CHƯƠNG I

GIỚI THIỆU VI ĐIỀU KHIỂN 8051

I. CẤU TẠO VI ĐIỀU KHIỂN 8051

1. TÓM TẮT PHẦN CỨNG HỌ MSC-51 (8051)

MCS-51 là họ IC vi điều khiển do hãng Intel sản xuất. Các IC tiêu biểu cho họ là 8031, 8051, 8951... Những đặc điểm chính và nguyên tắt hoạt động của các bộ vi điều khiển này khác nhau không nhiều. Khi đã sử dụng thành thạo một loại vi điều khiển thì ta có thể nhanh chóng vận dụng kinh nghiệm để làm quen và làm chủ các ứng dụng của một bộ vi điều khiển khác. Vì vậy để có những hiểu biết cụ thể về các bộ vi điều khiển cũng như để phục vụ cho đề tài tốt nghiệp này ta bắt đầu tìm hiểu một bộ vi điều khiển thông dụn g nhất, đó là họ MCS-51 và nếu như họ MCS-51 là họ điển hình thì 8051 lại chính là đại diện tiêu biểu

Các đặc điểm của 8051 được tóm tắt như sau :

√ 4 KB ROM 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 lý Boolean (hoạt động trên bit đơn).

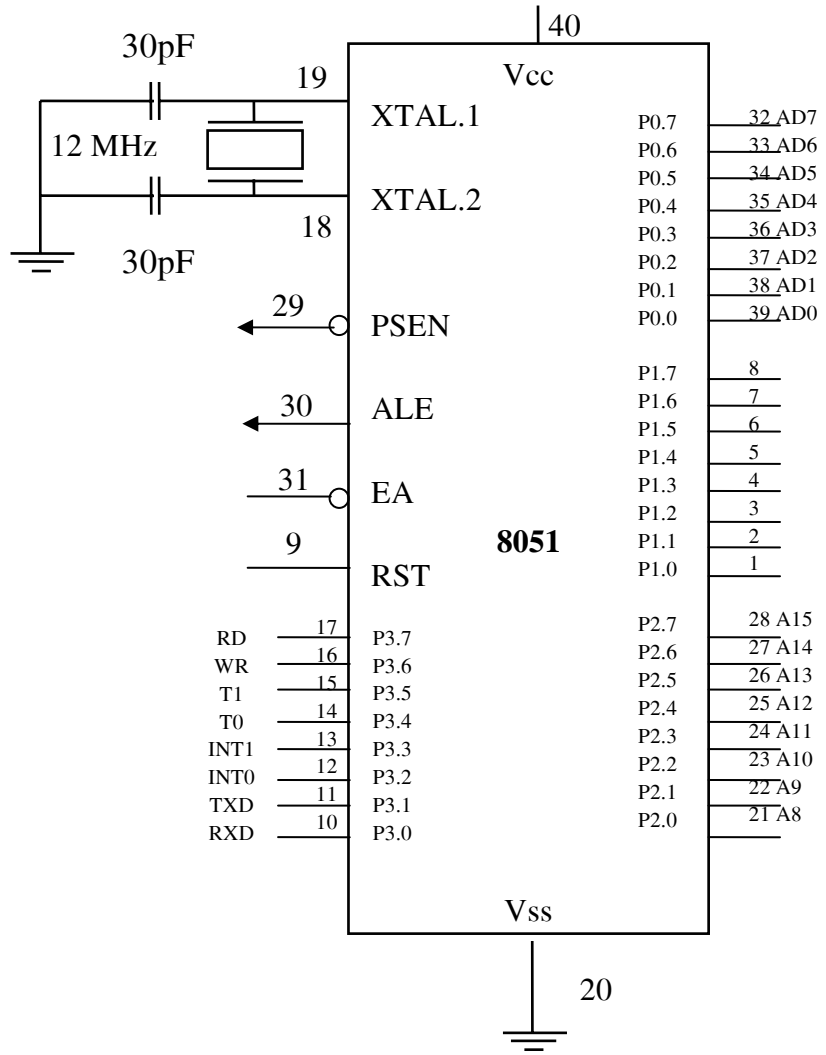
√ 210 vị trí nhớ có thể định vị bit.

√ 4 μs cho hoạt động nhân hoặc chia.

Bảng mô tả sự khác nhau của các IC trong họ MSC-51 :

Loại	Bộ Nhớ Mã Trên CHIP	Bộ Nhớ Dữ Liệu Trên CHIP	Số Timer
8051	4K ROM	128 Byte	2
8031	0K ROM	128 Byte	2
8751	4K ROM	128 Byte	2
8052	8 K ROM	256Byte	2
8032	0 K ROM	256Byte	2
8752	8K EPROM	256Byte	2

2. CẤU TRÚC VĐK 8051, CHỨC NĂNG TỪNG CHÂN



Chức năng hoạt động của từng chân (pin) được tóm tắt như sau :

√ Từ chân 1÷ 8 Port 1 (P1.0, . . ., P1.7) dùng làm Port xuất nhập I/O để giao tiếp bên ngoài.

√ Chân 9 (RST) là chân để RESET cho 8051. Bình thường các chân này ở mức thấp. Khi ta đưa tín hiệu này lên cao (tối thiểu 2 chu kỳ máy). Thì những thanh ghi nội của 8051 được LOAD những giá trị thích hợp để khởi động lại hệ thống.

Từ chân 10÷17 là Port3 (P3.0, P3.1, . . ., P3.7) dùng vào hai mục đích : dùng là Port xuất / nhập I/O hoặc mỗi chân giữ một chức năng cá biệt được tóm tắt sơ bộ như sau :

- P3.0 (RXD) : Nhận dữ liệu từ Port nối tiếp.
- P3.1 (TXD) : Phát dữ liệu từ Port nối tiếp.
- P3.2 ($\overline{\text{INT0}}$) : Ngắt 0 bên ngoài.
- P3.3 ($\overline{\text{INT1}}$) : Ngắt 1 từ bên ngoài.
- P3.4 (T0) : Timer/Counter 0 nhập từ bên ngoài.
- P3.5 (T1) : Timer/Counter 1 nhập từ bên ngoài.
- P3.6 ($\overline{\text{WR}}$) : Tín hiệu Strobe ghi dữ liệu lên bộ nhớ bên ngoài.
- P3.7 ($\overline{\text{RD}}$) : Tín hiệu Strobe đọc dữ liệu lên bộ nhớ bên ngoài.

√ Các chân 18,19 (XTAL2 và XTAL1) được nối với bộ dao động thạch anh 12 MHz để tạo dao động trên CHIP. Hai tụ 30 pF được thêm vào để ổn định dao động.

√ Chân 20 (Vss) nối đất (Vss = 0).

√ Từ chân 21÷28 là Port 2 (P2.0, P2.1, . . ., P2.7) dùng vào hai mục đích: làm Port xuất/nhập I/O hoặc dùng làm byte cao của bus địa chỉ thì nó không còn tác dụng I/O nữa. Bởi vì ta muốn dùng EPROM và RAM ngoài nên phải sử dụng Port 2 làm byte cao bus địa chỉ.

√ Chân 29 ($\overline{\text{PSEN}}$) là tín hiệu điều khiển xuất ra của 8051, nó cho phép chọn bộ nhớ ngoài và được nối chung với chân của OE (Outout Enable) của EPROM ngoài để cho phép đọc các byte của chương trình. Các xung tín hiệu $\overline{\text{PSEN}}$ hạ thấp trong suốt thời gian thi hành lệnh. Những mã nhị phân của chương trình được đọc từ EPROM đi qua bus dữ liệu và được chốt vào thanh ghi lệnh của 8051 bởi mã lệnh.

Chân 30 (ALE : Adress Latch Enable) là tín hiệu điều khiển xuất ra của 8051, nó cho phép phân kênh bus địa chỉ và bus dữ liệu của Port 0.

√ Chân 31 (EA : Eternal Acess) được đưa xuống thấp cho phép chọn bộ nhớ mã ngoàiđối với 8031.

Đối với 8051 thì :

- EA = 5V : Chọn ROM nội.
- EA = 0V : Chọn ROM ngoại.
- EA = 21V : Lập trình EPROM nội.

√ Các chân từ 32÷39 là Port 0 (P0.0, P0.1, . . . , P0.7) dùng cả hai mục đích : Vừa làm byte thấp cho bus địa chỉ, vừa làm bus dữ liệu, nếu vậy Port 0 không còn chức năng xuất nhập I/O nữa.

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

3. TỔ CHỨC BỘ NHỚ

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 (mặc định cho R0 -R7)							
00								

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
88	87	86	85	84	83	82	81	80	P0

CÁC THANH GHI CHỨC NĂNG ĐẶC BIỆT

Tóm tắt bộ nhớ dữ liệu trên chip.

3.1 RAM mục đích chung

Trong bản đồ bộ nhớ trên, 80 byte từ địa chỉ 30H÷7FH là RAM mục đích chung. Kể cả 32byte phần dưới từ 00H÷2FH cũng có thể sử dụng giống như 80 byte ở trên, tuy nhiên 32 byte còn có mục đích khác sẽ đề cập sau.

Bất kỳ vị trí nào trong RAM mục đích chung cũng có thể được truy xuất tùy ý giống như việc sử dụng các mode để định địa chỉ trực tiếp hay gián tiếp. Ví dụ để đọc nội dung của RAM nội có địa chỉ 5FH vào thanh ghi tích lũy thì ta dùng lệnh : MOV A, 5FH.

RAM nội cũng được truy xuất bởi việc dùng địa chỉ gián tiếp qua R0 và R1. Hai lệnh sau đây sẽ tương đương lệnh trên :

```
MOV R0, #5FH
MOV A, @R0
```

Lệnh thứ nhất dùng sự định vị tức thời để đưa giá trị 5FH vào thanh ghi R0, lệnh thứ hai dùng sự định vị gián tiếp để đưa dữ liệu "đã được trỏ đến bởi R0" vào thanh ghi tích lũy A.

3.2 RAM định vị

8051 chứa 210 vị trí có thể định vị bit, trong đó có 128 bit nằm ở các địa chỉ từ 20H÷2FH và phần còn lại là các thanh ghi chức năng đặc biệt.

3.3 Các băng thanh ghi (Register Banks)

32 vị trí nhớ cuối cùng của bộ nhớ từ địa chỉ byte 00H÷1FH chức các dãy thanh ghi. Tập hợp các lệnh của 8051 cung cấp 8 thanh ghi từ R0÷R7 ở địa chỉ 00H÷07H nếu máy tính mặc nhiên chọn để thực thi. Những lệnh tương đương dùng sự định vị trực tiếp. Những giá trị dữ liệu được dùng thường xuyên chắc chắn sẽ sử dụng một trong các thanh ghi này.

3.4 Các thanh ghi chức năng đặc biệt (Special Function Register)

Có 21 thanh ghichức năng đặc biệt SFR ở đỉnh của RAM nội từ địa chỉ các thanh ghi chức năng đặc biệt được định rõ, còn phần còn lại không định rõ.

Mặc dù thanh ghi A có thể truy xuất trực tiếp, nhưng hầu hết các thanh ghi chức năng đặc biệt được truy xuất bằng cách sử dụng sự định vị địa chỉ trực tiếp. Chú ý rằng vài thanh ghi SFR có cả bit định vị và byte định vị. Người thiết kế sẽ cẩn thận khi truy xuất bit mà không truy xuất byte.

3.4.1 Từ 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

a) Cờ Carry CY (Carry Flag) :

Cờ Carry được set lên 1 nếu có sự tràn ở bit 7 trong phép cộng hoặc có sự mượn vào bit 7 trong phép trừ.

Cờ Carry cũng là 1 "thanh ghi tích lũy luận lý", nó được dùng như một thanh ghi 1 bit thực thi trên các bit bởi những lệnh luận lý. Ví dụ lệnh : ANL C, 25H sẽ AND bit 25H với cờ Carryvà cất kết quả vào cờ Carry.

b) Cờ Carry phụ AC (Auxiliary Carry Flag) :

Khi cộng những giá trị BCD (Binary Code Decimal), cờ nhớ phụ AC đượcset nếu có sự tràn từ bit 3 sang 4 hoặc 4 bit thấp nằm trong phạm vi0AH÷0FH.

c) Cờ 0 (Flag 0) :

Cờ 0 (F0) là bit cờ có mục đích tổng hợp cho phép người ứng dụng dùng nó.

d). Những bit chọn dãy thanh ghi RS1 và RS0 :

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.

e. Cờ tràn OV (Over Flag) :

Cờ tràn được set sau một hoạt động cộng hoặc trừ nếu có sự tràn toán học. Bit OV được bỏ qua đối với sự cộng trừ không dấu. Khi cộng trừ có dấu, kết quả lớn hơn + 127 hay nhỏ hơn -128 sẽ set bit OV.

f. 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.

3.4.2 Thanh ghi B :

Thanh ghi B ở địa chỉ F0H được dùng đi đôi với thanh ghi A cho các hoạt động nhân chia.

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.

3.4.3 Con trỏ Stack SP (Stack Pointer) :

Stack Pointer là một thanh ghi 8 bit ở địa chỉ 81H. Nó chứa địa chỉ của dữ liệu đang hiện hành trên đỉnh Stack. Các hoạt động của Stack bao gồm việc đẩy dữ liệu vào Stack (PUSH) và lấy dữ liệu ra khỏi Stack (POP).

√ Việc PUSH vào Stack sẽ tăng SP lên 1 trước khi dữ liệu vào.

√ Việc POP từ Stack ra sẽ lấy dữ liệu ra trước rồi giảm SP đi 1.

3.4.4 Con trỏ dữ liệu DPTR (Data Pointer) :

Data Pointer được để truy xuất bộ nhớ mà ngoài hoặc bộ nhớ dữ liệu ngoài, nó là một thanh ghi 16 bit mà byte thấp là DPL ở địa chỉ 82H còn byte cao là DPH ở địa chỉ 83H. Để đưa nội dung 55H vào RAM ngoại có địa chỉ 1000H ta dùng 3 lệnh sau :

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

Lệnh thứ nhất dùng sự định vị trực tiếp đưa hằng số dữ liệu 55H vào A. Lệnh thứ hai cũng tương tự lệnh thứ nhất đưa hằng số dữ liệu 1000H vào

trong DPTR . lệnh cuối cùng dùng sự định vị gián tiếp để dịch chuyển giá trị 55H trong A vào vùng nhớ RAM ngoại 1000H nằm trong DPTR.

3.4.5 Các thanh ghi Port (Port Register) :

Các Port 0, Port 1, Port 2, Port 3 có địa chỉ tương ứng 80H, 90H, A0H, B0H. Các Port 0, Port 1, Port 2, Port 3 không còn tác dụng xuất nhập nữa nếu bộ nhớ ngoài được dùng hoặc một vài cá tính đặc biệt của 8051 được dùng (như Interrupt, Port nối tiếp . . .). Do vậy chỉ còn có Port1 có tác dụng xuất nhập I/O.

Tất cả các Port đều có bit địa chỉ, do đó nó có khả năng giao tiếp với bên ngoài mạnh mẽ.

3.4.6 Các thanh ghi Timer (Timer Register) :

8051 có 2 bộ : Một bộ Timer 16 bit và một bộ Counter 16 bit, hai bộ này dùng để định giờ lúc nghỉ của chương trình hoặc đếm các sự kiện quan trọng. Timer 0 có bit thấp TL0 ở địa chỉ 8AH và có bit cao TH0 ở địa chỉ 8CH. Timer 1 có bit thấp ở địa chỉ 8BH và bit cao TH1 ở địa chỉ 8DH.

Hoạt động định thời được cho phép bởi thanh ghi mode định thời TMOD (Timer Mode Register). Ở địa chỉ 89H và thanh ghi điều khiển định thời TCON (Timer Control Register) ở địa chỉ 88H. Chỉ có TCON có bit định vị.

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

8051 chứa một Port nối tiếp trên Chip cho việc truyền thông tin với những thiết bị nối tiếp như là những thiết bị đầu cuối, modem, hoặc để giao tiếp IC khác với những bộ biến đổi A/D, những thanh ghi di chuyển, RAM . . .). Thanh ghi đệm dữ liệu nối tiếp SBUF ở địa chỉ 99H giữ cả dữ liệu phát lẫn dữ liệu thu. Việc ghi lên SBUF để LOAD dữ liệu cho việc truyền và đọc SBUF để truy xuất dữ liệu cho việc nhận những mode hoạt động khác nhau được lập trình thông qua thanh ghi điều khiển Port nối tiếp SCON.

3.4.8 Các thanh ghi ngắt (Interrupt Register) :

8051 có hai cấu trúc ngắt ưu tiên, 5 bộ nguồn. Những Interrupt bị mất tác dụng sau khi hệ thống reset (bị cấm) và sau đó được cho phép bởi việc cho phép ghi lên thanh ghi cho phép ngắt IE (Interrup Enable Register) ở địa chỉ A8H. Mức ưu tiên được đặt vào thanh ghi ưu tiên ngắt IP (Interrupt Priority Level) tại địa chỉ B8H. Cả 2 thanh ghi trên đều có bit địa chỉ.

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

Thanh ghi PCON không có bit định vị. Nó ở địa chỉ 87H bao gồm các bit địa chỉ tổng hợp. Các bit 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 1 của cờ đa năng.

√ Bit 2 (GF0) : Bit 2 của cờ đa năng.

√ 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.

II. TÓM TẮT TẬP LỆNH CỦA 8051

Các chương trình được cấu tạo từ nhiều lệnh, chúng được xây dựng logic, sự nối tiếp của các lệnh được nghĩ ra một cách hiệu quả và nhanh, kết quả của chương trình thì khả quan.

Tập lệnh họ MSC-51 được sự kiểm tra của các mode định vị và các lệnh của chúng có các Opcode 8 bit. Điều này cung cấp khả năng 2^8=256 lệnh được thi hành và một lệnh không được định nghĩa. Vài lệnh có 1 hoặc 2 byte bởi dữ liệu hoặc địa chỉ thêm vào Opcode. Trong toàn bộ các lệnh có 139 lệnh 1 byte, 92 lệnh 2 byte và 24 lệnh 3 byte.

1. CÁC CHẾ ĐỘ ĐỊNH VỊ (ADDRESSING MODE)

Các mode định vị là một bộ phận thống nhất của tập lệnh mỗi máy tính. Chúng cho phép định rõ nguồn hoặc nơi gởi tới của dữ liệu ở các đường khác nhau tùy thuộc vào trạng thái của lập trình. 8051 có 8 mode định vị được dùng như sau :

√ Thanh ghi.

√ Trực tiếp.

√ Gián tiếp.

√ Tức thời.

√ Tương đối.

√ Tuyệt đối.

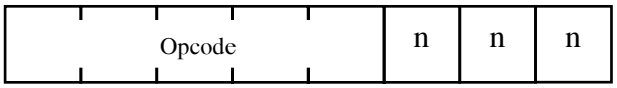
√ Dài.

√ Định vị.

1.1 Sự định vị thanh ghi (Register Addressing)

Có 4 dãy thanh ghi 32 byte đầu tiên của RAM dữ liệu trên Chip địa chỉ 00H ÷ 1FH, nhưng tại một thời điểm chỉ có một dãy hoạt động các bit PSW3, PSW4 của từ trạng thái chương trình sẽ quyết định dãy nào hoạt động.

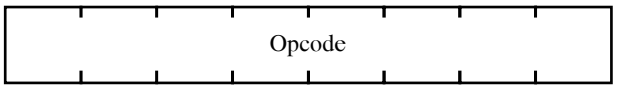
Các lệnh để định vị thanh ghi được ghi mật mã bằng cách dùng bit trọng số thấp nhất của Opcode lệnh để chỉ một thanh ghi trong vùng địa chỉ theo logic này. Như vậy 1 mã chức năng và địa chỉ hoạt động có thể được kết hợp để tạo thành một lệnh ngắn 1 byte như sau :



Register Addressing.

1.2 Sự định địa chỉ trực tiếp (Direct Addressing)

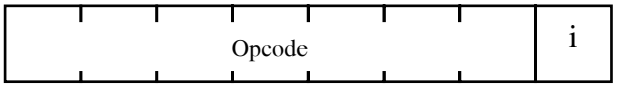
Sự định địa chỉ trực tiếp có thể truy xuất bất kỳ giá trị nào trên Chip hoặc thanh ghi phần cứng trên Chip. Một byte địa chỉ trực tiếp được đưa vào Opcode để định rỏ vị trí được dùng như sau :



Tùy thuộc các bit bậc cao của địa chỉ trực tiếp mà một trong 2 vùng nhớ được chọn. Khi bit 7 = 0, thì địa chỉ trực tiếp ở trong khoảng 0÷127 (00H÷7FH) và 128 vị trí nhớ thấp của RAM trên Chip được chọn.

1.3 Sự định vị địa chỉ gián tiếp (Indirect Addressing)

Sự định địa chỉ gián tiếp được tượng trưng bởi ký hiệu @ được đặt trước R0, R1 hay DPTR. R0 và R1 có thể hoạt động như một thanh ghi con trỏ mà nội dung của nó cho biết một địa chỉ trong RAM nội ở nơi mà dữ liệu được ghi hoặc được đọc. Bit có trọng số nhỏ nhất của Opcode lệnh sẽ xác định R0 hay R1 được dùng con trỏ Pointer.



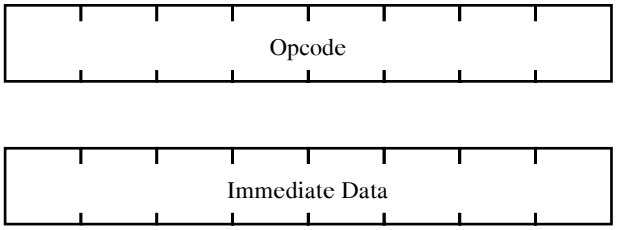
Ví dụ để đưa nội dung 60 H vào RAM nội tại địa chỉ 50H ta làm như sau:

MOV R1,#50H

MOV @R1,60H

1.4. Sự định vị địa chỉ tức thời (Immediate Addressing)

Sự định địa chỉ tức thời được tượng trưng bởi ký hiệu # được đứng trước một hằng số, 1 biến ký hiệu hoặc một biểu thức số học được sử dụng bởi các hằng, các ký hiệu, các hoạt động do người điều khiển. Trình biên dịch tính toán giá trị và thay thế dữ liệu tức thời. Byte lệnh thêm vô chứa trị số dữ liệu tức thời như sau :



Ví dụ :

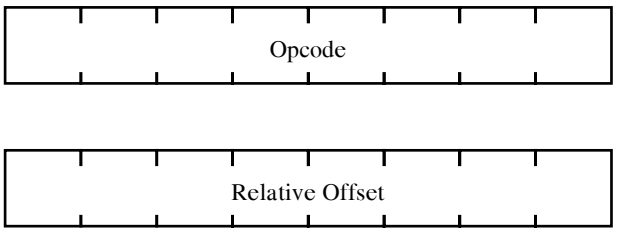
MOV A, # 12 ⇐ Đưa trực tiếp số thập phân 12 vào A.

MOV A, # 10 ⇐ Đưa trực tiếp số Hex 10H (16D) vào A.

MOV A, # 00010001B ⇐ Đưa trực tiếpsố nhị phân này vào A.

1.5 Sự định vị địa chỉ tương đối

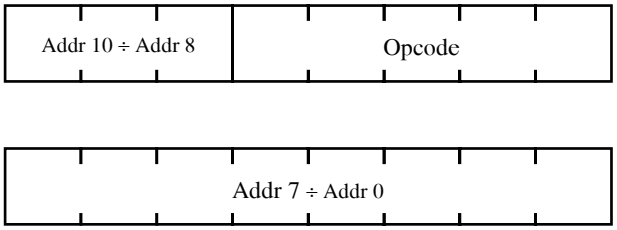
Sự định địa chỉ tương đối chỉ sử dụng với những lệnh nhảy nào đó. Một địa chỉ tương đối (hoặc Offset) là một giá trị 8 bit mà nó được cộng vào bộ đếm chương trình PC để tạo thành địa chỉ một lệnh tiếp theo được thực thi. Phạm vi của sự nhảy nằm trong khoảng -128 ÷ 127. Offset tương đối được gắn vào lệnh như một byte thêm vào như sau :



Sự định vị tương đối đem lại thuận lợi cho việc cung cấp mã vị trí độc lập, nhưng bất lợi là chỉ nhảy ngắn trong phạm vi -128÷127 byte.

1.6 Sự định địa chỉ tuyệt đối (Absolute Addressing)

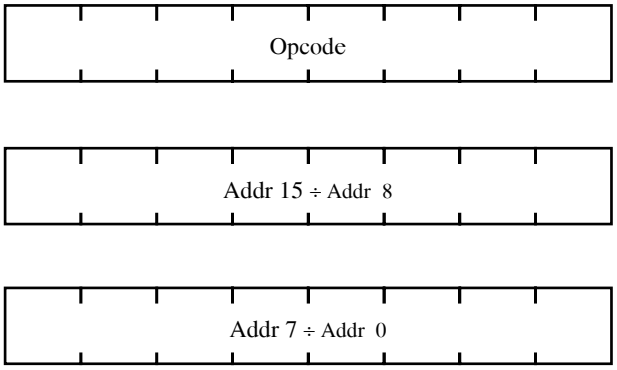
Sự định địa chỉ tuyệt đối được dùng với các lệnh ACALL và AJMP. Các lệnh 2 byte cho phép phân chia trong trang 2K đang lưu hành của bộ nhớ mã của việc cung cấp 11 bit thấp để xác định địa chỉ trong trang 2K (A0÷A10 gồm A10÷A8 trong Opcode và A7÷A0 trong` byte)và 5 bit cao để chọn trang 2K (5 bit cao đang lưu hành trong bộ đếm chương trình là 5 bit Opcode).



Sự định vị tuyệt đối đem lại thuận lợi cho các lệnh ngắn (2 byte), nhưng bất lợi trong việc giới hạn phạm vi nơi gởi đến và cung cấp mã có vị trí độc lập.

1.7 Sự định vị địa chỉ dài (Long Addressing)

Sự định vị dài được dùng với lệnh LCALL và LJMP. Các lệnh 3 byte này bao gồm một địa chỉ nơi gởi tới 16 bit đầy đủ là 2 byte và 3 byte của lệnh.



Ưu điểm của sự định dài là vùng nhớ mã 64K có thể được dùng hết, nhược điểm là các lệnh đó dài 3 byte và vị trí lệ thuộc. Sự phụ thuộc vào vị trí sẽ bất lợi bởi chương trình không thể thực thi tại địa chỉ khác.

1.8 Sự định địa chỉ phụ lục (Index Addressing)

Sự định địa chỉ phụ lục dùng một thanh ghi cơ bản (cũng như bộ đếm chương trình hoặc bộ đếm dữ liệu) và Offset (thanh ghiA) trong sự hình thành 1 địa chỉ liên quan bởi lệnh JMP hoặc MOVC.

Index Addressing.

2. CÁC KIỂU LỆNH (INSTRUCTION TYPES)

8051 chia ra 5nhóm lệnh chính :

√ Các lệnh số học.

√ Lệnh logic.

√ Dịch chuyển dữ liệu.

√ Lý luận.

√ Rẽ nhánh chương trình.

