

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

-----oOo-----

Giáo trình
VI ĐIỀU KHIỄN

Biên soạn: TS. Vũ Đức Lung
 TS. Lê Quang Minh
 ThS. Phan Đình Duy

2015

Lời nói đầu

Ngày nay, với sự tiến bộ của khoa học kỹ thuật, con người đã tìm ra nhiều giải pháp để giải phóng sức lao động và nâng cao đời sống vật chất lẫn tinh thần. Một trong những giải pháp đó là công nghệ tự động hóa dựa trên các ứng dụng kỹ thuật điện, kỹ thuật điều khiển vào trong máy móc và các thiết bị phục vụ đời sống như tivi, tủ lạnh, máy giặt, ô oto... Trong các thiết bị thông minh đó một thành phần không thể thiếu chính là vi điều khiển.

Vi điều khiển ra đời từ năm 1976 do hãng Intel phát triển, cho đến nay nó đã trở nên vô cùng phổ biến trong các thiết bị máy móc với rất nhiều chủng loại và được nhiều công ty khác nhau trên thế giới sản xuất. Nổi bật trong đó có vi điều khiển họ 8051 của Intel, vi điều khiển ARM của công ty cùng tên ARM, vi điều khiển PIC của Microchip, vi điều khiển AVR của Atmel. Vi điều khiển có thể được xem như là một máy tính nhỏ gọn bao gồm vi xử lý, các thành phần bộ nhớ (ROM, RAM), các thiết bị ngoại vi khác nhằm hỗ trợ cho vi điều khiển giao tiếp với các thiết bị trong việc nhận tín hiệu và xuất các tín hiệu điều khiển.

Xác định được tầm quan trọng của Vi điều khiển và nhu cầu của thị trường về lĩnh vực này rất lớn nên hầu hết các chương trình đào tạo lĩnh vực Điện tử - Kỹ thuật máy tính đều đưa nội dung vi điều khiển vào giảng dạy. Cuốn “Giáo trình vi điều khiển” này nhằm giúp cho sinh viên tìm hiểu cấu trúc bên trong họ vi điều khiển cơ bản nhất đó là họ vi điều khiển 8051, biết cách vi điều khiển hoạt động và ứng dụng họ vi điều khiển này để thực hiện một số sản phẩm điều khiển trong thực tế. Theo đó, nội dung cuốn sách này được chia làm 9 chương trình bày từ kiến thức cơ bản về vi điều khiển đến cấu trúc chi tiết một vi điều khiển 8051 và cuối cùng là các ví dụ minh họa và ứng dụng của họ vi điều khiển 8051 trong thực tế. Thêm vào đó cuốn sách

cũng dành một chương để đề cập đến các loại vi điều khiển khác tiên tiến hơn được sử dụng phổ biến hiện nay để sinh viên có cái nhìn tổng quan và có thể lựa chọn được loại vi điều khiển phù hợp cho từng ứng dụng. Cụ thể nội dung các chương như sau:

Chương 1: Trình bày sơ lược về lịch sử ra đời về các khái niệm vi xử lý - vi điều khiển, phân loại vi điều khiển.

Chương 2: Chương này sẽ đi sâu trình bày về tính năng, sơ đồ bố trí chân cũng như cấu trúc bên trong của họ vi điều khiển 8051. Cấu trúc bộ nhớ và các thanh ghi trong vi điều khiển 8051 là nội dung quan trọng của chương này, nó rất cần phải hiểu rõ để có thể tiếp tục tìm hiểu các chương tiếp theo trong sách này.

Chương 3: Sẽ đi vào cấu trúc tập lệnh của họ vi điều khiển 8051. Các nhóm lệnh, các lệnh gọi nhớ và chức năng của từng lệnh sẽ được mô tả cụ thể trong chương này. Chương này cũng trình bày các chế độ định địa chỉ trong quá trình lập trình cho vi điều khiển.

Chương 4: Bộ định thời/bộ đếm là một thành phần quan trọng không thể thiếu trong các vi điều khiển sẽ được thể hiện trong chương này. Các nội dung chính của chương trình bày về cách thức hoạt động của bộ định thời/bộ đếm, các mode hoạt động và các thanh ghi được sử dụng cho việc định thời/đếm.

Chương 5: Giới thiệu về giao tiếp nối tiếp, các thanh ghi sử dụng trong giao tiếp nối tiếp và các mode hoạt động của nó.

Chương 6: Trình bày về một thành phần cũng rất quan trọng khác trong họ vi điều khiển 8051 cũng như các vi điều khiển khác đó là ngắt (interrupt). Chương này sẽ giới thiệu các ngắt trong họ vi điều khiển 8051, các thanh ghi được sử dụng cho ngắt, chương trình phụ vụ ngắt và cách sử dụng ngắt trong lập trình.

Chương 7: Sẽ hướng dẫn cách lập trình vi điều khiển 8051 giao tiếp với một số thiết bị cơ bản như LED đơn, LED 7 đoạn, text LCD, motor, bộ nhớ ngoài và một số giao tiếp khác.

Chương 8: Mô tả một số ứng dụng đơn giản của vi điều khiển 8051 vào thực tế cuộc sống giúp cho sinh viên hình dung đầy đủ về việc phát triển một ứng dụng sử dụng vi điều khiển

Chương 9: Trình bày về một số vi điều khiển hiện đại được sử dụng rộng rãi ở Việt Nam hiện nay như AVR, PIC, ARM. Chương này sẽ giới thiệu sơ qua về tính năng, một số đặc điểm và ví dụ về việc sử dụng các vi điều khiển thông dụng hiện nay.

MỤC LỤC

Chương 1. Giới thiệu chung về Vi điều khiển	1
1.1 Tổng quan	1
1.2 Vi xử lý	2
1.3 Vi điều khiển	2
1.4 Các loại vi điều khiển	3
1.4.1 Dựa theo độ dài thanh ghi.....	3
1.4.2 Dựa theo kiến trúc CISC và RISC.....	3
1.4.3 Kiến trúc Harvard và kiến trúc Vonneumann	4
1.5 Các kiến thức đặc trưng của vi điều khiển	4
1.5.1 UART	6
1.5.2 SPI	7
1.5.3 ADC	8
1.5.4 I2C	9
1.6 Tổng kết	12
1.7 Bài tập	12
Chương 2. Kiến trúc họ Vi điều khiển 8051	14
Mục tiêu chương.....	14

2.1	Tính năng bên ngoài	14
2.2	Chân và tín hiệu	15
2.2.1	Port 0	16
2.2.2	Port 1	18
2.2.3	Port 2	19
2.2.4	Port 3	19
2.2.5	Các chân nguồn	20
2.2.6	Chân cho phép bộ nhớ chương trình PSEN.....	20
2.2.7	Chân cho phép chốt địa chỉ ALE.....	20
2.2.8	Chân truy xuất bộ nhớ ngoài EA	21
2.2.9	Chân Reset	21
2.2.10	Các chân vào bộ dao động trên chip.....	22
2.3	Kiến trúc bên trong	23
2.4	Tổ chức bộ nhớ	24
2.4.1	Tổ chức bộ nhớ trong	25
2.4.2	Tổ chức bộ nhớ ngoài	37
2.5	Tổng kết	41
2.6	Bài tập	42

Chương 3. Tập lệnh họ Vị điều khiển 8051	43
Mục tiêu chương.....	43
3.1 Tổng quan	43
3.2 Các chế độ định vị địa chỉ họ VĐK 8051.....	43
3.2.1 Địa chỉ tức thời	44
3.2.2 Địa chỉ theo thanh ghi.....	44
3.2.3 Địa chỉ trực tiếp	45
3.2.4 Địa chỉ gián tiếp.....	46
3.2.5 Địa chỉ chỉ số	46
3.3 Các lệnh di chuyển dữ liệu	47
3.4 Các lệnh toán học	49
3.5 Các lệnh rẽ nhánh	51
3.6 Các lệnh xử lý theo bit.....	53
3.7 Các lệnh logic	55
3.8 Ví dụ	56
3.9 Tổng kết	57
3.10 Bài tập	57
Chương 4. BỘ ĐỊNH THỜI/ BỘ ĐÊM	59

4.1	Giới thiệu	59
4.2	Bộ định thời 8051/8052	60
4.3	Thanh ghi chức năng đặc biệt cho bộ định thời	61
4.3.1	Thanh ghi điều khiển Timer (TCON).....	61
4.3.2	Thanh ghi chế độ định thời (TMOD)	62
4.3.3	Thanh ghi của bộ định thời 0.....	63
4.3.4	Thanh ghi của bộ định thời 1	64
4.4	Các chế độ định thời	64
4.4.1	Chế độ 0	64
4.4.2	Chế độ 1	65
4.4.3	Chế độ 2	66
4.4.4	Chế độ 3	67
4.5	Lập trình sử dụng các bộ định thời	68
4.5.1	Lập trình chế độ 1	68
4.5.2	Lập trình chế độ 2	71
4.6	Ví dụ	73
4.7	Tổng kết	76
4.8	Bài tập	76

Chương 5. GIAO TIẾP NỐI TIẾP (UART)	78
5.1 Giới thiệu	78
5.2 Thanh ghi cho giao tiếp nối tiếp	79
5.2.1 Thanh ghi bộ đệm cổng nối tiếp SBUF	80
5.2.2 Thanh ghi điều khiển cổng nối tiếp SCON	80
5.3 Các chế độ hoạt động của cổng nối tiếp	81
5.3.1 Chế độ 0	82
5.3.2 Chế độ 1	84
5.3.3 Chế độ 2	86
5.3.4 Chế độ 3	86
5.4 Tốc độ baud cổng nối tiếp	86
5.5 Ví dụ	89
5.6 Tổng kết	93
5.7 Bài tập	93
Chương 6. HOẠT ĐỘNG NGẮT	95
6.1 Giới thiệu	95
6.2 Ngắt của 8051	96
6.3 Thanh ghi đặc biệt cho ngắt.....	97

6.3.1	Thanh ghi cho phép ngắn	97
6.3.2	Thanh ghi ưu tiên ngắn	99
6.3.3	Thanh ghi chứa cờ ngắn	100
6.4	Chương trình phục vụ ngắn	100
6.4.1	Ngắn reset	103
6.4.2	Ngắn bộ định thời	103
6.4.3	Ngắn ngoài	104
6.4.4	Ngắn cổng nối tiếp	107
6.5	Mức ưu tiên ngắn của vi điều khiển 8051	109
6.6	Tổng kết	110
6.7	Bài tập	110
	Chương 7. Lập trình Assembly giao tiếp với 8051	112
7.1	Cấu trúc tổng quát chương trình	112
7.2	Giao tiếp với thiết bị hiển thị	114
7.2.1	Giao tiếp với LED đơn	114
7.2.2	Giao tiếp với LED 7 đoạn	116
7.3	Giao tiếp với bàn phím 4x4	120
7.4	Giao tiếp ADC/DAC	124

7.4.1	Giao tiếp ADC	125
7.4.2	Giao tiếp DAC	131
7.5	Giao tiếp với motor.....	134
7.6	Giao tiếp với bộ nhớ ngoài (RAM)	138
7.7	Tổng kết	141
7.8	Câu hỏi và bài tập	141
Chương 8. Ứng dụng của họ Vi điều khiển 8051		143
8.1	Ứng dụng điều khiển bảng quảng cáo	143
8.1.1	Phát biểu ứng dụng	143
8.1.2	Thiết kế mạch mô phỏng	143
8.1.3	Chương trình nhúng.....	149
8.2	Ứng dụng điều khiển giao thông ngã 4	154
8.2.1	Phát biểu ứng dụng	154
8.2.2	Thiết kế mạch mô phỏng	154
8.2.3	Chương trình nhúng.....	157
8.3	Ứng dụng điều khiển xe robot	164
8.3.1	Phát biểu ứng dụng	164
8.3.2	Thiết kế mạch mô phỏng	164

8.3.3	Chương trình nhúng.....	165
8.4	Câu hỏi và bài tập chương	171
Chương 9.	Các họ vi điều khiển hiện đại	172
Mục đích chương.....		172
9.1	Họ vi điều khiển AVR.....	172
9.1.1	Tổng quan về AVR.....	172
9.1.2	Đặc trưng phần cứng AVR	173
9.1.3	Một số giao tiếp ngoại vi đặc trưng của AVR	176
a.	BUS I2C	177
b.	Giao tiếp SPI	178
9.1.4	Tập lệnh của AVR	180
9.1.5	Giới thiệu vi điều khiển AVR	181
9.2	Họ vi điều khiển PIC	193
9.2.1	Tổng quan về PIC	193
9.2.2	Đặc trưng phần cứng của vi điều khiển PIC...	194
9.2.3	Tập lệnh của PIC	197
9.2.4	Cơ bản về dsPIC	201
9.2.5	Một số đặc tính cơ bản của dsPIC	201

9.2.6	Ví dụ ứng dụng vi điều khiển PIC	207
9.3	Họ vi điều khiển ARM	214
9.3.1	Tổng quan về ARM	214
9.3.2	Một số đặc trưng phần cứng ARM	215
9.3.3	Tập lệnh ARM	216
9.3.4	Giới thiệu bộ xử lý ARM Cortex-M3.....	219
9.3.5	Ví dụ ứng dụng ARM.....	226
9.4	Câu hỏi và bài tập	229

DANH MỤC HÌNH ẢNH

Hình 1-1 Sơ đồ các khối trong CPU	5
Hình 1-2 Kết nối trong giao tiếp SPI	7
Hình 1-3 Sơ đồ kết nối I2C	10
Hình 1-4 Dữ liệu truyền trên bus I2C theo từng bit....	11
Hình 1-5 Truyền dữ liệu I2C.....	12
Hình 2-1 Sơ đồ các chân của vi điều khiển 8051	16
Hình 2-2 Cấu tạo bên trong port 0.....	17
Hình 2-3 Điện trở kéo lên tại port 0.....	17
Hình 2-4 Cấu tạo bên trong port 1	18
Hình 2-5 Các lắp mạch reset.....	22
Hình 2-6 Cách lắp thạch anh ngoài	22
Hình 2-7 Kiến trúc bên trong vi điều khiển 8051	23
Hình 2-8 Tổ chức bộ nhớ của vi điều khiển 8051	24
Hình 2-9 Cấu trúc địa chỉ bên trong RAM nội	26
Hình 2-10 Sơ đồ nguyên lý giao tiếp bộ nhớ chương trình ngoài	38

Hình 2-11 Giản đồ thời gian giao tiếp với bộ nhớ chương trình ngoài.....	39
Hình 2-12 Sơ đồ nguyên lý giao tiếp bộ nhớ dữ liệu ngoài.....	40
Hình 2-13 Giản đồ thời gian giao tiếp bộ nhớ dữ liệu ngoài.....	40
Hình 2-14 Sơ đồ nguyên lý giao tiếp bộ nhớ dùng chung ngoài	41
Hình 3-1 Cấu trúc của lệnh định vị địa chỉ tức thời	44
Hình 3-2 Cấu trúc của lệnh định vị địa chỉ thanh ghi .	45
Hình 3-3 Cấu trúc của lệnh định vị địa chỉ trực tiếp...	45
Hình 4-1 Hoạt động của timer 3 bit.....	60
Hình 4-2 Thanh ghi bộ định thời 0	64
Hình 4-3 Thanh ghi bộ định thời 1	64
Hình 4-4 Mô hình chế độ 0 của bộ định thời	65
Hình 4-5 Mô hình chế độ 1 của bộ định thời	66
Hình 4-6 Mô hình chế độ 2 của bộ định thời	67
Hình 4-7 Mô hình chế độ 3 của bộ định thời	68
Hình 5-1 Truyền dữ liệu song song và nối tiếp.....	79

Hình 5-2 Hoạt động UART chế độ 0	83
Hình 5-3 Ứng dụng của chế độ 0	84
Hình 5-4 Hoạt động UART chế độ 1	85
Hình 5-5 Bộ chia tốc độ baud chế độ 0	87
Hình 5-6 Bộ chia tốc độ baud chế độ 1 và 3	87
Hình 6-1 Chuyển đổi trạng thái chương trình khi có ngắt	96
Hình 6-2 Sơ đồ các ngắt trong 8051	97
Hình 6-3 Cấu tạo ngắt ngoài	105
Hình 6-4 Sơ đồ nối công tắc và LED	106
Hình 6-5 Cờ ngắt cổng nối tiếp	107
Hình 7-1 Sơ đồ giao tiếp với LED đơn	115
Hình 7-2 Sơ đồ giao tiếp với 3 LED 7 đoạn	117
Hình 7-3 Sơ đồ giao tiếp với bàn phím 4x4	120
Hình 7-4 Sơ đồ giao tiếp ADC	126
Hình 7-5 Nguyên lý chuyển đổi ADC	127
Hình 7-6 Sơ đồ giao tiếp DAC	131
Hình 7-7 Nguyên lý chuyển đổi DAC	132
Hình 7-8 Nguyên lý băm xung	135

Hình 7-9 Sơ đồ mô phỏng điều khiển motor.....	136
Hình 7-10 Sơ đồ giao tiếp với RAM ngoài	139
Hình 7-11 Mô phỏng giao tiếp với RAM ngoài.....	140
Hình 8-1 Sơ đồ thiết kế giải pháp điều khiển ma trận LED	144
Hình 8-2 Sơ đồ chân IC 79HC595	145
Hình 8-3 Sơ đồ chân IC 74HC138	146
Hình 8-4 Sơ đồ kết nối IC 74HC138	147
Hình 8-5 Mạch mô phỏng ma trận LED 8x32 trên Proteus	148
Hình 8-6 Hiển thị chữ "T" trên ma trận led 8x8.....	149
Hình 8-7 Lưu đồ giải thuật điều khiển LED ma trận	150
Hình 8-8 Sơ đồ thiết kế giải pháp điều khiển đèn giao thông trên proteus	155
Hình 8-9 Sơ đồ mô phỏng hiển thị các đèn giao thông cùng thời gian chuyển đổi giữa các đèn	156
Hình 8-10 Lưu đồ giải thuật điều khiển đèn giao thông	158

Hình 8-11 Sơ đồ mạch mô phỏng điều khiển 2 động cơ	165
Hình 8-12 Lưu đồ giải thuật điều khiển robot	166
Hình 9-1 Cấu trúc nhân CPU của AVR	174
Hình 9-2 Ví dụ sử dụng I2C có 2 vi điều khiển kết nối	177
Hình 9-3 Truyền dữ liệu trên BUS I2C	178
Hình 9-4 Cấu hình SPI chế độ Multi Master	179
Hình 9-5 Cấu hình SPI chế độ Master/ Slaver.....	180
Hình 9-6 Kết nối AVR với bộ nhớ ngoài.....	184
Hình 9-7 Sơ đồ mạch mở rộng sử dụng 8255A	185
Hình 9-8 Sơ đồ mạch ứng dụng đo nhiệt độ	188
Hình 9-9 Cấu trúc nhân CPU dòng PIC 16	195
Hình 9-10 Cấu trúc lõi CPU.....	202
Hình 9-11 Tổ chức bộ nhớ DSPIC	204
Hình 9-12 Sơ đồ mạch máy tính đơn giản dùng PIC 18F452	208
Hình 9-13 Cortex – M3	222
Hình 9-14 Cortex-M3 bit-banding.....	223

Hình 9-15 Sơ đồ đồng hồ thời gian thực sử dụng ARM7	226
--	-----

DANH MỤC BẢNG BIỂU

Bảng 2-1 Đặc trưng các vi điều khiển họ MCS - 51 ..	15
Bảng 2-2 Chức năng thứ 2 của các chân tại port 3.....	19
Bảng 2-3 Giá trị các thanh ghi khi reset.....	21
Bảng 2-4 Lựa chọn các bank thanh ghi	27
Bảng 2-5 Các thanh ghi có chức năng đặt biệt.....	29
Bảng 2-6 Chức năng của thanh ghi trạng thái chương trình.....	30
Bảng 2-7 Chức năng các bit trong thanh ghi PCON ..	36
Bảng 3-1 Các lệnh di chuyển dữ liệu	47
Bảng 3-2 Cách lệnh số học.....	50
Bảng 3-3 Các câu lệnh rẽ nhánh.....	52
Bảng 3-4 Các lệnh xử lý theo bit.....	54
Bảng 3-5 Các lệnh logic	55
Bảng 4-1 Các thanh ghi sử dụng trong việc định thời của 8051	61
Bảng 4-2 Các bit chức năng trong thanh ghi điều khiển TCON	61

Bảng 4-3 Các bit chức năng trong thanh ghi điều khiển TMOD	62
Bảng 4-4 Các mode định thời trong 8051	63
Bảng 5-1 Các bit chức năng trong thanh ghi SCON ..	80
Bảng 5-2 Các chế độ hoạt động của cổng nối tiếp	82
Bảng 5-3 Các giá trị nạp vào TH1 cho tốc độ baud thông dụng	89
Bảng 6-1 Các bit chức năng của thanh ghi IE	98
Bảng 6-2 Các bit chức năng của thanh ghi IP	99
Bảng 6-3 Các cờ ngắt	100
Bảng 6-4 Bảng vector ngắt	101
Bảng 6-5 Độ ưu tiên các ngắt	110
Bảng 7-1 Số toán hạng trong câu lệnh	113

DANH SÁCH TỪ VIẾT TẮT

ACK	Acknowledge
ALU	Arithmetic-Logic Unit
AMR	Acorn RISC Machine
AVR	Advanced Virtual RISC
CISC	Complex Instruction Set Computer
CPU	Central Processing Unit
DCI	Data Converter Interface
I2C	Inter-Integrated Circuit
IC	Integrated Circuit
IE	Interrupt Enable
IP	Interrupt Priority
LED	Light Emitting Diode
MCU	Micro Controller Unit
NMI	Non-Maskable Interrupt
NVIC	Nested Vectored Interrupt Controller
PIC	Programmable Intelligent Computer
RB	Receiver bit
REN	Reception Enable
RI	Receive Interrupt
RISC	Reduced Instruction Set Computer
SCI	Serial Clock
SDA	Serial Data
SOC	Systems On Chip
SPI	Serial Peripheral Interface
TB	Transmitter bit
TI	Texas Instruments
VĐK	Vị điều khiển

Chương 1. Giới thiệu chung về Vi điều khiển

1.1 Tổng quan

Vi điều khiển (VĐK) đầu tiên được thương mại vào năm 1974 bởi công ty Texas Instruments (TI) với tên gọi TMS 1000 và chứa bộ nhớ chỉ đọc (ROM), bộ nhớ đọc/ghi (RAM), bộ xử lý và bộ định thời. Tiếp đó Intel cho ra đời VĐK Intel 8748, được thương mại rộng rãi vào năm 1977. Đây là một vi điều khiển trong họ vi điều khiển đầu tiên MCS-48 (bao gồm VĐK 8048, 8035 và 8748) của Intel, nó chứa hơn 17000 transistor gồm 1 vi xử lý, 1KB EPROM, 64 byte RAM, 27 chân nhập xuất và một bộ định thời 8 bit. Vi điều khiển 8748 và các vi điều khiển khác trong họ MCS-48 nhanh chóng trở nên phổ biến trong các ứng dụng điều khiển thời bấy giờ như điều khiển các thiết bị gia dụng và công nghiệp.

Tới năm 1980, Intel tiếp tục cho ra đời bộ vi điều khiển 8051 với kiến trúc, kích thước bộ nhớ và tính năng tăng lên rất nhiều so với vi điều khiển đời đầu. Vi điều khiển 8051 chứa hơn 60000 transistor gồm 4KB ROM, 128 byte RAM, 32 chân nhập xuất, 1 cổng truyền nối tiếp và 2 bộ định thời 16 bit, và đây là vi điều khiển đầu tiên của họ vi điều khiển MCS-51 của hãng Intel. Từ đây các vi điều khiển khác trong họ MCS-51 lần lượt ra đời với sự thay đổi về các thành phần bên trong kiến trúc như thay đổi dung lượng RAM, ROM, thay đổi số lượng bộ định thời ... Và tiếp đó là sự ra đời của các vi điều khiển của các hãng khác dựa trên kiến trúc của vi điều khiển 8051 làm cho các vi điều khiển thuộc họ 8051 được sử dụng vô cùng phổ biến trong các ứng dụng từ ngày nay ra đời cho đến tận ngày nay.

1.2 Vi xử lý

Vi xử lý là một thiết bị xử lý thông tin như các phép toán cộng trừ, nhân chia, dịch bit, quay ... Nó bao gồm các khối chức năng chính để thu thập, xử lý và xuất dữ liệu ra các khối phần cứng khác. Vi xử lý được sử dụng rất nhiều trong các hệ thống máy tính ngày nay, nó không thể giao tiếp với các thiết bị ngoại vi bên ngoài, và cũng chưa được kết nối với các thành phần bộ nhớ khác. Muốn dùng được vi xử lý thì phải thiết kế các mạch điện để kết nối nó với bộ nhớ RAM, ROM và các thiết bị ngoại vi khác.

1.3 Vi điều khiển

Vi điều khiển là một máy tính thu nhỏ, nó bao gồm 1 vi xử lý đơn giản bên trong kết hợp với các bộ phận khác như RAM, ROM và một số ngoại vi cần thiết. Tùy theo hãng sản xuất và mục đích sử dụng vi điều khiển có thể có nhiều dạng khác nhau về cấu trúc các thành phần cũng như số lượng và chức năng của các chân điều khiển. Nhưng nhìn chung cấu tạo của một vi điều khiển sẽ bao gồm các thành phần sau:

- Một bộ vi xử lý đơn giản với khả năng tính toán phù hợp cho việc xử lý tín hiệu đơn giản và điều khiển thiết bị.
- Bộ nhớ để lưu dữ liệu và chương trình (một vài vi điều khiển không chứa bộ nhớ bên trong, cần kết nối với bộ nhớ ngoài).
- Có các chân nhập/xuất tín hiệu, các bộ đếm, bộ định thời, bộ chuyển đổi tương tự - số,...
- Thường được tích hợp trong một vỏ IC tiêu chuẩn.

1.4 Các loại vi điều khiển

Ngày nay với việc phát triển mạnh mẽ của các dòng vi điều khiển, các hãng sản xuất nổi tiếng đã đưa ra thị trường các dòng vi điều khiển với nhiều tính năng khác nhau như: Tốc độ xử lý, tiết kiệm năng lượng, giá thành thấp... dựa vào nhu cầu sử dụng mà người dùng có thể chọn cho mình những dòng vi điều khiển mình cần như: ARM, INTEL, AVR, PIC, ... Và việc phân chia nhóm các loại vi điều khiển được phân chia như sau:

1.4.1 Dựa theo độ dài thanh ghi

Dựa vào độ dài của thanh ghi và các lệnh điều khiển của VĐK mà người ta chia ra các loại vi điều khiển 8bit, 16bit, 32bit... Các loại VĐK có độ dài lệnh lớn hơn thì tập lệnh nhiều hơn, xử lý chương trình được nhanh hơn. Một số dòng vi điều khiển dựa theo độ dài thanh ghi của các hãng sản xuất như:

- 8bit: 8051 family, Attiny13A (Atmel), STM8S003F3 (Stmicroelectronics), LPC7000 (NXP Semiconductor),...
- 16bit: Atmega16(Atmel), MSC-96 (Intel), PIC16F84A (Microchip), MSP430 (TI),...
- 32bit: Atmega32 (Atmel), ARM Cortex M4,...

1.4.2 Dựa theo kiến trúc CISC và RISC

VĐK CISC (Complex Instruction Set Computer) là dòng VĐK sử dụng bộ xử lý có tập lệnh phức tạp. Các VĐK này có một số lượng lớn lệnh nên giúp người lập trình có thể linh hoạt và dễ dàng trong khi viết chương trình. Một số dòng VĐK loại này như: Motorola 68xxx, Intel 8051,...

VĐK RISC (Reduced Instruction Set Computer) là dòng VĐK sử dụng bộ xử lý có tập lệnh khá đơn giản. Các VĐK này có một số lượng nhỏ các lệnh đơn giản. Vì vậy mà các dòng này đòi hỏi kiến trúc phần cứng ít hơn và không phức tạp, giá thành thấp hơn và nhanh hơn CISC, tuy nhiên đòi hỏi người lập trình phải viết các chương trình phức tạp hơn, nhiều lệnh hơn với cùng một bài toán. Các dòng VĐK loại này thường dựa theo kiến trúc ARM (STM32F4) và hầu hết thiết kế theo kiến trúc RISC Atmel AVR (Atmega8).

1.4.3 Kiến trúc Harvard và kiến trúc VonNeumann

Kiến trúc Harvard là kiến trúc sử dụng bộ nhớ riêng biệt cho chương trình và dữ liệu, bus địa chỉ và bus dữ liệu cũng độc lập với nhau nên quá trình truyền nhận dữ liệu và lệnh đơn giản hơn và nhanh hơn. VD: ARM7-LPC210x, Atmega128,...

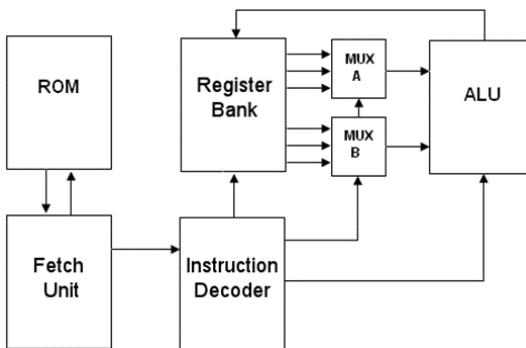
Kiến trúc VonNeumann sử dụng chung bộ nhớ chương trình và dữ liệu, việc truy xuất dữ liệu và tập lệnh phức tạp hơn, tuy nhiên đòi hỏi phần cứng ít hơn và giá thành rẻ hơn. VD: Motorola 68xxx...

1.5 Các kiến thức đặc trưng của vi điều khiển

Thông thường một vi điều khiển sẽ bao gồm các bộ phận chính như sau:

- CPU (Bộ xử lý trung tâm – Central Processing Unit): Là trung tâm của hệ thống, là nơi quản lý tất cả các hoạt động của VĐK. Bên trong CPU gồm:
 - + ALU (Arithmetic-Logic Unit): bộ tính toán số học thao tác trên các bộ dữ liệu được đưa vào.

- + Bộ giải mã lệnh và điều khiển: bộ phận này xác định được CPU cần làm gì và thực hiện thao tác gì với lệnh cần thực hiện.
- + Thanh ghi lệnh: lưu giữ các lệnh được tải từ RAM vào, đưa vào bộ giải mã lệnh để CPU thực thi lệnh đó.
- + Thanh ghi PC: lưu giữ địa chỉ của lệnh kế tiếp cần thực thi.
- + Bộ thanh ghi để lưu trữ dữ liệu tạm thời cho việc tính toán của ALU.



Hình 1-1 Sơ đồ các khối trong CPU

- ROM: Là bộ nhớ dùng để lưu trữ chương trình. Ngoài ra ROM còn dùng để chứa các tham số của hệ thống, các số liệu cố định của hệ thống. Trong quá trình chạy ROM không thay đổi, để thay đổi nội dung của ROM cần phải thiết lập VDK chế độ nạp chương trình hoặc xóa chương trình.
- RAM là bộ nhớ dữ liệu chính, là vùng nhớ trung gian giữa bộ nhớ phụ với CPU, là nơi lưu trữ các lệnh và dữ liệu tạm thời. Dữ liệu trên RAM sẽ mất hết khi mất nguồn hay khởi động lại hệ thống.

- BUS: Là các đường dẫn để di chuyển dữ liệu giữa các thành phần trong VĐK với nhau, gồm một số loại như: Bus địa chỉ, Bus dữ liệu, Bus điều khiển.
- Timer/Counter: Là module ngoại vi độc lập với CPU, chức năng chính của timer là định thời tạo ra khoảng thời gian, đếm thời gian, tạo ra các ngắt theo thời gian. Tùy thuộc vào từng loại VĐK mà độ dài các thanh ghi trong timer khác nhau.
- Watchdog: Là module dùng để reset lại hệ thống khi hệ thống gặp sự cố.
- ADC: Là module chuyển đổi tín hiệu từ Analog từ môi trường sang tín hiệu Digital mà VĐK có thể hiểu và xử lý được.

Bên cạnh các bộ phận chính trong một vi điều khiển, tùy theo chức năng và mục đích sử dụng mà nhà sản xuất có thể thêm các module giao tiếp cơ bản sau vào vi điều khiển. Các module này trình bày tiếp sau đây trong mục này.

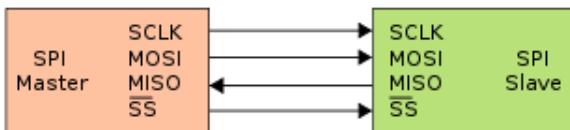
1.5.1 UART

UART (Universal Asynchronous serial Receiver and Transmitter) là bộ truyền nhận nối tiếp không đồng bộ. Là chuẩn giao tiếp giữa các thiết bị để trao đổi dữ liệu với nhau, dữ liệu sẽ được truyền từng bit trên 1 đường truyền, vì vậy mà dữ liệu dù lớn tới mấy thì cũng cần ít nhất một đường truyền. Và việc thiết lập chuẩn giao tiếp qua lại giữa 2 thiết bị với nhau thì cần 3 đường truyền: TX, RX, GND. UART khung dữ liệu đã được chuẩn hóa với định dạng cố định và 2 thiết bị ngầm định rằng cứ 1ms sẽ có 1bit truyền tới và kết hợp các bit lại với nhau thành dãy dữ liệu đúng, tuy nhiên cần phải quan tâm tới một số khái niệm sau:

- Baudrate: Là số bit truyền trong 1s được thiết lập và qui định giữa 2 thiết bị giao tiếp với nhau. VD: 9600, 115200,...
- Frame : Khung truyền qui định số bit trong mỗi lần truyền và các bit báo hiệu bắt đầu và kết thúc của một đoạn dữ liệu nào đó.
- Start bit: là bit đầu tiên được truyền trong một frame, báo cho thiết bị biết có 1 khung dữ liệu mới sắp truyền.
- Data: dữ liệu chính cần truyền đi.
- Parity bit: là bit kiểm tra xem dữ liệu truyền có đúng hay không.
- Stop bit: Là bit báo cho thiết bị nhận biết rằng gói dữ liệu gửi đã xong.

1.5.2 SPI

SPI (Serial Peripheral Bus) là chuẩn truyền thông nối tiếp đồng bộ tốc độ cao. Đây là kiểu Master và Slave trong đó một thiết bị đóng vai trò là Master điều phối quá trình truyền thông và các Slave được điều khiển bởi Master. SPI là chuẩn truyền dữ liệu song công, nghĩa là một lúc có thể thực hiện vừa truyền và nhận dữ liệu. SPI chuẩn giao tiếp thông qua 4 đường.



Hình 1-2 Kết nối trong giao tiếp SPI

- SCLK: xung nhịp cho giao tiếp SPI, mỗi xung nhịp biết được 1bit đến hay đi. SCLK giúp cho việc truyền dữ liệu đồng bộ và ít bị lỗi hơn và thực hiện tốc độ cao.
- MISO: Nếu thiết bị là Master thì đường này là input, còn nếu là Slave thì đường này là Output.
- MOSI: Nếu thiết bị là Master thì đường này là output, còn nếu là Slave thì đường này là Input.
- SS: là đường chọn Slave cần giao tiếp với Master.

1.5.3 ADC

Là module ngoại vi độc lập với CPU. Giúp chuyển đổi các tín hiệu Analog như: Nhiệt độ, độ ẩm,... sang tín hiệu Digital mà VDK có thể hiểu được, quá trình này gọi là “số hóa”.

Chuyển đổi ADC theo phương pháp flash ADC, bộ chuyển đổi được cấu thành từ dãy các bộ số so sánh, các bộ so sánh được kết nối trực tiếp với tín hiệu analog. Điện áp tham chiếu và mạch chia áp được sử dụng để tạo ra các mức điện áp khác nhau cho mỗi bộ so sánh. Các khái niệm liên quan khi làm việc với ADC:

- Độ phân giải: Là số bit cần thiết để chứa hết các mức giá trị digital ngõ ra. Trong trường hợp có 8 mức giá trị ngõ ra thì cần 3 bit để mã hóa hết các giá trị này. Tổng quát hóa thì nếu ADC có độ phân giải n bit thì cần tới $2^n - 1$ bộ so sánh điện áp. Độ phân giải liên quan tới chất lượng chuyển đổi và độ chính xác của tín hiệu analog vào.
- Điện áp tham chiếu: Điện áp tham chiếu thường là giá trị điện áp lớn nhất mà bộ ADC có thể chuyển đổi. Trong các bộ ADC, Vref thường là thông số

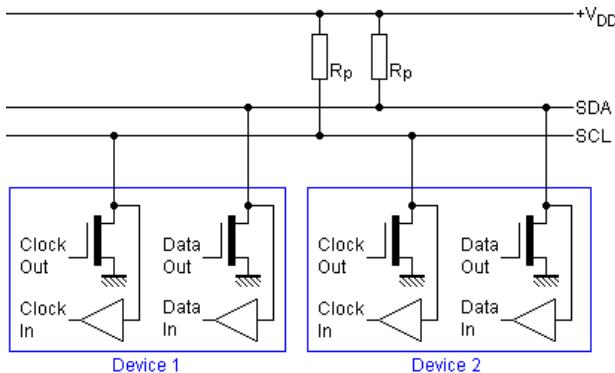
được đặt bởi người dùng, nó là điện áp lớn nhất mà thiết bị có thể chuyển đổi. Ví dụ, một bộ ADC 10 bit (độ phân giải) có $V_{ref}=3V$, nếu điện áp ở ngõ vào là 1V thì giá trị số thu được sau khi chuyển đổi sẽ là: $1023 \times (1/3) = 314$. Trong đó 1023 là giá trị lớn nhất mà một bộ ADC 10 bit có thể tạo ra ($1023=2^{10}-1$). Vì điện áp tham chiếu ảnh hưởng đến độ chính xác của quá trình chuyển đổi, chúng ta cần tính toán để chọn 1 điện áp tham chiếu phù hợp, không được nhỏ hơn giá trị lớn nhất của input nhưng cũng đừng quá lớn.

1.5.4 I2C

I²C hay thường được viết I2C là viết tắt của cụm từ tiếng Anh “Inter-Integrated Circuit”, là một loại bus nối tiếp được phát triển bởi hãng sản xuất linh kiện điện tử Philips. Ban đầu, loại bus này chỉ được dùng trong các linh kiện điện tử của Philips. Sau đó, do tính ưu việt và đơn giản của nó, I2C đã được chuẩn hóa và được dùng rộng rãi trong các mô đun truyền thông nối tiếp của vi mạch tích hợp ngày nay.

I2C sử dụng hai đường truyền tín hiệu:

- Một đường xung nhịp đồng hồ (SCL) chỉ do Master phát đi (thông thường ở 100kHz và 400kHz. Mức cao nhất là 1Mhz và 3.4MHz).
- Một đường dữ liệu (SDA) theo 2 hướng có Sơ đồ kết nối như Hình 1-3.



Hình 1-3 Sơ đồ kết nối I²C

SCL và SDA luôn được kéo lên nguồn bằng một điện trở kéo lên có giá trị xấp xỉ 4,7 KOhm (tùy vào từng thiết bị và chuẩn giao tiếp, có thể dao động trong khoảng 1KOhm đến 4.7 Kohm. Chú ý rằng theo cấu hình này, một thiết bị có thể ở mức logic LOW hay cao trở nhưng ko thể ở dạng HIGH => Chính trở pull up tạo ra mức logic HIGH).

Các chế độ hoạt động của I²C

- Dựa vào tốc độ ta chia làm 2 loại
- + Chế độ chuẩn (standard mode) hoạt động ở tốc độ 100 Kbit/s.
- + Chế độ tốc độ thấp (low-speed mode) hoạt động ở tốc độ 10 Kbit/s.
- Nếu chia theo quan hệ chủ tớ:
- + Một chủ một tớ.
- + Một chủ nhiều tớ.
- + Nhiều chủ nhiều tớ.

Quá trình truyền dữ liệu

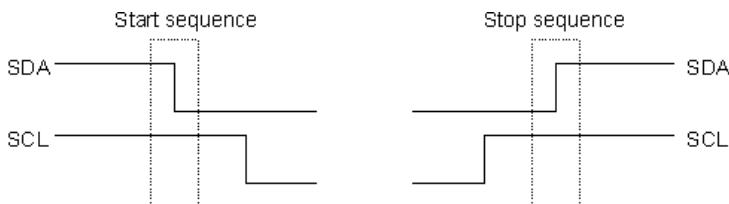
- Thiết bị A (chủ) xác định đúng địa chỉ của thiết bị B (Tớ), cùng với việc xác định địa chỉ, thiết bị A sẽ quyết định đọc hay ghi vào thiết bị tớ.
- Thiết bị A gửi gữi liệu tới thiết bị B
- Thiết bị A kết thúc quá trình truyền dữ liệu.
- Khi A muốn nhận dữ liệu từ B, quá trình diễn ra tương tự, chỉ khác A sẽ nhận dữ liệu từ B.

Cách đánh địa chỉ

I²C sử dụng 7 bit để định địa chỉ, do đó trên một bus có thể định địa chỉ tới 112 nút, 16 địa chỉ còn lại được sử dụng vào mục đích riêng. Bit còn lại quy định việc đọc hay ghi dữ liệu (1 là write, 0 là read)

Định dạng dữ liệu truyền

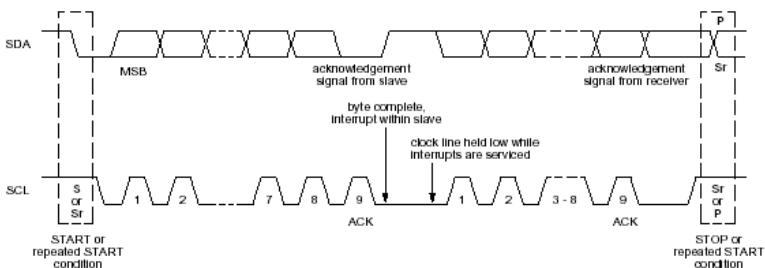
- Dữ liệu được truyền trên bus I²C theo từng bit, bit dữ liệu được truyền đi tại mỗi sườn lên của xung clock trên SCKL, quá trình thay đổi bit dữ liệu xảy ra khi SCL ở mức thấp.



Hình 1-4 Dữ liệu truyền trên bus I²C theo từng bit

- Mỗi byte dữ liệu được truyền có độ dài là 8 bits. Số 1 ượng byte có thể truyền trong một lần là không hạn chế.
- Mỗi byte được truyền sẽ chờ tín hiệu phản hồi là một bit ACK để báo hiệu đã

- Nhận dữ liệu. => Mỗi lần I2C sẽ truyền 8bit và nhận 1bit.



Hình 1-5 Truyền dữ liệu I2C

1.6 Tổng kết

Chương này đã trình bày tổng quan về vi xử lý, vi điều khiển. Trong đó nhấn mạnh về cấu trúc tổng quan, các thành phần bên trong vi điều khiển và cách phân loại vi điều khiển. Bên cạnh đó, nội dung chương cũng trình bày các chuẩn giao tiếp của vi điều khiển và trình bày một số ứng dụng của vi điều khiển trong cuộc sống. Qua đó, sẽ giúp người đọc có cái nhìn tổng quan về vi xử lý, vi điều khiển và biết được những khái niệm cơ bản để có thể học và hiểu được các môn liên quan đến lĩnh vực này

1.7 Bài tập

- Phân biệt khái niệm vi xử lý và vi điều khiển?
- Nêu các thành phần cơ bản bên trong một vi điều khiển?
- Nêu một số ứng dụng (ít nhất 2) của vi điều khiển trong thực tế cuộc sống, và mô tả chi tiết về ứng dụng đó?

4. Chuẩn giao tiếp UART có những đặc trưng gì và nó được ứng dụng để làm gì?
5. Nêu sự khác biệt của chuẩn giao tiếp SPI và I2C?
6. Vì điều khiển bao gồm những thành phần cơ bản nào? Kể tên cụ thể và nêu chức năng của các thành phần đó.

Chương 2. Kiến trúc họ Vi điều khiển 8051

Mục tiêu chương

Chương này sẽ đi sâu trình bày về tính năng, sơ đồ bộ trí chân cũng như cấu trúc bên trong của họ vi điều khiển 8051. Học xong chương này người học cần hiểu rõ về vi điều khiển 8051, nắm rõ các chân giao tiếp cùng chức năng của từng chân, nắm được cấu trúc bộ nhớ và các thanh ghi trong 8051.

2.1 Tính năng bên ngoài

Vi điều khiển 8051 là vi điều khiển tổng quát trong họ MCS-51 của Intel, nó chứa những đặc trưng cho 1 vi điều khiển như sau:

- 4KB ROM
- 128 Byte RAM
- 4 cổng nhập xuất 8 bit
- 2 bộ định thời 16 bit
- Cổng giao tiếp nối tiếp (UART)
- Có thể giao tiếp với bộ nhớ chương trình ngoài 64KB
- Có thể giao tiếp với bộ nhớ dữ liệu ngoài 64KB
- Có thể xử lý theo bit, bộ nhớ có 210 bit được định địa chỉ theo bit
- Thực hiện phép nhân/chia trong 4 chu kỳ máy

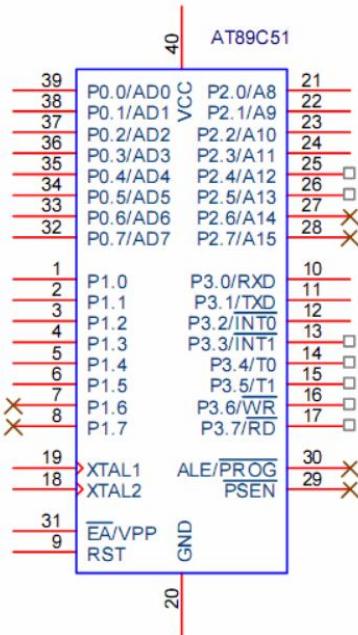
Các vi điều khiển khác trong họ vi điều khiển MCS-51 sẽ có thêm một vài đặc trưng khác về bộ nhớ và bộ định thời dựa vào mục đích sử dụng, được mô tả trong Bảng 2-1

Bảng 2-1 Đặc trưng các vi điều khiển họ MCS - 51

Vi điều khiển	ROM	RAM	Bộ định thời
8051	4 KB (ROM)	128 Byte	2
8031	0 KB	128 Byte	2
8751	4 KB (EPROM)	128 Byte	2
8052	8 KB (ROM)	256 Byte	3
8032	0 KB	256 Byte	3
8752	8 KB (EPROM)	256 Byte	3

2.2 Chân và tín hiệu

Do đặc điểm và chức năng hoạt động của các VĐK họ MSC-51 là hoàn toàn tương tự nhau nên trong giáo trình này sẽ chủ yếu dùng một VĐK thông dụng hiện nay trên thị trường là AT89C51 của hãng Atmel để mô tả các nội dung. Vi điều khiển 8051 có 40 chân tín hiệu để giao tiếp với bên ngoài như trong Hình 2-1.



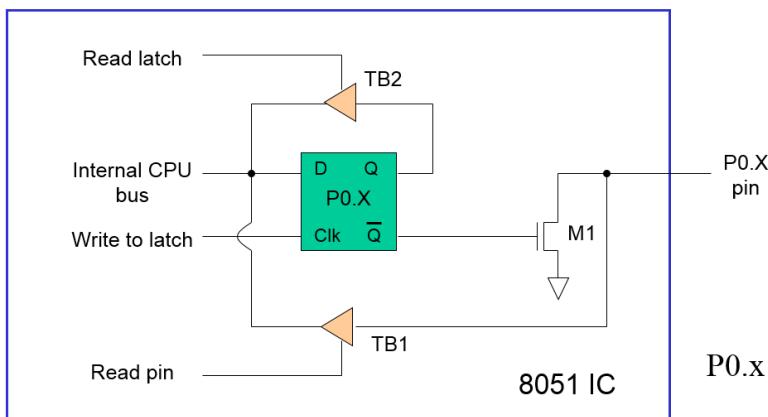
Hình 2-1 Sơ đồ các chân của vi điều khiển 8051

Trong 40 chân tín hiệu của 8051 có 32 chân phục vụ cho mục đích xuất nhập được chia làm 4 cổng là P0, P1, P2, P3; mỗi cổng có 8 chân. Các cổng P0, P2, P3 ngoài chức năng là cổng nhập xuất, còn chứa thêm một chức năng khác tùy vào các chân của cổng như: bus địa chỉ, ngắt ngoài, timer ngoài, cổng nối tiếp và một số tín hiệu khác.

Trong phần này chúng ta sẽ tìm hiểu cụ thể về các chân tín hiệu trên 8051 về chức năng và hoạt động của nó.

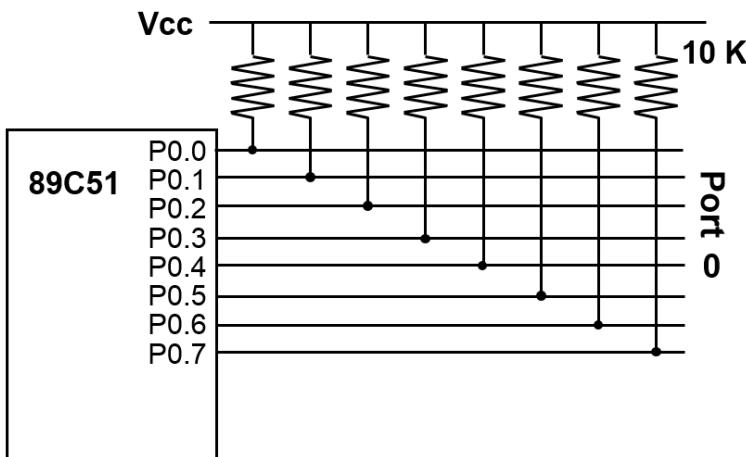
2.2.1 Port 0

Port 0 là port có 2 chức năng ở các chân từ 32 đến 39 của vi điều khiển 8051. Port 0 là port duy nhất trong vi điều khiển 8051 không có điện trở kéo lên bên trong vi điều khiển, Hình 2-2 là cấu trúc một chân của port 0.



Hình 2-2 Cấu tạo bên trong port 0

Trong các thiết kế không có bộ nhớ ngoài, port 0 được sử dụng làm nhiệm vụ xuất/nhập, khi dùng chức năng này thì port 0 phải dùng thêm các điện trở kéo lên được lắp thêm bên ngoài như Hình 2-3. Khi sử dụng port 0 làm input thì phải nạp giá trị 1 vào port 0 sau đó đọc dữ liệu.

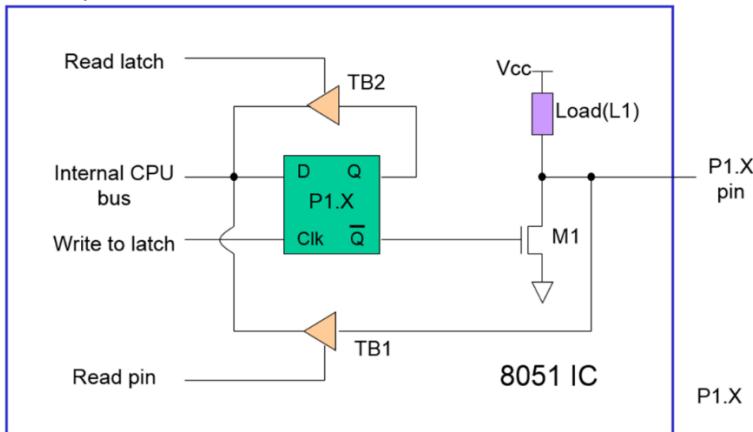


Hình 2-3 Điện trở kéo lên tại port 0

Trong các thiết kế có sử dụng bộ nhớ ngoài, port 0 còn có thêm chức năng là bus dữ liệu và bus địa chỉ đa hợp. Tức là trong nửa chu kỳ đầu của quá trình đọc bộ nhớ port 0 sẽ truyền 8 bit địa chỉ của bộ nhớ ngoài (đây là 8 bit thấp trong 16 bit địa chỉ), nó thường được nối với IC 74LS373 để chốt địa chỉ, sau đó truyền hoặc nhận 8 bit dữ liệu từ bộ nhớ ngoài. Ngoài ra, port 0 còn dùng để nhận mã khi lập trình và xuất mã khi kiểm tra trong quá trình nạp chương trình vào vi điều khiển.

