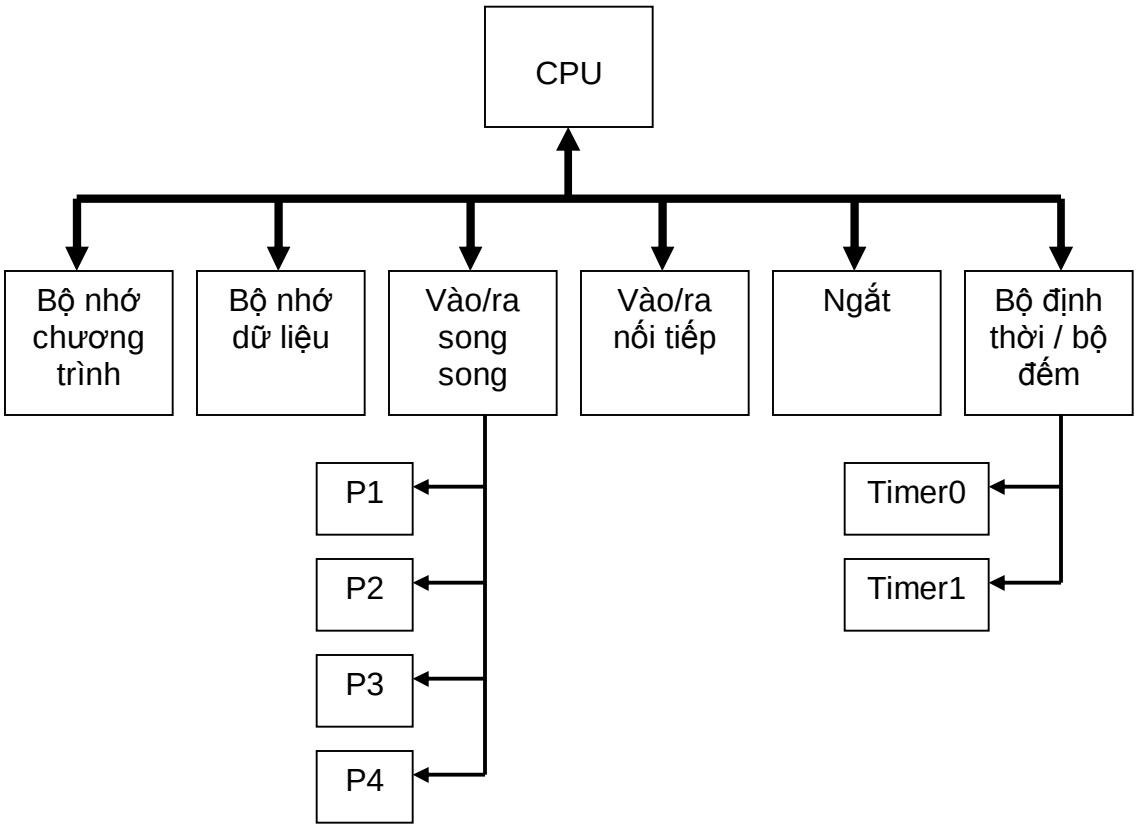
Chương I: Giới thiệu họ Vi điều khiển 8051

Vi điều khiển 8051 là một trong những vi điều khiển 8 bit thông dụng nhất hiện nay. Bắt đầu xuất hiện vào năm 1980, trải qua gần 30 năm, hiện đã có tới hàng trăm biến thể (derrivatives) được sản xuất bởi hơn 20 hãng khác nhau, trong đó phải kể đến các đại gia trong làng bán dẫn (Semiconductor) như ATMEL, Texas Instrument, Philips, Analog Devices… Tại Việt Nam, các biến thể của hãng ATMEL là AT89C51, AT89C52, AT89S51, AT89S52… đã có thời gian xuất hiện trên thị trường khá lâu và có thể nói là được sử dụng rộng rãi nhất trong các loại vi điều khiển 8 bit. Chương này sẽ tập trung mô tả tương đối chi tiết cấu trúc bên trong của các biến thể nói trên (tạm gọi chung là AT89) của hãng ATMEL.

Cấu trúc của AT89 ở dạng sơ đồ khối tổng quát



Cấu trúc bus

Bus địa chỉ của họ vi điều khiển 8051 gồm 16 đường tín hiệu (thường gọi là bus địa chỉ 16 bit). Với số lượng bit địa chỉ như trên, không gian nhớ của chip được mở rộng tối đa là 2^{16} = 65536 địa chỉ, tương đương 64K.

Bus dữ liệu của họ vi điều khiển 8051 gồm 8 đường tín hiệu (thường gọi là bus dữ liệu 8 bit), đó là lý do tại sao nói 8051 là họ vi điều khiển 8 bit. Với độ rộng của bus dữ liệu như vậy, các chip họ 8051 có thể xử lý các toán hạng 8 bit trong một chu kỳ lệnh.

CPU (Central Processing Unit)

CPU là đơn vị xử lý trung tâm, đó là bộ não của toàn bộ hệ thống vi điện tử được tích hợp trên chip vi điều khiển. CPU có cấu tạo chính gồm một đơn vị xử lý số học và lôgic ALU (Arithmethic Logic Unit) - nơi thực hiện tất cả các phép toán số học và phép lôgic cho quá trình xử lý.

Bộ nhớ chương trình (Program Memory)

Không gian bộ nhớ chương trình của AT89 là 64K byte, tuy nhiên hầu hết các vi điều khiển AT89 trên thị trường chỉ tích hợp sẵn trên chip một lượng bộ nhớ chương trình nhất định và chiếm dải địa chỉ từ 0000h trở đi trong không gian bộ nhớ chương trình.

AT89C51/AT89S51 có 4K byte bộ nhớ chương trình loại Flash tích hợp sẵn bên trong chip. Đây là bộ nhớ cho phép ghi/xóa nhiều lần bằng điện, chính vì thế cho phép người sử dụng thay đổi chương trình nhiều lần. Số lần ghi/xóa được thường lên tới hàng vạn lần.

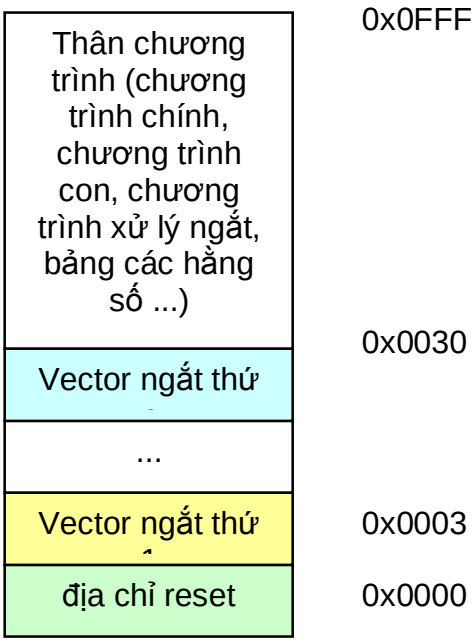
AT89C52/AT89S52 có 8K byte bộ nhớ chương trình cùng loại.

Bộ nhớ chương trình của các chip họ 8051 có thể thuộc một trong các loại: ROM, EPROM, Flash, hoặc không có bộ nhớ chương trình bên trong chip. Tên của từng chip thể hiện chính loại bộ nhớ chương trình mà nó mang bên trong, cụ thể là vài ví dụ sau:

STT	Tên chip	ROM	EPROM	Flash
1	8051	4 Kbyte	x	x
2	8052	8 Kbyte	x	x
3	8031	x	x	x
4	8032	x	x	x
5	87C51	x	4 Kbyte	x
6	87C52	x	8 Kbyte	x
7	AT89C51 / AT89S51	x	x	4 Kbyte
8	AT89C52 / AT89S52	x	x	8 Kbyte

Bộ nhớ chương trình dùng để chứa mã của chương trình nạp vào chip. Mỗi lệnh được mã hóa bởi 1 hay vài byte, dung lượng của bộ nhớ chương trình phản ánh số lượng lệnh mà bộ nhớ có thể chứa được. Địa chỉ đầu tiên của bộ nhớ chương trình (0x0000) chính là địa chỉ Reset của 8051. Ngay sau khi reset (do tắt bật nguồn, do mức điện áp tại chân RESET bị kéo lên 5V...), CPU sẽ nhảy đến thực hiện lệnh đặt tại địa chỉ này trước tiên, luôn luôn là như vậy. Phần còn trống trong không gian chương trình không dùng để làm gì cả. Nếu muốn mở rộng bộ nhớ chương trình, ta phải dùng bộ nhớ chương trình bên ngoài có dung lượng như ý muốn. Tuy nhiên khi dùng bộ nhớ chương trình ngoài, bộ nhớ chương trình onchip không dùng được nữa, bộ nhớ chương trình ngoài sẽ chiếm dải địa chỉ ngay từ địa chỉ 0x0000.

Hình ảnh minh họa bộ nhớ chương trình



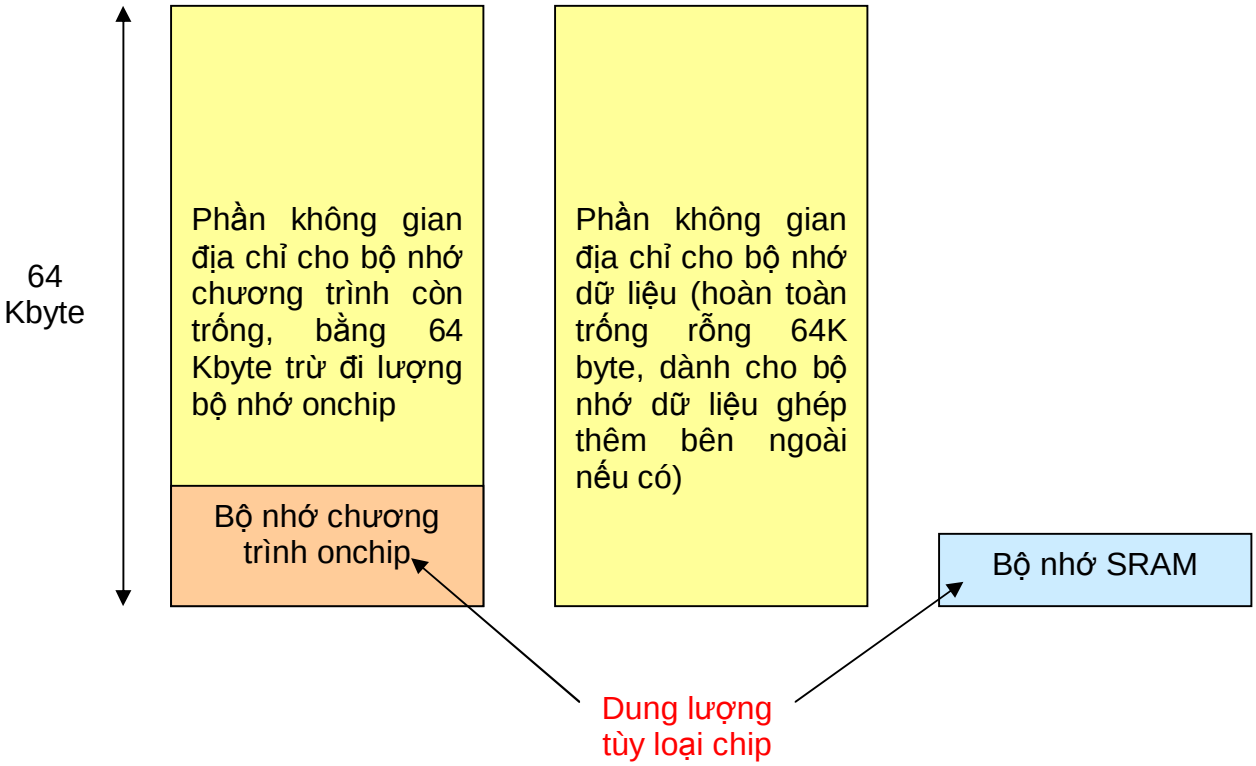
Bộ nhớ dữ liệu (Data Memory)

Vi điều khiển họ 8051 có không gian bộ nhớ dữ liệu là 64K địa chỉ, đó cũng là dung lượng bộ nhớ dữ liệu lớn nhất mà mỗi chip thuộc họ này có thể có được (nếu phối ghép một cách chính tắc, sử dụng các đường tín hiệu của bus địa chỉ và dữ liệu). Bộ nhớ dữ liệu của các chip họ 8051 có thể thuộc một hay hai loại: SRAM hoặc EEPROM. Bộ nhớ dữ liệu SRAM được tích hợp bên trong mọi chip thuộc họ vi điều khiển này, có dung lượng khác nhau tùy loại chip, nhưng thường chỉ khoảng vài trăm byte. Đây chính là nơi chứa các biến trung gian trong quá trình hoạt động của chip. khi mất điện, do bản chất của SRAM mà giá trị của các biến này cũng bị mất theo. Khi có điện trở lại, nội dung của các ô nhớ chứa các biến này cũng là bất kỳ, không thể xác định trước. Bên cạnh bộ nhớ loại SRAM, một số chip thuộc họ

8051 còn có thêm bộ nhớ dữ liệu loại EEPROM với dung lượng tối đa vài Kbyte, tùy từng loại chip cụ thể. Dưới đây là một vài ví dụ về bộ nhớ chương trình của một số loại chip thông dụng thuộc họ 8051.

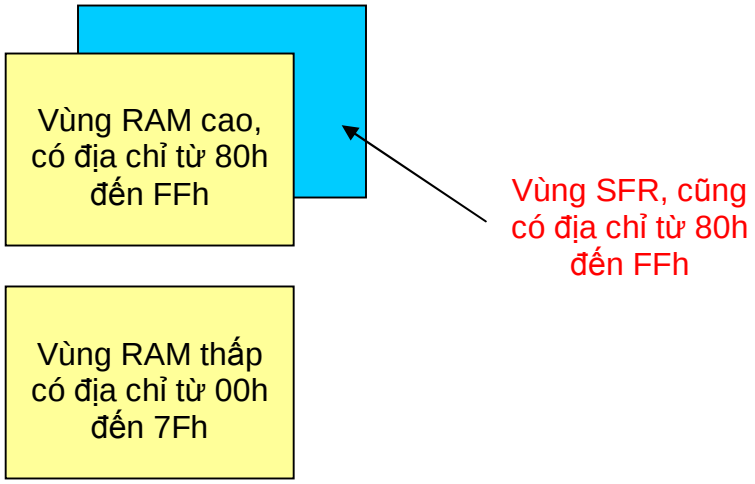
STT	Tên chip	Bộ nhớ SRAM	Bộ nhớ EEPROM
1	AT89C51	128 byte	0
2	AT89C52	256 byte	0
3	AT89C2051	128 byte	0
4	AT89S51	128 byte	0
5	AT89S52	256 byte	0
6	AT89S8252	256 byte	2048 byte

Tổng quát về bộ nhớ của 8051, ta có thể thấy mỗi chip 8051 gồm có những bộ nhớ sau:



Đối với các chip có bộ nhớ SRAM 128 byte thì địa chỉ của các byte SRAM này được đánh số từ 00h đến 7Fh. Đối với các chip có bộ nhớ SRAM 256 byte thì địa chỉ của các byte SRAM được đánh số từ 00h đến FFh. Ở cả hai loại chip, SRAM có địa chỉ từ 00h đến 7Fh được gọi là vùng RAM thấp, phần có địa chỉ từ 80h đến FFh (nếu có) được gọi là vùng RAM cao.

Bên cạnh các bộ nhớ, bên trong mỗi chip 8051 còn có một tập hợp các thanh ghi chức năng đặc biệt (SFR – Special Function Register). Các thanh ghi này liên quan đến hoạt động của các ngoại vi onchip (các cổng vào ra, timer, ngắt ...). Địa chỉ của chúng trùng với dải địa chỉ của vùng SRAM cao, tức là cũng có địa chỉ từ 80h đến FFh.

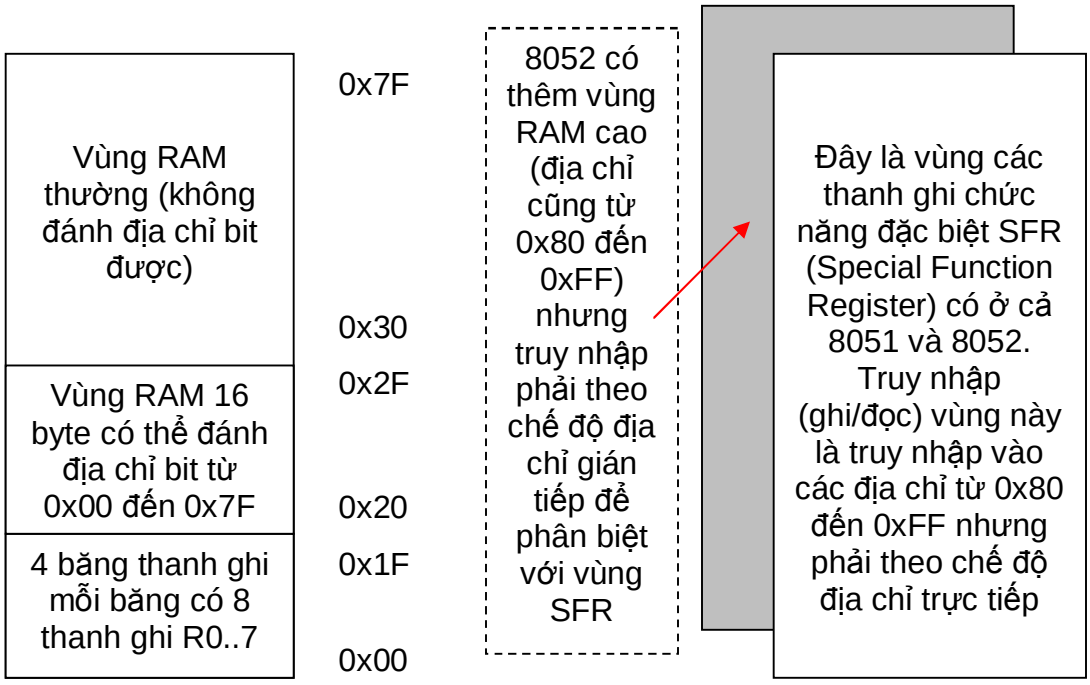


Vậy khi truy cập vào một địa chỉ thuộc dải từ 00h đến 7Fh thì sẽ truy cập đến ô nhớ thuộc vùng RAM thấp. Tuy nhiên khi truy cập đến một địa chỉ x thuộc dải từ 80h đến FFh thì xảy ra vấn đề cần giải quyết: sẽ truy cập đến thanh ghi SFR ở địa chỉ x hay truy cập đến ô nhớ ở địa chỉ x của vùng RAM cao? Nhà sản xuất quy định rằng, trong trường hợp này, nếu kiểu truy cập sử dụng chế độ địa chỉ trực tiếp thì sẽ truy cập vào vùng SFR, ngược lại nếu kiểu truy cập sử dụng chế độ địa chỉ gián tiếp thì sẽ truy cập vào vùng RAM cao.

