Chương 1 : Ôn lại về ngôn ngữ C theo chuẩn ANSI

1.1. Cấu trúc cơ bản của một chương trình C

Trước tiên ta xét ví du: Viết chương trình C hiện dòng thông báo " Chào các bạn đến với chương trình C" ra màn hình.

Cụ thể chương trình

```
    /* Chương trình thí dụ*/
// my first program in C
 #include <stdio.h>
#include <conio.h>
void main()
{
  clrscr();/* Câu lệnh xoá màn hình*/
printf("Chào các bạn đến với chương trình C!");
getch();
  }
```

Khai báo tệp tiêu đề

Trong ngôn ngữ lập trình C khi sử dụng các hàm chuẩn trong các thư viện chuẩn chúng ta phải khai báo tệp tiêu đề(header file) chứa các hàm nguyên mẫu tương ứng các hàm đó, các lệnh được bắt đầu bằng #include theo sau là tệp tiêu đề

Có hai cách viết như sau:

Cách 1: #include <[đường dẫn\] tentep>

Ví dụ: #include <a:\Baitap\Bai1.C>

#include <stdio.h>

Cách 2: #include "[đường dẫn\]tentep"

Ví dụ: #include "a:\Baitap\Bai2.C"

#include <conio.h>

Cách 1 tự động tìm tentep trong thư mục INCLUDE

Cách 2 tự động tìm tentep trong thư mục hiện thời nếu không có thì tìm trong thư mục INCLUDE

Trong thí dụ trên chúng ta có sử dụng hàm printf(...) là hàm chuẩn được khai báo trong tệp tiêu đề stdio.h và hàm getch(), clrscr() được khai báo trong tệp tiêu đề

conio.h. Do đó trong chương trình có hai dòng khai báo sau ở đầu chương trình:

#include <stdio.h>

#include <conio.h>

Chú thích và dấu kết thúc câu lệnh

Trong ngôn ngữ lập trình C những phần được viết trong /*...*/ được gọi là phần chú thích. Mọi ký tự nằm trong /*...*/ khi dịch chương trình dich bỏ qua, ta được phép dùng chúng để minh hoạ cho các thành phần chương trình làm cho chương trình dễ hiểu, mạch lạc. Lời chú thích có thể xuất hiện bất kỳ đâu trong chương trình và có thể trải trên nhiều dòng khác nhau trong chương trình.

Trong chương trình viết bằng ngôn ngữ C mỗi câu lệnh có thể viết trên một hay nhiều dòng và phải kết thúc bằng dấu chấm phẩy(;).

1.2. Các yếu tố cơ bản của ngôn ngữ C - ANSI

1.2.1 Bộ chữ viết

Ngôn ngữ C được xây dựng trên bộ ký tự sau:

Các chữ cái hoa: A B C .... Z

Các chữ cái thường:a b c ... z

Các chữ số:0 1 2... 9

Các dấu chấm câu: , . ; : / ? [ ] { } @ # $ % ^ * & ( ) + - = < > ‘ “...

Các dấu ngăn cách không nhìn thấy như dấu cách, dấu nhảy cách tab, dấu xuống dòng

Dấu gạch nối dưới _

1.2.2 Từ khoá

Là những từ có một ý nghĩa hoàn toàn xác định trong chương trình:

Ví dụ: void struct class while ....

Không được dùng từ khoá để đặt tên cho các hằng, biến, mảng, hàm ....

Từ khoá phải viết bằng chữ thường

Ví dụ từ khoá viết đúng: struct

Ví dụ từ khoá viết sai: Struct

1.2.3 Tên

Là một dãy ký tự được dùng để chỉ tên hằng, tên biến, tên mảng, tên hàm...Tên được tạo thành từ các chữ cái a..z, A..Z, chữ số 0..9, dấu gạch dưới. Tên không được bắt đầu bằng chữ số, chứa các kí tự đặc biệt như dấu cách, dấu phép toán...

Tên không được đặt trùng với từ khoá.

Ví dụ: Giai_Phuong_Trinh_Bac2

abc123

Chú ý:

-Trong ngôn ngữ lập trình C tên được phân biệt chữ hoa và chữ thường

-Thông thường chữ hoa thường được dùng để đặt tên cho các hằng, còn các đại lượng khác thì dùng chữ thường.

2.1.4 Một số kiểu dữ liệu cơ bản

- Kiểu ký tự (Char)

a Một giá trị kiểu char chiếm một byte và biểu diễn được một ký tự trong bảng mã ASCII.

- Kiểu số nguyên

Một giá trị kiểu số nguyên là một phần tử của một tập các số nguyên mà máy tính có thể biểu diễn. Trong ngôn ngữ lập trình C có nhiều kiểu dữ liệu số nguyên với dải giá trị khác nhau cụ thể:

Kiểu	Phạm vi biểu diễn	Kích thước(byte)
Char	-128 -> 127	1
Unsigned char	0->255	1
Int	-32768->32767	2
Unsigned int	0->65535	2
Short int	-32768->32767	2
Unsigigned Short	0-> 32767	2
Long Int	-2147483648->-2147483647	4
Unsigigned Long	0-> 4294967295	4

- Kiểu số thực

Một giá trị kiểu số thực là một phần tử của một tập các số thực mà máy tính có thể biểu diễn. Trong ngôn ngữ lập trình C có nhiều kiểu dữ liệu số thực với dải giá trị khác nhau cụ thể:

Kiểu	Phạm vi biểu diễn	Kích thước(byte)

Float	3.4E-38 -> 3.4E+38	4
Double	1.7E-311 -> 1.7E3+311	8
Long double	3.4E-4932->3.4E+4932	10

- Khai báo hằng, biến, mảng

+ Khai báo hằng

+ Hằng số thực

Được viết theo hai cách sau:

- Dạng thập phân gồm:Phần nguyên, dấu chấm thập phân, phần thập phân

Ví dụ:34.2 -344.122

- Dạng khoa học(dạng mũ) gồm: Phần định trị và phần mũ. Phần định trị là số nguyên hay số thực dạng thập phân, phần mũ bắt đầu bằng E hay e theo sau là số nguyên

Ví dụ: 1234.54E-122

+ Hằng số nguyên

- Hệ thập phân bình thường

VD: 545

- Hệ cơ số 8(Octal)

Bắt đầu bằng số 0 và chỉ biểu diễn số dương

Ví dụ: $024=20_{10}$

- Hệ cơ số 16(Hecxa)

Bắt đầu bằng 0x

Ví dụ: $0xAB = 163_{10}$

+ Hằng ký tự

Là một ký tự riêng biệt được đặt trong hai dấu nháy đơn

Ví dụ: ‘a’ ‘9’ .....

Chú ý: Hằng ký tự biểu thị mã của ký tự đó trong bảng mã ASCII. Do vậy một hằng ký tự cũng có thể tham gia vào các phép toán.

Ví dụ:

‘A’+10 có giá trị (65+10=75)

+ Hằng xâu ký tự

- Là một dãy các ký tự đặt trong hay dấu nháy “.....”

- Xâu ký được lưu trữ trong một mảng ô nhớ liền nhau song còn thêm ô nhớ cuối cùng chứa mã là 0(ký hiệu là ‘\0’)

Ví dụ: “Nguyen Van Anh”

+ **Cách khai báo một hằng**

Cách 1:#define Tenhang Giatri

Ví dụ: #define MAX 100

Cách 2: const kieu_du_kieu ten_hang=gia_tri_hang;

Ví dụ: const int n=20;

Sự khác nhau giữa định nghĩa hằng số dùng #define và const ở chỗ:

* Với const đây là hằng số cố định, một hằng số thực sự và chỉ có một hằng số chứa trong ô nhớ.

* Với #define khi gặp hằng số này chương trình dịch sẽ lắp giá trị hằng số này vào trong biểu thức cần tính với số lần thoải mái. Điều đó có nghĩa là mỗi khi gặp hằng này máy sẽ lắp đủ ô nhớ chứa hằng số này vào đó.

+ **Khai báo biến**

- Các biến trước khi sử dụng phải khai báo theo mẫu sau:

kieu_du_lieu danh_sach_cac_bien_can_khai_bao;

Ví dụ: int x,y;

float a;

- Khi khai báo một biến ta có thể khởi đầu giá trị cho nó theo mẫu sau:

kieu_du_lieu ten_bien = gia_tri;

Ví dụ: float x=5.;

int n=10;

- Để lấy địa chỉ của một biến ta dùng toán tử & cụ thể như sau:

&ten_bien

Ví dụ: &x lấy địa chỉ của biến a

&n lấy địa chỉ của biến n

+ Khai báo xâu ký tự.

char str[10]

+ Các phần tử của mảng là một ký tự

+ Xâu bao giờ cũng kết thúc bằng phần tử ký hiệu là NUL(‘\0’)

Một hằng xâu ký tự được đặt trong dấu nháy kép

VD: “DHSPKT” để lưu giữ xâu này thì hệ thống phải dùng 1 mảng có 7 ô nhớ.

D	H	S	P	K	T	\0

ký tự đơn ‘a’ a

xâu ký tự “a” 

VD: char ch[10]=”DHSPKT”

- Khai báo mảng

Mảng là một dãy biến liên tiếp cùng tên nhưng khác nhau bởi chỉ số. Tất cả các biến này có cùng một kiểu là kiểu của mảng.

+ Cách khái báo mảng

- Đối với mảng một chiều

kieu_du_lieu ten_mang[kich_thuc_mang];

- Đối với mảng hai chiều

kieu_du_lieu ten_mang[kich_thuc_hang][kich_thuoc_cot];

- Đối với mảng nhiều chiều

kieu_du_lieu ten_mang[kich_thuc_1][kich_thuoc_2]...[kich_thuoc_n];

Ví dụ:

int a[10];

float x[3][5];

char x[30];

+ Cách thức truy nhập các phần tử của mảng

Mỗi phần tử của mảng được truy nhập thông qua tên và chỉ số tương ứng, phần tử đầu tiên có chỉ số là 0.

Cách truy nhập

- Mảng một chiều: tenmang[chiso]
- Mang hai chiều: tenmang[chisodong][chisocot]

Ví dụ: m[0]

m[5]

- biến con trỏ

Ta có thể sử dụng tên con trỏ hoặc dạng khai báo của nó trong các biểu thức

Ví dụ:

float *px;

Ở đây: px là tên con trỏ

*px dạng khai báo của con trỏ

- Sử dụng tên con trỏ: Con trỏ cũng là một biến nên khi tên của nó xuất hiện trong các biểu thức thì giá trị của nó sẽ được sử dụng trong biểu thức này. Chỉ có một điều cần lưu ý ở đây: giá trị của một con trỏ là dịa chỉ của biến nào đó.

Ví dụ: float a,*p,*h;

p=&a;/* Gán địa chỉ của biến a cho p hay nói cách khác cho con trỏ p trỏ tới biến a */

h=p;/* Gán con trỏ p cho con trỏ h */

*p=5;// a=5

Các phép toán trên con trỏ

Có bốn nhóm phép toán liên quan đến con trỏ và địa chỉ: Phép gán, phép tăng giảm địa chỉ, phép truy nhập bộ nhớ và phép so sánh.

+ **Phép gán**

Ví dụ: int x,y,*trox,*troy;

char z;

trox=&x;

troy=&y;

trox=(int *)(&z); ép kiểu

+ **Phép tăng giảm địa chỉ**

Một con trỏ có thể cộng với một giá trị nguyên (int, long) để cho kết quả là một con trỏ cùng kiểu.

Ví dụ: int a[10], *tro1, *tro2, *tro3;

tro1=a; tương đương với tro1=a[0];

tro2=tro1+1;

tro3=tro1+9;

Cụ thể máy sẽ cung cấp các khoảng nhớ liên tiếp của mảng a như sau:

a[0]	a[1]								a[9]

tro1 ↑ tro2↑

tro3 ↑

+ **Hiệu hai con trỏ**

Hai con trỏ cùng kiểu trừ đi nhau cho ta một số nguyên

Ví dụ: float x[10],*trox,*troy;

int z;

trox=x+1; tương đương trox=&x[1]

troy=&x[5];

z=troy-trox;/* z có giá trị là 4 */

x[0]	x[1]								x[9]

Chú ý: Không được lấy tổng, hiệu, tích, thương, % hai con trỏ

- **Khối lệnh**

- Là một dãy các câu lệnh được bao bởi các dấu { và }

- Máy coi một khối lệnh tương tự như một lệnh riêng lẻ, chỗ nào viết được một lệnh riêng lẻ cũng có quyền đặt vào đó một khối lệnh. Việc bắt đầu một khối lệnh { và kết thúc một khối lệnh } tương tự như câu lệnh hợp thành trong Pascal sử dụng cặp từ khoá begin...end.

- Đầu mỗi khối lệnh có thể đặt các khai báo biến, mảng...

- Các khối lệnh có thể lồng nhau

- Các biến được khai báo trong khối lệnh nào thì chỉ có hiệu lực trong khối đó.

- Khi máy kết thúc phiên làm việc với khối lệnh nào thì tất cả các biến cục bộ bên trong khối lệnh đó đều bị giải phóng.

2.3 Biểu thức và Các phép toán

2.3.1 Phép toán số học hai ngôi

Các phép toán số học hai ngôi được thống kê ở bảng sau:

Phép toán	Ý nghĩa	Ví dụ
+	Phép cộng	2+4=6
-	Phép trừ	2-3=-1
*	Phép nhân	4*2=8
/	Phép chia	5/3=1
%	Phép lấy phần dư	6/2=0

Chú ý:

- Nếu phép chia hai toán hạng đều nguyên thì phép chia cho kết quả là phần nguyên của thương hai toán hạng đó.

- Nếu một trong hai toán hạng là kiểu thực thì lúc này kết quả của phép chia cho ta giá trị đúng.

- Phép toán lấy phần dư % chỉ áp dụng cho trường hợp hai toán hạng là số

nguyên.

2.3.2.Phép quan hệ và logic

Trong ngôn ngữ lập trình C coi mọi giá trị khác không là đúng(“TRUE”) và mọi giá trị bằng không là sai(“FALSE”)

Các phép toán quan hệ sau đây cho kết quả là 1 nếu điều khiện được thoả mãn và bằng 0 trong trường hợp ngược lại:

Phép toán quan hệ	Ý nghĩa	Ví dụ	Kết quả
>	Phép so sánh lớn hơn	1>2	0
>=	Phép so sánh lớn hơn hoặc bằng	2>=2	1
<	Phép so sánh nhỏ hơn	3<3	0
<=	Phép so sánh nhỏ hơn hoặc bằng	4<2	0
==	Phép so sánh bằng nhau	4==5	0
!=	Phép so sánh khác nhau	2!=7	1

Các phép toán logic được thể hiện dưới bảng sau:

Phép toán logic	Ý nghĩa	Ví dụ	Kết quả
!	Phép phủ định(not)	!(3>1)	0
&&	Phép và (and)	(2>1)&&(5=2)	0
||	Phép hoặc(or)	(4>3)||(1>8)	1

2.3.3. Sự chuyển đổi kiểu

Việc chuyển đổi kiểu dữ liểu trong C thường diễn ra tự động trong các trường hợp sau:

- Khi toán hạng trong một phép toán có kiểu khác nhau thì kiểu thấp hơn được chuyển thành kiểu cao hơn: int->long->float->double

- Khi gán một giá trị kiểu này cho một biến(hoặc phần tử mảng) kiểu kia.

Ví dụ: int c;

c=2.45;/* c sẽ nhận giá trị là 2*/

- Khi truyền giá trị cho các đối số của hàm, trong câu lênh return của hàm.

Ngoài ra ta có thể chuyển từ một kiểu giá trị này sang một kiểu giá trị khác bất kỳ ta muốn bằng cách ép kiểu theo mẫu sau:

(Kiểi_dữ_liệu)biểu_thức

Ví dụ:

```
float c=7.4;
int n;
```

n=(int)c*3;/* khi đó n có giá trị 21*/

2.3.4 Phép tăng giảm

Trong ngôn ngữ lập trình C đưa ra hai phép toán một ngôi để tăng và giảm các biến (nguyên và thực). Toán tử tăng ++ sẽ thêm 1 vào toán hạng của nó, toán tử giảm – sẽ trừ đi 1.

Ví dụ: n đang có giá trị là 5 thì

Sau phép toán ++ n có giá trị là 6

Sau phép toán – n có giá trị là 4

Dấu phép toán ++ và -- có thể đứng trước hoặc đứng sau toán hạng. Như vậy ta có thể viết: ++n, n++, --n, n--

Sự khác nhau của ++n và n++ ở chỗ: Trong phép toán n++ thì n tăng sau khi giá trị của nó được sử dụng, còn trong ++n thì giá trị của n tăng trước khi giá trị của nó được sử dụng. Trong phép toán n-- thì n giảm sau khi giá trị của nó được sử dụng, còn trong --n thì giá trị của n giảm trươc khi giá trị của nó được sử dụng.

Ví dụ:int x=2,y=4,n=4,m=5;

x+=n++;/* cho kết quả x có gía trị 6*/

y*=++m;/* cho kết quả y có giá trị 24*/

2.3.5 Câu lệnh gán

* Trong ngôn ngữ lập trình C dùng dấu “=” là dấu phép gán.

Ví dụ: a=a+3;

2.3.6. Biểu thức điều kiện

Biểu thức điều kiện có dạng: e1?e2:e3

Trong đó e1,e2,e3 là các biểu thức nào đó. Giá trị của biểu thức bằng e2 nếu e1 có giá trị khác không, giá trị của biểu thức bằng e3 nếu e1 có giá trị bằng không. Kiểu của biểu thức điều kiện là kiểu cao nhất giữa e2 và e3.

Ví dụ:int kq=3,x=5,y=2,z=1;

kq*=(x>y?x+z:y-z);/* cho kết quả kq có giá trị 18*/

2.4 Các toán tử điều khiển chương trình

2.4.1 Cấu trúc điều khiển if

2.4.1.2 Cấu trúc rẽ nhánh if dạng khuyết

Cú pháp câu lệnh

```
if (bt)
công_việc;
```

Trong đó:

- if là từ khoá
- bt là một biểu thức
- Công_việc có thể là một lệnh đơn hay một khối lệnh

2.4.1.2. Cấu trúc rẽ nhánh if dạng dầy đủ

Cú pháp câu lệnh

```
if (bt)
công_việc1;
else
công_việc2;
```

Trong đó:

- if, else là từ khoá
- bt là một biểu thức
- Công_việc1,Công_việc2 có thể là một lệnh đơn hay một khối lệnh

2.4.2 Cấu trúc điều khiển switch

Cú pháp câu lệnh

```
switch ( bieu_thuc)
                { case e1:Khối_lệnh_1;[break;]
                  case e2: Khối_lệnh_2;[break;]
                        .....................
                  case e2: Khối_lệnh_n;[break;]
                  [default: Khối_lệnh_n+1;]
                }
```

Trong đó: *switch, case, default là các từ khoá

* bieu_thuc: là một biểu thúc nguyên bất kỳ
* ei:là giá trị nguyên mà biểu thức có thể nhận được. Có thể là kiểu char

vì nó có thể được chuyển đổi thành kiểu int

* Những phần đặt trong hai dấu [và] có thể có hoặc không

2.4.3 Cấu trúc lặp while

Cú pháp câu lệnh

while(bt) Công_việc;

Trong đó:

- while là từ khoá
- bt là một biểu thức
- Công_việc có thể là một lệnh đơn hay một khối lệnh

2.4.4 Cấu trúc lặp do...while

Cú pháp câu lệnh

do

Công_việc;

while(bt);

Trong đó:

- while ,do là từ khoá
- bt là một biểu thức
- Công_việc liệt kê các câu lệnh cần phải thực hiện

2.4.5 Cấu trúc lặp for

Cú pháp câu lệnh

for(bt1;bt2;bt3)

Công_việc;

Trong đó:

- for là từ khoá
- bt1,bt2,bt3 là các biểu thức
- Công_việc có thể là một lệnh đơn hay một khối lệnh

2.5 Hàm, lập trình hướng hàm

2.5.1 Cách xây dựng một hàm:

Cấu trúc:

[kiểu_giá_trị_trả_về] tên_hàm([danh sách tham số]);

{

Các khai báo

.....

Các câu lệnh

}

Trong đó: *tên_hàm* là bất kỳ tên hợp lệ nào, *[kiểu_giá_trị_trả_về]* là kiểu dữ liệu của kết quả trả lại cho hàm gọi nó. *[danh sách tham số]* mô tả kiểu dữ liệu cùng thứ tự của các tham số hàm nhận được khi nó được gọi.

Các khai báo và các câu lệnh trong cặp dấu {} tạo thành phần thân của hàm(khối).

2.5.2 Sự hoạt động của một hàm

- Cấp phát bộ nhớ cho các đối và biến toàn cục
- Gán giá trị của các tham số thực sự cho các đối tương ứng.
- Thực hiện các câu lệnh trong thân hàm.
- Khi gặp câu lênh return hoặc dấu } cuối cùng của thân hàm thì máy sẽ xoá các đối và các biến cục bộ khỏi bộ nhớ và hàm kết thúc.
- Nếu hàm kết thúc bởi câu lệnh return có chứa biểu thức thì máy sẽ tính toán giá trị của biểu thức chuyển đổi kiểu phù hợp và gán cho tên hàm.

2.5.2.1 Biến mảng động

Các biến, mảng dược khai báo bên trong thân của một hàm gọi là biến, mảng tự động. Chúng chỉ có hiệu lực trong phạm vi hàm mà chúng được khai báo. Khi hàm kết thúc phiên làm việc thì chúng bị xoá khỏi bộ nhớ và trả lại ô nhớ cho máy.
Chú ý: Vì chương trình bắt đầu làm việc từ câu lệnh đầu tiên của hàm main() và kết thúc khi hàm này kết thúc. Do đó các biện tự động được khai báo bên trong hàm main() sẽ tồn tại trong suốt thời gian làm việc của chương trình.

2.5.2.2 Biến mảng ngoài

Là các biến, mảng được khai báo bên ngoài các hàm, chúng tồn tại trong suốt

thời gian làm việc của chương trình. Phạm vi sử dụng từ vị trí được khai báo đến cuối chương trình(kể cả trưởng hợp chương trình gồm nhiều tệp ghép nối bằng toán tử #include).

