CHƯƠNG I : GIỚI THIỆU VỀ VĐK 89C51

CHƯƠNG I : GIỚI THIỆU VỀ VĐK 89C51

I. GIỚI THIỆU CẤU TRÚC PHẦN CỨNG HỌ MSC-51 (8951) :

Đặc điểm và chức năng hoạt động của các IC họ MSC-51 hoàn toàn tương tự như nhau. Ở đây giới thiệu IC8951 là một họ IC vi điều khiển do hãng Intel của Mỹ sản xuất. Chúng có các đặc điểm chung như sau:

Các đặc điểm của 8951 được tóm tắt như sau :

√ 8 KB EPROM bên trong.

√ 128 Byte RAM nội.

√ 4 Port xuất /nhập I/O 8 bit.

√ Giao tiếp nối tiếp.

√ 64 KB vùng nhớ mã ngoài

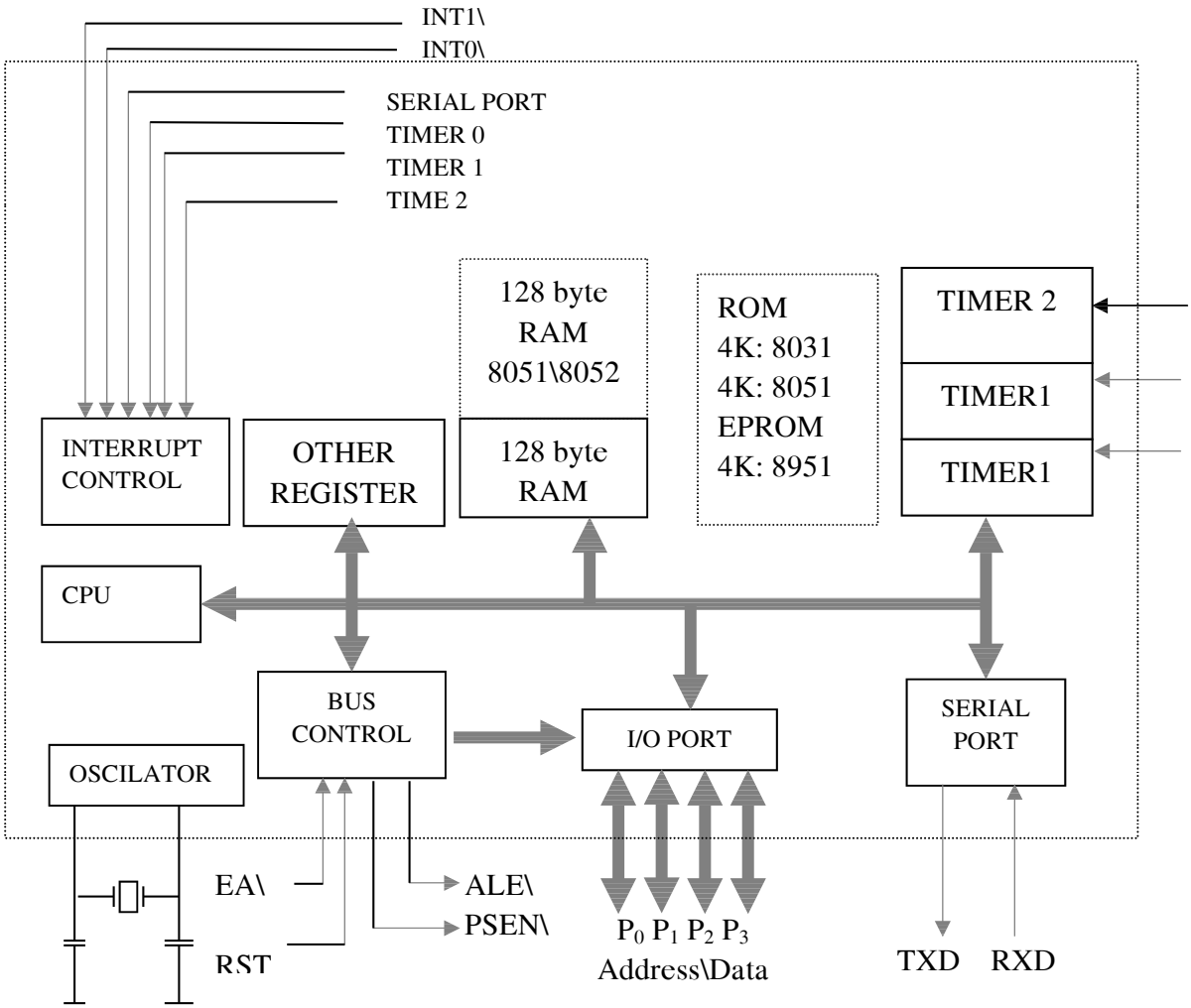
√ 64 KB vùng nhớ dữ liệu ngoại.

√ Xử lí Boolean (hoạt động trên bit đơn).

√ 210 vị trí nhớ có thể định vị bit.

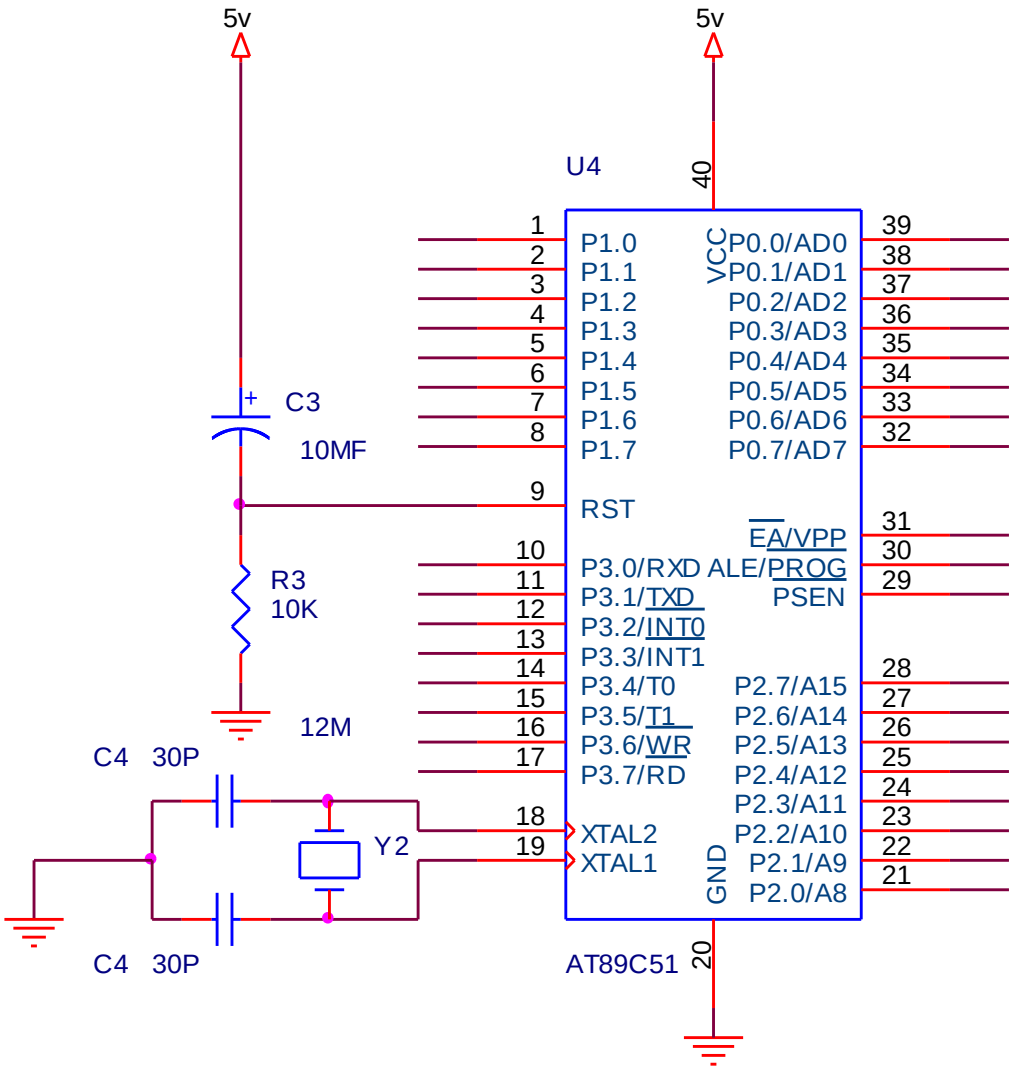
√ 4 μs cho hoạt động nhân hoặc chia.

Sơ đồ khối của 8951:



II. KHẢO SÁT SƠ ĐỒ CHÂN 8951, CHỨC NĂNG TỪNG CHÂN:

1.Sơ đồ chân 8951:



Sơ đồ chân IC 8951

2. Chức năng các chân của 8951:

- 8951 có tất cả 40 chân có chức năng như các đường xuất nhập. Trong đó có 24 chân có tác dụng kép (có nghĩa 1 chân có 2 chức năng), mỗi đường có thể hoạt động như đường xuất nhập hoặc như đường điều khiển hoặc là thành phần của các bus dữ liệu và bus địa chỉ.

a.Các Port:

- **Port 0 :**

- Port 0 là port có 2 chức năng ở các chân 32 – 39 của 8951. Trong các thiết kế cỡ nhỏ không dùng bộ nhớ mở rộng nó có chức năng như các đường IO. Đối với các thiết kế cỡ lớn có bộ nhớ mở rộng, nó được kết hợp giữa bus địa chỉ và bus dữ liệu.

- **Port 1:**

- Port 1 là port IO trên các chân 1-8. Các chân được ký hiệu P1.0, P1.1, P1.2, ... có thể dùng cho giao tiếp với các thiết bị ngoài nếu cần. Port 1 không có chức năng khác, vì vậy chúng chỉ được dùng cho giao tiếp với các thiết bị bên ngoài.

r **Port 2 :**

- Port 2 là 1 port có tác dụng kép trên các chân 21 - 28 được dùng như các đường xuất nhập hoặc là byte cao của bus địa chỉ đối với các thiết bị dùng bộ nhớ mở rộng.

r **Port 3:**

- Port 3 là port có tác dụng kép trên các chân 10 - 17. Các chân của port này có nhiều chức năng, các công dụng chuyển đổi có liên hệ với các đặc tính đặc biệt của 8951 như ở bảng sau:

Bit	Tên	Chức năng chuyển đổi
P3.0	RXT	Ngõ vào dữ liệu nối tiếp.
P3.1	TXD	Ngõ xuất dữ liệu nối tiếp.
P3.2	INT0\	Ngõ vào ngắt cứng thứ 0.
P3.3	INT1\	Ngõ vào ngắt cứng thứ 1.
P3.4	T0	Ngõ vào của TIMER/COUNTER thứ 0.
P3.5	T1	Ngõ vào của TIMER/COUNTER thứ 1.
P3.6	WR\	Tín hiệu ghi dữ liệu lên bộ nhớ ngoài.
P3.7	RD\	Tín hiệu đọc bộ nhớ dữ liệu ngoài.

Các ngõ tín hiệu điều khiển :

r **Ngõ tín hiệu PSEN (Program store enable):**

- PSEN là tín hiệu ngõ ra ở chân 29 có tác dụng cho phép đọc bộ nhớ chương trình mở rộng thường được nói đến chân 0E\ (output enable) của Eprom cho phép đọc các byte mã lệnh.

- PSEN ở mức thấp trong thời gian Microcontroller 8951 lấy lệnh. Các mã lệnh của chương trình được đọc từ Eprom qua bus dữ liệu và được chốt vào thanh ghi lệnh bên trong 8951 để giải mã lệnh. Khi 8951 thi hành chương trình trong ROM nội PSEN sẽ ở mức logic 1.

r **Ngõ tín hiệu điều khiển ALE (Address Latch Enable) :**

- Khi 8951 truy xuất bộ nhớ bên ngoài, port 0 có chức năng là bus địa chỉ và bus dữ liệu do đó phải tách các đường dữ liệu và địa chỉ. Tín hiệu ra ALE ở chân thứ 30 dùng làm tín hiệu điều khiển để giải đa hợp các đường địa chỉ và dữ liệu khi kết nối chúng với IC chốt.

- Tín hiệu ra ở chân ALE là một xung trong khoảng thời gian port 0 đóng vai trò là địa chỉ thấp nên chốt địa chỉ hoàn toàn tự động.

Các xung tín hiệu ALE có tốc độ bằng 1/6 lần tần số dao động trên chip và có thể được dùng làm tín hiệu clock cho các phần khác của hệ thống. Chân ALE được dùng làm ngõ vào xung lập trình cho Eprom trong 8951.

- **Ngõ tín hiệu EA\(External Access):**

- Tín hiệu vào EA\ ở chân 31 thường được mắc lên mức 1 hoặc mức 0. Nếu ở mức 1, 8951 thi hành chương trình từ ROM nội trong khoảng địa chỉ thấp 8 Kbyte. Nếu ở mức 0, 8951 sẽ thi hành chương trình từ bộ nhớ mở rộng. Chân EA\ được lấy làm chân cấp nguồn 21V khi lập trình cho Eprom trong 8951.

- **Ngõ tín hiệu RST (Reset)** :

-Ngõ vào RST ở chân 9 là ngõ vào Reset của 8951. Khi ngõ vào tín hiệu này đưa lên cao ít nhất là 2 chu kỳ máy, các thanh ghi bên trong được nạp những giá trị thích hợp để khởi động hệ thống. Khi cấp điện mạch tự động Reset.

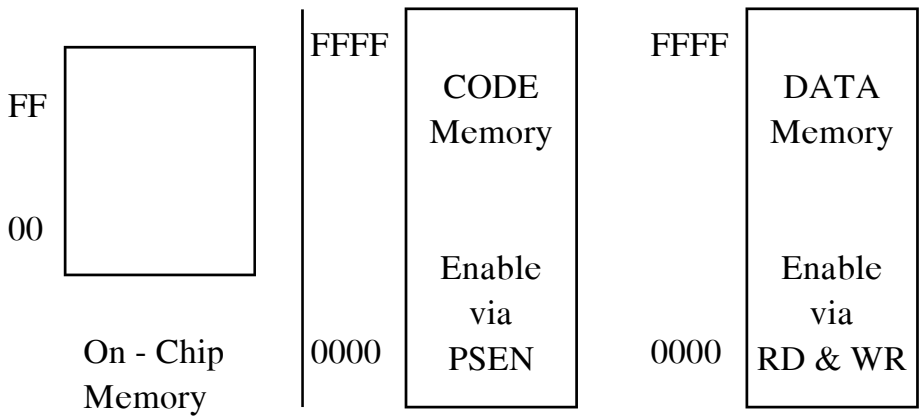
- **Các ngõ vào bộ dao động X1, X2:**

- Bộ dao động được tích hợp bên trong 8951, khi sử dụng 8951 người thiết kế chỉ cần kết nối thêm thạch anh và các tụ như hình vẽ trong sơ đồ. Tần số thạch anh thường sử dụng cho 8951 là 12Mhz.

- Chân 40 (Vcc) được nối lên nguồn 5V.

III. CẤU TRÚC BÊN TRONG VI ĐIỀU KHIỂN

1. Tổ chức bộ nhớ:



External Memory

Bảng tóm tắt các vùng nhớ 8951.

Bản đồ bộ nhớ Data trên Chip như sau :

Địa chỉ byte	Địa chỉ bit							
7F	RAM đa dụng							
30								
2F	7F	7E	7D	7C	7B	7A	79	78
2E	77	76	75	74	73	72	71	70
2D	6F	6E	6D	6C	6B	6A	69	68
2C	67	66	65	64	63	62	61	60
2B	5F	5E	5D	5C	5B	5A	59	58
2A	57	56	55	54	53	52	51	50
29	4F	4E	4D	4C	4B	4A	49	48
28	47	46	45	44	43	42	41	40
27	3F	3E	3D	3C	3B	3A	39	38
26	37	36	35	34	33	32	31	30
25	2F	2E	2D	2C	2B	2A	29	28
24	27	26	25	24	23	22	21	20
23	1F	1E	1D	1C	1B	1A	19	18
22	17	16	15	14	13	12	11	10
21	0F	0E	0D	0C	0B	0A	09	08
20	07	06	05	04	03	02	01	00
1F	Bank 3							
18								
17	Bank 2							
10								
0F	Bank 1							
08								
07	Bank thanh ghi 0							
00	(mặc định cho R0 -R7)							

RAM

Địa chỉ byte	Địa chỉ bit								
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	P.3
A8	AF			AC	AB	AA	A9	A8	IE
A0	A7	A6	A5	A4	A3	A2	A1	A0	P2
99	không được địa chỉ hoá bit								SBUF
98	9F	9E	9D	9C	9B	9A	99	98	SCON
90	97	96	95	94	93	92	91	90	P1
8D	không được địa chỉ hoá bit								TH1
8C	không được địa chỉ hoá bit								TH0
8B	không được địa chỉ hoá bit								TL1
8A	không được địa chỉ hoá bit								TL0
89	không được địa chỉ hoá bit								TMOD
88	8F	8E	8D	8C	8B	8A	89	88	TCON
87	không được địa chỉ hoá bit								PCON
83	không được địa chỉ hoá bit								DPH
82	không được địa chỉ hoá bit								DPL
81	không được địa chỉ hoá bit								SP
80	87	86	85	84	83	82	81	80	P0

CÁC THANH GHI CHỨC NĂNG ĐẶC BIỆT

- Bộ nhớ trong 8951 bao gồm ROM và RAM. RAM trong 8951 bao gồm nhiều thành phần: phần lưu trữ đa dụng, phần lưu trữ địa chỉ hóa từng bit, các bank thanh ghi và các thanh ghi chức năng đặc biệt.

- 8951 có bộ nhớ theo cấu trúc Harvard: có những vùng bộ nhớ riêng biệt cho chương trình và dữ liệu. Chương trình và dữ liệu có thể chứa bên trong 8951 nhưng 8951 vẫn có thể kết nối với 64K byte bộ nhớ chương trình và 64K byte dữ liệu.

Hai đặc tính cần chú ý là :

- Các thanh ghi và các port xuất nhập đã được định vị (xác định) trong bộ nhớ và có thể truy xuất trực tiếp giống như các địa chỉ bộ nhớ khác.
- Ngăn xếp bên trong Ram nội nhỏ hơn so với Ram ngoại như trong các bộ Microcontroller khác.

RAM bên trong 8951 được Phân chia như sau:

- Các bank thanh ghi có địa chỉ từ 00H đến 1FH.
- RAM địa chỉ hóa từng bit có địa chỉ từ 20H đến 2FH.
- RAM đa dụng từ 30H đến 7FH.
- Các thanh ghi chức năng đặc biệt từ 80H đến FFH.

RAM đa dụng:

- Mặc dù trên hình vẽ cho thấy 80 byte đa dụng chiếm các địa chỉ từ 30H đến 7FH, 32 byte dưới từ 00H đến 1FH cũng có thể dùng với mục đích tương tự (mặc dù các địa chỉ này đã có mục đích khác).

- Mọi địa chỉ trong vùng RAM đa dụng đều có thể truy xuất tự do dùng kiểu địa chỉ trực tiếp hoặc gián tiếp.

RAM có thể truy xuất từng bit:

- 8951 chứa 210 bit được địa chỉ hóa, trong đó có 128 bit có chứa các byte có chứa các địa chỉ từ 20F đến 2FH và các bit còn lại chứa trong nhóm thanh ghi có chức năng đặc biệt.

- Ý tưởng truy xuất từng bit bằng phần mềm là các đặc tính mạnh của microcontroller xử lý chung. Các bit có thể được đặt, xóa, AND, OR, . . . , với 1 lệnh đơn. Đa số các microcontroller xử lý đòi hỏi một chuỗi lệnh đọc – sửa - ghi để đạt được mục đích tương tự. Ngoài ra các port cũng có thể truy xuất được từng bit.

- 128 bit truy xuất từng bit này cũng có thể truy xuất như các byte hoặc như các bit phụ thuộc vào lệnh được dùng.

Các bank thanh ghi:

- 32 byte thấp của bộ nhớ nội được dành cho các bank thanh ghi. Bộ lệnh 8951 hổ trợ 8 thanh ghi có tên là R0 đến R7 và theo mặc định sau khi reset hệ thống, các thanh ghi này có các địa chỉ từ 00H đến 07H.

- Các lệnh dùng các thanh ghi RO đến R7 sẽ ngắn hơn và nhanh hơn so với các lệnh có chức năng tương ứng dùng kiểu địa chỉ trực tiếp. Các dữ liệu được dùng thường xuyên nên dùng một trong các thanh ghi này.

- Do có 4 bank thanh ghi nên tại một thời điểm chỉ có một bank thanh ghi được truy xuất bởi các thanh ghi RO đến R7 để chuyển đổi việc truy xuất các bank thanh ghi ta phải thay đổi các bit chọn bank trong thanh ghi trạng thái.

2. **Các thanh ghi có chức năng đặc biệt**:

- Các thanh ghi nội của 8951 được truy xuất ngầm định bởi bộ lệnh.
- Các thanh ghi trong 8951 được định dạng như một phần của RAM trên chip vì vậy mỗi thanh ghi sẽ có một địa chỉ (ngoại trừ thanh ghi bộ đếm chương trình và thanh ghi lệnh vì các thanh ghi này hiếm khi bị tác động trực tiếp). Cũng như R0 đến R7, 8951 có 21 thanh ghi có chức năng đặc biệt (SFR: Special Function Register) ở vùng trên của RAM nội từ địa chỉ 80H đến FFH.

Chú ý: tất cả 128 địa chỉ từ 80H đến FFH không được định nghĩa, chỉ có 21 thanh ghi có chức năng đặc biệt được định nghĩa sẵn các địa chỉ.

- Ngoại trừ thanh ghi A có thể được truy xuất ngầm như đã nói, đa số các thanh ghi có chức năng đặc biệt SFR có thể địa chỉ hóa từng bit hoặc byte.

- *Thanh ghi trạng thái chương trình (PSW: Program Status Word):*

Từ trạng thái chương trình ở địa chỉ D0H được tóm tắt như sau:

BIT	SYMBOL	ADDRESS	DESCRIPTION
PSW.7	CY	D7H	Cary Flag
PSW.6	AC	D6H	Auxiliary Cary Flag
PSW.5	F0	D5H	Flag 0
PSW4	RS1	D4H	Register Bank Select 1
PSW.3	RS0	D3H	Register Bank Select 0
			00=Bank 0; address 00H÷07H
			01=Bank 1; address 08H÷0FH
			10=Bank 2; address 10H÷17H
			11=Bank 3; address 18H÷1FH
PSW.2	OV	D2H	Overlow Flag
PSW.1	-	D1H	Reserved
PSW.0	P	DOH	Even Parity Flag

Chức năng từng bit trạng thái chương trình

- *Cờ Carry CY (Carry Flag):*

- Cờ nhớ có tác dụng kép. Thông thường nó được dùng cho các lệnh toán học: C=1 nếu phép toán cộng có sự tràn hoặc phép trừ có mượn và ngược lại C= 0 nếu phép toán cộng không tràn và phép trừ không có mượn.

- *Cờ Carry phụ AC (Auxiliary Carry Flag):*

Khi cộng những giá trị BCD (Binary Code Decimal), cờ nhớ phụ AC được set nếu kết quả 4 bit thấp nằm trong phạm vi điều khiển 0AH÷ 0FH. Ngược lại AC= 0.

- *Cờ 0 (Flag 0):*

Cờ 0 (F0) là 1 bit cờ đa dụng dùng cho các ứng dụng của người dùng.

- *Những bit chọn bank thanh ghi truy xuất:*

- RS1 và RS0 quyết định dãy thanh ghi tích cực. Chúng được xóa sau khi reset hệ thống và được thay đổi bởi phần mềm khi cần thiết.

- Tùy theo RS1, RS0 = 00, 01, 10, 11 sẽ được chọn Bank tích cực tương ứng là Bank 0, Bank1, Bank2, Bank3.

RS1	RS0	BANK
0	0	0
0	1	1
1	0	2
1	1	3

- *Cờ tràn OV (Over Flag) :*

- Cờ tràn được set sau một hoạt động cộng hoặc trừ nếu có sự tràn toán học. Khi các số có dấu được cộng hoặc trừ với nhau, phần mềm có thể kiểm tra bit này để xác định xem kết quả có nằm trong tầm xác định không. Khi các số không có dấu được cộng bit OV được bỏ qua. Các kết quả lớn hơn +127 hoặc nhỏ hơn –128 thì bit OV = 1.

- *Bit Parity (P):*

- Bit tự động được set hay Clear ở mỗi chu kỳ máy để lập Parity chẵn với thanh ghi A. Sự đếm các bit 1 trong thanh ghi A cộng với bit Parity luôn luôn chẵn.

Ví dụ A chứa 10101101B thì bit P set lên một để tổng số bit 1 trong A và P tạo thành số chẵn.

- Bit Parity thường được dùng trong sự kết hợp với những thủ tục của Port nối tiếp để tạo ra bit Parity trước khi phát đi hoặc kiểm tra bit Parity sau khi thu.

- *Thanh ghi B:*

- Thanh ghi B ở địa chỉ F0H được dùng cùng với thanh ghi A cho các phép toán nhân chia. Lệnh MUL AB ⇐ sẽ nhận những giá trị không dấu 8 bit trong hai thanh ghi A và B, rồi trả về kết quả 16 bit trong A (byte cao) và B(byte thấp). Lệnh DIV AB ⇐ lấy A chia B, kết quả nguyên đặt vào A, số dư đặt vào B.

- Thanh ghi B có thể được dùng như một thanh ghi đệm trung gian đa mục đích. Nó là những bit định vị thông qua những địa chỉ từ F0H÷F7H.

- *Con trỏ Ngăn xếp SP (Stack Pointer) :*

- Con trỏ ngăn xếp là một thanh ghi 8 bit ở địa chỉ 81H. Nó chứa địa chỉ của byte dữ liệu hiện hành trên đỉnh ngăn xếp. Các lệnh trên ngăn xếp bao gồm các lệnh cất dữ liệu vào ngăn xếp (PUSH) và lấy dữ liệu ra khỏi Ngăn xếp (POP). Lệnh cất dữ liệu vào ngăn xếp sẽ làm tăng SP trước khi ghi dữ liệu và lệnh lấy ra khỏi ngăn xếp sẽ làm giảm SP. Ngăn xếp của 8031/8051 được giữ trong RAM nội và giới hạn các địa chỉ có thể truy xuất bằng địa chỉ gián tiếp, chúng là 128 byte đầu của 8951.

- Để khởi động SP với ngăn xếp bắt đầu tại địa chỉ 60H, các lệnh sau đây được dùng:

```
MOV SP , #5F
```

- Với lệnh trên thì ngăn xếp của 8951 chỉ có 32 byte vì địa chỉ cao nhất của RAM trên chip là 7FH. Sở dĩ giá trị 5FH được nạp vào SP vì SP tăng lên 60H trước khi cất byte dữ liệu.

- Khi Reset 8951, SP sẽ mang giá trị mặc định là 07H và dữ liệu đầu tiên sẽ được cất vào ô nhớ ngăn xếp có địa chỉ 08H. Nếu phần mềm ứng dụng không khởi động SP một giá trị mới thì bank thanh ghi1 có thể cả 2 và 3 sẽ không dùng được vì vùng RAM này đã được dùng làm ngăn xếp. Ngăn xếp được truy xuất trực tiếp bằng các lệnh PUSH và POP để lưu trữ tạm thời và lấy lại dữ liệu, hoặc truy xuất ngầm bằng lệnh gọi chương trình con (ACALL, LCALL) và các lệnh trở về (RET, RETI) để lưu trữ giá trị của bộ đếm chương trình khi bắt đầu thực hiện chương trình con và lấy lại khi kết thúc chương trình con ...

- *Con trỏ dữ liệu DPTR (Data Pointer):*

-Con trỏ dữ liệu (DPTR) được dùng để truy xuất bộ nhớ ngoài là một thanh ghi 16 bit ở địa chỉ 82H (DPL: byte thấp) và 83H (DPH: byte cao). Ba lệnh sau sẽ ghi 55H vào RAM ngoài ở địa chỉ 1000H:

```
MOV A , #55H
MOV DPTR, #1000H
MOV @DPTR, A
```

- Lệnh đầu tiên dùng để nạp 55H vào thanh ghi A. Lệnh thứ hai dùng để nạp địa chỉ của ô nhớ cần lưu giá trị 55H vào con trỏ dữ liệu DPTR. Lệnh thứ ba sẽ di chuyển nội dung thanh ghi A (là 55H) vào ô nhớ RAM bên ngoài có địa chỉ chứa trong DPTR (là 1000H).

- *Các thanh ghi Port (Port Register):*

- Các Port của 8951 bao gồm Port0 ở địa chỉ 80H, Port1 ở địa chỉ 90H, Port2 ở địa chỉ A0H, và Port3 ở địa chỉ B0H. Tất cả các Port này đều có thể truy xuất từng bit nên rất thuận tiện trong khả năng giao tiếp.

- *Các thanh ghi Timer (Timer Register):*

- 8951 có chứa hai bộ định thời/ bộ đếm 16 bit được dùng cho việc định thời được đếm sự kiện. Timer0 ở địa chỉ 8AH (TLO: byte thấp) và 8CH (THO: byte cao). Timer1 ở địa chỉ 8BH (TL1: byte thấp) và 8DH (TH1: byte cao). Việc khởi động timer được SET bởi Timer Mode (TMOD) ở địa chỉ 89H và thanh ghi điều khiển Timer (TCON) ở địa chỉ 88H. Chỉ có TCON được địa chỉ hóa từng bit .

- Các thanh ghi Port nối tiếp (Serial Port Register) :

- 8951 chứa một Port nối tiếp cho việc trao đổi thông tin với các thiết bị nối tiếp như máy tính, modem hoặc giao tiếp nối tiếp với các IC khác. Một thanh ghi đệm dữ liệu nối tiếp (SBUF) ở địa chỉ 99H sẽ giữ cả hai dữ liệu truyền và dữ liệu nhập. Khi truyền dữ liệu ghi lên SBUF, khi nhận dữ liệu thì đọc SBUF. Các mode vận khác nhau được lập trình qua thanh ghi điều khiển Port nối tiếp (SCON) được địa chỉ hóa từng bit ở địa chỉ 98H.

