R7, #LOW var1
        mov    ptr, R7
; dòng 2
        mov    R0, ptr
        mov    var2, @R0
```

Nếu nhập SP_TYP là data, idata, pdata thì con trỏ chỉ được lưu vào một byte trong vùng nhớ định nghĩa bởi SP_TYP_PTRNếu SP_TYP là xdata hoặc code thì biến con trỏ được dùng 2 byte.

Trong ví dụ trên cho thấy con trỏ xác định bộ nhớ *ms_ptr trỏ đến các biến trong vùng nhớ data. Chính con trỏ cũng nằm trong vùng nhớ data. Từ đoạn mã hợp ngữ được tạo ra cho thấy con trỏ chỉ được dùng cho các biến trong vùng nhớ data. Nếu thử dùng con trỏ đó chỉ đến một biến trong vùng nhớ khác thì trình dịch sẻ tạo ra cảnh báo.

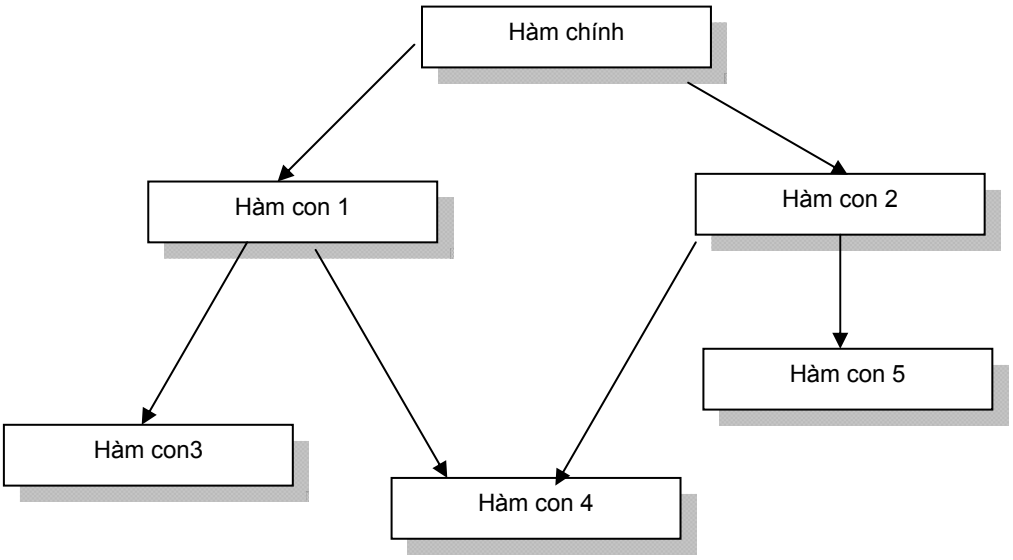
Các biến chồng nhau

Để sử dụng tốt hơn bộ nhớ dữ liệu hạn chế của 8051. Trình liên kết sẻ tạo một sự chồng lên nhau về không gian nhớ đối với các biến byte. Các biến này có thời điểm sử dụng khác nhau. Trình dịch C sẻ đánh dấu vùng nhớ xác định là “có thẻ chòng” tùy thuộc vào cấu hình bộ nhớ đã chọn. Các ký hiệu nhận dạng như sau :

- Cấu hình bộ nhớ SMALL – vùng nhớ data được phép chồng
- Cấu hình bộ nhớ COMPACT – vùng nhớ pdata được phép chồng
- Cấu hình bộ nhớ LARGE – vùng nhớ xdata được phép chồng.

Mỗi cấu hình bộ nhớ cũng chỉ có một vùng nhớ cho các biến byte được chồng nhau. Vùng nhớ bít trong mỗi cấu hình bộ nhớ được chồng nhau

Nếu một hàm được chỉ định trong một cấu hình bộ nhớ khác, khi được định nghĩa cho đơn thể chương trình thì các biến trong vùng nhớ đã được chọn cho hàm cũng sẻ được dánh dấu cho phép chồng. Từ đó dẫn đến khả năng có nhiều vùng nhớ trong một modul chồng nhau hơn. Sau đó trình liên kết sẻ chồng các vùng nhớ của các biến và thông số của các hàm không tự gọi lẫn nhau.



Vùng dữ liệu của hàm con 1 và 3 có thể được chồng khi gọi hàm con 2, vùng dữ liệu của hàm con 4 và 5 không được phép chồng vì chúng được gọi bởi hàm con 2. Nếu hàm con 4 được gọi thì vùng dữ liệu của hàm con 3 và 5 có thể được chồng

Hình 1.4 Sự chồng nhau giữa các biến

Nếu các hàm được gọi gián tiếp thông qua con trỏ thì trình liên kết có thể không nhận ra và dẫn đến tình huống chồng nhau không mong muốn. Trong trường hợp này trình liên kết phải được cung cấp thêm các tham khảo thông qua thông số điều khiển biên dịch OVERLAY

12.3 Gọi hàm trong 8051

Việc gọi hàm mở rộng một vài định nghĩa trong trình dịch C51. Chúng mang cấu trúc đặc biệt cho việc tính toán 8051 và tạo ra một mã lệnh hiệu quả

```
[static] [erg_typ] ten_ham ([datatyp] [var_name] [,…])
[model] [reentrant] [interrupt n] [using n]
{ }
```

Các định nghĩa in nghiêng là các mở rộng đặc biệt cho 8051

- *Model* : Có thể được nhập là small, compact và large nhằm giúp biên dịch các hàm có cấu hình bộ nhớ định sẳn không đúng.
- *Reentrant* : Một hàm được đánh dấu như thế có thể được gọi
- *Interrupt n* : hàm ngắt, n là số hiệu ngắt
- *Using n* : Chuyển sang dãy thanh ghi n khi gọi hàm, trở về dãy thanh khi cũ khi thoát ra khỏi hàm

Nếu áp dụng các định nghĩa model và reentrant chúng phải được khai báo không những khi định nghĩa mà còn trong lúc mô tả hàm

Khi khai báo model lúc định nghĩa hàm, các biến cà các khối thông số được đánh dấu là OVERLAYABLE nhờ đó chúng có thể được chồng lên các vùng nhớ khác nhờ trình liên kết.

Bàng cách sử dụng **reentrant** một ngăn xếp đặc biệt sẻ được đặt trong hàm, tùy theo cấu hình bộ nhớ là small, compact hoặc large, ngăn xếp này sẻ nằm trong IDATA, PDATA hoặc XDATA, các thông số được đưa vào đây mỗi khi gọi hàm

Ví dụ :

```
#pragma LARGE /* đủ chổ ngăn xếp trong XDATA */
long fakultaet (int anzahl) reentrant
{
        if (anz == 0)
                return (1) ;
        else
                return (anz * fakultaet (anz – 1)) ;
}
```

8051 hoạt động với 4 dãy thanh ghi với mỗi dảy gồm có 8 thanh ghi, dãy thanh ghi được chọn bằng định nghía using n. Định nghĩa này chỉ được khai báo khi định nghĩa hàm. Khi hàm trả về một kết quả cần phải lưu ý nếu hàm được gọi dùng chung dảy thanh ghi nếu không thì giá trị trả về sẻ ghi vào dảy thanh ghi khong đúng.

Ví dụ :

```
Extern void funtion2 (void) ;
Void funtion1 (void) using 3
{
        function2 ( ) ;
        /*….*/
}
```

Tất cả các hàm trong ví dụ được gọi bởi funtion1 làm việc với dảy thanh ghi 3. Ngoài ra chúng còn chứa một định nghĩa using riêng

Chuyển tham số cho hàm

Trong C có hai cách chuyển giao tham số cho hàm như sau :

- Chuyển thông qua khối tham số
- Chuyển thông qua thanh ghi

Trình dịch sẻ tối ưu hóa quá trình chuyển tham số. Trước tiên dùng các thanh ghi và ngay sau đó là các khối tham số, nếu không có tham số chuyển giao thì được khai báo là void. Nếu hàm không trả về một giá trị thì trong phần kiểu dữ liệu của hàm cũng được khai báo là void, nếu không trình dịch sẻ trả về trị mặc định là int.

Ví dụ :

```
Funct1 (int var1) ; /* kiểu trả về là int, kiểu tham số là int */
Long funct2 (int var1) ; /* kiểu trả về long, kiểu tham số int */
Void funct3 (char var2) ; /* không có trị trả về, kiểu tham số là char */
Void funct4 (void) ; /* không có trị trả về, không có tham số */
```

Chuyển tham số trong khối tham số

Với điều khiển biên dịch #pragma NOREGPARMS sẻ ngăn chế độ chuyển bằng thanh ghi. Điều khiển này không những được khai báo khi định nghĩa mà còn được khai

báo khi mô tả hàm. Điều khiển biên dịch này có lợi khi dùng các thư viện củ trước đây không có khả năng chuyển bằng thanh ghi hoặc để giao tiếp với các hàm của hợp ngữ.

Cấu hình bộ nhớ	Vùng nhớ	Vùng địa chỉ	Tên ký hiệu
Small	Data bít	00h – FFh 00h – 7Fh	?DT?FKT_NAME?MOD_NAME ?BI?FKT_NAME?MOD_NAME
Compact	Pdata bít	00h – FFh 00h – 7Fh	?PD?FKT_NAME?MOD_NAME ?BI?FKT_NAME?MOD_NAME
Large	Xdata Bít	0000h – FFFFh 00h – 7Fh	?XD?FKT_NAME?MOD_NAME ?BI?FKT_NAME?MOD_NAME

Hình 1.5 Khối tham số và cấu hình bộ nhớ

Khối tham số được đặt vào vùng nhớ thích hợp tùy thuộc vào cấu hình model khi định nghĩa hàm cũng như điều khiển biên dịch #pragma MODEL (small, compact, large). Những vùng nhớ này sẻ được trình biên dịch đánh dấu là cho phép chồng.

Chuyển tham số trong thanh ghi

Để chuyển tham số cho hàm dưới dạng mã và khả năng tổ chức không gian bộ nhớ dữ liệu được hiệu quả nhất. Trước tiên, trình biên dịch sẻ thử chuyển 3 tham số vào các thanh ghi R2-R7, nếu các thanh ghi này sẳn sàng sử dụng thì các tham số tương ứng sẻ được chuyển vào khối tham số.

Tham số	Kiểu biến				Con trỏ		
	Char	Int	Long	Float/double	8 bít	16 bít	Tổng quát
Tham số 1	R7	R6+R7	R4+R5+R6+R7	R4+R5+R6+R7	R7	R6+R7	R1+R2+R3
Tham số 2	R5	R4+R5	R4+R5+R6+R7	R4+R5+R6+R7	R5	R4+R5	R1+R2+R3
Tham số 3	R3	R2+R3	****	****	R3	R2+R3	R1+R2+R3

Hình 1.6 Chuyển tham số trong thanh ghi

Để việc chuyển giao thanh ghi được tối ưu, trước tiên nên chuyển các biến nhỏ (char hoặc int) cũng như con trỏ xác định bộ nhớ.

Các hàm ngắt

Nhằm khai thác tối đa tính năng của 8051. Trình dịch C51 còn hổ trợ việc xử lý các ngắt. Các hàm ngắt khác với các hàm thông thường ở những điểm sau đây :

- Không có tham số chuyển giao
- Không có kết quả trả về. Các hàm ngắt luôn được định nghĩa là void. Chương trình dịch cũng chấp nhận kiểu int vì nó là tối ưu.
- Không cho phép gọi trực tiếp hàm ngắt

Cách thức định nghĩa hàm ngắt như sau :

```
[void] int_name (void) interrupt  n [using  m]
{ }
```

Định nghĩa **interupt n** chỉ được phép khai báo chung với định nghĩa hàm

Trình dịch xử lý một hàm ngắt như sau :

- Cất các thanh ghi chức năng đặc biệt ACC, B, DPL, DPH và PSW trước khi thực hiện hàm
- Không dùng thuộc tính using vì tất cả các thanh ghi dùng trong hàm (R0 – R7) đều được cất
- Trước khi thoát khỏi hàm tất cả các thanh ghi đã cất trong ngăn xếp được phục hồi trở lại.

- Thoát khỏi hàm bằng lệnh RETI để bảo đãm hoạt động ổn định của hệ thống ngắt trong 8051

Trình dịch C51 có thể tạo ra chính xác các véc tơ ngắt nếu cho biết số hiệu ngắt **n**. Tiếp theo địa chỉ của véc tơ ngắt được tính theo công thức sau :

8 * n + 3

Một lệnh LJMP đến chương trình phục vụ ngắt được ghi vào địa chỉ tính toán

Số hiệu ngắt	Nguồn ngắt	Địa chỉ ngắt	Cờ ngắt	Reset
0	Ngắt 0 ngoài	0003h	IE0	Phần cứng
1	Ngắt Timer0	000Bh	TF0	Phần cứng
2	Ngắt 1 ngoài	0013h	IE1	Phần cứng
3	Ngắt Timer1	001Bh	TF1	Phần cứng
4	Nối tiếp Port 0	0023h	RI/TI(RI0/TI0)	Phần mềm
5	Ngắt Timer2	002Bh	TF2/EXF2	Phần mềm
8	Chuyển A/D	0043h	IADC	Phần mềm
9	Ngắt 2 ngoài	004Bh	IEX2	Phần cứng
10	Ngắt 3 ngoài	0053h	IEX3	Phần cứng
11	Ngắt 4 ngoài	005Bh	IEX4	Phần cứng
12	Ngắt 5 ngoài	0063h	IEX5	Phần cứng
13	Ngắt 6 ngoài	006Bh	IEX6	Phần cứng
16	*Nối tiếp Port 1*	*0083h*	*RI1/TI1*	*Phần mềm*
18	*Compare Match 0-7*	*0093h*	*ICMP0-7*	*Phần mềm*
19	*So sánh Timer tràn*	*009Bh*	*CTF*	*Phần cứng*
20	*Compare Match COMSET*	*00A3h*	*ICS*	*Phần cứng*
21	*Compare Match COMCLR*	*00Abh*	*ICR*	*Phần cứng*

In nghiêng : Chỉ có trong 80517

Hình 1.7 Địa chỉ ngắt của 80515/517

Ví dụ :

```
Extern  int  count ;
Void timer1 (void) interrupt 3
{
        TR1 = 0 ; /* ngừng Timer1 */
        count  = TL1 + 256 * TH1 ; /* đọc bộ định thời Timer1 */
        TR1 = 1 ; /* khởi động Timer1 */
}
```

Khi các phép tính float được thực hiện trong một chương trình phục vụ ngắt thì trạng thái các trình phục vụ dấu chấm di động phải được lưu trữ, chúng có thể mất khi các trình dấu chấm di động không sử dụng các hàm khác. Với các hàm fpsave ( ) và fprestore ( ) được mô tả trong file math.h có thể bảo đãm yêu cầu này.

Ví dụ :

```
#include<math.h>
```

```
struct FPBUF save_int ;

void isr_t0 (void) interrupt 1  using 1
{
        float  a, b ;
        fpsave (&save_int) ;/* lưu trạng thái của trình FP */
        a = b * b ;
        fprestore (&save_int) ; /* phục hồi trở lại */
}
```

Giao tiếp giữa các hàm hợp ngữ với các hàm C

Các hàm hợp ngữ được gọi theo quy ước giống như các hàm C, quá trình chuyển tham số đơn giản nhất qua các khối tham số. Tùy theo cấu hình bộ nhớ các tham số này được đặt vào một vùng nhớ xác định. Để trình liên kết có thể tạo nên một tham chiếu chính xác các hàm hợp ngữ phải được định nghĩa giống như các hàm C.

Trong một modul hợp ngữ, các biến toàn cục tùy theo kiểu dữ liệu sẻ được đặt trong một segment riêng. Cách định nghĩa như sau :

- Hằng số : ?CO?MODULNAME
- Biến XDATA : ?XD?MODULNAME
- Biến PDATA : ?PD?MODULNAME
- Biến IDATA : ?ID?MODULNAME
- Biến BDATA : ?BA?MODULNAME
- Biến DATA ?DT?MODULNAME
- Biến BIT : ?BI?MODULNAME

Ví dụ :

```
NAME  TEST1 ; tên modul là TEST1
?XD?TEST1 segment XDATA ; segment cho các biến toàn cục trong vùng XDATA
;….

Rseg?XD?TEST1
Var1 :    ds       2 ; int  var1
;…..
```

Đối với mỗi hàm có một segment CODE riêng được định nghĩa trong đó gồm có tên hàm và tên modul

- ?PR?FUNCTIONS_NAME?MODULNAME

Ví dụ :

```
NAME    TEST1 ; tên modul là TEST1
;*********************************
;**function : voi INIT (void) **
;*********************************

?PR?INIT?TEST1 segment CODE
rseg?PR?INIT?TEST1
INIT :
;……
  ret
```

Các biến cục bộ của hàm tùy theo cấu hình bộ nhớ cũng được chỉ định trong một segment riêng, mỗi một segment được định nghĩa cho các biến byte hoặc các biến bít

Các định nghĩa sau hợp lệ với bộ nhớ SMALL

- Biến byte : segmenttyp DATA ?DT?FUNCTIONS_NAME?MODUL_NAME
- Biến BIT : segmenttyp BIT ?BI?FUNCTIONS_NAME?MODUL_NAME

Các định nghĩa hợp lệ với bộ nhớ COMPACT

- Biến byte : Segmenttyp PDATA ?PD?FUNCTIONS_NAME?MODUL_NAME
- Biến bít : Segmenttyp BIT ?BI?FUNCTIONS_NAME?MODUL_NAME

Các định nghĩa hợp lệ với bộ nhớ LARGE

- Biến byte : Segmenttyp XDATA ?XD?FUNCTIONS_NAME?MODUL_NAME
- Biến bít : Segmenttyp BIT ?BI?FUNCTIONS_NAME?MODUL_NAME

Khi một hàm được gọi với các tham số thì phần bắt đầu của khối tham số được đánh dấu bởi tên ALIAS. Định nghĩa sau đây được dùng cho tham số byte :

- ?FUNCTIONS_NAME?BYTE :

Định nghĩa sau đây được dùng cho tham số bít :

- ?FUNCTIONS_NAME?BIT :

Những định nghĩa này được gán "PUBLIC". Các tham số đươc lưu trữ theo thứ tự nối tiếp định nghĩa của chúng trong lời gọi hàm bắt đầu từ địa chỉ này.

Nếu hàm có giá trị trả về thì chúng phải đặt trong các thanh ghi cố định.

Kiểu dữ liệu	**Thanh ghi**	**Ghi chú**
Bit	Cờ carry	Khong có #pragma DISABLE
Char	R7	
Int	R6+R7	MSB trong R6
Long	R4+R5+R6+R7	MSB trong R4
Float/double	R4+R5+R6+R7	MSB trong R4, dấu và số mũ trong R7
Con trỏ tổng quát	R1+R2+R3	MSB trong R2, kiểu bộ nhớ trong R3
Con trỏ 8 bít	R7	
Con trỏ 16 bít	R6+R7	MSB trong R6

Hình 1.8 Sắp xếp thanh ghi cho trị trả về của hàm

Nếu chú ý đến các quy ước này thì việc kết nối hàm hợp ngữ trong chương trình C được thực hiện một cách dễ dàng và cả trình liên kết cũng có thể phân tích việc chồng nhau cho các hàm này

Gọi hàm hợp ngữ và C trong cấu hình SMALL

Trong cấu hình SMALL, các thông số chuyển giao và các biến cục bộ được đặt trong vùng nhớ data. Hàm hợp ngữ được gọi trong file nguồn C như sau :

```
#pragma SMALL
#pragma NOREGPARMS /* không chuyển tham số trong thanh ghi */
int asm_funkt(int var1, char var2, bit bvar1) ;
void main (void)
{
        int var1, ret_val ;
        char var2 ;
```

```
        bit bvar1 ;
        /*….*/

        ret_val = asm_funkt (var1, var2, bvar1) ;
}
```

Cách định nghĩa hàm này trong file nguồn hợp ngữ

```
NAME    ASM_MODUL1          ; định nghĩa tên modul

?PR?asm_funkt?ASM_MODUL1 segment  CODE     ; Định nghĩa segment cho mã chương trình

?DT?asm_funkt?ASM_MODUL1 segment DATA OVERLAYABLE ;  Định  nghĩa segment cho tham số byte và các biến byte cục bộ

?BI?asm_funkt?ASM_MODUL1 segment BIT OVERLAYABLE      ;  Định  nghĩa segment cho tham số bít và các biến bít cục bộ

Public asm_funkt,  ?asm_funkt?BYTE,  ?asm_funkt?BIT      ; Khai báo các ký hiệu công cộng cho chương trình gọi C

;************************************************************************
;** FUNKTION : int asm_funkt (int var1, char var2, bit bvar1)  **
;************************************************************************

rseg      ?DT?asm_funkt?ASM_MODUL1      ; Segment cho biến byte cục bộ
?asm_funkt?BYTE :
var1 :    ds      2         ; Tham số int var1
var2 :    ds      1         ; Tham số char var2

lok_var : ds      2         ; Biến cục bộ int lok_var
rseg      ?BI?asm_funkt?ASM_MODUL1      ; Segment cho các biến bít
?asm_funkt?BIT :
bvar1 :   dbit1             ; Tham số bít bvar1
lok_bvar :        dbit1     ; Biên bít cục bộ bit lok_bvar

rseg      ?PR?asm_funkt?ASM_MODUL1      ; Bắt đầu phần chương trình của asm_funkt
asm_funkt :
; Có thể truy cập các biến và tham số dựa theo các tên “var1, var2 và lok_var”

mov       a, var2
add       a, var1 + 1       ; lok_var = var2 + var1
mov       lok_var + 1, a
clr       a
addc      a, var1
mov       lok_var, a

; Đối với các biến và tham số bít có thể truy cập theo các tên “bvar1 và lok_bvar”
mov       c, bvar1
mov       P1.1, c           ; P1.1 = bvar1

; Chuyển giá trị trả về, nội dung của biến lok_var
mov       R6, lok_var + 1
```

```
mov     R7, lok_var
ret                     ; Thoát khỏi hàm hợp ngữ
```

Một hàm C cũng có thể được gọi từ một chương trình hợp ngữ. Việc gọi hàm C phải được định nghĩa trong file hợp ngữ

```
NAME    ASM_MODUL1          ; Định nghĩa tên modul

extrn CODE (c_funkt)        ; Mô tả hàm C bên ngoài
extrn DATA (?c_funkt?BYTE)  ; Mô tả tham số byte bên ngoài
extrn BIT (?c_funkt?BIT)    ; Mô tả tham số bít bên ngoài

?PR?asm_funkt?ASM_MODUL1 segment CODE      ; Định nghĩa segment cho mã chương trình
?DT?asm_funkt?ASM_MODUL1 segment DATA OVERLAYABLE   ; Định nghĩa segment cho tham số byte và biến cục bộ byte
?BI?asm_funkt?ASM_MODUL1 segment BIT OVERLAYABLE    ; Định nghĩa segment cho tham số bit và biến cục bộ bit

*********************************************
** FUNKTION : void asm_funkt (void) **
*********************************************
rseg    ?DT?asm_funkt?ASM_MODUL1    ; Đoạn các biến byte cục bộ

lok_var1 :      ds      2       ; Biến cục bộ int lok_var1
lok_var2 :      ds      1       ; Biến cục bộ char lok_var2
lok_var3 :      ds      2       ; Biến cục bộ int lok_var3

rseg    ?BI?asm_funkt?ASM_MODUL1    ; Đoạn các biến bit cục bộ

lok_bvar :      dbit    1       ; Biến bít cục bộ bit lok_bvar

rseg    ?PR?asm_funkt?ASM_MODUL1    ; Bắt đầu phần chương trình của asm_funkt

asm_funkt :
mov     ?c_funkt?BYTE+0, lok_var1  ; int var1
mov     ?c_funkt?BYTE+1, lok_var1+1

mov     ?c_funkt?BYTE+2, lok_var2  ; char var2

mov     c, lok_bvar                ; bit var3
mov     ?c_funkt?BIT+0,  c

lcall   c_funkt                    ; Gọi hàm C

mov     lok_var3+0, R6             ; Trị trả về trong lok_var3
mov     lok_var3+1, R7
;……
```

Cách định nghĩa hàm “c_funkt” trong file nguồn C

```
#pragma SMALL
#pragma NOREGPARMS /* Chuyển tham số tiếp theo khối hàm */
```

```
/*…….*/

/*********************************************************************/
/** FUNKTION : int c_funkt (int var1, char var2, bit var3 ) **/
/*********************************************************************/

#pragma NOREGPARMS /* điều khiển biên dịch này cũng phải được khai báo khi định nghĩa */

int c_funkt (int var1, char var2, bit var3)
{
        int ret_val ; /* biến cục bộ */
        /* sử dụng tham số byte */
        ret_val = var2 + var1 ;

        /* sử dụng tham số bít */
        P1^1 = var3 ;

        return (ret_val) ; /* trị trả về */
}
```

Gọi hàm hợp ngữ và C trong cấu hình COMPACT

Với cấu hình bộ nhớ kiểu COMPACT, các tham số chuyển giao và các biến cục bộ được đặt vào vùng nhớ pdata. Hàm hợp ngữ có thể truy cập các biến bằng lệnh "MOVX @Ri". Hàm hợp ngữ được gọi trong file nguồn C như sau :

```
#pragma COMPACT
#pragma NOREGPARMS /* không chuyển tham số trong thanh ghi */
int asm_funkt (int var1, char var2, bit bvar1) ;

void main (void)
{
        int var1, ret_val ;
        char var2 ;
        bit bvar1 ;

        /*……*/

        ret_val = asm_funkt (var1, var2, bvar1)

        /*…….*/
}
```

Cách định nghĩa hàm trong file nguồn hợp ngữ như sau :

```
NAME    ASM_MODUL1          ; Định nghĩa tên modul

?PR?asm_funkt?ASM_MODUL1 segment CODE      ; Định nghĩa segment cho mã chương trình

?DT?asm_funkt?ASM_MODUL1 segment XDATA OVERLAYABLE ;  Định nghĩa segment cho tham số byte và các biến byte cục bộ

?BI?asm_funkt?ASM_MODUL1 segment BIT OVERLAYABLE      ;  Định nghĩa segment cho tham số bit và các biến bit cục bộ
```

```
Public asm_funkt, ?asm_funkt?BYTE, ?asm_funkt?BIT        ; Khai báo ký hiệu công cộng cho chương trình gọi C

;*************************************************************************
;** FUNKTION : int asm_funkt (int var1, char var2, bit bvar1) **
;*************************************************************************
rseg    ?PD?asm_funkt?ASM_MODUL1    ; Segment các biến byte cục bộ
?asm_funkt?BYTE :
var1 :  ds      2       ; Tham số int var1
var2 :  ds      1       ; Tham số char var2
lok_var : ds    2       ; Biến cục bộ int lok_var

rseg    ?BI?asm_funkt?ASM_MODUL1    ; Segment các biến bit cục bộ
?asm_funkt?BIT :
bvar1 :         dbit    1       ; Tham số bit bvar1
lok_bvar :      dbit    1       ; Biến bít cục bộ bit lok_bvar

rseg    ?PR?asm_funkt?ASM_MODUL1    ; Bắt đầu phần chương trình của asm_funkt
asm_funkt :

; Có thể truy cập các biến và tham số theo các tên var1, var2 và lok_var bằng lệnh MOVX @Ri
mov     R0, #LOW var2
movx    A, @R0                  ; lok_var = var1 + var2
mov     B, A
mov     R0, #LOW var1 + 1
movx    A, @R0
add     A, B
mov     R1, #LOW lok_var + 1
movx    @R1, A
dec     R0
dec     R1
movx    A, @R0
addc    A, #0
movx    @R1, A

; Truy cập tham số và các biến bít theo tên bvar1 và lok_bvar
mov     C, bvar1
mov     P1.1, C         ; P1.1 = bvar1

; Chuyển trị trả về, nội dung của biến lok_var
mov     R0, #LOW lok_var
movx    A, @R0
mov     R6, A
inc     R0
movx    A, @R0
mov     R7, A
ret                     ; Thoát khỏi hàm hợp ngữ
```

Một hàm C cũng có thể được gọi từ chương trình hợp ngữ. Việc gọi hàm C phải được định nghĩa trong file hợp ngữ

```
NAME    ASM_MODUL1      ; Định nghĩa tên modul
```

```
extrn CODE (c_funkt)            ; Khai báo hàm C bên ngoài
extrn XDATA (?c-funkt?BYTE) ; Khai báo tham số byte bên ngoài
extrn BIT (?c_funkt?BIT)        ; Khai báo tham số bít bên ngoài

?PR?asm_funkt?ASM_MODUL1 segment CODE     ; Định nghĩa đoạn mã chương trình
?PD?asm_funkt?ASM_MODUL1 segment XDATA OVERLAYABLE IPAGE ;     Định nghĩa đoạn tham số byte và biến byte cục bộ
?BI?asm_funkt?ASM_MODUL1 segment BIT OVERLAYABLE                ;     Định nghĩa đoạn tham số bit và biến bit cục bộ

;*********************************************
;** FUNKTION : void asm_funkt (void) **
;*********************************************
rseg     ?PD?asm_funkt?ASM_MODUL1     ; Đoạn biến byte cục bộ

lok_var1 :       ds      2       ; Biến cục bộ int lok_var1
lok_var2 :       ds      1       ; Biến cục bộ char lok_var2
lok_var3 :       ds      2       ; Biến cục bộ int lok_var3

rseg     ?BI?asm_funkt?ASM_MODUL1      ; Đoạn biến bit cục bộ

lok_bvar :       dbit    1       ; Biến bít cục bộ bit lok_bvar

rseg     ?PR?asm_funkt?ASM_MODUL1     ;     Bắt đầu đoạn chương trình asm_funkt
asm_funkt :
mov      R0, #LOW lok_var1
movx     A, @R0
mov      ?c_funkt?BYTE+0, A  ; int var1
inc      R0
movx     A, @R0
mov      ?c_funkt?BYTE+1, A
inc      R0
movx     A, @R0
mov      ?c_funkt?BYTE+2, A  ; char var2

mov      c, lok_bvar             ; bit var3
mov      ?c_funkt?BIT+0, c

lcall    c_funkt                 ; Gọi hàm c_funkt
mov      A, R6                   ; Trị trả về trong lok_var3
mov      R0, #LOW lok_var3
movx     @R0, A
inc      R0
mov      A, R7
movx     @R0, A
;…….
```

Cách định nghĩa hàm c_funkt trong file nguồn C

```
#pragma COMPACT
#pragma NOREGPARMS /* truyền tham số trong khối hàm */
int c_funkt (int var1, char var2, bit var3) /* khai báo hàm */
```

```
/*……..*/


/*****************************************************************/
/** FUNKTION : int c_funkt (int var1, char var2, bit var3) **/
/*****************************************************************/
#pragma NOREGPARMS /* điều khiển biên dịch này cũng được khai báo khi định nghĩa */
int c_funkt (int var1, char var2, bit var3)
{
        int ret_val ;

        /* áp dụng biến byte */
        ret_val = var1 + var2 ;

        /* áp dụng biến bít */
        P1^1 = var3 ;

        return (ret_val) ; /* trị trả về */
}
```

Gọi hàm hợp ngữ và C trong cấu hình LARGE

Trong cấu hình bộ nhớ kiểu LARGE, các tham số truyền và các biến cục bộ được đặt vào vùng nhớ XDATA. Hàm hợp ngữ có thể truy cập các biến này bằng lệnh MOVX @DPTR. Hàm hợp ngữ được gọi trong file nguồn C như sau :

```
#pragma LARGE
#pragma NOREGPARMS /* không truyền tham số trong thanh ghi */
int asm_funkt (int var1, char var2, bit bvar1) ;

void main (void)
{
        int var1, ret_val ;
        char var2 ;
        bit bvar1 ;
        /*……..*/

        ret_val = asm_funkt (var1, var2, bvar1) ;
        /*…….*/
}
```

Cách định nghĩa hàm trong file nguồn hợp ngữ

```
NAME    ASM_MODUL1          ; Định nghĩa tên modul

?PR?asm_funkt?ASM_MODUL1 segment CODE      ; Định nghĩa đoạn mã chương trình
?XD?asm_funkt?ASM_MODUL1 segment XDATA OVERLAYABLE ;  Định nghĩa đoạn tham số byte và các biến byte cục bộ
?BI?asm_funkt?ASM_MODUL1 segment BIT OVERLAYABLE       ;  Định nghĩa đoạn tham số bít và các biến bít cục bộ
Public asm_funkt, ?asm_funkt?BYTE, ?asm_funkt?BIT        ; Khai báo ký hiệu công cộng cho chương trình gọi C

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

```
;** FUNKTION : asm_funkt (int var1, char var2, bit bvar1) **
;*********************************************************************
rseg ?XD?asm_funkt?ASM_MODUL1 ; Đoạn biến byte cục bộ
?asm_funkt?BYTE :
var1 :    ds      2       ; Tham số int var1
var2 :    ds      1       ; Tham số chả var2

lok_var : ds      2       ; Biến cục bộ int lok_var

rseg ?BI?asm_funkt?ASM_MODUL1  ; Đoạn biến bít cục bộ
?asm_funkt?BIT :
bvar1 :           dbit    1       ; Tham số bít bvar1

lok_bvar          dbit    1       ; Biến bít cục bộ bit lok_bvar

rseg ?PR?asm_funkt?ASM_MODUL1 ; Bắt đầu chương trình asm_funkt
asm_funkt :
```

; Các biến và thông số được truy cập theo tên var1, var2 và lok_var bằng lệnh MOVX @DPTR

```
mov     DPTR, #var2
movx    A, @DPTR     ; lok_var = var1 + var2
mov     B, A
mov     DPTR, #var1+1
movx    A, @DPTR
add     A, B
mov     DPTR, #lok_var+1
movx    @DPTR, A
mov     DPTR, #var1
movx    A, @DPTR
addc    A, #0
mov     DPTR, #lok_var
movx    @DPTR, A

; Các biến và tham số bít được truy cập theo các tên bvar1 và lok_bvar
mov     c, bvar1
mov     P1.1, c         ; P1.1 = bvar1

; Trị trả về, nội dung của biến lok_var
mov     DPTR, #lok_var
movx    A, @R0
mov     R6, A
inc     DPTR
movx    A, @DPTR
mov     R7, A
ret                     ; Thoát khỏi hàm hợp ngữ
```

Hàm C cũng có thể được gọi từ chương trình hợp ngữ. Việc gọi hàm C phải được định nghĩa trong file hợp ngữ

```
NAME    ASM_MODUL1      ; Định nghĩa tên modul

extrn CODE (c_funkt)            ; Khai báo hàm C bên ngoài
extrn XDATA (?c_funkt?BYTE); Khai báo tham số byte bên ngoài
extrn BIT (?c_funkt?BIT)        ; Khai báo tham số bít bên ngoài
```

?PR?asm_funkt?ASM_MODUL1 segment CODE ; Định nghĩa đoạn mã chương trình

?XP?asm_funkt?ASM_MODUL1 segment XDATA OVERLAYABLE ; Định nghĩa đoạn tham số và các biến cục bộ byte

?BI?asm_funkt?ASM_MODUL1 segment BIT OVERLAYABLE ; Định nghĩa đoạn tham số và các biến cục bộ bít

```
;*********************************************
;** FUNKTION : void asm_funkt (void) **
;*********************************************
rseg ?XD?asm_funkt?ASM_MODUL1          ; Đoạn biến byte cục bộ

lok_var1:          ds    2             ; Biến cục bộ int lok_var1
lok_var2:          ds    1             ; Biến cục bộ char lok_var2
lok_var3 :         ds    2             ; Biến cục bộ int lok_var3

rseg ?BI?asm_funkt ?ASM_MODUL1         ; Đoạn biến bít cục bộ

lok_bvar :         dbit  1             ; Biến bít cục bộ bit lok_bvar

rseg ?PR?asm_funkt?ASM_MODUL1          ; Bắt đầu chương rình asm_funkt
asm_funkt :
        mov   DPTR, #lok_var1
        movx  A, @DPTR
        mov   ?c_funkt?BYTE+0, A  ; int var1
        inc   DPTR
        movx  A, @DPTR
        mov   ?c_funkt?BYTE+1, A

        inc   DPTR
        movx  A, @DPTR
        mov   ?c_funkt?BYTE+2, A  ; char var2

        mov   c, lok_bvar         ; bit var3
        mov   ?c_funkt?BIT+0, c

        lcall c_funkt             ; Gọi hàm c_funkt

        mov   A, R6               ; Trị trả về trong lok_var3
        mov   DPTR, #lok_var3
        movx  @DPTR, A
        inc   DPTR
        mov   A, R7
        movx  @DPTR, A
        ;……
```

Cách định nghĩa hàm c_funkt trong file nguồn C

```
#pragma LARGE
#pragma NOREGPARMS /* truyề tham số trong khối hàm */
int c_funkt (int var1, char var2, bit var3)
/*……*/

/*****************************************************************/
/** FUNKTION : int c_funkt (int var1, char var2, bit var3) **/
```

```
/*****************************************************************/
#pragma NOREGPARMS /* điều khiển biên dịch này cũng phải được khai báo khi định nghĩa */
int c_funkt (int var1, char var2, bit var3)
{
        int ret_val ; /* biến cục bộ */

        /* áp dụng các biến byte */
        ret_val = var1 + var2

        /* áp dụng các biến bít */
        P1^1 = var3

        return (ret_val) ; /* trị trả về */
}
```

12.4 Tối ưu hóa chương trình

Độ lớn và tốc độ của chương trình C51 phụ thuộc rất nhiều vào việc áp dụng một cách tối ưu các tính năng của vi điều khiển 8051. Do đó, lập trình viên cần phải chú ý đến cấu trúc của vi điều khiển :

- Là CPU 8 bít
- Dể truy cập bộ nhớ dữ liệu trong
- Thời gian truy cập bộ nhớ dữ liệu ngoài chậm
- Có vùng nhớ trong địa chỉ bít
- Khối nhân và chia 8 bít . Nên theo các hướng dẫn sau đây để tăng hiệu quả của chương trinh :
- Các biến được dùng thường xuyên nên đưa vào vùng data
- Con trỏ cũng nên được định nghĩa là data
- Tùy theo khả năng nên dùng con trỏ xác định bộ nhớ vì nó tạo ra mã trực tiếp. Con trỏ tổng quát dùng các hàm thư viện
- Nên dùng các kiểu dữ liệu ngắn. Các phép tính char có thể được thực hiện trực tiếp, trong khi đối với int, long hoặc float phải được đưa vào các hàm thư viện.
- Kiểu dữ liệu BIT rất phù hợp với các quyết định Y/N
- Những trường hợp có thể nên dùng bién unsigned. Vì khi xử lý các biến signed chương trình dịch phải tạo ra một mã lệnh phụ
- Các biến mảng nên đặt trong vùng nhớ địa chỉ gián tiếp (idata, pdata hoặc xdata)
- Các vùng và các cấu trúc lớn nên đặt trong vùng xdata để giãm tải cho bộ nhớ dữ liệu nội
- Nếu được nên chuyển các biến cục bộ thành toàn cục để có thể chồng nhau khi bắt đầu quá trình liên kết
- Nên cấu hình bộ nhớ thích hợp cho mỗi hàm để trình liên kết có thể tận dụng tối đa bộ nhớ
- Nên chọn các chuối tham số nối tiếp khi gọi hàm sao cho nhiều tham số có thể được truyền bằng thanh ghi

13. TRÌNH LIÊN KẾT

Nhiệm vụ của trình liên kết là tạo ra một file đối tượng địa chỉ tuyệt đối từ nhiều file đối tượng cho phép tái định vị. Sau đó file này sẻ được dùng cho chương trình thử hoặc máy nạp chương trình. Trình liên kết có thể được gọi theo hai cách :

```
L51 input-list [to output-file] [control-list]
```

Hoặc :

L51 @command-file

- *Input-list* : Là danh sách các file nhập được phân cách nhau bởi dấu phẩy. Phần mở rộng thường là .OBJ, các file này chứa các modul tái định vị cần liên kết lại với nhau để tạo ra chương trình địa chỉ tuyệt đối.
- *Output*-file : Là tên của file chương trình địa chỉ tuyệt đối, có thể không cần khai báo này khi đó tên của file đầu tiên trong input-list sẻ được chọn và không có phần mở rộng.
- *Control*-list : Chứa các lệnh và thông số của dòng lệnh gọi chương trình liên kết
- *Command*-file : Là file được đọc từ dòng lệnh nhập, lệnh và thông số trong file này tướng với trong dòng lệnh gọi

Ví dụ :

L51 PROG1
L51 PROG1, PROG2
L51 PROG1, PROG2 to PROGRAM. ABS
L51 PROG1, PROG2, PROG3 to PROGRAM. ABS RAMSIZE(256) CODE(100h)

L51 @LINK. MAK
LINK. MAK :
PROG1,
PROG2, PROG3
To PROGRAM. ABS
RAMSIZE(256) CODE(100h)

13.1 Lệnh điều khiển quá trình định vị

Các lệnh này tạo khả năng phân phát các địa chỉ tuyệt đối cho các segment tái định vị, sắp xếp các segment theo thứ tự và kiểm tra sự bố trí của bộ nhớ dữ liệu trong

RAMSIZE

Tham số RAMSIZE (giá trị) cố định độ lớn bộ nhớ dữ liệu trong tính theo byte nhờ đó trình liên kết có thể thích ứng với các loại vi điều khiển khác nhau trong họ 8051. Giá trị có thể từ 128 đến 256 byte, mặc định là 128 byte

Ví dụ :

L51 TEST. OBJ RAMSIZE(256)

Thông số định vị BIT, DATA, IDATA, XDATA, CODE

Thông số này cho phép phân phát các địa chỉ bộ nhớ cho các khối địa chỉ riêng của vi xử lý. Thông thường trình liên kết tự định vị các modul. Nhưng trong một số trường hợp cần phải đặt trước địa chỉ bắt đầu. Dạng tổng quát của thông số này có ý nghĩa như sau :

- Thông số (start-address [segmentname (start-address)…])

Địa chỉ ban đầu đã đươc khai báo có liên quan đến tất cả các segment tái định vị có trong vùng nhớ. Thông số này làm dễ dàng cho việc khai báo địa chỉ bắt đầu của một vi mạch nhớ bên ngoài

Các thông số có phạm vi địa chỉ như sau :

- BIT 00h – 7Fh (địa chỉ bít)
- DATA 00h – 7Fh
- IDATA 00h – 0FFh
- XDATA 0000h – 0FFFFh
- CODE 0000h – 0FFFFh

Với một ngoại lệ cho thông số BIT, tất cả các địa chỉ bít có liên quan đến địa chỉ byte. Các địa chỉ bít được khai báo với thông số BIT

Ví dụ :

L51 MODUL1 CODE(100h)
Mã chương trình được bắt đầu tại địa chỉ 100h

L51 MODUL2 XDATA (2000h INOUT(8000h))
Bộ nhớ dữ liệu ngoài bắt đầu từ địa chỉ 2000h, modul INOUT bắt đầu từ địa chỉ 8000h

13.2 Kỹ thuật chồng địa chỉ của trình liên kết

Trình dịch C51 tự động đánh dấu các vùng biến cục bộ là được phép chồng kể cả các biến trong các hàm hợp ngữ. Do đó, các đoạn dữ liệu và các đoạn chương trình phải được định nghĩa theo đúng quy ước của C51 (xem phần giao tiếp các hàm hợp ngữ và C). Trình liên kết phân tích việc gọi giữa các đoạn mã chương trình (chuẩn tham chiếu) để có thể thực hiện việc chồng hợp lý trong trường hợp gọi trực tiếp. Nhưng nên chú ý trong trường hợp gọi thông qua nhiều đoạn mã chương trình cũng như đối với hàm gián tiếp được gọi qua con trỏ, chương trình liên kết không thể tạo ra chuẩn tham chiếu. Đối với những trường hợp này cần thiết phải tạo chuẩn tham chiếu bằng tay.