Từng kiểu lệnh được mô tả như sau :

2.1 Các lệnh số học (Arithmetic Instrustion) :

ADD A, <src, byte>

ADD	A, Rn	: (A) ← (A) + (Rn)
ADD	A, direct	: (A) ← (A) + (direct)
ADD	A, @ Ri	: (A) ← (A) + ((Ri))
ADD	A, # data	: (A) ← (A) + # data
ADDC	A, Rn	: (A) ← (A) + (C) + (Rn)
ADDC	A, direct	: (A) ← (A) + (C) + (direct)
ADDC	A, @ Ri	: (A) ← (A) + (C) + ((Ri))
ADDC	A, # data	: (A) ← (A) + (C) + # data

SUBB A, <src, byte>

SUBB	A, Rn	: (A) ← (A) - (C) - (Rn)
SUBB	A, direct	: (A) ← (A) - (C) - (direct)
SUBB	A, @ Ri	: (A) ← (A) - (C) - ((Ri))
SUBB	A, # data	: (A) ← (A) - (C) - # data

INC <byte>

INC	A	: (A) ← (A) + 1
INC	direct	: (direct) ← (direct) + 1
INC	Ri	: ((Ri)) ← ((Ri)) + 1
INC	Rn	: (Rn) ← (Rn) + 1

INC	DPTR	: (DPTR) ← (DPTR) + 1

DEC <byte>

DEC	A	: (A) ← (A) - 1
DEC	direct	: (direct) ← (direct) - 1
DEC	@Ri	: ((Ri)) ← ((Ri)) - 1
DEC	Rn	: (Rn) ← (Rn) - 1
MULL	AB	: (A) ← LOW [(A) x (B)] ; có ảnh hưởng cờ OV
		: (B) ← HIGH [(A) x (B)] ; cờ Cary được xóa.
DIV	AB	: (A) ← Integer Result of [(A)/(B)]; cờ OV
		: (B) ← Remainder of [(A)/(B)]; cờ Carry xóa
DA	A	: Điều chỉnh thanh ghi A thành số BCD đúng trong phép cộng BCD (thường DA A đi kèm với ADD, ADDC)

√ Nếu [(A3-A0)>9] và [(AC)=1] ⇐ (A3÷A0) ← (A3÷A0) + 6.

√ Nếu [(A7-A4)>9] và [(C)=1] ⇐ (A7÷A4) ← (A7÷A4) + 6.

2.2 Các hoạt động logic (Logic Operation) :

Tất cả các lệnh logic sử dụng thanh ghi A như là một trong những toán hạng thực thi một chu kỳ máy, ngoài A ra mất 2 chu kỳ máy. Những hoạt động logic có thể được thực hiện trên bất kỳ byte nào trong vị trí nhớ dữ liệu nội mà không qua thanh ghi A.

Các hoạt động logic được tóm tắt như sau :

ANL <dest - byte> <src - byte>

ANL	A, Rn	: (A) ← (A) AND (Rn).
ANL	A, direct	: (A) ← (A) AND (direct).
ANL	A,@ Ri	: (A) ← (A) AND ((Ri)).
ANL	A, # data	: (A) ← (A) AND (# data).
ANL	direct, A	: (direct) ← (direct) AND (A).
ANL	direct, # data	: (direct) ← (direct) AND # data.

ORL <dest - byte> <src - byte>

ORL	A, Rn	: (A) ← (A) OR (Rn).
ORL	A, direct	: (A) ← (A) OR (direct).
ORL	A,@ Ri	: (A) ← (A) OR ((Ri)).
ORL	A, # data	: (A) ← (A) OR # data.
ORL	direct, A	: (direct) ← (direct) OR (A).
ORL	direct, # data	: (direct) ← (direct) OR # data.

XRL <dest - byte> <src - byte>

XRL	A, Rn	: (A) ← (A)) (Rn).
XRL	A, direct	: (A) ← (A)) (direct).
XRL	A,@ Ri	: (A) ← (A)) ((Ri)).
XRL	A, # data	: (A) ← (A)) # data.
XRL	direct, A	: (direct) ← (direct)) (A).
XRL	direct, # data	: (direct) ← (direct)) # data.

y = a) b = ab + ab

CLR	A	: (A) ← 0
CLR	C	: (C) ← 0
CLR	Bit	: (Bit) ← 0
RL	A	: Quay vòng thanh ghi A qua trái 1 bit
		(An + 1) ← (An); n = 0÷6
		(A_0) ← (A_7)
RLC	A	: Quay vòng thanh ghi A qua trái 1 bit có cờ Carry
		(An + 1) ← (An); n = 0÷6
		(C) ← (A_7)
		(A_0) ← (C)
RR	A	: Quay vòng thanh ghi A qua phải 1 bit
		(An + 1) → (An); n = 0÷6
		(A_0) → (A_7)

RRC A : Quay vòng thanh ghi A qua phải 1 bit có cờ Carry

$(An + 1) \rightarrow (An)$; n = 0÷6

$(C) \rightarrow (A_7)$

$(A_0) \rightarrow (C)$

SWAP A : Đổi chổ 4 bit thấp và 4 bit cao của A cho nhau (A3÷A0)√(A7÷A4).

2.3 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 rỏ địa chỉ, trình biên dịch sẽ đặt địa chỉ nơi cần nhảy tới vào đúng khẩu lệnh đã đưa ra.

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.

$(PC) \leftarrow (PC) + 2$

$(SP) \leftarrow (SP) + 1$

$((SP)) \leftarrow$ (PC7÷PC0)

$(SP) \leftarrow (SP) + 1$

$((SP)) \leftarrow$ (PC15÷PC8)

(PC10÷PC0) ← page Address.

LCALL addr16 : Lệnh gọi dài chương trình con trong 64K.

$(PC) \leftarrow (PC) + 3$

$(SP) \leftarrow (SP) + 1$

$((SP)) \leftarrow$ (PC7÷PC0)

$(SP) \leftarrow (SP) + 1$

((SP)) ← (PC15÷PC8)

(PC) ← Addr15÷Addr0.

RET : Kết thúc chương trình con trở về chương trình chính.

(PC15÷PC8) ← (SP)

(SP) ← (SP) - 1

(PC7÷PC0) ← ((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

(PC10÷PC0) ← 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 (-128÷127) 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.

2.4 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 Stack của 8051 chỉ chứa 128 byte RAM nội, nếu con trỏ Stack 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 chuyễn 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 máyvà 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 tắy như sau :

MOV	A,Rn	: (A) ← (Rn)
MOV	A, direct	: (A) ← (direct)
MOV	A, @ Ri	: (A) ← ((Ri))
MOV	A, # data	: (A) ← # data
MOV	Rn, A	: (Rn) ← (A)
MOV	Rn, direct	: (Rn) ← (direct)
MOV	Rn, # data	: (Rn) ← # data

MOV	direct, A	: (direct) ← (A)
MOV	direct, Rn	: (direct) ← (Rn)
MOV	direct, direct	: (direct) ← (direct)
MOV	direct, @ Ri	: (direct) ← ((Ri))
MOV	direct, # data	: (direct) ← data
MOV	@ Ri, A	: ((Ri)) ← (A)
MOV	@ Ri, direct	: ((Ri)) ← (direct)
MOV	@ Ri, # data	: ((Ri)) ← # data
MOV	DPTR, # data16	: (DPTR) ← # data16
MOV	A, @ A + DPTR	: (A) ← (A) + (DPTR)
MOV	@ A + PC	: (PC) ← (PC) + 1
		(A) ← (A) + (PC)
MOVX	A, @ Ri	: (A) ← ((Ri))
MOVX	A, @ DPTR	: (A) ← ((DPTR))
MOVX	@ Ri, A	: ((Ri)) ← (A)
MOVX	@ DPTR, A	: ((DPTR)) ← (A)
PUSH	direct	: Cất dữ liệu vào Stack
		(SP) ← (SP) + 1
		(SP) ← (Drirect)
POP	direct	: Lấy từ Stack 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))
		(A3÷A0) √ ((Ri3÷Ri0))

2.5 Các lệnh luận lý (Boolean Instruction) :

8051 chứa một bộ xử lý luận lý đầy đủ cho các hoạt động bit đơn, đây là một điểm mạnh của họ vi điều khiển MSC-51 mà các họ vi xử lý khác không có.

RAM nội chứa 128 bit đơn vị và các vùng nhớ các thanh ghi chức năng đặc biệt cấp lên đến 128 đơn vị khác. Tất cả các đường Port là bit định vị, mỗi đường có thể được xử lý như Port đơn vị riêng biệt. Cách truy xuất các bit này không chỉ các lệnh rẽ nhánh không, mà là một danh mục đầy đủ các lệnh MOVE, SET, CLEAR, COMPLEMENT, OR, AND.

Toàn bộ sư truy xuất của bit dùng sự định vị trực tiếp với những địa chỉ từ 00H÷7FH trong 128 vùng nhớ thấp và 80H÷FFH ở các vùng thanh ghi chức năng đặc biệt.

Bit Carry C trong thanh ghi PSW của từ trạng thái chương trình và được dùng như một sự tích lũy đơn của bộ xử lý luận lý. Bit Carry cũng là bit định vị và có địa chỉ trực tiếp vì nó nằm trong PSW. Hai lệnhCLR C và CLR CY đều có cùng tác dụng là xóa bit cờ Carry nhưng lệnh này mất 1 byte còn lệnh sau mất 2 byte.

Hoạt động của các lệnh luận lý được tóm tắt như sau :

CLR	C	: Xóa cờ Carry xuống 0. Có ảnh hưởng cờ Carry.
CLR	BIT	: Xóa bit xuống 0. Không ảnh hưởng cờ Carry
SET	C	: Set cờ Carry lên 1. Có ảnh hưởng cờ Carry.
SET	BIT	: Set bit lên 1. Không ảnh hưởng cờ Carry.
CPL	C	: Đảo bit cờ Carry. Có ảnh hưởng cờ Carry.
CPL	BIT	: Đảo bit. Không ảnh hưởng cờ Carry.
ANL	C, BIT	: (C) ← (C) AND (BIT) : Có ảnh hưởng cờ Carry.
ANL	C, /BIT	: (C) ← (C) AND NOT (BIT):Không ảnh hưởng cờ Carry.
ORL	C, BIT	: (C) ← (C) OR (BIT) : Tác động cờ Carry.
ORL	C, /BIT	: (C) ← (C) OR NOT (BIT) : Tác động cờ Carry.
MOV	C, BIT	: (C) ← (BIT) : Cờ Carry bị tác động.
MOV	BIT, C	: (BIT) ← (C) : Không ảnh hưởng cờ Carry.

III. HOẠT ĐỘNG CỦA PORT NỐI TIẾP 8051.

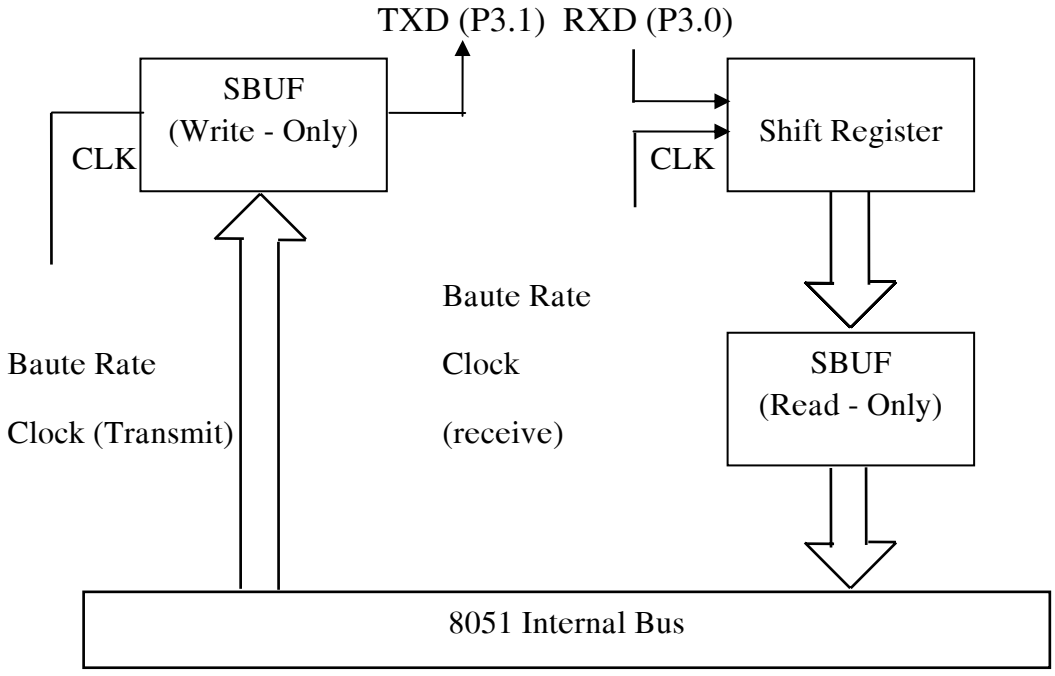
1. GIỚI THIỆU

Port nối tiếp của 8051 có thể hoạt động trong các mode riêng biệt trên phạm vi cho phép của tần số. Chức năng chủ yếu của Port nối tiếp là thực hiện sự chuyễn đổi song song thành nối tiếp cho dữ liệu ra và sự chuyễn đổi nối tiếp thành song song cho dữ liệu vào.

Phần cứng truy xuất tới Port nối tiếp qua các chân TXD (P3.1) và RXD (P3.0).

Port nối tiếp tham dự hoạt động đầy đủ (sự phát và thu cùng lúc), và thu vào bộ đệm mà nó cho phép1 ký tự nhận vào và được cất ở bộ đệm trong khi kí tự thứ hai được nhận vào. Nếu CPU đọc ký tự thứ nhất trước khi kí tự thứ hai được nhận vào hoàn toàn thì dữ liệu không bị mất.

Hai thanh ghi chức năng đặc biệt cung cấp cho phần mềm truy xuất đến Port nối tiếp là SBUF và SCON. Sự đệm Port nối tiếp (SBUF) ở địa chỉ 99H là 2 sự đệm thật sự : Ghi lên SBUF LOAD dữ liệu phát và đọc SBUF truy xuất dữ liệu đã nhận. Đây là hai thanh ghi riêng biệt và rõ rệt, mà thanh ghi phát chỉ ghi còn thanh ghi thu chỉ đọc. Sơ đồ khối của Port nối tiếp như sau :



Serial Port Block Dragram

Thanh ghi điều khiển Port nối tiếp SCON (98H) là thanh ghi được định vị bit bao gồm các trạng thái và các bit điều khiển. Các bit điều khiển set mode của Port nối tiếp, còn các bit trạng thái cho biết sự kết thúc việc thu phát 1 kí tự. Các bit trạng thái có thể được kiểm tra trong phần mềm hoặc có thể lập trình để sinh ra sự ngắt.

Tần số hoạt động của Port nối tiếp hoặc tốc độ BAUD có thể được lấy từ dao động trên Chip 8051 hoặc thay đổi. Nếu một tốc độ Baud thay đổi được

dùng, thì Timer cung cấp 1 tốc độ Baud ghi giờ và phải được lập trình một cách phù hợp.

2. THANH GHI ĐIỀU KHIỂN PORT NỐI TIẾP SCON (SERIAL PORT CONTROL REGISTER)

Mode hoạt động của Port nối tiếp 8051 được set bởiviệc ghi lên thanh ghi mode của Port nối tiếp SCON ở địa chỉ 99H. Bảng tóm tắt thanh ghi điều khiển Port nối tiếp SCON như sau :

Bit	Ký hiệu	Địa chỉ	Mô tả hoạt động
SCON.7	SM0	9FH	Bit 0 của mode Port nối tiếp.
SCON.6	SM1	9EH	Bit 1 của mode Port nối tiếp.
SCON.5	SM2	9DH	Bit 2 của mode Port nối tiếp. Cho phép sự truyền của bộ xử lý đa kênh ở mode 2 và 3; RI sẽ không tích cực nếu bit thứ 9 đã thu vào là 0.
SCON.4	REN	9CH	REN = 1 sẽ cho thu kí tự.
SCON.3	TB8	9BH	Phát bit 8. Bit 9 phát trong mode 2 và 3, nó được set hoặc xóa bởi phần mềm.
SCON.2	RB8	9AH	Thu bit 8, bit 9 thu.
SCON.1	TI	99H	Cờ ngắt phát. Được set khi kết thúc sự truyền kí tự và được xóa bởi phần mềm.
SCON.0	RI	98H	Cờ ngắt thu. Được set khi kết thúc sự thu và được xóa bởi phần mềm.

SCON Register sumary.

3. CÁC MODE HOẠT ĐỘNG (MODE OF OPERATION)

SM0	SM1	MODE	MÔ TẢ	TỐC ĐỘ BAUD
0	0	0	Thanh ghi dịch	Cố định (tần số dao động 1÷12).
0	1	1	URAT8 bit	Thay đổi (được đặt bởiTimer).
1	0	2	URAT 8 bit	Cố định (tần số dao động ÷12 c ÷16)

1	1	3	URAT 8 bit	Thay đổi (được đặt bởi Timer).

Serial Port Modes.

Trước khi dùng Port nối tiếp, SCON phải được gán đúng mode. Ví dụ để khởi gán Port nối tiếp MODE 1 (SM0/SM1 = 0/1), cho phép thu (REN =1), và set cờ ngắt của việc phát sẵn sàng hoạt động (TI = 1), ta dùng lệnh sau : MOV SCON, # 01010010H.

Port nối tiếp của 8051 có 4 mode hoạt động tùy thuộc theo 4 trạng thái của SM0/SM1.

Ba trong 4 mode cho phép truyền sự đồng bộ với mỗi kí tự thu hoặc phát sẽ được bố trí bởi bit Start hoặc bit Stop.

4. SỰ KHỞI ĐỘNG, TRUY XUẤT CÁC THANH GHI PORT NỐI TIẾP

4.1. Sự cho phép bộ thu (Recive Enable) :

Bit cho phép thu REN trong thanh ghi SCON phải được set bởi phần mềm để cho phép sự thu các ký tự. Điều này thường được làm ở đầu chương trình khi các Port nối tiếp và các Timer . . . được khởi động.

Ta có thể động bằng lệnh :

SETB REN hoặc MOV CON, # XXX1XXXXB

4.2. Bit data thứ 9 (the 9^{th} data bit) :

Bit data thứ 9 được phát trong mode 2 và mode 3 phải được LOAD vào TB8 bởi phần mềm, còn bit data thứ 9 được thu thì đặt trong RB8.

Phần mềm có thể (hoặc không) đòi hỏi một bit data thứ 9 tham gia vào những chi tiết kỹ thuật của thiết bị nối tiếp với điều kiện màsự truyền data được thành lập.

4.3. Sự thêm vào bit kiểm tra chẳn lẻ Parity :

Cách tổng quát dùng chung bit data thứ 9 là cộng bit Parity vào một ký tự .

Bit P (Parity) trong từ trạng thái chương trình PSW sẽ được set hoặc xóa với mọi chu kỳ máy để thành lập bit Parity chẳn với 8 bit trong thanh ghi tích lũyA

Ví dụ nếu sự truyền yêu cầu 8 bit data cộng thêm 1 bit Parity chẵn, thì các lệnh sau đây có thể được dùng để phát 8 bit vào thanh ghi A với Parity chẵn được cộng vào bit thứ 9.

```
MOV  C, P
MOV  TB8,  C
MOV  SBUF, A
```

Nếu Parity lẻ được yêu cầu thì các lệnh trên được sửa lại là:

```
MOV  C, P
CPL   C
MOV  TB8, C
MOV  SBUF, A
```

Việc dùng bit Parity không bị giới hạn trong mode 2 hoặc mode 3. 8 bit data được phát trong mode 1 có thể bao gồm 7 bit data, và 1 bit Parity. Để phát 1 mã ASCII 7 bit với 1 bit Parity chẵn vào 8 bit, các lệnh sau đây được dùng :

```
CLR   ACC,7        : Xóa bit MSB trong A đảm bảo Parity chẵn.
MOV  C, P          : Đưa Parity chẵn vào C
MOV  ACC.7, C      : Đưa Parity chẵn vào bit SB của A
MOV  SBUF, A       : Gởi bit data cùng bit Parity chẵn
```

4.4. Cờ ngắt :

Cờ ngắt thu RI và phát TI trong thanh ghi SCON vận hành 1 rơle quan trọng trong sự truyền nối tiếp 8051. Cả hai bit đều được set bởi phần cứng nhưng phải xóa bởi phần mềm.

Điển hình là RI được set ở cuối sự thu ký tự và cho biết : thanh ghi đệm thu đã đầy". Điều kiện này có thể kiểm tra trong phần mềm hoặc có thể được lập trình để sinh ra sự ngắt. Nếu phần mềm muốn nhập một ký tự từ một thiết bị đã được kết nối đến Port nối tiếp, thì nó phải chờ đến khi RI được set, sau khi xóa RI và đọc ký tự từ SBUF. Điều này được lập trình như sau :

```
WAIT :
        JNB    RI, WAIT     : Kiểm tra RI có set chưa.
        CLR    RI           : Xóa cờ ngắt thu RI
        MOV  A, SBUF        : CPU đọc ký tự
```

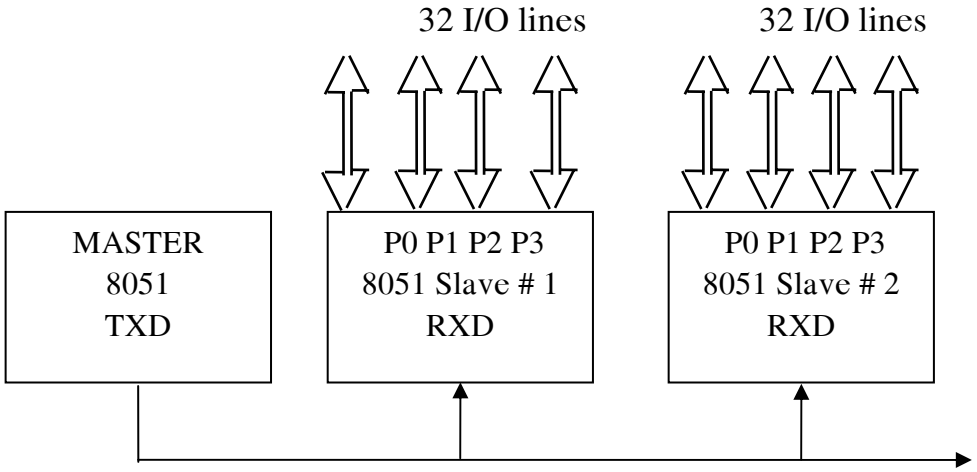
TI được set ở cuối sự phát ký tự và cho biết "thanh ghi đệm của sự phát đã rỗng". Nếu phần mềm muốn gởi 1 ký tự đến một thiết bị đã được kết nối

đến Port nối tiếp, trước tiên nó phải kiểm tra xem Port nối tiếp đã sẵn sàng chưa. Nếu ký tự trước đã được gởi đi, thì nó phải chờ cho đến khi sự phát đi hoàn thành. Các lệnh sau đây dùng để phát một ký tự trong thanh ghi A ra:

```
WAIT :
        JNB   TI, WAIT     : Kiểm tra TI có set chưa.
        CLR   TI           : Xóa cờ ngắt thu TI
        MOV   A, SBUF      : CPU đọc ký tự
```

5. SỰ TRUYỀN CỦA BỘ XỬ LÝ ĐA KÊNH

Mode 2 và mode 3 có một sự cung cấp đặc biệt cho việc truyền đa kênh xử lý. Ở các mode này, 9 bit data được thu và bit thứ 9 đi vào RB8. Port có thể lập trình như điều mà bit Stop được thu, sự ngắt của Port chỉ được tích cực nếu RB8 =1. Đặc điểm này cho phép bởi việc set bit MS2 trong thanh ghi SCON. Ứng dụng này là một sự cài đặt mạng được dùng bởi nhiều 8051 ở sự sắp đặt máy chủ và máy con như sau :



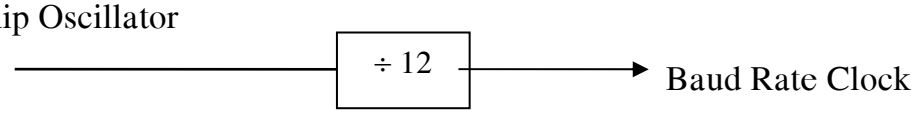
Multiprocessor Communication

Khi bộ xử lý chủ muốn phát một khối dữ liệu đến bộ xử lý con riêng lẻ, trước tiên nó gởi ra 1 byte địa chỉ để nhận diện bộ xử lý con mong muốn. Byte địa chỉ được phân biệt với byte dữ liệu bởi bit thứ 9 : bit thứ 9 bằng 1 trong byte địa chỉ và bằng 0 trong byte dữ liệu. Tuy nhiên byte địa chỉ sẽ ngắt toàn bộ các bộ xử lý con, do đó có thể khám phá byte đã thu để kiểm tra nếu nó đang định địa chỉ. Bộ xữ lý con đã được định địa chỉ sẽ xóa bit SM2 của nó và chuẩn bị thu các byte dữ liệu theo sau đó. Những bộ xử lý con không được định địa chỉ vẫn được giữ các bit SM2 của nó và set trở về các bận của chúng đồng thời lờ đi các byte dữ liệu đã thu thập. Chúng sẽ được ngắt lại khi byte địa chỉ kế tiếp được phát bởi bộ xữ lý cũ.

Bit SM2 không có tác dụng trong mode 0 và trong mode 1 nó có thể được dùng để kiểm tra sự thích hợp của bit Stop. Trong sự thu mode 1, nếu SM2 = 0 thì sự ngắt thu sẽ không tích cực trừ khi bit Stop thích hợp được thu.

6. TỐC ĐỘ BAUD CỦA PORT NỐI TIẾP :

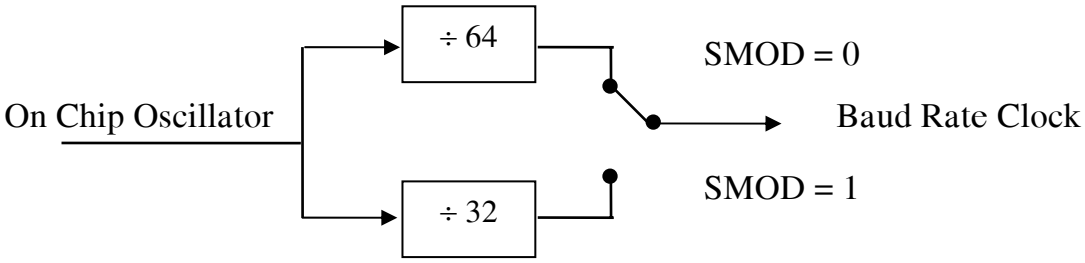
Tốc độ Baud của Port nối tiếp cố định ở mode 0 và mode 2. Trong mode 0 nó luôn luôn là tần số dao động trên Chip chia cho 12. Thông thường thạch anh 12 MHz lái dao động trên Chip 8051 nên tốc độ Baud của mode 0 là 1MHz.



MODE 0

Bằng sự mặc nhiên sau khi reset hệ thống, tốc độ Baud mode 2 là tần số dao động chia cho 64, tốc độ Baud cũng bị ảnh hưởng bởi bit SMOD của thanh ghi PCON.

Việc set bit SMOD sẽ tăng gấp đôi tốc độ Baud trong các mode1,2 và 3. Trong mode 2, tốc độ Baud có thể được gấp đôi từ giá trị mặc định 1/64 tần số/Chip (ứng SMOD = 0) lên đến 1/32 tần số dao động trên Chip (ứng với SMOD =1).