- *Các thanh ghi ngắt (Interrupt Register):*

- 8951 có cấu trúc 5 nguồn ngắt, 2 mức ưu tiên. Các ngắt bị cấm sau khi bị reset hệ thống và sẽ được cho phép bằng việc ghi thanh ghi cho phép ngắt (IE) ở địa chỉ A8H. Cả hai được địa chỉ hóa từng bit.

- *Thanh ghi điều khiển nguồn PCON (Power Control Register):*

- Thanh ghi PCON không có bit định vị. Nó ở địa chỉ 87H chứa nhiều bit điều khiển. Thanh ghi PCON được tóm tắt như sau:

√ Bit 7 (SMOD) : Bit có tốc độ Baud ở mode 1, 2, 3 ở Port nối tiếp khi set.

√ Bit 6, 5, 4 : Không có địa chỉ.

√ Bit 3 (GF1) : Bit cờ đa năng 1.

√ Bit 2 (GF0) : Bit cờ đa năng 2 .

√ Bit 1 (PD) : Set để khởi động mode Power Down và thoát để reset.

√ Bit 0 (IDL) : Set để khởi động mode Idle và thoát khi ngắt mạch hoặc reset.

Các bit điều khiển Power Down và Idle có tác dụng chính trong tất cả các IC họ MSC-51 nhưng chỉ được thi hành trong sự biên dịch của CMOS.

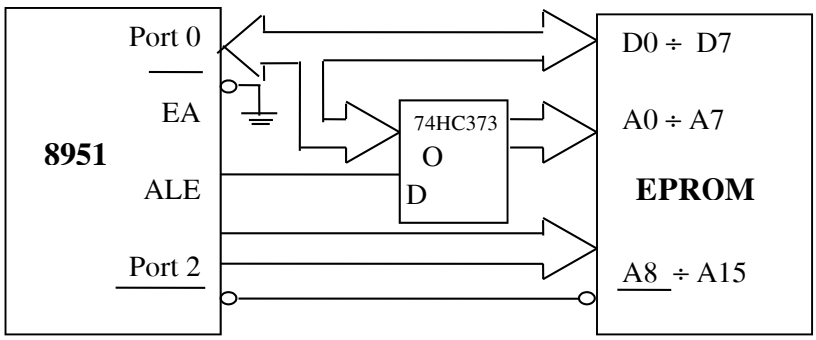
3. Bộ nhớ ngoài (external memory) :

- 8951 có khả năng mở rộng bộ nhớ lên đến 64K byte bộ nhớ chương trình và 64k byte bộ nhớ dữ liệu ngoài. Do đó có thể dùng thêm RAM và ROM nếu cần.

- Khi dùng bộ nhớ ngoài, Port0 không còn chức năng I/O nữa. Nó được kết hợp giữa bus địa chỉ (A0-A7) và bus dữ liệu (D0-D7) với tín hiệu ALE để chốt byte của bus địa chỉ khi bắt đầu mỗi chu kỳ bộ nhớ. Port được cho là byte cao của bus địa chỉ.

Truy xuất bộ nhớ mã ngoài (Accessing External Code Memory) :

- Bộ nhớ chương trình bên ngoài là bộ nhớ ROM được cho phép của tín hiệu PSEN\. Sự kết nối phần cứng của bộ nhớ EPROM như sau:



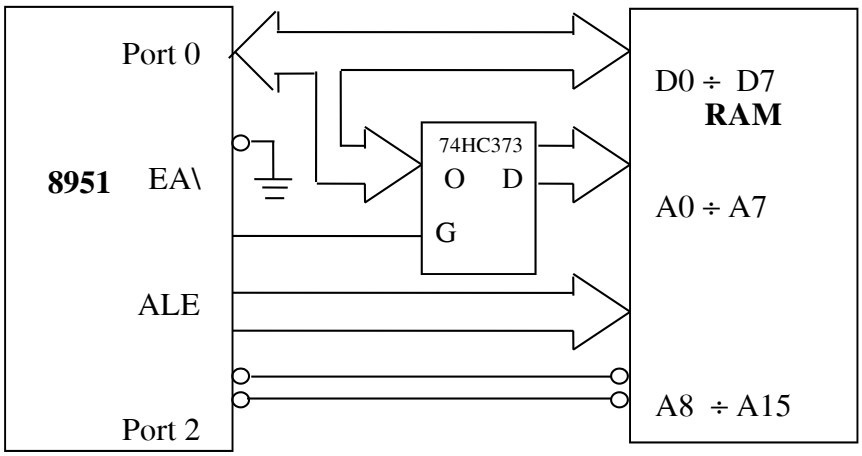
- Trong một chu kỳ máy tiêu biểu, tín hiệu ALE tích 2 lần. Lần thứ nhất cho phép 74HC373 mở cổng chốt địa chỉ byte thấp, khi ALE xuống 0 thì byte thấp và byte cao của bộ đếm chương trình đều có nhưng EPROM chưa xuất vì PSEN\ chưa tích cực, khi tín hiệu lên một trở lại thì Port 0 đã có dữ liệu là Opcode. ALE tích cực lần thứ hai

được giải thích tương tự và byte 2 được đọc từ bộ nhớ chương trình. Nếu lệnh đang hiện hành là lệnh 1 byte thì CPU chỉ đọc Opcode, còn byte thứ hai bỏ đi.

- ***Truy xuất bộ nhớ dữ liệu ngoài (Accessing External Data Memory):***

- Bộ nhớ dữ liệu ngoài là một bộ nhớ RAM được đọc hoặc ghi khi được cho phép của tín hiệu RD\ và WR. Hai tín hiệu này nằm ở chân P3.7 (RD) và P3.6 (WR). Lệnh MOVX được dùng để truy xuất bộ nhớ dữ liệu ngoài và dùng một bộ đệm dữ liệu 16 bit (DPTR), R0 hoặc R1 như là một thanh ghi địa chỉ.

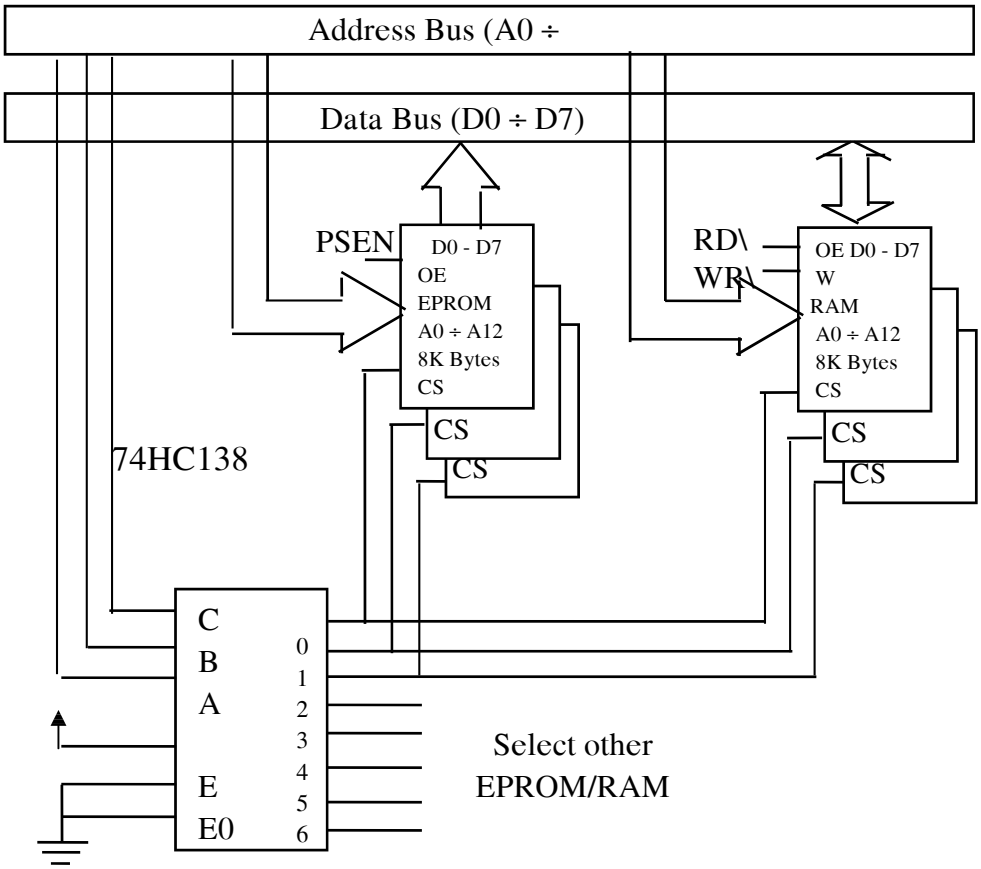
- Các RAM có thể giao tiếp với 8951 tương tự cách thức như EPROM ngoại trừ chân RD\ của 8951 nối với chân OE\ (Output Enable) của RAM và chân WR\ của 8951 nối với chân WE\ của RAM. Sự nối các bus địa chỉ và dữ liệu tương tự như cách nối của EPROM.



- ***Sự giải mã địa chỉ (Address Decoding):***

- Sự giải mã địa chỉ là một yêu cầu tất yếu để chọn EPROM, RAM, 8279, ... Sự giải mã địa chỉ đối với 8951 để chọn các vùng nhớ ngoài như các vi điều khiển. Nếu các con EPROM hoặc RAM 8K được dùng thì các bus địa chỉ phải được giải mã để chọn các IC nhớ nằm trong phạm vi giới hạn 8K: 0000H÷1FFFH, 2000H÷3FFFH, . .

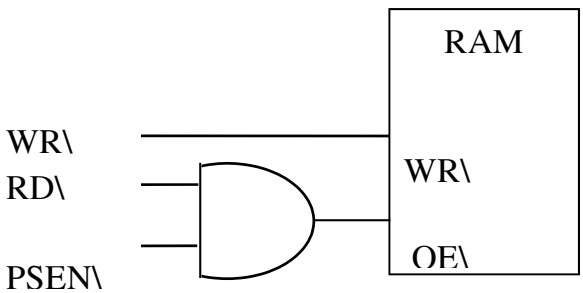
- Một cách cụ thể, IC giải mã 74HC138 được dùng với những ngõ ra của nó được nối với những ngõ vào chọn Chip CS (Chip Select) trên những IC nhớ EPROM, RAM, ... Hình sau đây cho phép kết nối nhiều EPROM và RAM.



Address Decoding (Giải mã địa chỉ)

- ***Sự đè lên nhau của các vùng nhớ dữ liệu ngoài:***

- Vì bộ nhớ chương trình là ROM, nên nảy sinh một vấn đề bất tiện khi phát triển phần mềm cho vi điều khiển. Một nhược điểm chung của 8951 là các vùng nhớ dữ liệu ngoài nằm đè lên nhau, vì tín hiệu PSEN\ được dùng để đọc bộ nhớ mã ngoài và tín hiệu RD\ được dùng để đọc bộ nhớ dữ liệu, nên một bộ nhớ RAM có thể chứa cả chương trình và dữ liệu bằng cách nối đường OE\ của RAM đến ngõ ra một cổng AND có hai ngõ vào PSEN\ và RD\. Sơ đồ mạch như hình sau cho phép cho phép bộ nhớ RAM có hai chức năng vừa là bộ nhớ chương trình vừa là bộ nhớ dữ liệu:

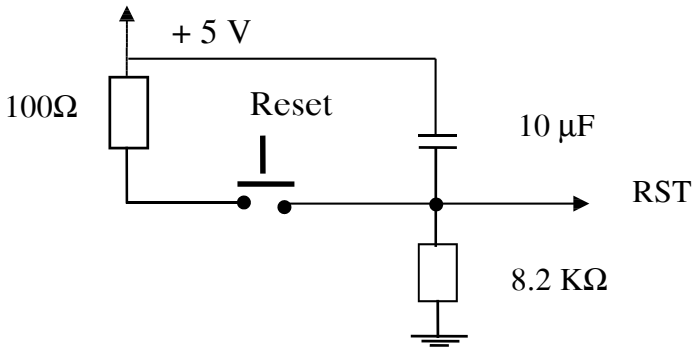


Overlapping the External code and data space

-Vậy một chương trình có thể được tải vào RAM bằng cách xem nó như bộ nhớ dữ liệu và thi hành chương trình băng cách xem nó như bộ nhớ chương trình.

Hoạt động Reset:

- 8951 có ngõ vào reset RST tác động ở mức cao trong khoảng thời gian 2 chu kỳ xung máy, sau đó xuống mức thấp để 8951 bắt đầu làm việc. RST có thể kích bằng tay bằng một phím nhấn thường hở, sơ đồ mạch reset như sau:



Manual Reset Reset bằng tay.

Trạng thái của tất cả các thanh ghi trong 8951 sau khi reset hệ thống được tóm tắt như sau:

Thanh ghi	Nội dung
Đếm chương trình PC	0000H
Thanh ghi tích lũy A	00H
Thanh ghi B	00H
Thanh ghi thái PSW	00H
SP	07H
DPRT	0000H
Port 0 đến port 3	FFH
IP	XXX0 0000 B
IE	0X0X 0000 B
Các thanh ghi định thời	00H 00H
SCON SBUF	00H
PCON (MHOS)	0XXX XXXXH
PCON (CMOS)	0XXX 0000 B

-Thanh ghi quan trọng nhất là thanh ghi bộ đếm chương trình PC được reset tai địa chỉ 0000H. Khi ngõ vào RST xuống mức thấp, chương trình luôn bắt đầu tại

địa chỉ 0000H của bộ nhớ chương trình. Nội dung của RAM trên chip không bị thay đổi bởi tác động của ngõ vào reset.

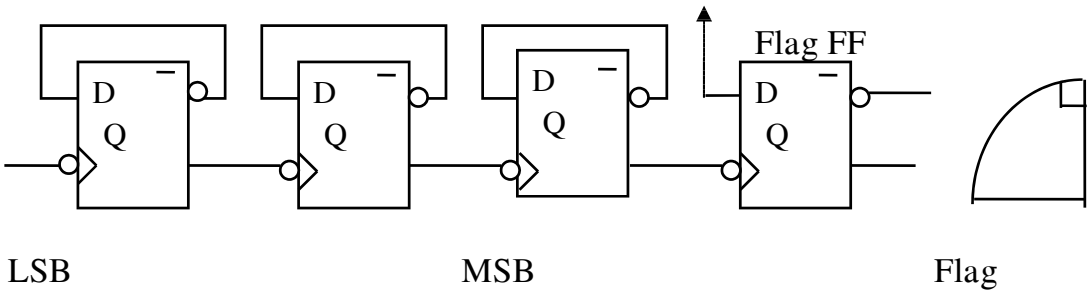
VI. HOẠT ĐỘNG TIMER CỦA 8951:

1. GIỚI THIỆU:

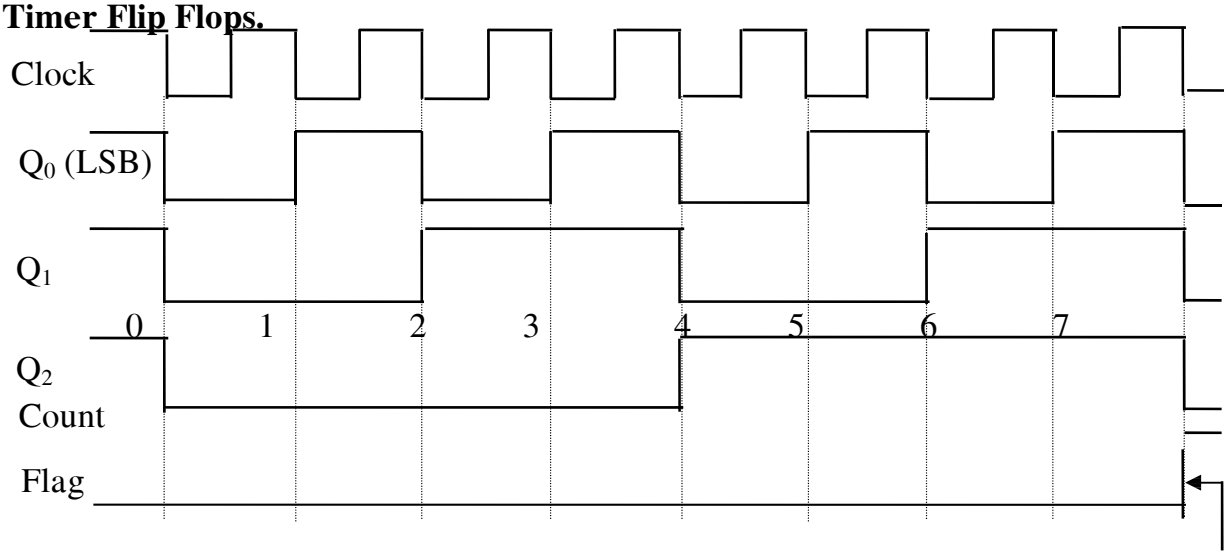
- Bộ định thời của Timer là một chuỗi các Flip Flop được chia làm 2, nó nhận tín hiệu vào là một nguồn xung clock, xung clock được đưa vào Flip Flop thứ nhất là xung clock của Flip Flop thứ hai mà nó cũng chia tần số clock này cho 2 và cứ tiếp tục.

- Vì mỗi tầng kế tiếp chia cho 2, nên Timer n tầng phải chia tần số clock ngõ vào cho 2^n. Ngõ ra của tầng cuối cùng là clock của Flip Flop tràn Timer hoặc cờ mà nó kiểm tra bởi phần mềm hoặc sinh ra ngắt. Giá trị nhị phân trong các FF của bộ Timer có thể được nghĩ như đếm xung clock hoặc các sự kiện quan trọng bởi vì Timer được khởi động. Ví dụ Timer 16 bit có thể đếm đến từ FFFFH sang 0000H.

- Hoạt động của Timer đơn giản 3 bit được minh họa như sau:



Timer Flip Flops.



- Trong hình trên mỗi tầng là một FF loại D phủ định tác động cạnh xuống được hoạt động ở mode chia cho 2 (ngõ ra Q\ được nối vào D). FF cờ là một bộ chốt

đơn giản loại D được set bởi tầng cuối cùng trong Timer. Trong biểu đồ thời gian, tầng đầu đổi trạng thái ở ½ tần số clock, tầng thứ hai đổi trạng thái ở tần số ¼ tần số clock . . . Số đếm được biết ở dạng thập phân và được kiểm tra lại dễ dàng bởi việc kiểm tra các tầng của 3 FF. Ví dụ số đếm "4" xuất hiện khi Q2=1, Q1=0, Q0=0 ($4_{10}=100_2$).

- Các Timer được ứng dụng thực tế cho các hoạt động định hướng. 8951 có 2 bộ Timer 16 bit, mỗi Timer có 4 mode hoạt động. Các Timer dùng để đếm giờ, đếm các sự kiện cần thiết và sự sinh ra tốc độ của tốc độ Baud bởi sự gắn liền Port nối tiếp.

- Mỗi sự định thời là một Timer 16 bit, do đó tầng cuối cùng là tầng thứ 16 sẽ chia tần số clock vào cho 2^{16} = 65.536.

- Trong các ứng dụng định thời, 1 Timer được lập trình để tràn ở một khoảng thời gian đều đặn và được set cờ tràn Timer. Cờ được dùng để đồng bộ chương trình để thực hiện một hoạt động như việc đưa tới 1 tầng các ngõ vào hoặc gởi dữ liệu đếm ngõ ra. Các ứng dụng khác có sử dụng việc ghi giờ đều đều của Timer để đo thời gian đã trôi qua hai trạng thái (ví dụ đo độ rộng xung).Việc đếm một sự kiện được dùng để xác định số lần xuất hiện của sự kiện đó, tức thời gian trôi qua giữa các sự kiện.

- Các Timer của 8951 được truy xuất bởi việc dùng 6 thanh ghi chức năng đặc biệt như sau :

Timer SFR	**Purpose**	**Address**	**Bit-Addressable**
TCON	Control	88H	YES
TMOD	Mode	89H	NO
TL0	Timer 0 low-byte	8AH	NO
TL1	Timer 1 low-byte	8BH	NO
TH0	Timer 0 high-byte	8CH	NO
TH1	Timer 1 high-byte	8DH	NO

2. CÁC THANH GHI ĐIỀU KHIỂN TIMER

2.1. Thanh ghi điều khiển chế độ timer TMOD (timer mode register) :

- Thanh ghi mode gồm hai nhóm 4 bit là: 4 bit thấp đặt mode hoạt động cho Timer 0 và 4 bit cao đặt mode hoạt động cho Timer 1. 8 bit của thanh ghi TMOD được tóm tắt như sau:

Bit	Name	Timer	Description
7	GATE	1	Khi GATE = 1, Timer chỉ làm việc khi INT1=1
6	C/T	1	Bit cho đếm sự kiện hay ghi giờ
			C/T = 1 : Đếm sự kiện
			C/T = 0 : Ghi giờ đều đặn
5	M1	1	Bit chọn mode của Timer 1
4	M0	1	Bit chọn mode của Timer 1
3	GATE	0	Bit cổng của Timer 0
2	C/T	0	Bit chọn Counter/Timer của Timer 0
1	M1	0	Bit chọn mode của Timer 0
0	M0	0	Bit chọn mode của Timer 0

Hai bit M0 và M1 của TMOD để chọn mode cho Timer 0 hoặc Timer 1.

M1	M0	MODE	DESCRIPTION
0	0	0	Mode Timer 13 bit (mode 8048)
0	1	1	Mode Timer 16 bit
1	0	2	Mode tự động nạp 8 bit
1	1	3	Mode Timer tách ra : Timer 0 : TL0 là Timer 8 bit được điều khiển bởi các bit của Timer 0. TH0 tương tự nhưng được điều khiển bởi các bit của mode Timer 1. Timer 1 : Được ngừng lại.

- TMOD không có bit định vị, nó thường được LOAD một lần bởi phần mềm ở đầu chương trình để khởi động mode Timer. Sau đó sự định giờ có thể dừng lại, được khởi động lại như thế bởi sự truy xuất các thanh ghi chức năng đặc biệt của Timer khác.

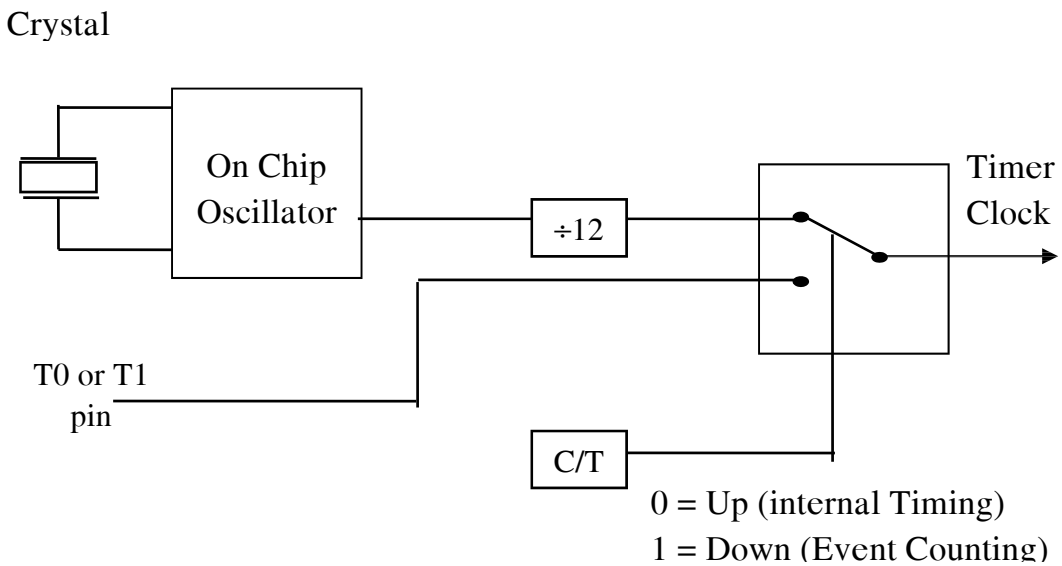
2.2. Thanh ghi điều khiển timer TCON (timer control register):

- Thanh ghi điều khiển bao gồm các bit trạng thái và các bit điều khiển bởi Timer 0 và Timer 1. Thanh ghi TCON có bit định vị. Hoạt động của từng bit được tóm tắt như sau :

Bit	Symbol	Bit Address	Description
TCON.7	TF1	8FH	Cờ tràn Timer 1 được set bởi phần cứng ở sự tràn, được xóa bởi phần mềm hoặc bởi phần cứng khi các vectơ xử lí đến thủ tục phục vụ ngắt ISR
TCON.6	TR1	8EH	Bit điều khiển chạy Timer 1 được set hoặc xóa bởi phần mềm để chạy hoặc ngưng chạy Timer.
TCON.5	TF0	8DH	Cờ tràn Timer 0(hoạt động tương tự TF1)
TCON.4	TR0	8CH	Bit điều khiển chạy Timer 0 (giống TR1)
TCON.3	IE1	8BH	Cờ kiểu ngắt 1 ngoài. Khi cạnh xuống xuất hiện trên INT1 thì IE1 được xóa bởi phần mềm hoặc phần cứng khi CPU định hướng đến thủ tục phục vụ ngắt ngoài.
TCON.2	IT1	8AH	Cờ kiểu ngắt 1 ngoài được set hoặc xóa bằng phấn mềm bởi cạnh kích hoạt bởi sự ngắt ngoài.
TCON.1	IE0	89H	Cờ cạnh ngắt 0 ngoài
TCON	IT0	88H	Cờ kiểu ngắt 0 ngoài.

2.3. Các nguồn xung nhịp cho timer (clock sources):

- Có hai nguồn xung clock có thể đếm giờ là sự định giờ bên trong và sự đếm sự kiện bên ngoài. Bit C/T trong TMOD cho phép chọn 1 trong 2 khi Timer được khởi động.



¨ ***Sự bấm giờ bên trong (Interval Timing):***

- Nếu bit C/T = 0 thì hoạt động của Timer liên tục được chọn vào bộ Timer được ghi giờ từ dao động trên Chip. Một bộ chia 12 được thêm vào để giảm tần số clock đến 1 giá trị phù hợp với các ứng dụng. Các thanh ghi TLx và THx tăng ở tốc độ 1/12 lần tần số dao động trên Chip. Nếu dùng thạch anh 12MHz thì sẽ đưa đến tốc độ clock 1MHz.

- Các sự tràn Timer sinh ra sau một con số cố định của những xung clock, nó phụ thuộc vào giá trị khởi tạo được LOAD vào các thanh ghi THx và TLx.

¨ ***Sự đếm các sự kiện (Event Counting) :***

- Nếu bit C/T = 1 thì bộ Timer được ghi giờ từ nguồn bên ngoài trong nhiều ứng dụng, nguồn bên ngoài này cung cấp 1 sự định giờ với 1 xung trên sự xảy ra của sự kiện. Sự định giờ là sự đếm sự kiện. Con số sự kiện được xác định trong phần mềm bởi việc đọc các thanh ghi Timer. Tlx/THx, bởi vì giá trị 16 bit trong các thanh này tăng lên cho mỗi sự kiện.

- Nguồn xung clock bên ngoài đưa vào chân P3.4 là ngõ nhập của xung clock bởi Timer 0 (T0) và P3.5 là ngõ nhập của xung clock bởi Timer 1 (T1).

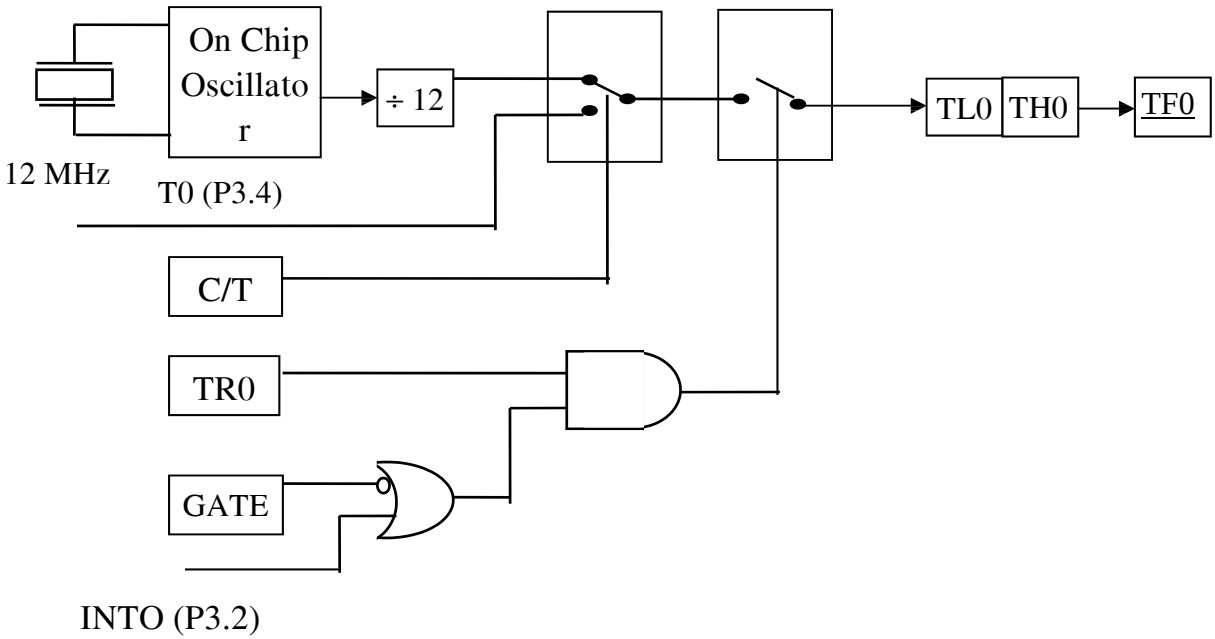
- Trong các ứng dụng đếm các thanh ghi Timer được tăng trong đáp ứng của sự chuyển trạng thái từ 1 sang 0 ở ngõ nhập Tx. Ngõ nhập bên ngoài được thử trong

suốt S5P2 của mọi chu kỳ máy: Do đó khi ngõ nhập đưa tới mức cao trong một chu kỳ và mức thấp trong một chu kỳ kế tiếp thì bộ đếm tăng lên một. Giá trị mới xuất hiện trong các thanh ghi Timer trong suốt S5P1 của chu kỳ theo sau một sự chuyển đổi. Bởi vì nó chiếm 2 chu kỳ máy (2µs) để nhận ra sự chuyển đổi từ 1 sang 0, nên tần số bên ngoài lớn nhất là 500KHz nếu dao động thạch anh 12 MHz.