Bộ nhớ dữ liệu RAM onchip thường dùng để chứa các biến tạm thời trong quá trình vi điều khiển hoạt động, đó cũng là nơi dành cho ngăn xếp hoạt động.
Không gian dữ liệu 64Kbyte được để trống hoàn toàn và chỉ dùng được khi ghép nối với bộ nhớ dữ liệu bên ngoài. Khi ghép nối thêm bộ nhớ dữ liệu bên ngoài, dung lượng của các bộ nhớ này sẽ chiếm dần các vị trí trong không gian, tuy nhiên không hề ảnh hưởng đến 128byte RAM onchip.

Ngăn xếp trong 8051 liên quan đến một thanh ghi tên là con trỏ ngăn xếp SP (Stack Pointer). Thanh ghi này luôn trỏ vào đỉnh của ngăn xếp, tức là nó chứa địa chỉ của vị trí ngay sát vị trí có thể lưu địa chỉ/dữ liệu tiếp theo vào. Khi cất 1 byte địa chỉ/dữ liệu vào ngăn xếp, SP tự động tăng lên 1 đơn vị sau đó mới cất địa chỉ/dữ liệu vào ô nhớ có địa chỉ bằng với giá trị của SP sau khi đã tăng. Khi lấy 1 byte địa chỉ/dữ liệu ra khỏi ngăn xếp, giá trị sẽ được lấy ra sau đó SP mới tự động trừ đi 1 đơn vị. Giá trị sau khi reset của SP là 0x07, do đó quy định ngăn xếp sẽ cất dữ liệu từ địa chỉ 0x08 trở đi. Tuy nhiên do đặc tính hoạt động bành trướng theo chiều tăng địa chỉ mà ngăn xếp thường được bố trí lên vùng trên cùng của bộ nhớ RAM onchip để tránh tranh chấp với các biến lưu trong RAM.

Hình ảnh minh họa bộ nhớ dữ liệu



Bản đồ các thanh ghi chức năng đặc biệt SFR

Table 5-1. AT89S52 SFR Map and Reset Values

0F8H									0FFH
0F0H	B 00000000								0F7H
0E8H									0EFH
0E0H	ACC 00000000								0E7H
0D8H									0DFH
0D0H	PSW 00000000								0D7H
0C8H	T2CON 00000000	T2MOD XXXXXX00	RCAP2L 00000000	RCAP2H 00000000	TL2 00000000	TH2 00000000			0CFH
0C0H									0C7H
0B8H	IP XX000000								0BFH
0B0H	P3 11111111								0B7H
0A8H	IE 0X000000								0AFH
0A0H	P2 11111111		AUXR1 XXXXXXX0				WDTRST XXXXXXXX		0A7H
98H	SCON 00000000	SBUF XXXXXXXX							9FH
90H	P1 11111111								97H
88H	TCON 00000000	TMOD 00000000	TL0 00000000	TL1 00000000	TH0 00000000	TH1 00000000	AUXR XXX00XX0		8FH
80H	P0 11111111	SP 00000111	DP0L 00000000	DP0H 00000000	DP1L 00000000	DP1H 00000000		PCON 0XXX0000	87H

Cổng vào ra song song (I/O Port)

8051 có 4 cổng vào ra song song, có tên lần lượt là P0, P1, P2 và P3. Tất cả các cổng này đều là cổng vào ra hai chiều 8bit. Các bit của mỗi cổng là một chân trên chip, như vậy mỗi cổng sẽ có 8 chân trên chip.

Hướng dữ liệu (dùng cổng đó làm cổng ra hay cổng vào) là độc lập giữa các cổng và giữa các chân (các bit) trong cùng một cổng. Ví dụ, ta có thể định nghĩa cổng P0 là cổng ra, P1 là cổng vào hoặc ngược lại một cách tùy ý, với cả 2 cổng P2 và P3 còn lại cũng vậy. Trong cùng một cổng P0, ta cũng có thể định nghĩa chân P0.0 là cổng vào, P0.1 lại là cổng ra tùy ý.

Liên quan đến mỗi cổng vào/ra song song của 8051 chỉ có một thanh ghi SFR (thanh ghi chức năng đặc biệt) có tên trùng với tên của cổng. Ta có các thanh ghi P0 dùng cho cổng P0, thanh ghi P1 dùng cho cổng P1 … Đây là các thanh ghi đánh địa chỉ đến từng bit (bit addressable), do đó ta có thể dùng các lệnh tác động bit đối với các bit của các thanh ghi này. Mỗi thanh ghi này gồm 8 bit tương ứng với các chân (bit) của cổng đó. Khi một chân (bit) cổng nào đó được dùng làm cổng vào thì trước đó bit tương ứng trong thanh ghi SFR phải được đặt ở mức 1. Nếu một chân (bit) cổng nào đó được dùng làm cổng ra thì giá trị của bit tương ứng trong thanh ghi SFR sẽ là giá trị lôgic muốn đưa ra chân cổng đó. Nếu muốn đưa ra mức lôgic cao (điện áp gần 5V), bit tương ứng trong thanh ghi phải được đặt bằng 1, hiển nhiên nếu muốn đưa ra mức lôgic thấp (điện áp gần 0V) thì bit tương ứng trong thanh ghi phải được đặt bằng 0. Như đã nói ở trên, các bit trong thanh ghi cổng có thể được đặt bằng 1/0 mà không làm ảnh hưởng đến các bit còn lại trong cổng đó bằng cách dùng các lệnh **setb** (đặt lên 1) hay **clr** (đặt về 0).

Sau khi đặt một chân cổng làm cổng vào, ta có thể dùng các lệnh kiểm tra bit để đọc vào và kiểm tra các mức lôgic của mạch ngoài đang áp vào là mức 0 hay mức 1. Các lệnh này là **jb** (nhảy nếu bit bằng 1), **jnb** (nhảy nếu bit bằng 0).

Mỗi cổng có cấu trúc gồm một latch (chính là các bit của thanh ghi cổng), mạch lái đầu ra (output driver) và mạch đệm đầu vào (input buffer).

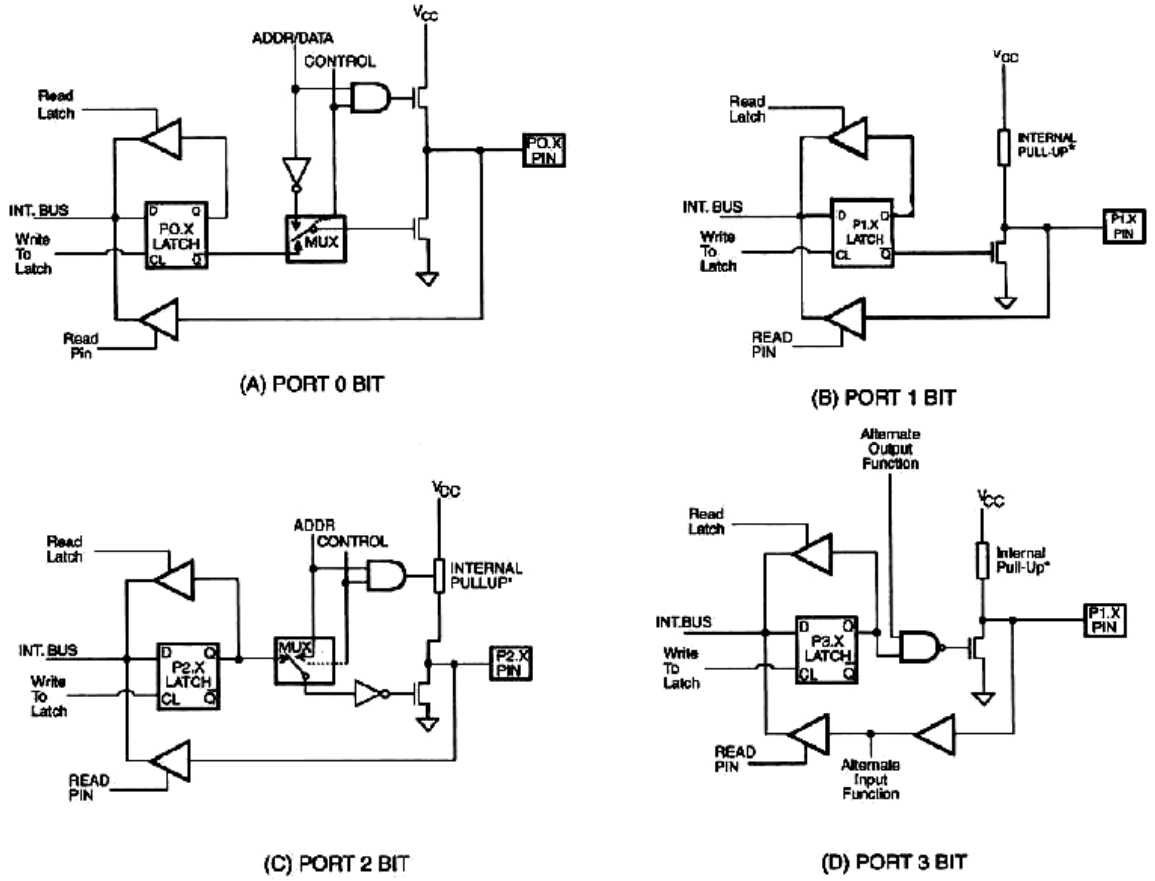
Ngoài chức năng vào/ra thông thường, một số cổng còn được tích hợp thêm chức năng của một số ngoại vi khác. Xem bảng liệt kê sau:

Port Pin	Alternate Function
(1)P1.0	T2 (Timer/Counter 2 external input) (If Timer 2 available)
(1)P1.1	T2EX (Timer/Counter 2 capture/reload trigger) (If Timer 2 available)
P3.0	RXD (serial input port)
P3.1	TXD (serial output port)
P3.2	$\overline{\text{INT0}}$ (external interrupt)
P3.3	$\overline{\text{INT1}}$ (external interrupt)
P3.4	T0 (Timer/Counter 0 external input)
P3.5	T1 (Timer/Counter 1 external input)
P3.6	$\overline{\text{WR}}$ (external Data memory write strobe)
P3.7	$\overline{\text{RD}}$ (external Data memory read strobe)

Các chân cổng P1.0 và P1.1 được tích hợp với các tín hiệu của timer2 trong trường hợp chip là 8052.

Khi dùng với các chức năng của các ngoại vi, chân cổng tương ứng phải được đặt lên 1. Nếu không các tín hiệu sẽ luôn bị ghim ở mức 0.

Sơ đồ của mạch của một chân cổng:



Cổng P0 không có điện trở treo cao (pullup resistor) bên trong, mạch lái tạo mức cao chỉ có khi sử dụng cổng này với tính năng là bus dồn kênh địa chỉ/dữ liệu. Như vậy với chức năng ra thông thường, P0 là cổng ra open drain, với chức năng vào, P0 là cổng vào cao trở (high impedance). Nếu muốn sử dụng cổng P0 làm cổng vào/ra thông thường, ta phải thêm điện trở pullup bên ngoài. Giá trị điện trở pullup bên ngoài thường từ 4K7 đến 10K.

Các cổng P1, P2 và P3 đều có điện trở pullup bên trong, do đó có thể dùng với chức năng cổng vào/ra thông thường mà không cần có thêm điện trở pullup bên ngoài. Thực chất, điện trở pullup bên trong là các FET, không phải điện trở tuyến tính thông thường, tuy vậy nhưng khả năng phun dòng ra của mạch lái khi đầu ra ở mức cao (hoặc khi là đầu vào) rất nhỏ, chỉ khoảng 100 micro Ampe. Trong datasheet của AT89S5x (một trong những biến thể của họ 8051 do Atmel sản xuất) có thống kê số liệu như sau:

Symbol	Parameter	Condition	Min	Max	Units
V_{OH}	Output High Voltage (Ports 1,2,3, ALE, $\overline{PSEN}$)	I_{OH} = -60 µA, V_{CC} = 5V ± 10%	2.4		V
		I_{OH} = -25 µA	0.75 V_{CC}		V
		I_{OH} = -10 µA	0.9 V_{CC}		V

Theo đó, nếu ta thiết kế để các cổng phải cung cấp cho tải ở đầu ra mức cao một lượng dòng điện I_{OH} = 60 micro Ampe thì mức điện áp ở đầu ra V_{OH} sẽ bị kéo sụt xuống, chỉ có thể đảm bảo từ 2.4V trở lên bởi nhà sản xuất, không thể cao sát với 5V như lý thuyết.

Trong khi đó, khả năng nuốt dòng của mạch lái khi đầu ra ở mức thấp lại cao hơn rất nhiều, có thể đạt từ vài đến hàng chục mili Ampe.

Symbol	Parameter	Condition	Min	Max	Units
V_{OL}	Output Low Voltage[1] (Ports 1,2,3)	I_{OL} = 1.6 mA		0.45	V
V_{OL1}	Output Low Voltage[1] (Port 0, ALE, $\overline{PSEN}$)	I_{OL} = 3.2 mA		0.45	V

Như vậy, khi thiết kế với các phần tử bên ngoài, ta nên để ý đến đặc tính vào/ra của các chân cổng. Ví dụ khi dùng để ghép nối với LED đơn hoặc LED 7 thanh, ta nên thiết kế chân cổng nuốt dòng từ LED để làm LED sáng (cổng nối với Cathode của LED), không nên thiết kế chân cổng phun dòng cho LED để làm LED sáng (cổng nối với Anode của LED).

Cổng vào ra nối tiếp (Serial Port)

Cổng nối tiếp trong 8051 chủ yếu được dùng trong các ứng dụng có yêu cầu truyền thông với máy tính, hoặc với một vi điều khiển khác. Liên quan đến cổng nối tiếp chủ yếu có 2 thanh ghi: SCON và SBUF. Ngoài ra, một thanh ghi khác là thanh ghi PCON (không đánh địa chỉ bit) có bit 7 tên là SMOD quy định tốc độ truyền của cổng nối tiếp có gấp đôi lên (SMOD = 1) hay không (SMOD = 0).

Dữ liệu được truyền nhận nối tiếp thông qua hai chân cổng P3.0(RxD) và P3.1(TxD).

Thanh ghi SBUF là thanh ghi 8bit chứa dữ liệu truyền hoặc nhận. Về thực chất có hai thanh ghi dữ liệu khác nhau, một dành để chứa dữ liệu truyền đi, một để chứa dữ liệ nhận được. Cả hai thanh ghi này đều có chung một tên là SBUF, tuy nhiên CPU hoàn toàn phân biệt được một cách dễ dàng. Khi ta muốn truyền dữ liệu đi, ta phải ghi vào thanh ghi SBUF (ví dụ viết lệnh **mov SBUF,a**), còn khi muốn đọc kiểm tra dữ liệu nhận về ta phải đọc thanh ghi SBUF (ví dụ viết lệnh **mov a,SBUF**). CPU sẽ căn cứ vào việc thanh ghi SBUF nằm ở vị trí toán hạng đích (toán hạng bên trái) hay toán hạng nguồn (toán hạng bên phải) để quyết định sẽ truy nhập (đọc/ghi) thanh ghi SBUF nào. Người lập trình không cần phải quan tâm xử lý vấn đề này.
Thanh ghi quy định chế độ hoạt động và điều khiển cổng nối tiếp là thanh ghi SCON (đánh địa chỉ bit).

SM0	SM1	SM2	REN	TB8	RB8	TI	RI

Bit SM0, SM1, SM2 quy định chế độ hoạt động của cổng nối tiếp. Thông thường để truyền thông giữa 2 vi điều khiển hoặc giữa 1 vi điều khiển và 1 máy tính, giá trị của bit SM2 được đặt bằng 0. Khi truyền thông theo kiểu mạng đa vi xử lý (multiprocessor communication), SM2 được đặt bằng 1. Hai bit SM0 và SM1 thực sự là các bit quy định chế độ hoạt động của cổng nối tiếp, chúng tạo ra 4 tổ hợp (00,01,10 và 11) ứng với 4 chế độ hoạt động mô tả trong bảng sau.

SM0	SM1	Chế độ	Khung dữ liệu	Baud rate
0	0	0 - Đồng bộ	8 bit SBUF	Fosc/12
0	1	1 - Dị bộ	8 bit SBUF	Thay đổi được
1	0	2 - Dị bộ	8bit SBUF + RB8/TB8	Fosc/32 hoặc Fosc/64
1	1	3 - Dị bộ	8bit SBUF + RB8/TB8	Thay đổi được

Chế độ 0: là chế độ truyền đồng bộ duy nhất. Chân RxD sẽ là tín hiệu truyền/nhận dữ liệu, chân TxD là tín hiệu xung nhịp. Bit LSB (bit 0) của dữ liệu được truyền đi trước tiên. Tốc độ truyền cố định và bằng 1/12 giá trị thạch anh.
Chế độ 1: là chế độ truyền dị bộ 8 bit. Dữ liệu 8 bit được đóng khung bởi một bit Start (= 0) ở đầu và một bit Stop (=1) ở cuối trước khi được truyền đi. Tốc độ truyền thay đổi được theo ý người lập trình.
Chế độ 2: là chế độ truyền dị bộ 9 bit. Dữ liệu 9 bit được ghép thành bởi 8bit trong thanh ghi SBUF và bit RB8 (trường hợp nhận về) hoặc TB8 (trường hợp truyền đi) trong thanh ghi SCON. Ngoài ra các bit Start và Stop vẫn được gắn bình ở đầu và cuối khung truyền. Trong chế độ này, tốc độ truyền chỉ có thể chọn được ở 1 trong 2 mức: 1/32 hoặc 1/64 giá trị của thạch anh (tùy thuộc vào giá trị của bit SMOD trong thanh ghi PCON đã nói ở trên).
Chế độ 3: cũng là chế độ truyền dị bộ 9 bit, khác với chế độ 2 ở chỗ tốc độ truyền có thể thay đổi được theo ý người lập trình như trong chế độ 1.