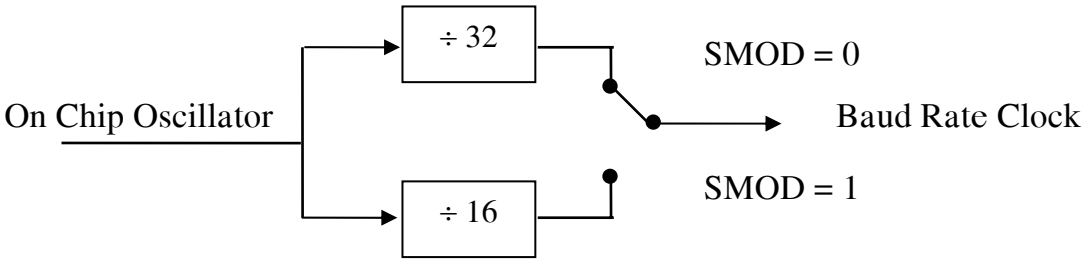
MODE 2

Bởi thanh ghi PCON không có bit định vị, nên để set bit SMOD mà không thay đổi các bit khác của thanh ghi PCON thì đòi hỏi phải có 1 hoạt động "đọc bổ sung ghi".

Các lệnh sau đây set bit SMOD :

```
MOV  A, PCON     : Nhập vào A giá trị hiện hành của PCON
SETB ACC, 7      : Set bit 7 của ACC (bit SMOD)
MOV  PCON, A     : Ghi giá trị trở về PCON mà SMOD đã được set.
```

Các tốc độ Baud trong mode 1 và mode 3 của 8051 được xác định bởi tốc độ tràn của Timer 1. Bởi vì Timer hoạt động ở tần số cao liên tục nên tràn xa hơn nữa được chia cho 32 (chia cho 16 nếu SMOD = 1) trước khi cung cấp xung clock tốc độ Baud đến Port nối tiếp. Tốc độ Baud ở mode 1 và 3 của 8051 được xác định bởi tốc độ tràn của Timer 1 hoặc Timer 2, hoặc cả 2.



MODE 1 and MODE 3

6.1 Dùng Timer 1 giống như sự đếm tốc độ Baud :

Muốn sinh ra tốc độ Baud, ta khởi gán TMOD ở mode tự động nạp 8 bit (mode 2 của Timer) và đặt giá trị Reload đúng vào byte cao của thanh ghi Timer 1 (TH1) để sinh ra tốc độ tràn chính xác cho tốc độ Baud. Có những tốc độ Baud rất chậm ta dùng mode 16 bit là mode 1 của Timer, nhưng ta phải khởi gán sau mỗi sự tràn cho TL1/TH1 trong thủ tục phục vụ ngắt ISR.

Hoạt động khác được đếm giờ bởi việc dùng Timer 1 ngoài là T1 (P3.5). công thức chung để xác định tốc độ Baud trong mode 1 và mode 3 là :

BAUD RATE = TIMER 1 OVERFLOW RATE÷32

Ví dụ một hoạt động 1200 Baud đòi hỏi một tốc độ tràn là 1200/32 = 38,4KHz. Nếu thạch anh 12 MHz lái dao động trên Chip, thì Timer 1 được đếm giờ ở tốc độ của tần số 1 MHz. Bởi vì Timer phải tràn ở tốc độ tần số 38,4 KHz và Timer đếm giờ ở tốc độ của tần số 1 MHz, nên một sự tràn được yêu cầu với 1000 : 38,4 = 26,04 clock (làm tròn 26). Bởi vì các Timer đếm lên và tràn trên sự chuyển đổi từ FFH $\rightarrow$ 00H của bộ đếm, nên 26 sự đếm thấp dưới 0 là giá trị Reload cần nạp cho TH1 (giá trị đúng là - 26). Ta dùng lệnh MOV TH1, # 26.

Ví dụ sau khởi động Port nối tiếp hoạt động giống như UART 8 bit ở tốc độ Baud 2400, dùng Timer 1 để cung cấp sự đếm giờ tốc độ Baud :

```
MOV     SCON, # 01010010B  : Port nối tiếp mode 1.
MOV     TMOD, # 20         : Timer 1 mode 2
```

```
MOV     TH1, # -13          : Nạp vào bộ đếm tốc độ 2400 Baud.
SETB    TR1                 : Start Timer 1.
```

Trong SCON có SM0/SM1 để vào mode UART 8 bit, REN = 1 cho phép Port nối tiếp thu các ký tự và TI = 1 cho phép phát ký tự đầu tiên bởi việc cho biết thanh ghi đếm rỗng. TMOD có M1/M0 = 1/0 để đặt Timer 1 vào mode tự động nạp 8 bit. Việc set bit TR1 để mở máy chạy Timer. Tốc độ Baud 2400 sẽ cho ta tốc độ tràn Timer 1 là 2400/32 = 76,8 KHz đồng thời Timer 1 được đếm giờ ở tốc độ của tần số 1000 KHz (ứng với thạch anh 12 MHz) sẽ cho số xung Clock sau mỗi sự tràn là 1000 : 76,8 = 13,02 (lấy tròn 13). Vậy - 13 là giá trị cần nạp vào TH1 để có tốc độ Baud là 2400 Baud.

Sau đây là bảng tóm tắt tốc độ Baud phổ biến ứng với 2 loại thạch anh 12 MHz và 11, 059 MHz :

Baud Rate	Crytal Frequency	SMOD	TH1 Reload Value	Actua Baud Rate	Error
9600	12MHz	1	- 7 (F9H)	8923	7%
2400	12MHz	0	-13 (F9H)	2404	0,16%
1200	12MHz	0	-26 (F9H)	1202	0%
19200	11,059MHz	1	-3 (F9H)	19200	0%
9600	11,059MHz	0	-3 (F9H)	9600	0%
2400	11,059MHz	0	-12 (F9H)	2400	0%
1200	11,059MHz	0	-24 (F9H)	1200	0%

Baud rate sumary.

IV. HOẠT ĐỘNG TIMER CỦA 8051 :

1. GIỚI THIỆU :

Bộ định thời của Timer là một chuổi 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 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. Gia trị nhị phân trong các FF của bộ Timer có thể được nghỉ 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.

Các Timer được ứng dụng thực tế cho các hoạt động định hướng. 8051 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 8051 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. THANH GHI MODE TIMER TMOD (TIMER MODE REGITER):

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.

3. THANH GHI ĐIỀU KHIỂN TIMER TCON (TIMER CONTROL REGISTER) :

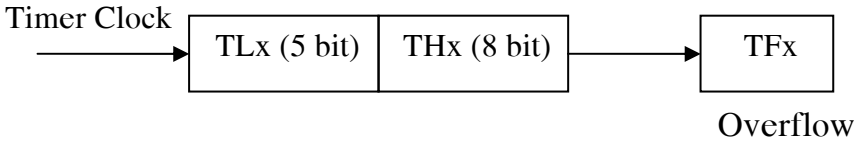
Thanh ghi điều khiển bao gồm các bit trạng thái và các bit điều khiển bởi Timer 0 và Timer 1. Thanh ghi TCON có bit định vị. Hoạt động của từng bit được tóm tắt như sau :

Bit	Symbol	Bit Address	Description
TCON.7	TF1	8FH	Cờ tràn Timer 1 được set bởi phần cứng ở sự tràn, được xóabởi phần mềm hoặc bởi phần cứng khi các vectơ xử lý 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.

4. CÁC MODE VÀ CỜ TRÀN (TIMER MODES AND OVERFLOW) :

8051 cờ 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 Tmer 1.

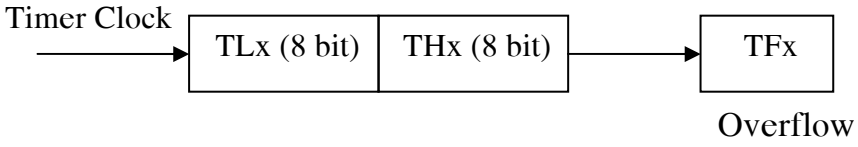
4.1. Mode Timer 13 bit (MODE 0) :



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.

4.2. Mode Timer 16 bit (MODE 1) :



MODE 1

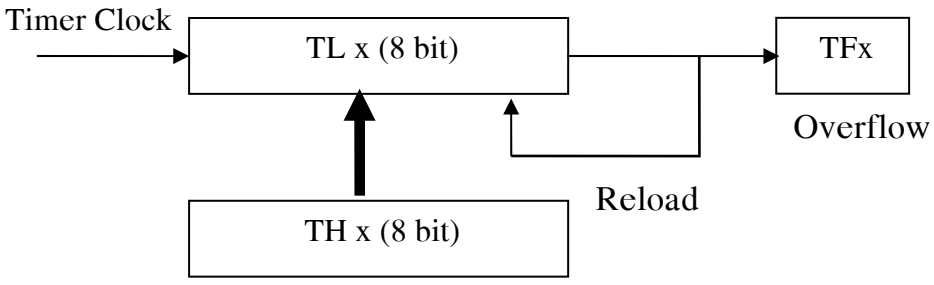
Mode 1 là mode Timer 16 bit, tương tư như mode 0 ngoại trừ Timer này hoạt động như một Timer đầy đủ 16 bit, xung clock được dùng với sự kết hợp các thanh ghi cao và thấp (TLx,THx). Khi xung clock được nhận vào, bộ đếm Timer tăng lên 0000H, 0001H,0002H, . . ., và một sự tràn sẽ xuất hiện khi có sự chuyển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

4..3. Mode tự động nạp 8 bit (MODE 2)

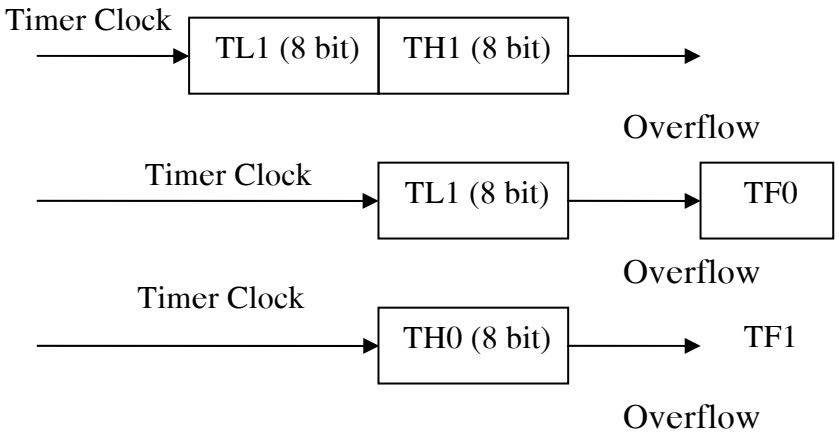


MODE 2

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.

4.4 Mode Timer tách ra (MODE 3) :



MODE 3

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

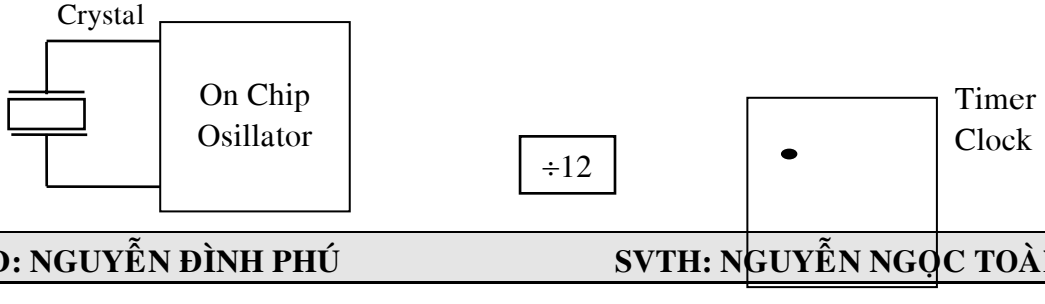
Timer 0 ở mode 3 được chia là 2 timer 8 bit. TL0 và TH0 hoạt động như những Timer riêng lẻ với sự tràn sẽ set các bit TL0 và TF1 tương ứng.

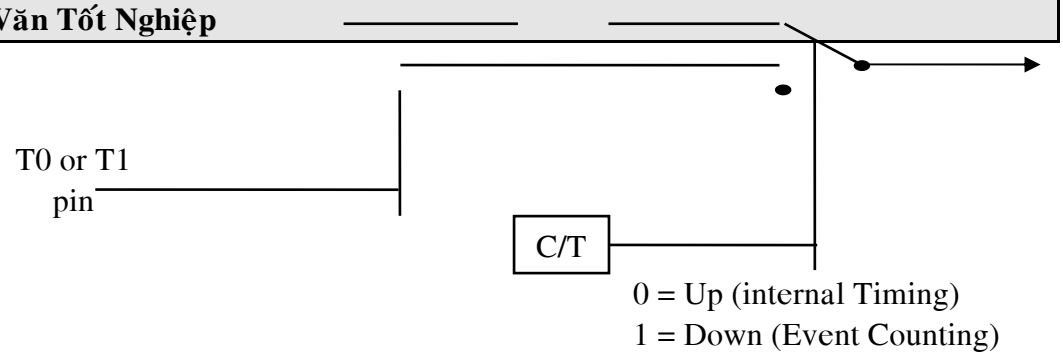
Timer 1 bị dừng lại ở mode 3, nhưng có thể được khởi động bởi việc ngắt nó vào một trong các mode khác. Chỉ có nhược điểm là cờ tràn TF1 của Timer 1 không bị ảnh hưởng bởi các sự tràn của Timer 1 bởi vì TF1 được nối với TH0.

Mode 3 tất yếu cung cấp 1 Timer ngoại 8 bit là Timer thứ ba của 8051. 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 đómà không sử dụng Interrupt.

5. CÁC NGUỒN XUNG CLOCK (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.





5.1 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 hầu hết các ứng dụng. Các thanh ghi TLx và THx tăng 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.

5.2 Sự đếm các sự kiện (Event Counting) :

Nếu bit C/T = 1 thì bộ Timer được ghi giờ từ bộ nguồn bên ngoài trong nhiều ứng dụng, bộ 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 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 được khám thấy. 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.

6. 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 khiểnbởi 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.

7. 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 thân chương trình các Timer đượcbắt 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 đều 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.

8. SỰ ĐỌC THANH GHI TIMER TRÊN TUYẾN :

Trong một số ứng dụng cần thiết đọc giá trị trong các thanh ghi Timer trên tuyến, có một vấn đề tiềm năng đơn giản để bảo vệ lại phần mềm. Bởi vì 2 thanh ghi Timer phải được đọc, nên "lỗi giai đoạn" có thể xuất hiện nếu byte tràn và byte cao giữa 2 hoạt động đọc. Một giải pháp để khắc phục là đọc byte cao trước, sau đó đọc byte thấp, và đọc lại byte cao : Nếu byte cao thay đổi thì lặp lại các hoạt động đọc.

V. HOẠT ĐỘNG INTERRUPT CỦA 8051 :

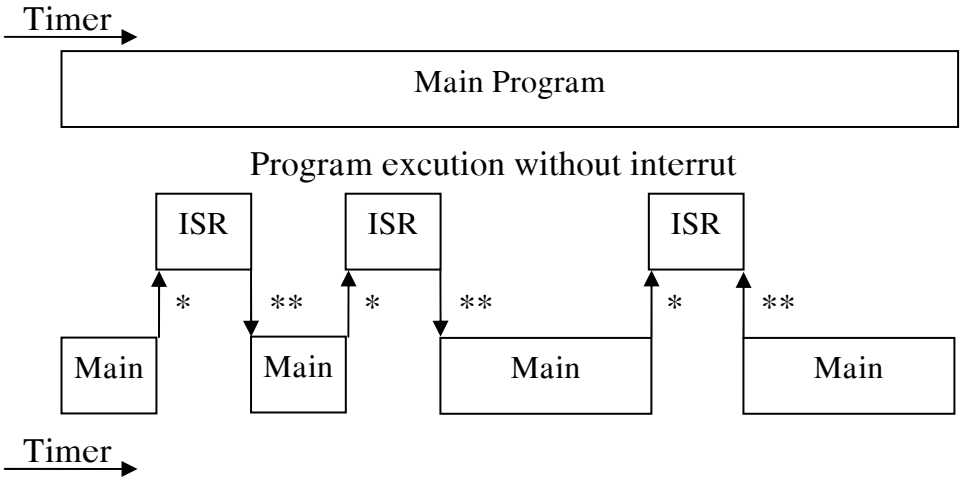
Trong nhiều ứng dụng đòi hỏi ta phải dùng Interrupt mà không dùng Timer bởi vì nếu dùng Timer ta phải mất thời gian để chờ cờ tràn TimerTFx set mới xử lý tiếp chương trình. Do đó ta không có thời gian để làm các việc quan trọng khác mà ứng dụng đòi hỏi. Đây là chương trình rất quan trọng của 8051 hay họ MSC-51.

1. GIỚI THIỆU :

Interrupt là một sự cố có điều kiện mà nó gây ra sự ngưng lại tạm thời của chương trình để phục vụ một chương trình khác. Các Interrupt vận hành một Relay rất quan trọng trong thiết bị và sự cung cấp đầy đủ các ứng dụng vi điều khiển. Chúng cho phép 1 hệ thống đáp ứng đồng bộ đến sự kiện quan trọng và giải quyết sự kiện đó trong khi chương trình khác đang thực thi. Một hệ thống được lái bởi Interrupt cho 1 kỹ xảo làm nhiều công việc cùng một lúc. Tất nhiên CPU không thể thực thi nhiều lệnh tại một thời điểm, nhưng nó có thể tạm thời treo việc thực thi của chương trình chính để thực thi chương trình khác và sau đó quay lại chương trình chính.

Khi chương trình chính đang thực thi mà có một sự ngắt xảy đến thì chương trình chính ngưng thực thi vàrẻ nhánh đến thủ tục phục vụ ngắt ISR (INTERRUPT SERVICE ROUTINE). ISR thực thi để thực hiện hoạt động và kết thúc với lệnh "trở lại từ sự ngắt" : Chương trình tiếp tục nơi mà nó dừng lại.

Ta có thể tóm tắt sự thực thi của 1 chương trình trong trường hợp có Interrupt và không có Interrupt như sau :



Program excution with interrupt.

Trong đó : Ký hiệu * cho biết ngắt chương trình chính để thực thi chương trình con trong thủ tục phục vụ ngắt ISR. Còn ký hiệu ** cho biết quay trở lại chương trình chính thực thi tiếp khi kết thúc chương trình con trong ISR.

2. TỔ CHỨC INTERRUPT CỦA 8051 (INTERRUPT ORGANIZATION)

8051 cung cấp 5 nguồn ngắt, 2 sự ngắt ngoài, 2 sự ngắt Timer và một sự ngắt Port nối tiếp. Tất cả các Interrrupt bị mất tác dụng bởi sự mặc định sau khi reset hệ thống và được cho phép cá biệt bởi phần mềm.

Trong trường hợp có hai hoặc nhiều hơn sự ngắt xảy ra đồng thời hoặc một sự ngắt đang được phục vụ mà xuất hiện một sự ngắt khác, thì sẽ có hai cách thực hiện sự ngắt là sự kiểm tra liên tiếp và sự ưu tiên cấp 2.

2.1 Sự cho phép ngắt và sự cấm ngắt

Mỗi nguồn Interrupt được cho phép riêng biệt hoặc sự cấm riêng biệt qua thanh ghi chức năng đặc biệt có bit định vị IE (Interrupt Enable) tại địa chỉ 0A8H. Cũng như sự cá biệt cho phép các bit của mỗi nguồn ngắt có 1 bit cho phép (hoặc cấm) chung mà nó được xóa để cấm tất cả các Interrupt hoặc được set để cho phép chung các Interrupt.

Hoạt động của từng bit trong thanh ghi cho phép ngắt IE được tóm tắt trong bảng sau :

Bit	Symbol	Bit Address	Sự mô tả (Enable = 1; Dissble)
IE.7	EA	AFH	Cho phép toàn bộ hoặc cấm toàn bộ.
IE.6	-	AEH	Không định nghĩa.
IE.5	ET2	ADH	Cho phép ngắt Timer 2 (8052).
IE.4	ES	ACH	Cho phép ngắt Port nối tiếp.
IE.3	ET1	ABH	Cho phép ngắt Timer 1.
IE.2	EX1	AAH	Cho phép ngắt ngoài External 1.
IE.1	ET0	A9H	Cho phép ngắt Timer 0.
IE.0	EX0	A8H	Cho phép ngắt ngoài External 0.

IE (Interrupt Enable) Register Sumary.

Hai bit phải set để cho phép 1 sự ngắt nào đó : Là bit cho phép riêng và bit cho phép chung. Ví dụ để cho phép ngắt Timer 1 ta có thể thực hiện trên

bit: SETB ET1 và SETB EA hoặc sự thực hiện trên byte : MOV IE, #10001000B. Cả 2 phương pháp này có kết quả chính xác sau khi reset hệ thống, nhưng kết quả khác nhau nếu thanh ghi IE được ghi trên tuyến ở giữa chương trình.

Giải pháp thứ nhất không có tác dụng trên các bit còn lại trong thanh ghi IE, còn giải pháp thứ hai xóa các bit còn lại trong thanh ghi IE. Ở đầu chương trình ta nên khởi gán IE với lệnh MOV BYTE, nhưng sự cho phép ngắt và cấm ngắt trên tuyến trong một chương trình sẽ dùng các lệnh SET BIT và CLR BIT để tránh kết quả phụ với các bit khác trong thanh ghi IE.

2.2 Sự ưu tiên ngắt (Interrupt Priority) :

Mỗi nguồn ngắt được lập trình cá biệt đến một trong hai mức ưu tiên qua thanh ghi chức năng đặc biệt có định vị IP (Interrupr Priority) tại địa chỉ 0B8H. Hoạt động của từng bit trong thanh ghi IP được tóm tắt trong bảng sau :

Bit	Symbol	Bit Address	Sự mô tả (Enable = 1; Disable)
IP.7	-	-	Không định nghĩa.
IP.6	-	-	Không định nghĩa.
IP.5	PT2	BDH	Ưu tiên cho sự ngắt Timer 2 (8052).
IP.4	PS	BCH	Ưu tiên cho sự ngắt Port nối tiếp.
IP.3	PT1	BBH	Ưu tiên cho sự ngắt Timer 1.
IP.2	PX1	BAH	Ưu tiên cho sự ngắt ngoài External 1.
IP.1	PT0	B9H	Ưu tiên cho sự ngắt Timer 0.
IP.0	PX0	B8H	Ưu tiên cho sự ngắt ngoài External 0.

IP (Interrupt Priority) Register Sumary.

Thanh ghi ưu tiên ngắt IP được xóa sau khi reset hệ thống để đặt tất cả các sự ngắt ở mức ưu tiên thấp hơn sự mặc định. Y tưởng "các sự ưu tiên" cho phép một thủ tục phục vụ ngắt ISR mới được ngắt nếu sự ngắt mới này ưu tiên cao hơn cho sự ngắt hiện hành đang phục vụ.

Trên 8051 có 2 mức ưu tiên thấp và 2 mức ưu tiên cao. Nếu một thủ tục phục vụ ngắt có mức ưu tiên thấp đang thực thi mà có một sự ngắt ưu tiên cao hơn xuất hiện, thì thủ tục phục vụ đó bị ngắt đi để thực thi thủ tục ngắt mới. Ngược lại thủ tục phục vụ ngắt có mức ưu tiên cao hơn đang thực thi mà có sự

ngắt ưu tiên thấp hơn xuất hiện, thì nó không thể bị ngắt mà phải chờ thực thi xong thủ tục phục vụ cao hơn mới nhảy tới thủ tục phục vụ ngắt thấp.

Chương trình thực thi ở mức cơ bản và không kết hợp với sự ngắt nào, nó có thể luôn luôn bị ngắt bất chấp sự ưu tiên ngắt ở mức cao hay thấp. Nếu 2 sự ngắt của các ưu tiên khác nhau xuất hiện đồng thời, sự ngắt ưu tiên cao hơn sẽ được phục vụ đầu tiên.

2.3 Sự kiểm tra vòng quét liên tiếp.

Nếu 2 sự ngắt có cùng mức ưu tiên xuất hiện đồng thời, thì sự kiểm tra vòng quét liên tiếp sẽ ấn định sự ngắt nào sẽ được phục vụ trước tiên. Sự kiểm tra vòng quét liên tiếp ưu tiên từ trên xuống theo thứ tự là : External 0, Timer 0, External 1, Timer 1, serial Port, Timer 2

Hình trên minh họa 5 nguồn ngắt của 8051, các kỹ xảo cho phép ngắt riêng và chung, sự kiểm tra vòng quét liên tiếp và mức ưu tiên. Trạng thái của tất cả các nguồn ngắt có hiệu lực thông qua các bit cờ tương ứng trong các thanh ghi chức năng đặc biệt. Nếu có sự ngắt nào bị cấm thì sự ngắt đó không xuất hiện nhưng phần mềm vẫn còn kiểm tra cờ ngắt.

Sự ngắt của Port nối tiếp đưa đến cổng OR logic của sự ngắt thu RI (Receive Interrupt) hoặc sự ngắt phát TI (Transmit Interrupt). Tương tự sự ngắt của Port nối tiếp, các sự ngắt của Timer 2 có thể được sinh ra bởi cờ tràn TF2 hoặc cờ nhập ngoài EXF2 (External Input Flag).

Các bit cờ sinh ra các sự ngắt được tóm tắt như sau :

Interrupt	Flag	SFR Register and Bit Position
External 0	IE 0	TCON 1
External 1	IE 1	TCON 3
Timer 1	TF 1	TCON 7
Timer 0	TF 0	TCON 5
Serial Port	TI	SCON 1
Serial Port	RI	SCON 0
Timer 2	TF 2	T2CON 7 (8052)
Timer 2	EXF 2	T2CON 6 (8052)

3. VIỆC XỬ LÝ CÁC SỰ NGẮT (PROCESSING INTERRUPT) :

Khi một sự ngắt xuất hiện và được chấp nhận bởi CPU thì chương trình chính bị ngắt. Các hoạt động sau đây xuất hiện :

√ Lệnh hiện hành và kết thúc thực thi.

√ Bộ đếm chương trình PC được cất giữ vào Stack.

√ Trạng thái ngắt hiện hành được cất giữ vào bên trong.

√ Những sự ngắt bị ngăn lại tại mức ngắt.

√ Bộ đếm chương trình PC được LOAD với địa chỉ vectơ của thủ tục phục vụ ngắt ISR.

√ Thủ tục phục vụ ngắt ISR được thực thi.

Thủ tục phục vụ ngắt ISR thực thi và đưa hoạt động vào đáp ứng ngắt, thủ tục phục vụ ngắt ISR kết thúc với lệnh RETI (quay trở về chương trình chính từ Stack). Điều này khôi phục lại giá htrị cũ của bộ đếm chương trình từ Stack và hoàn toàn dừng lại trạng thái cũ. Sự thực thi của chương trình chính tiếp tục ở nơi mà nó ngừng lại.