2.5.2.3 Biến mảng tĩnh

Cách khai báo

static khieu_du_lieu ten_bien;

Ví dụ: static int a,b,x;

Dòng khai báo có thể đặt ở trong(biến, mảng tĩnh trong) hay ngoài(biến, mảng tĩnh ngoài)

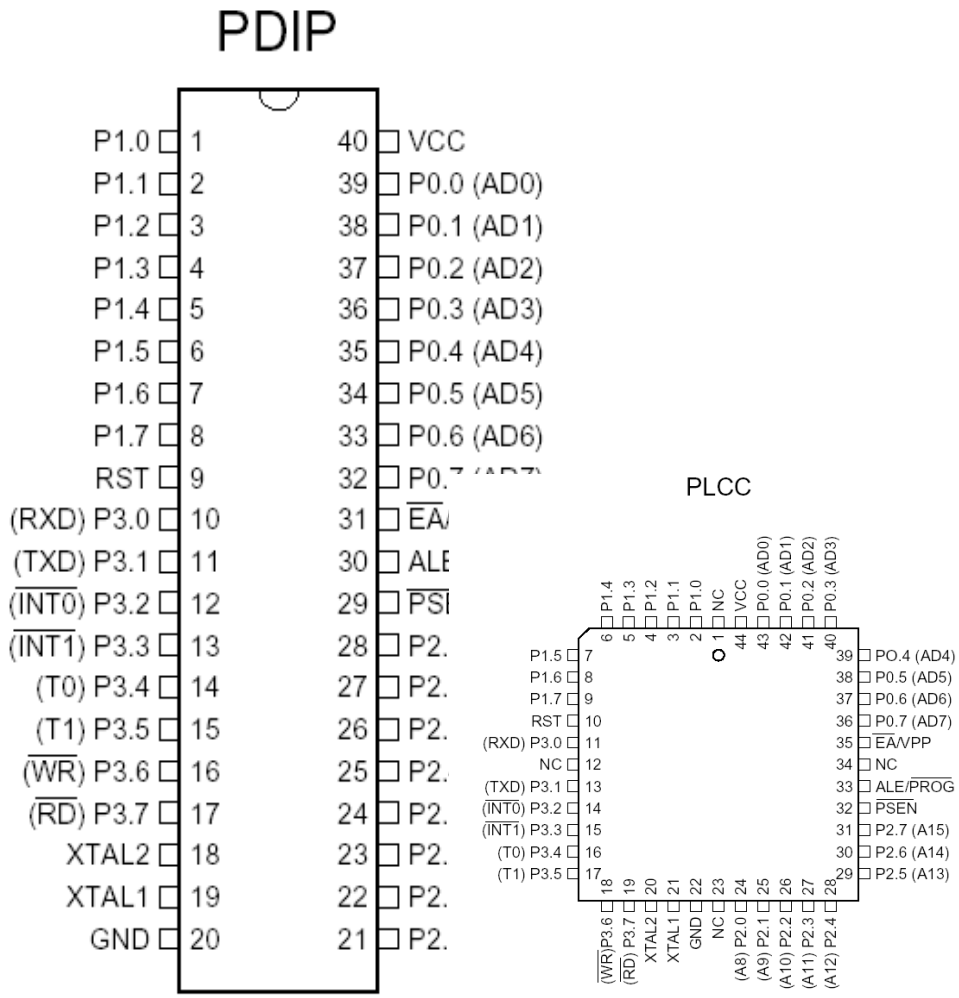
- Các biến, mảng tĩnh giống biến, mảng ngoài ở chỗ: Chúng đều tồn tại trong suốt thời gian làm việc của chương trình.
- Các biến, mảng tĩnh khác biến, mảng ngoài ở chỗ:

* Phạm vi hoạt động của biến, mảng tĩnh trong chỉ giới hạn bên trong hàm mà nó được khai báo. Tuy nhiên giá trị của nó vẫn được lưu giữ khi ra khỏi hàm và giá trị này có thể sử dụng mỗi khi hàm được thực hiện trở lại.

* Phạm vi hoạt động của biến, mảng tĩnh ngoài là từ vị trí khai báo đến cuối tệp và không bao gồm các tệp được kết nối bằng toán tử #include.

Chương 2: Ôn lại về vi điều khiển AT89C51

2.1. Sơ đồ chân tín hiệu của 80C51/AT89C51.



Chức năng của các chân tín hiệu như sau:

- P0.0 đến P0.7 là các chân của cổng 0.
- P1.0 đến P1.7 là các chân của cổng 1.
- P2.0 đến P2.7 là các chân của cổng 2
- P3.0 đến P3.7 là các chân của cổng 3
- RxD: Nhận tín hiệu kiểu nối tiếp.
- TxD: Truyền tín hiệu kiểu nối tiếp.
- /INT0: Ngắt ngoài 0.
- /INT1: Ngắt ngoài 1.
- T0: Chân vào 0 của bộ Timer/Counter 0.
- T1: Chân vào 1 của bộ Timer/Counter 1.
- /Wr: Ghi dữ liệu vào bộ nhớ ngoài.
- /Rd: Đọc dữ liệu từ bộ nhớ ngoài.
- RST: Chân vào Reset, tích cực ở mức logic cao trong khoảng 2 chu kỳ máy.

- XTAL1: Chân vào mạch khuyếch đại dao động
- XTAL2: Chân ra từ mạch khuyếch đại dao động.
- EA: Truy cập bộ nhớ ngoài.
- /PSEN : Chân cho phép đọc bộ nhớ chương trình ngoài (ROM ngoài).
- ALE (/PROG): Chân tín hiệu cho phép chốt địa chỉ để truy cập bộ nhớ ngoài, khi On-chip xuất ra byte thấp của địa chỉ. Tín hiệu chốt được kích hoạt ở mức cao, tần số xung chốt = 1/6 tần số dao động của bộ VĐK. Nó có thể được dùng cho các bộ Timer ngoài hoặc cho mục đích tạo xung Clock. Đây cũng là chân nhận xung vào để nạp chương trình cho Flash (hoặc EEPROM) bên trong On-chip khi nó ở mức thấp.
- /EA/Vpp: Cho phép On-chip truy cập bộ nhớ chương trình ngoài khi /EA=0, nếu /EA=1 thì On-chip sẽ làm việc với bộ nhớ chương trình nội trú (trường hợp cần truy cập vùng nhớ lớn hơn dung lượng bộ nhớ chương trình nội trú, thì bộ nhớ chương trình ngoài cũng được sử dụng). Khi chân này được cấp nguồn điện áp 12V (Vpp) thì On-chip đảm nhận chức năng nạp chương trình cho Flash bên trong nó.
- Vcc: Cung cấp dương nguồn cho On-chip (+ 5V).
- GND: nối Mass.

2.2. Sơ đồ khối

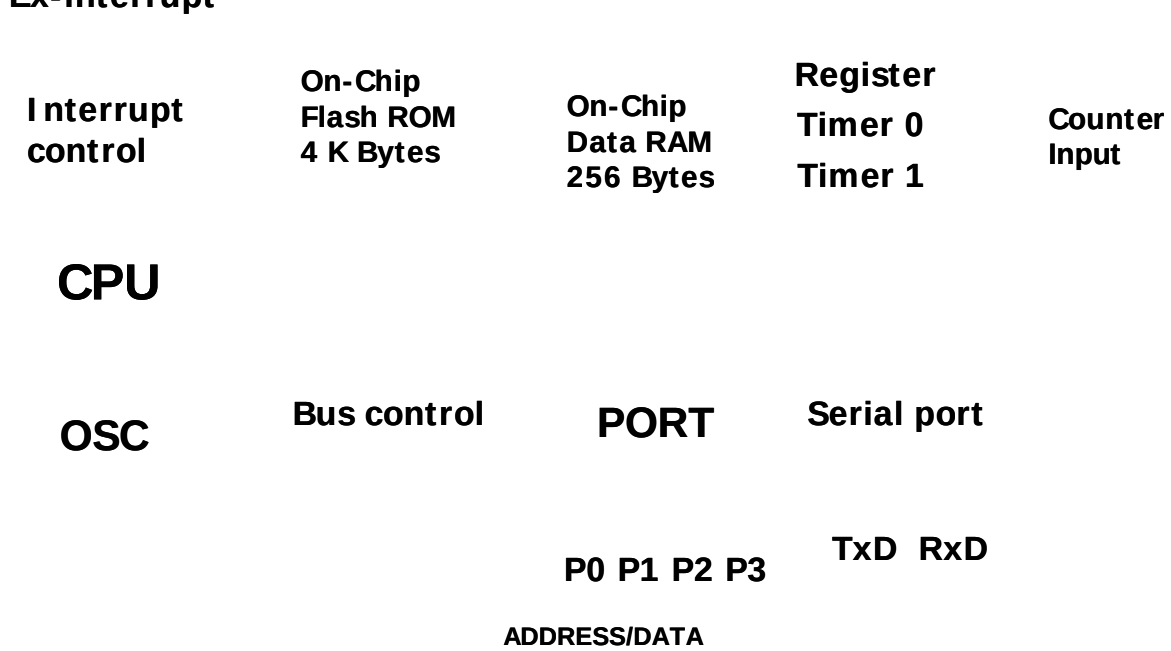
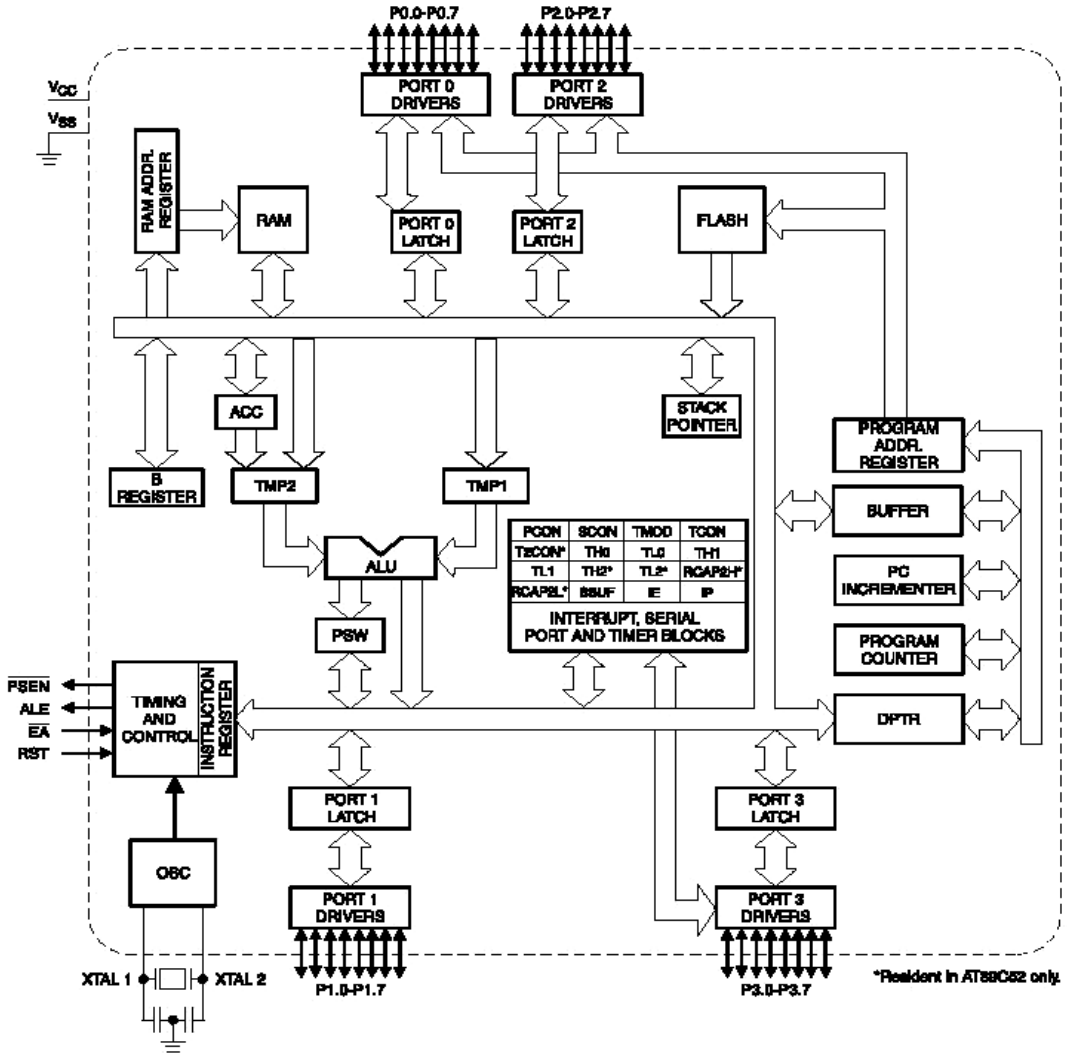


Figure 1. AT89 Series Flash-Based Microcontroller Architectural Block Diagram



Các thành phần chính:

2.3. Các thanh ghi chức năng đặc biệt.

SFR đảm nhiệm các chức năng khác nhau trong On-chip. Chúng nằm ở RAM bên trong On-chip, chiếm vùng không gian nhớ 128 Byte được định địa chỉ từ 80h đến FFh. Cấu trúc của SFR bao gồm các chức năng thể hiện ở bảng 2.3 và bảng 2.4.

Thanh ghi	**Nội dung**							
	MSB							*LSB*
IE	EA	-	ET2	ES	ET1	EX1	ET0	EX0
IP	-	-	PT2	PS	PT1	PX1	PT0	PX0
PSW	CY	AC	FO	RS1	RS0	OV	-	P
TMOD	GATE	C/(/T)	M1	M0	GATE	C/(/T)	M1	M0
TCON	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
SCON	SM0	SM1	SM2	REN	TB8	RB8	TI	RI
PCON	SMOD	-	-	-	GF1	GF0	PD	IDL
P1	T2	T2EX			/SS	MOSI	MISO	SCK
P3	RXD	TXD	/INT0	/INT1	T0	T1	/WR	/RD

Symbol	Name	Address	Reset Values
* ACC	Thanh ghi tích luỹ	0E0h	00000000b
* B	Thanh ghi B	0F0h	00000000b
* PSW	Từ trạng thái chương trình	0D0h	00000000b
SP	Con trỏ ngăn xếp	81h	00000111b
DP0L	Byte cao của con trỏ dữ liệu 0	82h	00000000b
DP0H	Byte thấp của con trỏ dữ liệu 0	83h	00000000b
* P0	Cổng 0	80h	11111111b
* P1	Cổng 1	90h	11111111b
Symbol	**Name**	**Address**	**Reset Values**
* P2	Cổng 2	0A0h	11111111b
* P3	Cổng 3	0B0h	11111111b
* IP	TG điều khiển ngắt ưu tiên	0B8h	xxx00000b
* IE	TG điều khiển cho phép ngắt	0A8h	0xx00000b
TMOD	Điều khiển kiểu Timer/Counter	89h	00000000b
* TCON	TG điều khiển Timer/Counter	88h	00000000b
TH0	Byte cao của Timer/Counter 0	8Ch	00000000b
TL0	Byte thấp của Timer/Counter 0	8Ah	00000000b
TH1	Byte cao của Timer/Counter 1	8Dh	00000000b
TL1	Byte thấp của Timer/Counter 1	8Bh	00000000b
* SCON	Serial Control	98h	00000000b
SBUF	Serial Data Buffer	99h	indeterminate
PCON	Power Control	87h	0xxx0000b

* : có thể định địa chỉ bit, x: không định nghĩa

Địa chỉ, ý nghĩa và giá trị của các SFR sau khi Reset

- **Thanh ghi ACC**: là thanh ghi tích luỹ, dùng để lưu trữ các toán hạng và kết quả của phép tính. Thanh ghi ACC dài 8 bits. Trong các tập lệnh của On-chip, nó thường được quy ước đơn giản là A.

- **Thanh ghi B :** Thanh ghi này được dùng khi thực hiện các phép toán nhân và chia. Đối với các lệnh khác, nó có thể xem như là thanh ghi đệm tạm thời. Thanh ghi B dài 8 bits. Nó thường được dùng chung với thanh ghi A trong các phép toán nhân hoặc chia.

- **Thanh ghi SP:** Thanh ghi con trỏ ngăn xếp dài 8 bit. SP chứa địa chỉ của dữ liệu hiện đang hiện hành ở đỉnh của ngăn xếp hay nói khác là SP luôn trỏ tới ngăn

nhớ sử dụng cuối cùng (gọi là đỉnh ngăn xếp). Giá trị của nó được tự động tăng lên khi thực hiện lệnh PUSH trước khi dữ liệu được lưu trữ trong ngăn xếp. SP sẽ tự động giảm xuống khi thực hiện lệnh POP.

- **Thanh ghi DPTR:** Thanh ghi con trỏ dữ liệu (16 bit) bao gồm 1 thanh ghi byte cao (DPH-8bit) và 1 thanh ghi byte thấp (DPL-8bit). DPTR có thể được dùng như thanh ghi 16 bit hoặc 2 thanh ghi 8 bit độc lập. Thanh ghi này được dùng để truy cập RAM ngoài.

- **Ports 0 to 3:** P0, P1, P2, P3 là các chốt của các cổng 0, 1, 2, 3 tương ứng. Mỗi chốt gồm 8 bit. Khi ghi mức logic 1 vào một bit của chốt, thì chân ra tương ứng của cổng ở mức logic cao. Còn khi ghi mức logic 0 vào mỗi bit của chốt thì chân ra tương ứng của cổng ở mức logic thấp. Khi các cổng đảm nhiệm chức năng như các đầu vào thì trạng thái bên ngoài của các chân cổng sẽ được giữ ở bit chốt tương ứng. Tất cả 4 cổng của on-chip đều là cổng I/O hai chiều, mỗi cổng đều có 8 chân ra, bên trong mỗi chốt bit có bộ "Pullup-tăng cường" do đó nâng cao khả năng nối ghép của cổng với tải (có thể giao tiếp với 4 đến 8 tải loại TTL).

- **Thanh ghi SBUF:** Đệm dữ liệu nối tiếp gồm 2 thanh ghi riêng biệt, một thanh ghi đệm phát và một thanh ghi đệm thu. Khi dữ liệu được chuyển tới SBUF, nó sẽ đi vào bộ đệm phát, và được giữ ở đấy để chế biến thành dạng truyền tin nối tiếp. Khi dữ liệu được truyền đi từ SBUF, nó sẽ đi ra từ bộ đệm thu.

- **Các Thanh ghi Timer:** Các đôi thanh ghi (TH0, TL0), (TH1, TL1) là các thanh ghi đếm 16 bit tương ứng với các bộ Timer/Counter 0 và 1.

- **Các thanh ghi điều khiển:** Các thanh ghi chức năng đặc biệt: IP, IE, TMOD, TCON, SCON, và PCON bao gồm các bit trạng thái và điều khiển đối với hệ thống ngắt, các bộ Timer/Counter và cổng nối tiếp. Chúng sẽ được mô tả ở phần sau.

- **Thanh ghi PSW:** Từ trạng thái chương trình dùng để chứa thông tin về trạng

thái chương trình. PSW có độ dài 8 bit, mỗi bit đảm nhiệm một chức năng cụ thể. Thanh ghi này cho phép truy cập ở dạng mức bit.

CY	AC	FO	RS1	RS0	OV	-	P

* CY: Cờ nhớ. Trong các phép toán số học, nếu có nhớ từ phép cộng bit 7 hoặc có số mượn mang đến bit 7 thì CY được đặt bằng 1.

* AC: Cờ nhớ phụ (Đối với mã BCD). Khi cộng các giá trị BCD, nếu có một số nhớ được tạo ra từ bit 3 chuyển sang bit 4 thì AC được đặt bằng 1. Khi giá trị được cộng là BCD, lệnh cộng phải được thực hiện tiếp theo bởi lệnh ***DA A*** (hiệu chỉnh thập phân thanh chứa A) để đưa các kết quả lớn hơn 9 về giá trị đúng.

* F0: Cờ 0 (Có hiệu lực với các mục đích chung của người sử dụng)

* RS1: Bit 1 điều khiển chọn băng thanh ghi.

* RS0: Bit 0 điều khiển chọn băng thanh ghi.

Lưu ý: RS0, RS1 được đặt/xoá bằng phần mềm để xác định băng thanh ghi đang hoạt động (Chọn băng thanh ghi bằng cách đặt trạng thái cho 2 bit này)

	RS1 (PSW. 4)	**RS0 (PSW. 3)**
Bank 0	0	0
Bank 1	0	1
Bank 2	1	0
Bank 3	1	1

Bảng Chọn băng thanh ghi

* OV: Cờ tràn. Khi thực hiện các phép toán cộng hoặc trừ mà xuất hiện một tràn số học, thì OV được đặt bằng 1. Khi các số có dấu được cộng hoặc được trừ, phần mềm có thể kiểm tra OV để xác định xem kết quả có nằm trong tầm hay không. Với phép cộng các số không dấu, OV được bỏ qua. Kết quả lớn hơn +128 hoặc nhỏ hơn -127 sẽ đặt OV=1.

* -: Bit dành cho người sử dụng tự định nghĩa(Nếu cần).

* P: Cờ chẵn lẻ. Được tự động đặt/ xoá bằng phần cứng trong mỗi chu trình lệnh để chỉ thị số chẵn hay lẻ của bit 1 trong thanh ghi tích luỹ. Số các bit 1 trong A cộng với bit P luôn luôn là số chẵn.

- **Thanh ghi PCON**: Thanh ghi điều khiển nguồn.

SMOD	-	-	-	GF1	GF0	PD	IDL

* SMOD: Bit tạo tốc độ Baud gấp đôi. Nếu Timer 1 được sử dụng để tạo tốc độ baud và SMOD=1, thì tốc độ Baud được tăng lên gấp đôi khi cổng truyền tin nối tiếp được dùng bởi các kiểu 1, 2 hoặc 3.

* -: Không sử dụng, các bit này có thể được dùng ở các bộ VXL trong tương lai. Người sử dụng không được phép tự định nghĩa cho các bit này.

* GF0, GF1: Cờ dùng cho các mục đích chung (đa mục đích).

* PD: bit nguồn giảm. Đặt bit này ở mức tích cực để vận hành chế độ nguồn giảm trong AT89C51. Chỉ có thể ra khỏi chế độ bằng Reset.

* IDL: bit chọn chế độ nghỉ. Đặt bit này ở mức tích cực để vận hành kiểu Idle (Chế độ không làm việc) trong AT89C51.

Lưu ý: Nếu PD và IDL cùng được kích hoạt cùng 1 lúc ở mức tích cực, thì PD được ưu tiên thực hiện trước. Chỉ ra khỏi chế độ bằng 1 ngắt hoặc Reset lại hệ thống.

- **Thanh ghi IE:** Thanh ghi cho phép ngắt

EA	-	ET2	ES	ET1	EX1	ET0	EX0

* EA: Nếu EA=0, không cho phép bất cứ ngắt nào hoạt động. Nếu EA=1, mỗi nguồn ngắt riêng biệt được phép hoặc không được phép hoạt động bằng cách đặt hoặc xoá bit Enable của nó.

* -: Không dùng, người sử dụng không nên định nghĩa cho Bit này, bởi vì nó có thể được dùng ở các bộ AT89 trong tương lai.

* ET2: Bit cho phép hoặc không cho phép ngắt bộ Timer 2.

* ES: Bit cho phép hoặc không cho phép ngắt cổng nối tiếp (SPI và UART).

* ET1: Bit cho phép hoặc không cho phép ngắt tràn bộ Timer 1

* EX1: Bit cho phép hoặc không cho phép ngắt ngoài 1.

* ET0: Bit cho phép hoặc không cho phép ngắt tràn bộ Timer 0

* EX0: Bit cho phép hoặc không cho phép ngắt ngoài 0.

- **Thanh ghi IP**: Thanh ghi ưu tiên ngắt**.**

-	-	PT2	PS	PT1	PX1	PT0	PX0

* - : Không dùng, người sử dụng không nên ghi “1” vào các Bit này.

* PT2: Xác định mức ưu tiên của ngắt Timer 2.

* PS: Định nghĩa mức ưu tiên của ngắt cổng nối tiếp.

* PT1: Định nghĩa mức ưu tiên của ngắt Timer 1.

* PX1: Định nghĩa mức ưu tiên của ngắt ngoàI 1.

* PT0: Định nghĩa mức ưu tiên của ngắt Timer 0.

* PX0: Định nghĩa mức ưu tiên của ngắt ngoàI 0.

- **Thanh ghi TCON** : Thanh ghi điều khiển bộ Timer/Counter

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

* TF1: Cờ tràn Timer 1. Được đặt bởi phần cứng khi bộ Timer 1 tràn. Được xoá bởi phần cứng khi bộ vi xử lý hướng tới chương trình con phục vụ ngắt.

* TR1: Bit điều khiển bộ Timer 1 hoạt động. Được đặt/xoá bởi phần mềm để điều khiển bộ Timer 1 ON/OFF

* TF0: Cờ tràn Timer 0. Được đặt bởi phần cứng khi bộ Timer 0 tràn. Được xoá bởi phần cứng khi bộ vi xử lý hướng tới chương trình con phục vụ ngắt.

* TR0: Bit điều khiển bộ Timer 0 hoạt động. Được đặt/xoá bởi phần mềm để điều khiển bộ Timer 0 ON/OFF.

* IE1: Cờ ngắt ngoài 1. Được đặt bởi phần cứng khi sườn xung của ngắt ngoài 1 được phát hiện. Được xoá bởi phần cứng khi ngắt được xử lý.

* IT1: Bit điều khiển ngắt 1 để tạo ra ngắt ngoài. Được đặt/xoá bởi phần mềm.