2.4. sự bắt đầu, kết thúc và sự điều khiển các timer (starting, stopping and controlling the timer) :

- Bit TRx trong thanh ghi có bit định vị TCON được điều khiển bởi phần mềm để bắt đầu hoặc kết thúc các Timer. Để bắt đầu các Timer ta set bit TRx và để kết thúc Timer ta Clear TRx. Ví dụ Timer 0 được bắt đầu bởi lệnh SETB TR0 và được kết thúc bởi lệnh CLR TR0 (bit Gate= 0). Bit TRx bị xóa sau sự reset hệ thống, do đó các Timer bị cấm bằng sự mặc định.

- Thêm phương pháp nữa để điều khiển các Timer là dùng bit GATE trong thanh ghi TMOD và ngõ nhập bên ngoài INTx. Điều này được dùng để đo các độ rộng xung. Giả sử xung đưa vào chân INT0 ta khởi động Timer 0 cho mode 1 là mode Timer 16 bit với TL0/TH0 = 0000H, GATE = 1, TR0 = 1. Như vậy khi INT0 = 1 thì Timer "được mở cổng" và ghi giờ với tốc độ của tần số 1MHz. Khi INT0 xuống thấp thì Timer "đóng cổng" và khoảng thời gian của xung tính bằng µs là sự đếm được trong thanh ghi TL0/TH0.



Timer Operating Mode 1.

2.5. Sự khởi động và truy xuất các thanh ghi timer:

- Các Timer được khởi động 1 lần ở đầu chương trình để đặt mode hoạt động cho chúng. Sau đó trong chương trình các Timer được bắt đầu, được xóa, các thanh ghi Timer được đọc và cập nhật ... theo yêu cầu của từng ứng dụng cụ thể.

- Mode Timer TMOD là thanh ghi đầu tiên được khởi gán, bởi vì đặt mode hoạt động cho các Timer. Ví dụ khởi động cho Timer 1 hoạt động ở mode 1 (mode Timer 16bit) và được ghi giờ bằng dao động trên Chip ta dùng lệnh : MOV TMOD, # 00001000B. Trong lệnh này M1 = 0, M0 = 1 để vào mode 1 và C/T = 0, GATE = 0 để cho phép ghi giờ bên trong đồng thời xóa các bit mode của Timer 0. Sau lệnh trên Timer vẫn chưa đếm giờ, nó chỉ bắt đầu đếm giờ khi set bit điều khiển chạy TR1 của nó.

- Nếu ta không khởi gán giá trị đầu cho các thanh ghi TLx/THx thì Timer sẽ bắt đầu đếm từ 0000Hlên và khi tràn từ FFFFH sang 0000H nó sẽ bắt đầu tràn TFx rồi tiếp tục đếm từ 0000H lên tiếp . . .

- Nếu ta khởi gán giá trị đầu cho TLx/THx, thì Timer sẽ bắt đầu đếm từ giá trị khởi gán đó lên nhưng khi tràn từ FFFFH sang 0000H lại đếm từ 0000H lên.

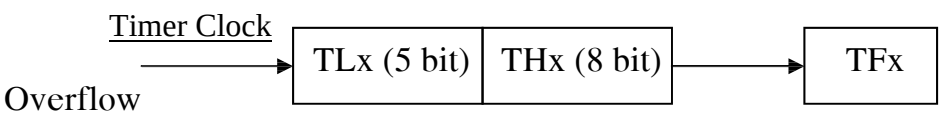
- Chú ý rằng cờ tràn TFx tự động được set bởi phần cứng sau mỗi sự tràn và sẽ được xóa bởi phần mềm. Chính vì vậy ta có thể lập trình chờ sau mỗi lần tràn ta sẽ xóa cờ TFx và quay vòng lặp khởi gán cho TLx/THx để Timer luôn luôn bắt đầu đếm từ giá trị khởi gán lên theo ý ta mong muốn.

- Đặc biệt những sự khởi gán nhỏ hơn 256 µs, ta sẽ gọi mode Timer tự động nạp 8 bit của mode 2. Sau khi khởi gán giá trị đầu vào THx, khi set bit TRx thì Timer sẽ bắt đầu đếm giá trị khởi gán và khi tràn từ FFH sang 00H trong TLx, cờ TFx tự động được set đồng thời giá trị khởi gán mà ta khởi gán cho Thx được nạp tự động vào TLx và Timer lại được đếm từ giá trị khởi gán này lên. Nói cách khác, sau mỗi tràn ta không cần khởi gán lại cho các thanh ghi Timer mà chúng vẫn đếm được lại từ giá trị ban đầu.

3. CÁC CHẾ ĐỘ TIMER VÀ CỜ TRÀN (TIMER MODES AND OVERFLOW):

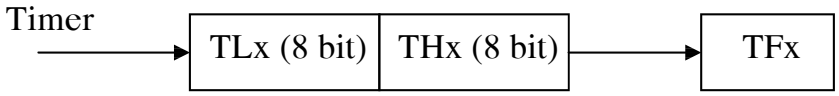
- 8951 có 2 Timer là Timer 0 và timer 1. Ta dùng ký hiệu TLx và Thx để chỉ 2 thanh ghi byte thấp và byte cao của Timer 0 hoặc Timer 1.

3.1. Mode Timer 13 bit (MODE 0) :



- Mode 0 là mode Timer 13 bit, trong đó byte cao của Timer (Thx) được đặt thấp và 5 bit trọng số thấp nhất của byte thấp Timer (TLx) đặt cao để hợp thành Timer 13 bit. 3 bit cao của TLx không dùng.

3.2. Mode Timer 16 bit (MODE 1) :



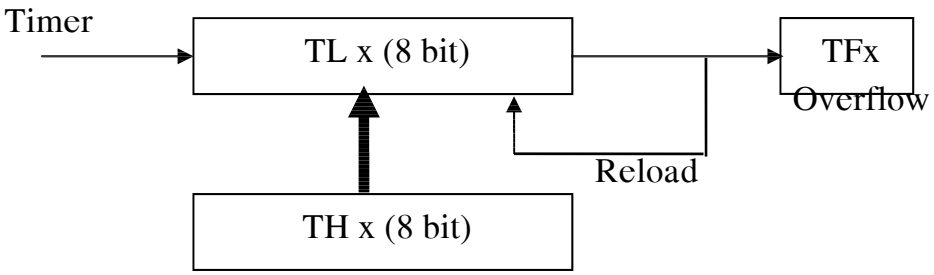
- Mode 1 là mode Timer 16 bit, tương tự như mode 0 ngoại trừ Timer này hoạt động như một Timer đầy đủ 16 bit, xung clock được dùng với sự kết hợp các thanh ghi cao và thấp (TLx, THx). Khi xung clock được nhận vào, bộ đếm Timer tăng lên 0000H, 0001H, 0002H, ..., và một sự tràn sẽ xuất hiện khi có sự chuyển trên bộ đếm Timer từ FFFH sang 0000H và sẽ set cờ tràn Time, sau đó Timer đếm tiếp.

- Cờ tràn là bit TFx trong thanh ghi TCON mà nó sẽ được đọc hoặc ghi bởi phần mềm.

- Bit có trọng số lớn nhất (MSB) của giá trị trong thanh ghi Timer là bit 7 của THx và bit có trọng số thấp nhất (LSB) là bit 0 của TLx. Bit LSB đổi trạng thái ở tần số clock vào được chia $2^{16} = 65.536$.

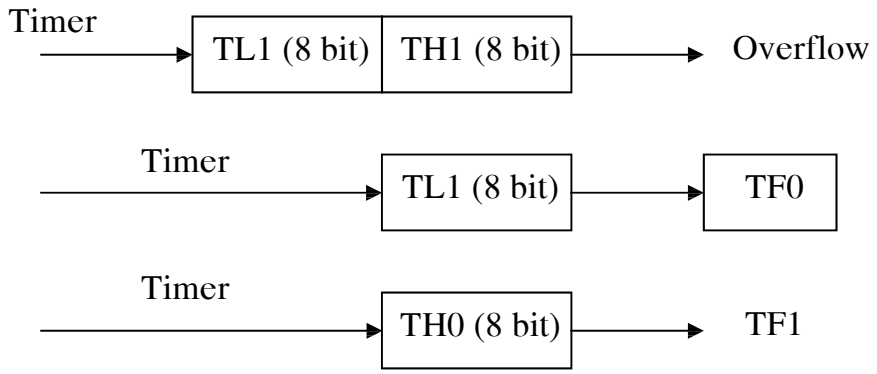
- Các thanh ghi Timer TLx và Thx có thể được đọc hoặc ghi tại bất kỳ thời điểm nào bởi phần mềm.

3.3. Mode tự động nạp 8 bit (MODE 2) :



-Mode 2 là mode tự động nạp 8 bit, byte thấp TLx của Timer hoạt động như một Timer 8 bit trong khi byte cao THx của Timer giữ giá trị Reload. Khi bộ đếm tràn từ FFH sang 00H, không chỉ cờ tràn được set mà giá trị trong THx cũng được nạp vào TLx : Bộ đếm được tiếp tục từ giá trị này lên đến sự chuyển trạng thái từ FFH sang 00H kế tiếp và cứ thế tiếp tục. Mode này thì phù hợp bởi vì các sự tràn xuất hiện cụ thể mà mỗi lúc nghỉ thanh ghi TMOD và THx được khởi động.

3.4 Mode Timer tách ra (MODE 3) :



- Mode 3 là mode Timer tách ra và là sự khác biệt cho mỗi Timer.

- Timer 0 ở mode 3 được chia là 2 timer 8 bit. TL0 và TH0 hoạt động như những Timer riêng lẻ với sự tràn sẽ set các bit TL0 và TF1 tương ứng.

- Timer 1 bị dừng lại ở mode 3, nhưng có thể được khởi động bởi việc ngắt nó vào một trong các mode khác. Chỉ có nhược điểm là cờ tràn TF1 của Timer 1 không bị ảnh hưởng bởi các sự tràn của Timer 1 bởi vì TF1 được nối với TH0.

- Mode 3 cung cấp 1 Timer ngoại 8 bit là Timer thứ ba của 8951. Khi vào Timer 0 ở mode 3, Timer có thể hoạt động hoặc tắt bởi sự ngắt nó ra ngoài và vào trong mode của chính nó hoặc có thể được dùng bởi Port nối tiếp như là một máy phát tốc độ Baud, hoặc nó có thể dùng trong hướng nào đó mà không sử dụng Interrupt.

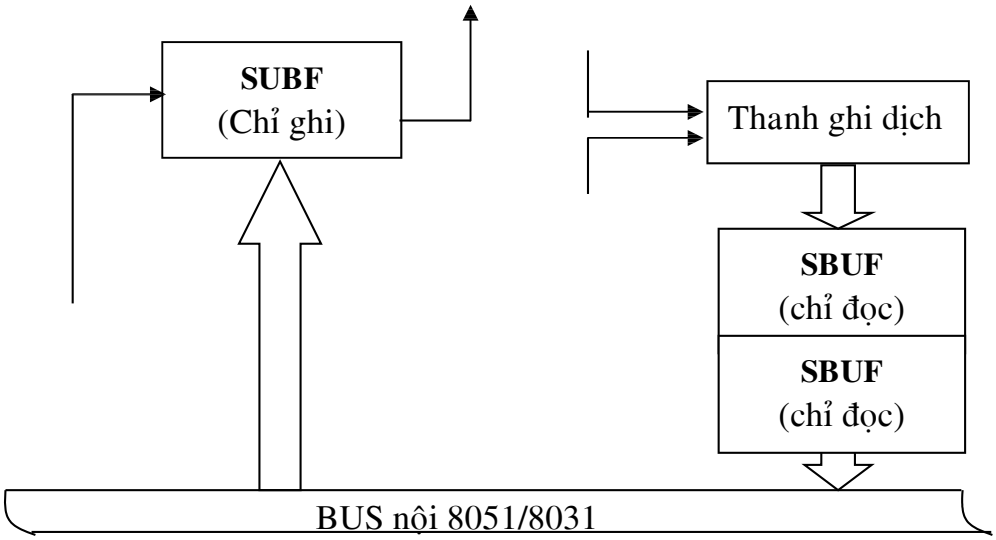
V. HOẠT ĐỘNG PORT NỐI TIẾP

1. Giới thiệu

8951 có một port nối tiếp trong chip có thể hoạt động ở nhiều chế độ trên một dãy tần số rộng. Chức năng chủ yếu là thực hiện chuyển đổi song song sang nối tiếp với dữ liệu xuất và chuyển đổi nối tiếp sang song song với dữ liệu nhập.

Port nối tiếp cho hoạt động song công (full duplex: thu và phát đồng thời) và đệm thu (receiver buffering) cho phép một ký tự sẽ được thu và được giữ trong khi ký tự thứ hai được nhận. Nếu CPU đọc ký tự thứ nhất trước khi ký tự thứ hai được thu đầy đủ thì dữ liệu sẽ không bị mất.

Hai thanh ghi chức năng đặc biệt cho phép phần mềm truy xuất đến port nối tiếp là: SBUF và SCON. Bộ đệm port nối tiếp (SBUF) ở địa chỉ 99H nhận dữ liệu để thu hoặc phát. Thanh ghi điều khiển port nối tiếp (SCON) ở địa chỉ 98H là thanh ghi có địa chỉ bit chứa các bit trạng thái và các bit điều khiển. Các bit điều khiển đặt chế độ hoạt động cho port nối tiếp, và các bit trạng thái Báo cáo kết thúc việc phát hoặc thu ký tự . Các bit trạng thái có thể được kiểm tra bằng phần mềm hoặc có thể lập trình để tạo ngắt.



2. Các thanh ghi và các chế độ hoạt động của port nối tiếp:

2.1. Thanh ghi điều khiển port nối tiếp:

Chế độ hoạt động của port nối tiếp được đặt bằng cách ghi vào thanh ghi chế độ port nối tiếp (SCON) ở địa chỉ 98H .Sau đây các bản tóm tắt thanh ghi SCON và các chế độ của port nối tiếp:

Bit	Ký hiệu	Địa chỉ	Mô tả
SCON.7	SM0	9FH	Bit 0 của chế độ port nối tiếp
SCON.6	SM1	9EH	Bit 1 của chế độ port nối tiếp
SCON.5	SM3	9DH	Bit 2 của chế độ port nối tiếp . Cho phép truyền thông xử lý trong các chế độ 2 và 3, RI sẽ không bị tác động nếu bit thứ 9 thu được là 0
SCON.4	REN	9CH	Cho phép bộ thu phải được đặt lên 1 để thu các ký tự
SCON.3	TB8	9BH	Bit 8 phát, bit thứ 9 được phát trong chế độ 2 và 3, được đặt và xóa bằng phần mềm.
SCON.2	RB8	9AH	B it 8 thu, bit thứ 9 thu được
SCON.1	TI	99H	Cờ ngắt phát. Đặt lên 1 khi kết thúc phát ký tự, được xóa bằng phần mềm
SCON.0	RI	98H	Cờ ngắt thu. Đặt lên 1 khi kết thúc thu ký tự, được xóa bằng phần mềm

Tóm tắt thanh ghi chế độ port nối tiếp

SM0	SM1	Chế độ	Mô tả	Tốc độ baud
0	0	0	Thanh ghi dịch	Cố định (Fosc /12)
0	1	1	UART 8 bit	Thay đổi (đặt bằng timer)
1	0	2	UART 9 bit	Cố định (Fosc /12 hoặc Fosc/64)
1	1	3	UART 9 bit	Thay đổi (đặt bằng timer)

Các chế độ port nối tiếp

Trước khi sử dụng port nối tiếp, phải khởi động SCON cho đúng chế độ. Ví dụ, lệnh sau:

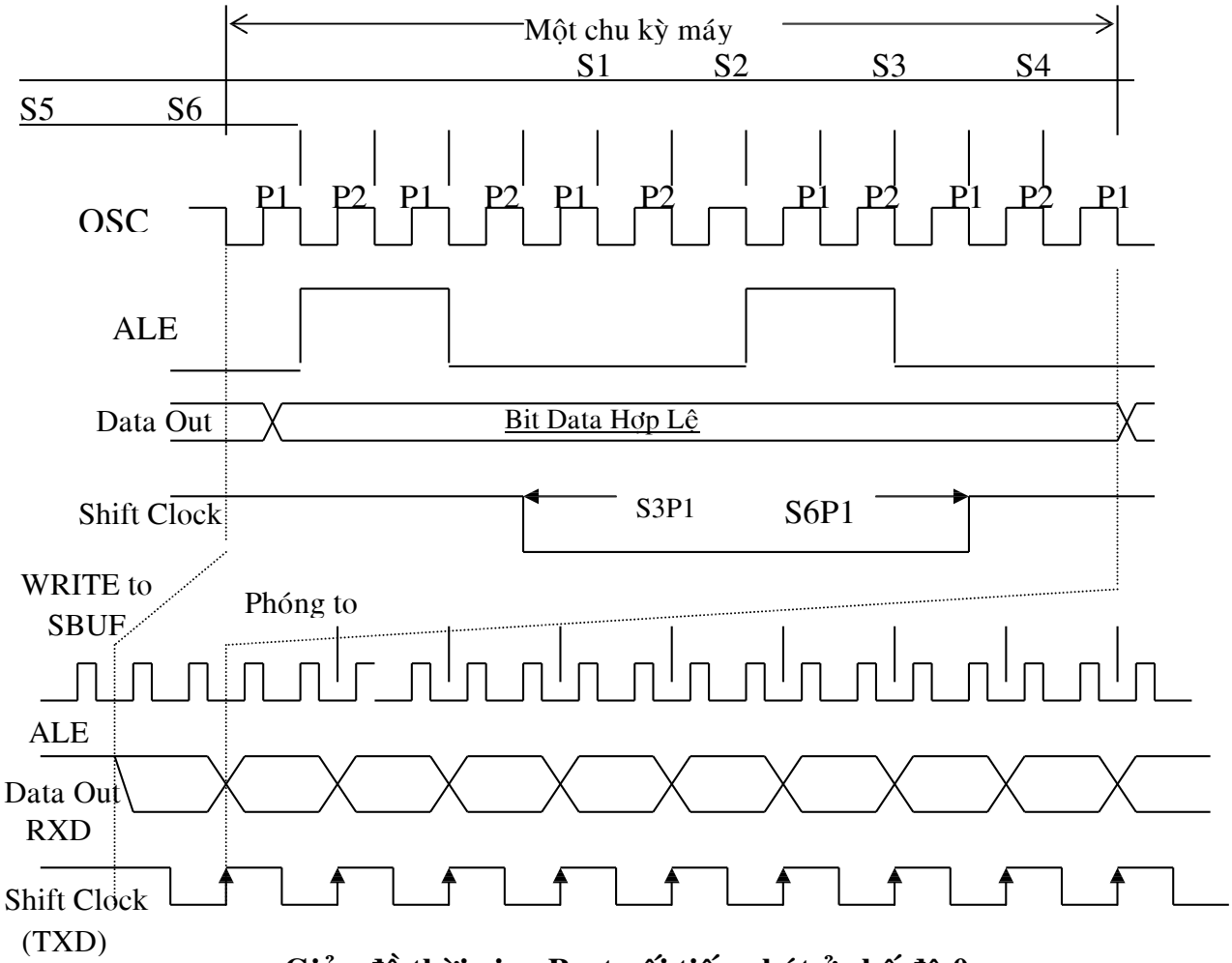
MOV SCON, #01010010B

Khởi động port nối tiếp cho chế độ 1 (SM0/SM1=0/1), cho phép bộ thu (REN=1) và cờ ngắt phát (TP=1) để bộ phát sẵn sàng hoạt động.

2.2. Chế độ 0 (Thanh ghi dịch đơn 8 bit) :

Chế độ 0 được chọn bằng các thanh ghi các bit 0 vào SM1 và SM2 của SCON, đưa port nối tiếp vào chế độ thanh ghi dịch 8bit. Dữ liệu nối tiếp vào và ra qua RXD và TXD xuất xung nhịp dịch, 8 bit được phát hoặc thu với bit đầu tiên là LSB. Tốc độ baud cố định ở 1/12 tần số dao động trên chip.

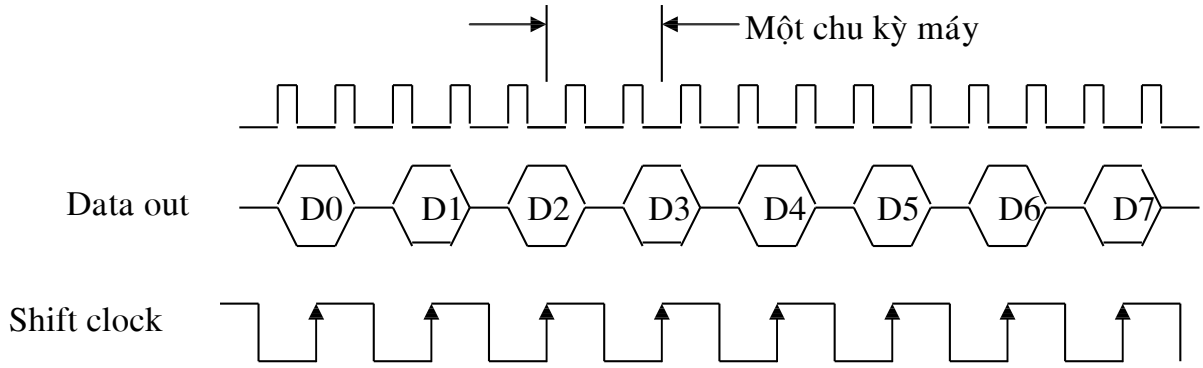
Việc phát đi được khởi động bằng bất cứ lệnh nào ghi dữ liệu vào SBUF. Dữ liệu dịch ra ngoài trên đường RXD (P3.0) với các xung nhịp được gửi ra đường TXD (P3.1). Mỗi bit phát đi hợp lệ (trên RXD) trong một chu kỳ máy, tín hiệu xung nhập xuống thấp ở S3P1 và trở về cao ở S6P1.



Giản đồ thời gian Port nối tiếp phát ở chế độ 0

II.

Việc thu được khởi động khi cho phép bộ thu (REN) là 1 và bit ngắt thu (RI) là 0. Quy tắc tổng quát là đặt REN khi bắt đầu chương trình để khởi động port nối tiếp, rồi xoá RI để bắt đầu nhận dữ liệu. Khi RI bị xoá, các xung nhịp được đưa ra đường TXD, bắt đầu chu kỳ máy kế tiếp và dữ liệu theo xung nhịp ở đường RXD. Lấy xung nhịp cho dữ liệu vào port nối tiếp xảy ra ở cạnh đường của TXD.



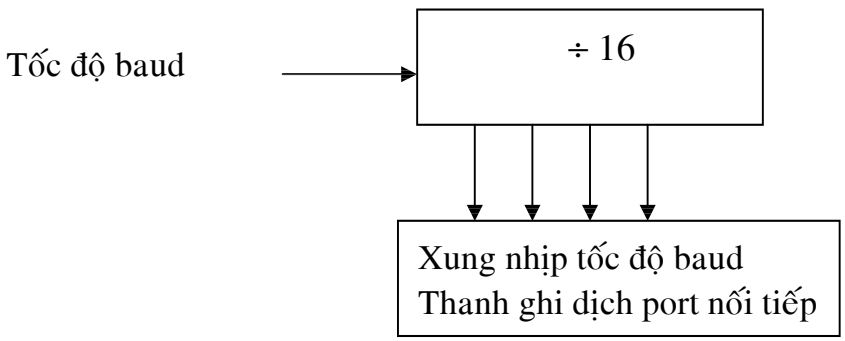
Giản đồ thời gian phát nối tiếp ở chế

2.3. Chế độ 1 (UART 8 bit với tốc độ baud thay đổi được):

Ở chế độ 1, port nối tiếp của 8951 làm việc như một UART 8 bit với tốc độ baud thay đổi được. Một UART (Bộ thu phát đồng bộ vạn năng) là một dụng cụ thu phát dữ liệu nối tiếp với mỗi ký tự dữ liệu đi trước là bit start ở mức thấp và theo sau bit stop ở mức cao. Đôi khi xen thêm bit kiểm tra chẵn lẻ giữa bit dữ liệu cuối cùng và bit stop. Hoạt động chủ yếu của UART là chuyển đổi song song sang nối tiếp với dữ liệu nhập.

Ở chế độ 1, 10 bit được phát trên TXD hoặc thu trên RXD. Những bit đó là: 1 bit start (luôn luôn là 0), 8 bit dữ liệu (LSB đầu tiên) và 1 bit stop (luôn luôn là 1). Với hoạt động thu, bit stop được đưa vào RB8 trong SCON. Trong 8951 chế độ baud được đặt bằng tốc độ báo tràn của timer 1.

Tạo xung nhịp và đồng bộ hóa các thanh ghi dịch của port nối tiếp trong các chế độ 1,2 và 3 được thiết lập bằng bộ đếm 4 bit chia cho 16, ngõ ra là xung nhịp tốc độ baud. Ngõ vào của bộ đếm này được chọn qua phần mềm



2.4. UART 9 bit với tốc độ baud cố định (chế độ 2):

Khi SM1=1 và SM0=0, cổng nối tiếp làm việc ở chế độ 2, như một UART 9bit có tốc độ baud cố định, 11 bit sẽ được phát hoặc thu:1bit start, 8 bit data, 1 bit data thứ 9 có thể được lập trình và 1 bit stop. Khi phát bit thứ 9 là bất cứ gì đã được

đưa vào TB8 trong SCON (có thể là bit Parity) .Khi thu bit thứ 9 thu được sẽ ở trong RB8. Tốc độ baud ở chế độ 2 là 1/32 hoặc 1/16 tần số dao động trên chip.

2.5. UART 9 bit với tốc độ baud thay đổi được (chế độ 3):

Chế độ này giống như ở chế độ 2 ngoại trừ tốc độ baud có thể lập trình được và được cung cấp bởi Timer.Thật ra các chế độ 1, 2, 3 rất giống nhau. Cái khác biệt là ở tốc độ baud (cố định trong chế độ 2, thay đổi trong chế độ 1 và 3) và ở số bit data (8 bit trong chế độ 1,9 trong chế độ 2 và 3).

2.6. Khởi động và truy xuất các thanh ghi cổng nối tiếp:

¨ ***Cho Phép Thu***

Bit cho phép bộ thu (REN=Receiver Enable) Trong SCON phải được đặt lên 1bằng phần mềm để cho phép thu các ký tự thông thường thực hiện việc này ở đầu chương trình khi khởi động cổng nối tiếp, timer ... Có thể thực hiện việc này theo hai cách. Lệnh:

SETB REN ; đặt REN lên 1

Hoặc lệnh

MOV SCON,#XXX1XXXXB ; đặt REN lên 1 hoặc xoá các bit khác trên SCON khi cần (các X phải là 0 hoặc 1 để đặt chế độ làm việc)

¨ ***Bit dữ liệu thứ 9:***

Bit dữ liệu thứ 9 cần phát trong các chế độ 2 và 3 phải được nạp vào trong TB8 bằng phần mềm. Bit dữ liệu thứ 9 thu được đặt ở RB8. Phần mềm có thể cần hoặc không cần bit dữ liệu thứ 9, phụ thuộc vào đặc tính kỹ thuật của thiết bị nối tiếp sử dụng (bit dữ liệu thứ 9 cũng đóng vai trò quan trọng trong truyền thông đa xử lý)

¨ ***Thêm 1 bit parity:***

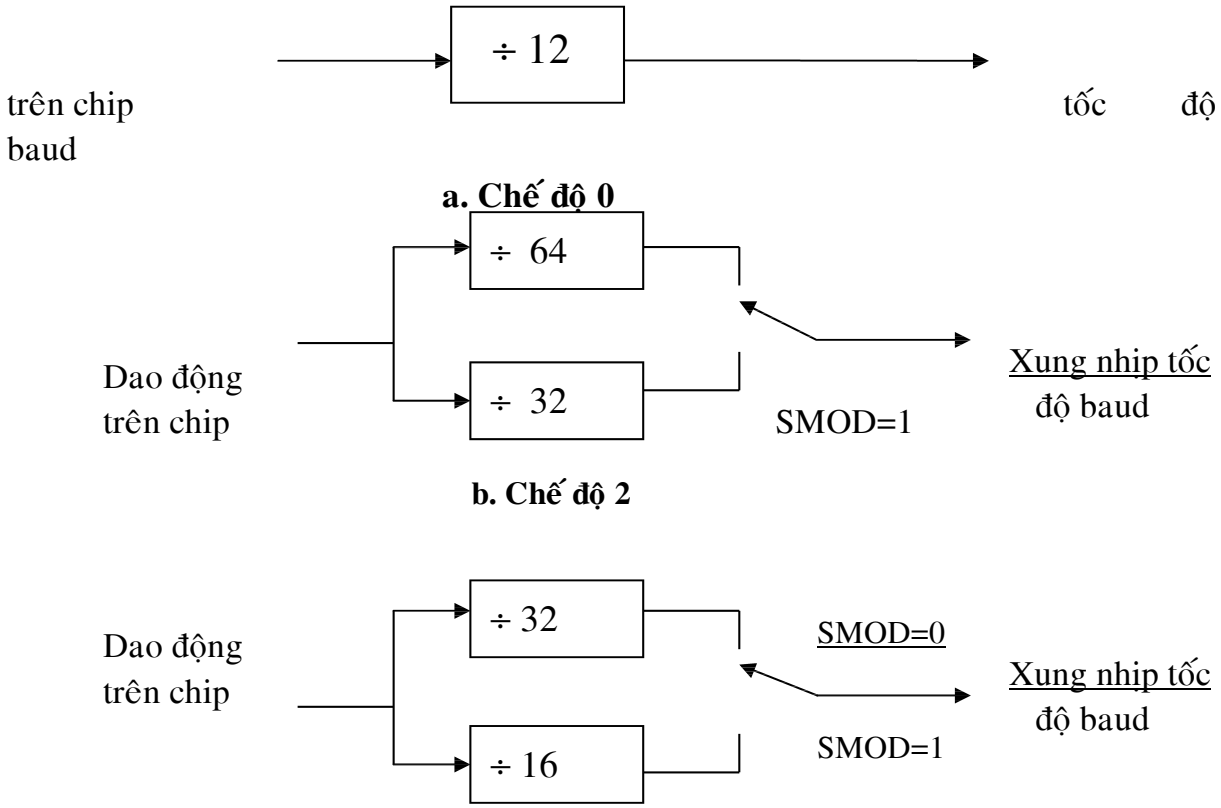
Thường sử dụng bit dữ liệu thứ 9 để thêm parity vào ký tự. Như đã nhận xét ở chương trước, bit P trong từ trạng thái chương trình (PSW) được đặt lên 1 hoặc bị xoá bởi chu kỳ máy để thiết lập kiểm tra chẳn với 8 bit trong thanh tích lũy.