3.1 Các vectơ ngắt (Interrupt Vectors) :

Khi có một sự ngắt được nhận giá trị được LOAD vào PC được gọi bởi vectơ ngắt. Nó là địa chỉ của sự khởi động thủ tục phục vụ ngắt ISR của nguồn ngắt. Các vectơ được cho trong bảng sau :

Interrupt	Flag	Vectors Address
System Reset	RST	0000H
External 0	IE 0	0003H
Timer 0	TF 0	000BH
External 1	IE 1	0013H
Timer 1	TF1	001BH
Serial Port	RI or TI	0023H
Timer 2	TF 2 or EXF2	002BH

Vectơ reset hệ thống RST tại địa chỉ 0000H được tính trong bảng này, bởi vì trong ý nghĩa này nó giống như 1 Interrupt : nó ngắt chương trình và LOAD vào PC với giá trị mới.

Khi đưa 1 vectơ ngắt đến sự ngắt thì cờ của nó gây ra sự ngắt, tự động bị xóa bởi phần cứng, ngoài trừ RI và TI của Port nối tiếp và TF2, EXF2 của Timer 2 được xóa bởi phần mềm. Nguyên nhân trên là do có 2 nguồn ngắt có thể chịu đựng được cho mỗi sự ngắt mà nó không ứng dụng cho CPU để xóa cờ ngắt.

Vì các vectơ ngắt nằm ở dưới cùng trong vùng mã nhớ, nên lệnh đầu tiên của chương trình chính là lệnh nhảy đến vị trí cao hơn vị trí này như LJMP 0030H.

CHƯƠNG2

GIAO TIẾP MÁY TÍNH

GIỚI THIỆU CÁC PHƯƠNG PHÁP GIAO TIẾP MÁY TÍNH

Việc giao tiếp giữa máy tính và thiết bị ngoại vi có thể giao tiếp bằng 3 cách

- ρ Giao tiếp bằng Slot-Card.
- ρ Giao tiếp qua cổng song song (máy in)
- ρ Giao tiếp qua cổng nối tiếp (COM)

1. GIAO TIẾP BẰNG SLOT CARD

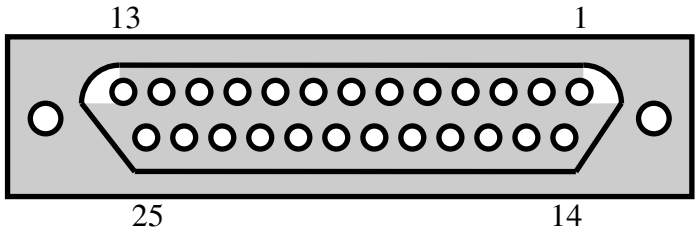
Bên trong máy tính, ngoài những khe cắm dùng cho card vào - ra, card màn hình, vẫn còn những rãnh cắm để trống. Để giao tiếp với máy tính, ta có thể thiết kế card mở rộng để gắn vào khe cắm mở rộng này. Ở máy tính PC/XT rãnh cắm chỉ có 1 loại với độ rộng 8 bit và tuân theo tiêu chuẩn ISA (**I**ndustry **S**tandard **A**rchitecture). Rãnh cắm theo tiêu chuẩn ISA có 62 đường tín hiệu, qua các đường tín hiệu này máy tính có thể giao tiếp dễ dàng với thiết bị bên ngoài thông qua card mở rộng.

Trên rãnh cắm mở rộng, ngoài 20 đường địa chỉ, 8 đường dữ liệu, còn có một số đường điều khiển như: RESET, IOR, IOW, AEN, CLK, ... Do đó card giao tiếp với máy tính qua slot card đơn giản, số bit có thể tăng dễ dàng, giảm được nhiều linh kiện, tốc độ truyền dữ liệu nhanh (truyền sonh song). Tuy nhiên, do khe cắm nằm bên trong máy tính nên khi muốn gắn card giao tiếp vào thì phải mở nắp ra, điều này gây bất tiện cho người sử dụng.

2. GIAO TIẾP BẰNG CỔNG SONG SONG

Việc giao tiếp giữa KIT Vi điều khiển 8051 với máy tính được thực hiện qua ổ cắm 25 chân ở phía sau máy tính. Qua cổng này dữ liệu được truyền đi song song, nên đôi khi còn được gọi là cổng ghép nối song song.

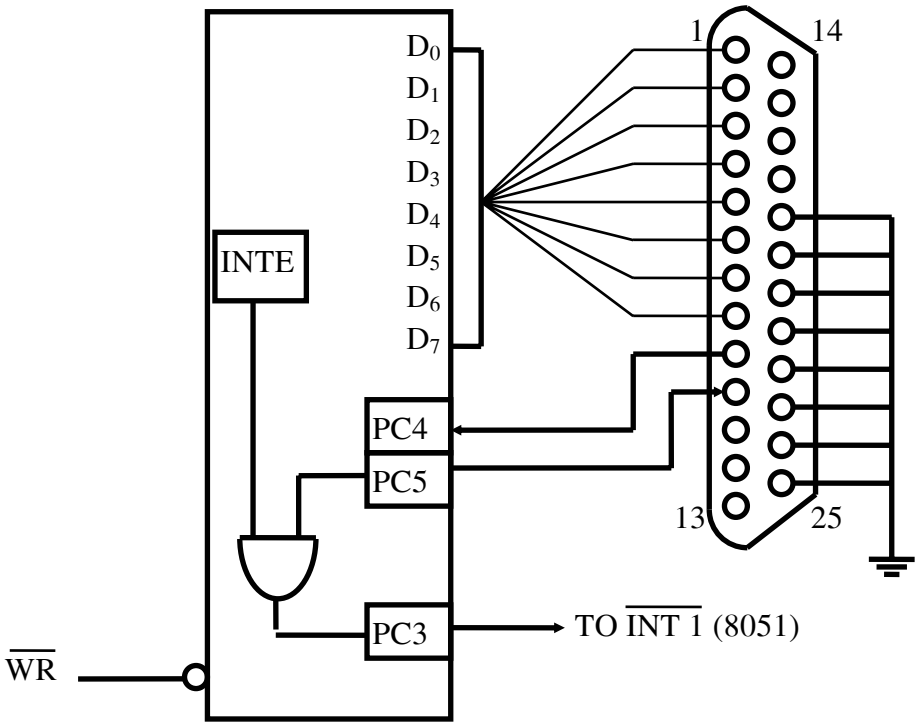
* Các chân và đường dẫn được mô tả như sau:



CHÂN	KÝ HIỆU	VÀO/RA	MÔ TẢ
1	STROBE	Lối ra (Output)	⎍ : Byte được in
2	D0	Lối ra	Đường dữ liệu D0
3	D1	Lối ra	Đường dữ liệu D1
4	D2	Lối ra	Đường dữ liệu D2
5	D3	Lối ra	Đường dữ liệu D3
6	D4	Lối ra	Đường dữ liệu D4
7	D5	Lối ra	Đường dữ liệu D5
8	D6	Lối ra	Đường dữ liệu D6
9	D7	Lối ra	Đường dữ liệu D7
10	ACK	Lối vào (Input)	Acknowledge (Xác nhận)
11	BUSY	Lối vào	1 : Máy in bận
12	PE	Lối vào	Hết giấy
13	SLCT	Lối vào	Select (Lựa chọn)
14	AF	Lối ra	Auto Feed (Tự nạp)
15	ERROR	Lối vào	Error (Lỗi)
16	INIT	Lối ra	0 : Đặt lại máy in
17	SLCTIN	Lối ra	Select in
18	GND		Nối đất
19	GND		“
20	GND		“
21	GND		“
22	GND		“
23	GND		“
24	GND		“
25	GND		“

Khi máy tính gởi dữ liệu ra cổng máy in muốn dữ liệu này qua KIT Vi điều khiển 8051 ta phải giao tiếp qua một vi mạch 8255. IC 8255 làm việc ở Mode 1 : Port A là nhập dữ liệu ; Port B xuất dữ liệu .

* Sơ đồ kết nối giữa IC 8255 với cổng máy in :



Vì 8255 được khởi tạo làm việc ở Mode 1 : Port A nhập dữ liệu Port B xuất dữ liệu nên khi máy tính gởi tín hiệu STROBE đến 8255, yêu cầu 8255 nhận dữ liệu do máy tính gởi đến và khi 8255 nhận dữ liệu thì nó tạo ra một tín hiệu ở PC5 đưa qua ACK báo cho máy tính biết là 8255 đã nhận dữ liệu do máy tính gởi đến, đồng thời lúc đó ở PC3 của 8255 tạo tín hiệu INTRA tác động đến chân ngắt INT1 (pin 13) của 8051 làm cho 8051 chạy chương trình phục vụ ngắt và dữ liệu từ máy tính qua 8255 sẽ được gởi đến CPU để xử lý.

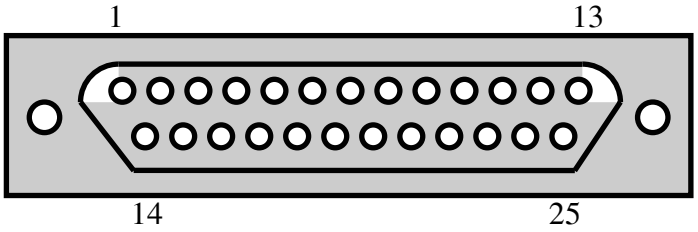
3. GIAO TIẾP BẰNG CỔNG NỐI TIẾP

Cổng nối tiếp RS232 là một giao diện phổ biến rộng rãi nhất. Người ta còn gọi cổng này là cổng COM1, còn cổng COM2 để tự do cho các ứng dụng khác. Giống như cổng máy in cổng COM cũng được sử dụng một cách thuận tiện cho việc giao tiếp với thiết bị ngoại vi.

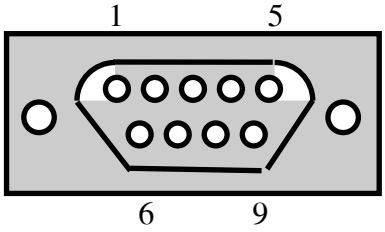
Việc truyền dữ liệu qua cổng COM được tiến hành theo cách nối tiếp. Nghĩa là các bit dữ liệu được truyền đi nối tiếp nhau trên một đường dẫn. Loại truyền này có khả năng dùng cho những ứng dụng có yêu cầu truyền khoảng cách lớn hơn, bởi vì các khả năng gây nhiễu là nhỏ đáng kể hơn khi dùng một cổng song song (cổng máy in).

Cổng COM không phải là một hệ thống bus nó cho phép dễ dàng tạo ra liên kết dưới hình thức điểm với điểm giữa hai máy cần trao đổi thông tin với nhau, một thành viên thứ ba không thể tham gia vào cuộc trao đổi thông tin này.

* Các chân và đường dẫn được mô tả như sau:



Loại 25 chân



Loại 25 chân

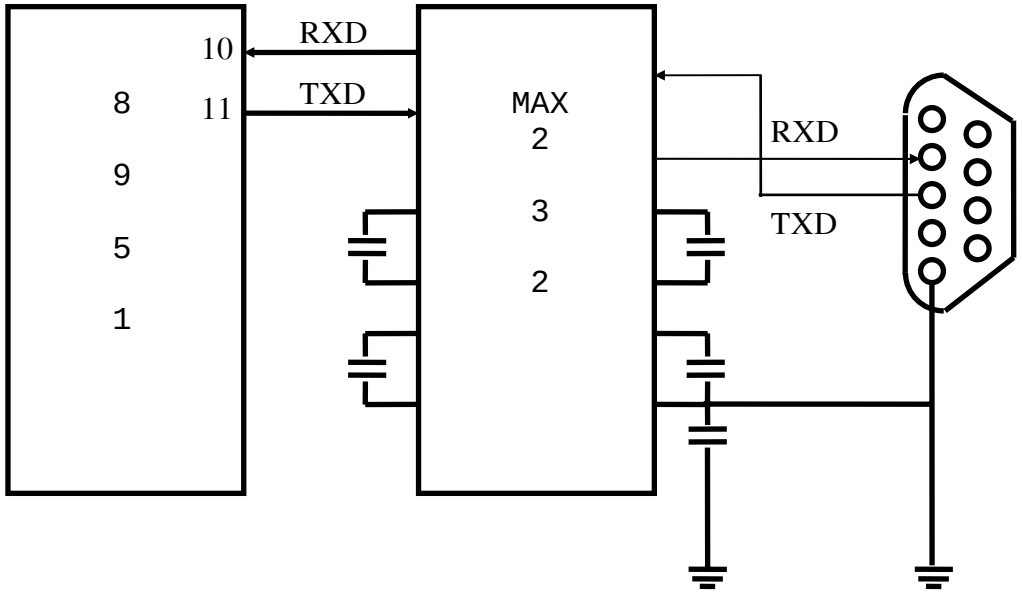
CHÂN (Loại 9 chân)	CHÂN (Loại 25 chân)	KÝ HIỆU	VÀO/RA	MÔ TẢ
1	8	DCD	Lối vào	Data Carrier Detect
2	3	RXD	Lối vào	Receive Data
3	2	TXD	Lối ra	Transmit Data
4	20	DTR	Lối ra	Data Terminal Ready
5	7	GND		Nối đất
6	6	DSR	Lối vào	Data Set Ready
7	4	RTS	Lối ra	Request to Send
8	5	CTS	Lối vào	Clear to Send
9	22	RI	Lối vào	Ring Indicator

Phích cắm COM có tổng cộng 8 đường dẫn, chưa kể đến đường nối đất. Trên thực tế có hai loại phích cắm, một loại 9 chân và một loại 25 chân. Cả hai loại này đều có chung một đặc điểm.

Việc truyền dữ liệu xảy ra ở trên hai đường dẫn. Qua chân cắm ra TXD máy tính gởi dữ liệu của nó đến KIT Vi điều khiển. Trong khi đó các dữ liệu mà máy tính nhận được, lại được dẫn đến chân RXD các tín hiệu khác đóng vai trò như là tín hiệu hổ trợ khi trao đổi thông tin, và vì thế không phải trong mọi trường hợp ứng dụng đều dùng hết.

Vì tín hiệu cổng COM thường ở mức +12V, -12V nên không tương thích với điện áp TTL nên để giao tiếp KIT Vi điều khiển 8051 với máy tính qua cổng COM ta phải qua một vi mạch biến đổi điện áp cho phù hợp với mức TTL, ta chọn vi mạch MAX232 để thực hiện việc tương thích điện áp.

** Sơ đồ kết nối giữa cổng COM với KIT Vi điều khiển 8051 :*



Vi mạch MAX232 dùng để tương thích tín hiệu ở mức (+12V, -12V) trên giao diện RS232.

Vi mạch này nhận mức RS232 đã được gởi tới từ máy tính và biến đổi tín hiệu náy thành tín hiệu TTL để cho tương thích với IC 8051 và nó cũng thực hiện ngược lại là biến đổi tín hiệu TTL từ Vi điều khiển thành mức +12V, -12V để cho phù hợp hoạt động của máy tính. Giao tiếp theo cách này, khoảng cách từ máy tính đến thiết bị ngoại vi có thể đạt tới trên 20 mét.

Đối với đề tài chỉ yêu cầu truyền dữ liệu từ máy tính qua KIT chứ không truyền dữ liệu từ KIT qua máy tính vì vậy chúng em chọn vi mạch MAX232 để thực hiện biến đổi tương thích mức tín hiệu.

Ưu điểm của giao diện này là có khả năng thiết lập tốc độ Baud.

Khi dữ liệu từ máy tính được gởi đến KIT Vi điều khiển 8051 qua cổng COM thì dữ liệu này sẽ được đưa vào từng bit (nối tiếp) vào thanh ghi SBUF (thanh ghi đệm), đến khi thanh ghi đệm đầy thì cờ RI trong thanh ghi điều khiển sẽ tự động Set lên 1 và lúc này CPU sẽ gọi chương trình con phục vụ ngắt và dữ liệu sẽ được đưa vào để xử lý.

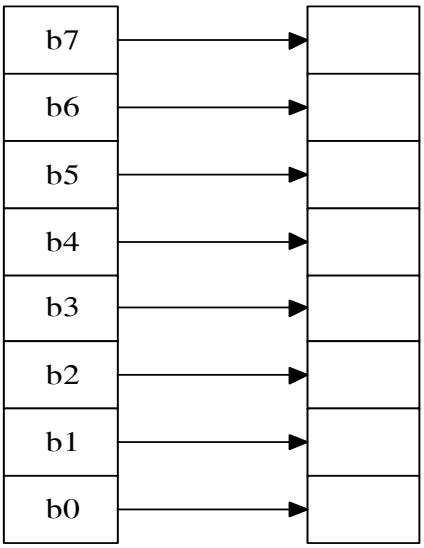
4. TRUYỀN DỮ LIỆU

1. Thông tin số liệu

Hệ thống thông tin số liệu dùng để xử lý và truyền các chuỗi mã nhị phân. Các mã này được tạo ra, lưu trữ và xử lý bởi máy tính và các thiết bị ngoại vi, bao gồm các loại như: các tin tức đã mã hóa, tập tin văn bản, hình ảnh, dữ liệu số và các thông tin khác.
Đường truyền là đường truyền ẫn tín hiệu số và các ký tự truyền phổ biến là mã ASCII.

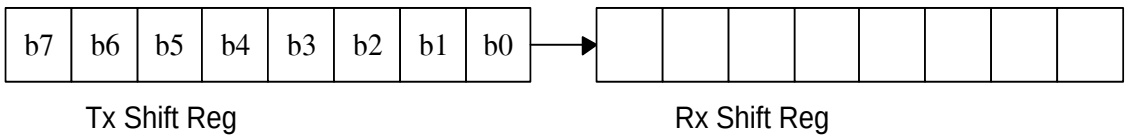
2. Phương thức truyền

a. Truyền nối tiếp/ Song song (Serial/ Parallel)
Truyền song song: truyền tất cả các bit của một ký tự cùng một lúc.



Hình 4.1 Truyền song song

Truyền nối tiếp: truyền tuần tự từng bit của 1 ký tự.



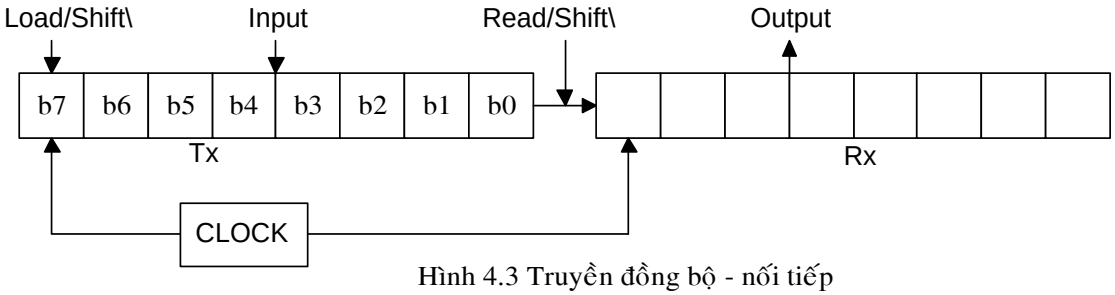
Hình 4.2 Truyền nối tiếp

Truyền song song nhanh hơn truyền nối tiếp (thường dùng ở cự ly thông tin gần).

Truyền nối tiếp ít tốn đường truyền hơn song song (thường dùng ở cự ly xa).

b. Truyền đồng bộ/ Bất đồng bộ (Synchoronous / Asynchonous)

- ***Truyền đồng bộ - nối tiếp:***



Hình 4.3 Truyền đồng bộ - nối tiếp

Dùng 1 xung clock để đồng bộ quá trình nhận theo từng bit ký tự. Máy sẽ cung cấp tín hiệu clock cho cả 2 đầu phát và thu.
Ưu điểm: chỉ truyền data, hông cần thêm tín hiệu đồng bộ vào chuỗi data → nhanh hơn.
Nhược điểm: phải thêm 1 kênh thứ 2 để truyền tín hiệu clock song song với kênh truyền data.

- ***Truyền bất đồng bộ nối tiếp:***

Thêm vào phía trước mỗi ký tự 1 bit START và phía sau 1 hoặc 2 bit STOP. Máy thu sẽ tách bit START 9ể khởi động tín hiệu đồng bộ dùng cho việc thu các bit ký tự. Các bit STOP được dùng để ngăn cách giữa các ký tữ. Phương pháp này cho phép truyền ngẫu nhiên, không cần truyền liên tục.

Vì phải thêm các bit START, STOP nên tốc độ truyền tổng quát là chậm so với truyền đồng bộ nhưng lại đơn giản rẻ tiền hơn.

Tốc Độ truyền bất đồng bộ: 75, 110, 300, 1200 bit/s
Tốc Độ truyền đồng bộ: 2400, 4800, 9600 bit/s

3. THÔNG TIN NỐI TIẾP BẤT ĐỒNG BỘ.

a/ Dẫn nhập

* Truyền số liệu nối tiếp cho phép trao đổi thông tin giữa máy tính và thiết bị ngoại vi từng bit một. Số liệu trao đổi thường được gởi theo các nhóm bit (tạo thành một kí tự hay một từ). Thí dụ: một ký tự được thể hiện bằng mã ASCII. Trao đổi nối tiếp chỉ cần một đường dây tín hiệu hay một kênh liên lạc.

* Truyền số liệu nối tiếp được sử dụng khi:

1. Thiết bị ngoại vi cần trao đổi số liệu vốn đã là vào/ra/nối tiếp.

   Ví dụ: Teletype, băng từ, catssete...

2. Khoảng cách giữa máy tính và thiết bị ngoại vi tương đối lớn. Nếu khoảng cách đó tăng thì giá thành tăng lên theo tổng số đường dây dẫn số liệu. Giá của hệ còn phụ thuộc vào các bộ khuyếch đại đường dây và bộ thu. Do đó sử dụng phương pháp trao đổi nối tiếp sẽ kinh tế hơn.

* Tốc độ truyền (còn gọi là tốc độ Baud-Rate) được xác định như tổn gsố lần thay đổi tín hiệu trong 1giây. Nếu tín hiệu truyền đi là nhị phân tốc độ truyền tương đối với số Bit truyền trong 1 giây. Các kênh thông tin được đánh giá bằng tốc độ truyền. Nếu số liệu được truyền với tốc độ ngoài khả năng của kênh sẽ sảy ra lỗi, bên thu sẽ không nhận đúntg được thông tin.

* Hệ thống truyền số liệu nối tiếp gồm các dạng:

- Đơn công: Số liệu chỉ được gửi đi theo một hướng.

- Bán song công: Số liệu được gửi theo hai hướng nhưng mỗi thời điểm chỉ được truyền theo một hướng.

- Song công: Số liệu được truyền đồng thời theo hai hướng.

* Truyền số liệu nối tiếp có thể là:

- Đồng bộ (DB)

- Bất đồng bộ (BDB)

Điểm chung của hai phương pháp này đều đòi hỏi thông tin khung (Frame) thêm vào thông tin số liệu để tạo điềui kiện cho bên thu/nhận biết dạng của số liệu.

Điểm khác nhau cơ bản là:

Trong truyền BDB, thông tin không cần cho từng ký tự, trong khí đó, ở truyền DB thông tin khung chỉ cần ở một chuỗi ký tự hay một khối (Block).

Truyền số liệu nối tiếp DB có hiệu suất lớn hơn truyền BDB nhưng đòi hỏi việc giải mã phức tạp hơn.

Phương pháp truyền DB sử dụng ở môi trường truyền dẫn có tốc độ cao, truyển BDB dùng ở môi trường có khả năng truyền dẫn chậm hơn.

Trong truyền BDB, dạng số liệu được cấu tạo từ các Bit số liệu cơ bản (các Bit thông tin và kiểm tra chẵn lẻ) và thêm vào phía trước một Bit khởi động

(Start) và phía sau một hay nhiều Bit dừng (Stop). Bit START có mức logic "0" được định nghĩa như mức điện áp dương trong chuẩn RS-232C. Bit STOP có mức logic "1". Bit START báo cho phía thu bắt đầu nhận ký tự và đồng bộ với bên phát. Quá trình đồng bộ này chỉ kéo dài trong thời gian "1" ký tự. Một hay nhiều Bit STOP được đưa vào sau ký tự để đảm bảo rằng Bit START của ký tự tiếp theo sẽ tạo ra quá trình truyển tiếp trên đường dây liên lạc. Bên thu có thể đuổi kịp bên phát nếu xung đồng bộ có chậm hơn. Mặt khác, nếu đồng bộ bên thu nhanh hơn bên phát, bên thu sẽ thấy có khoảng cách giữa các ký tự nhưng giải mã số liệu vẫn đúng. Như vậy, cho phép một sai số nhất định giữa bên thu và bên phát trong truyển nối tiếp bất đồng bộ.

Trong truyền nối tiếp đồng bộ, một hay vài ký tự khung sẽ được thêm vào một nhóm ký tự. Những ký tự này gọi là ký tự đồng bộ. Nhờ những ký tự này, thiết bị thu tái tạo được các ký tự thônh tin từ chuỗi Bit. Sự đồng bộ phải được giữ suốt trong một chuỗi số liệu dài. Ký tự đồng bộ thường được đưa vào từ kênh liên lạc ở MODEM ngay từ bên ngoài.

b/ Thủ tục truyền nối tiếp bất đồng bộ

* Đặc điểm của tín hiệu truyền nối tiếp bất đồng bộ là:
Tần số CLOCK thu, phát phân biệt với cùng một tần số danh định tùyheo tốc độ truyền bit.
Các ký tự truyền với những thời điểm không cần liên tục, truyền riêng biệt và ngẫu nhiên.
Đường truyền giữ ở trạng thái 1 trong khoảng cách giữa các ký tự, gọi là trạng thái rỗi (idle).
Đối với một ký tự thì LSB (Least Significant Bit) được truyền đầu tiên và lần lượt là các Bit kế tiếp.
Ở đầu phát:
Khi tín hiệu LOAD = 1 thì dữ liệu ở dạng song song sẽ được nạp vào TSR (từ ngõ nhập dữ liệu)
Khi tín hiệu LOAD = 0 thì các bit này sẽ được dịch nối tiếp ra đường truyền. Thanh ghi dịch phát TSR cũng sẽ bao gồm mạch logic tự động thêm các bit START và bit STOP.

Ở đầu thu:
Sẽ nhận biết điểm baa một ký tự bằng cách tách bit START nhờ mạch tách bit START (START BIT DETECT) khi trạng thái đường truyền dẫn chuyển từ 1 xuống 0 và lúc này bộ phận điều khiển sẽ điều khiển thanh ghi dịch bắt đầu dịch các bit trên đường dây vào. Sau 11 lần dịch (1 BIT START + 8 BIT

DATA + 2 BIT STOP) thì có thể đọc được ký tự thu dạng song song ở ngõ ra thanh ghi dịch khi có tín hiệu READ.

* Để kiểm tra sai khi truyền, trong 8 bit DATA sẽ có một bit kiểm tra theo một trong hai thủ tục sau:
Kiểm tra chẵn (Even parity): Tổng số bit một trong 8 bit phải luôn luôn chẵn.
Kiểm tra lẻ (Odd parity): Tổng số bit 1 luôn luôn lẻ.
Như vậy, ở đầu phát sẽ có bộ phận để đếm số bit 1 của 8 bit dữ liệu và tuỳ theo hình thưc kiểm, tra chẵn hay lẻ sẽ thêm vào bit cuối cùng giá trị 0 hoặc 1 cho thích hợp.
Ở đầu thu sẽ đếm số bit 1 của mỗi ký tự để xác định xem tính hình dữ liệu đến có đúng hay không?