Bit **REN** trong thanh ghi SCON là bit cho phép nhận dữ liệu. Dữ liệu chỉ được nhận qua cổng nối tiếp khi bit này = 1.

Bit **TB8** là bit dữ liệu thứ 9 trong trường hợp truyền đi 9 bit (8 bit kia trong thanh ghi SBUF).

Bit **RB8** là bit dữ liệu thứ 9 trong trường hợp nhận về 9 bit (8 bit kia trong thanh ghi SBUF).

Bit **TI** là cờ ngắt truyền, báo hiệu việc truyền 1 khung dữ liệu đã hoàn tất.

Bit **RI** là cờ ngắt nhận, báo hiệu việc nhận 1 khung dữ liệu đã hoàn tất.

Để tạo ra tốc độ truyền (Baud rate) của cổng nối tiếp trong 8051, phải dùng đến timer1 ở chế độ Auto Reload 8bit. Giá trị nạp lại chứa trong thanh ghi TH1 được tính toán theo công thức sau (phụ thuộc vào Baud rate mong muốn và giá trị của thạch anh).

$$\text{Modes 1, 3 Baud Rate} = \frac{2^{\text{SMOD}}}{32} \times \frac{\text{Oscillator Frequency}}{12x\ [256 - (\text{TH1})]}$$

Tóm lại để sử dụng cổng nối tiếp của 8051, hãy thực hiện các bước sau:
- Chọn chế độ cho cổng nối tiếp (đồng bộ/dị bộ, 8bit/9bit...), từ đó chọn được giá trị cho các bit trong thanh ghi SCON. Lưu ý xóa các bit TI và RI.

- Chọn tốc độ truyền mong muốn, từ đó tính ra giá trị của thanh ghi TH1. Cho timer1 chạy ở chế độ Auto Reload 8bit (không dùng ngắt tràn timer1).
- Đặt mức ưu tiên ngắt và cho phép ngắt cổng nối tiếp nếu muốn.
- Bắt đầu quá trình truyền dữ liệu bằng một lệnh ghi dữ liệu muốn truyền vào thanh ghi SBUF. Quá trình truyền kết thúc thì cờ TI sẽ tự động đặt lên 1.
- Khi một khung dữ liệu đã được nhận đầy đủ, cờ RI sẽ tự động đặt lên 1 và người lập trình lúc này có thể dùng lệnh đọc thanh ghi SBUF để lấy dữ liệu nhận được ra xử lý.

Ngắt (Interrupt)

8051 chỉ có một số lượng khá ít các nguồn ngắt (interrupt source) hoặc có thể gọi là các nguyên nhân ngắt. Mỗi ngắt có một vector ngắt riêng, đó là một địa chỉ cố định nằm trong bộ nhớ chương trình, khi ngắt xảy ra, *CPU sẽ tự động nhảy đến thực hiện lệnh nằm tại địa chỉ này*. Bảng tóm tắt các ngắt trong 8051 như sau:

STT	Tên ngắt	Mô tả	Cờ ngắt	Thanh ghi chứa cờ	Vector ngắt
1	INT0	Ngắt ngoài 0 khi có tín hiệu tích cực theo kiểu đã chọn ở chân P3.2	IE0	TCON	0x0003
2	Timer0	Ngắt tràn timer0 khi giá trị timer0 tràn từ giá trị max về giá trị min	TF0	TCON	0x000B
3	INT1	Ngắt ngoài 1 khi có tín hiệu tích cực theo kiểu đã chọn ở chân P3.3	IE1	TCON	0x0013
4	Timer1	Ngắt tràn timer1 khi giá trị timer1 tràn từ giá trị max về giá trị min	TF1	TCON	0x001B
5	Serial Port	Ngắt cổng nối tiếp khi vi điều khiển nhận hoặc truyền xong một byte bằng cổng nối tiếp	TI, RI	SCON	0x0023

Với 8052, ngoài các ngắt trên còn có thêm ngắt của timer2 (do vi điều khiển này có thêm timer2 trong số các ngoại vi onchip).

Mỗi ngắt được dành cho một vector ngắt kéo dài 8byte. Về mặt lý thuyết, nếu chương trình đủ ngắn, mã tạo ra chứa đủ trong 8 byte, người lập trình hoàn toàn có thể đặt phần chương trình xử lý ngắt ngay tại vector ngắt. Tuy nhiên trong hầu hết các trường hợp, chương trình xử lý ngắt có dung lượng mã tạo ra lớn hơn 8byte nên tại vector ngắt, ta chỉ đặt lệnh nhảy tới chương trình xử lý ngắt nằm ở vùng nhớ khác. Nếu không làm vậy, mã chương trình xử lý ngắt này sẽ lấn sang, đè vào vector ngắt kế cận.

Liên quan đến ngắt chủ yếu có hai thanh ghi là thanh ghi IE và thanh ghi IP.

Table 13-1. Interrupt Enable (IE) Register

(MSB)							(LSB)
EA	–	ET2	ES	ET1	EX1	ET0	EX0

Enable Bit = 1 enables the interrupt.

Enable Bit = 0 disables the interrupt.

Symbol	Position	Function
EA	IE.7	Disables all interrupts. If EA = 0, no interrupt is acknowledged. If EA = 1, each interrupt source is individually enabled or disabled by setting or clearing its enable bit.
–	IE.6	Reserved.
ET2	IE.5	Timer 2 interrupt enable bit.
ES	IE.4	Serial Port interrupt enable bit.
ET1	IE.3	Timer 1 interrupt enable bit.
EX1	IE.2	External interrupt 1 enable bit.
ET0	IE.1	Timer 0 interrupt enable bit.
EX0	IE.0	External interrupt 0 enable bit.
User software should never write 1s to reserved bits, because they may be used in future AT89 products.		

Để cho phép một ngắt, bit tương ứng với ngắt đó và bit EA phải được đặt bằng 1. Thanh ghi IE là thanh ghi đánh địa chỉ bit, do đó có thể dùng các lệnh tác động bit để tác động riêng rẽ lên từng bit mà không làm ảnh hưởng đến giá trị các bit khác. Cờ ngắt hoạt động độc lập với việc cho phép ngắt, điều đó có nghĩa là cờ ngắt sẽ tự động đặt lên bằng 1 khi có sự kiện gây ngắt xảy ra, bất kể sự kiện đó có được cho phép ngắt hay không. Do vậy, trước khi cho phép một ngắt, ta nên xóa cờ của ngắt đó để đảm bảo sau khi cho phép, các sự kiện gây ngắt trong quá khứ không thể gây ngắt nữa. Ví dụ trước khi cho phép ngắt timer0 mà timer 0 đã chạy và tràn (dù là tràn một hay nhiều lần) thì cờ TF0 sẽ bằng 1, nếu sau đó ta cho phép ngắt timer0 thì sẽ gây ra ngắt ngay do cờ tràn đang bằng 1 (sự kiện tràn gây ngắt trong trường hợp này là tràn trong quá khứ, không phải sự kiện ta quan tâm đến). Vì vậy hãy xóa cờ TF0 trước khi cho phép ngắt tràn timer0.

Ngoại trừ cờ của của ngắt nối tiếp (và cờ của ngắt timer2 trong 8052), các cờ ngắt khác đều tự động được xóa khi CPU thực hiện chương trình phục vụ ngắt. Lý do là ngắt cổng nối tiếp (và ngắt timer2 trong 8052) được gây ra bởi 2 nguyên nhân (có 2 cờ cho mỗi ngắt), khi xảy ra ngắt, người lập trình cần phải kiểm tra xem cờ nào được đặt bằng 1 để phân biệt nguyên nhân gây ra ngắt đó là nguyên nhân nào để xử lý thích hợp. Ví dụ ngắt cổng nối tiếp là ngắt được gây ra bởi 1 trong 2 nguyên nhân: vi điều khiển nhận xong hoặc truyền xong một byte dữ liệu qua cổng nối tiếp. Xảy ra sự kiện nào thì cờ ngắt tương ứng sẽ tự động được đặt lên bằng 1, nếu nhận xong thì cờ RI bằng 1, nếu truyền xong thì cờ TI bằng 1. Trong chương trình xử lý ngắt, người lập trình phải kiểm tra cờ TI hay cờ RI bằng 1 để quyết định xử lý ngắt truyền hay xử lý ngắt nhận. Sau khi kiểm tra, người lập trình phải viết lệnh xóa cờ đó vì việc này không được CPU thực hiện tự động như các cờ ngắt khác.

Nói đến ngắt không thể không nói đến mức ưu tiên của ngắt. Mức ưu tiên của ngắt ở đây có thể được hiểu là sự phân bậc, quyết định xử lý ngắt nào khi hai hay nhiều ngắt xảy ra. Có 2 cơ chế phân bậc ưu tiên. Thứ nhất là cơ chế phân bậc dành cho các ngắt xảy ra đồng thời, hai ngắt A và B xảy ra cùng một thời điểm nhìn từ phía vi điều khiển. Thứ hai là cơ chế phân bậc dành cho các ngắt xảy ra xen kẽ nhau, trong khi đang xử lý ngắt A thì ngắt B xảy ra, vậy thì trong từng trường hợp, CPU sẽ xử lý ra sao? Hãy xem dưới đây.

Với trường hợp các ngắt xảy ra đồng thời, CPU sẽ xem xét mức ưu tiên của các ngắt đó, từ đó quyết định xử lý ngắt có mức ưu tiên cao hơn trước. Mức ưu tiên trong trường hợp này là mức ưu tiên cứng (được quy định bởi nhà sản xuất, bởi cấu trúc sẵn có của 8051 và người lập trình **không thể thay đổi được**).

Table 2-27. Interrupt Priority Level

	Source	Priority Within Level
1	IE0	(highest)
2	TF0	
3	IE1	
4	TF1	
5	RI + TI	
6	TF2 + EXF2	(lowest)

Nhìn vào bảng trên ta thấy ngắt INT0 là ngắt có mức ưu tiên cao nhất và ngắt timer2 là ngắt có mức ưu tiên thấp nhất trong số các ngắt. Như vậy nếu ngắt ngoài 1 và ngắt timer0 cùng xảy ra một lúc, ngắt timer0 sẽ được CPU xử lý trước, sau đó mới xử lý ngắt ngoài 1.

Với trường hợp xảy ra ngắt xen kẽ, khi CPU đang xử lý ngắt A mà ngắt B xảy ra, CPU sẽ giải quyết theo 2 hướng: tiếp tục xử lý ngắt A nếu mức ưu tiên của ngắt B **không cao hơn** mức ưu tiên của ngắt A, hoặc sẽ dừng việc xử lý ngắt A lại, chuyển sang xử lý ngắt B nếu mức ưu tiên của ngắt B **cao hơn** mức ưu tiên của ngắt A. Mức ưu tiên cho các ngắt trong trường hợp này không phải là mức ưu tiên cứng do nhà sản xuất quy định (tức là không căn cứ vào bảng trên) mà là do người lập trình đặt. Lập trình viên có thể dùng thanh ghi IP để quy định mức ưu tiên cho các ngắt ở một trong hai mức: mức cao và mức thấp. Để đặt mức ưu tiên của một ngắt (trong trường hợp xảy ra xen kẽ) ở mức cao, ta đặt bit tương ứng với ngắt đó trong thanh ghi IP bằng 1, mức thấp ứng với giá trị bit = 0.

Thanh ghi IP (Interrupt Priority)

-	-	PT2	PS	PT1	PX1	PT0	PX0

Các bit trong thanh ghi IP tương ứng với các ngắt đúng như trong thanh ghi IE (bit PX0 dành cho ngắt ngoài 0, bit PT0 dành cho ngắt timer 0…)

Một điều dễ nhận ra là nếu một ngắt được đặt mức ưu tiên cao (bit tương ứng trong thanh ghi IP bằng 1) thì sẽ chẳng có ngắt nào có thể xen vào quá trình xử lý nó được nữa.

Nói về mức ưu tiên ngắt, có thể dùng một ví dụ tổng quát sau, giả sử hai ngắt timer0 và ngắt cổng nối tiếp cùng được cho phép (các bit tương ứng và bit EA trong thanh ghi IE được đặt bằng 1), bit PT0 = 0, bit PS = 1 thì:

- Nếu hai ngắt cùng xảy ra, ngắt timer0 sẽ thắng thế và được phục vụ trước.
- Nếu ngắt cổng nối tiếp xảy ra trước và đang được xử lý thì ngắt timer0 nếu có xảy ra cũng không thể chen vào, làm dừng quá trình xử lý ngắt cổng nối tiếp được.
- Nếu ngắt timer0 xảy ra trước và đang được xử lý mà ngắt cổng nối tiếp xảy ra thì CPU sẽ phải dừng việc xử lý ngắt timer0 lại, chuyển sang xử lý ngắt cổng nối tiếp, xử lý xong mới quay lại xử lý tiếp ngắt timer0.

Như đã nói ở trên, 8051 có 2 ngắt ngoài là INT0 và INT1. Ngắt ngoài được hiểu là ngắt được gây ra bởi sự kiện mức lôgic 0 (mức điện áp thấp, gần 0V) hoặc sườn xuống (sự chuyển mức điện áp từ mức cao về mức thấp) xảy ra ở chân ngắt tương ứng (P3.2 với ngắt ngoài 0 và P3.3 với ngắt ngoài 1). Việc lựa chọn kiểu ngắt được thực hiện bằng các bit IT (Interrupt Type) nằm trong thanh ghi TCON. Đây là thanh ghi điều khiển timer nhưng 4 bit LSB (bit0..3) được dùng cho các ngắt ngoài.

7	6	5	4	3	2	1	0
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

Khi bit ITx = 1 thì ngắt ngoài tương ứng được chọn kiểu là ngắt theo sườn xuống, ngược lại nếu bit ITx = 0 thì ngắt ngoài tương ứng được sẽ có kiểu ngắt là ngắt theo mức thấp. Các bit IE là các bit cờ ngắt ngoài, chỉ có tác dụng trong trường hợp kiểu ngắt được chọn là ngắt theo sườn xuống.

Khi kiểu ngắt theo sườn xuống được chọn thì ngắt sẽ xảy ra duy nhất một lần khi có sườn xuống của tín hiệu, sau đó khi tín hiệu ở mức thấp, hoặc có sườn lên, hoặc ở mức cao thì cũng không có ngắt xảy ra nữa cho đến khi có sườn xuống tiếp theo. Cờ ngắt IE sẽ dựng lên khi có sườn xuống và tự động bị xóa khi CPU bắt đầu xử lý ngắt.

Khi kiểu ngắt theo mức thấp được chọn thì ngắt sẽ xảy ra bất cứ khi nào tín hiệu tại chân ngắt ở mức thấp. Nếu sau khi xử lý xong ngắt mà tín hiệu vẫn ở mức thấp thì lại ngắt tiếp, cứ như vậy cho đến khi xử lý xong ngắt lần thứ n , tín hiệu đã lên mức cao rồi thì thôi không ngắt nữa. Cờ ngắt IE trong trường hợp này không có ý nghĩa gì cả.

Thông thường kiểu ngắt hay được chọn là ngắt theo sườn xuống.

Bộ định thời/Bộ đếm (Timer/Counter)

8051 có 2 timer tên là timer0 và timer1. Các timer này đều là timer 16bit, giá trị đếm max do đó bằng 2^{16} = 65536 (đếm từ 0 đến 65535).

Hai timer có nguyên lý hoạt động hoàn toàn giống nhau và độc lập. Sau khi cho phép chạy, mỗi khi có thêm một xung tại đầu vào đếm, giá trị của timer sẽ tự động được tăng lên 1 đơn vị, cứ như vậy cho đến khi giá trị tăng lên vượt quá giá trị max mà thanh ghi đếm có thể biểu diễn thì giá trị đếm lại được đưa trở về giá trị min (thông thường min = 0). Sự kiện này được hiểu là sự kiện tràn timer (overflow) và có thể gây ra ngắt nếu ngắt tràn timer được cho phép (bit ETx trong thanh ghi IE = 1).

Việc cho timer chạy/dừng được thực hiện bởi các bit TR trong thanh ghi TCON (đánh địa chỉ đến từng bit).

7	6	5	4	3	2	1	0
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

Khi bit TRx = 1, timerx sẽ đếm, ngược lại khi TRx = 0, timerx sẽ không đếm mặc dù vẫn có xung đưa vào. Khi dừng không đếm, giá trị của timer được giữ nguyên.

Các bit TFx là các cờ báo tràn timer, khi sự kiện tràn timer xảy ra, cờ sẽ được tự động đặt lên bằng 1 và nếu ngắt tràn timer được cho phép, ngắt sẽ xảy ra. Khi CPU xử lý ngắt tràn timerx, cờ ngắt TFx tương ứng sẽ tự động được xóa về 0.

Giá trị đếm 16bit của timerx được lưu trong hai thanh ghi THx (byte cao) và TLx (byte thấp). Hai thanh ghi này có thể ghi/đọc được bất kỳ lúc nào. Tuy nhiên nhà sản xuất khuyến cáo rằng nên dừng timer (cho bit TRx = 0) trước khi ghi/đọc các thanh ghi chứa giá trị đếm.

Các timer có thể hoạt động theo nhiều chế độ, được quy định bởi các bit trong thanh ghi TMOD (không đánh địa chỉ đến từng bit).