♦ *Các cờ ngắt*:

Hai cờ ngắt thu và phát (RI và TI) trong SCON đóng một vai trò quan trọng trong truyền thông nối tiếp dùng 8951/8051. Cả hai bit được đặt lên 1 bằng phần cứng, nhưng phải được xoá bằng phần mềm.

2.7. Tốc độ baud port nối tiếp

Như đã nói, tốc độ baud cố định ở các chế độ 0 và 2. Trong chế độ 0 nó luôn luôn là tần số dao động trên chip được chia cho 12. Thông thường thạch anh ấn định tần số dao động trên chip nhưng cũng có thể sử dụng nguồn xung nhịp khác.



Mặc nhiên sau khi reset hệ thống, tốc độ baud chế độ 2 là tần số bộ dao động chia cho 64, tốc độ baud cũng bị ảnh hưởng bởi 1 bit trong thanh ghi điều khiển nguồn cung cấp (PCON) bit 7 của PCON là bit SMOD. Đặt bit SMOD lên 1 làm gấp đôi tốc độ baud trong các chế độ 1, 2 và 3. Trong chế độ 2, tốc độ baud có thể bị gấp đôi từ giá trị mặc nhiên của 1/64 tần số dao động (SMOD=0) đến 1/32 tần số dao động (SMOD=1)

Vì PCON không được định địa chỉ theo bit, nên để đặt bit SMOD lên 1 cần phải theo các lệnh sau:

```
MOV A,PCON ; lấy giá trị hiện thời của PCON
SETB ACC.7  ; đặt bit SMOD lên 1
MOV PCON,A  ; ghi giá trị ngược về  PCON
```

Các tốc độ baud trong các chế độ 1 và 3 được xác định bằng tốc độ tràn của timer 1. Vì timer hoạt động ở tần số tương đối cao, tràn timer được chia thêm cho 32 (hoặc 16 nếu SMOD =1) trước khi cung cấp tốc độ xung nhịp cho port nối tiếp.

3. Tổ chức ngắt trong 8051

Vi Điều Khiển có 5 nguồn ngắt:2 nguồn ngắt ngoài,2 ngắt timer và 1 ngắt Port nối tiếp, tất cả các nguồn ngắt bị cấm sau khi reset hệ thống và cho phép bởi phần mềm

3.1.Cho Phép và Không Cho Phép Ngắt

Mỗi nguồn ngắt được cho phép hoặc không cho phép thông qua thanh ghi chức năng đặc biệt có các bit được địa chỉ hóa IE (Interrupt Enable) tại địa chỉ 0A8H.

BIT	SYMBOL	BIT ADDRESS	DESCRIPTION (1:ENABLE,0:DISABLE)
IE.7	EA	AFH	Global Enable/Disable
IE.6	EA	AEH	Undefined
IE.5	ET2	ADH	Enable Timer 2 Interrupt (8052)
IE.4		ACH	Enable Serial Port Interrupt
IE.3	ET1	ABH	Enable Timer 1 Interrupt
IE.2		AAH	Enable External 1 Interrupt
IE.1	ET0	A9H	Enable Timer 0 Interrupt
IE.0	EX0	A8H	Enable External 0 Interrupt

a. ***Ưu tiên ngắt.***

Mỗi nguồn ngắt được lập trình riêng vào một trong hai mức ưu tiên qua thanh ghi chức năng đặc biệt được địa chỉ bit Ip (Interrupt priority : ưu tiên ngắt) ở địa chỉ B8H.

Bit	Ký hiệu	Địa chỉ bit	Mô tả (1=mức cao hơn,0=mức thấp)
IP.7			Không được định nghĩa
IP.6			Không được định nghĩa
IP.5	PT2	BDH	Ưu tiên cho ngắt từ timer 2 (8052)
IP.4	PS	BCH	Ưu tiên cho ngắt Port nối tiếp
IP.3	PT1	BBH	Ưu tiên cho ngắt từ timer 1
IP.2	PX1	BAH	Ưu tiên cho ngắt ngoài
IP.1	PT0	B9H	Ưu tiên cho ngắt từ timer 0
IP.0	PX0	B8H	Ưu tiên cho ngắt ngoài 0

Tóm tắt thanh ghi IP.

Các ngắt ưu tiên được xóa sau khi reset hệ thống để đặ ttất cả các ngắt ở mức ưu tiên thấp hơn.

3.2 Xử lý ngắt.

Khi có một ngắn xẩy ra và được CPU chấp nhận, chương trình chính bị ngắt quãng. Những hoạt động sau xẩy ra:

- Thi hành hoàn chỉnh lệnh đang hiện hành.
- Các DC vào ngắt xếp.
- Trạng thái ngắt hiện hành được cất bên trong.
- Các ngắt được chặn tại mức của ngắt.

- Nap vàp DC địa chỉ Vector của ISR.
- ISR thực thi.

ISR thực thi và đáp ứng ngắt. ISR hoàn tất bằng lệnh RET1. Điều này làm lấy lại giá trị cũ của PC từ ngăn xếp và lấy lại trạng thái ngắt cũ. Chương trình lại tiếp tục thi hành tại nơi mà nó dừng.

3.3 VécTơ Ngắt

Khi ngắt được chấp nhận giá trị được đưa vào PC (Program Counter) gọi là vector ngắt (Interrupt Vector)

INTERRUPT	FLAG	VECTOR ADDRESS
System Reset	RST	0000 H
External 0	IE0	0003 H
Timer 0	TF0	000B H
External 1	IE1	0013 H
Timer 1	TF1	001B H
Serial Port	RI OR TI	0023 H
Timer 2	TF2 OR EXF2	002B H

3.4 Ngắt Port nối Tiếp

Ngắt Port nối tiếp xảy ra khi cả 2 cờ ngắt truyền (TI) hoặc cờ ngắt nhận (RI) được đặt. Ngắt truyền xảy ra khi bit cuối cùng trong SBUF truyền xong tức là lúc này thanh ghi SBUF rỗng .Ngắt nhận xảy ra khi SBUF đã hoàn thành việc nhận và đang đợi để đọc tức là lúc này thanh ghi SBUF đầy. Cả hai cờ ngắt này được đăt bởi phần cứng và xóa bằng phần mềm.

Các ngắt của 8051.

a. ***Các ngắt timer.***

Các ngắt timer có địa chỉ Vector ngắt là 000BH (timer 0) và 001BH (timer 1). Ngắt timer xẩy ra khi các thanh ghi timer (TLx ITHx) tràn và set cờ báo tràn (TFx) lên 1. Các cờ timer (TFx) không bị xóa bằng phần mềm. Khi cho phép các ngắt, TFx tự động bị xóa bằng phần cứng khi CPU chuyển đến ngắt.

b. Các ngắt cổng nối tiếp.

Ngắt cổng nối tiếp xẩy ra khi hoặc cờ phát (TI) hoặc cờ ngắt thu (KI) được đặt lên 1. Ngắt phát xẩy ra khi một ký tự đã được nhận xong và đang đợi trong SBUP để được đọc.

Các ngắt cổng nối tiếp khác với các ngắt timer. Cờ gây ra ngắt cổng nối tiếp không bị xóa bằng phần cứng khi CPU chuyển tới ngắt. Do có hai nguồn ngắt cổng nối tiếp Ti và RI. Nguồn ngắt phải được xác định trong ISR và cờ tạo ngắt sẽ được xóa bằng phần mềm. Các ngắt timer cờ ngắt cờ ngắt được xóa bằng phần cứng khi CPU hướng tới ISR.

c. ***Các ngắt ngoài.***

- Các ngắt ngoài xẩy ra khi có một mức thấp hoặc cạnh xuống trên chân INT0 hoặc INT1 của vi điều khiển. Đây là chức năng chuyển đổi của các bit Port 3.(Port 3.2 và Port 3.3).

Các cờ tạo ngắt này là các bit IE0 vá IE1 trong TCON. Khi quyền điều khiển đã chuyển đến ISR, cờ tạo ra ngắt chỉ được xóa nếu ngắt được tích cực bằng cạnh xuống. Nếu ngắt được tích cực theo mức, thì nguồn yêu cầu ngắt bên ngoài sẽ điều khiển mức của cờ thay cho phần cứng.

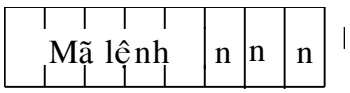
Sự lựa chọn ngắt tích cực mức thấp hay tích cực cạnh xuống được lập trình qua các bit IT0 và IT1 trong TCON. Nếu IT1 = 0, ngắt ngoài 1 được tác động bằng múc thấp ở chân IT1. Nếu IT1 = 1 ngắt ngoài 1 sẽ được tác động bằng cạnh xuống. trong chế độ này, nếu các mẫu liên tiếp trên chân INT1 chỉ mức cao trong một chu kỳ và chỉ mức thấp trong chu kỳ kế, cờ yêu cầu ngắt IE1 trong TCON được đặt lên 1, rồi bit IÉ yêu cầu ngắt.

Nếu ngắt ngoài được tác động bằng cạnh xuống thì nguồn bên ngoài phải giữ chân tác động ở mức cao tối thiểu một chu kỳ và giữ nó ở mức thấp thêm một chu kỳ nữa để đảm bảo phát hiện được cạnh xuống. Nếu ngắt ngoài được tác động theo mức thì nguồn bên ngoài phải giữ tín hiệu yêu cầu tác động cho đến khi ngắt được yêu cầu được thật sự tạo ra và không tác động yêu cầu ngắt trước khi ISR được hoàn tất . Nếu không một ngắt khác sẽ được lặp lại.

VI. CÁC CHẾ ĐỘ ĐÁNH ĐỊA CHỈ: TRONG TẬP LỆNH CÓ 8 CHẾ ĐỘ ĐÁNH ĐỊA CHỈ:

a. ***Thanh ghi địa ghi:***

8051/8031 có 4 bank thanh ghi, mỗi bank có 8 thanh ghi đ1nh số từ R0 đến R7. Tại mỗi thời điểm chỉ có một bank thanh ghi được tích cực. Muốn chọn bank thanh ghi nào ta chỉ cần gán các bit nhị phân thích hợp vào RSI (PSW.4) và RS0(PSW.3) trong thanh ghi trạng thái chương trình (PSW).



Địa chỉ thanh ghi.

Ngoài ra, một số thanh ghi đặc biệt như thanh ghi tích lũy, con trỏ dữ liệu.. cũng được xác định trong các lệnh nên không cần các bit địa chỉ. Trong các lệnh này thanh ghi tích lũy được xác định là “A”, con trỏ dữ liệu là “DPTR”, thanh ghi đếm chương trình là “PC”, cờ nhớ là “C”, cặp thanh ghi tích lũy B là “AB”.

b. Địi

c. a chỉ trực tiếp.

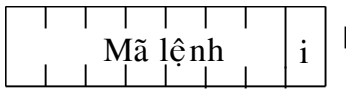
Trong chế độ này, các thanh ghi bên trong 8051/8031 được đánh địa chỉ trực tiếp bằng 8 bit địa chỉ nằm trong byte thứ hai của mã lệnh.

Địa chỉ trực tiếp.

Dù vậy, trình hợp dịch cho phép gọi tên các thanh ghi chức năng đặc biệt (có địa chỉ trực tiếp từ 80H đến FFH) ví dụ :P0 cho port 0, TMOD cho thanh ghi chế độ timer...

d. Địa chỉ gián tiếp.

R0 và R1 được dùng để chứa địa chỉ ô nhớ mà lệnh tác động đến. người ta quy ước dùng dấu @ trước R0 hoặc R1.



Địa chỉ gián tiếp.

e. Địa chỉ tức thời:

Người ta dùng # trước các toán hạng tức thời. Các toán hạng đó có thể là một hằng số, một ký số hay một biểu thức toán học... Trường hợp dịch sẽ tự động tính toán và thay thế dữ liệu trực tiếp vào mã lệnh.

Địa chỉ tức thời.

f. Địa chỉ tương đối:

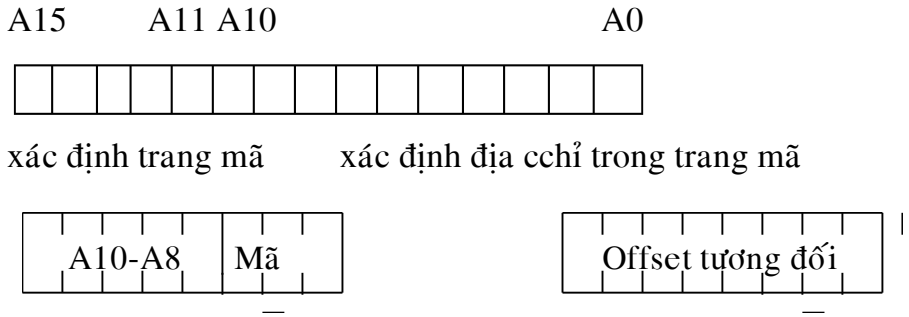
Địa chỉ tương đối được dùng trong các lệnh nhảy 8051/8031 dùng giá trị 8 bit có dấu để cộng thêm vào thanh ghi đếm chương trình (PC). Tầm nhảy của lệnh này trong khoảng từ –128 đến 127 ô nhớ. Trước khi cộng , thanh ghi PC sẽ tăng đến địa chỉ theo sau lệnh nhảy rồi tính toán địa chỉ offset cần thiết để nhảy đến địa chỉ yêu cầu. Như vậy địa chỉ mới là địa chỉ tương đối so với lệnh kế tiếp chứ không phải là bản thân lệnh nhảy. Thường lệnh này có liên quan đến nhãn được định nghĩa trước.

Địa chỉ tương đối.

g. Địa chỉ tuyệt đối:

Địa chỉ tuyệt đối chỉ dùng trong các lệnh ACALL và JIMP. Các lệnh 2 byte này dùng để rẽ nhánh vào một trang 2 Kbyte của bộ nhớ trương trình bằng cách cấp 11

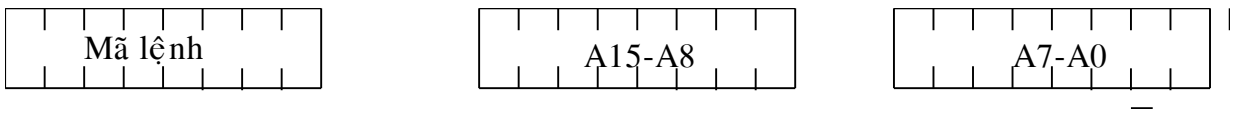
bit địa chỉ thấp (A0-A10) để xác định địa chỉ đích trong trang mã. Còn 5 bit cao của địa chỉ đích (A11-A15) chính là 5 bit cao hiện hành trong thanh ghi đếm chương trình. Vì vậy địa chỉ của lệnh theo sau lệnh rẽ nhánh và địa chỉ đích của lệnh rẽ nhánh và địa chỉ đích của lệnh rẽ nhánh cần phải cùng trang mã 2 Kbyte (có cùng 5 bit địa chỉ cao).



Địa chỉ tuyệt đối.

h. Địa chỉ dài:

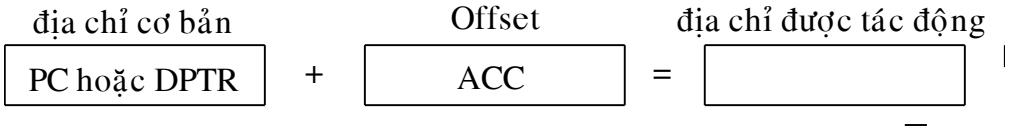
Địa chỉ dài chỉ dùng cho lệnh LCALL và LJIMP. Các lệnh này chiếm 3 byte và dùng 2 byte sau (byte 2 và byte 3) để định địa chỉ đích của lệnh (16 bit). Ưu điểm của lệnh này có thể sử dụng trong toàn bộ vùng nhớ 64 Kbyte. Tuy nhiên, lệnh này chiếm nhiều byte và lệ thuộc vào vị trí vùng nhớ.



Địa chỉ dài.

i. ***Địa chỉ tham chiếu:***

Địa chỉ tham chiếu dùng một thanh ghi cơ bản (hoặc thanh ghi đếm chương trình PC hoặc thanh ghi con trỏ dữ liệu DPTR) và địa chỉ offset (trong thanh ghi tích lũy A) để tạo địa chỉ được tác động cho các lệnh JMP hoặc MOVC. Các bảng nhảy và bảng tìm kiếm dễ dàng được tạo ra để sử dụng địa chỉ tham chiếu.



Địa chỉ tham chiếu.

VII. CÁC NHÓM LỆNH CỦA 8951

Tập lệnh của 8951 được chia thành 5 nhóm:

- Số học.
- Luận lý.
- Chuyển dữ liệu.
- Chuyển điều khiển.

Các chi tiết thiết lập lệnh:

Rn	:Thanh ghi R0 đến R7 của bank thanh ghi được chọn.
Data	: 8 bit địa chỉ vùng dữ liệu bên trong. Nó có thể là vùng RAM dữ liệu trong (0-127) hoặc các thanh ghi chức năng đặc biệt.
@Ri	: 8 bit vùng RAM dữ liệu trong (0-125) được đánh giá địa chỉ gián tiếp qua thanh ghi R0 hoặc R1.
#data	: Hằng 8 bit chức trong câu lệnh.
#data 16	: Hằng 16 bit chứa trong câu lệnh.
Addr16	: 16 bit địa chỉ đích được dùng trong lệnh LCALL và LJMP.
Addr11	: 11 bit địa chỉ đích được dùng trong lệnh LCALL và AJMP.
Rel	: Byte offset 8 bit có dấu được dùng trong lệnh SJMP và những lệnh nhảy có điều kiện.
Bit	: Bit được định địa chỉ trực tiếp trong RAM dữ liệu nội hoặc các thanh ghi chức năng đặc biệt.

a. Nhóm lệnh xử lý số học:

ADD A,Rn (1byte, 1 chu kỳ máy) : cộng nội dung thanh ghi Rn vào thanh ghi A.

ADD A,data	(2,1): Cộng trực tiếp 1 byte vào thanh ghi A.
ADD A,@Ri	(1,1): Cộng gián tiếp nội dung RAM chứa tại địa chỉ được khai báo trong Ri vào thanh ghi A.
ADD A,#data	(2,1):Cộng dữ liệu tức thời vào A.
ADD A,Rn	(1,1): Cộng thanh ghi và cờ nhớ vào A.
ADD A,data	(2,1): Cộng trực tiếp byte dữ liệu và cờ nhớ vào A.
ADDC A,@Ri	(1,1): Cộng gián tiếp nội dung RAM và cờ nhớ vào A.
ADDC A,#data	(2,1): Cộng dữ liệu tức thời và cờ nhớ vào A.
SUBB A,Rn	(1,1): Trừ nội dung thanh ghi A cho nội dung thanh ghi Rn và cờ nhớ.
SUBB A,data	(2,1): Trừ trực tiếp A cho một số và cờ nhớ.
SUBB A,@Ri	(1,1): Trừ gián tiếp A cho một số và cờ nhớ.
SUBB A,#data	(2,1): Trừ nội dung A cho một số tức thời và cờ nhớ.
INC A	(1,1): Tăng nội dung thanh ghi A lên 1.
INC Rn	(1,1): Tăng nội dung thanh ghi Rn lên 1.
INC data	(2,1): Tăng dữ liệu trực tiếp lên 1.
INC @Ri	(1,1): Tăng gián tiếp nội dung vùng RAM lên 1.
DEC A	(1,1): Giảm nội dung thanh ghi A xuống 1.
DEC Rn	(1,1): Giảm nội dung thanh ghi Rn xuống 1.
DEC data	(2,1): Giảm dữ liệu trực tiếp xuống 1
DEC @Ri	(1,1): Giảm gián tiếp nội dung vùng RAM xuống 1.
INC DPTR	(1,2): Tăng nội dng con trỏ dữ liệu lên 1.
MUL AB	(1,4): Nhân nội dung thanh ghi A với nội dung thanh ghi B.
DIV AB	(1,4): Chia nội dung thanh ghi A cho nội dung thanh ghi B.
DA A	(1,1,): hiệu chỉnh thập phân thanh ghi A.

b. **Nhóm lệnh luận lý:**

ANL A,Rn	(1,1): AND nội dung thanh ghi A với nội dung thanh ghi Rn.
ANL A,data	(2,1): AND nội dung thanh ghi A với dữ liệu trực tiếp.
ANL A,@Ri	(1,1): AND nội dung thanh ghi A với dữ liệu gián tiếp trong RAM.
ANL A,#data	(2,1): AND nội dung thanh ghi với dữ liệu tức thời.
ANL data,A	(2,1): AND một dữ liệu trực tiếp với A.
ANL data,#data	(3,2): AND một dữ liệu trực tiếp với A một dữ liệu tức thời.
ANL C,bit	(2,2): AND cờ nhớ với 1 bit trực tiếp.
ANL C,/bit	(2,2): AND cờ nhớ với bù 1 bit trực tiếp.
ORL A,Rn	(1,1): OR thanh ghi A với thanh ghi Rn.
ORL A,data	(2,1): OR thanh ghi A với một dữ liệu trực tiếp.
ORL A,@Ri	(1,1): OR thanh ghi A với một dữ liệu gián tiếp.
ORL A,#data	(2,1): OR thanh ghi A với một dữ liệu tức thời.
ORL data,A	(2,1): OR một dữ liệu trực tiếp với thanh ghi A.
ORL data,#data	(3,1) :OR một dữ liệu trực tiếp với một dữ liệu tức thời.
ORL C,bit	(2,2): OR cờ nhớ với một bit trực tiếp.
ORL C,/bit	(2,2): OR cờ nhớ với bù của một bit trực tiếp.
XRL A,Rn	(1,1): XOR thanh ghi A với thanh ghi Rn.

XRL A,data	(2,1): XOR thanh ghi A với mộ dữ liệu trực tiếp.
XRL A,@Ri	(1,1): XOR thanh ghi A với một dữ liệu gián tiếp.
XRL A,#data	(2,1): XOR thanh ghi A với mộ dữ liệu tức thời.
XRL data,A	(2,1): XOR một dữ liệu trực tiếp với thanh ghi A.
XRL dara,#data	(3,1): XOR một dữ liệu trực tiếp với một dữ liệu tức thời.
SETB C	(1,1): Đặt cờ nhớ.
SETB bit	(2,1): Đặt một bit trực tiếp.
CLR A	(1,1): Xóa thanh ghi A.
CLR C	(1,1): Xóa cờ nhớ.
CPL A	(1,1): Bù nội dung thanh ghi A.
CPL C	(1,1): Bù cờ nhớ.
CPL bit	(2,1): Bù một bit trực tiếp.
RL A	(1,1): Quay trái nội dung thanh ghi A.
RLC A	(1,1): Quay trái nội dung thanh ghi A qua cờ nhớ.
RR A	(1,1): Quay phải nội dung thanh ghi A.
RRC A	(1,1): Quay phải nội dung thanh ghi A qua cờ nhớ.
SWAP	(1,1): Quay trái nội dung thanh ghi A 1 nibble (1/2byte).

c. Nhóm lệnh chuyển dữ liệu:

MOV A,Rn	(1,1):Chuyển nội dung thanh ghi Rn vào thanh ghi A.
MOV A,data	(2,1): Chuyển dữ liệu trực tiếp vào thanh ghi A.
MOV A,@Ri	(1,1): Chuyển dữ liệu gián tiếp vào thanh ghi A.
MOV A,#data	(2,1): Chuyển dữ liệu tức thời vào thanh ghi A.
MOV Rn,data	(2,2): Chuyển dữ liệu trực tiếp vào thanh ghi Rn.
MOV Rn,#data	(2,1): Chuyển dữ liệu tức thời vào thanh ghi Rn.
MOV data,A	(2,1): Chuyển nội dung thanh ghi A vào một dữ liệu trực tiếp.
MOV data,Rn	(2,2): Chuyển nội dung thanh ghi Rn vào một dữ liệu trực tiếp.
MOV data,data	(3,2): Chuyển một dữ liệu trực tiếp vào một dữ liệu trực tiếp.
MOV data,@Ri	(2,2): Chuyển một dữ liệu gián tiếp vào một dữ liệu gián tiếp.
MOV data,#data	(3,2): Chuyển một dữ liệu tức thời vào một dữ liệu trực tiếp.
MOV @Ri,A	(1,1): Chuyển nội dung thanh ghi A vào một dữ liệu gián tiếp.
MOV @Ri,data	(2,2): Chuyển một dữ liệu trực tiếp vào một dữ liệu gián tiếp.
MOV @Ri,#data	(2,1): Chuyển dữ liệu tức thời vào dữ liệu gián tiếp.
MOV DPTR,#data	(3,2): Chuyển một hằng 16 bit vào thanh ghi con trỏ dữ liệu.
MOV C,bit	(2,1): Chuyển một bit trực tiếp vào cờ nhớ.
MOV bit,C	(2,2): Chuyển cờ nhớ vào một bit trực tiếp.
MOV A,@A+DPTR	(1,2): Chuyển byte bộ nhớ chương trình có địa chỉ là @A+DPRT vào thanh ghi A.
MOVC A,@A+PC	(1,2): Chuyển byte bộ nhớ chương trình có địa chỉ là @A+PC vào thanh ghi A.
MOVX A,@Ri	(1,2): Chuyển dữ liệu ngoài (8 bit địa chỉ) vào thanh ghi A.
MOVX A,@DPTR	(1,2): Chuyển dữ liệu ngoài (16 bit địa chỉ) vào thanh ghi A.
MOVX @Ri,A	(1,2): Chuyển nội dung A ra dữ liệu ngoài (8 bit địa chỉ).
MOVX @DPTR,A	(1,2): Chuyển nội dung A ra dữ liệu bên ngoài (16 bit địa chỉ).
PUSH data	(2,2): Chuyển dữ liệu trực tiếp vào ngăn xếp và tăng SP.

POP data (2,2): Chuyển dữ liệu trực tiếp vào ngăn xếp và giảm SP.
XCH A,Rn (1,1): Trao đổi dữ liệu giữa thanh ghi Rn v2 thanh ghi A.
XCH A,data (2,1): Trao đổi giữa thanh ghi A và một dữ liệu trực tiếp.
XCH A,@Ri (1,1): Trao đổi giữa thanh ghi A và một dữ liệu gián tiếp.
XCHD A,@R (1,1): Trao đổi giữa nibble thấp (LSN) của thanh ghi A và LSN của dữ liệu gián tiếp.

d. Nhóm lệnh chuyển điều khiển:

ACALL addr11 (2,2): Gọi chương trình con dùng địa chì tuyệt đối.
LCALL addr16 (3,2): Gọi chương trình con dùng địa chỉ dài.
RET (1,2): Trở về từ lệnh gọi chương trình con.
RET1 (1,2): Trở về từ lệnh gọi ngắt.
AJMP addr11 (2,2): Nhảy tuyệt đối.
LJMP addr16 (3,2): Nhảy dài.
SJMP rel (2,2):Nhảy ngắn.
JMP @A+DPTR (1,2): Nhảy gián tiếp từ con trỏ dữ liệu.
JZ rel (2,2): Nhảy nếu A=0.
JNZ rel (2,2): Nhảy nếu A không bằng 0.
JC rel (2,2): Nhảy nếu cờ nhớ được đặt.
JNC rel (2,2): Nhảy nếu cờ nhớ không được đặt.
JB bit,rel (3,2): Nhảy tương đối nếu bit trực tiếp được đặt.
JNB bit,rel (3,2):Nhảy tương đối nếu bit trực tiếp không được đặt.
JBC bit,rel (3,2): Nhảy tương đối nếu bit trực tiếp được đặt , rồi xóa bit.
CJNE A,data,rel (3,2): So sánh dữ liệu trực tiếp với A và nhảy nếu không bằng.
CJNE A,#data,rel (3,2): So sánh dữ liệu tức thời với A và nhảy nếu không bằng.
CJNE Rn,#data,rel (3,2): So sánh dữ liệu tức thời với nội dung thanh ghi Rn và nhảy nếu không bằng.
CJNE @Ri,#data,rel (3,2): So sánh dữ liệu tức thời với dữ liệu gián tiếp và nhảy nếu không bằng.
DJNZ Rn,rel (2,2): Giản thanh ghi Rn và nhảy nếu không bằng.
DJNZ data,rel (3,2): Giảm dữ liệu trực tiếp và nhảy nếu không bằng.

e. Các lệnh rẽ nhánh:

Có nhiều lệnh để điều khiển lên chương trình bao gồm việc gọi hoặc trả lại từ chương trình con hoặc chia nhánh có điều kiện hay không có điều kiện.
Tất cả các lệnh rẽ nhánh đều không ảnh hưởng đến cờ. Ta có thể định nhản cần nhảy tới mà không cần rõ địa chỉ, trình biên dịch sẽ đặt địa chỉ nơi cần nhảy tới vào đúng khẩu lệnh đã đưa ra.