* Tất cả các thủ tục trên không phải là bắt buộc mà có thể thay đổi khác nhau tùy theo việc cài đặt thông số ban đầu bởi người thực hiện việc truyền. Chỉ bắt buộc là các thủ tục ở hai đầu thu và phát tương ứng nhau. Tất cả những công việc đã mô tả ở trên sẽ được thực hiện bởi một bộ phận giao tiếp thông tin bất đồng bộ mà thành phần chính là chip LSI- gọi là UART.

4. THÔNG TIN NỐI TIẾP ĐỒNG BỘ.

Các thủ tục truyền nối tiếp bất đồng bộ đơn giản và rẻ tiền, nhưng chỉ thích hợp khi truyền các thông tin ngắn hoặc một vài ký tự cách quãng. Đối với các tập tin dài, sử dụng phương thức truyền thông tin đồng bộ sẽ hiệu quả hơn. Trong phương pháp này, thông tin được truyền theo từng khối (Blocks). Mỗi khối bao gồm một số tuần tự các ký tự và không cần các bit Start, bit Stop, mà sẽ đồng bộ theo từng khối cũng như việc kiểm tra sai.

Trong các hệ thống đồng bộ, tín hiệu Clock của máy phát sẽ được truyền qua máy thu song song với dữ liệu để dùng làm xung Clock cho việc dịch chuyển các bit thu. Nếu trong thực tế không thể thực hiện việc truyền tín hiệu Clock, thì máy thu phải tự tạo ra tín hiệu này. Do đó sẽ phức tạp hơn và có giá thành cao hơn so với thông tin bất đồng bộ. Để tránh trường hợp các chuỗi bit 0 hoặc 1 kéo dài đôi khi có thể dùng loại mã nhị phân đặc biệt để máy thu giữ được khả năng đồng bộ. Máy thu gửi một hoặc nhiều ký tự đồng bộ nhận dạng khi bắt đầu việc truyền và ngay khi nhận được bit đồng bộ, máy thu bắt đầu nhận bit. Phần lớn các mạng đồng bộ sử dụng các nghi thức do IBM tạo ra và nghi thức đồng bộ nhị phân BISYNC (Binary Synchronous) hoặc đồng bộ đường điều khiển dữ liệu SDLC (Synchronous Data Link Control).

Các giao tiếp chuẩn RS-232C và RS-449 cung cấp các chân sau để truyền tín hiệu Clock:

+ Đối với RS-232:

Chân 15: TCLK- Transmit Clock (từ DCE).

Chân 17: RCLK- Receive Clock (từ DCE).

Chân 24: ETCLK- Externel Transmit Clock.

+ Đối với RS-449:

Chân 6 và chân 23: Send Timing

Chân 8 và chân 26: Receive Timing.

Chân 17 và chân 35:Terminal Timing (từ DCE).

Khi dùng Modem đồng bộ thì tín hiệu định thời sẽ được cung cấp từ Modem đến máy tính. Tần số Clock phát có thể tạo từ Modem hoặc thiết bị đầu cuối.

LƯU ĐỒ VÀ CHƯƠNG TRÌNH DÀNH CHO PHẦN CỨNG

Để viết chương trình trênmáy cho PC, người ta có thể dùng các ngôn ngữ lập trình khác nhau. Dựa vào yêu cầu thiết kế mạch, dựa vào mức độ nhóm thực hiện thấy việc sử dụng ngôn ngữ Assembly kết hợp với các phục vụ ngắt của Bios để viết chương trình.

Các nhà thiết kế PC dành riêng Int 14H của Bios để phục vụ cho cổng nối tiếp. Ngắt này phục vụ khá đầy đủ các yêu cầu về xuất, nhập và kiểm tra trạng thái đường truyền và Modem. Việc sử dụng ngắt này làm cho chương trình trở nên dễ dàng, ngắn gọn.

I- Giới thiệu ngắt INT 14h của Bios:

Bios truy cập tới khối ghép nối nối tiếp nhờ ngắt INT 14h với các hàm như sau:

Hàm:	Vai trò.
00h	Khởi phá khối ghép nối tiếp
01h	Gửi một ký tự
02h	Nhận một ký tự
03h	Đọc trạng thái của khối ghép nối tiếp
04h	Khởi phát cảng nối tiếp mở rộng
05h	Điều khiển truyền thông của cảng nối tiếp mở rộng

Bios có thể điều hành tối đa 4 khối ghép nối tiếp, có tên từ COM1 đến COM4 với cá địa chỉ như sau:

Khối ghép nối	Địa chỉ cơ sở	Ngắt cứng IRQ
COM1	3F8h	IRQ4
COM2	2F8h	IRQ3
COM3	3E8h	IRQ4 (hoặc hỏi vòng)
COM4	2E8h	IRQ3 (hoặc hỏi vòng)

Ở mức độ chương trình, ta có thể chọn một khối ghép nối tiếp bằng cách gắn các mã tương ứng vào thanh ghi DX với các giá trị:

00h Cho	COM1
01h Cho	COM2
02h Cho	COM3
03h Cho	COM4

* Phục vụ 00h: Khởi phát khối ghép nối tiếp. Phục vụ 00h ấn định những thông số khác nhau của các khối ghép nối tiếp cũng như RS – 232C. Đó là các thông số:

- Số baud: Tốc độ trao đổi thông tin
- Tín chẵn lẻ
- Số bit dừng

Kích thước ký tự hây số bit nối tiếp. Những thông số này được tổ hợp trong mã 8 bit, được đặt vào thanh ghi AL, theo thứ tự các bit như sau:

+ D_7, D_6, D_5: mẫu cả vận tốc (tính bằng baud)
+ D_4, D_3: mã của tính chẵn lẻ
+ D_2: mã của bit dừng
D_1, D_0: mã của kích thước ký tự

Các mã trên theo bảng sau:

D_7	D_6	D_5	Vận tốc (bit persec)	D_4	D_3	Tính chẵn lẻ
0	0	0	110	0	0	Không có
0	0	1	150	0	1	Tính lẻ
0	1	0	300	1	0	Không có
0	1	1	600	1	1	Tính chẵn
1	0	0	1200			
1	0	1	2400			
1	1	0	4800			
1	1	1	9600			

D_1	Kích thước ký tự	D_1	D_0	Tính chẵn lẻ
0	Một bit Stop	0	0	Không dùng
1	Hai bit Stop	0	1	Không dùng
		1	0	7 bit
		1	1	8 bit

- Phục vụ 01h: Gửi một ký tự

Hàm này gửimột ký tự ra thiết bị ngoài với khối ghép nối tiếp. Muốn vậy, thực hiện chuỗi hành động sau:

- Đặt số liệu từ khối ghép nối tiếp vào thanh ghi DX (ví dụ COM1 với 00h)
- Gửi mã ký tự vào thanh ghi AL.
- Gửi 01h vào thanh ghi AH.
- Gọi INT 14h

Sau khi thực hiện chương trình con, thanh ghi AH chứa kết quả chương trình.

Nếu:

- Bit $D_7 = 1$, ký tự không được truyền đi
- Bit $D_7 = 0$, ký tự đã được truyền đi

- Phục vụ 02h: Nhận một ký tự. Trình tự nhận một ký tự cũng như trên, tức là:

Đặt số liệu từ khối ghép nối tiếp vào DX.

Đặt giá trị AH bằng 02h

Gọi INT 14h

Kết quả của chương trình con là ký tự được gửi vào khối ghép nối tiếp trong thanhghi AL. Thanh ghi AH cũng chứa kết quả của việc thực hiện chương trình như trường hợp AH = 01h, tức là:

Bit D_7=1, ký tự không tự nhận

Khi D_7=0, ký tự đã được nhận.

- Phục vụ 02h: Đọc trạng thái của khối ghép nối.

Muốn vậy cũng phải theo các trình tự:

Đặt số hiệu khối ghép nối vào DX

Đặt 03h vào AH

Kết quả các chương trình con là:

Trạng thái của đường dây (của khối ghép nối) được đặt trong thanh ghi AH, có các bit như hình dưới.

Trạng thái của Modem được đặt trong thanh ghi AL như hình dưới:

Bit	Ý nghĩa
D_7	Vượt qua độ trễ =0: Không có sai số =1 Có sai số
D_6	Thanh ghi dịch chuyển =0 Thanh ghi bận =1 Thanh ghi rỗi
D_5	Thanh ghi đợi =0 Thanh ghi bận =1 Thanh ghi rỗi

Bit	Ý nghĩa
D_7	Tín hiệu của sóng mang = 0 Không được phát hiện = 1 Đã được phát hiện
D_6	Chỉ báo chuông = 0 Không reo chuông = 1 Reo chuông
D_5	Thiết bị đầu cuối của thanh ghi đã sẵn sàng = 0 Modem không sẵn sàng = 1 Modem sẵn sàng

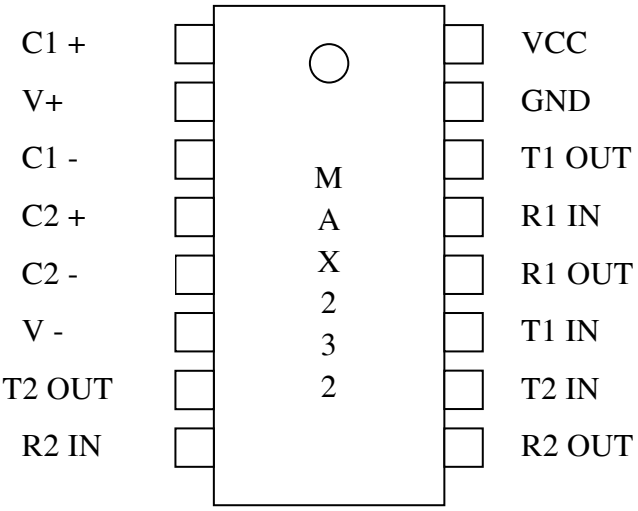
D_4	Ngắt bởi tín hiệu Break =0 Không biết =1 Có tín hiệu Break		D_4	Sẵn sàng phát (truyền) = 0 Chưa sẵn sàng = 1 Sẵn sàng
D_3	Giao thức = 0 Không có lỗi = 1 Có lỗi		D_3	Tín hiệu sóng mang đã thay đổi =0 Không thay đổi = 1 Thay đổi
D_2	Tính chẵn lẻ = 0 Không cò lỗi = 1 Có lỗi		D_2	Đường chỉ báo chuông thay đổi = 0 Không thay đổi = 1 Có thay đổi
D_1	Số liệu = 0 Không có tràn = 1 Bị tràn		D_1	Đường dây "trạm số liệu sẵn sàng" đã thay đổi =0 Không có thai = 1 Có thay đổi
D_0	Số liệu đã sẵn sàng = 0 Không có số liệu sẵn sàng = 1 Có Số liệu sẵn sàng		D_0	Đường dây "sẵn sàng truyền" đã thay đổi = 0 Không thay đổi = 1 Thay đổi

CHƯƠNG III

THIẾT KẾ VÀ THI CÔNG MODUL GIAO TIẾP MÁY TÍNH VỚI KIT 8051

3. GIỚI THIỆU VI MẠCH GIAO TIẾP MAX 232

Vi mạch MAX 232 của hãng MAXIM là một vi mạch chuyên dùng trong giao diện nối tiếp với máy tính. Chúng có nhiệm vụ chuyển đổi mức TTL ở lối vào thành mức +10V hoặc –10V ở phía truyền và các mức +3...+15V hoặc -3...-15V thành mức TTL ở phía nhận.

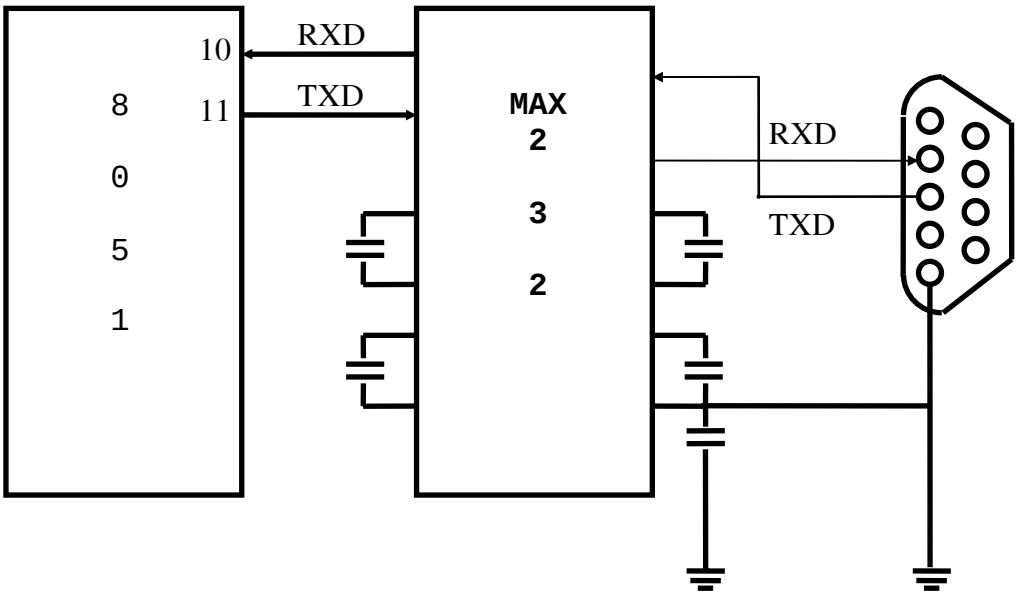


Vi mạch MAX 232 có hai bộ đệm và hai bộ nhận. Đường dẫn điều khiển lối vào CTS, điều khiển việc xuất ra dữ liệu ở cổng nối tiếp khi cần thiết, được nối với chân 9 của vi mạch MAX 232. Còn chân RST (chân 10 của vi mạch MAX) nối với đường dẫn bắt tay để điều khiển quá trình nhận. Thường thì các đường dẫn bắt tay được nối với cổng nối tiếp qua các cầu nối, để khi không dùng đến nữa có thể hở mạch các cầu này. Cách truyền dữ liệu đơn giản nhất là chỉ dùng ba đường dẫn TxD, RxD và GND (mass)

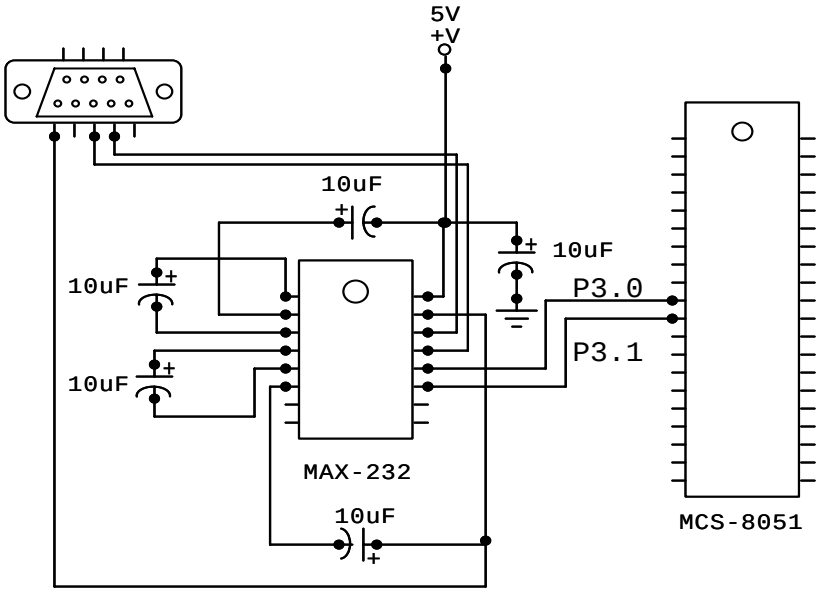
THIẾT KẾ VÀ THI CÔNG MODUL GIAO TIẾP

1. Sơ đồ kết nối

** Sơ đồ kết nối giữa cổng COM với KIT Vi điều khiển 8051 :*



2. Sơ đồ thực thực tế



CHƯƠNG 1

GIỚI THIỆU NGÔN NGỮ HỢP NGỮ

1. NGÔN NGỮ MÁY VÀ HỢP NGỮ

Chương trình là một tệp lệnh được đưa vào bộ nhớ cho máy thực hiện. Các lệnh có thể được thể hiện theo nhiều dạng (ngôn ngữ) khác nhau, dạng cơ bản nhất mà máy (CPU) có thể hiểu ngay gọi là ngôn ngữ máy (Machine Language). Tùy theo CPU mà ngôn ngữ máy có một dạng nhất định, điều đó có nghĩa với một loại CPU có một ngôn ngữ máy riêng. Sau đây là một đoạn chương trình ngôn ngữ máy thuộc họ Intel 8086/80x86 :

Lệnh	Dạng thập lục phân	Dạng nhị phân
1	B4 02	1011 0100 0000 0010
2	80 C2 30	1000 0000 1100 0010 0011 0000
3	50	0101 0000

Đọan chương trình trên gồm 3 câu lệnh có chiều dài lần lược là 2, 3 và1 byte. Byte đầu tiên gọi là mã lệnh hay tác tử (Operation Code) xác định tác vụ mà CPU phải thực hiện, phần còn lại là tác tố (Operand) xác định dữ liệu hoặc nơi chứa dữ liệu mà lệnh tác động vào. Chiều dài các câu lệnh theo qui định của CPU.

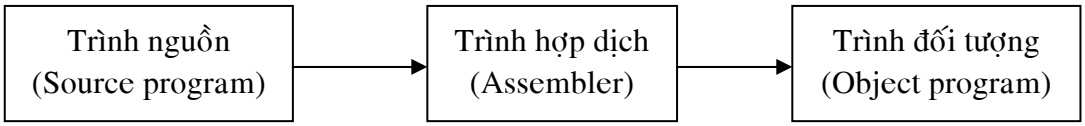
Để có thể lập trình với loại ngôn ngữ này, lập trình viên phải biết về tổ chức của máy đang sử dụng.

Vì là ngôn ngữ riêng của máy nên chương trình viết bằng ngôn ngữ này thực hiện rất nhanh và chiếm ít chỗ trong bộ nhớ tuy nhiên vì chương trình viết dưới dạng nhị phân nên rất khó viết và khó nhớ dễ nhầm lẫn.

Hợp ngữ (Assembly Language) là một loại ngôn ngữ giúp lập trình viên viết chương trình dễ dàng hơn thay cho ngôn ngữ máy. Hợp ngữ có dạng như ngôn ngữ máy tức là một lệnh hợp ngữ tương đương với một lệnh của ngôn ngữ máy và cũng có thể một lệnh hợp ngữ tương đương với nhiều lệnh ngôn ngữ máy nhưng khác với ngôn ngữ máy ở chỗ thay vì viết chương trình dưới dạng nhị phân người ta dùng một số ký hiệu tượng trưng dễ nhớ như MOV là lệnh chuyển, ADD là lệnh cộng, SUB là lệnh trừ. Ví dụ 3 lệnh ngôn ngữ máy ở trên có thể viết dưới dạng hợp ngữ như sau:

Lệnh	Dạng ngôn ngữ máy	Dạng hợp ngữ
1	B4 02	MOV AH, 02h
2	80 C2 30	ADD DL, 30h
3	50	PUSH AX

Dĩ nhiên là máy không thể hiểu được chương trình viết bằng hợp ngữ nên phải qua giai đoạn dịch, để dịch chương trình từ hợp ngữ sang ngông ngữ máy. Chương trình làm nhiệm vụ dịch một chương trình sang ngôn ngữ máy gọi là trình hợp dịch (Assembler). Chương trình viết bằng hợp ngữ gọi là chương trình nguồn(hay gốc –sourse program) và chương trình dưới dạng ngôn ngữ máy dịch từ chương trình nguồn gọi là chương trình đích (hay đối tượng -object program) như sơ đồ sau:



TẠO VÀ CHẠY CHƯƠNG TRÌNH HỢP NGỮ

Để tạo và chạy một chương trình hợp ngữ bạn cần có một trong các bộ trình hợp dịch như Turbo Assembler của hãng Borland International (gồm trình hợp dịch TASM.EXE và trình liên kết TLINK.EXE) hoặc Microsoft Assembler của hãng Microsoft (gồm trình hợp dịch MASM.EXE và trình liên kết LINK.EXE) ngoài ra còn một số tập tin khác trong các bộ chương trình này. Dù đang sử dụng của hãng nào cũng phải theo qui trình sau:

Bước 1: Trước hết bạn cần có mốt trình soạn thảo văn bản để tạo chương trình nguồn hợp ngữ như NC (Norton Commander), Turbo trong Turbo Pascal..., sau khi soạn được ghi lên đĩa thành một tập tin có họ là ASM (ví dụ HELLO.ASM)

Bước 2: Dịch chương trình đã soạn (HELLO.ASM) với trình hợp dịch TASM.EXE (đối với sử dụng bộ dịch của hãng turbo). Sau khi dịch trên đĩa sẽ có một tập tin mới gọi là tập đối tượng (HELLO.OBJ) dòng lệnh dịch chương trình như sau:

```
C:\ TASM HELLO.ASM
Turbo Assembler Version 2.01 Copyright (c) 1990 Borland International
Assembling file:      hello.asm to hello.obj
Error message:        None
Warning message:      None
Passes:               1
Remaining memory:     370k
```

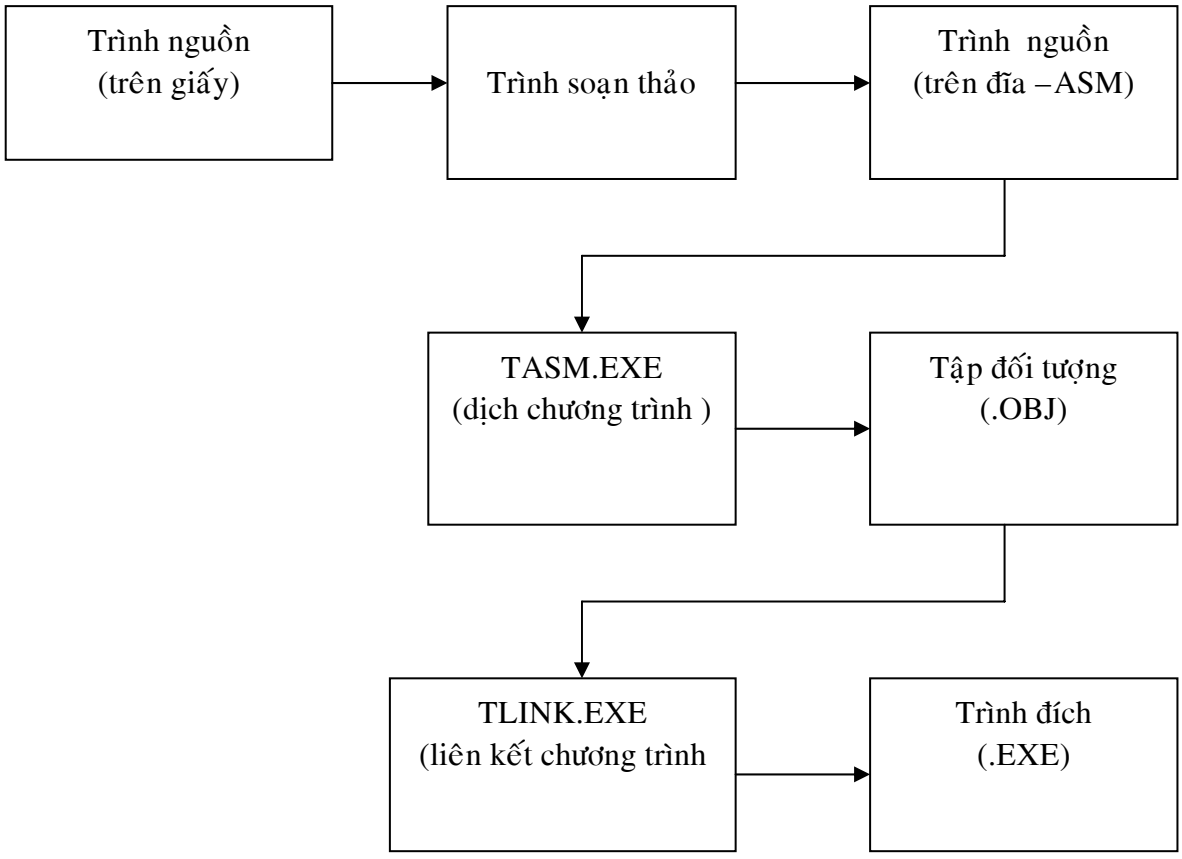
Thông báo trên cho biết chương trình của bạn không có lỗi sai. Nếu có, phải sửa lại chương trình (với trình soạn thảo) và cho dịch lại. Bây giờ trên đĩa của bạn có hai tập tin HELLO.ASM (chương trình nguồn do bạn tạo ra) và HELLO.OBJ (tập tin đối tượng). Nếu chương trình không lỗi thì qua bước 3

Bước 3: Liên kết chương trình với trình liên kết TLINK.EXE. dòng lệnh thực hiện liên kết như sau:

C:\TLINK HELLO.OBJ
Turbo Link Version 3.01 Copyright (c) 1990 Borland International

Nếu chương trình của bạn không có lỗi sai, TLINK sẽ tạo tập thực hiện HELLO.EXE.