7	6	5	4	3	2	1	0
GATE1	C/T1#	M11	M01	GATE0	C/T0#	M10	M00

Bit Number	Bit Mnemonic	Description
7	GATE1	**Timer 1 Gating Control Bit** Clear to enable timer 1 whenever the TR1 bit is set. Set to enable timer 1 only while the INT1# pin is high and TR1 bit is set.
6	C/T1#	**Timer 1 Counter/Timer Select Bit** Clear for timer operation: timer 1 counts the divided-down system clock. Set for Counter operation: timer 1 counts negative transitions on external pin T1.
5	M11	**Timer 1 Mode Select Bits** M11 M01 Operating mode 0 0 Mode 0: 8-bit timer/counter (TH1) with 5-bit prescaler (TL1). 0 1 Mode 1: 16-bit timer/counter. 1 0 Mode 2: 8-bit auto-reload timer/counter (TL1). Reloaded from TH1 at overflow. 1 1 Mode 3: timer 1 halted. Retains count.
4	M01	
3	GATE0	**Timer 0 Gating Control Bit** Clear to enable timer 0 whenever the TR0 bit is set. Set to enable timer/counter 0 only while the INT0# pin is high and the TR0 bit is set.
2	C/T0#	**Timer 0 Counter/Timer Select Bit** Clear for timer operation: timer 0 counts the divided-down system clock. Set for counter operation: timer 0 counts negative transitions on external pin T0.
1	M10	**Timer 0 Mode Select Bit** M10 M00 Operating mode 0 0 Mode 0: 8-bit timer/counter (TH0) with 5-bit prescaler (TL0). 0 1 Mode 1: 16-bit timer/counter. 1 0 Mode 2: 8-bit auto-reload timer/counter (TL0). Reloaded from TH0 at overflow. 1 1 Mode 3: TL0 is an 8-bit timer/counter. TH0 is an 8-bit timer using timer 1's TR0 and TF0 bits.
0	M00	

Để xác định thời gian, người ta chọn nguồn xung nhịp (clock) đưa vào đếm trong timer là xung nhịp bên trong (dành cho CPU). Nguồn xung nhịp này thường rất đều đặn (có tần số ổn định), do đó từ số đếm của timer người ta có thể nhân với chu kỳ xung nhịp để tính ra thời gian trôi qua. Timer lúc này được gọi chính xác với cái tên "timer", tức bộ định thời.

Để đếm các sự kiện bên ngoài, người ta chọn nguồn xung nhịp đưa vào đếm trong timer là tín hiệu từ bên ngoài (đã được chuẩn hóa về dạng xung vuông 0V/5V). Các tín hiệu này sẽ được nối với các bit cổng có dồn kênh thêm các tính năng T0/T1/T2. Khi có sự kiện bên ngoài gây ra thay đổi mức xung ở đầu vào đếm, timer sẽ tự động tăng lên 1 đơn vị giống như trường hợp đếm xung nhịp bên trong. Lúc này, timer được gọi chính xác với cái tên khác: "counter", tức bộ đếm (sự kiện).

Nhìn vào bảng mô tả thanh ghi TMOD bên trên, ta có thể nhận thấy có 2 bộ 4 bit giống nhau (gồm GATEx, C/Tx, Mx0 và Mx1) dành cho 2 timer0 và 1. Ý nghĩa các bit là như nhau đối với mỗi timer.

Bit **GATEx** quy định việc cho phép timer đếm (run timer). Nếu GATEx = 0, timerx sẽ đếm khi bit TRx bằng 1, dừng khi bit TRx bằng 0. Nếu GATEx = 1, timerx sẽ chỉ đếm khi bit TRx = 1 và tín hiệu tại chân INTx = 1, dừng khi một trong hai điều kiện trên không còn thỏa mãn. Thông thường người ta dùng timer với GATE = 0, chỉ dùng timer với GATE = 1 trong trường hợp muốn đo độ rộng xung vì lúc đó timer sẽ chỉ đếm thời gian khi xung đưa vào chân INTx ở mức cao.

Bit **C/Tx** quy định nguồn clock đưa vào đếm trong timer. Nếu C/Tx = 0, timer sẽ được cấu hình là bộ định thời, nếu C/Tx = 1, timer sẽ được cấu hình là bộ đếm sự kiện.

Hai bit còn lại (Mx0 và Mx1) tạo ra 4 tổ hợp các giá trị (00,01,10 và 11) ứng với 4 chế độ hoạt động khác nhau của timerx. Trong 4 chế độ đó thường chỉ dùng chế độ timer/counter 16bit (Mx1 = 0, Mx0 = 1) và chế độ Auto Reload 8bit timer/counter (Mx1 = 1, Mx0 = 0).

Trong chế độ timer/counter 16bit, giá trị đếm (chứa trong hai thanh ghi THx và TLx) tự động được tăng lên 1 đơn vị mỗi lần nhận được thêm một xung nhịp. Khi giá trị đếm tăng vượt quá giá trị max = 65535 thì sẽ tràn về 0, cờ ngắt TFx được tự động đặt = 1. Chế độ này được dùng trong các ứng dụng đếm thời gian và đếm sự kiện.

Trong chế độ Auto Reload 8bit, giá trị đếm sẽ chỉ được chứa trong thanh ghi TLx, còn giá trị của thanh ghi THx bằng một số n (từ 0 đến 255) do người lập trình đưa vào. Khi có thêm 1 xung nhịp, giá trị đếm trong TLx đương nhiên cũng tăng lên 1 đơn vị như bình thường. Tuy nhiên trong trường hợp này, giá trị đếm lớn nhất là 255 chứ không phải 65535 như trường hợp trên vì timer/counter chỉ còn 8bit. Do vậy sự kiện tràn lúc này xảy ra nhanh hơn, chỉ cần vượt quá 255 là giá trị đếm sẽ tràn. Cờ ngắt TFx vẫn được tự động đặt = 1 như trong trường hợp tràn 16bit. Điểm khác biệt là thay vì tràn về 0, giá trị THx sẽ được tự động nạp lại (Auto Reload) vào thanh ghi TLx, do đó timer/counter sau khi tràn sẽ có giá trị bằng n (giá trị chứa trong THx) và sẽ đếm từ giá trị n trở đi. Chế độ này được dùng trong việc tạo Baud rate cho truyền thông qua cổng nối tiếp.

Để sử dụng timer của 8051, hãy thực hiện các bước sau:

- Quy định chế độ hoạt động cho timer bằng cách tính toán và ghi giá trị cho các bit trong thanh ghi TMOD.
- Ghi giá trị đếm khởi đầu mong muốn vào 2 thanh ghi đếm THx và TLx. Đôi khi ta không muốn timer/counter bắt đầu đếm từ 0 mà từ một giá trị nào đó để thời điểm tràn gần hơn, hoặc chẵn hơn trong tính toán sau này. Ví dụ nếu cho timer đếm từ 15535 thì sau 50000 xung nhịp (tức 50000 micro giây với thạch anh 12MHz) timer sẽ tràn, và thời gian một giây có thể dễ dàng tính ra khá chính xác = 20 lần tràn của timer (đương nhiên mỗi lần tràn lại phải nạp lại giá trị 15535).
- Đặt mức ưu tiên ngắt và cho phép ngắt tràn timer (nếu muốn).
- Dùng bit TRx trong thanh ghi TCON để cho timer chạy hay dừng theo ý muốn.

Chương II: Các ngôn ngữ lập trình cho vi điều khiển

Trong kỹ thuật vi xử lý nói chung, ngôn ngữ lập trình thường được chia làm 2 loại: *Ngôn ngữ bậc thấp* và *Ngôn ngữ bậc cao*.

Ngôn ngữ bậc thấp là ngôn ngữ máy hoặc ngôn ngữ gần với máy. Ngôn ngữ máy là ngôn ngữ ở bậc thấp nhất, chính là mã máy ở dạng nhị phân. Lập trình với ngôn ngữ này đồng nghĩa với việc lập trình viên phải viết từng bit 0/1 cho từng mã lệnh cụ thể, đương nhiên đó là việc rất vất vả và khó khăn. Kế đến là ngôn ngữ gần với máy, chính là hợp ngữ (Assembly). Với ngôn ngữ này, lập trình viên có thể viết các lệnh cụ thể ở dạng ký tự, tuân theo một tập hợp các ký tự nhất định gọi là tập lệnh. Nói cách khác, ở cấp độ này, lập trình viên sẽ viết các lệnh ở dạng mã gợi nhớ (mnemonic) thay vì phải viết các bit 0/1 cho các mã lệnh cụ thể. Trình hợp ngữ (Assembler) - một phần mềm trên máy tính - sẽ đảm nhiệm việc dịch các lệnh do lập trình viên viết ở dạng mã gợi nhớ sang dạng mã máy 0/1.

Ngôn ngữ bậc cao là các ngôn ngữ gần với ngôn ngữ con người hơn, do đó việc lập trình bằng các ngôn ngữ này trở nên dễ dàng và đơn giản hơn. Có thể kể đến một số ngôn ngữ lập trình bậc cao như C, Basic, Pascal… trong đó C là ngôn ngữ thông dụng hơn cả trong kỹ thuật vi xử lý. Về bản chất, sử dụng các ngôn ngữ này thay cho ngôn ngữ bậc thấp là sự giảm tải cho lập trình viên trong việc nghiên cứu các tập lệnh và xây dựng các cấu trúc giải thuật. Chương trình viết bằng ngôn ngữ bậc cao cũng sẽ được một phần mềm trên máy tính gọi là trình biên dịch (Compiler) chuyển sang dạng hợp ngữ trước khi chuyển sang mã máy.

Mỗi loại ngôn ngữ có ưu và nhược điểm riêng.

Với hợp ngữ (đại diện cho ngôn ngữ bậc thấp):

- *Ưu điểm:* mã máy sinh ra rất ngắn gọn, thời gian xử lý của CPU vì thế cũng được giảm thiểu, trình hợp ngữ (Assembler) của các họ vi điều khiển đều miễn phí đối với người sử dụng.
- *Nhược điểm:* khó khăn trong việc tiếp cận với tập lệnh (tuy ở dạng mã gợi nhớ nhưng vẫn chưa thực sự gần với ngôn ngữ con người), các cấu trúc giải thuật (if…else, for…, switch…case…) hầu hết không có sẵn, vì vậy quá trình lập trình khó khăn, mất nhiều thời gian và công sức, việc kế thừa và phát triển là gần như không thể.

Với ngôn ngữ C (đại diện cho ngôn ngữ bậc cao):

- *Ưu điểm:* ngôn ngữ gần với ngôn ngữ con người, các cấu trúc giải thuật có sẵn, do đó tạo sự thuận tiện, dễ dàng trong sự diễn đạt thuật toán, việc kế thừa và phát triển là khả thi, tốn ít thời gian.

- *Nhược điểm:* mã máy sinh ra thường dài hơn so với hợp ngữ (tất nhiên cũng còn tùy vào năng lực của lập trình viên), thời gian xử lý của CPU vì thế cũng dài hơn, các trình biên dịch (Compiler) tùy theo cấp độ tối ưu mà được thiết kế và bán với giá rất cao.

Trong thực tế hiện nay, các vi điều khiển có tài nguyên (resourse) nói chung và bộ nhớ (memory) nói riêng rất phong phú và dồi dào. Mặt khác các trình biên dịch (Compiler) cũng được thiết kế ngày càng tối ưu, hỗ trợ rất nhiều các thao tác xử lý giải thuật, cho phép trộn lệnh hợp ngữ vào những tình huống yêu cầu khắt khe về mặt thời gian và lượng mã máy sinh ra. Chính vì thế yêu cầu về tối giản mã máy khi lập trình không còn quá bức xúc như trước kia. Sử dụng ngôn ngữ bậc cao giúp rút ngắn rất nhiều thời gian nghiên cứu, thiết kế sản phẩm trước khi đưa ra thị trường (time to market), nâng cao khả năng kế thừa, phát triển, cải tiến các tính năng sản phẩm, từ đó kéo dài chu kỳ sống (life time) của sản phẩm trên thị trường. Đó là lý do tại sao ngôn ngữ bậc cao (điển hình là ngôn ngữ C) là sự lựa chọn của hầu hết những người tác nghiệp trên lĩnh vực kỹ thuật vi xử lý.

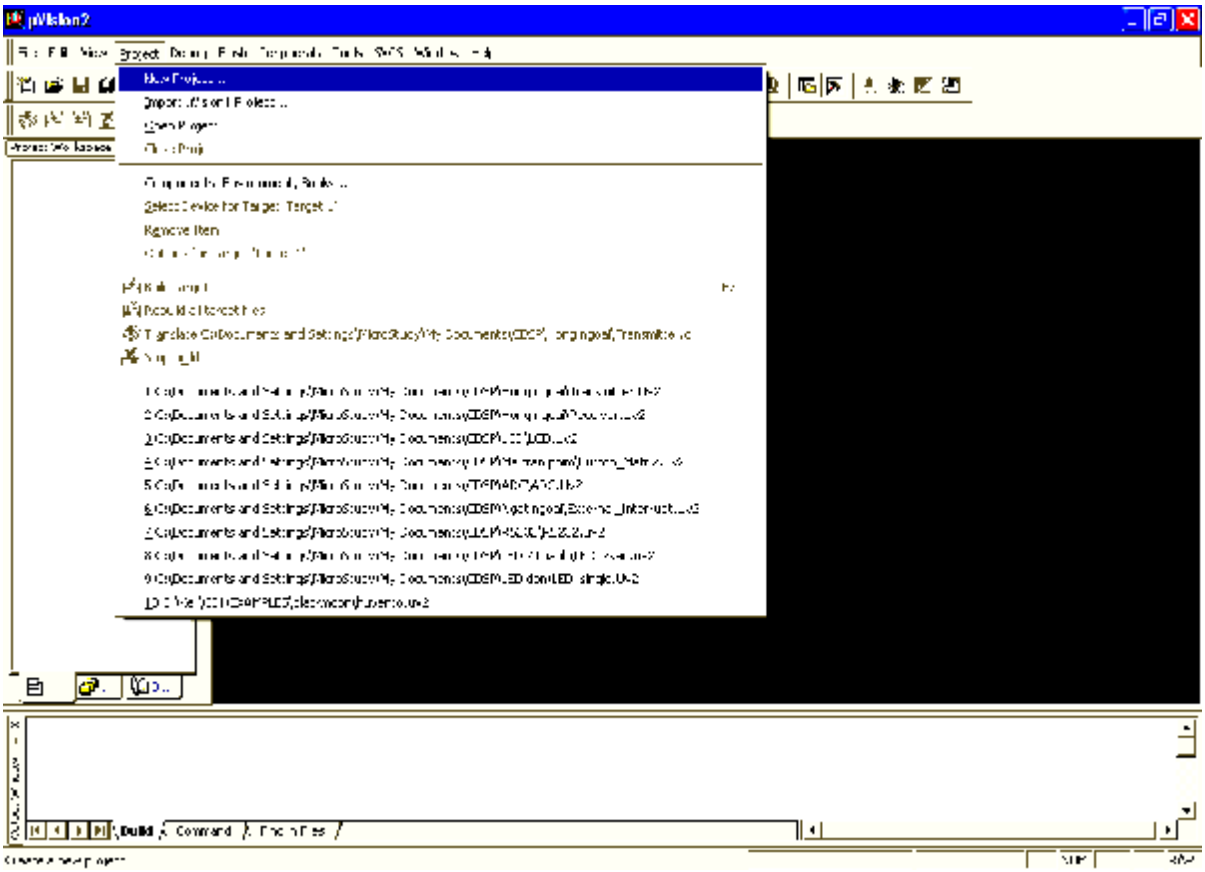
Chương III: Hướng dẫn sử dụng trình biên dịch Keil C cho họ vi điều khiển 8051

Trong số các trình biên dịch C (C Compiler) cho họ vi điều khiển 8051, Keil C là một trình biên dịch tối ưu, được sử dụng rộng rãi. Chương này chủ yếu hướng dẫn sử dụng trình biên dịch này trong việc thiết kế phần mềm cho họ vi điều khiển 8051.

Cài đặt

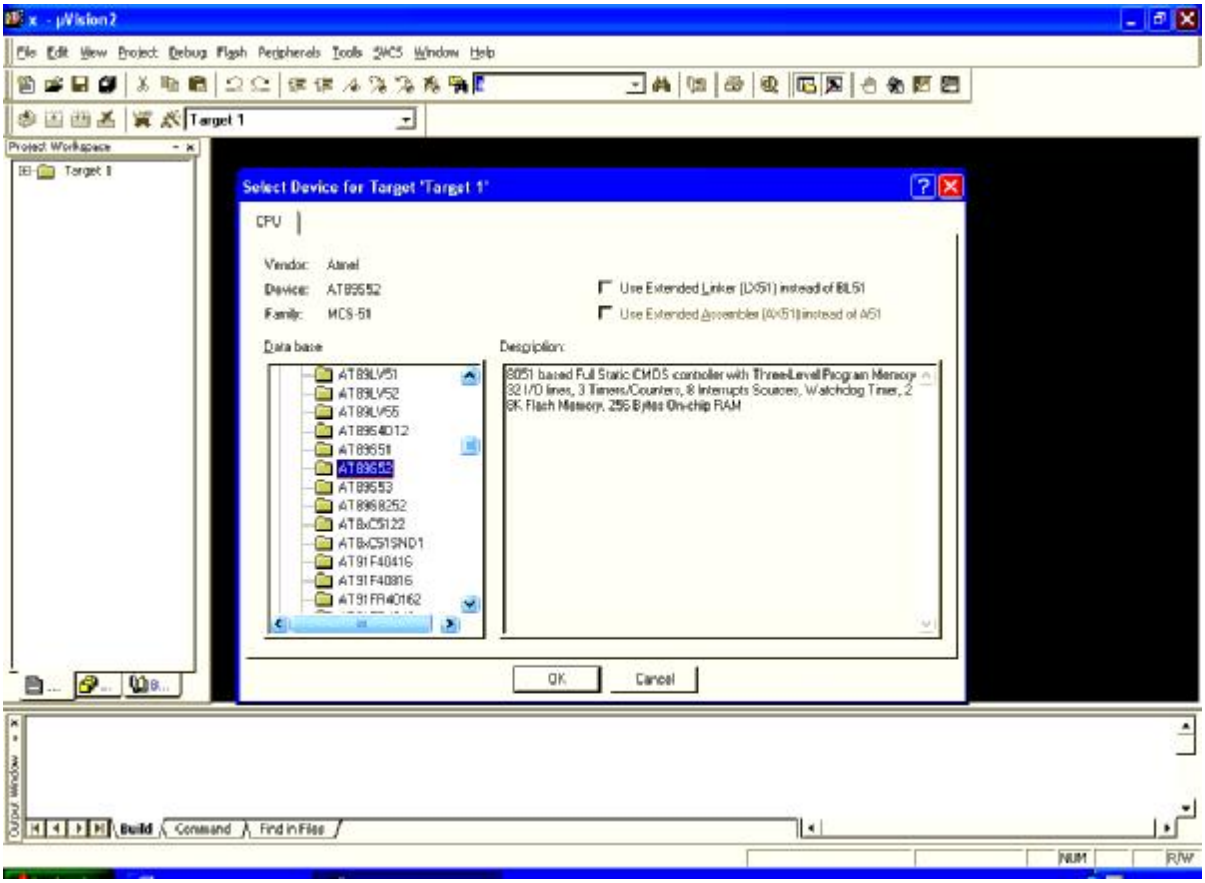
Keil C chạy được trên các Hệ điều hành Win98, Windows2000, WindowsME, WindowsXP. Để cài đặt, hãy chạy file setup.exe trong thư mục Setup của Keil C trên đĩa CD kèm theo. Tiếp đó hãy theo các chỉ dẫn của giao diện cài đặt.