Mô tả tập lệnh :

Tóm Tắt Các Lệnh NHẢY (JMP)

Nhảy có điều kiện:

<condition>	Jump_if_not <conditon>	Jump_if_<conditon>
C = 1	**JNC** rel	**JC** rel
bit = 1	**JNB** bit, rel	**JB** bit, rel / **JBC** bit, rel
A = 0	**JNZ** rel	**JZ** rel
Rn = 0	**DJNZ** Rn, rel	
direct = 0	**DJNZ** direct, rel	
A ≠ direct	**CJNE** A, direct, rel	
A ≠ #data	**CJNE** A, #data, rel	
Rn ≠ #data	**CJNE** Rn, #data, rel	
@Ri ≠ #data	**CJNE** @Ri, #data, rel	

Nhảy không điều kiện: **AJMP, LJMP, SJMP.**

1. **Cấu trúc "repeat... until"**

```
Repeat
                <action>
Until           <condition>
```

Ngôn ngữ Assembly

```
LOOP:
                <action>
                JUMP_if_not_<condition>,LOOP
```

VD: Cấu trúc "repeat... until"

```
Repeat
                ...
Until           A = 0
```

Ngôn ngữ Assembly

```
LOOP:
                ...

                JNZ LOOP
```

2. **Cấu trúc "while... do"**

```
while <condition> do <action>
```

Ngôn ngữ Assembly

```
LOOP:           JUMP_if_not_<condition>,DO
                SJMP            STOP
DO:             <action>
                SJMP            LOOP
STOP:           ...
```

VD: Cấu trúc "while... do"

```
R7 = 0
while R7 ¹ 10 do {
                ...
                R7 = R7 + 1
                  }
```

Ngôn ngữ Assembly

```
                MOV             R7,#0
LOOP:           CJNE            R7,#10,DO
                SJMP            STOP
DO:             ...
                INC             R7
                SJMP            LOOP
STOP:           ...
```

3. **Cấu trúc "if... then... else"**

```
if <condition> then
                <action 1>
else            <action 2>
```

Ngôn ngữ Assembly

```
                JUMP_if_not_<condition>,ELSE
                <action 1>
SJMP            DONE
ELSE:           <action 2>
DONE:           ...
```

VD: Cấu trúc "if... then... else"

```
if P0.1 = 0 then
                R7 = R7 + 1
else            R7 = 0
```

Ngôn ngữ Assembly

```
                JB              P0.1,ELSE
INC             R7
SJMP            DONE
ELSE:           MOV             R7,#0
DONE:           ...
```

4. Cấu trúc "case... of..."

```
case P1 of
                #11111110b: P2.0 = 1
                #11111101b: P2.1 = 1
                #11111011b: P2.2 = 1
                else P2 = 0
end
```

Ngôn ngữ Assembly

```
                CJNE            P1,#11111110b,
SKIP1           SETB            P2.0
                SJMP            EXIT
SKIP1:          CJNE            P1,#11111101b,SKIP2
                SETB            P2.1
                SJMP            EXIT
SKIP2:          CJNE            P1,#11111011b,SKIP3
                SETB            P2.2
                SJMP            EXIT
SKIP3:          MOV             P2,#0
EXIT:           ...
```

Sau đây là sự tóm tắt từng hoạt động của lệnh nhảy.

JC rel : Nhảy đến "rel" nếu cờ Carry C = 1.
JNC rel : Nhảy đến "rel" nếu cờ Carry C = 0.
JB bit, rel : Nhảy đến "rel" nếu (bit) = 1.
JNB bit, rel : Nhảy đến "rel" nếu (bit) = 0.
JBC bit, rel : Nhảy đến "rel" nếu bit = 1 và xóa bit.
ACALL addr11: Lệnh gọi tuyệt đối trong page 2K.
(PC) (PC) + 2
(SP) (SP) + 1
((SP)) (PC7PC0)
(SP) (SP) + 1
((SP)) (PC15PC8)
(PC10PC0) page Address.
LCALL addr16: Lệnh gọi dài chương trình con trong 64K.
(PC) (PC) + 3
(SP) (SP) + 1
((SP)) (PC7PC0)
(SP) (SP) + 1
((SP)) (PC15PC8)
(PC) Addr15Addr0.
RET : Kết thúc chương trình con trở về chương trình chính.
(PC15PC8) (SP)
(SP) (SP) - 1
(PC7PC0) ((SP))
(SP) (SP) -1.

RETI : Kết thúc thủ tục phục vụ ngắt quay về chương trình chính hoạt động tương tự như RET.

AJMP Addr11 : Nhảy tuyệt đối không điều kiện trong 2K.
(PC) (PC) + 2
(PC10PC0) page Address.

LJMP Addr16 : Nhảy dài không điều kiện trong 64K
Hoạt động tương tự lệnh LCALL.

SJMP rel :Nhảy ngắn không điều kiện trong (-128127) byte
(PC) (PC) + 2
(PC) (PC) + byte 2

JMP @ A + DPTR:Nhảy không điều kiện đến địa chỉ (A) + (DPTR)
(PC) (A) + (DPTR)

JZ rel : Nhảy đến A = 0. Thực hành lệnh kế nếu A = 0.
(PC) (PC) + 2
(A) = 0 (PC) (PC) + byte 2

JNZ rel : Nhảy đến A 0. Thực hành lệnh kế nếu A = 0.
(PC) (PC) + 2
(A) < > 0 (PC) (PC) + byte 2

CJNE A, direct, rel : So sánh và nhảy đến A direct
(PC) (PC) + 3
(A) < > (direct) (PC) (PC) + Relative Address.
(A) < (direct) C = 1
(A) > (direct) C = 0
(A) = (direct). Thực hành lệnh kế tiếp

CJNE A, # data, rel : Tương tự lệnh CJNE A, direct, rel.

CJNE Rn, # data, rel : Tương tự lệnh CJNE A, direct, rel.

CJNE @ Ri, # data, rel : Tương tự lệnh CJNE A, direct, rel.

DJNE Rn, rel : Giảm Rn và nhảy nếu Rn 0.
(PC) (PC) + 2
(Rn) (Rn) -1
(Rn) < > 0 (PC) (PC) + byte 2.

DJNZ direct, rel : Tương tự lệnh DJNZ Rn, rel.

Các lệnh dịch chuyển dữ liệu:

Các lệnh dịch chuyển dữ liệu trong những vùng nhớ nội thực thi 1 hoặc 2 chu kỳ máy. Mẫu lệnh MOV <destination>, <source> cho phép di chuyển dữ liệu bất kỳ 2 vùng nhớ nào của RAM nội hoặc các vùng nhớ của các thanh ghi chức năng đặc biệt mà không thông qua thanh ghi A.

Vùng Ngăn xếp của 8951 chỉ chứa 128 byte RAM nội, nếu con trỏ Ngăn xếp SP được tăng quá địa chỉ 7FH thì các byte được PUSH vào sẽ mất đi và các byte POP ra thì không biết rõ.

Các lệnh dịch chuyển bộ nhớ nội và bộ nhớ ngoại dùng sự định vị gián tiếp. Địa chỉ gián tiếp có thể dùng địa chỉ 1 byte (@ Ri) hoặc địa chỉ 2 byte (@ DPTR). Tất cả các lệnh dịch chuyển hoạt động trên toàn bộ nhớ ngoài thực thi trong 2 chu kỳ máy và dùng thanh ghi A làm toán hạng DESTINATION.

Việc đọc và ghi RAM ngoài (RD và WR) chỉ tích cực trong suốt quá trình thực thi của lệnh MOVX, còn bình thường RD và WR không tích cực (mức 1).
Tất cả các lệnh dịch chuyển đều không ảnh hưởng đến cờ. Hoạt động của từng lệnh được tóm tắt như sau:

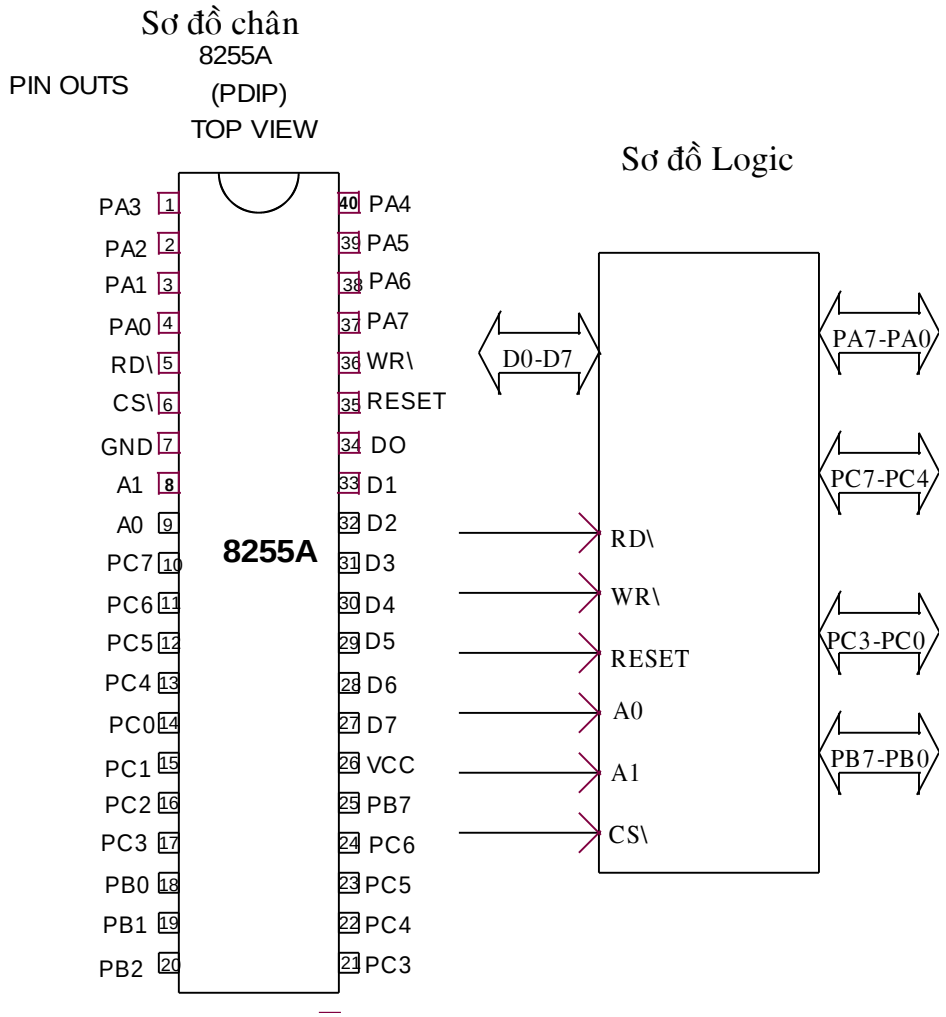
PUSH direct : Cất dữ liệu vào Ngăn xếp
(SP) (SP) + 1
(SP) (Drirect)
POP direct : Lấy từ Ngăn xếp ra direct
(direct) ((SP))
(SP) (SP) - 1
XCH A, Rn : Đổi chổ nội dung của A với Rn
(A) (Rn)
XCH A, direct : (A) (direct)
XCH A, @ Ri : (A) ((Ri))
XCHD A, @ Ri : Đổi chổ 4 bit thấp của (A) với ((Ri))
(A3A0) ((Ri3Ri0))

Các lệnh xen vào (Miscellamous Intstruction):
NOP : Không hoạt động gì cả, chỉ tốn 1 byte và 1 chu kỳ máy. Ta dùng để delay những khoảng thời gian nhỏ.

Chương 2 :
KHẢO SÁT VI MẠCH GIAO TIẾP NGOẠI VI 8255.

I CẤU TRÚC PHẦN CỨNG 8255A:

8255A là IC ngoại vi được chế tạo theo công nghệ LSI dùng để giao tiếp song song giữa Microprocrssor và thiết bị điều khiển bên ngoài.



Hình 3.1: Sơ đồ chân và sơ đồ logic 8255A.

Tên các chân 8255A:

D_7-D_0 Data bus (Bi-Direction).

RESET Reset input.

CS\ Chip select

RD\ Read input

WR\ Write input

A_0A_1 Prot Address

PA7-PA0 Port A

PB7-PB0 Port B

PC7-PC0 Port C

8255A giao tiếp với Microprocrssor thông qua 3 bus : bus dữ liệu bit D_7-D_0 bus địa chỉ A_1A_0, bus điếu khiển RD\,WR\.SC\.Reset.

Mã lệnh, thông tin trạng thái và dữ liệu đều đượ truyền trên 8 đường dữ liệu D_7-D_0. Microprocrssor gởi dữ liệu đến 8255A hoặc Microprocrssor đọc dữ liệu từ 8255A tùy thuộc vào lệnh diều khiển. Các đường tín hiệu RD\,WR\ của 8255A được kết nối với các đường RD\, WR\ của Microprocrssor.

Tín hiệu Reset dùng để khởi động 8255A khi cấp điện, khi bị Reset các thanh ghi bên trong của 8255A đều bị xóa và 8255A ở trạng thái sẵn sàng làm việc. Khi giao tiếp với Microprocrssor, ngõ vào tín hiệu Reset này được kết nối tín hiệu Reset Out của Microprocrssor.

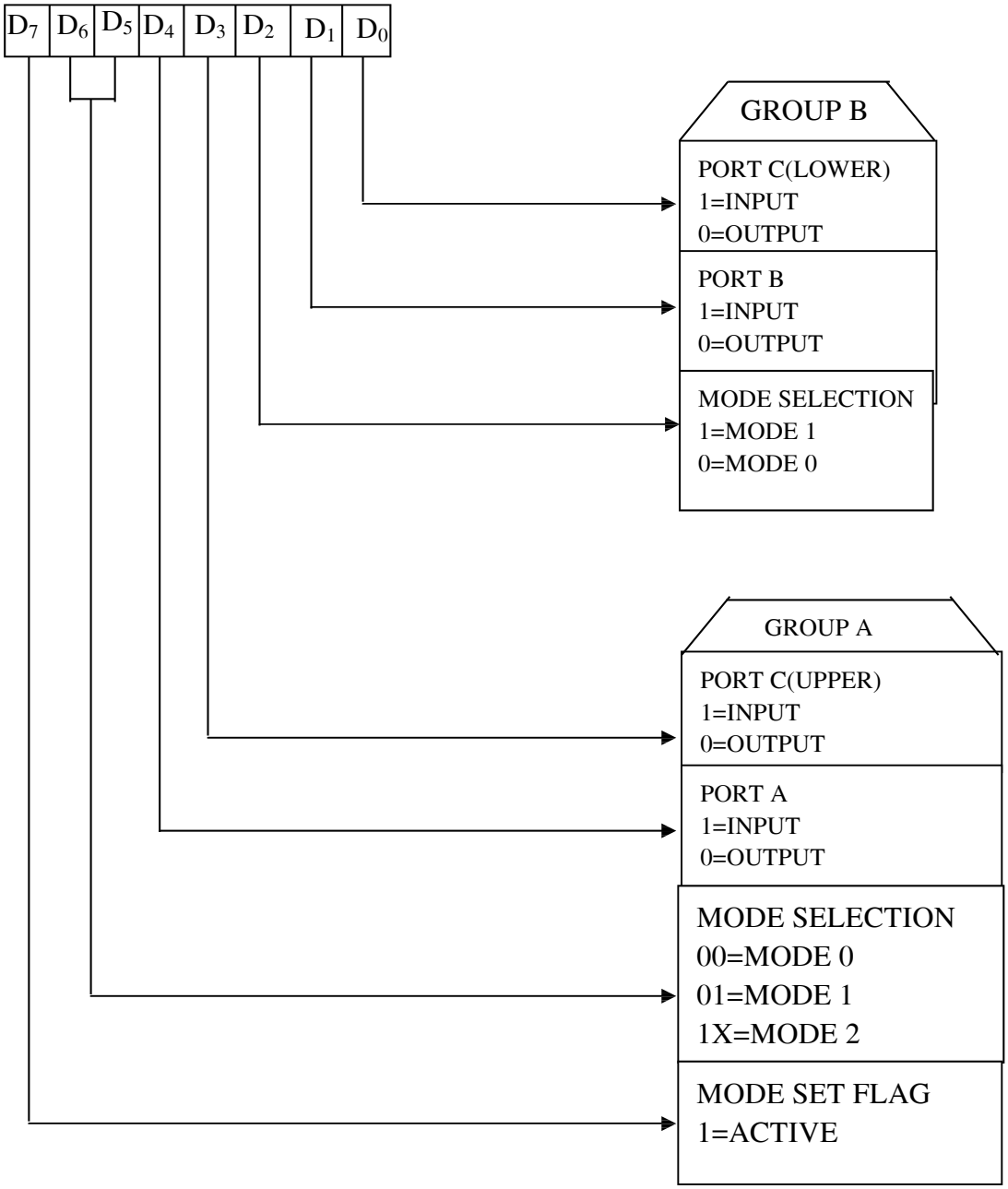
Tín hiệu Chip select CS\ dùng để lựa chọn 8255A khi Microprocrssor, giao tiếp với nhiều 8255A.

8255A có 3 Port xuất nhập (I/O) có tên là Port A,Port B, Port C, mỗi Port 8255A bit. Port A gồm PA_0-PA_7, Port B gồm PB_0-PB_7, Port C gồm các bit PC_0-PC_7. Các Port này có thể là các Port Input hay Output tùy thuộc vào lệnh điều khiển, lệnh điều khiển do Microprocrssor gởi đến chứa trong thanh ghi lệnh (còn gọi là thanh ghi điều khiển) để điều khiển 8255A .

các đường địa chỉ A_1A_0 của 8255A dùng để lựa chọn các Port và thanh ghi $A_1A_0=00_2$ dùng để chọn Port A, $A_1A_0=01_2$ dùng để chọn Port B, $A_1A_0=10_2$ dùng để chọn Port C, $A_1A_0=11_2$ dùng để chọn thanh ghi điều khiển.

Trong sơ đồ khối của 8255A , các Port I/O của 8255A chia ra làm 2 nhóm : nhóm A gồm Port A và bit cao của Port C,nhóm B gồm Port B và 4 bit thấp của Port C. Để sử dụng các Port của 8255A người lập trình phải gởi từ điều khiển ra thanh ghi điều khiển để 8255A định cấu hình cho các Port đúng theo yêu cầu mà người lập trình mong muốn.

Cấu trúc từ điều khiển của 8255A.



II. CẤU TRÚC PHẦN MỀM CỦA 8255.

Do các Port ra của 8255A được chia ra làm 2 nhóm A và nhóm B tách rời nên từ điều khiển của 8255A cũng được chia làm 2 nhóm.

Các bit $D_2D_1D_0$ dùng để định cấu hình cho nhóm B:

- Bit D_0 dùng để thiết lập 4 bit thấp của Port C, D_0=0 Port C xuất dữ liệu (output), D_0=1 – Port thấp là port nhập dữ liệu (Input).
- Bit D_1 dùng để thiết lập Port B , D_1=0- Port B là Port xuất dữ liệu (output), D_1=1 –Port B là Port nhập dữ liệu (input).
- Bit D_2 dùng để thiết lập Mode điều khiển của nhóm B:

§ D_2=0: nhóm B hoạt động ở modem 0.
§ D_2=1: nhóm B hoạt động ở modem 1.

Các bit $D_6D_5D_4D_3$ dùng để định cấu hình cho nhóm A:

- Bit D_3 dùng để thiết lập 4 bit cao của Port C, D_3=0-Port C là Port xuất dữ liệu (output),D_3=1 Port C là Port nhập dữ liệu (input).
- Bit D_4 dùng để thiết lập Port A, D_4=0- Port A là Port xuất dữ liệu (output), D_4=1-Port A là Port nhập dữ liệu (input).
- Bit D_6D_5 dùng để thiết lập Mode điều khiển của nhóm B:
  § D_6D_5=00:nhóm A hoạt động ở modem 0.
  § D_6D_5=01: nhóm A hoạt động ở modem 1.
  § D_6D_5=1x: nhóm A hoạt động ở modem 2.

III. GIAO TIẾP GIỮA VI XỬ LÍ VỚI 8255A .

- Vi mạch 8255A có thể giao tiếp với vi xử lý theo hai kiểu xuất nhập (I/O) và kiểu bộ nhớ.
- Khi vi xử lý giao tiếp với 8255A. Theo kiểu I/O thì nó chỉ dùng 8255A đường địa chỉ từ A_0 đến A_7, còn khi giao tiếp theo kiểu bộ nhớ thì nó dùng 16 đường A_0 đến A_{15} để giao tiếp, vì vậy dung lượng giao tiếp theo kiểu I/O thấp hơn dung lượng giao tiếp theo kiểu bộ nhớ.

1. Giao tiếp kiểu I/O.

Khi thiết kế vi xử lýgiao tiếp với 8255A theo kiểu I/O thì việc giao tiếp thông qua hai lệnh: In addr – Port và Out addr – Port. Dữ liệu giao tiếp luôn chứa trong thanh ghi A, địa chỉ port(addr port) có độ dài 8255A bit.

Cũng giống như bộ nhớ. Vi xử lý có thể giao tiếp với nhiều vi mạch 8255A. Với 8255A bit địa chỉ, nếu xem mỗi một địa chỉ truy xuất một ô nhớ thì vi xử lý có khả năng truy xuất 255 ô nhớ(với 256 địa chỉ). Mỗi vi mạch 8255A chiếm 4 địa chỉ 93 port và 1 thanh ghi điều khiển, nên số lượng vi mạch 8255A có thể giao tiếp với vi xử lý là 64.

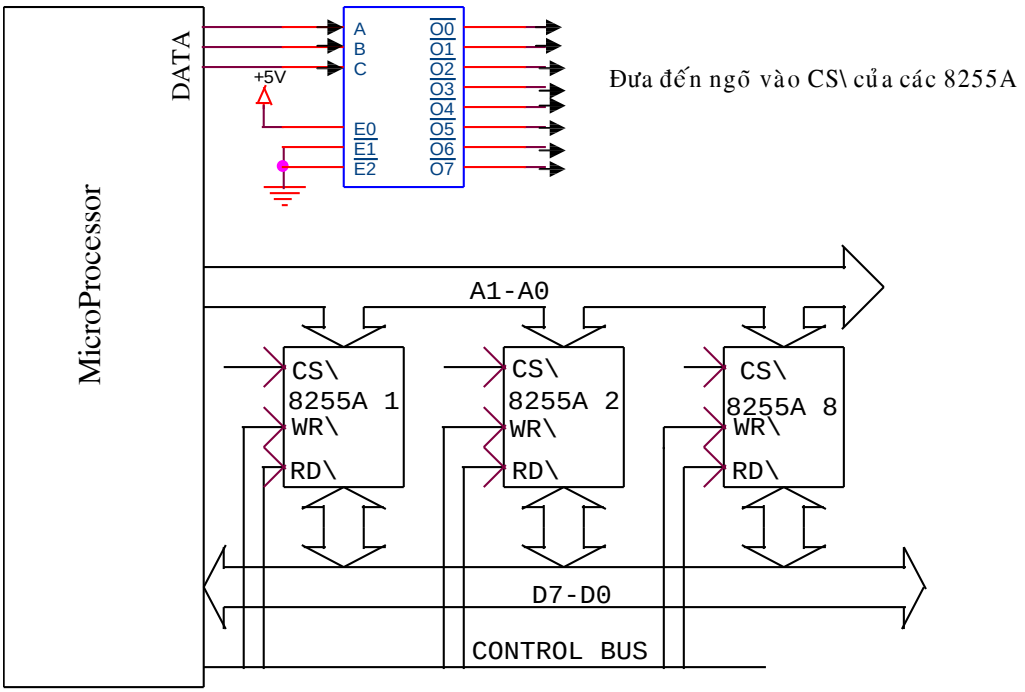
- khi kết nối giữa vi xử lý và vi mạch 8255A thì đường địa chỉ A_0 và A_1 dùng để lựa chọn các cổng và thanh ghi điều khiển, còn các đường A_2-A_7 dùng để lựa chọn vi mạch hoạt động, thông thường các đường địa chỉ này được đưa vào vi mạch giải mã rồi các ngõ ra của vi mạch giải mã sẽ đưa chân CS\ của các vi mạch 8255A.
- Ví dụ: thiết kế 2 vi mạch 8255 A giao tiếp với vi xử lý theo kiểu I/O. Ta có bảng địa chỉ các vi mạch 8255A.

IC	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	HEX
8255I	0 0	0 0	0 0	0 0	0 0	0 0	0 1	0 1	00 03
8255II	0 0	0 0	0 0	0 0	0 0	1 1	0 1	0 1	04 07

- 8255I chiếm 1 vùng địa chỉ từ 00_H đến 03_H địa chỉ của port A=00_H, port B=01_H ,port C=02_H và địa chỉ của thanh ghi điều khiển =03_H.
- 8255-I chiếm một vùng địa chỉ từ 04H đến 07H, địa chỉ của: port A=04H, port B=05H, port C=06H và địa chỉ của thanh ghi điều khiển=07H.

2. **Giao tiếp kiểu bộ nhớ.**

- Khi thiết kế giao tiếp 8255 với vi xử lý theo kiểu bộ nhớ; về chức năng của 8255 không có gì thay đổi chỉ thay đổi về địa chỉ truy xuất. Kiểu I/O, địa chỉ của port hay thanh ghi có độ dài 8255A bit, kiểu bộ nhớ, địa chỉ của port hay thanh ghi sẽ có độ dài 16 bit giống như bộ nhớ nên gọi là kiểu bộ nhớ.
- Khi thiết kế IO theo kiểu bộ nhớ thì mỗi port hay thanh ghi điều khiển của 8255, được xem là từng ô nhớ. Khi đó vi xử lý giao tiếp với 8255 giống như bộ nhớ và 2 lệnh IN và OUT không còn tác dụng.
- Kiểu bộ nhớ chỉ sử dụng trong các hệ thống nhỏ đơn giản.



Hình 3.2: Giao tiếp IC8255A với Microprocessor.

3. **Ứng dụng của 8255:**

IC giao tiếp IO 825 có rất nhiều ứng dụng trong các hệ thống điều khiển dùng MicroProcessor, 8255 đóng vai trò là IC giao tiếp giữa MicroProcessor và đối tượng điều khiển.

Các ứng dụng của 8255 là truyền dữ liệu, giải mã hiển thị, giải mã bàn phím, giao tiếp điều khiển tùy theo yêu cầu.

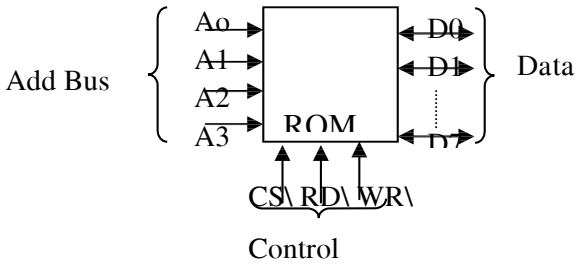
Chương 3 : KHẢO SÁT BỘ NHỚ BÁN DẪN

Vi điều khiển (Microcontroller) là IC chuyên về xử lý dữ liệu điều khiển theo một chương trình, muốn vi điều khiển thực hiện một công việc gì thì người sử dụng phải lập trình. Chương trình phải được lưu trữ ở một bộ phận nào đó, để vi điều khiển nhận lệnh và thi hành, đôi khi trong lúc xử lý,chương trình của vi điều khiển cần nơi để lưu trữ tạm thời dữ liệu chính của bộ nhớ. Các bộ nhớ của vi điều khiển là các IC, các IC nhớ này có thể đọc dữ liệu ra, ghi dữ liệu vào hoặc chỉ đọc dữ liệu ra. Đôi khi bộ nhớ của vi đều khiển không đủ để lưu trữ những thông tin cần thiết khi chạy chương trình, khi đo phải dùng kỹ thuật mỡ rộng bộ nhớ.

I BỘ NHỚ CHỈ ĐỌC(ROM:Read Only Memory)

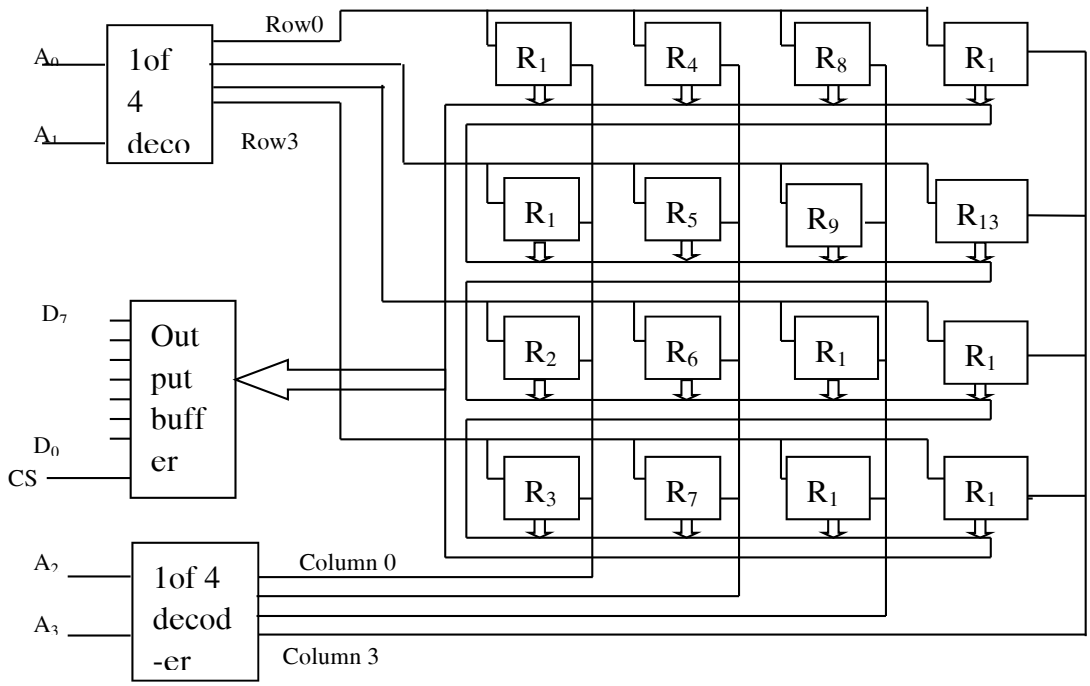
Loại bộ nhớ này được thiết kế để lưu trữ các dữ liệu cố định. Trong lúc hoạt động bình thường dữ liệu mới không thể nào ghi được vào ROM, mã dữ liệu chỉ đọn ra từ ROM. ROM dùng để lưu trữ các chương trình của máy tính do không bị mất dữ liệu khi mất điện

Sơ đồ ROM có dung lượng 32 x4bit



ROM có 3 bus:bus dữ liệu,bus địa chỉ,bus điều khiển.Với bộ nhớ ROM ở trên bus địa chỉ có 4 đường nên có dung lượng bộ nhớ là 2^4=16.Bus dữ liệu có 8 đường,từ dữ liệu là 8bit hay 1byte,vậy bộ nhớ ROM này có dung lượng là 16byte.Bus điều khiển cho phép ROM hoạt động đọc hay viết,để đọc dữ liệu của ô nhớ nào phải cung cấp địa chỉ của ô nhớ đó tới các ngõ vào địa chỉ tác động đến ngõ vào cho phép CS\.