Bước 4: Cuối cùng là thực hiện chương trình của bạn. Qui trình tạo và thực hiện chương trình có thể tóm tắt như ở hình sau:



CÚ PHÁP LỆNH HỢP NGỮ

Chương trình hợp ngữ gồm nhiều lệnh, mỗi lệnh viết trên một dòng. Lệnh hợp ngữ phân làm hai loại là chỉ thị và chỉ dẫn. Chỉ thị là lệnh sẽ được dịch sng mã máy, tức là lệnh sẽ được thi hành, còn chỉ dẫn (còn gọi là lệnh giả) chỉ là lệnh hướng dẫn trình hợp dịch trong quá trình dịch chương trình. Dạng tổng của một lệnh gồm 4 chương trình như sau:

<tên> <tác tử> <tác tố> < ;ghi chú>
vd: DoAddition: ADD AX, DX ;Tăng AX lượng DX

Các trường hợp cách nhau ít nhất là một khoảng trắng hoặc kí tự nhảy (Tab)

a/ Trường tên

Trường tên có thể là nhãn (Label) hoặc kí hiệu (Symbol). Nhãn là một tên đại diện cho một vị trí trong chương trình (trường hợp này có dấu : theo sau), hoặc tên thủ tục (chương trình con) hoặc tên biến vùng nhớ chứa dữ liệu).

b/ Trường tác tử

Trường tác tử là tên gợi nhớ của lệnh. Nếu là chỉ thị như MOV, ADD, ... thì lệnh sẽ được dịch sang mã máy còn nếu là chỉ dẫn như ENDS, PROC, ... thì đó là lệnh hướng dẫn trình hợp dịch trong quá trình dịch chương trình sang mã máy.

c/ Trường tác tố

Trường tác tố xác định dữ liệu sẽ được xử lý bởi lệnh. Lệnh có thể có hoặc không có tác tố. Nếu có hai tác tố thì chúng cách nhau bằng dấp phẩy, tác tố thứ nhất (từ trái qua) gọi là tác tố đích, tác tố thứ hai gọi là tác tố nguồn.

d/ Trường ghi chú

Sau mỗi câu lệnh có thể viết dòng ghi chú sau dấu chấm phẩy với mục đích là để giải thích ý nghĩa của lệnh

KHAI BÁO DỮ LIỆU

Dữ liệu trong chương trình đều được chuyển sang dưới dạng nhị phân, tuy nhiên bạn có thể viết dưới dạng thậo phân, thập lục phân hoặc chuỗi ký tự

a/ Cách viết số

Trong các chương trình bình thường được hiểu là thập phân, khi cần có thêm chữ D hoặc d đằng sau số (ví dụ 10,10D, 10d) đều có giá trị như nhau

Số viết theo hệ thập lục phân kết thúc bằng chữ H hoặc h phải bắt đầu là một số (ví dụ 10h, 10H, 2F8h, 2F8H)

Số nhị phân kết thúc bằng B hoặc b (ví dụ 1001b, 1001B)

b/ Chuỗi ký tự

Ký tự hoặc chuỗi ký tự phải rào giữa hai dấu nháy đơn (‘) hoặc dấu nháy kép(“) (ví dụ ‘Hello’,”hello”,’A’, “A”) các ký tự được chuyển thành mã ASCII tương ứng, do đó ‘A’,”A” ,41h hoặc 65 đều có nghĩa như nhau.

c/ Định nghĩa dữ liệu

Các chỉ dẫn thông dụng dùng định nghĩa dữ liệu kiểu byte, từ (2 byte – Word) hoặc từ kép (4 byte – Double word) như sau:

Nhãn DB trị, trị, ;byte

Nhãn DW trị, trị, ;word
Nhãn DD trị, trị, ;double word

Với nhãn là tên vùng nhớ (còn gọi là biến, thực chất là địa chỉ tượng trưng của vùng nhớ và được chuyển thành địa chỉ thật sau khi dịch chương trình) được định nghĩa với kích thước 1 byte (DB), 2 byte (DW) hoặc 4 byte (DD). Mỗi trị ghi trong phần tác tố sẽ là trị được gán cho vùng nhớ được cấp phát. Nếu thay trị bằng dấu ? thì sẽ không gán trị cho vùng nhớ

Vd : B DB 5 có nghĩa là vùng nhớ được cấp phátcó địa chỉ là B, chiếm 1 byte và có trị là 5

Vd : W DW 10 có nghĩa là vùng nhớ được cấp phát có địa chỉ là W chiếm 2 byte và có giá trị là 0Ah (W là 0A còn W+1 là 00)

d/ Định nghĩa hằng

Thay vì viết trực tiếp các hằng số hoặc chuỗi trong chương trình, ta có thể đặt tên (gọi là kí hiệu) cho rằng ở đầu chương trình, sau đó chỉ cần dùng các tên đó thay cho các hằng. Cách đặt tên cho hằng này làm chương trình dễ đọc và dễ hiểu hơn với cú pháp sau:

Tên EQU Hằng

Vd:

```
CR      EQU  0D h
LF      EQU  0A h
STR     EQU  'Du lieu nhap sai!!!!'
```

Sau đó có hai dòng sau là tương đương

```
MESS  DB   STR, CR, LF , 'S'
MESS  DB   'Du lieu nhap sai!!!!', 0Dh, 0Ah, '$'
```

CẤU TRÚC CHƯƠNG TRÌNH

Như đã trình bày, chương trình mã máy gồm 3 phần chứa trong 3 đoạn là đoạn mã dữ liệu và ngăn xếp do đó trình hợp ngữ cũng được tổ chức tương tự với các lệnh thích hợp.

a/ Kiểu bộ nhớ

Kích thước bộ nhớ dùng cho đoạn mã và dữ liệu được xác định bằng chỉ dẫn MODEL như sau:

MODEL kiểu

Với kiểu là:

- TINY

Mã và dữ liệu nằm cho phạm vi một đoạn

- SMALL

Mã nằm trong một đoạn 64K nhưng dữ liệu ở trong phạm vi một đoạn 64K

- COMPACT

Mã trong phạm vi một đoạn 64K và dữ liệu có thể lớn hơn 64K

Thường có ít chương trình nào có mã hoặc dữ liệu lớn hơn 64K nên kiểu SMALL là đủ. Kiểu TINYdùng để dịch chương trình sang dạng .COM.

b/ Đoạn ngăn xếp

Đoạn ngăn xếp khai báo kích thước vùng ngăn xếp với chỉ dẫn :

. STACK Kích Thước

Kích thước là độ lớn ngăn xếp tính bằng byte, nếu không ghi sẽ mặc nhiên là 1024. Ví dụ sau khai báo vùng ngăn xếp 256 byte

. STACK 100h

c/ Đoạn dữ liệu

Đoạn dữ liệu dùng khai báo biến hoặc hằng bắt đầu bằng chỉ dẫn .DATA. Ví dụ:

```
. DATA
CR          EQU  13
LF          EQU  10
VungNho1    DW   2
VungNho2    DW   3
ThongBao    DB   'CHUONG TRINH ABC '
```

d/ Đoạn mã

Đoạn mã chứa các lệnh của chương trình bắt đầu bằng chỉ dẫn :

. CODE

Lệnh cuối cùng của chương trình là chỉ dẫn END. Tóm lại một chương trình hợp ngữ thông thường có dạng sau:

```
. MODEL     SMALL
. STACK     100h
. DATA
; phần khai báo dữ liệu
. CODE
; phần lệnh
END
```

CHƯƠNG 2

GIỚI THIỆU NGÔN NGỮ LẬP TRÌNH VISUAL C++

I. Tổng quan ngôn ngữ visual c

Tại thời điểm hiện nay đã có rất nhiều ngôn ngữ lập trình khác nhau phục vụ cho nhiều mục đích khác nhau, tùy vào ứng dụng cụ thể mà người lập trình sẽ sử dụng một ngôn ngữ nào mà mình biết để viết chương trình.

Trong cuốn Luận văn tốt nghiệp này để tạo phần giao diện cho chương trình tôi sử dụng ngôn ngữ lập trình Visual C++ một ngôn ngữ khá phổ biến hiện nay.

Visual C++ là một ngôn ngữ lập trình trực quan nó dựa trên nền tảng của Ngôn ngữ C/C++ vì thế những ai đã biết đến ngôn ngữ C/C++ đều có thể tự học và tự viết cho mình một chương trình. Đây là ngôn ngữ chạy trên môi trường Windows và có thể liên kết với các chương trình trong môi trường Dos

2. Cách viết một chương trình bằng visual c++

Tôi không thể giới thiệu một cách đầy đủ về Visual C++ nhưng có thể tóm tắt quá trình viết chương trình bằng Visual C++ như sau:

Bước 1: Thiết kế giao diện

Bước 2: Viết mã lệnh

- Đối với bước thiết kế giao diện, bạn sẽ thiết kế "bộ mặt" của chương trình. Bạn dùng các công cụ Visual C++ để đưa các đối tượng khác nhau (như là các nút bấm, thanh cuộn, nút radio...) vào trong cửa sổ chương trình của bạn. Đặc biệt trong phần thiết kế giao diện bạn không phải viết một mã lệnh nào.

- Đối với bước viết mã lệnh bạn dùng trình soạn thảo của Visual C++ và ngôn ngữ lập trình C++ để viết mã lệnh cho chương trình

CHƯƠNG 3

CHƯƠNG TRÌNH

CHƯƠNG TRÌNH GIAO DIỆN

CÁC MÃ LỆNH VIẾT CHO CHƯƠNG TRÌNH GIAO DIỆN

♦ **HIỂN THỊ FILE**

```
void CLUANVANTOTNGHIEPDoc::Serialize(CArchive& ar)
{
	if (ar.IsStoring())
	{
		// TODO: add storing code here
	}
	else
	{
		// TODO: add loading code here
	}

	// Calling the base class CRichEditDoc enables serialization
	//  of the container document's COleClientItem objects.
	m_bRTF=FALSE;
	CRichEditDoc::Serialize(ar);
}
```

♦ **CHUYỂN ĐỔI FILE ASM THÀNH LILE LIST**

```
void CChildFrame::OnCovertlst()
{
	// TODO: Add your command handler code here
	system("C:\\LVTN\\ASM51 TEST1.ASM ");

}
```

♦ **CHUYỂN ĐỔI FILE OBJ THÀNH FILE HEX**

```
void CChildFrame::OnConverthex()
{
        // TODO: Add your command handler code here
        system("C:\\LVTN\\RL51 TEST1.OBJ");
        system("C:\\LVTN\\OH TEST1.OBJ");

}
```

♦ **GỌI CHƯƠNG TRÌNH MÁY TÍNH TRỢ GIÚP CHO VIỆC ĐỔI CÁC CƠ SỐ KHÁC NHAU**

```
void CLUANVANTOTNGHIEPView::OnCalCulator()
{
        // TODO: Add your command handler code here
        system("Calc.exe");
}
```

♦ **GỌI CHƯƠNG TRÌNH TRUYỀN TỪNG BYTE KÍ TỰ**

```
void CLUANVANTOTNGHIEPView::OnTransmitted()
{
        // TODO: Add your command handler code here
        system("C:\\LVTN\\TERMINAL.EXE");
}
```

♦ **GỌI CHƯƠNG TRÌNH TRUYỀN FILE DỮ LIỆU (*.HEX)**

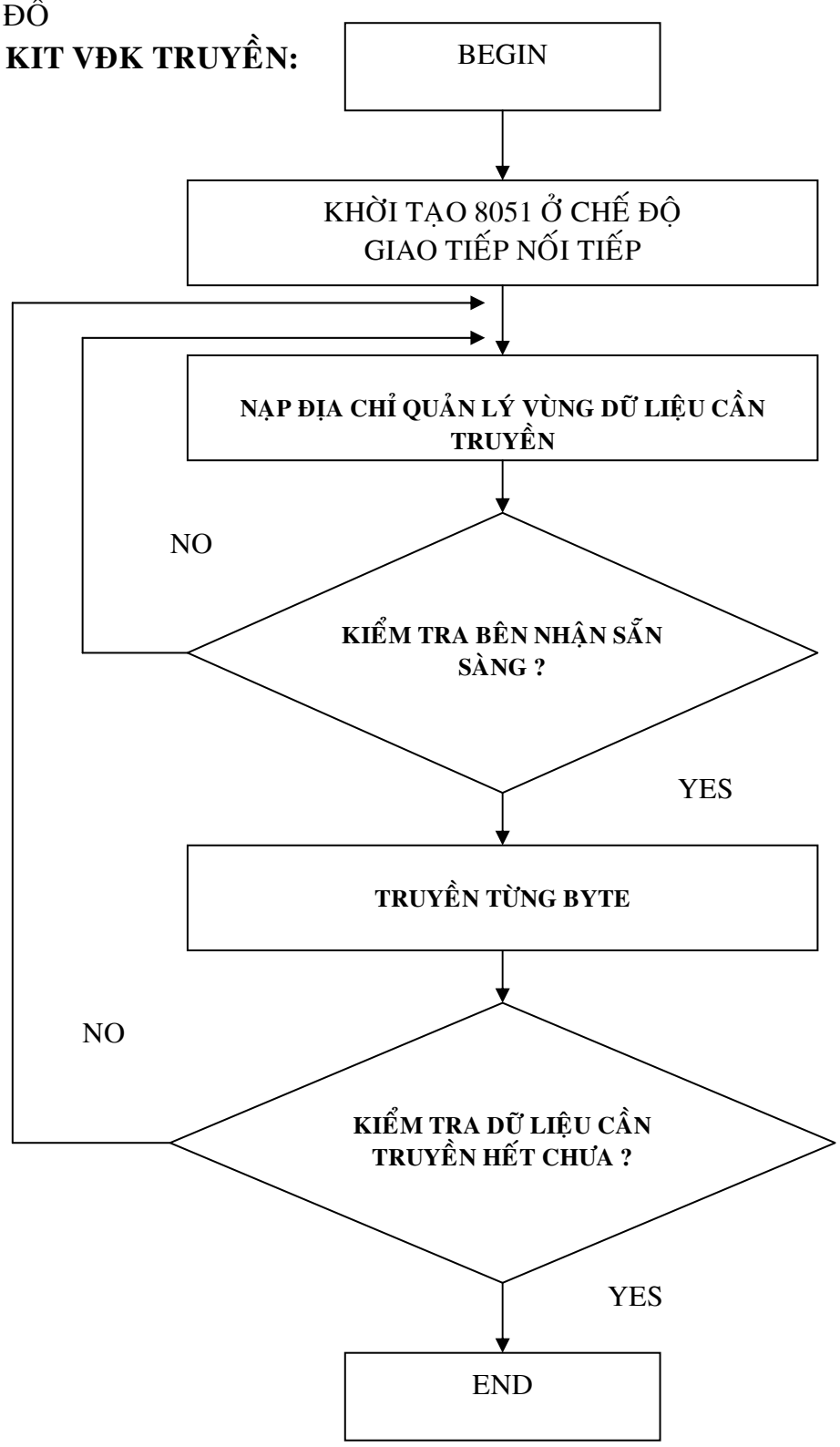
```
void CChildFrame::OnLoadfile()
{
        // TODO: Add your command handler code here
        system("C:\\LVTN\\TERMINA3.EXE");
}
```

CHƯƠNG TRÌNH TRUYỀN DỮ LIỆU

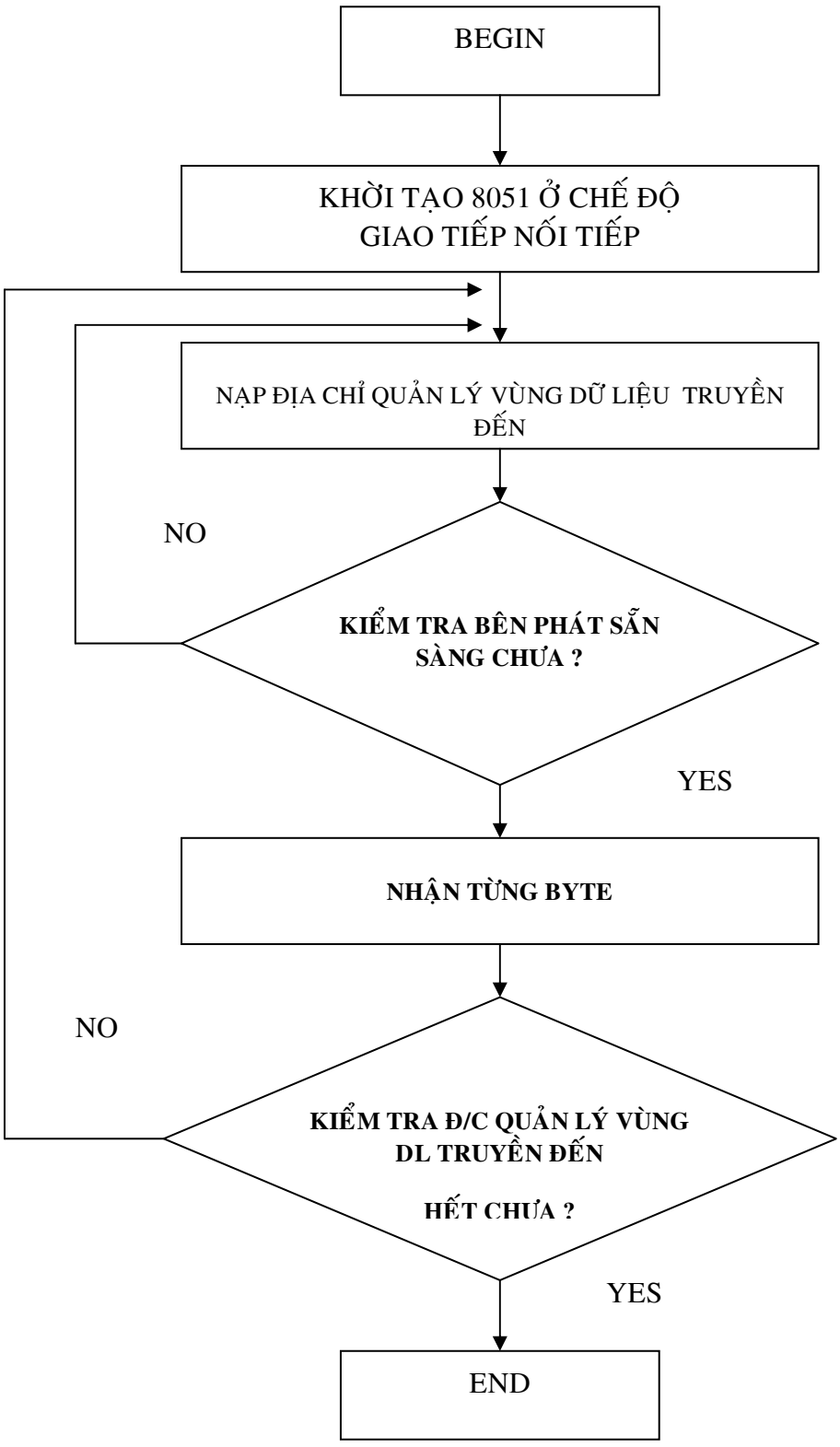
TRUYỀN GIỮA 2 KIT VI ĐIỀU KHIỂN

1. LƯU ĐỒ

♦ **BÊN KIT VĐK TRUYỀN:**



- **BÊN KIT VĐK 8051 NHẬN:**



2. CHƯƠNG TRÌNH

♦ BÊN KIT VĐK TRUYỀN:

```
        org 5000h
        mov IE,#00h         ;khoi tao khong cho phep ngat
        mov tmod,#20h
        mov th1,#-13        ;timer mode 2, 2400baud
        setb tr1            ;cho phep chay
        mov scon,#0fch      ;khoi tao truyen data mode 3
        mov dptr,#6000h     ;nap dia chi quan li vung ma
x2:     jb p1.1,x2          ;neu bang 1 thi cho
        movx a,@dptr        ;lay du lieu de truyen di
        mov sbuf,a          ;goi len thanh ghi dem
x1:     jnb ti,x1           ;kiem tra ti vi sau khi goi 1 byte thi ti=1

                            ;neu dung bang 1 thi xoa de goi tiep byte thu 2
        clr ti
        inc dptr
        mov a,dpl
        cjne a,#0ffh,x2
        mov a,#76h
        mov dptr,#0c000h
        movx @dptr,a
        sjmp $
end
```

♦ BÊN KIT VĐK 8051 NHẬN:

```
        org 5000h
        setb p1.1
        mov IE,#00h         ;cam ngat
        mov tmod,#20h
        mov th1,#-13
        setb tr1
        mov scon,#0fch      ;khoi tao giao tiep noi tiep
        mov dptr,#6000h
xr2:    clr p1.1
xr1:    jnb ri,xr1
        clr ri              ;xoa vi da co du lieu
        mov a,sbuf          ;lay du lieu tu may phat goi toi
        movx @dptr,a        ;cat du lieu
        inc dptr
        mov a,dpl
        cjne a,#0ffh,xr2
        mov a,#79h
```

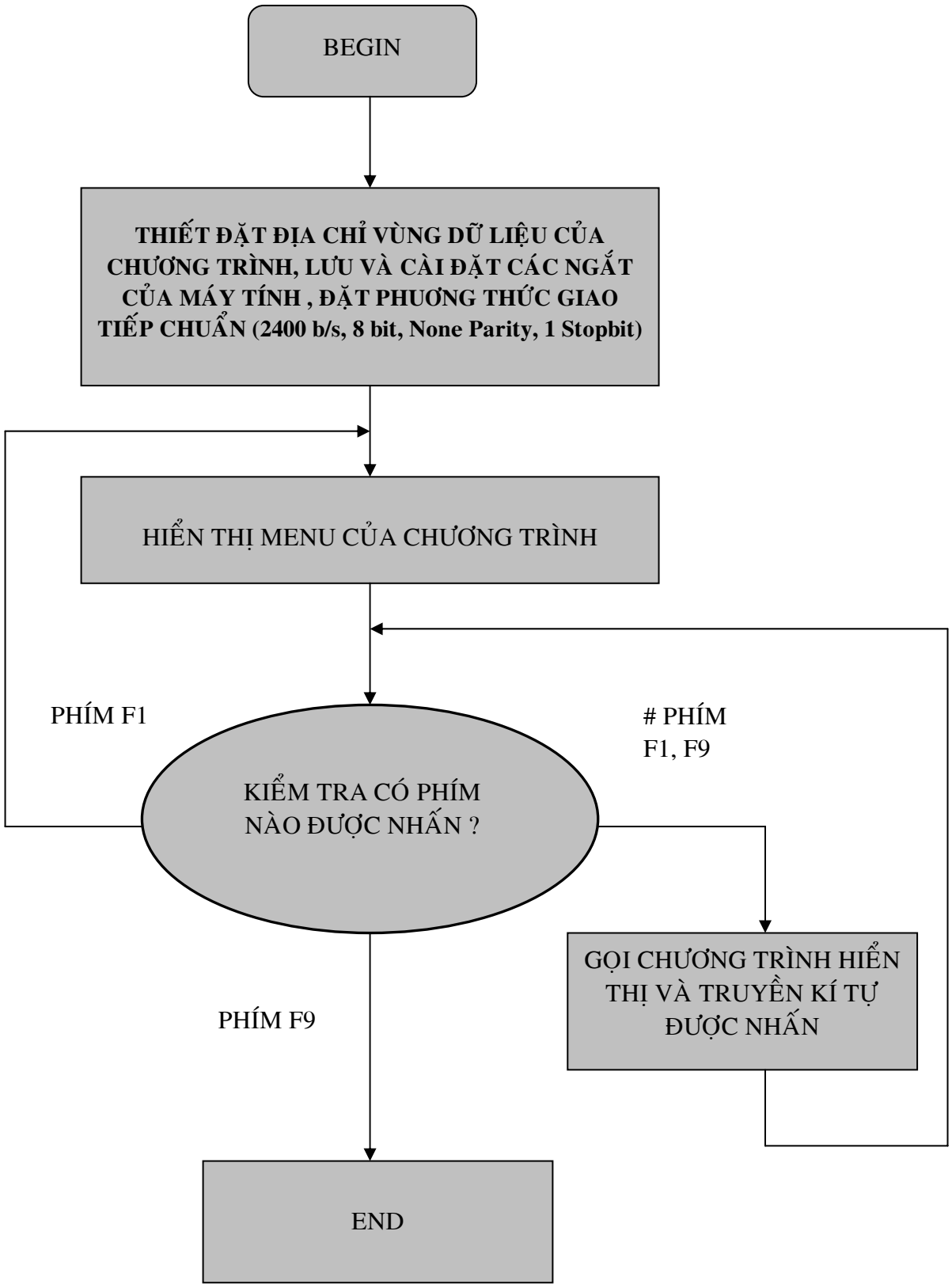
```
        mov dptr,#0c000h
        movx @dptr,a
        sjmp $                  ;nhay tai cho
end
;RxD cua may phat va RxD cua may thu duoc noi voi nhau lam duong truyen
data
;TxD cua may phat va TxD cua may thu duoc noi voi nhau lam xung clk
```

TRUYỀN DỮ LIỆU TỪ MÁY TÍNH ĐẾN KIT VĐK 8051

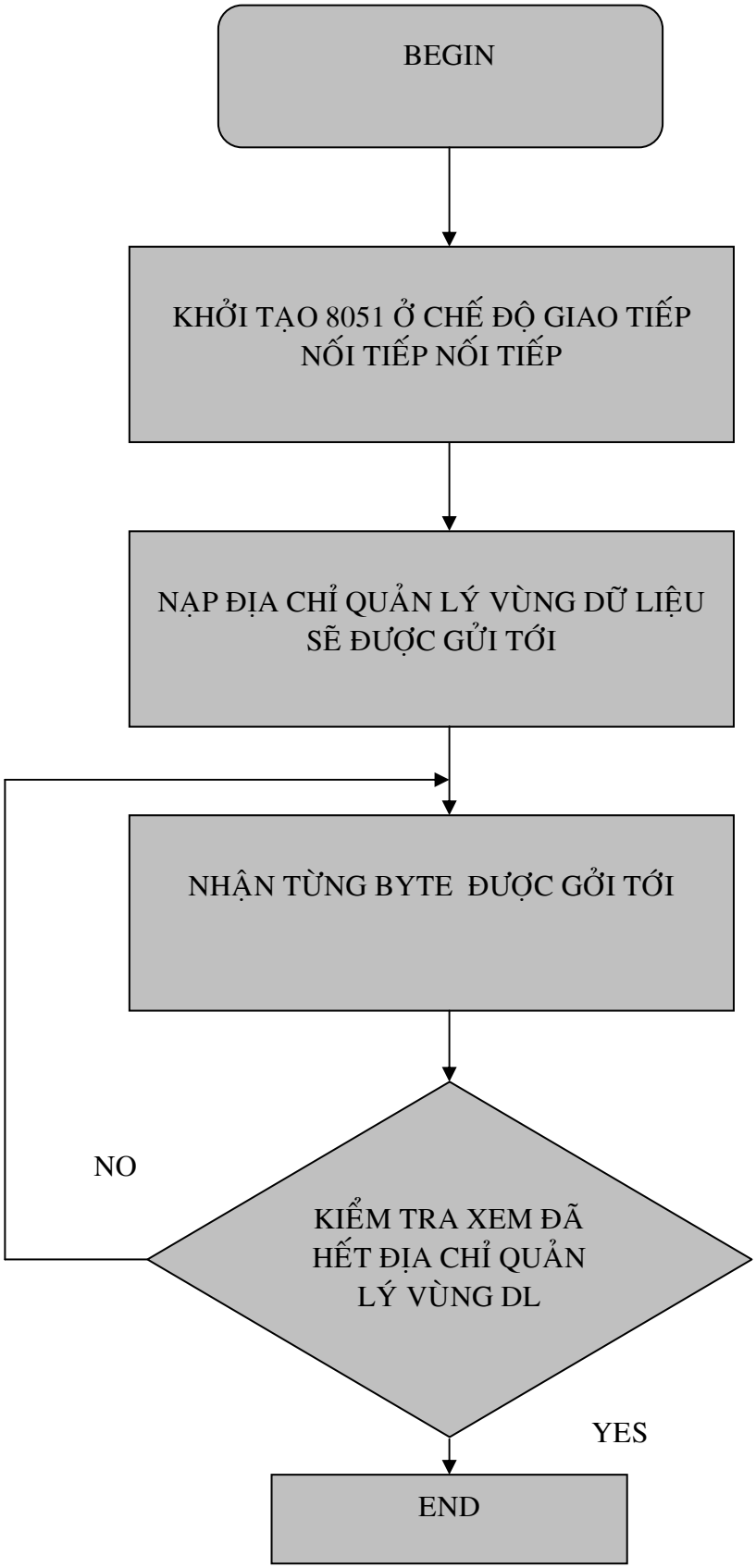
B1. TRUYỀN TỪNG BYTE KÍ TỰ TỪ MÁY TÍNH ĐẾN KIT VĐK

1. LƯU ĐỒ

- **BÊN TRUYỀN (MÁY TÍNH)**


BEGIN
THIẾT ĐẶT ĐỊA CHỈ VÙNG DỮ LIỆU CỦA CHƯƠNG TRÌNH, LƯU VÀ CÀI ĐẶT CÁC NGẮT CỦA MÁY TÍNH , ĐẶT PHƯƠNG THỨC GIAO TIẾP CHUẨN (2400 b/s, 8 bit, None Parity, 1 Stopbit)
HIỂN THỊ MENU CỦA CHƯƠNG TRÌNH
PHÍM F1
PHÍM F1, F9
KIỂM TRA CÓ PHÍM NÀO ĐƯỢC NHẤN ?
GỌI CHƯƠNG TRÌNH HIỂN THỊ VÀ TRUYỀN KÍ TỰ ĐƯỢC NHẤN
PHÍM F9
END