Tạo một Project mới



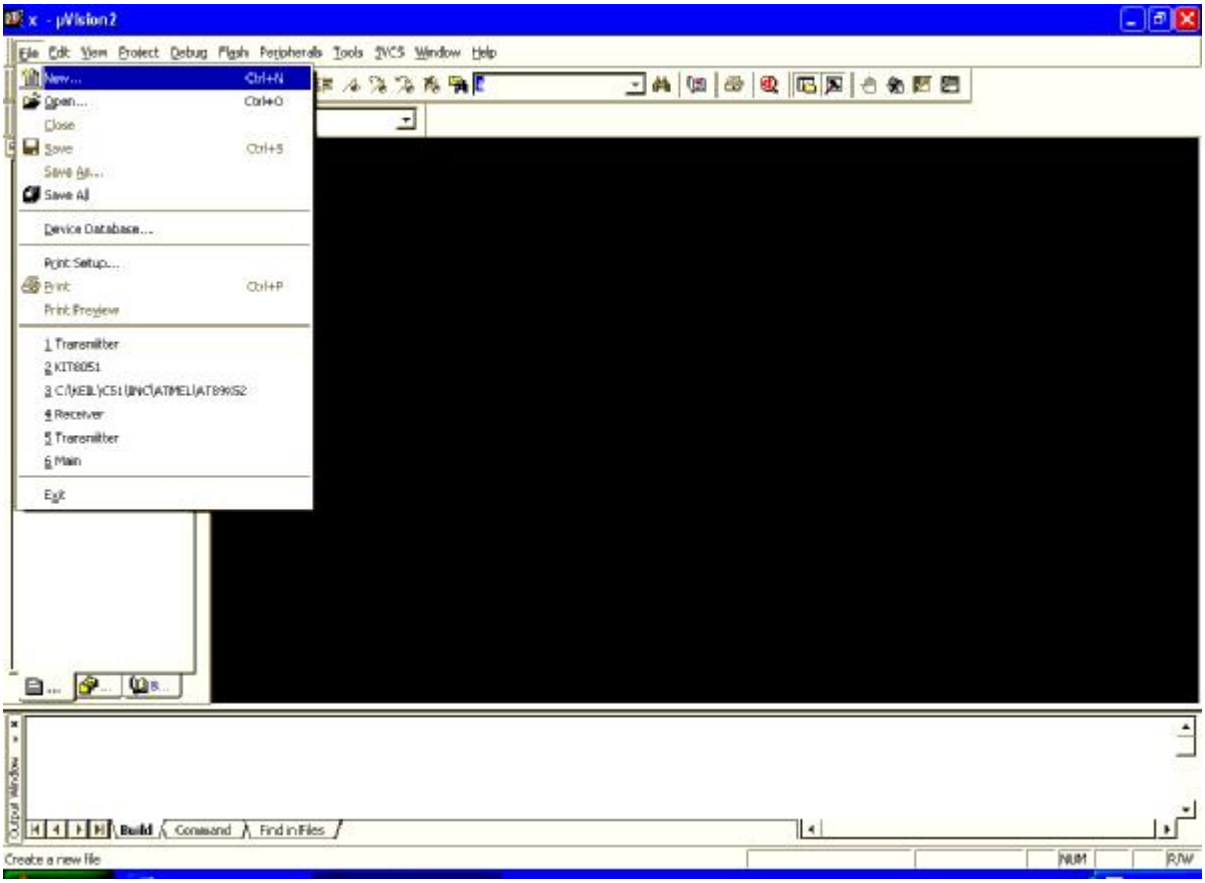
Tiếp đó gõ tên project vào hộp thoại. Chọn đường dẫn và bấm OK.

Chọn tiếp loại vi điều khiển sẽ sử dụng (trong trường hợp này là AT89S52).



Chọn câu trả lời "**No**" khi được hỏi "*Copy Standart Startup Code to Project Folder and Add File to Project?*"

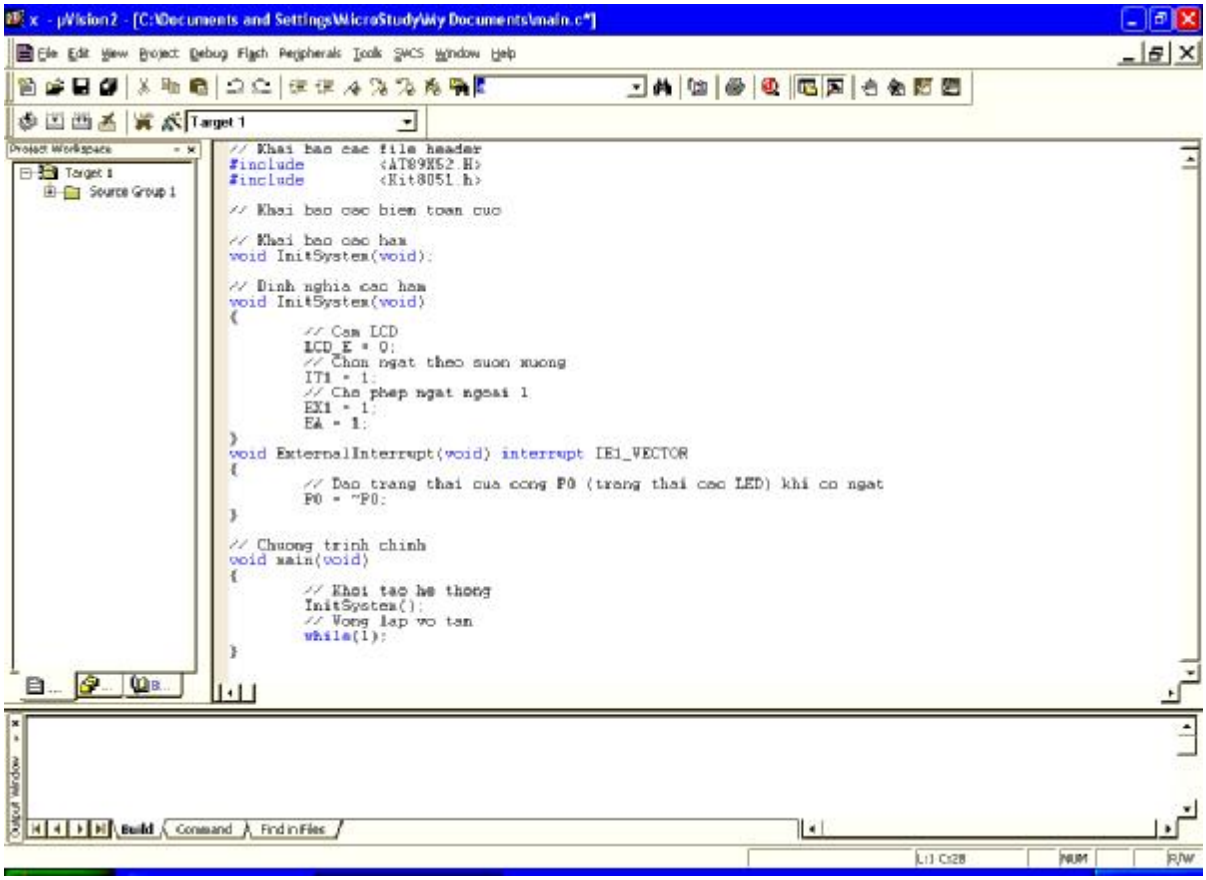
Tạo một file mới



Sau khi Keil tạo cho ta một file mặc định dạng Text, hãy Save File lại dưới dạng mong muốn (*.c nếu là file mã nguồn, hoặc *.h nếu là file header).
Một Project chủ yếu sử dụng hai loại file nói trên.

Tiếp đó thực hiện soạn thảo các file theo ý muốn.

Cấu trúc một chương trình



Một số lưu ý khi lập trình với Keil C

Khai báo các file header

Khi sử dụng loại vi điều khiển 8051 nào (đã lựa chọn trong khi tạo Project mới) thì phải sử dụng file header của loại đó. Trong trường hợp này ta sử dụng file "AT89X52.H" cho vi điều khiểể AT89S52. Các file header được tìm trong thư mục …C51\INC\ của Keil C đã cài ra.

Định nghĩa hằng số trong bộ nhớ chương trình

unsigned char code <tên biến>;

Ví dụ định nghĩa một mảng 3 hằng số:

unsigned char code array[3] = {1,2,3};

Định nghĩa các chương trình con phục vụ ngắt

void <tên chương trình> **(void) interrupt** <tên Vector ngắt>

<tên chương trình> do lập trình viên tùy ý đặt.

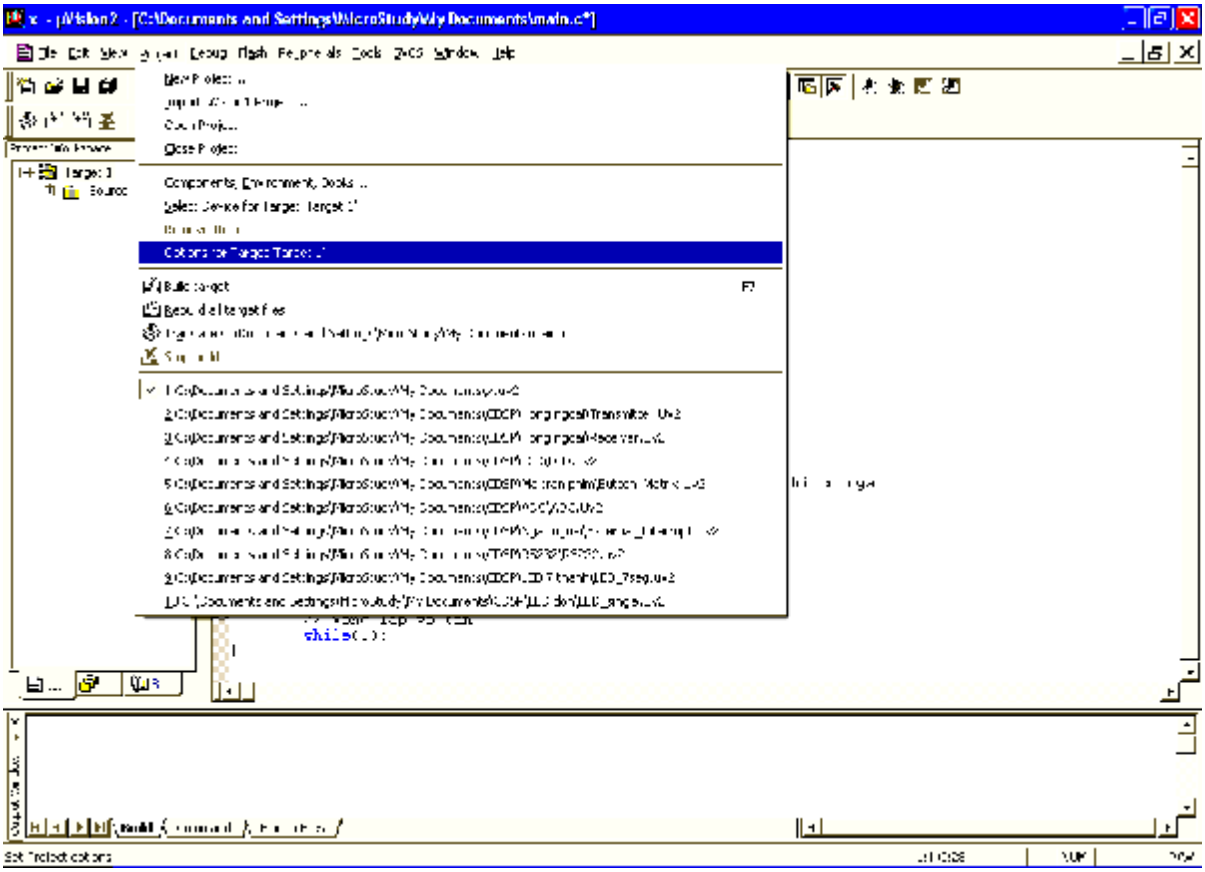
<tên Vector ngắt> được tra ở phần cuối file header (AT89X52.H).

Không nên viết lệnh ở dạng biểu thức dài mà nên tách ra thành từng phép tính nhỏ thực hiện lần lượt.

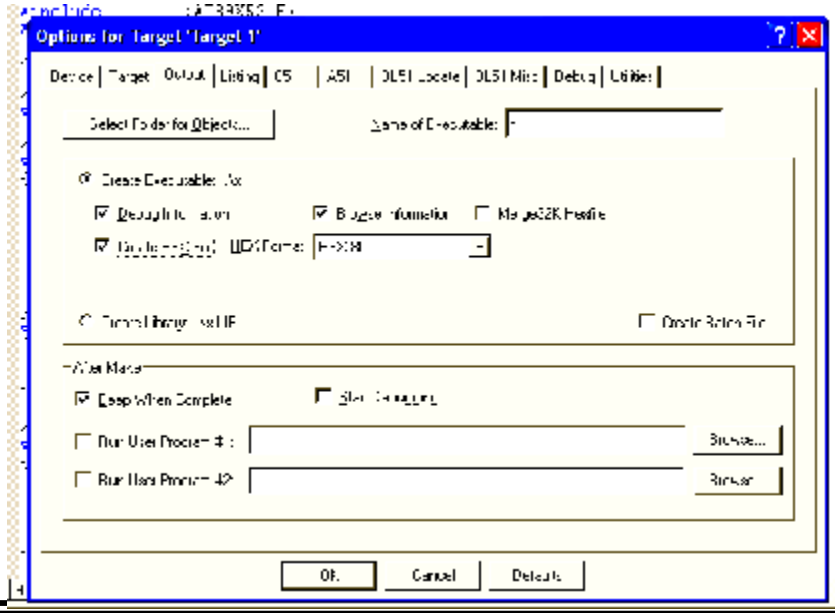
Các tính năng khác giống như ANSI C.

Biên dịch Project

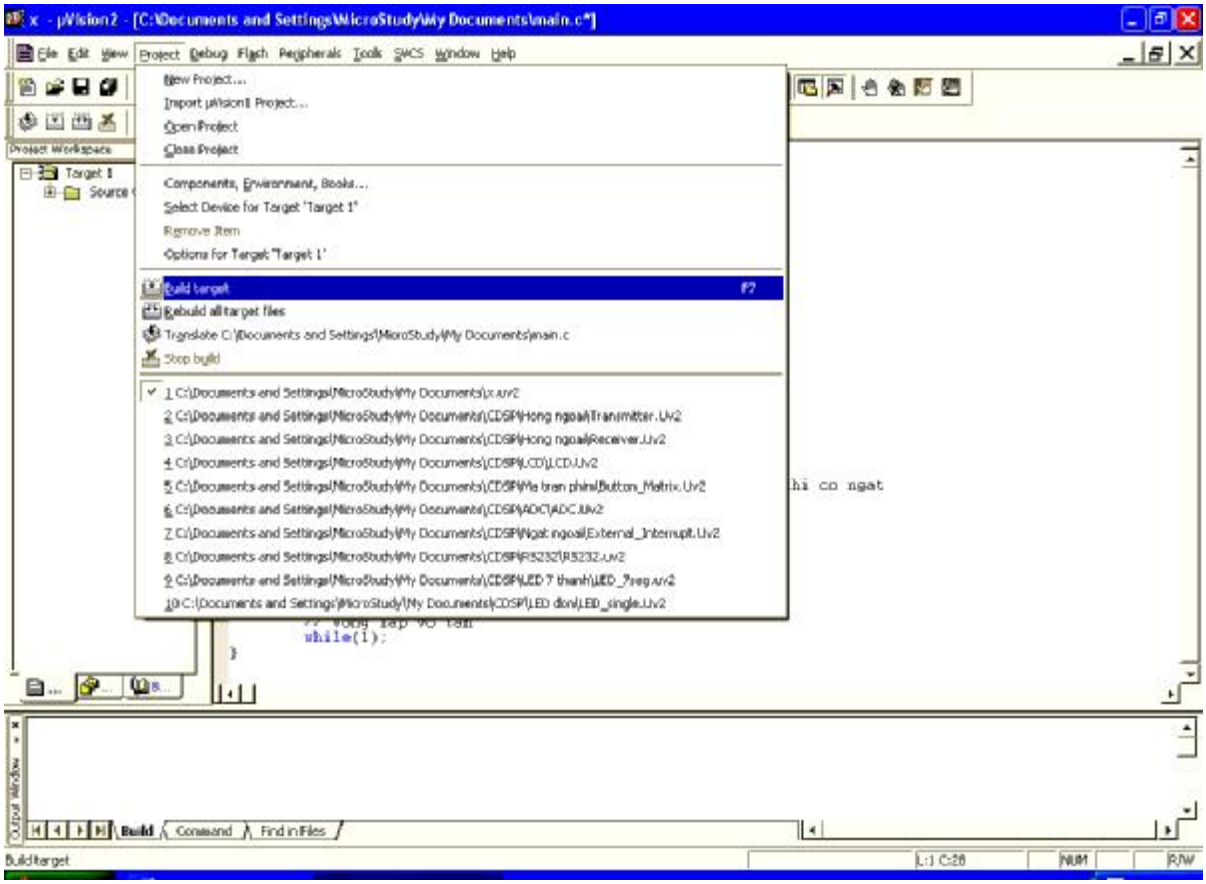
Để tạo ra file mã máy dạng *.hex nạp vào chip, click chuột phải như hình vẽ.



Trong hộp thoại hiện ra, hãy check vào Creat Hex File như chỉ dẫn.



Cuối cùng chọn theo Menu như hình ảnh, hoặc bấm phím F7.



