

2.5 Lập trình C với 8051

- Tham khảo Keil C51 Manual

<http://www.keil.com/support/man/docs/c51/>

- Công cụ: Keil C51 (bản dùng thử giới hạn 2KB code)
- Tuân thủ ANSI-C
- Thêm một số mở rộng (language extension) để hỗ trợ tính năng của 8051.

địa chỉ thanh
ghi, ngăn nhớ
ở địa chỉ 90

Kiểu dữ liệu đặc biệt

- sfr: thanh ghi

VD: sfr P1=0x90;

tại đây, thì khác vs C thông
thường. Còn lại thuần túy

- sbit: bit thuộc sfr

VD: sbit Led_pin=P1^1;

dùng nhiều, cho từng bit
P1^1 chân cổng số 1 của P1 =>
P1^1 đc ánh xạ vào địa chỉ 91

- sbit: bit, allocate
trong vùng bit-
addressable RAM

VD: bit value;

- Khi làm việc vs byte nhớ =>
byte addr ở tay trái
- Khi khai báo vs biến => bit
addr ở tay phải

Byte address	Bit address	Byte address	Bit address
7F	General purpose RAM	FF	
		F0	F7 F6 F5 F4 F3 F2 F1 F0 B
		E0	E7 E6 E5 E4 E3 E2 E1 E0 ACC
		D0	D7 D6 D5 D4 D3 D2 D1 D0 PSW
		B8	- - - BC BB BA B9 B8 IP
		B0	B7 B6 B5 B4 B3 B2 B1 B0 P3
		A8	AF - - AC AB AA A9 A8 IE
		A0	A7 A6 A5 A4 A3 A2 A1 A0 P2
		99	not bit addressable SBUF
		98	9F 9E 9D 9C 9B 9A 99 98 SCON
		90	97 96 95 94 93 92 91 90 P1
		8D	not bit addressable TH1
		8C	not bit addressable TH0
		8B	not bit addressable TL1
		8A	not bit addressable TL0
		89	not bit addressable TMOD
		88	8F 8E 8D 8C 8B 8A 89 88 TCON
		87	not bit addressable PCON
		83	not bit addressable DPH
		82	not bit addressable DPL
		81	not bit addressable SP
		80	87 86 85 84 83 82 81 80 P0

Bản đồ bộ nhớ 8051.

Chú ý phân biệt địa chỉ bit và byte

Ví dụ chương trình LedBlinky

```
//Nhấp nháy led trên chân P1.0
```

```
#include <at89x51.h>
```

```
void delay(int interval){  
    int i,j;  
    for(i=0; i<255; i++){  
        for(j=0; j<interval; j++);  
    }  
}
```

record 22/4 giới thiệu về tool keil

```
void main(){  
    while(1){  
        P1_0=1;  
        delay(100);    //Tre mot khoang thoi gian  
        P1_0=0;  
        delay(100);    //Tre mot khoang thoi gian  
    }  
}
```

bật tắt chân 0 ở cổng P1

Làm việc với Keil C51

LED blinking - µVision4

File Edit View Project Flash Debug Peripherals Tools SVCS Window Help



Registers

Register	Value
Reg	
r0	0x00
r1	0x00
r2	0x00
r3	0x00
r4	0x00
r5	0x00
r6	0x00
r7	0x00
Sys	
a	0x00
b	0x00
sp	0x07
sp_max	0x07
dptr	0x0000
PC	0x0825
states	389

Disassembly

```
9: void main() {  
10:     while(1) {  
11:         P1_0=1;  
12:         C:0x0825 D290 SETB P1_0(0x90.0)
```

main.c

```
11:         P1_0=1;  
12:         delay(100); //Tre mot khoang thoi gian  
13:         P1_0=0;  
14:         delay(100); //Tre mot khoang thoi gian  
15:     }
```

Parallel Port 1

Port 1

P1: 0xFF 7 Bits 0

Pins: 0xFF

Command

Running with Code Size Limit: 2K
Load "C:\\Users\\ngola\\OneDrive\\Documents\\Giang day\\He nhung\\2019-202
BS \\MAIN\\11
BS \\MAIN\\13