2.2.2 Port 1

Port 1 chỉ có một chức năng là nhập xuất ở các chân từ 1 đến 8 của vi điều khiển 8051, không sử dụng cho mục đích khác (trừ 8052 thì sử dụng chân P1.0 và P1.1 cho bộ định thời thứ 3). Bên trong Port 1 đã có điện trở kéo lên, nên khi sử dụng làm cổng nhập/xuất thì không cần phải gắn thêm điện trở kéo lên bên ngoài, điều này tương tự ở các port 2,3. Cấu trúc bên trong của port 1 được mô tả như Hình 2-4



Hình 2-4 Cấu tạo bên trong port 1

Ngoài ra, port 1 còn dùng làm 8 bit địa chỉ thấp trong quá trình nạp hoặc kiểm tra chip.

2.2.3 Port 2

Port 2 là port có 2 chức năng ở các chân từ 21 đến 28 của vi điều khiển 8051. Trong các thiết kế không có bộ nhớ ngoài port 2 chỉ làm nhiệm vụ nhập/xuất, còn khi có bộ nhớ ngoài thì port 2 sẽ được sử dụng làm byte địa chỉ cao trong 16 bit địa chỉ. Ngoài ra, port 2 còn dùng làm 8 bit địa chỉ cao hoặc tín hiệu điều khiển trong quá trình nạp hoặc kiểm tra chip.

2.2.4 Port 3

Port 3 là port có 2 chức năng ở các chân từ 10 đến 17 của vi điều khiển 8051. Ngoài chức năng nhập/xuất thông thường như các port khác, port 3 còn có nhiều chức năng riêng, mỗi chân trên port 3 có chức năng liên quan đến một đặt trung cụ thể trong 8051 được mô tả trong **Error! Reference source not found.**

Bảng 2-2 Chức năng thứ 2 của các chân tại port 3

Bit	Tên	Địa chỉ bit	Chức năng
P3.0	RxD	B0H	Ngõ vào cổng nối tiếp
P3.1	TxD	B1H	Ngõ ra cổng nối tiếp
P3.2	INT0	B2H	Ngắt ngoài 0
P3.3	INT1	B3H	Ngắt ngoài 1
P3.4	T0	B4H	Ngõ vào bộ định thời 0
P3.5	T1	B5H	Ngõ vào bộ định thời 1
P3.6	WR	B6H	Tín hiệu điều khiển ghi dữ liệu lên bộ nhớ ngoài
P3.7	RD	B7H	Tín hiệu điều khiển đọc dữ liệu từ bộ nhớ ngoài

2.2.5 Các chân nguồn

Vi điều khiển 8051 sử dụng nguồn điện một chiều +5V. Cực dương (VCC) 5V được nối vào chân 40 của vi điều khiển. Cực âm (GND) được nối vào chân 20 của vi điều khiển.

2.2.6 Chân cho phép bộ nhớ chương trình PSEN

Chân PSEN là chân 29 của vi điều khiển 8051 cho phép đọc bộ nhớ chương trình mở rộng đối với các ứng dụng sử dụng ROM ngoài, thường được nối với chân OE (Output Enable) của EPROM hoặc Rom để cho phép đọc các byte mã lệnh. Chân PSEN sẽ ở mức logic 0 trong thời gian vi điều khiển 8051 tìm nạp lệnh lên từ ROM ngoài, mã lệnh của chương trình được đọc từ ROM thông qua bus dữ liệu (port 0) và bus địa chỉ (port 0 + 1). Khi chương trình thực thi trong ROM nội thì chân PSEN sẽ ở mức logic 1.

2.2.7 Chân cho phép chốt địa chỉ ALE

Chân ALE là chân 30 của vi điều khiển 8051, cho phép xuất tín hiệu chốt địa chỉ để giải đa hợp bus dữ liệu và bus địa chỉ ở port 0 trong quá trình giao tiếp với bộ nhớ ngoài. Khi port 0 được sử dụng làm bus dữ liệu/địa chỉ đa hợp, chân ALE xuất tín hiệu để chốt địa chỉ (byte của địa chỉ 16 bit) vào một IC chốt (74LS373) trong nửa đầu của chu kỳ bộ nhớ, sau đó các chân của port 0 sẽ đọc/ghi dữ liệu trong nửa cuối của chu kỳ bộ nhớ. Chân ALE thường được nối với chân G của IC chốt (74LS373). Các xung tín hiệu tại chân ALE có tần số bằng 1/6 tần số của mạch dao động bên trong vi điều khiển và có thể được dùng làm tín hiệu clock cho các thành phần khác của hệ thống. Nếu mạch dao động có tần số 12MHz, tín hiệu tại chân ALE có tần số là 2MHz. Chân ALE còn được dùng để làm ngõ vào cho xung lập trình EPROM trong vi điều khiển 8051 khi nạp chương trình.

2.2.8 Chân truy xuất bộ nhớ ngoài EA

Chân truy xuất bộ nhớ ngoài EA là chân 31 của vi điều khiển 8051 dùng để cho phép thực thi chương trình từ ROM ngoài. Khi nối chân EA với Vcc, vi điều khiển 8051 sẽ thực thi chương trình từ ROM nội (tối đa 4KB), ngược lại khi nối chân EA với GND thì vi điều khiển 8051 sẽ thực thi chương trình từ ROM ngoại (tối đa 64KB). Ngoài ra chân EA còn được dùng để làm chân cấp nguồn 12V khi lập trình cho EPROM trong 8051.

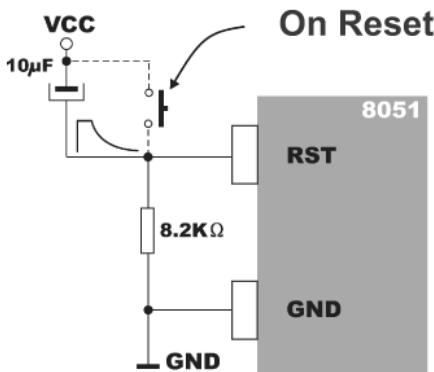
2.2.9 Chân Reset

Chân Reset là chân 9 của vi điều khiển 8051 dùng để thiết lập lại trạng thái ban đầu cho hệ thống hay còn gọi là reset hệ thống. Khi chân được này được đưa lên mức cao (nối với Vcc) trong ít nhất 2 chu kỳ máy, thì các thanh ghi bên trong 8051 được nạp lại các giá trị như khi khởi động hệ thống và chương trình sẽ chạy lại từ đầu (ngoại trừ các thanh ghi SBUF, và các thanh ghi port không có khởi tạo giá trị). Giá trị của các thanh ghi sau khi reset hệ thống được mô tả trong bảng..

Bảng 2-3 Giá trị các thanh ghi khi reset

Thanh ghi	Giá trị sau khi Reset
PC	0000
ACC	00
B	00
PSW	00
SP	07
DPTR	0000
Các byte nhớ khác trong RAM	00

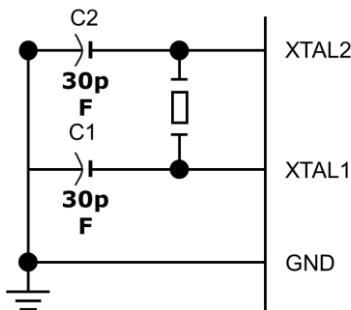
Thông thường chân Reset sẽ được nối với một nút ấn theo sơ đồ mạch trong Hình 2-5



Hình 2-5 Các lắp mạch reset

2.2.10 Các chân vào bộ dao động trên chip

Thông thường mạch dao động bên trong vi điều khiển 8051 được kết nối với bộ tạo xung thạch anh ngoài ở 2 chân 18 và 19 của vi điều khiển 8051. Với vi điều khiển 8051 có thể sử dụng thạch anh ngoài có tần số từ 3MHz đến 30 MHz tùy theo mục đích sử dụng, thông thường vi điều khiển sẽ sử dụng thạch anh ở tần số 11.0592MHz hoặc 12MHz. Để sử dụng thạch anh ngoài chúng ta cần thiết lập mạch như Hình 2-6

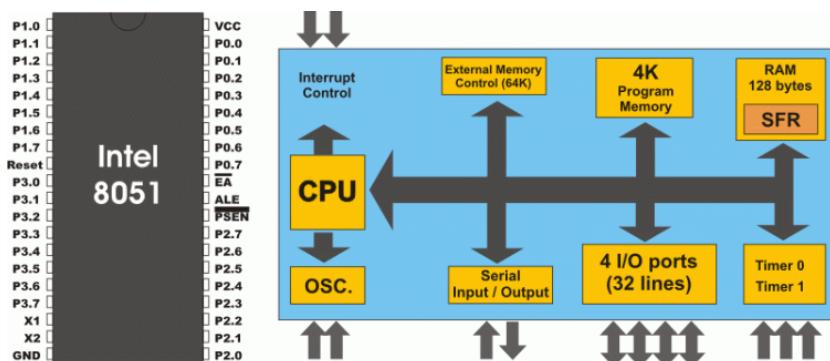


Hình 2-6 Cách lắp thạch anh ngoài

Chu kỳ máy của vi điều khiển sẽ được tính theo tần số thạch anh. Ví dụ tần số thạch anh là 11.0592MHz, thì tần số đầu ra của bộ tạo dao động nội là $11.0592\text{MHz}/12 = 921.6\text{kHz}$, suy ra chu kỳ máy = $1/921.6\text{kHz} = 1.085ms$.

2.3 Kiến trúc bên trong

Vi điều khiển họ 8051 có nhiều phiên bản được các nhà sản xuất khác nhau, mỗi phiên bản có thể thêm bớt một số thành phần bên trong cấu trúc của vi điều khiển nhưng nhìn chung kiến trúc bên trong của vi điều khiển họ 8051 được mô tả như **Hình 2-7**



Hình 2-7 Kiến trúc bên trong vi điều khiển 8051

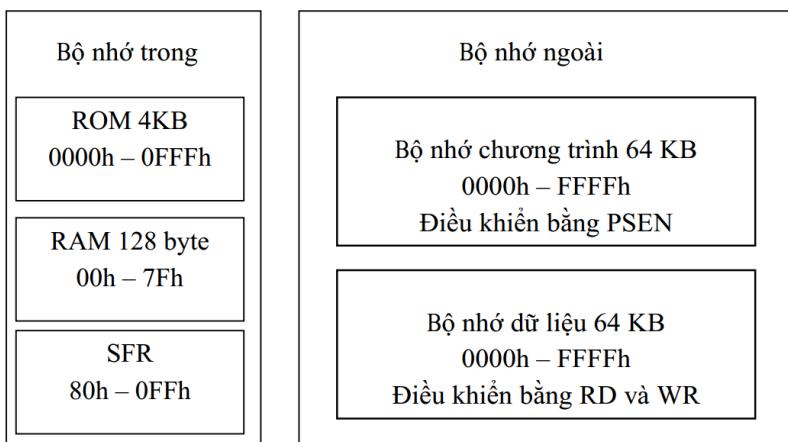
Cụ thể cấu trúc bên trong vi điều khiển 8051 (lấy AT89C51 làm tiêu biểu) bao gồm các thành phần sau:

- CPU để thực thi các câu lệnh cho vi điều khiển
- OSC dùng để tạo xung nhịp hoạt động cho vi điều khiển bao gồm bộ tạo dao động nội và bộ tạo giao động ngoại (thường sử dụng thạch anh)
- Cổng vào ra nối tiếp (UART)
- 4 cổng nhập xuất, mỗi cổng có 8 bit.
- 2 bộ timer

- Bộ nhớ RAM 128 và các thanh ghi có chức năng đặt biệt
- Bộ nhớ ROM 4KB dùng để lưu chương trình và các biến cố định

2.4 Tổ chức bộ nhớ

Bộ nhớ của vi điều khiển 8051 có thể chia thành 2 thành phần bộ nhớ: bộ nhớ trong và bộ nhớ ngoài. Bộ nhớ trong bao gồm 4 KB ROM, 128 byte RAM và vùng thanh ghi có chức năng đặt biệt. Các byte RAM có địa chỉ từ 00H đến 7FH và các thanh chức năng đặt biệt (SFR) có địa chỉ từ 80H đến FFH. Bộ nhớ ngoài bao gồm bộ nhớ chương trình (điều khiển đọc bằng tín hiệu PSEN) và bộ nhớ dữ liệu (điều khiển bằng tín hiệu RD à WR để cho phép đọc hay ghi dữ liệu). Do đường địa chỉ của vi điều khiển 8051 có 16 bit (8 bit thấp ở port 0 và 8 bit cao ở port 2) nên có thể truy cập bộ nhớ ngoài tối đa 64KB. Mô hình bộ nhớ được mô tả trong Hình 2-8



Hình 2-8 Tổ chức bộ nhớ của vi điều khiển 8051

2.4.1 Tổ chức bộ nhớ trong

Bộ nhớ trong của vi điều khiển 8051 được thiết kế theo cấu trúc Harvard bao gồm bộ nhớ chương trình (ROM) và bộ nhớ dữ liệu (RAM). ROM dùng để lưu trữ chương trình cần thực thi (các chỉ lệnh) và chương trình không bị mất đi khi vi điều khiển không được cấp nguồn. RAM sử dụng để lưu các kết quả tạm thời trong quá trình tính toán, những giá trị tức thời, stack và được sử dụng trong suốt quá trình vi điều khiển hoạt động. Các giá trị trong RAM sẽ bị mất khi hệ thống reset hoặc bị ngừng cấp nguồn.

2.4.1.1 Bộ nhớ chương trình

Bộ nhớ chương trình ROM dùng để lưu chương trình mà vi điều khiển cần thực hiện để giải quyết công việc, ngoài ra ROM còn sử dụng để chứa số liệu các bảng, các tham số hệ thống, các số liệu cố định của hệ thống. Chương trình do người lập trình viết trên các IDE, sau đó được biên dịch thành các file HEX để nạp vào ROM của 8051 tại các địa chỉ theo byte. Địa chỉ bắt đầu là 0000H cho đến địa chỉ kết thúc là 0FFFH, trong quá trình lập trình có thể định vị trí bắt đầu các câu lệnh bằng câu lệnh ORG. Trong vi điều khiển 8051, vùng nhớ chương trình từ 0000H đến 002FH được dùng để chứa bản vector ngắn, nên khi lập trình có sử dụng ngắn thông thường chương trình sẽ bắt đầu tại địa chỉ 0030H.

Trong quá trình hoạt động của vi điều khiển, nội dung của ROM là cố định. Nội dung này chỉ bị thay đổi khi ROM ở chế độ xóa hoặc nạp lại chương trình bằng mạch nạp và chương trình nạp.

2.4.1.2 Bộ nhớ dữ liệu

Bộ nhớ dữ liệu trong vi điều khiển được chia thành nhiều vùng có mục đích khác nhau bao gồm trong 2 vùng lớn đó là

vùng RAM nội (có địa chỉ từ 00H đến 7FH) và vùng các thanh ghi đặc biệt (có địa chỉ từ 80H đến FFH).

a. RAM nội

Vùng RAM nội lại được chia thành các vùng nhỏ với nhiều chức năng khác nhau: các bank thanh ghi có địa chỉ từ 00H đến 1FH, vùng bộ nhớ có thể định địa chỉ theo bit gồm 128 bit có địa chỉ từ 20H đến 2FH và vùng bộ nhớ định địa chỉ theo byte có địa chỉ từ 30H đến 7FH.

Dịa chỉ byte	Địa chỉ bit								Chức năng
7F									
30									Vùng RAM đa dụng
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	Vùng có thể định địa chỉ bit
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 18	Bank 3								
17 10	Bank 2								Các bank thanh ghi
1F 08	Bank 1								
07 00	Bank thanh ghi 0 (mặc định cho R0-R7)								

Hình 2-9 Cấu trúc địa chỉ bên trong RAM nội

i. Các Bank thanh ghi

Vùng địa chỉ từ 00H đến 1FH được chia thành 4 bank thanh ghi: bank 0 từ 00H đến 07H, bank 1 từ 08H đến 0FH, bank 2 từ 10H đến 17H, bank 3 từ 18H đến 1FH. Các byte trong mỗi bank thanh ghi này được đại diện bằng các ký hiệu từ R0 đến R7. Sau khi khởi động hệ thống thì bank thanh ghi mặc định được sử dụng là bank 0. Do có 4 bank thanh ghi nên tại một thời điểm chỉ có một bank thanh ghi được sử dụng bởi các thanh ghi từ R0 đến R7. Các bit RS0 và RS1 của thanh ghi trạng thái chương trình PSW được sử dụng để chọn bank thanh ghi khác ngoài bank mặc định là bank 0 theo Bảng 2-4

Bảng 2-4 Lựa chọn các bank thanh ghi

RS1	RS0	Bank thanh ghi
0	0	Bank 0
0	1	Bank 1
1	0	Bank 2
1	1	Bank 3

ii. Vùng RAM định địa chỉ theo bit

Vùng RAM có địa chỉ từ 20H đến 2FH gồm 16 byte (128 bit) có thể truy xuất theo byte hoặc truy xuất theo bit bằng các lệnh xử lý bit. Các bit của vùng RAM này có địa chỉ bit bắt đầu tại 00H và kết thúc tại 7FH. Như vậy địa chỉ byte bắt đầu 20H gồm 8 bit có địa chỉ bit từ 00H đến 07H, địa chỉ byte kết thúc có địa chỉ bit từ 78H đến 7Fh.

Ví dụ để thiết lập bit 07H (MSB của địa chỉ byte 20H) lên 1 ta có thể dùng lệnh sau:

- Theo bit: SETB 07H
- Theo byte: MOV A, 20H
ORL A, #10000000B

MOV 20H, A

iii. Vùng RAM đa dụng

Vùng RAM đa dụng có 80 byte từ địa chỉ 30H đến 7FH có thể truy xuất theo byte (8 bit) bằng cách dùng chế độ địa chỉ trực tiếp hay gián tiếp. Vùng này thường dùng để lưu các dữ liệu trong quá trình hoạt động của vi điều khiển.

Ví dụ, để đọc nội dung ở địa chỉ 34H của RAM nội vào thanh ghi tích lũy ta có thể dùng 2 chế độ địa chỉ như sau:

- Địa chỉ trực tiếp:

MOV A, 34H ;chuyển giá trị byte dữ liệu
 tại địa chỉ trực tiếp 34H vào
 thanh ghi tích lũy A

- Địa chỉ gián tiếp qua R1:

MOV R1, ;chuyển giá trị 34H vào
 #34H thanh ghi R1

MOV A, @R1 ;chuyển dữ liệu tại địa chỉ
 lưu trong R1 vào thanh ghi
 tích lũy A

b. Vùng các thanh ghi có chức năng đặc biệt

Các thanh ghi có chức năng đặc biệt trong vi điều khiển 8051 là một phần trong bộ nhớ dữ liệu. Vì vậy mỗi thanh ghi sẽ có một địa chỉ riêng (ngoại trừ thanh ghi PC và thanh ghi lệnh vì các thanh ghi này ít bị truy xuất trực tiếp nên chúng không được đặt trong bộ nhớ dữ liệu). Cũng như các thanh ghi R0 đến R7, ngoài việc truy cập theo tên trong Bảng 2-5 thì 21 thanh ghi có chức năng đặc biệt của vi điều khiển 8051 có thể truy cập bằng địa chỉ từ 80H đến FFH. Tuy nhiên trong vùng địa chỉ từ 80H đến FFH thì chỉ có 1 số byte nhớ được sử dụng.

Bảng 2-5 Các thanh ghi có chức năng đặc biệt

Địa chỉ byte	Có thể định địa chỉ bit	Không định địa chỉ bit							
F8h									
F0h	B								
E8h									
E0h	ACC								
D8h									
D0h	PSW								
C8h	(T2CON)		(RCAP2L)	(RCAP2H)	(TL2)	(TH2)			
C0h									
B8h	IP	SADEN							
B0h	P3								
A8h	IE	SADDR							
A0h	P2								
98h	SCON	SBUF	BRL	BDRCON					
90h	P1								
88h	TCON	TMOD	TL0	TH0	TL1	TH1	AUXR	CKCON	
80h	P0	SP	DPL	DPH					PCON

Trong đó, đa số các thanh ghi có chức năng đặc biệt được truy xuất bằng chế độ địa chỉ trực tiếp hay tên, riêng thanh ghi A có thể được truy xuất ngầm định trong một số lệnh. Một số thanh ghi có thể được định địa chỉ theo bit hoặc theo byte. Ví dụ lệnh SETB 0E0H sẽ lập bit 0 trong thanh ghi tích lũy, các bit khác trong thanh ghi này không bị thay đổi. Địa chỉ 0E0H vừa là địa chỉ byte của thanh ghi tích lũy vừa là địa chỉ bit có trọng số nhỏ nhất trong thanh ghi tích lũy nhưng vì lệnh SETB chỉ tác động lên bit nên địa chỉ bit sẽ có tác dụng trong lệnh này.

i. Thanh ghi tích lũy – Accumulator (A)

Thanh ghi tích lũy là thanh ghi được sử dụng nhiều nhất trong vi điều khiển 8051, được ký hiệu là thanh ghi A hoặc ACC (trong các lệnh xử lý bit). Thanh ghi tích lũy có thể truy xuất

trực tiếp thông qua địa chỉ 0E0H (theo byte) hay truy xuất theo từng bit thông qua địa chỉ từ E0H đến E7H.

Ví dụ câu lệnh:

MOV A, #2 và

MOV 0E0H, # 2 có cùng kết quả

Hoặc

SETB ACC.4 và

SETB 0E4H cũng có cùng kết quả

ii. Thanh ghi phụ (B)

Thanh ghi B dùng chung với thanh ghi A cho các phép toán nhân, chia và có thể dùng như một thanh ghi tạm để chứa các kết quả trung gian. Lệnh “*MUL AB*” nhân 2 số 8 bit không dấu chứa trong A và B, kết quả 16 bit được lưu vào cặp thanh ghi B:A (thanh ghi A chứa byte thấp, thanh ghi B chứa byte cao). Lệnh chia “*DIV AB*” chia A cho B và kết quả lưu trong A, số dư lưu trong thanh ghi B. Thanh ghi B có địa chỉ byte là F0H và cũng có thể truy cập theo từng bit với địa chỉ bit từ F0H đến F7H.

iii. Thanh ghi trạng thái chương trình (PSW)

Thanh ghi trạng thái chương trình là một thanh ghi có chức năng đặc biệt quan trọng nhất. Nó chứa các bit phản ánh trạng thái hiện tại của CPU bao gồm: cờ nhớ, cờ nhớ phụ, cờ zero, 2 bit để chọn bank thanh ghi, cờ tràn và cờ chẵng lẻ (có 1 bit không được sử dụng). Thanh ghi này có địa chỉ byte là D0H, và có thể truy cập theo từng bit thông qua tên bit hoặc địa chỉ bit từ D0H đến D7H. Các bit chức năng của thanh ghi trạng thái chương trình được mô tả trong Bảng 2-6.

Bảng 2-6 Chức năng của thanh ghi trạng thái chương trình

Bit	Tên	Địa chỉ	Chức năng
PSW.7	CY	D7H	Cờ nhớ

PSW.6	AC	D6H	Cờ nhớ phụ
PSW.5	F0	D5H	Cờ zero
PSW.4	RS1	D4H	Chọn bank thanh ghi
PSW.3	RS0	D3H	Chọn bank thanh ghi
PSW.2	OV	D2H	Cờ tràn
PSW.1	-	D1H	Không sử dụng
PSW.0	P	D0H	Cờ chẵn lẻ

Trong đó:

- Cờ nhớ (CY): được dùng cho các lệnh toán học trên các số không dấu, nó sẽ được lập bằng 1 nếu có số nhớ trong phép cộng hoặc có mượn số trong phép trừ. Ví dụ nếu thanh ghi A đang chứa giá trị 15H thì lệnh “ADD A, #0EBH” sẽ cho kết quả là 00H trong thanh ghi A và lập cờ nhớ CY lên 1 (do kết quả là 100H mà thanh ghi A chỉ có 8 bit). Cờ nhớ cũng có thể dùng như một bit nhớ thông thường cho các lệnh xử lý bit. Ví dụ lệnh “SETB C” sẽ lập cờ nhớ lên 1.
- Cờ nhớ phụ (AC): được sử dụng trong cộng các số BCD, nếu có một số nhớ được tạo ra từ bit 3 chuyển sang bit 4 trong phép cộng hoặc kết quả của 4 bit thấp nằm trong khoảng 0AH đến 0FH thì cờ nhớ phụ sẽ được lập lên 1. Nếu thực hiện cộng các số BCD thì sau lệnh cộng cần hiệu chỉnh thập phân bằng lệnh “DA A” để các kết quả lớn hơn 9 trở về khoảng từ 0 đến 9.
- Cờ zero (F0): cờ này được sử dụng cho nhiều chức năng tùy thuộc vào người lập trình như trong các lệnh so sánh, nhảy... Nó được lập bằng 1 khi kết quả phép tính bằng 0.
- Các bit chọn thanh ghi (RS1, RS0): được dùng để chọn bank thanh ghi nào trong 4 bank thanh ghi được sử dụng như được mô tả trong Bảng 2-4 . Các bit này được xóa khi hệ thống reset, tức là bank 0 được sử dụng mặc định và có thể lập trình bằng phần mềm

theo bit qua tên hoặc qua địa chỉ bit là D3H và D4H. Ví dụ các lệnh sau cho phép bank 3 được sử dụng và sau đó di chuyển nội dung của thanh ghi R2 (địa chỉ byte 1AH) vào thanh ghi A:

```
SETB RS1  
SETB RS0  
MOV A, R2
```

- Cờ tràn (OV): được dùng để báo tràn trong các lệnh toán học có dấu. Cờ OV được lập bằng 1 khi phép cộng hoặc trừ số có dấu có xuất hiện một tràn số học, từ là kết quả có giá trị lớn hơn +128 hoặc nhỏ hơn -127. Với phép cộng trừ các số không dấu cờ tràn được bỏ qua. Ví dụ phép tính các số có dấu sau sẽ gây ra hiện tượng tràn và set cờ OV lên 1:

Số hex	Số thập phân
OF	15
<u>+7F</u>	<u>+127</u>

8E (biểu diễn số âm -116) 142

- Cờ chẵn lẻ (P): được dùng để kiểm tra chẵn lẻ cho thanh ghi A, để đảm bảo số bit 1 trong thanh ghi A cộng với bit P luôn luôn là 1 số chẵn. Tức là nếu số bit 1 trong thanh ghi A là số lẻ thì P bằng 1 và ngược lại. Ví dụ nếu thanh ghi A chứa nội dung 10101011B, có 5 bit 1 thì P sẽ bằng 1 để thành 6 bit 1. Bit chẵn lẻ được sử dụng nhiều trong việc truyền và nhận qua cổng nối tiếp.

iv. Thanh ghi con trả stack SP

Stack là một dạng bộ nhớ lưu trữ dạng vào sau ra trước (LIFO – Last In First Out) thường được dùng để lưu trữ địa chỉ trả về khi gọi một chương trình con, hoặc cũng có thể sử dụng để lưu các kết quả trong quá trình tính toán bằng lệnh PUSH và POP.

Thanh ghi con trỏ stack SP là 1 thanh ghi 8 bit có địa chỉ 81H, nó chứa địa chỉ của dữ liệu nằm ở đỉnh stack. Các lệnh liên quan đến stack bao gồm lệnh cất dữ liệu và stack (PUSH) và lệnh lấy dữ liệu ra khỏi stack (POP). Việc cất dữ liệu vào stack sẽ làm tăng SP lên 1 đơn vị trước khi ghi dữ liệu, và việc lấy dữ liệu ra khỏi stack sẽ làm giảm SP. Stack trong vi điều khiển 8051 được chứa trong RAM nội (vùng 128 byte), giá trị khởi tạo ban đầu của stack khi reset hệ thống là 07H, nghĩa là stack bắt đầu chứa dữ liệu từ địa chỉ 08H (vì SP sẽ tăng trước khi lưu dữ liệu). Như vậy nếu không gán giá trị khởi tạo cho thanh ghi SP thì vùng bank 1 của thanh ghi sẽ sử dụng làm stack và sẽ tiếp tục ở các bank 2,3 và vùng nhớ tiếp theo. Nếu trong các ứng dụng cần dùng nhiều bank thanh ghi hoặc cần stack lớn ta có thể khởi động stack bằng phần mềm. Ví dụ lệnh “*MOV SP, #5FH*” sẽ khởi động vùng stack ở địa chỉ 60H.

v. Thanh ghi con trỏ dữ liệu (DPTR)

Thanh ghi con trỏ dữ liệu được dùng để truy xuất bộ nhớ chương trình ngoài hoặc bộ nhớ dữ liệu ngoài. Đây là một thanh ghi 16 bit bao gồm 2 thanh ghi 8 bit có địa chỉ ở 2 byte liên tiếp trong bộ nhớ là 82H (DPL, byte thấp) và 83H (DPH, byte cao). Ví dụ muốn ghi giá trị 12H vào RAM ngoài ở địa chỉ 1234H thì ta phải dùng đoạn lệnh sau:

<i>MOV</i>	<i>A, #12H</i>	; chuyển giá trị 12H vào thanh ghi A
<i>MOV</i>	<i>DPTR,</i> <i>#1234H</i>	; chuyển giá trị 1234H vào thanh ghi DPTR
<i>MOV</i>	<i>@DPTR, A</i>	; chuyển giá trị A

vi. Các thanh ghi cổng vào ra

Các thanh ghi cổng vào ra của vi điều khiển 8051 bao gồm thanh ghi cổng 0 (P0) tại địa chỉ 80H, thanh ghi cổng 1 (P1) tại địa chỉ 90H, thanh ghi cổng 2 (P2) tại địa chỉ A0H và thanh ghi

cổng 3 (P3) tại địa chỉ B0H. Tất cả các thanh ghi này đều cho phép định địa chỉ theo bit trong đó địa chỉ của thanh ghi P0 từ 80H đến 87H, P1 từ 90H đến 97H, P2 từ A0H đến A7H và P3 từ B0H đến B7H. Các bit địa chỉ này có thể truy cập bằng tên hoặc bằng địa chỉ. Ví dụ 2 lệnh sau: SETB P0.0 và SETB 80H có tác dụng như nhau là lập bit có trọng số thấp nhất của cổng 0 lên 1.

Khi vi điều khiển 8051 giao tiếp với bộ nhớ ngoài thì các thanh ghi cổng 0 và 2 không được dùng để nhập xuất mà nó sẽ có nhiệm vụ truyền/nhận địa chỉ và dữ liệu với bộ nhớ ngoài. Thanh ghi cổng 3 còn được sử dụng cho một số tính năng riêng biệt của 8051 ngoài chức năng nhập xuất như là: ngắn, timer ngoài, cổng nối tiếp...

vii. Thanh ghi bộ đệm dữ liệu nối tiếp (SBUF)

Vi điều khiển 8051 có một cổng nối tiếp bên trong dành cho việc trao đổi dữ liệu với các thiết bị nối tiếp như máy tính, vi điều khiển khác hoặc các IC khác có giao tiếp nối tiếp. Thanh ghi bộ đệm dữ liệu nối tiếp ở địa chỉ 99H sẽ giữ cả hai dữ liệu truyền và nhận. Khi truyền dữ liệu thì ghi dữ liệu lên thanh ghi SBUF, khi nhận dữ liệu thì đọc thanh ghi SBUF. Cổng nối tiếp có nhiều chế độ hoạt động khác nhau được lựa chọn bằng các bit trên thanh ghi điều khiển port nối tiếp SCON sẽ được giới thiệu trong phần sau.

viii. Thanh ghi định thời (Timer)

Vi điều khiển 8051 có 2 bộ định thời/đếm 16 bit để xác định các khoảng thời gian hoặc đếm các sự kiện. Mỗi bộ định thời 16 bit được cấu thành từ 2 thanh ghi 8 bit được gọi là thanh ghi định thời, thanh ghi này sẽ chứa giá trị ban đầu của bộ định thời được nạp vào trong quá trình lập trình. Tùy thuộc vào khoảng thời gian cần định thời mà người lập trình sẽ tính toán các giá trị cho các thanh ghi định thời này. Bộ định thời 0 gồm 2 thanh ghi định thời là TH0 có địa chỉ 8CH chứa các 8 bit cao của bộ định

thời 0 và TL0 có địa chỉ 8AH chứa 8 bit thấp của bộ định thời 0. Tương tự bộ định thời 1 gồm 2 thanh ghi định thời là TH1 có địa chỉ 8DH chứa các 8 bit cao của bộ định thời 1 và TL1 có địa chỉ 8BH chứa 8 bit thấp của bộ định thời 1. Các thanh ghi định thời này có thể truy cập theo tên hoặc địa chỉ. Ngoài ra trong vi điều khiển 8052 còn có cặp thanh ghi TH2/TL2 và cặp thanh ghi RCAP2L/RCAP2H dùng cho bộ định thời 2.

ix. Các thanh ghi điều khiển

Các thanh ghi điều khiển bao gồm các thanh ghi có chức năng điều khiển một số phần nào đó trong vi điều khiển 8051 bao gồm: thanh ghi IP và IE điều khiển ngắt, thanh ghi TMOD và TCON điều khiển timer, thanh ghi SCON điều khiển cổng nối tiếp, thanh ghi PCON điều khiển nguồn. Trong phần này chỉ giới thiệu sơ lược về các thanh ghi này, chi tiết các thanh ghi sẽ được đề cập ở các chương nói về các chức năng ngắt, timer, cổng nối tiếp. Riêng thanh ghi điều khiển nguồn PCON sẽ được nói rõ luôn trong phần này.

- Thanh ghi IP có địa chỉ tại B8H cho phép chọn mức ưu tiên ngắt khi có 2 ngắt xảy ra đồng thời. Thanh ghi này cho phép định địa chỉ theo bit từ B8H đến BFH.
- Thanh ghi IE có địa chỉ tại A8H cho phép hay cấm các ngắt trong 8051. Thanh ghi này cũng cho phép định địa chỉ theo bit từ A8H đến AFH.
- Thanh ghi TMOD có địa chỉ tại 89H dùng để chọn chế độ hoạt động cho các bộ định thời 0 và 1. Thanh ghi này không cho phép định địa chỉ theo bit nên mỗi lần cần phải nạp 8 bit cho cả thanh ghi.
- Thanh ghi TCON có địa chỉ tại 88H dùng để điều khiển hoạt động của bộ định thời và ngắt, thanh ghi này có thể định địa chỉ theo bit từ 88H đến 8FH.
- Thanh ghi SCON có địa chỉ tại 98H dùng để điều khiển hoạt động của cổng nối tiếp, có thể định địa chỉ bit từ 98H đến 9FH.

- Thanh ghi điều khiển nguồn (PCON) có địa chỉ tại 87H có nhiều bit dùng để xác định các chức năng và chế độ hoạt động khác nhau của vi điều khiển. Thanh ghi này không cho phép định địa chỉ theo bit. Chức năng của các bit trong thanh ghi PCON được mô tả trong Bảng 2-7.

Bảng 2-7 Chức năng các bit trong thanh ghi PCON

Bit	Ký hiệu	Mô tả
PCON.7	SMOD	Bit tăng gấp đôi tốc độ baud cho cổng nối tiếp
PCON.6	-	
PCON.5	-	
PCON.4	-	
PCON.3	GF1	Cờ đa dụng 1 (dành cho người dùng)
PCON.2	GF0	Cờ đa dụng 0 (dành cho người dùng)
PCON.1	PD	Cho phép vi điều khiển hoạt động trong chế độ giảm nguồn
PCON.0	IDL	Cho phép vi điều khiển hoạt động trong chế độ rỗi

- + Bit SMOD khi được thiết lập sẽ làm tăng gấp đôi tốc độ baud trong giao tiếp cổng nối tiếp khi cổng nối tiếp hoạt động trong các chế độ 1, 2 hoặc 3
- + Bit PD là bit cho phép vi điều khiển hoạt động trong chế độ nguồn giảm. Khi PD được lập bằng 1, thì vi điều khiển sẽ đi vào hoạt động ở chế độ nguồn giảm. Trong chế độ này bộ dao động tạo xung nhịp sẽ ngưng hoạt động, sẽ làm cho các khôi chức năng bên trong vi điều khiển ngừng hoạt động. Nội dung của RAM nội sẽ được giữ nguyên, mức logic của các cổng cũng

được giữ nguyên, cổng ALE và PSEN sẽ được giữ ở mức thấp. Nguồn cung cấp cho vi điều khiển có thể giảm xuống 2V sau khi vi điều khiển chuyển vào chế độ này. Vi điều khiển sẽ thoát khỏi chế độ giảm nguồn khi reset hệ thống, nhưng nguồn cấp Vcc phải phục hồi 5V trước 10 chu kỳ khi tín hiệu RST được kích hoạt.

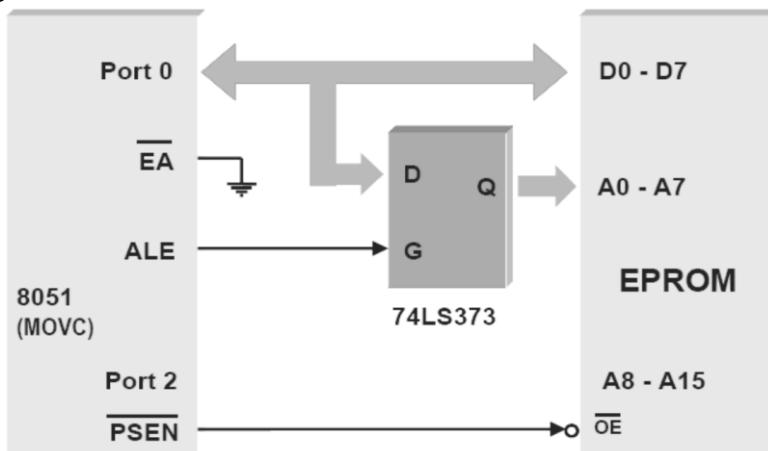
- + Bit IDL cho phép thiết lập chế độ rỗi cho vi điều khiển. Khi bit IDL được lập bằng 1, vi điều khiển sẽ vào hoạt động ở chế độ rỗi. Trong chế độ này xung nhịp bên trong sẽ bị cắt không cung cấp tới CPU, nhưng vẫn cung cấp tới bộ điều khiển ngắn, bộ định thời và công nối tiếp. Trạng thái hiện tại của CPU và các thanh ghi sẽ được giữ nguyên, các cổng vào ra cũng được giữ ở mức logic hiện tại, chân ALE và PSEN được giữ ở mức cao. Chế độ rỗi sẽ được thoát khi có yêu cầu ngắn, khi đó bit IDL sẽ được xóa.

2.4.2 Tổ chức bộ nhớ ngoài

Trong phần trên chúng ta đã tìm hiểu về bộ nhớ trong của vi điều khiển 8051 và thấy rằng bộ nhớ của nó rất ít: 128 byte RAM và 4KB ROM vì vậy nhu cầu mở rộng bộ nhớ ngoài để có thể nạp những chương trình lớn hơn hoặc lưu nhiều giá trị tính toán hơn là rất cần thiết. Vi điều khiển 8051 có khả năng mở rộng đến 64KB bộ nhớ chương trình và 64 KB bộ nhớ dữ liệu bên ngoài. Khi sử dụng bộ nhớ ngoài, cổng 0 sẽ không làm nhiệm vụ của một cổng vào/ra nữa mà được sử dụng như là một cổng đa hợp giữa phần thấp của bus địa chỉ (A0-A7) và 8 bit thấp của bus dữ liệu (D0-D7). Chân ALE sẽ làm nhiệm vụ chốt địa chỉ trên cổng 0 này, tín hiệu ALE sẽ tác động mức cao khi bắt đầu mỗi chu kỳ truy cập bộ nhớ để chốt 8 bit thấp của địa chỉ. Cổng 2 sẽ được sử dụng làm 8 bit cao của bus địa chỉ (A8-A15) khi giao tiếp với bộ nhớ ngoài.

2.4.2.1 Bộ nhớ chương trình

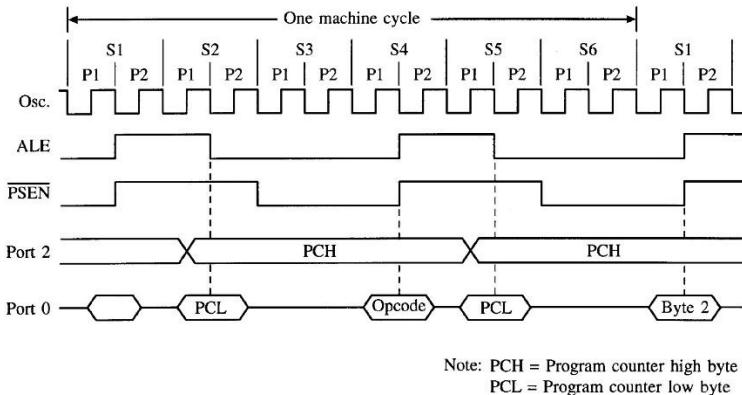
Bộ nhớ chương trình ngoài là một bộ nhớ chỉ đọc (ROM) được cho phép bởi tín hiệu PSEN. Sơ đồ kết nối bộ nhớ chương trình ngoài được mô tả trong Hình 2-10, ở đây sử dụng 74LS373 để chốt địa chỉ thấp của lệnh cần truy cập và chân ALE được nối với chân G của IC chốt. Trong quá trình truy cập bộ nhớ chương trình ngoài chân EA phải được nối với GND. Với bus địa chỉ 16 bit, vi điều khiển 8051 có thể giao tiếp với bộ nhớ chương trình ngoài tối đa lên đến 64KB.



Hình 2-10 Sơ đồ nguyên lý giao tiếp bộ nhớ chương trình ngoài

Trong một chu kỳ máy, vi điều khiển 8051 có thể đọc được 2 byte lệnh từ bộ nhớ chương trình ngoài vì tín hiệu ALE sẽ tích cực cao hai lần trong một chu kỳ. Giản đồ thời gian ở Hình 2-11 mô tả chu kỳ tìm nạp lệnh tại bộ nhớ chương trình ngoài của vi điều khiển 8051 trong một chu kỳ máy (12 chu kỳ giao động của thạch anh). Trong $\frac{1}{2}$ chu kỳ đầu của chu kỳ bộ nhớ (6 chu kỳ thạch anh), byte thấp của địa chỉ được xuất ra tại cổng 0 và được chốt vào IC chốt (74LS373) nhờ tín hiệu tại chân ALE. Trong $\frac{1}{2}$ chu kỳ sau của chu kỳ bộ nhớ, cổng 0 được sử dụng chuyển dũ

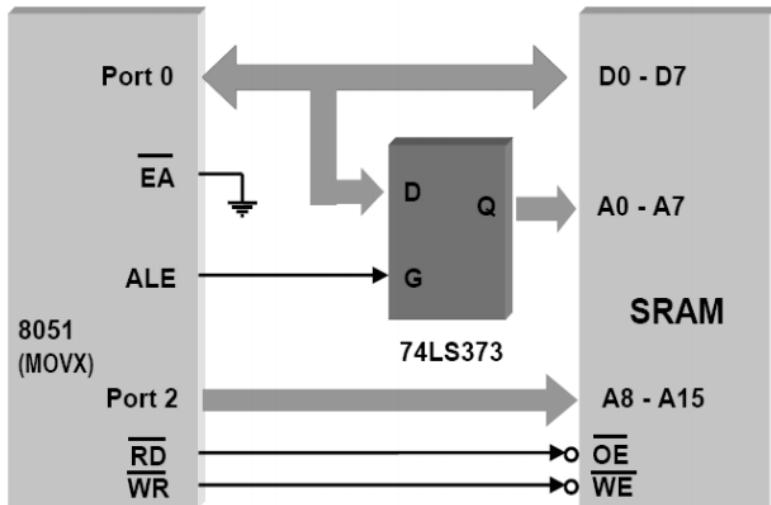
liệu vào vi điều khiển. Trong khi đó cổng 2 luôn luôn chứa byte cao của địa chỉ.



Hình 2-11 Giản đồ thời gian giao tiếp với bộ nhớ chương trình ngoài

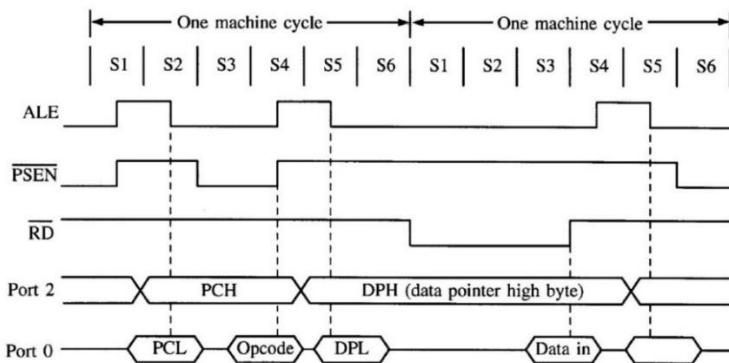
2.4.2.2 Bộ nhớ dữ liệu

Bộ nhớ dữ liệu ngoài là bộ nhớ đọc/ghi (RAM) được cho phép hoạt động bởi các tín hiệu RD và WR cở các chân P3.7 và chân P3.6 của vi điều khiển 8051. Để truy xuất bộ nhớ dữ liệu ngoài cần dùng lệnh MOVX với cách định địa chỉ gián tiếp, với thanh ghi chứa địa chỉ là thanh ghi con trỏ dữ liệu DPTR hoặc thanh ghi R0 hoặc R1. Hình 2-12 mô tả các kết nối bộ nhớ dữ liệu ngoài với vi điều khiển 8051, trong đó bus địa chỉ và bus dữ liệu cũng được kết nối tương tự như với bộ nhớ chương trình. Chỉ khác ở hai chân điều khiển, tín hiệu yêu cầu đọc RD của vi điều khiển 8051 được nối tới chân cho phép xuất OE của RAM, và tín hiệu yêu cầu ghi WR của 8051 được nối tới chân cho phép ghi của RAM.



Hình 2-12 Sơ đồ nguyên lý giao tiếp bộ nhớ dữ liệu ngoài

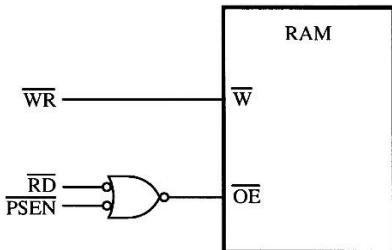
Giản đồ thời gian Hình 2-13 mô tả thao tác đọc dữ liệu ở bộ nhớ dữ liệu ngoài cho lệnh MOVX A, DPTR. Quá trình ghi dữ liệu vào bộ nhớ ngoài cũng tương tự ngoại trừ các xung WR ở mức thấp tại thời điểm ghi và dữ liệu được xuất ra ở cổng 0 (trong quá trình ghi RD phải ở mức cao).



Hình 2-13 Giản đồ thời gian giao tiếp bộ nhớ dữ liệu ngoài

2.4.2.3 Bộ nhớ dùng chung

Vì bộ nhớ chương trình là bộ nhớ chỉ đọc, nên khi phát triển phần mềm mỗi lần cần sửa đổi chương trình cần phải nạp lại chương trình từ đầu, điều này rất bất tiện và tốn thời gian. Để giải quyết việc này, chúng ta có thể sử dụng chung một vùng nhớ RAM ngoài cho cả chương trình và dữ liệu. Điều này có thể thực hiện bằng cách AND hai chân PSEN và RD của vi điều khiển 8051 và nối đầu ra của công AND vào chân OE của RAM. Lúc này một chương trình có thể nạp vào RAM bằng cách ghi vào bộ nhớ dữ liệu ngoài, và thực hiện bằng cách truy xuất RAM như bộ nhớ chương trình ngoài. Hình 2-14 mô tả cách sử dụng bộ nhớ ngoài chung cho dữ liệu và chương trình.



Hình 2-14 Sơ đồ nguyên lý giao tiếp bộ nhớ dùng chung ngoài

2.5 Tổng kết

Chương này đã trình bày kiến trúc tổng quan cũng như chi tiết các thành phần bên trong vi điều khiển 8051, từ chức năng bên ngoài đến các thành phần cấu tạo bên trong. Bên trong vi điều khiển cũng được trình bày chi tiết về cấu trúc bộ nhớ và các thanh ghi có chức năng đặc biệt và các nội dung trong thanh ghi đó. Một số phương pháp kết nối các ngoại vi đến vi điều khiển cũng được trình bày ở chương này.

2.6 Bài tập

1. Cấu trúc bên trong vi điều khiển 8051 gồm những thành phần nào?
2. Các thông số cơ bản của VĐK 8051?
3. VĐK 8051 có bao nhiêu chân? Bao nhiêu chân dùng cho nhập xuất?
4. Có bao nhiêu cổng (port)? Chức năng của các cổng?
5. Bộ nhớ của VĐK có các loại nào và dung lượng từng loại bao nhiêu?
6. Phân biệt bộ nhớ trong và bộ nhớ ngoài, bộ nhớ chương trình và bộ nhớ dữ liệu?
7. VĐK 8051 có bao nhiêu thanh ghi, độ dài và chức năng của từng loại?
8. Nêu cấu trúc bộ nhớ của vi điều khiển 8051?

Chương 3. Tập lệnh họ Vi điều khiển 8051

Mục tiêu chương

Chương này sẽ mô tả chi tiết cấu trúc tập lệnh của họ vi điều khiển 8051. Học xong chương này người học cần nắm rõ 8051 có các nhóm lệnh nào? Cách sử dụng các lệnh này trong các ứng dụng thực tế ra sao? Cách viết một chương trình đơn giản cho một hệ thống nhúng sử dụng vi điều khiển 8051.

3.1 Tổng quan

Chương này giới thiệu về tập lệnh của họ vi điều khiển 8051 cũng như các chế độ định vị địa chỉ trên tập lệnh này. Tập lệnh của 8051 được xây dựng phù hợp cho các ứng dụng điều khiển 8 bit, nó có các chế độ định địa chỉ cho phép việc thực hiện các phép toán, đọc/ghi vào bộ nhớ dữ liệu một cách nhanh chóng, thích hợp với các ứng dụng có cấu trúc dữ liệu nhỏ. Các lệnh của vi điều khiển 8051 có các opcode 8 bit, có thể định nghĩa tới 256 lệnh. Thực tế tập lệnh của 8051 có 255 lệnh, trong đó có 139 lệnh 1 byte, 92 lệnh 2 byte và 24 lệnh 3 byte. Ngoài các byte opcode các lệnh 2 hoặc 3 byte chứa thêm 1 hoặc 2 byte là byte địa chỉ hoặc dữ liệu của lệnh đó. Cụ thể các lệnh sẽ được trình bày thành các nhóm lệnh có chức năng tương tự nhau trong các phần tiếp theo.

3.2 Các chế độ định vị địa chỉ họ Vi điều khiển 8051

Định vị địa chỉ (định địa chỉ) trong họ vi điều khiển 8051 là xác định vị trí nơi chứa dữ liệu cần thao tác (trong thanh ghi, trong RAM, trong bộ nhớ ngoài) trong các lệnh xử lý dữ liệu của vi điều khiển 8051 (tính toán, di chuyển, ...). Các chế độ định địa chỉ là phần cần thiết cho toàn bộ tập lệnh của mỗi một

bộ vi điều khiển. Chúng cho phép xác định rõ nguồn và đích dữ liệu theo nhiều cách khác nhau phụ thuộc và các tình huống lập trình. Trong giới hạn của giáo trình này ta xem xét 5 kiểu định địa chỉ cơ bản thường được sử dụng trong lập trình đối với 8051.

3.2.1 Địa chỉ tức thời

Địa chỉ tức thời là cách định địa chỉ trong câu lệnh mà trong đó toán hạng nguồn là một hằng số, hằng số này nằm ngay trong mã lệnh như một byte dữ liệu tức thời. Byte dữ liệu tức thời này sẽ nằm ngay sau byte opcode của lệnh như trong Hình 3-1

Opcode	Giá trị tức thời
--------	------------------

Hình 3-1 Cấu trúc của lệnh định vị địa chỉ tức thời

Trong lệnh assembly, định vị địa chỉ tức thời được ký hiệu bằng dấu “#” nằm trước giá trị tức thời đó. Giá trị tức thời có thể là 1 số, một tên được định nghĩa trước hoặc 1 biểu thức số học biểu diễn bằng số, các tên, các hoạt động. Trình thông dịch sẽ tính toán giá trị và thay thế dữ liệu trực tiếp vào trong câu lệnh. Ví dụ lệnh MOV A, #25 sẽ nạp giá trị 25 (19H) vào thanh ghi A.

Hầu hết các lệnh định vị địa chỉ tức thời đều có dữ liệu tức thời 8 bit, trừ trường hợp ngoại lệ là nạp giá trị 16 bit vào thanh ghi DPTR. Ví dụ lệnh MOV DPTR, #6123H sẽ bao gồm 3 byte, nhằm nạp giá trị 16 bit (6123H) và thanh ghi con trỏ dữ liệu.

