HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA KỸ THUẬT ĐIỆN TỬ

*****

GIÁO TRÌNH BÀI GIẢNG

(Phương pháp đào tạo theo tín chỉ)

TÊN HỌC PHẦN: KỸ THUẬT VI XỬ LÝ

Mã học phần: ELE1317

(03 tín chỉ)

Biên soạn

TS. VŨ HỮU TIẾN

LƯU HÀNH NỘI BỘ

Hà Nội, 11/2014

LỜI NÓI ĐẦU

Ngày nay, kỹ thuật điện tử đã có những tiến bộ vượt bậc, đặc biệt trong kỹ thuật chế tạo mạch vi điện tử. Sự phát triển nhanh chóng của kỹ thuật vi điện tử mà đặc trưng là kỹ thuật vi xử lý đã tạo ra bước ngoặt quan trọng trong sự phát triển của khoa học tính toán và xử lý thông tin. Nhờ đó mà hiện nay vi xử lý được sử dụng rộng rãi không chỉ trong lĩnh vực máy tính điện tử mà còn trong rất nhiều hệ thống điều khiển khác.

Để đáp ứng nhu cầu tìm hiểu về kỹ thuật vi xử lý và lập trình cho vi xử lý, bài giảng "Kỹ thuật vi xử lý" được biên soạn nhằm giới thiệu các khái niệm cơ bản về vi xử lý nói chung, bộ vi xử lý ARM và cách lập trình hợp ngữ ARM nói riêng. Bên cạnh đó, bài giảng cũng giới thiệu về bộ vi điều khiển 8051. Đây là một trong những bộ vi điều khiển đơn giản và cơ bản, giúp sinh viên có được những kiến thức nền tảng để tiếp cận các bộ vi điều khiển phức tạp hơn.

Với các nội dung trên, bài giảng được chia làm 6 chương như sau:

Chương 1. Tổng quan về vi xử lý

Chương 2. Bộ vi xử lý ARM

Chương 3. Lập trình hợp ngữ ARM

Chương 4. Vi điều khiển 8051

Chương 5. Bộ đếm/định thời và UART trong 8051

Chương 6. Lập trình ngắt trong 8051

Trong quá trình biên soạn không tránh khỏi sơ suất, tác giả rất mong nhận được các ý kiến đóng góp của độc giả để hoàn thiện nội dung bài giảng.

Xin trân trọng cảm ơn!

Chủ biên

TS. Vũ Hữu Tiến

MỤC LỤC

DANH SÁCH HÌNH VẼ

DANH SÁCH CÁC BẢNG

CHƯƠNG 1. TỔNG QUAN VỀ VI XỬ LÝ

1.1. Giới thiệu chung

Một máy tính thông thường bao gồm các khối chức năng cơ bản như: khối xử lý trung tâm CPU (Central Processing Unit), bộ nhớ, và khối phối ghép với thiết bị ngoại vi (I/O - Input/output). Tùy theo quy mô, độ phức tạp hiệu năng của các khối chức năng kể trên mà người ta phân các máy tính điện tử đã và đang sử dụng ra thành các loại sau:

Máy tính lớn (Mainframe) là loại máy tính được thiết kế để giải các bài toán lớn với tốc độ nhanh. Máy tính này thường làm việc với số liệu từ 64 bit hoặc lớn hơn nữa và được trang bị nhiều bộ xử lý tốc độ cao và bộ nhớ rất lớn. Chính vì vậy máy tính cũng lớn về kích thước vật lý. Chúng thường được dùng để tính toán điều khiển các hệ thống thiết bị dùng trong quân sự hoặc các hệ thống máy móc của chương trình nghiên cứu vũ trụ, để xử lý các thông tin trong ngành ngân hàng, ngành khí tượng, các công ty bảo hiểm,... Loại máy lớn nhất trong các máy lớn được gọi là Supercomputer (như loại máy Y-MP/832 của Cray).

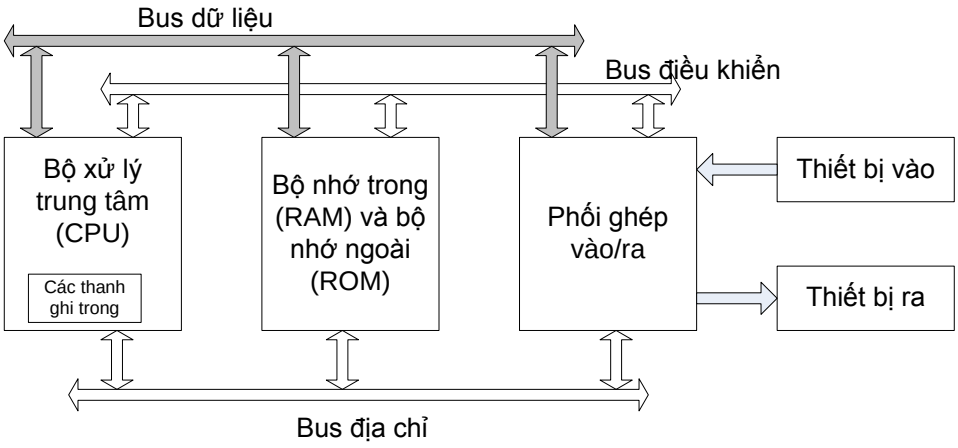
Máy tính con (Minicomputer) là một dạng thu nhỏ về kích thước cũng như về tính năng của máy tính tính lớn. Nó ra đời nhằm thỏa mãn các nhu cầu sử dụng máy tính cho các ứng dụng vừa phải mà nếu dùng máy tính lớn vào đó thì sẽ gây lãng phí. Máy tính con thường được dùng cho các tính toán khoa học kỹ thuật, gia công dữ liệu quy mô nhỏ hay để điều khiển quy trình công nghệ. Tiêu biểu cho nhóm này là loại máy VAX 6360 của Digital Equipment Corporation và MV/8000II của Data General.