Symbols

Name	Address	Type
VT Simulator VTREG		
Peripheral Registers		
LED blinking		Application
Runtime Library		
MAIN		Module

ASM ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet

Call Stack Locals Watch 1 Memory 1 Symbols

Simulation

t1: 0.00019450 sec

CAP NUM SCRL OVR R/W

LED Blinky v2

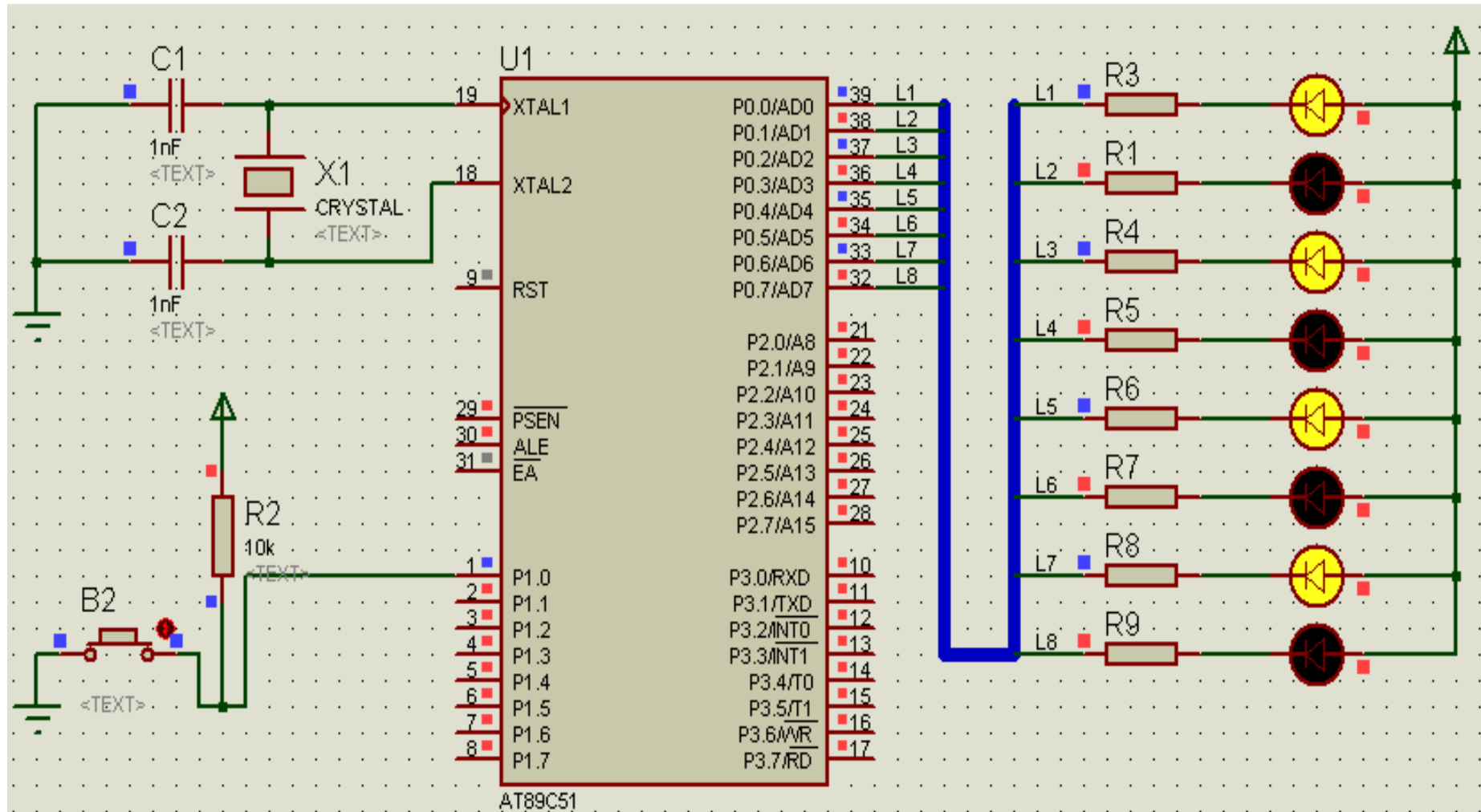
```
//Nhập nháy led trên chân P1.0
#include <at89x51.h>
sbit LED_pin = P1^0;
bit LED_data = 0;
void delay(int interval){
    int i,j;
    for(i=0; i<255; i++){
        for(j=0; j<interval; j++);
    }
}
void main(){
    while(1){
        LED_pin = LED_data;
        LED_data = ~LED_data;
        delay(100); //Tre mot khoang thoi gian
    }
}
```