3.2.2 Địa chỉ theo thanh ghi

Trong cấu trúc các lệnh sử dụng chế độ định địa chỉ theo thanh ghi, các thanh ghi được mã hóa bằng 3 bit trọng số thấp nhất trong chuỗi bit của bit của mã lệnh. Như vậy mã chức năng và toán hạng của lệnh được kết hợp trong một byte như Hình 3-2

Opcode	Toán hạng 1	Toán hạng 2	Toán hạng 3
--------	-------------	-------------	-------------

Hình 3-2 Cấu trúc của lệnh định vị địa chỉ thanh ghi

Các thanh ghi được sử dụng trong chế độ định địa chỉ này là 32 thanh ghi 8 bit nằm ở 32 byte đầu tiên trong RAM nội của vi điều khiển 8051 từ địa chỉ từ 00H đến 1FH. Các thanh ghi này được chia thành 4 tập thanh ghi giống nhau, mỗi tập gồm 8 thanh ghi được ký hiệu từ R0 đến R7. Các tập thanh ghi được chọn bằng các bit PSW[3,4] của thanh ghi trạng thái chương trình (PSW).

3.2.3 Địa chỉ trực tiếp

Một lệnh sử dụng cách định vị địa chỉ trực tiếp cho phép truy xuất bất kỳ địa chỉ hoặc thanh ghi bất kỳ bên trong vi điều khiển. Trong kiểu định địa chỉ này, một byte địa chỉ trực tiếp được sử dụng sau opcode của lệnh nhị phân như mô tả trong hình Hình 3-3

Opcode	Địa chỉ trực tiếp
--------	-------------------

Hình 3-3 Cấu trúc của lệnh định vị địa chỉ trực tiếp

Để truy cập vùng nhớ thấp trong RAM có tầm từ 0 đến 127 (00H-7FH) thì bit có trọng số cao nhất (bit 7) có giá trị bằng 0. Khi bit 7 bằng 1, thì các thanh ghi có chức năng đặt biệt được truy cập với địa chỉ từ 128 đến 255 (80H-FFH). Ví dụ port0 và port1 được gán địa chỉ trực tiếp là 80H và 90H. Trong lập trình hợp ngữ ta cũng có thể sử dụng các mã gọi nhớ để thay thế cho địa chỉ trực tiếp này. Ví dụ hai lệnh sau đây là tương đương nhau:

MOV P0, A

MOV 80H, A

để chuyển nội dung của thanh ghi A vào port 0.

3.2.4 Địa chỉ gián tiếp

Trong cách định địa chỉ gián tiếp, chương trình sử dụng các thanh ghi để lưu địa chỉ của một ô nhớ chứa dữ liệu cần xử lý trong lệnh. Cách định địa chỉ này đặc biệt tiện lợi khi truy cập một chuỗi dữ liệu trong bộ nhớ, khi đó chỉ cần tăng hoặc giảm giá trị thanh ghi để trỏ tới dữ liệu kế tiếp của chuỗi dữ liệu.

Vì điều khiển 8051 chỉ sử dụng các thanh ghi R0 và R1 cho việc định vị địa chỉ gián tiếp. Bit ở vị trí thấp nhất trong mã lệnh sử dụng để xác định thanh ghi nào trong hai thanh ghi này được chọn để chứa địa chỉ của ô nhớ cần truy xuất.

Trong lập trình hợp ngữ, dấu '@' được đặt trước thanh ghi R0 hoặc R1 để mô tả cách định địa chỉ gián tiếp. Ví dụ nếu R0 chứa giá trị 60H và nội dung ô nhớ 60H là 35H, thì lệnh “*MOV A,@R0*” sẽ chuyển giá trị 35H vào thanh ghi A. Để sử dụng cách định địa chỉ gián tiếp này trong các đoạn chương trình xử lý các chuỗi dữ liệu trong bộ nhớ ta cần sử dụng các vòng lặp để tăng hoặc giảm các giá trị của thanh ghi R0 hoặc R1 này. Ví dụ để xóa các ô nhớ từ địa chỉ 60H đến 70H ta thực hiện đoạn chương trình sau:

```
MOV R0, #60H  
LOOP: MOV @R0,#0  
INC R0  
CJNE R0, #71H, LOOP
```

3.2.5 Địa chỉ chỉ số

Chế độ định địa chỉ chỉ số được sử dụng khi cần truy cập các dữ liệu trong không gian ROM của vi điều khiển 8051. Thanh ghi 16 bit DPTR và thanh ghi A được sử dụng để tạo ra địa chỉ của dữ liệu được lưu trong ROM, lệnh “*MOVC A, @A+DPTR*” được sử dụng cho cách định địa chỉ này. Ở lệnh này, nội dung của A được cộng với nội dung của thanh ghi 16bit DPTR để tạo ra địa chỉ 16 bit.

3.3 Các lệnh di chuyển dữ liệu

Các lệnh di chuyển dữ liệu cho phép sao chép dữ liệu từ một hàng số đến thanh ghi; từ một hàng số đến một ô nhớ; từ thanh ghi đến thanh ghi; từ thanh ghi đến ô nhớ hoặc ngược lại từ ô nhớ đến thanh ghi. Các lệnh này được chia làm 5 nhóm lệnh nhỏ:

- MOV: di chuyển dữ liệu trong vùng RAM nội hoặc các thanh ghi có chức năng đặc biệt
- MOVC: đọc giá trị bộ nhớ chương trình
- MOVX: đọc/ghi giá trị với bộ nhớ ngoài
- PUSH/POP: di chuyển dữ liệu vào ra vùng stack
- XCH: hoán đổi dữ liệu với giữa nguồn với thanh ghi A

Bảng 3-1 Các lệnh di chuyển dữ liệu

STT	Lệnh	Mô tả	Số byte	Số chu kỳ
1	MOV A, Rn	A = Rn	1	1
2	MOV A, direct	A = direct	2	2
3	MOV A, @Ri	A = @Ri	1	2
4	MOV A, #data	A = data	2	2
5	MOV Rn, A	Rn = A	1	2
6	MOV Rn, direct	Rn = direct	2	4
7	MOV Rn, #data	Rn = data	2	2
8	MOV direct, A	direct = A	2	3
9	MOV direct, Rn	direct = Rn	2	3
10	MOV direct1, direct2	direct1 = direct2	3	4
11	MOV direct, @Ri	direct = @Ri	2	4
12	MOV direct, #data	direct = data	3	3
13	MOV @Ri, A	@Ri = A	1	3
14	MOV @Ri, direct	@Ri = direct	2	5

15	MOV @Ri, #data	@Ri = data	2	3
16	MOV DPTR , #data16	DPTR = data16	3	3
17	MOVC A, @A+DPTR	Đọc giá trị bộ nhớ chương trình tại địa chỉ = A + DPTR, lưu vào thanh ghi A	1	3
18	MOVC A, @A + PC	Đọc giá trị bộ nhớ chương trình tại địa chỉ = A + PC, lưu vào thanh ghi A	1	3
19	MOVX A, @Ri	Đọc giá trị bộ nhớ ngoài tại địa chỉ = Ri, lưu vào thanh ghi A	1	3-10
20	MOVX A, @DPTR	Đọc giá trị bộ nhớ ngoài tại địa chỉ = DPTR, lưu vào thanh ghi A	1	3-10
21	MOVX @Ri, A	Ghi giá trị của thành ghi A vào bộ nhớ ngoài tại địa chỉ = Ri	1	4-11
22	MOVX @DPTR, A	Ghi giá trị của thành ghi A vào bộ nhớ ngoài tại địa chỉ = DPTR	1	4-11
23	PUSH direct	Lưu nội dung tại direct vào đỉnh ngăn xếp	2	4
24	POP direct	Lấy nội dung ở đỉnh ngăn xếp ghi vào direct	2	3
25	XCH A, Rn	Hoán đổi A và Rn	1	2
26	XCH A, direct	Hoán đổi A và direct	2	3
27	XCH A, @Ri	Hoán đổi A và @Ri	1	3

28	XCHD A, @Ri	Hoán đổi 4 bit thấp giữa thanh ghi A và nội dung bộ nhớ tại địa chỉ được lưu trong Ri	1	3
----	-------------	---	---	---

Ví dụ về các lệnh di chuyển dữ liệu:

- Lệnh MOV A, R1 ;di chuyển dữ liệu thanh thi trong R1 vào thanh ghi A
- Lệnh MOV R2, #58H ;di chuyển giá trị tức thời 58H vào thanh ghi R2
- Lệnh PUSH 2; di chuyển nội dung trong thanh ghi R2 vào đinh ngăn xếp

3.4 Các lệnh toán học

Các lệnh toán học dùng để thực hiện các phép tính cơ bản như +, -, *, /, tăng, giảm, ...kết quả sau khi thực hiện được lưu vào toán hạng đầu tiên trong lệnh. Các lệnh toán học có 4 cách để định vị địa chỉ gồm: trực tiếp, gián tiếp, thanh ghi, và tức thời. Phần lớn các lệnh số học được thực thi trong 1 chu kỳ máy trừ lệnh “INC DPTR” được thực thi trong 2 chu kỳ máy, các lệnh “MUL AB” và “DIV AB” được thực thi trong 4 chu kỳ máy (12 chu kỳ xung clock). Sau đây, chúng ta sẽ xem xét một số lệnh toán học đặt trung của vi điều khiển 8051.

Với lệnh INC vi điều khiển 8051 cung cấp nhiều kiểu định địa linh hoạt trong không gian bộ nhớ nội. Chúng ta có thể dùng một câu lệnh INC để tăng giá trị tại một ô nhớ trong RAM bằng cách định địa chỉ trực tiếp. Ví dụ, tại vị trí ô nhớ có địa chỉ 1EH của RAM đang chứa giá trị 32H thì lệnh “ INC 1EH sẽ tăng giá trị chức tại ô nhớ này thành 33H.

Với lệnh MUL AB sẽ nhân dữ liệu 8 bit chứa trong thanh ghi A với dữ liệu 8 bit chứa trong thanh ghi B và sẽ đưa kết quả

16 bit của phép nhân vào cặp thanh ghi BA (thanh ghi A chứa byte thấp, thanh ghi B chứa byte cao).

Với lệnh DIV AB sẽ lấy dữ liệu chứa thanh ghi A chia cho dữ liệu chứa trong thanh ghi B, thương của phép chia sẽ đưa vào thanh ghi A, số dư của phép chia sẽ được đưa vào thanh ghi B.

Với các phép toán số học đổi với số BCD, sau các lệnh toán học phải là lệnh DA A để hiệu chỉnh kết quả. Lệnh này không biến đổi số chứa trong thanh ghi A thành số BCD mà chỉ tạo ra một kết quả hợp lệ khi thao tác toán học trên số BCD. Ví dụ nếu thanh ghi A chứa giá trị BCD là 28 thì lệnh “ADD A, #2” sẽ có kết quả là thanh ghi A sẽ chứa giá trị 2A, không nằm trong tâm số của BCD vì vậy cần có lệnh “DA A” tiếp sau đó để hiệu chỉnh lại thành 30.

Bảng 3-2 Cách lệnh số học

ST T	Lệnh	Mô tả	Số byte	Số chu kỳ
1	ADD A, Rn	$A = A + Rn$	1	1
2	ADD A, direct	$A = A + direct$	2	1
3	ADD A, @Ri	$A = A + @Ri$	1	1
4	ADD A, #data	$A = A + data$	2	1
5	ADDC A, Rn	$A = A + Rn + C$	1	1
6	ADDC A, direct	$A = A + direct + C$	2	1
7	ADDC A, @Ri	$A = A + @Ri + C$	1	1
8	ADDC A, #data	$A = A + data + C$	2	1
9	SUBB A, Rn	$A = A - Rn - C$	1	1
10	SUBB A, direct	$A = A - direct - C$	2	1
11	SUBB A, @Ri	$A = A - @Ri - C$	1	1
12	SUBB A, #data	$A = A - #data - C$	2	1
13	INC A	$A = A + 1$	1	1
14	INC Rn	$Rn = Rn + 1$	1	1

15	INC direct	direct = direct + 1	2	1
16	INC @Ri	@Ri = @Ri + 1	1	1
17	INC DPTR	DPTR = DPTR + 1		
18	DEC A	A = A - 1	1	1
19	DEC Rn	Rn = Rn - 1	1	1
20	DEC direct	direct = direct - 1	2	1
21	DEC @Ri	@Ri = @Ri - 1	1	1
22	MUL AB	B:A = A*B	1	4
23	DIV AB	A = thương A/B B = phần dư A/B	1	4
24	DA A	Hiệu chỉnh số BCD trong thanh ghi kết quả A sau phép toán trên số BCD	1	1

3.5 Các lệnh rẽ nhánh

Các lệnh rẽ nhánh trong tập lệnh của vi điều khiển 8051 được chia làm 2 nhóm dựa vào chức năng của nó là:

- Lệnh rẽ nhánh không điều kiện: khi thực thi chúng chương trình sẽ nhảy đến vị trí bất kỳ mà lệnh qui định và thực hiện lệnh tiếp theo mà chương trình muốn thực thi
- Lệnh rẽ nhánh có điều kiện: khi lệnh này được thực thi thì chương trình sẽ chỉ nhảy tới vị trí thực thi mới khi điều kiện trong câu lệnh được thỏa mãn. Nếu điều kiện không được thỏa, thì câu lệnh sẽ bị bỏ qua và lệnh tiếp theo của câu lệnh sẽ nhánh sẽ được thực thi.

Bảng 3-3 Các câu lệnh rẽ nhánh

S T T	Lệnh	Mô tả	Số byte	Số ch u kỳ
1	ACALL <addr11>	Gọi chương trình con nằm trong phạm vi 2KB	3	2
2	LCALL <add16>	Gọi chương trình con nằm trong phạm vi 64KB	3	2
3	RET	Thoát khỏi chương trình con	1	2
4	RETI	Thoát khỏi chương trình con phục vụ ngắt	1	2
5	AJMP <addr11>	Nhảy đến nhãn trong phạm vi 2KB	2	2
6	LJMP <addr16>	Nhảy đến nhãn trong phạm vi 64KB	3	2
7	SJMP <rel>	Nhảy đến nhãn	2	2
8	JMP @A + DPTR	Nhảy đến địa chỉ A + DPTR	1	2
9	JZ <rel>	Nhảy đến nhãn nếu A = 0	2	2
10	JNZ <rel>	Nhảy đến nhãn nếu A ≠ 0	2	2
11	CJNE A, direct, <rel>	So sánh và nhảy đến nhãn nếu A ≠ direct	3	2
12	CJNE A, #data, <rel>	So sánh và nhảy đến nhãn nếu A ≠ data	3	2
13	CJNE Rn, #data, <rel>	So sánh và nhảy đến nhãn nếu Rn ≠ data	3	2
14	CJNE @Ri, #data, <rel>	So sánh và nhảy đến nhãn nếu nội dung bộ nhớ tại Ri ≠ data	3	2
15	DJNZ Rn, <rel>	Giảm Rn đi 1 đơn vị và	2	2

5		nhảy đến nhãn nếu Rn # 0		
1 6	DJNZ direct, <rel>	Giảm direct đi 1 đơn vị và nhảy đến nhãn nếu giá trị tại direct # 0	3	2
1 7	NOP	Lệnh này ko làm gì, thường dùng trong hàm delay	1	1

3.6 Các lệnh xử lý theo bit

Để hỗ trợ việc xử lý các bit trong quá trình lập trình, tập lệnh của vi điều khiển bao gồm các lệnh để di chuyển, thiết lập, xóa, lấy bù, OR, AND và các lệnh rẽ nhánh trên từng bit riêng lẻ. Các lệnh xử lý theo bit chỉ sử dụng kiểu định địa chỉ trực tiếp với các địa chỉ theo bit từ 00H đến 7FH tức là từ địa chỉ theo byte 20H đến 2FH của vùng nhớ dữ liệu có thể định địa chỉ theo bit trong RAM. Ngoài ra, các lệnh xử lý theo bit cũng được sử dụng cho các bit tại các thanh ghi có chức năng đặc biệt có định địa chỉ theo bit như thanh ghi A, IP, IE, P0, P1, P2, P3, PSW, ...

Những lệnh xử lý bit trên các thanh ghi đặc biệt có thể sử dụng địa chỉ trực tiếp hoặc thông qua các ký hiệu gọi nhớ. Ví dụ để thiết lập cờ nhớ C ta có thể sử dụng các lệnh sau:

SETB C

SETB CY

SETB 0D7H

Cụ thể các lệnh xử lý theo bit được mô tả trong bản bên dưới:

Bảng 3-4 Các lệnh xử lý theo bit

STT	Lệnh	Mô tả	Số byte	Số chu kỳ
1	CLR C	Xóa cờ nhớ	1	1
2	CLR bit	Xóa bit theo địa chỉ trực tiếp	2	3
3	SETB C	Thiết lập cờ nhớ	1	1
4	SETB bit	Thiết lập bit theo địa chỉ trực tiếp	2	3
5	CPL C	Lấy bù cờ nhờ	1	1
6	CPL bit	Lấy bù bit theo địa chỉ trực tiếp	2	3
7	ANL C, bit	AND bit trực tiếp với cờ nhớ	2	2
8	ANL C, /bit	AND bù bit trực tiếp với cờ nhớ	2	2
9	ORL C, bit	OR bit trực tiếp với cờ nhớ	2	2
10	ORL C, /bit	OR bù bit trực tiếp với cờ nhớ	2	2
11	MOV C, bit	Di chuyển bit trực tiếp tới cờ nhớ	2	2
12	MOV bit, C	Di chuyển cờ nhớ tới bit trực tiếp	2	3
13	JC <rel>	Nhảy đến nhãn <rel> nếu C = 1	2	2
14	JNC <rel>	Nhảy đến nhãn <rel> nếu C # 1	3	2
15	JB bit, <rel>	Nhảy đến nhãn <rel> nếu bit = 1	3	2
16	JNB bit, <rel>	Nhảy đến nhãn <rel> nếu bit # 1	3	2
17	JBC bit, <rel>	Nhảy đến nhãn <rel> nếu	3	2

		bit = 1 và sau đó xóa bit về 0		
--	--	-----------------------------------	--	--

3.7 Các lệnh logic

Nhóm các lệnh logic của vi điều khiển 8051 thực hiện các phép toán logic trên các byte dữ liệu hoặc các bit dữ liệu cụ thể tại các vị trí ô nhớ cho phép định địa chỉ theo bit. Đôi với các lệnh có toán hạng là thanh ghi A thì được thực thi trong một chu kỳ máy, ngược lại các toán hạng không phải là thanh ghi A thì sẽ thực thi trong hai chu kỳ máy. Các lệnh logic chủ yếu thực hiện cho các phép toán: AND, OR, XOR, NOT, các lệnh quay và lệnh hóa đổi.

Bảng 3-5 Các lệnh logic

STT	Lệnh	Mô tả	Số byte	Số chu kỳ
1	ANL A, Rn	$A = A \text{ and } Rn$	1	1
2	ANL A, direct	$A = A \text{ and } \text{direct}$	2	1
3	ANL A, @Ri	$A = A \text{ and } @R1$	1	1
4	ANL A, #data	$A = A \text{ and } \text{data}$	2	1
5	ANL direct, A	$\text{direct} = \text{direct and } A$	2	1
6	ANL direct, #data	$A = \text{direct and } \text{data}$	3	2
7	ORL A, Rn	$A = A \text{ or } Rn$	1	1
8	ORL A, direct	$A = A \text{ or } \text{direct}$	2	1
9	ORL A, @Ri	$A = A \text{ or } @R1$	1	1
10	ORL A, #data	$A = A \text{ or } \text{data}$	2	1
11	ORL direct, A	$\text{direct} = \text{direct or } A$	2	1
12	ORL direct, #data	$A = \text{direct or } \text{data}$	3	2
13	XRL A, Rn	$A = A \text{ xor } Rn$	1	1
14	XRL A, direct	$A = A \text{ xor } \text{direct}$	2	1

15	XRL A, @Ri	$A = A \text{ xor } @R1$	1	1
16	XRL A, #data	$A = A \text{ xor } \text{data}$	2	1
17	XRL direct, A	$\text{direct} = \text{direct xor } A$	2	1
18	XRL direct, #data	$A = \text{direct xor } \text{data}$	3	2
19	CLR A	$A = 0$	1	1
20	CPL A	$A = \text{NOT } A$	1	1
21	RL A	Quay trái A	1	1
22	RLC A	Quay trái A qua cờ C	1	1
23	RR A	Quay phải A	1	1
24	RRC	Quay phải A qua cờ C	1	1
25	SWAP A	Hoán đổi 1 nửa của A	1	1

3.8 Ví dụ

Viết chương trình lưu thanh ghi R1, R2 và B vào ngăn xếp và sau đó lấy ra và đưa vào các thanh ghi R3, R4, R5.

Chương trình mẫu:

PUSH R1	;lưu thanh ghi R1 vào đỉnh ngăn xếp
PUSH R2	;lưu thanh ghi R2 vào đỉnh ngăn xếp
PUSH OF0H	;lưu thanh ghi B vào đỉnh ngăn xếp

POP R3	; lấy đỉnh ngăn xếp đưa vào thanh ghi R3 ;sau khi thực hiện lệnh này R3 = B
POP R4	; lấy đỉnh ngăn xếp đưa vào thanh ghi R3 ;sau khi thực hiện lệnh này R4 = R2
POP R5	; lấy đỉnh ngăn xếp đưa vào thanh ghi R3 ;sau khi thực hiện lệnh này R5 = R1

3.9 Tổng kết

Nội dung chương 3 trình bày cụ thể về các kiểu định địa chỉ được sử dụng trong lập trình ASM cho vi điều khiển 8051 bao gồm các kiểu định địa chỉ: tức thời, theo thanh ghi, trực tiếp, gián tiếp qua thanh ghi và chỉ số. Mỗi kiểu định địa chỉ được sử dụng trong từng tình huống và có phạm vi hoạt động khác nhau, kết hợp với các lệnh gọi nhớ của vi điều khiển 8051 để tạo ra thư việc các câu lệnh phong phú trong quá trình lập trình. Các lệnh cụ thể cũng được trình bày trong chương này, được chia thành các nhóm lệnh theo chức năng của các câu lệnh bao gồm: các lệnh di chuyển dữ liệu, các lệnh toán học, các lệnh rẽ nhánh, các lệnh xử lý theo bit và các lệnh logic.

3.10 Bài tập

1. Viết đoạn chương trình thực hiện việc AND giá trị tại ô nhớ 50H với giá trị tại ô nhớ 45H và lưu kết quả vào địa chỉ 50H bằng ASM trên 8051
2. Viết đoạn chương trình lưu giá trị đọc được tại cổng P2 vào RAM ngoài tại địa chỉ 2A54H bằng ASM trên 8051
3. Viết chương trình xóa thanh ghi A sau đó cộng 5 vào thanh ghi A 20 lần.
4. Một vi điều khiển 8051 có tần số dao động thạch anh là 12MHz. Hãy tìm thời gian cần thiết để thực thi các câu lệnh sau đây:
 - a. MOV A, R2
 - b. MUL AB
 - c. DJNZ R1, HERE
 - d. ACALL AGAIN
 - e. RETI
 - f. NOP

5. Viết chương trình để lưu giá trị lần lượt từ 23H đến 26H vào các ô nhớ trong RAM tại các địa chỉ lần lượt từ 63H đến 66H sử dụng các chế độ định địa chỉ:
 - a. Tức thời
 - b. Trực tiếp
 - c. Gián tiếp không dùng vòng lặp
 - d. Gián tiếp dùng vòng lặp
6. Viết chương trình xóa 25 ô nhớ trong RAM bắt đầu từ địa chỉ 30H đến 54H
7. Giả sử chuỗi ký tự “KHOA KTMT” được lưu trong bộ nhớ ROM bắt đầu từ địa chỉ 50H. Hãy viết chương trình đọc chuỗi ký tự này và lưu vào bộ nhớ RAM bắt đầu từ địa chỉ 60H.
8. Viết chương trình cộng 2 số 16 bit được lưu lần lượt từ ô nhớ 14H đến 17H trong bộ nhớ RAM và lưu kết quả vào thanh ghi R2 và R3.

Chương 4. BỘ ĐỊNH THỜI/ BỘ ĐÉM

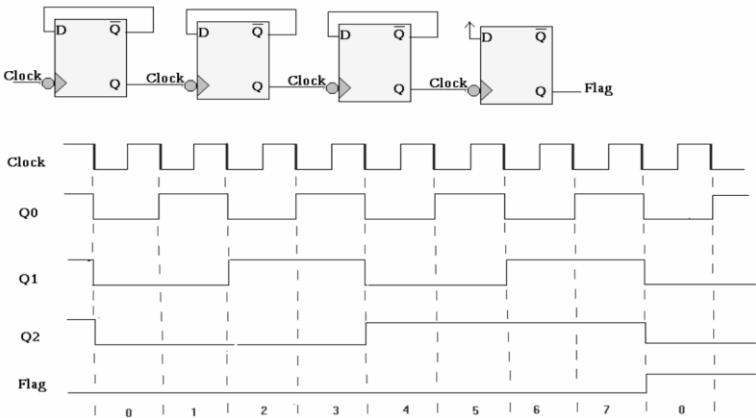
Mục tiêu chương

Chương này sẽ trình bày về khái niệm, hoạt động và các yếu tố liên quan tới bộ định thời và bộ đếm của vi điều khiển họ 8051. Học xong chương này người học có thể hiểu thế nào là bộ định thời, cách dùng bộ định thời trong việc tạo thời gian trễ hoặc sử dụng cho việc đếm sản phẩm

4.1 Giới thiệu

Bộ định thời bên trong vi điều khiển là một bộ chia tần số được tạo thành từ chuỗi các flip-flop D nối tiếp với nhau, mỗi flip-flop là một bộ chia 2. Tín hiệu xung nhịp (clock) được đưa vào chân clock của flip-flop thứ nhất, ngõ ra Q của flip-flop thứ nhất được dùng làm ngõ vào chân clock của flip-flop thứ 2, tại đây tần số clock chỉ bằng $\frac{1}{2}$ tần số của clock đầu vào timer. Mỗi tầng kế tiếp nhau tín hiệu clock được chia cho 2 nên bộ định thời có n tầng (bit) sẽ chia tần số xung clock ở ngõ vào cho 2^n . Ngõ ra của flip-flop ở tầng cuối làm ngõ vào clock cho flip-flop báo tràn của timer (cờ tràn của timer), cờ tràn này có thể được kiểm tra bằng phần mềm trong lúc lập trình hay để tạo ngắt.

Trong Hình 4-1 trình bày một ví dụ về bộ định thời 3 bit. Ở mỗi tầng là một flip-flop D nối tiếp với nhau. Cờ tràn Flag là một flip-flop D được nối ngõ vào D vào 1, nó sẽ tăng lên 1 sau khi clock đầu vào đếm được $2^3 = 8$ chu kỳ.



Hình 4-1 Hoạt động của timer 3 bit

Giá trị nhị phân trong các flip-flop của timer thay đổi như trong một bộ đếm nhị phân, nó sử dụng cho việc đếm xung clock (hay các sự kiện) trong vi điều khiển từ thời điểm timer được khởi động. Ví dụ timer 8 bit sẽ đếm lên từ 00H đến FFH, và cờ tràn sẽ lên 1 khi số đếm tràn từ FFH xuống 00H.

4.2 Bộ định thời 8051/8052

Trong vi điều khiển 8051 có hai timer 16 bit: timer 0 và timer1; mỗi timer có bốn chế độ hoạt động. Hai timer này thường được dùng làm bộ định thời để tạo thời gian trễ (delay), làm bộ đếm các sự kiện xảy ra bên ngoài vi điều khiển hoặc dùng để tạo tốc độ baud cho cổng nối tiếp (UART). Trong vi điều khiển 8052 còn chứa thêm 1 bộ định thời thứ 3, timer 2 cũng có các chức năng tương tự timer 0 và 1. Phần tiếp theo sẽ trình bày về các thanh ghi bên trong vi điều khiển được sử dụng cho các bộ time này.

4.3 Thanh ghi chức năng đặc biệt cho bộ định thời

Các bộ định thời của vi điều khiển 8051 được điều khiển bằng cách sử dụng 6 thanh ghi chức năng đặc biệt trong Bảng 4-1

Bảng 4-1 Các thanh ghi sử dụng trong việc định thời của 8051

SFR	Chức năng	Địa chỉ	Địa chỉ theo bit
TCON	Điều khiển	88H	Có
TMOD	Chọn chế độ	89H	Không
TL0	Byte thấp của Timer0	8AH	Không
TL1	Byte thấp của Timer1	8BH	Không
TH0	Byte cao của Timer0	8CH	Không
TH1	Byte cao của Timer1	8DH	Không

4.3.1 Thanh ghi điều khiển Timer (TCON)

Thanh ghi TCON là một thanh ghi 8 bit và có thể định địa chỉ theo bit. Nó chứa các bit điều khiển và trạng thái của bộ định thời 0 và bộ định thời 1, và các bit dùng cho ngắt như mô tả trong Bảng 4-2. Trong đó, chỉ có bốn bit cao (TCON.7-TCON.4) được dùng để điều khiển các bộ định thời hoạt động hoặc chứa các cờ tràn. Bốn bit thấp của TCON (TCON.3-TCON.0) được sử dụng để phát hiện và khởi động các ngắt.

Bảng 4-2 Các bit chức năng trong thanh ghi điều khiển TCON

BIT	Tên	Địa chỉ	Chức năng
7	TF1	8FH	Cờ báo tràn của Timer1, được lập bởi phần cứng khi có tràn Timer1, được xóa bởi phần mềm, hoặc bởi phần cứng khi phục vụ chương trình ngắt

6	TR1	8EH	Bit điều khiển hoạt động của Timer1, được lập hoặc xóa bởi phần mềm để điều khiển Timer1 hoạt động hay ngưng hoạt động
5	TF0	8DH	Cờ báo tràn của Timer0
4	TR0	8CH	Bit điều khiển hoạt động của Timer1
3	IE1	8BH	Được sử dụng trong phục vụ ngắn
2	IT1	8AH	
1	IE0	89H	
0	IT0	88H	

4.3.2 Thanh ghi chế độ định thời (TMOD)

Thanh ghi TMOD dùng để thiết lập chế độ hoạt động cho bộ định thời 0 và bộ định thời 1. Thanh ghi này là thanh ghi 8 bit, không được định địa chỉ theo bit và được nạp một lần bởi phần mềm ở thời điểm bắt đầu của một chương trình để khởi động chế độ hoạt động của bộ định thời. Trong đó 4 bit thấp dành cho bộ Timer 0 và 4 bit cao dành cho bộ Timer 1, mỗi bộ gồm 2 bit dành cho việc thiết lập chế độ định thời, còn 2 bit dùng để chọn chức năng đếm hoặc định thời.

Cụ thể chức năng của từng bit của thanh ghi TMOD được mô tả trong Bảng 4-3

Bảng 4-3 Các bit chức năng trong thanh ghi điều khiển TMOD

BIT	Tên	Timer	Chức năng
7	GATE	1	Bit điều khiển cỗng. Khi GATE = 1, Timer chỉ hoạt động khi INT1 ở mức cao.
6	C/T'	1	Bit chọn chức năng đếm hoặc định thời: 1 – đếm sự kiện; 0 – định thời
5	M1	1	Bit chọn chế độ cho Timer1

4	M0	1	Bit chọn chế độ cho Timer1
3	GATE	0	Bit điều khiển cỗng. Khi GATE = 1, Timer chỉ hoạt động khi INT1 ở mức cao.
2	C/T'	0	Bit chọn chức năng đếm hoặc định thời: 1 – đếm sự kiện; 0 – định thời
1	M1	0	Bit chọn chế độ cho Timer0
0	M0	0	Bit chọn chế độ cho Timer0

Mỗi Timer có bốn chế độ hoạt động và chế độ hoạt động sẽ được chọn phụ thuộc vào giá trị của các bit M1 và M0 tương ứng với mỗi Timer theo Bảng 4-4

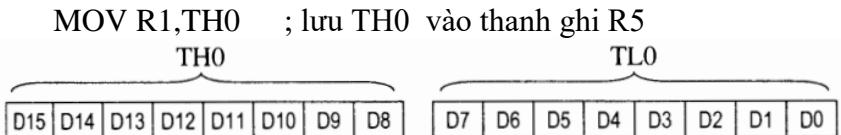
Bảng 4-4 Các mode định thời trong 8051

M1	M0	Chế độ	Mô tả
0	0	0	Chế độ định thời 13 bit
0	1	1	Chế độ định thời 16 bit
1	0	2	Chế độ định thời tự động nạp lại 8 bit
1	1	3	Chế độ định thời chia tách: Timer 0: TL0 là bộ định thời 8 bit điều khiển bởi chế độ của Timer 0; TH0 là bộ định thời 8 bit điều khiển bởi chế độ của Timer 1 Timer 1: ngưng hoạt động

4.3.3 Thanh ghi của bộ định thời 0

Thanh ghi 16 bit của bộ định thời 0 được truy cập theo 2 byte là byte thấp và byte cao. Thanh ghi byte thấp được gọi là TL0 và thanh ghi byte cao là TH0. Các thanh ghi này có thể được truy cập như mọi thanh ghi khác, ví dụ:

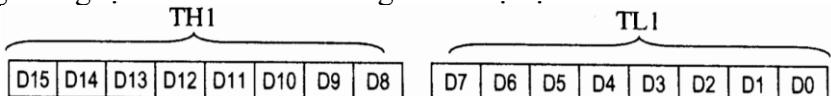
MOV TL0, #32H ; chuyển giá trị 32H vào TL0



Hình 4-2 Thanh ghi bộ định thời 0

4.3.4 Thanh ghi của bộ định thời 1

Các thanh ghi của bộ định thời 1 cũng được cấu tạo và hoạt động tương tự như các thanh ghi của bộ định thời 0.



Hình 4-3 Thanh ghi bộ định thời 1

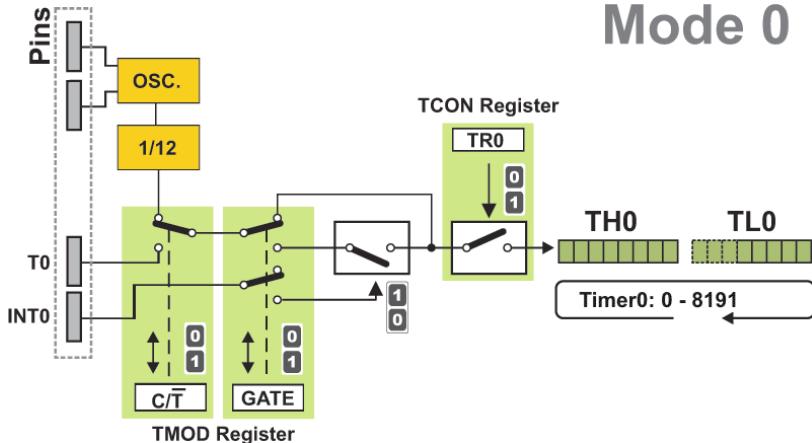
4.4 Các chế độ định thời

Các Timer 0 và 1 trong 8051 có các chế độ hoạt động tương đương nhau, nên trong phần này tác giả sử dụng ký hiệu “x” để chỉ hoặc Timer 0 hoặc Timer 1. Ví dụ THx có thể là TH0 hoặc TH1. Phần này sẽ trình bày rõ hơn về các chế độ định thời bao gồm cách thức cài đặt và cách thức hoạt động.

4.4.1 Chế độ 0

Chế độ định thời 0 là chế độ định thời 13 bit, chế độ này được thiết kế nhằm cung cấp khả năng tương thích với bộ vi điều khiển trước là 8048. Chế độ này hoạt động bằng cách ghép 5 bit thấp của thanh ghi TLx và thanh ghi THx để tạo thành bộ định thời 13 bit, 3 bit cao của thanh ghi TLx không được sử dụng. Sơ đồ hoạt động của chế độ định thời 0 được mô tả như hình ...

Mode 0

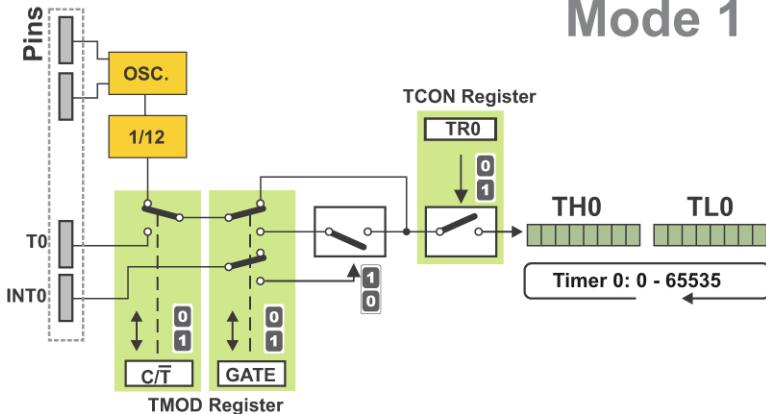


Hình 4-4 Mô hình chế độ 0 của bộ định thời

4.4.2 Chế độ 1

Chế độ định thời 1 là chế độ định thời 16 bit, hoạt động tương tự như chế độ 0. Khi nhận được xung nhịp bộ định thời sẽ đếm lên từ giá trị chưa bên trong thanh ghi định thời, nếu thanh ghi định thời chưa được nạp thì bộ định thời sẽ đếm lên từ 0000H. Cờ tràn sẽ bật lên 1 khi bộ định thời chuyển trạng thái từ FFFFH về 0000H, và bộ định thời sẽ tiếp tục đếm. Cờ tràn là bit TFx trong thanh ghi TCON có thể đọc hoặc ghi bằng phần mềm. Sơ đồ hoạt động của bộ định thời 1 được mô tả trong hình ...

Mode 1



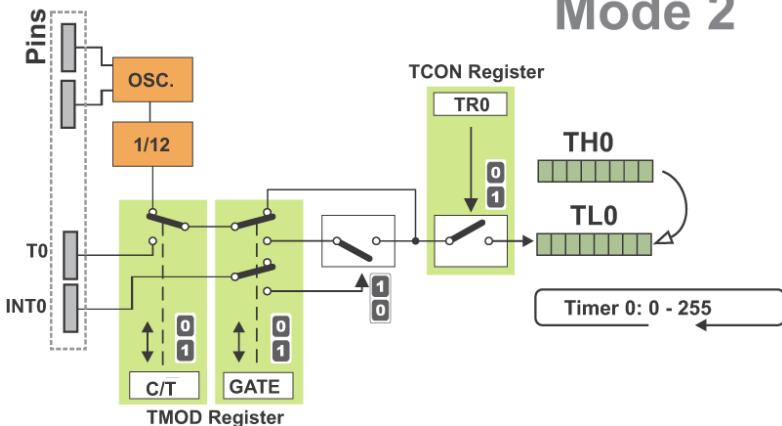
Hình 4-5 Mô hình chế độ 1 của bộ định thời

Bit có trọng số lớn nhất (MSB) của giá trị trong các thanh ghi định thời là bit 7 của THx và bit có trọng số thấp nhất (LSB) là bit 0 của thanh ghi TLx. Giá trị bắt đầu của bộ định thời được nạp vào các thanh ghi TLx và THx trước khi cho phép bộ định thời hoạt động. THx và TLx có thể đọc và ghi bằng phần mềm.

4.4.3 Chế độ 2

Chế độ định thời 2 là chế độ định thời tự nạp lại 8 bit, trong chế độ này TLx hoạt động như một timer 8 bit, còn THx vẫn giữ nguyên giá trị bắt đầu của đếm. Và sẽ nạp lại giá trị từ THx và TLx mỗi khi bộ định thời tràn từ FFH đến 00H, đồng thời bật cờ tràn TFX tại thời điểm này. Chế này được sử dụng thường xuyên bởi vì bộ định thời sẽ tự động hoạt động sau khi cài đặt các thanh ghi TMOD và THx mà không cần phải nạp giá trị vào các thanh ghi định thời mỗi khi cờ tràn bật lên. Sơ đồ hoạt động của chế độ định thời 2 được mô tả như trong hình ...

Mode 2



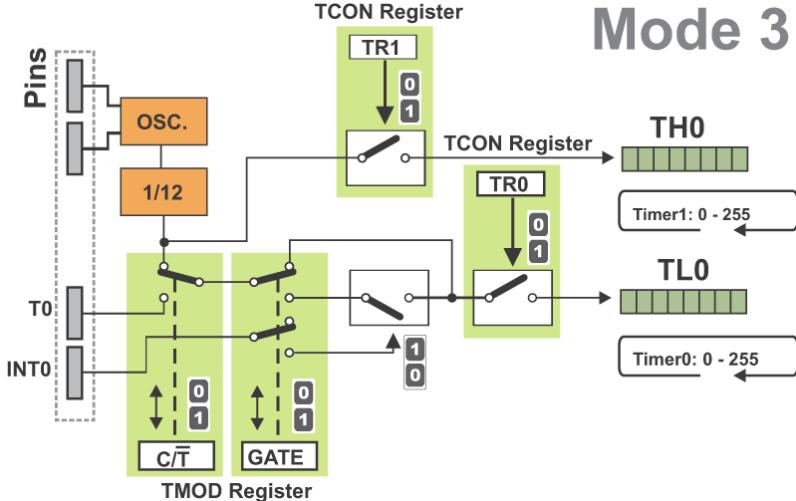
Hình 4-6 Mô hình chế độ 2 của bộ định thời

4.4.4 Chế độ 3

Chế độ định thời 3 là chế độ định thời chia sẻ và trong chế độ này 2 bộ định thời trong 8051 có hoạt động khác nhau. Bộ định thời 0 ở chế độ 3 được chia thành 2 bộ định thời 8 bit hoạt động độc lập với nhau TL0 có cờ báo tràn là TF0, còn TH0 có cờ báo tràn là TF1. Khi bộ định thời 0 ở chế độ 3 thì bộ định thời 1 không hoạt động định thời bởi vì cờ tràn TF1 đã được nối với TH0, nhưng có thể được khởi động bằng thiết lập bit TR1 bằng 1 trong chế độ khác chế độ 3.

Chế độ định thời 3 nhằm cung cấp thêm một bộ định thời 8 bit nữa cho 8051. Khi đó 8051 có 3 bộ định thời, bộ định thời 0 ở chế độ 3 hoạt động như 2 bộ định thời, và bộ định thời 1 không hoạt động định thời nên có thể sử dụng để tạo tốc độ baud trong giao tiếp cổng nối tiếp (UART) hoặc sử dụng như một bộ đếm thông thường mà không yêu cầu ngắn. Sơ đồ hoạt động của chế độ định thời 3 được mô tả trong hình..

Mode 3



Hình 4-7 Mô hình chế độ 3 của bộ định thời

4.5 Lập trình sử dụng các bộ định thời

Trong các chế độ định thời thì chế độ 1 và chế độ 2 là hai chế được thường được sử dụng trong các tinh huân lập trình vì vậy trong phần này sẽ trình bày rõ hơn về cách lập trình sử dụng các bộ định thời trong 2 chế độ này.

4.5.1 Lập trình chế độ 1

Chế độ định thời 1 có những đặt trung và hoạt động như sau:

- Là bộ định thời 16 bit, do vậy giá trị có thể được nạp vào các thanh ghi định thời TLx và THx của bộ định thời là từ 0000H đến FFFFH.
- Sau khi TL và TH được nạp giá trị ban đầu 16 bit thì bộ định thời được khởi động bằng các lệnh “SETB TR0” cho bộ định thời 0 và “STETB TR1” cho bộ định thời 1.

- Bộ định thời sau khi được khởi động thì bắt đầu tiến hành đếm tăng một đơn vị sau mỗi chu kỳ của bộ định thời. Khi giá trị bộ đếm đạt giá trị FFFFH thì bộ định thời sẽ quay vòng về 0000H đồng thời bật cờ bộ định thời TFx lên mức cao (TF0 cho bộ định thời 0 và TF1 cho bộ định thời 1).
- Nếu không sử dụng ngắt thì khi cờ bộ định thời được thiết lập thì có thể kiểm tra bằng phần mềm để thực hiện câu lệnh nhảy tới hàm con tương ứng và cho dừng bộ định thời bằng phần mềm sử dụng lệnh “CLR TR0”.
- Nếu sử dụng ngắt cho bộ định thời thì khi cờ bộ định thời được bật, chương trình sẽ tự động chạy tới chương trình con phục vụ ngắt và tự động xóa cờ TFx bằng phần cứng.
- Để lặp lại quá trình đếm của bộ định thời thì cần nạp lại giá trị ban đầu vào các thanh ghi định thời TLx và THx và cờ TFx cần được xóa về 0.

Từ những đặt trưng của chế độ định thời 1, bộ định thời ở chế độ 1 thường sử dụng cho mục đích tạo thời gian trễ hoặc tạo xung vuông. Để tạo ra độ trễ thời gian dùng bộ định thời chế độ 1 thì cần phải thực hiện các bước sau:

- Nạp giá trị cho thanh ghi TMOD để xác định bộ định thời và chế độ định thời được chọn
- Nạp giá trị đếm ban đầu cho các thanh ghi định thời TLx và THx
- Khởi động bộ định thời
- Kiểm tra trạng thái bật của cờ định thời TFx bằng lệnh “JNB TFx, đích”. Vòng lặp được thoát khi TFx được bật lên cao
- Dừng bộ định thời
- Xóa cờ TFx cho vòng tiếp theo

- Quay lại bước 2 để nạp lại giá trị vào các thanh ghi định thời TLx và THx

Trong các bước trên có một bước quan trọng là bước 2 cần có các giá trị đếm ban đầu để nạp vào các thanh ghi định thời TLx và THx. Để tính được giá trị đếm ban đầu này thì cần phải có 2 giá trị là thời gian trễ và tần số giao động của thạch anh mà vi điều khiển sử dụng. Các bước để tính giá trị đếm ban đầu như sau:

- Tần số bộ định thời = **tần số thạch anh** chia 12
- Thời gian 1 chu kỳ bộ định thời = 1 chia cho tần số bộ định thời
- Số chu kỳ trễ = **thời gian trễ** chia cho thời gian 1 chu kỳ bộ định thời
- Giá trị thập phân ban đầu = 65536 trừ số chu kỳ trễ
- Chuyển giá trị thập phân ban đầu về dạng số Hexa dưới dạng xxyyH
- Nạp TLx = yy và THx = xx là giá trị đếm ban đầu

Ví dụ: Viết chương trình cho bộ định thời Timer 0 để tạo ra xung trên chân P1.1 có độ rộng là 5 ms với tần số thạch anh = 11.0592

Tính toán các giá trị:

- Tần số bộ định thời: Ftimer = $11.0592/12 = 0.9216$
- Thời gian 1 chu kỳ bộ định thời = $1/0.9216 = 1.085 \mu s$
- Số chu kỳ trễ = $5ms/1.085\mu s = 4608$
- Giá trị thập phân ban đầu = $65536 - 4608 = 60928$
- Đổi sang Hexa = EE00H
- Giá trị cần nạp TL0 = 00H và TH0 = EEH

Đoạn chương trình mẫu:

<i>ORG 0000H</i>	<i>;Bắt đầu chương trình</i>
<i>LJMP MAIN</i>	

<i>MAIN:</i>	<i>;chương trình chính</i>
<i>CLR P1.1</i>	<i>;xóa bit P1.1</i>
<i>MOV TMOD, #01</i>	<i>;chọn Timer 0 chế độ 1</i>
<i>HERE: MOV TL0,#0</i>	<i>;nạp TL0</i>
<i>MOV TH0, #0EEH</i>	<i>;nạp TH0</i>
<i>CPL P1.1</i>	<i>;bù bit P1.1</i>
<i>SETB TR0</i>	<i>;khởi động bộ định thời Timer 0</i>
<i>AGAIN: JNB TF0, AGAIN</i>	<i>;lặp cho đến khi TF0 bật lên 1</i>
<i>CLR TR0</i>	<i>;dừng bộ định thời</i>
<i>CLR TF0</i>	<i>;xóa cờ TF0 cho xung tiếp theo</i>
<i>JMP HERE</i>	<i>;vòng lặp tiếp theo</i>
<i>END</i>	

4.5.2 Lập trình chế độ 2

Đặt trung của bộ định thời ở chế độ 2:

- Là một bộ định thời 8 bit, chỉ có thanh ghi THx của bộ định thời được nạp giá trị đếm ban đầu, với giá trị từ 00H đến FFH
- Sau khi giá trị THx được nạp thì bộ định thời được khởi động bằng lệnh “SETB TR0” đối với Timer0 và “SETB TR1” đối với Timer1. Khi đó giá trị THx được vi điều khiển 8051 đưa vào TLx
- Bộ định thời sau khi được khởi động thì bắt đầu đếm tăng từ giá trị ban đầu của thanh ghi TLx đến giá trị giới hạn là FFH và trở về 00H và thiết lập cờ TFx lên 1.
- Khi cờ TFx bật lên 1 thì thanh ghi TLx được tự động nạp lại với giá trị ban đầu được giữ trong thanh ghi THx, người lập trình chỉ cần xóa cờ TFx để lặp lại quá trình định thời. Vì vậy chế độ 2 còn gọi là chế độ

định thời tự nạp lại, và giá trị THx không thay đổi trong quá trình bộ định thời hoạt động.

Tương tự với chế độ 1, để tạo ra độ trễ thời gian dùng bộ định thời chế độ 2 cần thực hiện các bước sau:

- Nạp vào thanh ghi TMOD giá trị phù hợp để chọn bộ định thời và chế độ làm việc
- Nạp vào thanh ghi THx giá trị đếm ban đầu, giá trị này được tính tương tự như ở chế độ 1. Tuy nhiên, ở bước 4 thì giá trị ban đầu = 256 - số chu kỳ trễ
- Khởi động bộ định thời
- Kiểm tra trạng thái bật của cờ định thời TFx bằng lệnh “JNB TFx, đích”. Vòng lặp được thoát khi TFx được bật lên cao
- Xóa cờ TFx cho vòng tiếp theo
- Quay trở lại bước kiểm tra trạng thái bật cờ

Ví dụ: Viết chương trình cho bộ định thời Timer 0 để tạo ra xung trên chân P1.0 có độ rộng là 0.1 ms với tần số thạch anh = 11.0592

Tính toán các giá trị:

- Tần số bộ định thời: $F_{timer} = 11.0592/12 = 0.9216$
- Thời gian 1 chu kỳ bộ định thời = $1/0.9216 = 1.085 \mu s$
- Số chu kỳ trễ = $100\mu s / 1.085\mu s = 92$
- Giá trị thập phân ban đầu = $256 - 92 = 164$
- Đổi sang Hexa = A4H
- Giá trị cần nạp TH0 = A4H

Đoạn chương trình mẫu:

<i>ORG 0000H</i>	<i>;Bắt đầu chương trình</i>
<i>LJMP MAIN</i>	

<i>MAIN:</i>	<i>;chương trình chính</i>
<i>CLR P1.0</i>	<i>;xóa bit P1.0</i>

<i>MOV TMOD, #02</i>	<i>;chọn Timer 0 chế độ 2</i>
<i>HERE:</i>	
<i>MOV TH0, #0A4H</i>	<i>;nạp TH0</i>
<i>CPL P1.0</i>	<i>;bù bit P1.0</i>
<i>SETB TR0</i>	<i>;khởi động bộ định thời Timer 0</i>
<i>AGAIN: JNB TF0, AGAIN</i>	<i>;lặp cho đến khi TF0 bật lên 1</i>
<i>CLR TF0</i>	<i>;xóa cờ TF0 cho xung tiếp theo</i>
<i>JMP HERE</i>	<i>;vòng lặp tiếp theo</i>
<i>END</i>	

Có thể sử dụng hai nguồn để tạo xung nhịp cung cấp cho các bộ định thời. Chúng ta sử dụng bit C/T' trong thanh ghi TMOD để chọn nguồn tạo xung nhịp là bộ dao động trong chip để dùng cho định thời hoặc bit 4 của cổng thứ 3 (P3.4) để nhận xung đếm sự kiện từ bên ngoài để đếm sự kiện.