* IE0: Cờ ngắt ngoài 0. Được đặt bởi phần cứng khi sườn xung của ngắt ngoài 0 được phát hiện. Được xoá bởi phần cứng khi ngắt được xử lý.

* IT0: Bit điều khiển ngắt 0 để tạo ra ngắt ngoài. Được đặt/xoá bởi phần mềm.

- **Thanh ghi TMOD:** Thanh ghi điều khiển kiểu Timer/Counter

GATE	C/(/T)	M1	M0	GATE	C/(/T)	M1	M0
Dành cho Timer 1				Dành cho Timer 0			

* GATE: Khi GATE=1 và TRx =1, bộ TIMER/COUTERx hoạt động chỉ khi chân INTx ở mức cao. Khi GATE=0, bộ TIMER/COUNTERx sẽ hoạt động chỉ khi

TRx=1.

* C/(/T): Bit này cho phép chọn chức năng là Timer hay Counter.

- Bit này =0 thì thực hiện chức năng Timer

- Bit này =1 thì thực hiện chức năng Counter

* M0, M1: Bit chọn Mode, để xác định trạng thái và kiểu Timer/Counter:

- M1=0, M0=0: Chọn kiểu bộ Timer 13 bit. Trong đó THx dài 8 bit, còn TLx dài 5 bit.

- M1=0, M0=1: Chọn kiểu bộ Timer 16 bit. THx và TLx dài 16 bit được ghép tầng.

- M1=1, M0=0: 8 bit Auto reload. Các thanh ghi tự động nạp lại mỗi khi bị tràn. Khi bộ Timer bị tràn, THx dài 8 bit được giữ nguyên giá trị, còn giá trị nạp lại được đưa vào TLx.

- M1=1, M0=1: Kiểu phân chia bộ Timer. TL0 là 1 bộ Timer/Counter 8 bit, được điều khiển bằng các bit điều khiển bộ Timer 0, Còn TH0 chỉ là bộ Timer 8 bit, được điều khiển bằng các bit điều khiển Timer 1.

- M1=1, M0=1: Timer/Counter 1 Stopped

- Thanh ghi SCON:

SM0	SM1	SM2	REN	TB8	RB8	TI	RI

SCON là thanh ghi trạng thái và điều khiển cổng nối tiếp. Nó không những chứa các bit chọn chế độ, mà còn chứa bit dữ liệu thứ 9 dành cho việc truyền và nhận tin (TB8 và RB8) và chứa các bit ngắt cổng nối tiếp.

* SM0, SM1: Là các bit cho phép chọn chế độ cho cổng truyền nối tiếp.

SM0	**SM1**	**Mode**	**Đặc điểm**	**Tốc độ Baud**
0	0	0	Thanh ghi dịch	$F_{osc}/12$
0	1	1	8 bit UART	Có thể thay đổi (được đặt bởi bộ Timer)
1	0	2	9 bit UART	$F_{osc}/64$ hoặc $F_{osc}/32$
1	1	3	9 bit UART	Có thể thay đổi (được đặt bởi bộ Timer)

Bảng 2.6. Chọn Mode trong SCON

* SM2: Cho phép truyền tin đa xử lý, thể hiện ở Mode 2 và 3. ở chế độ 2 hoặc 3, nếu đặt SM2 = 1 thì RI sẽ không được kích hoạt nếu bit dữ liệu thứ 9 (RB8) nhận được giá trị bằng 0. ở Mode 1, nếu SM2=1 thì RI sẽ không được kích hoạt nếu bit dừng có hiệu lực đã không được nhận. ở chế độ 0, SM2 nên bằng 0

* REN: Cho phép nhận nối tiếp. Được đặt hoặc xoá bởi phần mềm để cho phép hoặc không cho phép nhận.
* TB8: Là bit dữ liệu thứ 9 mà sẽ được truyền ở Mode 2 và 3. Được đặt hoặc xoá bởi phần mềm.
* RB8: Là bit dữ liệu thứ 9 đã được nhận ở Mode 2 và 3. ở Mode 1, nếu SM2=0 thì RB8 là bit dừng đã được nhận. ở Mode 0, RB8 không được sử dụng.
* TI: Cờ ngắt truyền. Được đặt bởi phần cứng tại cuối thời điểm của bit thứ 8 trong Mode 0, hoặc đầu thời điểm của bit dừng trong các Mode khác. ở bất kỳ quá trình truyền nối tiếp nào, nó cũng phải được xoá bằng phần mềm.
* RI: Cờ ngắt nhận. Được đặt bởi phần cứng tại cuối thời điểm của bit thứ 8 trong Mode 0, hoặc ở giữa thời điểm của bit dừng trong các Mode khác. ở bất kỳ quá trình nhận nối tiếp nào (trừ trường hợp ngoại lệ, xem SM2), nó cũng phải được xoá bằng phần mềm.

2.4. Khối tạo thời gian và bộ đếm (Timer/Counter).

On-chip AT89C51 có 2 thanh ghi Timer/Counter dài 16 bit, đó là: Timer 0 và Timer 1. Trong On-chip AT89C52, ngoài Timer 0 và Timer 1 nó còn có thêm bộ Timer 2. Cả 3 bộ Timer này đều có thể được điều khiển để thực hiện chức năng thời gian hay bộ đếm, thông qua thanh ghi TMOD.

Khi thanh ghi Timer/Counter làm việc ở kiểu Timer, thì sau mỗi chu kỳ máy nội dung trong thanh ghi được gia tăng thêm 1 đơn vị. Vì vậy thanh ghi này đếm số chu kỳ máy. Một chu kỳ máy có 12 chu kỳ dao động, do đó tốc độ đếm của thanh ghi là 1/12 tần số dao động.

Khi thanh ghi Timer/Counter làm việc ở kiểu Counter, xung nhịp bên ngoài được đưa vào để đếm ở T0 hoặc T1. Nội dung thanh ghi được tăng lên khi có sự chuyển trạng thái từ 1 về 0 tại chân đầu vào ngoài T0 hoặc T1.

Do xung nhịp bên ngoài có tần số bất kỳ nên các bộ Timer (0 và 1) có 4 chế độ làm việc khác nhau để lựu chọn: (13 bit Timer, 16 bit Timer, 8 bit auto-reload(tự lặp lai), split Timer(định thời chia tách)).

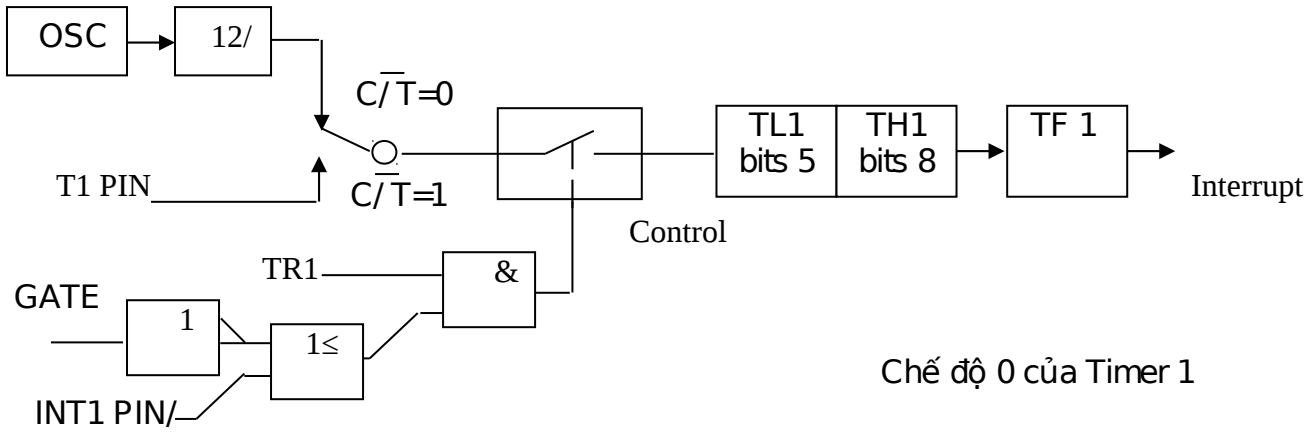
Timer 0 và Timer 1:

Trong AT89C51 và AT89C52 đều có các bộ Timer 0 và 1. Chức năng Timer hay Counter được chọn lựa bởi các bit điều khiển C/(/T) trong thanh ghi TMOD. Hai bộ

Timer/Counter này có 4 chế độ hoạt động, được lựa chọn bởi cặp bit (M0, M1) trong TMOD. Chế độ 0, 1 và 2 giống nhau cho các chức năng Timer/Counter, nhưng chế độ 3 thì khác. Bốn chế độ hoạt động được mô tả như sau:

+ ***Chế độ 0***: Cả 2 bộ Timer 0 và 1 ở chế độ 0 có cấu hình như một thanh ghi 13 bit, bao gồm 8 bit của thanh ghi THx và 5 bit thấp của TLx. 3 bit cao của TLx không xác định chắc chắn, nên được làm ngơ. Khi thanh ghi được xoá về 0, thì cờ ngắt thời gian TFx được thiết lập. Bộ Timer/Counter hoạt động khi bit điều khiển TRx được thiết lập (TRx=1) và, hoặc Gate trong TMOD bằng 0, hoặc /INTx=1. Nếu đặt GATE=1 thì cho phép điều khiển Timer/ Counter bằng đường vào ngoài /INTx, để dễ dàng xác định độ rộng xung.

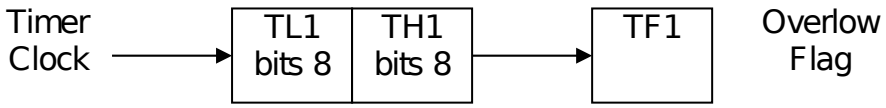
Khi hoạt động ở chức năng thời gian thì bit C/(/T)=0, do vậy xung nhịp từ bộ dao động nội, qua bộ chia tần cho ra tần số $f=f_{osc}/12$ được đưa vào để đếm trong



Chế độ 0 của Timer 1

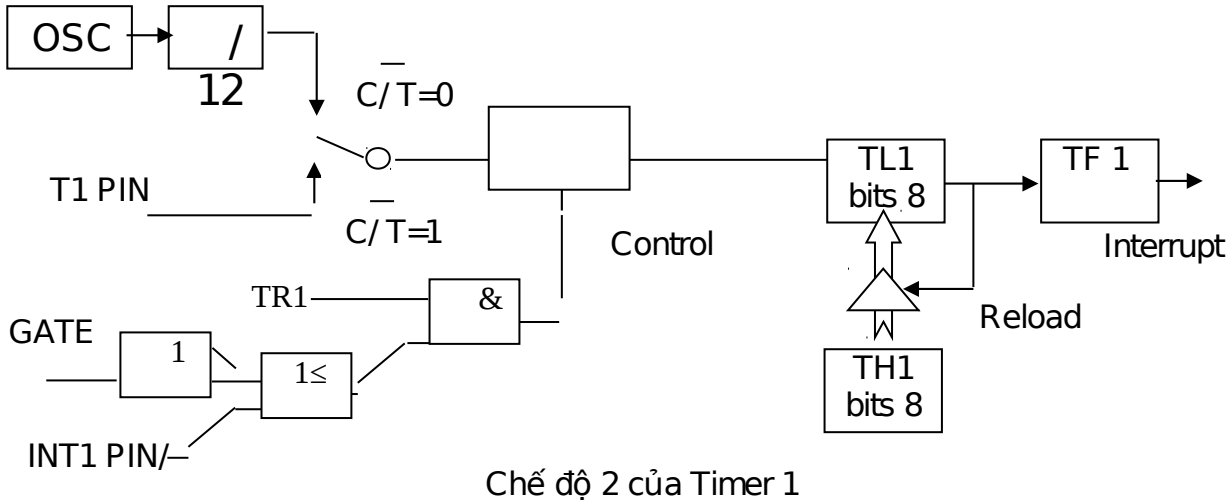
thanh ghi Timer/Counter. Khi hoạt động ở chức năng bộ đếm thì bit C/(/T)=1, lúc đó xung nhịp ngoài đưa vào sẽ được đếm.

+ ***Chế độ 1***: hoạt động tương tự như chế độ 0, chỉ khác là thanh ghi Timer/Counter được sử dụng cả 16 bit. Xung nhịp được dùng kết hợp với các thanh ghi thời gian byte thấp và byte cao (TH1 và TL1). Khi xung Clock được nhận, bộ Timer sẽ đếm tăng lên: 0000h, 0001h, 0002, Khi hiện tượng tràn xẩy ra, cờ tràn sẽ chuyển FFFFh về 0000h, và bộ Timer tiếp tục đếm. Cờ tràn của Timer 1 là bit TF1 ở trong TCON, nó được đọc hoặc ghi bởi phần mềm, xem hình 2.5 (Timer/Counter 1 Mode 1: 16 bit Counter).



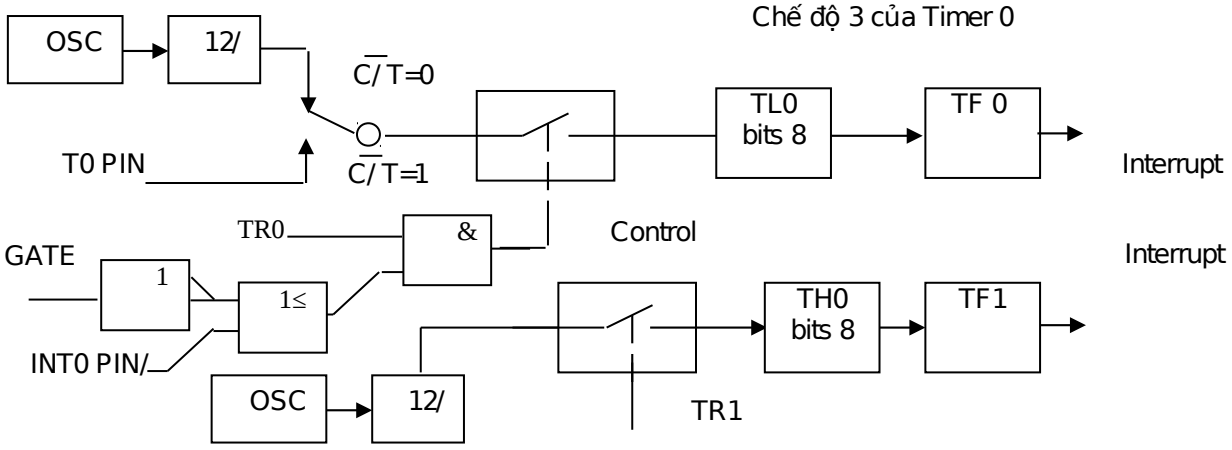
Chế độ 1 của Timer 1

+ ***Chế độ 2***: Chế độ này của thanh ghi Timer cũng hoạt động tương tự như 2 chế độ trên, nhưng nó được tổ chức như bộ đếm 8 bit (TL1) với chế độ tự động nạp lại, như hình 2.6. Khi xẩy ra hiện tượng tràn ở TL1, không chỉ thiết lập bit TF1 mà còn tự động nạp lại cho TL1 bằng nội dung của TH1, đã được thiết lập bởi phần mềm. Quá trình nạp lại cho phép nội dung của TH1 không bị thay đổi. Chế độ 2 của Timer/Counter 0 cũng tương tự như Timer/Counter 1.



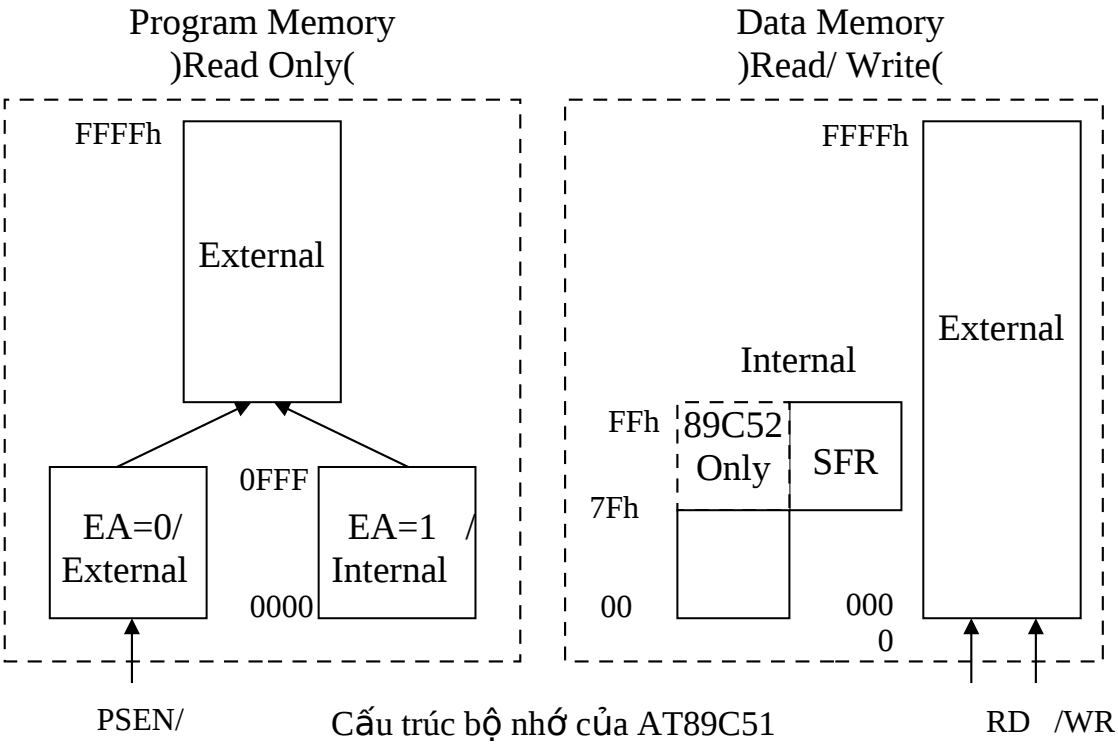
Chế độ 2 của Timer 1

+ ***Chế độ 3***: ở chế độ này, chức năng Timer/Counter 0 và chức năng Timer/Counter 1 khác nhau. Bộ Timer 1 ở chế độ 3 chỉ chứa chức năng đếm của nó, kết quả giống khi đặt TR1=0. Bộ Timer 0 ở chế độ 3 thiết lập TH0, TL0 như là 2 bộ đếm riêng biệt. Mạch Logic đối với chế độ 3 của Timer 0 thể hiện ở hình 2.7. Bộ đếm TL0 được điều khiển bởi các bit: C/(/T), GATE, TR0, /INT0 và khi đếm tràn nó thiết lập cờ ngắt TF0. Bộ đếm TH0 chỉ được điều khiển bởi bit TR1, và khi đếm tràn nó thiết lập cờ ngắt TF1. Vậy, TH0 điều khiển ngắt Timer/Counter 1.



2.5. Bộ nhớ chương trình và bộ nhớ dữ liệu nội trú.

Tất cả các bộ Flash Microcontrollers của Atmel đều tổ chức các vùng địa chỉ tách biệt đối với bộ nhớ chương trình và bộ nhớ dữ liệu, được mô tả ở hình dưới đây. Các vùng nhớ chương trình và dữ liệu tách biệt cho phép bộ nhớ dữ liệu được truy cập bởi địa chỉ 8 bit, có thể được lưu trữ với tốc độ cao và được vận hành bởi một bộ CPU 8 bit. Tuy nhiên, địa chỉ bộ nhớ dữ liệu 16 bit cũng có thể được tạo ra thông qua thanh ghi con trỏ dữ liệu (DPTR).



Cấu trúc bộ nhớ của AT89C51

- **Bộ nhớ chương trình nội trú.**

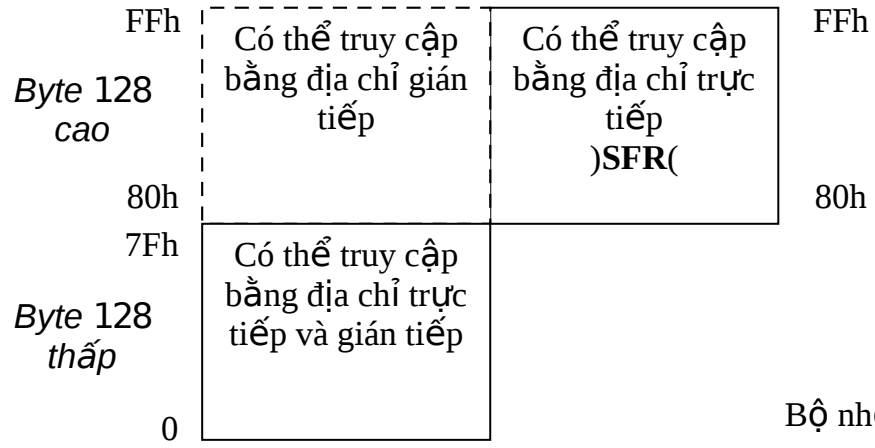
Bộ nhớ chương trình của AT89C51 được tổ chức như thể hiện ở hình trên. Không gian nhớ cực đại của bộ nhớ này chiếm 64 Kbyte, được định địa chỉ từ 0000h đến FFFFh, trong đó có 4 Kbyte Flash nội trú bên trong nó và được định địa chỉ từ 0000h đến 0FFFh. Do đó có thể mở rộng thêm 60 Kbyte bộ nhớ chương trình bên ngoài, được định địa chỉ từ 1000h đến FFFFh. Tuy nhiên bộ VĐK này cũng có thể sử dung toàn bộ bộ nhớ chương trình ngoài bao gồm 64 Kbyte được định địa chỉ từ 0000h đến FFFFh.

Cũng từ hình trên ta thấy, thông qua việc chọn mức logic cho bit /EA có thể lựa chọn để truy cập bộ nhớ chương trình nội trú (4Kb), bộ nhớ chương trình mở rộng ngoại trú (60Kb), hoặc toàn bộ bộ nhớ chương trình ngoại trú bên ngoài On-

chip (64Kb). Cụ thể, khi /EA = 1 thì bộ VĐK sử dụng cả bộ nhớ chương trình nội trú và ngoại trú. Ngược lại, khi /EA = 0 thì bộ VĐK chỉ sử dụng bộ nhớ chương trình ngoại trú.

Mỗi khi được Reset, bộ VĐK sẽ truy cập bộ nhớ chương trình tại địa chỉ khởi đầu là 0000h, sau đó nếu cơ chế ngắt được sử dụng thì nó sẽ truy cập tới địa chỉ quy định trong bảng vecter ngắt.

- Bộ nhớ dữ liệu nội trú.



Bộ nhớ dữ liệu trong AT89C51 có bộ nhớ dữ liệu chiếm một khoảng không gian bộ nhớ độc lập với bộ nhớ chương trình. Dung lượng của RAM nội trú ở họ VĐK này là 128 Byte, được định địa chỉ từ 00h đến 7Fh. Phạm vi địa chỉ từ 80h đến FFh dành cho SFR. Tuy nhiên bộ VĐK cũng có thể làm việc với RAM ngoại trú có dung lượng cực đại là 64 Kbyte được định địa chỉ từ 0000h đến FFFFh.

- Vùng nhớ 128 Byte thấp

Vùng nhớ 128 Byte thấp được định địa chỉ từ 00h đến 7Fh, được chia thành 3 vùng con như thể hiện ở hình 2.10.

- Vùng thứ nhất có độ lớn 32 Byte được định địa chỉ từ 00h đến 1Fh bao gồm 4 băng thanh ghi (băng 0...băng 3), mỗi băng có 8 thanh ghi 8 bit. Các thanh ghi trong mỗi băng có tên gọi từ R0 đến R7. Vùng RAM này được truy cập bằng địa chỉ trực tiếp mức Byte, và quá trình chọn để sử dung băng thanh ghi nào là tùy thuộc vào việc lựa chon giá trị cho RS1 và RS0 trong PSW.
- Vùng thứ 2 có độ lớn 16 Byte được định địa chỉ từ 20h đến 2Fh, cho phép truy cập trực tiếp bằng địa chỉ mức bit. Bộ VĐK cung cấp các lệnh có khả năng truy cập tới

vùng nhớ 128 bit này (nếu truy cập ở dạng mức bit thì vùng này có địa chỉ được định từ 00h đến 7Fh) ở mức bit..