Với thông số điều khiển NOOVERLAY se gián đoạn hoàn toàn việc chồng các đoạn bộ nhớ

Ví dụ :

L51 BEISP1. OBJ, BEISP2. OBJ to BEISP NOOVERLAY

Đoạn dữ liệu và đoạn bít được chồng nhau khi tất cả các điều kiện sau đây được thỏa mản

- Chỉ được phép gọi một đoạn mã lệnh từ một kiểu chương trình. Các kiểu chương trình là chương trình chính và chương trình ngắt
- Định nghĩa đoạn phải phù hợp với chuẩn C51
- Không tồn tại tham chiếu giữa các đoạn mã phụ thuộc

Quá trình phân tích chồng nhau được điều khiển bởi thông số điều khiển OVERLAY. Có nhiều loại khác nhau :

- OVERLAY : Tự động phân tích OVERLAY. Thông số này là mặc định
- OVERLAY (name ! cname [,…]) hoặc OVERLAY (name ! (cname1, cname2 [,…]) [,…]) : Các chuẩn tham chiếu phụ được định nghĩa, kế đó gọi name cname
- OVERLAY (name ~ cname [,…]) hoặc OVERLAY (name ~ (cname1, cname2 [,…]) [,…]) : Gở bỏ chuẩn tham chiếu, sau đó phải lưu vào nơi an toàn để không gọi name cname hoặc không được ghi đè lên các tham số và các biến
- OVERLAY (*! Name) hoặc OVERLAY (name !*) : hàm name được lấy ra từ quá trình xử lý OVERLAY. Không được liên quan đến các đoạn khác hoặc các hàm khác

Ví dụ :

L51 BEISP1. OBJ, BEISP2. OBJ to BEISP OVERLAY (FUNK1 ~ FUNK2)
Gở bỏ tham chiếu từ FUNK1 đến FUNK2. Chúng có thể được chồng

L51 BEISP1. OBJ, BEISP2. OBJ to BEISP OVERLAY (FUNK1 ~ (FUNK2, FUNK3))
Tham chiếu từ FUNK1 trên FUNK2 và FUNK3 được gở bỏ

L51 BEISP1. OBJ, BEISP2. OBJ to BEISP OVERLAY (FUNK1 ~ (FUNK2, FUNK3), FUNK1 ! (FUNK4, FUNK5))

Tham chiếu từ FUNK1 trên FUNK2 và FUNK3 được gở bỏ và tạo ra một tham chiếu từ FUNK1 trên FUNK4 và FUNK5

L51 BEISP1. OBJ BEISP2. OBJ to BEISP OVERLAY (FUNK1 !*)
FUNK1 được nhận từ quá trình xử lý OVERLAY

14. TRÌNH QUẢN LÝ THƯ VIỆN

Trình quản lý thư viện LIB51 được dùng để tập trung các hàm riêng lẻ vào chung trong một thư viện. Mẫu của các hàm được ghi vào các file tiêu đề (name.h). Khi một hàm trong thư viện được cần dùng đến thì file tiêu đề tương ứng phải được gắn trong file nguồn. Sau đó thư viện sẻ kết nối với trình liên kết tại nhưng nơi gắn này của modul. Trình liên kết sẻ tự tìm các hàm được dùng có trong thư viện

```
/****DATEI : myprog. C51 ****/
#include “mylib.h” /* file tiêu đề của thư viện mylib */

void main (void)
{
        int x, y, z ;

        z = myfunct1 (x, y) ; /* gọi một hàm từ thư viện mylib */
}
```

Sau đó trình liên kết sẻ được gọi

L51 myprog. Obj, mylib. Lib

Trình quản lý thư viện LIB51 có thể được điều khiển và tương tác bởi lệnh điều khiển trên dòng lệnh gọi. Khi gọi LIB51 không có lệnh điều khiển thì nó sẻ chuyển sang chế độ tương tác và hoển thị ký hiệu “*”. Bây giờ có thể nhập vào tất cả các lệnh điều khiển. Các lệnh sau đây có thể được dùng :

- Create FILENAME tạo thư viện mới

LIB51 c mylib. Lib

- Add FILENAME [(MODULNAME,…), FILENAME,…] to FILENAME : Nhập một hoặc nhiều modul chương trình vào thư viện. Nếu không khai báo tên modul thì toàn bộ nội dung của file đối tượng sẻ được nhập vào thư viện

LIB51 a myfunkt1. obj, myfunkt2. obj (funkt1, funkt2) to mylib. Lib

- List FILENAME [to listfile] [publics] : Hiển thị nội dung của một thư viện. Nếu khai báo listfile thì nội dung sẻ được ghi vào file này. Với publics các ký hiêuh public sẻ được hiển thị

REM Xuất ra trên màn hình
LIB51 l mylib.lib

REM xuất ra trên máy in
LIB51 l mylib.lib to LPT1 :

- Delete FILENAME (MODULNAME,…) : Xóa một hoặc nhiều moul chương trình (MODULNAME) khỏi thư viện (FILENAME). Tên modul phải được khai báo trong dấu ngoặc

LIB51 d mylib.lib (myfunkt1)

Nếu một hàm được thay đổi thì trước tiên phải gở bỏ phiên bản củ ra khỏi thư viện trước khi nhập phiên bản mới vào.

LIB51 d mylib.lib (myfunkt1)
LIB51 a myfunkt1.obj to mylib.lib

- Help : Gọi trợ giúp sử dụng LIB51
- Exit : Chấm dứt tương tác với LIB51

Tất cả các lệnh có thể được nhập tắt bằng ký tự đầu tiên

BÀI 2

Tên bài: **Vi điều khiển AT90S8535** Mã bài: **CIO 02 27 02**

GIỚI THIỆU

Bài này trình bày về đặc tính và cấu trúc của họ vi điều khiển AVR thông qua vi mạch cụ thể là AT90S8535. Nội dung bài thuần túy lý thuyết.

MỤC TIÊU THỰC HIỆN

- Biết được các tính năng mới của họ AVR so với họ vi điều khiển đã hoc là 8051
- Hiểu cấu tạo, nguyên lý hoạt động các khối chức năng tích hợp trong AVR
- Biết được sơ đồ chân và tín hiệu các chân của AT80S8535

NỘI DUNG CHÍNH

Nội dung bài học tập trung về các chủ đề chính như sau:

- Cấu trúc chung về AVR
- Đặc tính của AT90S8535
- Sơ đồ khối cấu tạo AT90S8535
- Tổ chức bộ nhớ
- Cấu tạo và hoạt động của bộ timer/counter
- Cấu tạo và hoạt động của bộ ADC
- Cấu tạo và hoạt động của bộ so sánh analog

1. MỞ ĐẦU

AT90S8535 l bộ vi điều khiển CMOS 8 bit tiêu thụ ít năng lượng dựa trên cấu trúc RISC (reduced instruction set computer) AVR. Nhờ thực hiện các lệnh mạnh trong một chu kì xung nhịp (a single clock cycle), AT90S8535 đạt được tốc độ xử lý dữ liệu lên đến 1 triệu lệnh/ giây ở tần số 1 MHz. Vi điều khiển này còn cho phép người thiết kế hệ thống tối ưu hóa mức độ tiêu thụ năng lượng mà vẫn đảm bảo tốc độ xử lý.

Phần cốt lỏi của AVR (AVR core) kết hợp tập lệnh với 32 thanh ghi làm việc đa năng. Tất cả 32 thanh ghi đều được nối trực tiếp với khối ALU (Arithmetic Logic Unit), cho phép truy cập hai thanh ghi độc lập bằng một lệnh đơn lẻ trong một chu kì xung nhịp. Cấu trúc đạt được tốc độ xử lí nhanh gấp 10 lần vi điều khiển CISC thông thường.

Các vi điều khiển AT90S8535 được hổ trợ bằng tất cả các công cụ lập trình phát triển hệ thống hiện có như trình dịch C, trình dịch Macro Assembler, trình gỡ rối/ mô phỏng và các bản mạch đánh giá.

Các đặc điểm của AT90S8535 được tóm tắt như sau:

- AVR cấu trúc RISC có chỉ tiêu chất lượng cao vì tiêu thụ ít năng lượng.
- 118 lệnh mạnh, hầu hết được thực hiện trong 1 chu kì xung nhịp (clock).
- 32 thanh ghi đa năng 8 bit.
- Tốc độ xử lý dữ liệu 8 triệu lệnh/ giây ở tần số 8MHz.

Bộ nhớ dữ liệu và bộ nhớ chương trình không tự mất dữ liệu (nonvolatile).

- Bộ nhớ flash 8k byte lập trình được ngay trong hệ thống, có thể ghi/xóa được 1000 lần, giao diện nối tiếp SPI để lập trình ngay trong hệ thống.
- 512 bytes EEPROM, có thể ghi/xóa được 100.000 lần.
- 512 bytes SRAM nội (internal SRAM).
- Lập trình khóa để bảo vệ phần mềm.

Các tính năng ngoại vi (peripheral features).

- Bộ biến đổi ADC 8 kênh, 10 bit.
- UART nối tiếp lập trình được.
- Giao diện SPI chủ / tớ nối tiếp.
- 2 Timer/Counter 8 bit với chế độ chia tần số và chế độ so sánh.
- 1 Timer/ Counter 16 bit với bộ chia tần số riêng biệt, chế độ so sánh và bắt mẫu (capture).
- Bộ định thời watchdog lập trình được với bộ dao động trên chip.
- Bộ so sánh tương tự có sẵn trên chip.

Các tính năng điều khiển đặc biệt.

- Có mạch power-on reset.
- Đồng hồ thời gian thực (Real-time clock) với bộ dao động riêng biệt và chế độ bộ đếm (counter mode).
- Các nguồn ngắt ngoài và trong.
- Có 3 chế độ ngủ: nghỉ (Idle), tiết kiệm năng lượng (Power save), và giảm năng lượng (Power-down).

Mức tiêu thụ năng lượng ở 4 MHz, 3V, 20^{o}C.

- Hoạt động: 6.4 mA.
- Chế độ nghỉ: 1.9 mA.
- Chế độ tiết kiệm năng lượng: < 1 μA

Vào/Ra và các hình thức đóng vỏ

- 32 đường xuất/ nhập có thể lập trình được.
- PDIP: 40 chân; PLCC, TQFP, MLF: 44 chân.

Điện áp hoạt động.

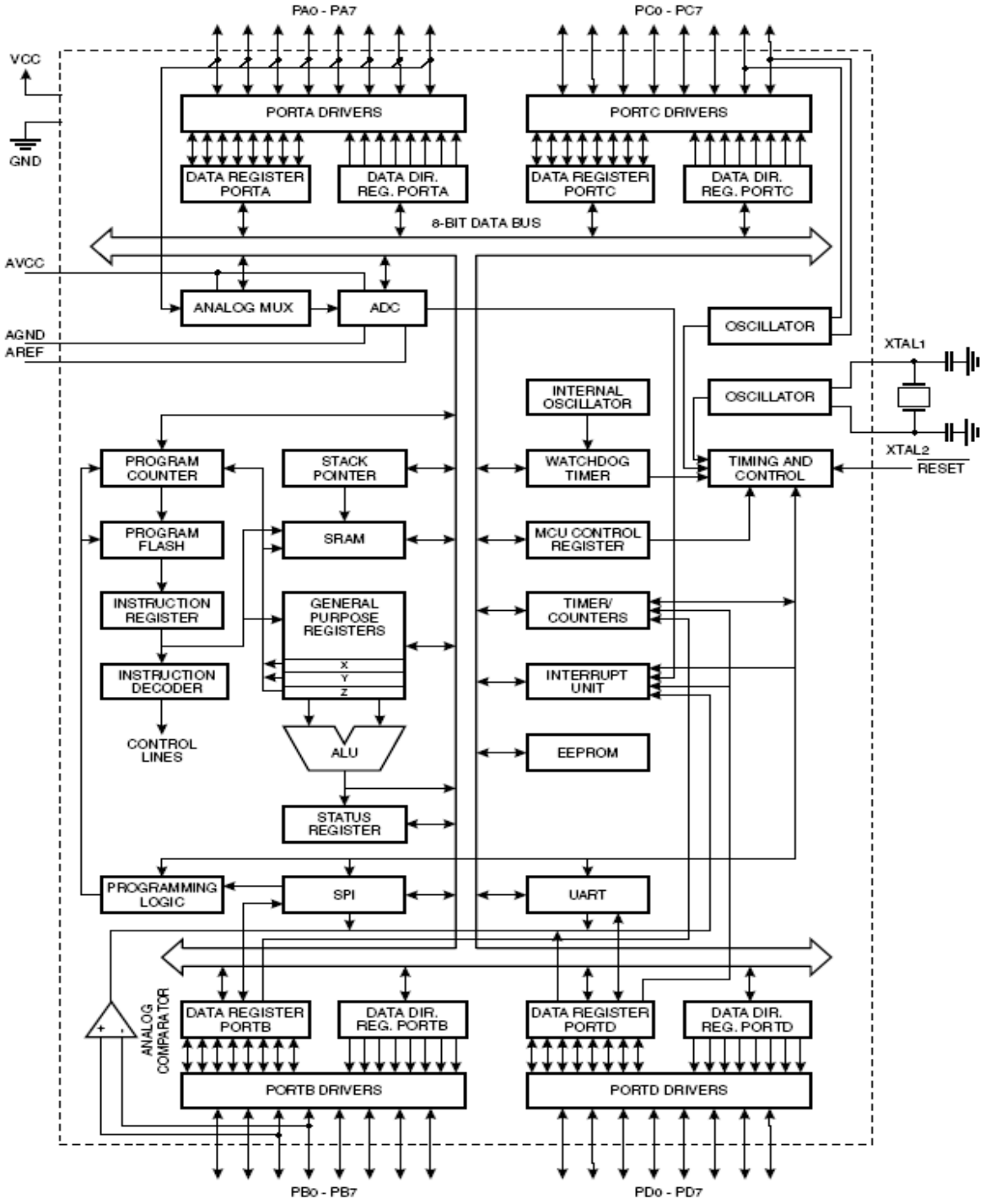
- Vcc: 4.0 V đến 6.0 V đối với AT90S8535.
- Vcc: 2.7 V đến 6.0 V đối với AT90LS8535.

Phân loại tốc độ xử lý.

- 0 đến 8 MHz đối với AT90S8535.

- 0 đến 4 MHz đối với AT90LS8535.

2. SƠ ĐỒ KHỐI AT90S8535



Hình 2.1 Sơ đồ khối AT90S8535

Vi điều khiển AT90S8535 có những tính năng sau: bộ nhớ flash có thể lập trình được ngay trong hệ thống với dung lượng 8Kbyte, bộ nhớ EEPROM 512 byte, bộ nhớ SRAM 512 byte, 32 đường vào/ ra đa năng, 32 thanh ghi làm việc đa năng, đồng hồ thời

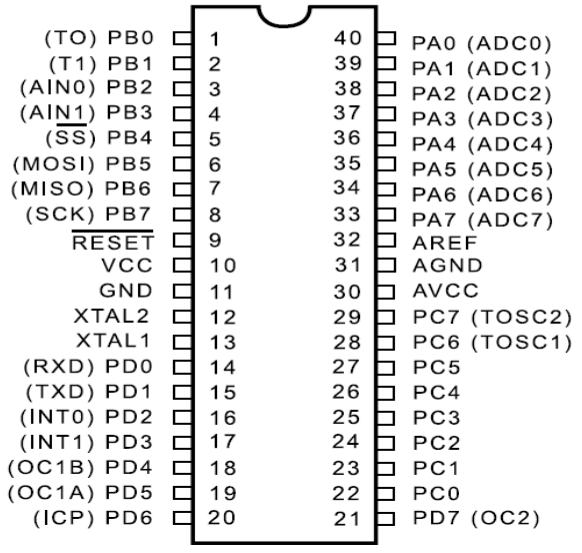
gian thực (Real-time clock), 3 timer/counter linh hoạt với chế độ so sánh, các ngắt bên trong và ngoài, một bộ UART nối tiếp lập trình được, bộ ADC 8 kênh 10 bit, bộ định thời watchdog lập trình được với bộ dao động bên trong, một cổng nối tiếp ISP và ba chế độ tiết kiệm năng lượng lựa chọn được bằng phần mềm. Chế độ Idle làm ngưng hoạt động của CPU trong khi bộ nhớ SRAM, các bộ định thời/ bộ đếm, cổng ISP và hệ thống ngắt vẫn tiếp tục hoạt động. Chế độ Power-down lưu trữ nội dung thanh ghi nhưng giải phóng bộ dao động, cấm tất cả các chức năng khác trên chip cho đến khi xuất hiện một ngắt kế tiếp hoặc tín hiệu reset phần cứng. Trong chế độ power-save, bộ dao động định thời (timer oscillator) tiếp tục chạy, cho phép người dùng tiếp tục trong một khoảng thời gian nào đấy (đặt trước) trong khi những phần còn lại ở trong trạng thái ngủ.

Vi điều khiển AVR được công ty Atmel chế tạo với công nghệ bộ nhớ không tự mất dữ liệu(non-volatile), mật độ cao. Bộ nhớ flash ISP trên chip cho phép bộ nhớ chương trình có thể được lập trình ngay trên hệ thống thông qua giao diện nối tiếp SPI hoặc một bộ nạp chương trình vào bộ nhớ không tự mất dữ liệu thông thường (a conventional nonvolatile memory programmer). Bằng cách kết hợp một CPU 8 bit có cấu truc RISC với bộ nhớ flash lập trình được trong hệ thống trên một chip đơn lẻ, AT90S8535 là l vi điều khiển mạnh với tính linh hoạt cao, giá thành hợp lý đối với nhiều ứng dụng điều khiển nhúng (embedded control applications).

3. MÔ TẢ CÁC CHÂN RA

3.1 Sơ đồ chân

PDIP



Hình 2.2 Sơ đồ chân PDIP-40

3.2 Chức năng các chân

VCC	Điện áp nuôi số (Digital Supply Voltage).
GND	Masse số

Port A (PA0…PA7)

Port A (cổng A) là cổng vào/ra hai hướng 8 bit. Các chân của cổng có các điện trở nối lên nguồn dương (pull – up resistor) (được chọn cho mỗi bit). Các chân ra của cổng A có thể chịu được dòng điện 20 mA đi qua và trực tiếp điều khiển Led hiển thị. Khi các chân PA0 đến PA7 là các ngỏ vào và được đặt xuống mức thấp từ bên ngoài, chúng sẽ là nguồn dòng nếu các điện trở kéo lên được kích hoạt.

Port A còn đóng vai trò là ngỏ vào của bộ chuyển đổi A/D (Analog to Digital Converter).

Các chân của port A được đặt ở trạng tháii thứ 3 (không cho phép xuất/ nhập dữ liệu) khi đang reset, hoặc ngay cả khi xung nhịp không hoạt động.

Port B (PB0…PB7)

Port B là cổng vào/ra hai hướng 8 bit có các điện trở kéo lên nguồn dương bên trong. Các chân ra của cổng B có thể chịu được dòng điện 20 mA đi qua. Khi các chân của cổng B là các ngỏ vào và được đặt xuống mức thấp từ bên ngoài, chúng sẽ là nguồn dòng nếu các điện trở nối lên nguồn dương được kích hoạt. Cổng B cũng cung cấp các chức năng ứng với các tính năng đặc biệt của AT90S8535 được trình bày ở bảng sau:

BẢNG 2.1 Chức năng các chân Port B

Chân	**Các chức năng chuyển đổi**
PB0	T0 (Timer/Counter0 External Counter Input)
PB1	T1 (Timer/Counter1 External Counter input)
PB2	AIN0 (Analog Comparator Positive Input)
PB3	AIN1 (Analog Comparator Negative Input)
PB4	$\overline{SS}$ (SPI Slave Select Input)
PB5	MOSI (SPI Bus Master Output/ Slave Input)
PB6	MISO (SPI Bus Master Input/ Slave Output)
PB7	SCK (SPI Bus Serial Clock)

Các chân của port B được đặt ở trạng thái thứ 3 (không cho phép xuất/ nhập dữ liệu) khi đang reset, hoặc ngay cả khi xung nhịp không hoạt động.

Port C (PC0…PC7)

Port C là cổng vào/ra hai hướng 8 bit có các điện trở nối lên nguồn dương bên trong. Các chân ra của cổng C có thể chịu được dòng điện 20 mA đi qua. Khi các chân của cổng C là các ngỏ vào và được đặt xuống mức thấp từ bên ngoài, chúng sẽ là nguồn dòng nếu các điện trở kéo lên nguồn dương được kích hoạt. Hai chân của port C có thể được sử dụng như bộ dao động cho timer/counter2.

Các chân của port C được đặt ở trạng thái thứ 3 (không cho phép xuất/ nhập dữ liệu) khi đang reset, hoặc ngay cả khi xung nhịp không hoạt động.

Port D (PD0…PD7)

Cổng D là cổng vào/ ra hai hướng 8 bit có các điện trở nối lên nguồn dương bên trong. Các chân ra của cổng D có thể chịu được dòng điện 20 mA đi qua. Khi các chân của cổng D là các ngỏ vào và được đặt xuống mức thấp từ bên ngoài, chúng sẽ là nguồn dòng nếu các điện trở nối lên nguồn dương được kích hoạt.

Cổng D cung cấp các chức năng ứng với các tính năng đặc biệt của AT90S8535 như trình bày ở bảng sau:

BẢNG 2.2 Chức năng các chân của port D

Chân	Các chức năng chuyển đổi
PD0	RXD (UART Input line)
PD1	TXD (UART Output line)
PD2	INT0 (External interrupt 0 input)
PD3	INT1 (External interrpupt 1 input)
PD4	OC1B (Timer/counter1 output compareB match output)
PD5	OC1A (Timer/counter1 output compareA match output)
PD6	ICP (Timer/counter1 input capture pin)
PD7	OC2 (Timer/counter2 output compare match output)

Các chân của port D được đặt ở trạng thái thứ 3 (không cho phép xuất/ nhập dữ liệu) khi đang reset, hoặc ngay cả khi xung nhịp không hoạt động.

RESET : Ngỏ vào đặt lại. Bộ vi điều khiển sẽ được đặt lại khi chân này ở mức thấp trong thời gian hơn 50 ns hoặc ngay cả khi không có tín hiệu giữ nhịp. Các xung ngắn hơn không tạo ra tín hiệu đặt lại.

XTAL 1: Lối vào bộ khuếch đại đảo và lối vào mạch tạo xung nhịp bên trong.

XTAL 2: Lối ra bộ khuếch đại dao động đảo (inverting oscillator amplifier).

AV_{cc} : AV_{CC} là chân cung cấp điện áp cho port A và bộ ADC. Nếu bộ ADC không được sử dụng thì chân này phải được nối lên VCC. Nếu bộ ADC được sử dụng thì chân này phải được nối với VCC qua mạch lọc thông thấp (low-pass filter).

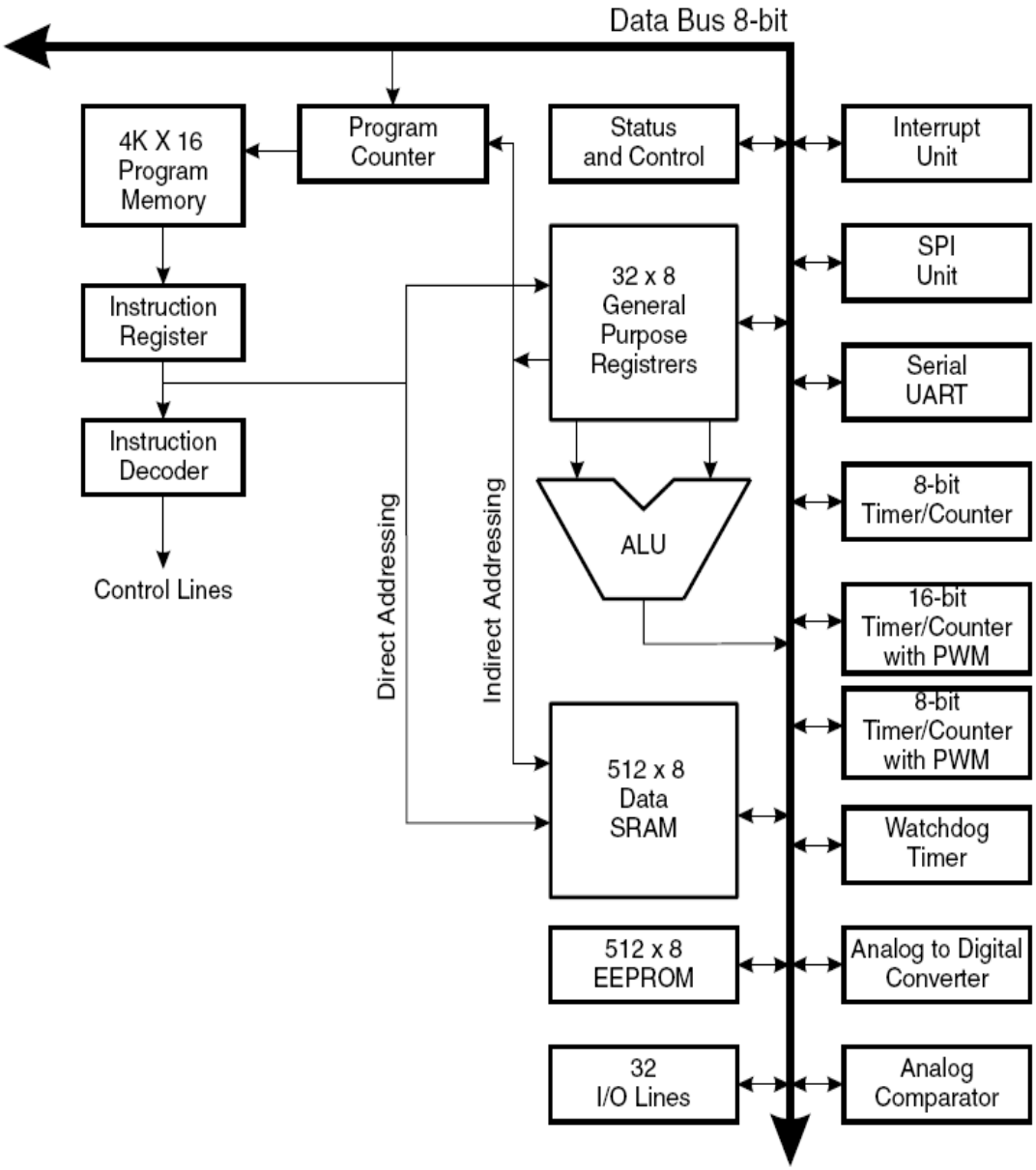
AREF : AREF là ngỏ vào điện áp chuẩn dùng cho bộ chuyển đổi ADC (Analog to Digital Converter). Để bộ ADC hoạt động, thì chân này phải được đặt mức điện áp khoảng từ 2V đến AV_{CC} .

AGND: AGND là chân nối đất cho phần mạch tương tự (analog ground).

4. CẤU TRÚC AT90S8535

4.1 Sơ lược

Khái niệm tập thanh ghi (register file) truy nhập nhanh gồm 32 thanh ghi làm việc đa năng 8 bit với thời gian truy nhập trong một chu kỳ xung nhịp (one single clock cycle). Điều này có nghĩa là trong một chu kỳ xung nhịp, ALU thực hiện được một phép toán. Hai toán hạng được xuất từ tập thanh ghi, phép toán được thực hiện với kết quả được lưu trở lại vào tập thanh ghi trong một chu kỳ xung nhịp. 6 trong số 32 thanh ghi này có thể làm 3 con trỏ địa chỉ gián tiếp 16 bit để định địa chỉ không gian dữ liệu (data space addressing) và cho phép tính địa chỉ hiệu dụng. Một trong 3 con trỏ địa chỉ còn được dùng làm con trỏ địa chỉ cho chức năng tìm kiếm bảng hằng số. Các thanh ghi có các chức năng bổ sung này là các thanh ghi 16 bit X,Y và Z.



Hình 2.3 Cấu trúc AT90S8535

Khối ALU hỗ trợ chức năng logic và số học giữa các thanh ghi hoặc giữa một hằng số với một thanh ghi. Các phép toán trên thanh ghi đơn (single register) cũng được thực hiện trong khối ALU. Hình trên thể hiện cấu trúc RISC AVR của vi điều khiển AT90S8535.

Thêm vào sự hoạt động của thanh ghi, những chế độ định địa chỉ bộ nhớ thông thường có thể được sử dụng trên tập thanh ghi (register file). Vì tập các thanh ghi đa năng được đặt tại 32 địa chỉ thấp nhất trong không gian dữ liệu (từ $00 đến $1F) nên chúng có thể truy nhập như những vị trí trong bộ nhớ thông thường.

Không gian bộ nhớ xuất/ nhập chứa 64 địa chỉ cho các chức năng ngoại vi của CPU các thanh ghi điều khiển, Timer/counter, bộ ADC và các chức năng xuất/ nhập khác. Bộ nhớ xuất/ nhập có thể truy nhập trực tiếp hoặc theo vị trí trong không gian dữ liệu của các tập thanh ghi (register file), từ địa chỉ $20 đến $5F.

AVR sử dụng cấu trúc Harvard với bộ nhớ và bus riêng biệt cho chương trình và dữ liệu. Bộ nhớ chương trình được hoạt động với một đường ống 2 tầng. Trong khi một lệnh

được thực hiện thì lệnh tiếp theo được gửi vào bộ nhớ chương trình. Điều này cho phép các lệnh được thực hiện trong mỗi chu kỳ xung nhịp (every clock cycle). Bộ nhớ chương trình là bộ nhớ flash có thể nạp ngay trong hệ thống.

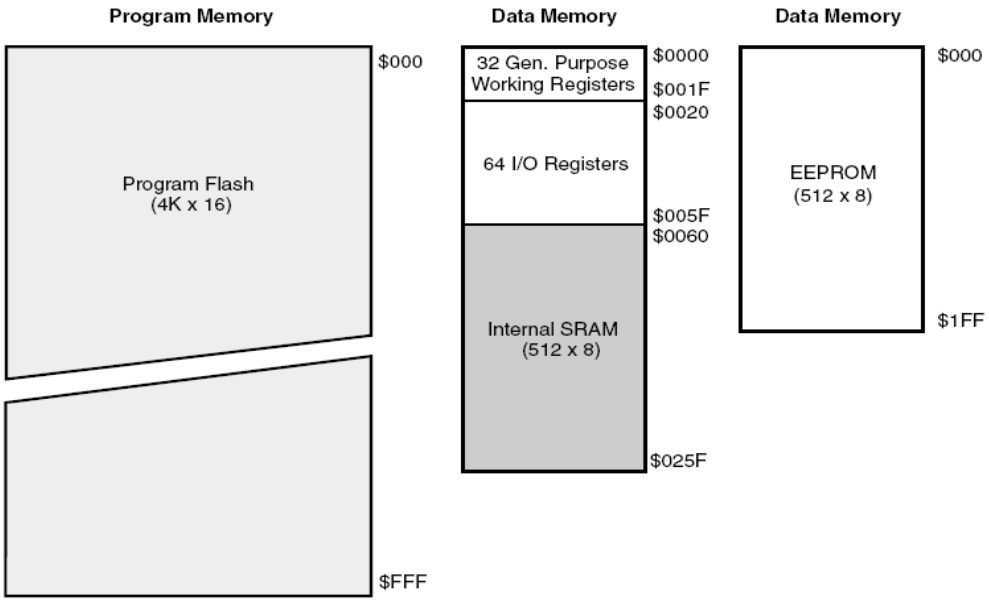
Toàn bộ không gian 4K địa chỉ được truy nhập trực tiếp nhờ các lệnh nhảy và lệnh gọi. Hầu hết các lệnh của AVR có định dạng từ đơn 16 bit. Mỗi địa chỉ của bộ nhớ chương trình chứa một lệnh 16 bit hoặc 32 bit.

Trong quá trình gọi ngắt và chương trình con (interrupts and subroutine) địa chỉ trở về của bộ đếm lệnh hay bộ đếm chương trình được lưu vào ngăn xếp. Ngăn xếp được cấp phát trong bộ nhớ dữ liệu SRAM nên kích thước của ngăn xếp chỉ bị giới hạn bởi dung lượng và với không gian đã sử dụng của bộ nhớ SRAM. Tất cả chương trình của người dùng phải khởi tạo con trỏ ngăn xếp SP (Stack Pointer) khi bắt đầu chương trình (trước khi các chương trình con và các ngắt được thực thi). Con trỏ ngăn xếp SP 10 bit được truy nhập để đọc/viết trong không gian vào/ra.

Bộ nhớ dữ liệu SRAM 512 byte có thể được truy nhập dễ dàng thông qua 5 chế độ định địa chỉ được cung cấp trong cấu trúc của AVR.

Các không gian nhớ trong cấu trúc AVR được định địa chỉ theo kiểu ánh xạ bộ nhớ tuyến tính đều.

4.2 Tổ chức bộ nhớ



Hình 2.4 Tổ chức bộ nhớ

Khối ngắt linh động (flexible interrupt module) có các thanh ghi điều khiển trong không gian vào/ra và một bit cho phép ngắt toàn cục trong thanh ghi trạng thái. Mỗi ngắt đều có một véc tơ ngắt riêng trong bảng véc tơ ngắt ở đầu bộ nhớ chương trình. Các ngắt có thứ tự ưu tiên tương ứng với vị trí véc tơ ngắt của chúng. Địa chỉ véc tơ ngắt càng thấp thì thứ tự ưu tiên càng cao.