1. Cấu trúc bên trong của ROM



Cấu trúc của ROM rất phức tạp,từ sơ đồ trên thì cấu trúc của ROM gồm có 4 phần chính

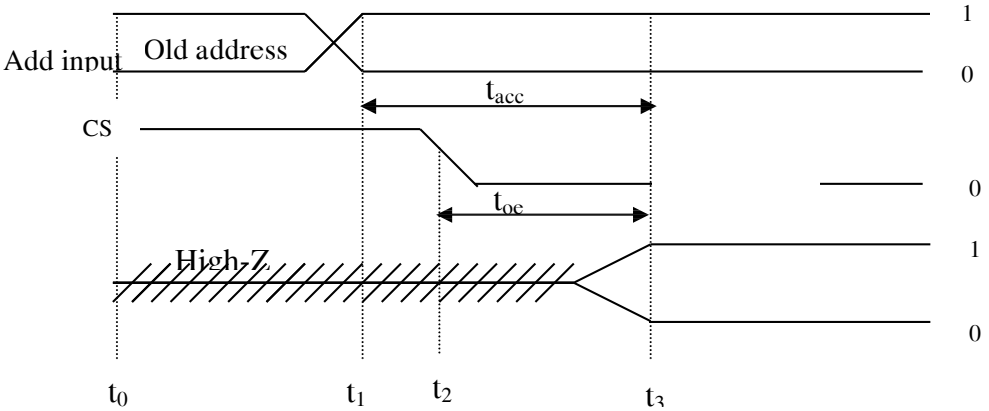
+ Giải mã hàng
+ Giải mã cột
+ Ma trận thanh ghi
+Đệm ngõ ra

- Ma trận thanh ghi: Lưu trữ dữ liệu đã được lập trình từ ROM,mỗi thanh ghi chứa một từ dữ liệu,như trong trường hợp trên mỗi thanh ghi lưu trữ bốn từ dữ liệu bit. Ngõ ra của từ dữ liệu 8 bit được kết nối với bit dữ liệu bên trong. Mỗi thanh ghi có hai ngõ vào cho phép.Thanh ghi nào có hai ngõ vào cho phép ở mức cao thì dữ liệu sẽ gởi là bus dữ liệu.
- Giải mã địa chỉ: mã địa chỉ $A_3A_2A_1A_0$ dùng để xác định thanh ghi nào trong ma trận được phép đặt từ dữ liệu 8bit lên bus dữ liệu .Hai bit địa chỉ A0A1 được đưa đến bộ giải mã hai đường sang bốn đường để lựa chọn một trong bốn dòng,hai bit địa chỉ A_2A_3 được đưa đến bộ giải mã thứ hai để chọn một trong bốn cột. Chỉ duy nhất một thanh ghi ở trong một hàng và một cột được chọn bởi một địa chỉ ở ngõ vào,và thanh ghi nầy được phép gửi dữ liệu lên bus.
- Đệm ngõ ra: dữ liệu do thanh ghi gửi ra sẽ được đưa vào bộ đệm,bộ đệm sẽ gửi dữ liệu ra các đường dữ liệu bên ngoài,khi tín hiệu điều khiển CS ở mức cao. Nếu CS ở mức thấp thì bộ đệm ngõ ra ở trạng thái tổng trở cao và các đường dữ liệu $D_0 – D_7$ sẽ được thã nổi

2. Thời hằng truy xuất bộ nhớ ROM

Có một khoảng thời gian từ lúc áp đặt địa chỉ tới các ngõ vào địa chỉ của ROM đến lúc dữ liệu xuất hiện ở ngõ ra(trong lúc ROM hoạt động) thời gian nầy gọi là thời gian trễ hay thời gian truy xuất.Khoảng thời gian từ lúc ngõ vào cho phép CS\ đến lúc dữ liệu xuất hiện gọi là thời gian cho phép xuất dữ liệu.

Giản đồ thời hằng truy xuất của Rom



3. Các loại bộ nhớ ROM

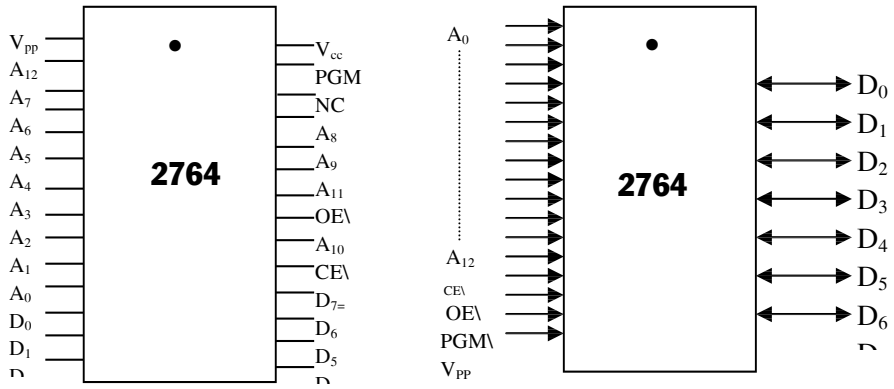
Maskable Programmed ROM(ROMmặt nạ): đây là loại ROM do nhà sản xuất nạp sẳn chương trình,khi đã nạp chương trình thì các bit tring ROM này không được thay đổi nữa.

Programmable ROM(PROM): loại ROM này người sử dụng có thể nạp chương trình và chỉ nạp một lần không thể xóa được.

ErasableProrammable ROM(EPROM): loại ROM này có thể lập trình bởi người sử dụng và có thể xóa nạp nhiều lần .Để xóa dữ liệu trong EPROM phải dùng ánh sáng cực tím để xóa,để lập trình cho ROM phải dùng mạch nạp EPROM. EPROM có hai điểm bất lợi: phải lấy EPROM ra khổi soket để xóa và lập trình lại khi muốn thay đổi chương trình .Khi muốn thay đổi dữ liệu ô nhớ thì phải xóa dữ liệu của ô nhớ đó,nhưng khi dùng ánh sáng cực tím thì tất cả dữ liệu trong EPROM bị xóa sạch và phải nạp lại toàn bộ dữ liệu.

4. Khảo sát bộ nhớ EPROM 2764

Trong các mạch điều khiển dùng vi xử lý PROM được sử dụng rất phổ biến vì nó cho phép người sử dụng có thể nạp và xóa các chương trình dễ dàng theo yêu cầu của mỗi người. EPROM 2764 có dung lượng 8kbyte có sơ đồ chân và sơ đồ logic như sau:



Hình 4.2 Sơ đồ chân và sơ đồ logic EPROM 2764

– EPROM 2764 có 13 đường địa chỉ và 8 đường dữ liệu nên dung lượng của 2764 là 2^{13}=8192byte dữ lệu hay 8kbyte ,có 2 nguồn cung cấpVcc và Vpp ngõ vào Vcc luôn nối tới nguồn 5v ngõ vào Vpp được nối tới nguồn+5v khi EPROM đang làm việc ở chế độ đọc dữ liệu và nối tới nguồn 26v khi lập trình cho EPROM

Hai ngõ vào điều khiển:

OE\ được dùng để điều khiển bộ đệm cho phép dữ liệu của EPROM xuất ra ngoài hay không .

CE\ là ngõ vào cho phép có hai chức năng :khi hoạt động bình thườngCE\ là it1n hiệu cho phép để dọc dữ liệu từ EPROM,CE\ phải ở mức thấp để mạch điện bên tronglựa chọn dữ liệu và chuyển nó đến output buffer kết hợp với tín hiệu cho OE\ ở mức thấp,thì dữ liệu mới xuất ở các ngõ raD0-D7.Khi CE\ ở mức cao thì EPROM ở trạng thái chờ(Standby).công suất tiêu tán lúc này 132mw.

Bảng trạng thái làm việc của EPROM

MODE	CE\	OE\	PGM\	Vpp	Vcc	Output
READ	V_{il}	V_{il}	V_{ih}	Vcc	Vcc	D_{out}
STANDBY	V_{ih}	X	X	Vcc	Vcc	HighZ
PROGAM	V_{il}	X	V_{il}	Vpp	Vcc	D_{in}

PROGRAM VERYFY	V_{il}	V_{il}	V_{ih}	V_{pp}	V_{cc}	D_{out}
PROGRAM INHIBIT	V_{ih}	X	X	V_{pp}	V_{cc}	HighZ

II.BỘ NHỚ RAM

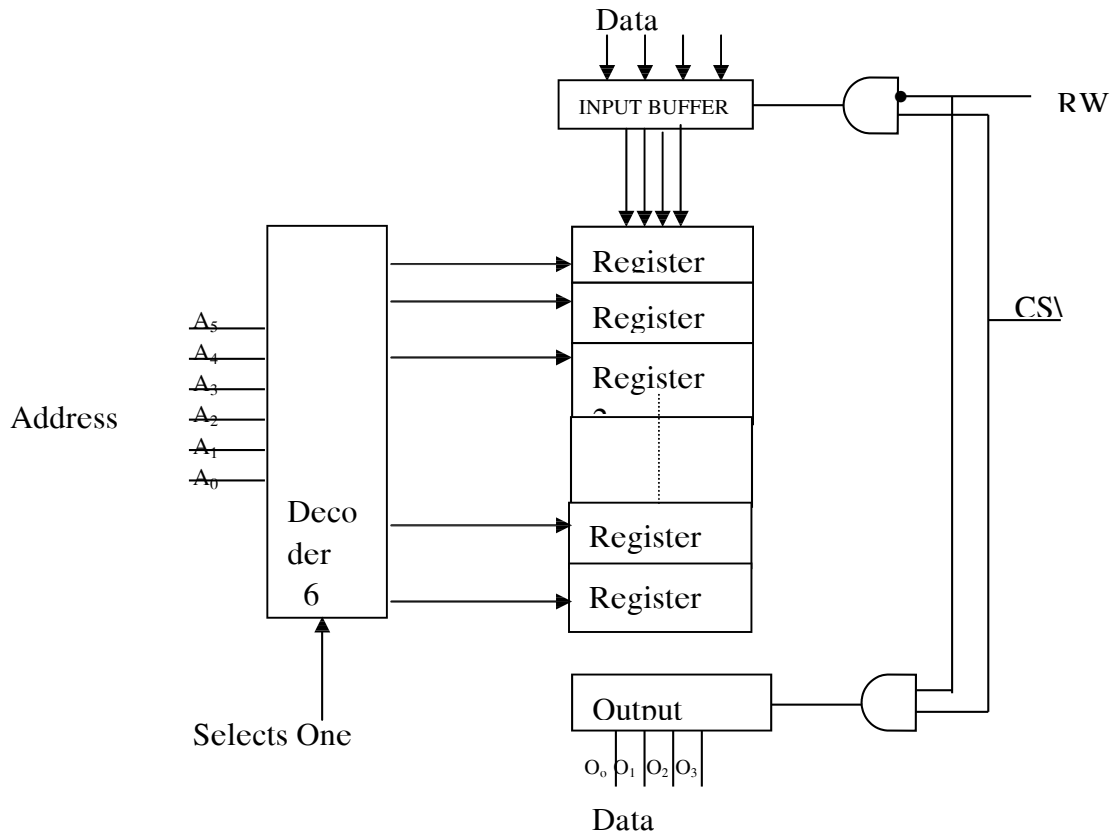
-Ram là bộ nhớ truy xuất ngẩu nhiên, có nghĩa là bất kì ô nhớ nào cũng dễ dàng truy xuất như những ô nhớ khác.

-Khuyết điểm của Ram là` dữ liệu lưu trữ trong Ram sẽ mất khi mất điện.

-Ưu điểm chính của Ram là có thể đọc và ghi nhanh chóng

1.Cấu Trúc Của Ram

Tương tự như bộ nhó Rom,bộ nhớ Ram cũng gồm có một số thanh ghi .mổi thanh ghi lưu trữ 1 từ dữ liệu duy nhất và một dữ liệu duy nhất.Dung lượng của bô nhớ Ram là 1K,2K ,8K, 16K ,32K, 64K, 128K, 256K, 512K, và 1024K.và từ 72 dữ liệu là 8 hoặc 4 bit.



Hình 4.3 Sơ đồ cấu trúc bên trong Ram 64x4

a.Hoạt động đọc dữ liệu từ Ram

Mã địa chỉ của ô nhớ cần đọc dữ liệu được đưa đến ngõ vào địa chỉ cuả Ram đồng thời ngõ tín hiệu điều khiển R/W phải ở mức logic 1 và ngõ vào cho phép(CS) phải ở mức logic1.khi đó dữ liệu mới xuất hiện ở ngõ ra dữ liệu.

Khi R/W=1 sẽ không cho phép bộ đệm ngõ vào, do đó dữ liệu ngõ vào không ảnh hưởng gì đến ô nhớ đang truy xuất.

b. Hoạt động ghi dữ liệu lên Ram

Để ghi dữ liệu vào thanh ghi đã được lựa chọn bởi các ngõ vào địa chỉ của bộ nhớ Ram,đòi hỏi ngõ vàoR/W=0 và CS=1.Tổ hợp hai mức logic này sẽ cho phép bộ đệm ngõ vào để đưa từ dữ liệu (4bit) ở các ngõ vào sẽ được nạp thanh ghi được chọn

KhiR/W ở mức thấp sẽ không cho phép bộ đệm ngõ ra và ngõ ra ở trạng thái tổng trở cao(trong lúc ghi dữ liệu).Khi ghi dữ liệu vào ô nhớ thì dữ liệu trước đó sẽ mất đi .

c. Chip selet (cs)

Hầu hết các bộ nhớ đều có hoặt nhiều ngõ vào CS ,được dùng để cho phép hoaặ không cho phép bộ nhớ hoạt động trong nhiều trường hợp kết nối nhiều bộ nhớ.Khi không cho tất cả các ngõ váo dữ liệu và ngõ ra dữ liệu ở trạng thái tổng trở cao.

d. Những chân data input-output

Để giảm số chân cho một Icnhà chế tạo kết hợp 2 chức năng data input và data output thành một chân Input/output, chúng có chức năng của các chân I/O.Khi hoạt động đọc,cá chân I/O hoạt động như lá các chân xuất dữ liệu.Khi ghi dữ liệu, các chân I/o hoạt động như là các chân dữ liệu.

2 . Các loại Ram

Ram được chia làm 2 loại:

-SRAM(Static RAM);là một loại linh kiện mà việc lưu trữ dữ liệu dựa vào nguyên tắc hoạt động của flip flop D.Dữ liệu vào tồn tại ở một trong haitrạng thái logic của mạch số.

DRAM(Dynamic Ram):là loại linh kiện nhớ mà dữ liệu lưu trữ như điện tích trữ trong tụ điện.

Chương 4 : ĐO NHIỆT ĐỘ

I. Hệ Thống Đo Lường

1. Giới thiệu

Để thực hiện phép đo của một đại lượng nào đó thì tuỳ thuộc vào đặc tính của đại lượng cần đo,điều kiện đo,cũng như độ chính xác theo yêu cầu của một phép đo mà ta có thể thực hiện đo bằng nhiều cách khác nhau trên cơ sỡ của các hệ thống đo lường khác nhau.

Sơ đồ khối của một hệ thống đo lường tổng quát

_ Khối chuyển đổi: làm nhiệm vụ nhận trực tiếp các đại lượng vật lý đặc trưng cho đối tượng cần đo biến đổi các đại lượng thành các đại lượng vật lý thống nhất(dòng điện hay điện áp) để thuận lợi cho việc tính toán.

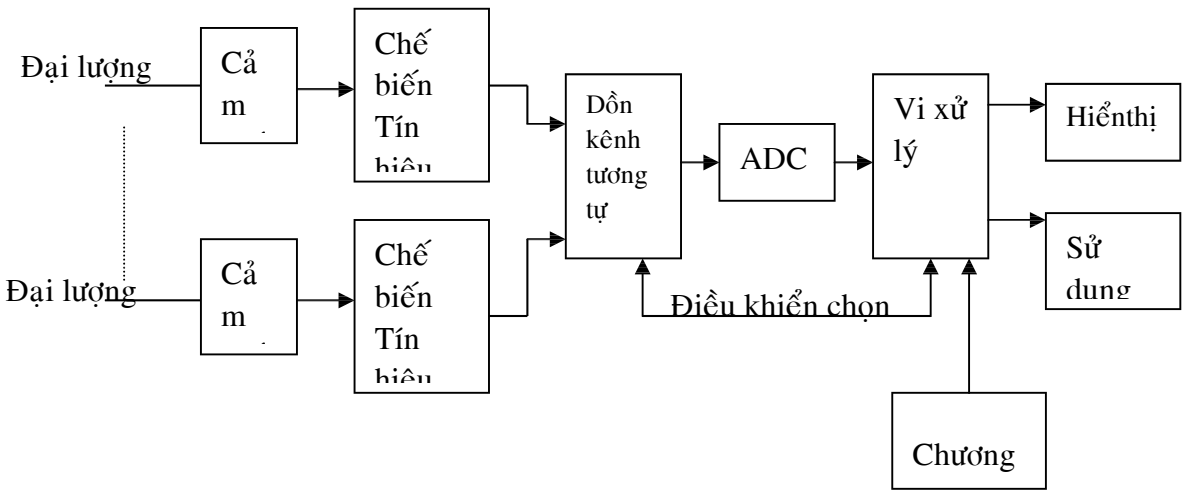
_ Mạch đo: có nhiệm vụ tính toán biến đổi tín hiệu nhận được từ bộ chuyển đổi sao cho phù hợp với yêu cầu thể hiện kết quả đo của bộ chỉ thị.

_ Khối chỉ thị:làm nhiệm vụ biến đổi tín hiệu điện nhận được từ mạch đo để thể hiện kết quả đo.

2. Hệ thống đo lường số

Hệ thống đo lường số được nhóm áp dụng để thực hiện luận văn nầy vì có các ưu điểm:các tín hiệu tương tự qua biến đổi thành các tín hiệu số có các xung rỏ ràng ở trạng thái 0,1 sẽ giới hạn được nhiều mức tín hiệu gây sai số .Mặt khác ,hệ thống này tương thích với dữ liệu của máy tính,qua giao tiếp với máy tính ứng dụng rộng rãi trong kỹ thuật.

a. Sơ đồ khối



Hình 5.1 Sơ đồ khối của hệ thống đo lường số

b. Nguyên lý hoạt động

Đối tượng cần đo là đại lượng vật lý,dựa vào các đặc tính của đối tượng cần đo mà ta chọn một loại cảm biến phù hợp để biến đổi thông số đại lượng vật lý cần đo thành đại lượng điện ,đưa vào mạch chế biến tín hiệu(gồm:bộ cảm biến,hệ thống khuếch đại,xử lý tín hiệu).

Bộ chuyển đổi tín hiệu sang số ADC(Analog Digital Converter) làm nhiệm vụ chuyển đổi tín hiệu tương tự sang tín hiệu số và kết nối với vi xử lý.

Bộ vi xử lý có nhiệm vụ thực hiện những phép tính và xuất ra những lệnh trên cơ sở trình tự những lệnh chấp hành đã thực hiện trước đó.

Bộ dồn kênh tương tự (multiplexers) và bộ chuyển ADC được dùng chung tất cả các kênh . Dữ liệu nhập vào vi xử lý sẽ có tín hiệu chọn đúng kênh cần xử lý đê đưa vào bộ chuyển đổi ADC và đọc đúng giá trị đặc trưng của nó qua tính toán để có kết quả của đại lượng cần đo.

II. <u>Các Phương Pháp Đo Nhiệt Độ</u>

Đo nhiệt độ là một phương thức đo lường không điện,đo nhiệt độ được chia thành nhiều dãi:

+ Đo nhiệt độ thấp
+ Đo nhiệt độ trung bình
+ Đo nhiệt độ cao.

Việc đo nhiệt độ được tiến hành nhờ các dụng cụ hổ trợ chuyên biệt như:

+ Cặp nhiệt điện
+ Nhiệt kế điện kế kim loại
+ Nhiệt điện trở kim loại
+ Nhiệt điện trở bán dẫn
+ Cảm biến thạch anh.

Việc sử dụng các IC cảm biến nhiệt để đo nhiệt độ là một phương pháp thông dụng được nhóm sử dụng trong tập luận văn nầy,nên ở đây chỉ giới thiệu về IC cảm biến nhiệt.

- Nguyên lý hoạt động chung của IC đo nhiệt độ

IC đo nhiệt độ là một mạch tích hợp nhận tín hiệu nhiệt độ chuyển thành tín hiệu điện dưới dạng dòng điện hay điện áp.Dựa vào đặc tính rất nhạy của các bán dẫn với nhiệt độ,tạo ra điện áp hoặc dòng điện,tỉ lệ thuận với nhiệt độ tuyệt đối.Đo tín hiệu điện ta biết được giá trị của nhiệt độ cần đo.Sự tác động của nhiệt độ tạo ra điện tích tự do và các lổ trống trong chất bán dẫn . Bằng sự phá vỡ các phân tư , bứt các electron thành dạng tự do di chuyển qua vùng cấu trúc mạng tinh thể tạo sự xuất hiện các lỗ trống . Làm cho tỉ lệ điện tử tự do và lổ trống tăng lên theo qui luật hàm mũ với nhiệt độ .

- Đặc tính của một số IC đo nhiệt độ thông dụng

+AD590

Ngõ ra là dòng điện.

Độ nhạy $1A/^0K$.

Độ chính xác $+4^0C$.

Nguồn cung cấp $V_{cc} = 4 - 30V$.

Phạm vi sử dụng -55^oc đến 150^oc

+ LX5700

Ngõ ra là điện áp.

Độ nhạy $-10mv/^0K$.

Phạm vi sử dụng $-55^0C - 150^0C$.

+ LM135,LM335

Ngõ ra là điện áp.

Độ nhạy $10mv/^0C$.

Sai số cực đại $1,5^0C$ khi nhiệt độ lớn hơn 100^0C.

Phạm vi sử dụng $-55^0C - 150^0C$.

Chương 5 : CHUYỂN ĐỔI TƯƠNG TỰ – SỐ

I. KHÁI NIỆM CHUNG

Ngày nay việc truyền đạt tín hiệy cũng như quá trình điều khiển và chỉ thị phần lớn được thực hiện theo phương pháp số. Trong khi đó tín hiệu tự nhiên có dạng tương tự như:nhiệt độ,áp suất ,cường độ ánh sáng,tốc độ quay,tín hiệu âm thanh...Để kết nối giữa nguồn tín hiệu tượng tự với các hệ thống xử lý số người ta dùng các mạch chuyển đổi tương tự sang số(ADC) nhằm biến đổi tín hiệu tương tự sang số hoặc trong trường hợp ngược lại cần biến đổi tín hiệu số sang tương tự thi dùng các mạch DAC (Digital Analog Converter).

II. NGUYÊN TẮT THỰC HIỆN CHUYỂN ĐỔI ADC

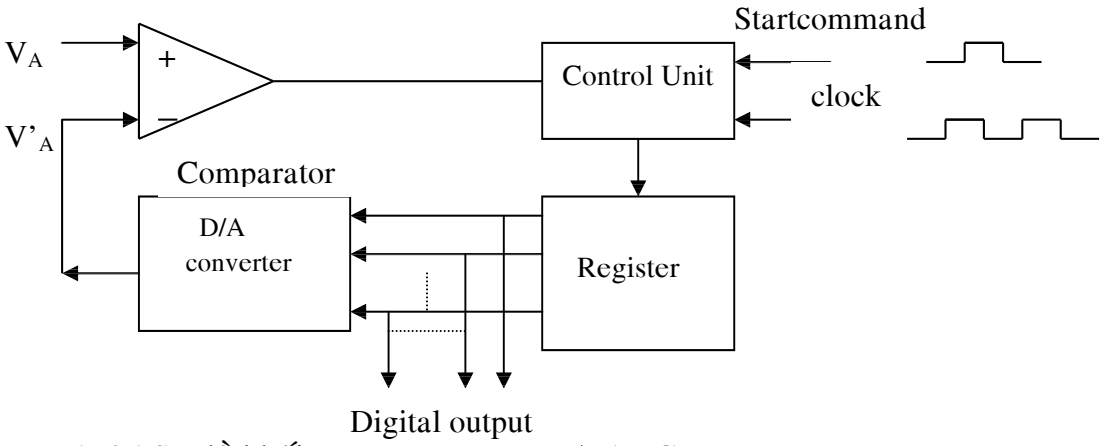
Mạch chuyển đổi tin hiệu tương tự sang số,chuyển một tín hiệu ngõ vào tương tự (dòng điện hay điện áp) thành dạng mã số nhị phân có giá trị tương ứng.

Chuyển đổi ADC có rất nhiều phương pháp.Tuy nhiên,mỗi phương pháp điều có những thông số cơ bản khác nhau:

+Độ chính xác của chuyển đổi AD.

+ Tốc độ chuyển đổi .

+ Dãi biến đổi của tín hiệu tương tự ngõ vào



Hình 6.1 Sơ đồ khối tổng quát của mạch ADC

- Hoạt động

-Đầu tiên kích xung start để bộ ADC hoạt động

-Tại một tần số được xác định bằng xung clock bộ điều khiển làm thay đổi thành số nhị phân được lưu trữ trong thanh ghi(Register).-Số nhị phân trong thanh ghi được chuyển thành dạng điện áp V'_a bằng bộ chuyển đổi DA.

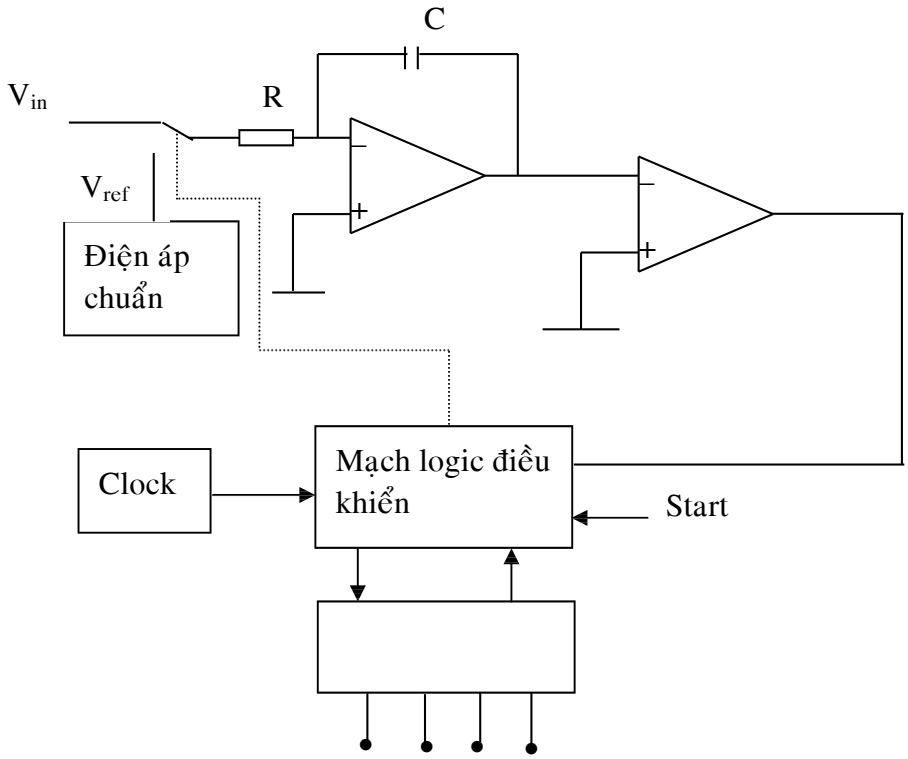
-Bộ so sánh,so sánh V'_a với điện áp ngõ vào V_a .Nếu $V'_a < V_a$ thì ngõ ra của bộ so sánh vẫn giữ mức cao. Khi $V'_a > V_a$ ngõ ra của bộ so sánh xuống

mức thấp và quá trình thay đổi số của thanh ghi ngưng. Lúc này V'_a gần bằng V_a , những số trong thanh ghi là những số cần chuyển đổi .

III. CÁC PHƯƠNG PHÁP CHUYỂN ĐỔI AD

1. Phương pháp tích phân (Intergration method)

Phương pháp tích phân cũng giống như phương pháp chuyển đổi ADC dùng tín hiệu dốc đôi (Dual-Slope-ADC). Cấu trúc mạch điện đơn giản hơn nhưng tốc độ chuyển đổi chậm.



Hình 6.2 : Sơ đồ nguyên lý cơ bản của mạch chuyển đổi AD dùng phương pháp tích phân

* Hoạt động

-Khi có xung start mạch đếm đưa về trạng thái reset. Mạch logic điều khiển khóa K ở vị tri 1, điện áp tương tự V_{in} được nạp vào tụ điện C với thời hằng t_1 tín hiệu ngõ ra của mạch tích phân giảm dần,và cho đến khi nhỏ hơn 0V thì ngõ ra của bộ so sánh lên mức 1,do đó mạch logic điều khiển mở cổng cho xung clock vào mạch đếm. Sau khoảng thời gian t_1 mạch đếm tràn mạch logic điều khiển khóa K ở vị trí 0,khi đó điện áp âm V_{ref} được đưa vào ngõ vào của mạch tích phân,tụ điện C xả điện với tốc độ không đổi, sau khoảng thời gian t_2 tín hiệu ngõ ra của mạch tích phân tăng dần,do đó ngõ ra của mạch so sánh xuống ,mức thấp làm cho mạch logic điều khiển đống cổng và báo kết thúc chuyển đổi. Trong suốt khoảng thời gian xả điện t_2 mạch đếm

vẫn tiếp tục đếm kết quả của mạch đếm cũng chính là tín hiệu số cần chuyển đổi tương ứng với điện áp tương tự ngõ vào V_{in} .