- Vùng nhớ còn lại gồm 80 Byte có địa chỉ từ 30h đến 7Fh được dành riêng cho người sử dụng để lưu trữ dữ liệu. Đây có thể xem là vùng RAM đa mục đích. Có thể truy cập vùng nhớ này bằng địa chỉ trực tiếp hoặc gián tiếp thông qua các thanh ghi (R0 hoặc R1) ở dạng mức Byte.

- Vùng nhớ 128 Byte cao (dành cho SFR)

Vùng nhớ này được định địa chỉ từ 80h đến FFh, và được truy cập bằng địa chỉ trực tiếp.

Bộ nhớ dữ liệu RAM (Data Menmory)

Có thể chọn bank
bằng bit RS1,RS0
trong thanh ghi PSW

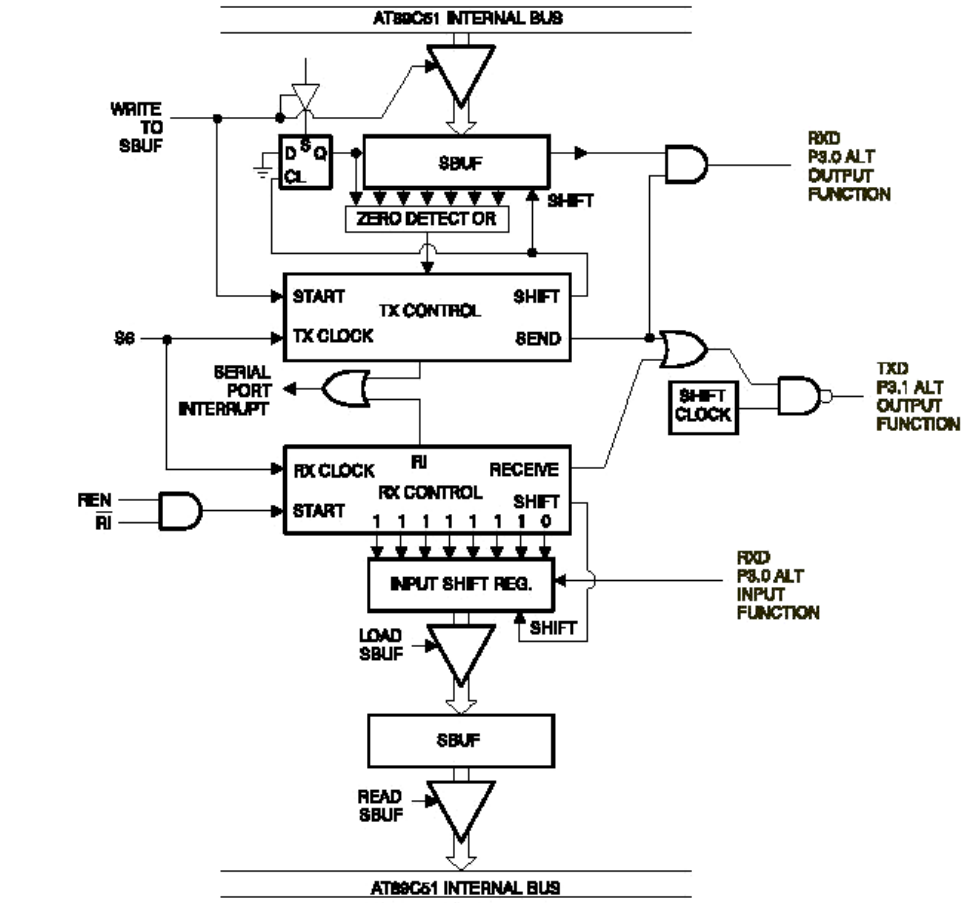
2.6. Nguyên lý truyền tin nối tiếp của AT89C51.

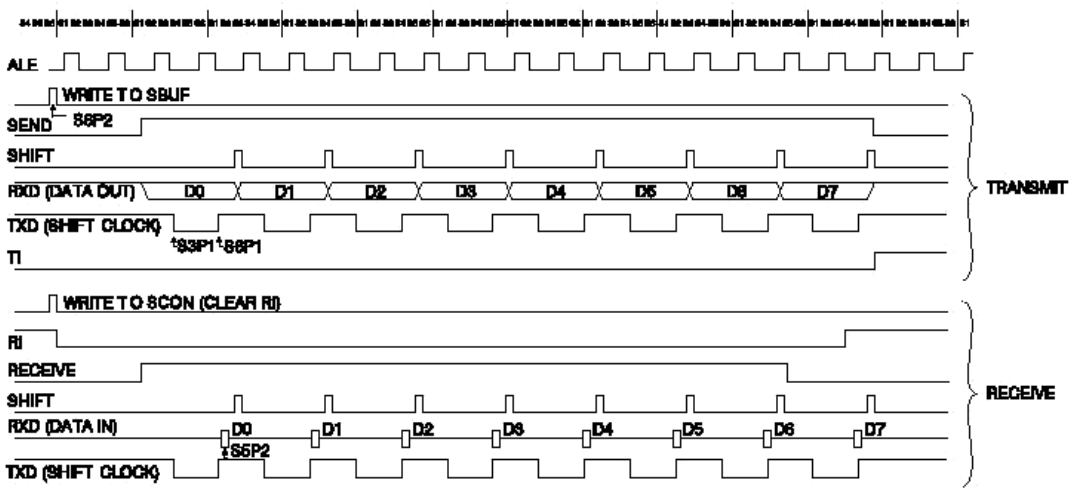
- Phương thức truyền tin nối tiếp(Serial Interface):

Hệ VXL on-chip này truyền tin nối tiếp bằng cổng RxD và TxD, dữ liệu xuất

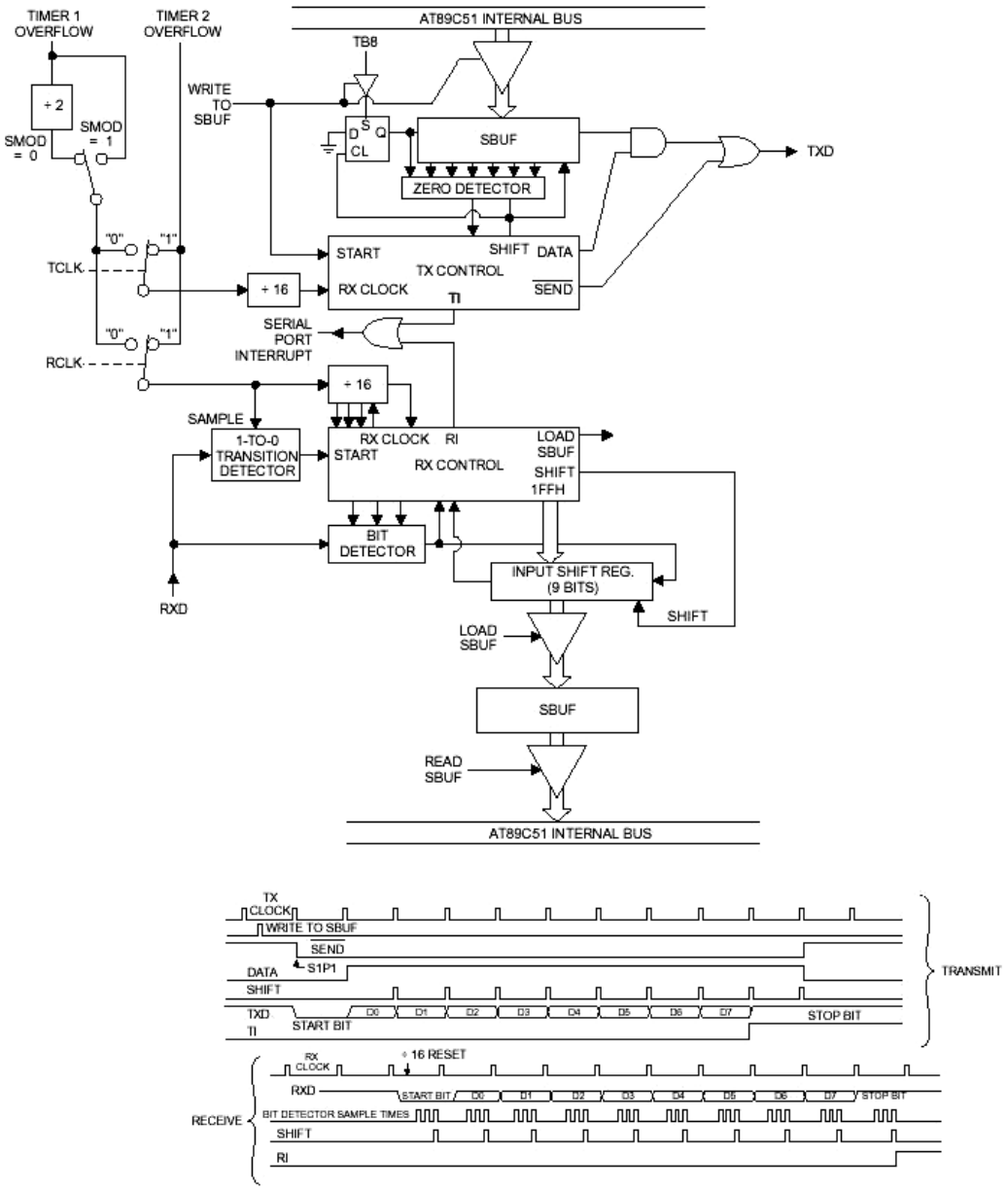
nhập truyền qua cổng nối tiếp bằng tốc độ Baud và đều qua vùng đệm nối tiếp SBUF. Cổng truyền nối tiếp là cổng truyền tin 2 chiều, nghĩa là nó có thể đồng thời truyền và nhận thông tin cùng 1 lúc. Nó cũng có khả năng vừa thực hiện chức năng nhận vừa thực hiện chức năng đệm, tức là nó có thể nhận byte kế tiếp trước khi byte được nhận trước đó được đọc từ thanh ghi đệm. (Tuy nhiên, nếu byte đầu tiên vẫn chưa được đọc tại thời điểm nhận của byte thứ 2, thì một trong 2 byte này sẽ bị mất). Điều khiển cổng nối tiếp bằng thanh ghi SCON, trạng thái của 2 bit SM0 và SM1 trong thanh ghi này thiết lập nên 4 chế độ hoạt động giao tiếp nối tiếp chuẩn như sau:

+ ***Chế độ 0***: Dữ liệu nối tiếp vào và ra sẽ thông qua chân RxD. Chân TxD đưa ra xung nhịp đồng hồ. 8 bit dữ liệu được truyền/nhận nối tiếp, với bit LSB được thực hiện đầu tiên. Tốc độ Baud được cố định bằng 1/12 tần số của bộ dao động.

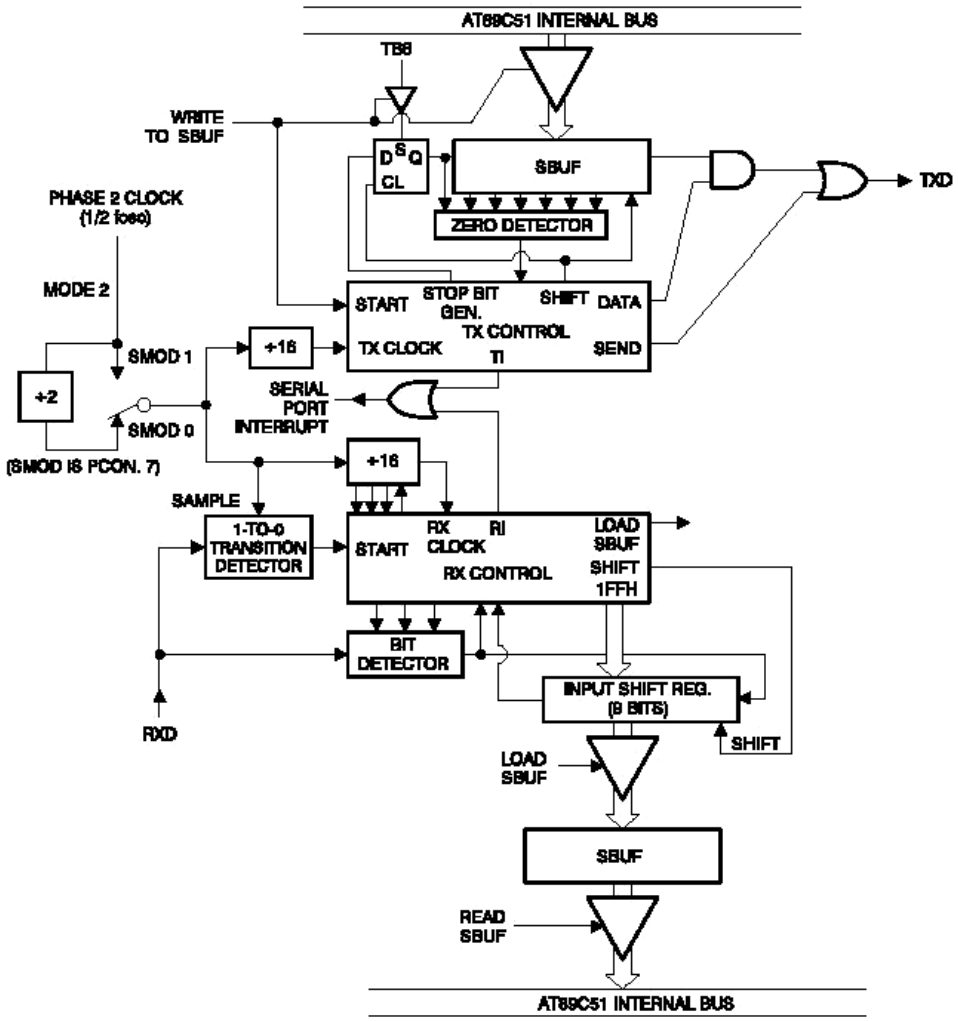




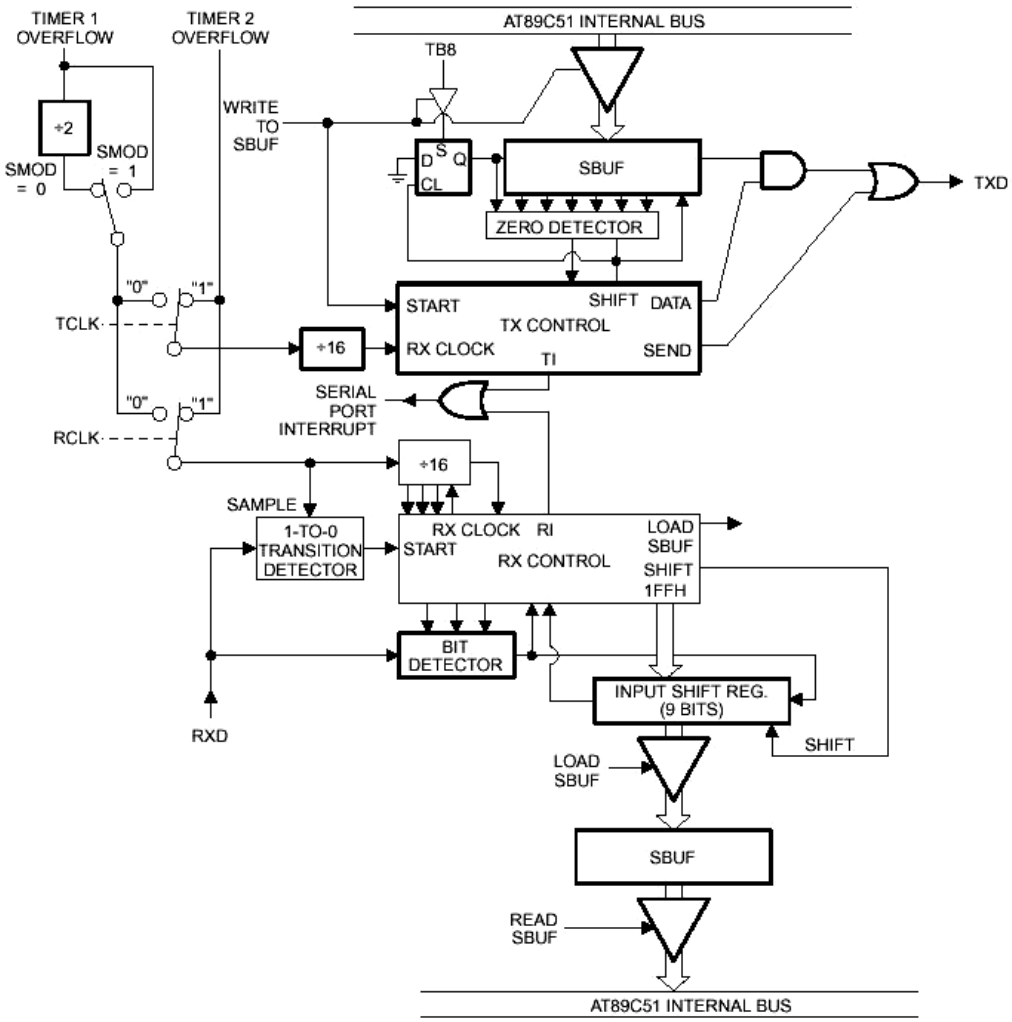
+ ***Chế độ 1***: 10 bit được truyền (thông qua TxD) hoặc nhận (thông qua RxD), trong đó gồm có: 1 bit khởi động (có giá trị 0), 8 bit dữ liệu (đầu tiên là LSB), và 1 bit dừng (có giá trị là 1). Khi nhận, bit dừng được chuyển vào RB8 của thanh ghi SCON. Tốc độ Baud có thể thay đổi được.



+ ***Chế độ 2***: 11 bit được truyền (thông qua TxD) hoặc nhận (thông qua RxD) bao gồm: bit khởi động (có giá trị 0), 8 bit dữ liệu (đầu tiên là LSB), một bit dữ liệu thứ 9 có thể lập trình được, và một bit dừng (có giá trị 1). Khi truyền, bit dữ liệu thứ 9 (TB8 ở trong SCON) có thể được gán giá trị 0 hoặc 1. Chẳng hạn như bit chẵn lẻ (P ở trong PSW) có thể được chuyển vào TB8. Khi nhận, bit dữ liệu thứ 9 được chuyển vào RB8 ở thanh ghi SCON, trong khi bit dừng được lọc bỏ. Tốc độ Baud có thể lập trình được bằng 1/32 hoặc 1/64 tần số bộ dao động.



+ ***Chế độ 3***: 11 bit được truyền (thông qua TxD) hoặc được nhận (thông qua RxD) bao gồm: 1 bit khởi động (có giá trị 0), 8 bit dữ liệu (đầu tiên là LSB), 1 bit dữ liệu thứ 9 có thể lập trình được, và 1 bit dừng (có giá trị 1). Trên thực tế, chế độ 3 giống chế độ 2 ở mọi góc độ trừ tốc độ Baud. Tốc độ Baud ở chế độ 3 là khả biến và được xác định theo bộ Timer 1.



Trong cả 4 chế độ trên, việc truyền được bắt đầu bởi bất kỳ một lệnh nào mà sử dụng thanh ghi SBUF như là một thanh ghi đích. Việc nhận được bắt đầu ở chế độ 0 khi RI=0 và REN=1. Đối với các chế độ khác, việc nhận được bắt đầu khi bit REN=1.

2.5.6.3. Các tốc độ Baud:

+ Tốc độ Baud ở chế độ 0 được cố định, và bằng ***Tần số bộ dao động/12***

+ Tốc độ Baud ở chế độ 2 phụ thuộc vào giá trị của bit SMOD trong thanh ghi PCON. Nếu SMOD=0 (giá trị sau khi reset), thì tốc độ Baud =1/64 tần số của bộ dao động. Nếu SMOD=1 thì tốc độ Baud =1/32 tần số của bộ dao động.

Tốc độ Baud chế độ 2 = (2^{SMOD}*Tần số bộ dao động)/64

Trong AT89C51, các tốc độ Baud ở chế độ 1 và 3 do Timer 1 quyết định, Trong AT89C52 tốc độ Baud của các chế độ này có thể được quyết định bởi Timer 1 hoặc Timer 2, hoặc cả hai (một bộ timer xác định tốc độ truyền, bộ kia xác định

tốc độ nhận).

2.5.6.4. Sử dụng Timer 1 để tạo ra các tốc độ Baud :

Khi bộ Timer 1 được dùng để tạo tốc độ Baud, thì các tốc độ Baud ở các chế độ 1 và 3 do tốc độ tràn của timer 1 và giá trị của SMOD quyết định:

Tốc độ Baud ở chế độ 1 và 3 = (2^{SMOD}*(Tốc độ tràn của timer 1))/32

Ngắt của Timer 1 sẽ mất tác dụng trong ứng dụng này.

Bản thân bộ Timer có thể được thiết lập để thực hiện chức năng thời gian hay bộ đếm ở bất kỳ một trong 3 chế độ hoạt động. Trong hầu hết các kiểu ứng dụng, nó thường được thiết lập để thực hiện chức năng thời gian, hoạt động ở chế độ Auto-reload (nửa byte cao của TMOD = 0010b). Trong trường hợp này, tốc độ baud được tính bằng công thức:

Tốc độ Baud chế độ 1 và 3 = (2^{SMOD}*Tần số bộ dao động)/(32*(12*[256-(TH1)])

Ta có thể nhận được các tốc độ Baud rất thấp với bộ Timer 1 bằng cách làm cho ngắt của timer 1 có tác dụng, và thiết lập Timer 1 để hoạt động như một bộ đếm thời gian 16 bit (Nửa byte cao của TMOD=0001b). Bảng 2.8 liệt kê các tốc độ Baud khác nhau thường được sử dụng và cách chúng có thể nhận được từ Timer 1.

Tốc độ Baud (Hz)	**Tần số d.động (MHz)**	**SMODE**	**Timer 1**		
			C/(/T)	***Mode***	***Giá trị nạp lại***
Mode 0 Max: 1M	12	x	X	X	X
Mode 2 Max: 375K	12	1	X	X	X
Mode 1,3 Max:62,5K	12	1	0	2	FFh
19,2K	11,059	1	0	2	FDh
9,6K	11,059	0	0	2	FDh
4,8K	11,059	0	0	2	FAh
2,4K	11,059	0	0	2	F4h
1,2K	11,059	0	0	2	E8h
137,5	11,966	0	0	2	1Dh
110	6	0	0	2	72h
110	12	0	0	1	FEEBh

Bảng . Các tốc độ Baud được tạo ra khi sử dụng Timer 1

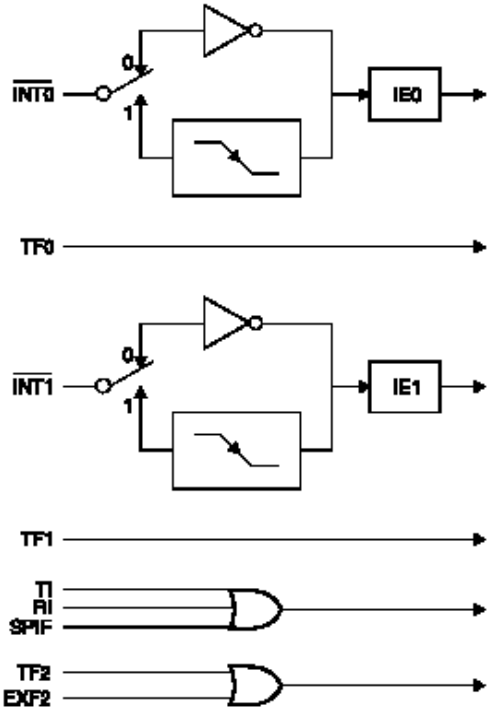
2.7. Cơ chế ngắt trong On-chip AT89C51

- Phân loại ngắt trong On-chip:

Bộ AT89C51 có tất cả 5 Vectors ngắt bao gồm: 2 ngắt ngoài (/INT0 và /INT1), 2

ngắt của khối thời gian (Timer 0, 1), và ngắt cổng truyền tin nối tiếp.