Định khoảng thời gian: Khi C/T' = 0

4.6 Ví dụ

Ví dụ 4.1: Hãy xác định chế độ định thời và bộ định thời được sử dụng trong các trường hợp sau:

- MOV TMOD, #02H
- MOV TMOD, #10H
- MOV TMOD, #23H

Hướng dẫn:

Chuyển các số HEX được nạp vào thanh ghi TMOD sang dạng nhị phân để đối chiếu với thứ tự các bit có trong thanh ghi TMOD chúng ta sẽ suy ra được chế độ định thời và bộ định thời được sử dụng

- TMOD = 00000010 => chế độ định thời 2 của Timer 0

- được chọn
- b. TMOD = 00010000 => chế độ định thời 1 của Timer 2 được chọn
 - c. TMOD = 00100011 => chế độ định thời 3 của Timer 0 và chế độ định thời 2 của Timer 1 được chọn

Ví dụ 4.2: Hãy xác định tần số và chu kỳ của bộ định thời của các mạch vi điều khiển 8051 sử dụng tần số thạch anh ngoài như sau:

- a. 12 MHz
- b. 16 MHz
- c. 11,0592 MHz

Hướng dẫn:

Bộ định thời trong vi điều khiển 8051 có tần số bằng tần số của thạch anh chia cho 12. Và để tính chu kỳ của bộ định thời, ta tính theo công thức: $T = \frac{1}{F}$

a. Tần số bộ định thời là $F_{\text{định thời}} = \frac{1}{12} \times 12MHz = 1MHz$

Chu kỳ của bộ định thời là $T_{\text{định thời}} = \frac{1}{1MHz} = 1\mu s$

b. Tần số bộ định thời là $F_{\text{định thời}} = \frac{1}{12} \times 16MHz = 1.333MHz$

Chu kỳ của bộ định thời là $T_{\text{định thời}} = \frac{1}{1.333MHz} = 0.75\mu s$

c. Tần số bộ định thời là $F_{\text{định thời}} = \frac{1}{12} \times 11.0592MHz = 921.6kHz$

Chu kỳ của bộ định thời là $T_{\text{định thời}} = \frac{1}{921.6kHz} = 1.085\mu s$

Ví dụ 4.3: Viết một chương trình sử dụng bộ định thời 1 để tạo ra một sóng vuông có tần số 5KHz trên chân P0.1 (tần số thạch anh ngoài 12MHz)

Tính toán các giá trị:

- Để tạo một sóng vuông có tần số 5KHz, trước tiên cần tính thời gian mỗi chu kỳ sóng vuông

$$T = \frac{1}{F} = \frac{1}{5000} = 0.0002s = 200\mu s$$

- Từ đó ta tính ra thời gian trễ tại mỗi giá trị 0 và 1 của sóng vuông tại chân P0.1 là $100\mu s$
- Tần số bộ định thời: $F_{timer} = 11.0592/12 = 0.9216$
- Thời gian 1 chu kỳ bộ định thời = $1/0.9216 = 1.085\mu s$
- Số chu kỳ trễ = $100\mu s / 1.085\mu s = 92$
- Giá trị thập phân ban đầu = $256 - 92 = 164$
- Đổi sang Hexa = A4H
- Giá trị cần nạp TH0 = A4H

Chương trình mẫu:

```
ORG 0000H      ;Reset
LJMP MAIN
;-----
ORG 0030H      ;vùng nhớ bắt đầu chương trình
                ;sau bảng vector ngắn
MAIN:          ;chương trình chính
    CLR P1.0    ;xóa bit P1.0
    MOV TMOD, #20 ;khởi động chế độ 2 của Timer 1
;-----
HERE:          ;địa chỉ của HERE
    MOV TH0, #0A4H ;nạp TH0
    CPL P1.0      ;bù bit P1.0
    SETB TR0       ;khởi động bộ định thời Timer 0
;-----
AGAIN: JNB TF0, AGAIN ;lặp cho đến khi TF0 bật lên 1
    CLR TF0        ;xóa cờ TF0 cho xung tiếp theo
    JMP HERE       ;vòng lặp tiếp theo
```

4.7 Tổng kết

Chương 4 trình bày các kiến thức tổng quan về bộ định thời, từ đó đi sâu vào các bộ định thời của vi điều khiển họ 8051. Trong đó cũng trình bày cụ thể các thanh ghi tham gia vào quá trình hoạt động của các bộ định thời trong vi điều khiển 8051 như thanh ghi TMOD, thanh ghi TCON và các thanh ghi định thời THx, TLx. Sau đó mô tả các chế độ hoạt động của các bộ định thời và đặt trung của nó. Cuối cùng là hướng dẫn cách lập trình sử dụng các bộ định thời, cụ thể là 2 chế độ định thời thường được sử dụng là chế độ 1 và chế độ 2 để tạo thời gian trễ. Tuy nhiên, trong các sử dụng các bộ định thời để tạo thời gian trễ này vẫn còn sử dụng vòng lặp để kiểm tra trạng thái của các cờ TFX, làm vốn rất nhiều thời gian hoạt động của vi điều khiển. Để khắc phục vấn đề đó chúng ta có thể thay thế việc kiểm qua cờ TFX bằng cách sử dụng ngắt timer sẽ được trình bày trong chương 6 của giáo trình.

4.8 Bài tập

1. Thế nào là bộ định thời? Bộ định thời dùng làm gì?
2. Bộ định thời có các chế độ hoạt động nào? Sự khác biệt của các chế độ này?
3. Các thanh ghi nào tham gia vào hoạt động của các bộ định thời? Giá trị của nó cần cài đặt thế nào để các bộ định thời hoạt động.
4. Viết chương trình tạo sóng vuông có tần số 50Hz tại chân P2.0 trên vi điều khiển 8051 với tần số thạch anh 11.0592 sử dụng timer 1 chế độ 2

5. Viết chương trình tạo sóng vuông có tần số 2KHz tại chân P2.0 trên vi điều khiển 8051 với tần số thạch anh 11.0592 sử dụng timer 0 chế độ 1

Chương 5. GIAO TIẾP NỐI TIẾP (UART)

Mục tiêu chương

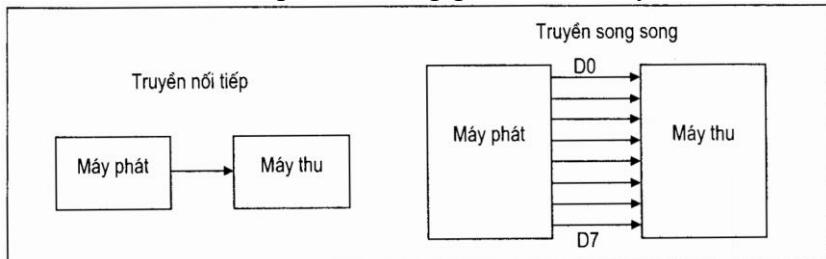
Chương này sẽ trình bày về khái niệm, hoạt động và các yếu tố liên quan tới giao tiếp nối tiếp của vi điều khiển họ 8051. Học xong chương này người học có thể hiểu thế nào là giao tiếp nối tiếp, hiểu được các giao thức trong giao tiếp nối tiếp và có thể lập trình để giao tiếp vi điều khiển 8051 với các thiết bị điện tử khác thông qua giao tiếp nối tiếp.

5.1 Giới thiệu

Thông thường để giao tiếp với nhau bằng dây dẫn, các thiết bị điện tử sẽ có 2 phương pháp truyền dữ liệu đó là truyền song song và truyền nối tiếp như mô tả ở Hình 5-1. Ở mỗi phương pháp truyền sẽ có những ưu khuyết điểm khác nhau:

- Truyền song song phù hợp cho những thiết bị điện tử ở khoảng cách ngắn và cần thời gian truyền nhanh, ví dụ như giữa mainboard của máy tính và ổ cứng ATA hoặc giữa máy tính và máy in đời cũ. Ở phương pháp truyền này, số lượng dây dẫn truyền đi chính bằng độ rộng bit cần truyền, ví dụ truyền dữ liệu 8 bit thì phải có 8 dây dẫn nối giữa 2 thiết bị vì vậy sẽ rất tốn kém khi phải truyền giữa 2 thiết bị có khoảng cách xa. Nhưng bù lại với phương pháp truyền này tốc độ truyền/nhận sẽ rất nhanh.
- Truyền nối tiếp phù hợp cho những thiết bị ở khoảng cách xa hơn. Đây là phương pháp truyền được sử dụng rộng rãi trong thời đại ngày nay. Ở phương pháp truyền dữ liệu này, trước hết byte dữ liệu được chuyển thành các bit nối tiếp nhau 1 thanh ghi dịch

vào song song ra nối tiếp. Sau đó dữ liệu được truyền qua một đường dữ liệu đơn, và ở thiết bị nhận phải có 1 thanh ghi dịch vào nối tiếp ra song song để nhận dữ liệu nối tiếp đó và đóng gói lại thành byte.



Hình 5-1 Truyền dữ liệu song song và nối tiếp

Vì điều khiển 8051 có một cổng giao tiếp nối tiếp dùng để thực hiện việc chuyển đổi dữ liệu song song thành nối tiếp khi truyền và chuyển đổi dữ liệu nối tiếp thành song song khi nhận trong quá trình giao tiếp với một thiết bị khác như máy tính, vi điều khiển khác, hoặc các thiết bị có kiểu giao tiếp nối tiếp. Các thiết bị giao tiếp với 8051 qua cổng nối tiếp thông qua hai chân tín hiệu TxD (truyền dữ liệu) và chân tín hiệu RxD (nhận dữ liệu) được phân bố tại 2 chân của cổng 3 là P3.1 và P3.0.

Cổng nối tiếp trên 8051 có thể hoạt động song công nghĩa là có khả năng truyền và nhận đồng thời. Bên cạnh đó cổng nối tiếp trên 8051 còn có khả năng đệm dữ liệu khi nhận, nên khi một ký tự đang được nhận và lưu trong bộ đệm nhận thì ký tự kế tiếp cũng được nhận vào.

5.2 Thanh ghi cho giao tiếp nối tiếp

Hai thanh ghi chức năng đặc biệt cho phép lập trình điều khiển và truy xuất cổng nối tiếp là SBUF và SCON.

5.2.1 Thanh ghi bộ đệm cổng nối tiếp SBUF

Thanh ghi bộ đệm cổng nối tiếp SBUF có địa chỉ là 99H bao gồm 2 bộ đệm. Dữ liệu ghi vào SBUF sẽ được truyền ra thiết bị bên ngoài qua chân TxD, và việc đọc SBUF để nhận dữ liệu cần truyền đến vi điều khiển. Đây là 2 thanh ghi riêng biệt, một thanh ghi chỉ ghi để truyền dữ liệu và một thanh ghi chỉ đọc để nhận dữ liệu.

5.2.2 Thanh ghi điều khiển cổng nối tiếp SCON

Thanh ghi điều khiển cổng nối tiếp SCON có địa chỉ là 89H là thanh ghi được định địa chỉ theo bit. Thanh ghi này chứa các bit trạng thái và các bit điều khiển cho việc giao tiếp nối tiếp. Các bit được mô tả cụ thể trong Bảng 5-1

Bảng 5-1 Các bit chức năng trong thanh ghi SCON

Bit	Ký hiệu	Địa chỉ	Chức năng
7	SM0	9FH	Bit chọn chế độ
6	SM1	9EH	Bit chọn chế độ
5	SM2	9DH	Bit chọn chế độ, cho phép truyền thông tin trong chế độ đa xử lý trong chế độ 2 và 3; RI sẽ không tác động nếu nhận bit thứ 9 bằng 0
4	REN	9CH	Cần được thiết lập để cho phép nhận dữ liệu
3	TB8	9BH	Khi TB8 = 1 thì bit thứ 9 sẽ được truyền trong chế độ 2 và 3
2	RB8	9AH	Khi RB8 = 1 thì bit thứ 9 sẽ được nhận trong chế độ 2 và 3
1	TI	99H	Được lập khi kết thúc truyền 1 byte dữ liệu, có thể xóa bằng phần mềm

0	RI	98H	Được lập khi kết thúc nhận 1 byte dữ liệu, có thể xóa bằng phần mềm
---	----	-----	---

Cụ thể chức năng các bit trong thanh ghi SCON như sau:

- SM0, SM1 dùng để chọn các chế độ hoạt động của cổng giao tiếp nối tiếp như mô tả trong Bảng 5-2.
- SM2 là bit được sử dụng trong các hệ đa xử lý. Trong giới hạn của giáo trình này không đề cập đến vấn đề đa xử lý đối với vi điều khiển 8051 nên khi lập trình cần đặt SM2 = 0.
- REN là bit cho phép thu, khi bit REN ở mức cao thì nó cho phép vi điều khiển 8051 nhận dữ liệu trên chân RxD. Nếu muốn vi điều khiển 8051 vừa truyền vừa nhận thì bit REN phải được đặt lên 1 bằng lệnh “*SETB SCON.4*”.
- TB8/RB8 được sử dụng trong việc truyền nhận bit thứ 9 trong chế độ 2 và chế độ. Thông thường ta không sử dụng 2 bit này, vì vậy cần đặt TB8 = RB8 = 0 khi lập trình
- TI là cờ ngắt truyền, cờ này được bật khi kết thúc việc truyền một ký tự 8 bit (bật khi bắt đầu truyền bit Stop) để báo cho bộ vi điều khiển biết rằng nó đang sẵn sàng để truyền byte tiếp theo.
- RI là cờ ngắt nhận, cờ này được bật khi kết thúc việc nhận một ký tự 8 bit vào thanh ghi SBUF (bật khi đang tách bit Stop ra khỏi 8 bit dữ liệu) để báo cho bộ vi điều khiển sẵn sàng nhận byte dữ liệu tiếp theo.

5.3 Các chế độ hoạt động của cổng nối tiếp

Cổng nối tiếp của 8051 có 4 chế độ hoạt động, các chế độ này được chọn bằng cách ghi giá trị vào các bit SM0 và SM1 của thanh ghi SCON theo Bảng 5-2

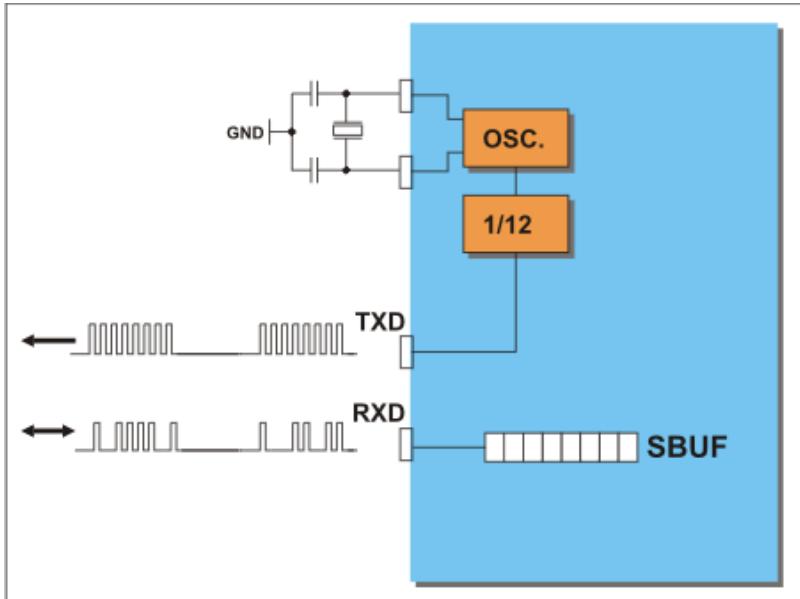
Bảng 5-2 Các chế độ hoạt động của cổng nối tiếp

SM0	SM1	Chế độ	Chức năng	Tốc độ baud
0	0	0	Thanh ghi dịch	Cố định (tần số dao động/12)
0	1	1	8 bit UART	Thay đổi (theo giá trị timer)
1	0	2	9 bit UART	Cố định (tần số dao động/12 hoặc 64)
1	1	3	9 bit UART	Thay đổi (theo giá trị timer)

Trước khi sử dụng cổng nối tiếp phải khởi động thanh ghi SCON để chọn các chế độ thích hợp. Ví dụ lệnh: “MOV SCON, #10010010B” sẽ khởi động cổng nối tiếp ở chế độ 2, cho phép thu và đặt cờ ngắt truyền để chỉ bộ truyền đã sẵn sàng hoạt động.

5.3.1 Chế độ 0

Ở chế độ 0 cổng nối tiếp hoạt động như một thanh ghi dịch 8 bit, được lựa chọn khi SM0=SM1=0. Dữ liệu nối tiếp được truyền và nhận thông qua chân RxD (P3.0), chân TxD (P3.1) xuất tín hiệu xung nhịp đồng bộ khi dịch bit. Khi truyền và nhận dữ liệu 8 bit, LSB được truyền và nhận trước tiên. Tốc độ xung nhịp cố định và bằng 1/12 tần số của mạch tạo dao động bên trong chip.

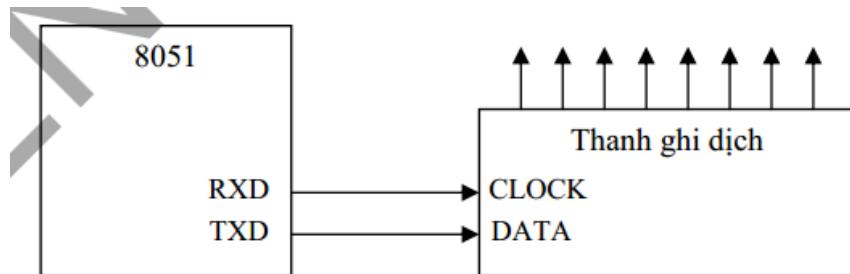


Hình 5-2 Hoạt động UART chế độ 0

Việc truyền dữ liệu được bắt đầu bằng một lệnh ghi dữ liệu cần truyền vào thanh ghi SBUF. Dữ liệu sẽ được dịch ra ngoài qua chân RXD với các xung nhịp dịch bit được đưa ra trên chân TXD. Mỗi bit được truyền đi trong một chu kỳ máy.

Việc nhận dữ liệu được bắt đầu khi bit cho phép nhận REN = 1 và bit ngắt nhận RI = 0. Thông thường bit REN được thiếp lập bằng 1 ở thời điểm bắt đầu chương trình để khởi động cổng nối tiếp, sau đó xóa bit RI để bắt đầu việc nhận dữ liệu.

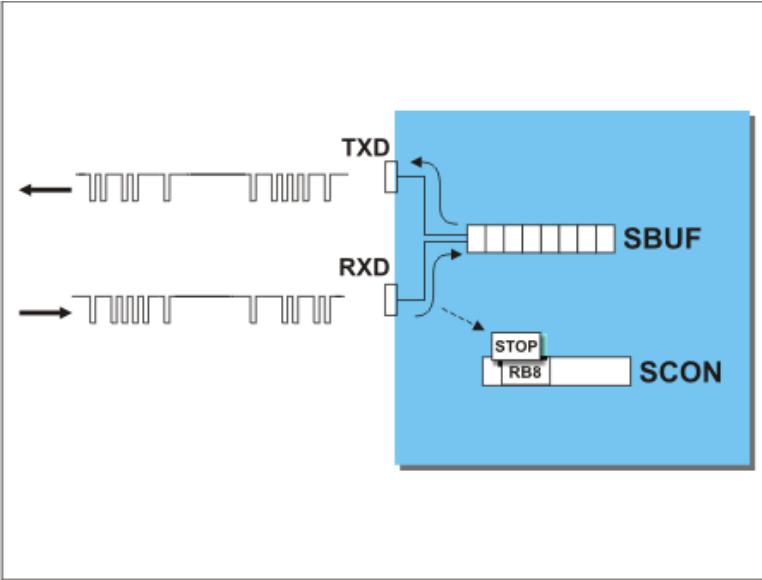
Chế độ 0 thường được ứng dụng để mở rộng thêm các cổng cho 8051 bằng cách nối với các IC thanh ghi dịch nối tiếp – song song (74HC595) qua các chân TXD và RXD như trong Hình 5-3



Hình 5-3 Ứng dụng của chế độ 0

5.3.2 Chế độ 1

Chế độ 1 là chế độ cổng nối tiếp của 8051 truyền nhận bát đồng bộ (UART-Universal Asynchronous Receiver/Transmitter) 8 bit với tốc độ baud thay đổi được. UART thực hiện việc truyền/nhận dữ liệu nối tiếp với mỗi ký tự dữ liệu, bắt đầu bằng bit start ở mức thấp, tiếp theo là 8 bit dữ liệu và cuối cùng là bit stop ở mức cao. Như vậy để truyền/nhận 1 ký tự chúng ta cần phải truyền/nhận 10 bit dữ liệu thông qua các chân Rxd/TxD.



Hình 5-4 Hoạt động UART chế độ 1

Việc truyền dữ liệu được bắt đầu bằng cách ghi dữ liệu vào SBUF, sau đó phải chờ đến khi bộ đếm cung cấp tốc độ baud tràn thì dữ liệu mới được truyền đi tại chân TxD. Thời gian truyền của mỗi bit tỉ lệ nghịch với tốc độ baud được lập trình trong Timer . Cờ ngắt truyền (TI) sẽ được đặt lên 1 khi bit stop xuất hiện trên chân TxD.

Việc thu dữ liệu được bắt đầu khi có sự chuyển trạng thái từ 1 xuống 0 trên đường RxD. Bộ đếm ngay lập tức được xóa để bắt đầu đếm thời gian nhận các bit. Bộ thu sẽ kiểm tra lại bit start ở lần đếm tiếp theo, khi có sự chuyển trạng thái từ 1 đến 0 đầu tiên trên RxD. Nếu thời điểm này RxD không còn giữ trạng thái 0 thì bộ thu sẽ xem nwh tín hiệu start không hợp lệ. Khi đó bộ thu sẽ được reset và quay về trạng thái nghỉ để chờ sự chuyển trạng thái từ 1 đến 0 kế tiếp.

Khi đã phát hiện được bit start hợp lệ, các bit tiếp theo của ký tự sẽ được thu. Bit start được bỏ qua và 8 bit dữ liệu được

đưa vào thanh ghi dịch SBUF, bit stop sẽ được nạp vào bit RB8 của thanh ghi SCON, sau đó cờ ngắt nhận (RI) sẽ được lập lên 1.

5.3.3 Chế độ 2

Chế độ 2 là chế độ cổng nối tiếp truyền/nhận bát đồng bộ 9 bit với tốc độ baud cố định bằng tần số chia cho 12 hoặc 64, được lựa chọn khi SM1 = 1 và SM0 = 0. Trong chế độ này 11 bit sẽ được truyền và nhận bao gồm 1 bit start, 8 bit dữ liệu, bit dữ liệu thứ 9 có thể lập trình được và 1 bit stop. Quá trình truyền nhận cũng tương tự chế độ 2, chỉ khác ở chỗ:

- Khi truyền bit thứ 9 là một bit do người lập trình quy định được đưa vào bit TB8 trong thanh ghi SCON (có thể là bit chẵn/lẽ)
- Khi nhận bit thứ 9 thu được sẽ ở trong bit RB8 trong thanh ghi SCON dùng để so sánh để xác định dữ liệu nhận được có đúng không?

5.3.4 Chế độ 3

Chế độ 3 là chế độ cổng nối tiếp truyền/nhận bát đồng bộ 9 bit với tốc độ baud có thể thay đổi được bằng cách lập trình Timer 1, được lựa chọn khi SM1 = 1 và SM0 = 1. Hoạt động cổng nối tiếp ở chế độ này hoàn toàn giống chế độ 2, chỉ khác ở chỗ có thể lập trình được tốc độ baud.

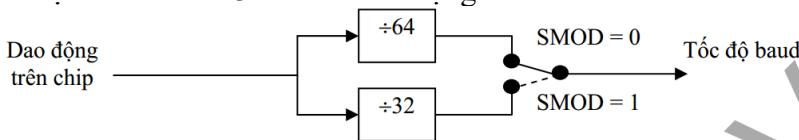
5.4 Tốc độ baud cổng nối tiếp

Tốc độ baud cổng nối tiếp là cố định ở các chế độ 0 và 2. Trong chế độ 0 nó luôn là tần số dao động trên vi điều khiển được chia cho 12. Giả sử với tần số dao động của thạch anh cung cấp cho chip là 16MHz, thì tốc độ baud ở chế độ 0 sẽ là 1.33MHz.



Hình 5-5 Bộ chia tốc độ baud chế độ 0

Trong chế độ 2 tốc độ baud mặc định là tần số dao động trên vi điều khiển chia cho 64. Ngoài ra nó còn được quyết định bởi 1 bit trong thanh ghi điều khiển nguồn (PCON), đó là bit PCON.7 còn được gọi là bit SMOD. Nếu SMOD = 1 thì tốc độ baud sẽ được tăng gấp đôi trong chế độ 2 cũng như trong các chế độ 1 và 3. Vậy trong chế độ 2, nếu SMOD = 0 (mặc định) thì tốc độ baud sẽ là 1/64 tần số giao động, nếu SMOD = 1 thì tốc độ baud sẽ là 1/32 tần số dao động.



Hình 5-6 Bộ chia tốc độ baud chế độ 1 và 3

Tốc độ baud cổng nối tiếp ở chế độ 1 và 3 được xác định bằng cách cài đặt giá trị của Timer 1. Vì timer hoạt động ở tần số tương đối cao nên tốc độ baud cho cổng nối tiếp bằng tốc độ tràn của timer chia thêm cho 32 (SMOD = 0) hoặc chia cho 16 (SMOD = 1).

Thông thường trong chế độ 1 và 3, để tạo tốc độ baud sẽ nạp vào thanh ghi TMOD = 0010xxxxB để chọn chế độ định thời 8 bit tự nạp lại (chế độ 2), và nạp giá trị vào TH1. Chúng ta cũng có thể cài đặt tốc độ baud lớn hơn bằng cách sử dụng chế độ 16 bit (chế độ 1) bằng cách nạp TMOD = 0001xxxxB. Tuy nhiên, việc này làm chương trình phần mềm phức tạp hơn vì sau mỗi lần timer tràn thì giá trị thanh ghi TH1 và TL1 phải được nạp lại. Ngoài ra chúng ta cũng có thể cấp xung nhịp cho Timer 1 từ bên ngoài vi điều khiển bằng chân T1 (P3.5). Trong tất cả các trường hợp thì tốc độ baud được tính bằng tốc độ tràn của Timer 1 chia cho 32 (hoặc 16 nếu SMOD = 1).

Công thức tính tốc độ baud:

$$\text{Tốc độ baud} = \text{Tốc độ tràn của Timer 1} / (32/2^{\text{SMOD}})$$

Vậy khi biết tốc độ baud, chúng ta sẽ tính toán giá trị cần nạp vào thanh ghi TH1 nếu sử dụng Timer 1 chế độ 2 bằng cách nào?

Ví dụ nếu tốc độ baud cần thiết là 2400, tần số dao động được cung cấp bởi thạch anh 12MHz và SMOD = 0, thì khi đó tốc độ tràn của Timer1 là:

$$2400 \times 32 = 76800 \text{ Hz} = 76.8 \text{ KHz}$$

Tần số xung nhịp của bộ tạo dao động là:

$$12 : 12 = 1 \text{ MHz} = 1000 \text{ KHz}$$

Vậy Timer 1 cần tràn sau:

$$1000 : 76.8 = 13.02 \text{ xung nhịp}$$

~ 13 xung nhịp (sai số 0.16 %)

Vì Timer 1 sẽ tràn khi có sự thay đổi từ FFH xuống 00H ở bộ định thời, nên giá trị cần nạp vào TH1 là -13 bằng lệnh

MOV TH1, # -13 hay MOV TH1, #0F3H

Do việc làm tròn số xung nhịp nên có sai số ở tốc bộ baud, sau số cho phép nhỏ hơn 5% trong truyền thông bất đồng bộ. Chúng ta cũng có thể có được tốc bộ baud chính xác khi sử dụng thạch anh 11.059 MHz. Bảng ... tóm tắt các giá trị nạp vào TH1 cho các tốc độ baud thông dụng.

Bảng 5-3 Các giá trị nạp vào TH1 cho tốc độ baud thông dụng

Baud Rate	Tần số thạch anh					Bit SMOD
	11.0592	12	14.7456	16	20	
150	40 h	30 h	00 h			0
300	A0 h	98 h	80 h	75 h	52 h	0
600	D0 h	CC h	C0 h	BB h	A9 h	0
1200	E8 h	E6 h	E0 h	DE h	D5 h	0
2400	F4 h	F3 h	F0 h	EF h	EA h	0
4800		F3 h	EF h	EF h		1
4800	FA h		F8 h		F5 h	0
9600	FD h		FC h			0
9600					F5 h	1
19200	FD h		FC h			1
38400			FE h			1
76800			FF h			1

5.5 Ví dụ

Ví dụ 1: Viết chương trình cho vi điều khiển 8051 để truyền nối tiếp liên tục một ký tự “A” với tốc độ baud là 4800, biết vi điều khiển sử dụng thạch anh 11.0592

Các bước lập trình:

- Nạp giá trị 20H vào thanh thi TMOD để sử dụng Timer 1 chế độ 2 cho việc tạo tốc độ baud
- Nạp giá trị FAH vào thanh ghi TH1 để thiết lập tốc độ baud cho việc truyền dữ liệu (theo Bảng 5-3 hoặc tự tính theo công thức)
- Nạp giá trị 50H vào thanh ghi SCON để thiết lập chế độ 1 với khung dữ liệu 8 bit, 1 bit Start và 1 bit Stop
- Set bit TR1 để khởi động timer 1
- Xóa bit TI bằng lệnh “CLR TI”

- Ghi byte ký tự “A” cần truyền vào SBUF
- Kiểm tra việc truyền đã hoàn tất chưa bằng câu lệnh lặp “HERE: JNB TI, HERE”
- Xóa cờ TI khi truyền xong
- Trở lại bước thứ 5 để truyền ký tự “A” lần tiếp theo

Đoạn chương trình mẫu:

*ORG 0000H ;Bắt đầu chương trình
LJMP MAIN*

*MAIN: ;chương trình chính
MOV TMOD, #20H ;chọn Timer 0 chế độ 1
MOV TH1, #0FAH ;nạp TH1
MOV SCON, #50H ;nạp thanh ghi SCON
SETB TR1 ;khởi động bộ định thời Timer 1*

AGAIN: MOV SBUF, # “A” ;truyền ký tự “A” vào SBUF

*HERE: JNB TI, HERE ;lặp cho đến khi TI bật lên 1
CLR TI ;xóa bit TI để truyền tiếp*

*JMP AGAIN ;vòng lặp tiếp theo
END*

Ví dụ 2: Viết chương trình cho vi điều khiển 8051 để truyền nối tiếp liên tục chữ “KTMT” với tốc độ baud là 19200, biết vi điều khiển sử dụng thạch anh 11.0592

Các bước lập trình:

- Nạp giá trị 20H vào thanh thi TMOD để sử dụng Timer 1 chế độ 2 cho việc tạo tốc độ baud
- Nạp giá trị FDH vào thanh ghi TH1 để thiết lập tốc độ baud cho việc truyền dữ liệu (theo Bảng 5-3 hoặc tự tính theo công thức)
- Nạp giá trị 50H vào thanh ghi SCON để thiết lập chế

- độ 1 với khung dữ liệu 8 bit, 1 bit Start và 1 bit Stop
- Set bit TR1 để khởi động timer 1
 - Xóa bit TI bằng lệnh “CLR TI”
 - Ghi byte ký tự lần lượt “K”, “T”, “M”, “T” cần truyền vào thanh ghi A và gọi tới hàm truyền (hàm truyền là một chương trình con)
 - Hàm truyền sẽ truyền ký tự được lưu trong thanh ghi A vào thanh ghi SBUF
 - Kiểm tra việc truyền đã hoàn tất chưa bằng câu lệnh lặp “HERE: JNB TI, HERE”
 - Xóa cờ TI khi truyền xong
 - Trở lại bước thứ 5 để truyền ký tự tiếp theo

Đoạn chương trình mẫu:

<i>ORG 0000H</i>	<i>;Bắt đầu chương trình</i>
<i>LJMP MAIN</i>	
<i>MAIN:</i>	<i>;chương trình chính</i>
<i>MOV TMOD, #20H</i>	<i>;chọn Timer 0 chế độ 1</i>
<i>MOV TH1, #0FDH</i>	<i>;nạp TH1</i>
<i>MOV SCON, #50H</i>	<i>;nạp thanh ghi SCON</i>
<i>SETB TR1</i>	<i>;khởi động bộ định thời Timer 1</i>
<i>AGAIN: MOVA, # "K"</i>	<i>;truyền ký tự "K" vào A</i>
<i>ACALL TRUYEN</i>	<i>;gọi hàm truyền</i>
<i>MOVA, # "T"</i>	<i>;truyền ký tự "T" vào A</i>
<i>ACALL TRUYEN</i>	<i>;gọi hàm truyền</i>
<i>MOVA, # "M"</i>	<i>;truyền ký tự "M" vào A</i>
<i>ACALL TRUYEN</i>	<i>;gọi hàm truyền</i>
<i>MOVA, # "T"</i>	<i>;truyền ký tự "T" vào A</i>
<i>ACALL TRUYEN</i>	<i>;gọi hàm truyền</i>

TRUYEN: MOV SBUF, A ;chương trình con truyền dữ liệu

HERE: JNB TI, HERE ;lặp cho đến khi TI bật lên 1

CLR TI ;xóa bit TI để truyền tiếp

RET ;thoát khỏi chương trình con

END

Ví dụ 3: Viết chương trình cho vi điều khiển 8051 để nhận các byte dữ liệu nối tiếp và lưu vào trong stack với tốc độ baud là 4800, biết vi điều khiển sử dụng thạch anh 12 MHz

Các bước lập trình:

- Nạp giá trị 20H vào thanh thi TMOD để sử dụng Timer 1 chế độ 2 cho việc tạo tốc độ baud
- Nạp giá trị FDH vào thanh ghi TH1 để thiết lập tốc độ baud cho việc truyền dữ liệu (theo Bảng 5-3 hoặc tự tính theo công thức)
- Nạp giá trị 50H vào thanh ghi SCON để thiết lập chế độ 1 với khung dữ liệu 8 bit, 1 bit Start và 1 bit Stop
- Set bit TR1 để khởi động timer 1
- Kiểm tra việc nhận đã hoàn tất chưa bằng câu lệnh lặp “HERE: JNB RI, HERE”
- Lưu ký tự nhận được vào stack
- Xóa cờ RI để nhận ký tự kế tiếp
- Trở lại bước thứ 5 để nhận ký tự tiếp theo

Đoạn chương trình mẫu:

ORG 0000H ;Bắt đầu chương trình

LJMP MAIN

MAIN: ;chương trình chính

MOV TMOD, #20H ;chọn Timer 0 chế độ 1

MOV TH1, #0FDH ;nạp TH1

MOV SCON, #50H ;nạp thanh ghi SCON

<i>SETB RI</i>	<i>;khởi động bộ định thời Timer 1</i>
<i>HERE: JNB RI, HERE</i>	<i>;lặp cho đến khi RI bật lên 1</i>
<i>MOVA, SBUF</i>	<i>;lưu ký tự nhận được vào A</i>
<i>PUSH A</i>	<i>;lưu ký tự nhận được vào stack</i>
<i>CLR RI</i>	<i>;xóa bit RI để nhận tiếp</i>
<i>JMP HERE</i>	<i>;quay lại nhận ký tự kế tiếp</i>
<i>END</i>	

5.6 Tổng kết

Chương này mô tả một số khái niệm cơ bản về truyền nối tiếp với vi điều khiển 8051 bao gồm các thanh ghi dùng cho mục đích truyền nối tiếp, các chế độ truyền nối tiếp với từng đặc trưng của mỗi chế độ. Tiếp đó trình bày cách tính tốc độ baud trong các chế độ truyền thường được sử dụng: chế độ 1 và chế độ 3. Kết thúc chương là một số ví dụ về truyền nhận nối tiếp với vi điều khiển 8051.

5.7 Bài tập

- Cho tần số thạch anh là 16MHz, hãy tìm giá trị cần nạp vào TH1 để cài đặt tốc độ baud cho các trường hợp sau
 - 4800
 - 19200 với SMOD = 1
- Hãy tìm tốc độ baud nếu TH1 = -46, SMOD = 1 và tần số thạch anh là 11.0592

3. Viết chương trình khởi động cổng nối tiếp ở chế độ 8 bit với tốc độ baud là 2400 sử dụng bộ định thời 1. Giải xử lý số hoạt động của vi điều khiển 8051 là 16MHz
4. Viết chương trình gửi chuỗi “ĐH CNTT” qua cổng nối tiếp của vi điều khiển 8051 với tốc độ baud 2400 và tần số thạch anh là 12MHz.
5. Giả sử vi điều khiển 8051 kết nối với máy tính thông qua cổng nối tiếp. Viết chương trình trên vi điều khiển 8051 để nhận ký tự từ máy tính truyền xuống: nếu ký tự là “ON” thì bật LED tại chân P1.2 và truyền lại máy tính chữ “OK”, nếu ký tự khác thì không làm gì.

Chương 6. HOẠT ĐỘNG NGẮT

Mục tiêu chương

Chương này sẽ trình bày về khái niệm, hoạt động và các yếu tố liên quan tới hoạt động ngắt của vi điều khiển họ 8051. Học xong chương này người học có thể hiểu thế nào là hoạt động ngắt, hiểu được các cơ chế phần cứng trong hoạt động ngắt. Chương này cũng sẽ giúp cho sinh viên việc lập trình được các chương trình liên quan đến ngắt bộ định thời, ngắt ngoài, và ngắt giao tiếp nối tiếp.

6.1 Giới thiệu

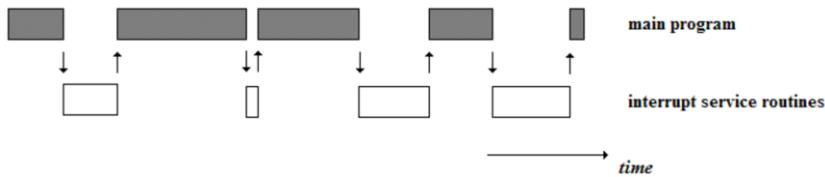
Một ngắt là một sự kiện bên trong hoặc bên ngoài làm ngắt bộ vi điều khiển để báo cho nó biết rằng có sự kiện cần dịch vụ của nó.

Một vi điều khiển có thể phục vụ vài thiết bị để thực hiện một số công việc điều khiển khác nhau, để thực hiện nhiều hoạt động khác nhau đó thông thường có hai cách là thăm dò (polling) và sử dụng các ngắt. Trong phương pháp thăm dò, vi điều khiển sẽ liên tục thăm dò các thiết bị một cách tuần tự để xem thiết bị nào cần phục vụ. Khi đó nếu có một thiết bị nào đó cần dịch vụ nhưng chưa tới lượt vi điều khiển thăm dò tới nó thì thiết bị đó cũng không được phục vụ. Và khi không có thiết bị nào cần phục vụ thì CPU cũng sẽ liên tục thực hiện việc thăm dò dẫn đến tiêu hao nhiều năng lượng.

Để khắc phục những điều đó, phương pháp sử dụng ngắt được áp dụng. Trong phương pháp này thì mỗi khi có một thiết bị bất kỳ cần đến dịch vụ thì nó báo cho bộ vi điều khiển bằng cách gửi một tín hiệu ngắt. Khi nhận được tín hiệu ngắt thì bộ vi điều khiển có thể ngưng tạm thời việc thực thi chương trình nó đang thực hiện để chuyên sang phục vụ thiết bị. Và khi không có

thiết bị nào cần phục vụ thì CPU sẽ nghỉ ngoại để giảm thiểu năng lượng tiêu thụ. Ngoài ra, trong phương pháp sử dụng ngắt chúng ta còn có thể cài đặt độ ưu tiên cho các thiết bị nào cần phục vụ trước, hoặc chúng ta có thể cài đặt để bỏ qua ngắt ở một số thiết bị nào không cần thiết.

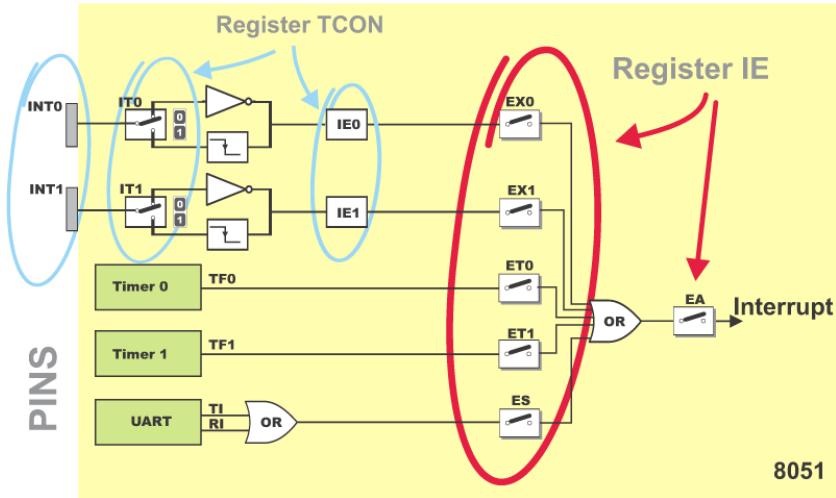
Chương trình đi cùng với ngắt gọi là chương trình phục vụ ngắt (ISR) hoặc chương trình quản lý ngắt. Chương trình phục vụ ngắt được thực thi để đáp ứng một ngắt như bộ định thời, công giao tiếp nối tiếp, các ngắt ngoài. Chương trình phục vụ ngắt được kết thúc bằng lệnh “RETI”, khi đó chương trình chính sẽ tiếp tục chạy tại nơi nó bị dừng trước khi ngắt xảy ra. Chương trình chính thực hiện ở mức nền còn chương trình phục vụ ngắt thực thi ở mức ngắt như sơ đồ Hình 6-1.



Hình 6-1 *Chuyển đổi trạng thái chương trình khi có ngắt*

6.2 Ngắt của 8051

Vì điều khiển 8051 có 6 nguồn yêu cầu ngắt: 1 ngắt reset có mức ưu tiên cao nhất, 2 nguồn ngắt ngoài, 2 nguồn ngắt từ 2 timer và 1 nguồn ngắt từ cổng nối tiếp. Khi hệ thống reset thì tắt cả các nguồn ngắt đều bị cấm (trừ ngắt reset), để cho phép các ngắt hoạt động cần lập trình nạp các giá trị phù hợp vào thanh ghi IE.



Hình 6-2 Sơ đồ các ngắt trong 8051

Khi có nhiều nguồn ngắt yêu cầu ngắt đồng thời, hoặc một ngắt xảy ra trong khi một ngắt khác đang được phục vụ, vì điều khiển sẽ đáp ứng bằng cách xét theo vòng tuần tự (cố định thứ tự các ngắt) hoặc dựa vào mức ưu tiên các ngắt đã được lập trình trước đó để xác định ngắt nào cần thực hiện.

6.3 Thanh ghi đặc biệt cho ngắt

6.3.1 Thanh ghi cho phép ngắt (IE – Interrupt Enable)

Thanh ghi cho phép ngắt IE cho phép cài đặt nguồn ngắt nào trong 5 nguồn ngắt được phép ngắt (không tính ngắt reset), có địa chỉ là A8H và được định địa chỉ theo bit. Mỗi nguồn ngắt sẽ được quy định bởi 1 bit trong thanh ghi IE, ngoại ra còn có một bit cho phép cấm ngắt toàn bộ các ngắt. Các bit trong thanh ghi IE được trình bày cụ thể trong Bảng 6-1

Bảng 6-1 Các bit chức năng của thanh ghi IE

Bit	Ký hiệu	Địa chỉ	Chức năng (1: cho phép, 0: cấm)
IE7	EA	AFH	Cho phép/cấm toàn bộ các ngắt
IE6	-	AEH	Không sử dụng
IE5	ET2	ADH	Cho phép ngắt Timer 2 (8052)
IE4	ES	ACH	Cho phép ngắt công nôii tiếp
IE3	ET1	ABH	Cho phép ngắt Timer 1
IE2	EX1	AAH	Cho phép ngắt ngoài từ INT1
IE1	ET0	A9H	Cho phép ngắt Timer 0
IE0	EX0	A8H	Cho phép ngắt ngoài từ INT0

Như vậy khi muốn cho phép bất kỳ ngắt nào hoạt động chúng ta cần phải lập 2 bit: bit cho phép toàn bộ và bit cho phép ngắt riêng. Ví dụ muốn ngắt từ Timer 0 được cho phép như sau:

*SETB EA ; Cho phép ngắt toàn bộ
SETB ET0 ; Cho phép ngắt từ Timer 0*

Hoặc

MOV IE, #10000010B

Mặc dù hai cách này có cùng kết quả là cho phép ngắt Timer 0 được hoạt động, nhưng nếu sử cách thứ 2 ở giữa chương trình thì các ngắt khác đã được cho phép trước đó sẽ bị cấm, còn sử dụng cách thứ 1 thì sẽ không bị ảnh hưởng đến các ngắt khác. Vậy cách thứ 2 thường được sử dụng ở đầu chương trình để khởi tạo các ngắt, còn khi muốn cấm/cho phép các ngắt ngay trong chương trình thì nên sử dụng cách 1 để không ảnh hưởng đến các ngắt khác trong hệ thống.

6.3.2 Thanh ghi ưu tiên ngắt (IP – Interrupt Priority)

Thanh ghi ưu tiên ngắt cho phép mỗi nguồn ngắt được lập trình riêng vào một trong hai mức ưu tiên, có địa chỉ là B8H và được định địa chỉ theo bit. Chức năng của các bit trong thanh ghi IP được mô tả trong Bảng 6-2

Bảng 6-2 Các bit chức năng của thanh ghi IP

Bit	Ký hiệu	Địa chỉ	Chức năng (1: cho phép, 0: cấm)
IP7	-	-	Không sử dụng
IP6	-	-	Không sử dụng
IP5	PT2	BDH	Ưu tiên cho ngắt Timer 2 (8052)
IP4	PS	BCH	Cho phép ngắt công nối tiếp
IP3	PT1	BBH	Ưu tiên cho ngắt Timer 1
IP2	PX1	BAH	Ưu tiên cho ngắt ngoài từ INT1
IP1	PT0	B9H	Ưu tiên cho ngắt Timer 0
IP0	PX0	B8H	Ưu tiên cho ngắt ngoài từ INT0

Thanh ghi IP mặc định bị xóa sau khi reset hệ thống, để đặt tất cả các ngắt ở mức ưu tiên thấp. Sau khi cài đặt độ ưu tiên, nếu một trình phục vụ ngắt (ISR) có độ ưu tiên thấp đang được thực thi, khi có một ngắt có ưu tiên cao xảy ra thì ISR hiện hành sẽ bị ngắt để phục vụ cho ISR có độ ưu tiên cao hơn. Nếu hai ngắt có độ ưu tiên khác nhau xảy ra đồng thời, thì ngắt có độ ưu tiên cao hơn sẽ được phục vụ trước. Nếu hai ngắt cùng độ ưu tiên xảy ra đồng thời, thứ tự các ngắt sẽ được phục vụ theo thứ tự tuần tự: INT0, Timer 0, INT1, Timer 1, công nối tiếp.

6.3.3 Thanh ghi chúa cờ ngắt

Các cờ ngắt trong vi điều khiển 8051 được chứa trong các thanh ghi đã được chúng ta tìm hiểu trong phần Timer và công nối tiếp đó là thanh ghi TCON và SCON. Cụ thể các cờ ngắt được mô tả trong Bảng 6-3

Bảng 6-3 Các cờ ngắt

Ngắt	Cờ	Bit cụ thể
INT0	IE0	TCON.1
INT1	IE1	TCON.3
Timer 0	TF0	TCON.5
Timer 1	TF1	TCON.7
Công nối tiếp	TI	SCON.1
Công nối tiếp	RI	SCON.3
Timer 2 (8052)	TF2	T2CON.7
Timer 2 (8052)	EXF2	T2CON.6

6.4 Chương trình phục vụ ngắt

Khi một ngắt xảy ra và được vi điều khiển chấp nhận, chương trình chính sẽ bị ngắt quãng và những hoạt động sau sẽ xảy ra:

- Hoàn tất việc thực thi lệnh hiện hành
- Cắt thanh ghi PC vào ngăn xếp
- Trạng thái ngắt hiện hành được lưu lại
- Các ngắt được chặn lại ở mức ngắt
- Bộ đếm chương trình PC được nạp địa chỉ vector của mình phục vụ ngắt ISR
- ISR được thực thi

ISR được thực thi để phục vụ công việc của ngắt đó. Việc thực thi ISR được kết khi gặp lệnh RETI. Lệnh này lấy lại giá

trị cũ của bộ đếm chương trình PC từ ngăn xếp và phục hồi trạng thái của ngắt cũ. Chương trình chính sẽ thực thi tại nơi mà nó bị tạm ngưng trước đó.

Khi chấp nhận ngắt, giá trị được nạp vào thanh ghi PC được gọi là vector ngắt, nó là địa chỉ bắt đầu của ISR cho nguồn tạo ngắt. Các vector ngắt của vi điều khiển 8051 có độ dài 8 byte, và được tập hợp thành 1 bản gọi là bảng vector ngắt (Bảng 6-4)

Bảng 6-4 Bảng vector ngắt

Ngắt	Còn	Địa chỉ vector ngắt
Reset	RST	0000H
INT0	IE0	0003H
Timer 0	TF0	000BH
INT1	IE1	0013H
Timer 1	TF1	001BH
Cổng nối tiếp	RI hoặc TI	0023H
Timer 2 (8052)		

Các chương trình phục vụ ngắt phải bắt đầu ở các địa chỉ trong bảng vector ngắt. Vì vector ngắt cho mỗi ngắt chỉ có 8 byte nên đối với chương trình phục vụ ngắt có kích thước dưới 8 byte thì chúng ta có thể đặt chương trình phục vụ ngắt vào ô nhớ dành cho vector ngắt. Ví dụ viết chương trình sử dụng ngắt ngoài 1 để bật LED tại cổng P1, và sử dụng ngắt Timer 1 để bật đèn LED ở cổng P2 như sau:

<i>ORG 0000H</i>	<i>;Reset</i>
<i>LJMP MAIN</i>	
<i>ORG 0013H</i>	<i>; bắt đầu ngắt ngoài 1</i>
<i>MOV P1,#0FFH</i>	<i>; bật LED tại cổng P1</i>
<i>RETI</i>	<i>; thoát khỏi chương trình ngắt</i>
<i>ORG 001BH</i>	<i>; bắt đầu ngắt Timer 1</i>
<i>MOV P2, #0FFH</i>	<i>; bật LED tại cổng P2</i>

<i>RETI</i>	<i>; thoát khỏi chương trình ngắn</i>
<i>ORG 0030H</i>	<i>;vùng nhớ bắt đầu chương trình sau bảng vector ngắn</i>
<i>MAIN:</i>	<i>;chương trình chính</i>
....	

Đối với chương trình phục vụ ngắn có kích thước dài hơn 8 byte, cần có lệnh nhảy để chuyển nó tới nơi khác trong bộ nhớ chương trình, nếu không chương trình sẽ bị nằm qua vùng nhớ của vector ngắn khác.

Ví dụ 6.1: viết chương trình sử dụng ngắn Timer 1 để bật lần lượt từng LED trên cổng P0.

Chương trình mẫu:

<i>ORG 0000H</i>	<i>;Reset</i>
<i>LJMP MAIN</i>	
<i>ORG 001BH</i>	<i>; bắt đầu ngắn Timer 1</i>
<i>LJMP ISRTimer1</i>	<i>; nhảy tới đoạn chương trình phục vụ ngắn cho Timer 1</i>
<i>ORG 0030H</i>	<i>;vùng nhớ bắt đầu chương trình ngắn</i>
<i>sau bảng vector</i>	
<i>MAIN:</i>	<i>;chương trình chính</i>
....	
<i>ISRTimer1:</i>	<i>;chương trình phục vụ ngắn cho Timer 1</i>
<i>MOV P1,#01H</i>	
<i>MOV P1,#02H</i>	
<i>MOV P1,#04H</i>	
<i>MOV P1,#08H</i>	
<i>MOV P1,#10H</i>	
<i>MOV P1,#20H</i>	
<i>MOV P1,#40H</i>	
<i>MOV P1,#80H</i>	
<i>RETI</i>	<i>;thoát khỏi chương trình ngắn</i>

6.4.1 Ngắt reset

Ngắt reset có địa chỉ vector ngắt là 0000H, đây là ngắt đặc biệt và có độ ưu tiên cao nhất. Nó không cần cài đặt các thanh ghi để sử dụng, cũng như không cần chương trình phục vụ ngắt. Trong vi điều khiển 8051, ngắt reset được quyết định duy nhất bởi việc nối chân Reset với Vcc thông qua công tắt hoặc nút ấn. Khi ngắt reset xảy ra chương trình sẽ chạy lại ở vị trí 0000H và đồng thời reset toàn bộ các thanh ghi và nội dung trong RAM.

6.4.2 Ngắt bộ định thời

Các ngắt bộ định thời có địa chỉ vector ngắt là 000BH (Timer 0) và 001BH (Timer 1). Các ngắt này xảy ra khi các thanh ghi Timer (THx/TLx) tràn và lập cờ báo tràn TFX lên 1. Khi chương trình phục vụ ngắt được gọi thì cờ TFX sẽ tự động xóa bằng phần cứng. Ngắt bộ định thời thông thường được sử dụng để tạo delay trong các chương trình hoặc tạo sóng vuông ở một số chân vi điều khiển.