sbit: là bit được đặt vào cố định địa vào
bit0 ở địa chỉ P1 (là đc90)
bit: 1 trong 256 byte của chip

lấy dl LED data ở LED_pin

Mỗi chu kì đảo chiều 1 lần

Lập trình với cổng vào ra



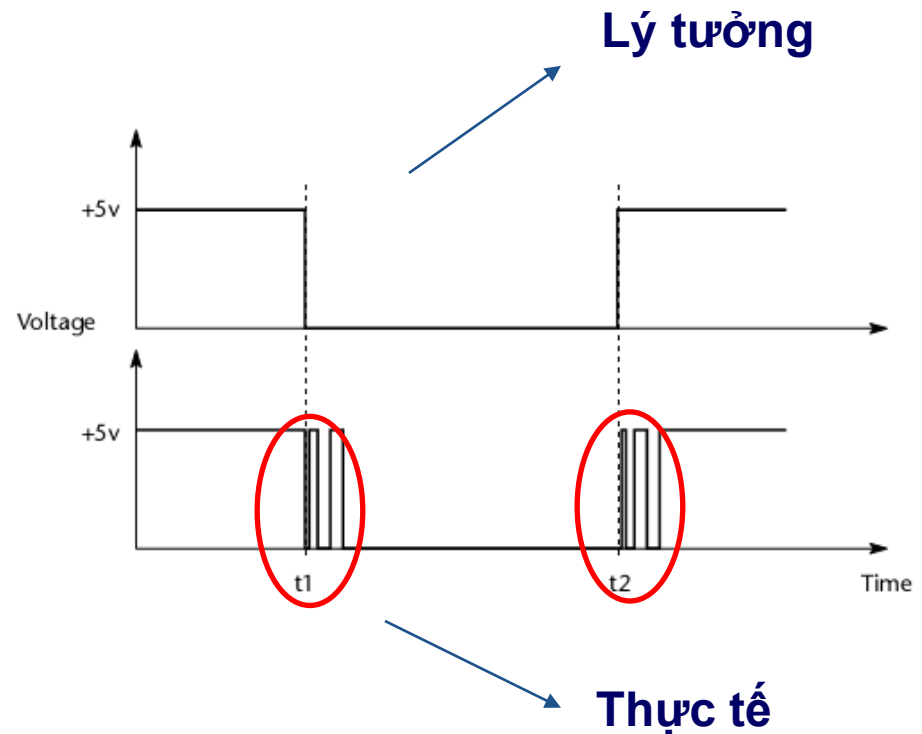
Đọc dữ liệu từ chân vào

```
//Đọc dữ liệu từ chân vào P1_0 ghép với nút bấm
#include <at89x51.h>
void delay(int interval){
    int i,j;
    for(i=0;i<255;i++){
        for(j=0;j<interval;j++);
    }
}
void main(){
    while(1){
        //Kiem tra trạng thái chân P1_0 (dau voi cong tac)
        if(P1_0 == 1){
            P0=0x55;
            delay(100);
            P0=0xAA;
            delay(100);
        }
    }
}
```

thêm P1_0=1;
//để đặt P1_0 ở trạng thái input
cho an toàn

Hiện tượng nảy phím (key bounce)

- Khi sử dụng các phím bấm cơ khí sẽ có hiện tượng nảy phím, đó là hiện tượng tín hiệu thay đổi mức liên tục trong một khoảng thời gian ngắn trước khi chuyển sang trạng thái ổn định.
- Dẫn đến số lần bấm phím sẽ không còn chính xác.
- Giữa thời điểm nhấn phím và nhả phím sẽ xuất hiện nhiều lần “giả” bấm và nhả phím