- **Bộ nhớ chương trình (Program Memory).**

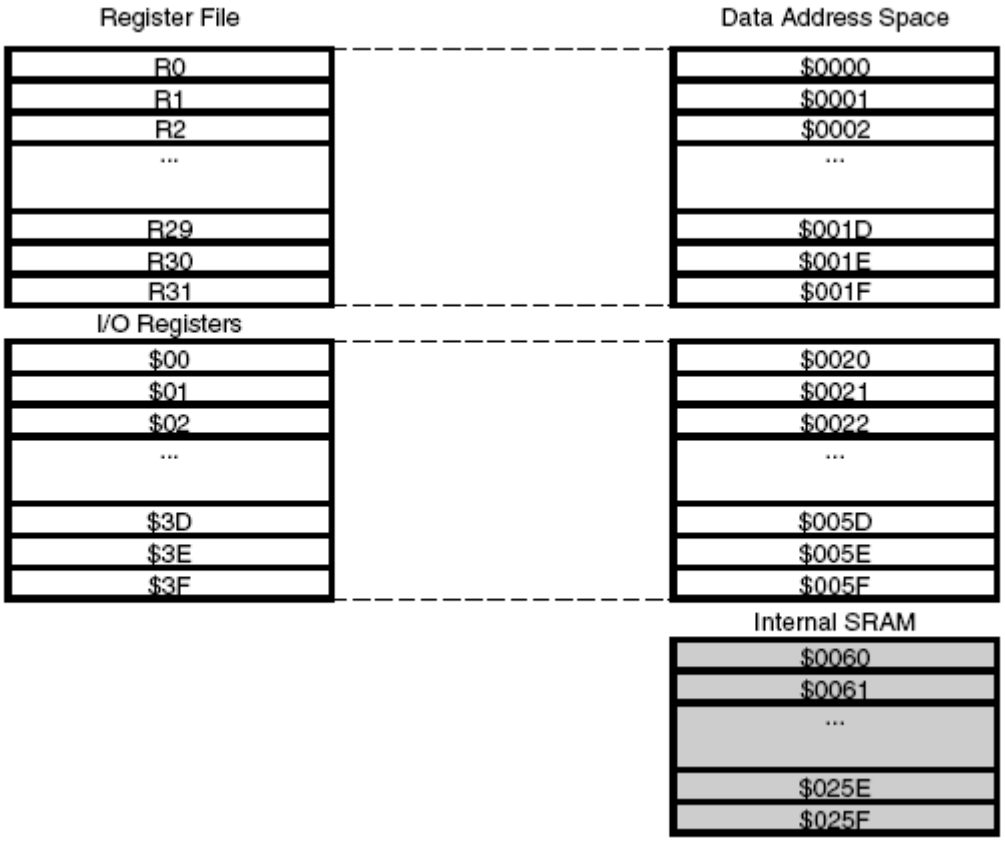
AT90S8535 chứa 8 K byte bộ nhớ flash lập trình được ngay trong hệ thống, dùng để lưu chương trình. Bộ nhớ này có thể chịu đựng ít nhất 1000 lần ghi xóa

.

- **Bộ nhớ dữ liệu SRAM (Static RAM Data Memory).**

Có năm chế độ định địa chỉ cho tất cả bộ nhớ dữ liệu : trực tiếp, gián tiếp với độ dịch chuyển, gián tiếp, gián tiếp với sự giảm trước (các thanh ghi địa chỉ X,Y hoặc Z bị giảm), gián tiếp với sự tăng sau (các thanh ghi địa chỉ X,Y hoặc Z tăng lên).

32 thanh ghi làm việc đa năng, 64 thanh ghi xuất/ nhập và 512 byte SRAM dữ liệu (internal data SRAM) của AT90S8535 có thể truy nhập được từ tất cả các chế độ định địa chỉ này.



Hình 2.5 Không gian dữ liệu SRAM

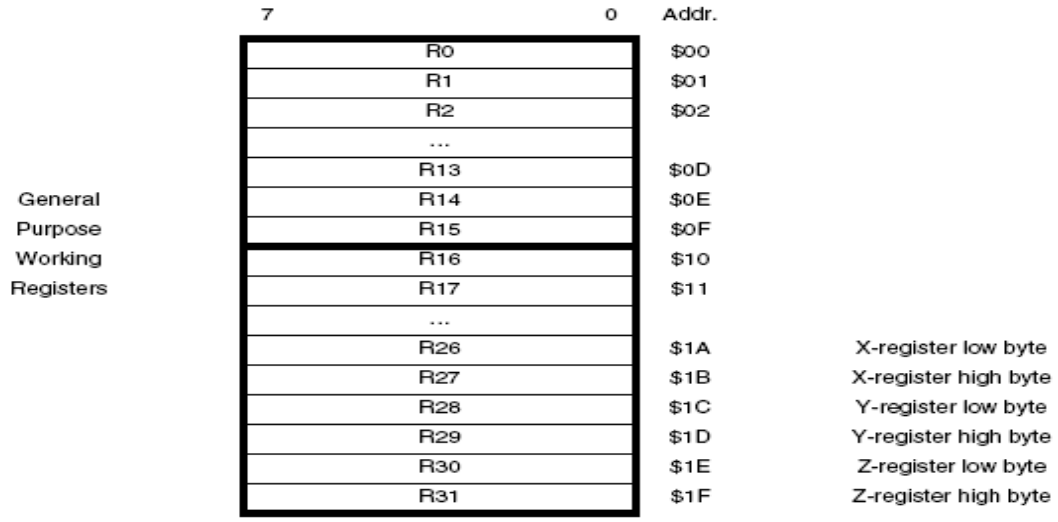
- **Bộ nhớ dữ liệu EEPROM (Electrically Erasable Programmable Read-Only Memory).**

AT90S8535 chứa 512 byte bộ nhớ dữ liệu EEPROM. Nó được tổ chức như một không gian dữ liệu riêng biệt gồm các byte đơn (single byte) có thể được đọc và ghi. Bộ nhớ EEPROM chịu đựng ít nhất 100.000 lần ghi xóa.

- **Thanh ghi làm việc đa năng (General Purpose Working Register).**

Tất cả các lệnh thực hiện trên thanh ghi (register operating instructions) đều có thể truy nhập trực tiếp đến mọi thanh ghi trong một chu kỳ máy. Ngoại trừ năm lệnh số học và logic giữa một hằng số với một thanh ghi: SBCI, SUBI, CPI, ANDI, ORI và lệnh nạp dữ liệu hằng số tức thời LDI. Các lệnh này áp dụng cho các thanh ghi đa năng từ R16 đến R31. Các lệnh SBC, SUB, CP, AND, OR và tất cả các lệnh khác giữa hai thanh ghi hoặc trên một thanh ghi áp dụng cho toàn bộ tập thanh ghi.

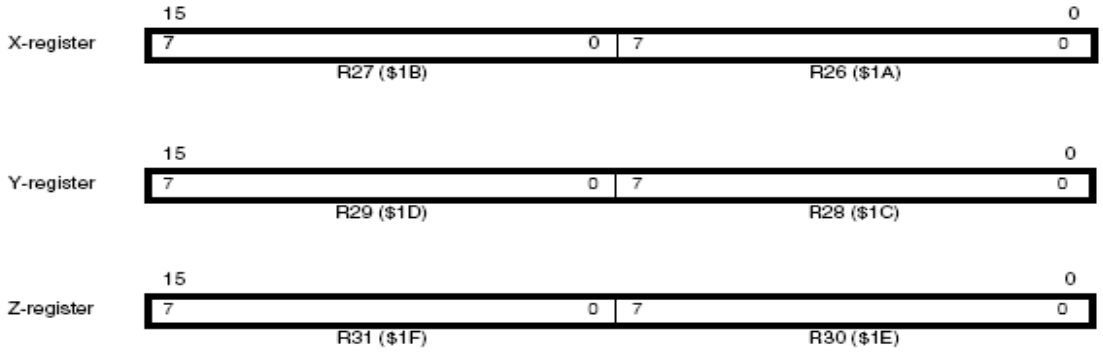
Trong các chế độ định địa chỉ khác, những thanh ghi địa chỉ (address registers) này đóng vai trò làm độ dịch cố định, tăng và giảm tự động.



Hình 2.6 Thanh ghi đa năng

Mỗi thanh ghi còn được gán một địa chỉ bộ nhớ dữ liệu (data memory address) ánh xạ (mapping) trực tiếp với 32 vị trí đầu tiên của không gian dữ liệu dành cho người sử dụng. Mặc dù không có các đặc điểm vật lý như các vị trí trong bộ nhớ của SRAM nhưng cách tổ chức bộ nhớ này làm cho việc truy nhập các thanh ghi rất linh động khi các thanh ghi X, Y, Z có thể được dùng để đánh số các thanh ghi trong tập thanh ghi đa năng.

Các thanh ghi từ R26 đến R31 có một số chức năng bổ sung. Những thanh ghi này là những con trỏ địa chỉ (address pointer) để định địa chỉ gián tiếp của không gian dữ liệu. Ba thanh ghi địa chỉ gián tiếp X, Y và Z được định nghĩa như sau:



Hình 2.7 Ba thanh ghi X, Y, Z

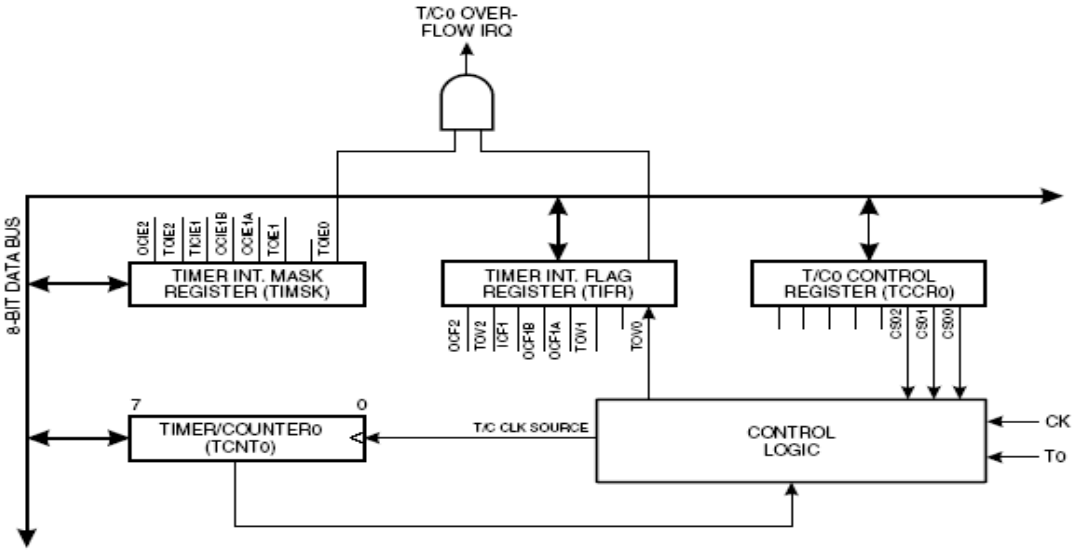
- **Bộ nhớ xuất/ nhập (I/O memory).**

Tất cả các địa chỉ xuất/ nhập ngoại vi được đặt trong không gian xuất/ nhập (I/O space). Các vị trí xuất/ nhập được truy nhập bởi lệnh IN và OUT chuyển đổi dữ liệu giữa 32 thanh ghi đa năng và không gian xuất nhập. Bằng cách sử dụng các lệnh SBI và CBI có thể truy cập bit trực tiếp đến các thanh ghi có địa chỉ từ $00 đến $1F. Trong những thanh ghi này, giá trị của từng bit đơn có thể được kiểm tra bằng cách sử dụng lệnh SIBS và SBIC. Khi sử dụng lệnh IN, OUT, địa chỉ xuất/ nhập $00 đến $3F phải được sử dụng. Khi định địa chỉ thanh ghi như SRAM, $20 phải được thêm vào những địa chỉ này.

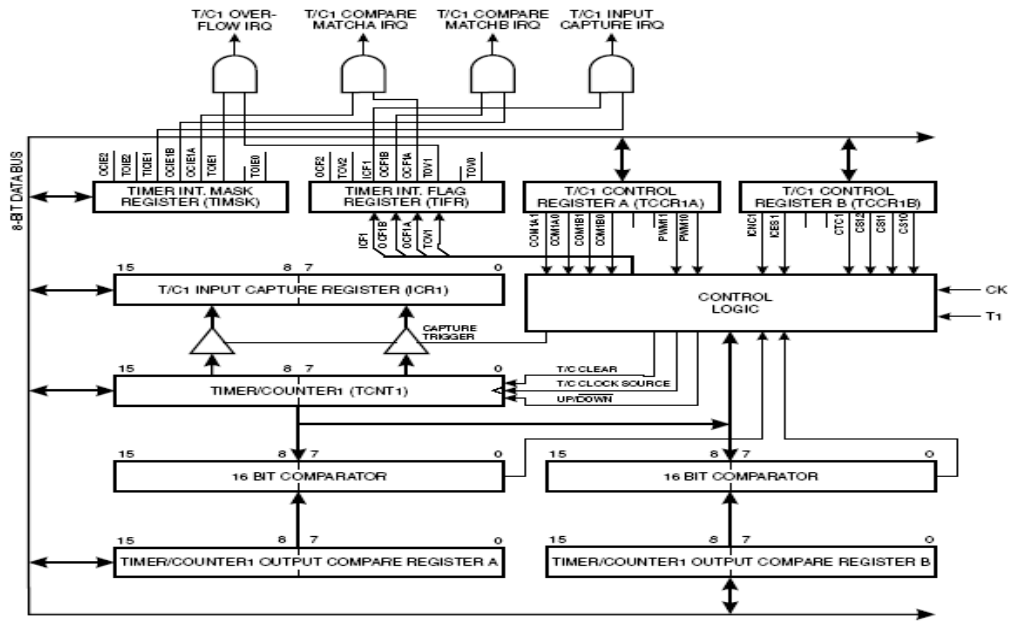
4.3 Hoạt động của Timer/Counter

AT90S8535 cung cấp ba bộ Timer/ Counter đa năng gồm hai bộ Timer/ Counter 8 bit và một bộ Timer/ Counter 16 bit. Timer/ Counter2 có thể được kích bởi bộ dao động ngoại (external oscillator). Bộ dao động ngoại này được sử dụng tối ưu nhất với thạch anh 32.768 kHz, nó cho phép Timer/ Counter2 hoạt động như một đồng hồ thời gian thực

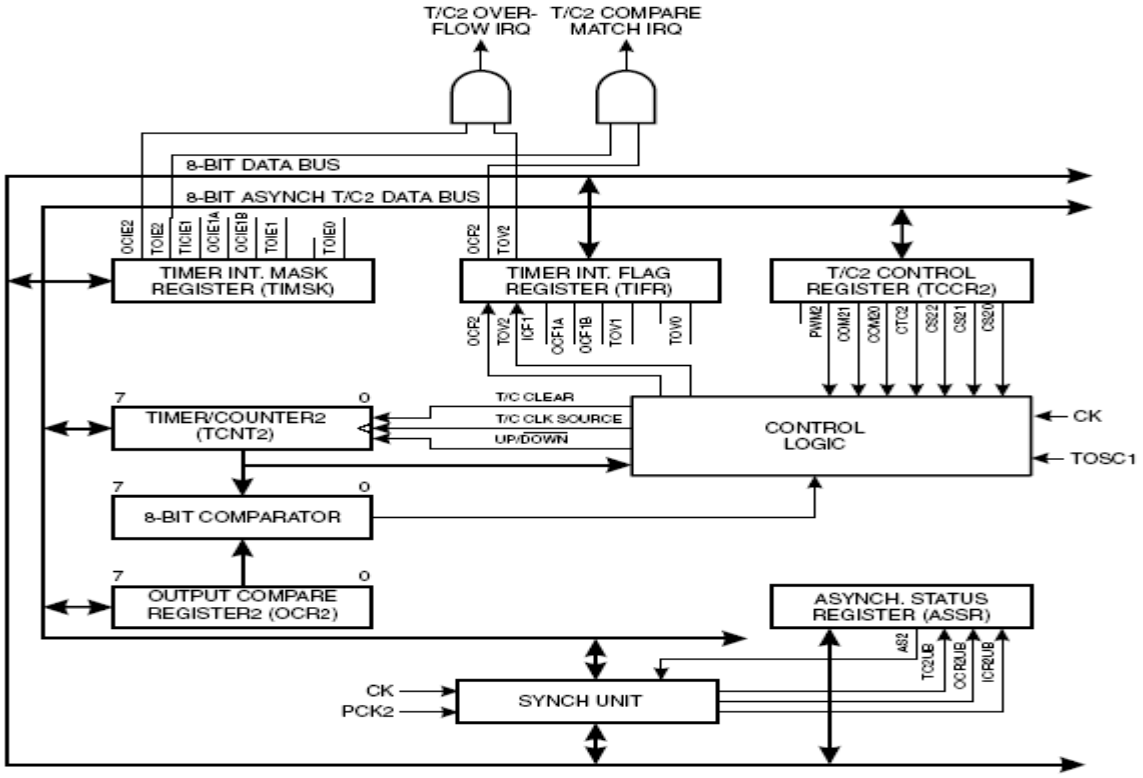
(Real-time clock). Timer/ Counter0 và Timer/ Counter1 có phần chia tần số riêng từ bộ chia tần số 10 bit. Timer/ Counter2 có bộ chia tần số ngay trong nó. Những Timer/ Counter này được sử dụng như một bộ định thời khi có tín hiệu giữ nhịp bên trong, hoặc được sử dụng như một bộ đếm khi có sự kết nối với xung kích của mạch đếm bên ngoài. Sau đây là các sơ đồ khối của Timer/ Counter0, Timer/ Counter1, Timer/ Counter2:



Hình 2.8 Sơ đồ khối timer/counter 0



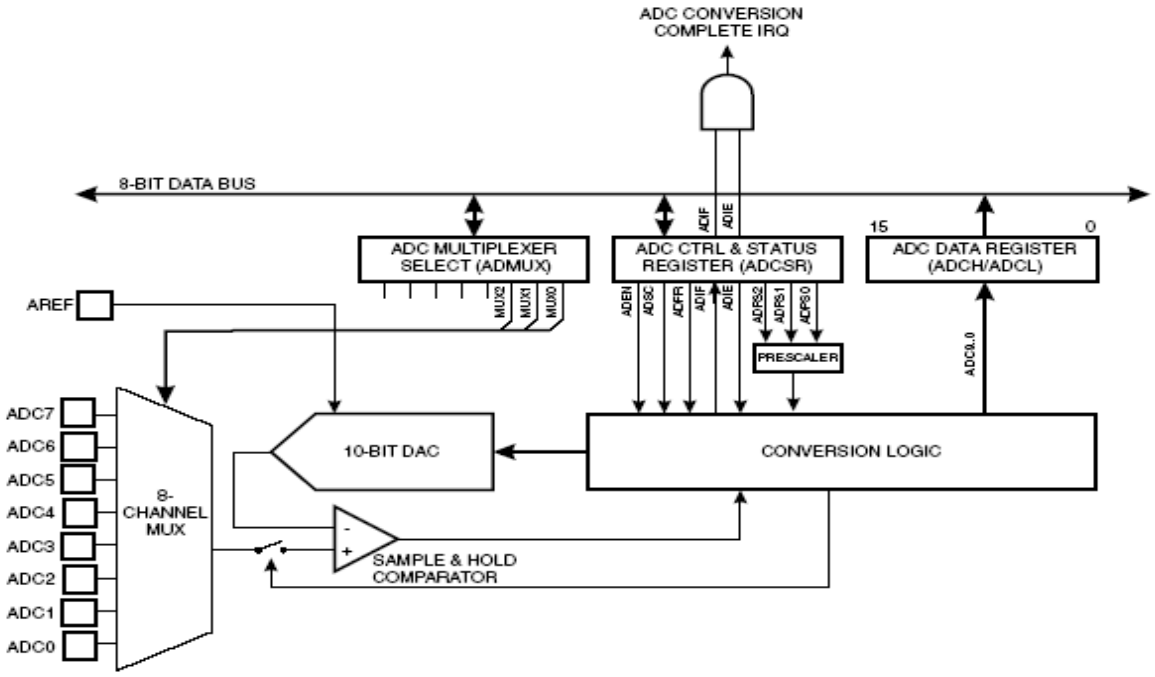
Hình 2.9 Sơ đồ khối timer/counter 1



Hình 2.10 Sơ đồ khối timer/counter 2

4.4 Bộ chuyển đổi ADC (Analog to Digital Converter)

Đặc trưng của AT90S8535 là có bộ ADC. Bộ ADC được nối với bộ dồn kênh Analog 8 kênh nó cho phép mỗi chân của port A được sử dụng như là ngỏ vào của bộ ADC. Bộ ADC chứa một mạch khuếch đại và lấy mẫu, nó bảo đảm điện áp ngỏ vào của bộ ADC được giữ ở một hằng số trong suốt quá trình chuyển đổi.



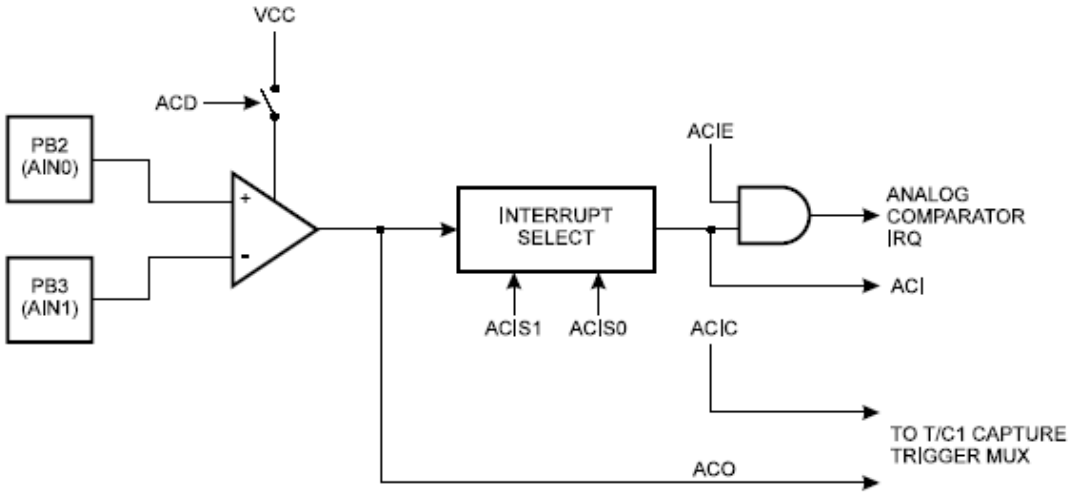
Hình 2.11 Sơ đồ khối bộ ADC

Bộ ADC có 2 chân cấp nguồn analog riêng biệt là AV_{CC} và AGND. AGND phải được nối đến GND và điện áp AVcc phải không khác nhiều hơn $\pm 0.3V$ so với Vcc.

Điện áp tham chiếu bên ngoài (external reference voltage) phải được đặt vào chân AREF. Điện áp này phải ở khoảng từ 2V-Avcc.

4.5 Bộ so sánh tương tự (analog comparator)

Bộ so sánh tương tự so sánh gía trị điện áp ở ngỏ vào dương PB2(AIN0) với ngỏ vào âm PB3 (AIN1). Khi điện áp ở ngỏ vào PB2 cao hơn điện áp đặt vào chân PB3 thì ngỏ ra của bộ so sánh tương tự được đặt lên mức “1”. Ngỏ ra này có thể được sử dụng cho bộ Timer/Counter1 để kích hoặc xóa ngắt bộ so sánh tương tự.



Hình 2.12 Bộ so sánh analog

BÀI 3

Tên bài Vi điều khiển PIC16F8x **Mã bài** CIO 02 27 03

GIỚI THIỆU

Bài này trình bày về đặc tính và cấu trúc của họ vi điều khiển của công tyMicrrochip thông qua họ vi mạch thông dụng là PIC16F8x. Nội dung bài gồm cả lý thuyết và thực hành nhờ các phần mềm biện dịch Assembler (MPASM) và C (MPLAB C17) kết hợp với các board mô phỏng phần cứng

MỤC TIÊU THỰC HIỆN

- Biết được các tính năng của họ PIC dùng ROM và flash ROM
- Hiểu cấu tạo, nguyên lý hoạt động các khối chức năng tích hợp trong PIC16F8x
- Biết được sơ đồ chân và tín hiệu các chân của PIC16F8x
- Viết được chương trình ứng dụng

NỘI DUNG CHÍNH

Nội dung bài học tập trung về các chủ đề chính như sau:

- Cấu trúc chung về họ PIC
- Đặc tính chung của PIC16CRxx và PIC16F8x
- Sơ đồ khối cấu tạo họ PIC16F8x
- Tổ chức bộ nhớ
- Cấu tạo và hoạt động của bộ timer/counter
- Tổ chức và cách truy cập bộ nhớ dữ liệu EEPROM
- Cấu tạo và tổ chức ngắt
- Bộ định thời canh chừng
- Các phương pháp reset
- Tập lệnh họ PIC16F8x

1. MÔ TẢ CHUNG

PIC16F8x là một nhóm trong ho vi điều khiển CMOS 8 bít PIC16Cxx giá rẻ, hiệu năng cao, chế độ tĩnh toàn phần. Nhóm này bao gồm các thiết bị sau

- PIC16F83
- PIC16F84
- PIC16CR83
- PIC16CR84

Tất cả các vi điều khiển họ PIC đều có cấu trúc RISC cải tiến. các thiết bị 16F8x được tăng cường các đặc tính chủ yếu, ngăn xếp có độ lồng sâu 8 cấp và nhiều nguồn tín hiệu ngắt trong cũng như ngoài. Bus mã lệnh và dữ liệu riêng biệt theo cấu trúc Harvard cho phép một lệnh 16 bít tách biệt với bus dữ liệu 8 bít. Đường ống lệnh hai tầng cho phép tất cả các lệnh được thực hiện chỉ trong một chu kỳ ngoại trừ các lệnh rẻ nhánh (các lệnh này cần đến hai chu kỳ) tổng cộng có 35 lệnh khả thi (tập lênh rút gọn). Thêm vào đó là một tập hợp các thanh ghi được sử dụng nhằm đạt mức độ thực thi cao.

Các vi điều khiển PIC16F8x điển hình đạt mức nén mã lệnh 2:1 và tốc độ tăng 4:1 (tại 20 MHz) so với các vi điều khiển 8 bít khác cùng loại.

PIC16F8x có đến 68 byte RAM, 64 byte bộ nhớ dữ liệu EEPROM và 13 chân I/O và một bộ timer/counter

Họ PIC16Cxx có những tính năng đặc biệt để giảm thiết bị ngoài, hạ giá thành, tăng độ tin cậy của hệ thống và giảm công suất tiêu thụ. Có bốn tùy chọn dao động : Dao động RC đơn cực cung cấp giải pháp rẻ tiền, dao động LP tối thiểu hóa công suất tiêu thụ, XT là một thạch anh chuẩn và HS đối với các thạch anh tốc độ cao. Chế độ ngũ (hạ nguồn) cho phép tiết kiệm năng lượng. Người dùng có thể đánh thức thiết bị từ chế độ này bằng các ngắt trong và ngoài và reset.

Một bộ định thời watchdog có độ tin cậy cao, bộ định thời này có mạch dao động RC riêng trên chip cung cấp khả năng treo máy

Các thiết bị với bộ nhớ chương trình flash cho phép đóng gói thiết bị giống nhau trong quy trình tạo mẫu và sản xuất. Với khả năng lập trình trong hệ thống cho phép cập nhật chương trình ngay trong ứng dụng mà không cần phải tháo rời thiết bị. Điều này rất hữu dụng cho việc phát triển nhiều ứng dụng mà việc tiếp cận thiết bị không dễ dàng nhưng lại cần thiết phải cập nhật mã lệnh và cũng rất có lợi cho các ứng dụng cần cập nhật mã lệnh từ xa VD thông tin về tốc độ.

Bảng 3.1 liệt kê các tính năng của họ PIC16F8x và một sơ đồ khối đơn giản được trình bày trong hình 3.1

PIC16F8x rất thích hợp trong các lĩnh vực ứng dụng tự động tốc độ cao, hệ thống điều khiển động cơ cho đến các cảm biến từ xa công suất thấp, khóa điện tử, thiết bị bảo mật và card thông minh. Công nghệ Flash/EEPROM cho phép khả năng tùy biến chương trình ứng dụng theo ý khách hàng (mã phát, tốc độ động cơ, tần số nhận, mã bảo mật…) rất nhanh và thuân tiện

Kích thước đóng gói nhỏ gọn thích hợp với tất cả các ứng dụng có không gian hạn chế. Giá thành rẻ, công suất thấp, hiệu năng cao và khả năng I/O linh động làm cho PIC16F8x rất linh hoạt ngay cả ở những nơi trước đây không dùng vi điều khiển như : Định thời, so sánh, PWM và các ứng dụng đồng xử lý.

Đặc tính lập trình trong hệ thống nối tiếp (thông qua hai chân) tạo nên tính linh hoạt tùy biến sản phẩm sau khi lắp ráp và kiểm tra hoàn tất. Tính năng này có thể được dùng để sản xuất sản phẩm, chứa dữ liệu định chuẩn hoặc lập trình thiết bị với phần dẻo hiện hành trước khi phân phối.

1.1 Khả năng tương thích

Với các người dùng đã quen thuộc với họ PIC16C5x sẽ thấy rằng đây là một phiên bản nâng cấp cấu trúc của họ PIC16C5x. Mã lệnh được viết cho PIC16C5x sẽ dễ dàng được thực hiện bởi PIC16F8x.

1.2 Công cụ hỗ trợ phát triển

Họ vi điều khiển PIC16Cxx được hỗ trợ bởi một trình dịch hợp ngữ đầy đủ tính năng, một phần mềm mô phỏng, một phần cứng thử nghiệm trong hệ thống, một máy nạp giá rẻ và đầy đủ tính năng. Một trình dịch "C" và cũng còn nhiều công cụ hỗ trợ logic mờ.

		PIC16F83	PIC16CR83	PIC16F84	PIC16CR84
Đồng hồ	Tần số (MHz)	10	10	10	10
Bộ nhớ	Bộ nhớ chương trình flash	512	-	1K	-
	Bộ nhớ chương trình EEPROM	-	-	-	-
	Bộ nhớ chương trình ROM	-	512	-	1K
	Bộ nhớ dữ liệu (bytes)	36	36	68	68
	Bộ nhớ dữ liệu EEPROM (bytes)	64	64	64	64
Ngoại vi	Timer	TMR0	TMR0	TMR0	TMR0
	Nguồn ngắt	4	4	4	4
	Chân I/O	13	13	13	13
Đặc tính	Điện áp (V)	2.0 – 6.0	2.0 – 6.0	2.0 – 6.0	2.0 – 6.0
	Dạng vỏ	18 pins DIP, SOIC	18 pins DIP, SOIC	18 pins DIP, SOIC	18 pins DIP, SOIC

Bảng 1.1 Họ PIC16F8x

2. CÁC LOẠI 16F8x

Tùy thuộc vào ứng dụng và yêu cầu sản xuất, tùy chọn thiết bị tương ứng có thể được chọn dựa trên các thông tin ở phần này.

Có 4 chủng loại thiết bị được chỉ ra trong số hiệu của thiết bị.

1. **F** như PIC16F84, đây là các thiết bị có bộ nhớ chương trình flash và hoạt động trong dải điện áp chuẩn.
2. **LF** như PIC16LF84, các thiết bị này có bộ nhớ chương trình flash và hoạt động trên dải điện áp mở rộng
3. **CR** như PIC16CR83, các thiết bị này có bộ nhớ chương trình loại ROM và hoạt động trong dải điện áp chuẩn
4. **LCR** như PIC16LCR83, là các thiết bị có bộ nhớ chương trình ROM và hoạt động trên bộ nhớ mở rộng

Khi đề cập đến bản đồ bộ nhớ và các tính năng cấu trúc khác, các ký hiệu F và CR cũng được dùng để ám chỉ LF và LCR.

2.1 Các thiết bị flash

Các thiết bị này được cung cấp dưới dạng vỏ plastic giá rẻ mặc dù chúng có thể xóa và lập trình lại. Việc này cho phép dùng một thiết bị giống nhau để phát triển mẫu và cũng như sản xuất các chương trình mẫu

Một ưu điểm nữa của phiên bản flash xóa bằng điện là chúng có thể được xóa và nạp chương trình lại trong hệ thống hoặc bằng máy nạp rời từ ngoài

2.2 Các thiết bị QTP

Công ty microchip cung cấp một dịch vụ lập trình QTP (quic-turnaround-production) cho các yêu cầu đặt hàng sản xuất công nghiệp. Dịch vụ này giúp người dùng có thể

chọn không lập trình một số lượng lớn thiết bị mà mã lệnh trong chúng đã được ổn định. Các thiết bị này có tất cả các vị trí là loại flash và các tùy chọn cấu hình đã được lập trình bởi nhà sản xuất. Các thủ tục kiểm mẫu và mã lệnh được áp dụng trước khi xuất xưởng.

2.3 Các thiết bị SQTP

Hảng microchip cung cấp một dịch vụ lập trình đơn lẻ. qua đó một vài vị trí trong thiết bị được xác định bởi người dùng được lập trình theo các số hiệu sản xuất khác nhau. Các số hiệu sản xuất có thể là ngẫu nhiên hoặc theo thứ tự.

Khả năng lập trình nối tiếp cho phép mỗi thiết bị có một số duy nhất và được dùng làm mã nhập, mật khẩu hoặc số nhận dạng

2.4 Các thiết bị ROM

Trong số thiết bị của microchip có một thiết bị tương ứng có bộ nhớ chương trình là ROM, các thiết bị này có giá rẻ hơn loại lập trình theo người dùng truyền thống (EPROM và EEPROM)

Các thiết bị ROM PIC16CR8x không cho biết thông tin sản xuất trong vùng bộ nhớ chương trình. Người dùng có thể nạp các thông tin này trong vùng EEPOM dữ liệu

3. SƠ LƯỢC VỀ CẤU TRÚC

Họ PIC16Cxx hiệu năng cao được xếp vào loại vi xử lý RISC thông dụng, họ này áp dụng cấu trúc Harvard có bộ nhớ chương trình và bộ nhớ dữ liệu được truy cập riêng, các thiết bị như thế có một bus bộ nhớ chương trình và một bus bộ nhớ dữ liệu, điều này làm cải thiện băng thông so với cấu trúc Neumann truyền thống ở đó chương trình và dữ liệu đều được lấy từ cùng một bộ nhớ (truy cập trên cùng một bus). Do bộ nhớ chương trình và dữ liệu riêng biệt còn cho phép độ dài các lệnh khác với độ dài dữ liệu 8 bít. Mã lệnh của PIC16Cxx có độ dài là 14 bít, toàn bộ bus bộ nhớ chương trình 14 bít nhận lệnh 14 bít chỉ trong một chu kỳ. Một đường ống hai tầng gối chồng các lệnh đọc vào với các lệnh đang thực hiện (ví dụ 3.1). Do đó, tất cả các lệnh được thực hiện trong một chu kỳ ngoại trừ các lệnh rẻ nhánh chương trình.

PIC16F83 và PIC16CR83 địa chỉ hóa 512 x 14 bộ nhớ chương trình, PIC16F84 và PIC16CR84 địa chỉ hóa 1 K x 14 bộ nhớ chương trình. Tất cả cả bộ nhớ chương trình là vùng nhớ bên trong.

Họ PIC16Cxx có thể định địa chỉ trực tiếp các dãy thanh ghi và bộ nhớ dữ liệu của chúng. Tất cả các thanh ghi chức năng đặc biệt bao gồm cả bộ đếm chương trình đều được ánh xạ trong vùng nhớ dữ liệu. Một tập lênh trực giao (đối xứng) giúp nó có khả năng thực hiện mọi thao tác trên mọi thanh ghi bàng cách định địa chỉ bất kỳ, dẫn đến kết quả là chương trình đơn giản và hiệu quả. Thêm vào đó quá trình học cũng giảm một cách đáng kể.

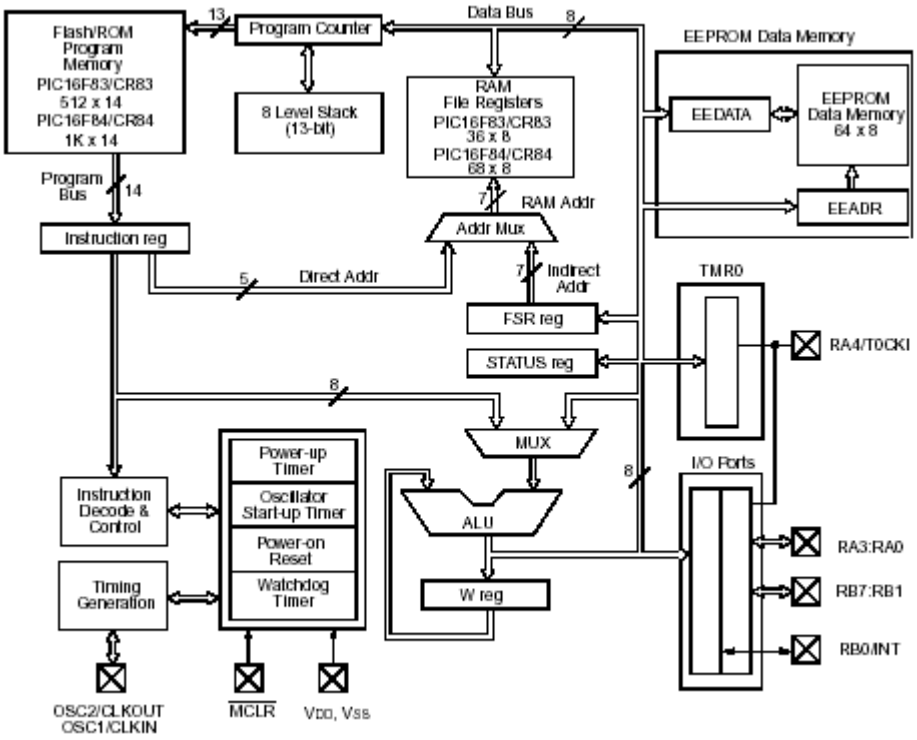
Các thiết bị PIC16Cxx gồm một khối ALU 8 bít và các thanh ghi hoạt động. Khối ALU là một đơn vị số học công dụng chung. Nó thực hiện các phép tính số học và logic giữa dữ liệu trong thanh ghi đang hoạt động với dãy thanh ghi bất kỳ.

ALU có độ lớn 8 bít có khả năng : Cộng, trừ, dịch chuyển và các phép toán logic. Trừ khi được lưu ý, nếu không thì các phép tính số học đều dựa trên cơ sở số bù bậc hai. Đối với các lệnh hai toán hạng : Một toán hạng là thanh ghi hoạt động và toán hạng khác là một thanh ghi dãy hoặc một hằng số tức thời. Đối với các lệnh một toán hạng, toán hạng là thanh ghi hoạt động hoặc một thanh ghi dãy.