Ví dụ 6.2: Viết một chương trình sử dụng bộ định thời 1 và các ngắt timer để tạo ra một sóng vuông có tần số 5KHz trên chân P0.1 (tần số thạch anh ngoài 12MHz)

Hướng dẫn: Chương trình tạo sóng vuông có tần số 5KHz sử dụng ngắt cũng tương tự với chương trình tạo sóng vuông trong chương Bộ định thời. Nhưng khác ở chỗ, thay vì trong chương Bộ định thời thì chương trình sẽ liên tục kiểm tra cờ TFX để biết thời điểm bộ định thời tràn, làm cho CPU không thể làm thêm việc gì khác. Còn ở chương này, với phương pháp sử dụng ngắt thì chương trình có thể làm các công việc khác, chờ khi cờ tràn TFX bật lên thì chương trình sẽ thông báo một ngắt đến CPU và khi đó CPU sẽ được sử dụng để phục vụ chương trình ngắt. Trong chương trình phục vụ ngắt, ta không cần phải xóa cờ TFX mà cờ này sẽ được xóa tự động bởi phần cứng.

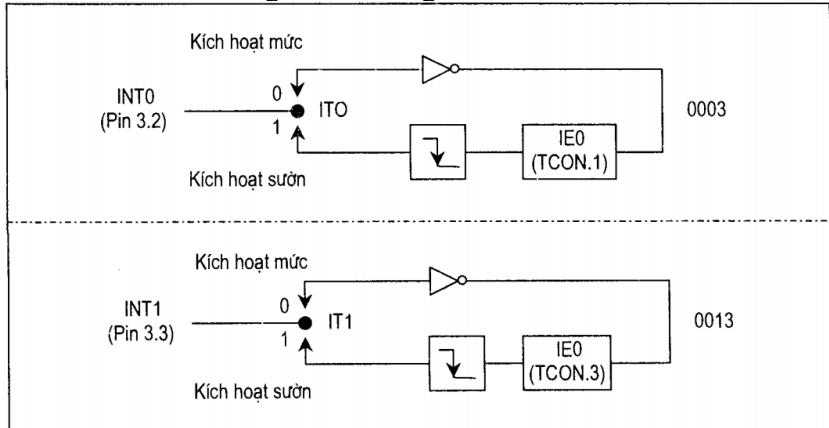
Chương trình mẫu:	
<i>ORG 0000H</i>	;Reset
<i>LJMP MAIN</i>	
<i>ORG 001BH</i>	; bắt đầu ngắt Timer 1
<i>LJMP ISRTimer1</i>	; nhảy tới đoạn chương trình phục vụ ngắt cho Timer 1
<i>ORG 0030H</i>	;vùng nhớ bắt đầu chương trình sau bảng vector ngắt
<i>MAIN:</i>	;chương trình chính
<i>MOV TMOD,#20</i>	; khởi động chế độ 2 của Timer 1
<i>MOV TH0, #-100</i>	; trì hoãn 100ms
<i>SETB TR1</i>	;bắt đầu hoạt động Timer1
<i>MOV IE, #88H</i>	;cho phép ngắt Timer 1
<i>SJMP \$</i>	;không làm gì
<i>ISRTimer1:</i>	;chương trình phục vụ ngắt cho Timer1
<i>CPL P0.1</i>	;lấy bù
<i>RETI</i>	;thoát khỏi chương trình ngắt
<i>END</i>	

6.4.3 Ngắt ngoài

Ngắt ngoài xảy ra khi có mức thấp hoặc có cạnh xuống kích hoạt lên chân INT0 (P3.2) hoặc INT1 (P3.3) của vi điều khiển 8051. Tín hiệu từ các chân này sẽ tạo ra các cờ ngắt tại cái bit IE0 hoặc IE1 của thanh ghi TCON. Khi một ngắt ngoài xảy ra, vi điều khiển sẽ chuyển đến ISR cho ngắt yêu cầu và cờ tạo ra ngắt được xóa bởi phần cứng nếu ngắt theo cạnh. Nếu ngắt theo mức thì cờ tạo ngắt được xóa khi có mức cao kích hoạt lên chân tạo ngắt.

Việc chọn ngắt tích cực mức thấp hoặc tích cực cạnh xuống được lập trình thông qua các bit IT0 và IT1 trong thanh ghi TCON như trong Hình 6-3. Nếu IT0 = 0 thì ngắt ngoài 0 được

kích hoạt bằng mức thấp ở chân INT0. Nếu IT0 = 1, ngắt ngoài 0 được kích hoạt bằng cạnh xuống ở chân INT0.



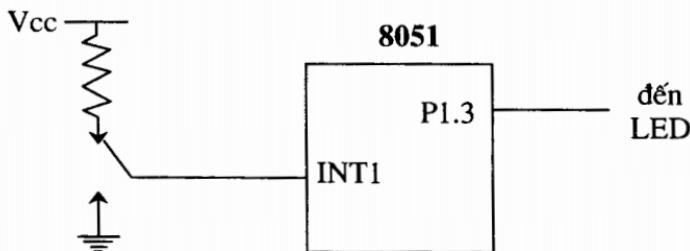
Hình 6-3 Cấu tạo ngắt ngoài

6.4.3.1 Ngắt theo mức

Ngắt ngoài theo mức được mặc định khi sử dụng ngắt ngoài sau khi vi điều khiển hoạt động hoặc reset. Hoặc để cài đặt sử dụng ngắt ngoài theo mức, thì phải nạp $ITx = 0$ và khi chân INTx đang ở mức cao ít nhất một chu kỳ máy được chuyển xuống mức thấp liên tục 4 chu kỳ máy thì một ngắt theo mức sẽ được xác lập. Khi đó, vi điều khiển sẽ dừng công việc đang thực hiện và nhảy đến bảng vector ngắt để thực thi chương trình phục vụ ngắt. Và tín hiệu mức thấp phải được thả ra trước khi thực hiện lệnh cuối cùng của ISR nếu không thì sẽ có 1 ngắt khác được tạo ra. Điều này làm chúng ta sẽ rất khó xác định án nút bao lâu để thực hiện 1 lần ngắt, nếu án chưa đủ 4 chu kỳ máy thì chưa tạo ra ngắt, còn án quá lâu thì sẽ xảy ra nhiều ngắt.

Ví dụ 6.3: Giả sử chân INT1 được nối đến công tắc có điện trở kéo lên nối đến GND, khi công tắc được án thì một LED đơn được nối đến chân P1.3 sáng một khoảng thời gian ngắn, nếu

công tắc được ấn liên tục thì LED đơn sẽ sáng liên tục. Sơ đồ mạch được nối theo Hình 6-4



Hình 6-4 Sơ đồ nối công tắc và LED

Hướng dẫn: Chương trình này nhận tín hiệu công tắc tại chân INT1 (P3.3) để nhảy đến bảng vector ngắt tại địa chỉ 0013H, và tại đây sẽ nhảy đến chương trình phục vụ ngắt ISRINT1. Chương trình phục ngắt sẽ bật LED sáng 1 khoảng thời gian bằng 255 lần thực hiện câu lệnh “DJNZ” rồi sau đó tắt LED, và cuối cùng thực hiện lệnh RETI để trở về chờ ở chương trình chính cho lần ngắt tiếp theo .

Chương trình mẫu:

```

ORG 0000H           ;Reset
LJMP MAIN

;-----;
ORG 0013H           ; bắt đầu ngắt INT1
LJMP ISRINT1 ; nhảy tới đoạn chương trình phục
vụ ngắt cho INT1
;-----;

ORG 0030H           ;vùng nhớ bắt đầu chương trình
                     ;sau bảng vector ngắt
MAIN:               ;chương trình chính
    MOV IE, #82      ;cho phép ngắt ngoài INT1
    SJMP $            ;chờ đến khi phục vụ ngắt
;-----;
```

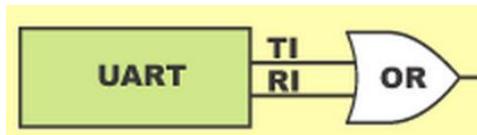
<i>ISRINT1:</i>	<i>;chương tình phục vụ ngắt cho INT1</i>
<i>SETB P1.3</i>	<i>;bật đèn LED</i>
<i>MOV R3, #255</i>	<i>;nap giá trị delay</i>
<i>BACK: DJNZ R3, BACK</i>	<i>;lặp để delay</i>
<i>CLR P1.3</i>	<i>;tắt đèn LED</i>
<i>RETI</i>	<i>;thoát khỏi chương trình ngắt</i>
<i>END</i>	

6.4.3.2 Ngắt ngoài theo cạnh

Ngắt ngoài theo cạnh xuống sẽ xử lý được vấn đề này, ngắt theo mức cần nạp ITx = 1 và khi chân INTx đang ở mức cao một chu kỳ máy xuống mức thấp 1 chu kỳ máy thì ngắt được xác lập trong khoảng thời gian xuống đó và bật bit IEEx lên 1 sau đó bỏ qua các sườn xuống khác.

6.4.4 Ngắt cổng nối tiếp

Các ngắt do cổng nối tiếp xảy ra khi cò ngắt truyền TI hoặc cò ngắt nhận RI bằng 1. Một ngắt truyền được gọi khi hoàn tất việc truyền một ký tự trong thanh ghi SBUF. Một ngắt nhận được gọi khi một ký tự được nhận đầy đủ và đang ở trong SBUF để chờ được vi điều khiển đọc. Vậy ngắt truyền xảy ra khi bộ đếm truyền SBUF rỗng, và ngắt nhận xảy ra khi bộ đếm nhận SBUF đầy.



Hình 6-5 Cò ngắt cổng nối tiếp

Các ngắt do cổng nối tiếp khác với các ngắt do bộ định thời ở chỗ cò ngắt không được xóa bởi phần cứng khi CPU nhảy đến chương trình phục vụ ngắt. Vì chúng ta có 2 nguyên nhân tạo ra ngắt ở cổng nối tiếp là cờ TI OR với cờ RI như Hình 6-5 nên cần phải được xác định cụ thể nguyên nhân nào và được xóa bởi phần mềm trong quá trình xử lý ngắt.

Ví dụ 6.4: Viết chương trình sử dụng vi điều khiển 8051 đọc dữ liệu từ cổng P1 và ghi liên tục tới cổng P2, đồng thời đưa dữ liệu đó tới cổng nối tiếp để truyền đi, sử dụng ngắt truyền nối tiếp. Biết tần số thạch anh là 11.0592 và tốc độ baud là 9600

Hướng dẫn: Chương trình này cũng tương tự chương trình truyền các ký tự qua cổng nối tiếp ở chương 5 nhưng khác ở chỗ trong chương trình này cùng lúc làm 2 công việc là vừa chuyển dữ liệu tới cổng P2, vừa chuyển dữ liệu tới cổng nối tiếp. Vì vậy nếu ta dùng phương pháp trong chương 5 thì chương trình sẽ bị dừng ở chỗ kiểm tra cờ TI vì khi chương trình kiểm tra cờ TI thì vi điều khiển không thể chuyển dữ liệu từ P1 tới P2 nếu P1 có thay đổi dữ liệu. Thay vì vậy trong chương này chúng ta sử dụng ngắt cho cổng nối tiếp sẽ giải quyết được vấn đề này. Với việc sử dụng ngắt chương trình sẽ liên tục truyền dữ liệu từ P1 đến P2, đồng thời khi cổng nối tiếp truyền xong thì sẽ xảy ra 1 ngắt gọi CPU truyền ký tự kế tiếp rồi lại quay về với việc truyền P1 tới P2. Lưu ý một điểm ở đây là khi sử dụng ngắt cho cổng nối tiếp ta phải kiểm tra cờ TI hoặc RI được bật để xử lý ngắt chính xác vì 2 cờ này được nối chung vào một ngắt xử dụng cổng OR như đã trình bày trong phần lý thuyết.

Chương trình mẫu:

<i>ORG 0000H</i> <i>LJMP MAIN</i> ; <i>ORG 0023H</i> <i>LJMP ISRUART</i>	<i>;Reset</i> <i>; bắt đầu ngắt UART</i> <i>; nhảy tới đoạn chương trình phục vụ ngắt cho UART</i>
--	--

```

;-----;
ORG 0030H ;vùng nhớ bắt đầu chương trình
             ;sau bảng vector ngắt
MAIN:      ;chương trình chính
    MOV P1, #FFH ;đặt P1 làm cổng vào dữ liệu
    MOV TMOD,#20 ;khởi động chế độ 2 của Timer 1
    MOV TH0, #FDH ;chọn tốc độ baud 9600
    MOV SCON, #50H ;chọn chế độ 1 cho phép nhận
    MOV IE, #90H ;cho phép ngắt truyền nối tiếp
    SETB TR1     ;bắt đầu hoạt động Timer1

BACK:       ;đọc dữ liệu từ cổng P1
    MOVA, P1   ;gửi dữ liệu tới cổng P2
    MOV P2, A   ;quay lại truyền theo vòng lặp
;-----;

ISRUART:   ;chương tình phục vụ ngắt cho
            ;UART
    JB TI, TRUYEN ;nếu bit TI bật thì truyền
    RETI          ;thoát khỏi chương trình ngắt

TRUYEN:    ;truyền dữ liệu tới cổng nối tiếp
    MOV SBUF, A
    RETI          ;-----;

END

```

6.5 Mức ưu tiên ngắt của vi điều khiển 8051

Khi vi điều khiển 8051 được bắt đầu hoặc reset thì các mức ưu tiên ngắt được gán như Bảng 6-5 Theo bảng này chúng ta thấy nếu ngắt bộ định thời 0 và ngắt truyền thông nối tiếp xảy ra cùng lúc thì ngắt bộ định thời 0 sẽ được đáp ứng trước. Sau khi chương trình phục vụ ngắt cho bộ định thời 0 thực thi xong, thì

chương trình mới tiếp tục phục vụ cho ngắt truyền thông nối tiếp vì nó có độ ưu tiên thấp hơn.

Bảng 6-5 Độ ưu tiên các ngắt

STT (Mức ưu tiên)	Ngắt	Còn
1	Reset	
2	Ngắt ngoài 0	INT0
3	Ngắt bộ định thời 0	TF0
4	Ngắt ngoài 1	INT1
5	Ngắt bộ định thời 1	TF1
6	Ngắt truyền nối tiếp	RI hoặc TI

Tuy nhiên trong lập trình chúng ta có thể thay đổi độ ưu tiên các ngắt trên (trừ Reset) bằng cách gán mức ưu tiên cao hơn cho bất kỳ ngắt nào. Điều này chúng ta thực hiện bằng cách lập trình bật bit tương ứng trong thanh ghi IP như đã mô tả ở phần trên lên cao để tăng độ ưu tiên cho ngắt tương ứng. Nếu có nhiều ngắt cùng có độ ưu tiên cao thì thứ tự ưu tiên của các ngắt cao đó cũng theo thứ tự ở bảng ưu tiên trên.

6.6 Tổng kết

Chương này đã trình bày khái niệm về ngắt, các thanh ghi đặc biệt phục vụ cho quá trình cài đặt và sử dụng các ngắt. Bên cạnh đó trình bày phương pháp lập trình sử dụng các ngắt trong vi điều khiển 8051 bao gồm: ngắt bộ định thời, ngắt ngoài và ngắt nối tiếp. Cuối cùng là trình bày về độ ưu tiên của các ngắt trong vi điều khiển 8051.

6.7 Bài tập

1. Viết chương trình tạo ra sóng vuông tần số 50Hz trên chân P1.2 sử dụng ngắt Timer 0 với tần số thạch anh sử dụng cho vi điều khiển 8051 là 11.0592 MHz.

2. Viết chương trình nhận liên tục dữ liệu 8 bit ở cổng P0 và gửi đến cổng P1. Đồng thời trong thời gian này tạo ra trên chân P2.1 một sóng vuông có chu kỳ $200\mu s$. Sử dụng Timer 1 để tạo sóng vuông, biết tần số thạch anh sử dụng cho vi điều khiển 8051 là 11.0592 MHz .
3. Hãy viết chương trình cho vi điều khiển 8051 nhận dữ liệu ở cổng P3 và truyền đến cổng P1, đồng thời nhận dữ liệu ở cổng nối tiếp và truyền đến cổng P2. Với tần số thạch anh là 11.0592MHz và tốc độ baul là 4800.
4. Hãy viết chương trình tạo sóng vuông, với mức cao kéo dài $155\text{ }\mu s$ và mức thấp kéo dài $15\text{ }\mu s$ với tần số thạch anh 12MHz , sử dụng ngắt timer.
5. Viết chương trình sử dụng ngắt ngoài 0, khi bấm giữ phím được nối với ngắt ngoài 0 thì đèn LED nhấp nháy với tần số 1ms.

Chương 7. Lập trình Assembly giao tiếp với 8051

Mục tiêu chương

Chương này sẽ trình bày về phương pháp lập trình Assembly và cấu trúc của một chương trình Assembly cho vi điều khiển 8051 và trình bày ví dụ về các giao tiếp cơ bản với vi điều khiển 8051. Học xong chương này người học có thể lập trình 1 chương trình hoàn thiện sử dụng ngôn ngữ Assembly cho vi điều khiển 8051 và có thể thực hiện các chương trình giao tiếp với các thiết bị ngoại vi với vi điều khiển.

7.1 Cấu trúc tổng quát chương trình

Một chương trình hợp ngữ cho vi điều khiển 8051 bao gồm một tập các dòng lệnh hợp ngữ được phân bố theo một cấu trúc quy định, bắt đầu với lệnh ORG chỉ đến vùng nhớ bắt đầu của chương trình và kết thúc bởi lệnh END.

Ví dụ:

ORG 00H

MAIN:

MOV R2, #04H

MOV A, R2

END

Một dòng lệnh hợp ngữ trong chương trình bao gồm các thành phần sau:

Nhân: Lệnh gọi Toán hạng ;chú thích
nhớ

Ví dụ:

MAIN: MOV A, R2 ;chuyển nội dung thanh ghi
R2 vào thanh ghi A

Trong đó:

- Trường nhãn dùng để đánh dấu vị trí bắt đầu một đoạn lệnh, có thể nhảy đến vị trí của nhãn bằng các câu lệnh nhảy trong lập trình hợp ngữ. Nhãn là một trong các loại ký hiệu và nó được phân biệt bằng dấu hai chấm “..”. Một nhãn phải bắt đầu bằng một chữ cái, dấu chấm hỏi “?” hoặc dấu gạch dưới “_” và có thể chứa tối đa 31 ký tự. Các ký tự không phân biệt chữ hoa hoặc chữ thường, và không được trùng với các lệnh gọi nhớ, các nhãn đã được định nghĩa trước đó và các chỉ thị hợp dịch. Một câu lệnh có thể có nhãn hoặc không có nhãn.
- Lệnh là một từ gọi nhớ đại diện cho câu lệnh mà người lập trình muốn thực thi, mỗi lệnh sẽ có 1 từ gọi nhớ duy nhất và bắt buộc phải có trong một câu lệnh.
- Toán hạng là các thành phần được viết ngay sau phần mã gọi nhớ trong câu lệnh. Trường này chứa địa chỉ hoặc dữ liệu sử dụng trong câu lệnh và sẽ bị tác động trong khi câu lệnh đó được thực thi. Tùy vào câu lệnh mà có thể có 3 toán hạng, 2 toán hạng, 1 toán hạng hoặc không có toán hạng nào. Trong câu lệnh nhiều toán hạng thì các toán hạng được phân cách nhau bởi dấu phẩy “,”.

Bảng 7-1 Số toán hạng trong câu lệnh

Câu lệnh	Số toán hạng
RET	Không có toán hạng
JZ TEMP	Có 1 toán hạng
ADD A, R3	Có 2 toán hạng

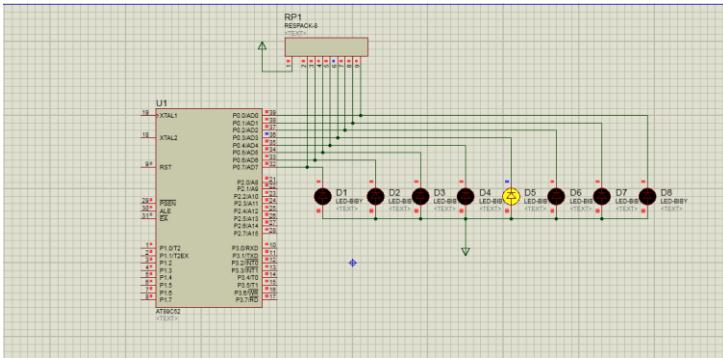
- Trường chú thích được bắt đầu bằng dấu chấm phẩy “;”, có thể đặt chú thích ở bất cứ chỗ nào trong chương trình. Chương trình có thể có hoặc không có chú giải, chú giải sẽ được trình hợp dịch bỏ qua khi biên dịch chương trình, nhưng nó cần thiết để lập trình viên nhớ được chương trình của mình hoặc có thể cho người phát triển sau này có thể hiểu được chương trình.

7.2 Giao tiếp với thiết bị hiển thị

7.2.1 Giao tiếp với LED đơn

Phần này sẽ giới thiệu chương trình đơn giản để sinh viên làm quen với vi điều khiển 8051, chương trình sẽ điều khiển dãy 8 LED đơn nhấp nháy theo hiệu ứng sáng dần từng LED từ trái qua phải sau đó quay lại trạng thái ban đầu.

Sơ đồ nguyên lý của mạch được thiết kế trên phần mềm Proteus bao gồm những kết nối về mặt nguyên lý để có thể mô phỏng được chương trình như trong Hình 7-1. Các kết nối khác như nguồn, một số tụ được bỏ qua vì mặc định chương trình đã có sẵn. Sơ đồ cho mạch giao tiếp vi điều khiển 8051 với LED đơn được mô tả trong hình. Lưu ý khi kết nối cổng P0 của 8051 thì cần có điện trở kéo lên vì kết cấu của port P0 theo dạng mở (Open Drain)



Hình 7-1 Sơ đồ giao tiếp với LED đơn

Trong sơ đồ này sử dụng bộ tạo xung nhịp thạch anh 12MHz.

Chương trình mẫu

ORG 0000H ; Nhấn bắt đầu chương trình chính
ORG 0030H; Nhấn bắt đầu chương trình sau bảng
vector ngắn
MOV A, #7FH ; Lưu giá trị 01111111 vào thanh ghi tạm
A
MOV R0, #14H ; Lưu số vòng lặp tạo trễ cho LED
14H=20
MAIN: ;Nhấn chương trình chính
DJNZ R0, LOOP2 ; R0=R0-1 nhảy tới nhãn LOOP2 để
thực hiện
MOV 80H, #FFH ; 80H là địa chỉ của Port0, set tất cả
các chân của Port0 lên mức logic 1
RL A ; Dịch trái thanh ghi A đang có giá trị 01111111

MOV 80H, A ; Xuất giá trị A ra Port0

MOV R0, #14H ; Nạp lại giá trị tạo trễ cho thanh ghi R0
SJMP MAIN ; Quay lại hàm MAIN

LOOP2:

MOV 8AH, #LOW(15535) ; 8AH Địa chỉ thanh ghi TL0
của Timer0

MOV 8CH, #HIGH(15535); 8CH Địa chỉ thanh ghi TH0
của Timer0

MOV 89H,#01H ; 89H Địa chỉ thanh ghi TMOD dùng để
chọn mode cho Timer0

SETB 8CH ; 8CH Địa chỉ tahnh ghi TR0 set mức logic 1
để bắt đầu chạy Timer0

LOOP3:

JNB 8DH, LOOP3 ; 8DH Địa chỉ của thanh ghi cờ TF0
khi Timer0 tràn cờ sẽ set lên 1 lệnh JNB dùng để check
xem cờ TF0 đã được bật hay chưa nếu không thì sẽ nhảy
tới nhãn LOOP3

CLR 8DH ;Xóa bit cơ để bắt đầu lại Timer0

CLR 8CH ;Tắt Timer0

SJMP MAIN ; Nhảy tới nhãn MAIN

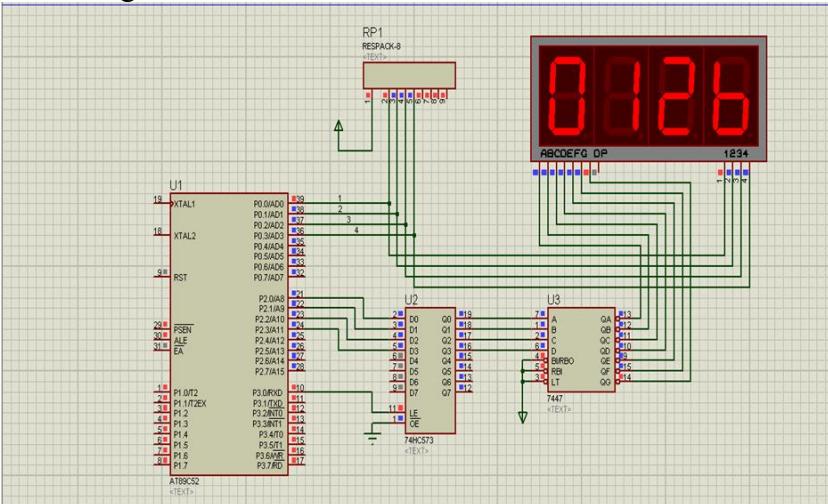
END

Ở chương trình trên kết nối LED theo dương chung nên
việc điều khiển là xuất ra tín hiệu logic 0 LED sẽ sáng.

7.2.2 Giao tiếp với LED 7 đoạn

LED 7 đoạn là thiết bị hiển thị thường được sử dụng trong
các mạch vi điều khiển được sử dụng để hiển thị các thông số
như: thời gian, nhiệt độ, độ ẩm, đếm xung hoặc hiển thị các số

nhập vào từ bàn phím, các số trong quá trình tính toán. LED 7 đoạn có đặt trung là có 7 đoạn LED sáng tắt luân phiên để tạo ra các số mong muốn. Trong ví dụ này sẽ hướng dẫn sinh viên cách lập trình điều khiển 4 LED 7 đoạn bằng vi điều khiển 8051 theo nguyên lý quét LED. Sơ đồ nguyên lý của mạch được thể hiện trong Hình 7-2.



Hình 7-2 Sơ đồ giao tiếp với 3 LED 7 đoạn

Trong sơ đồ nguyên lý này LED 7 đoạn được sử dụng là loại 4 số đã được nối chung các đoạn với nhau và chân nguồn cho mỗi LED 1, 2, 3, 4 là Anode chung. Thông thường trong thực tế các chân Anode chung này sẽ được nối với vi điều khiển thông qua 1 transistor để đảm bảo nguồn cung cấp cho LED hoạt động. Trong sơ đồ kết nối trên có sử dụng IC 74HC573 để chốt dữ liệu và IC 7447 để giải mã, vì Port0 có cấu trúc cực máng hở (Open Drain) nên dùng thêm trở kéo để điều khiển quét LED.

Để có thể điều khiển được LED 7 đoạn trước hết phải hiểu được quét LED là gì. Quét LED là cho từng LED sáng nhưng với tốc độ nhanh mắt người sẽ cảm nhận được cả 4 LED đều sáng cùng 1 lúc, nguyên lý này giống như máy chiếu phim ảnh thời xưa.

Chương trình điều khiển được thiết kế theo nguyên tắc quét LED hiển thị ra số 0126.

Chương trình mẫu:

```
ORG 000H ; Địa chỉ bắt đầu chương trình chính
MOV R0,#2 ; Lưu giá trị tạo trễ vào thanh ghi R0 vì quét
với tốc độ cao nên giá trị tạo trễ thường là rất nhỏ
LJMP MAIN ; Nhảy tới nhãn MAIN thực hiện
ORG 0030H ; Địa chỉ bắt đầu chương trình sau địa chỉ
các vector ngắt
MAIN:
;So 0
MOV 80H,#1 ; set chân P0.0 lên 1 để hiển cho LED số
1
MOV A0H,#00 ;xuất số 0 ra Port2
SETB B0H ; Set chân P3.0 lên mức logic 1
LCALL TRE ;Gọi hàm tạo trễ để chờ chốt dữ liệu
CLR B0H ;Kéo chân P3.0 xuống mức logic 0 để
xuất dữ liệu sang IC giải mã 7447 để hiển thị ra LED số 1
;So 1
MOV 80H,#2 ;set chân P0.1 lên 1 để hiển cho LED số 2
MOV A0H,#1 ;xuất số 1 ra Port2
SETB B0H ;Set chân P3.0 lên mức logic 1
LCALL TRE ;Gọi hàm tạo trễ để chờ chốt dữ liệu
CLR B0H ; Kéo chân P3.0 xuống mức logic 0 để xuất
dữ liệu sang IC giải mã 7447 để hiển thị ra LED số 2
;So 2
```

MOV 80H,#4 ; set chân P0.2 lên 1 để hiển cho LED số 3

MOV A0H,#2 ;xuất số 2 ra Port2

SETB B0H ;Set chân P3.0 lên mức logic 1

LCALL TRE ;Gọi hàm tạo trẽ để chờ chốt dữ liệu

CLR B0H ,Kéo chân P3.0 xuống mức logic 0 để xuất dữ liệu sang IC giải mã 7447 để hiển thị ra LED số 3

;So 6

MOV 80H,#8

MOV A0H,#6 ;

SETB B0H

LCALL TRE ;

CLR B0H;

LJMP MAIN ; Quay lại nhãn MAIN để quét liên tục TRE:

LOOP2:

MOV 8AH, #LOW(50536) ; 8AH Địa chỉ thanh ghi TLO

MOV 8CH, #HIGH(50536); 8CH Địa chỉ thanh ghi TH0

MOV 89H,#01H ; 89H Địa chỉ thanh ghi TMOD để chọn mode cho Timer0

SETB 8CH ; 8CH địa chỉ thanh ghi TR0 set lên mức logic 1 để bắt đầu Timer0

LOOP3:

JNB 8DH, LOOP3 ; 8DH Địa chỉ thanh ghi cờ TF0 dùng để check xem Timer0 đã tràn hay chưa nếu chưa thì quay lại nhãn LOOP3 để chờ

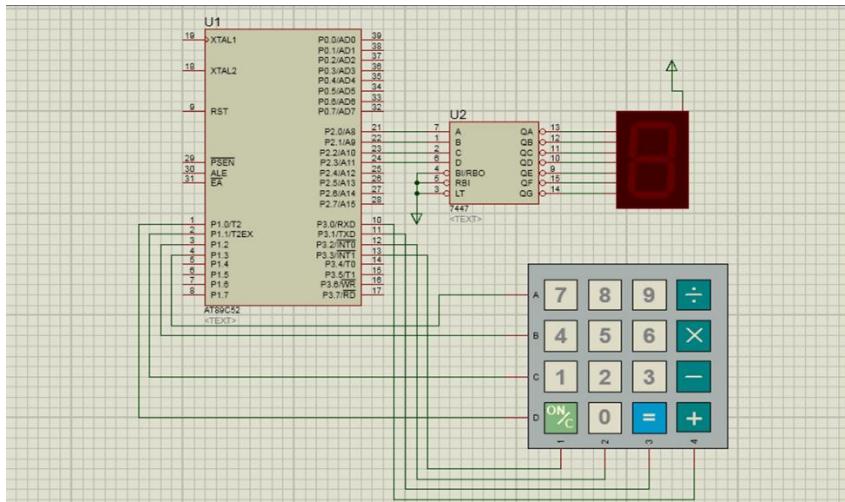
CLR 8DH ; xóa bit cờ TF0

CLR 8CH ; Tắt Timer0

RET ; Thoát hàm tạo trẽ

7.3 Giao tiếp với bàn phím 4x4

Bàn phím sử dụng nhiều trong các thiết bị trong cuộc sống từ những thiết bị đơn giản như máy tính bỏ túi , hay đến các thiết bị như điện thoại di động ,nút bấm điều khiển đầu DVD . Đặc điểm của bàn phím 4x4 là các bút tinh được nối với nhau theo cột chung và hàng chung, vì vậy khi lập trình thì người dùng có thể quét theo hàng hoặc theo cột tùy thích. Nếu như quét theo hàng thì sẽ đọc giá trị theo cột và ngược lại.Trong ví dụ này sẽ hướng dẫn sinh viên quét phím và hiển thị số mà trên bàn phím có hiển thị lên LED 7 đoạn. Sơ đồ kết nối như trong hình sau.



Hình 7-3 Sơ đồ giao tiếp với bàn phím 4x4

Sơ đồ kết nối hàng bàn phím 4x4 được nối với Port3 còn cột được nối với Port1. LED 7 đoạn sử dụng IC 7447 để giải mã

Để có thể giao tiếp được với bàn phím 4x4 cần phải hiểu quét phím là gì. Quét phím gần giống với quét LED, quét từng hàng hoặc cột giống như quét LED nhưng mỗi lần quét ta phải đọc giá trị để kiểm tra xem phím nào được bấm hay không từ đó xuất giá trị ra LED 7 đoạn.

Chương trình mô phỏng quét bàn phím 4x4

ORG 00H ; Nhấn bắt đầu chương trình

*ORG 0030H ; Nhấn bắt đầu chương trình sau vector ngắt
MAIN: ; Nhấn chương trình chính hàm MAIN*

MOV 90H,#00 ; Đưa Port1 về 0 để quét LED

*MOV A,B0H ; Đọc giá trị Port3 để kiểm tra nhấn phím
hay không*

*ANL A,#0FH ; and Port3 với 00001111 để kiểm tra 4
chân đầu của Port3 vì sơ đồ kết của keypad 4x4 có cột nối
với 4 chân đầu của Port3*

*CJNE A,#0FH,MAIN ; so sánh nếu Port3 không có gì
thay đổi thì quay về MAIN để chờ có sự thay đổi từ phím
nhấn*

CHONGDOI: ; Hàm chống dội phím

LCALL DELAY ; gọi hàm tạo trễ

MOV A,B0H ; Đọc giá trị Port3

*ANL A,#0FH ; And Port3 với 00001111 để kiểm tra 4
chân đầu của
Port3*

*CJNE A,#0FH,DUOCNHAN ; nếu có sự thay đổi ở Port3
thì xác nhận là có phím nhấn và nhảy tới Nhấn
DUOCNHAN để xử lý*

*SJMP CHONGDOI ; Quay về hàm CHONGDOI
DUOCNHAN:*

LCALL DELAY ; Gọi hàm trễ
MOV A,B0H ; Đọc giá trị Port3
ANL A,#0FH ; Kiểm tra giá trị thay đổi ở Port3
CJNE A,#0FH,KIEMTRACOT ; Nếu có thay đổi nhảy
trởi hàm KIEMTRACOT để kiểm tra cột được nhấn
SJMP CHONGDOI ; Không có sự thay đổi nhảy về hàm
CHONGDOI để chờ
KIEMTRACOT:
MOV 90H,#0FEH ; Đưa hàng 1 về 0 để check xem nút
nào được bấm ở hàng 1
MOV A,B0H ; đọc giá trị ở Port3 để xác định phím được
nhấn
ANL A,#0FH ; kiểm tra xem thực sự có giá trị được nhấn
hay không
CJNE A,#0FH,SCAN_R0 ; Nếu có phím được nhấn ở
hàng 1 thì nhảy tới Hàm SCAN_R0 để xử lý xác định vị trí
nút nhấn
MOV 90H,#0FDH ; Đưa hàng 2 về 0 để check xem nút
nào được bấm ở hàng 2

MOV A,B0H
ANL A,#0FH
CJNE A,#0FH,SCAN_R1
MOV 90H,#0FBH
MOVA,B0H
ANL A,#0FH
CJNE A,#0FH,SCAN_R2
MOV 90H,#0F7H
MOVA,B0H
ANL A,#0FH
CJNE A,#0FH,SCAN_R3
LJMP CHONGDOI ; Sau khi hoàn tất việc quét phím thì

Nhảy về Hàm CHONGDOI để chờ phím nhấn tiếp theo

DELAY:

LOOP2:

*MOV 8AH, #LOW(55536) ; 8AH địa chỉ thanh ghi
TL0*

*MOV 8CH, #HIGH(55536); 8CH địa chỉ thanh ghi
TH0*

MOV 89H,#01H ; 89H địa chỉ thanh ghi TMOD

SETB 8CH ; 8CH địa chỉ thanh ghi TR0

LOOP3:

JNB 8DH, LOOP3 ; 8DH thanh ghi cờ tràn TF0

CLR 8DH ; xóa cờ tràn TF0 cho Timer0

CLR 8CH ; Tắt Timer0

RET ; Thoát khỏi hàm DELAY

SCAN_R0:

*MOV DPTR,#MAHANG0 ; Lấy địa chỉ ở mảng
MAHANG0 để lưu vào thanh ghi DPTR*

*LCALL SCAN ; Nhảy tới hàm SCAN để hiển thị ra LED
7 đoạn*

LJMP MAIN ; Quay về lại hàm MAIN để chờ phím nhấn

SCAN_R1:

MOV DPTR,#MAHANG1

LCALL SCAN

LJMP MAIN

SCAN_R2:

MOV DPTR,#MAHANG2

LCALL SCAN

LJMP MAIN

SCAN_R3:

MOV DPTR,#MAHANG3

LCALL SCAN

LJMP MAIN

SCAN:

RRC A ; Quay phải qua cờ nhớ C giá trị của A

JNC MATCH ; Nhảy tới hàm MATCH nếu như cờ C=0 để hiển thị Nếu như có tín hiệu bằng 0 tức là có nhấn phím

INC DPTR ; tăng con trỏ DPTR lên 1

SJMP SCAN; Nhảy về lại hàm SCAN

MATCH:

MOV A,#0; Gán A=0

MOVC A,@A+DPTR ; Giá trị hiển thị bằng địa chỉ của con trỏ DPTR + giá trị A

MOV A0H,A ; Xuất giá trị ra Port2 để hiển thị ra LED 7

Đoạn

RET ;Thoát khỏi hàm SCAN

ORG 300H ; Nhấn nút lưu giá trị mảng

MAHANG0: DB '0','0','0','0'

MAHANG1: DB '0','3','2','1'

MAHANG2: DB '0','6','5','4'

MAHANG3: DB '0','9','8','7'

END

Nếu chỉ mô phỏng trên phần mềm proteus thì ta có thể bỏ qua hàm CHONGDOI ở trên , kết quả mô phỏng sẽ không có gì thay đổi . Trong thực tế nếu như không có cơ chế chống dội phím thì phím bấm sẽ bị nhiễu và làm sai đi kết quả. Việc chống nhiễu có thể xử lý bằng phần cứng và cũng có thể xử lý bằng phần mềm như đoạn code mẫu vừa trình bày ở trên.

7.4 Giao tiếp ADC/DAC

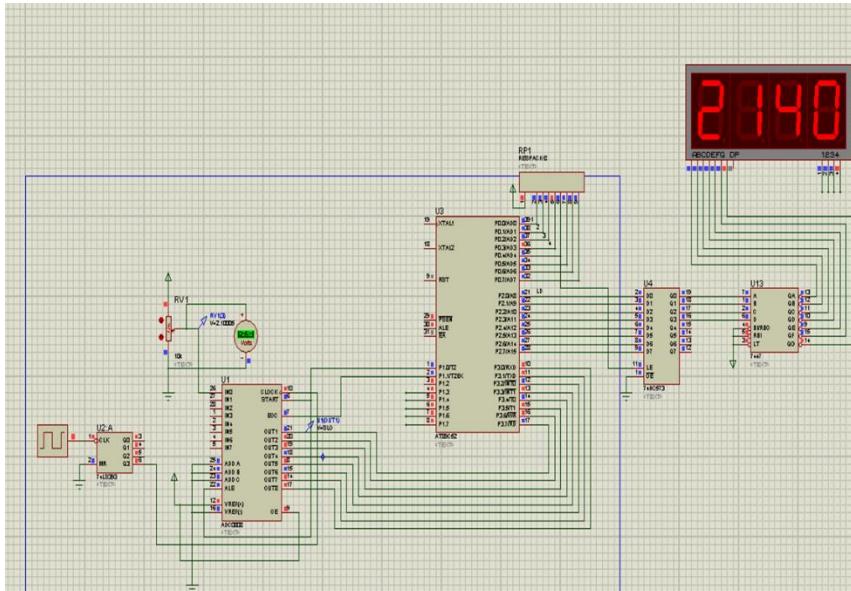
Giao tiếp ADC/DAC là viết tắt của Analog Digital Converter và Digital Analog Converter . Để hiểu chúng trước

tiên ta phải hiểu Analog là gì. Analog là tín hiệu tương tự như âm thanh con người, băng đĩa nhạc hay bất kỳ loại tín hiệu âm thanh nào có trong cuộc sống cũng là một tín hiệu tương tự. Tín hiệu điện cũng là một loại tín hiệu tương tự chúng có biên độ khác nhau ở mỗi thời điểm . Để đưa các tín hiệu tương tự đó vào trong các vi điều khiển để xử lý thì ta cần chuyển chúng về tín hiệu số Digital. Digital là loại tín hiệu 0 và 1 mà máy tính hay vi điều khiển có thể hiểu được nên việc chuyển đổi này rất quan trọng và nó được áp dụng rất nhiều trong thực tế.

Ứng dụng ADC/DAC trong cuộc sống có rất nhiều như xử lý để đó được nhiệt độ của môi trường áp dụng trong khí tượng dự báo thời tiết. Trong y học thì nó áp dụng trong các máy đo nhịp tim , nồng độ và hơi thở . Trong giải trí ta có thể bắt gặp được các dàn LED nháy theo nhạc cũng sử dụng ADC hoặc cao cấp hơn nữa là nhạc nước.

7.4.1 Giao tiếp ADC

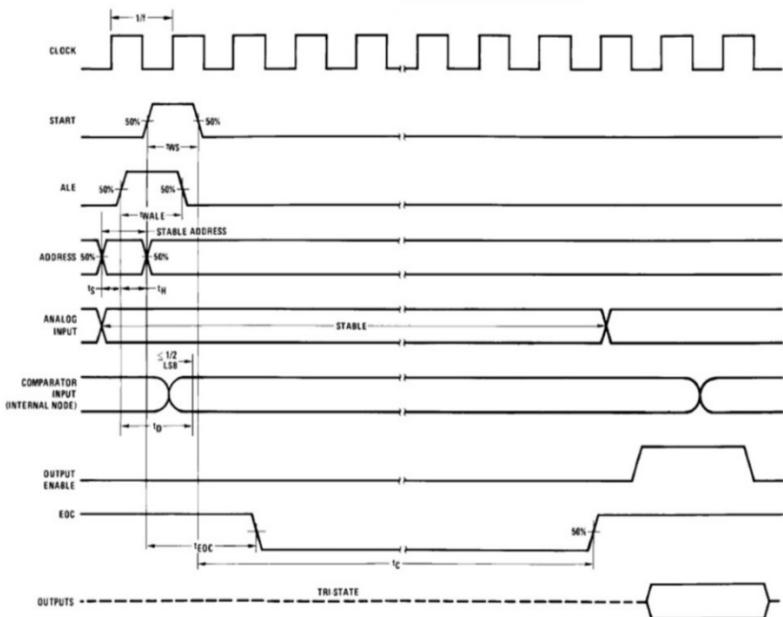
Thiết kế mạch mô phỏng giao tiếp ADC như Hình 7-4



Hình 7-4 Sơ đồ giao tiếp ADC

Trong mạch mô phỏng ở Hình 7-4 trên ta sử dụng clock 640 Khz để tạo xung lock cho IC 74LS393. IC 74LS393 dùng để điều chế clock chuẩn cho IC xử lý ADC ADC0808. Mạch mô phỏng hiển thị LED bảy đoạn được kết nối giống với bài trước. Sơ đồ kết nối chân ADC0808 với vi điều khiển được kết nối như trong hình.

Để có thể giao tiếp được với ADC0808 ta cần hiểu cơ chế và cách thức chuyển đổi của IC này.



Hình 7-5 Nguyên lý chuyển đổi ADC

Trong sơ đồ Hình 7-5 trên có thể thấy tín hiệu START và ALE xảy ra gần như cùng 1 lúc nên ta có thể kết nối 2 chân này lại với nhau, như vậy ta chỉ cần tạo đỗ trễ hợp lý để tín hiệu START và ALE đồng bộ giống như hình trên thường ta tạo trễ 120us là phù hợp để điều khiển.Các biến địa chỉ kênh chuyển đổi ta thiết lập bằng phần cứng bằng cách nối 3 chân ADD A, ADD B, ADD C nối xuống đất (GND) ứng với mức logic 0 , như vậy giá trị ADDRESS là 000 ứng với kênh chuyển đổi số 1. Để có thể đọc giá trị ADC0808 với địa chỉ đã chọn ta cần xử lý chúng trước khi hiển thị ra LED 7 đoạn. Vì ADC0808 có độ phân giải 8bit nên giá trị sẽ nằm trong khoảng từ 0-255 và điện áp ngõ vào ở chân In0 của ADC0808 là từ 0-5V nên ta sẽ có công thức

$5/255 \sim 19.6\text{mV}$ vì vậy để đọc giá trị điện áp (mV) ta sẽ nhân giá trị này với 19.6 . Tuy nhiên số 19.6 là con số rất khó xử lý trong vi điều khiển nên ta sẽ nhân với 20 và chấp nhận sai số nhỏ.

Chương trình mẫu:

```
#include <sfr51.inc>
ORG 000H ; Nhấn bắt đầu chương trình chính

LJMP MAIN ; Nhảy tới hàm MAIN để thực hiện
ORG 0030H ; Nhấn bắt đầu chương trình sau bảng vector
ngắt
MAIN:
CLR 90H ; Dưa chân P1.0 về 0 để chuẩn bị set tín hiệu
ALE và START
ACALL TRE ; gọi hàm tạo trễ
SETB 90H ; Set tín hiệu ALE và START lên 1 để chuẩn bị
bắt đầu quá trình chuyển đổi
ACALL TRE ; Gọi hàm tạo trễ
CLR 90H ; bắt đầu qua trình chuyển đổi
ACALL TRE
LOOP:
JNB 91H,LOOP ; Chờ chân P1.1 được bật lên 1 chưa nếu
được set lên 1 có nghĩa là quá trình chuyển đổi đã hoàn
tất nếu không quay về LOOP chờ
MOV A,B0H ; đọc giá trị chuyển đổi ADC vào thanh ghi A
MOV B,#100; Nạp giá trị 100 cho thanh ghi B để tách số
hàng trăm hiển thị ra LED 7 đoạn
DIV AB ;Lấy giá trị ADC chia cho 100 và A sẽ lưu phần
nguyên và B lưu phần dư
MOV R1,A ; Lưu số hàng trăm vào thanh ghi R1
MOV A,B ; Lưu phần dư còn lại vào thanh ghi A để tách
```

số tiếp

MOV B,#10 ; Nạp giá trị 10 cho thanh ghi B để tách số hàng trực

DIV AB

MOV R2,A ; Lưu hàng trực vào thanh ghi R2

MOV R3,B;Lưu hàng đơn vị vào thanh ghi R3

MOV B,#2 ; Nạp giá trị 2 cho thanh ghi B

MOV A,R3 ; nạp giá trị hàng đơn vị vào thanh ghi A

MUL AB ; Nhân giá trị hàng đơn vị với 2 thể hiện mức điện áp là V

MOV B,#10 ;Nạp cho thanh ghi B giá trị 10

DIV AB ; Tách số đơn vị sau khi nhân 2 nếu như lớn hơn 9

MOV R3,B ; lưu giá trị đơn vị vào thanh ghi R3

MOV R0,A ; Lưu số nhớ cho hàng chục

MOV B,#2 ; Nạp giá trị 2 cho thanh ghi B

MOV A,R2; nạp giá trị hàng chục vào thanh ghi A

MUL AB; Nhân giá trị hàng chục với 2 thể hiện mức điện áp là V

MOV B,R0 ; Nạp số nhớ hàng chục vào thanh ghi B

ADD A,B; Cộng giá trị hàng trực và số nhớ

MOV B,#10;Nạp cho thanh ghi B giá trị 10

DIV AB ; Tách số hàng chục nếu như lớn hơn 9

MOV R2,B ;Lưu giá trị hàng chục vào thanh ghi R2

MOV R0,A ; Lưu số nhớ hàng chục vào thanh ghi R0

MOV B,#2

MOVA,R1

MUL AB

MOV B,R0

ADD A,B

MOV R1,A

MOV 80H,#1; Chọn LED 7 đoạn đầu tiên để hiển thị
MOV A0H,R1 ; xuất giá trị hàng trăm ra Port2
SETB 84H ; Set chân chót 74HC573 lên 1
CLR 84H ;Đưa chân chót về 0 để xuất dữ liệu ra LED
ACALL TRE ; Gọi hàm tạo Trẽ
MOV 80H,#2
MOV A0H,R2
SETB 84H
CLR 84H
ACALL TRE
MOV 80H,#4
MOV A0H,R3
SETB 84H
CLR 84H
ACALL TRE
MOV 80H,#8
MOV A0H,R4
SETB 84H
CLR 84H
ACALL TRE
SJMP MAIN

TRE:

LOOP2:

MOV 8AH, #LOW(65406) ; 8AH Địa chỉ thanh ghi TL0 của Timer0

MOV 8CH, #HIGH(65506); 8CH Địa chỉ thanh ghi TH0 của Timer0

MOV 89H,#01H ; 89H Địa chỉ thanh ghi TMOD

SETB 8CH ; 8CH Địa chỉ thanh ghi TR0

LOOP3:

JNB 8DH, LOOP3 ; 8DH Chờ bit cờ TF0 bật lên 1

CLR 8DH ;xóa bit cờ

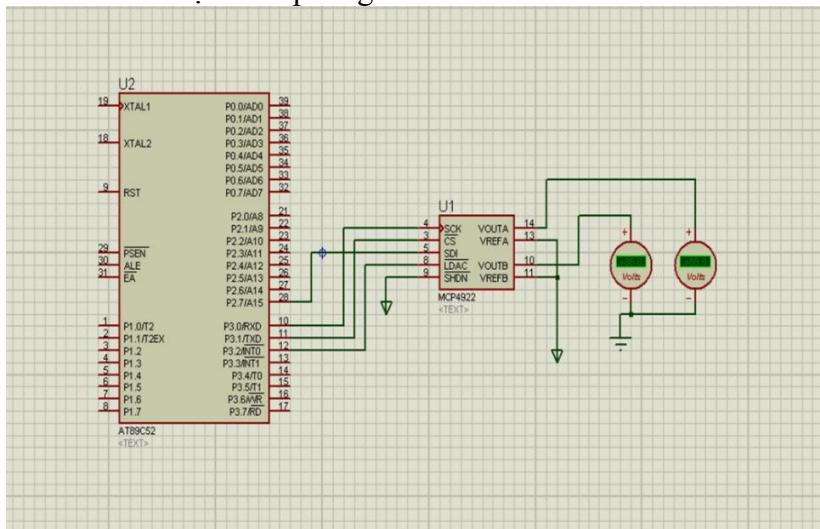
CLR 8CH ;dừng Timer0

RET ; thoát hàm tạo trẽ

END

7.4.2 Giao tiếp DAC

Thiết kế mạch mô phỏng DAC như Hình 7-6 sau:



Hình 7-6 Sơ đồ giao tiếp DAC

Trong thiết kế mô phỏng trong Hình 7-6 trên ta sử dụng DAC MCP4922 để mô phỏng và dùng 2 Volt kế để đo điện áp đầu ra của DAC. Điện áp tham chiếu VREFA và VREFB được kéo lên 5V .

Để có thể giao tiếp được với DAC MCP4922 ta cần biết được cách thức động và cách truyền dữ liệu cho IC này. Cách thức hoạt động được mô tả như Hình 7-7 sau

Chip Select (\overline{CS})

\overline{CS} is the chip select input, which requires an active-low signal to enable serial clock and data functions.

Serial Clock Input (SCK)

SCK is the SPI compatible serial clock input.

Serial Data Input (SDI)

SDI is the SPI compatible serial data input.

Latch DAC Input (LDAC)

LDAC (the latch DAC synchronization input) transfers the input latch registers to the DAC registers (output latches) when low. Can also be tied low if transfer on the rising edge of \overline{CS} is desired.

Hardware Shutdown Input (\overline{SHDN})

\overline{SHDN} is the hardware shutdown input that requires an active-low input signal to configure the DACs in their low-power Standby mode.