♦ **BÊN NHẬN (KIT VI ĐIỀU KHIỂN 8051):**



2. CHƯƠNG TRÌNH

♦ **BÊN TRUYỀN (MÁY TÍNH):**

CHƯƠNG TRÌNH TRUYỀN TỪNG BYTE KÍ TỰ TỪ MÁY TÍNH

```
;               STACK
stack  segment      stack
               db      300 dup('?')
stack  ends
;---------------------------------------      DATA      ----------------------
Data   Segment
;Messages
MENU_MS  db      ",0dh,0ah
         db      ",0dh,0ah
         db      '* THE TRANSMITTED  CHARACTERS * ‘,0dh,0ah
         db      '** --> Press <F1> To Redisplay This MENU  ** ',0dh,0ah
         db      '** --> Press <F9> To Exit This Program ** ’, 0dh,0ah
         db      'Input characters from keyboard:'
         db      '$'
;
PROT_MS   db          ",0dh,0ah
ERR1_MS          db     0dh,0ah,'*** Cannot Transmit ***',0dh,0ah
;
;PROGRAM PARAMETER STORAGE
CARD_BASE        dw     03f8h          ;Address of RS 232 card for all
harware
                                       ;type except
INT_NUM          db     0ch            ;Offset in Bios table as follows
SETUP_BYTE       db     0bbh           ;Default value
;
;Origin interrupt vector address to restore on exit
O_INT_SEG dw     0000h                  ;segment
O_INT_OFF dw     0000h                  ;offset
;
;circular buffer and pointers:
CIRC_BUF  db     20 dup(00h)            ;Circular buffer
               dw     0
DATA_IN          dw     0              ;imput pointer
DATA_OUT dw      0                     ;output pointer
;
DATA          Ends
;-----------------------------------  CODE  ----------------------------------
CODE          SEGMENT
         ASSUME     CS:CODE
START:
```

```
;
;Establish addressability of program's data Segment
        mov   ax,data
        mov   ds,ax
        assume        ds:data
        mov   es,ax
        assume        es:data
;
;Display MENU at cursor
        mov   dx,offset menu_ms ;messages
        call    show_message
;
;---------------------|
;Hardware Type  |
;---------------------|
;
;Examine Ram location F000:FFFE to determine IBM hardware
        push  ds                    ;Save program DS
        mov   dx,0f000h
        mov   ds,dx
        mov   al,ds:[0fffeh]        ;Code to AL
;
;Get addr of the RS232 card from BIOS data area
        mov   dx,0                  ;Bios data area segment
        mov   ds,dx                 ;Data segment to Bios area
        mov   cx,ds:0400h           ;Offset card 1
        pop   ds                    ;Restore program ds
        mov   Card_Base,cx          ;Save in program's data
;
;determine interrupt number:
;       0bh    IRQ4  all other hardware
        cmp   al,0fdh               ;Code for PCjr
        jne     Set_Address
        mov   int_num,0bh
;
;---------------------------------|
;Save/Install Interrupt        |
;---------------------------------|
;Obtain and save the segment/offset of the original communications
;interrup installed on entry using Dos service number 53
;of Int 21h
Set_Address:
```

```
        mov   ah,53             ;Dos service request number
        mov   al,Int_Num        ;Interrup number (0bh or 0ch)
        int   21h
;
;ES:BX = segment/offset of original handler
        mov   O_Int_Seg,es      ;Save segment
        mov   O_Int_Off,bx      ;and offset
;
;Insert address of the interrup service routine in the Bios
;Label for interrup service routine is: RS232_INT
;use DOS service number 37 of INT 21h
        mov   ah,37             ;Dos service request number
        mov   al,int_num        ;Machine interrup number
        mov   dx,offset cs:RS232_Int
        push  ds                ;Save program data segment
        push  cs
        pop   ds                ;Set DS to segment base of
        int   21h               ;interrup service routine
        pop   ds                ;Restore program's ds
;
;-----------------|
;Set protocol    |
;-----------------|
;Set default communication parameters
        mov al,10100011b        ;Control code
        mov ah,0                ;Bios request number
        mov dx,0                ;Comm1 in all hardware types
        int 14h                 ;Bios service request
        call comm_on
        call flush              ;Flush keyboard buffer
;

;-------------------------------------------------------------------------------|
;                       Send and receive characters                             |
;                         monitor funtion key                                   |
;-------------------------------------------------------------------------------|
Monitor:
        mov ah,1                ;Code for read keyboard status
        int 16h                 ;Bios service
        jz ser_imp              ;Nothing in keyboard buffer
        jmp char_typed          ;Character in keyboard buffer
;
```

```
;Delay loop to allow interrupt to occur
Ser_imp:
        sti                         ;interrupt on
        mov cx,50
Delay:      nop
        nop
        loop delay
;
;-------------------------------------|
;Test for new data received      |
;-------------------------------------|
        cli                         ;Interrup off while reading pointer
        mov bx,data_out             ;Compare pointers
        cmp bx,data_in
        jne new_data                        ;New data item or items
        sti                         ;Interrup on
        jmp Monitor                 ;Reapeat cycle
;
;------------------|
;Process char   |
;------------------|
;Receive character type from keyboard buffer
char_typed:
        mov ah,0                    ;Code for read keyboard char
        int 16h                     ;Bios service
;Test for <F1> and <F9> keys
        cmp ax,3b00h                ;<F1>
        jne test_f9
        jmp show_menu               ;<F1> key pressed
Test_F9:
        cmp ax,4300h                ;<F9>
        je dos_exit
        jmp show_and_send           ;<F9> key pressed
;
;----------------|
;       Exit    |
;----------------|
Dos_Exit:
;Communications interrupts OFF
        call Comm_Off
;
;Restore orginal interrupt vector for communications interrup number
```

```
        mov ah,37               ;Dos service request number
        mov al,int_num          ;Machine interrup number
        mov dx,o_int_off        ;Offset to DX
        mov ax,o_int_seg        ; Segment
        mov ds,ax               ;to DS
        int 21h
;Exit
        mov ah,76               ;Dos service request number
        mov al,0                ;No return code
        int 21h                 ;Exit to dos
;
;---------------------|
;Redisplay Menu       |
;---------------------|
Show_Menu:
        mov dx,offset menu_ms
        call show_message       ;Display message routine
        jmp monitor
;
;-------------------------|
;New Data Receiver        |
;-------------------------|
New_Data:
        lea si,circ_buf         ;Circular buffer address
        mov bx,data_out         ;Output pointer
        add si,bx               ;Buffer start +displacement
        mov al,byte ptr[si]     ;Get character
;
;Update output pointer
        inc bx                  ;Bump
        cmp bx,20               ;Pointer overflows buffer?
        jne ok_out_ptr
        mov bx,0                ;Request to start of buffer
;
Ok_Out_Ptr:
        mov data_out,bx         ;Update
;
;Display byte taken from buffer
        sti
        call tty
        jmp monitor
;
```

```
;-----------------------|
;Display Protocol   |
;-----------------------|
Show_Protocol:
        mov dx,offset prot_ms
        call show_message       ;Display message routine
        jmp monitor
;
;-------------------------|
; Output and Display  |
;-------------------------|
Show_and_Send:
;Send through RS-232c line
;Wait loop for tranmitter holding register empty
        mov cx,2000             ;Prime wait counter
        push ax                 ;Save character to transmit
;
Thre_Wait:
        mov dx,card_base
        add dx,5                ;Line status register
        in al,dx                ;Get byte at port
        jmp short $+2           ;Thre bit set?
        test al,20h
        jnz ok_2_send
        loop thre_wait
;
;Wait period timed out,display error message and exit
        pop ax                          ;Restore stack
        mov dx,offset err1_ms
        call show_message       ;Error to screen
        jmp monitor
;
Ok_2_Send:
        pop ax                          ;Retrieve byte
;Place in transmitter hoding register to send
        mov dx,Card_Base        ;THR register
        out dx,al               ;Send
        jmp short $+2           ;I/O delay
;Display character
        call tty
        jmp monitor
;-------------------------------------------------------------------------|
```

```
;                          PROCEDURES                                    |
;--------------------------------------------------------------------------|
Comm_On         proc    near
;Set communication line for interrupt operation received data
        cli                             ;interrupt off
;Reset buffer pointer to start of buffer
        mov data_in,0
        mov data_out,0
;
;Set dx to base address of RS 232 card from BIOS
        mov dx,card_base
;
;Init mode control register for data terminal ready
;(bit 0) request to send (bit1) and output 2 (bit3)
;DX is still holding port address
        mov dl,0fch             ;MCR address
        mov al,00001011b        ;Bit 0,1 and 3 set
        out dx,al
        jmp short $+2
;
;Set bit 7 of the line control register (DLAB) to access
;the interrupts enable register at xF9h
        mov dl,0fbh             ;xFBH =line control register
        in al,dx                ;Read byte at port
        jmp short $+2           ;I/O delay
        and al,7fh              ;Reset DLAB
        out dx,al               ;Write to LCR
        jmp short $+2           ;I/O delay
;
;Enable interrupts for DATA READY only
        mov dl,0f9h             ;Interrupt enable register
        mov al,1                ;Data ready interrupt
        out dx,al
        jmp short $+2           ;I/O delay
;
;Enable communications interrupts by resetting the bits
;corresponding to the irq3 and iqr4 line on the interrupt mask
;register(port address =21h)
        in al,21h               ;Read byte at port
        jmp short $+2           ;I/O delay
        and al,0e7h             ;Reset bit 3 and bit 4
        out 21h,al
```

```
        jmp short $+2                ;I/o delay
;
;Reenable interrupt
        sti
        ret
Comm_on             Endp
;
;--------------------------------------|
;       Communication line off  |
;--------------------------------------|

Comm_off    proc    near
;Disable communications interrup by setting for irq3 and iqr4 line
;on the interrupt mask register (port address=21)
        in al,21h
        or al,18h               ;Set bit 3 and 4
        out 21h,al
        jmp short $+2
        ret
Comm_off    Endp
;
Show_Message      proc    near
;Display string ->by the DX register using Dos function 09h
        mov ah,9                ;Service request number
        int 21h                 ;Dos interrupt
        ret
Show_Message      Endp
;
;----------------------------|
;       Teletype write  |
;----------------------------|
tty     proc    near
;Display character or control code at cursor position
tty_one:
        push ax                 ;Save character
        mov ah,14               ;Bios service request number
                                ;for ASCII teletype write
        mov bx,0                ;Display page
        int 10h                 ;Bios service request
        pop ax
;
;Test for carriage return and add line feed
```

```
        cmp al,0dh
        jne not_cr
        mov al,0ah
        jmp tty_one
not_cr:
        ret
tty     endp
;
;-------------------------|
;       Flush Buffer      |
;-------------------------|
flush   proc    near
flush_1:
        mov ah,1            ;Bios service request code
        int 16h
        jz no_old_chars
;Flush old character
        mov ah,0
        int 16h
        jmp flush_1
no_old_chars:
        ret
flush   endp
;
get_key     proc    near
        mov ah,0            ;Bios service request number
        int 16h
        ret
get_key             endp
;
;-----------------------------------------------------------------------------|
;               Interrupt Service Routine                                     |
;-----------------------------------------------------------------------------|
rs232_int:
        sti                 ;interrupt on
                            ;communications
;Save register to be used by the service routine
        push ax
        push bx
        push dx
        push di
        push ds
```

```
;Set Ds establish addressability of main program data
        mov dx,data
        mov ds,dx
        assume ds:data
;
;Check line status register for reception error and data ready
Data_check:
        mov dx,card_base
        mov dl,0fdh         ;line status register
        in al,dx            ;Read port byte
        jmp short $+2       ;I/O delay
;Check for error codes
        test al,1eh
        jnz data_error
        jmp data_check
;
data_error:
        mov al,'?'          ;Error symbol
        jmp store_byte
;
;Pull data from the receiver data register and store in
;the circular buffer
Data_Ready:
        mov dl,0f8h         ;RDR
        in al,dx            ;Get byte
        jmp short $+2       ;I/O delay
        and al,7fh          ;Mask off high bit
;
;Place byte in circular buffer
Store_Byte:
        lea di,circ_buf     ;Buffer pointer
        mov bx,data_in      ;Input pointer
        add di,bx           ;Point Di to active byte
        mov byte ptr[di],al ;Store in Circ_Buf
;
;Index input pointer. Reset if pointer overflows buffer
        inc bx              ;Bump pointer
        cmp bx,20           ;Past end of buffer ?
        jne ok_in_ptr
;
;Reset pointer to start of buffer
        mov bx,0
```

```
ok_in_ptr:
        mov data_in,bx      ;Store new pointer displacement
;
;Signal end of -interrupt to the interrupt command register
        mov al,20h          ;Code
        out 20h,al          ;EOl port address
        jmp short $+2       ;I/O delay
;
;Requset register from stack
        pop ds
        pop di
        pop dx
        pop bx
        pop ax
;
;Return from interrupt
        iret
code    ends
        End start
```

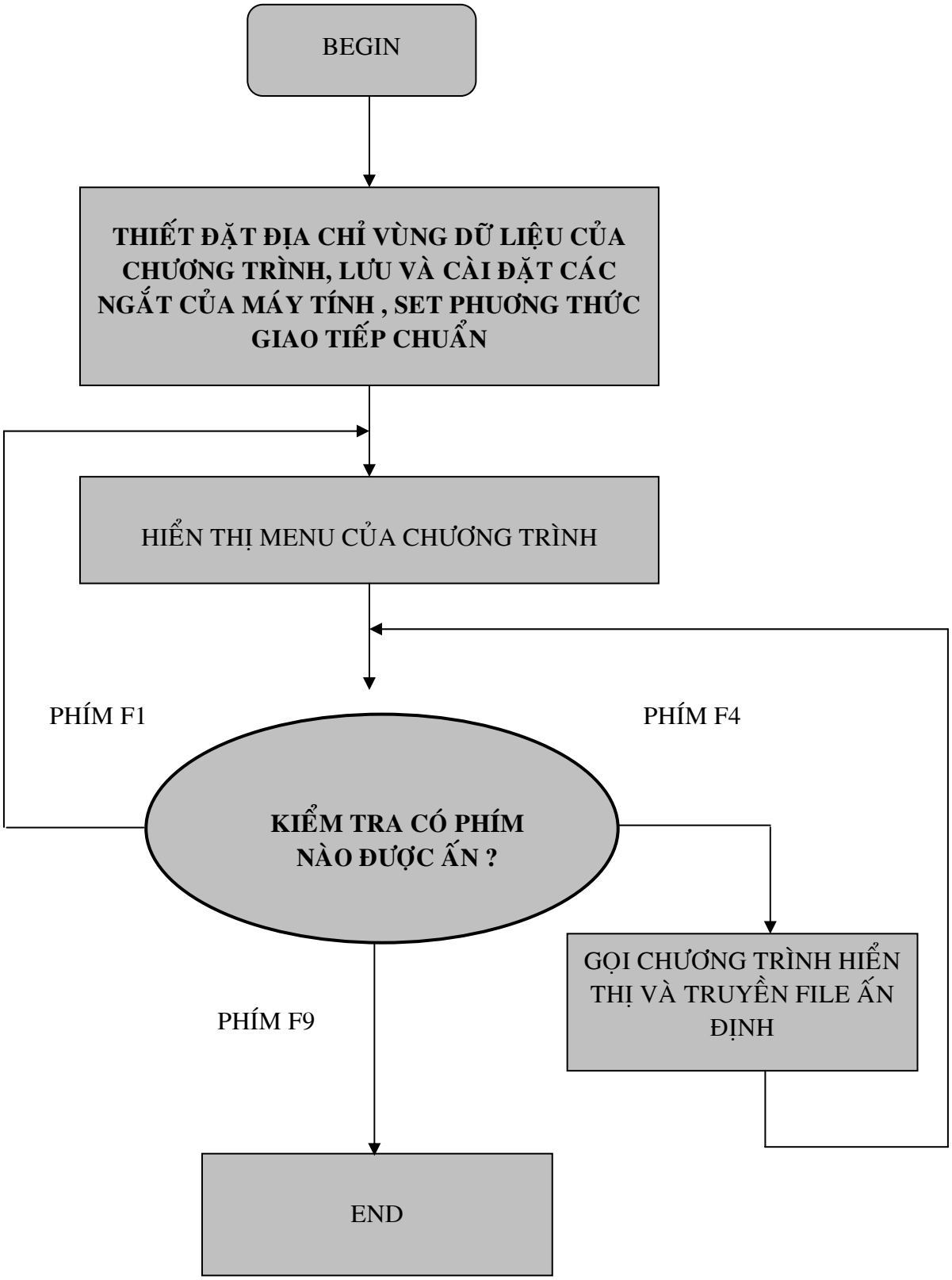
♦ **BÊN NHẬN (KIT VĐK 8051):**

```
        org 5000h
        setb p1.1
        mov IE,#00h         ;cam ngat
        mov tmod,#20h
        mov th1,#-13
        setb tr1
        mov scon,#0fch      ;khoi tao giao tiep noi tiep
        mov dptr,#6000h
xr1:    jnb ri,xr1
        clr ri              ;xoa vi da co du lieu
        mov a,sbuf          ;lay du lieu tu may phat goi toi
        movx @dptr,a        ;cat du lieu
        inc dptr
        mov a,dpl
        cjne a,#0ffh,xr1
        sjmp $              ;nhay tai cho
end
```

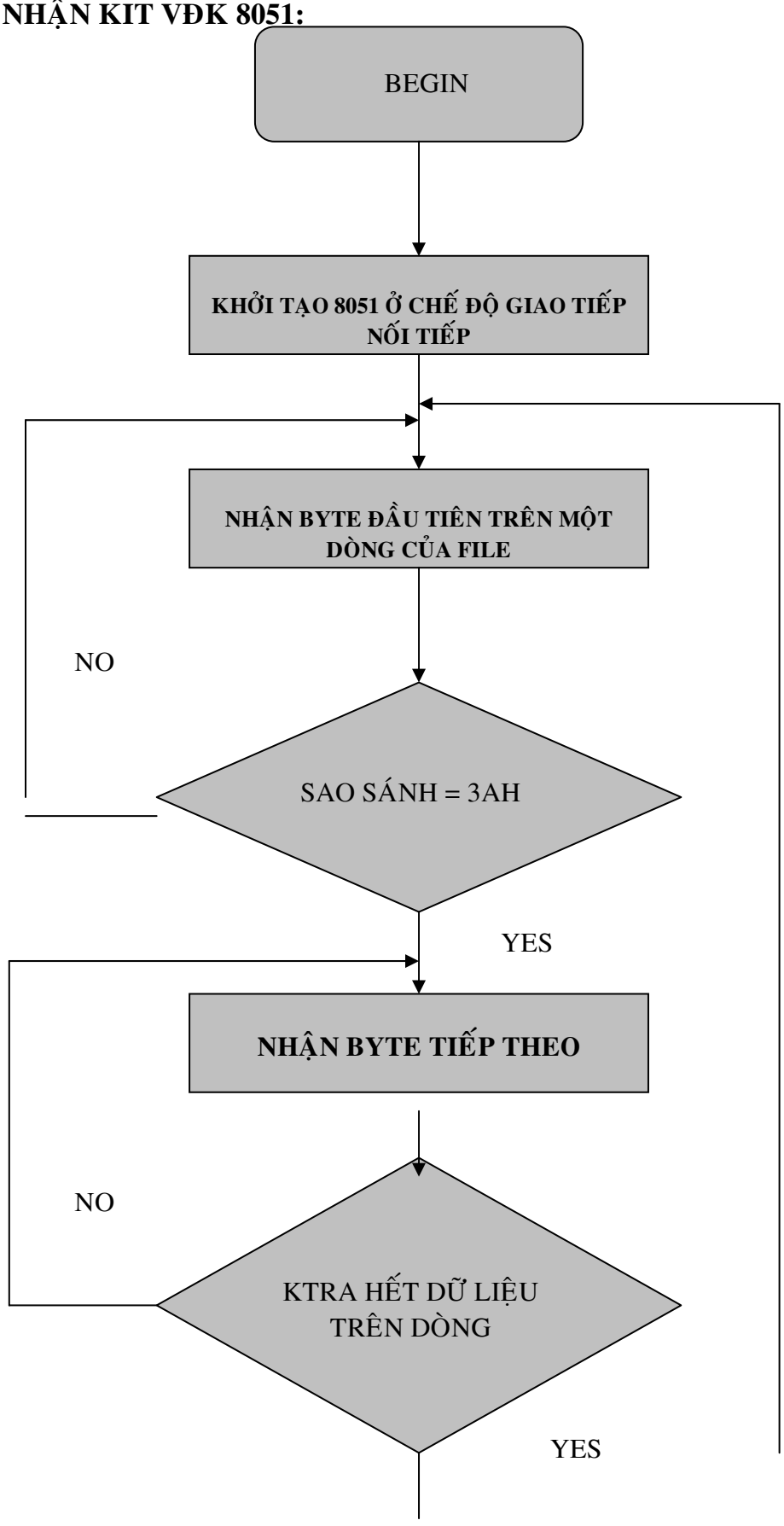
B2. TRUYỀN FILE DỮ LIỆU TỪ MÁY TÍNH ĐẾN KIT VĐK

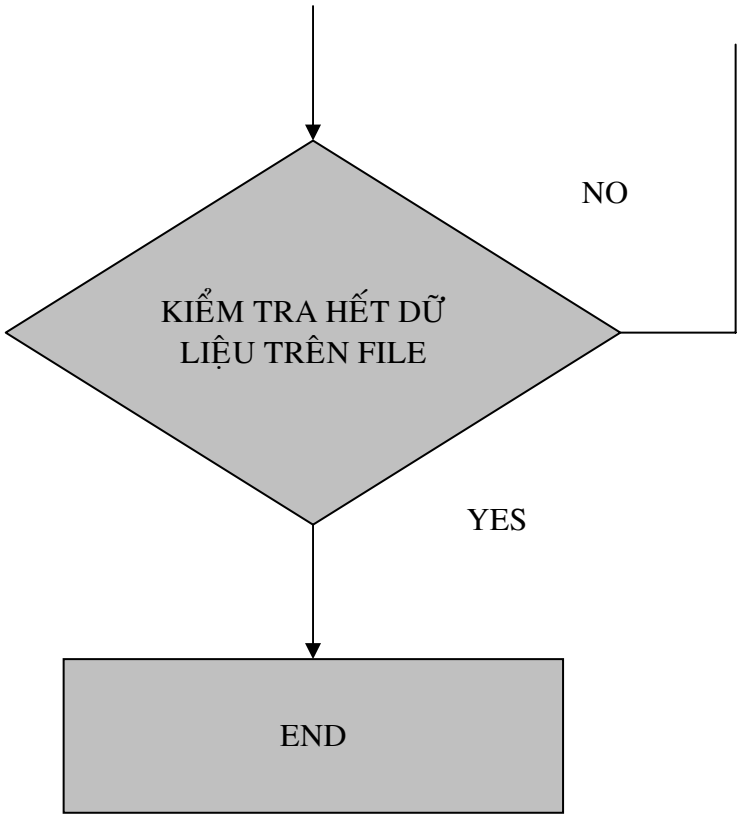
1. LƯU ĐỒ

♦ **BÊN TRUYỀN (MÁY TÍNH):**



♦ **BÊN NHẬN KIT VĐK 8051:**





2. CHƯƠNG TRÌNH

♦ **BÊN TRUYỀN (MÁY TÍNH)**

CHƯƠNG TRÌNH TRUYỀN FILE DỮ LIỆU (*.HEX)

Dựa vào chương trình truyền từng byte kí tự có sự thêm vào một số hàm

```
;       TERMINAL.ASM
;               STACK
stack   segment     stack
                db      300 dup('?')
stack   ends
;               DATA
data    segment
        prompt              db      'File name:$'
        filename    db      30 dup(0)
        buffer      db      512 dup(0)
        buffrr      db      512 dup(0)
        handle              dw      ?
        openerr             db      0dh,0ah,'OPEN ERROR  - COPDE'
        errcode             db      30h,'$'
;messages
        MENU_MS  db     '       **TERMINAL PROGRAM**',0dh,0ah
        db              '<F1> to redisplay this MENU',0dh,0ah
```

```
        db          '<F4> to open file.hex and trasnmit',0dh,0ah
        db          '<F9> to exit the TERMINAL program',0dh,0ah
        db          0dh,0ah,'$'
PROT_MS         db          ' ',0dh,0ah
err1_ms         db          0dh,0ah,'*** cannot transmit ***',0dh,0ah
card_base   dw      02f8h       ;address of RS 232 card
inT_num     db      0ch
setup_byte  db      0bbh        ;Origin
O_int_seg   dw      0000h       ;segment
O_int_off   dw      0000h       ;offset
;circular buffer and pointer:
circ_buf    db      20 dup(00h)
            dw      0
data_in     dw      0           ;imput pointer
data_out    dw      0           ;output pointer

        data    ends
```

phần code giống có thay đổi như sau:

```
;---------------------------------------- CODE  ----------------------------
;-----------------|
;Process char  |
;-----------------|
;Receive character type from keyboard buffer
char_typed:
        mov ah,0            ;Code for read keyboard char
        int 16h                     ;Bios service
;Test for <F1> ,<F4> and <F9> keys
        cmp ax,3b00h                ;<F1>
        jne test_f4
        jmp show_menu               ;<F1> key pressed
Test_F4:
        cmp ax,3e00h
        jne test_F9
        jmp tran_file               ;<F4> key pressed
Test_F9:
        cmp ax,4300h                ;<F9>
        je dos_exit
        jmp show_and_send           ;<F9> key pressed
;
```