Thanh ghi hoạt động là một thanh ghi 8 bít được dùng cho khối ALU, thanh ghi này không có địa chỉ.

Tùy theo lệnh được thực hiện, ALU có thể ảnh hưởng đến giá trị của các bít : Carry (C), Digit Carry (DC), và Zero (Z) trong thanh ghi trạng thái. Các bít C và DC có chức năng lần lượt như là bít ra $\overline{\text{borrow}}$ và $\overline{\text{digit borrow}}$ trong phép trừ

Sơ đồ khối của PIC16F8x được trình bày trong hình 3.1 với các chân tương ứng được mô tả trong bảng 3.2



Hình 3.1 So đồ khối PIC16F8x

Tên chân	Số DIP	Số SOIC	Kiểu I/O/P	Kiểu đệm	Mô tả
OSC1/CLKIN	16	16	I	ST/CMOS [1]	Ngỏ vào thạch anh hoặc dao động từ bên ngoài
OSC2/CLKOUT	15	15	O	-	Ngỏ ra thạch anh, nối đến thạch anh hoặc mạch cộng hưởng trong chế độ dao động thạch anh. Trong chế độ RC, tần số tại đây bằng ¼ OSC1 và biểu thị tốc độ chu kỳ lệnh
$\overline{\text{MCLR}}$	4	4	I/P	ST	Ngỏ vào reset tích cực mức thấp/ngỏ vào điện áp lập trình
RA0	17	17	I/O	TTL	Port A là một port I/O hai chiều
RA1	18	18	I/O	TTL	
RA2	1	1	I/O	TTL	
RA3	2	2	I/O	TTL	
RA4/T0CK1	3	3	I/O	ST	Ngỏ vào TMR0 của timer/counter, ngỏ ra là loại cực thoát để hở
					Port B là một port I/O hai chiều, port B có thể được lập trình bằng phần mềm cho tất cả các ngỏ vào kéo lên
RB0/INT	6	6	I/O	TTL/ST[2]	Có thẻ dược dùng làm ngỏ vào ngắt ngoài
RB1	7	7	I/O	TTL	
RB2	8	8	I/O	TTL	
RB3	9	9	I/O	TTL	
RB4	10	10	I/O	TTL	Thay đổi với ngắt
RB5	11	11	I/O	TTL	Thay đổi với ngắt
RB6	12	12	I/O	TTL/ST [3]	Thay đổi với ngắt, đồng hồ nạp nối tiếp
RB7	13	13	I/O	TTL/ST [3]	Thay đổi với ngắt, dữ liệu nạp nối tiếp
VSS	5	5	P	-	GND
VCC	14	14	P	-	Nguồn nuôi

Bảng 3.2 Mô tả chức năng các chân PIC16F8x3.1 Đồ thị thời gian/Chu kỳ lệnh

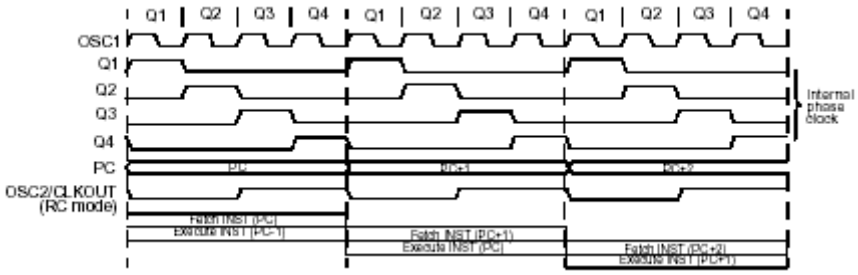
[1] Là ngỏ vào Smith trigger khi hoạt động với dao động RC, các trường hợp khác là ngỏ vào CMOS
[2] Là ngỏ vào Smith trigger khi được cấu hình là ngắt ngoài
[3] Là ngỏ vào Smĩth trigger ở chế độ nạp nối tiếp

Ngỏ vào đồng hồ (từ OSC1) được chia 4 ở bên trong để tạo ra bốn xung vuông lệch nhau Q1, Q2, Q3 và Q4. Bộ đếm chương trình (PC) được tăng bởi xung Q1, lệnh cần thực hiện được đọc từ bộ nhớ chương trình và chốt vào thanh ghi lệnh trong Q4. Lệnh được giải mã và thực hiện trong khoảng sau Q1 đến Q4. Các xung đồng hồ và tiến trình thực hiện lệnh được chỉ trong hình 3.2

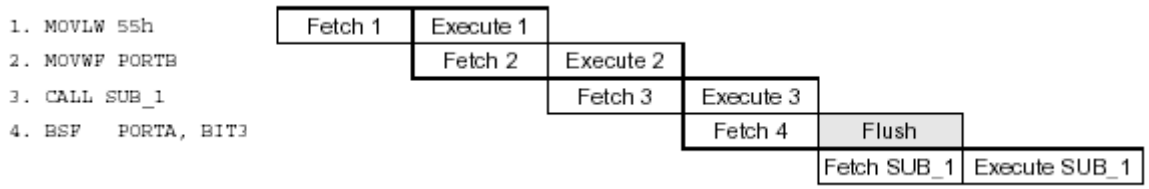
Thực hiện lệnh/đường ống

Một chu kỳ lệnh bao gồm bốn chu kỳ Q (Q1, Q2, Q3 và Q4), Thao tác nhập và thực hiện lệnh theo nguyên tắc đường ống. Thao tác đọc lệnh chiếm một chu kỳ lệnh trong khi giải mã và thực hiện lệnh chiếm một chu kỳ lệnh khác. Tuy nhiên, do hiệu ứng đường ống, mỗi lệnh được thực hiện trong một chu kỳ. Nếu một lệnh nào đó làm cho bộ đếm chương trình thay đổi VD lệnh goto, thì phải cần hai chu kỳ để hoàn tất lệnh này (VD 3.1)

Chu kỳ đọc lệnh bắt đầu qua việc tăng bộ đếm chương trình trong Q1. Trong chu kỳ thực hiện, lệnh đã đọc được chốt vào thanh ghi lệnh trong chu kỳ Q1. Lệnh này sau đó được giải mã và thực hiện trong khoảng thời gian Q2, Q3 và Q4. Bộ nhớ dữ liệu được đọc trong khoảng thời gian Q2 (đọc toán hạng) và được ghi trong khoảng thời gian Q4 (ghi vào đích)



Hình 3.2 Chu kỳ lệnh



Tất cả các lệnh là lệnh 1 chu kỳ trừ các lệnh rẻ nhánh vì phải lấy khỏi đường ống lệnh đã đọc vào trong khi đọc lệnh khác và thực hiện nó

Hình 3.3 Thực hiện lệnh theo đường ống

4. TỔ CHỨC BỘ NHỚ

Có hai khối nhớ trong PIC16F8x đó là bộ nhớ chương trình và bộ nhớ dữ liệu. Mỗi khối có bus riêng do đó việc truy cập đến mỗi khối có thể xảy ra trong cùng chu kỳ dao động.

Hơn nữa, bộ nhớ dữ liệu có thể phân ra thành vùng RAM công dụng chung và các thanh ghi chức năng đặc biệt (SFRs). Hoạt động của các SFR là trọng tâm mô tả trong phần này. Các SFR được dùng để điều khiển thiết bị ngoại vi và được trình bày theo từng khối ngoại vi tương ứng.

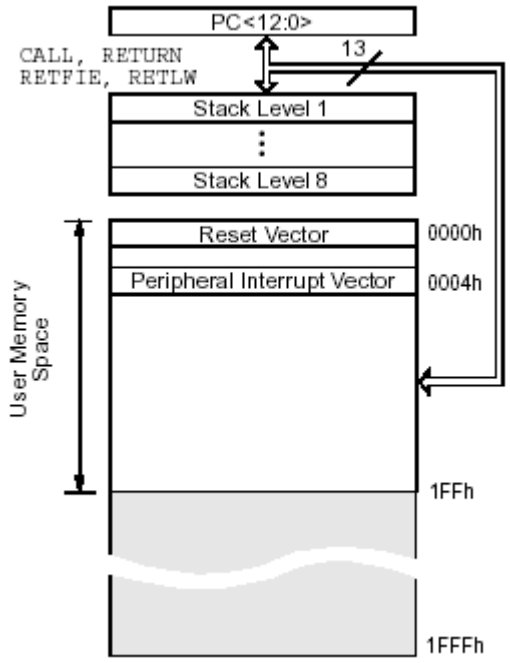
Vùng nhớ dữ liệu cũng bao gồm vùng dữ liệu EEPROM, vùng nhớ này không được ánh xạ trực tiếp trong vùng nhớ dữ liệu mà chỉ được ánh xạ gián tiếp. Có nghĩa là một

con trỏ địa chỉ gián tiếp sẽ xác định địa chỉ vùng nhớ EEPROM cho các yêu cầu đọc/viết. 64 byte dữ liệu EEPROM có địa chỉ trong khoảng từ 0h – 3Fh.

4.1 Tổ chức bộ nhớ chương trình

Họ PIC16Fxx có một bộ đếm chương trình 13 bít có khả năng định địa chỉ không gian nhớ chương trình 8 K x 14. Đối với PIC16F83 và PIC16CR83, không gian địa chỉ đầu tiên 512 x 14 (0000h – 01FFh) là địa chỉ vật lý (hình 3.4). Đối với PIC16F84 và PIC16CR84, không gian đầu tiên 1 K x 14 (0000h – 03FFh) là địa chỉ vật lý (hình 3.5). Việc truy cập một vị trí trên vùng địa chỉ vật lý sẽ gây ra một sự trùng lặp. Ví dụ : Đối với 16F84 các vị trí 20h, 420h, 820h, C20h, 1020h, 1420h, 1820h và 1C20h sẽ có cùng mã

Vectơ reset tại 0000h và vectơ ngắt tại 0004h



Hình 3.4 Bản đồ bộ nhớ chương trình và ngăn xếp – PIC16F83/CR83

4.2 Tổ chức bộ nhớ dữ liệu

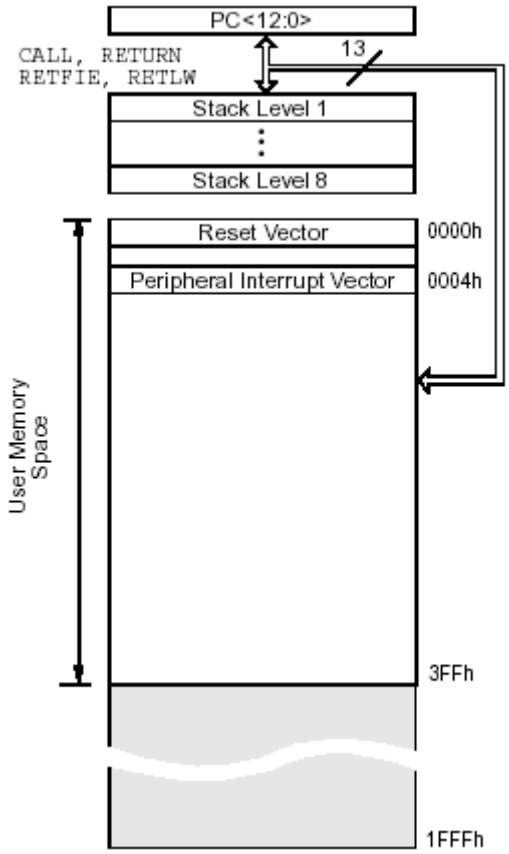
Bộ nhớ dữ liệu được chia thành hai vùng. Thứ nhất là vùng các thanh ghi đặc biệt và thứ hai là vùng các thanh ghi công dụng chung GPR (general purpose registers), các thanh ghi SFR điều khiển hoạt động của thiết bị.

Bộ nhớ dữ liệu được chia thành nhiều dãy cho cả hai vùng SFR và GPR. Các dãy GPR lớn hơn 116 byte RAM công dụng chung, dãy SFR là các thanh ghi điều khiển ngoại vi. Sự chia dãy cần cho việc sử dụng các bít điều khiển cho việc chọn dãy. Các bít điều khiển này nằm trong thanh ghi trạng thái. Hình 3.4 và hình 3.5 trình bày bản đồ tổ chức bộ nhớ dữ liệu.

Các lệnh MOVWF và MOVF chuyển các giá trị từ thanh ghi hoạt động (Working register) đến một vị trí bất kỳ trong dãy thanh ghi và ngược lại.

Toàn bộ bộ nhớ dữ liệu có thể được truy cập hoặc trực tiếp bằng địa chỉ tuyệt đối của mỗi dãy thanh ghi hoặc gián tiếp thông qua thanh ghi chọn dãy FSR (file select register) Địa chỉ gián tiếp sử dụng giá trị các bít RP1:RP0 để truy cập vào các vùng phân giải của bộ nhớ dữ liệu.

Bộ nhớ dữ liệu được phân thành hai dãy trong đó chứa các thanh ghi công dụng chung và các thanh ghi chức năng đặc biệt, dãy 0 được chọn bằng cách xóa bít RP0 (STATUS <5>), dãy 1 được chọn bằng cách đặt 1 cho bít RP0. Mỗi dãy trải dài đến 7Fh (128 byte). 12 vị trí đầu tiên của mỗi dãy được dành cho các thanh ghi chức năng đặc biệt phần còn lại là các thanh ghi công dụng chung là phần RAM tĩnh



Hình 3.5 Bản đồ bộ nhớ chương trình và ngăn xếp – PIC16F84/CR84

4.2.1 Dãy thanh ghi công dụng chung

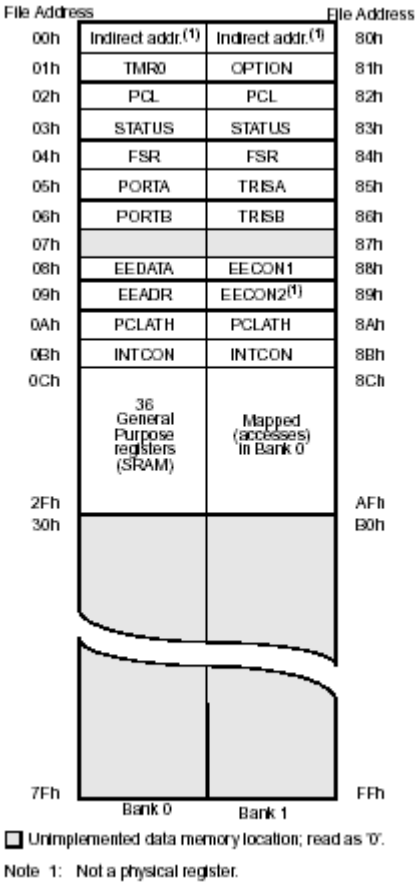
Tất cả các thiết bị đều có một số vùng các thanh ghi công dụng chung. Mỗi thanh ghi GPR có độ dài 8 bít và được truy cập hoặc trực tiếp hoặc gián tiếp thông qua FSR (mục 4.5).

Các địa chỉ GPR trong dãy 1 được ánh xạ vào các địa chỉ trong dãy 0 VD: Định địa chỉ 0Ch hoặc 8Ch sẽ truy cập cùng một GPR.

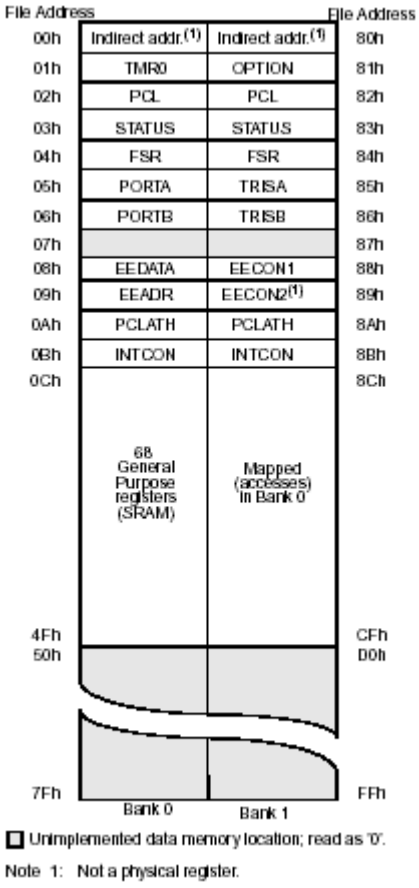
4.2.2 Các thanh ghi chức năng đặc biệt

Các thanh ghi chức năng đặc biệt (hình 3.6, 3.7 và bảng 3.3) được dùng bởi CPU và các chức năng ngoại vi để điều khiển hoạt động thiết bị, các thanh ghi này là RAM tĩnh.

Các thanh ghi đặc biệt có thể được chia thành hai nhóm : Trung tâm và ngoại vi. Chúng thích hợp với các chức năng trung tâm được mô rả trong phần này, nhóm còn lại liên quan đến hoạt động của các tính năng ngoại vi



Hình 3.6 Bản đồ dãy thanh ghi 16F83/CR83



Hình 3.7 Bản đồ dãy thanh ghi 16F84/CR84

Địa chỉ	Tên	Bít7	Bít 6	Bít 5	Bít 4	Bít 3	Bít 2	Bít 1	Bít 0	Giá trị khi power on rest	Các trường hợp reset khác ([1])
Dãy 0											
00h	**INDF**	**Đinh địa chỉ bộ nhớ bằng FSR (không phải thanh ghi vật lý)**								**---- ----**	**---- ----**
01h	**TMR0**	**Đồng hồ thời gian thực 8 bít / bộ đếm**								**xxxx xxxx**	**uuuu uuuu**
02h	**PCL**	**8 bít thấp của bộ đếm chương trình (PC)**								**0000 0000**	**0000 0000**
03h	**STATUS [2]**	**IRP**	**RP1**	**RP0**	$\overline{\text{T0}}$	$\overline{\text{PD}}$	**Z**	**DC**	**C**	**0001 1xxx**	**000q quuu**
04h	**FSR**	**Con trỏ 0 định địa chỉ gián tiếp bộ nhớ dữ liệu**								**xxxx xxxx**	**uuuu uuuu**
05h	**Port A**	**-**	**-**	**-**	**RA4/T0CK1**	**RA3**	**RA2**	**RA1**	**RA0**	**---x xxxx**	**---u uuuu**
06h	**Port B**	**RB7**	**RB6**	**RB5**	**RB4**	**RB3**	**RB2**	**RB1**	**RB0/INT**	**xxxx xxxx**	**uuuu uuuu**
07h		**Không dùng, có giá trị "0" khi đọc**								**---- ----**	**---- ----**
08h	**EEDATA**	**Thanh ghi dữ liệu EEPROM**								**xxxx xxxx**	**uuuu uuuu**
09h	**EEADR**	**Thanh ghi địa chỉ EEPROM**								**xxxx**	**uuuu**

[1] Reset ngoài bởi $\overline{\text{MCLR}}$ và reset do bộ định thời watchdog
[2] Các bít trạng thái T0 và PD không bị ảnh hưởng bởi MCLR

										xxxx	uuuu
0Ah	PCLATH	-	-	-	Bộ đệm 5 bít cao của PC					---0 0000	---0 0000
0bh	INTCON	GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	0000 000u
Dãy 1											
80h	INDF	Dùng nội dung FSR xác định bộ nhớ dữ liệu (không phải thg vật lý)								---- ----	---- ----
81h	OPTIONREG	$\overline{\text{RBPU}}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	1111 1111
82h	PCL	8 bít thấp của bộ đếm chương trinh (PC)								0000 0000	0000 0000
83h	STATUS [2]	IRP	RP1	RP0	$\overline{\text{T0}}$	$\overline{\text{PD}}$	Z	DC	C	0001 1xxx	000q quuu
84h	FSR	Con trỏ địa chỉ 0 định địa chỉ gián tiếp bộ nhớ dữ liệu								xxxx xxxx	uuuu uuuu
85h	TRISA	-	-	-	Thanh ghi dữ liệu port A					---1 1111	---1 1111
86h	TRISB	Thanh ghi dữ liệu port B								1111 1111	1111 1111
87h		Không dùng, khi đọc có giá trị 0								---- ----	---- ----
88h	EECON1	-	-	-	EEIF	WRERR	WREN	WR	RD	---0 x000	---0 q000
89h	EECON2	Thanh ghi điều khiển 2 EEPROM								---- ----	---- ----
0Ah	PCLATH	-	-	-	Bộ đệm ghi 5 bít cao của PC [1]					---0 0000	---0 0000
0bh	INTCON	GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	0000 000u

Ghi chú: x = Không xác định ; u = Không thay đổi ; - = không dùng khi đọc bằng 0 ; q = phụ thuộc vào điều kiện

Bảng 3.3 Tóm tắt các dãy thanh ghi

Thanh ghi trạng thái

Thanh ghi trạng thái chứa trạng thái số học của khối ALU, trạng thái reset và bít chọn dãy của bộ nhớ dữ liệu.

Như mọi thanh ghi khác, thanh ghi trạng thái có thể là đích của một lệnh bất kỳ. Nếu thanh ghi trạng thái là đích của một lệnh có ảnh hưởng đến các bít Z, DC hoặc C thì việc ghi vào ba bít này sẽ không có hiệu lực. Các bít này được đặt hoặc xóa là tùy thuộc vào thiết bị logic. Hơn nửa, các bít $\overline{\text{T0}}$ và $\overline{\text{PD}}$ không cho phép ghi. Vì vậy, kết quả của một lệnh lấy thanh ghi trạng thái làm đích sẽ khác với dự kiến. VD : CLRF STATUS sẽ xóa 3 bít trên và đặt 1 cho bít Z, các bít còn lại là 000u u1uu (u = không thay đổi).

Chỉ nên dùng các lệnh BCF, BSF, SWAPF và MOVWF để thay đổi thanh ghi trạng thái bởi vì các lệnh này không ảnh hưởng đến các bít trạng thái

Lưu ý 1:

Các bít IRP và RP1 (STATUS <7:6>) không được dùng đối với 16F8x mà nên lập trình như thông thường. Không nên dùng các bít này như các bít công dụng chung R/W vì điều này có thể ảnh hưởng đến khả năng tương thích với các sản phẩm trong tương lai

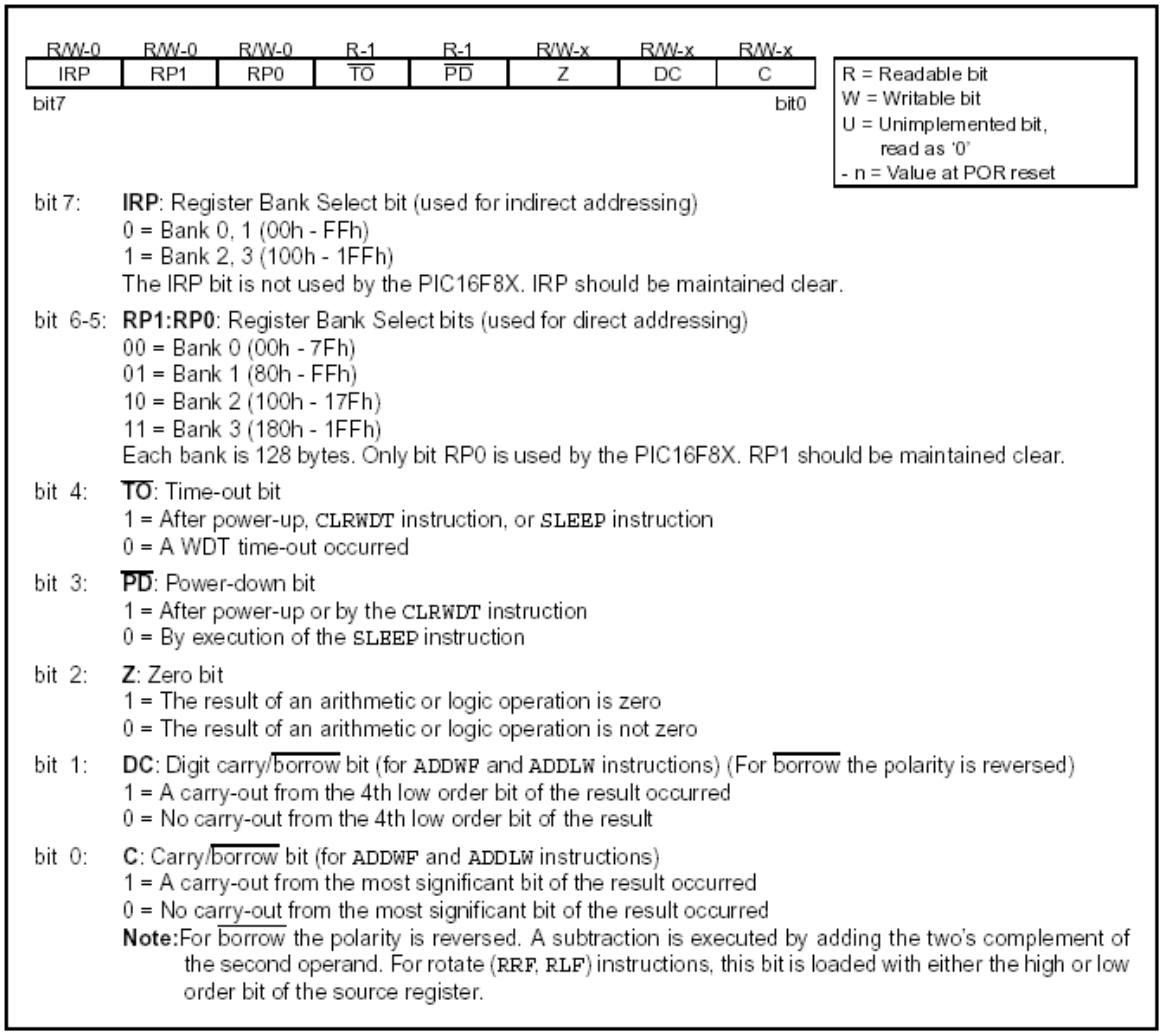
Lưu ý 2:

Các bít C và DC hoạt động như các bít ra số nhớ và số thiếu trong phép trừ. Xem ví dụ về các lệnh SUBLW và SUBWF

Lưu ý 3:

Khi thanh ghi trạng thái là đích của một lệnh có ảnh hưởng đến các bít Z, DC hoặc C thì việc ghi vào ba bít này không được cho phép, các bít xác định sẽ được cập nhật tùy thuộc vào thiết bị logic.

[1] Byte cao của PC không được truy cập trực tiếp. PCLATH là thanh ghi slave đối với PC<12:8>. Nội dung của PCLATH có thể chuyển lên byte cao của PC nhưng nội dung của PC<12:8> không thể chuyển vào PCLATH



Hình 3.8 Thanh ghi trạng thái (địa chỉ 03h, 83h)

Thanh ghi tùy chọn

Thanh ghi này cho phép đọc và ghi gồm các bít điều khiển khác nhau để cấu hình trước thang chia cho TMR0/WDT, ngắt ngoài INT, TMR0 và mạch kéo lên port B

Lưu ý:

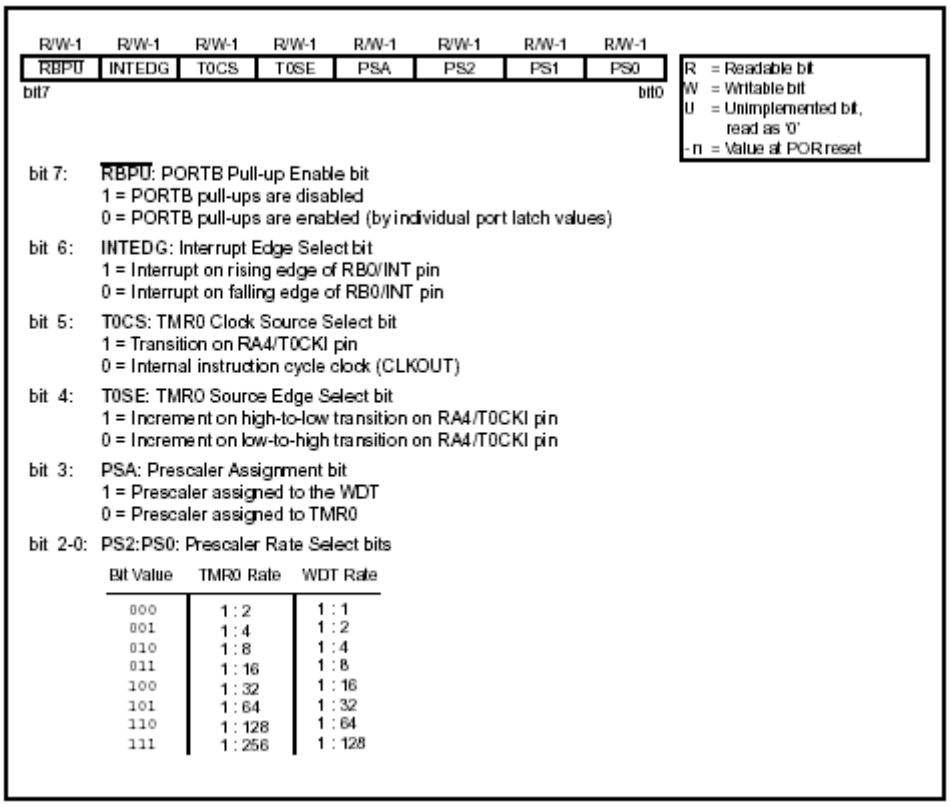
Khi cài đặt WDT (PSA = 1) thì TMR0 sẽ có thang chia là 1:1

Thanh ghi điều khiển ngắt INTCON

Thanh ghi điều khiển ngắt là thanh ghi đọc/viết, nó bao gồm các bít cho phép tất cả các nguồn tín hiệu ngắt

Lưu ý:

Các bít cờ ngắt được đặt khi một yêu cầu ngắt xảy ra bất chấp trạng thái của bít cho phép tương ứng hoặc bít cho phép toàn cục GIE (INTCON <7>)



R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit7							bit0

R = Readable bit
W = Writable bit
U = Unimplemented bit, read as '0'
-n = Value at POR reset

bit 7: RBPU: PORTB Pull-up Enable bit
1 = PORTB pull-ups are disabled
0 = PORTB pull-ups are enabled (by individual port latch values)

bit 6: INTEDG: Interrupt Edge Select bit
1 = Interrupt on rising edge of RB0/INT pin
0 = Interrupt on falling edge of RB0/INT pin

bit 5: T0CS: TMR0 Clock Source Select bit
1 = Transition on RA4/T0CKI pin
0 = Internal instruction cycle clock (CLKOUT)

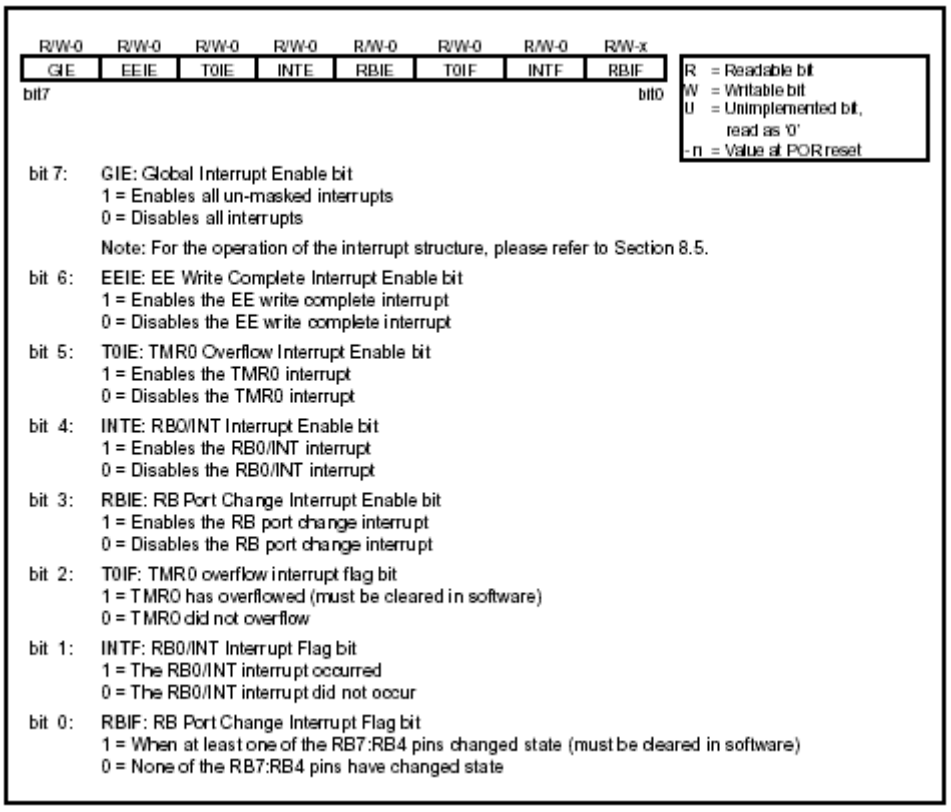
bit 4: T0SE: TMR0 Source Edge Select bit
1 = Increment on high-to-low transition on RA4/T0CKI pin
0 = Increment on low-to-high transition on RA4/T0CKI pin

bit 3: PSA: Prescaler Assignment bit
1 = Prescaler assigned to the WDT
0 = Prescaler assigned to TMR0

bit 2-0: PS2:PS0: Prescaler Rate Select bits

Bit Value	TMR0 Rate	WDT Rate
000	1 : 2	1 : 1
001	1 : 4	1 : 2
010	1 : 8	1 : 4
011	1 : 16	1 : 8
100	1 : 32	1 : 16
101	1 : 64	1 : 32
110	1 : 128	1 : 64
111	1 : 256	1 : 128

Hình 3.9 Thanh ghi tùy chọn (địa chỉ 81h)



R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
bit7							bit0

R = Readable bit
W = Writable bit
U = Unimplemented bit, read as '0'
-n = Value at POR reset

bit 7: GIE: Global Interrupt Enable bit
1 = Enables all un-masked interrupts
0 = Disables all interrupts

Note: For the operation of the interrupt structure, please refer to Section 8.5.

bit 6: EEIE: EE Write Complete Interrupt Enable bit
1 = Enables the EE write complete interrupt
0 = Disables the EE write complete interrupt

bit 5: T0IE: TMR0 Overflow Interrupt Enable bit
1 = Enables the TMR0 interrupt
0 = Disables the TMR0 interrupt

bit 4: INTE: RB0/INT Interrupt Enable bit
1 = Enables the RB0/INT interrupt
0 = Disables the RB0/INT interrupt

bit 3: RBIE: RB Port Change Interrupt Enable bit
1 = Enables the RB port change interrupt
0 = Disables the RB port change interrupt

bit 2: T0IF: TMR0 overflow interrupt flag bit
1 = TMR0 has overflowed (must be cleared in software)
0 = TMR0 did not overflow

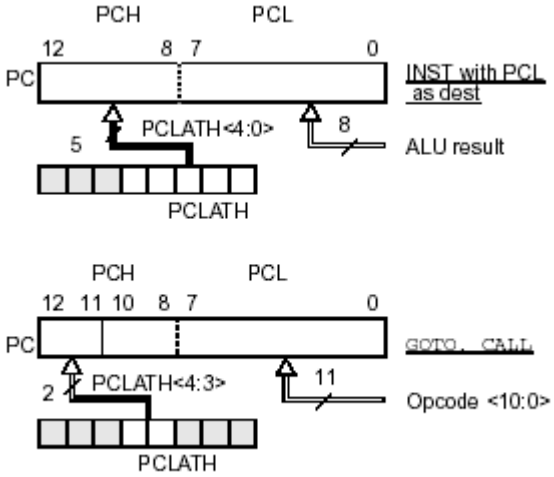
bit 1: INTF: RB0/INT Interrupt Flag bit
1 = The RB0/INT interrupt occurred
0 = The RB0/INT interrupt did not occur

bit 0: RBIF: RB Port Change Interrupt Flag bit
1 = When at least one of the RB7:RB4 pins changed state (must be cleared in software)
0 = None of the RB7:RB4 pins have changed state

Hình 3.10 Thanh ghi điều khiển ngắt (địa chỉ 08h, 88h)

4.3 Bộ đếm chương trình, PCL và PCLATH

Bộ đếm chương trình PC dài 13 bít, byte thấp là thanh ghi PCL là một thanh ghi đọc/viết. Byte cao của PC (PC <12:8>) không thể đọc hoặc viết trực tiếp và thông qua thanh ghi PCLATH. Thanh ghi PCLATH (PC latch high) là thanh ghi chốt cho PC <12:8>. Nội dung của PCLATH được chuyển byte cao của bộ đếm chương trình khi PC được nạp giá trị mới. Điều này xảy ra trong thời gian lệnh CALL, GOTO hoặc ghi vào PCL. Các bít cao của PC được nạp từ PCLATH như trình bày trong hình 3.11



Hình 3.11 Nạp PC từ các vị trí khác nhau

4.3.1 Nhảy đến địa chỉ đã xác định

Việc nhảy đến địa chỉ đã xác định được hoàn thành bằng cách cộng một độ lệch (offset) vào bộ đếm chương trình (ADDWF PCL). Trong khi đọc một bảng bằng phương pháp GOTO nên kiểm tra cẩn thận sự trùng lặp giữa vị trí của bảng với phạm vi bộ nhớ PCL (mỗi khối 256 word).

4.3.2 Phân trang bộ nhớ chương trình

PIC16F83 và PIC16CR83 có 512 từ bộ nhớ chương trình. PIC16F84 và PIC16CR84 có 1K từ nhớ chương trình, các lệnh CALL và GOTO có phạm vi địa chỉ là 11 bít, độ dài địa chỉ này cho phép một lệnh nhảy trong vòng một trang có độ lớn là 2K. Để mở rộng bộ nhớ chương trình của họ PIC16F8x trong tương lai phải dùng hai bít khác để xác định trang của bộ nhớ chương trình. Các bít phân trang này là các bít PCLATH<4:3> (hình 3.11). Khi thực hiện một lệnh CALL hoặc lệnh GOTO, người dùng phải chắc chắn rằng các bít phân trang đã được lập trình phù hợp với trang mong muốn. Nếu một lệnh CALL (hoặc ngắt) đã được thực hiện, toàn bộ 13 bít của PC sẽ được cất vào ngăn xếp. Do đó, việc sử dụng PCLATH<4:3> là không cần thiết cho các lệnh trở về.

Lưu ý:

PIC16F8x bỏ qua các bít PCLATH<4:3>, các bít này được dùng cho các trang 1, 2 và 3 của bộ đếm chương trình (0800h – 1FFFh), không nên dùng các bít phân trang như các bít R/W công dụng chung vì điều này sẽ làm ảnh hưởng đến sự tương thích với các sản phẩm trong tương lai.