Mỗi nguồn ngắt có thể được kích hoạt hoặc không kích hoạt bằng cách đặt hoặc xoá Bit ở trong IE. IE cũng chứa bit có thể không cho tất cả các ngắt hoạt động EA (Nếu EA=0). Các ngắt ngoài có thể được kích hoạt theo mức hoặc theo sườn xung, tuỳ thuộc vào giá trị của các bit IT0, IT1 trong TCON. Ngắt ngoài có 2 cờ ngắt tương ứng là IE0, IE1 cũng nằm trong TCON. Khi một ngắt được thực hiện thì cờ ngắt tương ứng của nó bị xoá bằng phần cứng. Chương trình con phục vụ ngắt hoạt động chỉ khi ngắt được kích hoạt theo sườn xung. Nếu ngắt được kích hoạt theo mức thì nguồn yêu cầu ngắt từ bên ngoài điều khiển cờ ngắt.



Các ngắt trong, với ngắt Timer/Counter 0, 1 được phát sinh bởi cờ ngắt TF0, TF1. Hai cờ ngắt này được thiết lập khi thanh ghi Timer/Counter thực hiện quay vòng, tại thời điểm S5P2 của chu trình máy. Khi một ngắt được thực hiện thì cờ ngắt tương ứng phát sinh ra ngắt sẽ bị xoá bằng phần cứng trong On-chip.

Ngắt cổng nối tiếp được phát sinh bởi các ngắt RI, TI, SPIF thông qua phần tử Logic OR, khi chương trình con phục vụ ngắt được kích hoạt thì các cờ ngắt phát sinh tương ứng được xoá bằng phần mềm. Các ngắt trong có thể được phép hoặc không được phép kích hoạt bằng cách đặt hoặc xoá một bit trong IE.

-.*Các bước thực hiện ngắt.*

Theo đúng trình tự, để sử dụng các ngắt trong Flash Microcontroller, cần thực hiện các bước như sau:

- Đặt bit EA ở trong IE mức logic 1.

- Đặt bit cho phép ngắt tương ứng ở trong IE mức logic 1.

- Bắt đầu chương trình con phục vụ ngắt tại địa chỉ của ngắt tương ứng đó.

(Xem bảng địa chỉ Vector của các nguồn ngắt)

Ngoài ra, đối với các ngắt ngoài, các chân /INT0, /INT1 phải được đặt mức 1. Và tuỳ thuộc vào ngắt được kích hoạt bằng mức hay sườn xung, mà các bit IT0, IT1 ở trong TCON có thể cần phải đặt mức 1.

ITx=0: Kích hoạt bằng mức

ITx=1: Kích hoạt bằng sườn xung.

- *Mức ngắt ưu tiên trong on-chip:*

Mỗi nguồn ngắt có thể được lập trình riêng cho 1 hoặc 2 mức ưu tiên bằng cách đặt hoặc xoá 1 bit trong IP của SFR. Mỗi ngắt ưu tiên ở mức thấp có thể được ngắt bằng ngắt ưu tiên ở mức cao hơn nhưng không thể ngắt bằng ngắt có mức ưu tiên ở mức thấp hơn được. Một ngắt ưu tiên ở mức cao có thể được ngắt bởi bất kỳ nguồn ngắt nào khác.

Nếu có yêu cầu ngắt của 2 mức ưu tiên cùng nhau (cùng 1 lúc), yêu cầu của mức ưu tiên cao hơn sẽ được phục vụ (Ngắt nào có mức ưu tiên cao hơn sẽ được phục vụ). Nếu các yêu cầu ngắt có cùng mức ưu tiên, thì thứ tự quay vòng bên trong sẽ quyết định ngắt nào được phục vụ.

Thứ tự ưu tiên ngắt từ cao xuống thấp của AT89C51 như sau:

IE0, TF0, IE1, TF1, RI hoặc TI.

-. *Nguyên lý điều khiển ngắt của AT89:*

Các cờ ngắt được thiết lập tại thời điểm S5P2 của mỗi chu kỳ máy. Chu kỳ máy tiếp theo sau chu kỳ máy có cờ ngắt được thiết lập, thì chương trình con được thiết lập khi có lệnh gọi LCALL. Lệnh LCALL phát sinh nhưng lại bị cấm hoạt động khi gặp các tình huống sau:

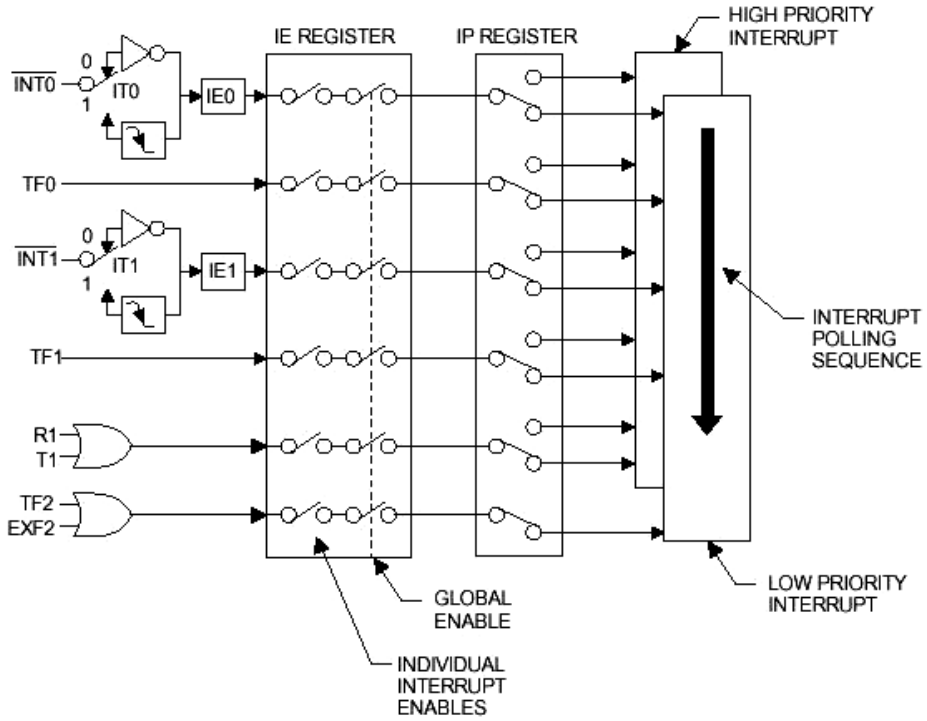
a- Đồng thời có ngắt với mức ưu tiên cao hơn hoặc bằng ngắt đang phục vụ.

(Một ngắt có mức ưu tiên bằng hoặc cao hơn đang sẵn sàng để được phục vụ)

b- Chu kỳ máy hiện hành không phải là chu kỳ máy cuối cùng của lệnh đang thực hiện.

c- Lệnh đang thực hiện là RETI hoặc bất kỳ lệnh nào ghi vào thanh ghi IE

hoặc IP.



Hệ thống ngắt của AT89C51

Bất kỳ một trong 3 điều kiện này xuất hiện sẽ cản trở việc tạo ra LCALL đối với chương trình phục vụ ngắt. Điều kiện 2 đảm bảo rằng, lệnh đang thực hiện sẽ được hoàn thành trước khi trỏ tới bất kỳ chương trình phục vụ nào. Điều kiện 3 đảm bảo rằng, nếu lệnh đang thực hiện là RETI hoặc bất kỳ sự truy cập nào vào IE hoặc IP, thì ít nhất một lệnh nữa sẽ được thực hiện trước khi bất kỳ ngắt nào được trỏ tới. Chu trình kiểm tra vòng được lặp lại với mỗi chu trình máy, và các giá trị được kiểm tra là các giá trị mà đã xuất hiện ở thời điểm S5P2 của chu trình máy trước đó. Nếu một chỉ thị ngắt có hiệu lực nhưng không được đáp ứng vì các điều kiện trên và nếu chỉ thị này vẫn chưa có hiệu lực khi điều kiện cản trở được loại bỏ, thì ngắt bị từ chối này sẽ không được phục vụ nữa.

LCALL do phần cứng tạo ra sẽ chuyển nội dung của bộ đếm chương trình vào ngăn xếp (Nhưng không ghi vào PSW) và nạp lại cho PC một địa chỉ phụ thuộc vào nguồn gây ngắt đang được phục vụ, như bảng dưới đây:

Ngắt	Nguồn ngắt	Địa chỉ Véc tơ
External 0	IE0	0003h
Timer 0	TF0	000Bh
External 1	IE1	0013h
Timer 1	TF1	001Bh
Serial Port	RI hoặc TI	0023h
Timer 2 (AT89C52)	TF2 hoặc EXF2	002Bh
System Reset	RST	0000h

Địa chỉ véc tơ ngắt

Lệnh RETI thông báo cho bộ VXL rằng thủ tục ngắt này đã kết thúc, sau đó lấy ra 2 Byte từ ngăn xếp và nạp lại cho PC để trả lại quyền điều khiển cho chương trình chính.

- Các ngắt ngoài:

Vì các chốt ngắt ngoài được tạo mẫu mỗi lần trong mỗi chu trình máy, nên một giá trị cao hoặc thấp của đầu vào sẽ duy trì trong ít nhất là 12 chu kỳ xung nhịp của bộ dao động để đảm bảo tạo mẫu. Nếu ngắt ngoài được kích hoạt bằng sườn xung , thì nguồn ngắt ngoài phải duy trì ở chốt yêu cầu giá trị cao ít nhất 1 chu kỳ máy và sau đó duy trì giá trị thấp ít nhất 1 chu kỳ máy nữa. Việc này được thực hiện để đảm bảo rằng quá trình chuyển tiếp cho thấy chỉ thị yêu cầu ngắt IEx sẽ được xác lập. IEx sẽ tự động được xoá bởi CPU khi thủ tục ngắt đáp ứng được gọi.

Nếu ngắt ngoài được kích hoạt theo mức, thì nguồn ngắt bên ngoài phải duy trì cho yêu cầu này có hiệu lực cho đến khi ngắt đã được yêu cầu thực sự được tạo ra. Sau đó nguồn ngắt ngoài phải huỷ yêu cầu đó trước khi thủ tục phục vụ ngắt hoàn thành, nếu không ngắt khác sẽ được tạo ra.

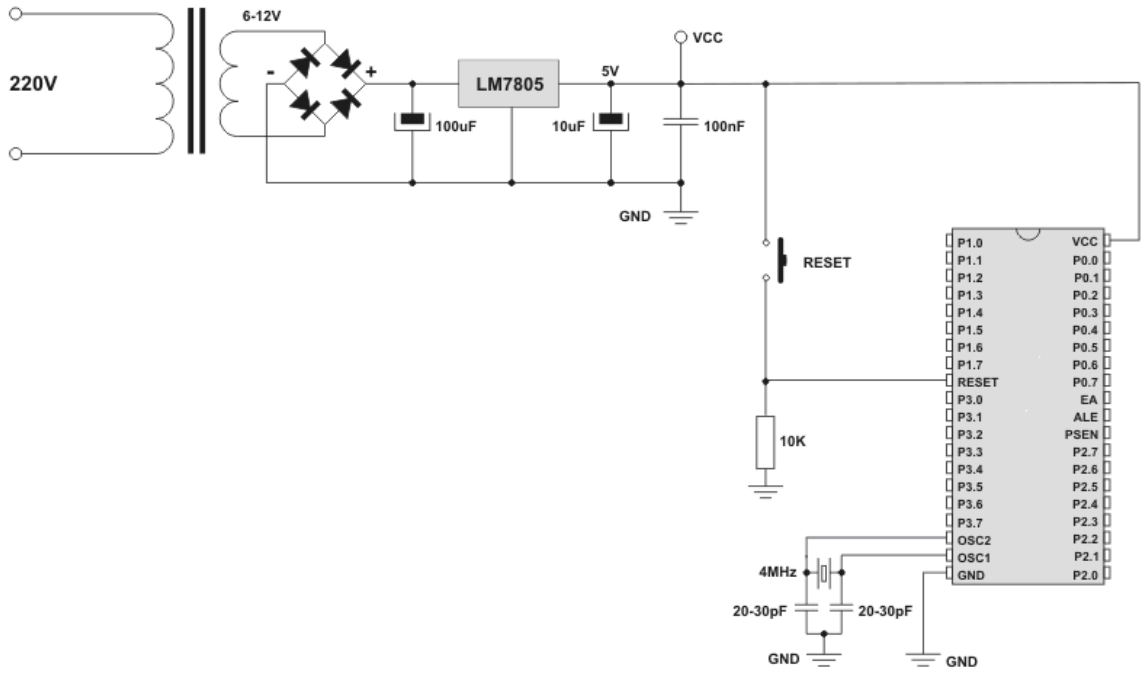
- Vận hành Single-Step:

Cấu trúc ngắt AT89C51 cho phép thực hiện các bước đơn với sự tham gia của rất ít phần mềm. Như đã lưu ý trước đây, một yêu cầu ngắt sẽ không được đáp ứng khi một ngắt khác có cùng mức ưu tiên vẫn đang hoạt động, nó cũng không được đáp ứng sau khi có lệnh RETI cho đến khi có ít nhất một lệnh khác đã được thực hiện. Do đó mỗi khi một thủ tục ngắt được đưa vào, thì nó không thể được đưa vào lần nữa cho đến khi ít nhất một lệnh của chương trình ngắt được thực

hiện. Một cách để sử dụng đặc điểm này đối với hoạt động theo bước đơn lẻ là lập trình cho 1 trong những ngắt ngoài(chẳng hạn /INT0) được kích hoạt theo mức.

Nếu chân /INT0 được duy trì ở mức thấp, thì CPU sẽ chuyển ngay đến thủ tục ngắt ngoài 0 và dừng ở đó cho tới khi INT0 được nhận xung từ thấp lên cao rồi xuống thấp. Sau đó nó sẽ thực hiện lệnh RETI, trở lại nhiệm vụ chương trình, thực hiện một lệnh, và ngay sau đó nhập lại thủ tục ngfắt ngoài 0 để đợi xung nhịp tiếp theo của P3.2. Mỗi bước của nhiệm vụ chương trình được thực hiện vào mỗi thời điểm chân P3.2 được nhận xung.

2.8 Kết nối cơ bản của vi điều khiển 8051



Chương 3: C cho vi điều khiển 8051

3.1 Keil C cho vi điều khiển

3.1.1 Keil Compiler C51 bao gồm phần mở rộng (cho ANSI C) cho:

- các vùng và kiểu bộ nhớ của 8051
- Các chế độ nhớ
- Các kiểu nhớ đặc biệt
- Các kiểu biến dữ liệu đặc biệt
- Biến Bit và biến bit dữ liệu
- Các thanh ghi đặc biệt
- Con trỏ
- Thuộc tính hàm

3.1.2Những kiểu dữ liệu riêng của C51

Những kiểu dữ liệu riêng của C51

- bit

```
static bit done_flag=0;
```

- sbit

```
sbit EA= oxAF; /*defines EA to be the SFR bit at 0xAF*/
```

- sfr(Special Function Registers, 0x80-0xFF)

```
sfr P0 = 0x80;  /* Port-0, address 80h*/
sfr P2 = 0xA0;  /* Port-2, address 0A0h */
```

- sfr16

```
sfr16 T2=0xCC;  /* Timer 2: T2L 0CCh, T2H 0CDh
```

Các chế độ nhớ (Memory Models)

- SmallModel -

Tất cả các biến được mặc định xắp xếp hết trong bộ nhớ dữ liệu trong

Tất cả các đối tượng, như stack phải được đặt trong internal RAM

- Compact Model -

Tất cả các biến được mặc định xắp xếp trong một page của external data memory

Có thể được cung cấp lớn nhất 256 biến

Chậm hơn chế độ SmallModel

- Large Model -

Tất cả các biến được mặc định xắp xếp trong external data memory

Data Pointer (DPTR) được sử dụng để định địa chỉ

Truy nhập bộ nhớ không hiệu quả

Tạo ra nhiều mã hơn các chế độ small và compact model

- Các con trỏ bộ nhớ đặc biệt (Memory-specific Pointers)

Bao gồm các kiểu nhớ đặc biệt trong con trỏ

Có thể được sử dụng để truy nhập các vùng nhớ đã định trước

```
char data *str;
int xdata *numtab;
long code *powtab;
```

Bộ nhớ chương trình

- code có thể mở rộng tới 64Kbyte bộ nhớ chương trình

  char code text[] = “ENTER PARAMETER”;
- Truy nhập bởi lệnh MOVC @A+DPTR

Bộ nhớ chương trình thì chỉ cho phép đọc (trong chương trình)và không thể ghi vào khi chương trình đang thực hiện.

Bộ nhớ dữ liệu

Có tới 256 bytes của bộ nhớ dữ liệu trong

- data : Vùng nhớ 128 bytes đầu tiên của internal memory

```
char data var1;
```

- idata : Tất cả vùng nhớ 256 bytes của internal data memory

```
float idata x,y,z;
```

- bdata : Vùng nhớ 16 bytes của vùng nhớ định địa chỉ bit của internal data memory (20h tới 2Fh)

```
char bdata flags;
```

Bộ nhớ dữ liệu mở rộng

- xdata chỉ bất kỳ vùng nhớ nào trong không gian 64KByte của vùng nhớ dữ liệu mở rộng unsigned long xdata array[100];
- pdata chỉ 1 page của 256 bytes của vùng nhớ mở rộng

```
unsigned char xdata vector[10][4][4];
```

Vùnh nhớ các thanh ghi đặc biệt

- SFRs được mô tả như các biến trong C

- sfr (giống như từ khóa char hoặc int)

sfr P0 = 0x80; /*Port0, address 80h*/

- sfr16 truy nhập 2 SFRs như 16-bit SFR

- sfr16 T2 = 0xCC /*Timer 2; T2L 0CCh, T2H 0CDh)

- sbit cho phép truy nhập tới từng bit riêng của các thanh ghi SFR

sfr PSW=0xD0;

sfr IE=0xA8;

sbit EA=IE^7;

sbit OV=0xD0^2;

sbit CY=0xD7;

3.1.3 Hàm với phần định nghĩa mở rộng.

Trong KeilC có hàm với phần định nghĩa mở rộng cho phép :

Định rõ các hàm như thủ tục ngắt

Chọn register bank sử dụng

Chọn chế độ nhớ

Hàm đệ quy

Cấu trúc hàm mở rộng:

[return_type] funcname ([args]) [{small|compact|large}][reentrant][interrupt n][using n]

Trong đó:

small, compact, large – Chế độ nhớ

reentrant - Hàm đệ quy

interrupt n- Nguồn ngắt (bảng vector ngắt)

using - Chọn bank thanh ghi

Truyền tham số qua các thanh ghi:

Argument Number	char byte ptr 1	int bytes ptr 2	long float	generic ptr

1	R7	R6&R7	R4-R7	R1-R3
2	R5	R4&R5	R4-R7	R1-R3
3	R3	R2&R3		

Giá trị trả về cho hàm

Return Type	Register	Description
bit	Carry Flag	
char	R7	
int	R6&R7	MSB in R6, LSB in R7
long	R4-R7	MSB in R4, LSB in R7
float	R4-R7	bit IEEE format-32
generic ptr	R1-R3	,Memory type in R3 MSB R2, LSB R1

Định nghĩa chế độ nhớ cho một hàm:

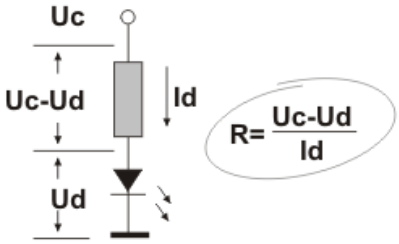
```
#pragma small /*default small model */
extern int calc (char i, int b) large reentrant;
extern int func (char i, float f) large;
extern void *tcp (char xdata *xp, int ndx) small;
int mtest (int i, int y){  /*small model*/
      return (i*y + y*i + func(-1, 4.75);}
int large_func (int i, int k) large {  /*large model*/
      return (mtest(i,k) * 2)}
```

3.1 Project 1 Led đơn

3.1.1 Mạch và nguyên ly hoạt động

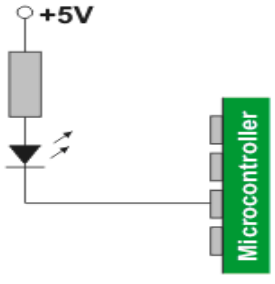
Đây là sơ đồ nguyên lí của 1 led. Led đơn được sử dụng như một phương tiện truyền tín hiệu, có nhiều nhà sản xuất Led với các hình dáng kích thước và màu sắc khác nhau.

Để đảm bảo Led được sáng thì dòng qua Led phải được đảm bảo lớn hơn hoặc bằng dòng điển hình, và cũng phải chú y để đảm bảo dòng điện qua Led phải nhỏ hơn dòng điện max. Với mỗi loại Led, điện áp rơi trên Led sẽ không đổi thường khoảng từ 1.4 tới 4 V do đó người ta thường phải mắc thêm một điện trở có giá trị được tính theo công thức cho trên hình vẽ:



Color	Type	Typical current Id (mA)	Maximal current If (mA)	Voltage drop Ud (V)
Infrared	-	30	50	1.4
Red	Standard	20	30	1.7
Red	Super Bright	20	30	1.85
Red	Low Current	2	30	1.7
Orange	-	10	30	2.0
Green	Low Current	2	20	2.1
Yellow	-	20	30	2.1
Blue	-	20	30	4.5
White	-	25	35	4.4

Một Led đơn được nối với chân của vi điều khiển như hình bên, giả sử chân đó là P1.2 vậy làm thế nào để điều khiển cho Led sáng, tắt:



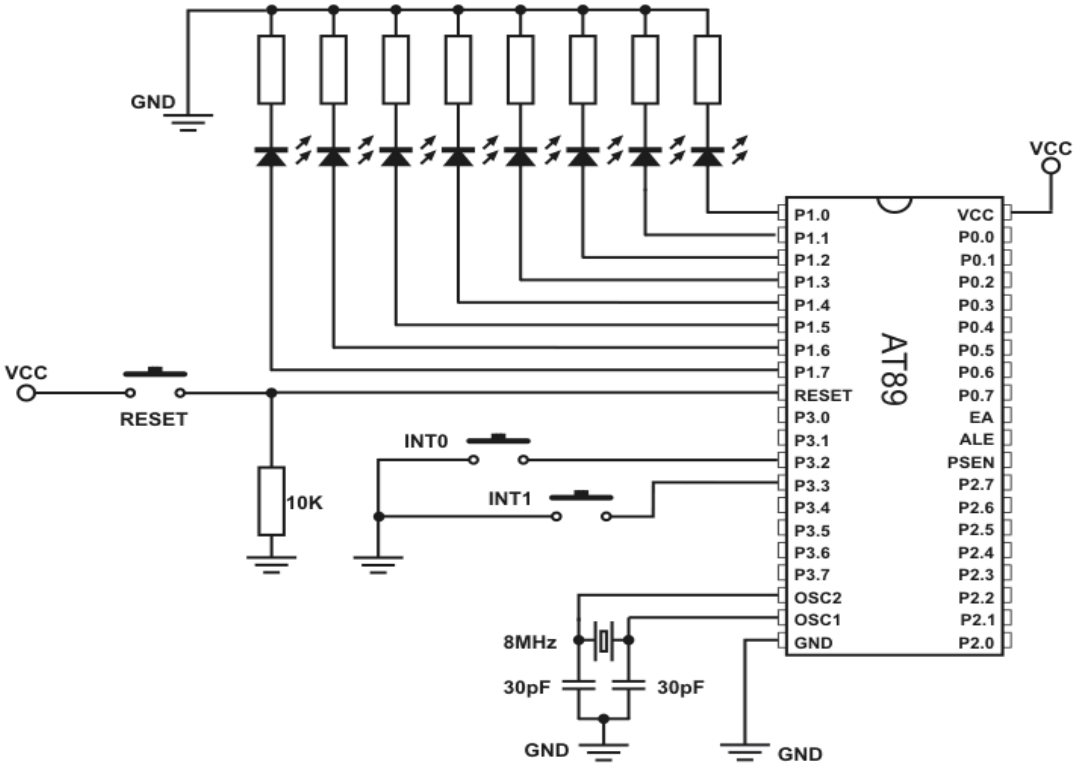
Biến Led1 được khai báo (gán cho) chân P1_2 của vi điều khiển bằng câu lệnh:
sbit Led1=P1^2;
Khi gán : Led1= 0; trong hàm **main** thì chân P1_0 của AT89C51 có mức logic là 0V. Theo sơ đồ nguyên lí: 5V Trở R1 Led1 P1_2 (0 V). Có chênh lệch áp có dòng điện qua led Led sáng. Chúng ta có thể tính tóan chỗ này dễ dàng giá trị của điện trở. Điện áp rơi trên led là Uak (chọn Led vàng) lấy =2 V. Điện áp chân P1_0 là 0V. Điện áp hai đầu trở : 5V - 2V = 3 V. Dòng qua trở = dòng qua led = xấp xỉ 10 mA vậy phải chọn điện trở có giá trị xấp xỉ 3/10*1000=300 Ω.