Phần procedure thêm vao một số chương trình con

```
;-------------------------------------------------------------------------------|
;                              PROCEDURES                                       |
;-------------------------------------------------------------------------------|
tran_file:
        call get_name           ;doc ten file
        lea dx,filename         ;dx chua offset cua ten file
        mov al,0
        call open
        jc open_error
        mov handle,ax
read_loop:
        lea dx,buffer           ;tro toi vung dem
        mov bx,handle           ;lay the file
        call read               ;doc file,AX = so byte doc duoc
        or ax,ax                ;ket thuc file
        je pexit                ;dung, ket thuc file
        mov cx,ax               ;CX chua so byte doc duoc
        call display            ;hien thi file
        jmp read_loop           ;lap lai
open_error:
        lea dx,openerr          ;lay thong bao loi
        add errcode,al
        mov ah,9
        int 21h                 ;hien thi thong bao loi
;output and display
;show_and_send:
;
pexit:
        mov cx,2000
pthre_wait:
        mov dx,card_base
        add dx,5
        in al,dx
        jmp short $+2
        test al,20h
        jnz pok_2_send
        loop pthre_wait
;wait period timed out,display error message and exit
        mov dx,offset err1_ms
        call show_message
        jmp ppexit
pok_2_send:
```

```
        call con_hex            ;goi chtr con chuyen sang so hex
        lea dx,buffrr
        mov cx,256
        call display
        mov cx,256
;place in transmitter hoding register to send
        mov dx,card_base
        lea di,buffrr           ;tro toi vung dem
ppl:    mov al,[di]             ;lay byte data
        out dx,al
        jmp short $+2
        call edelay
        inc di
        loop ppl
;display character
;       call tty
ppexit:     mov bx,handle       ;lay the file
        call close              ;dong the file
        jmp monitor
;-------------------------------------
get_name    proc    near
        push ax
        push dx
        push di
        mov ah,9                ;ham hien thi chuoi
        lea dx,prompt
        int 21h
        cld
        lea di,filename         ;DI tro toi ten file
        mov ah,1                ;ham doc ki tu tu ban phim
read_name:
        int 21h
        cmp al,0dh    ;co phai CR
        je done                 ;dung ket thuc
        stosb                   ;luu no vao trong chuoi
        jmp read_name           ;tiep tuc doc vao
done:  mov al,0
        stosb                   ;luu byte 0
        pop di
        pop dx
        pop ax
        ret
```

```
get_name    endp
open  proc  near
      mov ah,3dh            ;ham mo file
      mov al,0              ;chi doc
      int 21h
      ret
open  endp
read  proc  near
      push cx
      mov ah,3fh            ;ham mo file
      mov cx,512            ;chi doc
      int 21h
      pop cx
      ret
read  endp
display     proc  near
      push bx
      mov ah,40h            ;ham ghi file
      mov bx,1              ;the file cho man hinh
      int 21h               ;dong file
      pop bx
      ret
display     endp

close proc  near
      mov ah,3eh            ;ham dong file
      int 21h               ;dong file
      ret
close endp
edelay      proc  near
      push ax
      push bx
      mov ax,06h
edel2: mov bx,0ffffh
edel1: dec bx
      jnz edel1
      dec ax
      jnz edel2
      pop bx
      pop ax
      ret
edelay      endp
```

```
;--------------------
con_hex     proc   near
        push ax
        push bx
        push cx
        push dx
        push di
        push si
        mov ax,0b800h
        mov bx,0
        cld
        lea si,buffrr
        mov cx,260
        mov al,0
xxx8:   mov [si],al
        inc si
        loop xxx8
        lea si,buffrr
        lea di,buffer
xxx3:   mov al,[di]             ;lay byte data
        cmp al,3ah              ;so sanh voi ma dau ':'
        jz xxx2                 ;nhay neu la dau ':'
        inc di
        jmp xxx3                ;quay lai de tim dau ':'
xxx2:   call ktra_end           ;goi chuong trinh kiem tra ket thuc
        cmp ax,0                ;dung la het data thi lam cho AX=0000
        jnz xxx4
xxx6:   pop si
        pop di
        pop dx
        pop cx
        pop bx
        pop ax
        ret
xxx4:   mov al,[di]             ;lay byte data
        mov [si],al             ;dung la byte 3Ah can luu vao
        call goi_ht
        inc di
        inc si
;xu li so byte can goi
        mov al,[di]             ;lay so can goi MSD
        sub al,30h              ;tru di 30 de thanh so hex
```

```
        call so_lon             ;kiem tra so ABCDEF
        mov cl,4
        rol al,cl
        mov ah,al
        inc di
        mov al,[di]             ;lay so can go LSD
        sub al,30h              ;tru di 30 de thanh so hex
        call so_lon             ;kiem tra so ABCDEF
        or al,ah                ;or 2 data lai thanh 1 byte
        mov [si],al             ;cat so HEX ADDR_H
        add al,1
        mov dl,al               ;luu so byte can xu li con lai
        call goi_ht
;xu li phan dia chi can goi
        inc di
        inc si
        mov al,[di]             ;lay byte ADDR_L-MSD
        sub al,30h              ;tru di 30 de thanh so hex
        call so_lon             ;kiem tra so ABCDEF
        mov cl,4
        rol al,cl
        mov ah,al
        inc di
        mov al,[di]             ;lay byte ADDR_L-LSD
        sub al,30h              ;tru di 30 de thanh so hex
        call so_lon             ;kiem tra so ABCDEF
        or al,ah                ;or 2 data lai thanh 1 byte
        mov [si],al             ;cat so HEX ADDR_H
        call goi_ht
        inc di
        inc si
        mov al,[di]             ;lay byte ADDR_H-MSD
        sub al,30h              ;tru di 30 de thanh so hex
        call so_lon             ;kiem tra so ABCDEF
        mov cl,4
        rol al,cl
        mov ah,al
        inc di
        mov al,[di]             ;lay byte  ADDR_H-LSD
        sub al,30h              ;tru di 30 de thanh so hex
        call so_lon             ;kiem tra so ABCDEF
        or al,ah                ;or 2 data lai thanh 1 byte
```

```
        mov [si],al             ;cat so byte can goi dang HEX
        call goi_ht
;xu li cac byte con lai
        inc di                  ;bo byte 00
        inc di
xxx1:   inc di
        inc si
        mov al,[di]             ;lay so byte thu nhat
        sub al,30h              ;tru di 30 de thanh so hex
        call so_lon             ;kiem tra so ABCDEF
        mov cl,4
        rol al,cl
        mov ah,al
        inc di
        mov al,[di]             ;lay so byte thu 2
        sub al,30h              ;tru di 30 de thanh so hex
        call so_lon             ;kiem tra so ABCDEF
        or al,ah                ;or 2 data lai thanh 1 byte
        mov [si],al             ;cat so byte can goi dang HEX
        call goi_ht
        dec dl
        cmp dl,0
        jnz xxx1                ;quay lai vi chua
        inc di                  ;bo byte cuoi thu nhat
        inc di                  ;bo byte thu 2
        inc di                  ;bo ma xuong hang
        jmp xxx3                ;het 1 hang data
con_hex     endp
;chuong trinh kiem tra ket thuc
ktra_end proc       near
        push di
        inc di
        mov al,[di]             ;lay byte data
        cmp al,30h              ;kiem tra byte 0 thu nhat
        jnz kt_exit             ;nhay den de thoat vi khong phai
        inc di                  ;neu dung thi kiem tra byte 0 thu 2
        mov al,[di]             ;lay byte data
        cmp al,30h
        jnz kt_exit             ;nhay den de thoat vi khong phai
        inc di                  ;neu dung thi kiem tra byte 0 thu 3
        mov al,[di]             ;lay byte data
        cmp al,30h
```

```
        jnz kt_exit             ;nhay den de thoat vi khong phai
        inc di                  ;neu dung thi kiem tra byte 0 thu 4
        mov al,[di]             ;lay byte data
        cmp al,30h
        jnz kt_exit             ;nhay den de thoat vi khong phai
        inc di                  ;neu dung thi kiem tra byte 0 thu 5
        mov al,[di]             ;lay byte data
        cmp al,30h
        jnz kt_exit             ;nhay den de thoat vi khong phai
        inc di                  ;neu dung thi kiem tra byte 0 thu 6
        mov al,[di]             ;lay byte data
        cmp al,30h
        jnz kt_exit             ;nhay den de thoat vi khong phai
        inc di                  ;neu dung thi kiem tra byte 0 thu 7
        mov al,[di]             ;lay byte data
        cmp al,30h
        jnz kt_exit             ;nhay den de thoat vi khong phai
        inc di                  ;neu dung thi kiem tra byte 1 thu 8
        mov al,[di]             ;lay byte data
        cmp al,31h
        jnz kt_exit             ;nhay den de thoat vi khong phai
        inc di                  ;neu dung thi kiem tra byte F thu 9
        mov al,[di]             ;lay byte data
        cmp al,46h
        jnz kt_exit             ;nhay den de thoat vi khong phai
        inc di                  ;neu dung thi kiem tra byte F thu 10
        mov al,[di]             ;lay byte data
        cmp al,46h
        jnz kt_exit             ;nhay den de thoat vi khong phai
        mov ax,0
        pop di
        ret
kt_exit:
        mov ax,1111h            ;nap data sao cho khac khong la 1
        pop di
        ret
ktra_end    endp
goi_ht proc near
        RET
        push di
        mov di,bx
        mov ah,1
```

```
        stosw
        mov bx,di
        pop di
        ret
goi_ht  endp
so_lon  proc near
        cmp al,9
        jg yyy
        ret
yyy:    sub al,7
        ret
so_lon  endp
```

♦ **BÊN NHẬN (KIT VĐK 8051):**

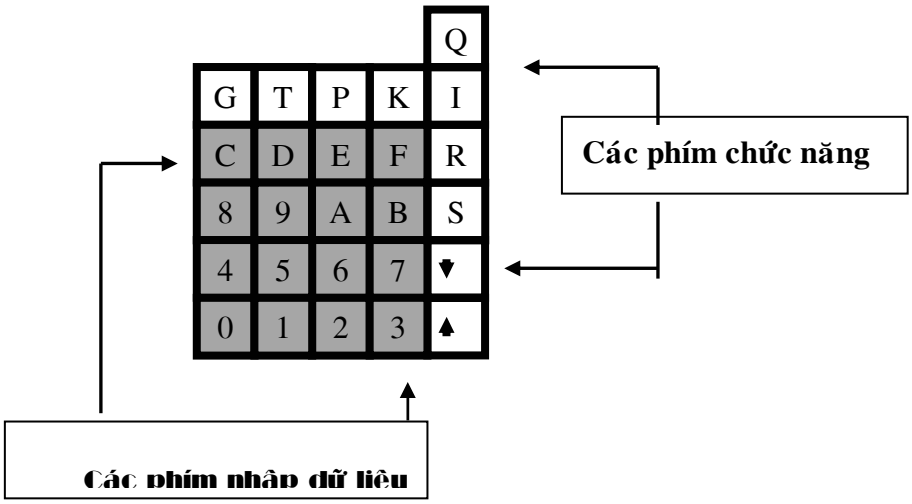
```
;chuong trinh cua may nhan du lieu (receiver)
        dk      equ     0c001h
        ht      equ     0c000h
        org 5200h
        mov ie,#00h
        mov tmod,#20h
        mov th1,#-13
        setb tr1
        mov scon,#0fch
        mov 24h,#00
        mov 25h,#00
xr1:    jnb ri,xr1      ;nhan ma dau ':'
        clr ri
        mov a,sbuf
        cjne a,#3ah,xr1
xr2:    jnb ri,xr2      ;nhan so byte can goi
        clr ri
        mov r1,sbuf
xr3:    jnb ri,xr3
        clr ri
        mov dph,sbuf
xr4:    jnb ri,xr4
        clr ri
        mov dpl,sbuf
xr5:    jnb ri,xr5
        clr ri
        mov 11h,dph ;chuyen dia chi de giai ma hthi
        mov 10h,dpl
        mov a,sbuf
```

```
        movx @dptr,a
        mov 14h,a     ;cat de giai ma hien thi
        lcall decode
        inc dptr
        djnz r1,xr5
        sjmp xr1



;chuong trinh con giai ma hien thi
decode:     push dpl
        push dph
        mov dptr,#0200h
        mov a,11h     ;lay byte B_A_H de giai ma hien thi
        push 0e0h     ;cat A
        swap a        ;xu li so thu nhat
        anl a,#0fh
        mov dpl,a
        movx a,@dptr
        mov 20h,a
        pop 0e0h      ;lay lai A
        anl a,#0fh    ;xu li so thu 2
        mov dpl,a
        movx a,@dptr
        mov 21h,a
        mov a,10h     ;lay byte B_A_L de giai ma hien thi
        push 0e0h     ;cat A
        swap a        ;xu li so thu nhat
        anl a,#0fh
        mov dpl,a
        movx a,@dptr
        mov 22h,a
        pop 0e0h      ;lay lai A
        anl a,#0fh    ;xu li so thu 2
        mov dpl,a
        movx a,@dptr
        mov 23h,a
        mov a,14h     ;lay byte DATA de giai ma hien thi
        push 0e0h     ;cat A
        swap a        ;xu li so thu nhat
        anl a,#0fh
        mov dpl,a
```

```
        movx a,@dptr
        mov 26h,a
        pop 0e0h      ;lay lai A
        anl a,#0fh    ;xu li so thu 2
        mov dpl,a
        movx a,@dptr
        mov 27h,a
        lcall display
        pop dph
        pop dpl
        ret
display:
        mov r2,#80h         ;tu dieu khien 8279 chong nhap nhay
        mov r0,#20h         ;quan li dia chi ma hien thi
dis1:   mov dptr,#dk
        mov a,r2
        movx @dptr,a
        mov dptr,#ht
        mov a,@r0
        movx @dptr,a
        inc r2
        inc r0
        mov a,r0
        cjne a,#28h,dis1
        ret
end
```

CHƯƠNG 1

HƯỚNG DẪN SỬ DỤNG KIT VI ĐIỀU KHIỂN 8051



I. GIỚI THIỆU CẤU TRÚC PHẦN CỨNG KIT VI XỬ LÝ:

1. Tần số làm việc:

- Kít vi điều khiểnsử dụng vi điều khiển 8051 hoặc 8951 của Intel với tần số hoạt động 12MHZ.
- Các chương trình về thời gian được viết tương ứng với địa chỉ này.

2. Tổ chức bộ nhớ:

a. Bộ nhớ EPROM:

Có dung lượng 16kbyte sử dụng 2 EPROM 2764, chương trình hệ thống chứa ở EPROM thứ nhất, EPROM thứ 2 chưa sử dụng được thiết kế ở dạng socket.

- EPROM 1 có địa chỉ từ 0000_H - $1FFF_H$.
- EPROM 2 có địa chỉ từ 2000_H - $3FFF_H$.

b. Bộ nhớ RAM:

Bộ nhớ RAM có dung lượng 16kbyte sử dụng 2 IC 6264.

- RAM 1 có địa chỉ từ 4000_H - $5FFF_H$.
- RAM 2 có địa chỉ từ 6000_H - $7FFF_H$.
- Chương trình có thể sử dụng toàn bộ các vùng nhớ RAM.

3. **Các IC ngoại vi:** trong hệ thống có sử dụng 2 IC 8255A dùng để giao tiếp với thiết bị ngoại vi.

Bảng đồ nhớ của 2 IC 8255:

Địa chỉ của các port	**8255_1**	**8255-2**
Port A	8000H	A000H
Port B	8001H	A001H
Port C	8002H	A002H
Thanh ghi điều khiển	8003H	A003H

- Các ngõ ra của IC 8255A -1, 8255 -2, được đưa ra bên ngoài bằng connect 64 chân có sơ đồ chân tra ở bảng tra. Mỗi IC 8255A có 3 port, mỗi port có 8 chân điều khiển nên số chân đưa ra bên ngoài để điều khiển là 48.

4. Khối giải mã hiển thị – quét phím sử dụng IC 8279:

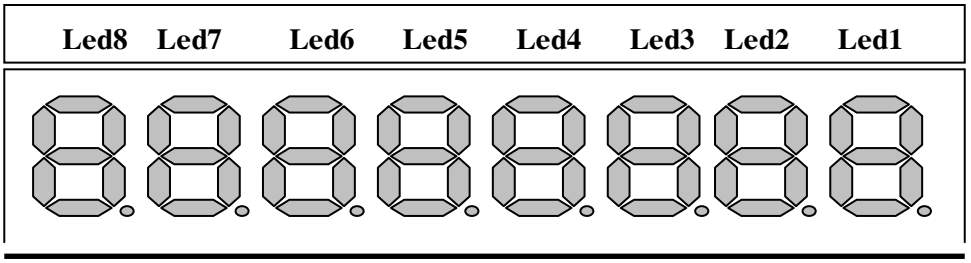
Vùng địa chỉ sử dụng của IC 8279 là $C000_H$ - $C001_H$, trong đó:

- Địa chỉ C000H là địa chỉ dùng để gởi **dữ liệu cần hiển thị và đọc mã phím**.
- Địa chỉ C001H là địa chỉ dùng để gởi **từ điều khiển** ra 8279 – đọc thanh ghi trạng thái.

a. Phần giải mã hiển thị:

Gồm có 8 led với thứ tự Led 1 đến led 8 theo hướng từ phải sang trái như hình 2:

Hình 2.



- Cấu trúc byte dữ liệu của led:

p	g	f	e	d	c	b	a

Hệ thống sử dụng Led loại Anode chung nên muốn đoạn nào sáng thì bit dữ liệu tương ứng với đoạn đó bằng 1. Đoạn nào tắt thì bit tương ứng với đoạn đó bằng 0.

Ví dụ muốn sáng số "9" thì byte dữ liệu sẽ gởi ra led là:

0	1	1	0	1	1	1	1

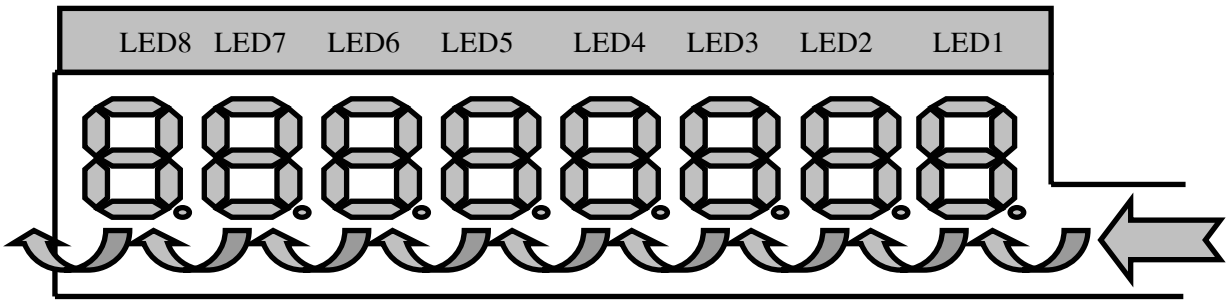
Tương ứng với số hex là 6FH. Sau đây là mã 7 đoạn của 1 số chữ số và chữ cái:

	p	g	f	e	d	c	b	a	hex
Số 0	0	0	1	1	1	1	1	1	3F
Số 1	0	0	0	0	0	1	1	0	06
Số 2	0	1	0	1	1	0	1	1	5B
Số 3	0	1	0	0	1	1	1	1	4F
Số 4	0	1	1	0	0	1	1	0	66
Số 5	0	1	1	0	1	1	0	1	6D
Số 6	0	1	1	1	1	1	0	1	7D
Số 7	0	0	0	0	0	1	1	1	07
Số 8	0	1	1	1	1	1	1	1	7F
Số 9	0	1	1	0	1	1	1	1	6F
Chữ A	0	1	1	1	0	1	1	1	77
Chữ b	0	1	1	1	1	1	0	0	7C
Chữ C	0	0	1	1	1	0	0	1	39
Chữ d	0	1	0	1	1	1	1	0	5E
Chữ E	0	1	1	1	1	0	0	1	79
Chữ F	0	1	1	1	0	0	0	1	71
Chữ P	0	1	1	1	0	0	1	1	73
Chữ H	0	1	1	1	0	1	1	0	76
Chữ U	0	0	1	1	1	1	1	0	3E

Có thể tìm các mã tương ứng còn lại.

- Có 2 cách hiển thị dữ liệu trên các Led:

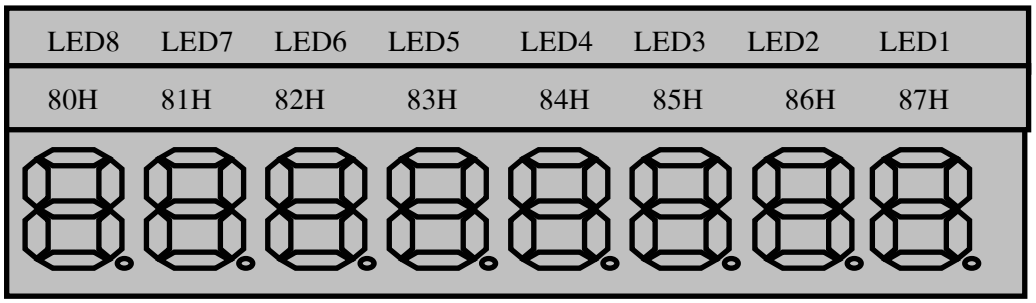
- ***Cách 1:*** khi gởi dữ liệu mới ra địa chỉ C000H thì dữ liệu này sẽ hiển thi ở led 1, dữ liệu trước đó của các led sẽ dịch sang trái theo chiều mũi tên trong hình 3. Riêng byte dữ liệu trước đó của Led8 sẽ dịch và mất đi.



Mũi tên nằm ngang chỉ chiều nhận dữ liệu từ vi điều khiển đưa đến led 1. Các mũi tên vòng cung chỉ chiều dịch chuyển dữ liệu.

> ***Chú ý:*** nếu muốn xóa hết màn hình 8 led thì gởi 8 byte 00h liên tiếp ra A000h.

- **Cách 2**: kiểu gởi dữ liệu ở cách 1 còn được gọi là kiểu dịch chuyển dữ liệu tuần tự. Bên cạnh đó 8279 còn cho phép gởi dữ liệu trực tiếp đến bất kỳ led nào trong 8 led – tổ chức của led không có gì thay đổi địa chỉ gởi dữ liệu vẫn là C000H nhưng mỗi led còn có thêm 1 địa chỉ điều khiển như trong hình 4. Địa chỉ điều khiển của led phải gởi ra địa chỉ C001H trước khi gởi dữ liệu ra địa chỉ C000H.



b. Phần giải mã bàn phím:

Chương trình con giải mã bàn phím được viết tại địa chỉ 0223H **sử dụng các thanh ghi R2, A, DPTR, R6, R7, ô nhớ 41h.** Khi gọi chương trình con 0223H:

- Nếu không ấn phím thì sau khi thực hiện xong chương trình sẽ trở về chương trình chính với nội dung thanh ghi A =FFH.
- Nếu có ấn phím thì mã của phím ấn chứa trong A.

Chương trình này nếu có ấn phím hoặc không ấn phím đều trở về chương trình sau khi thực hiện xong và phải chú ý cất dữ liệu trong các thanh ghi khi gọi chương trình con này.

Bảng mã các phím số:

Phím	Mã		Phím	Mã		Phím	Mã		Phím	Mã
0	00		4	04		8	08		C	0C
1	01		5	05		9	09		D	0D
2	02		6	06		A	0A		E	0E
3	03		7	07		B	0B		F	0F

Bảng mã các phím chức năng:

Phím	Mã		Phím	Mã
T	10		S	14
G	11		↑	15
R	12		P	16
↓	13		K	17

II. HƯỚNG DẪN SỬ DỤNG KIT VI ĐIỀU KHIỂN 8051

1. Bàn phím:

- Kít vi điều khiển có tất cả là 26 phím nhấn như hình 1 được chia thành các nhóm như sau:

- 16 phím nhập dữ liệu của chương trình dạng số thập lục phân từ 0 đến F
- Các phím chức năng.

2. Chức năng của phím: Q

- Khi mới cấp điện cho máy 4 Led bên trái sẽ hiện thị 4 số 0000, bốn led bên phải tắt.
- Nếu không hiển thị đúng hãy nhấn phím "Q". Phím "Q" có chức năng Reset mạch khi khởi động hoặc khi muốn thoát khỏi chương trình vi điều khiển đang thực hiện (chức năng như phím RESET của máy vi tính).

3. Chức năng của phím: S

- Muốn nhập dữ liệu mới vào ô nhớ có địa chỉ ví dụ 4000, hãy dùng các phím nhập dữ liệu đánh số 4000, địa chỉ này sẽ xuất hiện ở 4 led bên phải.
- Nhấn phím "S" thì địa chỉ 4000 sẽ thay thế cho địa chỉ trước đó ở 4 led bên trái.
- 4 led còn lại chỉ có 2 led sáng đó chính là nội dung của ô nhớ tương ứng với địa chỉ 4 led bên trái.

4. Chức năng của phím: ↑

- Dùng để lưu trữ dữ liệu vào ô nhớ có địa chỉ ở 4 Led bên trái, ví dụ muốn lưu trữ dữ liệu là "3F" vào ô nhớ có địa chỉ là 4000, hãy đánh "3F" từ các phím dữ liệu, dữ liệu mới "3F" sẽ thay thế dữ liệu cũ trước đó.
- Sau đó nhấn phím " ↑ " để lưu trữ dữ liệu này vào ô nhớ 4000. Địa chỉ sẽ tăng lên 1 là 4001 để sẵn sàng nhận dữ liệu tiếp theo và 2 led bên trái hiển thị nội dung của ô nhớ 4001.
- Chức năng của phím này là lưu trữ dữ liệu đồng thời tăng địa chỉ của ô nhớ.

5. Chức năng của phím: ↓

- Có chức năng giảm địa chỉ của ô nhớ xuống 1 đơn vị tương ứng với mỗi lần nhấn. Ví dụ muốn kiểm tra lại ô nhớ mới vừa nhập là 4000 xem có đúng là dữ liệu "3F" không, hãy nhấn phím " ↓ ". Nếu sai thì nhập lại, nếu đúng thì nhấn phím tăng địa chỉ để nạp các dữ liệu tiếp theo.

P

6. **Chức năng của phím:**
   - ♦ Sau khi nhập dữ liệu của một chương trình tại địa chỉ 4000, để vi điều khiển thực hiện chương trình này hãy nhấn phím "P". Khi đó trên màn hình 8 Led sẽ xuất hiện "PC 4000". Nếu muốn thực hiện chương trình tại địa chỉ 4000 hãy nhấn phím tăng địa chỉ, khi đó trên màn hình sẽ xuất hiện thêm dấu "=" như sau: "PC =4000'. Sau đó nhấn phím "G". Chương trình sẽ được thi hành.
   - ♦ Nếu chương trình lưu tại địa chỉ khác với địa chỉ 4000 thì trước khi nhấn phím tăng địa chỉ hãy đánh địa chỉ của chương trình đó vào bằng các phím nhập dữ liệu. Sau đó nhấn phím tăng địa chỉ, ví dụ muốn thực hiện chương trình tại địa chỉ 5000 thì trên màn hình 8 led sẽ hiển thị "PC =5000". Nhấn tiếp phím "G" chương trình sẽ được thi hành tại địa chỉ 5000.

7. **Chức năng của phím:** R
   - ♦ Dùng để xem nội dung các thanh ghi, trước tiên nhấn phím "R" và sau đó nhấn các phím thập phân tương ứng từ "6" cho đến "F".

- Nhấn phím thập phân "A": xem nội dung thanh ghi A.
- Nhấn phím thập phân "B": xem nội dung thanh ghi B.
- Nhấn phím thập phân "C": xem nội dung thanh ghi C.
- Nhấn phím thập phân "D": xem nội dung thanh ghi D.
- Nhấn phím thập phân "E": xem nội dung thanh ghi E.
- Nhấn phím thập phân "F": xem nội dung thanh ghi F.
- Nhấn phím thập phân "8": xem nội dung thanh ghi H.
- Nhấn phím thập phân "9": xem nội dung thanh ghi L.
- Nhấn phím thập phân "7": xem nội dung cặp thanh ghi SP.
- Nhấn phím thập phân "6": xem nội dung cặp thanh ghi PC.

8. **Chức năng của phím:** I
   - ♦ Phím này tác động đến ngắt cứng của hệ thống vi xử lý. Chương trình sẽ bị ngừng sau khi nhấn phím "I", nếu nhấn "I" thêm lần nữa hệ thống sẽ được đặt lại trạng thái mặc định ban đầu tương đương với reset máy bằng phím Q.

9. **Chức năng của phím:** T
   - ♦ Chức năng của phím này là thực hiện chương trình từng bước. Trình tự nhấn phím giống như phím "G". Nếu nhấn phím "G" để thực hiện cả chương trình tại địa chỉ chứa trong cặp thanh ghi PC, ta nhấn phím "T" chương trình sẽ được thực hiện từng lệnh tại địa chỉ chứa trong PC.