4.4 Ngăn xếp

Họ PIC16Fxx có độ lồng sâu 8 cấp x 13 bít. Vùng ngăn xếp không phải là một phần của bộ nhớ chương trình hoặc của bộ nhớ dữ liệu và con trỏ ngăn xếp không cho phép đọc và ghi.

Toàn bộ 13 bít của PC được cất vào ngăn xếp khi một lệnh CALL được thực hiện hoặc một ngắt được chấp nhận. Ngăn xếp được lấy ra khi thực hiện lệnh RETURN, RETLW hoặc RETFIE. PCLATH không bị ảnh hưởng bởi thao tác cất và lấy ra từ ngăn xếp.

Lưu ý:

Không có lênh nào cất hoặc lấy ra từ ngăn xếp. Các thao tác này chỉ xảy ra khi thực hiện các lệnh CALL, RETURN, RETLW và RETFIE hoặc việc trỏ đến một địa chỉ ngắt.

Ngăn xếp hoạt động như một bộ đệm quay vòng. Có nghĩa là sau tám lần cất vào ngăn xếp thì giá trị được cất vào tại lần thú chín sẽ được ghi đè lên giá trị đã được cất vào đầu tiên, giá trị cất lần thứ mười sẽ đè lên giá trị đã cất vào lần thứ hai và tiếp tục như thê. Nếu ngăn xếp được lấy ra chín lần thì PC sẽ có giá trị giống như giá trị lấy ra lần đầu tiên.

Lưu ý:

Không có bít trạng thái nào báo cho biết hiện tượng tràn ngăn xếp

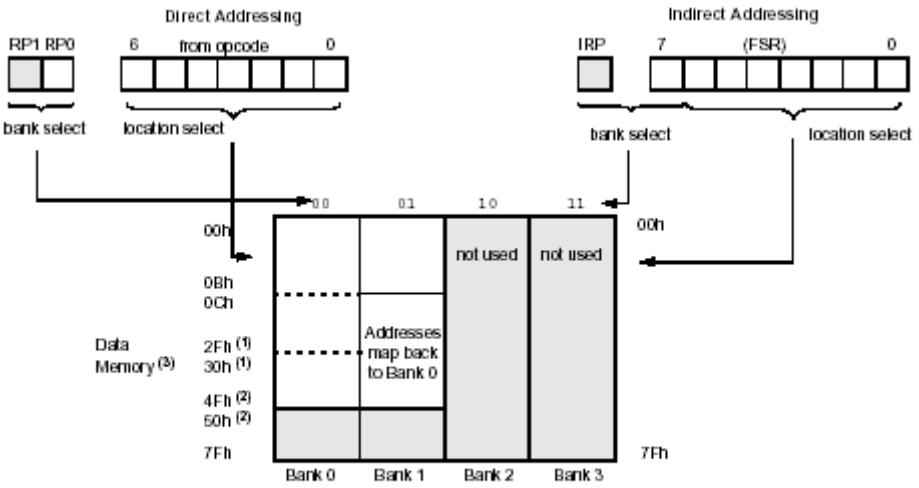
Định địa chỉ gián tiếp: Thanh ghi INDFF và FSR

Thanh ghi INDFF không phải là thanh ghi vật lý. Trong thực tế việc định địa chỉ INDF sẽ xác định địa chỉ của thanh ghi có địa chỉ được chứa trong thanh ghi FSR (FSR là một con trỏ). Đây là phương pháp định địa chỉ gián tiếp.

Ví dụ 4-1: Định địa chỉ gián tiếp

- Dãy thanh ghi 05 chứa giá trị 10h
- Dãy thanh ghi 06 chứa giá trị 0Ah
- Nạp giá trị 05 vào FSR
- Một lệnh đọc thanh ghi INDF sẽ trả về giá trị 10h
- Tăng thanh ghi FSR lên 1 (FSR = 06)
- Bây giờ lệnh đọc thanh ghi INDF sẽ trả về giá trị 0Ah

Quá trình tự đọc gián tiếp INDF sẽ cho kết quả là 00h. Việc ghi gián tiếp thanh ghi INDF sẽ không cho ra kết quả gì cả (mặc dù các bít trạng thái có thể bị ảnh hưởng)



Note 1: PIC16F83 and PIC16CR83 devices.
2: PIC16F84 and PIC16CR84 devices
3: For memory map detail see Figure 4-1.

Hình 3.12 Định địa chỉ trực tiếp/gián tiếp

Ví dụ 4-2 trình bày một chương trình đơn giản xóa vùng RAM 20h – 2Fh bằng cách định địa chỉ gián tiếp

Ví dụ 4-2:

```
                movlw 0x20    ; khởi tạo con trỏ
                movwf FSR     ; chỉ đến RAM
NEXT            clrf  INDF    ; xóa thanh ghi INDF
                incf  FSR     ; tăng con trỏ
                btfss FSR, 4  ; xong ?
                goto  NEXT    ; chưa, xóa tiếp
CONTINUE
```

Một địa chỉ hiệu dụng 9 bít nhận được bằng cách ghép 8 bít thanh ghi FSR với bít IRP (STATUS<7>) như chỉ trong hình 3.12. Tuy nhiên, IRP không được dùng trong PIC16F8x

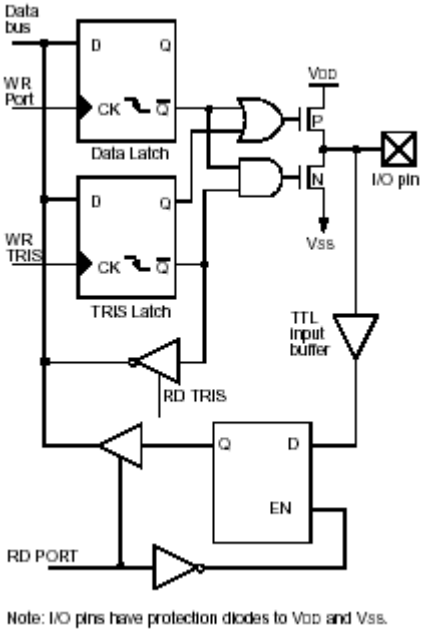
5. CỔNG GIAO TIẾP I/O

PIC16F8x có hai cổng: Port A và port B, một số chân của cổng là loại đa chức năng tương ứng với các chức năng khác của thiết bị.

5.1 Port A và thanh ghi TRISA

Port A là một chốt 5 bít, RA4 là một ngỏ vào schmitt trigger và cũng là một ngỏ ra cực thoát để hở. Tất cả các chân khác của port A là ngỏ vào TTL và là ngỏ ra CMOS. Tất cả các chân có các bít định hướng dữ liệu tương ứng (thanh ghi TRIS) để cấu hình vào hoặc ra cho các chân.

Việc thiết lập 1 cho một bít TRISA sẽ xác định chân tương ứng là vào có nghĩa là mạch điều khiển ra tại chân này được đưa lên trạng thái Z cao. Xóa một bít TRISA sẽ cấu hình chân tương ứng là ra có nghĩa là đưa nội dung của mạch chốt ngỏ ra lên chân này. Việc đọc thanh ghi port A sẽ đọc trạng thái tại các chân. Việc ghi vào chân sẽ ghi vào mạch chốt của port. Tất cả các thao tác ghi port đều là loại đọc – sửa – ghi. Nên việc ghi vào port có nghĩa là đầu tiên chân port được đọc vào sau đó sửa đổi và cuối cùng ghi lại vào chốt dữ liệu của port. Chân RA4 là chân đa hợp với ngỏ vào đồng hồ TMR0

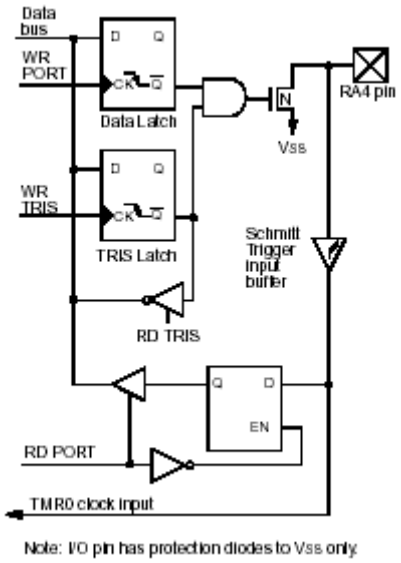


Note: I/O pins have protection diodes to VDD and VSS.

Hình 3.13 Sơ đồ khối các chân RA3:RA0

Ví dụ 5-1: Khởi tạo port A

```
Clrf    PORT A          ; Khởi tạo port A bằng cách thiết lập các chốt
                        ; dữ liệu ngỏ ra
Bsf     STATUS, RP0     ; Chọn dãy 1
Movlw   0x0F            ; Dữ liệu được dùng để chọn hướng dữ liệu
Movwf   TRISA           ; Thiết lập RA<3:0> là ngỏ vào
                        ; RA4 là ngỏ ra
                        ; Giá trị đọc của TRISA<7:5> luôn là “0”
```



Hình 3.14 Sơ đồ khối chân RA4

Tên	Bít	Kiểu đệm	Chức năng
RA0	Bít 0	TTL	Input/output
RA1	Bít 1	TTL	Input/output
RA2	Bít 2	TTL	Input/output
RA3	Bít 3	TTL	Input/output
RA4/T0CK1	Bít 4	ST	Input/output hoặc ngỏ vào xung đồng hồ của TMR0, ngỏ ra cực thoát để hở

Bảng 3.4 Chức năng port A

Địa chỉ	Tên	Bít 7	Bít 6	Bít 5	Bít 4	Bít 3	Bít 2	Bít 1	Bít 0	Giá trị khi mở nguồn	Giá trị khi reset
05h	**PORTA**	**-**	**-**	**-**	**RA4/T0CK1**	**RA3**	**RA2**	**RA1**	**RA0**	**---x xxxx**	**---u uuuu**
85h	**TRISA**	**-**	**-**	**-**	**TRISA4**	**TRISA3**	**TRISA2**	**TRISA1**	**TRISA0**	**---1 1111**	**---1 1111**

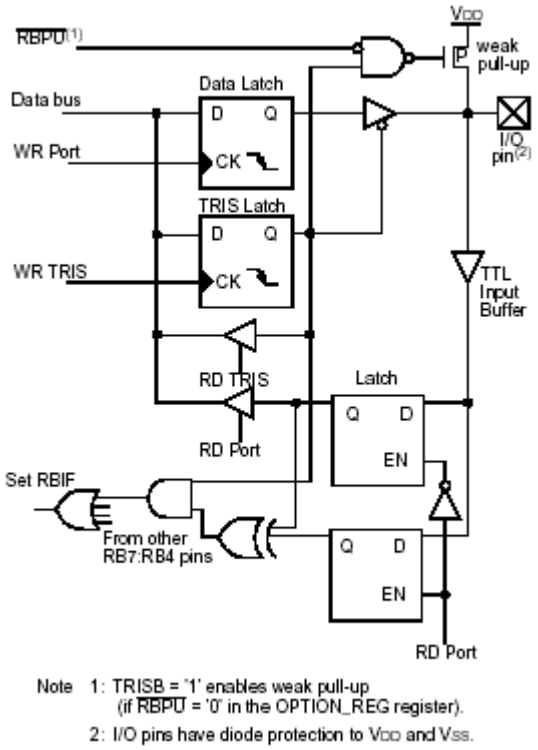
Bảng 3.5 Tóm tắt các thanh ghi của port A

5.2 Port B và các thanh ghi TRISB

Port B là port hai chiều 8 bít, các thanh ghi chọn hướng dữ liệu là TRISB. Bít nào trong thanh ghi TRISB bằng 1 thì mạch lái ra tương ứng sẽ có trạng thái Z cao. Khi một bít bất kỳ trong TRISB bằng 0, thì nội dung của chốt dữ liệu ra sẽ được đặt vào chân ra tương ứng với bít đó.

Mỗi chân ra của port B có điện trở kéo lên bên trong. Một bít điều khiển đơn có thể kích hoạt tất cả các điện trở kéo lên này, điều này được thực hiện bằng cách xóa bít $\overline{\text{RBPU}}$ (OPTION_REG<7>). Mạch điện trở kéo lên tự động ngắt khi chân port được cấu hình là chân ra, mạch điện trở kéo lên cũng mất tác dụng khi mở nguồn.

Bốn chân RB7:RB4 có tính năng thay đổi ngắt. Ngắt chỉ có thể xảy ra khi các chân này được cấu hình là ngỏ vào, có nghĩa là chân nào được cấu hình là ra thì ngắt bị chặn). Giá trị tại chân ở chế độ vào được so sánh với giá trị cũ đã được chốt tại lần đọc port B cuối cùng. Các ngỏ ra không phù hợp của các chân được OR lại với nhau để tạo biến thiên ngắt port RB.



Hình 3.15 Sơ đồ khối các chân RB7:RB4

Ngắt có thể đánh thức thiết bị khỏi trạng thái SLEEP. Người dùng có thể xóa ngắt theo các cách sau :

a) Đọc hoặc ghi port B, điều này sẽ chấm dứt giá trị biến thiên
b) Xóa cờ ngắt RBIF

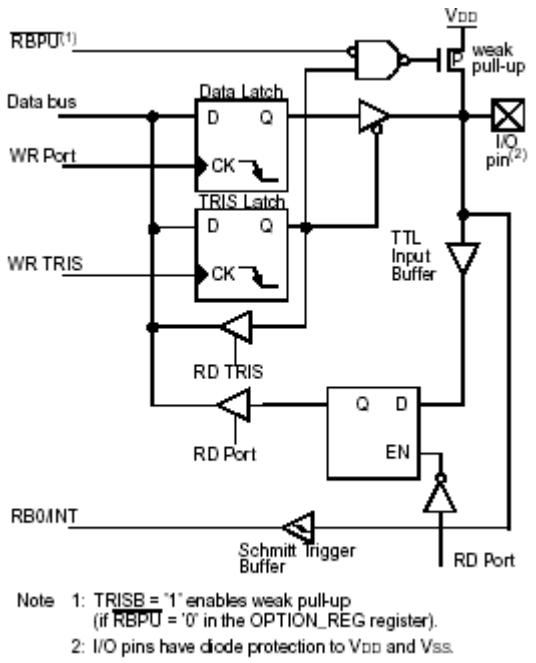
Một giá trị biến thiên sẽ tiếp tục đặt bít RBIF. Việc đọc port B sẽ kết thúc sự biến thiên này và cho phép xóa bít RBIF

Ngắt dựa trên đặc tính biến thiên (tác động cạnh) cùng với mạch kéo lên cấu hình theo phần mềm cho phép dễ dàng giao tiếp với bàn phím để thực hiện việc đánh thức thiết bị bằng cách ấn phím.

Lưu ý:

Để có thể nhận ra sự thay đổi ở chân I/O, bề rộng xung tối thiểu phải bằng Tcy (fOSC/4)

Ngắt tác động cạnh nên được dùng để đánh thức thiết bị bằng phím nhấn và trong các thao tác mà ở đó port B chỉ được dùng cho ngắt tác động cạnh. Không nên dùng chuỗi vòng của port B cho ngắt tác động cạnh.



Hình 3.16 Sơ đồ khối các chân RB3:RB0

Ví dụ 5-1: Khởi tạo port B

```
clrf    PORTB           ; Khởi tạo port B bằng cách đặt các chốt dữ liệu ra
bsf     STATUS, RP0     ; Chọn dãy 1
movlw   0xCF            ; Giá trị chọn hướng dữ liệu
movwf   TRISB           ; RB<3:0> là vào
                        ; RB<5:4> là ra         ; RB<7:6> là vào
```

Tên	Bít	Kiểu đệm	Chức năng
RB0/INT	Bits 0	TTL/ST (1)	Chân I/O hoặc ngỏ vào ngắt ngoài, mạch kéo lên bên trong lập trình bằng phần mềm
RB1	Bits 1	TTL	Chân I/O, mạch kéo lên bên trong lập trình bằng phần mềm
RB2	Bits 2	TTL	Chân I/O, mạch kéo lên bên trong lập trình bằng phần mềm
RB3	Bits 3	TTL	Chân I/O, mạch kéo lên bên trong lập trình bằng phần mềm
RB4	Bits 4	TTL	Chân I/O (ngắt cạnh), mạch kéo lên bên trong lập trình bằng phần mềm
RB5	Bits 5	TTL	Chân I/O (ngắt cạnh), mạch kéo lên bên trong lập trình bằng phần mềm
RB6	Bits 6	TTL/ST (2)	Chân I/O (ngắt cạnh), mạch kéo lên bên trong lập trình bằng phần mềm, ngỏ vào đồng hồ nạp nối tiếp
RB7	Bits 7	TTL/ST (2)	Chân I/O (ngắt cạnh), mạch kéo lên bên trong lập trình bằng phần mềm, dữ liệu nạp nối tiếp

Bảng 3.6 Chức năng port B

[1] Là ngỏ vào schmitt trigger khi được cấu hình là ngỏ vào ngắt
[2] Là ngỏ vào schmitt trigger ở chế độ nạp chương trình nối tiếp

Địa chỉ	Tên	Bít 7	Bít 6	Bít 5	Bít 4	Bít 3	Bít 2	Bít 1	Bít 0	Giá trị khi mở máy	Giá trị khi reset
06h	**PORTB**	**RB7**	**RB6**	**RB5**	**RB4**	**RB3**	**RB2**	**RB1**	**RB0/INT**	**xxxx xxxx**	**uuuu uuuu**
86h	**TRISB**	**TRISB7**	**TRISB6**	**TRISB5**	**TRISB4**	**TRISB3**	**TRISB2**	**TRISB 1**	**TRISB0**	**1111 1111**	**1111 1111**
81h	**OPTION _REG**	$\overline{\text{RBPU}}$	**INTEDG**	**T0CS**	**T0SE**	**PSA**	**PS2**	**PS1**	**PS0**	**1111 1111**	**1111 1111**

Bảng 3.7 Tóm tắt các thanh ghi port B

5.3 Lập trình I/O

5.3.1 Port I/O hai chiều

Các lệnh ghi bất kỳ, diễn tiến bên trong như một thao tác đọc được theo sau bởi thao tác ghi. Ví dụ các lệnh BCF và BSF sẽ đọc thanh ghi vào CPU, thực hiện phép tính bít và sau đó ghi kết quả trở lại vào thanh ghi. Cần phải chú ý khi các lệnh này được áp dụng cho port đã được định nghĩa theo cả hai hướng vào và ra. Ví dụ lệnh BSF trên bít 5 của port B sẽ đọc 8 bít của port B vào CPU, sau đó lệnh BSF xảy ra trên bít 5 và port B được ghi vào chốt dữ liệu ra. Nếu một bít khác của port B được dùng như một chân I/O hai chiều (VD: Bít 0) và cùng lúc đó đã được định nghĩa là ngỏ vào thì tín hiệu vào tại chân này tự nó được đọc vào CPU và ghi lại vào chốt dữ liệu của chân đặc biệt này bằng cách ghi đè nội dung trước đó. Cho đến khi chân này vẫn còn ở chế độ vào thì không vấn đề gì xảy ra. Tuy nhiên, nếu sau đó bit0 được chuyển sang chế độ ra thì nội dung của chốt dữ liệu sẽ không xác định

Thao tác đọc thanh ghi port sẽ đọc giá trị tại các chân port. Thao tác ghi vào thanh ghi port sẽ ghi giá trị vào chốt của port. Khi dùng các lệnh đọc-sửa-ghi như BCF, BSF… thì giá trị tại chân port được đọc, xử lý giá trị này và sau đó ghi trở lại vào chốt port.
Một chân ra đang tích cực ở mức thấp hoặc cao không nên được điều khiển đồng thời bởi thiết bị ngoài để tránh sự thay đổi mức logic tại chân này (OR nối dây hoặc AND nối dây) dẫn đến kết quả là dòng ra cao có thể làm hư vi mạch

5.3.2 Thao tác liên tục trên port I/O

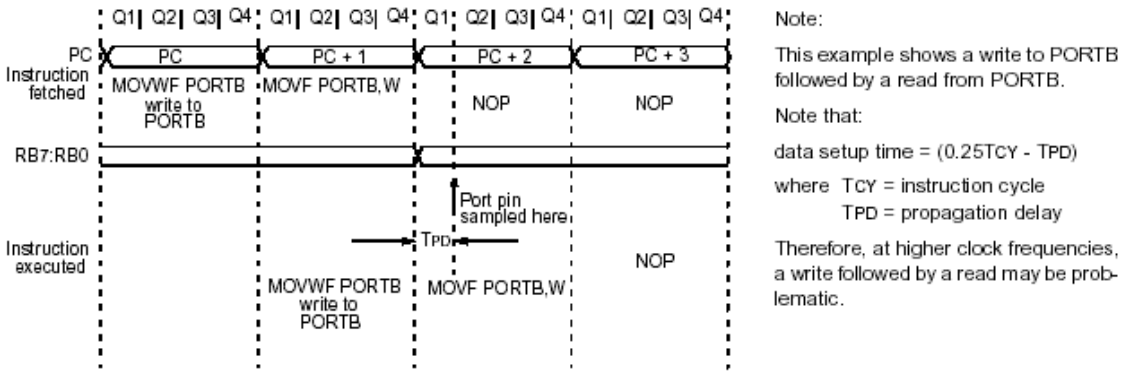
Trên thực tế việc ghi vào một port I/O xảy tại thời điểm cuối của chu kỳ lệnh, để đọc thì dữ liệu phải xác lập tại lúc bắt đầu chu kỳ lệnh (hình 3.17). Do đó, cần phải cẩn thận khi một thao tác đọc theo sau một thao tác ghi trên cùng một port. Các lệnh phải tuần tự sao cho điện áp chân ổn định (tùy theo tải) trước khi thực hiện lệnh tiếp theo đọc dữ liệu vào CPU. Nếu không trạng thái trước đó tại chân port sẽ được đọc vào CPU thay vì là trạng thái mới. Nếu không chắc chắn thì cách tốt nhất là nên cách ly các lệnh này bằng một lệnh NOP hoặc một lệnh khác không truy cập đến port I/O này.

Ví dụ 5-2 cho thấy ảnh hưởng của hai lệnh đọc-sửa-ghi BCF, BSF trên một port I/O

Ví dụ 5-2: Các lệnh đọc-sửa-ghi trên một port I/O

```
; Khởi tạo port PORTB<7:4> vào
;                PORTB<3:0> ra
; PORTB<7:6> có điện trở kéo lên bên ngoài và không được nối với mạch khác
;
;                          Chốt port      Chân port
;                          ------------   -------------
        bcf     PORTB, 7   ; 01pp ppp     11pp ppp
        bcf     PORTB, 6   ; 10pp ppp     11pp ppp
        bsf     STATUS, RP0;
        bcf     TRISB, 7   ; 10pp ppp     11pp ppp
        bcf     TRISB, 6   ; 10pp ppp     10pp ppp
```

Lưu ý là người dùng có thể đã nghỉ rằng giá trị các chân là 00pp ppp. Lệnh bcf thứ nhì làm cho RB7 bị chốt tại giá trị chân (high)



Hình 3.17 Thao tác I/O liên tục

6. BỘ ĐỊNH THỜI TIMER0 VÀ THANH GHI TMR0

Bộ định thời timer0 có những tính năng sau :

- Là timer/counter 8 bít
- Cho phép đọc/ghi
- Thang chia 8 bít lập trình bằng phần mềm
- Cho phép chọn xung đồng hồ trong hoặc ngoài
- Tạo ngắt khi có tràn từ FFh đến 00h
- Chọn tác động cạnh đối với xung đồng hồ ngoài

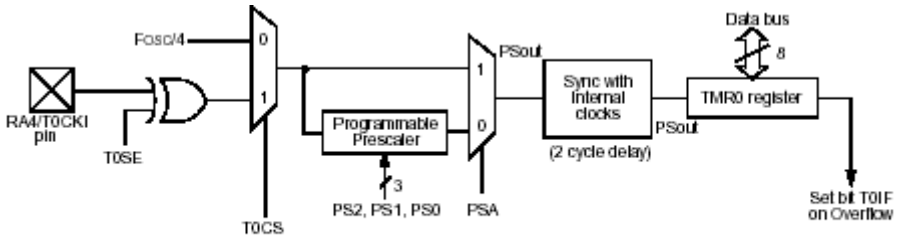
Chế độ timer được chọn bằng cách xóa bít T0CS (OPTION_REG<5>). Trong chế độ định thời, mô đun timer0 (hình 3.18) sẽ tăng theo mỗi chu kỳ lệnh (không có định thang). Nếu thanh ghi TMR0 được ghi, quá trình tăng sẽ bị cấm sau hai chu kỳ (hình 3.19 và hình 3.20). Người dùng có thể thực hiện việc này bằng cách ghi giá trị đã đặt trước vào thanh ghi TMR0.

Chế độ đếm được chọn bằng cách đặt bít T0CS. Trong chế độ này TMR0 sẽ tăng theo mỗi cạnh lên hoặc mỗi cạnh xuống tại chân RA4/T0CK1. Việc tăng theo cạnh được xác định bởi bít chọn cạnh T0SE (OPTION_REG<4>). Xóa bít T0SE khi muốn chọn cạnh lên.

Thang chia được chia xẻ giữa bộ định thời timer0 với bộ định thời canh chừng (watchdog timer). Việc định thang chia được điều khiển bằng phần mềm bằng cách điều khiển bít PSA (OPTION_REG<3>). Bít PSA bị xóa sẽ gán thang chia cho timer0. Thang chia không cho phép đọc/viết. Khi thanh chia được gán cho timer0, các tỉ số chia (1:2, 1:4, …,1:256) có thể chọn được bằng phần mềm.

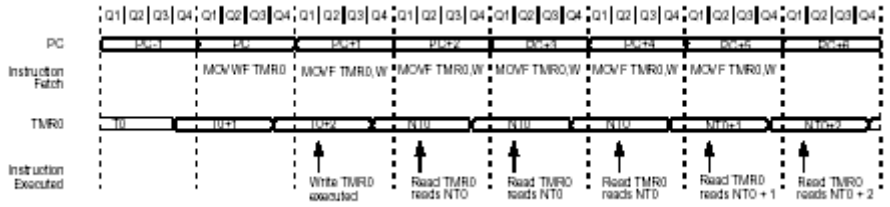
6.1 Ngắt TMR0

Ngắt TMR0 được tạo ra khi thanh ghi TMR0 tràn từ FFh đến 00h. Hiện tượng tràn này sẽ đặt bít T0IF (INTCON<2>). Ngắt có thể bị ngăn bằng cách xóa bít cho phép T0IE (INTCON<5>). Bít T0IF phải bị xóa bằng phần mềm thông qua chương trình phục vụ ngắt timer0 trước khi ngắt này được cho phép trở lại. Ngắt TMR0 (hình 3.21) không thể đánh thức vi xử lý khỏi trạng thái SLEEP vì timer đã ngừng hoạt động trong thời gian SLEEP.

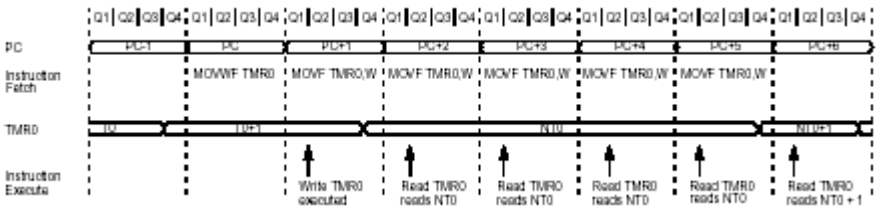


Note 1: Bits T0CS, T0SE, PS2, PS1, PS0 and PSA are located in the OPTION_REG register.
2: The prescaler is shared with the Watchdog Timer (Figure 6-6)

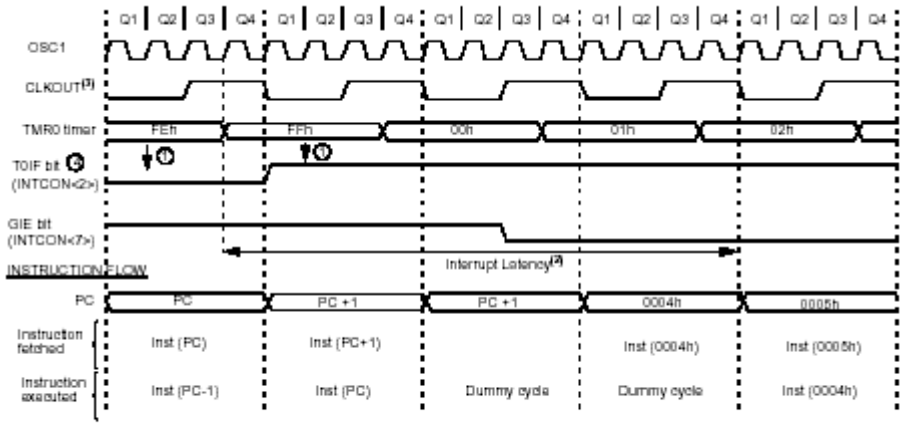
Hình 3.18 Sơ đồ khối bộ định thời



Hình 3.19 Đồ thị thời gian timer0 đồng hồ bên trong/ không định thang chia



Hình 3.20 Đồ thị thời gian Timer0 đồng hồ trong/thang chia 1:2



Note 1: T0IF interrupt flag is sampled here (every Q1).
2: Interrupt latency = 3.25Tcy, where Tcy = instruction cycle time.
3: CLKOUT is available only in RC oscillator mode.
4: The timer clock (after the synchronizer circuit) which increments the timer from FFh to 00h immediately sets the T0IF bit.
The TMR0 register will roll over 3 Tosc cycles later.

Hình 3.21 Đồ thị thời gian ngắt TMR0

6.2 Timer0 với xung đồng hồ bên ngoài

Khi một ngỏ vào đồng hồ bên ngoài được áp dụng cho Timer0 thì phải thỏa mãn một số yêu cầu nhất định. Xung đồng hồ ngoài phải đồng pha với xung đồng hồ bên trong (T_{OSC}). Cũng có một độ trễ trong lượng tăng thực tế của TMR0 sau khi đồng bộ.

6.2.1 Đồng bộ xung đồng hồ ngoài

Khi không dùng thang chia, ngỏ vào đồng hồ ngoài có cùng thang chia như ngỏ ra. Việc đồng bộ chân RA4/T0CK1 với pha đồng hồ bên trong bằng cách lấy mẫu thang chia ngỏ ra theo các chu kỳ Q2 và Q4 của pha đồng hồ bên trong (hình 3.23). Do đó, T0CK1 cần thiết phải cao hơn ít nhất $2T_{OSC}$ (cộng thêm thời gian trể bằng mạch RC) và thấp xuống ít nhất $2T_{OSC}$.

Khi sử dụng thang chia, xung đồng hồ bên ngoài được chia bởi một bộ đếm không đồng bộ có hệ số chia sao cho thang chia ngỏ ra đồng bộ. Xung đồng hồ ngoái cần phải được lấy mẫu và phải tính hệ số chia bộ đếm không đồng bộ. Do đó, T0CK1 cần thiết phải có chu kỳ ít nhất $4T_{OSC}$ (thêm mạch trể RC) được chia bởi giá trị thang chia, yêu cầu đối với thời gian cao và thấp của T0CK1 là chúng không được nhỏ hơn bề rộng xung tối thiểu là 10 nS.

6.2.2 Tăng độ trể TMR0

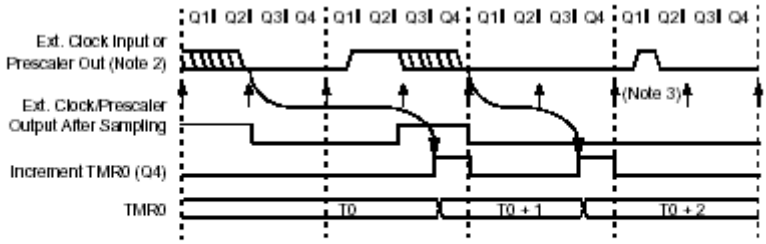
Kể từ khi thang chia ngỏ ra được đồng bộ với đồng hồ bên trong, có một độ trể nhỏ từ thời điểm xuất hiện cạnh xung đồng hồ bên ngoài đến thời điểm bộ định thời Timer0 thực sự tăng lên. Hình 3.22 trình bày độ trể này

6.3 Bộ định thang chia

Một bộ đếm 8 bít có thể dùng để định thang chia cho Timer0 hoặc cho bộ định thời canh chừng (hình 3.23) . Lưu ý là chỉ có một bộ định thang chia duy nhất cho cả hai bộ Timer0 và bộ định thời watchdog. Vì vậy nếu bộ định thang gán cho Timer0 thf sẽ không hiệu lực với watchdog và ngược lại.

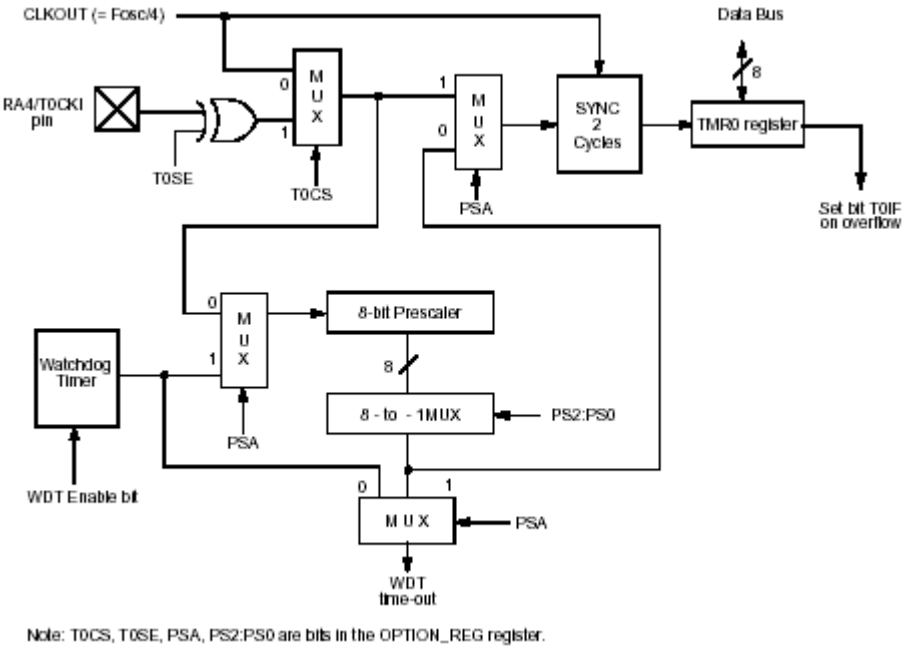
Các bít PSA và PS2:PS0 (OPTION_REG<3:0>) xác định nhiệm vụ của bộ định thang và giá trị thang chia.

Khi đã được gán cho Timer0, tất cả các lệnh ghi vào Timer0 như CLRF 1, MOVWF 1, BSF 1,….sẽ xóa bộ định thang. Khi đã được gán cho WDT, lệnh CLRWDT sẽ xóa bộ định thang cùng với WDT. Bộ định thang không cho phép đọc/ghi.



Note 1: Delay from clock input change to TMR0 increment is 3Tosc to 7Tosc. (Duration of Q = Tosc).
Therefore, the error in measuring the interval between two edges on TMR0 input = ± 4Tosc max.
2: External clock if no prescaler selected, Prescaler output otherwise.
3: The arrows ↑ indicate where sampling occurs. A small clock pulse may be missed by sampling.

Hình 3.22 Đồ thị thời gian Timer0 với đồng hồ ngoài



Hình 3.23 Sơ đồ khối bộ định thang TMR0/WDT

Hoán chuyển nhiệm vụ bộ định thang

Nhiệm vụ bộ định thang được điều khiển hoàn toàn bằng phần mềm, có nghĩa là nó có thể được thay đổi trong khi đang thực hiện chương trình.

Lưu ý:

Để tránh hiện tượng reset thiết bị ngoài ý muốn cần phải thực hiện đoạn lệnh sau đây (ví dụ 6-1) khi chuyển nhiệm vụ bộ định thang từ Timer0 sang WDT ngay cả khi WDT không được cho phép. Để chuyển bộ định thang từ WDT sang Timer0 phải dùng đoạn mã lệnh trình bày ở ví dụ 6-2

Ví dụ 6-1: Chuyển bộ định thang (Timer0 → WDT)

```
BCF    STATUS, RP0 ; Dãy 0
CLRF   TMR0        ; Xóa TMR0 và bộ định thang
BSF    STATUS, RP0 ; Dãy 1
CLRWDT             ; Xóa WDT
MOVLW      b'xxxx1xxx'    ; Chọn giá trị
MOVWF      OPTION_REG     ; bộ định thang mới
BCF    STATUS, RP0 ; Dãy 0
```

Ví dụ 6-2: Chuyển bộ định thang (WDT → Timer0)

```
CLRWDT             ; Xóa WDT và bộ định thang
BSF    STATUS, RP0 ; Dãy 1
MOVLW      b'xxxx0xxx'    ; Chọn TMR0, giá trị định thang mới và
                   ; nguồn xung đồng hồ
MOVWF      OPTION_REG     ;
BCF    STATUS, RP0 ; Dãy 0
```

Địa chỉ	Tên	Bít 7	Bít 6	Bít 5	Bít 4	Bít 3	Bít 2	Bít 1	Bít 0	Giá trị mở máy	Giá trị reset
01h	**TMR0**	**Thanh ghi Timer0**								**xxxx xxxx**	**uuuu uuuu**
0Bh	**INTCON**	**GIE**	**EEIE**	**T0IE**	**INTE**	**RBIE**	**T0IF**	**ÌNT**	**RBIF**	**0000 000x**	**0000 0000**
81h	**OPTION_REG**	$\overline{\text{RBPU}}$	**INTEDG**	**T0CS**	**T0SE**	**PSA**	**PS2**	**PS1**	**PS0**	**1111 1111**	**1111 1111**
85h	**TRISA**	**-**	**-**	**-**	**TRISA4**	**TRISA3**	**TRISA2**	**TRISA1**	**TRISA0**	**---1 1111**	**---1 1111**

Bảng 3.8 Các thanh ghi của Timer0

7. BỘ NHỚ DỮ LIỆU EEPROM

Bộ nhớ dữ liệu EEPROM cho phép đọc/ghi trong khi hoạt động bình thường (toàn bộ dãi điện áp VDD). Bộ nhớ này không được ánh xạ trực tiếp lên vùng không gian thanh ghi. Thay vào đó nó được định địa chỉ gián tiếp thông qua các thanh ghi chức năng đặc biệt. Có bốn thanh ghi chức năng đặc biệt được dùng để đọc/ghi bộ nhớ này, đó là :

- EECON1
- EECON2
- EEDATA
- EEADR

EEDATA lưu giữ 8 bít dữ liệu đọc/ghi và EEADR lưu giữ địa chỉ của vị trí EEPROM đang được truy cập. Các thiết bị PIC16F8x có 64 byte dữ liệu EEPROM chiếm địa chỉ từ 0h đến 3Fh.