Khi gán: Led1= 1; tức là chân P1_0 có giá trị 1 tương ứng điện áp của nó là 5V . Hiệu điện thế giữa hai đầu +5V và P1_0 là 0V . Nên không có dòng qua led Led tắt.Nhưng nếu trong hàm main các bạn viết chỉ có như sau: While(1) { Led1=bat; Led1=tat; } Khi chạy debug thì vẫn thấy led nhấp nháy. Nhưng khi nạp chương trình vào chíp lắp vào mạch thì led không nháy hoặc chỉ sáng mờ hoặc tắt ngóm. Vì lệnh Led1=bat; là lệnh 1 chu kì máy , tần số thạch anh là 12 Mhz, 1 chu kì máy có thời gian là 1uS. Vừa bật lên 1 uS rồi lại tắt ngay. Led không đáp ứng được tần số cao vậy nên không nhấp nháy. Do đó cần tới hàm trễ . Bật led lên trễ 1 thời gian khá lâu(0,5 giấy), rồi tắt led di khá lâu(0,5s) rồi lại bật lại tạo thành vòng lặp sẽ được led nhấp nháy. Tác dụng của câu lệnh while(1) . Điều kiện bên trong vòng while là 1 luôn luôn đúng nên nó là vòng lặp vô hạn lần. Nếu không có vòng while(1) thì led của các bạn chỉ sáng lên 1 lần rồi tắt

3.2.2 Chương trình mẫu

```
/*---------------------------------------------------------------------------
Định nghĩa P1.2
---------------------------------------------------------------------------*/
sbit Led1 = P1^2;     /* SFR cho P1.2 */
/*---------------------------------------------------------------------------
Chương trình chính MAIN
---------------------------------------------------------------------------*/
void main (void)
{
/*-----------------------------------------------
Vòng lặp sau liên tục cho Led1 sáng rồi tắt
-----------------------------------------------*/
while (1)
  {
       Led1=0;  /*Led sang*/
       Delay(500);/*Giu cho led sang de nhin thay*/
       Led1=0; /*Tat Led*/
       Delay(500);/*Giu cho led tat de nhin thay*/
  }
}
/*---------------------------------------------------------------------------*/
```

3.3 Project 2 dãy 8 Led đơn



3.3.1 Nguyên lí hoạt động:

Led nối từ chân vđk xuống đất vậy nếu chân vi điều khiển 5V thì led sẽ sáng, nếu chân vi điều khiển 0V thì led sẽ tối. Điện áp 5V vì sao led không cháy mà lại còn sáng yếu? Vì vi điều khiển 8051 chỉ có thể cung cấp dòng nhỏ không đủ 10mA ở 1 chân nên led sáng yếu. Còn nếu muốn led sáng đẹp thì lắp như sau từ dương 5V chân dài của led - chân ngắn của led chân vi điều khiển.

3.3.2 Lập trình :

Trước hết điều khiển 1 led từng Led một. Để điều khiển 1 led thì chỉ việc gán chân nối với led dó bằng 0 hoặc 1, thì điện áp ở chân đó sẽ là 0V hoặc 5V, tùy vào điện áp đèn sẽ sáng hoặc tối.

```
/*================================================  Mo ta:
Dieu khien led don.
Phan cung:    8 led noi tu +5V qua dien tro han dong vao 8 chan cong 1.
Thach anh:   12 Mhz
================================================*/
/************************************************************/
 #include <AT89X51.H>
```

```
/*************************************************************/
/*******************Khai bao bien toan cuc**********************/
sbit Led1=P1^0; //Khai bao bien Led1 kieu bit chan P1.0
sbit Led2=P1^1; // ...
sbit Led3=P1^2;
sbit Led4=P1^3;
sbit Led5=P1^4;
sbit Led6=P1^5;
 sbit Led7=P1^6;
sbit Led8=P1^7;//Khai bao bien Led8 kieu bit chan P1.7
/*************************************************************/
/*********************Khai bao ham*****************************/
/*----------------Delay —Ham tao thoi gian tre-----------------------------
  Dau vao: 1 bien thoi gian.
   Dau ra: khong
 ------------------------------------------------------*/
void Delay(unsigned int time) {
unsigned int 1;// Khai bao bien cuc bo
  for(i=0; i<time; i++)//Lap tao thoi gian tre
 {
 ; // Khong lam gi
}
 }
  /*************************************************************/
/*******************Chuong trinh chinh**************************/
void main(void) {
 while(1)// Vong lap vo han
 {   Led1= 1;// Cho led 1  sang
 tre(1000);// Tre 1 khoang thoi gian
  Led1= 0;// Tat led 1
 tre(1000);// Tre 1 khoang thoi gian
 }
```

```
 }
/*************************************************************/
```

Điều khiển 8 led, chương trình chính được sửa lại như sau:

```
void main(void) {
while(1)// Lap vo han
{
 Led1= 1;// Cho led 1 sang
Delay(500);// Goi ham tao thoi gian tre
 Led1= 0;// Tat led 1
Delay(500);// Goi ham tao thoi gian tre
Led2= 1;// Cho led 2 sang
Delay(500);// Goi ham tao thoi gian tre
 Led2= 0;// Tat led 2
Delay(500);// Goi ham tao thoi gian tre
  Led3= 1;// Cho led 3 sang
Delay(500);// Goi ham tao thoi gian tre
 Led3= 0;// Tat led 3
Delay(500);// Goi ham tao thoi gian tre
 Led4= 1;// Cho led 4 sang
Delay(500);// Goi ham tao thoi gian tre
Led4= 0;// Tat led 4
Delay(500);// Goi ham tao thoi gian tre
Led5= 1;// Cho led 5 sang
Delay(500);// Goi ham tao thoi gian tre
Led5= 0;// Tat led 5
Delay(500);// Goi ham tao thoi gian tre
 Led6= 1;// Cho led 6 sang
Delay(500);// Goi ham tao thoi gian tre
  Led6= 0;// Tat led 6
Delay(500);// Goi ham tao thoi gian tre
Led7= 1;// Cho led 7 sang
Delay(500);// Goi ham tao thoi gian tre
```

```
Led7= 0;// Tat led 7
Delay(500);// Goi ham tao thoi gian tre
Led8= 1;// Cho led 8 sang
Delay(500);// Goi ham tao thoi gian tre
Led8= 0;// Tat led 8
Delay(500);// Goi ham tao thoi gian tre
}
 }
```

Với chương trình này chúng ta có thể cho thứ tự các led tắt bật khác nhau để có các kiểu nháy khác nhau.

3.3.3 Điều khiển ra cả cổng

Nếu các bạn nhàm chán với việc điều khiển từng chân 1 viết code rất tốn công các bạn có thể xuất giá trị ra cả cổng. Trước hết các bạn cần nắm các điều như sau:

- Một cổng có 8 bit tổ hợp, 8bit có 2 = 256 trạng thái. Khi các bạn đưa ra cổng 1 giá trị a (thập phân) từ 0 đến 255 thì số a sẽ được đổi ra hệ nhị phân rồi đưa ra các bit(chân) của cổng. Ví dụ:

Nếu có lệnh:

P1=1;

vì 1(10) nên chân P1_0(bit 0) sẽ bằng 1(5V) còn lại các từ P1_1(bit 1) đến P1_7(bit 7) sẽ bằng 0(0V).

P1=10;

vì 10(10) = 0000 0001 = 0000 1001(2)(2) thì sẽ có P1_0 và P1_3 bằng 1(5V) còn lại các chân khác sẽ là 0(0V).

- Các bạn có thể đưa ra cổng 1 giá trị số hex từ 0 đền ff tương ứng từ 0 đến 255. Các số cơ sở trong hệ hex. (HEX)0 1 2 3 4 5 6 7 8 9 A B C D E F (10) 10 11 12 13 14 15 Cách đổ số hex ra số thập phân: có số hex : N=abf1 đổi ra hệ số 10 N(10)=1.160 + 15.161+ 11.162+ 10.163(16). Đổi số nhị phân sang hex: Gộp 4 số nhị phân thành 1 số hex: Ví dụ: 0010 0001(2)(16) 1 4 số đầu có bit 1 = 1 nên 1x20= 2

4 số sau có bit 0 =1 nên 1x2=1. Cách đưa ra như sau:

Ví dụ lệnh P1=1; tương dương với P1=0x01;

P1=10; tương đương với P1=0x0A;

Chương trình xuất ra cả cổng tương đương với chương trình điều khiển 8 led từng cái 1 như sau:

```
void main(void) {
while(1)// Lap vo han
{
 P1=0x01;// Bat led 1
Delay(1000);// Tre 1 khoang thoi gian
P1=0x00;// Tat led 1
Delay (1000);// Tre 1 khoang thoi gian
P1=0x02;// Bat led 2
Delay (1000);// Tre 1 khoang thoi gian
P1=0x00;// Tat led 2
Delay (1000);// Tre 1 khoang thoi gian
P1=0x04;// Bat led 3
Delay (1000);// Tre 1 khoang thoi gian
P1=0x00;// Tat led 3
Delay (1000);// Tre 1 khoang thoi gian
P1=0x08;// Bat led 4
Delay (1000);// Tre 1 khoang thoi gian
P1=0x00;// Tat led 4
Delay (1000);// Tre 1 khoang thoi gian
P1=0x10;// Bat led 5
Delay (1000);// Tre 1 khoang thoi gian
 P1=0x00;// Tat led 5
Delay (1000);// Tre 1 khoang thoi gian
P1=0x20;// Bat led 6
Delay (1000);// Tre 1 khoang thoi gian
P1=0x00;// Tat led 6
Delay (1000);// Tre 1 khoang thoi gian
P1=0x40;// Bat led 7
Delay (1000);// Tre 1 khoang thoi gian
```

```
P1=0x00;// Tat led 7
Delay (1000);// Tre 1 khoang thoi gian
 P1=0x80;// Bat led 8
Delay (1000);// Tre 1 khoang thoi gian
P1=0x00;// Tat led 8
Delay (1000);// Tre 1 khoang thoi gian
  }
  }
```

Như vậy gõ code vẫn mỏi tay lắm để dạt được mục đích 8 đèn nháy liên tiếp các bạn có thể làm như sau:

```
 /*******************Ham chinh*****************************/
void main(void) {
 unsigned char n; // Khai bao them bien n cho vong for
 while(1)// Lap vo han
 {
 P1=0x01;// Bat led 1
 for(n=0 ; n<8;n++)// Lap 8 lan
 {
P1=P1<<1; // Dich trai 1 bit
   Delay (1000);
}
 }
 }
  /*************************************************************/
```

Debug quan sát sự thay đổi của cổng 1 để thấy được tác dụng phép dịch bit xang trái. Để hiểu thao tác xuất ra cổng, chân, các các bạn làm 1 ví dụ nữa như sau:

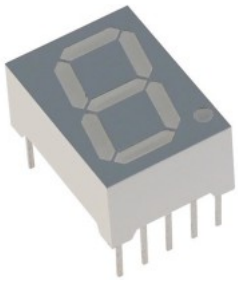
```
/************************Ham chinh*******************************/
void main(void) {
unsigned char n; // Khai bao them bien n cho vong for
while(1)// Lap vo han
{
   P1=0x01;// Bat led 1
```

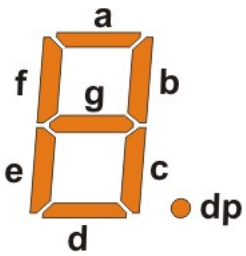
```
 for(n=0 ; n<256;n++)// Lap 8 lan
{
   P1=n; // Dich bit xang trai
  Delay (5000);
 }
 }
 }
  /*****************************************************************/
```

Debug để thấy sự thay đổi các cổng.

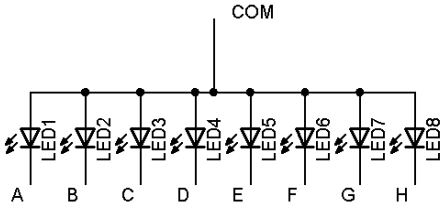
3.4 Project 3: điều khiển Led 7 thanh

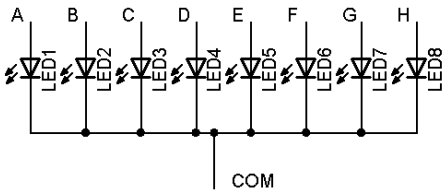
3.4.1 Hình dạng và cấu tạo:



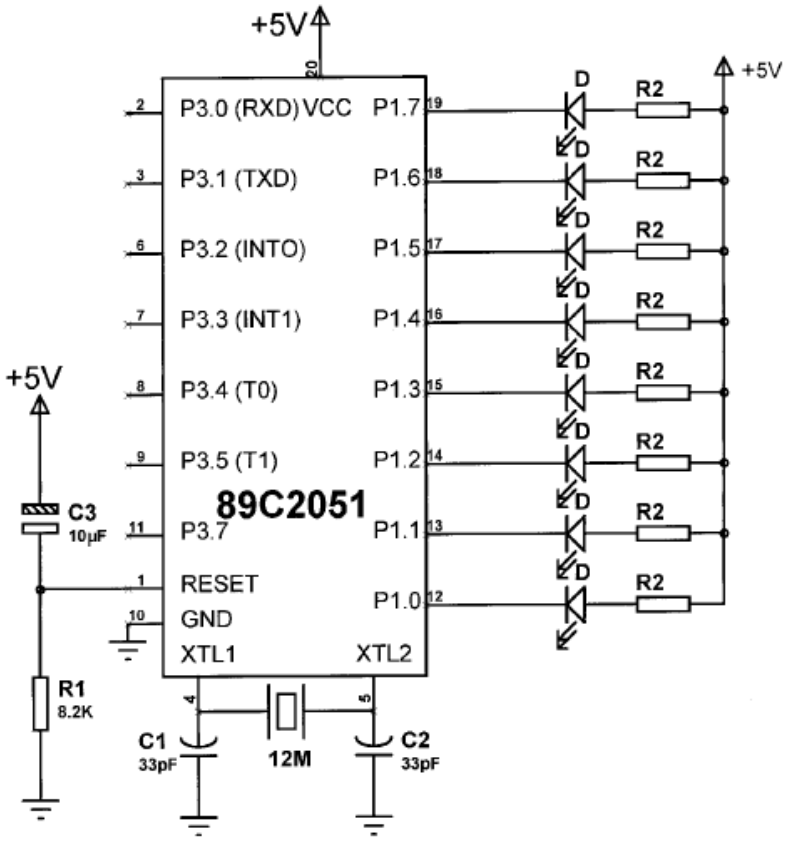


Có hai loại led 7 thanh: Anốt chung và Catốt chung. Hình trên là so đồ chân của hai loại led. Nó có cấu tạo như sau:





Mạch lắp sử dụng led Anốt chung nhu sau:



3.4.2 Nguyên lí hoạt động:

Khi cắm nguồn vào mạch tất cả các chân của các cổng IO của VĐK là 5V(Nếu cổng P0 không lắp điện trở treo thì sẽ là 0V). Nhìn sơ đồ mạch không có chênh lệch điện áp nên không có đèn nào sáng. Chúng ta muốn sáng thanh nào chỉ việc đưa ra điện áp 0V ở chân vi điều khiển nối với thanh đó.

Thanh hiện Thanh tắt Giá trị(P1)

Để hiện thị số 1: B,C các thanh còn lại 1111 1001

Để hiện thị số 2: A,B,D,E,G các thanh còn lại 1010 0100 ….

Để hiện thị số 8: Tất cả các thanh không thanh nào 1000 0000

Bít thứ 8 P1.7 không dùng.

Ngoài ra led 7 thanh còn có thể hiện thị 1 số chữ

Để hiển thị chữ B: Giống số 8

Hiển thị chữ A: A,B,C,E,F,G D 1000 1000

3.4.3 Lập trình :

3.4.3.1 Cách 1: Lập trình dễ hiểu không cần phải tính toán nhưng phải viết và copy, paste và sửa nhiều.

Code như sau:

```
#include <AT89X52.H> /* Khai bao cac bien bit gan voi chan vi điеu khien*/
sbit a = P1^0;
sbit b = P1^1;
sbit c = P1^5;
sbit d = P1^4;
sbit e = P1^3;
sbit f = P1^2;
sbit g = P1^3;
/* Khai bao bien */
long n;// Cho vong for
 /* Khai bao ham */
/* Ham tre */
void delay(long time) {
for(n=0; n<time; n++) {
;
}
}

/* Ham tat tat ca cac thanh */
 void tat(void)   {
a =1; b =1; c =1; d =1; e =1; f =1; g =1;
}
 /* Cac ham hien thi chu va so */
void so1(void) {
tat();
a =1; b =0; c =0; d =1;e =1; f =1;  g =1;
}
 void so2(void) {
tat();
a =0; b =0; c =1; d =0; e=0; f =1; g =0;
}
```

```
void so3(void) {
tat();
a =0; b =0;c =0; d =0;e =1; f =1; g =0;
}
void so4(void) {
tat();
a =1;b =0; c =0; d =1;e =1; f =0; g =0;
}
void so5(void) {
tat();
a =0; b =1; c =0; d =0; e =1; f =0; g =0;
}
void so6(void) {
 tat();
a =0; b =1; c = 0;d =0; e =0; f =0; g =0;
}
 void so7(void) {
tat();
a =0; b =0; c =0; d =1; e =1; f =1; g =1;
}
void so8(void) {
 tat();
 a =0; b =0; c =0; d =0;e =0; f =0; g =0;
 }
void so9(void) {
 tat();
a =0; b =0; c =0; d =0; e =1;f =0; g =0;
}
void chuA(void) {
tat();
a =0; b =0; c =0;d =1; e =0; f =0; g =0;
 }
```

```
void chuB(void) {
 tat();
a =0; b =0; c =0; d =0; e =0; f =0; g =0;
 }
void chuC(void) {
tat();
 a =0; b =1; c =1; d =0; e =0; f =0; g =1; }
void chuD(void) {
 tat();
a =0; b =0; c =0; d =0; e =0; f =0; g =1; }
void chuE(void) {
 tat();
a =0; b =1; c =1; d =0; e =0; f =0; g =0; }
void chuF(void) {
tat();
a =0; b =1; c =1; d =1; e =0; f =0; g =0; }
void chuG(void) {
tat();
 a =0; b =1; c =0; d =0; e =0; f =0; g =1; }
void chuH(void) {
tat();
a =1; b =0; c=0; d =1; e =0; f =0; g =0; }
void chuI(void) {
 tat();
a =1; b =1; c =1; d =1; e =0; f =0; g =1; }
void chuL(void) {
 tat();
a =1; b =1; c =1; d =0; e =0; f =0; g =1; }
void chuO(void) {
tat();
a =0; b =0; c =0; d =0; e =0; f =0; g =1; }
void chuP(void) {
```

```
 tat();
 a =0; b =0; c =1; d =1; e =0; f =0; g =0; }
void chuR(void) {
 tat();
a =0; b =0; c =0; d =1; e =0; f =0; g =0; }
 void chuS(void) {
 tat();
a =0; b =1; c =0; d =0; e =1; f =0; g =0; }
void chuU(void) {
tat();
 a =1; b =0; c =0; d =0; e =0; f =0; g =1; }
void chuY(void) {
 tat();
a =1; b =0; c =0; d =0; e =1; f =0; g =0; }
  /* Ham chinh */
 void main(void) {
while(1) {
so0();
 delay(20000);
 so1();
delay(20000);
so2();
delay(20000);
 so3();
delay(20000);
so4();
 delay(20000);
 so5();
delay(20000);
so6();
delay(20000);
 so7();
 delay(20000);
 so8();
```

```
delay(20000);
so9();
delay(20000);
 chuA();
 delay(20000);
 chuB();
delay(20000);
chuC();
delay(20000);
chuD();
delay(20000);
 chuE();
delay(20000);
chuF();
delay(20000);
chuG();
delay(20000);
chuH();
delay(20000);
 chuI();
 delay(20000);
 chuL();
delay(20000);
chuO();
delay(20000);
chuP();
 delay(20000);
chuR();
delay(20000);
 chuS();
 delay(20000);
chuU();
delay(20000);
chuY();
```

```
delay(20000);
} }
```

3.4.3.2 Cách 2:

Các bạn viết 1 chương trình đơn giản rồi dùng công cụ Debug để xem số hex rồi viết vào rất ngắn gọn.

Ví dụ: Hàm hiển thị số 1:

```
void so1(void) {
 tat();
P1=0xF5;
}
```

Các bạn debug cho hiển thị cổng P1 lên. Để dấu tích ở các đèn tắt(1) , bỏ dấu tích ở các đèn cần bật(0). Rồi đọc giá trị hex như tôi hướng dẫn ở bài trước. Dùng cấu trúc lệnh switch case để viết lại chương trình sẽ rất gọn.

```
void Hienthiled(unsigned char x)
// Co 1 bien dau vao de xac dinh xem la hien thi so nao
{
switch(x)  {
 case 1: {
 tat(); P1=0xF5; break;
} // So 1
case 2: {
 tat(); P1=0xFF; break;
}// So 2   …
case 9: {
 tat(); P1=0xFF; break;
}// So 9
case 10: {
tat(); P1=0xFF; break;
}// Chu A   ….
 case 20: {
 tat(); P1=0xFF; break;
```

}// Chu Y

}

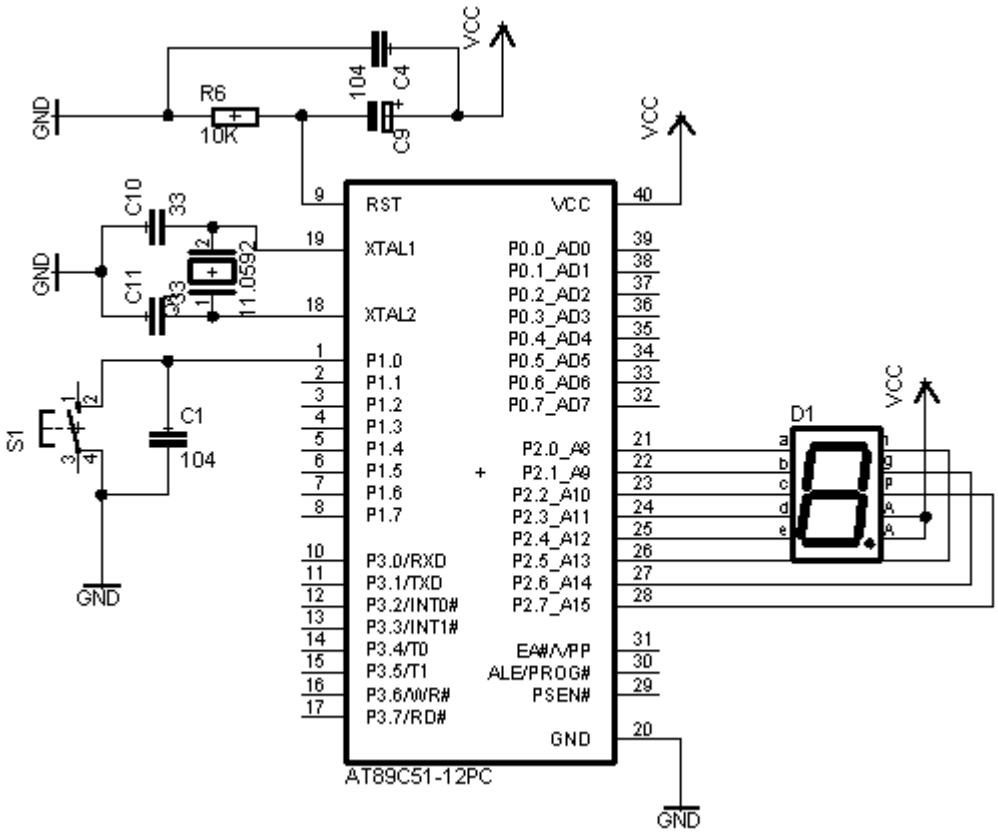
}

Các giá trị ở trên chỉ là ví dụ các bạn đã rút gọn và tự copy vào. Với hàm hiển thị led các bạn đã viết để hiện các số và các chữ giờ hàm main chỉ cần như sau:

```
 void main (void)  {
while(1)  {
 for(n=0; n<20; n++)
{
 Hienthiled(n);
delay(20000);
  }
 }
```

3.5 Phím nhấn

3.5.1 Đếm số lần phím bấm giới hạn từ 0 đến 9 hiển thị ra led 7 thanh.