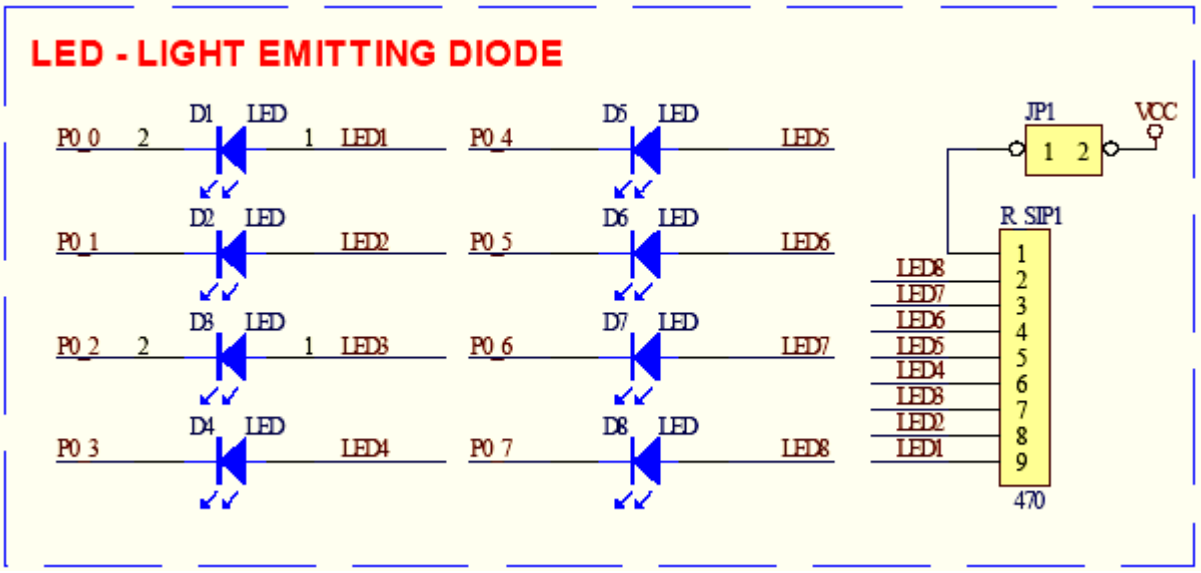
File mã máy *.hex tạo ra sẽ nằm trong cùng một thư mục với các file khác của Project.

Sử dụng phần mềm SPI – Flash Programmer có sẵn trên đĩa CD kèm theo để mở file mã máy (bằng nút Open) và nạp xuống chip (nút Program). Phần mềm này chạy trực tiếp không cần cài đặt và chạy tốt nhất trên các Win98, 2000 và ME, với WindowsXP đôi khi không tương thích ở một số rất ít các trường hợp.

ChươngIV: Hướng dẫn thực hành một số bài thí nghiệm mẫu với 8051 Starter Kit

Bài 1: Ghép nối với LED đơn – Light Emitting Diode

Nguyên lý thiết kế



LED đơn được nối trực tiếp với cổng vào ra của vi điều khiển AT89, sử dụng điện trở hạn dòng 470 Ohm. Vi điều khiển đưa ra mức logic 0 làm đèn sang, mức logic 1 làm đèn tắt.

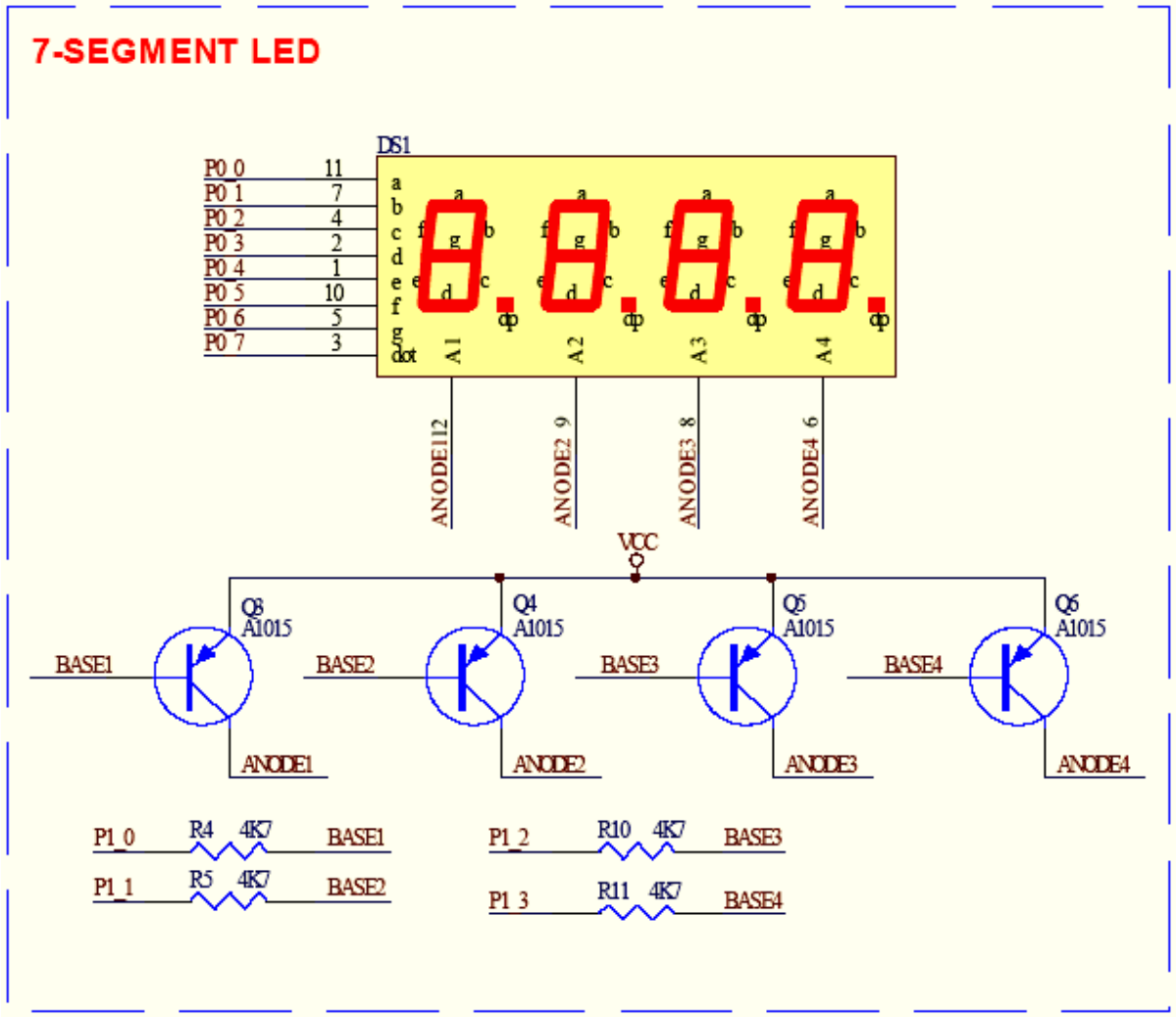
Ví dụ: Thực hiện nhấp nháy LED.

```
// Khai bao cac file header
#include      <AT89X52.H>
#include      <Kit8051.h>

// Khai bao cac ham
void Delay(unsigned int n);
void InitSystem(void);

// Dinh nghia cac ham
void Delay(unsigned int n)
{
        unsigned int i,j;
        for(i=0;i<n;i++)
                for(j=0;j<100;j++);
```

```
}
void InitSystem(void)
{
        LCD_E = 0;
}
// Chuong trinh chinh
void main(void)
{
        // Khoi tao he thong
        InitSystem();
        // Vong lap vo tan
        while(1)
        {
                P0 = ~P0;
                Delay(1000);
        }
}
```

Bài 2: Ghép nối với LED 7 thanh – 7 Segment LED

Nguyên lý thiết kế



LED 7 thanh sử dụng là loại Anode chung. Các LED 7 thanh được nối chung chân dữ liệu với nhau và nối với một cổng vào ra 8 bit của vi điều khiển. Việc cấp điện cho từng LED được thực hiện bởi các transistor, điều khiển bởi các chân vào ra khác của vi điều khiển AT89. Các LED được cấp nguồn cho sáng ở các thời điểm khác nhau, việc quét các LED này được thực hiện liên tục, do hiện tượng ảnh lưu võng mạc mà con người nhìn thấy các LED hiển thị một cách liên tục.

Ví dụ: Hiển thị 4 số 1234 ở 4 LED 7 thanh.

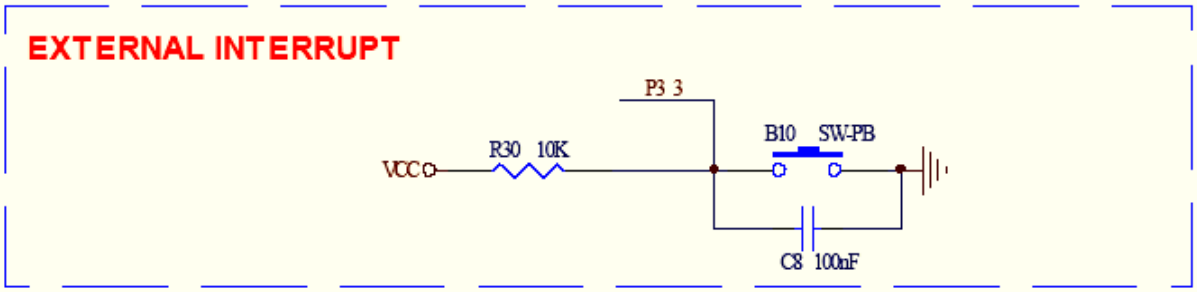
```
// Khai bao cac file header
#include      <AT89X52.H>
#include      <Kit8051.h>
```

```
// Khai bao cac bien toan cuc
unsigned char code LED_code[] =  {Number0,
                                  Number1,
                                  Number2,
                                  Number3,
                                  Number4,
                                  Number5,
                                  Number6,
                                  Number7,
                                  Number8,
                                  Number9};

// Khai bao cac ham
void Delay(unsigned int n);
void InitSystem(void);
void Display(unsigned char digit1,
             unsigned char digit2,
             unsigned char digit3,
             unsigned char digit4);


// Dinh nghia cac ham
void Delay(unsigned int n)
{
        unsigned int i,j;
        for(i=0;i<n;i++)
                for(j=0;j<100;j++);
}
void Display(unsigned char digit1,
             unsigned char digit2,
             unsigned char digit3,
             unsigned char digit4)
{
        // Hien thi so thu nhat
        LED = LED_code[digit1];
        K1 = 0;
        Delay(1);
        K1 = 1;
```

```
        // Hien thi so thu hai
        LED = LED_code[digit2];
        K2 = 0;
        Delay(1);
        K2 = 1;

        // Hien thi so thu ba
        LED = LED_code[digit3];
        K3 = 0;
        Delay(1);
        K3 = 1;


        // Hien thi so thu tu
        LED = LED_code[digit4];
        K4 = 0;
        Delay(1);
        K4 = 1;
}
void InitSystem(void)
{
        LCD_E = 0;
}
// Chuong trinh chinh
void main(void)
{
        // Khoi tao he thong
        InitSystem();
        // Vong lap vo tan
        while(1)
        {
                Delay(10);
                Display(1,2,3,4);
        }
}
```

Bài 3: Sử dụng ngắt ngoài của vi điều khiển 8051 – External Interrupt

Nguyên lý thiết kế



Ngắt ngoài của AT89 được kích hoạt bởi một phím bấm bên ngoài. Khi phím bấm, mức lôgic 0 sẽ được đưa vào chân ngắt của vi điều khiển, khi nhả phím, mức lôgic trở lại mức cao. Tụ C8 được mắc nhằm hạn chế rung phím do cơ khí chế tạo không hoàn toàn chính xác của công tắc.

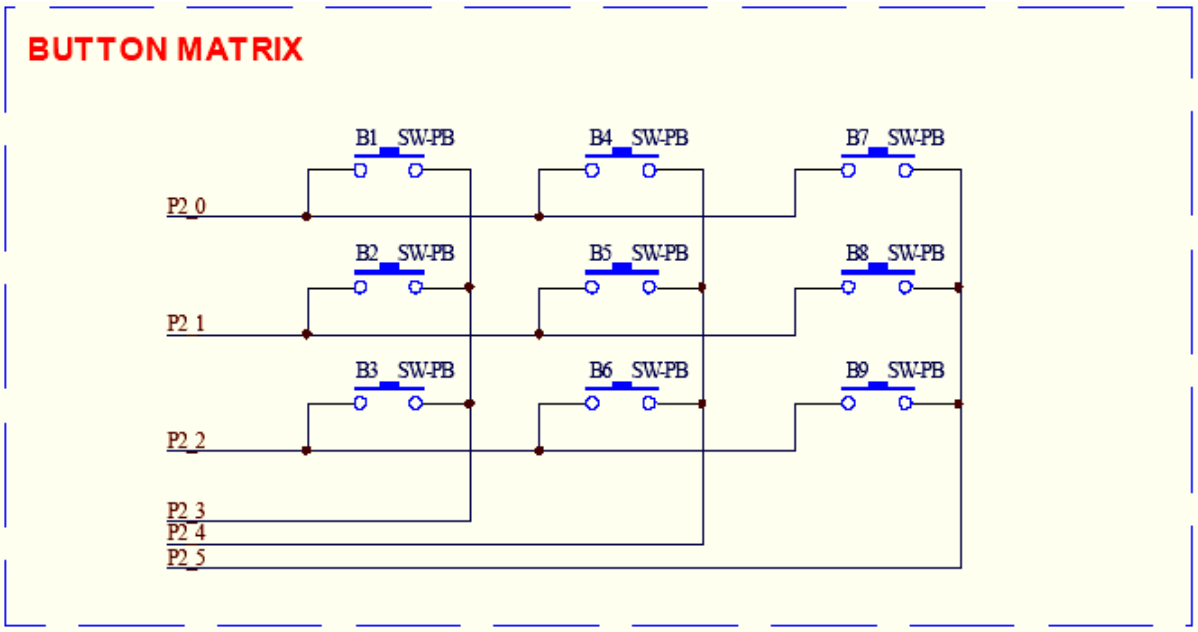
Ví dụ: Nhận tín hiệu phím bấm bên ngoài bằng ngắt, thể hiện bằng việc đảo trạng thái LED đơn.

```
// Khai bao cac file header
#include      <AT89X52.H>
#include      <Kit8051.h>

// Khai bao cac bien toan cuc

// Khai bao cac ham
void InitSystem(void);

// Dinh nghia cac ham
void InitSystem(void)
{
      // Cam LCD
      LCD_E = 0;
      // Chon ngat theo suon xuong
      IT1 = 1;
      // Cho phep ngat ngoai 1
      EX1 = 1;
      EA = 1;
}
void ExternalInterrupt(void) interrupt IE1_VECTOR
{
```

```
        // Dao trang thai cua cong P0 (trang thai cac LED) khi co ngat
        P0 = ~P0;
}

// Chuong trinh chinh
void main(void)
{
        // Khoi tao he thong
        InitSystem();
        // Vong lap vo tan
        while(1);
}
```

Bài 4: Ghép nối với ma trận phím – Button Matrix

Nguyên lý thiết kế



Phím bấm được nối thành ma trận 3 hàng x 3 cột, các hàng và cột được nối với các chân cổng vào ra của vi điều khiển AT89. Khi một phím được bấm, nó sẽ nối một hàng và một cột tương ứng. Thuật toán quét phím được sử dụng là lần lượt tìm hàng và tìm cột (hoặc ngược lại). Khi tìm hàng, các hàng sẽ được đặt làm đầu vào, các cột được đặt làm đầu ra mức thấp. Sau đó kiểm tra các hàng xem có hàng nào ở mức thấp hay không (có phím nào bấm gây ra nối với cột hay không)? Sau khi xác định được hàng sẽ đặt các cột làm đầu vào, hàng vừa tìm được làm đầu ra mức

thấp. Việc kiểm tra được tiến hành với các cột. Sau khi xác định được hàng và cột sẽ suy ra phím được bấm.

Ví dụ: Quét ma trận phím và hiển thị số thứ tự phím được bấm lên LED 7 thanh.

```
// Khai bao cac file header
#include      <AT89X52.H>
#include      <Kit8051.h>

// Khai bao cac bien toan cuc
unsigned char code LED_code[] =  {Number0,
                                  Number1,
                                  Number2,
                                  Number3,
                                  Number4,
                                  Number5,
                                  Number6,
                                  Number7,
                                  Number8,
                                  Number9};
unsigned char stt=0;
// Khai bao cac ham
void Delay(unsigned int n);
void InitSystem(void);
void ScanMatrix(void);
void Display(unsigned char number);


// Dinh nghia cac ham
void Delay(unsigned int n)
{
        unsigned int i,j;
        for(i=0;i<n;i++)
                for(j=0;j<100;j++);
}
void Display(unsigned char number)
{
        // Hien thi so thu tu phim bam ra LED cuoi cung
        LED = LED_code[number];
        K4 = 0;
```