- Chống nảy phím bằng phần mềm
 - Thêm một khoảng trễ (khoảng 10-20ms) sau khi nhận sự kiện nhấn phím đầu tiên -> bỏ qua thời gian nảy phím, rồi đọc lại giá trị nút nhấn.
 - Đọc liên tục nhiều lần theo chu kỳ ngắn (dùng timer): <https://www.embedded.com/my-favorite-software-debouncers/>
- Chống nảy phím bằng phần cứng: tụ điện

Chống náy phím bằng phần mềm

```
void main(){
    int count=0;
    P1_0=1;//Chặn P1_0 làm chặn vào
    while(1){
        if(P1_0==0){
            count++;
        }
        delay(10); //Tạo trễ chống náy phím
        //đọc P1_0 và kiểm tra lại
    }
}
```

Lập trình ngắt với C51

■ Hàm xử lý ngắt

`void Tên_chương_trình_con()` **interrupt** Số_hiệu

- Tên chương trình con ngắt: do người lập trình đặt, tuân thủ các quy tắc giống như chương trình con thông thường
- Số hiệu ngắt: số hiệu vector ngắt (Xem trong file at89x51.h)

```
#define IE0_VECTOR      0  /* External Interrupt 0 */
#define TF0_VECTOR      1  /* Timer 0 */
#define IE1_VECTOR      2  /* External Interrupt 1 */
#define TF1_VECTOR      3  /* Timer 1 */
#define SIO_VECTOR      4  /* Serial port */
```

Ví dụ: ghép nối ngắt ngoài 0

- Code mẫu chương trình xử lý ngắt ngoài 1 (chế độ ngắt theo sườn)

```
//External interrupt 1 handler
void EX1_Process() interrupt IE1_VECTOR {
    /*short job here*/
}

//main prog
void main(){
    IE=0x84;    //Cho phép ngắt ngoài 1
    IT1=1;      //Chế độ ngắt theo sườn
    while(1){
        /*main loop*/
    }
}
```

Lập trình ghép nối timer

- TMOD
- TCON
- Interrupt handler
 - Reset timer value
- Start/stop timer

Các thanh ghi Counter/Timer

■ Thanh ghi TCON

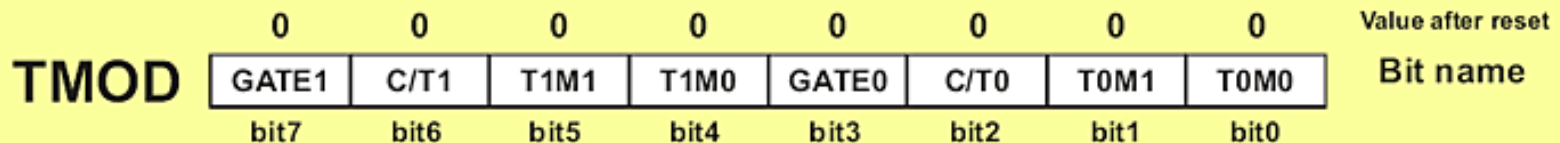
- TR1/TR0: bit khởi động/tắt
- TF1/TF0: cờ báo tràn bộ đếm/định thời
- IE1, IT1, IE0, IT0: liên quan tới ngắt phần cứng ngoài

TCON	0	0	0	0	0	0	0	Value after Reset
	TF1	TR1	TF0	TR0	IE1	IT1	IE0	Bit name
	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0

Các thanh ghi Counter/Timer

■ Thanh ghi TMOD

- Gate: sử dụng cho bộ đếm
- C/T: chọn chế độ (bộ đếm hay bộ định thời)
 - ✓ C/T=0: bộ định thời
 - ✓ C/T=1: bộ đếm
- M1,M0: chọn chế độ làm việc. Hai chế độ thông dụng
 - ✓ M1=0, M0=1: chế độ 1, bộ Timer/Counter 16 bit
 - ✓ M1=1, M0=0: chế độ 2, bộ Timer/Counter 8 bit tự động nạp lại