Mối quan hệ giữa điện áp ngõ vào V_{in} và điện áp chuẩn V_{ref} với t_1,t_2

$$t_2=t_1.v_{in}/v_{ref}$$

$t_1=2^n/f_{ck}$:thời gian mạch đếm từ 0 đến khi tràn

$t_2=N/f_{ck}$: thời gian mạch đếm từ khi tràn đến kết quả sau cùng

-Biểu thức nầy không phụ thuộc vào thời hằng RC,cũng như số xung clock(nếu mạch làm việc ổn định).

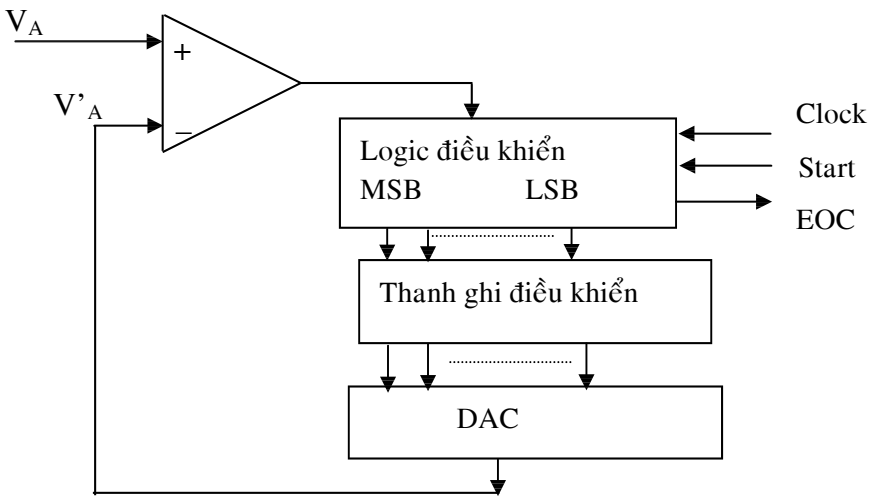
-Các tín hiệu tương tự V_{in} qua mạch tích phân nên các tín hiệu nhiểu đều bị loại bỏ.

-Nhược điểm của mạch nầy là thời gian chuyển đổi chậm,giữa 2^n chu kỳ xung clock trong lần lấy tích phân trong thời gian t_1 va` N chu kỳ trong lần lấy tích phân trong thời gian t_2. Thời gian chuyển đổi lớn nhất khi $t_1=t_2$.

Thời gian chuyển đổi: $T = t_1+t_2$

2. Phương pháp ADC xấp xỉ liên tiếp(Successive- Approximation ADC)

Đây là một trong những phương pháp d9uo75c sử dụng rộng rãi. Tuy nhiên,mạch điện có phức tạp nhưng thời gian chuyển đổi ngắn hơn. Phương pháp chuyển đổi ADC xấp xỉ liên tiếp có thời gian chuyển đổi cố định không phụ thuộc vào điện áp ngõ vào.



Hình 6.3 : Sơ đồ khối chuyển đổi ADC dùng phương pháp xấp xỉ liên tiếp.

* Hoạt động

Khi tác động cạnh xuống của xung start thì ADC bắt đầu chuyển đổi .

-Mạch logic điều khiển đặt bit có nghĩa lớn nhất(Most Signifi cant Bit)của thanh ghi điều khiển lên mức cao và tất cả các bit còn lại ở mức thấp.Số nhị phân ra ở mạch thanh ghi điều khiển đượ cqua mạch DAC để tạo ra điện áp tham chiếu V'_a.

Nếu $V'_a > V_a$ thì ngõ ra bộ so sánh xuống mức thấp ,làm cho mạch logic điều khiển xóa bit MSB xuống mức thấp.

Nếu $V'_a < V_a$ thì ngõ ra của bộ so sánh vẫn ở mức cao và làm cho mạch logic điều khiển giữ bit MSB ở mức cao.

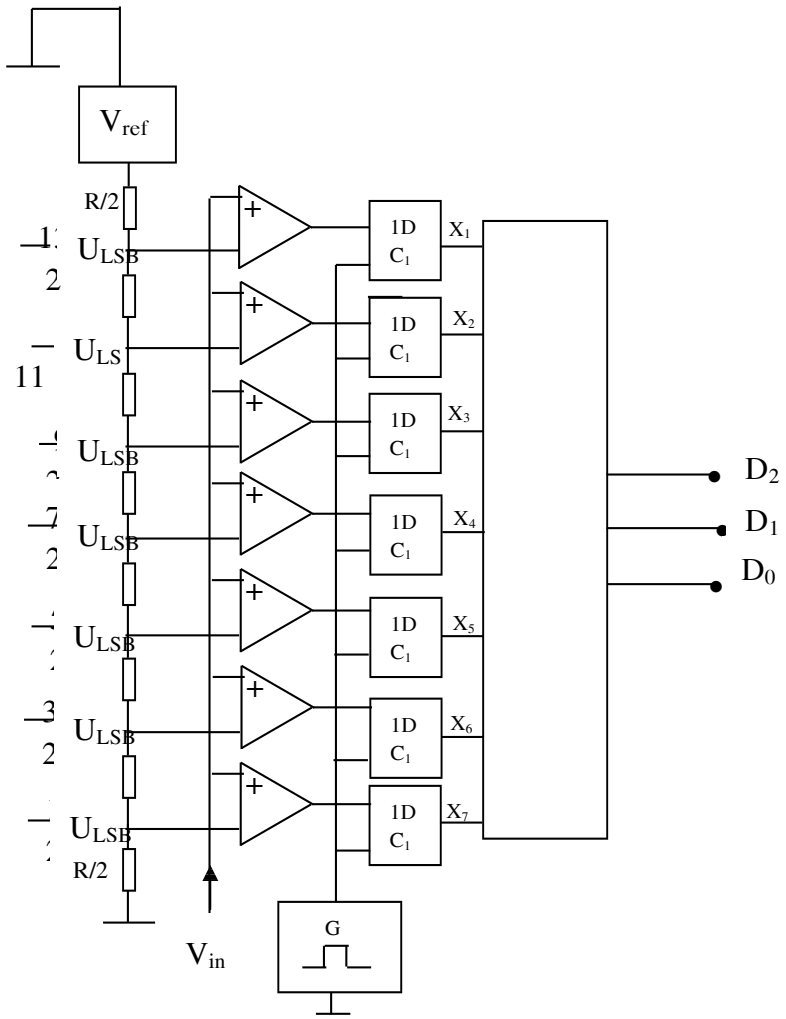
Tiếp theo mạch logic điều khiển đưa bit có nghĩa kế bit MSB lên mức cao và tạo ở ngõ ra khối DAC một điện áp tham chiếu v'_a rồi đem so sánh tương tự như bit MSB ở trên .Quá trình này cứ tiếp tục cho đến bit cuối cùng trong thanh ghi điều khiển. Lúc đó v'_a gần bằng V_a ngõ ra của mạch logic điều khiển báo kết thúc chuyển đổi.

Như vậy mạch đổi ra n bit chỉ mất n chu kỳ xung clock nên có thể đạt tốc độ rất cao. Tuy nhiên mạch ADC xấp xỉ liên tiếp lại không thể đáp ứng với tín hiệu tương tự vào biến đổi cực nhanh .

3. Phương pháp song song (paralled method)

Mạch ADC dùng nguyên tắc chuyển đổi song song hay còn gọi là phương pháp ADC nhanh, có cấu trúc mạch điện phức tạp nhưng tốc độ chuyển đổi rất cao .

Trong vài trường hợp người ta cần mạch chuyển đổi ADC có tốc độ rất cao vì những tín hiễu biến đổi nhanh nên khi chuyển sang dạng số người ta cầ mạch ADC có tốc độ cao .



Hình 6.4 Sơ đồ khối mạch chuyển đổi AD dùng phương pháp song song

* Hoạt đông

Mạch bao gồm: khối so sánh song song và mạch mã hoá. Tín hiệu tương tự được vào các mạch so sánh cùng một lúc, các trạng thái ra của mạch so sánh được đưa vào các flip flop D để đưa đến bộ mã hóa,đầu ra của mạch mã hóa chính là đầu ra của mạch ADC.

Mạch so sánh và mạch mã hóa là loại mạch có tốc độ xử lý rất cao nên tổng thời gian trễ chỉ vài chục ns,nhờ vậy sự chuyển đổi xẩy ra rất nhanh. Tuy nhiên với mạch ADC nhanh ở 3 bit thì nó đồi hỏi bảy bộ so sánh khi ở 6 bit thì cần đến 63 bộ so sánh đó là nhược điểm của mạch ADC dùng phương pháp so sánh .

Bảng sự thật của mạch chuyển đổi

Điện áp vào	Ngõ ra bộ so sánh							Tín hiệu số ngõ ra		
V_{in}/V_{LSB}	K_7	K_6	K_5	K_4	K_3	K_2	K_1	D_1	D_2	D_3
0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	1	0	0	1
2	0	0	0	0	0	1	1	0	1	0
3 3	0	0	0	0	1	1	1	0	1	1
4	0	0	0	1	1	1	1	1	0	0
5	0	0	1	1	1	1	1	1	0	1
6	0	1	1	1	1	1	1	1	1	0
7	1	1	1	1	1	1	1	1	1	1

PHỤ LỤC : CÁC VÍ DỤ CƠ BẢN.

1. INSTUCTIONSET

• Add

```
ORG     0H
        MOV     R5,#25H ; na.p 25H va`o R5
        MOV     R7,#34H ; na.p 34H va`o R7
        MOV     A,#0    ; na.p 0 va`o A
        ADD     A,R5    ; co^.ng R5 vo+'i A
                        ; A = A + R5
        ADD     A,R7    ; co^.ng R7 vo+'i A
                        ; A = A + R7
        ADD     A,#12H  ; add 12H va`o A
                        ; A = A + 12H
HERE:   SJMP    HERE    ; du+`ng chuo+ng tri`nh ta.i dda^y
        END
```

• BCD2ASCII

```
ORG     0
        MOV     A,#29H    ; A = 29H, packed BCD
        MOV     R2,A      ; sao lu+u A va`o R2
        ANL     A,#0FH    ; che nibble cao (A=09)
        ORL     A,#30H    ; chuye^?n tha`nh ma~ ASCII, A=39H (`9')
        MOV     R6,A      ; lu+u ke^'t qua? va`o R6 (R6=39H ASCII char)
        MOV     A,R2      ; la^'y la.i gia' tri. A ban dda^`u
        ANL     A,#0F0H   ; che nibble tha^'p (A=20)
        RR      A         ; quay pha?i 4 la^`n
        RR      A         ;
        RR      A         ;
        RR      A         ; -> A=02
        ORL     A,#30H    ; chuye^? tha`nh ma~ ASCII
        MOV     R2,A      ; lu+u va`o R2
        SJMP    $
```

• Bin2BCD

```
; ddo^?i Binary (P1) -> BCD (R5 R6 R7)
        MOV     A,#0FFH
        MOV     P1,A    ; P1: input port
        MOV     A,P1    ; ddo.c P1
        MOV     B,#10   ; B=0A hex (10 dec)
        DIV     AB      ; chia cho 10
```

```
        MOV     R7,B    ; lu+u digit tha^'p
    MOV     B,#10   ;
    DIV     AB      ; chia cho 10
    MOV     R6,B    ; lu+u digit tie^'p theo va`o R6
    MOV     R5,A    ; lu+u digit cuo^'i va`o R6
    SJMP    $

; Ba.n ha~y vie^'t la.i ddoa.n chuo+ng tri`nh tre^n
; tha`nh mo^.t chuo+ng tri`nh con, dda(.t te^n la` BIN2BCD
```

- **Cong_16bit**

```
; co^.ng so^' 16-bit: 3CE7h + 3B8Dh
; ke^'t qua? lu+u trong: R7 R6

    CLR     C
    MOV     A,#0E7H
    ADD     A,#8DH
    MOV     R6,A
    MOV     A,#3CH
    ADDC    A,#3BH
    MOV     R7,A
    SJMP    $

; Ba.n ha~y vie^'t chuo+ng tri`nh con co^.ng 2 so^' 16-bit
```

- **Cong_5byte_BCD**

```
; co^.ng 5 byte chu+'a so^' BCD, ddi.a chi? ba('t dda^`u la` 40h
    ORG     0
    MOV     R0,#40H         ; na.p con tro?
    MOV     R2,#5           ; na.p bie^'n dde^'m
    CLR     A               ; A=0
    MOV     R7,A            ; xo'a R7
AGAIN:      ADD     A,@R0           ; co^.ng o^ nho+' tro? bo+?i R0
            DA      A               ; hie^.u chi?nh BCD
            JNC     NEXT            ; ne^'u CY=0 -> kho^ng ta(ng R7
            INC     R7              ; CY=1 -> ta(ng R7
NEXT:       INC     R0              ; ta(ng con tro?
            DJNZ    R2,AGAIN        ; la(.p dde^'n khi R2=0
            SJMP    $
```

- **Cong_Don**

```
; co^.ng do^`n 5 byte
    ORG     0
    MOV     R0,#40H         ; na.p ddi.a chi? cho con tro?
    MOV     R2,#5           ; R2: bie^'n dde^'m
```

```
        CLR     A           ; A = 0
    MOV     R7,A            ; xo'a R7
AGAIN:      ADD     A,@R0       ; co^.ng o^ nho+' tro? bo+?i R0
            JNC     NEXT        ; ne^'u CY=0 thi` kho^ng ta(ng R7
            INC     R7          ; ne^'u CY=1 thi` ta(ng R7
NEXT:       INC     R0          ; di.ch con tro? le^n 1 ddi.a chi?
            DJNZ    R2,AGAIN    ; la(.p cho dde^'n khi R2 = 0
            SJMP    $
```

- **Copy_String**

```
; copy mo^.t chuo^~i tu+` bo^. nho+' chuo+ng tri`nh va`o RAM no^.i
    ORG     0
    MOV     DPTR,#MYDATA    ; con tro? nguo^`n
    MOV     R0,#40H         ; con tro? ddi'ch
BACK:       CLR     A           ; A=0
            MOVC    A,@A+DPTR   ; la^'y data tu+` bo^. nho+' CT
            JZ      HERE        ; thoa't ne^'u data = 0 (NULL)
            MOV     @R0,A       ; lu+u va`o RAM
            INC     DPTR        ; ta(ng con tro? nguo^`n
            INC     R0          ; ta(ng con tro? ddi'ch
            SJMP    BACK        ;
HERE:       SJMP    HERE

    ORG     250H
MYDATA:     DB      'HUTECH',0      ; chuo^~i du+~ lie^.u
                    ; ke^'t thu'c la` 0 (NULL char)
    END
```

- **Copyblock**

```
; copy kho^'i du+~ lie^.u 10 byte tu+` 35h dde^'n 60h
    ORG     0
    MOV     R0,#35H   ; con tro? nguo^`n
    MOV     R1,#60H   ; con tro? ddi'ch
    MOV     R3,#10    ; bie^'n dde^'m (10 bytes)
BACK:   MOV     A,@R0     ; ddo.c 1 byte tu+` data nguo^`n
    MOV     @R1,A     ; copy va`o ddi'ch
    INC     R0        ; ta(ng con tro? nguo^`n
    INC     R1        ; ta(ng con tro? ddi'ch
    DJNZ    R3,BACK   ;
    SJMP    $
```

- P1_55_AA

```
; ba^.t/ta('t ca'c bit cu?a P1 xen ke~: AAh <-> 55h
;
    ORG     0
```

```
BACK:   MOV     A,#55H          ; A = 55h
        MOV     P1,A            ; P1 = 55h
        LCALL   DELAY           ;
        MOV     A,#0AAH         ; A = AAh
        MOV     P1,A            ; P1 = AAh
        LCALL   DELAY
        SJMP    BACK            ;

;this is the delay subroutine
        ORG 300H
DELAY:  PUSH    4               ; PUSH R4
        PUSH    5               ; PUSH R5
        MOV     R4,#0FFH        ; R4=FFH
NEXT:   MOV     R5,#0FFH        ; R5=255
AGAIN:  DJNZ    R5,AGAIN
        DJNZ    R4,NEXT
        POP     5               ; POP INTO R5
        POP     4               ; POP INTO R4
        RET                     ;
        END
```

- **Trabang_X2**

```
; ddo.c x tu+` P1
; tra ba?ng ti'nh x^2
; xua^'t ke^'t qua? ra P2
        ORG     0
        MOV     DPTR,#300H      ; na.p ddi.a chi? ba?ng tra
        MOV     A,#0FFH         ;
        MOV     P1,A            ; P1: input
BACK:   MOV     A,P1            ; ddo.c x
        MOVC    A,@A+DPTR       ; tra ba?ng ti'nh x^2
        MOV     P2,A            ; xua^'t ra P2
        SJMP    BACK            ;

        ORG     300H
XSQR_TABLE:
        DB      0,1,4,9,16,25,36,49,64,81
        END
```

- **Tru_16bit**

```
; tru+` 16-bit: 2762h - 1296h
        CLR     C               ; CY=0
        MOV     A,#62H          ; A=62H
        SUBB    A,#96H          ; 62H-96H=CCH, CY=1
        MOV     R7,A            ; lu+u ke^'t qua?
```

```
        MOV     A,#27H          ; A=27H
    SUBB    A,#12H          ; 27H-12H-1=14H
    MOV     R6,A            ; lu+u ke^'t qua?

    SJMP    $
```

- **Tru_8bit**

```
; tru+`: 4Ch - 6Eh
;
    CLR     C
    MOV     A,#4Ch    ; A=4CH
    SUBB    A,#6EH    ; A=A-6Eh
    JNC     NEXT      ; ne^'u CY=0 nha?y dde^'n NEXT
    CPL     A         ; ne^'u CY=1 la^'y bu` 2
    INC     A         ;
NEXT:   MOV     R1,A      ; lu+u ke^'t qua? va`o R1

    SJMP    $
```

2. INTERRUPT

- **INT1**

```
; Button no^'i vo+'i /INT1
; Nha^'n button -> LED (P1.3) sa'ng mo^.t lu'c ro^`i ta('t
    ORG  0000H
    LJMP MAIN               ;nha?y qua vu`ng vector nga('t

; ISR cu?a INT1
    ORG  0013H              ;INT1 ISR
    SETB P1.3               ;ba^.t LED sa'ng (1 byte)
    MOV  R3,#255            ;(2 byte)
BACK:   DJNZ R3,BACK            ;delay 1 chu't (2 byte)
    CLR  P1.3               ;ta('t LED (1 byte)
    RETI                    ;(1 byte)
; MAIN program for initialization
    ORG  30H
MAIN:   MOV  IE,#10000100B      ;cho phe'p nga('t ngoa`i 1 (/INT1)
HERE:   SJMP HERE               ;cho+` nha^.n nga('t
    END
```

- **Int1_Edge_Trigger**

```
; Cha^n 1.3 no^'i vo+'i loa
; Khi co' ca.nh xuo^'ng o+? INT1 -> ba^.t loa 1 lu'c ro^`i ta('t
    ORG     0000H
    LJMP    MAIN
;ISR cu?a INT1
```

```
        ORG     0013H          ;INT1 ISR
    SETB    P1.3           ;ba^.t loa
    MOV     R3,#255
BACK:   DJNZ    R3,HERE        ;delay 1 chu't
    CLR     P1.3           ;ta('t loa
    RETI                   ;

;MAIN program for initialization
    ORG     30H
MAIN:   SETB    TCON.2         ;INT1 ta'c ddo^.ng ca.nh
    MOV     IE,#10000100B   ;cho phe'p nga('t ngoa`i 1
HERE:   SJMP    HERE           ;cho+` nga('t
    END
```

- **Pulse**

```
;Pha't xung vuo^ng o+? P1.2 du`ng nga('t
    ORG     0
    LJMP    MAIN
    ORG     000BH
    CPL     P1.2
    MOV     TL0,#0H
    MOV     TH0,#0DCH
    RETI

    ORG     30H
MAIN:
    MOV     TMOD,#01H
    MOV     TH0,#0DCH
    MOV     IE,#82H
    SETB    TR0
HERE:   SJMP    HERE
    END
```

- **Read_P0_Write_P1_Pulse_P21_1**

```
; DDo.c data o+? P0, xua^'t ra P1, trong khi P2.1 pha't xung vuo^ng
; Du`ng Timer 0, mode 2 (auto reload)
    ORG  0000H
    LJMP MAIN      ;nha?y qua vu`ng vector nga('t
;
; ISR cua? Timer 0 -> pha't xung vuo^ng
    ORG  000BH     ;vector cu?a Timer 0
    CPL  P2.1      ;dda?o P2.1
    RETI
;
; Chuo+ng tri`nh chi'nh
```

```
        ORG  0030H
MAIN:   MOV  TMOD,#02H ;Timer 0,mode 2(auto reload)
        MOV  P0,#0FFH  ;P0: input port
        MOV  TH0,#-92
        MOV  IE,#82H   ;IE=10000010b cho phe'p nga('t Timer 0
        SETB TR0       ;cho phe'p Timer 0 cha.y
BACK:   MOV  A,P0      ;ddo.c ddu+~ lie^.u tu+` P0
        MOV  P1,A      ;xua^'t ra P1
        SJMP BACK
        END
```

- **Read_P0_Write_P1_Pulse_P21_2**

```
; DDo.c data tu+` P0, xua^'t ra P1, trong khi P2.1 pha't xung
; Du`ng Timer 1, mode 1
        ORG     0000H
        LJMP    MAIN      ;nha?y qua vu`ng vector nga('t
; ISR cu?a Timer 1 -> pha't xung
        ORG     001BH     ;vector nga('t Timer 1
        LJMP    ISR_T1

; Chuo+ng tri`nh chi'nh
        ORG     0030H
MAIN:   MOV     TMOD,#10H       ;timer 1, mode 1
        MOV     P0,#0FFH        ;P0: input port
        MOV     TL1,#low(-1000)
        MOV     TH1,#high(-1000)
        MOV     IE,#88H         ;IE=10001000b cho phe'p nga('t Timer 1
        SETB    TR1             ;cho phe'p Timer 1 cha.y
BACK:   MOV     A,P0            ;ddo.c data tu+` P0
        MOV     P1,A            ;xua^'t ra P1
        SJMP    BACK
;
; Timer 1 ISR. Timer 1 pha?i dduo+.c na.p la.i vi` mode 1 kho^ng na.p tu+. ddo^.ng
ISR_T1: CLR     TR1             ;du+`ng Timer 1
        CLR     P2.1            ;P2.1=0
        MOV     R2,#4           ;2 MC
HERE:   DJNZ    R2,HERE         ;4x2MC = 8MC
        MOV     TL1,#low(-1000) ;2 MC
        MOV     TH1,#high(-1000);2 MC
        SETB    TR1             ;cho phe'p Timer 1 cha.y, 1 MC
        SETB    P2.1            ;P2.1=1, 1 MC
        RETI
        END
```

- **Serial_Port_Interrupt_1**

```
        ; DDo.c data tu+` P1, xua^'t ra P2 va` serial port
        ORG     0
        LJMP    MAIN
        ORG     23H
        LJMP    SERIAL          ; nha?y dde^'n ISR cu?a nga('t port nt
        ORG     30H
MAIN:   MOV     P1,#0FFH        ; P1: input port
        MOV     TMOD,#20H       ; timer 1, mode 2 (auto reload)
        MOV     TH1,#0FDH       ; 9600 baud rate
        MOV     SCON,#50H       ; 8-bit, REN enabled
        MOV     IE,#10010000B   ; cho phe'p nga('t port nt
        SETB    TR1             ; cho phe'p timer 1 cha.y
BACK:   MOV     A,P1            ; ddo.c data tu` port 1
        MOV     SBUF,A          ; xua^'t ra port nt
        MOV     P2,A            ; xua^'t ra P2
        SJMP    BACK
;
;------------------SERIAL PORT ISR
        ORG     100H
SERIAL: JNB     RI,CHK_TI       ; RI = 0 -> nha?y dde^'n CHK_TI
        MOV     A,SBUF          ; RI = 1 -> receive
        CLR     RI              ; xo'a RI
CHK_TI: JNB     TI,EXIT
        CLR     TI              ; xo'a TI
EXIT:   RETI
        END
```

- **Serial_Port_Interrupt_2**

```
; DDo.c data tu+` P1, xua^'t ra P2
; Nha^.n data tu+` serial port, xua^'t ra P0
        ORG     0
        LJMP    MAIN
        ORG     23H
        LJMP    SERIAL          ;nha?y dde^'n serial ISR
        ORG     30H
MAIN:   MOV     P1,#0FFH        ;P1: input port
        MOV     TMOD,#20H       ;timer 1, mode 2 (auto reload)
        MOV     TH1,#0FDH       ;9600 baud rate
        MOV     SCON,#50H       ;8-bit, REN enabled
        MOV     IE,#10010000B   ;cho phe'p serial interrupt
        SETB    TR1             ;cho phe'p timer 1 cha.y
BACK:   MOV     A,P1            ;ddo.c data tu+` Port 1
        MOV     P2,A            ;xua^'t ra P2
        SJMP    BACK
```

```
;SERIAL PORT ISR
SERIAL: JNB     RI,TRANS        ;RI=0 -> nha?y
        MOV     A,SBUF          ;RI=1
        MOV     P0,A            ;xua^'t data nha^.n dduo+.c ra P0
        CLR     RI              ;xo'a RI
        RETI
TRANS:  CLR     TI              ;xo'a TI
        RETI
        END
```

- **Serial_Port_Timer_Interrupt**

```
; Pha't xung vuo^ng 5KHz o+? P0.1,
; nha^.n data tu+` serial port, xua^'t ra P0
; DDc.c data tu++` P1, ghi va`o 30h, va` xua^'t ra serial port
        ORG     0
        LJMP    MAIN
        ORG     000BH           ;ISR cu?a Timer 0
        CPL     P0.1            ;dda?o P0.1
        RETI
        ORG     23H
        LJMP    SERIAL          ;nha?y dde^'n ISR cu?a nga('t port nt
        ORG     30H
MAIN:   MOV     P1,#0FFH        ; P1: input port
        MOV     TMOD,#22H       ; Timer 0&1, mode 2, AUTO RELOAD
        MOV     TH1,#0F6H       ; 4800 BAUD RATE
        MOV     SCON,#50H       ; 8-bit, REN = 1
        MOV     TH0,#-92        ; TH0 = -92 -> pha't xung 5 KHz
        MOV     IE,#10010010B   ; cho phe'p nga('t serial port, Timer 0
        SETB    TR1             ; cho phe'p Timer 1 cha.y
        SETB    TR0             ; cho phe'p Timer 0 cha.y
BACK:   MOV     A,P1            ; ddo.c data tu+` port 1
        MOV     SBUF,A          ; xua^'t ra serial port
        MOV     P2,A            ; xua^'t ra P2
        SJMP    BACK

;SERIAL PORT ISR
SERIAL: JNB     RI,TRANS        ; RI = 0 -> nha?y
        MOV     A,SBUF          ; RI = 1: receive
        MOV     30h,A           ; lu+u data va`o o^ nho+' 30h
        CLR     RI              ; xo'a RI
        RETI
TRANS:  CLR     TI              ; xo'a TI
```

```
        RETI
    END
```

3. Keypad

- **Scankp**

```
; Ba`n phi'm hex no^'i va`o P1
; Chuo+ng tri`nh hie^?n thi. phi'm nha^'n ra LED 7 ddoa.n
; P1.0-P1.3: columns
; P1.4-P1.7: rows
; DDi.a chi? LED: A000h)

LOOP:   LCALL   READKB          ; tri. tra? ve^`: A = 0-15
        MOV     DPTR,#T7SEG
        MOVC    A,@A+DPTR
        MOV     DPTR,#0A000H    ; A000h: LED 1
        MOVX    @DPTR,A
        SJMP    LOOP

READKB: PUSH    7
SCAN:   MOV     A,#11111110B    ; col_0 -> GND
        MOV     R7,#0           ; R7 = i
CONT:   MOV     P1,A            ; no^'i col i -> GND
        MOV     A,P1            ; ddo.c row
        JNB     ACC.4,ROW_0     ; xe't xem row na`o?
        JNB     ACC.5,ROW_1
        JNB     ACC.6,ROW_2
        JNB     ACC.7,ROW_3
        RL      A               ; chua^?n bi. no^'i GND
        INC     R7              ; co^.t tie^'p theo
        CJNE    R7,#4,CONT      ; la^`n luo+.t no^'i GND 4 co^.t
        SJMP    SCAN            ; quay la.i que't tu+` co^.t 0

ROW_0:  MOV     A,R7            ; Row=0, Col=R7
        ADD     A,#0            ; A = 0 + R7
        SJMP    EXIT
ROW_1:  MOV     A,R7            ; Row=1, Col=R7
        ADD     A,#4            ; A = 4 + R7
        SJMP    EXIT
ROW_2:  MOV     A,R7            ; Row=2, Col=R7
        ADD     A,#8            ; A = 8 + R7
        SJMP    EXIT
ROW_3:  MOV     A,R7            ; Row=3, Col=R7
        ADD     A,#12           ; A = 12 + R7
EXIT:   POP     7
```

```
        RET

T7SEG:  DB      40H,79H,24H,30H,19H,12H,02H,78H,00H,10H,
        DB      08H,03H,46H,21H,04H,0EH

        END
```