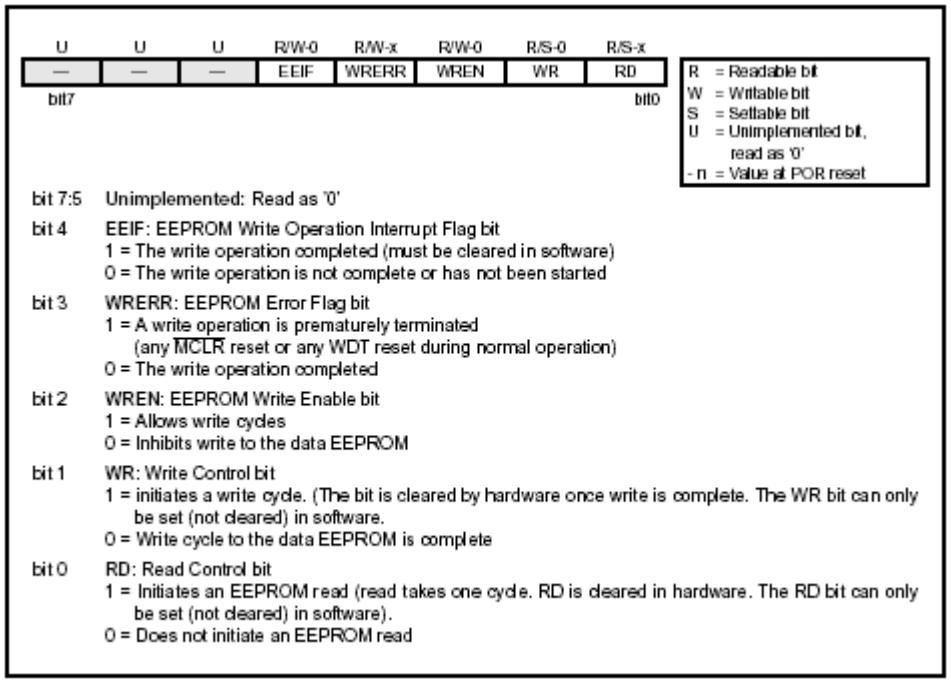
Bộ nhớ dữ liệu EEPROM cho phép ghi/đọc theo byte. Khi ghi vị trí cần ghi sẽ bị xóa trước khi ghi byte mới. Bộ nhớ dữ liệu EEPROM được xem như có chu kỳ đọc/ghi cao, thời gian ghi được điều khiển bởi một bộ định thời tích hợp trên chip, thời gian ghi thay đổi theo điện áp và nhiệt độ cũng như theo chip. Tham khảo phần đặc tính AC để biết chính xác hơn.

Khi một thiết bị được mã hóa bảo vệ, CPU có thể tiếp tục đọc và ghi bộ nhớ EEPROM. Máy nạp chương chương trình có thể không còn truy cập được bộ nhớ này.

7.1 EEADR

Thanh ghi EEADR có thể định địa chỉ tối đa đến 256 byte bộ nhớ EEPROM. Chỉ có 64 byte đầu tiên của bộ nhớ được cung cấp.

Hai bít ở phần trên được giải mã địa chỉ. Có nghĩa là hai bít này phải luôn bằng 0 để chắc chắn rằng địa chỉ nằm trong vùng 64 byte



Hình 3.24 Thanh ghi EECON1 (địa chỉ 88h)

7.2 Thanh ghi EECON1 và EECON2

EECON1 là thanh ghi điều khiển có năm bít vật lý được cung cấp ở phần thấp, ba bít cao không tồn tại và có giá trị '0' khi đọc.

Các bít điều khiển RD và WR khởi tạo quá trình đọc và ghi. Những bít này không cho phép xóa, chỉ được đặt bằng phần mềm. Chúng bị xóa bằng phần cứng sau khi hoàn tất thao tác đọc hoặc ghi. Việc không cho phép xóa bít WR bằng phần mềm nhằm tránh hiện tượng kết thúc ngẫu nhiên quá trình ghi.

Bít WREN khi được đặt sẽ cho phép thao tác ghi. Bít này bị xóa khi mở máy. Bít WRERR được đặt khi quá trình ghi bị dừng bởi $\overline{\text{MCLR}}$ hoặc do bộ định thời WDT trong hoạt động bình thường. Trong những trường hợp này, sau khi reset, người dùng có thể kiểm tra bít WRERR và thực hiện việc ghi lại. Dữ liệu và địa chỉ sẽ không bị thay đổi trong các thanh ghi EEDATA và EEADR.

Bít cờ ngắt EEIF được đặt khi hoàn tất thao tác ghi. Nó phải được xóa bằng phần mềm.

EECON2 không phải là một thanh ghi vật lý, khi đọc EECON2 sẽ cho kết quả là '0'. Thanh ghi EECON2 được dùng để ghi tuần tự vào bộ nhớ dữ liệu EEPROM

7.3 Đọc bộ nhớ dữ liệu EEPROM

Để đọc một ô nhớ dữ liệu, người dùng phải ghi địa chỉ vào thanh ghi EEADR và sau đó đặt bít điều khiển RD (EECON1<0>), dữ liệu trong thanh ghi EEDATA có thể dùng trong nhiều chu kỳ kế tiếp. Do đó, nó có thể được đọc bởi lệnh tiếp theo. EEDATA sẽ duy trì giá trị cho đến lệnh đọc khác hoặc cho đến khi được ghi vào bởi người dùng.

Ví dụ 7-1: Đọc dữ liệu EEPROM

```
BCF    STATUS, RP0; Dãy 0
MOVLW        CONFIG_ADDR       ;
MOVWF        EEADR             ; Địa chỉ cần đọc
BSF    STATUS, RP0; Dãy 1
BSF    EECON1, RD  ; Đọc EE
BCF    STATUS, RP0; Dãy 0
MOVF  EEDATA, W    ; W = EDATA
```

7.4 Ghi vào bộ nhớ dữ liệu EEPROM

Để ghi một vị trí dữ liệu trong EEPROM. Trước tiên người dùng phải đưa địa chỉ vào thanh ghi EEADR và dữ liệu vào thanh ghi EEDATA sau đó phải áp dụng đoạn mã lênh sau đây để khởi động ghi cho mỗi byte.

Ví dụ 7-2: Ghi bộ nhớ dữ liệu EEPROM

```
BSF     STATUS, RP0 ; Dãy 1
BCF     INTCON, GIE  ; Cấm các ngắt
BSF     EECON1, WREN        ; Cho phép ghi
MOVLW         55h           ;
```

Đoạn mã lệnh cần thiết

```
        MOVWF     EECON2        ; Ghi 55h
        MOVLW     AAh           ;
        MOVWF     EECON2        ; Ghi AAh
        BSFEECON1, WR ; Đặt bít WR, bắt đầu ghi

        BSFINTCON, GIE  ; Cho phép các ngắt
```

Quá trình ghi sẽ không khởi động nếu đoạn mã lệnh trên không được viết tiếp theo sau một cách chính xác (ghi 55h vào EECON2, ghi AAh vào EECON2 sau đó đặt bít WR) cho mỗi byte. Nên cấm các ngắt trong đoạn mã lệnh này.

Thêm vào đó, bít WREN trong EECON1 phải được đặt để cho phép ghi. Kỹ thuật này ngăn hiện tượng ghi ngẫu nhiên vào EEPROM (không mong muốn) do việc thực hiện sai mã lệnh (mất chương trình) nên duy trì trạng thái xóa cho bít WREN ở mọi thời điểm trừ khi cần cập nhật EEPROM. Bít WREN không bị xóa bằng phần cứng.

Sau khi một chuỗi ghi tuần tự được khởi động việc xóa bít WREN không ảnh hưởng đến chu kỳ ghi. Bít WR bị cấm khi đang đặt trừ phi bít WREN được đặt.

Khi hoàn tất chu kỳ ghi, bít WR bị xóa bởi phần cứng và bít cờ ngắt hoàn tất ghi (EEIF) được đặt, người dùng có thể hoặc cho phép hoặc thăm dò bít này. EEIF phải được xóa bằng phần mềm.

7.5 Kiểm tra kết quả ghi

Tùy theo ứng dụng, trong thực tế một chương trình gọi là tốt thì phải có khả năng kiểm tra lại kết quả đã ghi so với dữ liệu cần ghi vào EEPROM. Điều này nên áp dụng cho những ứng dụng có bít EEPROM gần với mức giới hạn cho phép.

Nói chung một bít khi vào EEPROM bị sai có nghĩa là khi ghi giá trị là ‘1’ nhưng đọc lại giá trị là ‘0’ (do mất bít).

Ví dụ 7-3: Kiểm tra ghi

```
        BCF  STATÚ, RP0  ; Dãy 0
        ;
        ;
        MOVF        EEDATA, w    ; Phải nằm trong dãy 0
        BSF  STATUS, RP0; Dãy 1

Read
       BSF    EECON1, RD  ; Đọc giá trị đã ghi
       BCF    STATUS, RP0; Dãy 0
                                 ;
        SUBWF       EEDATA, w    ; Có phải giá trị đã ghi trong thanh ghi w và giá trị
đọc
                                 ; trong EEDATA bằng nhau ?
       BTFSS, z              ; Hiệu số bằng 0 ?
```

```
        GOTO WRITE_ERR  ; Không, ghi sai
                        ; Đúng, ghi đúng
        ;               ; Tiếp tục chương trình
```

7.6 Bảo vệ chống ghi

Có nhiều trường hợp không muốn ghi vào EEPROM để chống giả mạo. Có nhiều kỹ thuật khác nhau được cài đặt sẵn trong thiết bị. Khi mở máy WREN bị xóa. Bộ định thời mở máy (trong khoảng 72 mS) cũng ngăn việc ghi vào EEPROM.
Chuỗi khởi động ghi và bít WREN kết hợp lẫn nhau để ngăn việc ghi ngẫu nhiên do nhiễu nguồn hoặc phần mềm hoạt động sai

7.7 Hoạt động của EEPROM trong trường hợp bảo vệ mã lệnh

Khi thiết bị được bảo vệ mã lệnh. CPU có thể đọc và ghi dữ liệu đã phục hồi vào EEPROM.

Đối với các thiết bị ROM có hai bít bảo vệ mã lệnh. Một cho bộ nhớ chương trình ROM và một cho bộ nhớ dữ liệu EEPROM

Địa chỉ	Tên	Bít 7	Bít 6	Bít 5	Bít 4	Bít 3	Bít 2	Bít 1	Bít 0	Giá trị khi mở máy	Giá trị khi reset
08h	**EEDATA**	**Thanh ghi dữ liệu EEPROM**								**xxxx xxxx**	**uuuu uuuu**
09h	**EEADR**	**Thanh ghi địa chỉ EEPROM**								**xxxx xxxx**	**uuuu uuuu**
88h	**EECON1**	**-**	**-**	**-**	**EEIF**	**WRERR**	**WREN**	**WR**	**RD**	**---0 x000**	**---0 q000**
89h	**EECON2**	**Thanh ghi điều khiển EEPROM**								**---- ----**	**---- ----**

Bảng 3.9 Các thanh ghi EEPROM

8. TÍNH NĂNG ĐẶC BIỆT CỦA CPU

Những điểm khác nhau giữa một vi điều khiển với các vi xử lý khác là các mạch đặc biệt cho các ứng dụng theo thời gian thực. PIC16F8x có một loạt các tính năng như thế nhằm tăng tối đa độ tin cậy của hệ thống, tối thiểu hóa giá thành thông qua việc giảm các linh kiện phụ bên ngoài, cung cấp các chế độ hoạt động tiết kiệm năng lượng và tính năng bảo vệ mã lệnh. Có thể kể ra như sau :

- Khả năng chọn OSC
- Reset
  - Khi mở máy (POR)
  - Bộ định thời tăng nguồn (PWRT)
  - Bộ định thời khởi động dao động (OST)
- Các ngắt
- Bộ định thời canh chừng (WDT)
- Chế độ ngũ
- Bảo vệ mã lệnh
- Vùng ID
- Lập trình trong hệ thống

PIC16F8x có một bộ định thời canh chừng nó chỉ có thể tắt bởi các bít cấu hình. Bộ định thời này hoạt động với mạch dao động RC riên nhằm tăng độ tin cậy. Có hai bộ định thời tạo ra các thời gian trể cần thiết khi tăng nguồn. Một là bộ định thời khởi động dao động (Oscilator Start-up Timer) nhằm duy trì chip vẫn ở trong trạng thái reset cho đến khi dao động thạch anh ổn định. Thứ hai là bộ định thời tăng nguồn (Power-up Timer). Bộ định thời này có thời gian trể cố định là 72 mS (danh định) khi tăng nguồn để giủ chip ở trạng thái reset cho đến khi nguồn nuôi ổn định. Với hai timer đượoc tích hợp này hầu như tất cả các ứng dụng đều không cần đến mạch reset bên ngoài

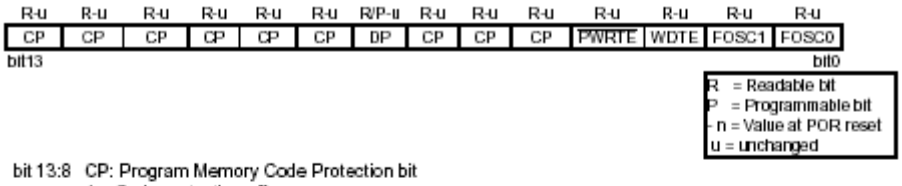
Chế độ ngũ cung cấp một trạng thái hạ nguồn với dòng điện nuôi rất thấp. Người dùng có thể đánh thức khỏi chế độ này bằng reset bên ngoài, WDT hoặc một tín hiệu

ngắt. Một vài tùy chọn dao động được cung cấp để cho phép thích ứng với ứng dụng. Dao động RC giảm giá thành hệ thống trong khi dao động LP lại giảm công suất. Các tùy chon được chọn bởi một tập hợp các bít cấu hình.

8.1 Các bít cấu hình

Các bít cấu hình có thể được lập trình (đọc bằng '0') hoặc không lập trinh (đọc bằng '1') để chọn cấu hình cho thiết bị, các bít này được ánh xạ tại vị trí 2007h trong bộ nhớ chương trình

Địa chỉ 2007h nằm ngoài không gian bộ nhớ chương trình của người dùng và ở trong vùng kiểm tra/cấu hình đặc biệt (2000h – 3FFFh). Vùng này chỉ có thể được truy cập trong khi nạp chương trình.



bit 13:8 CP: Program Memory Code Protection bit
1 = Code protection off
0 = Program memory is code protected

bit 7 DP: Data Memory Code Protection bit
1 = Code protection off
0 = Data memory is code protected

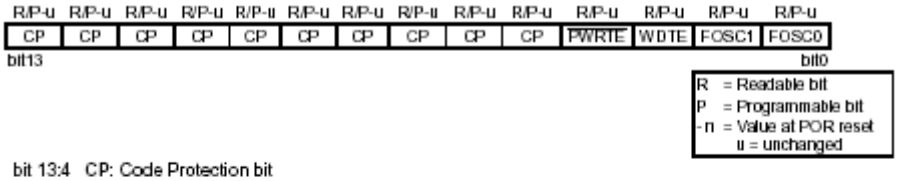
bit 6:4 CP: Program Memory Code Protection bit
1 = Code protection off
0 = Program memory is code protected

bit 3 $\overline{\text{PWRTE}}$: Power-up Timer Enable bit
1 = Power-up timer is disabled
0 = Power-up timer is enabled

bit 2 WDTE: Watchdog Timer Enable bit
1 = WDT enabled
0 = WDT disabled

bit 1:0 FOSC1:FOSC0: Oscillator Selection bits
11 = RC oscillator
10 = HS oscillator
01 = XT oscillator
00 = LP oscillator

Hình 3.25 Từ cấu hình PIC16CR83 và PIC16CR84



bit 13:4 CP: Code Protection bit
1 = Code protection off
0 = All memory is code protected

bit 3 $\overline{\text{PWRTE}}$: Power-up Timer Enable bit
1 = Power-up timer is disabled
0 = Power-up timer is enabled

bit 2 WDTE: Watchdog Timer Enable bit
1 = WDT enabled
0 = WDT disabled

bit 1:0 FOSC1:FOSC0: Oscillator Selection bits
11 = RC oscillator
10 = HS oscillator
01 = XT oscillator
00 = LP oscillator

Hình 3.26 Từ cấu hình PIC16F83 và PIC16F84

8.2 Cấu hình dao động

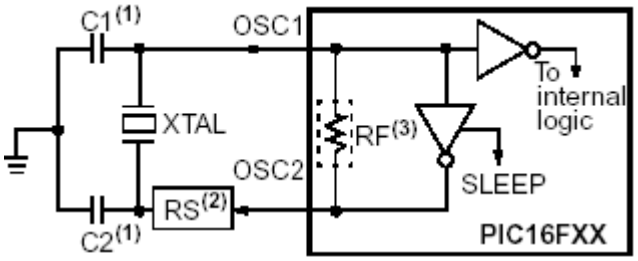
8.2.1 Các kiểu dao động

PIC16F8x có thể hoạt động ở bốn chế độ dao động :

- LP Low power crystal
- XT Crystal/Resonator
- HS High speed Crystal/Resonator
- RC Resistor/Capacitor

8.2.2 Dao động thạch anh/Cộng hưởng gốm

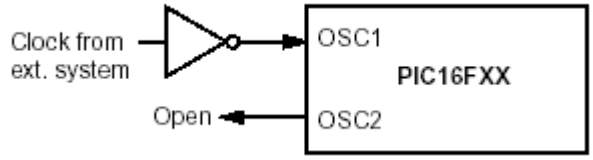
Các chế độ XT, LP và HS cần dùng một thạch anh hoặc một mạch cộng hưởng gốm nối vào các chân OSC1/CLKIN và OSC2/CLKOUT để ổn định dao động (hình 3.27)



Hình 3.27 Cấu hình dao động XT. LP và HS

Note1: See Table 8-1 for recommended values of C1 and C2.
2: A series resistor (RS) may be required for AT strip cut crystals.
3: RF varies with the crystal chosen.

PIC16F8x được thiết kế để dao động với một thạch anh song song, nếu dùng thạch anh nối tiếp thì tần số sẽ vượt khỏi phạm vi cho phép của nhà sản xuất. Ở chế độ XT. LP hoặc HS, thiết bị có thể nhận một nguồn dao động ngoài từ ngỏ vào OSC1/CLKIN (hình 3.28)



Hình 3.28 Dao động ngoài cho cấu hình XT, LP hoặc HS

Phạm vi thử nghiệm			
Chế độ	Tần số	OSC1/C1	OSC2/C2
XT	455 KHz 2 MHz 4 MHz	47 – 100 pF 15 – 33 pF 15 – 33 pF	47 – 100 pF 15 – 33 pF 15 – 33 pF
HS	8 MHz 10 MHz	15 – 33 pF 15 – 33 pF	15 – 33 pF 15 – 33 pF
Lưu ý : Nên chọn C1 và C2 bằng nhau trong bảng phạm vi thử nghiệm Tụ giá trị cao làm tăng độ ổn định nhưng cũng làm tăng thời gian khởi động Các giá trị này chỉ là gợi ý. Do mỗi mạch cộng hưởng có đặc tính riêng. Nên tham khảo tài liệu nhà sản xuất để chọn cho phù hợp			
Mạch cộng hưởng thử nghiệm			
455 KHz	Panasonic EFO-A455K04B		± 0.3%
2 MHz	Murata Erie CSA2.00MG		± 0.5%
4 MHz	Murata Erie CSA4.00MG		± 0.5%
8 MHz	Murata Erie CSA8.00MT		± 0.5%
10 MHz	Murata Erie CSA10.00MTZ		± 0.5%
Không có tụ trong các mạch cộng hưởng			

Bảng 3.10 Chọn tụ cho mạch cộng hưởng gốm

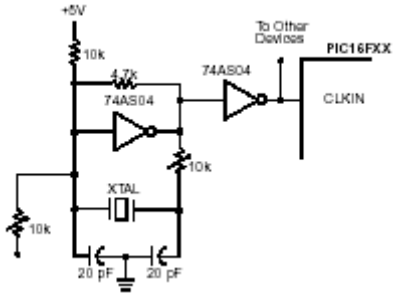
Chế độ	Tần số	OSC1/C1	OSC2/C2
LP	32 KHz 200 KHz	68 – 100 pF 15 – 33 pF	68 – 100 pF 15 – 33 pF
XT	100 KHz 2 MHz 4 MHz	100 – 150 pF 15 – 33 pF 15 – 33 pF	100 – 150 pF 15 – 33 pF 15 – 33 pF
HS	4 MHz 10 MHz	15 – 33 pF 15 – 33 pF	15 – 33 pF 15 – 33 pF
Lưu ý : Nên chọn C1 và C2 bằng nhau trong bảng phạm vi thử nghiệm Tụ giá trị cao làm tăng độ ổn định nhưng cũng làm tăng thời gian khởi động Các giá trị này chỉ là gợi ý. Rs có thể cần thiết trong chế độ HS và XT để tránh điều khiển quá mức so với mức điều khiển thấp của thạch anh theo sổ tay. Do các thạch anh có đặc tính không giống nhau. Nên tham khảo tài liệu nhà sản xuất để chọn cho phù hợp Với V_{DD} > 4,5 V nên chọn C1 = C2 = 30 pF			
Thạch anh thử nghiệm			
32,768 KHz	Epson C-001R32.768K-A		± 20 PPM
100 KHz	Epson C-2 100.00 KC-P		± 20 PPM
200 KHz	STD XTL 200.000 KHz		± 20 PPM
1 MHz	ECS ECS-10-13-2		± 50 PPM
2 MHz	ECS ECS-20-S-2		± 50 PPM
4 MHz	ECS ECS-40-S-4		± 50 PPM
10 MHz	ECS ECS-100-S-4		± 50 PPM

Hình 3.29 Giá trị tụ trong chế độ dao động thạch anh

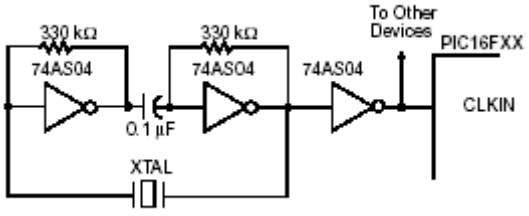
8.2.3 Mạch dao động thạch anh ngoài

Có thể dùng các mạch dao động đóng gói sẵn hoặc ghép lại từ các cổng TTL. Các mạch dao động chế tạo sẵn có dãi làm việc và độ ổn định cao hơn. Một mạch dao động thạch anh được thiết kế tốt sẽ cung cấp hiệu quả tốt với các cổng TTL. Sau đây là hai sơ đồ điển hình : Một là cộng hưởng nối tiếp và một là cộng hưởng song song.

Hình 3.30 trình bày mạch dao động cộng hưởng song song, mạch được thiết kế với tần số cơ bản của thạch anh. Cổng đảo 74AS04 làm nhiệm vụ đảo pha đây là yêu cầu của mạch dao động song song. Điện trở hồi tiếp 4,7 KΩ làm nhiệm vụ ổn định. Biến trở phân cực 10 KΩ chọn điểm làm việc cho 74AS04 nằm trong vùng tuyến tính



Hình 3.30 Mạch dao động thạch anh ngoài cộng hưởng song song



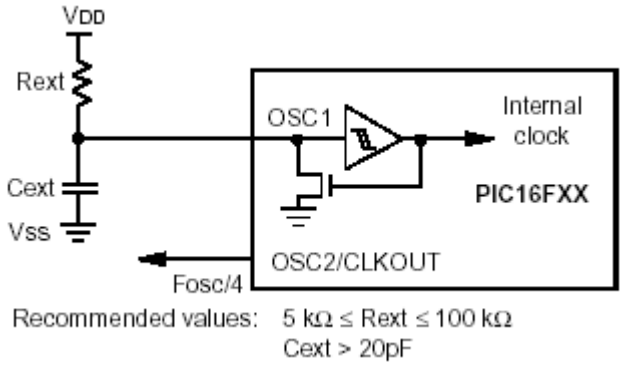
Hình 3.31 Mạch dao động thạch anh ngoài cộng hưởng nối tiếp

8.2.4 Dao động RC

Với những ứng dụng không cần thời gian chính xác thì tùy chọn dao động RC sẽ giúp giảm giá thành. Tần số dao động RC thay đổi theo điện áp nguồn, R_{ext} và C_{ext} và cả nhiệt độ làm việc. Hơn nữa tần số dao động còn thay đổi theo thiết bị do sự thay đổi thông số xử lý. Và điện dung liên cực giữa các dạng vỏ khác nhau của chip củng ảnh hưởng đặc biệt đối với giá trị C_{ext} thấp. Người dùng cần lưu ý đến dung sai của R và C. Hình 3.32 trình bày một mạch RC kết hợp với PIC16F8x. Với những R_{ext} thấp hơn 4 KΩ hoạt động của mạch có thể không ổn định hoặc không dao động. Nếu Rext quá lớn VD. 1 MΩ dao động dễ bị ảnh hưởng bởi nhiễu, độ ẩm và độ rỉ điện. Do đó nên chọn R_{ext} trong khoảng từ 5 KΩ đến 100 KΩ.

Mặc dù mạch dao động không cần tụ ngoài nhưng (Cext = 0 pF) nhưng cũng nên thêm tụ khoảng 20 pF để chống nhiễu và ổn định hơn. Nếu không có C_{ext} tần số dao động có thể thay đổi bất thường do điện dung bên ngoài biến thiên như điện dung lắp ráp, điện dung liên cực.

Tần số được chia 4 xuất hiện ở chân OSC2/CLKOUT có thể được dùng để kiểm tra hoặc đồng bộ các linh kiện khác.



Hình 3.32 Chế độ dao động RC

Lưu ý :

Khi đang dao động bằng RC không nên điều khiển chân OSC1 bằng một nguồn dao động khác có thể làm hư thiết bị

8.3 Reset

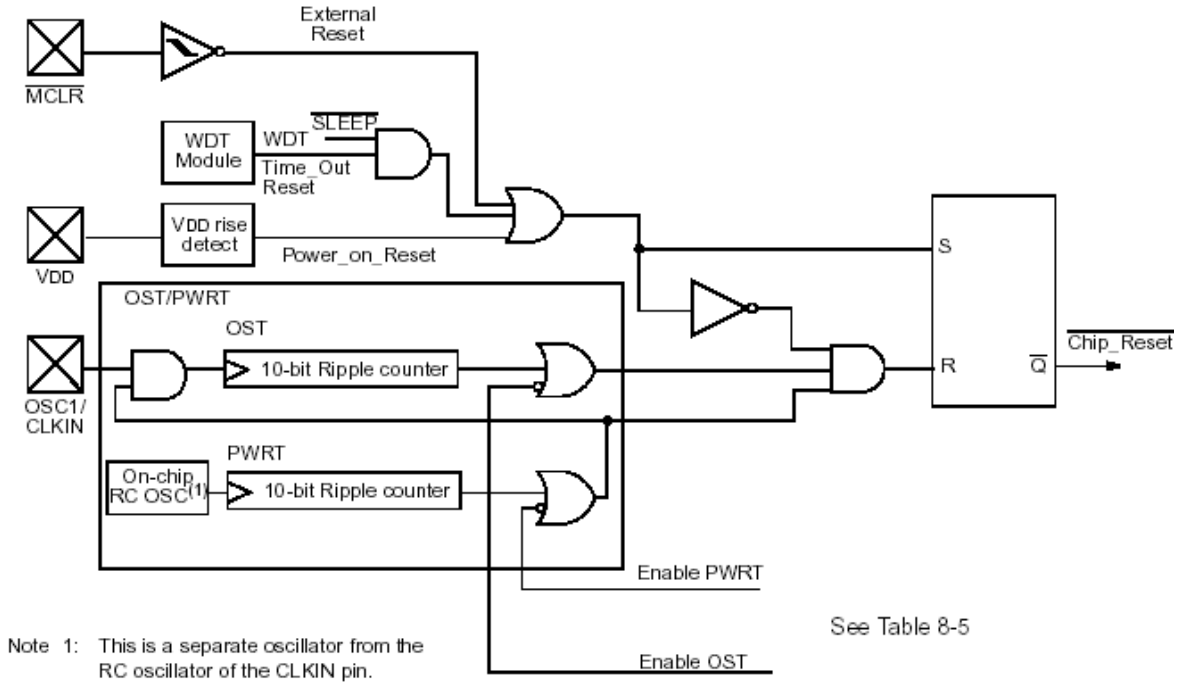
Có nhiều cách để reset PIC16F8x :

- Reset khi mở máy (POR)
- $\overline{MCLR}$ rest trong khi SLEEP
- WDT reset khi hoạt động bình thường
- WDT đánh thức trong khi SLEEP

Hình 3.33 trình bày sơ đồ khối của mạch reset tích hợp trên chip. Trên đường reset $\overline{MCLR}$ có một bộ lọc nhiễu. Bề rộng cần thiết của xung $\overline{MCLR}$ được cho trong sổ tay. Một số thanh ghi không bị ảnh hưởng với mọi chế độ reset. Trạng thái của chúng không xác định khi mở máy và không thay đổi ở các phương pháp reset khác. Hầu hết các thanh ghi khác được đưa về trạng thái ban đầu khi mở máy, reset $\overline{MCLR}$ khi hoạt động bình thường, reset WDT và reset $\overline{MCLR}$ khi SLEEP. Do đó các thao tác reset này được xem như là sự bắt đầu lại khi hoạt động bình thường.

Bảng 3.11 cho thấy các điều kiện reset của bộ đếm chương trình PC và thanh ghi trạng thái. Bảng 3.12 trình bày đầy đủ các trạng thái reset của tất cả các thanh ghi.

Các bít $\overline{TO}$ và $\overline{PD}$ được đặt và xóa một cách khác nhau trong các trường hợp reset khác nhau. Hai bít này được dùng trong phần mềm để xác định bản chất của reset.



Hình 3.33 Sơ đồ khối mạch reset trên chip

Điều kiện	**Bộ đếm chương trình**	**Thanh ghi trạng thái**
Reset mở máy	000h	0001 1xxx
Reset $\overline{MCLR}$ khi hoạt động bình thường	000h	000u uuuu
Reset $\overline{MCLR}$ trong khi SLEEP	000h	0001 0uuu
Reset WDT trong khi hoạt động bình thường	000h	0000 1uuu
Đánh thức bằng WDT	PC + 1	uuu0 0uuu
Đánh thức SLEEP bằng ngắt	PC + 1 [1]	uuu1 0uuu

Bảng 3.11 Điều kiện ngắt của PC và thanh ghi STATUS

Thanh ghi	Địa chỉ	Reset mở máy	Reset $\overline{MCLR}$ khi : Hoạt động bình thường SLEEP Reset WDT khi hoạt động bình thường	Đánh thức SLEEP Bằng ngắt Bằng WDT
W	**-**	**Xxxx xxxx**	**Uuuu uuuu**	**Uuuu uuuu**

[1] Khi đánh thức bằng ngắt và bít GIE được đặt, PC được nạp véc tơ ngắt 0004h

INDF	**00h**	**---- ----**	**---- ----**	**---- ----**
TMR0	**01h**	**xxxx xxxx**	**uuu uuuu**	**uuuu uuuu**
PCL	**02h**	**0000h**	**0000h**	**PC + 1** [1]
STATUS	**03h**	**0001 1xxx**	**000q quuu**	**uuuq quuu** [2]
FSR	**04h**	**xxxx xxxx**	**uuuu uuuu**	**uuuu uuuu**
PORTA	**05h**	**---x xxxx**	**---u uuuu**	**---u uuuu**
PORTB	**06h**	**xxxx xxxx**	**uuuu uuuu**	**uuuu uuuu**
EEDATA	**08h**	**xxxx xxxx**	**uuuu uuuu**	**uuuu uuuu**
EEADR	**09h**	**xxxx xxxx**	**uuuu uuuu**	**uuuu uuuu**
PCLATH	**0Ah**	**---0 0000**	**---0 0000**	**---u uuuu**
INTCON	**0Bh**	**0000 000x**	**0000 000u**	**uuuu uuuu** [3]
INDF	**80h**	**---- ----**	**---- ----**	**---- ----**
OPTION_REG	**81h**	**1111 1111**	**1111 1111**	**uuuu uuuu**
PCL	**82h**	**0000h**	**0000h**	**PC + 1**
STATUS	**83h**	**0001 1xxx**	**000q quuu**	**uuuq quuu**
FSR	**84h**	**xxxx xxxx**	**uuuu uuuu**	**uuuu uuuu**
TRISA	**85h**	**---1 1111**	**---1 1111**	**---u uuuu**
TRISB	**86h**	**1111 1111**	**1111 1111**	**uuuu uuuu**
EECON1	**88h**	**---0 x000**	**---0 q000**	**---0 uuuu**
EECON2	**89h**	**---- ----**	**---- ----**	**---- ----**
PCLATH	**8Ah**	**---0 0000**	**---0 0000**	**---u uuuu**
INTCON	**8Bh**	**0000 000x**	**0000 000u**	**uuuu uuuu** [3]

Bảng 3.12 Các điều kiện reset đối với tất cả các thanh ghi

8.4 Khởi động khi mở máy (POR)

Một xung POR được tạo ra trong chip khi độ tăng của VDD được nhận dạng (trong khoảng 1,2 V – 1,7 V). Để đạt được ưu điểm của POR, chân $\overline{MCLR}$ phải nối trực tiếp (hoặc ngang qua điện trở) đến V_{DD}, việc này sẽ loại bỏ các linh kiện RC ghép thêm bên ngoài vẫn thường được dùng để tạo reset khi mở máy. V_{DD} phải đạt thời gian tăng tối thiểu để bảo đãm hoạt động phù hợp.

Khi thiết bị bắt đầu hoạt động bình thường (tồn tại điều kiện reset) thì các thông số điện áp, nhiệt độ, tần số … phải có giá trị thích hợp để bảo đãm hoạt động. Nếu các thông số này không đạt yêu cầu thì thiết bị phải được duy trì ở trạng thái reset. Mạch POR không tạo ra reset bên trong khi điện áp nguồn nuôi VDD giảm xuống.

8.5 Bộ định thời tăng nguồn (PWRT)

Bộ định thời tăng nguồn (Power–up Timer) tạo ra một thời gian trễ cố định T_{PWRT} là 75 mS từ POR. Bộ định thời này hoạt động với mạch dao động RC bên trong. Chip được duy trì ở trạng thái reset cho đến khi PWRT tác động, thời gian PWRT cho phép VDD tăng đến một mức chấp nhận được (hình 8.13).

Bít cấu hình PWRTE cho phép hoặc không cho phép PWRT, Xem hình 3.25 hoặc hình 3.26 về chức năng của bít PWRTE đối với một thiết bị đặc biệt.

Thời gian trì hoãn tăng nguồn TPWRT thay đổi tùy theo thiết bị do VDD, nhiệt độ và những thay đổi xử lý.

8.6 Bộ định thời khởi động dao động (OST)

Bộ định thời khởi động dao đông (Oscilator Start-up Timer) tạo ra một thời gian trể bằng 1024 chu kỳ dao động (từ ngỏ vào OSC1) sau khi kết thúc trì hoãn PWRT để bảo đãm dao động thạch anh hoặc dao động gốm được khởi động và ổn định.

Thời gian OST (T_{OST}) chỉ được yêu cầu trong các chế độ XT, LP và HS và khi mở máy hoặc đánh thức từ trạng thái SLEEP.

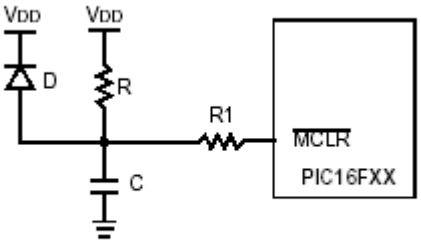
Khi V_{DD} tăng lên quá chậm có thể dẫn đến thời gian T_{PWRT} và T_{OST} kết thúc trước khi V_{DD} đạt giá trị cuối cùng. Trong trường hợp này (hình 3.38) có thể phải cần thêm một mạch reset mở máy bên ngoài (hình 3.34).