Định thời dùng polling

```
#include <at89x51.h>
//Chương trình tạo độ trễ chính xác sử dụng Timer
void delay_hardware_50ms(){
    //Xóa phần thiết lập cũ của Timer0
    TMOD=TMOD & 0xF0;
    TMOD=TMOD | 0x01;
    ET0=0;      //Không phát sinh ngắt
    TH0=0x3C;   //Khởi tạo giá trị là 3CB0
    TL0=0xB0;   //Tương đương 15536
    TF0=0;      //Xóa cờ tràn timer 0
    TR0=1;      //Khởi động timer 0
    while(TF0==0); //Chờ đến khi tràn
    TR0=0;      //Dừng timer 0
}
```


Định thời dùng interrupt

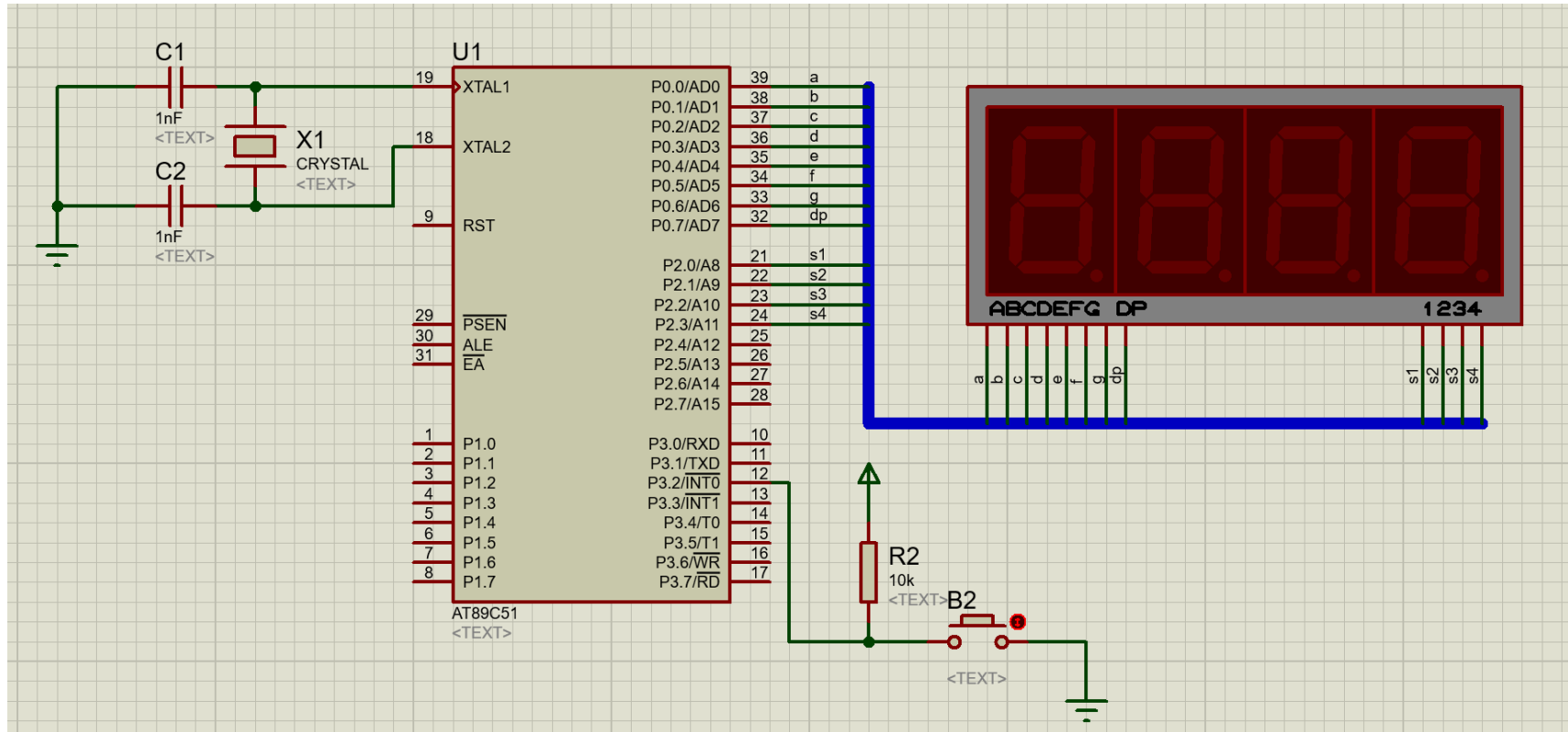
```
void initilize_timer0()
{
    TMOD = TMOD & 0xF0;
    TMOD = TMOD | 0x01;
    TH0 = 0x3C;      //Khoi tao T0
    TL0 = 0xB0;      //Tuong duong 15536
    TF0 = 0;         //Xoa co tran timer 0
    TR0 = 1;         //Khoi dong timer 0
    ET0 = 1;         //cho phep ngat
    EA = 1;          //global interrupt enable
}
```