4. LCD

• LCD_BusyFlag

```
;Xua^'t ra LCD "Hello"
;P1=data pin
;P3.0 -> RS pin
;P3.1 -> R/W pin
;P3.2 -> E pin

RS      EQU     P3.0
RW      EQU     P3.1
E       EQU     P3.2
        ORG     0
        MOV     A,#38H          ;init. LCD 2 do`ng, ma tra^.n 5x7
        ACALL   CSTROBE
        MOV     A,#0CH          ;LCD on, cursor on
        ACALL   CSTROBE
        MOV     A,#01H          ;clear LCD
        ACALL   CSTROBE
        MOV     A,#06H          ;cursor di.ch pha?i
        ACALL   CSTROBE

        MOV     A,#86H          ;chuye^?n cursor dde^'n line 1, pos. 6
        ACALL   CSTROBE
        MOV     A,#'H'
        ACALL   DSTROBE
        MOV     A,#'e'
        ACALL   DSTROBE
        MOV     A,#'l'
        ACALL   DSTROBE
        MOV     A,#'l'
        ACALL   DSTROBE
        MOV     A,#'o'
        ACALL   DSTROBE
HERE:   SJMP    HERE

CSTROBE:                        ;command strobe
        ACALL   READY           ;is LCD ready?
        MOV     P1,A            ;xua^'t ma~ le^.nh
        CLR     RS              ;RS=0: le^.nh
        CLR     RW              ;R/W=0 -> ghi ra LCD
        SETB    E               ;E=1 -> ta.o ca.nh xuo^'ng
        CLR     E               ;E=0 ,cho^'t
        RET
```

```
DSTROBE:                ;data strobe
      ACALL   READY       ;is LCD ready?
      MOV     P1,A        ;xua^'t du+~ lie^.u
      SETB    RS          ;RS=1 for data
      CLR     RW          ;R/W=0 to write to LCD
      SETB    E           ;E=1 -> ta.o ca.nh xuo^'ng
      CLR     E           ;E=0, cho^'t
      RET

; kie^?m tra co+` BF
READY:  SETB    P1.7        ;P1.7: input
      CLR     RS          ;RS=0: thanh ghi le^.nh
      SETB    RW          ;R/W=1: ddo.c
BACK:   CLR     E           ;E=0 -> ta.o ca.nh le^n
      SETB    E           ;E=1
      JB      P1.7,BACK   ;cho+` busy flag=0
      RET
      END
```

- **LCD_ScanKB**

```
;P1 = data/command pin
;P3.0 -> RS pin
;P3.1 -> R/W pin
;P3.2 -> E pin
;P2 -> Keypad
      ORG     0
RS      EQU     P3.0
RW      EQU     P3.1
EN      EQU     P3.2

      MOV     A,#38H      ;init. LCD 2 lines,5x7 matrix
      ACALL   CSTROBE
      MOV     A,#0EH      ;LCD on, cursor on
      ACALL   CSTROBE
      MOV     A,#01H      ;clear LCD
      ACALL   CSTROBE
      MOV     A,#06H      ;cursor di.ch pha?i
      ACALL   CSTROBE
      MOV     A,#80H      ;cursor: line 1, pos. 0
      ACALL   CSTROBE

AGAIN:  LCALL   READKP
      ORL     A,#30h
```

```
            ACALL   DELAY
        ACALL   DSTROBE
        SJMP    AGAIN

;command strobe
CSTROBE:
        ACALL   READY       ;is LCD ready?
        MOV     P1,A        ;xua^'t ma~ le^.nh
        CLR     RS          ;RS=0: le^.nh
        CLR     RW          ;R/W=0: ghi ra LCD
        SETB    EN          ;EN=1 -> ta.o ca.nh xuo^'ng
        CLR     EN          ;EN=0 ,cho^'t
        RET

;data strobe
DSTROBE:
        ACALL   READY       ;is LCD ready?
        MOV     P1,A        ;xua^'t du+~ lie^.u ra P1
        SETB    RS          ;RS=1: du+~ lie^.u
        CLR     RW          ;R/W=0 ghi ra LCD
        SETB    EN          ;EN=1 -> ta.o ca.nh xuo^'ng
        CLR     EN          ;EN=0, cho^'t
        RET

READY:  SETB    P1.7        ;P1.7: input
        CLR     RS          ;RS=0: le^.nh
        SETB    RW          ;R/W=1: ddo.c
BACK:   CLR     EN          ;EN=0 -> ta.o ca.nh le^n
        SETB    EN          ;EN=1
        JB      P1.7,BACK   ;cho+` busy flag=0
        RET

; DDo.c ba`n phi'm
READKP: PUSH    7
SCAN:   MOV     A,#11111110B    ; col_0 -> GND
        MOV     R7,#0           ; R7 = i
CONT:   MOV     P2,A            ; no^'i col i -> GND
        MOV     A,P2            ; ddo.c row
        JNB     ACC.4,ROW_0     ; xe't xem row na`o?
        JNB     ACC.5,ROW_1
        JNB     ACC.6,ROW_2
        JNB     ACC.7,ROW_3
        RL      A               ; chua^?n bi. no^'i GND
        INC     R7              ; co^.t tie^'p theo
```

```
            CJNE    R7,#4,CONT      ; la^`n luo+.t no^'i GND 4 co^.t
      SJMP    SCAN            ; quay la.i que't tu+` co^.t 0
ROW_0:  MOV     A,R7            ; Row=0, Col=R7
      ADD     A,#0            ; A = 0 + R7
      SJMP    EXIT
ROW_1:  MOV     A,R7            ; Row=1, Col=R7
      ADD     A,#4            ; A = 4 + R7
      SJMP    EXIT
ROW_2:  MOV     A,R7            ; Row=2, Col=R7
      ADD     A,#8            ; A = 8 + R7
      SJMP    EXIT
ROW_3:  MOV     A,R7            ; Row=3, Col=R7
      ADD     A,#12           ; A = 12 + R7
EXIT:   POP     7
      RET

DELAY:  PUSH    6
      PUSH    7
      MOV     R7,#0FFh
LP1:    MOV     R6,#0FFh
LP0:    DJNZ    R6,LP0
      DJNZ    R7,LP1
      POP     7
      POP     6
      RET
      END
```

5. LED

- **counter_led**

```
; Que't LED
; a,b,c,d,e,f,g -> Port 2
; P3.0 -> LED1
; P3.1 -> LED2
; P3.2 -> LED3
; P3.4(T0) -> Button
; 40h: ha`ng do+n vi.
; 41h: ha`ng chu.c
; 42h: ha`ng tra(m

      ORG     0H
      MOV     DPTR,#LED7SEG   ; DPTR tro? dde^'n ba?ng ma~ LED
      MOV     TMOD,#06h       ; counter 0, mode 2
      MOV     TH0,#0
```

```
        SETB    P3.0        ; ta('t ta^'t ca? ca'c LED
    SETB    P3.1
    SETB    P3.2
    SETB    P3.4        ; P3.4: input
    SETB    TR0         ; cho phe'p counter 0 cha.y
BEGIN:  MOV     A,TL0
    LCALL   BIN2BCD
    ; tra ba?ng, ddo^?i BCD -> LED 7 ddoa.n
    MOV     A,40h
    MOVC    A,@A+DPTR
    MOV     40h,A
    MOV     A,41h
    MOVC    A,@A+DPTR
    MOV     41h,A
    MOV     A,42h
    MOVC    A,@A+DPTR
    MOV     42h,A
    LCALL   DISPLAY
    SJMP    BEGIN

DISPLAY:
    MOV     P2,40H      ; LED1
    CLR     P3.0        ; ba^.t LED1 sa'ng
    ACALL   DELAY       ; delay
    SETB    P3.0        ; ta('t LED1

    MOV     P2,41H      ; LED2
    CLR     P3.1        ; ba^.t LED2 sa'ng
    ACALL   DELAY       ; delay
    SETB    P3.1        ; ta('t LED2

    MOV     P2,42H      ; LED 3
    CLR     P3.2        ; ba^.t LED3 sa'ng
    ACALL   DELAY       ; delay
    SETB    P3.2        ; ta('t LED3
    RET

BIN2BCD:
    MOV     B,#10       ; B=10
    DIV     AB          ; chia cho 10
    MOV     40h,B       ; lu+u digit tha^'p
    MOV     B,#10       ;
    DIV     AB          ; chia cho 10
    MOV     41h,B       ; lu+u digit tie^'p theo va`o 41h
```

```
        MOV     42h,A           ; lu+u digit cuo^'i va`o 42h
    RET

DELAY:
    MOV     R1,#10
    MOV     R0,#0FFh
LOOP:   DJNZ    R0,LOOP
    DJNZ    R1,LOOP
    RET
;
LED7SEG:
    DB      0C0H,0F9H,0A4H,0B0H,99H,92H,82H,0F8H,80H,98H
    DB      88H,0C6H,86H,8EH,82H,89H
    END
```

- **Quetled**

```
; a,b,c,d,e,f,g -> Port 2
; P3.0 -> LED1
; P3.1 -> LED2
; P3.1 -> LED3
    ORG     0H
    MOV     P3,#0FFH
    MOV     DPTR,#LED7SEG
BEGIN:
    MOV     A,#4
    MOVC    A,@A+DPTR
    MOV     40H,A

    MOV     A,#3
    MOVC    A,@A+DPTR
    MOV     41H,A

    MOV     A,#2
    MOVC    A,@A+DPTR
    MOV     42H,A

    MOV     A,#1
    MOVC    A,@A+DPTR
    MOV     43H,A

    LCALL   DISPLAY
    SJMP    BEGIN

DISPLAY:
```

```
        ; LED1
        MOV     P2,40H
        CLR     P3.0
        ACALL   DELAY_25
        SETB    P3.0
; LED2
        MOV     P2,41H
        CLR     P3.1
        ACALL   DELAY_25
        SETB    P3.1
; LED 3
        MOV     P2,42H
        CLR     P3.2
        ACALL   DELAY_25
        SETB    P3.2
; LED 4
        MOV     P2,43H
        CLR     P3.3
        ACALL   DELAY_25
        SETB    P3.3
        RET
;
DELAY_25:
        MOV     R1,#10
        MOV     R0,#0
LOOP:   DJNZ    R0,LOOP
        DJNZ    R1,LOOP
        RET;
LED7SEG:
        DB      0C0H,0F9H,0A4H,0B0H,99H,92H,82H,0F8H,80H,98H
        DB      88H,0C6H,86H,8EH,82H,89H
        END
```

- **Quetled_123**

```
; a,b,c,d,e,f,g -> Port 2
; P3.0 -> LED1
; P3.1 -> LED2
; P3.1 -> LED3
        ORG     0H
        MOV     P3,#0FFh        ; ta('t ta^'t ca? ca'c LED

BEGIN:  MOV     P2,#0B0h        ; xua^'t ra P2 ma~ cu?a '3'
        CLR     P3.0            ; ba^.t LED1
        ACALL   DELAY           ; delay
```

```
        SETB    P3.0            ; ta('t LED1

    MOV     P2,#0A4h        ; xua^'t ra P2 ma~ cu?a '2'
    CLR     P3.1            ; ba^.t LED2
    ACALL   DELAY           ; delay
    SETB    P3.1            ; ta('t LED2

    MOV     P2,#0F9h        ; xua^'t ra P2 ma~ cu?a '1'
    CLR     P3.2            ; ba^.t LED3
    ACALL   DELAY           ; delay
    SETB    P3.2            ; ta('t LED3
    SJMP    BEGIN

DELAY:  MOV     R1,#10
    MOV     R0,#0FFh
LOOP:   DJNZ    R0,LOOP
    DJNZ    R1,LOOP
    RET
    END
```

- **Quetled_8255**

```
ORG     0H
    MOV     DPTR,#4003H
    MOV     A,#80H
    MOVX    @DPTR,A
    MOV     P3,#0FFH
    MOV     DPTR,#LED7SEG
BEGIN:
    MOV     A,#4
    MOVC    A,@A+DPTR
    MOV     40H,A

    MOV     A,#3
    MOVC    A,@A+DPTR
    MOV     41H,A

    MOV     A,#2
    MOVC    A,@A+DPTR
    MOV     42H,A

    MOV     A,#1
    MOVC    A,@A+DPTR
    MOV     43H,A
```

```
        LCALL   DISPLAY
    SJMP    BEGIN

DISPLAY:
    PUSH    DPH
    PUSH    DPL
    MOV     A,40H
    MOV     DPTR,#4000H
    MOVX    @DPTR,A
    MOV     DPTR,#4001H     ; chon LED o PB
    MOV     A,#0FEH
    MOVX    @DPTR,A
    ACALL   DELAY_25

    MOV     A,41H           ; xuat nd o nho 41h ra PA
    MOV     DPTR,#4000H
    MOVX    @DPTR,A
    MOV     DPTR,#4001H     ; cho.n
    MOV     A,#0FDH
    MOVX    @DPTR,A
    ACALL   DELAY_25

    MOV     A,42H
    MOV     DPTR,#4000H
    MOVX    @DPTR,A
    MOV     DPTR,#4001H
    MOV     A,#0FBH
    MOVX    @DPTR,A
    ACALL   DELAY_25

    MOV     A,43H
    MOV     DPTR,#4000H
    MOVX    @DPTR,A
    MOV     DPTR,#4001H
    MOV     A,#0F7H
    MOVX    @DPTR,A
    ACALL   DELAY_25
    POP     DPL
    POP     DPH
    RET
;
DELAY_25:
    MOV     R1,#10
    MOV     R0,#0
```

```
    LOOP:   DJNZ    R0,LOOP
    DJNZ    R1,LOOP
    RET
;
LED7SEG:
    DB      0C0H,0F9H,0A4H,0B0H,99H,92H,82H,0F8H,80H,98H
    DB      88H,0C6H,86H,8EH,82H,89H

    END
```

- **Quetled_8255_Ram**

```
ORG     0H
    MOV     DPTR,#4003H
    MOV     A,#80H
    MOVX    @DPTR,A
    MOV     P3,#0FFH
    MOV     DPTR,#LED7SEG
BEGIN:
    MOV     A,#9
    MOVC    A,@A+DPTR
    MOV     DPTR,#2000H
    MOVX    @DPTR,A
    MOV     DPTR,#LED7SEG

    MOV     A,#3
    MOVC    A,@A+DPTR
    MOV     41H,A

    MOV     A,#2
    MOVC    A,@A+DPTR
    MOV     42H,A

    MOV     A,#1
    MOVC    A,@A+DPTR
    MOV     43H,A

    LCALL   DISPLAY
    SJMP    BEGIN

DISPLAY:
    PUSH    DPH
    PUSH    DPL
    MOV     DPTR,#2000H
    MOVX    A,@DPTR
```

```
        MOV     DPTR,#4000H
    MOVX    @DPTR,A
    MOV     DPTR,#4001H     ; chon LED o PB
    MOV     A,#0FEH
    MOVX    @DPTR,A
    ACALL   DELAY_25

    MOV     A,41H           ; xuat nd o nho 41h ra PA
    MOV     DPTR,#4000H
    MOVX    @DPTR,A
    MOV     DPTR,#4001H     ; cho.n
    MOV     A,#0FDH
    MOVX    @DPTR,A
    ACALL   DELAY_25

    MOV     A,42H
    MOV     DPTR,#4000H
    MOVX    @DPTR,A
    MOV     DPTR,#4001H
    MOV     A,#0FBH
    MOVX    @DPTR,A
    ACALL   DELAY_25

    MOV     A,43H
    MOV     DPTR,#4000H
    MOVX    @DPTR,A
    MOV     DPTR,#4001H
    MOV     A,#0F7H
    MOVX    @DPTR,A
    ACALL   DELAY_25
    POP     DPL
    POP     DPH
    RET
;
DELAY_25:
    MOV     R1,#10
    MOV     R0,#0
LOOP:   DJNZ    R0,LOOP
    DJNZ    R1,LOOP
    RET
;
LED7SEG:
    DB      0C0H,0F9H,0A4H,0B0H,99H,92H,82H,0F8H,80H,98H
    DB      88H,0C6H,86H,8EH,82H,89H
```

```
      END
```

- **Led Blink**

```
      ORG     0
LOOP:   SETB    P2.0
      ACALL   DELAY
      CLR     P2.0
      ACALL   DELAY
      SJMP    LOOP

DELAY:  MOV     R6,#0FFh
LP2:    MOV     R7,#0FFh
LP1:    DJNZ    R7,LP1
      DJNZ    R6,LP2
      RET
```

- **Switch Led On**

```
      ORG     0
      SETB    P3.0          ;P3.0: input
LOOP:   JNB     P3.0,LOOP
LOOP1:  JB      P3.0,LOOP1
      CLR     P2.0
      ACALL   DELAY
      SETB    P2.0
      SJMP    LOOP

DELAY:  MOV     R6,#0FFh
LP2:    MOV     R7,#0FFh
LP1:    DJNZ    R7,LP1
      DJNZ    R6,LP2
      RET
```

6. SERIALPORT

- **Receive_Char_Send_To_P1**

```
; Nha^.n ky' tu+. tu+` serial port, xua^'t ra P1
      ORG     0
      MOV     TMOD,#20H ;timer1, mode 2 (auto reload)
      MOV     TH1,#-6   ;4800 baud
      MOV     SCON,#50H ;8-bit, REN enabled
      SETB    TR1       ;cho phe'p timer 1 cha.y
HERE:   JNB     RI,HERE   ;cho+` thu xong (cho+` RI=1)
      MOV     A,SBUF    ;ddo.c du+~ lie^.u va`o A
      MOV     P1,A      ;xua^'t ra port 1
```

```
        CLR     RI        ;xo'a RI dde^? chua^?n bi. nha^.n byte tie^'p theo
    SJMP    HERE
```

- **Transmit**

```
; pha't ki' tu+. 'A' lie^n tu.c
; XTAL 11.0592MHz
    MOV     TMOD,#20H       ; timer 1, mode 2
    MOV     TH1,#-6         ; 4800 baud rate
    MOV     SCON,#50H       ; 8-bit UART, REN enable
    SETB    TR1             ; cho phe'p Timer 1 cha.y
AGAIN:  MOV     SBUF,#'A'       ; pha't ky' tu+. 'A'
HERE:   JNB     TI,HERE         ; cho+` pha't xong
    CLR     TI              ; xo'a TI
    SJMP    AGAIN           ; tie^'p tu.c pha't
```

- **Transmit_B**

```
; xua^'t ki' tu+. 'B' lie^n tu.c
    ORG     0
    MOV     A,PCON          ;A=PCON
    SETB    ACC.7           ;
    MOV     PCON,A          ;SMOD=1

    MOV     TMOD,#20H       ;Timer 1, mode 2,auto reload
    MOV     TH1,-3          ;baud rate 19200
    MOV     SCON,#50H       ;8-bit data, RI enabled
    SETB    TR1             ;cho phe'p Timer 1 cha.y
    MOV     A,#'B'          ;luu+ ma~ ASCII cu?A 'B' va`o ACC
A_1:    CLR     TI              ;xo'a TI
    MOV     SBUF,A          ;pha't
H_1:    JNB     TI H_1          ;cho+` pha't xong
    SJMP    A_1             ;tie^'p tu.c pha't
```

- **Transmit_String_Receive_Char**

```
; Pha't chuo^~i "We are ready!"
; Sau ddo' nha^.n ki' tu+. tu+` serial port, xua^'t no' ra P1
    ORG     0
    MOV     P2,#0FFH        ;P2: input port
    MOV     TMOD,#20H       ;timer 1,mode 2(auto-reload)
    MOV     TH1,#0FAH       ;4800 baud rate
    MOV     SCON,#50H       ;8-bit, REN enabled
    SETB    TR1             ;cho phe'p timer 1 cha.y
    MOV     DPTR,#MYDATA    ;na.p ddi.a chi? chuo^~i va`o DPTR
H_1:    CLR     A
    MOVC    A,@A+DPTR       ;ddo.c 1 ki' tu+.
    JZ      B_1             ;nha?y ne^'u la` ki' tu+. NULL (0)
```

```
            ACALL  SEND          ;ne^'u kho^ng, go.i CT con SEND
      INC    DPTR          ;ta(ng con tro? DPTR
      SJMP   H_1

B_1:    MOV    A,P2          ;ddo.c du+~ lie^.u o+? P2
      ACALL  SEND          ;xua^'t ki' tu+. ddo' ra port nt
      ACALL  RECV          ;nha^.n du+~ lie^.u tu+` port nt
      MOV    P1,A          ;xua^'t ra P1
      SJMP   B_1           ;

;CTC pha't data. ACC chu+'a data ca^`n pha't.
SEND:   MOV    SBUF,A        ;na.p data va`o SBUF dde^? pha't
H_2:    JNB    TI,H_2        ;cho+` pha't xong
      CLR    TI            ;xo'a TI
      RET                  ;

;CTC nha^.n data tu+` port nt.
RECV:   JNB    RI,RECV       ;cho+` thu xong
      MOV    A,SBUF        ;ca^'t data va`o ACC
      CLR    RI            ;xo'a RI
  RET                      ;
;
MYDATA:  DB     'We Are Ready!',0
      END
```

- **Transmit_Yes**

```
      ORG     0h
      MOV     TMOD,#20H      ;timer 1, mode 2
      MOV     TH1,#-3        ;9600 baud
      MOV     SCON,#50H      ;8-bit, REN enabled
      SETB    TR1            ;cho phe'p Timer 1 cha.y
AGAIN:  MOV     A,#'Y'         ;pha't 'Y'
      ACALL   TRANS
      MOV     A,#'E'         ;pha't 'E'
      ACALL   TRANS
      MOV     A,#'S'         ;pha't 'S'
      ACALL   TRANS
      SJMP    AGAIN

;CTC pha't du+~ lie^.u
TRANS:  MOV     SBUF,A         ;na.p data va`o SBUF
HERE:   JNB     TI,HERE        ;cho+` pha't xong
      CLR     TI             ;xo'a TI
      RET
```

4. Timer

• Counter

```
; DDe^'m xung ngoa`i tu+` ngo~ T1 (P3.5)
      MOV     TMOD,#01100000B        ; counter 1,mode 2,C/T=1
                                     ; xung ngoa`i
      MOV     TH1,#0                 ; xo'a TH1
      SETB    P3.5                   ; T1: input
AGAIN:  SETB    TR1                  ; cho phe'p dde^'m
BACK:   MOV     A,TL1                ; ddo.c tri. dde^'m o+? TL1
      MOV     P2,A                   ; xua^'t ra port 2
      JNB     TF1,Back               ; dde^'m cho dde^'n khi TF=0
      CLR     TR1                    ; du+`ng counter 1
      CLR     TF1                    ; xo'a co+` TF
      SJMP    AGAIN
```

• Delay

```
; Delay da`i (Timer tra`n nhie^`u la^`n)
      MOV     TMOD,#10H       ; Timer 1,mode 1(16-bit)
      MOV     R3,#200         ; bie^'n dde^'m so^' la^`n tra`n
AGAIN:  MOV     TL1,#08         ; TL1=08,low byte
      MOV     TH1,#01         ; TH1=01,Hi byte
      SETB    TR1             ; cho phe'p Timer 1 cha.y
BACK:   JNB     TF1,BACK        ; cho+` Timer 1 tra`n
      CLR     TR1             ; du+`ng timer 1
      CLR     TF1             ; xo'a TF1
      DJNZ    R3,AGAIN        ; tie^'p tu.c ne^'u R3 chu+a = 0
                              ;
      END
```

• Pulse

```
; pha't xung o+? P1.5
      ORG     0
      MOV     TMOD,#01        ; Timer 0,mode 1(16-bit mode)
HERE:   MOV     TL0,#0F2H       ; TL0=F2H, low byte
      MOV     TH0,#0FFH       ; TH0=FFH, high byte
      CPL     P1.5            ; dda?o bit P1.5
      ACALL   DELAY
      SJMP    HERE            ;
; delay using timer 0
DELAY:  SETB    TR0             ; cho phe'p Timer 0 cha.y
AGAIN:  JNB     TF0,AGAIN       ; cho+` Timer 0 tra`n
      CLR     TR0             ; du+`ng Timer 0
      CLR     TF0             ; xo'a TF0
      RET
```

- **Pulse1**

```
; pha't xung ta.i P1.5 du`ng Timer 1, mode 1
      ORG     0
      MOV     TMOD,#10H       ; timer 1, mode 1(16-bit)
AGAIN:  MOV     TL1,#34H        ; TL1=34H,low byte
      MOV     TH1,#76H        ; TH1=76H,Hi byte
                        ; (tri. dde^'m = 7634H)
      SETB    TR1             ; cho phe'p timer 1 cha.y
BACK:   JNB     TF1,BACK        ; cho+` Timer 1 tra`n
      CLR     TR1             ; ddu+`ng timer 1
      CPL     P1.5            ; dda?o bit P1.5
      CLR     TF1             ; xo'a TF1
      SJMP    AGAIN           ;
```

- **Pulse2**

```
; pha't xung ta.i P1.5
       MOV  TMOD,#10H      ;timer 1, mode 1(16-bit)
AGAIN:    MOV  TL1,#1AH       ;TL1=1A,low byte
       MOV  TH1,#0FFH      ;TH1=FF,Hi byte
       SETB TR1            ;cho phe'p Timer 1 cha.y
BACK:     JNB  TF1,BACK       ;cho+` Timer 1 tra`n
       CLR  TR1            ;du+`ng Timer 1
       CPL  P1.5           ;dda?o bit P1.5
       CLR  TF1            ;xo'a TF1
       SJMP AGAIN          ;
```

- **Pulse3**

```
; pha't xung ta.i P2.3
      ORG     0
      MOV     TMOD,#10H       ;timer 1, mode 1 (16-bit)
AGAIN:  MOV     TL1,#00         ;TL1=00, low byte
      MOV     TH1,#0DCH       ;TH1=DC, hi byte
      SETB    TR1             ;cho phe'p Timer 1 cha.y
BACK:   JNB     TF1,BACK        ;cho+` tra`n
      CLR     TR1             ;du+`ng Timer 1
      CPL     P2.3            ;dda?o bit P2.3
      CLR     TF1             ;xo'a TF1
      SJMP    AGAIN           ;
```

- **Pulse4**

```
 ORG     0
      MOV     TMOD,#2H        ; Timer 0,mode 2
                        ; (8-bit,auto reload)
      MOV     TH0,#-150       ; TH0=6AH = bu`2 cu?a -150
AGAIN:  SETB    P1.3            ; P1.3=1
```

```
        ACALL   DELAY
    ACALL   DELAY
    CLR     P1.3        ; P1.3=0
    ACALL   DELAY
    SJMP    AGAIN

DELAY:  SETB    TR0         ; cho phe'p Timer 0 cha.y
BACK:   JNB     TF0,BACK    ; cho+` TF0 tra`n
    CLR     TR0         ; du+`ng Timer0
    CLR     TF0         ; xo'a TF0
    RET
```

- **Pulse5**

```
; pha't xung o+? P1.0
    ORG     0
    MOV     TMOD,#2H        ;Timer 0,mode 2
                    ;(8-bit,auto reload)
    MOV     TH0,#0          ;TH0=0
AGAIN:  MOV     R5,#250         ;dde^'m so^' la^`n tra`n (250 la^`n)
    ACALL   DELAY
    CPL     P1.0
    SJMP    AGAIN
DELAY:  SETB    TR0         ;cho phe'p Timer0 cha.y
BACK:   JNB     TF0,BACK        ;cho+` tra`n
    CLR     TR0         ;du+`ng timer 0
    CLR     TF0         ;xo'a TF0
    DJNZ    R5,DELAY
    RET
```

5. ADC

- **ADC0804**

```
;P1 <- D0-D7
;P3.0 <- /INTR
;P3.1 -> /WR
    ORG     0
    MOV     P1,#0FFH        ;
    SETB    P3.0        ;P3.0: input

LOOP:   CLR     P3.1        ;pha't xung START
    SETB    P3.1

    JB      P3.0,$          ;cho+` bie^'n ddo^?i AD

    MOV     A,P1        ;ddo.c data va`o A
```

```
        MOV     40h,A           ;lu+u va`o o^ nho+' 40h
    MOV     P2,A            ;xua^'t ra P2
    SJMP    LOOP
```

- **Read_Ad**

```
ORG     0
    MOV     DPTR,#LED7SEG   ; DPTR tro? dde^'n ba?ng ma~ LED
    MOV     P1,#0FFH
AGAIN:  MOV     A,P1
    LCALL   BIN2BCD
    ; tra ba?ng, ddo^?i BCD -> LED 7 ddoa.n
    MOV     A,40h
    MOVC    A,@A+DPTR
    MOV     40h,A
    MOV     A,41h
    MOVC    A,@A+DPTR
    MOV     41h,A
    MOV     A,42h
    MOVC    A,@A+DPTR
    MOV     42h,A
    LCALL   DISPLAY
    SJMP    AGAIN

DISPLAY:
    MOV     P2,40H          ; LED1
    CLR     P3.0            ; ba^.t LED1 sa'ng
    ACALL   DELAY           ; delay
    SETB    P3.0            ; ta('t LED1

    MOV     P2,41H          ; LED2
    CLR     P3.1            ; ba^.t LED2 sa'ng
    ACALL   DELAY           ; delay
    SETB    P3.1            ; ta('t LED2

    MOV     P2,42H          ; LED 3
    CLR     P3.2            ; ba^.t LED3 sa'ng
    ACALL   DELAY           ; delay
    SETB    P3.2            ; ta('t LED3
    RET
```

```
        BIN2BCD:
        MOV     B,#10           ; B=10
        DIV     AB              ; chia cho 10
        MOV     40h,B           ; lu+u digit tha^'p
        MOV     B,#10           ;
        DIV     AB              ; chia cho 10
        MOV     41h,B           ; lu+u digit tie^'p theo va`o 41h
        MOV     42h,A           ; lu+u digit cuo^'i va`o 42h
        RET

DELAY:  PUSH    7
        PUSH    6
        MOV     R7,#10
LP2:    MOV     R6,#0FFh
LP1:    DJNZ    R6,LP1
        DJNZ    R7,LP2
        POP     6
        POP     7
        RET
;
LED7SEG:
        DB      0C0H,0F9H,0A4H,0B0H,99H,92H,82H,0F8H,80H,98H
        DB      88H,0C6H,86H,8EH,82H,89H
        END
```

My Name : Pham Trung Hieu
My E-mail : Hieutechnology@Gmail.com or Hieutechnology@Yahoo.com
My Phone : 08-8349063 0958612485