---

[1] Khi đánh thức do ngắt ngoài và bít GIE được đặt thì PC được nạp véc tơ ngắt là 0004h

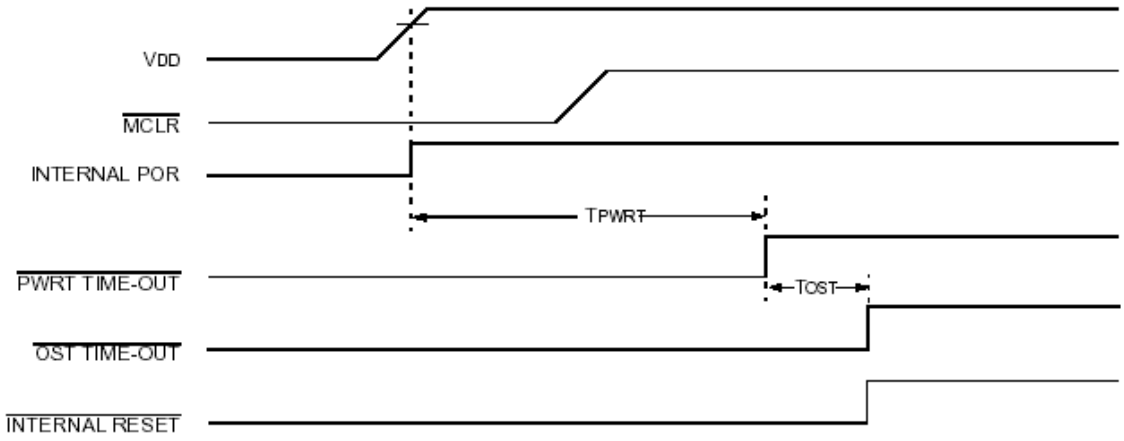
[2] Bảng 8.3 liệt kê giá trị reset cho mối điều kiện xác định

[3] Một hoặc nhiều bít của INTCON bị ảnh hưởng (để đánh thức)

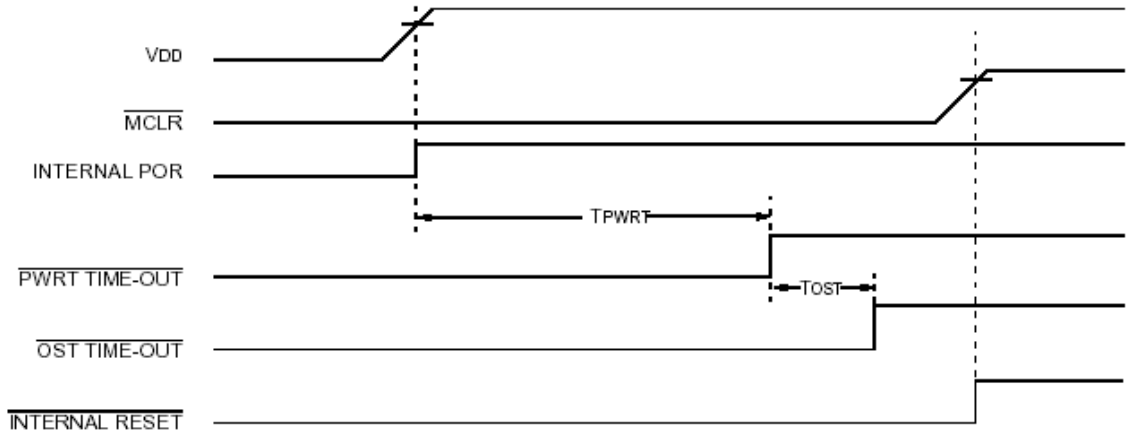


Hình 3.34 Mạch reset mở máy bên ngoài (khi V_{DD} tăng chậm)

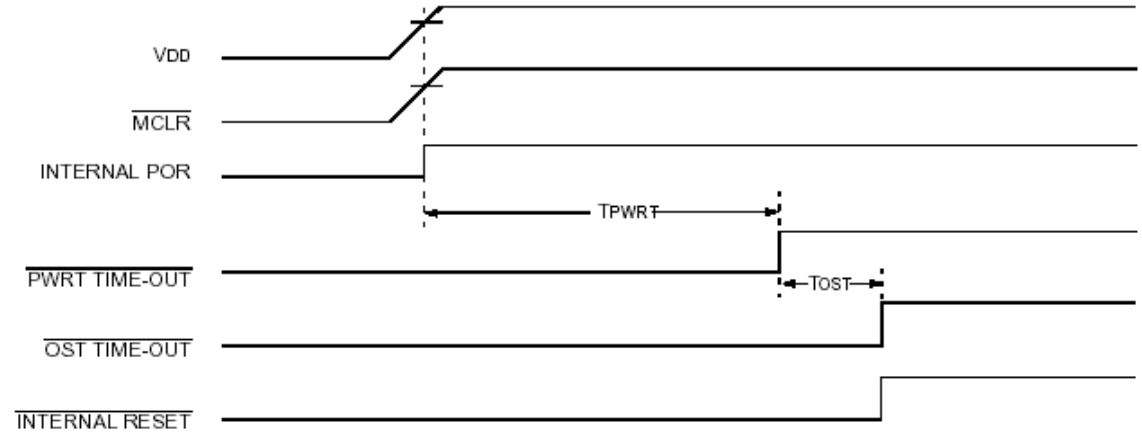
Note 1: External Power-on Reset circuit is required only if VDD power-up rate is too slow. The diode D helps discharge the capacitor quickly when VDD powers down.
2: R < 40 kΩ is recommended to make sure that voltage drop across R does not exceed 0.2V (max leakage current spec on $\overline{MCLR}$ pin is 5 µA). A larger voltage drop will degrade VIH level on the $\overline{MCLR}$ pin.
3: R1 = 100Ω to 1 kΩ will limit any current flowing into $\overline{MCLR}$ from external capacitor C in the event of an $\overline{MCLR}$ pin breakdown due to ESD or EOS.



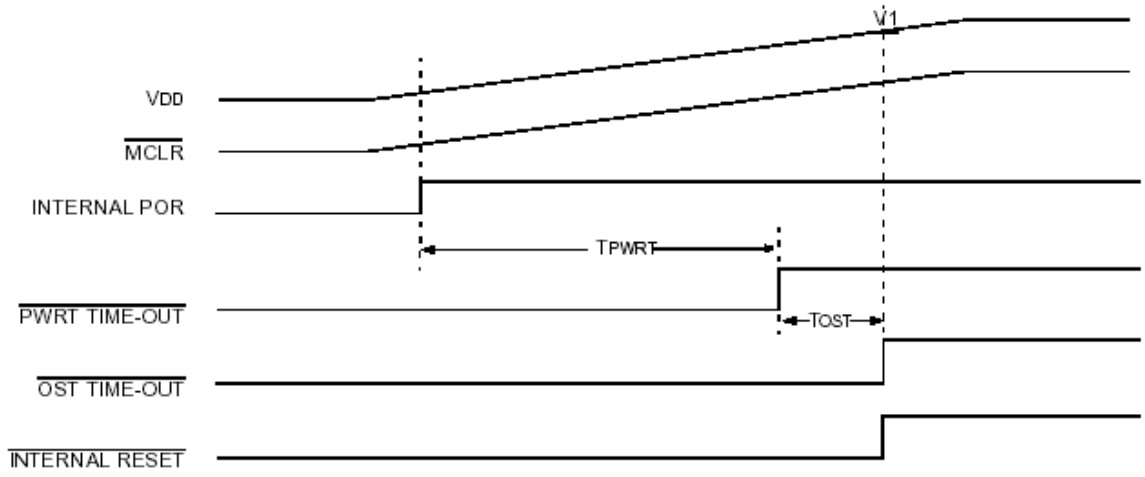
Hình 3.35 Đồ thị thời gian tăng nguồn ($\overline{MCLR}$ không nối lên V_{DD}) : Trường hợp 1



Hình 3.36 Đồ thị thời gian tăng nguồn ($\overline{MCLR}$ không nối lên V_{DD}) : Trường hợp 2



Hình 3.37 Thời gian tăng nguồn ($\overline{MCLR}$ nối lên V_{DD}) : V_{DD} tăng nhanh



When VDD rises very slowly, it is possible that the TPWRT time-out and TOST time-out will expire before VDD has reached its final value. In this example, the chip will reset properly if, and only if, V1 ≥ VDD min.

Hình 3.38 Thời gian tăng nguồn ($\overline{MCLR}$ nối lên V_{DD}) : V_{DD} tăng chậm

8.7 Thời gian trì hoản và các bít trạng thái hạ nguồn $\overline{T0}$ / $\overline{PD}$

Thời gian trễ khi tăng nguồn (hình 3.35 – 3.38) diễn tiến như sau : Trước tiên, bắt đầu thời gian trễ PWRT sau khi POR kết thúc. Sau đó OST tác động. Tổng thời gian trì hoản thay đổi theo cấu hình dao động và trạng thái bít cấu hình PWRTE. Ví dụ trong chế độ dao động RC và bít PWRTE không cho phép thì không có thời gian trì hoản.

Oscillator Configuration	Power-up		Wake-up from SLEEP
	PWRT Enabled	PWRT Disabled	
XT, HS, LP	72 ms + 1024TOSC	1024TOSC	1024TOSC
RC	72 ms	—	—

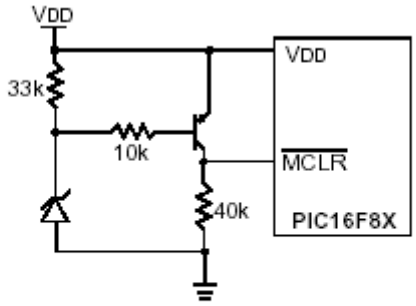
Bảng 3.13 Thời gian trì hoản tại các trạng thái khác nhau

Kể từ khi thời gian trễ bắt đầu sau xung POR. Nếu thời gian duy trì ở mức thấp của $\overline{MCLR}$ đủ lâu thì thời gian trễ sẽ kết thúc. Sau khi $\overline{MCLR}$ lên mức cao. Mạch lập tức hoạt động (hình 3.35). Điều này rất hữu dụng cho yêu cầu kiểm tra hoặc đồng bộ nhiều thiết bị PIC16F8x khi làm việc song song với nhau.

Bảng 3.14 cho thấy ý nghĩa của các bít $\overline{T0}$ và $\overline{PD}$. Bảng 3.11 liệt kê các điều kiện reset đối với một số các thanh ghi đặc biệt trong khi bảng 3.12 thì liệt kê đối với tất cả các thanh ghi

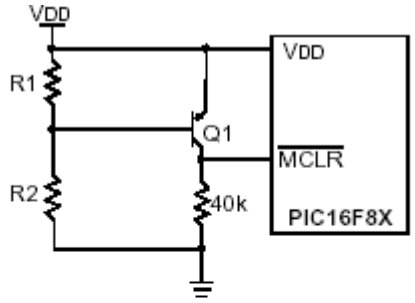
8.8 Reset khi sụt áp nguồn (Brown-out)

Đây là hiện tượng điện áp nguồn sụt thấp hơn giá trị tối thiểu nhưng sau đó phục hồi trở lại, trong trường hợp này nên duy trì thiết bị ở trạng thái reset.
Để reset PIC16F8x khi xảy ra sụt áp cần phải thêm một mạch bảo vệ bên ngoài như trình bày trong hình 3.39 và 3.40



Hình 3.39 Sơ đồ bảo vệ sụt áp 1

This circuit will activate reset when VDD goes below (Vz + 0.7V) where Vz = Zener voltage.



Hình 3.40 Sơ đồ bảo vệ sụt áp 2

This brown-out circuit is less expensive, although less accurate. Transistor Q1 turns off when VDD is below a certain level such that:

$$V_{DD} \cdot \frac{R1}{R1 + R2} = 0.7V$$

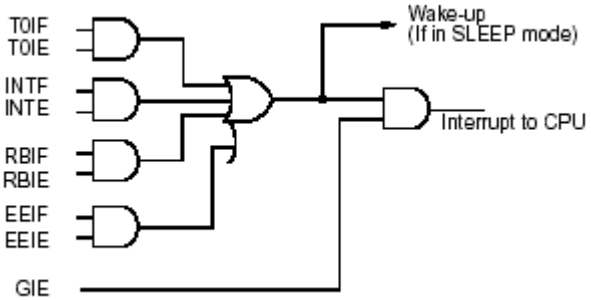
8.9 Tín hiệu ngắt

PIC16F8x có bốn nguồn tín hiệu ngắt :

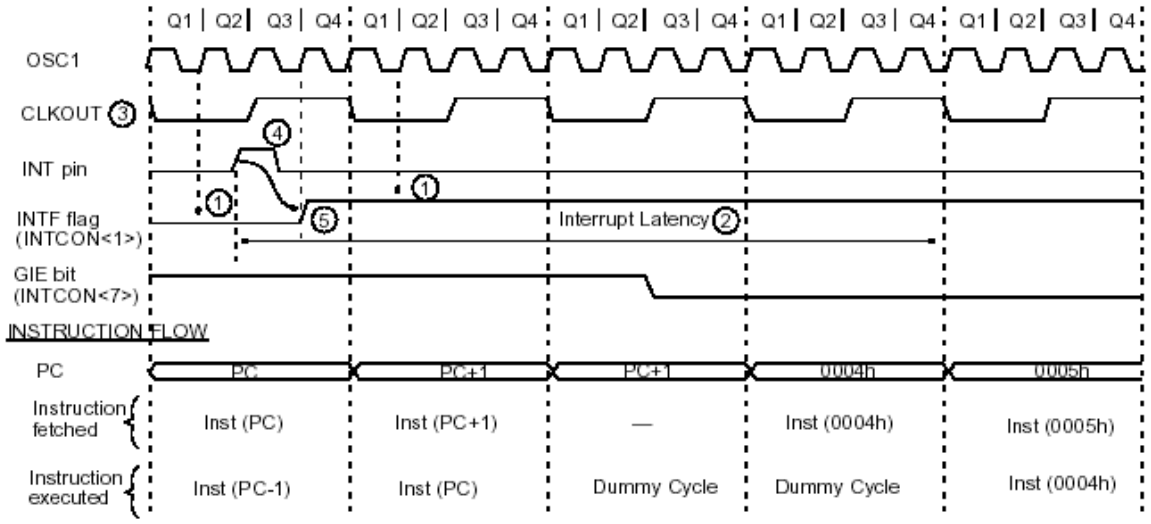
- Chân ngắt ngoài RB0/INT
- Ngắt timer TMR0
- Các ngắt thay đổi PORTB (chân RB7:RB4)
- Ngắt hoàn tất quá trình ghi vào bộ nhớ dữ liệu EEPROM

Thanh ghi điều khiển ngắt INTCON ghi nhớ các yêu cầu ngắt riêng biệt vào các bít cờ. Thanh ghi này còn chứa bít cho phép ngắt toàn cục và cho phép từng ngắt.
Bít cho phép ngắt toàn cục GIE (INTCON<7>) khi được đặt sẽ cho phép tất cả các ngắt và sẽ cấm tất cả các ngắt khi bị xóa. Các ngắt riêng có thể bị cấm bởi bít cho phép ngắt tương ứng trong thanh ghi INTCON. Bít GIE bị xóa khi reset.

Lệnh trở về từ chương trình phục vụ ngắt RETPIE sẽ đặt bít GIE để cho phép trở lại các ngắt.



Hình 3.41 Sơ đồ logic ngắt



Note 1: INTF flag is sampled here (every Q1).
2: Interrupt latency = 3-4Tcy where Tcy = instruction cycle time.
Latency is the same whether Inst (PC) is a single cycle or a 2-cycle instruction.
3: CLKOUT is available only in RC oscillator mode.
4: For minimum width of INT pulse, refer to AC specs.
5: INTF is enabled to be set anytime during the Q4-Q1 cycles.

Hình 3.42 Đồ thị thời gian chân INT

8.9.1 Ngắt INT

Ngắt ngoài tại chân RB0/INT tác động bằng cạnh : Cạnh lên nếu bít INTEDG (OPTION_REG<6>) được đặt hoặc tại cạnh xuống nếu bít này bị xóa. Khi một cạnh hợp lệ xảy ra trên chân INT thì bít INTF sẽ được đặt (INTCON<1>). Ngắt này có thể bị cấm bằng cách xóa bít điều khiển INTE (INTCON<4>). Cờ ngắt INTF phải được xóa bằng phần mềm trong chương trình phục vụ ngắt trước khi ngắt này được cho phép trở lại. Ngắt INT có thể đánh thức vi điều khiển khỏi trạng thái SLEEP chỉ khi nào bít INTE đã được đặt trước đó để đi vào trạng thái SLEEP. Trạng thái của bít GIE quyết định sự rẻ nhánh của vi điều khiển đến véc tơ ngắt sau khi thức dậy.

8.9.2 Ngắt TMR0

Khi xảy ra tràn từ FFh → 00h trong TMR0 sẽ đặt bít cờ T0IF (INTCON<2>) ngắt có thể được cho phép hoặc cấm bằng cách đặt hoặc xóa bít T0IF (INTCON<5>)

8.9.3 Ngắt PORT RB

Một sự thay đổi trên PORT<7:4> sẽ đặt cờ bít RBIF (INTCON<0>). Ngắt có thể được cho phép hoặc cấm bằng cách đặt hoặc xóa bít RBIE (INTCON<3>)

Lưu ý :

Sự thay đổi trên chân I/O port chỉ được nhận ra khi bề rộng xung nhỏ nhất phải bằng T_{CY}

8.10 Lưu trử nội dung trong khi ngắt

Trong khi ngắt, chỉ có giá trị trở về của PC được lưu trử trong ngăn xếp. Trong trường hợp người dùng muốn lưu nội dung thanh ghi khóa trong thời gian một ngắt ví dụ : Thanh ghi W và thanh ghi STÁTUS thì có thể thực hiện yêu cầu này bằng phần mềm.
Ví dụ 8-1 lưu trử và phục hồi nội dung của thanh ghi trạng thái và thanh ghi W. Người dùng đã định nghía các thanh ghi W_TEMP và STATUS_TEMP là các vị trí lưu trử tạm của thanh ghi W và thanh ghi STATUS.

Ví dụ 8-1 : Lưu trử thanh ghi W và STATUS trong RAM

```
PUSH  MOVWF                W_TEMP       ; Copy W vào thanh ghi tạm
```

```
                SWAPF       STATUS, W    ; Hoán chuyển trạng thái được lưu
                                         ; vào W
                MOVWF       STATUS_TEMP  ; Lưu trạng thái vào
                                         ; thanh ghi STATUS_TEMP
ISR     :                                ;
                :                        ;
                :                        ;
POP     SWAPF       STATUS_TEMP, W       ; Hoán chuyển bốn bít
                                         ; trong thanh ghi STATUS_TEMP
                                         ; và đặt kết quả vào W
                MOVWF       STATUS       ; Chuyển W vào thanh ghi STATUS
                SWAPF       W_TEMP, F    ; Hoán chuyển 4 bít trong W_TEMP
                                         ; và đặt kết quả vào W_TEMP
                SWAPF       W_TEMP, W    ; Hoán chuyển 4 bít trong W_TEMP
                                         ; và đặt kết quả vào W
```

Kết quả của ví dụ 8-1 :

a) Cất thanh ghi W
b) Cất thanh ghi STÁTUS vào STATUS_TEMP
c) Thực hiện chương trình phục vụ ngắt
d) Phục hồi thanh ghi STATUS
e) Phục hồi thanh ghi W

8.11 Bộ định thời canh chừng

Bộ định thời canh chừng WDT là một mạch dao động đa hài tự kích RC tích hợp trên chip và không cần thêm linh kiện bên ngoài. Mạch dao động này độc lập với mạch dao động RC tại chân OSC1/CLKIN, điều này có nghĩa là WDT vẫn chạy trong khi không có xung đồng hồ ở các chân OSC1 và OSC2. Ví dụ khi thực hiện lệnh SLEEP. Trong chế độ làm việc bình thường thời gian trễ WDT sẽ tạo ra tín hiệu reset thiết bị. Nếu thiết bị đang trong trạng thái SLEEP tín hiệu WDT sẽ đánh thức thiết bị để trở về trạng thái hoạt động bình thường. WDT có thể bị cấm bằng cách lập trình bít cấu hình WDTE là '0'.

8.11.1 Chu kỳ WDT

WDT có thời gian trễ danh định là 18 mS (không có định thang), thời gian này thay đổi theo nhiệt độ, VDD và phương pháp xử lý từng thiết bị (xem đặc tính DC). Nếu cần thời gian trễ dài hơn, một bộ định thang với tỉ số chia lên đến 1:128 có thể được gán cho WDT thông qua phần mềm bằng cách ghi vào thanh ghi OPTION_REG và thời gian trễ có thể lên đến 2, 3 giây.

Các lệnh CLRWDT và SLEEP sẽ xóa WDT và bộ định thang (nếu đã được gán cho WDT) để ngăn thời gian trễ và tạo ra tín hiệu reset.

Bít $\overline{T0}$ trong thanh ghi trạng thái sẽ bị xóa trong thời gian trễ WDT

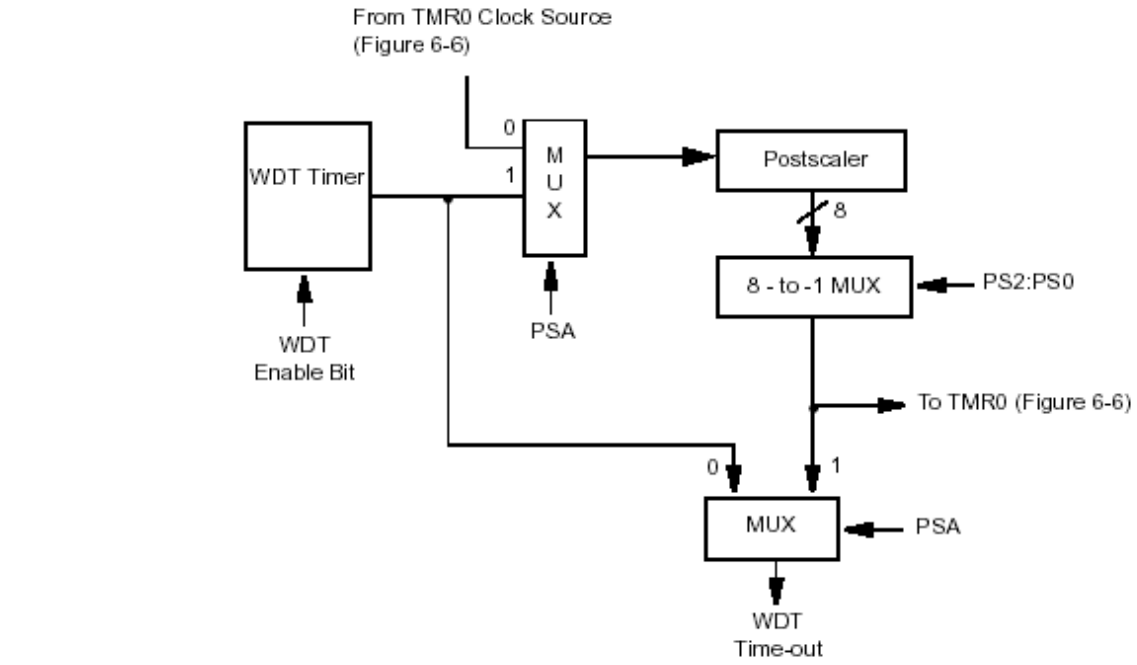
8.11.2 Lập trình WDT

Khi lập trình nên quan tâm đến các điều kiện xấu nhất (V_{DD} = min; nhiệt độ max, bộ định thang max) cần vài giây trước khi xảy ra WDT

Địa chỉ	Tên	Bít 7	Bít 6	Bít 5	Bít 4	Bít 3	Bít 2	Bít 1	Bít 0	Giá trị khi mở máy	Giá trị khi reset
2007h	**Bít cấu hình**	**(2)**	**(2)**	**(2)**	**(2)**	**PWRTE ([1])**	**WDTE**	**FOSC1**	**FOSC0**	**([2])**	
81h	**OPTION_REG**	$\overline{\text{REP0}}$	**INTEDG**	**T0CS**	**T0SE**	**PSA**	**PS2**	**PS1**	**PS0**	**1111 1111**	**1111 1111**

Bảng 3.15 Các thanh ghi tương ứng với WDT

[1] Xem chức năng của bít PWRTE ở hình 8.1 và 8.2
[2] Xem chức năng các bít bảo vệ mã lệnh và dữ liệu ở hình 8.1; 8.2 và mục 8.13



Note: PSA and PS2:PS0 are bits in the OPTION_REG register.

Hình 3.43 Sơ đồ khối WDT

8.12 Chế độ hạ nguồn (SLEEP)

Một thiết bị có thể được hạ nguồn (SLEEP) và sau đó tăng nguồn trở lại (thức dậy) để tiếp tục hoạt động bình thường.

8.12.1 Ngũ (SLEEP)

Để vào chế độ hạ nguồn phải thực hiện lệnh SLEEP, WDT sẽ bị xóa (nhưng vẫn đang chạy), bít $\overline{PD}$ (STATUS<3>) bị xóa, bít $\overline{T0}$ (STATUS<4>) được đặt và tắt mạch điều khiển dao động. Các cổng I/O duy trì trạng thái đã có trước khi lệnh SLEEP được thực hiện (high, low hoặc high Z).

Dòng tiêu thụ giảm xuống đến mức nhỏ nhất trong trạng thái SLEEP, các chân I/O được đặt lên V_{DD} hoặc V_{SS} và không cấp dòng ra mạch ngoài, xung đồng hồ ngoài bị cấm, các chân I/O vào có trạng thái Z cao nên được nối lên mức cao hoặc xuống mức thấp từ bên ngoài để tránh dòng điện chuyển mạch tạo ra bởi các ngỏ vào để trống. Ngỏ vào T0CK1 cũng nên nối đến V_{DD} hoặc V_{SS}. Cũng nên lưu ý đến các mạch kéo lên tại port B bên trong chip.

Chân $\overline{MCLR}$ phải ở mức cao (V_{IHMC}). Nên lưu ý rằng một tín hiệu reset tạo ra từ thời gian trễ WDT sẽ không đưa $\overline{MCLR}$ xuống mức thấp.

8.12.2 Đánh thức từ trạng thái SLEEP

Thiết bị có thể được đánh thức từ trạng thái SLEEP bởi các sự kiện sau đây :

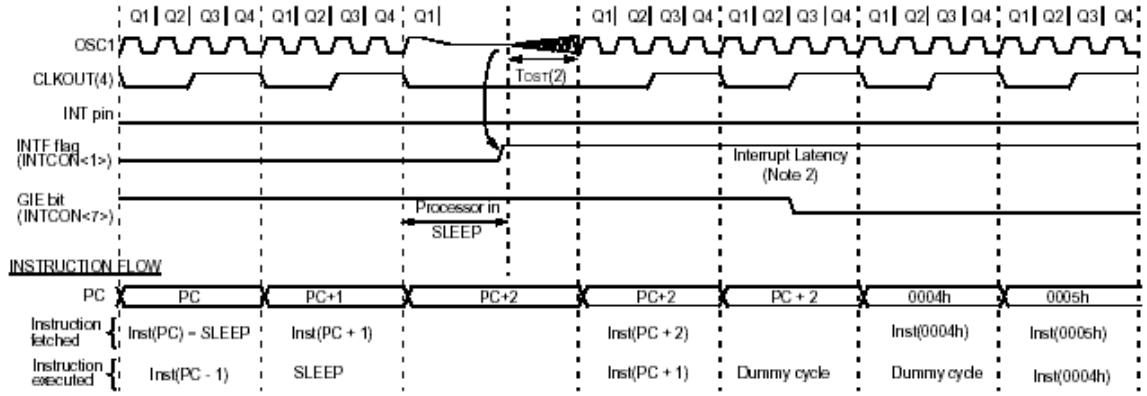
1. Reset bên ngoài tại chân $\overline{MCLR}$
2. WDT (nếu WDT đã được cho phép)
3. Ngắt từ chân RB0/INT, sự thay đổi ở port RB hoặc khi hoàn tất quá trình ghi dữ liệu EEPROM.

Các thiết bị ngoại vi không thể tạo ra ngắt trong khi SLEEP vì không có xung đồng hồ trong chip.

Sự kiện đầu tiên (reset $\overline{MCLR}$) sẽ gây nên một reset cho thiết bị. Hai sự kiện sau được xem như là một phần mở rộng của việc thực thi chương trình. Các bít $\overline{T0}$ và $\overline{PD}$ có thể được dùng để xác định nguyên nhân reset thiết bị. Bít $\overline{PD}$ được đặt khi tăng nguồn và

bị xóa khi cần SLEEP. Bít $\overline{T0}$ bị xóa nếu xuất hiện thời gian trễ WDT (và gây ra tín hiệu đánh thức).

Trong khi lệnh SLEEP đang được thực hiện, lệnh kế tiếp (PC + 1) được đọc vào. Muốn đánh thức thiết bị bằng ngắt thì bít cho phép của ngắt tương ứng phải được đặt. Việc đánh thức xảy ra bất chấp trạng thái bít cho phép toàn cục GIE. Nếu bít GIE bị xóa, thiết bị sẽ tiếp tục thực hiện lệnh theo sau lệnh SLEEP. Nếu bít GIE được đặt, thiết bị sẽ thực hiện lệnh sau lệnh SLEEP và sau đó nhảy đến địa chỉ ngắt 0004h. Trong trường hợp không muốn thực hiện lệnh sau lệnh SLEEP thì nên đặt một lệnh NOP sau lệnh SLEEP.



Note 1: XT, HS or LP oscillator mode assumed.
2: TOST = 1024TOSC (drawing not to scale) This delay will not be there for RC osc mode.
3: GIE = '1' assumed. In this case after wake- up, the processor jumps to the interrupt routine. If GIE = '0', execution will continue in-line.
4: CLKOUT is not available in these osc modes, but shown here for timing reference.

Hình 3.44 Đánh thức bằng ngắt

8.12.3 Đánh thức bằng các ngắt

Khi các ngắt toàn cục bị cấm (GIE bị xóa) và một nguồn ngắt nào đó có cả hai bít cho phép và cờ ngắt được đặt thì một trong những điều sau đây sẽ xảy ra :

- Nếu ngắt xảy ra trước khi thực hiện lệnh SLEEP, lệnh SLEEP sẽ giống như lệnh NOP. Do đó, WDT và bộ định thang WDT sẽ không bị xóa, bít $\overline{T0}$ không được đặt và bít $\overline{PD}$ không bị xóa
- Nếu ngắt xảy trong hoặc sau khi thực hiện lệnh SLEEP, thiết bị sẽ được đánh thức ngay lập tức khỏi trạng thái SLEEP. Lệnh SLEEP sẽ được hoàn tất trước khi thức dậy. Do đó, WDT và bộ định thang WDT sẽ bị xóa, bít $\overline{T0}$ được đặt và bít $\overline{PD}$ bị xóa.

Mặc dù các cờ được kiểm tra trước khi thực hiện lệnh SLEEP nhưng chúng có thể được đặt trước khi lệnh SLEEP hoàn tất . Để xác định lệnh SLEEP đã được thực hiện hay chưa thì phải kiểm tra bít $\overline{PD}$. Nếu $\overline{PD}$ được đặt thì lệnh SLEEP đã được thực hiện như một lệnh NOP.

Để chắc chắn WDT bị xóa, nên thực hiện lênh CLRWDT trước lệnh SLEEP.

8.13 Kiểm lại chương trình – Bảo vệ mã lệnh

Nếu các bít bảo vệ mã lệnh không được lập trình thì bộ nhớ chương trình có thể được đọc ra để phục vụ yêu cầu kiểm tra lại.

Lưu ý : Microchip không khuyến khích bảo vệ mã lệnh các thiết bị đơn lẻ.

8.14 Vị trí ID

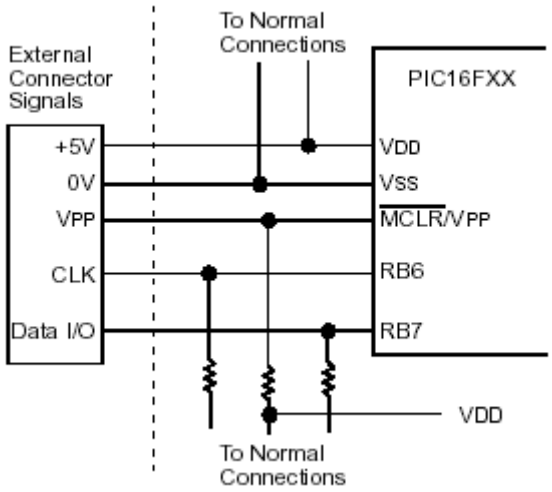
Có 4 ô nhớ được thiết kế để lưu trử các số kiểm tra hoặc các mã nhận dạng khác. Những vị trí này không thể truy cập khi đang hoạt động bình thường mà chỉ có thể đọc hoặc ghi trong khi đang lập trình. Chỉ có 4 bít bít thấp trong các vị trí này được sử dụng.

Đối với các thiết bị loại ROM, các giá trị này là mã của ROM

8.15 Lập trình nối tiếp trong hệ thống

Họ vi điều khiển PIC16F8x có thể lập trình nối tiếp ngay trên mạch ứng dụng cuối. Việc này được thực hiện một cách đơn giản bằng hai đường truyền xung đồng hồ và dữ liệu và 3 đường khác là nguồn, GND và điện áp lập trình. Người dùng có thể sản xuất các board mạch với các thiết bị chưa được lập trình và sau đó nạp chương trình vào vi điều khiển trước khi phân phối sản phẩm. Khả năng này cho phép nạp hầu hết các chương trình được tạo sẵn trước hoặc các chương trình theo ý người dùng.

Thiết bị được chuyển vào chế độ nạp/kiểm tra. Bộ đếm chương trình PC trỏ đến vị trí 00h, sau đó một lệnh 6 bít được gởi đến thiết bị và sau dó là dữ liệu 14 bít gởi đến hoặc đọc ra từ thiết bị bằng các lệnh nạp hoặc đọc



Hình 3.45 Sơ đồ kết nối lập trình nối tiếp trong hệ thống

Đối với các thiết bị ROM, cả 2 bộ nhớ chương trình và bộ nhớ dữ liệu EEPROM đều cho phép đọc nhưng chỉ có bộ nhớ EEPROM cho phép nạp

9. TẬP LỆNH

Mỗi lệnh của PIC16Cxx là một từ dài 14 bít gồm có phần mã công tác xác định thao tác cần thực hiện và một hoặc nhiều toán hạng cần thiết cho thao tác dó. Tóm tắt tập lệnh của PIC16Cxx trong bảng 3.17 liệt kê các lệnh byte, lệnh bít, ký tự và các lệnh điều khiển. Bảng 3.16 trình bày các mô tả vùng mã lệnh.

Đối với các lệnh byte. ‘f’ biểu diễn một thanh ghi được chỉ định và ‘d’ là đích, thanh ghi được chỉ định là thanh ghi được lệnh sử dụng.

Đích là nơi đặt kết quả của phép tính. Nếu ‘d’ bằng 0 thì kết quả được đặt vào thanh ghi W, nếu ‘d’ bằng 1 thì kết quả được đặt vào thanh xác định trong lệnh

Đối với các lệnh bít, ‘b’ biểu diễn một vùng bít được chỉ định, vùng bít này chọn số bít bị ảnh hưởng bởi thao tác. Trong khi ‘f’ biểu diễn số file trong đó chứa các bít.

Đối với các thao tác hằng số và điều khiển, ‘k’ biểu diễn một hằng 8 bít hoặc 11 bít hoặc giá trị ký tự.

Vùng	**Mô tả**
f	Địa chỉ tập thanh ghi 0x00 đến 0x7f
w	Thanh ghi làm việc (bộ tích lủy
b	Địa chỉ bít trong một thanh ghi 8 bít
k	Vùng hằng số, hằng dữ liệu hoặc nhản
x	Vùng không xác đinh (= 0 hoặc 1)
d	Đích chọn, d = 0 : kết quả lưu vào W D = 1 : kết quả lưu vào thanh ghi f. Mặc định d = 1
label	Tên nhản
TOS	Đỉnh ngăn xếp

PC	Bộ đếm chương trình
PCLATH	Chốt phần cao bộ đếm chương trình
GIE	Bít cho phép ngắt toàn cục
WDT	Watchdog Timer/Counter
$\overline{T0}$	Bít thời gian trễ
$\overline{PD}$	Bít hạ nguồn
dest	Đích hoặc thanh ghi w hoặc vị trí tập thanh ghi xác định
[ ]	Tùy chọn
( )	Nội dung
$\rightarrow$	Gán đến
< >	Vùng bít
$\in$	Trong tập hợp của
nghiêng	Ký hiệu được định nghĩa bởi người dùng

Bảng 3.16 Mô tả vùng mã lệnh

Tập lệnh có tính trực giao cao và được chia thành 3 nhóm :

- Nhóm lệnh byte
- Nhóm lệnh bít
- Nhóm lệnh hằng số và điều khiển

Tất cả các lệnh được thực hiện trong vòng một chu kỳ lệnh trừ phi một phép thử điều kiện là đúng hoặc bộ đếm chương trình bị thay đổi do một lệnh. Trong trường hợp này thời gian thực hiện là hai chu kỳ lệnh với chu kỳ thứ hai giống như một lệnh NOP. Một chu kỳ lệnh bao gồm 4 chu kỳ đồng hồ. Vì vậy với tần số đồng hồ là 4 MHz thì thời gian thực hiện một lệnh là 1 μS.

Bảng 3.17 liệt kê các lệnh được nhận ra bởi trình dịch hợp ngữ MPASM. Bảng 9.1 cho thấy các dạng tổng quát của các lệnh.