3.5.1.1Nguyên lí hoạt động:

- Phần nút bấm: (khi không có tụ 104) ban đầu chân P1.0 ở mức cao +5V, nếu bấm nút 2 đầu nút bấm thông với nhau. Chân P1.0 thông với GND. Led sáng do có chênh áp. Chân P1.0 thông đất. Nếu có tụ 104 tụ điện được nạp điện, khi bấm nút tụ điện sẽ phóng điện từ cực dương sang cực âm làm chân P1.0 thông với GND nhưng lâu về 0 V hơn 1 chút.
- Do tiếp điểm cơ khí của nút bấm nên khi bấm nút nó sẽ có 1 số xung điện. Tụ 104 có tác dụng giảm nhiễu đó. Tụ 104 cũng có thể bỏ đi không lắp vì ta có thể khử nhiễu bằng phần mềm.

3.5.1.2 Lập trình:

Code bài trước giữ nguyên: soạn thêm một số hàm như sau hàm đọc phím bấm.
Hàm đọc số lần ấn phím
Đây là code các hàm bổ sung:

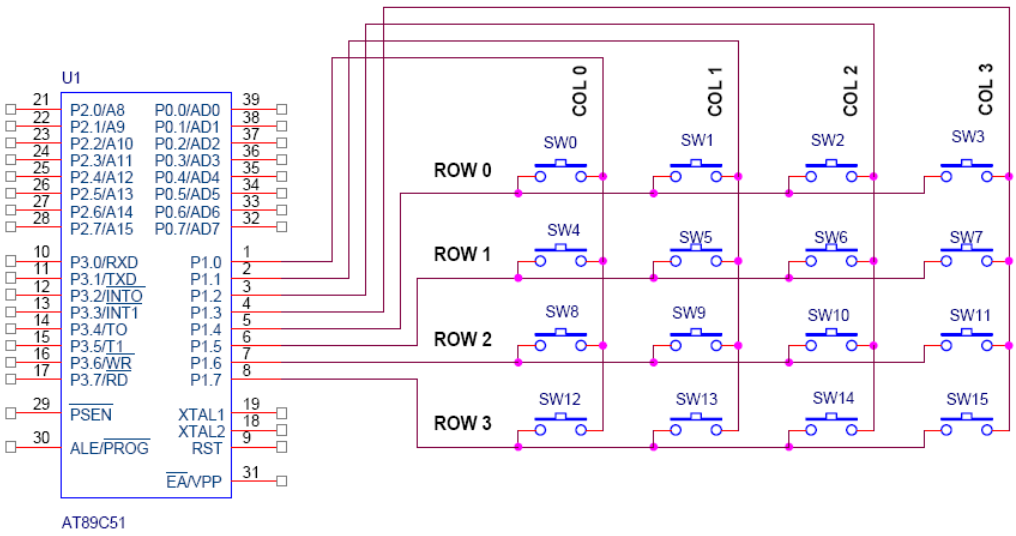
```
unsigned char i=0;// Khai bao them bien toan cuc de dem so lan an nut
 unsigned char Docnutnbam(void)// Ham tra lai gia tri unsigned char
{    if( P1_0 ==0)// Neu nut duoc bam
{
delay(300);// Tre 1 khoang thoi gian qua xung nhieu
while(P1_0 ==0) // Cho toi khi nha tay khoi nut bam
{
 ;//Khong lam gi
}
 i++;// Nha tay thi tang i
 if( i ==10) i=0;// Quay vong gia tri cua i
  }
return i;
}
void hienthisolannhanphim(unsigned char solan) {
 switch(solan)// Tuy vao so lan
{
case 0:
```

```
{
so0(); break;
 }// Neu so lan =0 hien so 0 thoat khoi
 switch
 case 1: {
 so1(); break;
}// Neu so lan =1 hien so 1 thoat khoi switch
    case 2: {
 so2(); break;
}// ....
 case 3: {
so3(); break;
}
case 4: {
 so4(); break;
}
 case 5: {
so5(); break;
}
  case 6: {
 so6(); break;
}
case 7: {
so7(); break;
 }
case 8: {
 so8(); break;
}
  case 9: {
so9(); break;
 }// Neu so lan =9 hien so 9 thoat khoi switch
 }
```

```
 }
 void main(void) {
 while(1) {
  Docnutbam(); // Goi ham doc so lan nhan phim
hienthisolannhanphim(i);// Hien thi so lan nhan phim, bien i la bien toan cuc
 }
}
```

3.5.2 Đọc ma trận phím

Nhiệm vụ: Quét bàn phím 16 phím bấm(4x4), xem phím nào được bấm, các phím được đánh số từ 0 đến 15 rồi hiển thị giá trị ra led 7 thanh



3.5.2.1 Nguyên lí quét phím:

- Vì sao mạch phím đấu theo ma trận. Nếu để đọc từ 16 nút bấm bình thường phải dùng 16 chân vi điều khiển. Nếu đấu theo dạng ma trận thì chỉ mất 8 chân ta cũng có thể đọc được 16 phím bấm.

- Có 2 cách quét phím theo cột và theo hàng, tôi chọn cách quét theo cột, quét theo hàng các bạn có thể làm tương tự.

- Bước 1 : Ta đưa chân P1.0 nối với cột 1 xuống 0V.Rồi ta kiểm tra giá trị logic của các chân P1.4,P1.5,P1.6,P1.7.Nếu phím 1 được bấm thì hàng 1_ P1.4 sẽ có giá trị bằng 0.

Nếu phím 2 được bấm thì hàng 2_ P1.5 sẽ có giá trị bằng 0. Nếu phím 3 được bấm thì
hàng 3_ P1.6 sẽ có giá trị bằng 0. Nếu phím 4 được bấm thì hàng 4_ P1.7 sẽ có giá trị bằng 0. Ta căn cứ vào đó để xác định xem phím nào được bấm.
- Bước 2 : Ta đưa chân P1.1 nối với cột 2 xuống 0V.Rồi ta kiểm tra giá trị logic của các chân P1.4,P1.5,P1.6,P1.7.Nếu phím 5 được bấm thì hàng 1_ P1.4 sẽ có giá trị bằng 0.
Nếu phím 6 được bấm thì hàng 2_ P1.5 sẽ có giá trị bằng 0. Nếu phím 7 được bấm thì
hàng 3_ P1.6 sẽ có giá trị bằng 0. Nếu phím 8 được bấm thì hàng 4_ P1.7 sẽ có giá trị bằng 0. Ta căn cứ vào đó để xác định xem phím nào được bấm. Tương tự ta thực hiện cho các cột còn lại. Ta sẽ dùng câu lệnh if để kiểm tra.

3.5.2.2 Lập trình:

- Tạo 1 project mới, copy phần hiển thị các số 0…9 các chữ A…Y của bài trước. Rồi bổ sung các hàm sau. Hàm hiện thị phím ấn.

```
void phim_duoc_an(unsigned char phim) {
 switch(phim)// Tuy vao so lan
 {
case 0: {
so0(); break;
 }// Neu so lan =0 hien so 0 thoat khoi switch
case 1: {
so1(); break;
}// Neu so lan =1 hien so 1 thoat khoi switch
case 2: {
 so2(); break;
 }// ....
case 3: {
so3(); break;
 }
case 4: {
```

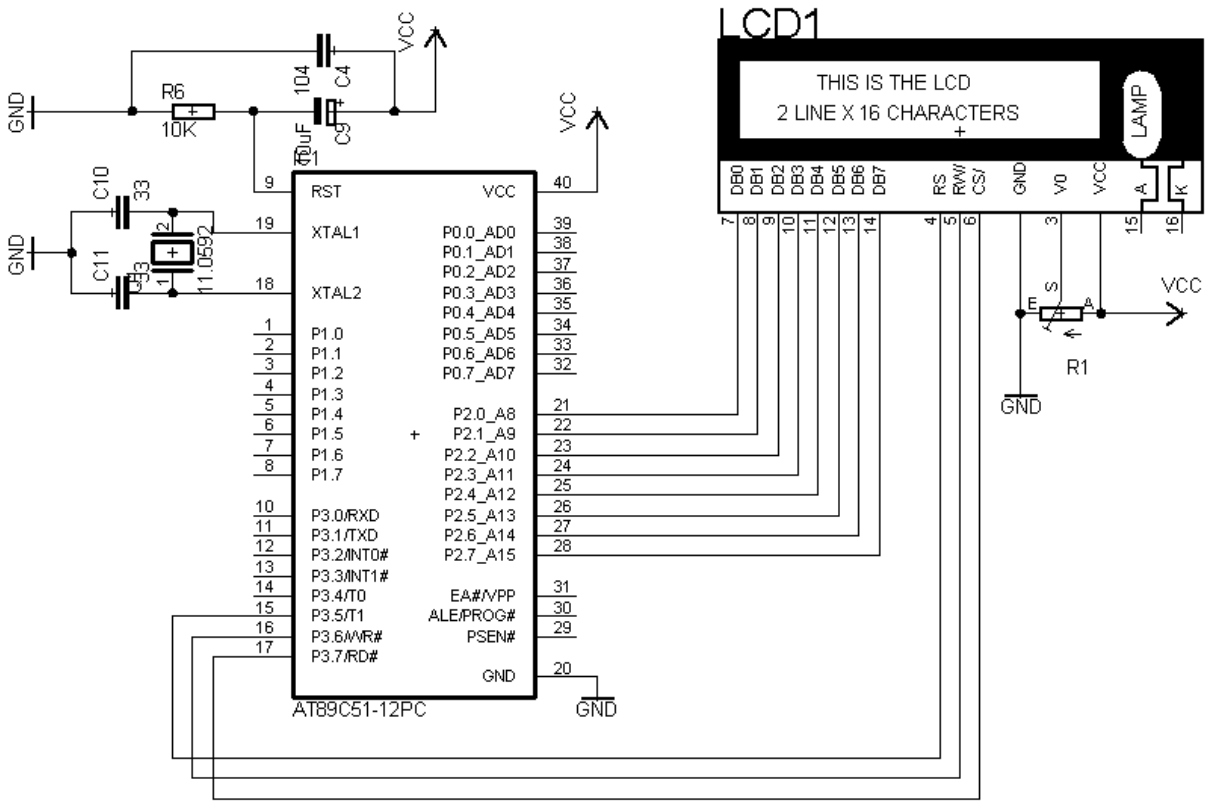
```
 so4(); break;
}
case 5: {
so5(); break;
}
case 6: {
so6(); break;
}
case 7: {
so7(); break;
}
case 8: {
so8(); break;
}
case 9: {
so9(); break;
}// Neu so lan =9 hien so 9 thoat khoi switch
} }
```

Hàm quét phím:

```
/*Khai bao 1 mang 4 phan tu nhu sau:
quetphim[4]={P0=0xFE,0xFD,0xFB,0xF7}
De dua 0 ra lan luot cac hang phim, khi do neu nut nao đuoc an thi chan vi điieu khien
se xuong 0.Chu y fai kiem tra phim khoang 100 lan.*/
 unsigned char quetphim[4]={0xFE,0xFD,0xFB,0xF7};
 // Dinh nghia so lan quet phim
#define solanquetphim 100 // Cac ban co the thay doi gia tri nay cho phu hop unsigned
char quetbanphim(void) {
unsigned char giatribanphim;// Bien de luu gia tri phim an tu 0 den 15 ma hoa 16 phim
unsigned char x,y;    //Quet 4 hang phim
for(x=0; x<4;x++)  {
P1=quetphim[x];// Dua lan luot cac hang xuong 0
 for(y=0;y<solanquetphim;y++)// Kiem tra solanquetphim lan
```

```
{
if(P1_4==0) giatribanphim=0+4*x;// Gia tri phim tuong ung
 if(P1_5==0) giatribanphim=1+4*x;// Tuy thuoc vao hang x
if(P1_6==0) giatribanphim=2+4*x;// La may ma gia tri cua
if(P1_7==0) giatribanphim=3+4*x;// gia tri ban phim tuong ung.
}
 }
return(giatribanphim);
} Hàm Main.
 void main(void) {
unsigned char i;
 while(1)  {
 i=quetbanphim();
  phim_duoc_an(i);
 }
}
```

3.6 Ghép nối với LCD



3.6.1 Nguyên lý hoạt động của LCD:

- Chân VCC, V và VSS: Các chân VEECC, V: Cấp dương nguồn - 5v và đất tương ứng thì V0 được dùng để điều khiển độ tương phản của LCD. - Chân chọn thanh ghi RS (Register Select): Có hai thanh ghi trong LCD, chân RS(Register Select) được dùng để chọn thanh ghi, như sau: Nếu RS = 0 thì thanh ghi mà lệnh được chọn để cho phép người dùng gửi một lệnh chẳng hạn như xoá màn hình, đưa con tr ỏ về đầu dòng v.v… Nếu RS = 1 thì thanh ghi dữ liệu được chọn cho phép người dùng gửi dữ liệu cần hiển thị trên LCD.

- Chân đọc/ ghi (R/W): Đầu vào đọc/ ghi cho phép người dùng ghi thông tin lên LCD khi R/W = 0 hoặc đọc thông tin từ nó khi R/W = 1.

- Chân cho phép E (Enable): Chân cho phép E dược sử dụng bởi LCD để chốt dữ liệu của nó. Khi dữ liệu được cấp dến chân dữ liệu thì một xung mức cao xuống thấp phải được áp đến chân này để LCD chốt dữ liệu trên các chân dữ liệu. Xung này phải rộng tối thiểu là 450ns.

- Chân D0 - D7: Đây là 8 chân dữ liệu 8 bít, được dùng dể gửi thông tin lên LCD hoặc đọc nội dung của các thanh ghi trong LCD. Để hiển thị các chữ cái và các con số, chúng ta gửi các mã ASCII của các chữ cái từ A đến Z, a đến f và các con số từ 0 - 9 đến các chân này khi bật RS = 1.

Cũng có các mã lệnh mà có thể được gửi đến LCD để xoá màn hình hoặc đưa con trỏ về đầu dòng hoặc nhấp nháy con trỏ.

- Chú ý:Chúng ta cũng sử dụng RS = 0 để kiểm tra bít cờ bận để xem LCD có sẵn sàng nhận thông tin. Cờ bận là bít D7 và có thể được đọc khi R/W = 1 và RS = 0 như sau:

Nếu R/W = 1, RS = 0 khi D7 = 1 (cờ bận 1) thì LCD bận bởi các công việc bên trong và

sẽ không nhận bất kỳ thông tin mới nào. Khi D7 = 0 thì LCD sẵn sàng nhận thông tin mới. Lưu ý chúng ta nên kiểm tra cờ bận trước khi ghi bất kỳ dữ liệu nào lên LCD.

3.6.2 Mã (Hex) Lệnh đến thanh ghi của LCD

1: Xoá màn hình hiển thị

2: Trở về đầu dòng

4: Giảm con trỏ (địch con trỏ sang trái) 6 Tang con trỏ (dịch con trỏ sang phải)

5: Dịch hiển thị sang phải

7: Dịch hiển thị sang trái

8: Tắt con trỏ, tắt hiển thị

A: Tắt hiển thị, bật con trỏ

C: Bật hiển thị, tắt con trỏ

E: Bật hiển thị, nhấp nháy con trỏ

F: Tắt con trỏ, nhấp nháy con trỏ

10: Dịch vị trí con trỏ sang trái

14: Dịch vị trí con trỏ sang phải

18: Dịch toàn bộ hiển thị sang trái

1C: Dịch toàn bộ hiển thị sang phải

80: Dịch con trỏ về đầu dòng thứ nhất

C0: ép con trỏ về đầu dòng thứ hai

38: Hai dòng và ma trận 5 × 7

- Điều khiển LCD qua các bước sau:

Bước 1 : Chuẩn bị phần cứng. Dùng tuốc vít hay cái gì bạn có xoay biến trở 5 K điều chỉnh độ tương phản của LCD. Xoay cho đến khi các ô vuông(các điểm ảnh) của LCD hiện lên thì xoay ngược biến trở lại 1 chút.

Bước 2 : Khởi tạo cho LCD.

Bước 3 : Gán các giá trị cho các bit điều khiển các chân RS,RW,EN cho phù hợp với các chế dộ : Hiển thị kí tự lên LCD hay thực hiện 1 lệnh của LCD.

Bước 4: Xuất byte dữ liệu ra cổng diều khiển 8 bit dữ liệu của LCD.

Bước 5: Kiểm tra cờ bận xem LCD sẵn sàng nhận dữ liệu mới chua.

Bước 6: Quay vòng lại bước 1.

3.6.3 Lập trình:

- Để có thể lập trình cho LCD ta thêm vào thư viện string.h của trình biên dịch

bằng câu lệnh:

```
 #include <string.h>
```

- Khai báo các chân của LCD gắn với các cổng:

```
/*   RS chon thanh ghi
 =0 ghi lenh
 =1 ghi du lieu
RW doc ghi
=0 ghi
=1 doc
  E cho phep chot du lieu
xung cao xuong thap toi thieu 450 ns.
Bit co ban D7
   khi RS=0 RW=1 neu D7=1 LCD ban
       D7=0 LCD san sang.
 */
sfr LCDdata = 0xA0;// Cong 2 , 8 bit du lieu  P0 co dia chi 0x80, P1 0x90 , P2 0xA0
sbit BF = 0xA7; // Co ban bit 7  sbit RS = P3^5;  sbit RW = P3^4; sbit EN = P3^3;
```

- Viết 1 số hàm điều khiển LCD nhu sau:

* Hàm kiểm tra LCD có bận hay không:

```
void wait(void) {
  long n = 0;
EN=1;// Dua chan cho fep len cao
RS=0;// Chon thanh ghi lenh
RW=1;// Doc tu LCD
LCDdata=0xFF;// Gia tri 0xFF
while(BF){
n++;
if(n>100) break;
}// Kiem tra co ban
// Neu ban dem n den 100 roi thoat khoi
 while   EN=0;// Dua xung cao xuong thap de cho
   RW=0;// Doc tu LCD
```

```
}
```
* Hàm điều khiển LCD thực hiện 1 lệnh:
```
 void LCDcontrol(unsigned char x) {
EN=1;// Dua chan cho fep len cao
RS=0;// Chon thanh ghi lenh
RW=0;// Ghi len LCD
 LCDdata=x;// Gia tri x
EN=0;// Xung cao xuong thap
wait();// Doi LCD san sang
}
```
Hàm có 1 biến đầu vào là các giá trị trong bảng mã lệnh của LCD.
* Hàm khởi tạo LCD:
```
void LCDinit(void) {
 LCDcontrol(0x30);//Che do 8 bit.
 LCDcontrol(0x30);
 LCDcontrol(0x30);
LCDcontrol(0x38);// 2 dong va ma tran 5x7
LCDcontrol(0x0C);// Bat con tro
LCDcontrol(0x06);// Tang con tro xang fai
 LCDcontrol(0x01);// Xoa man hinh
}
```
* Hàm l ệnh cho LCD hiển thị 1 kí tự :
```
void LCDwrite(unsigned char c) {
EN=1;// Cho phep muc cao
RS=1;// Ghi du lieu
RW=0;// Ghi len LCD
LCDdata=c;// Gia tri C
 EN=0;// Xung cao xuong thap
   wait();// Cho
 }
```
Hàm có 1 bi ến dầu vào là mã của kí tự trong bảng ASCII.
* Hàm l ệnh cho LCD hiển thị 1 xâu kí tự (dòng chữ):

```
 void LCDputs(unsigned char *s,unsigned char row) {
 unsigned char len;
if(row==1)
LCDcontrol(0x80);// dich con tro ve dau dong 1
 else
LCDcontrol(0xC0);// dich con tro ve dau dong 2
len=strlen(s);// Lay do dai bien duoc tro boi con tro
while(len!=0)// Khi do dai van con
{
 LCDwrite(*s);// Ghi ra LCD gia tri duoc tro boi con tro
   s++;// Tang con tro
  len--;// Tru do dai
 }
}
```

Hàm hiển thị 1 số integer:

```
void LCDwritei(int d) {
unsigned char i,j,k,l;
i=d%10;// Chia lay phan du, duoc chu so hang don vi
d=d/10;// Chia lay phan nguyen, duoc nhung chu so da bo hang don vi
j=d%10;// Duoc chu so hang chuc
d=d/10;// Nhung chu so da bo hang don vi va hang chuc
k=d%10;// Duoc hang tram
l=d/10;// Duoc hang nghin
 LCDwrite(48+l);// Hien thi ki tu trong bang ascii
 LCDwrite(48+k);// Trong bang ascii so 0 co co so thu tu la 48
 LCDwrite(48+j);
LCDwrite(48+i);
}
```

Hàm có 1 biến đầu vào là số int lớn dến hàng nghìn cần hiển thị.

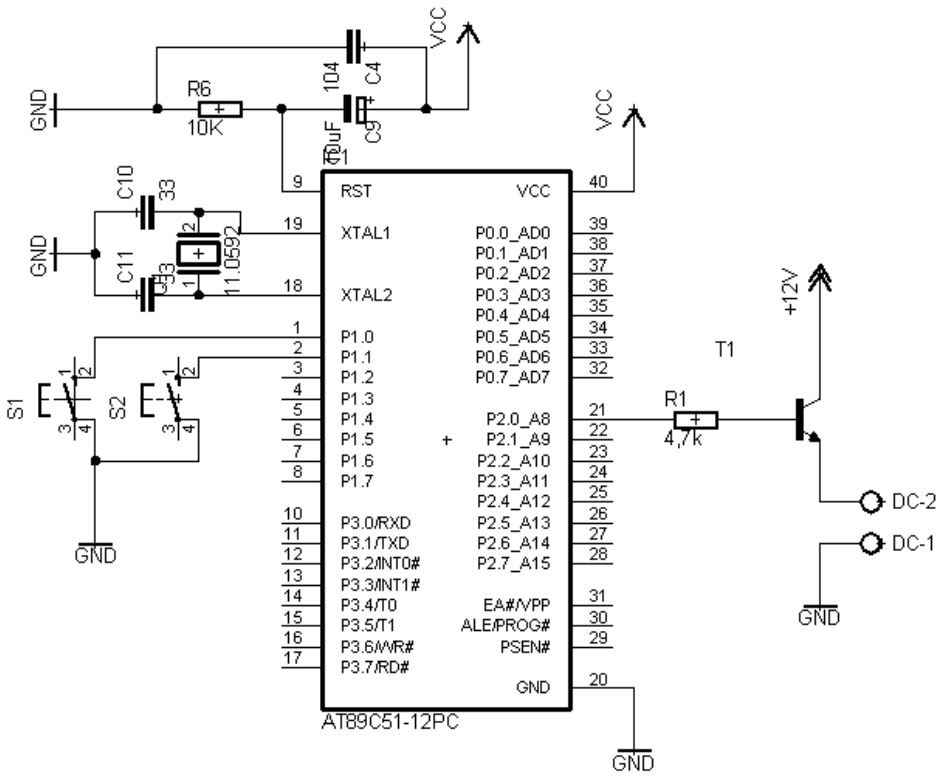
* Hàm trễ:

```
void delay(long time) {
 long n;
```

```
 for(n=0;n<time;n++) ;
}
* Hàm main:
void main(void) {
 char x;
LCDinit();
LCDputs("8052 MCU",1);
delay(30000);
while(1)  {
 for(x=0;x<16;x++)// Dich 16 lan.
 {
LCDputs("8052 MCU",1);
LCDcontrol(0x18);// Dich hien thi sang trai.
delay(5000);// Tre
 }
 }
 }
```

3.7 Điều khiển động cơ DC

3.7.1 Mạch nguyên ly



Nhiệm vụ: Tạo ra xung có độ rộng thay đổi, 10 cấp, tần số 1Khz, để điều khiển tốc độ động cơ (10 cấp tốc độ).