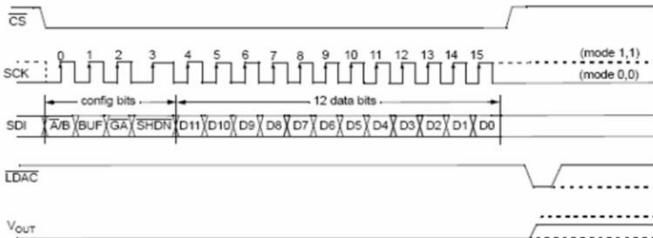
DAC_x Outputs (V_{OUTA} , V_{OUTB})

V_{OUTA} and V_{OUTB} are DAC outputs. The DAC output amplifier drives these pins with a range of V_{SS} to V_{DD} .

DAC_x Voltage Reference Inputs

(V_{REFA} , V_{REFB})

V_{REFA} and V_{REFB} are DAC voltage reference inputs. The analog signal on these pins is utilized to set the reference voltage on the string DAC. The input signal can range from V_{SS} to V_{DD} .



A/B: DAC_A or DAC_B Select bit

- 1 = Write to DAC_B
- 0 = Write to DAC_A

BUF: V_{REF} Input Buffer Control bit

- 1 = Buffered
- 0 = Unbuffered

GA: Output Gain Select bit

- 1 = $1x$ ($V_{OUT} = V_{REF} * D/4096$)
- 0 = $2x$ ($V_{OUT} = 2 * V_{REF} * D/4096$)

SHDN: Output Power Down Control bit

- 1 = Output Power Down Control bit
- 0 = Output buffer disabled, Output is high impedance

D11:D0: DAC Data bits

12 bit number "D" which sets the output value. Contains a value between 0 and 4095.

Hình 7-7 Nguyên lý chuyển đổi DAC

Ta có thể thấy DAC MCP4922 được giao tiếp theo chuẩn SPI 16bit chính vì vậy việc đồng bộ xung clock trong việc truyền là hết sức quan trọng. Trong sơ đồ trên xung clock được kết nối với chân P3.0 của vi điều và giá trị được truyền đi ở cạnh lên của xung clock nên ban đầu ta sẽ phải set chân P3.0 và 0 và sau đó đưa dữ liệu vào chân SDI được gắn với chân P2.7 của vi điều khiển. Lúc này ta đưa chân P3.0 lên 1 thì dữ liệu ở chân P2.7 sẽ được truyền, tương tự như vậy khi đã truyền hết 16 bit ta sẽ đưa chân LDAC xuống mức logic 0 để báo kết thúc quá trình truyền.

Chương trình mẫu truyền giá trị FFH(5V) cho DAC MCP4922

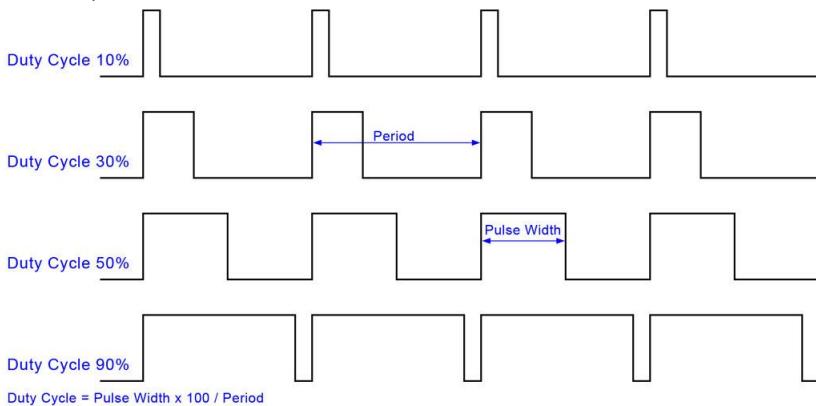
```
#include <sfr51.inc>
ORG 00H ; Nhấn bắt đầu chương trình chính
MOV R1,#0011111B ; Thiết lập giá trị 8bit đầu
DAC
MOV R2,#1111111B ; Thiết lập giá trị 8bit sau DAC
MOV R3,#8 ; Nạp giá trị 8 vào thanh ghi R3 có vai trò
kiểm tra xem truyền đủ 8 bit hay chưa
LJMP MAIN ; Nhảy tới hàm MAIN thực hiện
ORG 0030H ; Nhấn bắt đầu chương trình sau bảng vector
ngắt
MAIN:
CLR B1H ; Đưa chân CS về 0
SETB B2H ; Đưa chân LDAC lên 1 để sẵn sàng quá trình
transmission dữ liệu
MOV A,R1; Nạp 8 bit đầu vào thanh ghi A để truyền dữ
liệu
TRUYEN_HIGH:
CLR B0H ; đưa chân clock về 0
MOV A0H,A ; xuất dữ liệu cần truyền ra Port2
```

SETB B0H ; Đưa xung clock lên 1 để truyền dữ liệu đi
 RL A ; dịch trái A để chuẩn bị truyền bit tiếp theo
 DJNZ R3,TRUYEN_HIGH ; Kiểm tra truyền đủ 8 bit
 đầu hay chưa nếu chưa thì nhảy tới hàm TRUYEN_HIGH
 để tiếp tục truyền
 MOV R3,#8 ; Nạp lại giá trị check cho thanh ghi R3
 MOV A,R2 ; Nạp 8bit sau vào thanh ghi A để chuẩn bị
 truyền
TRUYEN_LOW:
 CLR B0H; đưa chân clock về 0
 MOV A0H,A ; xuất dữ liệu cần truyền ra Port2
 SETB B0H;Đưa xung clock lên 1 để truyền dữ liệu đi
 RL A;dịch trái A để chuẩn bị truyền bit tiếp theo
 DJNZ R3,TRUYEN_LOW;Kiểm tra truyền đủ 8 bit sau
 hay chưa nếu chưa thì nhảy tới hàm TRUYEN_LOW để
 tiếp tục truyền
 MOV R3,#8 ; Nạp lại giá trị check cho thanh ghi R3
 SETB B1H ; Đưa chân CS lên 1
 CLR B2H ; đưa chân LDAC về 0 để kết thúc quá trình
 truyền
 LJMP MAIN
 END

7.5 Giao tiếp với motor

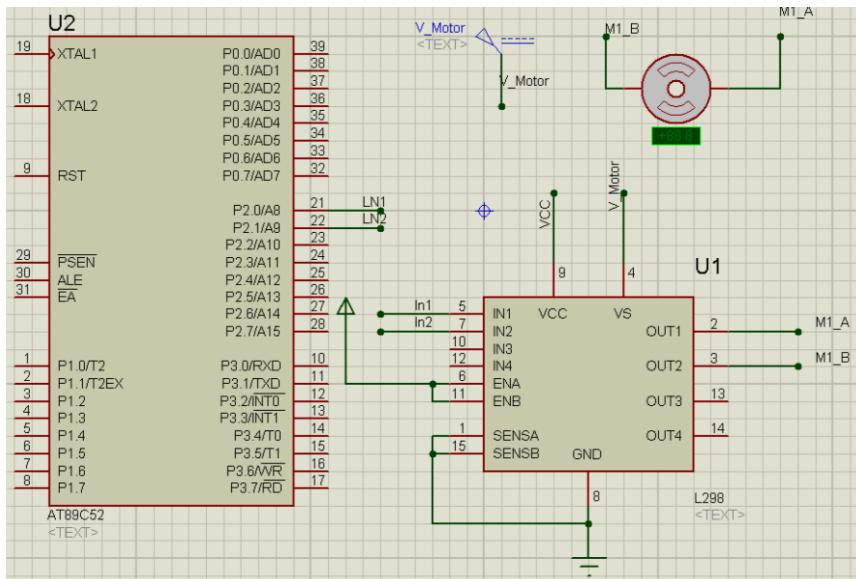
Băm xung hay còn gọi là điều khiển độ rộng xung – PWM là một giải thuật cho phép ta điều chỉnh tỉ lệ của xung cao và xung thấp của 1 chu kỳ xung clock. Nói một cách đơn giản trong 1 chu kỳ xung clock ta thường quen với việc 1 nửa chu kỳ sẽ là xung cao, nửa còn lại sẽ là xung thấp, băm xung sẽ giúp chúng ta thay đổi tỉ lệ cao thấp này để cho ra 1 xung có độ rộng xung

cao, thấp tùy ý như Hình 7-8. Băm xung chủ yếu được dùng trong các ứng dụng điều khiển tốc độ động cơ, điều khiển động cơ bước, ...



Hình 7-8 Nguyên lý băm xung

Sau đây là một ví dụ đơn giản nhất cho việc ứng dụng giải thuật băm xung để điều khiển tốc độ động cơ. Trong ví dụ ta sử dụng IC cầu H L298 để cấp nguồn cho động cơ hoạt động. Khi hai ngõ vào IN1 và IN2 của L298 có sự khác nhau về mức logic thì các ngõ OUT1 và OUT2 tương ứng sẽ có sự chênh lệch điện áp làm động cơ quay như Hình 7-9. Trong code nếu muốn thay đổi tốc độ động cơ bạn có thể tùy chỉnh giá trị R2 từ 0x01 đến 0x0B tương ứng (0% đến 100%). Chương trình dùng ngắt timer để tạo 1 xung clock có chu kỳ là [thời gian 1 ngắt * (R0 - 1)] với độ rộng xung cao là R1 được nạp lại sau mỗi chu kỳ với giá trị của R2. Trong ví dụ chương trình tạo ra 1 ngắt mỗi 20ms với R0 là 11 tương đương với chu kỳ 200ms. Như vậy độ rộng xung cao sẽ có độ phân giải là 1/10 2/10 ... 10/10 chu kỳ.



Hình 7-9 Sơ đồ mô phỏng điều khiển motor

Chương trình mẫu:

```

ORG 0000H           ;Reset
ORG 00h
LJMP MAIN

```

```

ORG 01bh //Địa chỉ ngắt timer 1
LJMP TIMER1

```

```
ORG 0030h
```

MAIN:

```

MOV R0, #0BH // nạp giá trị ban đầu cho R0
MOV R2, #02H // nạp giá trị ban đầu cho R2

```

*MOV R1, #01H // nạp giá trị ban đầu cho R1
CLR P2.0 // Kéo các chân điều khiển về 0
CLR P2.1
MOV TMOD,#10H // Timer 1 mode 1 – 16 bit
MOV IE, #88H // Cho phép ngắt và cho phép timer 1 ngắt
MOV TL1, #0D0H // nạp giá trị cho các thanh ghi trong timer 1
MOV TH1, #07H
SETB TF1 // Nhảy vào ngắt*

SJMP \$

TIMER1:

*CLR TF1 // Xóa cờ ngắt
CLR TRI // Dừng timer
MOV TL1, #0D0H // Nạp lại giá trị cho timer 1
MOV TH1, #07H*

DJNZ R0, PWM1 // tạo chu kỳ cho xung clock

MOV R0, #0BH // Khi hết chu kỳ nạp lại giá trị cho các thanh ghi để bắt đầu chu kỳ tiếp theo

MOVA,R2

MOV R1,A

SETB TRI // Cho timer chạy

RETI // Thoát khỏi ngắt

PWM1:

DJNZ R1, PWM2

INC R1

CLR P2.1 // Tạo xung thấp khi R1 đã trù hết

SETB TRI

RETI

PWM2: //R1 > 0 - tạo xung cao

SETB P2.1

SETB TR1

RETI

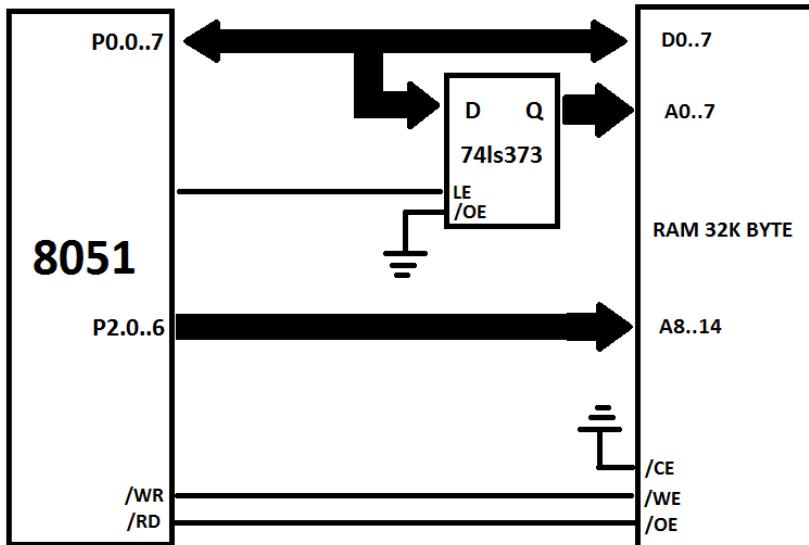
END

7.6 Giao tiếp với bộ nhớ ngoài (RAM)

Bộ nhớ của 8051 là khá thấp, do đó để phục vụ nhu cầu làm việc với các ứng dụng cần không gian lưu trữ lớn ta sẽ phải ghép nối thêm bộ nhớ ngoài cho 8051. Cụ thể trong phần này ta sẽ thực hiện việc ghép nối vi điều khiển 8051 với bộ nhớ RAM 32K x 8bit.

Để có thể ghép nối 8051 và RAM ta cần dùng các port 0 và port 1 làm port địa chỉ. Trong đó port 0 là port ghép dữ liệu và địa chỉ nên ta cần thêm 1 IC chốt 74LS373 để tách đường dữ liệu và địa chỉ (lưu ý khi dùng port 0 để ghép nối bộ nhớ ngoài ta không cần dùng trở kéo). Để điều khiển việc đọc ghi dữ liệu cần sử dụng thêm các chân ALE (chốt địa chỉ) nối với IC chốt, và chân /WR, /RD nối với RAM.

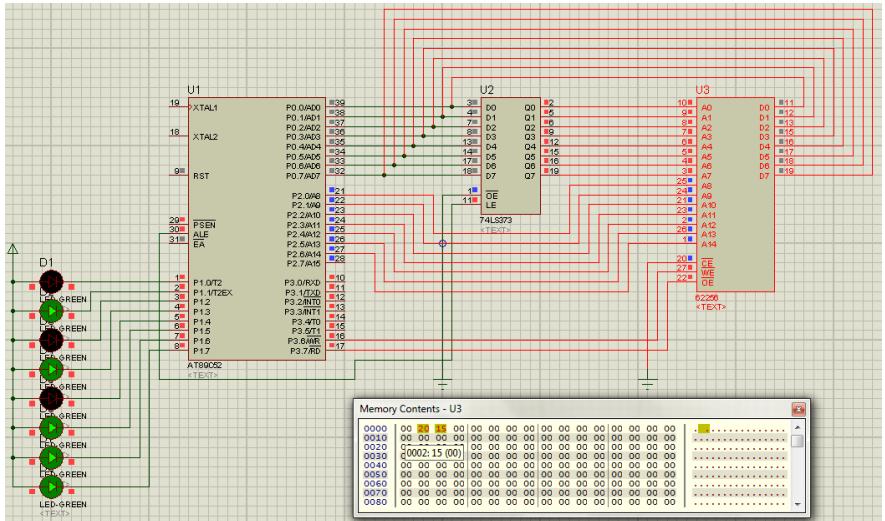
Sơ đồ nguyên lý của việc kết nối vi điều khiển 8051 với RAM ngoài như Hình 7-10 sau:



Hình 7-10 Sơ đồ giao tiếp với RAM ngoài

Để thực hiện việc mô phỏng kết nối với RAM ngoài, ta sẽ tiến hành mô phỏng trên protues với mạch như Hình 7-11. Trong phần code của chương trình này ta sẽ thực hiện các công việc: ghi giá trị #20H vào ô nhớ có địa chỉ #01H trên RAM ngoài và giá trị #15H vào ô nhớ có địa chỉ #02H. Sau đó ta sẽ đọc giá trị từ ô nhớ #02H ra và hiển thị giá trị đọc được lên LED đơn tương ứng với từng bit.

Việc thực hiện đọc/ghi và điều khiển các chân ALE, /RD, /WR là tự động khi sử dụng lệnh đọc ghi dữ liệu từ bộ nhớ ngoài.



Hình 7-11 Mô phỏng giao tiếp với RAM ngoài

Chương trình mẫu:

```
ORG 00h
LJMP MAIN
```

```
ORG 0030h
```

MAIN:

MOV P2,#00H //kéo P2 xuống 0 để không ghi vào đai chỉ cao

MOV R0,#01H //Nạp giá trị địa chỉ vào R1

MOV A,#20H //Nạp dữ liệu vào A

MOVX @R0,A //Chép dữ liệu vào ô nhớ ngoại có địa chỉ chứa trong R0

MOV R0,#02H

```
MOVA,#15H  
MOVX @R0,A
```

```
MOVX A,@R0 //Đọc dữ liệu từ ô nhớ ngoài có địa chỉ  
trong R0 vào A  
MOV P1,A
```

```
SJMP $
```

```
END
```

7.7 Tổng kết

Chương này trình bày một số giao tiếp cơ bản mà vi điều khiển 8051 có thể làm, đây sẽ là những mẫu giao tiếp cơ bản để chúng ta có thể phát triển lên các ứng dụng có ý mà vi điều khiển 8051 có thể hỗ trợ. Những giao tiếp này đã bao gồm các giao tiếp cơ bản như: LED đơn, LED 7 đoạn, bàn phím, ADC, DAC, RAM ngoài ...sẽ giúp sinh viên có thể dựa vào để có thể làm mô phỏng các giao tiếp này bằng phần mềm Proteus.

7.8 Câu hỏi và bài tập

1. Giao tiếp ADC hoạt động theo nguyên lý gì? Thường được sử dụng trong những ứng dụng nào?
2. Nguyên lý điều khiển dây 4 LED 7 đoạn được thực hiện như thế nào?
3. Phân quan trọng nhất trong việc xử lý án phím là gì? Nếu giải pháp khắc phục?

4. Viết chương trình đọc giá trị ADC từ cảm biến nhiệt độ và hiển thị giá trị độ C lên 2 LED 7 đoạn.
5. Viết chương trình nhận giá trị từ bàn phím 4x4 và hiển thị các số được ấn lên LED 7 đoạn.
6. Viết chương trình giao tiếp với LCD 16x2 để hiển thị dòng chữ “Khoa Kỹ thuật May tính”.

Chương 8. Ứng dụng của họ Vi điều khiển 8051

Mục tiêu chương

Mô tả một số ứng dụng đơn giản của vi điều khiển 8051 vào thực tế. Học xong chương này người học sẽ hình dung đầy đủ về việc phát triển một ứng dụng thực tế sử dụng vi điều khiển và có các kỹ năng bước đầu cho việc lập trình, xây dựng ứng dụng trên vi điều khiển.

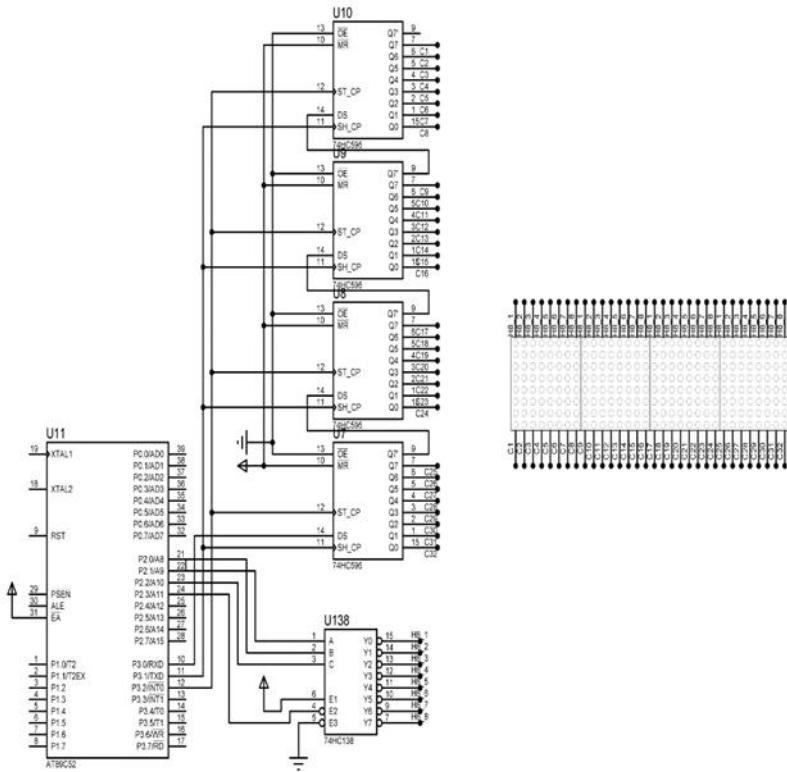
8.1 Ứng dụng điều khiển bảng quảng cáo

8.1.1 Phát biểu ứng dụng

Thiết kế ứng dụng hiển thị bảng quảng cáo dưới dạng ma trận LED 8x32 hiển thị dòng chữ : “TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN”.

8.1.2 Thiết kế mạch mô phỏng

Mạch mô phỏng được thiết kế dựa trên hướng mở rộng số lượng chân của vi điều khiển. Số chân mà vi điều khiển 8051 cung cấp chỉ đáp ứng được 32 chân ứng với 4 Port. Nhưng yêu cầu của ứng dụng là điều khiển LED ma trận 8x32 như vậy ta sẽ cần 8 chân điều khiển hàng và 32 chân điều khiển cột , giải pháp để giải quyết vấn đề không đủ chân điều khiển trên là ta sẽ sử dụng IC ghi và dịch 74hc595 để ghi và dịch. Hàng ta sẽ sử dụng Decoder 74LS138 để giải mã thuận tiện cho việc điều khiển. Sơ đồ thiết kế của giải pháp để xuất như Hình 8-1. Lưu ý đây chỉ là giải pháp để xuất và còn có thể có nhiều giải pháp khác mà người học có thể tự nghĩ ra.



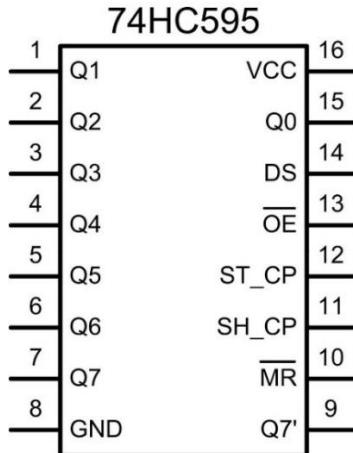
Hình 8-1 Sơ đồ thiết kế giải pháp điều khiển ma trận LED

Để có thể điều khiển được ma trận LED này trước tiên ta cần hiểu cách điều khiển IC ghi và dịch 74HC595 và Decoder 74LS138

a. 74HC595:

- 74HC595 là IC ghi dịch 8bit kết hợp chốt dữ liệu, đầu vào nối tiếp đầu ra song song với sơ đồ chân như Hình 8-2. Đây là IC dịch được sử dụng rất phổ biến trong các ứng dụng điều khiển đèn LED như LED 7 đoạn, LED ma trận. Việc sử dụng IC dịch sẽ giúp tiết

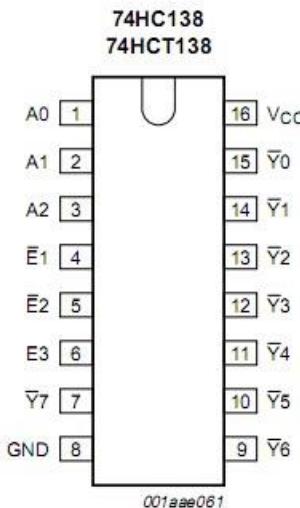
kiêm phần lớn chân của vi điều khiển. Có thể mở rộng điều khiển tùy thích khi ghép nối nhiều IC 79HC595 lại với nhau.



Hình 8-2 Sơ đồ chân IC 79HC595

- Sơ đồ chân:
- + Chân 15 và 1 đến 7 (Q0 đến Q7): Các chân xuất dữ liệu của 74HC595
- + Chân 14: Chân vào dữ liệu nối tiếp
- + Chân 13 : Chân cho phép tích cực ở mức thấp. Khi ở mức cao, tất cả các đầu ra của 74HC595 trở về trạng thái cao, không có đầu ra nào được cho phép.
- + Chân 9: Chân dữ liệu nối tiếp. Nếu dùng nhiều 74HC595 mắc nối tiếp nhau thì chân này đưa vào đầu vào của 74HC595 tiếp theo
- + Chân 11: Chân vào xung clock .
- + Chân 12 : chân chốt dữ liệu.
- + Chân 10: chân reset tích cực mức thấp.
- Hoạt Động:

- + Tại 1 thời điểm xung clock ở chân 11 có sườn 0 -> 1 thì 1 bit ở chân 14 sẽ được đưa vào IC dịch. Chiều dịch dữ liệu là từ Q0 -> Q7
 - + Khi muốn chốt dữ liệu đầu ra ta cần kéo chân 12 từ 0 -> 1 thì dữ liệu sẽ được chốt ở các chân dữ liệu ra.
- b. 74HC138:
- Đây là IC giải mã 3 -> 8 đường tín hiệu với các chân cho loại IC phổ biến 74HC138 như Hình 8-3. Đây là IC được dùng phổ biến trong các ứng dụng quét LED.



Hình 8-3 Sơ đồ chân IC 74HC138

- Các chân A2A1A0 tương ứng với dữ liệu đầu vào (input) và có giá trị từ 000 đến 111
- Các ngõ ra Y7 → Y0 tích cực mức thấp
- Hoạt động: khi ngõ vào A2A1A0 = 000 thì tương ứng với ngõ ra Y0 = 0 các ngõ ra khác bằng 1, A2A1A0 = 001 thì Y1 = 0 các ngõ ra còn lại bằng 1 và tương tự cho các trường hợp còn lại.

- Bảng sự thật (Truth table) cho thấy hoạt động của IC 74HC138 như trong Bảng 8-1.

Bảng 8-1 Bảng sự thật của IC 74HC138

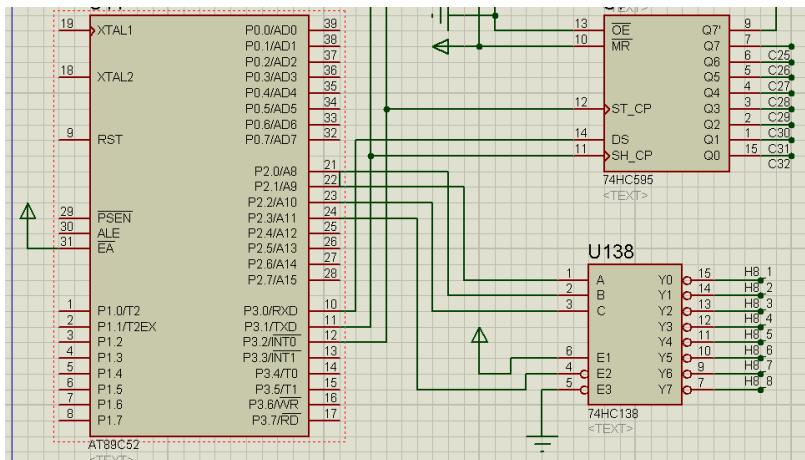
FUNCTION TABLE

INPUTS						OUTPUTS							
\bar{E}_1	\bar{E}_2	E_3	A_0	A_1	A_2	\bar{Y}_0	\bar{Y}_1	\bar{Y}_2	\bar{Y}_3	\bar{Y}_4	\bar{Y}_5	\bar{Y}_6	\bar{Y}_7
H	X	X	X	X	X	H	H	H	H	H	H	H	H
X	H	X	X	X	X	H	H	H	H	H	H	H	H
X	X	L	X	X	X	H	H	H	H	H	H	H	H
L	L	H	L	L	L	L	H	H	H	H	H	H	H
L	L	H	H	L	L	L	L	H	H	H	H	H	H
L	L	H	L	H	L	L	L	L	H	H	H	H	H
L	L	H	H	H	L	H	L	L	L	H	H	H	H
L	L	H	L	L	H	H	L	L	L	L	L	H	H
L	L	H	H	L	H	H	L	L	L	L	L	H	H
L	L	H	H	H	H	H	L	L	L	L	L	L	H
L	L	H	L	H	H	H	L	L	L	L	L	L	H
L	L	H	H	H	H	H	L	L	L	L	L	L	H
L	L	H	H	H	H	H	L	L	L	L	L	L	L

Notes

- 1. H = HIGH voltage level
- L = LOW voltage level
- X = don't care

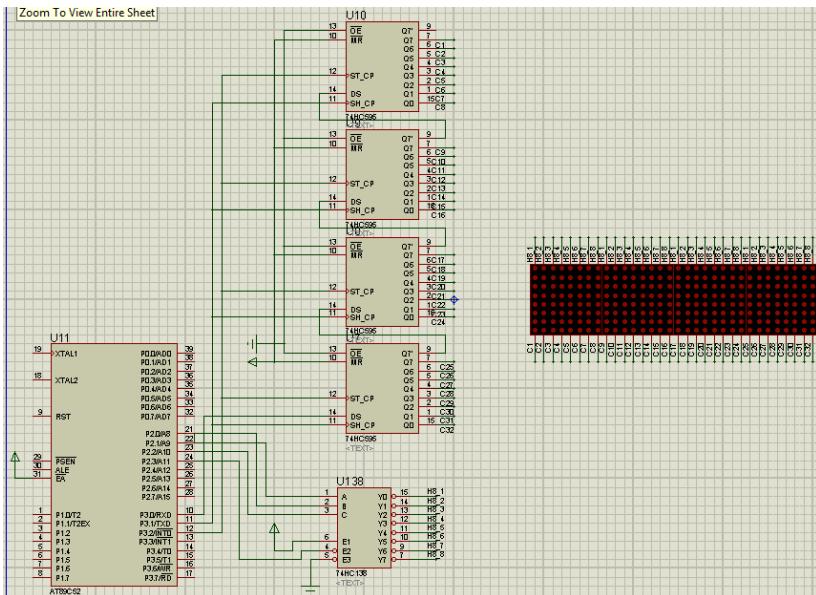
- Kết nối VDK với IC decoder 74HC138 theo Hình 8-4



Hình 8-4 Sơ đồ kết nối IC 74HC138

- c. Mạch mô phỏng LED quảng cáo

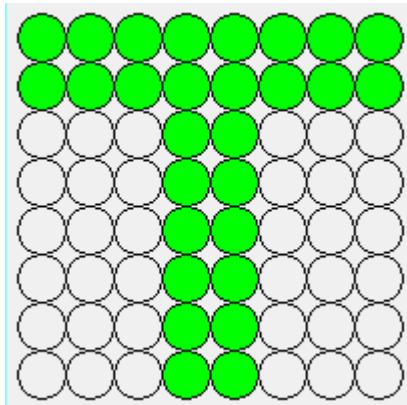
Để kiểm tra giải pháp thiết kế trước khi mua board và gắn các IC trên đó cần sử dụng một phần mềm mô phỏng thử hoạt động. Phần mềm thông dụng và rất tốt cho việc này có thể dùng là Proteus. Sử dụng phần mềm này thiết kế giải pháp mô phỏng như Hình 8-5.



Hình 8-5 Mạch mô phỏng ma trận LED 8x32 trên Proteus

Cách giải mã LED 8x32.

Để có thể giải mã được LED ma trận 8x32 thì trước hết ta sẽ giải mã từng con LED ma trận 8x8 sau đó ta sẽ ghép 4 con LED ma trận 8x8 lại với nhau sẽ được ma trận LED 8x32. Trong ví dụ về cách giải mã này sẽ hướng dẫn sinh viên giải mã chữ "T" như Hình 8-6.



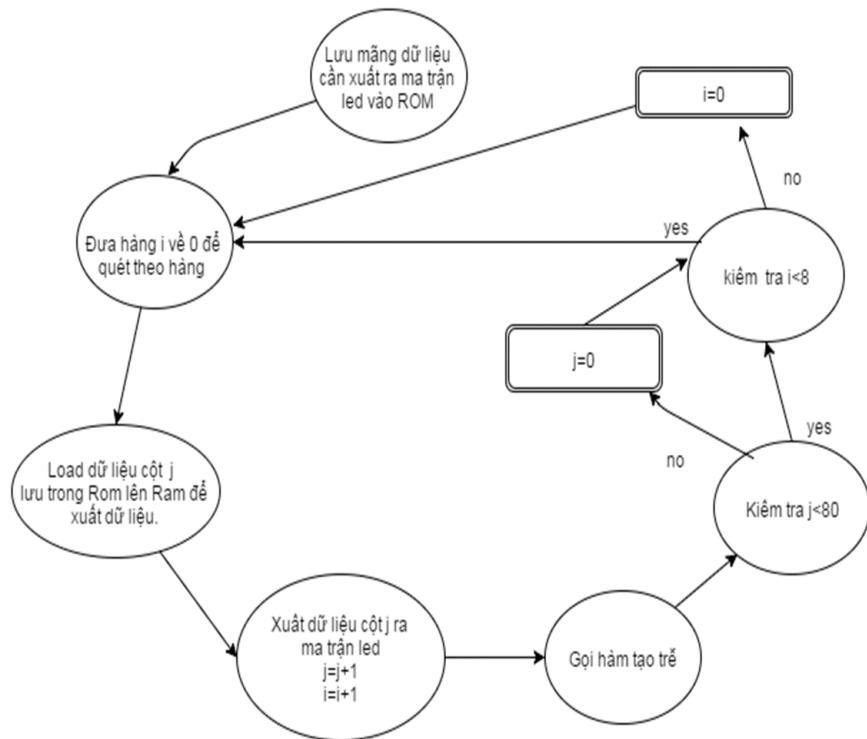
Hình 8-6 Hiển thị chữ "T" trên ma trận led 8x8

Ta có thể thấy nếu như LED được tích cực mức 1 là sáng thì ở cột đầu tiên ta sẽ được giá trị mã hóa như sau 00000011(Binary) ứng với giá trị là 3H(Hexa decimal). Tương tự như vậy cho hết 8 cột ta sẽ được 1 mảng mã hóa chữ "T" như sau: DB 3H,3H,3H,0FFH,0FFH,3H,3H,3H

Để có thể hiển thị được số lượng chữ lớn hơn ta cũng làm tương tự để giải mã.

8.1.3 Chương trình nhúng

Lưu đồ giải thuật của chương trình được trình bày trong Hình 8-7



Hình 8-7 Lưu đồ giải thuật điều khiển LED ma trận

Chương trình nhúng sẽ hiển thị chữ TRƯỜNG ĐH CNTT lên ma trận LED 8x32 như chương trình ngay dưới đây. Trong chương trình này chữ sẽ được dịch từ trái sang phải.

```

#include <sfr51.inc>

;QUETLED Hiển thị 32 cột và 8 hàng
SI BIT P3.0 ;pin 14 74HC595
SCK BIT P3.1 ;pin 11 74HC595
RCK BIT P3.2 ;pin 12 74HC595
;---

```

*ORG 0000H ; Nhấn bắt đầu chương trình chính
LJMP MAIN ; Nhảy tới hàm MAIN thực hiện chương trình
ORG 0030H ; Nhấn bắt đầu sau bảng vector ngắt*

MAIN:

*MOV SP,#60H ;Khai báo địa chỉ Stack Pointer
MOV DPTR,#CNTT ;Địa chỉ chữ đã được mã hóa có thể thay bằng chữ khác*

PLAY:

MOV R1,#-80 ;R1 là địa chỉ trung gian

DICH_TRAI:

MOV R3,#2 ;R3 là tốc độ quét dùng để tạo trễ thường thì số sẽ rất thấp để mắt người có thể nhìn thấy ở đây chọn giá trị là 2

LRAM:

LCALL LOADLENRAM ; Nhảy tới hàm LOADLENRAM để load dữ liệu lên ram để hiển thị

EX_00:

LCALL QUET8HANG ; gọi hàm QUET8HANG

DJNZ R3,EX_00 ; tạo trễ khi quét

MOV R3,#2 ;nạp lại giá trị tạo trễ

INC R1 ; Tăng địa chỉ trung gian lên 1

CJNE R1,#80,EX_01 ; Kiểm tra xem đã hiển thị đủ 80bye dữ liệu chưa nếu chưa nhảy tới hàm EX_01 để thực hiện tiếp

MOV R1,#-80 ; Nạp lại giá trị địa chỉ

LJMP DICH_TRAI ; Quay lại hàm DICH_TRAI để quét liên tục

EX_01: LJMP LRAM ; Nhảy tới hàm LRAM

LJMP PLAY ; Nhảy tới hàm PLAY

LOADLENRAM:

PUSH 00H

```

PUSH 01H
PUSH 02H
MOV R0,#9EH ;Địa chỉ bắt đầu con trỏ lưu vào
thanh ghi R0
MOV A,R1 ; Địa chỉ trung gian
MOV R2,#80 ; số cột hiển thị
LAPL_01:MOV A,R1
    MOVC A, @A+DPTR ;Lấy dữ liệu
    MOV @R0,A ;Ghi vào ram
    INC R0      ;tăng con trỏ địa chỉ lên 1
    INC R1      ;Tăng địa chỉ trung gian
    DJNZ R2,LAPL_01;Kiểm tra xem đủ số cột chưa
    POP 00H
    POP 01H
    POP 02H
RET

```

;*****
*

QUET8HANG:

```

PUSH 07H
PUSH 00H
MOV R7,#7 ;Chọn hàng để quét
LAPHANG: MOV R0,#9EH ; Nạp lại giá trị con trỏ

```

CLR RCK ; đưa chân chót 74hc595 về 0

LAPCOT:CLR SCK ; tạo xung sườn âm xung clock 74hc595

```

MOV A,@R0      ; đưa giá trị đầu vào thanh ghi A
CLR C          ; xóa cờ C
RRC A          ; Quay phải A qua cờ C
MOV @R0,A ;Đưa giá trị tiếp theo vào đầu ram
SETB SCK ; tạo sườn dương xung clock 74hc595 để

```

đưa giá trị vào

MOV SI,C ; giá trị được đưa vào 74hc595

INC R0 ; tăng con trỏ lên 1

*CJNE R0,#0EFH,LAPCOT ; kiểm tra xem đã quét
đủ chưa*

SETB RCK; xuất dữ liệu ra các chân 74hc595

MOV A0H,R7; Chọn hàng đầu tiên để quét

LCALL DELAYLED ; gọi hàm tạo trễ

MOV A0H,#8 ; đưa 3 bit đầu Port2 về 0

DEC R7 ; R7=R7-1

*CJNE R7,#-1,LAPHANG ; kiểm tra xem quét đủ
hàng chưa*

POP 00H

POP 07H

RET

DELAYLED: PUSH 04H

PUSH 03H

MOV R4,#10

D1 :MOV R3,#250

DJNZ R3,\$

DJNZ R4,D1

POP 03H

POP 04H

RET

CNTT:

DB

*3H,3H,0FFH,0FFH,3H,3H,3H,0H,0FFH,0FFH,11H,31H,
71H,0DFH,8EH,0H,7FH,0FFH,0C0H,0C0H,0C0H,0FFH
,7FH,0H,*

3CH,7EH,0C3H,0C3H,0C3H,7EH,3CH,0H,0FFH,0FFH,

```
0CH,18H,30H,0FFH,0FFH,0H,7EH,0FFH,0C3H,0C3H,0  
D3H,0F3H,72H,0H,  
  
0H,0H,0H,0H,0H,0H,0H,18H,0FFH,0FFH,99H,99H,81H,  
0FFH,7EH,0H,0FFH,0FFH,18H,18H,18H,0FFH,0FFH,0  
H,  
  
0H,0H,0H,0H,0H,0H,0H,7EH,0FFH,0C3H,0C3H,0C3  
H,0C3H,0C3H,0H,0FFH,0FFH,0CH,18H,30H,0FFH,0FF  
H,0H,  
  
3H,3H,0FFH,0FFH,3H,3H,3H,0H,3H,3H,0FFH,0FFH,3  
H,3H,3H,0H  
END
```

8.2 Ứng dụng điều khiển giao thông ngã 4

8.2.1 Phát biểu ứng dụng

Ứng dụng thực tế thứ hai trong chương này là điều khiển đèn giao thông, một thiết bị không thể thiếu trong giao thông hàng ngày. Việc tạo ra 1 hệ thống điều khiển tự động đèn giao thông đơn giản, hiệu quả là nhu cầu tất yếu.

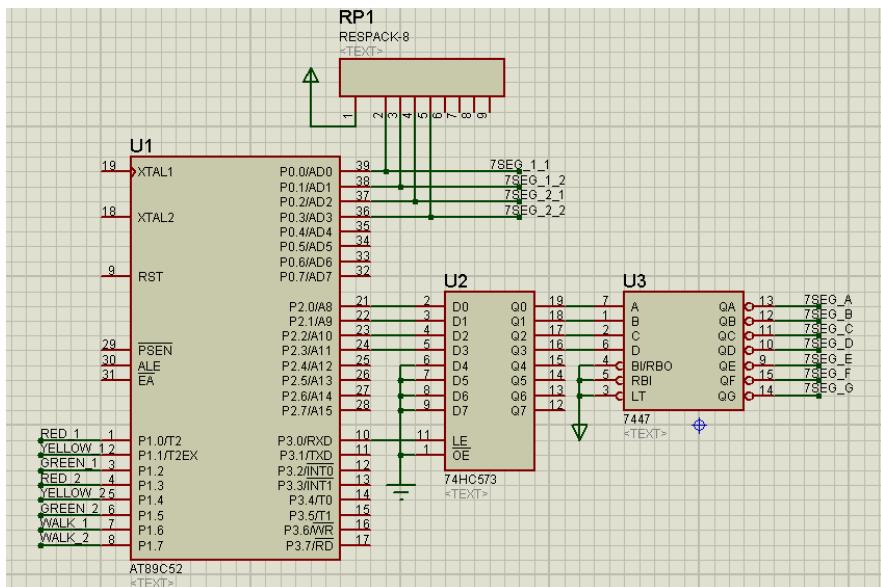
Trong ứng dụng này ta sẽ thiết kế một hệ thống điều khiển đèn giao thông đơn giản bao gồm đèn xanh, vàng, đỏ, đèn đi bộ và hộp số đếm lùi. Do đây là 1 hệ thống cơ bản nên thời gian sáng của các đèn sẽ được cài đặt sẵn.

8.2.2 Thiết kế mạch mô phỏng

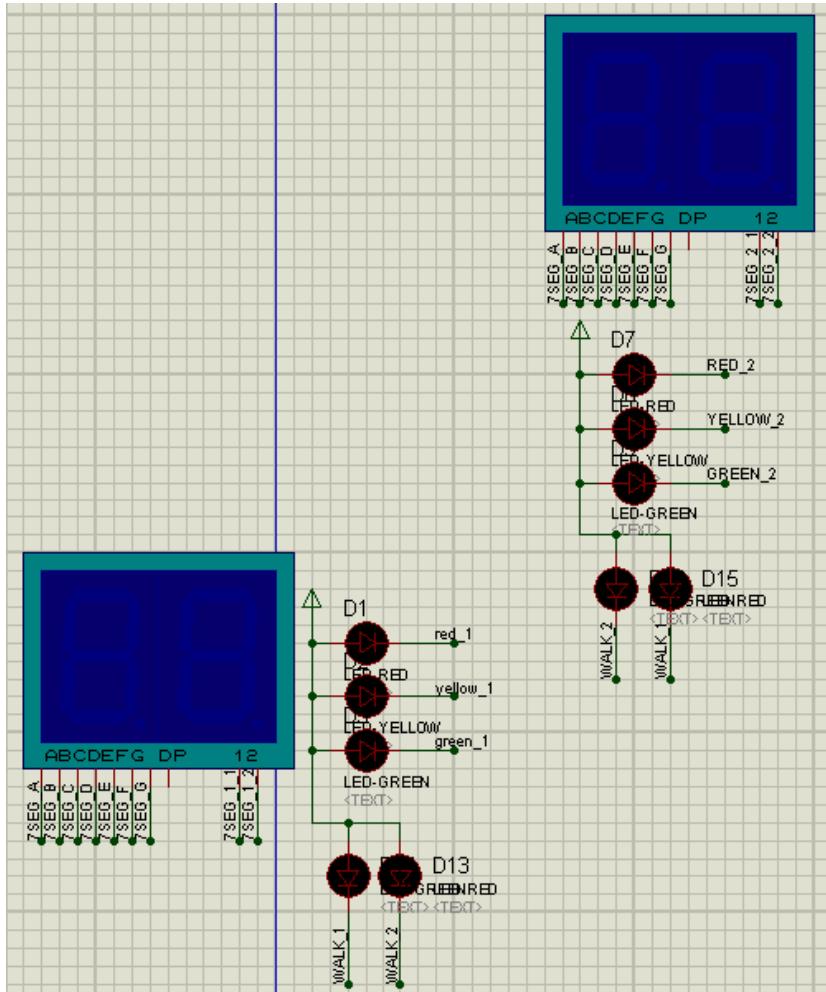
Trong mạch mô phỏng, ta sử dụng phần cứng quét LED 7 đoạn như ở phần 7.2.2 đã trình bày, bao gồm IC chốt và IC giải

mã LED 7 đoạn, tuy nhiên ta sử dụng 2 cặp LED 7 đoạn, nguyên lý quét giống như đã trình bày.

Port 1 sẽ dùng để điều khiển các pha đèn, mỗi pha gồm có đèn xanh, đỏ, vàng và đèn đi bộ xanh đỏ. Sơ đồ thiết kế sử dụng các IC cơ bản trên Proteus của một giải pháp để xuất có thể như hình 8-6 và sơ đồ mô phỏng hệ thống đèn cùng led hiển thị thời gian chuyển đổi như Hình 8-8.



Hình 8-8 Sơ đồ thiết kế giải pháp điều khiển đèn giao thông trên proteus

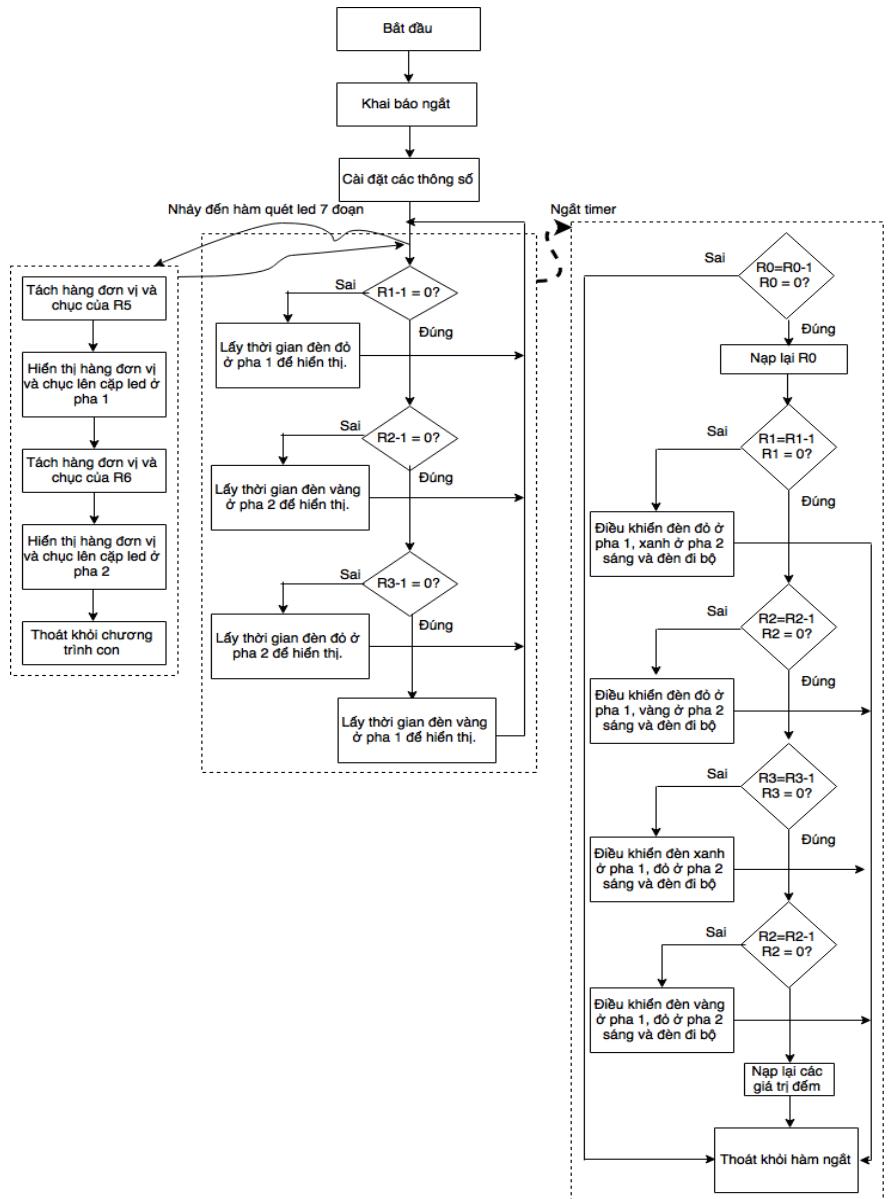


Hình 8-9 Sơ đồ mô phỏng hiển thị các đèn giao thông cùng thời
gian chuyển đổi giữa các đèn

8.2.3 Chương trình nhúng

Chương trình nhúng sử dụng giải thuật quét LED 7 đoạn như đã trình bày ở trên. Trong đó thanh ghi R5 sẽ chứa giá trị hiển thị trên cặp LED 7 đoạn ở pha 1 (7SEG_1), R6 sẽ chứa giá trị hiển thị trên cặp LED 7 đoạn ở pha 2 (7SEG_2). Tuy nhiên giá trị hiển thị là số thập phân nên chương trình cũng đã được đưa vào giải thuật chuyển số HEX sang hệ thập phân và tách số thành chục và đơn vị để hiển thị lên đèn.

Chương trình sử dụng Ngắt timer tạo thời gian 1 giây để đếm thời gian và điều khiển đèn tín hiệu trên các pha đèn. Chương trình chính sẽ dựa vào tín hiệu và thời gian trên pha đèn để điều khiển giá trị hiển thị trên LED 7 đoạn cho phù hợp (tác động lên R5 và R6). Mỗi lần ngắt giá trị đếm sẽ thay đổi cùng với tín hiệu đèn, từ đó ta có thể hiển thị đồng bộ thời gian đang được đếm và tín hiệu của đèn. Do thời gian khi chạy chương trình là thời gian đếm ngược và phân biệt xanh vàng đỏ nên thời gian đèn đỏ khi hiển thị trên đèn tín hiệu và đèn LED 7 đoạn là thời gian đèn đỏ + thời gian đèn vàng. Tổng thời gian đèn trên 2 pha là như nhau.



Hình 8-10 Lưu đồ giải thuật điều khiển đèn giao thông

ORG 00h
LJMP MAIN

ORG 00bh
LJMP TIMER0

ORG 0030h

MAIN:

MOV R0, #10H
MOV R1, #14H // Red
MOV R2, #04H // yellow1
MOV R3, #0FH // Green
MOV R4, #04H // yellow2
MOV R5, #14H
MOV R6, #14H
MOV P0, #00H
MOV P1, #0FFH
MOV P2, #00H
MOV P3, #00H
MOV TMOD, #11H
MOV IE, #82H

MOV TL0, #0AFH
MOV TH0, #03CH

SETB TF0

LOOP:

LCALL SCAN_SEG

MOVA, RI // Dieu khien thoi gian hien thi

DECA

JZ SEG_Y2 // nhay den den vang 2

ADD A,#02H // den do 1

MOV R5, A

MOVA, RI

DECA

DECA

MOV R6, A

SJMP LOOP

SEG_Y2:

MOVA, R2

DECA

JZ SEG_R2 // nhay den den do 2

MOVA, R2 // den vang 2

DECA

DECA

MOV R5, A

MOVA, R2

DECA

DECA

MOV R6, A

SJMP LOOP

SEG_R2:

MOVA, R3

DECA

JZ SEG_Y1 // nhay den den vang 1

ADD A, #03H //den do 2

MOV R6, A

```
MOVA, R3
DECA
DECA
MOV R5, A
SJMP LOOP
```

```
SEG_Y1:
MOVA, R4      // den vang 1
DECA
MOV R5, A
MOVA, R4
DECA
MOV R6, A
SJMP LOOP
```

```
TIMER0:
CLR TF0
CLR TR0
DJNZ R0, EXIT_0
MOV R0, #10H
```

```
DJNZ R1, R_1_G_2
INC R1
DJNZ R2, Y_2
INC R2
DJNZ R3, R_2_G_1
INC R3
DJNZ R4, Y_1
INC R4
MOV R1, #14H // Red
MOV R2, #04H // yellow1
MOV R3, #0FH // Green
MOV R4, #04H // yellow2
```

SETB TR0

RETI

R_1_G_2: //Den do pha 1 / xanh pha 2 sang

MOV P1, #10011110B

SETB TR0

RETI

Y_2: //Den vang pha 2 sang

MOV P1, #10101110B

SETB TR0

RETI

R_2_G_1: //Den do pha 2 / xanh pha 1 sang

MOV P1, #01110011B

SETB TR0

RETI

Y_1: //Den vang pha 1 sang

MOV P1, #01110101B

SETB TR0

RETI

EXIT_0:

SETB TR0

RETI

SCAN SEG:

MOV P0, #00000001B

MOV B, #0AH

MOVA, R5

DIV AB

MOV P2, A

```
SETB P3.0
LCALL DELAY
CLR P3.0
MOV P0, #00000010B
MOV P2,B
SETB P3.0
LCALL DELAY
CLR P3.0
MOV P0, #000000100B
MOV B,#0AH
MOVA, R6
DIV AB
MOV P2, A
SETB P3.0
LCALL DELAY
CLR P3.0
MOV P0, #00001000B
MOV P2,B
SETB P3.0
LCALL DELAY
CLR P3.0
RET
```

DELAY:

LOOP2:

```
MOV TL1, #67H
MOV TH1, #0C5H
```

SETB TR1

LOOP3:

```
JNB TF1, LOOP3
```

CLR TR1

CLR TF1

RET

END

8.3 Ứng dụng điều khiển xe robot

8.3.1 Phát biểu ứng dụng

Ứng dụng thứ ba trong chương nhắm đến bài toán điều khiển chuyên động và giáo trình sử dụng bài toán điều khiển robot chạy tới chạy lui, qua trái, qua phải sau 1 khoảng thời gian cố định làm ví dụ.