Lưu ý :

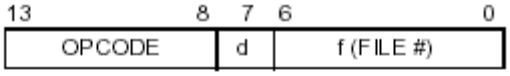
Để duy trì khả năng tương thích với các sản phẩm PIC16Cxx trong tương lai, không nên dùng các lệnh OPTION và TRIS

Tất cả các ví dụ dùng dạng thức sau đây để biểu diễn số thập lục phân

0xhh

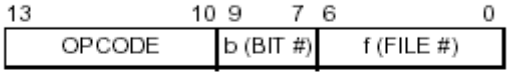
Trong đó h là một ký số hexa

Byte-oriented file register operations



d = 0 for destination W
d = 1 for destination f
f = 7-bit file register address

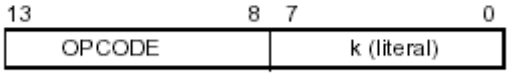
Bit-oriented file register operations



b = 3-bit bit address
f = 7-bit file register address

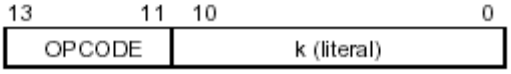
Literal and control operations

General



k = 8-bit immediate value

CALL and GOTO instructions only



k = 11-bit immediate value

Hình 3.46 Dạng tổng quát các lệnh

Lệnh gợi nhớ	Mô tả	Chu kỳ	Mã 14 bít **MSB LSB**	Bít ảnh hưởng	Ghi chú
Nhóm lệnh byte về tập thanh ghi					
ADDWF f, d	**Cộng w với f**	**1**	**00 0111 dfff ffff**	**C,DC,Z**	**1, 2**
ANDWF f, d	**AND w với f**	**1**	**00 0101 dfff ffff**	**Z**	**1, 2**
CLRF f	**Xóa f**	**1**	**00 0001 1fff ffff**	**Z**	**2**
CLRW	**Xóa w**	**1**	**00 0001 0xxx xxxx**	**Z**	
COMF f, d	**Bù f**	**1**	**00 1001 dfff ffff**	**Z**	**1, 2**
DECF f, d	**Giảm f**	**1**	**00 0011 dfff ffff**	**Z**	**1, 2**
DECFSZ f, d	**Giảm f, bỏ qua nếu bằng 0**	**1(2)**	**00 1011 dfff ffff**		**1,2,3**
INCF f, d	**Tăng f**	**1**	**00 1010 dfff ffff**	**Z**	**1, 2**
INCFSZ f, d	**Tăng f, bỏ qua nếu bằng 0**	**1(2)**	**00 1111 dfff ffff**		**1,2,3**
IORWF f, d	**OR w với f**	**1**	**00 0100 dfff ffff**	**Z**	**1, 2**
MOVF f, d	**Chuyển f**	**1**	**00 1000 dfff ffff**	**Z**	**1, 2**
MOVWF f	**Chuyển w vào f**	**1**	**00 0000 1fff ffff**		
NOP	**Không làm gì cả**	**1**	**00 0xx0 0000**		
RLF f, d	**Dịch trái f ngang qua Cy**	**1**	**00 1101 dfff ffff**	**C**	**1, 2**
RRF f, d	**Dịch phải f ngang qua Cy**	**1**	**00 1100 dfff ffff**	**C**	**1, 2**
SUBWF f, d	**Trừ w cho f**	**1**	**00 0010 dfff ffff**	**C,DC,Z**	**1, 2**
SWAPF f, d	**Hoán chuyển 4 bít trong f**	**1**	**00 1110 dfff ffff**		**1, 2**
XORWF f, d	**XOR w với f**	**1**	**00 0110 dfff ffff**	**Z**	**1, 2**
Nhóm lệnh bit về tập thanh ghi					
BCF f, b	**Xóa bít f**	**1**	**01 00bb bfff ffff**		**1, 2**
BSF f, b	**Set bít f**	**1**	**01 01bb bfff ffff**		**1, 2**
BTFSC f, b	**Kiểm tra bít f, bỏ qua nếu xóa**	**1(2)**	**01 10bb bfff ffff**		**3**
BTFSS f, b	**Kiểm tra bít f, bỏ qua nếu set**	**1(2)**	**01 11bb bfff ffff**		**3**
Nhóm lệnh hằng số và điều khiển					
ADDLW k	**Cộng hằng số với w**	**1**	**11 111x kkkk kkkk**	**C,DC,Z**	
ANDLW k	**AND hằng số với w**	**1**	**11 1001 kkkk kkkk**	**Z**	
CALL k	**Gọi chương trình con**	**2**	**10 0kkk kkkk kkkk**		
CLRWDT	**Xóa WDT**	**1**	**00 0000 0110 0100**	$\overline{TO}$, $\overline{PD}$	
GOTO k	**Nhảy đến địa chỉ**	**2**	**10 1kkk kkkk kkkk**		
IORLW k	**OR hằng số với w**	**1**	**11 1000 kkkk kkkk**	**Z**	
MOVLW k	**Chuyển hằng số vào w**	**1**	**11 00xx kkkk kkkk**		
RETFIE	**Trở về từ chương trình ngắt**	**2**	**00 0000 0000 1001**		
RETLW k	**Trở về với hằng số trong w**	**2**	**11 01xx kkkk kkkk**		
RETURN	**Trở về từ chương trình con**	**2**	**00 0000 0000 1000**		
SLEEP	**Vào chế độ ngũ**	**1**	**00 0000 0110 0011**	$\overline{TO}$, $\overline{PD}$	
SUBLW k	**Trừ w cho hằng số**	**1**	**11 110x kkkk kkkk**	**C.DC.Z**	
XORLW k	**XOR w với hằng số**	**1**	**11 1010 kkkk kkkk**	**Z**	

Ghi chú 1 : Khi một thanh ghi I/O được thay đổi như hàm của chính nó VD: MOVF PORT, 1 thì giá trị được sử dụng chính là giá trị tại chân của nó. VD: Nếu chốt địa chỉ là '1' đối với 1 chân được cấu hình là ngỏ vào và được kéo xuống thấp bởi thiết bị ngoài thì dữ liệu sẽ được ghi vào trở lại là '0'

Ghi chú 2 : Nếu lệnh này được thực hiện trên thanh ghi TMR0 và d =1 thì bộ định thang sẽ bị xóa nếu đang được gán cho Timer0

Ghi chú 3 : Nếu PC bị thay đổi hoặc một kiểm tra điều kiện là đúng thì lệnh cần đến 2 chu kỳ máy, chu kỳ thứ hai được thực hiện như 1 lệnh NOP

Bảng 3.17 Tập lệnh PIC16Fxx

ADDLW	**Cộng hằng số với W**
Cú pháp :	[label] ADDLW k
Toán hạng :	$0 \le k \le 255$
Thao tác :	$(W) + k \rightarrow (W)$
Bít ảnh hưởng :	C, DC, Z

Mã lệnh :	11 111x kkkk kkkk
Mô tả :	Nội dung của thanh ghi W được cộng với hằng số ‘k’ 8 bít và kết quả được đặt vào thanh ghi W
Words :	1
Cycles :	1

Chu kỳ hoạt động Q :	Q1	Q2	Q3	Q4
	Giãi mã	Đọc hằng ‘k’	Xử lý dữ liệu	Ghi vào W

Ví dụ :

```
ADDLW  0x15
Trước khi thực hiện lệnh :
W = 0x10
Sau khi thực hiện lênh :
W = 0x25
```

ADDWF **Cộng W với f**

Cú pháp :	[label] ADDWF f, d
Toán hạng :	$0 \le f \le 127$ $d \in [0,1]$
Thao tác :	(W) + (f) $\rightarrow$ (đích)
Bít ảnh hưởng :	C, DC, Z
Mã lệnh :	00 0111 dfff ffff
Mô tả :	Công nội dung của thanh ghi W với thanh ghi ‘f’ . Nếu ‘d’ là 0 thì kết quả được chứa trong thanh ghi W. Nếu ‘d’ là 1 thì kết quả được chứa trở lại trong thanh ghi ‘f’
Words :	1
Cycles :	1

Chu kỳ hoạt động Q :	Q1	Q2	Q3	Q4
	Giãi mã	Đọc thanh ghi ‘f’	Xử lý dữ liệu	Ghi vào đích

Ví dụ :

```
ADDWF  FSR, 0
Trước khi thực hiện lệnh :
W = 0x17
FSR = 0xC2
Sau khi thực hiện lênh :
W = 0xD9
FSR = 0xC2
```

ANDLW **AND hằng số với W**

Cú pháp :	[label] ANDLW k
Toán hạng :	$0 \le k \le 255$
Thao tác :	(W) AND (k) $\rightarrow$ (W)
Bít ảnh hưởng :	Z
Mã lệnh :	11 1001 kkkk kkkk
Mô tả :	Nội dung của thanh ghi W được AND với hằng số ‘k’ 8 bít và kết quả được đặt vào thanh ghi W
Words :	1

Cycles : 1

Chu kỳ hoạt động Q :	Q1	Q2	Q3	Q4
	Giãi mã	Đọc hằng ‘k’	Xử lý dữ liệu	Ghi vào W

Ví dụ :
```
ANDLW  0x5F
Trước khi thực hiện lệnh :
W = 0xA3
Sau khi thực hiện lênh :
W = 0x03
```

ANDWF **AND W với f**

Cú pháp : [label] ANDWF f, d

Toán hạng : $0 \le f \le 127$, $d \in [0,1]$

Thao tác : (W) AND (f) $\rightarrow$ (đích)

Bít ảnh hưởng : Z

Mã lệnh : 00 0101 dfff ffff

Mô tả : AND nội dung của thanh ghi W với thanh ghi ‘f’ . Nếu ‘d’ là 0 thì kết quả được chứa trong thanh ghi W. Nếu ‘d’ là 1 thì kết quả được chứa trở lại trong thanh ghi ‘f’

Words : 1

Cycles : 1

Chu kỳ hoạt động Q :	Q1	Q2	Q3	Q4
	Giãi mã	Đọc thanh ghi ‘f’	Xử lý dữ liệu	Ghi vào đích

Ví dụ :
```
ANDWF  FSR, 1
Trước khi thực hiện lệnh :
W = 0x17
FSR = 0xC2
Sau khi thực hiện lênh :
W = 0x17
FSR = 0x02
```

BCF **Xóa bít f**

Cú pháp : [label] BCF f, b

Toán hạng : $0 \le f \le 127$, $0 \le b \le 7$

Thao tác : $0 \rightarrow$ (f<b>)

Bít ảnh hưởng : không

Mã lệnh : 01 00bb bfff ffff

Mô tả : Xóa bít ‘b’ trong thanh ghi ‘f’

Words : 1

Cycles : 1

Chu kỳ hoạt động Q :	Q1 Giãi mã	Q2 Đọc thanh ghi ‘f’	Q3 Xử lý dữ liệu	Q4 Ghi vào thanh ghi ‘f’

Ví dụ :
```
BCF  FLAG_REG, 7
Trước khi thực hiện lệnh :
FLAG_REG = 0xC7
Sau khi thực hiện lênh :
FLAG_REG = 0x47
```

BSF **Đặt bít f**

Cú pháp : [label] BSF f, b
Toán hạng : $0 \le f \le 127$
$0 \le b \le 7$
Thao tác : $1 \rightarrow$ (f<b>)
Bít ảnh hưởng : không

Mã lệnh : 01 01bb bfff ffff

Mô tả : Đặt bít ‘b’ trong thanh ghi ‘f’
Words : 1
Cycles : 1

Chu kỳ hoạt động Q :	Q1 Giãi mã	Q2 Đọc thanh ghi ‘f’	Q3 Xử lý dữ liệu	Q4 Ghi vào thanh ghi ‘f’

Ví dụ :
```
BSF  FLAG_REG, 7
Trước khi thực hiện lệnh :
FLAG_REG = 0x0A
Sau khi thực hiện lênh :
FLAG_REG = 0x8A
```

BTFSC **Kiểm tra bít, bỏ qua nếu xóa**

Cú pháp : [label] BTFSC f, b
Toán hạng : $0 \le f \le 127$
$0 \le b \le 7$
Thao tác : Bỏ qua nếu (f<b>) = 0
Bít ảnh hưởng : không

Mã lệnh : 01 10bb bfff ffff

Mô tả : Nếu bít ‘b’ trong thanh ghi ‘f’ là ‘1’ thì thực hiện lệnh kế tiếp
Nếu bít ‘b’ trong thanh ghi ‘f’ là ‘0’ thì bỏ qua lệnh kế tiếp và thay vào đó là thực hiện lệnh NOP. Lúc này thời gian thực hiện cần 2 T_{CY}
Words : 1
Cycles : 1 (2)

Chu kỳ hoạt động Q :

Q1	Q2	Q3	Q4
Giải mã	Đọc thanh ghi ‘f’	Xử lý dữ liệu	NOP

Nếu bỏ qua (chu kỳ thứ 2)

Q1	Q2	Q3	Q4
NOP	NOP	NOP	NOP

```
HERE    BTFSC  FLAG, 1
FALSSE  GOTO   PROCESS_CODE
TRUE    *
        *
        *
```

Ví dụ :

Trước khi thực hiện lệnh
PC = địa chỉ của HERE
Sau khi thực hiện lệnh
Nếu FLAG<1> = 0
PC = địa chỉ của TRUE
Nếu FLAG<1> = 1
PC = địa chỉ của FALSSE

BTFSS **Kiểm tra bít, bỏ qua nếu đặt**

Cú pháp : [label] BTFSS f, b
Toán hạng : $0 \le f \le 127$
$0 \le b \le 7$
Thao tác : Bỏ qua nếu (f<b>) = 1
Bít ảnh hưởng : không

Mã lệnh : 01 11bb bfff ffff

Mô tả : Nếu bít ‘b’ trong thanh ghi ‘f’ là ‘0’ thì thực hiện lệnh kế tiếp
Nếu bít ‘b’ trong thanh ghi ‘f’ là ‘1’ thì bỏ qua lệnh kế tiếp và thay vào đó là thực hiện lệnh NOP. Lúc này thời gian thực hiện cần 2 T_{CY}

Words : 1
Cycles : 1 (2)
Chu kỳ hoạt động Q :

Q1	Q2	Q3	Q4
Giải mã	Đọc thanh ghi ‘f’	Xử lý dữ liệu	NOP

Nếu bỏ qua (chu kỳ thứ 2)

Q1	Q2	Q3	Q4
NOP	NOP	NOP	NOP

Ví dụ :

```
HERE    BTFSS  FLAG, 1
FALSSE  GOTO   PROCESS_CODE
TRUE    *
        *
        *
```

Trước khi thực hiện lệnh
PC = địa chỉ của HERE
Sau khi thực hiện lệnh

Nếu FLAG<1> = 0
PC = địa chỉ của FALSSE
Nếu FLAG<1> = 1
PC = địa chỉ của TRUE

CALL	**Gọi chương trình con**
Cú pháp :	[label] CALL k
Toán hạng :	$0 \le k \le 2047$
Thao tác :	(PC) + 1 → TOS k → PC<10:0> (PCLATH<4:3>) → PC<12:11>
Bít ảnh hưởng :	không
Mã lệnh :	10 0kkk Kkkk kkkk
Mô tả :	Gọi chương trình con : Trước tiên, địa chỉ trở về (PC + 1) được cất vào ngăn xếp. Địa chỉ tức thời 11 bít được nạp vào các bít PC<10:0>, các bít cao của pC được nạp từ PCLATH. Lệnh CALL cần hai chu kỳ lệnh
Words :	1
Cycles :	2
Chu kỳ hoạt động Q :	

Q1	Q2	Q3	Q4
Giãi mã	Đọc 'k', cất PC vào ngăn xếp	Xử lý dữ liệu	Ghi vào PC
NOP	NOP	NOP	NOP

Ví dụ :
HERE CALL THERE
Trước khi thực hiện lệnh
PC = địa chỉ của HERE
Sau khi thực hiện lênh
PC = địa chỉ của THERE
TOS = địa chỉ HERE +1

CLRF	**Xóa f**
Cú pháp :	[label] CLRF f
Toán hạng :	$0 \le f \le 127$
Thao tác :	00h → (f) 1 → Z
Bít ảnh hưởng :	Z
Mã lệnh :	00 0001 1fff ffff
Mô tả :	Xóa nội dung thanh ghi 'f' và đặt bít Z
Words :	1
Cycles :	1
Chu kỳ hoạt động Q :	

Q1	Q2	Q3	Q4
Giãi mã	Đọc thanh ghi 'f'	Xử lý dữ liệu	Ghi vào thanh ghi 'f'

Ví dụ : CLRF FLAG_REG
Trước khi thực hiện lênh
FLAG_REG = 0x5A
Sau khi thực hiện lệnh
FLAG_REG = 0x00
Z = 1

CLRW **Xóa w**

Cú pháp : [label] CLRW
Toán hạng : không
Thao tác : 00h → (W)
1 → Z
Bít ảnh hưởng : Z

Mã lệnh : 00 0001 0xxx xxxx

Mô tả : Xóa nội dung thanh ghi w và đặt bít Z
Words : 1
Cycles : 1
Chu kỳ hoạt động Q :

Q1	Q2	Q3	Q4
Giãi mã	NOP	Xử lý dữ liệu	Ghi vào thanh ghi w

Ví dụ : CLRW
Trước khi thực hiện lênh
w = 0x5A
Sau khi thực hiện lệnh
w = 0x00
Z = 1

CLRWDT **Xóa Watchdog Timer**

Cú pháp : [label] CLRWDT
Toán hạng : không
Thao tác : 00h → WDT
0 → bộ định thang WDT
1 → $\overline{T0}$
1 → $\overline{PD}$
Bít ảnh hưởng : $\overline{T0}$, $\overline{PD}$

Mã lệnh : 00 0000 0110 0100

Mô tả : Reset WDT và bộ định thang của WDT, đặt bít $\overline{T0}$ và $\overline{PD}$
Words : 1
Cycles : 1
Chu kỳ hoạt động Q :

Q1	Q2	Q3	Q4
Giãi mã	NOP	Xử lý dữ liệu	Xóa WDT

Ví dụ : CLRWDT
Trước khi thực hiện lênh
WDT counter = ?
Sau khi thực hiện lệnh

WDT counter = 0x00
Bộ định thang WDT = 0
$\overline{T0}$ = 1
$\overline{PD}$ = 1

COMF **Đảo f**

Cú pháp : [label] COMF f, d
Toán hạng : $0 \le f \le 127$
$d \in [0,1]$
Thao tác : $(\bar{f}) \rightarrow$ (đích)
Bít ảnh hưởng : Z

Mã lệnh : 00 1001 dfff ffff

Mô tả : Đảo nội dung của thanh ghi 'f'. Nếu 'd' là 0 thì kết quả được chứa trong W, nếu 'd' là 1 thì kết quả được đưa trở lại vào thanh ghi 'f'
Words : 1
Cycles : 1
Chu kỳ hoạt động Q :

Q1	Q2	Q3	Q4
Giãi mã	Đọc thanh ghi 'f'	Xử lý dữ liệu	Ghi vào đích

Ví dụ : COMF REG1, 0
Trước khi thực hiện lênh
REG1 = 0x13
Sau khi thực hiện lệnh
REG1 = 0x13
W = 0xEC

DECF **Giảm f**

Cú pháp : [label] DECF f, d
Toán hạng : $0 \le f \le 127$
$d \in [0,1]$
Thao tác : (f) -1 $\rightarrow$ (đích)
Bít ảnh hưởng : Z

Mã lệnh : 00 0011 dfff ffff

Mô tả : Giảm nội dung của thanh ghi 'f'. Nếu 'd' là 0 thì kết quả được chứa trong W, nếu 'd' là 1 thì kết quả được đưa trở lại vào thanh ghi 'f'
Words : 1
Cycles : 1
Chu kỳ hoạt động Q :

Q1	Q2	Q3	Q4
Giãi mã	Đọc thanh ghi 'f'	Xử lý dữ liệu	Ghi vào đích

Ví dụ : DECF CNT, 1
Trước khi thực hiện lênh
CNT = 0x01
Sau khi thực hiện lệnh

CNT = 0x00
Z = 1

DECFSZ **Giảm f, bỏ qua nếu bằng 0**

Cú pháp : [label] DECFSZ f, d

Toán hạng : $0 \le f \le 127$
$d \in [0,1]$

Thao tác : (f) -1 $\rightarrow$ (đích)
bỏ qua nếu kết quả bằng 0

Bít ảnh hưởng : không

Mã lệnh : 00 1011 dfff ffff

Mô tả : Giảm nội dung của thanh ghi 'f'. Nếu 'd' là 0 thì kết quả được chứa trong W, nếu 'd' là 1 thì kết quả được đưa trở lại vào thanh ghi 'f'
Nếu kết quả bằng khác 0 thì thực hiện lệnh kế tiếp, nếu kết quả bằng 0 thì thực hiện lệnh NOP, lúc này cần 2TCY

Words : 1

Cycles : 1 (2)

Chu kỳ hoạt động Q :

Q1	Q2	Q3	Q4
Giải mã	Đọc thanh ghi 'f'	Xử lý dữ liệu	Ghi vào đích

Nếu bỏ qua (chu kỳ thứ 2

Q1	Q2	Q3	Q4
NOP	NOP	NOP	NOP

Ví dụ :

```
HERE        DECFSZ  CNT, 1
            GOTO    LOOP
CONTINUE    *
            *
            *
```

Trước khi thực hiện lênh
PC = địa chỉ HERE
Sau khi thực hiện lệnh
CNT = CNT - 1
Nếu CNT = 0
    PC = địa chỉ CONTINUE
Nếu CNT $\neq$ 0
    PC = địa chỉ HERE + 1

GOTO **Nhảy không điều kiện**

Cú pháp : [label] GOTO k

Toán hạng : $0 \le k \le 2047$

Thao tác : K $\rightarrow$ PC<10:0>
PCLATH<4:3> $\rightarrow$ PC<12:11>

Bít ảnh hưởng : không

Mã lệnh : 10 1kkk kkkk kkkk

Mô tả : Giá trị tức thời 11 bít được nạp vào các bít PC<10:0>, các bít trên của PC được nạp từ PCLATH<4:3>, lệnh này cần 2 chu kỳ

Words : 1
Cycles : 2
Chu kỳ hoạt động Q :

Q1	Q2	Q3	Q4
Giãi mã	Đọc ‘k’	Xử lý dữ liệu	Ghi vào PC
NOP	NOP	NOP	NOP

Ví dụ : GOTO THERE
Sau khi thực hiện lệnh
PC = địa chỉ của THERE

INCF **Tăng f**

Cú pháp : [label] INCF f, d
Toán hạng : $0 \le f \le 127$
$d \in [0,1]$
Thao tác : (f) + 1 $\rightarrow$ (đích)
Bít ảnh hưởng : Z

Mã lệnh : 00 1010 dfff ffff

Mô tả : Tăng nội dung của thanh ghi ‘f’. Nếu ‘d’ là 0 thì kết quả được chứa trong W, nếu ‘d’ là 1 thì kết quả được đưa trở lại vào thanh ghi ‘f’
Words : 1
Cycles : 1
Chu kỳ hoạt động Q :

Q1	Q2	Q3	Q4
Giãi mã	Đọc thanh ghi ‘f’	Xử lý dữ liệu	Ghi vào đích

Ví dụ : INCF CNT, 1
Trước khi thực hiện lênh
CNT = 0xFF
Z = 0
Sau khi thực hiện lệnh
CNT = 0x00
Z = 1

INCFSZ **Tăng f, bỏ qua nếu bằng 0**

Cú pháp : [label] INCFSZ f, d
Toán hạng : $0 \le f \le 127$
$d \in [0,1]$
Thao tác : (f) + 1 $\rightarrow$ (đích)
bỏ qua nếu kết quả bằng 0
Bít ảnh hưởng : không

Mã lệnh : 00 1111 dfff ffff

Mô tả : Tăng nội dung của thanh ghi ‘f’. Nếu ‘d’ là 0 thì kết quả được chứa trong W, nếu ‘d’ là 1 thì kết quả được đưa trở lại vào thanh ghi ‘f’
Nếu kết quả khác 0 thì thực hiện lệnh kế tiếp, nếu kết quả bằng 0 thì thực hiện lệnh NOP, lúc này cần $2T_{CY}$
Words : 1
Cycles : 1 (2)

Chu kỳ hoạt động Q :

Q1	Q2	Q3	Q4
Giải mã	Đọc thanh ghi 'f'	Xử lý dữ liệu	Ghi vào đích

Nếu bỏ qua (chu kỳ thứ 2)

Q1	Q2	Q3	Q4
NOP	NOP	NOP	NOP

Ví dụ :

```
HERE      INCFSZ  CNT, 1
          GOTO    LOOP
CONTINUE  *
          *
          *
```

Trước khi thực hiện lênh
PC = địa chỉ HERE
Sau khi thực hiện lệnh
CNT = CNT + 1
Nếu CNT = 0
    PC = địa chỉ CONTINUE
Nếu CNT $\neq$ 0
    PC = địa chỉ HERE + 1

IORLW **OR hằng số với W**

Cú pháp : [label] IORLW k
Toán hạng : $0 \leq k \leq 255$
Thao tác : (W) OR (k) $\rightarrow$ (W)
Bít ảnh hưởng : Z

Mã lệnh : 11 1000 kkkk kkkk

Mô tả : Nội dung của thanh ghi W được OR với hằng số 'k' 8 bít và kết quả được đặt vào thanh ghi W
Words : 1
Cycles : 1

Chu kỳ hoạt động Q :

Q1	Q2	Q3	Q4
Giải mã	Đọc hằng 'k'	Xử lý dữ liệu	Ghi vào W

Ví dụ :

IORLW 0x35
Trước khi thực hiện lệnh :
W = 0x9A
Sau khi thực hiện lênh :
W = 0xBF
Z = 1

IORWF **OR W với f**

Cú pháp : [label] IORWF f, d
Toán hạng : $0 \leq f \leq 127$
$d \in [0,1]$
Thao tác : (W) OR (f) $\rightarrow$ (đích)
Bít ảnh hưởng : Z

Mã lệnh : 00 0100 dfff ffff

Mô tả : OR nội dung của thanh ghi W với thanh ghi ‘f’ . Nếu ‘d’ là 0 thì kết quả được chứa trong thanh ghi W. Nếu ‘d’ là 1 thì kết quả được chứa trở lại trong thanh ghi ‘f’

Words : 1

Cycles : 1

Chu kỳ hoạt động Q :

Q1	Q2	Q3	Q4
Giãi mã	Đọc thanh ghi ‘f’	Xử lý dữ liệu	Ghi vào đích

Ví dụ :

IORWF RESULT, 0
Trước khi thực hiện lệnh :
W = 0x91
RESULT = 0x13
Sau khi thực hiện lênh :
W = 0x93
RESULT = 0x13
Z = 1

MOVF Chuyển f

Cú pháp : [label] MOVF f, d

Toán hạng : $0 \leq f \leq 127$
$d \in [0,1]$

Thao tác : (f) $\rightarrow$ (đích)

Bít ảnh hưởng : Z

Mã lệnh : 00 1000 dfff ffff

Mô tả : Chuyển nội dung thanh ghi ‘f’ vào đích. Nếu ‘d’ là 0 thì kết quả được chứa trong thanh ghi W. Nếu ‘d’ là 1 thì kết quả được chứa trở lại trong thanh ghi ‘f’. ‘d’ =1 thường được dùng để thử một thanh ghi do bít Z bị ảnh hưởng

Words : 1

Cycles : 1

Chu kỳ hoạt động Q :

Q1	Q2	Q3	Q4
Giãi mã	Đọc thanh ghi ‘f’	Xử lý dữ liệu	Ghi vào đích

Ví dụ :

MOVF FSR, 0
Sau khi thực hiện lênh :
W = giá trị của thanh ghi FSR
Z = 1

MOVLW Chuyển hằng số vào W

Cú pháp : [label] MOVLW k

Toán hạng : $0 \leq k \leq 255$

Thao tác : k $\rightarrow$ (W)

Bít ảnh hưởng : không

Mã lệnh : 11 00xx kkkk kkkk

Mô tả : Nạp hằng số 8 bít vào W, những giá trị không xác định được

hợp dịch là ‘0’

Words : 1

Cycles : 1

Chu kỳ hoạt động Q :	Q1	Q2	Q3	Q4
	Giải mã	Đọc hằng ‘k’	Xử lý dữ liệu	Ghi vào W

Ví dụ :
```
MOVLW  0x5A
Sau khi thực hiện lênh :
W = 0x5A
```

MOVWF **Chuyển W vào f**

Cú pháp : [label] MOVWF f

Toán hạng : $0 \le f \le 127$

Thao tác : $(W) \rightarrow (f)$

Bít ảnh hưởng : không

Mã lệnh : 00 0000 dfff ffff

Mô tả : Chuyển nội dung thanh ghi W vào thanh ghi ‘f’

Words : 1

Cycles : 1

Chu kỳ hoạt động Q :	Q1	Q2	Q3	Q4
	Giải mã	Đọc thanh ghi W	Xử lý dữ liệu	Ghi vào thanh ghi ‘f’

Ví dụ :
```
MOVWF   OPTION_REG
Trước khi thực hiện lệnh :
W = 0x4F
OPTION = 0xFF
Sau khi thực hiện lênh :
W = 0x4F
OPTION = 0x4F
```

NOP **Không làm gì cả**

Cú pháp : [label] NOP

Toán hạng : không

Thao tác : Không làm gì cả

Bít ảnh hưởng : không

Mã lệnh : 00 0000 0xx0 0000

Mô tả : Không làm gì cả

Words : 1

Cycles : 1

Chu kỳ hoạt động Q :	Q1	Q2	Q3	Q4
	Giải mã	NOP	NOP	NOP

Ví dụ :
```
NOP
```

RETFIE **Trở về từ ngắt**

Cú pháp : [label] RETFIE
Toán hạng : không
Thao tác : TOS → PC
1 → GIE
Bít ảnh hưởng : không

Mã lệnh : 00 0000 0000 1001

Mô tả : Đỉnh ngăn xếp được nạp vào PC, đặt bít cho phép ngắt toàn cục GIE (INTCON<7>), lệnh này 2 chu kỳ
Words : 1
Cycles : 2
Chu kỳ hoạt động Q :

Q1	Q2	Q3	Q4
Giãi mã	NOP	Đặt bít GIE	Lấy ra từ ngăn xếp
NOP	NOP	NOP	NOP

Ví dụ : RETFIE
Sau ngắt
PC = TOS
GIE = 1

OPTION **Nạp thanh ghi OPTION**

Cú pháp : [label] OPTION
Toán hạng : không
Thao tác : (W) → OPTION
Bít ảnh hưởng : không

Mã lệnh : 00 0000 0110 0010

Mô tả : Nạp nội dung thanh ghi W vào thanh ghi OPTION. Lệnh này được hỗ trợ để tương thích với PIC16C5x do OPTION cho phép đọc/viết nên người dùng có thể định địa chỉ trực tiếp
Words : 1
Cycles : 1

Chu kỳ hoạt động Q :

Q1	Q2	Q3	Q4
Giãi mã	NOP	NOP	NOP

Ví dụ : Mục đích nhằm tương thích với các sản phẩm PIC16Cxx trong tương lai

RETLW **Trở về với hằng số trong W**

Cú pháp : [label] RETLW k
Toán hạng : $0 \le k \le 255$
Thao tác : K → (W)

TOS → PC

Bít ảnh hưởng : không

Mã lệnh : 11 01xx kkkk kkkk

Mô tả : Nạp hằng số k 8 bít vào W. nạp đỉnh ngăn xếp vào PC (địa chỉ trở về), lệnh này 2 chu kỳ

Words : 1

Cycles : 2

Chu kỳ hoạt động Q :

Q1	Q2	Q3	Q4
Giải mã	Đọc 'k'	NOP	Ghi vào W, lấy ra từ ngăn xếp
NOP	NOP	NOP	NOP

Ví dụ :

```
CALL   TABLE  ; W chứa giá trị offset của bảng
:             ; Bây giờ W là giá trị bảng
:
:
TABLE  ADDWF  PC  ; W = offset
       RETLW  K1  ; Bắt đầu bảng
       RETLW  K2  ;
       :
       :
       RETLW  Kn  ; Cuối bảng
```

Trước khi thực hiện lệnh
W = 0x07
Sau khi thực hiện lệnh
W = giá trị của K8

RETURN **Trở về từ chương trình con**

Cú pháp : [label] RETURN

Toán hạng : không

Thao tác : TOS → PC

Bít ảnh hưởng : không

Mã lệnh : 00 0000 0000 1000

Mô tả : Đỉnh ngăn xếp được nạp vào PC, lệnh này 2 chu kỳ

Words : 1

Cycles : 2

Chu kỳ hoạt động Q :

Q1	Q2	Q3	Q4
Giải mã	NOP	NOP	Lấy ra từ ngăn xếp
NOP	NOP	NOP	NOP

Ví dụ : RETURN

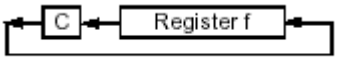
Sau khi thực hiện lệnh
PC = TOS

RLF **Quay trái f ngang qua carry**

Cú pháp : [label] RLF f, d
Toán hạng : $0 \le f \le 127$
$d \in [0,1]$
Thao tác : Xem mô tả bên dưới
Bít ảnh hưởng : C

Mã lệnh : 00 1101 dfff ffff

Mô tả :



Nội dung của thanh ghi 'f' bị quay trái 1 bít ngang qua cờ carry. Nếu 'd' = 0, kết quả được đặt vào W, nếu 'd' =1, kết quả được đặt trở lại vào thanh ghi 'f'

Words : 1
Cycles : 1

Chu kỳ hoạt động Q :

Q1	Q2	Q3	Q4
Giãi mã	Đọc 'f'	Xử lý dữ liệu	Ghi vào đích

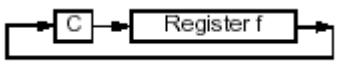
Ví dụ : RLF REG1, 0
Trước khi thực hiện lệnh
REG1 = 1110 0110
C = 0
Sau khi thực hiện lệnh
REG1 = 1110 0110
W = 1100 1100
C = 1

RRF **Quay phải f ngang qua carry**

Cú pháp : [label] RRF f, d
Toán hạng : $0 \le f \le 127$
$d \in [0,1]$
Thao tác : Xem mô tả bên dưới
Bít ảnh hưởng : C

Mã lệnh : 00 1100 dfff ffff

Mô tả :



Nội dung của thanh ghi 'f' bị quay phải 1 bít ngang qua cờ carry. Nếu 'd' = 0, kết quả được đặt vào W, nếu 'd' =1, kết quả được đặt trở lại vào thanh ghi 'f'

Words : 1
Cycles : 1

Chu kỳ hoạt động Q :

Q1	Q2	Q3	Q4
Giãi mã	Đọc 'f'	Xử lý dữ liệu	Ghi vào đích

Ví dụ : RRF REG1, 0

Trước khi thực hiện lệnh
REG1 = 1110 0110
C = 0
Sau khi thực hiện lệnh
REG1 = 1110 0110
W = 0111 0011
C = 0

SLEEP **Quay phải f ngang qua carry**

Cú pháp : [label] SLEEP
Toán hạng : không
Thao tác : 00h → WDT
0 → bộ định thang WDT
1 → $\overline{T0}$
0 → $\overline{PD}$
Bít ảnh hưởng : $\overline{T0}$, $\overline{PD}$

Mã lệnh : 00 0000 0110 0011

Mô tả : Bít trạng thái hạ nguồn $\overline{PD}$ bị xóa, bít trạng thái định thời $\overline{T0}$ được đặt, bộ định thời canh chừng và bộ định thang của nó bị xóa
Dao động đồng hồ ngừng hoạt động, vi xử lý đi vào chế độ SLEEP
Words : 1
Cycles : 1

Chu kỳ hoạt động Q :

Q1	Q2	Q3	Q4
Giãi mã	NOP	NOP	Đi vào SLEEP

Ví dụ : SLEEP

SUBLW **Trừ W với hằng số**

Cú pháp : [label] SUBLW k
Toán hạng : $0 \le k \le 255$
Thao tác : K – (W) → (W)
Bít ảnh hưởng : C, DC, Z

Mã lệnh : 11 110x kkkk kkkk

Mô tả : Nội dung của thanh ghi W bị trừ (phương pháp số bù 2) từ hằng số 'k' 8 bít và kết quả được đặt vào thanh ghi W
Words : 1
Cycles : 1

Chu kỳ hoạt động Q :

Q1	Q2	Q3	Q4
Giãi mã	Đọc hằng 'k'	Xử lý dữ liệu	Ghi vào W

Ví dụ 1 : SUBLW 0x02
Trước khi thực hiện lệnh :
W = 1
C = ?

Z = ?
Sau khi thực hiện lênh :
W = 1
C = 1 ; kết quả dương
Z = 0

Ví dụ 2 : Trước khi thực hiện lệnh
W = 2
C = ?
Z = ?
Sau khi thực hiện lệnh
W = 0
C = 1 ; kết quả là 0
Z = 1

Ví dụ 3 : Trước khi trù
W = 3
C = ?
Z = ?
Sau khi trừ
W = 0xFF
C = 0 ; Kết quả âm
Z = 0

SUBWF **Trừ W từ f**

Cú pháp : [label] SUBWF f, d

Toán hạng : $0 \le f \le 127$
$d \in [0,1]$

Thao tác : (f) – (W) $\rightarrow$ (đích)

Bít ảnh hưởng : C, DC, Z

Mã lệnh : 00 0010 dfff ffff

Mô tả : Trừ thanh ghi W từ thanh ghi ‘f’ . Nếu ‘d’ là 0 thì kết quả được chứa trong thanh ghi W. Nếu ‘d’ là 1 thì kết quả được chứa trở lại trong thanh ghi ‘f’

Words : 1

Cycles : 1

Chu kỳ hoạt động Q :

Q1	Q2	Q3	Q4
Giãi mã	Đọc thanh ghi ‘f’	Xử lý dữ liệu	Ghi vào đích

Ví dụ 1 : SUBWF REG1, 1
Trước khi trừ
REG1 = 3
W = 2
C = ?
Z = ?
Sau khi trừ
REG1 = 1
W = 2
C = 1 ; Kết quả dương
Z = 0

Ví dụ 2 : Trước khi trừ

REG1 = 2
W = 2
C = ?
Z = ?
Sau khi trừ
REG1 = 0
W = 2
C =1 ; Kết quả là 0
Z = 1

Ví dụ 3 : Trước khi trừ
REG1 = 1
W = 2
C = ?
Z = ?
Sau khi trừ
REG1 = 0xFF
W = 2
C = 0 ; Kết quả âm
Z = 0

SWAP **Hoán chuyển 4 bít trong f**

Cú pháp : [label] SWAP f, d

Toán hạng : $0 \le f \le 127$
$d \in [0,1]$

Thao tác : (f<3:0>) $\rightarrow$ (đích<7:4>)
(f<7:4>) $\rightarrow$ (đích<3:0>)

Bít ảnh hưởng : không

Mã lệnh : 00 1100 dfff ffff

Mô tả : 4 bít cao và 4 bít thấp của thanh ghi 'f' sẽ hoán chuyển cho nhau. Nếu 'd' = 0 thì kết quả được chứa vào W, nếu 'd' =1 thì kết quả được đưa trở lại vào 'f'

Words : 1

Cycles : 1

Chu kỳ hoạt động Q :

Q1	Q2	Q3	Q4
Giãi mã	Đọc 'f'	Xử lý dữ liệu	Ghi vào đích

Ví dụ : SWAP REG, 0
Trước khi thực hiện
REG1 = 0xA5
Sau khi thực hiện
REG1 = 0xA5
W = 0x5A

TRIS **Nạp thanh ghi TRIS**

Cú pháp : [label] TRIS f

Toán hạng : $0 \le f \le 127$

Thao tác : (W) $\rightarrow$ TRIS register f

Bít ảnh hưởng : không

Mã lệnh : 00 0000 0110 0fff

Mô tả : Lệnh được hỗ trợ cho yêu cầu tương thích với họ PIC16C5x trong tương lai, vì các thanh ghi TRIS cho phép đọc/viết nên có thể định địa chỉ trực tiếp

Words : 1

Cycles : 1

Chu kỳ hoạt động Q :

Ví dụ : Mục đích tương thích trong tương lai

XORLW **XOR hằng số với W**

Cú pháp : [label] XORLW k

Toán hạng : $0 \leq k \leq 255$

Thao tác : (W) XOR (k) $\rightarrow$ (W)

Bít ảnh hưởng : Z

Mã lệnh : 11 1010 kkkk kkkk

Mô tả : Nội dung của thanh ghi W được XOR với hằng số 'k' 8 bít và kết quả được đặt vào thanh ghi W

Words : 1

Cycles : 1

Chu kỳ hoạt động Q :

Q1	Q2	Q3	Q4
Giãi mã	Đọc hằng 'k'	Xử lý dữ liệu	Ghi vào W

Ví dụ :

XORLW 0xÀ
Trước khi thực hiện lệnh :
W = 0x85
Sau khi thực hiện lênh :
W = 0x1A

XORWF **XOR W với f**

Cú pháp : [label] XORWF f, d

Toán hạng : $0 \leq f \leq 127$
$d \in [0,1]$

Thao tác : (W) XOR (f) $\rightarrow$ (đích)

Bít ảnh hưởng : Z

Mã lệnh : 00 0110 dfff ffff

Mô tả : XOR nội dung của thanh ghi W với thanh ghi 'f' . Nếu 'd' là 0 thì kết quả được chứa trong thanh ghi W. Nếu 'd' là 1 thì kết quả được chứa trở lại trong thanh ghi 'f'

Words : 1

Cycles : 1

Chu kỳ hoạt động Q :	Q1	Q2	Q3	Q4
	Giãi mã	Đọc thanh ghi ‘f’	Xử lý dữ liệu	Ghi vào đích

Ví dụ :

XORWF REG, 1
Trước khi thực hiện lệnh :
W = 0x85
REG = 0xAF
Sau khi thực hiện lênh :
W = 0x85
REG = 0x1A

TÀI LIỆU THAM KHẢO

1.	PIC16F8x Data Sheet	MICROCHIP Technology Inc
2.	Giáo trình vi điều khiển họ PIC	Trần văn Trọng
3.	8051 Mikrocontroller erfolgreich anwenden	Jurgen Maier-Wolf
4.	Kỹ thuật lập trình C	Phạm văn Ất
5.	AT90S8535 Data Sheet	ATMEL
6.	Kỹ thuật vi điều khiển AVR	Ngô diên Tập