Định thời dùng interrupt

//Timer 0 interrupt handler

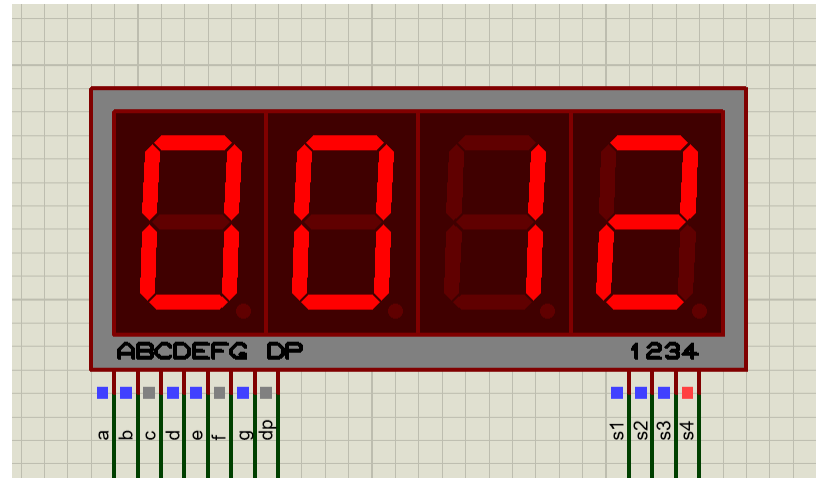
```
void TMR0_Process() interrupt TF0_VECTOR{  
    TF0    = 0;                //clear flag  
    TR0    = 0;                //stop timer  
    TH0    = TH0_50ms;         //reset T0 value  
    TL0    = TL0_50ms;         //  
    /*Do the job here*/  
    TR0    = 1;                //restart timer  
}
```

Ghép nối LED 7 thanh (LED scanning)



- Không thể hiện cùng lúc tất cả 4 module
 - Liên tục hiển thị lần lượt từng module
 - Do lưu ảnh võng mạc → giống như cả 4 cùng sáng

LED scanning



**4x7SEG
common
anode**

■ Step 1:

- $s_1s_2s_3s_4 = 1000$
- a..f = "0"
- delay



■ Step 2:

- $s_1s_2s_3s_4 = 0100$
- a..f = "0"
- delay



■ Step 3:

- $s_1s_2s_3s_4 = 0010$
- a..f = "1"
- delay



■ Step 4:

- $s_1s_2s_3s_4 = 0001$
- a..f = "0"
- delay



```
void main(){
    init();
    while(1){
        display_number(count);
    }
}

void init(){
    P3_2 = 1;           //P3_2 input for interrupt 0
    EX0  = 1;           //Cho phép ngắt ngoài 0
    IT0  = 1;           //Ngắt theo sườn
    ET   = 1;           //Global interrupt
}
```