Về phần cứng để điều khiển 2 động cơ DC ta có thể sử dụng IC cầu H L298 nhằm cung cấp nguồn cho động cơ và phục vụ điều khiển động cơ bằng giải thuật băm xung.

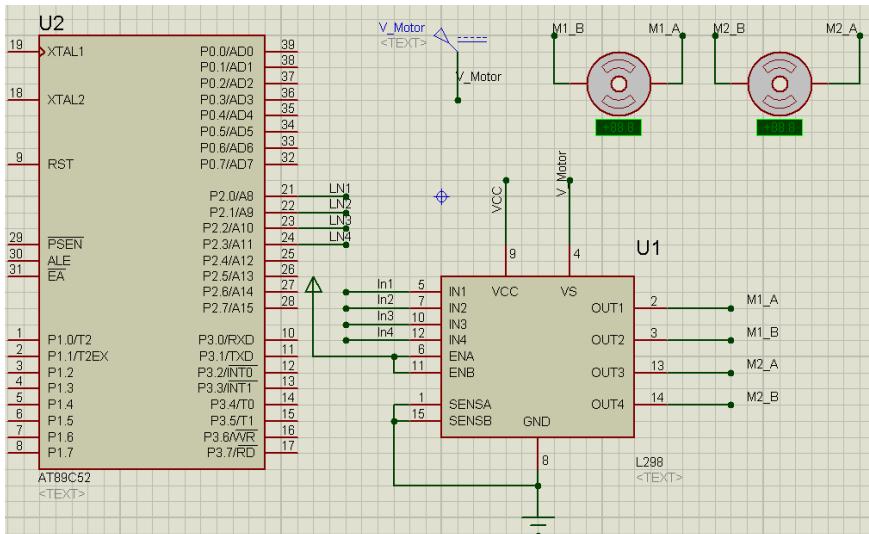
Để thực hiện ứng dụng ta sẽ sử dụng 2 timer của 8051 phục vụ cho 2 mục đích: băm xung điều khiển tốc độ động cơ, định thời để điều khiển robot.

Việc điều khiển robot theo cách đơn giản nhất ta sẽ điều khiển tốc độ 2 động cơ. Hai động cơ quay cùng vận tốc robot sẽ đi thẳng, hoặc lui tùy theo chiều quay động cơ, hai động cơ quay khác tốc độ robot sẽ xoay theo hướng động cơ có tốc độ thấp hơn.

8.3.2 Thiết kế mạch mô phỏng

Mạch mô phỏng Hình 8-11 sử dụng 2 động cơ DC kết nối với L298, DC 1 (bên trái) sẽ được cấp nguồn từ OUT1 và OUT2 từ L298, DC2 (bên phải) sẽ được cấp nguồn từ OUT3 và OUT4 từ L298. Vậy nếu muốn điều khiển động cơ 8051 sẽ điều khiển L298 thông qua 4 chân IN1 IN2 IN3 IN4 được nối với P2.0 P2.1 P2.2 P2.3.

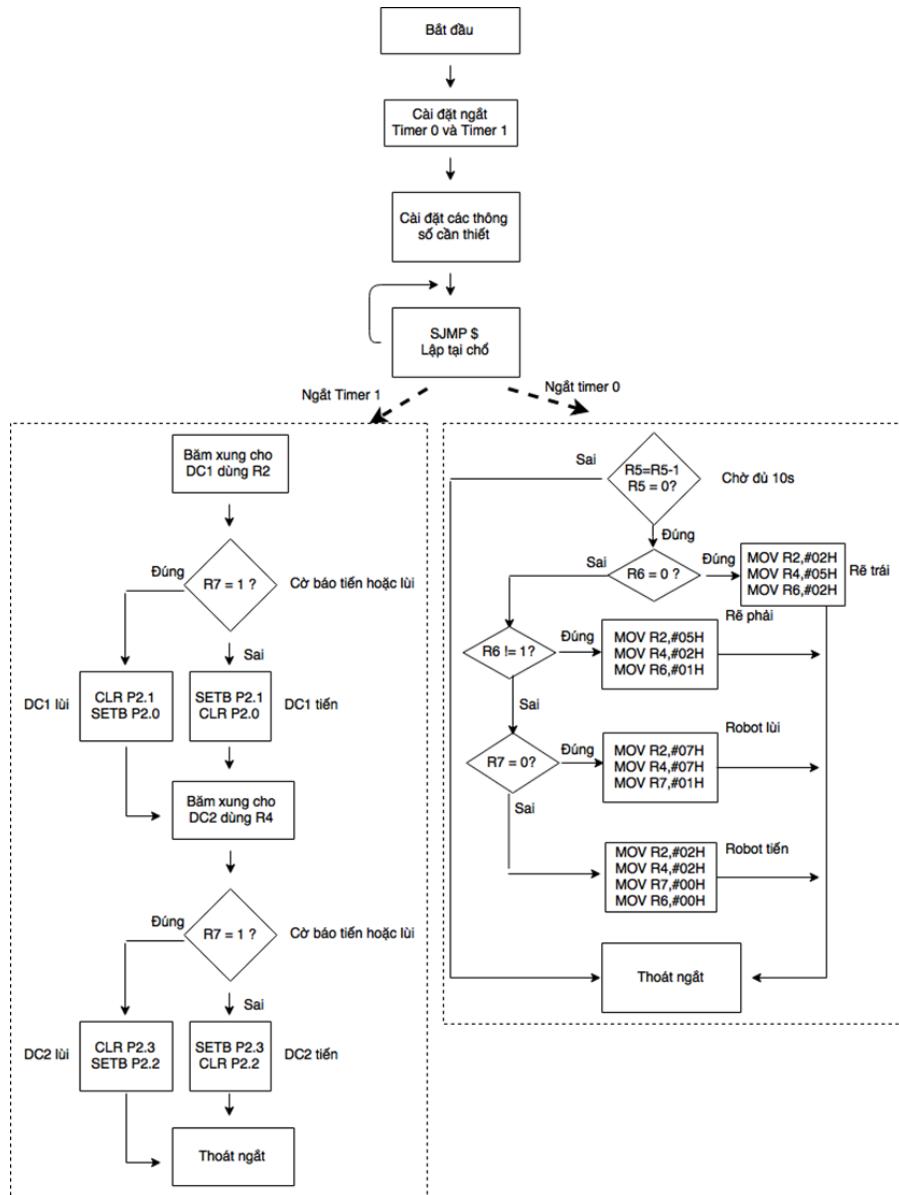
Để xe robot tiến ta điều khiển sao cho 2 động cơ quay cùng tốc độ, để robot lùi ta sẽ đảo chân điều khiển input cho L298. Nếu muốn robot rẽ trái ta sẽ điều khiển sao cho tốc độ động cơ bên phải lớn hơn tốc độ động cơ bên trái như vậy robot sẽ quay trái, góc quay và tốc độ quay phụ thuộc vào độ chênh lệch tốc độ giữa 2 động cơ. Tương tự như vậy robot sẽ quay phải khi tốc độ động cơ bên trái lớn hơn tốc độ động cơ bên phải.



Hình 8-11 Sơ đồ mạch mô phỏng điều khiển 2 động cơ

8.3.3 Chương trình nhúng

Chương trình được thiết kế để thay đổi đường chạy robot mỗi 10s 1 lần theo giải thuật được mô tả trong Hình 8-12.



Hình 8-12 Lưu đồ giải thuật điều khiển robot

Chương trình mẫu mô tả chi tiết giải thuật điều khiển được trình bày dưới đây.

ORG 00h
LJMP MAIN

ORG 00bh //Ngắt Timer 0

LJMP TIMER0

ORG 01bh //Ngắt Timer 1 để bấm xung
LJMP TIMER1

ORG 0030h

MAIN:

MOV R0, #0BH

MOV R2, #02H //Xung DC 1

MOV R4, #02H //Xung DC 2

MOV R3, #01H

MOV R1, #01H

MOV R5, #0C8H

MOV R6, #00H //thanh ghi báo rẽ trái / phải

MOV R7, #00H //thanh ghi báo tiến / lùi

CLR P2.0

CLR P2.1

CLR P2.2

CLR P2.3

MOV TMOD,#11H

MOV IE, #8AH

MOV TL1, #0D0H

MOV TH1, #07H

```
MOV TL0, #0AFH
MOV TH0, #03CH
SETB TR0
SETB TF1
SJMP $
```

TIMER0: // Trình phục vụ ngắt timer 0

```
CLR TF0
CLR TR0
MOV TL0, #0AFH
MOV TH0, #03CH
DJNZ R5, CTR // Lặp đủ 10s
MOV R5, #0C8H
```

```
MOVA,R6
JZ LEFT
CJNE R6,#01H, RIGHT
MOVA, R7
JZ RETROGRADE
```

```
MOV R6,#00H //robot tiến
MOV R7,#00H
MOV R2, #02H
MOV R4, #02H
SETB TR0
RETI
```

LEFT: // Robot rẽ trái

```
MOV R6,#02H
MOV R2, #02H
MOV R4, #05H
SETB TR0
RETI
```

RIGHT: //Robot rẽ phải

MOV R6,#01H

MOV R2, #05H

MOV R4, #02H

SETB TR0

RETI

RETROGRADE: //robot lùi

MOV R2, #07H

MOV R4, #07H

MOV R7,#01H

SETB TR0

RETI

CTR:

SETB TR0

RETI

TIMER1:

CLR TF1

CLR TR1

MOV TL1, #0D0H

MOV TH1, #07H

DJNZ R0, PWM_DC_1

MOV R0, #0BH

MOVA,R2

MOV R1,A

MOVA,R4

MOV R3,A

SETB TR1

RETI

PWM_DC_1: // Bấm xung cho DC 1

DJNZ R1, PWM2

*INC R1
CLR P2.1
CLR P2.0
SJMP PWM_DC_2*

PWM2:

*MOV A, R7 //Điều khiển robot tiến hoặc lùi
JZ GO_1
CLR P2.1
SETB P2.0
SJMP PWM_DC_2
GO_1:
SETB P2.1
CLR P2.0
SJMP PWM_DC_2*

*PWM_DC_2: //Bấm xung cho DC 2
DJNZ R3, PWM3
INC R3
CLR P2.2
CLR P2.3
SETB TR1
RETI*

PWM3:

*MOV A, R7 //Điều khiển robot tiến hoặc lùi
JZ GO_2
CLR P2.3
SETB P2.2
SETB TR1
RETI
GO_2:
SETB P2.3
CLR P2.2*

SETB TRI

RETI

END

8.4 Câu hỏi và bài tập chương

1. Các IC dịch và giải mã nào là phổ biến? Hoạt động của các IC này như thế nào?
2. Để băm xung có thể dùng IC nào?
3. Cách viết một chương trình nhúng và thử nghiệm nhúng lên board chạy thật.
4. Hãy mô tả ứng dụng mạch đo nhiệt độ, sau đó vẽ sơ đồ nguyên lý và viết chương trình mô phỏng mạch đo nhiệt độ sử dụng vi điều khiển 8051.
5. Đồng hồ điện tử báo thức là một ứng dụng được sử dụng nhiều trong cuộc sống. Bằng những kiến thức đã học về vi điều khiển 8051 và những ứng dụng đã được mô tả trong chương 8, hãy mô tả và viết chương trình mô phỏng một mạch đồng hồ báo thức. Lưu ý cần sử dụng thêm IC thời gian thực DS1307 để lấy các giá trị thời gian chính xác, và có thể lưu lại thời gian khi vi điều khiển mất nguồn.
6. Viết chương trình mô phỏng mạch đo tốc độ quay của động cơ sử dụng encoder và bộ định thời 0 của vi điều khiển 8051.

Chương 9. Các họ vi điều khiển hiện đại

Mục đích chương

Chương này sẽ trình bày về một số vi điều khiển hiện đại được sử dụng rộng rãi ở Việt Nam hiện nay như AVR, PIC, ARM. Học xong chương này người học sẽ nắm được những tính năng và đặc điểm của một vài vi điều khiển hiện đại và có được kỹ năng ứng dụng các vi điều khiển thông dụng hiện nay trong giải quyết các vấn đề thực tế.

9.1 Họ vi điều khiển AVR

9.1.1 Tổng quan về AVR

Vi điều khiển AVR do hãng Atmel sản xuất được giới thiệu lần đầu năm 1996. AVR có rất nhiều dòng khác nhau bao gồm dòng Tiny AVR (như ATtiny13, ATtiny22,...) có kích thước bộ nhớ nhỏ, ít bộ phận ngoại vi. Ké đến là dòng AVR (chẳng hạn AT90S8535, AT90S8515,...) có kích thước bộ nhớ vào loại trung bình và mạnh hơn là dòng Mega (như ATmega32, ATmega128,...) với bộ nhớ có kích thước vài Kbyte đến vài trăm Kb cùng với các bộ ngoại vi đa dạng được tích hợp trên chip. Một vài dòng tích hợp cả bộ LCD trên chip (dòng LCD AVR). Tốc độ của dòng Mega cũng cao hơn so với các dòng khác. Sự khác nhau cơ bản giữa các dòng chính là cấu trúc ngoại vi, còn nhân thì vẫn như nhau.

Năm 2008, Atmel lại tiếp tục cho ra đời dòng AVR mới là megaAVR, với những tính năng mạnh mẽ chưa từng có ở các

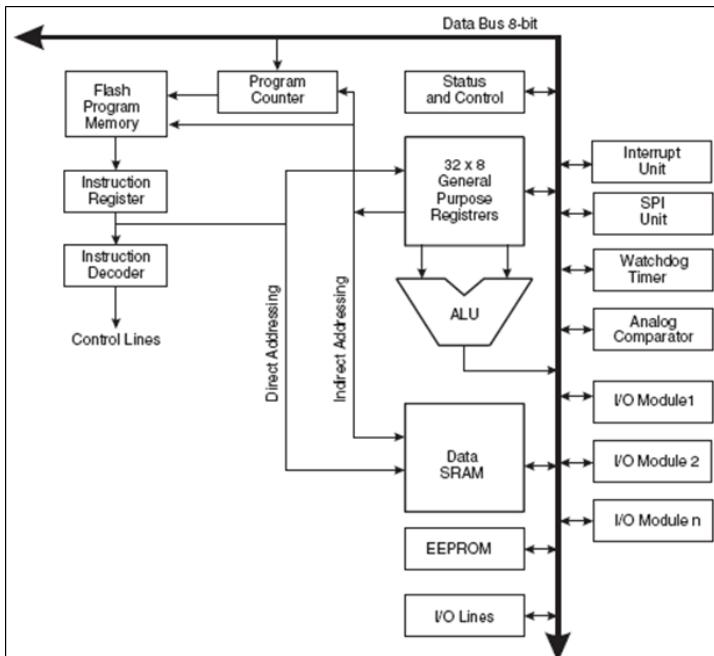
dòng AVR trước đó. Có thể nói XMEGA AVR là dòng MCU 8 bit mạnh mẽ nhất hiện nay.

- TinyAVR - chuỗi Attiny
 - + 0,5-8 kB bộ nhớ chương trình
 - + Đóng vỏ 6-32-chân.
 - + Ngoại vi hữu hạn thường ít.
 - MegaAVR - chuỗi Atmega
 - + 4-256 kB bộ nhớ chương trình
 - + Đóng vỏ 28-100-chân
 - + Tập lệnh mở rộng (Lệnh nhân và lệnh cho quản lý bộ nhớ lớn hơn).
 - + Mở rộng hơn về thiết bị ngoại vi
 - XMEGA - chuỗi ATXMEGA
 - + 16-384 kB bộ nhớ chương trình.
 - + Đóng vỏ 44-64-100-chân (A4, A3, A1)
 - + Mở rộng các tính năng hiệu suất, chẳng hạn như DMA, "Sự kiện hệ thống", và hỗ trợ mật mã.
 - + Thiết bị ngoại vi được mở rộng với DACs

9.1.2 Đặc trưng phần cứng AVR

9.1.2.1 CPU

Đơn vị xử lý trung tâm(CPU) của AVR được cấu tạo theo kiến trúc RISC một cấu trúc đặc biệt cho ngôn ngữ C để giảm thiểu sự chênh lệch kích thước mã lệnh và tốc độ thực thi mã lệnh. Sơ đồ cấu trúc cơ bản của AVR được trình bày trong Hình 9-1.



Hình 9-1 Cấu trúc nhân CPU của AVR

Hỗn vi điều khiển AVR với việc làm giảm kích thước đoạn mã khi biên dịch và thêm vào đó là thực hiện hầu hết các lệnh lệnh đúng trong 1 chu kỳ máy (trừ các lệnh truy xuất bộ nhớ).

9.1.2.2 Bộ nhớ

Bộ nhớ vi điều khiển AVR có cấu trúc Harvard là cấu trúc có đường Bus riêng cho bộ nhớ chương trình và bộ nhớ dữ liệu. Bộ nhớ AVR được chia làm 2 phần chính: Bộ nhớ chương trình (program memory) và bộ nhớ dữ liệu (Data memory).

- Bộ Nhớ Chương Trình : Bộ nhớ chương trình của AVR là bộ nhớ Flash có dung lượng từ 8 - 128 K bytes. Bộ nhớ chương trình có độ rộng bus là 16 bit.

Những địa chỉ đầu tiên của bộ nhớ chương trình được dùng cho bảng vec tơ ngắn. Cần để ý là ở vi điều khiển ATmega 32 hay 128 và một số khác, bộ nhớ chương trình còn có thể được chia làm 2 phần : phần boot loader (Boot loader program section) và phần ứng dụng (Application program section). Phần boot loader chứa chương trình boot loader. Chương trình Boot loader là một phần mềm nhỏ nạp trong vi điều khiển và được chạy lúc khởi động. Phần mềm này có thể tải vào trong vi điều khiển chương trình của người sử dụng và sau đó thực thi chương trình này. Mỗi khi reset vi điều khiển CPU sẽ nhảy tới thực thi chương trình boot loader trước, chương trình boot loader sẽ dò xem có chương trình nào cần nạp vào vi điều khiển hay không, nếu có chương trình cần nạp, boot loader sẽ nạp chương trình vào vùng nhớ ứng dụng (Application program section), rồi thực thi chương trình này. Ngược lại, boot loader sẽ chuyển tới chương trình ứng dụng có sẵn trong vùng nhớ ứng dụng để thực thi chương trình này. Phần ứng dụng (Application program section) là vùng nhớ chứa chương trình ứng dụng của người dùng. Kích thước của phần boot loader và phần ứng dụng có thể tùy chọn.

- Bộ nhớ dữ liệu : Bộ nhớ dữ liệu của AVR chia làm 2 phần chính là bộ nhớ SRAM và bộ nhớ EEPROM. Tuy cùng là bộ nhớ dữ liệu nhưng hai bộ nhớ này lại tách biệt nhau và được đánh địa chỉ riêng. Bộ nhớ SRAM có dung lượng từ 512Bytes đến 4 K bytes. Bộ nhớ EEPROM là bộ nhớ dữ liệu có thể ghi xóa ngay trong lúc vi điều khiển đang hoạt động và không bị mất dữ liệu khi nguồn điện cung cấp bị cắt. Có thể

ví bộ nhớ dữ liệu EEPROM giống như là ổ cứng (Hard disk) của máy vi tính. EEPROM được xem như là một bộ nhớ vào ra được đánh địa chỉ độc lập với SRAM, điều này có nghĩa là ta cần sử dụng các lệnh in, out ... khi muốn truy xuất tới EEPROM. Bộ nhớ EEPROM được tích hợp sẵn trên họ vi điều khiển AVR giúp dễ dàng hơn trong việc lưu trữ các dữ liệu xử lý trung gian hay các kết quả tính toán ngay cả khi mất điện. Đây là một điểm mới so với họ vi điều khiển 8051. Hơn nữa dung lượng bộ nhớ SRAM trên họ AVR cũng thường lớn hơn giúp dễ dàng thực thi các ứng dụng. Với việc phát triển hệ thống trên AVR thì ta có các lợi thế về phần cứng tích hợp sẵn, phải kể đến trong số đó là ADC, giao tiếp SPI, I2C, 1 wire... mà ở các vi điều khiển thế hệ cũ chưa có. Vì điều khiển AVR với kiến trúc RISC (Reduced Instruction Set Computer) cho phép đẩy cao tốc độ xử lý lệnh của modul đáp ứng các yêu cầu thời gian thực. Bộ truyền nhận USART trên AVR cho phép chúng truyền nhận nối tiếp không đồng bộ hay bộ ghép nối đồng bộ SPI giúp chúng ghép nối với các ngoại vi hay mạng các vi điều khiển dễ dàng hơn.

9.1.3 Một số giao tiếp ngoại vi đặc trưng của AVR

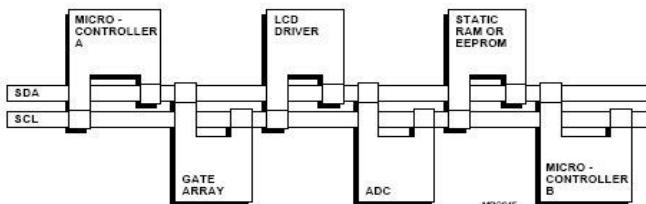
Các vi điều khiển AVR cũng được trang bị sẵn các ngoại vi thông dụng như cổng vào ra nối tiếp, song song, ngắt ngoài,... Một số vi điều khiển được trang bị sẵn các cổng ghép nối với chuẩn giao tiếp USB như AT90USB, hay bộ chuyển đổi dữ liệu tương tự sang số ADC, bộ chia xung PWM, giao tiếp nối tiếp đồng bộ SPI, bus nối tiếp I2C, hay giao tiếp chỉ với 1 dây one wire.

a. BUS I²C

I²C là viết tắt của từ tiếng anh *Inter-Integrated Circuit* đây là một loại BUS nối tiếp cho phép nhiều thiết bị giao tiếp với nhau chỉ trên 2 dây ký hiệu là SDA và SCL cho phép các thiết bị đấu nối với BUS truyền và nhận thông tin. Ví dụ sử dụng I²C cho 2 CĐK kết nối như Hình 9-2.

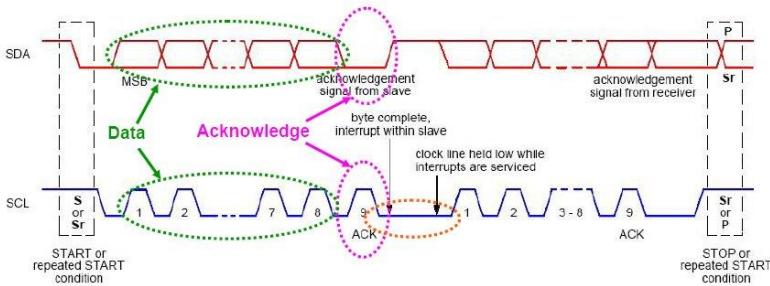
Trong vi điều khiển AVR BUS I²C

- Có thể cấu hình theo 2 chế độ chủ, tớ và chế độ đa chủ.
- Một thiết bị nối với BUS I²C có thể là thiết bị truyền hoặc thiết bị nhận.
- AVR dùng 7 bit để đánh địa chỉ cho các thiết bị trên BUS do vậy có thể kết nối tới 128 thiết bị trên cùng một BUS.
- Tốc độ truyền dữ liệu lên tới 400KHz



Hình 9-2 Ví dụ sử dụng I²C có 2 vi điều khiển kết nối

Trên SDA (Serial Data), số lượng byte có thể truyền là không giới hạn và mỗi byte được truyền cần phải kèm theo 1 bit ACK (Acknowledge). Nếu bộ tớ (slave) nào đó không truyền hoặc nhận dữ liệu, nó có thể giữ SCL (Serial Clock) ở mức thấp để ép bộ tớ vào trạng thái chờ.



Hình 9-3 Truyền dữ liệu trên BUS I2C

Việc truyền dữ liệu trên I2C như Hình 9-3. Truyền dữ liệu với 1 bit ACK (bit kiểm tra) là bắt buộc. ACK có liên quan tới xung nhịp phát ra bởi bộ chủ. Bộ phát sẽ để SDA ở mức cao trong suốt chu kỳ xung nhịp ACK.

Thông thường một bộ thu đã được định địa chỉ thì phải khởi phát một bit ACK sau mỗi byte nó nhận được.

Nếu bộ thu gặp rắc rối với việc nhận dữ liệu nó phát tín hiệu kết thúc dữ liệu tới bộ phát từ bằng cách không khởi tạo ACK trên byte cuối cùng. Bộ phát từ giải thoát SDA và cho phép bộ chủ kết thúc cuộc truyền hay khởi tạo phiên truyền mới.

b. Giao tiếp SPI

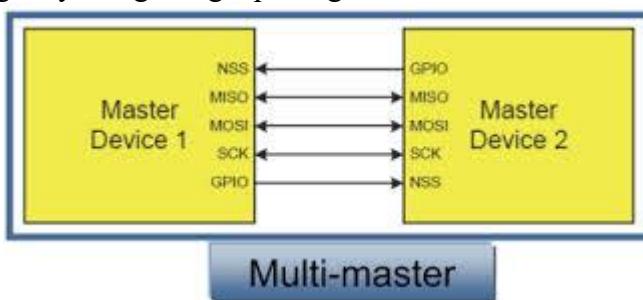
SPI là viết tắt của *Serial Peripheral Interface*, SPI là một phương thức truyền thông nối tiếp đồng bộ giữa các vi điều khiển và ngoại vi. Thông thường một giao tiếp trong cấu hình chuẩn giao tiếp SPI gồm 2 đường điều khiển và 2 đường dữ liệu.

Các vi điều khiển AVR hầu hết đều được trang bị bộ truyền thông nối tiếp đồng bộ SPI với các tính năng cơ bản như sau:

- Truyền song công với 3 dây dẫn MOSI, MISO, SCK đồng bộ quá trình truyền.
- Ghép nối theo dạng chủ tớ.

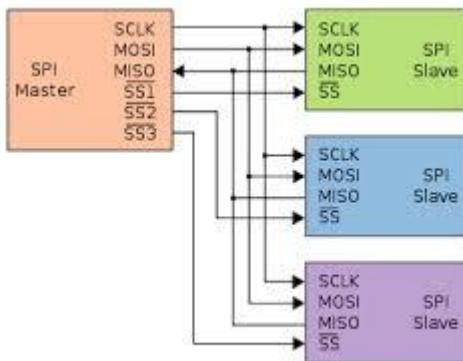
- Bit LBS và MBS có thể truyền tùy thuộc vào người lập trình.
- Có thể lập trình tốc độ truyền thông báo kết thúc truyền bằng ngắt, vận hành từ chế độ ngủ.
- Cấu hình là Master hay Slave bằng cách thiết lập cho chân SS.

Trong cấu hình Multi Master (Hình 9-4) thì bất kỳ thiết bị nào cũng có thể cấp tín hiệu xung và tín hiệu chọn chip. Chế độ này thường thấy trong các ghép nối giữa các vi điều khiển.



Hình 9-4 Cấu hình SPI ché độ Multi Master

Trong cấu hình Master/Slave (Hình 9-5), một master sẽ cung cấp tín hiệu xung clock và tín hiệu chọn chip- Chip Select (CS) để chọn slave mà nó cần giao tiếp. Các slave sẽ nhận tín hiệu clock và chip select từ Master. SS là tín hiệu cho phép và SCK là tín hiệu xung nhịp đồng bộ.



Hình 9-5 Cấu hình SPI chế độ Master/ Slaver

9.1.4 Tập lệnh của AVR

Tập lệnh được chia thành các nhóm sau:

- Các lệnh toán học và logic: Một số lệnh đáng chú ý trong nhóm này là cộng (ADD), trừ (SUB), nhân (MUL), chia (FMUL), xóa (CLR), tăng (INC), giảm (DEC).
- Lệnh rẽ nhánh: Gồm có các lệnh nhảy, lệnh gọi chương trình con, lệnh trở về, lệnh so sánh, lệnh rẽ nhánh.
- Lệnh di chuyển dữ liệu: Gồm có các lệnh sao chép dữ liệu (MOV), xuất nhập dữ liệu, PUSH, POP.
- Các lệnh thao tác BIT: SBI, CBI, LSL, ...
- Các lệnh điều khiển MCU: NOP, SLEEP, WDR

Tuy nhiên hiện nay việc lập trình cho AVR chủ yếu sử dụng ngôn ngữ C, sử dụng trình biên dịch Codevision AVR. Với việc sử dụng trình biên dịch này các thao tác về cấu hình vi điều

khiển có thể được thực hiện hoàn toàn bằng giao diện đồ họa. sau đó trình biên dịch tự sinh mã, người lập trình chỉ cần quan tâm tới phần thân chương trình.

9.1.5 Giới thiệu vi điều khiển AVR ATMEGA-8515.

- **AVR®** – Kiến trúc RISC có tính năng cao và dùng ít năng lượng. Tập lệnh với 130 lệnh hâu hết thực hiện trong 1 chu kỳ.
- Có 32 thanh ghi 8 bit.
- Tốc độ 16MIPS (16 Milion Instructions per second) triệu lệnh/s ở tần số dao động 16MHz
- Bộ nhớ lập trình 8KB. Bộ nhớ dữ liệu và chương trình Flash-loại Nonvolatile (không mất dữ liệu khi mất nguồn). Chu kì sử dụng Flash: 1,000 lần nạp/xóa chương trình.
- 8 KBytes ISP Flash, Giao tiếp ngoại vi nối tiếp SPI.
- 512 Bytes EEPROM Chu kì sử dụng: 100,000 lần nạp/xóa EEPROM.
- 512 Bytes Internal SRAM
- Có bit bảo mật chương trình.

Các tính năng ngoại vi:

- ADC10bits, 8 kênh.
- UART có thể lập trình
- Master/Slave SPI Serial Interface
- 2 Timer/Counters 8 bit với chế độ so sánh và Prescaler độc lập.
- 1 Timer/Counters 16 bit với chế độ so sánh và độc lập và PWM 8, 9, 10bit.
- Watchdog Timer với dao động trong chip có thể lập trình.

- Bộ so sánh Analog trên chip.

Các tính năng nổi bật khác:

- Power-on Reset Circuit
- Real-time Clock (RTC).
- Nhiều nguồn ngắt với 3 ngắt ngoài và nhiều ngắt trong.
- 3 chế độ nghỉ: Idle, Power Save và Power-down

Sử dụng năng lượng ở 4 MHz, 3V, 20°C

- Hoạt động tích cực: 6.4 mA
- Chế độ Idle: 1.9 mA
- Chế độ Power-down: <1 μA

a. Các tính năng cơ bản.

ATMEGA8515 được thiết kế với nhiều tính năng mạnh mẽ. Với 35 chân I/O, 32 thanh ghi 8 bit lưu trữ thông tin trong quá trình xử lý, tần số xung nhịp cho phép tới 16MHz, 130 lệnh hầu hết thực hiện trong 1 chu kỳ xung nhịp. AVR8515 có sẵn 8Kbyte bộ nhớ lập trình cho phép tới 10.000 lần ghi xóa lưu trữ được các chương trình lập trình lớn. Trong chip cũng chứa sẵn 512 byte EEPROM và 512 byte SRAM ngoài ra có thể mở rộng thêm 64Kbyte bộ nhớ SRAM bên ngoài, hỗ trợ sẵn 3 ngắt ngoài, 6 ngắt cho 2 bộ Timer, 3 ngắt cho bộ truyền nhận nối tiếp song song USART và một số ngắt khác như Reset, SPI, STC...v..v...

Một số tính năng mà AT8515 hỗ trợ:

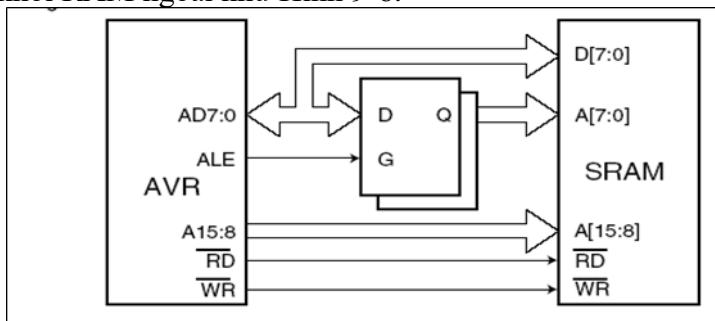
- Bộ truyền nhận nối tiếp song song lập trình được. Đặc biệt khi dùng với chương trình viết mã CodeVision AVR thì ta có thể đặt được tần số, biết trước được sẽ có khoảng bao nhiêu phần trăm thông tin truyền đi bị lỗi.

- Hỗ trợ bộ truyền nhận nối tiếp SPI - Serial Peripheral Interface, cho phép truyền song công toàn phần.
- Có sẵn bộ so sánh Analog, bộ so sánh này sẽ so sánh điện áp giữa 2 chân của bộ so sánh và tạo ra 1 ngắt của bộ so sánh khi phát hiện chênh lệch.
- Ngoài ra chương trình CodeVision AVR còn hỗ trợ rất nhiều cho chúng ta trong việc viết code khi cho AVR giao tiếp với các thiết bị bên ngoài như LCD, LM75, DS1621, DS1307, DS1820/1822...v.v..

b. Ghép nối với bộ nhớ SRAM bên ngoài.

Khả năng mở rộng bộ nhớ cho phép AVR ATMEGA8515 linh hoạt hơn khi thiết kế các hệ thống cần nhiều bộ nhớ. AVR MEGA8515 cho phép mở rộng bộ nhớ tới 64KB. Khi mở rộng bộ nhớ một vài IC giao tiếp được thêm vào và khi đó nó cho phép bộ nhớ ngoài trở thành một bộ phận lưu trữ của vi điều khiển.

Khi bộ nhớ ngoài được sử dụng thì PORTA và PORTC không còn được sử dụng cho mục đích vào ra thông thường nữa, nó trở thành các dây địa chỉ và dữ liệu phục vụ cho việc truy xuất tới bộ nhớ ngoài. chân WR, RD trên PD được sử dụng để làm các dây cho phép đọc/ghi dữ liệu và chân ALE trên PE được sử dụng để làm tín hiệu chốt phần byte thấp của . Sơ đồ khối kết nối khỏi RAM ngoài như Hình 9-6:



Hình 9-6 Kết nối AVR với bộ nhớ ngoài

c. Bộ truyền nhận nội tiếp.

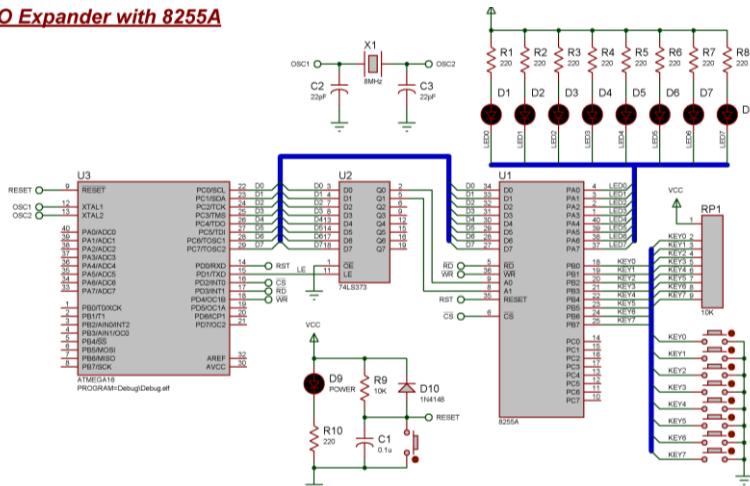
Bộ truyền nhận USART của AVR với nhiều tính năng được tích hợp sẵn dễ dàng trong việc truyền tin như hỗ trợ khung truyền 5-8 hoặc 9 bit dữ liệu, tính và kiểm tra parity phần cứng, tự động đặt cờ ngắt khi nhận được 1 byte khi truyền được 1 byte hay đếm truyền rồi. Tốc độ truyền baud được đặt bởi thanh ghi tốc độ USART baud rate UBRR0H, hỗ trợ phát hiện frame lỗi, tự động phát hiện tràn đếm dữ liệu và đặt cờ ngắt khi dữ liệu trong đếm vẫn còn mà nó lại phát hiện 1 frame khác...v.v. Sơ đồ cấu tạo bên trong của khối USART trong AVR sẽ nói rõ hơn điều này.

10.1.5 Một ví dụ ứng dụng cho AVR

Bài 1: Trong các ứng dụng với vi điều khiển thì số lượng cổng vào ra luôn được quan tâm. Hãy thực hiện mở rộng cổng vào ra cho AVR ATmega 16/32 (Hai vi điều khiển này có chức năng như nhau, chỉ khác nhau về dung lượng bộ nhớ, ngoài ra có thể dùng ATmega 8535 hay AT mega 64 cũng có sơ đồ chân và chức năng tương tự) dùng 8255A để thực hiện điều khiển các đèn LED và bàn phím đơn giản.

Giải: Ta thực hiện xây dựng mạch như Hình 9-7 sau:

I/O Expander with 8255A



Hình 9-7 Sơ đồ mạch mở rộng sử dụng 8255A

Chương trình điều khiển mẫu sử dụng ngôn ngữ C

```
#include <avr/io.h>
#define uchar unsigned char
#define uint unsigned int
// Low level port/pin definitions
#define sbit(x,PORTE) ((PORTE) /= (1<<x))
#define cbit(x,PORTE) ((PORTE) &= ~(1<<x))
#define pin(x,PIN) ((PIN) & (1<<x))
// 8255 pins definition
// RST
#define srst sbit(0,PORTE)
#define crst cbit(0,PORTE)
// LE
#define sle sbit(1,PORTE)
```

```

#define cle cbit(1,PORTD)
//CS
#define scs sbit(2,PORTD)
#define ccs cbit(2,PORTD)
//RD
#define srd sbit(3,PORTD)
#define crd cbit(3,PORTD)
//WR
#define swr sbit(4,PORTD)
#define cwr cbit(4,PORTD)
//I/O BUS
#define out PORTC
#define in PINC
//Address latch multiplexer.
void latch_it(void)
{
    cle;
    asm("nop");
    sle;
    asm("nop");
    cle;
}

int main()
{
    uchar temp;
    //Initialize Stack Pointer
    SPL = 0x54;
    SPH = 0x04;
    //Initialize PORTD & C for all output pins.
    DDRD = 0xff;
    //Disable 8255A
    srd;
    swr;
}

```

```

scs;
// PORTC pins can now safely initialized as outputs
DDRC=0xff;
// Enable the 8255A and reset it
ccs;
srst;
asm("nop"); asm("nop"); asm("nop");
crst;
//
out = 0x03;
latch_it();
out = 0x82;
cwr;
asm("nop");
swr;
while(1)
{
//
out = 0x01;
latch_it();
DDRC = 0;
asm("nop");
crd;
asm("nop");
temp = in;
srd;
//
DDRC = 0xff;
out = 0x00;
latch_it();
out = temp;
cwr;
asm("nop");

```

```

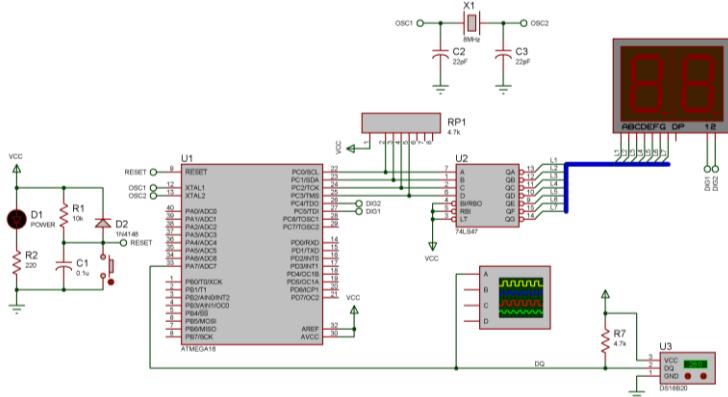
        SWR;
    }
}

```

Bài 2: Dùng vi điều khiển AVR để đo nhiệt độ sử dụng cảm biến DS18B20

Thực hiện thiết kế mạch trên phần mềm mô phỏng Proteus như Hình 9-8.

Temperature meter with DS18B20



Hình 9-8 Sơ đồ mạch ứng dụng đo nhiệt độ

Chương trình điều khiển mẫu sử dụng ngôn ngữ C

```

#include <avr/io.h>
#include <util/delay.h>

#define uchar unsigned char
#define uint unsigned int
#define BUS PORTC

// Low level port/pin definitions

```

```

#define sbit(x,PORT) (PORT) /=(1<<x)
#define cbit(x,PORT) (PORT) &= ~(1<<x)
#define pin(x,PIN) (PIN) & (1<<x)

// Pins definition
#define s_digit1 sbit(5,PORTC)
#define c_digit1 cbit(5,PORTC)
#define s_digit2 sbit(4,PORTC)
#define c_digit2 cbit(4,PORTC)
#define out PORTC
#define DQ_IN DDRA&= ~(1<<7)
#define DQ_OUT DDRA/= (1<<7)
#define S_DQ sbit(7,PORTA)
#define C_DQ cbit(7,PORTA)
#define DQ pin(7,PINA)

// Function Prototypes
void init_ds18b20(void);
uchar readbyte(void);
void writecommand(uchar);
uchar readtemp(void);
uchar a, b, tt;

// Main program
int main(void)
{
    uchar i=0, temp;
    // Initialize Stack Pointer
    SPL=0x54;
    SPH=0x04;
    // Configure port pins
    DDRC = 0xff;
    DDRA = 0xff;
}

```

```

while(1)
{
    temp = readtemp();
    for(i=0; i<10; i++) // 10 measures
    {
        // output the units
        out = (temp/10) & 0x0f;
        s_digit1;
        c_digit2;
        _delay_ms(5);
        // output the tens
        out = (temp%10) & 0x0f;
        c_digit1;
        s_digit2;
        _delay_ms(5);
    }
}

// Start transaction with 1-wire line.
void init_ds18b20(void)
{
    DQ_OUT;
    C_DQ;
    _delay_us(600);
    S_DQ;
    _delay_us(50);
    DQ_IN;
    while(DQ);
    _delay_us(240);
    DQ_OUT;
    S_DQ;
    _delay_us(300);
}

```

```

// Read a byte from the sensor
uchar readbyte(void)
{ uchar i = 0,data = 0;
  DQ_OUT;
  for (i=0; i<8; i++)
  { C_DQ ;
    data >>= 1;
    _delay_us(3);
    S_DQ;
    DQ_IN;
    _delay_us(12);
    if(DQ)
      data /= 0x80;
    DQ_OUT;
    S_DQ;
    _delay_us(45);
    _delay_us(5);
  }
  return(data);
}

// Write a command to the sensor
void writecommand(uchar data)
{ uchar i;
  for(i=0; i<8; i++)
  { C_DQ;
    _delay_us(15);
    if(data & 0x01)
      S_DQ;
    else
      C_DQ;
    _delay_us(45);
  }
}

```

```
    data >>= 1;
    S_DQ;
    _delay_us(2);
}

// Read value from the sensor
uchar readtemp(void)
{
    uint t;
    init_ds18b20();
    // Convert
    writecommand(0xCC);
    writecommand(0x44);
    init_ds18b20();
    // Read Scratch memory area
    writecommand(0xCC);
    writecommand(0xBE);
    a = readbyte();
    b = readbyte();
    t = b;
    t <<= 8;
    t = t/a;
    tt = t*0.0625;
    return(tt);
}
}
```

9.2 Họ vi điều khiển PIC

9.2.1 Tổng quan về PIC

PIC nói chung là họ vi điều khiển 8-bit/16-bit, dựa trên kiến trúc Harvard sửa đổi, với tập lệnh rút gọn (do vậy PIC thuộc loại kiến trúc RISC). PIC được sản xuất từ dòng cơ bản như PIC10 hay PIC12, qua dòng cấp thấp PIC16, cho đến dòng cấp cao PIC18. Hiện nay hãng Microchip đã có các họ vi điều khiển 16-bit, gồm PIC24H và PIC24F, DSPic 30F, DSPic33F. Hãng Microchip cũng đã giới thiệu vào tháng 11 năm 2007 họ vi điều khiển 32-bit, PIC32MX, dựa trên lõi MIPS32 M4K.

Hiện nay có khá nhiều dòng PIC và có rất nhiều khác biệt về phần cứng, nhưng chúng ta có thể điểm qua một vài nét như sau:

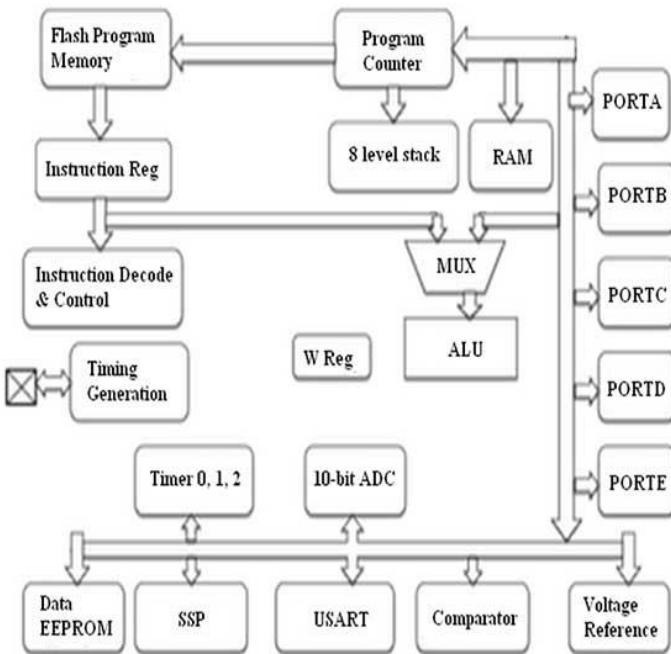
- 8/16 bit CPU, xây dựng theo kiến trúc Harvard có sửa đổi
- Flash và ROM có thể tùy chọn từ 256 byte đến 256 Kbyte
- Các cổng Xuất/Nhập (I/O ports) (mức logic thường từ 0V đến 5.5V, ứng với logic 0 và logic 1)
- 8/16 Bit Timer
- Công nghệ tiết kiệm năng lượng Nanowatt
- Tích hợp sẵn ác chuẩn giao tiếp ngoại vi nối tiếp đồng bộ/không đồng bộ USART, AUSART, EUSARTs
- Bộ chuyển đổi ADC Analog-to-digital converters, 10/12 bit
- Bộ so sánh điện áp (Voltage Comparators)

- Các module so sánh và phát xung Capture/Compare/PWM.
- Giao tiếp với LCD
- MSSP Peripheral dùng cho các giao tiếp I²C, SPI, và I²S
- Bộ nhớ nội EEPROM - có thể ghi/xoá lên tới 1 triệu lần
- Module Điều khiển động cơ, đọc encoder
- Hỗ trợ giao tiếp USB như PIC 18F4550
- Hỗ trợ điều khiển Ethernet hoặc dễ dàng giao tiếp SPI với chip Ethernet.
- Hỗ trợ giao tiếp theo chuẩn truyền thông công nghiệp CAN
- Hỗ trợ giao tiếp LIN
- Hỗ trợ giao tiếp IrDA
- Một số dòng có tích hợp bộ RF (PIC16F639, và rfPIC)
- KEELOQ Mã hoá và giải mã
- DSP những tính năng xử lý tín hiệu số (dsPIC)

9.2.2 Đặc trưng phần cứng của vi điều khiển PIC

9.2.2.1 CPU

Cấu trúc nhân CPU của PIC như Hình 9-9. Giống như AVR, đơn vị xử lý trung tâm của PIC cũng được cấu tạo theo kiến trúc RISC với một số lượng nhỏ các lệnh đơn giản. Do đó, chúng đòi hỏi phần cứng ít hơn, giá thành thấp hơn, và nhanh hơn so với CISC. Tuy nhiên nó đòi hỏi người lập trình phải viết các chương trình phức tạp hơn, nhiều lệnh hơn.



Hình 9-9 Cấu trúc nhân CPU dòng PIC 16

9.2.2.2 Bộ nhớ

Cấu trúc bộ nhớ của vi điều khiển PIC bao gồm bộ nhớ chương trình (Program memory) và bộ nhớ dữ liệu (Data Memory).

Bộ nhớ chương trình của vi điều khiển PIC là bộ nhớ flash, dung lượng bộ nhớ tùy theo từng vi điều khiển từ 2KB - 128KB word (1 word = 14 bit) và được phân thành nhiều trang. Khi vi điều khiển được reset, bộ đếm chương trình sẽ chỉ đến địa chỉ 0000h (Reset vector). Khi có ngắt xảy ra, bộ đếm chương trình sẽ chỉ đến địa chỉ 0004h (Interrupt vector). Bộ nhớ chương trình

không bao gồm bộ nhớ stack và không được địa chỉ hóa bởi bộ đếm chương trình.

Bộ nhớ dữ liệu của PIC là bộ nhớ EEPROM được chia ra làm nhiều bank. Giả sử đối với PIC16F877A bộ nhớ dữ liệu được chia ra làm 4 bank. Mỗi bank có dung lượng 128 byte, bao gồm các thanh ghi có chức năng đặc biệt SFG (Special Function Register) nằm ở các vùng địa chỉ thấp và các thanh ghi mục đích chung GPR (General Purpose Register) nằm ở vùng địa chỉ còn lại trong bank. Các thanh ghi SFR thường xuyên được sử dụng (ví dụ như thanh ghi STATUS) sẽ được đặt ở tất cả các bank của bộ nhớ dữ liệu giúp thuận tiện trong quá trình truy xuất và làm giảm bớt lệnh của chương trình.

9.2.2.3 Một số giao tiếp ngoại vi đặc trưng của PIC

Về cơ bản các dòng vi điều khiển PIC cũng được trang bị sẵn các ngoại vi thông dụng như cổng vào ra nối tiếp, song song, ngắt ngoài... Một số loại có chức năng đặc biệt được trang bị sẵn các cổng ghép nối với chuẩn giao tiếp USB (PIC 18F4550) hay Ethernet như nhau, hay bộ chuyển đổi dữ liệu tương tự sang số ADC, bộ chia xung PWM ngoài ra còn có một số ngoại vi đặc trưng như giao tiếp theo chuẩn SPI, I2C, 1 wire, CAN, ...

a. Giao tiếp theo chuẩn truyền thông công nghiệp CAN.

- Một số dòng PIC hỗ trợ giao tiếp theo chuẩn truyền thông công nghiệp CAN với modul ECANTM theo chuẩn 2.0B như dòng PIC24xx hay PIC33xx.
- Đệm gửi 8 từ, và đệm nhận 32 từ
- Hỗ trợ chế độ địa chỉ theo chuẩn DeviceNetTM, các chế độ lắng nghe, tự động xử lý các yêu cầu gửi dữ liệu, điều khiển đường truyền, Wake – up từ chế độ Sleep bằng thông điệp của chuẩn CAN.

b. Giao tiếp theo chuẩn Ethernet.

Đây là chuẩn không được thiết kế phần cứng trên chip, nhưng nhà sản xuất Microchip đã đưa ra bộ thư viện về lập trình giao tiếp theo chuẩn Ethernet có tên là TCP/IP Stack cho phép phần lớn các vi điều khiển PIC có chuẩn giao tiếp SPI kết nối với một vi mạch Ethernet controller để ghép nối và giao tiếp mạng. Thông thường các ghép nối dùng vi mạch điều khiển ghép nối mạng NIC ENC 28J60 có tốc độ 10Mbps hay ENC 28J600 có tốc độ 100Mbps, các bộ giao tiếp mạng cũng được Microchip giới thiệu và bán sẵn như *PIC32 Ethernet Starter Kit*.