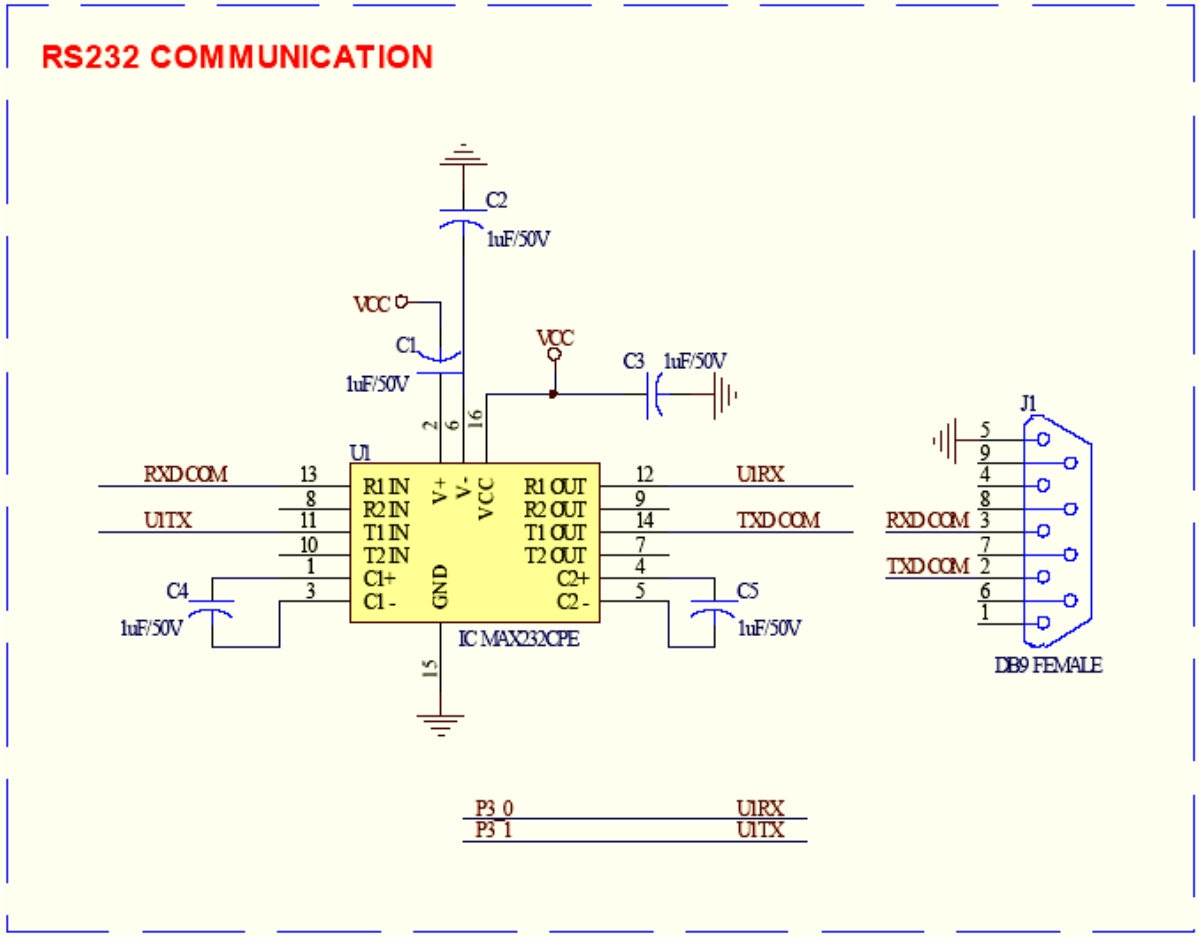
```
        Delay(1);
        K4 = 1;
}
void InitSystem(void)
{
        // Cam LCD
        LCD_E = 0;
        // Cau hinh Timer0 o che do 16 bit
        TMOD = 0x01;
        // Cho phep ngat tran Timer 0
        ET0 = 1;
        EA = 1;
        // Cho Timer 0 chay
        TR0 = 1;
}
void Timer0Interrupt(void) interrupt TF0_VECTOR
{
        // Dung Timer 0
        TR0 = 0;
        // Nap lai gia tri = 55536--> con 10000 clock nua la tran
        TH0 = (unsigned char)(55536>>8);
        TL0 = (unsigned char)(55536);
        // Tiep tuc cho chay Timer 0
        TR0 = 1;
        Display(stt);
}
void ScanMatrix(void)
{
        // Chuyen cac hang lam dau vao, cac cot lam dau ra muc thap
        ROW_1 = 1;
        ROW_2 = 1;
        ROW_3 = 1;
        COL_1 = 0;
        COL_2 = 0;
        COL_3 = 0;
        // Kiem tra xem co phim bam?
        if((!ROW_1)|(!ROW_2)|(!ROW_3))
        {
                // Chong rung phim
```

```
Delay(100);
// Kiem tra lai
// Neu la hang 1 co phim bam
if(!ROW_1)
{
        // Chuyen cac cot lam dau vao
        COL_1 = 1;
        COL_2 = 1;
        COL_3 = 1;
        // Hang 1 lam dau ra muc thap
        ROW_1 = 0;
        // Kiem tra cac cot de xac dinh phim
        if(!COL_1)          stt = 1;
        else if (!COL_2)    stt = 2;
        else if (!COL_3)    stt = 3;
}
// Neu la hang 2 co phim bam
if(!ROW_2)
{
        // Chuyen cac cot lam dau vao
        COL_1 = 1;
        COL_2 = 1;
        COL_3 = 1;
        // Hang 2 lam dau ra muc thap
        ROW_2 = 0;
        // Kiem tra cac cot de xac dinh phim
        if(!COL_1)          stt = 4;
        else if (!COL_2)    stt = 5;
        else if (!COL_3)    stt = 6;
}
// Neu la hang 3 co phim bam
if(!ROW_3)
{
        // Chuyen cac cot lam dau vao
        COL_1 = 1;
        COL_2 = 1;
        COL_3 = 1;
        // Hang 3 lam dau ra muc thap
        ROW_3 = 0;
```

```
                        // Kiem tra cac cot de xac dinh phim
                        if(!COL_1)          stt = 7;
                        else if (!COL_2)    stt = 8;
                        else if (!COL_3)    stt = 9;
                }
        }
}
// Chuong trinh chinh
void main(void)
{
        // Khoi tao he thong
        InitSystem();
        // Vong lap vo tan
        while(1)
        {
                ScanMatrix();
        }
}
```

Bài 5: Ghép nối với máy tính qua giao diện RS232
RS232 Communication

Nguyên lý thiết kế



Giao thức truyền thông giữa cổng COM của máy tính và cổng nối tiếp UART của vi điều khiển AT89 là tương thích. Tuy nhiên về mặt điện học thì có sự khác biệt về mức lôgic giữa máy tính và vi điều khiển. Vi mạch MAX232CPE được sử dụng để tạo ra sự tương thích về mặt điện học, tự động chuyển đổi mức lôgic giữa vi điều khiển và máy tính.

Ví dụ: Thiết kế thiết bị tự động truyền trả lại PC những dữ liệu nhận được.

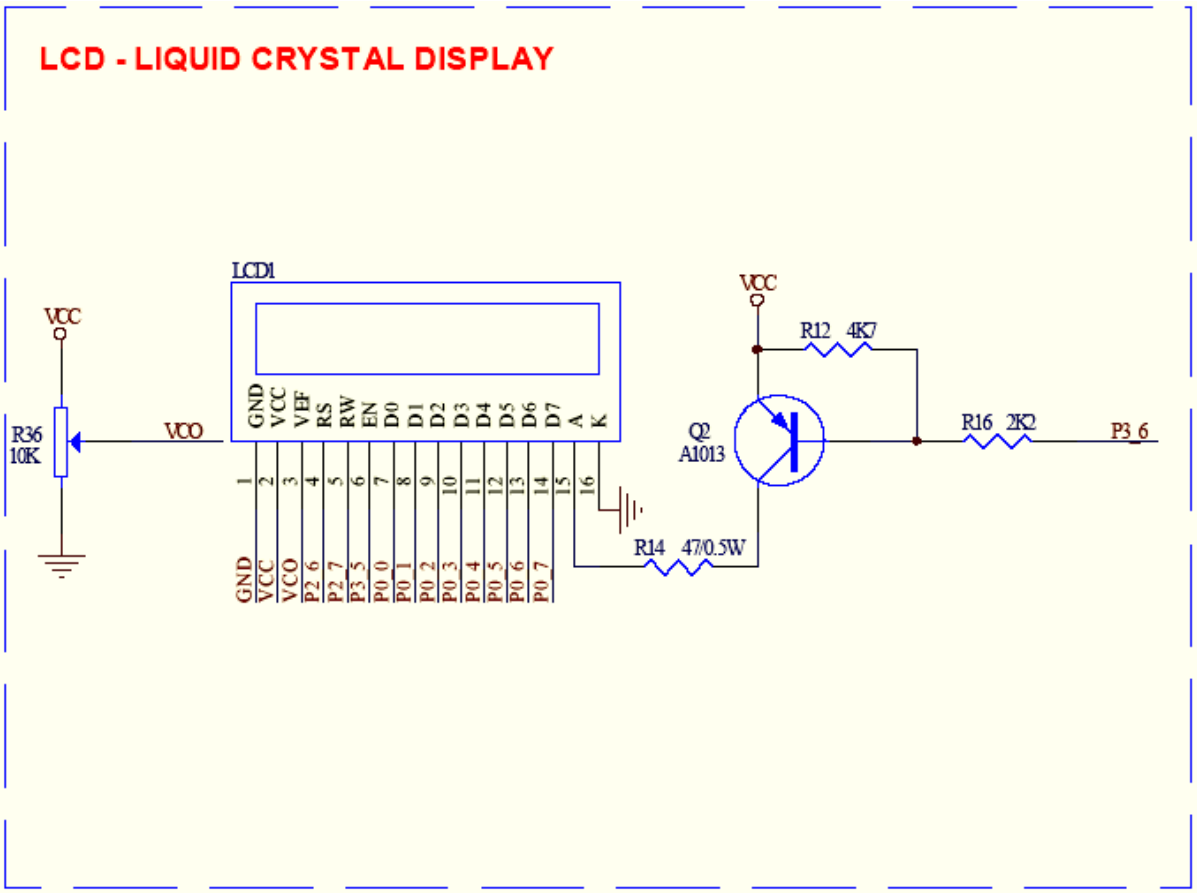
```
// Khai bao cac file header
#include      <AT89X52.H>
#include      <Kit8051.h>
```

```
// Khai bao cac bien toan cuc

// Khai bao cac ham
void InitSystem(void);

// Dinh nghia cac ham
void InitSystem(void)
{
	// Cam LCD
	LCD_E = 0;
	// Khoi tao cong noi tiep
	SCON = 0x50;
	// Khoi tao Timer1 dung de tao Baud Rate
	TMOD = 0x20;
	TH1 = 0xFD;
	TR1 = 1;
	// Cho phep ngat cong noi tiep
	ES = 1;
	EA = 1;
}
void SerialPortInterrupt(void) interrupt SIO_VECTOR
{
	unsigned char temp;
	// Kiem tra xem ngat la do nhan duoc du lieu hay truyen xong du lieu?
	if(RI==1)
	{
		RI = 0;
		// Doc du lieu nhan duoc tu bo dem
		temp = SBUF;
		// Truyen tra lai may tinh
		SBUF = temp;
	}
	else
	{
		TI = 0;
	}
}

// Chuong trinh chinh
```

```
void main(void)
{
        // Khoi tao he thong
        InitSystem();
        // Vong lap vo tan
        while(1);
}
```

Bài 6: Ghép nối với LCD – Liquid Crystal Display

Nguyên lý thiết kế



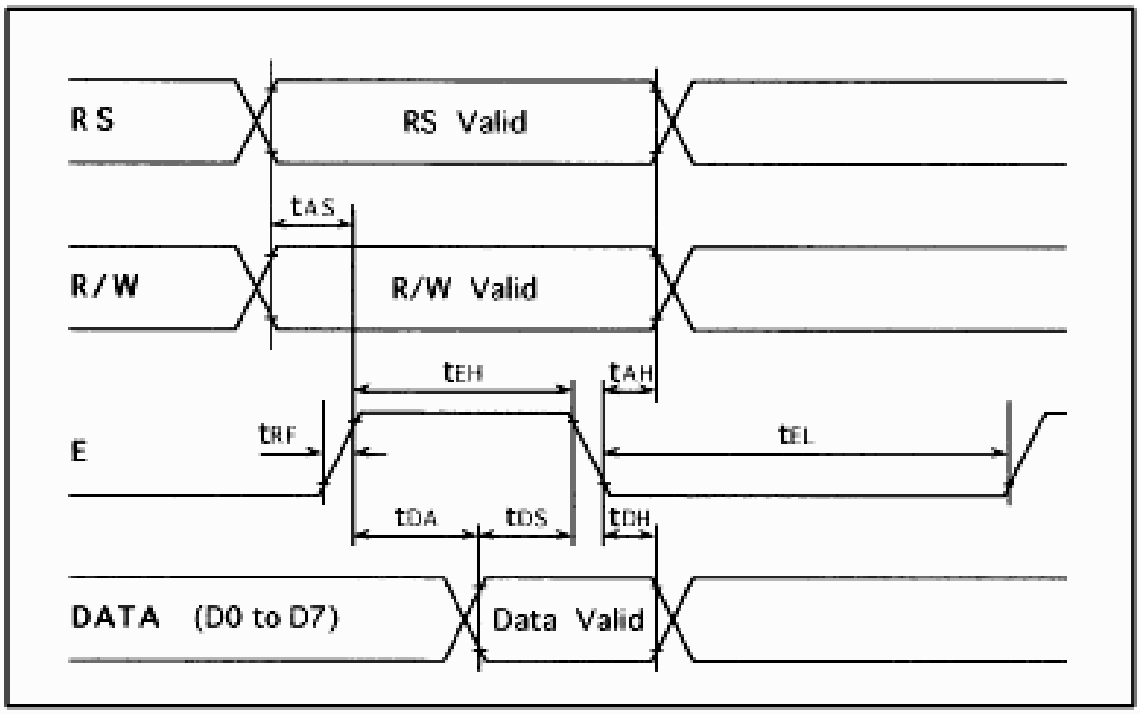
LCD là thiết bị hiển thị chuẩn, tiết kiệm năng lượng và không đòi hỏi phải quét liên tục như LED 7 thanh. Các loại LCD ký tự (character LCD) đều tuân theo một tập lệnh chung.

Table 2. The Command Control Codes.

Command	Binary								Hex
	D7	D6	D5	D4	D3	D2	D1	D0	
Clear Display	0	0	0	0	0	0	0	1	01
Display & Cursor Home	0	0	0	0	0	0	1	x	02 or 03
Character Entry Mode	0	0	0	0	0	1	I/D	S	04 to 07
Display On/Off & Cursor	0	0	0	0	1	D	U	B	08 to 0F
Display/Cursor Shift	0	0	0	1	D/C	R/L	x	x	10 to 1F
Function Set	0	0	1	8/4	2/1	10/7	x	x	20 to 3F
Set CGRAM Address	0	1	A	A	A	A	A	A	40 to 7F
Set Display Address	1	A	A	A	A	A	A	A	80 to FF

I/D: 1=Increment*, 0=Decrement
S: 1=Display shift on, 0=Display shift off*
D: 1=Display On, 0=Display Off*
U: 1=Cursor underline on, 0=Underline off*
B: 1=Cursor blink on, 0=Cursor blink off*
D/C: 1=Display shift, 0=Cursor move
R/L: 1=Right shift, 0=Left shift
8/4: 1=8 bit interface*, 0=4 bit interface
2/1: 1=2 line mode, 0=1 line mode*
10/7: 1=5x10 dot format, 0=5x7 dot format*
x = Don't care
* = Initialisation settings

Các thao tác ghi đọc được thực hiện theo biểu đồ thời gian sau:



Ví dụ: Hiển thị dòng chữ “Truong CDSP HN” lên dòng thứ nhất, dòng chữ “8051 Starter Kit” lên dòng thứ hai của LCD.

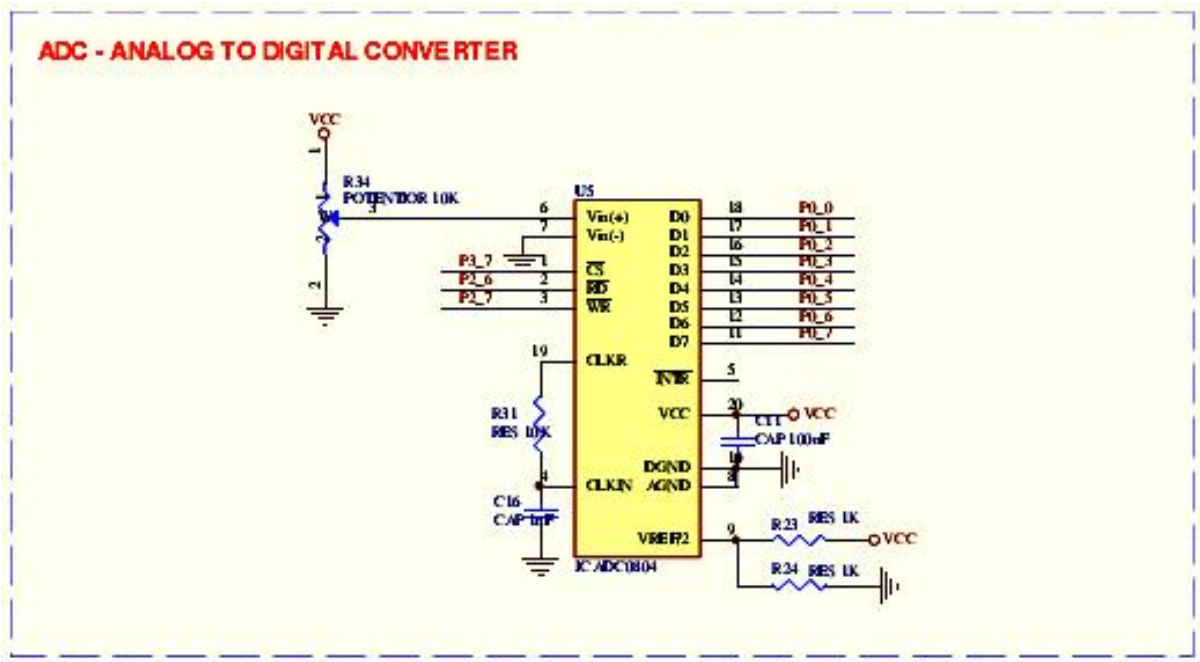
```
// Khai bao cac file header
#include	<AT89X52.H>
#include	<Kit8051.h>

// Khai bao cac bien toan cuc
unsigned char code string1[] = "Truong CDSP HN";
unsigned char code string2[] = "8051 Starter Kit";

// Khai bao cac ham
void InitSystem(void);
void Delay(unsigned int n);
void InitLCD(void);
void WriteCommand(unsigned char command);
void WriteCharacter(unsigned char character);
void WriteLCD(unsigned char x);
void SendString2LCD(unsigned char code *p);
void DisplayText(void);

// Dinh nghia cac ham
void InitSystem(void)
{
	// Cam LCD
	LCD_E = 0;
	// Sang den backlight
	LCD_BL = 0;
	// Tre de LCD tu khoi tao ben trong (it nhat 15ms)
	Delay(100);
	// Tat den backlight
	LCD_BL = 1;
	InitLCD();
}
void Delay(unsigned int n)
{
	unsigned int i,j;
	for(i=0;i<n;i++)
		for(j=0;j<100;j++);
}
```

```
void InitLCD(void)
{
	WriteCommand(0x30);
	WriteCommand(0x30);
	WriteCommand(0x30);

	// 8 bit, 2 lines, font 5x7
	WriteCommand(0x38);
	// Display on, hide cursor
	WriteCommand(0x0C);

	// Xoa man hinh
	WriteCommand(0x01);
}
void DisplayText(void)
{
	// Dich con tro den vi tri thu 2, dong thu nhat
	WriteCommand(0x81);
	SendString2LCD(string1);
	// Dich con tro den dau dong thu hai
	WriteCommand(0xC0);
	SendString2LCD(string2);
}
void WriteLCD(unsigned char x)
{
	LCD_RW = 0;
	LCD_DATA = x;

	LCD_E = 1;
	LCD_E = 0;
	Delay(5);
}
void WriteCommand(unsigned char command)
{
	LCD_RS = 0;
	WriteLCD(command);
}
void WriteCharacter(unsigned char character)
{
```

```
        LCD_RS = 1;
        WriteLCD(character);
}
void SendString2LCD(unsigned char code *p)
{
        unsigned char i=0;
        while(p[i]!=0)
        {
                WriteCharacter(p[i]);
                i++;
                Delay(200);
        }
}
// Chuong trinh chinh
void main(void)
{
        // Khoi tao he thong
        InitSystem();
        DisplayText();
        // Vong lap vo tan
        while(1);
}
```

Bài 7: Ghép nối với ADC – Analog to Digital Converter

Nguyên lý thiết kế



ADC0804 là một trong những IC cho phép chuyển đổi điện áp dạng tương tự (analog) sang giá trị dạng số (digital), được ứng dụng trong rất nhiều các thiết bị đo lường. Đầu vào điện áp có thể trong dải từ 0 đến +5V. Giá trị số đầu ra trong dải từ 0 đến 255. Các tín hiệu D0-D7 dùng để đưa dữ liệu đã chuyển đổi xong tới vi điều khiển. Các tín hiệu CS, RD, WR (đều tích cực ở mức thấp) dùng để ra lệnh cho ADC bắt đầu chuyển đổi hoặc đưa dữ liệu đã chuyển đổi ra. Kể từ khi nhận được lệnh yêu cầu chuyển đổi từ vi điều khiển, ADC sẽ mất một khoảng thời gian để thực hiện nhiệm vụ. Trong khoảng thời gian này tính hiệu INTR sẽ ở mức cao (báo bận). Khi quá trình chuyển đổi kết thúc, tín hiệu này sẽ tự động trở lại mức thấp để thông báo rằng ADC đã chuyển đổi xong và có thể thực hiện lần chuyển đổi tiếp theo. Lập trình viên có thể thăm dò tín hiệu INTR để biết khi nào ADC bận, khi nào ADC sẵn sàng hoặc đơn giản chỉ cần đợi một vài mili giây trước khi đọc kết quả hoặc ra lệnh chuyển đổi tiếp theo.