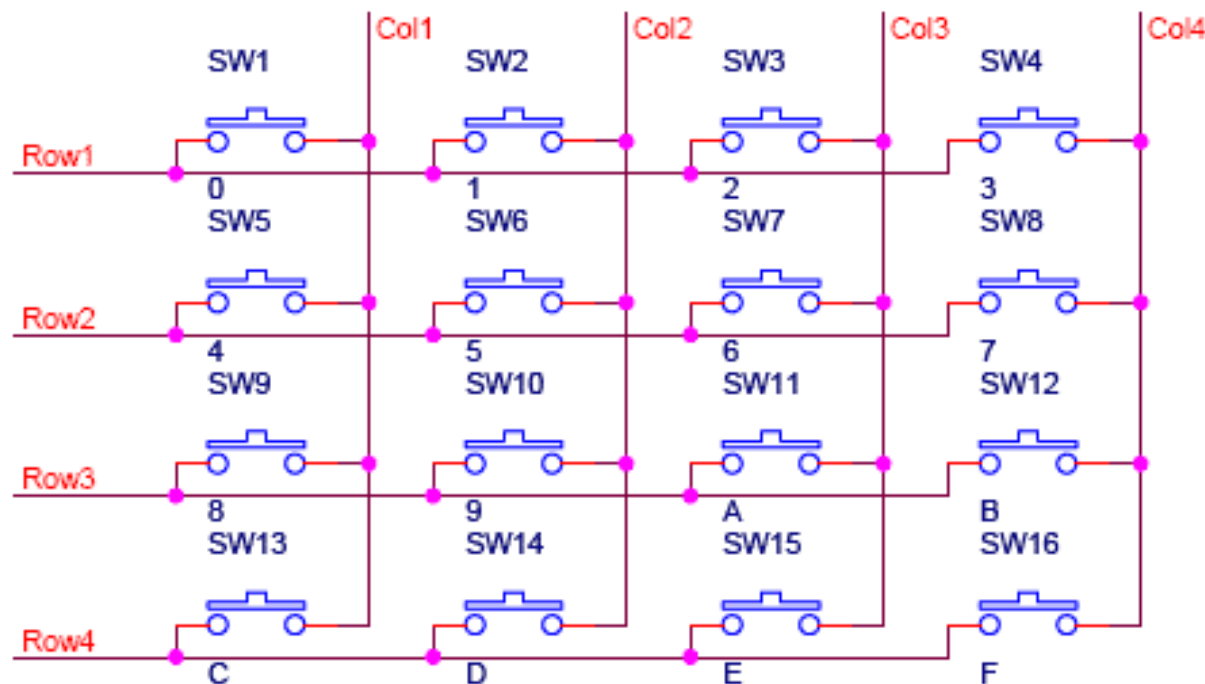
```
//display 4 digits
void display_number(int iNum){
    int i;
    unsigned char pos=0x08;
    unsigned char temp;
    for(i=0; i<4; i++){
        temp = iNum % 10;
        iNum = iNum / 10;
        P2 = pos;
        output_7seg(temp);
        delay(5);
        pos = pos>>1;
    }
}
```

```
//EXT 0 interrupt handler
void EXT0_Process() interrupt IE0_VECTOR{
    EA=0;          //Cam ngat
    count++;
    EA=1;          //Cho phep ngat
}
//7-seg display
void output_7seg(unsigned char value)
{
    unsigned char const mask[10]={0xC0, 0xF9,
0xA4, 0xB0, 0x99, 0x92, 0x82, 0xF8, 0x80, 0x90};
    if(value < 10){
        P0=mask[value];
    }
}
```

- Hiển thị bằng timer, không dùng main loop
- Không dùng toán tử %, /

Ghép nối bàn phím (keypad)

- Cấu trúc bàn phím ma trận (keypad)



Hoạt động của ma trận phím

- Bình thường, tín hiệu tại các cột là mức cao
- Tín hiệu mức thấp được đưa tới tất cả các hàng
- Khi có nút được bấm trên cột nào thì tín hiệu đọc tại cột đó sẽ ở mức thấp
- Để xác định được chính xác nút ở hàng nào, cột nào được bấm thì phải thực hiện thủ tục quét phím
 - Mỗi thời điểm chỉ đưa tín hiệu mức thấp tới 1 hàng, các hàng còn lại đưa tín hiệu mức cao
 - Kiểm tra tín hiệu tại các cột

Ví dụ ghép nối với ma trận phím

Nguyên tắc hoạt động?