9.2.3 Tập lệnh của PIC

9.2.3.1 Giới thiệu về trình dịch CCS c:

- CCS là trình biên dịch dùng ngôn ngữ C lập trình cho PIC. Đây là ngôn ngữ lập trình đầy sức mạnh, giúp bạn nhanh chóng trong việc viết chương trình hơn là Assembly
- CCS chứa rất nhiều hàm phục vụ cho mọi mục đích và có rất nhiều cách lập trình mà cho cùng 1 vấn đề với tốc độ thực thi và độ dài chương trình khác nhau. Sự tối ưu là do kĩ năng lập trình của mỗi người
- CCS cung cấp các công cụ tiện ích giám sát hoạt động chương trình như:
 - + C/ASM list: cho phép m\$ ASM của file bạn biên dịch, giúp bạn quản lý và nắm rõ cách thức nó được sinh ra, là công cụ rất quan trọng giúp bạn có thể gỡ rối chương trình
 - + SYMBOL: hiển thị bộ nhớ cấp phát cho từng biến, giúp bạn quản lý bộ nhớ các biến của chương trình

- + CALLTREE: hiển thị phân bố bộ nhớ

9.2.3.2 Chỉ thị tiền xử lý :

1) #include :

Cú pháp: #include<filename>

Filename: tên file cho thiết bị *.h, *.c . Chỉ định đường dẫn cho trình biên dịch , luôn phải có để khai báo chương trình viết cho VDK nào và phải luôn đặt ở dòng đầu tiên
VD: #include<16F877A.H>

2) #bit :

Cú pháp: #bit name = x.y

Name: tên biến

- + X: biến C(8,16,32...bit) hay hàng số địa chỉ thanh ghi
- + Y: vị trí của bit trong x
- + Tạo biến 1bit đặt ở byte x vị trí y tiện dùng kiểm tra hay gán giá trị cho thanh ghi.

VD : #Bit TMR1IF = 0x0B.2;

3) #byte :

- Cú pháp: #byte name = x

Name: tên biến

X:địa chỉ

Gán tên biến name cho địa chỉ x , name thòng dùng để gán cho các thanh ghi

VD : #Byte portb = 0x06;

4) #define :

- Cú pháp: #define name text

Name: tên biến

Text : chuỗi hay số

VD : #Define A 12345

5) #use :

- Cú pháp: #use delay(clock = speed) Speed: tốc độ dao động của thạch anh

Có chỉ thị này chúng ta mới dùng được hàm delay_ms hoặc delay_us

VD: #use delay(clock = 4000000);

6) #use fast_io :

- Cú pháp: #use fast_io(port)

Port : các cổng vào ra của PIC(từ A-G)

Dùng cái này chúng ta có thể điều chỉnh các port với chỉ 1 lệnh

VD: # use fast_io(a);

9.2.3.3 Các hàm delay :

1) delay_ms(time)

Time: giá trị thời gian cần tạo trễ

VD : delay_ms(1000); // trễ 1s

2) delay_us(time)

Time: giá trị thời gian cần tạo trễ

VD : delay_us(1000); // trễ 1ms

Hàm delay này không sử dụng bất cứ Timer nào cả mà chỉ là 1 nhóm lệnh vô nghĩa thực hiện trong khoảng thời gian bạn đã định sẵn

Trước khi sử dụng các hàm này cần phải khai báo tiền định #use_delay(....)

9.2.3.4 Các hàm vào ra trong CCS c

1) Output_low(pin) – Output_high(pin)

Thiết lập mức 0v(low) hoặc 5v(high) cho các chân của PIC

VD : output_low(pin_D0) ;

2) Output_bit(pin,value)

Pin: tên chân của PIC

Value: giá trị 0 hay 1

VD: output_bit(pin_C0,1);

3) Output_X(value)

X: tên các port trên chip

Value: giá trị 1 byte

VD: output_B(255);

4) Input_X()

X: tên các port trên chip

Hàm này trả giá trị 8 bit là giá trị hiện hữu của port đó

VD: n = input_A();

5) Set_tris_X(value)

X: tên chân (A – G)

Value: là giá trị 8bit điều khiển vào ra cho các chân của chip

1: nhập dữ liệu 0: xuất dữ liệu

VD: set_tris_B(0); // tất cả các chân của portb là ngõ ra

9.2.3.5 Các hàm phục vụ ngắt

- enable_interrupts(level)

Trong đó level là:

- + GLOBAL : cho phép ngắt toàn cục
- + INT_TIMER0 : ngắt do tràn Timer0
- + INT_TIMER1 : ngắt do tràn Timer1
- + INT_TIMER2 : ngắt do tràn Timer2
- + INT_RB : có thay đổi 1 trong các chân RB4 RB7
- + INT_EXT : ngắt ngoài trên chân RB0

9.2.3.6 Các hàm điều chế độ rộng xung -

ETUP_CCPx(mode):

Dùng trước hết để thiết lập chế độ hoạt động hay vô hiệu hóa tính năng CCP

X: tên chân CCP trên chip (với PIC 16F877A đó là các chân RC1-CCP2 ; RC2-CCP1)

Mode: CCP_PWM (bật chế độ PWM)

- SET_CCPx_DUTY(value)

X: tên chân CCP trên chip

Value: giá trị 8 hay 16 bit

Nó ghi 10 bit giá trị vào thanh ghi CCPx , nếu value chỉ có 8 bit thì nó sẽ dịch thêm 2 bit nữa để đủ 10 bit nạp vào CCPx. Tuỳ độ phân giải mà giá trị của value không phải lúc nào cũng đạt tới giá trị 1023

9.2.4 Cơ bản về dsPIC

Chỉ xét riêng phần vi điều khiển, dsPIC giống như PIC24, là các vi điều khiển 16-bit, dựa trên kiến trúc Harvard sửa đổi, với tập lệnh rút gọn (như vậy dsPIC và PIC24 cũng thuộc loại RISC). DSPIC hiện nay gồm có hai dòng: dsPIC30F và dsPIC33F.

Sự khác biệt giữa dsPIC với các dòng PIC thông thường là nó có sẵn bộ xử lý tín hiệu số trên chíp.

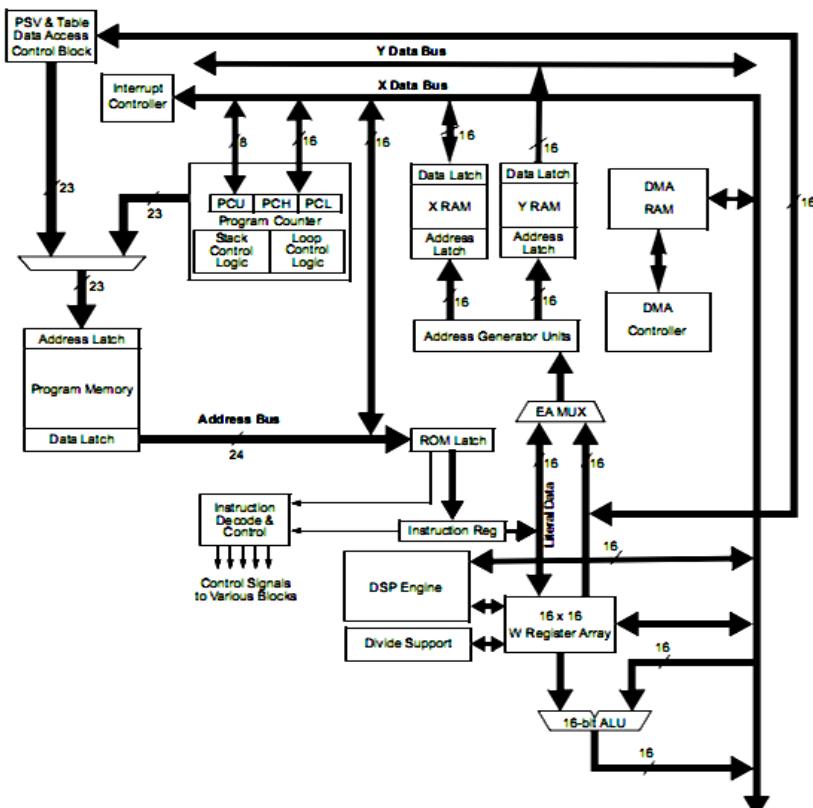
9.2.5 Một số đặc tính cơ bản của dsPIC họ dsPIC 33FJ

Dòng dsPIC là dòng vi xử lý 16 bit được trang bị nhiều tính năng cải tiến mới thuận tiện cho việc xây dựng các ứng dụng.

- Tốc độ xử lý lệnh lên tới 40MHz (40 triệu lệnh 1 giây).
- Điện áp cung cấp 3 – 3,6V

- Nhiệt độ hoạt động nằm trong dải -40°C tới +85°C
- dsPIC 33FJ256 với 256 KB bộ nhớ FLASH, 30KB bộ nhớ RAM bao gồm cả 2KB bộ nhớ RAM ghép nối thông qua giao diện DMA
- Nhiều chế độ tiết kiệm điện.

9.2.5.1 Kiến trúc CPU:



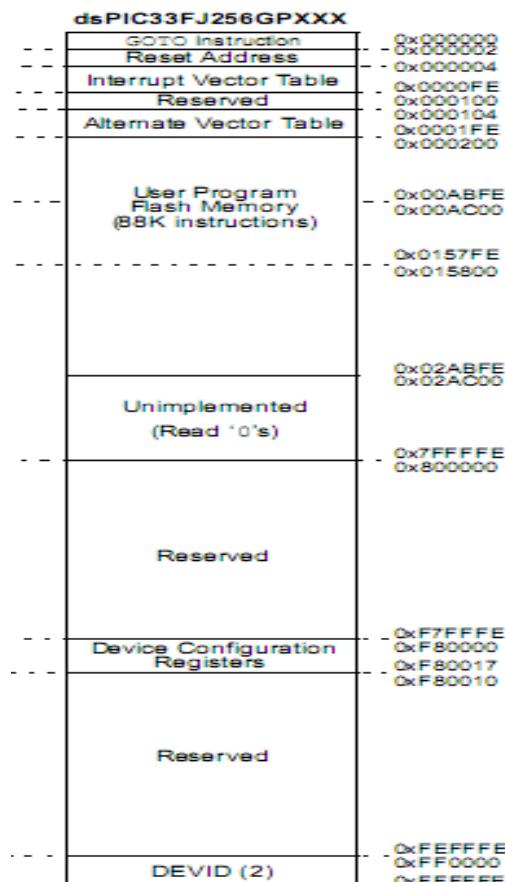
Hình 9-10 Cấu trúc lõi CPU

- Lõi CPU có kiến trúc Harvard sử dụng tập lệnh RISC cho phép giảm kích thước của chương trình viết bằng ngôn ngữ C.
 - Bên trong vi xử lý có 16 thanh ghi đa năng 16bit, 2 thanh ghi tích lũy 40 bit với các chế độ làm tròn trong tính toán.
 - Thực thi được các phép toán trên số thập phân và phép toán trên số nguyên, các phép chia số 16bit và số 32bit.
 - Bus dữ liệu 16 bit
 - Mở rộng bộ nhớ ngoài tới 64KB
 - Hệ lệnh với 83 lệnh hầu hết thực hiện trong 1 chu kỳ xung nhịp
 - Cấu trúc lõi CPU theo kiến trúc RISC.

9.2.5.2 Tổ chức bộ nhớ trên dsPIC

- DSPic 33FJ256 có 24 bit địa chỉ
 - Vùng địa chỉ từ 0x000000 tới 0x000002 dùng để chứa các lệnh GOTO.
 - Vùng địa chỉ từ 0x000002 tới 0x000004 dùng để lưu địa chỉ Reset
 - Vùng địa chỉ từ 0x000004 tới 0x0000FE dùng để lưu bảng vecto ngắt. Như vậy DSPic 33FJ256 có thể lưu 252 ngắt tại bảng vecto ngắt.
 - Vùng địa chỉ từ 0x000100 tới 0x000104 là vùng dành riêng.
 - Vùng địa chỉ từ 0x000104 tới 0x0001FE là bảng vecto luân phiên.
 - Vùng địa chỉ từ 0x000200 tới 0x0157FE được dùng làm nơi chứa các lệnh lập trình. Vùng này có độ lớn 88KB

- Ngoài ra là các vùng nhớ dành riêng để chứa các dữ liệu riêng tùy từng yêu cầu cụ thể của từng bài toán điều khiển
- Hình dưới đây là sơ đồ tổ chức bộ nhớ của DSPIC 33FJ256.



Hình 9-11 Tổ chức bộ nhớ DSPIC

Một số vi điều khiển PIC đời mới có sẵn phần cứng hỗ trợ chuẩn giao tiếp DMA thường dùng trong các giao tiếp với bộ nhớ. Với DSPIC 33FJ256 có 8 kênh DMA.

- Bộ đệm DMA là 2 KB phục vụ cho đệm truyền nhận dữ liệu với Ram được ghép nối theo DMA với chip.
- Cho phép truy nhập trao đổi dữ liệu theo kiểu DMA ngay trong khi CPU thực hiện lệnh.

9.2.5.3 Các ngắt của DSPIC 33FJ256

- DSPIC 33FJ256 hỗ trợ 5 ngắt ngoài
- Có tới 118 vectors ngắt và 63 nguồn gây ngắt.
- Với 7 mức độ ưu tiên cho các ngắt thuận tiện cho việc phân cấp các ngắt.
- Cung cấp 5 ngắt ngoại lệ để xử lý các ngoại lệ có thể xảy ra, tránh việc khởi động lại.

9.2.5.4 3 Các chân vào ra

- DSPIC 33FJ256 có tới 85 chân vào ra có thể lập trình được.
- Chế độ Wake – up từ ngắt/ thay đổi tới từ 24 chân
- Điện áp ra trên các chân từ 3 tới 3,6 V
- Cho phép mức điện áp 5V ở các chân vào.
- Dòng điện ra ở mức 4mA ở tất cả các chân đầu ra.

9.2.5.5 Các bộ Timer

- DSPIC 33FJ256 có tất cả 9 bộ timer 16 bit
- Có thể kết hợp hai bộ timer lại (2+3, 4+5, 6+7, 8+9) để tạo ra một bộ timer 32 bit như vậy ta có 4 bộ timer 32 bit.

- Một bộ timer có khả năng chạy với xung nhịp thời gian thực được cấp từ một bộ dao động 32.768KHz
- Có thể lập trình để tới 8 kênh so sánh đầu vào, hay 8 kênh so sánh đầu ra 16 bit độ phân giải tối đa < 100ns
- Bộ PWM 16 bit
- Bộ đếm sự kiện và so sánh chân vào(8 kênh) và chân ra(8 kênh)

9.2.5.6 Các cổng giao tiếp

a. Giao tiếp SPI

- Có 2 modul SPI
- Hỗ trợ 8 hoặc 16 bit dữ liệu với nhiều tốc độ xung nhịp và trích mẫu khác nhau.
- Giống như AVR chuẩn SPI trên PIC cũng có thể cấu hình theo chế độ multi master hay master/ slaver.

b. Giao tiếp I²C

- Có 2 modul I²C hỗ trợ đầy đủ cho chế độ chủ tớ.
- Hỗ trợ từ 7 hoặc 10 bit địa chỉ tự động phát hiện tranh chấp đường truyền, sử dụng mặt lạ địa chỉ slaver để truyền nhận dữ liệu

c. Giao tiếp nối tiếp USART

- Có 2 modul USART giao tiếp theo các chuẩn RS-232, RS-485, LIN
- Cho phép ngắt khi phát hiện bit địa chỉ, ngắt khi phát hiện dữ liệu lỗi, ngắt tự động Wake – up từ chế độ Sleep.
- Hỗ trợ việc mã hóa và giải mã bằng phần cứng với chuẩn IrDA.

d. Modul chuyển đổi dữ liệu DCI

- Modul chuyển đổi dữ liệu hỗ trợ chuyển đổi dữ liệu sang chuẩn I²S và chuẩn AC'97.
- Độ rộng từ 16bit và 1 frame có 16 từ.
- Đệm nhận 4 từ và đệm gửi 4 từ.

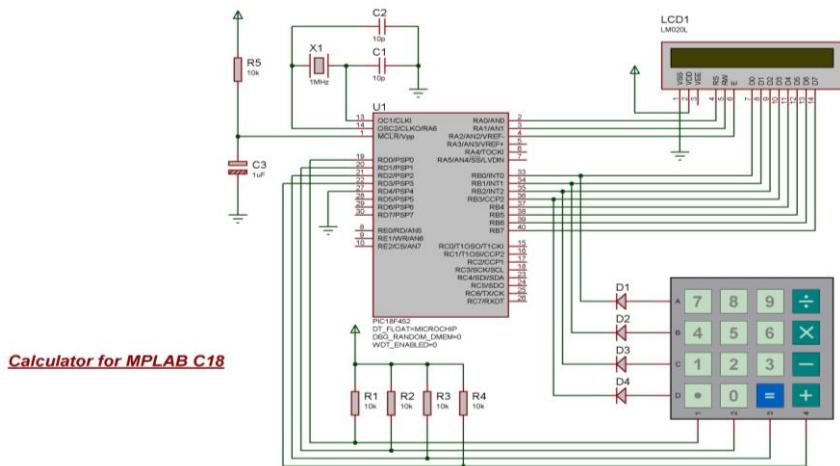
9.2.5.7 Bộ chuyển đổi ADC

- Có 2 modul ADC trong 1 chip với 2 chế độ hoạt động, ở chế độ 10bit tốc độ chuyển đổi lên tới 1,1 Msps và chế độ 12 bit với tốc độ 500Ksp
- 2, 4, 8 mẫu đồng thời với 32 kênh đầu vào tự động quét

9.2.6 Ví dụ ứng dụng vi điều khiển PIC

Bài 3: Xây dựng một máy tính đơn giản sử dụng vi điều khiển PIC

Ta xây dựng mạch sử dụng vi điều khiển PIC 18F452 trên phần mềm mô phỏng như sau:



```

{ lcd_init();
  calc_evaluate();
}
VOID calc_evaluate()
{ CHAR number[MAX_DISPLAY_CHAR+1], key;
  INT8 pos;
  FLOAT tmp;
  // Initialize static values:
  lvalue = 0;
  rvalue = 0;
  lastop = 0;
  // Display a Zero to start:
  calc_format(0);
  // Clear the buffer before we start.
  pos = 0;

  for (;;)
  { key = calc_getkey();
    if (calc_testkey(key))
    { // Key test positive for digit so we read it into the
      // buffer and then write the buffer to the screen/LCD.
      // Size limit the number of digits - allow for
      termination
      // and possible negative results.
      if (pos != MAX_DISPLAY_CHAR - 2)
      { number[pos++] = key;
        number[pos] = 0;
        calc_display(number);
      }
    }
    else
    { // If a number has been entered, then evaluate it.

```

```

// The number is stored to lvalue if we have no
current operator,
// or else rvalue if we do.
if(pos != 0)
{ tmp = atof(number);
if(lastop == 0)
    lvalue = tmp;
else
    rvalue = tmp;
}
// Reset the input buffer.
pos = 0;
if(lastop != 0)
    calc_opfunctions(lastop);
if(key != '=')
    lastop = key;
else
    lastop = 0;
}
}
}

VOID calc_opfunctions (CHAR token)
// Handle the operations. Lvalue holds the result and we
test for
// consecutive operator presses.
{ INT8 result = OK;
switch (token)
{ case '+': lvalue += rvalue; break;
case '-': lvalue -= rvalue; break;
case '*': lvalue *= rvalue; break;
case '/':
    if(rvalue != 0)

```

```

    lvalue /= rvalue;
else
    result = ERROR;
break;
}
if(result == OK)
    calc_format(lvalue);
else if(result == ERROR)
{ char buf[] = "*ERROR*";
    calc_display(buf);
}
}

VOID calc_format (FLOAT f)
// Microchip C18's math libraries are not as accurate as
// they should be, so we have to use a table of divisors.
{ static const float divisors[] =
    { 100000000,
        10000000,
        1000000,
        100000,
        10000,
        1000,
        100,
        10,
        1,
        0.1,
        0.01,
        0.001,
        0.0001,
        0.00001,
        0.000001,
        0.0000001,

```

```

0
};

CHARdbuf[MAX_DISPLAY_CHAR+1];
FLOAT divisor, tmp;
INT count = 0, digit;
INT pad=0, p=0;
// Sort out minus sign:
if(f>= 0)
   dbuf[p++] = ' ';
else
{ dbuf[p++] = '-';
f = -f;
}
if(f>= divisors[0])
   dbuf[p++] = 'E';
else
while(p < MAX_DISPLAY_CHAR &&
((divisor=divisors[count++]) >= 1 //f > 0.0000001))
{ if(fabs(f)<1)
    digit = f/divisor+0.05;
else
    digit = f/divisor;
if(divisor == 0.1)
   dbuf[p++] = '.';
if(digit != 0 // divisor < 10)
{ dbuf[p++] = digit + '0';
pad = TRUE;
}
else if(pad)
dbuf[p++] = '0';
tmp = digit*divisor;
f -= tmp;
}

```

```

        }
        dbuf[p] = 0;
        calc_display(dbuf);
    }
BOOL calc_testkey (CHAR key)
// Test whether the key is a digit, a decimal point or an
operator.
// Return 1 for digit or decimal point, 0 for op.
{ if ((key == '.') || ((key >= '0') && (key <= '9')))
    return TRUE;
else
    return FALSE;
}
CHAR calc_getkey (VOID)
// Use the input routine from the *Keypad_Read*
// Scan for a key and return ASCII value of the Key
pressed.
{ CHAR mykey;
while ((mykey = keypadread()) == 0x00)
    /* Poll again */;
return mykey;
}
VOID calc_display (CHAR *buf)
// *LCD_Write* assembly file to output ASCII values to
the LCD.
{ INT8 i;
    clearscreen();
    for (i=0 ; buf[i] != 0; i++)
// { if (buf[calc_testkey(buf[i]) || buf[i] == 0x2D])
        { wrdata(buf[i]); }
// }
}

```

9.3 Họ vi điều khiển ARM

9.3.1 Tổng quan về ARM

Cấu trúc ARM (viết tắt từ tên gốc là Acorn RISC Machine) là một loại cấu trúc vi xử lý 32-bit kiểu RISC được sử dụng rộng rãi trong các thiết kế nhúng. Do có đặc điểm tiết kiệm năng lượng, các bộ CPU ARM chiếm ưu thế trong các sản phẩm điện tử di động, mà với các sản phẩm này việc tiêu tán công suất thấp là một mục tiêu thiết kế quan trọng hàng đầu.

Ngày nay, hơn 75% CPU nhúng 32-bit là thuộc họ ARM, điều này khiến ARM trở thành cấu trúc 32-bit được sản xuất nhiều nhất trên thế giới. CPU ARM được tìm thấy khắp nơi trong các sản phẩm thương mại điện tử, từ thiết bị cầm tay (PDA, điện thoại di động, máy đa phương tiện, máy trò chơi cầm tay, và máy tính cầm tay) cho đến các thiết bị ngoại vi máy tính (ổ đĩa cứng, bộ định tuyến để bàn.) Một nhánh nổi tiếng của họ ARM là các vi xử lý Xscale của Intel.

Ngày 26/4/1985, mẫu sản phẩm ARM đầu tiên sản xuất tại công ty kỹ thuật VLSI, SanJose, bang California được chuyển tới trung tâm máy tính Acorn ở Cambridge, Anh Quốc. Một vài giờ sau, chương trình thử nghiệm đầu tiên đã thành công.

Nửa thập niên sau đó, ARM được phát triển rất nhanh chóng để làm nhân máy tính để bàn của Acorn, nền tảng cho các máy tính hỗ trợ giáo dục ở Anh. Trong thập niên 1990, dưới sự phát triển của Acorn Limited, ARM đã thành một thương hiệu đứng đầu thế giới về các ứng dụng sản phẩm nhúng đòi hỏi tính năng cao, sử dụng năng lượng ít và giá thành thấp.

Chính nhờ sự nổi trội về thị phần đã thúc đẩy ARM liên tục được phát triển và cho ra nhiều phiên bản mới. Những thành công quan trọng trong việc phát triển ARM ở thập niên sau này:

- Giới thiệu ý tưởng về định dạng các chỉ lệnh được nén lại (thumb) cho phép tiết kiệm năng lượng và giá thành ở những hệ thống nhỏ.
- Giới thiệu họ điều khiển ARM9, ARM10 và ‘Strong ARM’
- Phát triển môi trường làm việc ảo của ARM trên PC.
- Các ứng dụng cho hệ thống nhúng dựa trên nhân xử lý ARM ngày càng trở nên rộng rãi.

Hầu hết các nguyên lý của hệ thống trên chip (Systems on chip-SoC) và cách thiết kế bộ xử lý hiện đại được sử dụng trong ARM, ARM còn đưa ra một số khái niệm mới (như giải nén động các dòng lệnh). Việc sử dụng 3 trạng thái nhận lệnh-giải mã-thực thi trong mỗi chu kỳ máy mang tính quy phạm để thiết kế các hệ thống xử lý thực. Do đó, nhân xử lý ARM được sử dụng rộng rãi trong các hệ thống phức tạp

Hiện nay các dòng vi điều khiển ARM đã phát triển lên thế hệ 7, 8, 9 - thế hệ vi điều khiển 64 bit, đi kèm theo đó là các công nghệ xử lý đồ họa, tích hợp thông minh và tiết kiệm năng lượng.

9.3.2 Một số đặc trưng phần cứng ARM

Để đạt được một thiết kế gọn, đơn giản và nhanh, các nhà thiết kế ARM xây dựng nó theo kiểu nối cứng không có vi chương trình, giống với bộ vi xử lý 8-bit 6502 đã từng được dùng trong các máy vi tính trước đó của hãng Acorn.

Cấu trúc ARM bao gồm các đặc tính của RISC như sau:

- Cơ chế truy xuất bộ nhớ chương trình và bộ nhớ dữ liệu đồng thời.
- Tập lệnh trực giao.
- Bộ nhớ thanh ghi lớn gồm 16 thanh ghi x 32-bit
- Chiều dài mã máy cố định là 32 bit để dễ giải mã và thực hiện pipeline, để đạt được điều này phải chấp nhận giảm mật độ mã máy.
- Hầu hết các lệnh đều thực hiện trong vòng một chu kỳ máy.

So với các bộ vi xử lý cùng thời như Intel 80286 và Motorola 68020, trong ARM có một số tính chất khá độc đáo như sau:

- Hầu hết tất cả các lệnh đều cho phép thực thi có điều kiện, điều này làm giảm việc phải viết các tiêu đề rẽ nhánh cũng như bù cho việc không có một bộ dự đoán rẽ nhánh.
- Trong các lệnh số học, để chỉ ra điều kiện thực hiện, người lập trình chỉ cần sửa mã điều kiện
- Có một thanh ghi dịch đóng thùng 32-bit mà có thể sử dụng với chức năng hoàn hảo với hầu hết các lệnh số học và việc tính toán địa chỉ.
- Có các kiểu định địa chỉ theo chỉ số rất mạnh
- Có hệ thống con thực hiện ngắt hai mức ưu tiên đơn giản nhưng rất nhanh, kèm theo cho phép chuyển từng nhóm thanh ghi.

9.3.3 Tập lệnh ARM

Quá trình thực thi một lệnh trong vi điều khiển hầu hết được thực hiện có thể các bước sau:

- Nhận lệnh từ bộ nhớ <fetch>

- Giải mã lệnh, xác định các tác động cần có và kích thước lệnh <decode>
- Truy cập các toán hạng có thể được yêu cầu từ thanh ghi <reg>
- Kết hợp toán hạng đầy với để tạo thành kết quả hay địa chỉ bộ nhớ <ALU>
- Truy cập vào bộ nhớ cho toán hạng dữ liệu nếu cần thiết <mem>
- Viết kết quả ngược lại bằng thanh ghi <res>

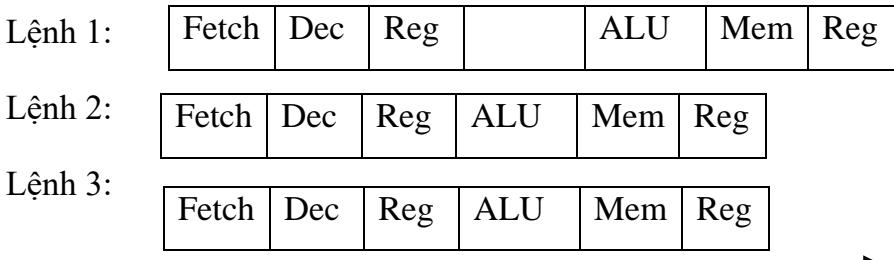
Tập lệnh cho vi điều khiển thông thường có hai tập lệnh CISC và RISC. Ta có thể thấy sự khác biệt qua các đặc điểm quan trọng sau:

- Kích thước các chỉ lệnh là cố định <32 bit> với chỉ một vài định dạng. (CISC có kích thước tập lệnh thay đổi với rất nhiều định dạng khác nhau)
- Sử dụng kiến trúc load-store các chỉ lệnh xử lý dữ liệu hoạt động chỉ trong thanh ghi và cách ly với các chỉ lệnh truy cập bộ nhớ (CISC cho phép giá trị trong bộ nhớ được dùng như toán hạng trong các chỉ lệnh xử lý dữ liệu)
- Gồm một số lớn các thanh ghi đa dụng 32 bit, cho phép cấu trúc load-store hoạt động hiệu quả. (CISC có rất nhiều thanh ghi, nhưng hầu hết chỉ để sử dụng cho một mục đích riêng biệt nào đây-ví dụ các thanh ghi dữ liệu và địa chỉ trong Motorola MC68000)

Tổ chức tập lệnh RISC:

Giải mã các chỉ lệnh logic bằng kết nối phần cứng (CISC sử dụng rất nhiều code trong ROM giải mã các chỉ lệnh).

Thực thi chỉ lệnh theo cấu trúc dòng chảy <xem hình dưới> (CISC ít khi cho phép các dòng lệnh thực thi kiểu này, chúng phải tuân tự hết dòng lệnh này mới đến dòng lệnh khác)



Một chỉ lệnh thực thi trong 1 chu kì xung nhịp (CISC cần nhiều chu kì xung nhịp để hoàn thành một lệnh).

Tập lệnh

Tất cả lệnh của ARM đều là 32bit:

- Có cấu trúc dạng load-store.
- Cấu trúc lệnh định dạng 3 địa chỉ (nghĩa là địa chỉ của 2 toán hạng nguồn và toán hạng đích đều là các địa chỉ riêng biệt)
- Mỗi chỉ lệnh thực thi một điều kiện.
- Có cả chỉ lệnh load-store nhiều thanh ghi đồng thời.
- Có khả năng dịch bit kết hợp với thực thi lệnh ALU trong chỉ 1 chu kì máy.

Các lệnh của ARM có thể chia thành các loại sau:

- Cấu trúc chỉ lệnh có 4 địa chỉ, dạng lệnh là:

Tên lệnh Địa chỉ 1 Địa chỉ 2 Địa chỉ Đích Địa chỉ kế tiếp
Ví dụ:

ADD d, s1, s2, next_i ;d := s1 +s2

- Cấu trúc chỉ lệnh có 3 địa chỉ: Dạng lệnh:

Tên lệnh Địa chỉ 1 Địa chỉ 2 Địa chỉ Đích

Ví dụ:

ADD d, s1, s2 ; d := s1 + s2

- Cấu trúc chỉ lệnh có 2 địa chỉ: Dạng lệnh:

Tên lệnh Địa chỉ 1 Địa chỉ Đích

Ví dụ:

ADD d, s1 ; d := d + s1

- Cấu trúc chỉ lệnh có 1 địa chỉ: Dạng lệnh:

Tên lệnh Địa chỉ 1

Ví dụ:

ADD s1 ; accumulator := accumulator + s1

- Cấu trúc chỉ lệnh không truy cập địa chỉ: Dạng lệnh:

Tên lệnh

Ví dụ

ADD ; top_of_stack := top_of_stack +next_on_stack

9.3.4 Giới thiệu bộ xử lý ARM Cortex-M3

Trong gần một thập kỷ qua, dòng vi xử lý ARM7 đã được sử dụng rất rộng rãi. Bộ vi xử lý Cortex-M3 được xây dựng trên nền tảng này nên việc nâng cấp từ dòng ARM7 lên Cortex-M3 là hợp lý và dễ dàng. Lõi trung tâm làm việc hiệu quả hơn, mô hình lập trình đơn giản, cách xử lý ngắt tắt định (deterministic interrupt behaviour), việc tích hợp các thiết bị ngoại vi giúp nâng cao hiệu năng làm việc mà vẫn giữ được chi phí thấp.

9.3.4.1 Kiến trúc và tính năng vi xử lý Cortex-M3

Bộ vi xử lý Cortex-M3 dựa trên kiến trúc ARMv7-M có cấu trúc thứ bậc. Nó tích hợp lõi xử lý trung tâm, gọi là CM3Core,

với các thiết bị ngoại vi hệ thống tiên tiến để tạo ra các khả năng như kiểm soát ngắn, bảo vệ bộ nhớ, gỡ lỗi và theo vết hệ thống.

Các thiết bị ngoại vi có thể được cấu hình một cách thích hợp, cho phép bộ vi xử lý Cortex-M3 đáp ứng được rất nhiều ứng dụng và yêu cầu khắt khe của hệ thống. Lõi của bộ vi xử lý Cortex-M3 và các thành phần tích hợp (hình 3) đã được thiết kế đặc biệt để đáp ứng yêu cầu bộ nhớ tối thiểu, năng lượng tiêu thụ thấp và thiết kế nhỏ gọn.

9.3.4.2 Lõi Cortex-M3

Lõi trung tâm Cortex-M3 dựa trên kiến trúc Harvard, được đặc trưng bằng sự tách biệt giữa vùng nhớ chứa dữ liệu và chương trình do đó có các bus riêng để truy cập (hình 3). Đặc tính này khác với dòng ARM7 dựa trên kiến trúc Von Neumann sử dụng chung vùng nhớ để chứa dữ liệu và chương trình, do đó dùng chung bus cho việc truy xuất. Vì có thể đọc cùng lúc lệnh và dữ liệu từ bộ nhớ, bộ vi xử lý Cortex-M3 có thể thực hiện nhiều hoạt động song song, tăng tốc thực thi ứng dụng.

Lõi Cortex có cấu trúc đường ống gồm 3 tầng: Instruction Fetch, Instruction Decode và Instruction Execute. Khi gấp một lệnh nhánh, tầng decode chứa một chỉ thị nạp lệnh suy đoán có thể dẫn đến việc thực thi nhanh hơn. Bộ xử lý nạp lệnh dự định rẽ nhánh trong giai đoạn giải mã. Sau đó, trong giai đoạn thực thi, việc rẽ nhánh được giải quyết và bộ vi xử lý sẽ phân tích xem đâu là lệnh thực thi kế tiếp. Nếu việc rẽ nhánh không được chọn thì lệnh tiếp theo đã sẵn sàng. Còn nếu việc rẽ nhánh được chọn thì lệnh rẽ nhánh đó cũng đã sẵn sàng ngay lập tức, hạn chế thời gian rồi chỉ còn một chu kỳ.

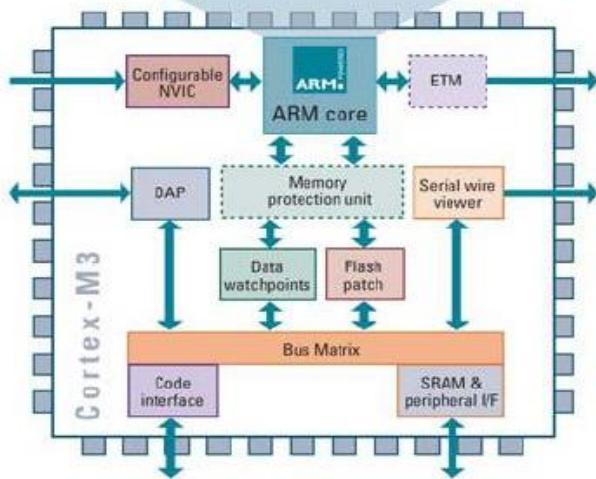
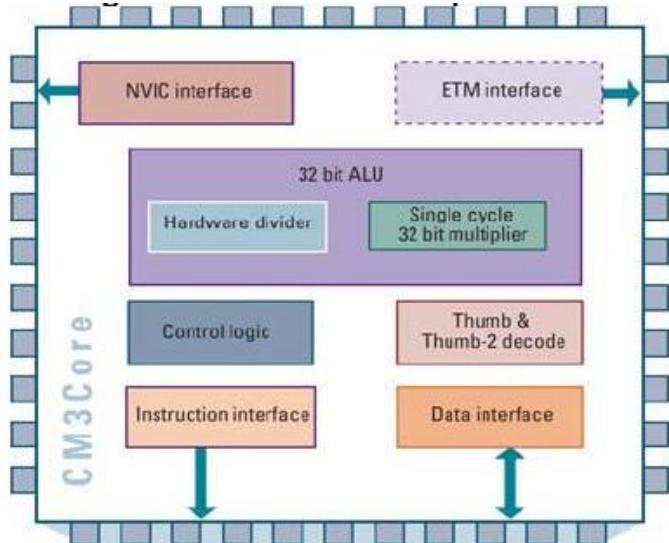
Lõi Cortex-M3 chứa một bộ giải mã cho tập lệnh Thumb truyền thống và Thumb-2 mới, một ALU tiên tiến hỗ trợ nhân

chia phần cứng, điều khiển logic, và các giao tiếp với các thành phần khác của bộ xử lý.

Bộ vi xử lý Cortex-M3 là một bộ vi xử lý 32-bit, với độ rộng của đường dẫn dữ liệu 32 bit, các dải thanh ghi và giao tiếp bộ nhớ. Có 13 thanh ghi đa dụng, hai con trỏ ngăn xếp, một thanh ghi liên kết, một bộ đếm chương trình và một số thanh ghi đặc biệt trong đó có một thanh ghi trạng thái chương trình.

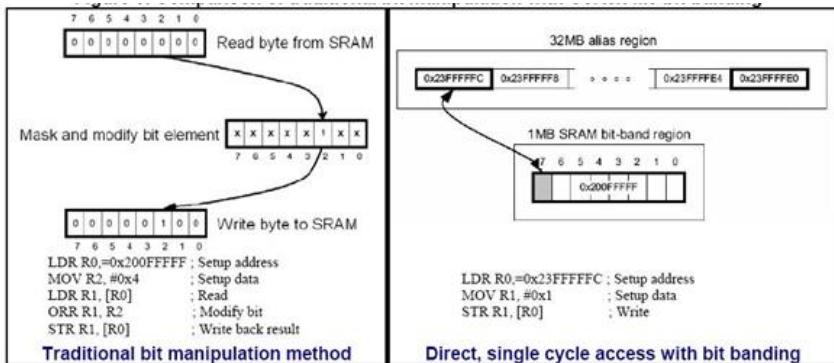
Bộ vi xử lý Cortex-M3 hỗ trợ hai chế độ hoạt động (Thread và Handler) và hai mức truy cập tài nguyên của lõi xử lí (đặc quyền và không đặc quyền), tạo điều kiện cho việc cài đặt các hệ thống mở và phức tạp nhưng vẫn bảo mật. Những dòng mã không đặc quyền bị giới hạn hoặc không cho phép truy cập vào một số tài nguyên quan trọng (một số lệnh đặc biệt và các vùng nhớ nhất định). Chế độ Thread là chế độ hoạt động tiêu biểu hỗ trợ cả mã đặc quyền và không đặc quyền. Bộ vi xử lý sẽ vào chế độ Handler khi một ngoại lệ (exception) xảy ra và tắt cả các mã là đặc quyền trong chế độ này. Ngoài ra, tất cả các hoạt động trong bộ vi xử lý đều thuộc một trong hai trạng thái hoạt động: Thumb cho chế độ thực thi bình thường và Debug cho việc gỡ lỗi.

Bộ vi xử lý Cortex-M3 là một hệ thống ánh xạ bộ nhớ đơn giản, quản lí vùng nhớ cố định lên tới 4 gigabyte với các địa chỉ định nghĩa sẵn, dành riêng cho mã lệnh (vùng mã lệnh), SRAM (vùng nhớ), bộ nhớ/thiết bị bên ngoài, thiết bị ngoại vi bên trong và bên ngoài. Ngoài ra còn có một vùng nhớ đặc biệt dành riêng cho nhà cung cấp.



Hình 9-13 Cortex – M3

Bộ vi xử lý Cortex-M3 cho phép truy cập trực tiếp đến từng bit dữ liệu trong các hệ thống đơn giản bằng cách thực thi một kỹ thuật được gọi là bit-banding (hình 5). Bộ nhớ bao gồm hai vùng bit-band (mỗi vùng 1MB) trong SRAM và vùng bí danh 32MB của vùng không gian ngoại vi (Mỗi byte trong vùng bí danh sẽ tương ứng với một bit trong vùng bit-band). Mỗi hoạt động nạp/lưu tại một địa chỉ trong khu vực bí danh (alias region) sẽ trực tiếp tương ứng với hoạt động trên bit được đại diện bởi bí danh đó. Cụ thể, khi ghi giá trị 0x01 vào một địa chỉ trên vùng bí danh thì có nghĩa là xác định bit tương ứng sẽ có giá trị là 1, tương tự giá trị 0x00 sẽ xác định bit tương ứng có giá trị 0. Còn đọc giá trị tại một địa chỉ vùng bí danh có nghĩa là đọc được giá trị của bit tương ứng. Một vấn đề cần chú ý nữa là hoạt động này mang tính nguyên tử (không chia nhỏ được nữa), không thể bị gián đoạn bởi các hoạt động khác trên bus.



Hình 9-14 Cortex-M3 bit-banding

9.3.4.3 Bộ điều khiển vector ngắt lồng nhau (NVIC)

NVIC (Nested Vectored Interrupt Controller) là thành phần tích hợp của bộ vi xử lý Cortex-M3 có khả năng xử lý ngắt rất linh hoạt và nhanh chóng. Trong cài đặt chuẩn, nó cung cấp một NMI (Non-Maskable Interrupt) và 32 ngắt vật lý đa dụng với 8 mức ưu tiên pre-emption. Nó có thể được cấu hình từ 1 đến 240 ngắt vật lý với tối đa 256 mức độ ưu tiên.

Bộ vi xử lý Cortex-M3 sử dụng một bảng vector có thể tái định vị được, dùng để chứa địa chỉ của hàm xử lý ngắt. Khi nhận một ngắt, bộ xử lý sẽ lấy địa chỉ từ bảng vector thông qua bus chương trình. Bảng vector ngắt được đặt ở địa chỉ 0 khi reset, nhưng có thể được di chuyển đến vị trí khác bằng cách lập trình một thanh ghi điều khiển.

NVIC hỗ trợ ngắt lồng nhau, cho phép một ngắt được xử lý trước một ngắt khác dựa trên mức độ ưu tiên. Nó cũng hỗ trợ cấu hình mức ưu tiên động cho các ngắt. Độ ưu tiên có thể được thay đổi bằng phần mềm trong thời gian chạy (run time). Các ngắt đang được xử lý đều bị khóa cho đến khi hàm xử lý ngắt hoàn thành, do đó, độ ưu tiên của ngắt có thể thay đổi mà không cần lo đến chuyện trùng lắp.

Trong trường hợp các ngắt nối đuôi nhau, các hệ thống cũ sẽ lặp lại hai lần việc lưu trạng thái hoàn thành và khôi phục, dẫn đến độ trễ cao. Bộ vi xử lý Cortex-M3 đơn giản hóa việc chuyển đổi giữa các ngắt đang hoạt động và đang chờ bằng cách cài đặt công nghệ tail-chaining trong phần cứng NVIC. Tail-chaining đạt độ trễ thấp hơn nhiều bằng cách thay thế chuỗi các thao tác pop và push vốn mất hơn 30 chu kỳ xung nhịp bằng một thao tác nạp lệnh đơn giản chỉ mất 6 chu kỳ. Trạng thái bộ vi xử lý được tự động lưu khi ngắt bắt đầu được xử lý và phục hồi ngay khi kết thúc, ít chu kỳ hơn so với việc thực thi bảng phân

mềm, nâng cao hiệu suất đáng kể ở hệ thống hoạt động dưới 100MHz.

9.3.4.4 Đơn vị bảo vệ bộ nhớ (MPU)

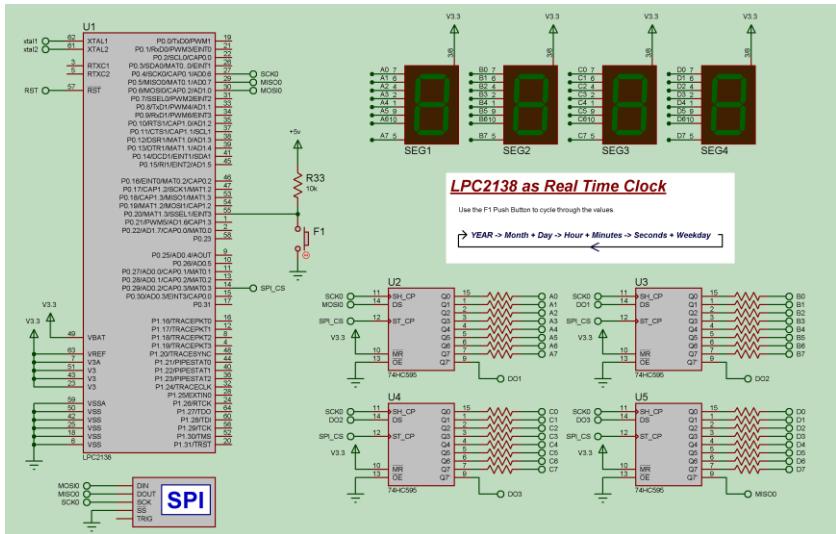
MPU là một thành phần tùy chọn của bộ vi xử lý Cortex-M3, có thể nâng cao độ tin cậy của hệ thống nhúng bằng cách bảo vệ các dữ liệu quan trọng được hệ điều hành sử dụng khỏi các ứng dụng khác, tách biệt độc lập các tác vụ đang thực thi bằng cách không cho phép truy cập vào dữ liệu của nhau, vô hiệu hóa quyền truy cập vào một số vùng nhớ, cho phép các vùng nhớ được định nghĩa là chỉ đọc (read only) và phát hiện các truy cập bộ nhớ có thể phá vỡ hệ thống.

MPU cho phép một ứng dụng được chia nhỏ thành các tiến trình. Mỗi tiến trình sẽ có bộ nhớ (code, dữ liệu, ngăn xếp, heap) và thiết bị riêng, cũng như có quyền truy cập vào bộ nhớ và các thiết bị được chia sẻ. MPU cũng có các quy tắc (rule) truy cập của người dùng và đặc quyền bao gồm việc thực thi mã tại mức đặc quyền thích hợp cũng như quyền sở hữu bộ nhớ và các thiết bị của mã đặc quyền và mã người dùng.

MPU chia bộ nhớ thành các vùng riêng biệt và thực hiện việc bảo vệ bằng cách ngăn các truy cập trái phép. MPU có thể chia bộ nhớ thành tối đa 8 vùng trong đó mỗi vùng có thể được chia thành 8 vùng con. Kích thước vùng có thể bắt đầu từ 32 byte và tăng gấp đôi dần cho đến tối đa 4 gigabyte. Các vùng được đánh số thứ tự bắt đầu từ 0. Có thể xác định một bản đồ bộ nhớ (memory map) nền mặc định để truy cập đặc quyền. Việc truy cập đến các địa chỉ bộ nhớ không được xác định trong vùng MPU hoặc không được phép sẽ tạo ra ngoại lệ lỗi về quản lý bộ nhớ (Memory Management Fault Exception).

9.3.5 Ví dụ ứng dụng ARM

Bài 4: Xây dựng một đồng hồ thời gian thực sử dụng ARM7



Hình 9-15 Sơ đồ đồng hồ thời gian thực sử dụng ARM7

Chương trình mẫu:

```
#include "Target.h"
#include "Config.h"

__irq void IRQ_Handler (void)
{
    void (*interrupt_function)();
    unsigned int vector;

    vector = VICVectAddr; // G
```

```

interrupt_function = (void(*)())vector;
(*interrupt_function)(); // Call vectored interrupt
function.
}
/**Function Name: FIQ_Exception
** Function Desc: Fast Interrupt exception
handler*****/
void FIQ_Exception(void)
{
    while(1); // You should replace the code here
if you are using FIQ
}

/**Function Name: TargetInit
** Function Desc: Target board initialisation
routine*****/
void TargetInit(void)
{
    /* Add your code here */
}

/** Function Name: TargetResetInit
** Function Desc: Initialisation after reset
*****/
void TargetResetInit(void)
{
#ifndef NDEBUG
    MEMMAP = 0x2; //remap
#else
    MEMMAP = 0x1; //remap
#endif
}

```

```

/* Set up System clock */
PLLCON = 1;
#if(Fpclk / (Fcclk / 4)) == 1
    VPBDIV = 0;
#endif
#if(Fpclk / (Fcclk / 4)) == 2
    VPBDIV = 2;
#endif
#if(Fpclk / (Fcclk / 4)) == 4
    VPBDIV = 1;
#endif

#if(Fcco / Fcclk) == 2
    PLLCFG = ((Fcclk / Fosc) - 1) / (0 << 5);
#endif
#if(Fcco / Fcclk) == 4
    PLLCFG = ((Fcclk / Fosc) - 1) / (1 << 5);
#endif
#if(Fcco / Fcclk) == 8
    PLLCFG = ((Fcclk / Fosc) - 1) / (2 << 5);
#endif
#if(Fcco / Fcclk) == 16
    PLLCFG = ((Fcclk / Fosc) - 1) / (3 << 5);
#endif

    PLLFEED = 0xaa;
    PLLFEED = 0x55;
    while((PLLSTAT & (1 << 10)) == 0);
    PLLCON = 3;
    PLLFEED = 0xaa;
    PLLFEED = 0x55;

/* Set up memory accelerate module */

```

```

MAMCR = 0;
#if Fcclk < 20000000
    MAMTIM = 1;
#else
#if Fcclk < 40000000
    MAMTIM = 2;
#else
    MAMTIM = 3;
#endif
#endif
MAMCR = 2;

/* Init VIC */
VICIntEnClear = 0xffffffff;
VICVectAddr = 0;
VICIntSelect = 0;

/* Function Name: Delayms()
 * Function Desc: software delay
 */
void Delayms(uint32 dly)
{
    uint32 i;

    for(; dly>0; dly--)
        for(i=0; i<2000; i++);
}

```

9.4 Câu hỏi và bài tập

1. Hãy kể tên một vài công ty sản xuất các loại vi điều khiển hiện đại thông dụng ngày nay mà bạn biết? Và những công ty đó sản xuất loại gì?

2. Vi điều khiển ARM có đặc điểm gì nổi bật so với những vi điều khiển khác?
3. Nêu ưu điểm của vi điều khiển PIC? Vi điều khiển PIC thường được sử dụng cho mục đích gì? Vì sao?
4. Vi điều khiển AVR thường được sử dụng trong các loại ứng dụng nào?
5. Viết chương trình giao tiếp giữa 1 vi điều khiển AVR với máy tính thông qua cổng COM.
6. Viết chương trình trên vi điều khiển AVR cho đồng hồ điện tử sử dụng IC thời gian thực DS1307.
7. Viết chương trình đọc giá trị của cảm biến nhiệt độ bằng ADC với vi điều khiển PIC và so sánh tính ổn định với bài tập sử dụng vi điều khiển 8051.

TÀI LIỆU THAM KHẢO

1. Intel. MCS®51 Microcontroller family user's manual. 1994.
2. Kenneth Ayala. 8051 Microcontroller: Architecture, Programming and Applications Second Edition. Nhà xuất bản Delmar Learning, 1996.
3. Muhammad Ali Mazidi, Janice Gillispie Mazidi and Rolin D. McKinlay. The 8051 Microcontroller and Embedded Systems 2nd Edition. Nhà xuất bản Prentice-Hall, 2005.
4. Subrata Ghosal. 8051 Microcontroller: Internals, Instructions, Programming and Interfacing. Nhà xuất bản Pearson. 2010.
5. Tống Văn On. Thiết kế hệ thống với họ 8051. Nhà xuất bản Phương Đông, 2006.
6. Nguyễn Tăng Cường. Cấu trúc và lập trình họ vi điều khiển 8051. Nhà xuất bản Khoa học và kỹ thuật, năm 2004.