3.7.2 Lập trình:

- Cách tạo xung có độ rộng thay đổi bằng VĐK.

+ Cách 1: Như các bạn điều khiển nhấp nháy 1 con led, đó là tạo ra 1 xung ở 1 chân của vi điều khiển, nhưng xung đó có độ rộng cố định, tần số lớn, cách bạn có thể điều chỉnh lại hàm delay để tần số của nó đúng 1 Khz. Tuy nhiên vì là dùng hàm delay nên trong

thời gian có xung lên 1(5V) và thời gian không có xung(0V) vi điều khiển không làm gì cả, hơn nữa tạo xung bằng việc delay mà các bạn có nhu cầu cần 2 bộ phát xung ở 2 kênh, có cùng tần số mà khác độ rộng xung thì trở nên rất khó khăn. Cho nên chúng ta dùng bộ định thời Timer của vi diều khiển trong trường hợp này rất tiện.

+ Cách 2: Dùng ngắt Timer của bộ vi điều khiển. Trước hết nhắc lại về ngắt của vi điều khiển:

+ Ngắt là gì ? để trả lời câu hỏi này tôi xin trích đoạn về ngắt timer:

- Hàm ngắt: Cấu trúc: Void Tênhàm(void) interrupt nguồnngắt using bangthanhghi

{ // Chuong trinh phuc vu ngat o day }

+ Hàm ngắt không được phép trả lại giá trị hay truyền biến vào hàm.

+ Tên hàm bất kì.

+ interrupt là từ khóa phân biệt hàm ngắt với hàm thường.

+ Nguồn ngắt từ 0 tới 5 theo bảng vector ngắt.

+ Bảng thanh ghi trên ram chọn từ 0 đến 3.

Tùy theo bạn viết hàm ngắt cho nguồn nào bạn chọn nguồn ngắt từ bảng sau:

Ngắt do	Cờ	Địa chỉ vector
Reset hệ thống	RST	0000H
Ngắt ngoài 0	IE0	0003H
Bộ định thời 0	TF0	000BH
Ngắt ngoài 1	IE1	0013H
Bộ định thời 1	TF1	001BH
Port nối tiếp	RI hoặc TI	0023H
Bộ định thời 2	TF2 hoặc EXF2	002BH

- Về using 0: Có 4 bang thanh ghi bạn có thể chọn cho chuong trình phục vụ ngắt, cái này cũng không quan trọng. Trong hàm ngắt các bạn có thể bỏ đi từ using 0, khi đó vi điều khiển sẽ tự sắp xếp là dùng bang thanh ghi nào.

- Hàm ngắt khác hàm bình thuờng chỗ nào. Hàm bình thuờng ví dụ hàm delay, cứ khi

bạn gọi nó thì nó sẽ được thực hiện, có nghĩa là nó có vị trí cố định trong tiến trình hàm main, có nghĩa là bạn biết nó xảy ra khi nào. Còn hàm ngắt thì không có tiến trình cố định, điều kiện ngắt có thể xảy ra bất kì lúc nào trong tiến trình hàm main và cứ khi nào có điều kiện ngắt thì hàm ngắt sẽ được gọi tự động.

- Để sử dụng ngắt ta phải làm các công việc sau:

1) Khởi tạo ngắt: dùng ngắt nào thì cho phép ngắt đó hoạt động bằng cách gán giá trị tương ứng cho thanh ghi cho phép ngắt IE(Interrupt Enable):

IE là thanh ghi có thể xử lí từng bít. Ví dụ : bạn muốn cho phép ngắt timer 1 bạn dùng

lệnh: ET1=1; Không cho phép nữa bạn dùng lệnh : ET1=0; Hoặc bạn có thể dùng lệnh IE= 0x08; thì bit 3 của thanh ghi IE tức(IE) sẽ lên 1. Nhưng cách thứ nhất tiện hơn.

2) Cấu hình cho ngắt: Trong 1 ngắt nó lại có nhiều chế độ ví dụ: với ngắt timer. Bạn phải cấu hình cho nó chạy ở chế độ nào, chế độ timer hay counter, chế độ 16 bit, hay 8 bit,… bằng cách gán các giá trị tuong ứng cho thanh ghi TMOD(Timer MODe).

3)Bắt đầu chương trình có ngắt:

-Trước khi bắt đầu cho chạy chuong trình ta phải cho phép ngắt toàn cục được xảy ra bằng cách gán EA (Enable All interrupt) bằng 1, thì ngắt mới xảy ra.

-Thường thì ngay vào đầu chương trình (hàm main) trước vòng while(1) chúng ta đặt công việc khởi tạo, cấu hình và cho phép kiểm tra ngắt.

```
void khoitaotimer0(void)// Ham khoi tao
{
 EA=0;// Cam ngat toan cuc
TMOD=0x02;// Timer 0 che do 2 8 bit auto reload
TH0=0x9B;// Gia tri nap lai 155 doi ra so hex
TL0=0x9B;// Gia tri khoi tao 155 doi ra so hex
ET0=1;// Cho phep ngat timer 0
EA=1;// Cho phep ngat toan cuc
TR0=1;// Chay timer 0 bat dau dem so chu ki may
}
```

* Hàm ngắt:

```
unsigned char dem=0;// Khai bao bien dem de dem tu 1 den 10
 unsigned char phantramxung;// Bien chua phan tram xung(0...10)
 void timer0(void) interrupt 1 //Ngat timer 0
 { TR0=0;// Dung chay timer 0
TF0=0;// Xoa co, o che do co tu duoc xoa,che do khac can toi cu viet vao day
dem++;
if(dem<phantramxung) P2_0=1;// Neu bien dem < phan tram xung thi dua gia tri 1
ra //chan, xung 5V
 else P2_0=0;// Neu dem = phan tram xung
 if(dem==10) dem=0;// Neu dem du 10 thi gan lai bang 0 de bat dau chu ki moi
TR0=1;
// Cho chay timer
 }
```

Để có thể thay đổi độ rộng xung thì ta lưu độ rộng xung vào 1 biến, vì hàm ngắt không cho truyền biến vào ta khai báo biến đó là biến toàn cục để có thể gán giá trị ở mọi hàm. 100 uS ngắt 1 lần để xác định đủ chu kì 1000 uS ta cần đếm từ 1 đến 10 ta khai báo biến đếm.

```
void timer0(void) interrupt 1 //Ngat timer 0
{
TR0=0;// Dung chay timer 0
TF0=0;// Xoa co
TH0=0xAB;
TL0=0xAB;
 ….
TR0=1;// Cho chay timer
}
```

Cấu trúc hàm ngắt timer nào cung phải theo, do chế độ 2 tự động nạp lại nên không cần gán giá trị cho TH0 và TL0. Về biến đếm sẽ đếm từ 1 đến 10 nếu bằng 10 kết thúc 1 chu kì 10*100 =1000 uS, ta gán lại nó bằng 0 để sang chu kì mới.

```
if(dem<phantramxung) P2_0=1;// Neu bien dem < phan tram xung thi đua gia tri 1
ra //chan, xung 5V
else P2_0=0;// Neu dem = phan tram xung
```

Câu lệnh này kiểm tra nếu đếm nhỏ hon phantramxung thì sẽ đua ra cổng giá trị 1, bằng hoặc lớn hơn sẽ đưa ra giá trị 0. Khi vào chương trình chính ta chỉ việc thay đổi giá trị biến phantramxung thì độ rộng xung sẽ thay đổi. * Hàm main:

```
void main(void) {
khoitaotimer0();
while(1) {
phantramxung=9;
delaylong(20000);
phantramxung=4;
delaylong(20000);
}
}
```

Giả sử khi các bạn gán

phantramxung=4;

Thì cứ mỗi 100uS ngắt xảy ra 1 lần, và kiểm tra

biến dếm. Lần đầu đếm=1 <4 nên giá trị P2_0 = 1 mức cao, lần thứ 2 , 200 uS, dem =2<4 P2_0 = 1 mức cao, lần thứ 3, 300uS, dem=3<4, P2_0=1 mức cao, lần thứ 4, 400uS, dem =4 <4 sai, P2_0=0, bắt dầu xuỗng mức thấp, có xung từ cao xuống thấp, dem = 5<4 sai , P2_0=0 mức thấp, …, dem =10 <4 sai P2_0 mức thấp dủ 1000 uS , 400uS cao, 600uS

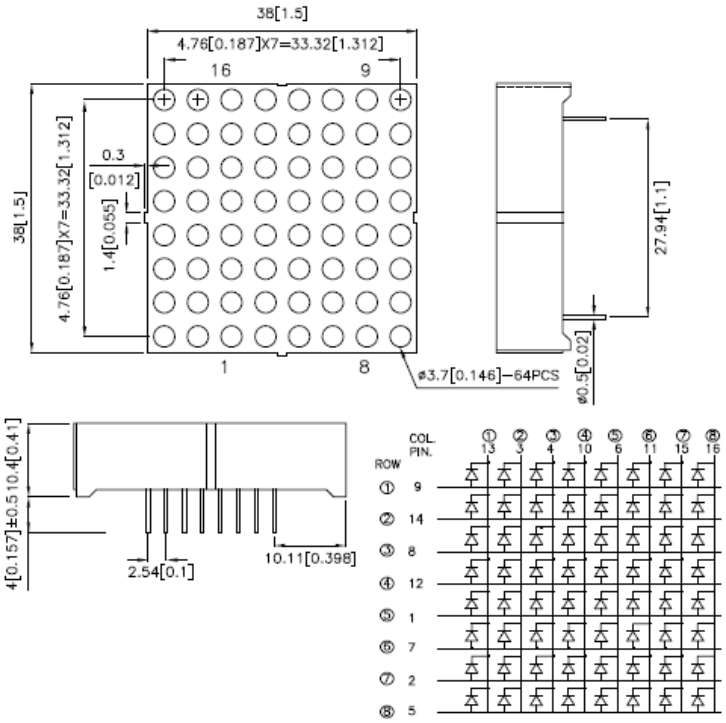
thấp quay vòng dem=0, ngắt lần thứ 11, dem=1 < 4 , P2_0=1 mức cao, có xung thấp lên cao…. Để PWM 2 chân P2_0 và P3_5, các bạn khai báo thêm 1 biến phantramxung2 và đua thêm dòng l ệnh sau vào hàm ngắt. if(dem<phantramxung) P3_5=1;// Neu bien dem < phan tram xung thi đua gia tri 1 ra //chan, xung 5V

else P3_5=0;// Neu dem = phan tram xung

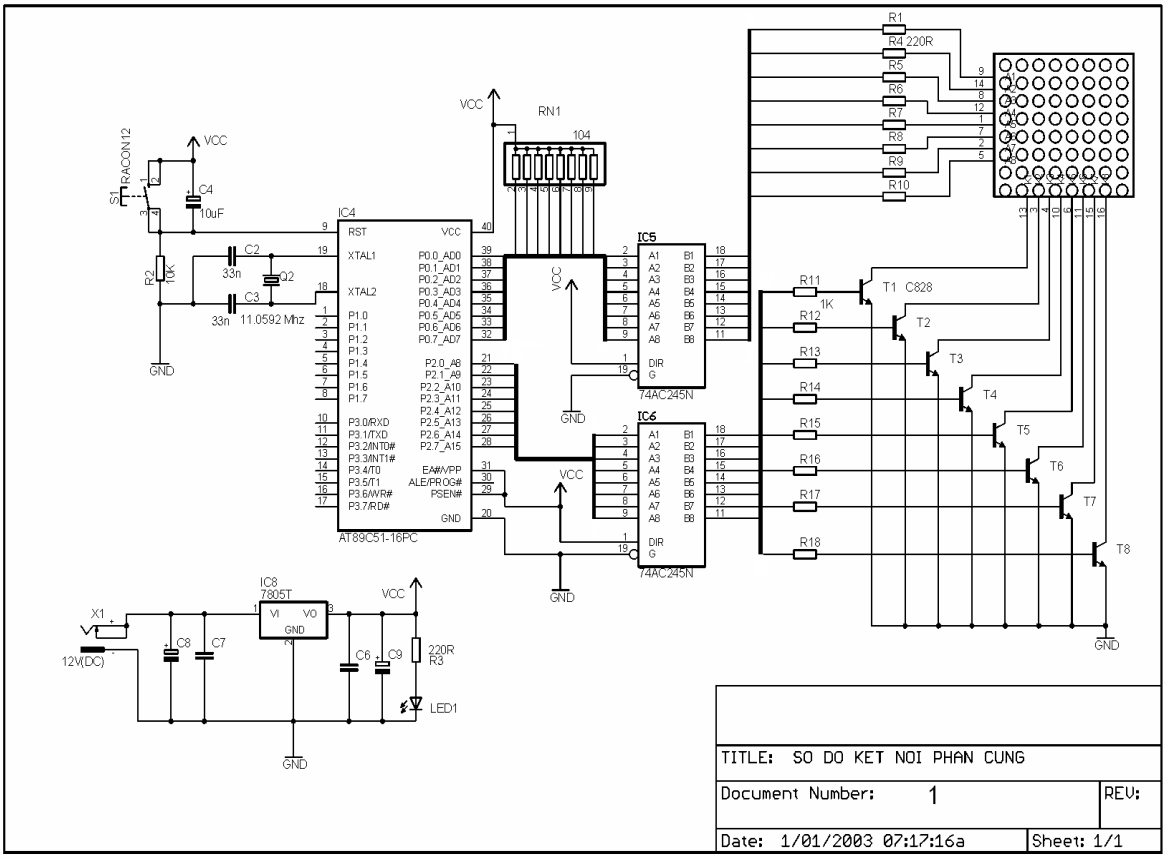
3.8 Ghép nối Matrix Led

- Dạng Led:

3.8.1 Sơ đồ cấu tạo:



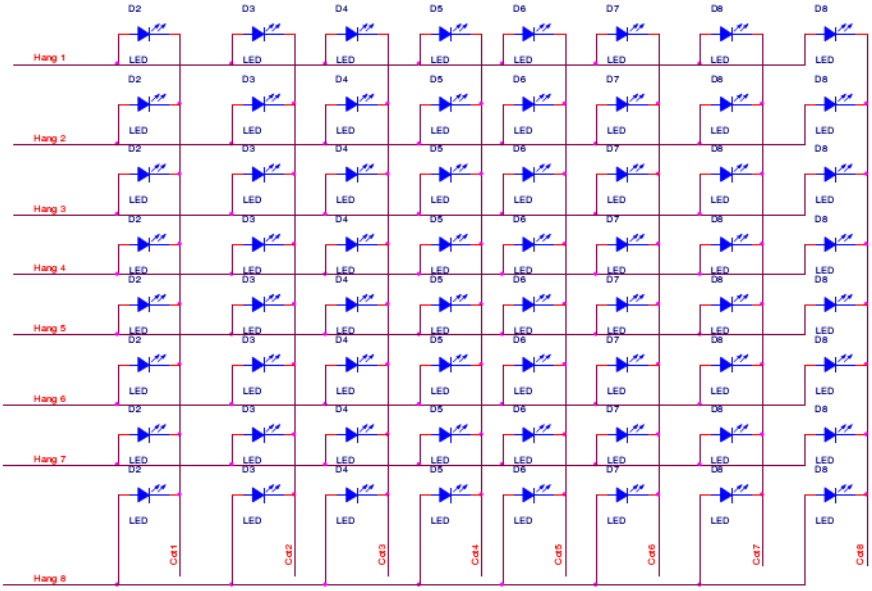
3.8.2 Sơ đồ kết nối Matrix Led 8x8:



Nhiệm vụ:

Điều khiển Led ma trận 8x8. Hiển thị dòng chữ chạy “MTC”.

3.8.3 Nguyên lí hoạt động:



Muốn cho led sáng, cấp điện dương 5V vào hàng, 5V vào cột, dòng 10mA đến 15 mA. Ví dụ: muốn đèn led ở vị trí 5x4 sáng, ta đưa điện áp cột 4(P2_3) lên 5V, điện áp hàng 5(P2_5) lên 5V. Hiển thị chữ: thống kê các điểm sáng thành chữ rồi cho các hàng cột điện áp tương ứng. Có thể dùng công cụ debug để lấy giá trị cổng tương ứng với các led sáng. Giống như quét bàn phím, đưa điện áp 0V ra từng cột nối với cổng 0. Như vậy sẽ có 8 giá trị: 0xFE, 0xFD, 0xFB, 0xF7, 0xEF, 0xDF, 0xBF, 0x7F phải đưa vào 1 mảng 8 phần tử, rồi sau đó đưa vào 1 vòng for tăng dần 1 biến để tăng phần tử mảng cột[8]. Với mỗi lần 1 chân cổng 0 lên 5V ta dùng cổng 2 đưa ra 1 giá trị 8 bít để điều khiển trong 1cột những đèn nào sáng. Ví dụ muốn hàng 1 và hàng 3 sáng thì hàng 1 và 3 có giá tr ị 5V còn các hàng khác 0V, ta được giá trị 8 bít sau: 0x05 (1010 000). Tại mỗi thời điểm chỉ có một số đèn trên 1 cột sáng, nhưng do ta quét 8 cột với tần số nhanh, vì mắt có hiện tượng lưu ảnh nên ta thấy trong 1 thời điểm ta thấy toàn bộ kí tự. Với 8 cột lần lượt bằng 5V ta phải đưa ra tương ứng 8 giá trị 8 bit ra cổng 2, do đó ta fải lưu 8 giá trị đó vào 1 mảng 8 kí tự_ kytu1[8], ta sẽ viết các ký tự trên 7 cột. Để mỗi kí tự sẽ cách nhau 1 cột không sáng. Ta khai báo mảng kytu1[9] có 9 phần tử và phần tử đầu tiên có giá trị đẩy ra cổng 2 là 0xff để tắt toàn bộ cột đó.

Quá trình điều khiển hiển thị như sau: Cột 1, hàng 1, cột 2 hàng 2, …, cột 8 , hàng 8. Để làm chữ chạy: Thêm 1 biến vào để điều khiển thứ tự hiển thị hàng. Hiển 1 chữ trên led như trên đã đưa ra: Cột 1, hàng 1, cột 2 hàng 2, …, cột 8 , hàng 8. Muốn chữ đó dịch chuyển sang trái ta hiển thị như sau: Cột 1, hàng 2, cột 2 hàng 3,

…, cột 7, hàng 8,cột 8 , hàng 1 ký tự sau. Cột 1, hàng 3, cột 2 hàng 4, …, cột 7 hàng 1 ký tự sau,cột 8 , hàng 2 ký tự sau.

3.8.4 Lập trình

```
#include <REGX51.H> /* Cot tu P2.0 den P2.7 Hang tu P0.0 den P0.7 De quet dua muc logic 5v lan luot ra cong P2 */
/* Ham tre */
void delay(long time) {
long n;
for(n=0; n<time; n++) {
 ; }
 }
unsigned char kytu1[9];// Mang 9 phan tu chua gia tri cac hang day ra cong 2 unsigned char k=0;// Bien xac dinh cac ky tu
 /* Ham nap gia tri hien thi cac ky tu vao mang kytu1 co 8 gia tri dua ra va 1 gia tri khong bat den nao de cac ky tu cach nhau 1 cot */
void mahoa(unsigned char x) {
switch(x) {
 // Dau trang
 case 0: {
kytu1[0]=0x00; kytu1[1]=0x00; kytu1[2]=0x00; kytu1[3]=0x00; kytu1[4]=0x00; kytu1[5]=0x00; kytu1[6]=0x00; kytu1[7]=0x00; kytu1[8]=0x00;
break;
} // Chu M
Case 1: { kytu1[0]=0x00; kytu1[1]=0xFF; kytu1[2]=0x02; kytu1[3]=0x04; kytu1[4]=0x08; kytu1[5]=0x04; kytu1[6]=0x02; kytu1[7]=0xFF; kytu1[8]=0x00;
break;
} // Chu T
case 2: {
kytu1[0]=0x00; kytu1[1]=0x01; kytu1[2]=0x01; kytu1[3]=0x01; kytu1[4]=0xFF; kytu1[5]=0x01; kytu1[6]=0x01; kytu1[7]=0x01; kytu1[8]=0x00;
break;
```

```
} // Chu C
case 3: {
kytu1[0]=0x00; kytu1[1]=0x7E; kytu1[2]=0x81; kytu1[3]=0x81; kytu1[4]=0x81;
kytu1[5]=0x81; kytu1[6]=0x42; kytu1[7]=0x00; kytu1[8]=0x00;
break;
 } // Dau trang
case 4: {
kytu1[0]=0x00; kytu1[1]=0x00; kytu1[2]=0x00; kytu1[3]=0x00; kytu1[4]=0x00;
kytu1[5]=0x00; kytu1[6]=0x00; kytu1[7]=0x00; kytu1[8]=0x00;
break;
}
}
} /* Ham quet led ma tran_ vua hien thi vua dich ky tu dan sang trai*/
void hienthi(void) {
 unsigned char n,m,lap;
unsigned char cot[8]={0x80,0x40,0x20,0x10,0x08,0x04,0x02,0x01};
 // Cac phan tu quet cot
for(m=0; m<8 ; m++)// Dich hien thi
{ for(lap=0; lap<10; lap ++) // Lap hien thi
 { for(n=0; n<8 ; n++)// Quet cot
{
if((n+m)<9 )// Neu n+m < 9 hien thi ky tu 1
{
 mahoa(k); // Nap cac gia tri ma hoa ky tu dua ra cac hang (Cong P0)
P2=cot[n]; // Day gia tri 5V ra cong P2 (cac cot)
 P0=kytu1[n+m];// Day cac gia tri cac hang (ma hoa ky tu) ra cong P0(cac hang)
delay(45);// Tre du de led sang
 }
if((n+m) > 7)// Neu n+m >7 hien thi ky tu 2 {
mahoa(k+1);// Nap gia tri ma hoa ky tu tiep de dua ra cac hang(Cong P0)
 P2=cot[n];// Day gia tri logic 5V ra cong P2(cac cot)
```

```
 P0=kytu1[n+m-8];// Day cac gia tri cac hang (ma hoa ky tu) ra cong P0(cac hang)
delay(45);// Tre du de led sang
      }
P2=0x00;// Day cac cot xuong muc thap
 P0=0x00;// Dua cac hang xuong thap de tat toan bo cac led.
 }
}
}
}
void main(void) {
while(1)// Vong lap vo han.
 {
 hienthi(); // Hien thi 2 ky tu dau tien dau trang va chu M
k=k+1; // Tang k de hien thi chu M va chu T lan tiep
if(k==4) k=0;// Quay vong hien thi
}
}
```

MỤC LỤC

Chương 1 : Ôn lại về ngôn ngữ C theo chuẩn ANSI

Chương 2: Ôn lại về vi điều khiển AT89C51

Chương 3 C cho vi điều khiển 8051