Ví dụ: Đo điện áp (trong dải từ 0 đến +5V) được tạo ra từ chiết áp và hiển thị giá trị đo được lên LCD.

```
// Khai bao cac file header
#include      <AT89X52.H>
#include      <Kit8051.h>

// Khai bao cac bien toan cuc
unsigned char code string1[] = "8051 Starter Kit";
unsigned char code string2[] = "Dien ap = ";
unsigned char voltage,digit1,digit2;

// Khai bao cac ham
void InitSystem(void);
void Delay(unsigned int n);
void DelayShort(void);
void InitLCD(void);
void WriteCommand(unsigned char command);
void WriteCharacter(unsigned char character);
void WriteLCD(unsigned char x);
void SendString2LCD(unsigned char code *p);
void DisplayText(void);
void DisplayVoltage(void);
void Convert();
void Calculate();
```

```
// Dinh nghia cac ham
void InitSystem(void)
{
	// Cam LCD
	LCD_E = 0;
	// Sang den backlight
	LCD_BL = 0;
	// Tre de LCD tu khoi tao ben trong (it nhat 15ms)
	Delay(100);
	// Tat den backlight
	LCD_BL = 1;
	InitLCD();
}
void Delay(unsigned int n)
{
	unsigned int i,j;
	for(i=0;i<n;i++)
		for(j=0;j<100;j++);
}
void DelayShort(void)
{
	unsigned char i;
	for(i=0;i<10;i++);
}
void InitLCD(void)
{
	WriteCommand(0x30);
	WriteCommand(0x30);
	WriteCommand(0x30);

	// 8 bit, 2 lines, font 5x7
	WriteCommand(0x38);
	// Display on, hide cursor
	WriteCommand(0x0C);

	// Xoa man hinh
	WriteCommand(0x01);
}
void DisplayText(void)
```

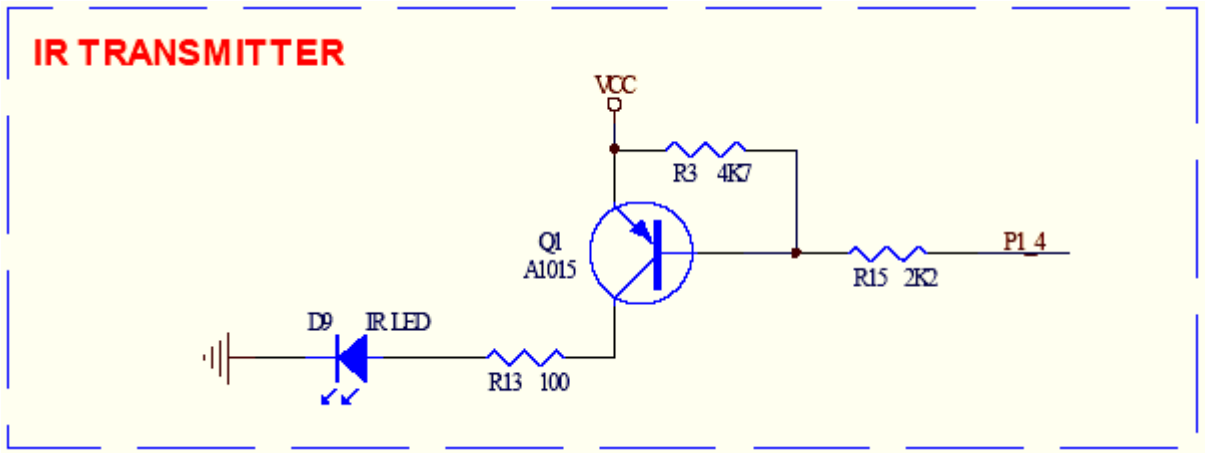
```
{
        // Dich con tro den dau dong thu nhat
        WriteCommand(0x80);
        SendString2LCD(string1);
        // Dich con tro den dau dong thu hai
        WriteCommand(0xC1);
        SendString2LCD(string2);
        // Dich con tro den vi tri don vi do
        WriteCommand(0xCE);
        WriteCharacter('V');
}
void WriteLCD(unsigned char x)
{
        LCD_RW = 0;
        LCD_DATA = x;

        LCD_E = 1;
        LCD_E = 0;
        Delay(5);
}
void WriteCommand(unsigned char command)
{
        LCD_RS = 0;
        WriteLCD(command);
}
void WriteCharacter(unsigned char character)
{
        LCD_RS = 1;
        WriteLCD(character);
}
void SendString2LCD(unsigned char code *p)
{
        unsigned char i=0;
        while(p[i]!=0)
        {
                WriteCharacter(p[i]);
                i++;
                Delay(50);
        }
```

```
}
void Convert(void)
{
        ADC_CS = 0;
        DelayShort();
        ADC_WR = 0;
        DelayShort();
        ADC_WR = 1;
        DelayShort();
        ADC_CS = 1;

        // Tre cho chuyen doi xong
        Delay(10);
        // Chuyen Port thanh cong vao de chuan bi doc du lieu
        ADC_DATA = 0xFF;
        ADC_CS = 0;
        DelayShort();
        ADC_RD = 0;
        DelayShort();
        // Doc du lieu vao
        voltage = ADC_DATA;
        ADC_RD = 1;
        DelayShort();
        ADC_CS = 1;
}
void Calculate(void)
{
        unsigned int temp;
        temp = (voltage*10);
        temp = temp/52;
        // tinh ra dien ap tu gia tri ADC dua ve
        // tach phan nguyen va phan thap phan
        digit1 = (unsigned char) (temp/10);
        digit2 = (unsigned char)(temp%10);
        // chuyen sang ma ASCII
        digit1 = digit1 + 0x30;
        digit2 = digit2 + 0x30;
}
void DisplayVoltage(void)
```

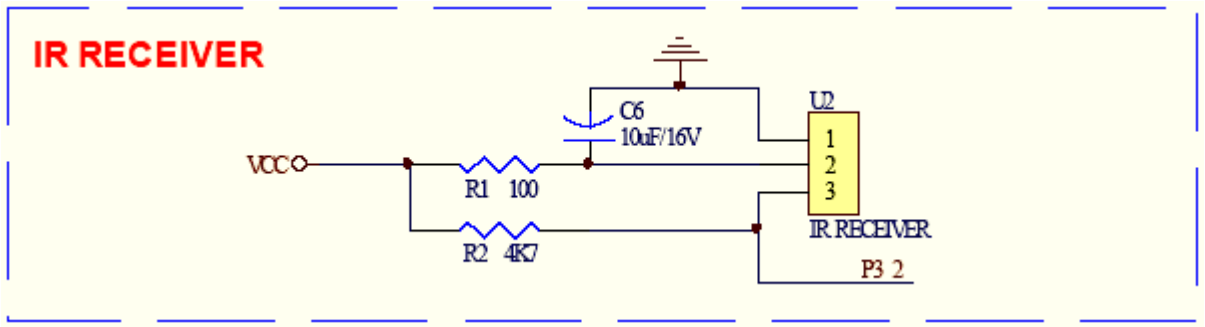
```
{
        // Dua con tro den vi tri can hien thi
        WriteCommand(0xCB);
        WriteCharacter(digit1);
        WriteCharacter('.');
        WriteCharacter(digit2);
}
// Chuong trinh chinh
void main(void)
{
        // Khoi tao he thong
        InitSystem();
        DisplayText();
        // Vong lap vo tan
        while(1)
        {
                Convert();
                Calculate();
                DisplayVoltage();
        }
}
```

Bài 8: Thiết kế hệ giao tiếp từ xa bằng hồng ngoại
Infrared Communication

Nguyên lý thiết kế



Mạch phát được thiết kế với một transistor công suất nhỏ có thể đóng cắt nguồn cấp cho LED hồng ngoại ở tần số ~36KHz. Đây là tần số được dùng trong các thiết bị thu phát hồng ngoại phổ biến trong sinh hoạt (TV, điều hòa nhiệt độ, đầu đĩa CD,…)



Mạch thu được thiết kế với mắt thu hồng ngoại sẵn có trên thị trường. Mắt thu này được thiết kế với bộ lọc thông một dải, chỉ cho phép các tần số xấp xỉ 36KHz đi qua. Do loại mắt thu này đặc biệt nhạy cảm với nhiễu nguồn nuôi nên đầu vào nguồn nuôi cho mắt được thiết kế với một mạch lọc RC thay vì nối trực tiếp lên nguồn của mạch thí nghiệm. Bình thường đầu ra của mắt thu ở mức cao, khi nhận được tín hiệu hồng ngoại ở tần số ~36KHz truyền tới, đầu ra của mắt thu sẽ lập tức xuống mức lôgic thấp (mức 0). Khi hết tín hiệu hồng ngoại, đầu ra lại trở về mức logic cao (mức 1). Nhờ vào việc đo khoảng thời gian ở mức thấp của đầu ra, người ta có thể biết được thời gian phát xung tương đương ở bên phát, từ đó suy ra mã được truyền đi. Trong ví dụ sau sẽ sử dụng tia hồng ngoại để truyền đi 3 loại mã, phân biệt nhau bởi thời gian phát đi (2ms, 4ms và 6ms). Các ngưỡng để phân biệt các mã với nhau và các mã với nhiễu là 1ms, 3ms, 5ms và 7ms.

Mã thứ nhất kéo dài 2ms tương đương với khoảng 80 xung phát đi ở tần số ~36KHz. Mã này được phát đi khi mạch bên phát nhận được tín hiệu từ một trong ba phím bấm của hàng 1.

Mã thứ hai kéo dài 4ms tương đương với khoảng 160 xung phát đi ở tần số ~36KHz. Mã này được phát đi khi mạch bên phát nhận được tín hiệu từ một trong ba phím bấm của hàng 2.

Mã thứ ba kéo dài 6ms tương đương với khoảng 240 xung phát đi ở tần số ~36KHz. Mã này được phát đi khi mạch bên phát nhận được tín hiệu từ một trong ba phím bấm của hàng 3.

Ví dụ: Thiết kế hệ thu phát tín hiệu hồng ngoại, thực hiện thu phát ba loại mã và hiển thị lên LCD loại mã thu được.

Code mẫu cho bên phát

```
// Khai bao cac file header
#include      <AT89X52.H>
```

```
#include	<Kit8051.h>

// Khai bao cac bien toan cuc

// Khai bao cac ham
void Delay(unsigned int n);
void Delay12us(void);
void InitSystem(void);
void ScanMatrix(void);
void Transmit(unsigned char n);


// Dinh nghia cac ham
void Delay(unsigned int n)
{
	unsigned int i,j;
	for(i=0;i<n;i++)
		for(j=0;j<100;j++);
}
void Delay12us(void)
{
	unsigned char i;
	i++;
	i++;
	i++;
	i++;
	i++;
	i++;
}
void InitSystem(void)
{
	// Cam LCD
	LCD_E = 0;
}
void ScanMatrix(void)
{
	// Chuyen cac hang lam dau vao, cac cot lam dau ra muc thap
	ROW_1 = 1;
	ROW_2 = 1;
```

```
        ROW_3 = 1;
        COL_1 = 0;
        COL_2 = 0;
        COL_3 = 0;
        // Kiem tra xem co phim bam?
        if((!ROW_1)|(!ROW_2)|(!ROW_3))
        {
                // Chong rung phim
                Delay(100);
                // Kiem tra lai
                // Neu la hang 1 co phim bam
                if(!ROW_1)
                {
                        Transmit(80);
                        // Doi nha phim
                        while(!ROW_1);
                }
                // Neu la hang 2 co phim bam
                if(!ROW_2)
                {
                        Transmit(160);
                        // Doi nha phim
                        while(!ROW_2);
                }
                // Neu la hang 3 co phim bam
                if(!ROW_3)
                {
                        Transmit(240);
                        // Doi nha phim
                        while(!ROW_3);
                }
        }
}
void Transmit(unsigned char n)
{
        unsigned char i;
        for(i=0;i<n;i++)
        {
                IR_LED = 0;
```

```
                Delay12us();
                IR_LED = 1;
                Delay12us();
        }
}
// Chuong trinh chinh
void main(void)
{
        // Khoi tao he thong
        InitSystem();
        // Vong lap vo tan
        while(1)
        {
                ScanMatrix();
        }
}
```

Code mẫu cho bên thu

```
// Khai bao cac file header
#include    <AT89X52.H>
#include    <Kit8051.h>

// Khai bao cac bien toan cuc
unsigned char code string1[] = "8051 Starter Kit";
unsigned char code string2[] = "Ma nhan duoc = ";

// Khai bao cac ham
void InitSystem(void);
void Delay(unsigned int n);
void InitLCD(void);
void WriteCommand(unsigned char command);
void WriteCharacter(unsigned char character);
void WriteLCD(unsigned char x);
void SendString2LCD(unsigned char code *p);
void DisplayText(void);

// Dinh nghia cac ham
void InitSystem(void)
{
        // Cam LCD
```

```
	LCD_E = 0;

	// Tre de LCD tu khoi tao ben trong (it nhat 15ms)
	Delay(100);

	InitLCD();

	// Khoi tao Timer0 o che do 16bit
	TMOD = 0x01;

	// Cau hinh ngat ngoai 0 theo suon xuong
	IT0 = 0;
	// Cho phep ngat ngoai 0
	EX0 = 1;
	EA = 1;
}
void Delay(unsigned int n)
{
	unsigned int i,j;
	for(i=0;i<n;i++)
		for(j=0;j<100;j++);
}
void InitLCD(void)
{
	WriteCommand(0x30);
	WriteCommand(0x30);
	WriteCommand(0x30);

	// 8 bit, 2 lines, font 5x7
	WriteCommand(0x38);
	// Display on, hide cursor
	WriteCommand(0x0C);

	// Xoa man hinh
	WriteCommand(0x01);
}
void DisplayText(void)
{
	// Dich con tro den vi tri thu 2, dong thu nhat
```

```
        WriteCommand(0x80);
        SendString2LCD(string1);
        // Dich con tro den dau dong thu hai
        WriteCommand(0xC0);
        SendString2LCD(string2);
}
void WriteLCD(unsigned char x)
{
        LCD_RW = 0;
        LCD_DATA = x;

        LCD_E = 1;
        LCD_E = 0;
        Delay(5);
}
void WriteCommand(unsigned char command)
{
        LCD_RS = 0;
        WriteLCD(command);
}
void WriteCharacter(unsigned char character)
{
        LCD_RS = 1;
        WriteLCD(character);
}
void SendString2LCD(unsigned char code *p)
{
        unsigned char i=0;
        while(p[i]!=0)
        {
                WriteCharacter(p[i]);
                i++;
                Delay(100);
        }
}
void ExternalInterrupt0(void) interrupt IE0_VECTOR
{
        unsigned char temp;
        // Xoa Timer 0
```

```
        TH0 = 0;
        TL0 = 0;
        // Cho Timer 0 chay
        TR0 = 1;
        // Doi cho den khi tin hieu nhan duoc ket thuc
        while(!P3_2);
        // Dung Timer 0
        TR0 = 0;
        // So sanh gia tri cua Timer voi cac nguong de xac dinh ma nhan duoc
        temp = TH0;
        if((temp>THRESHOLD_1)&&(temp<THRESHOLD_2))
        {
                // Dua con tro den vi tri mong muon
                WriteCommand(0xCF);
                // Hien thi ma nhan duoc
                WriteCharacter('1');
        }
        else if((temp>THRESHOLD_2)&&(temp<THRESHOLD_3))
        {
                // Dua con tro den vi tri mong muon
                WriteCommand(0xCF);
                // Hien thi ma nhan duoc
                WriteCharacter('2');
        }
        else if((temp>THRESHOLD_3)&&(temp<THRESHOLD_4))
        {
                // Dua con tro den vi tri mong muon
                WriteCommand(0xCF);
                // Hien thi ma nhan duoc
                WriteCharacter('3');
        }
}
// Chuong trinh chinh
void main(void)
{
        // Khoi tao he thong
        InitSystem();
        DisplayText();
        // Vong lap vo tan
```

```
        while(1);
}
```

File header cho toàn bộ các chương trình mẫu ở trên bao gồm file "AT89X52.H" có sẵn của trình biên dịch Keil. Ngoài ra còn có thêm một file header khác tên là "Kit8051.h" do người sử dụng tự tạo ra nhằm định nghĩa các hằng số, các tên gọi khác của các tín hiệu giao tiếp, điều khiển. Chi tiết xin tham khảo trong đĩa CD kèm theo.