Máy vi tính (Microcomputer) là loại máy tính rất thông dụng hiện nay. Một máy vi tính có thể là một bộ vi điều khiển (microcontroller), một máy vi tính trong vi mạch (one-chip microcomputer), và một hệ vi xử lý có khả năng làm việc với số liệu có độ dài 1 bit, 4 bit, 8 bit, 16 bit hoặc lớn hơn. Hiện nay một số máy vi tính có tính năng có thể so sánh được với máy tính con, làm việc với số liệu có độ dài từ là 32 bit (thậm chí là 64 bit). Ranh giới để phân chia giữa máy vi tính và máy tính con chính vì thế ngày càng không rõ nét.

Các bộ vi xử lý hiện có trên thị trường thường được xếp theo các họ phụ thuộc vào các nhà sản xuất và chúng rất đa dạng về chủng loại. Các nhà sản xuất vi xử lý nổi tiếng có thể kể tới Intel với các sản phẩm x86, Motorola với 680xx, Sun Microsystems với SPARC.

1.2. Hệ vi xử lý

Bộ vi xử lý là một thành phần rất cơ bản, không thể thiếu được để tạo nên máy vi tính. Trong thực tế, bộ vi xử lý có thể kết hợp thêm với các bộ phận điện tử khác nhau như bộ nhớ và bộ phối ghép vào/ra để tạo nên một hệ vi xử lý hoàn chỉnh. Cần lưu ý rằng, để chỉ một hệ thống có cấu trúc như trên, thuật ngữ "hệ vi xử lý" mang ý nghĩa tổng quát hơn so với thuật ngữ "máy vi tính", vì máy vi tính chỉ là một ứng dụng cụ thể của hệ vi xử lý. Hình 1.1 giới thiệu sơ đồ khối tổng quát của một hệ vi xử lý.



Hình 1. 1 Sơ đồ khối của hệ vi xử lý

Trong sơ đồ này ta thấy rõ các khối chức năng chính của hệ vi xử lý bao gồm:

- Khối xử lý trung tâm (Central Processing Unit, CPU)
- Bộ nhớ bán dẫn (ROM - RAM)
- Khối phối ghep với các thiết bị ngoại vi (Input/Output - I/O)
- Các bus truyền thông tin

Ba khối chức năng đầu tiên liên hệ với nhau thông qua các đường truyền tín hiệu gọi chung là Bus hệ thống. Bus hệ thống bao gồm 3 bus thành phần bus địa chỉ, bus dữ liệu và bus điều khiển được sử dụng để truyền các tín hiệu địa chỉ, dữ liệu và điều khiển tương ứng.

CPU đóng vai trò chủ đạo trong hệ vi xử lý. Đây là một mạch vi điện tử có độ tích hợp rất cao. Khi hoạt động, CPU đọc mã lệnh được ghi dưới dạng các bit 0 và bit 1 từ bộ nhớ, sau đó sẽ giải mã các lệnh này thành các dãy xung điều khiển ứng với các thao tác trong lệnh để điều khiển các khối khác thực hiện từng bước các thao tác đó. Để làm được việc này bên trong CPU có các thanh ghi dùng để chứa địa chỉ của lệnh sắp thực hiện gọi là thanh ghi con trỏ lệnh (Instruction Pointer, IP) hoặc bộ đếm chương trình (Program Counter, PC), một số thanh ghi đa năng khác cùng bộ tính toán số học và logic (Arithmetic Logic Unit ALU) để thao tác với dữ liệu. Ngoài ra ở đây còn có các hệ thống mạch điện tử rất phức tạp để giải mã lệnh và từ đó tạo ra các xung điều khiển cho toàn bộ hệ thống.

Bộ nhớ bán dẫn hay còn gọi là bộ nhớ trong là một bộ phận khác rất quan trọng của hệ vi xử lý. Tại đây (trong ROM) ta có thể chứa chương trình điều khiển hoạt động của toàn hệ để khi bật điện thì CPU có thể lấy lệnh từ đây để khởi động hệ thống. Một phần của chương trình điều khiển hệ thống, các chương trình ứng dụng, dữ liệu cùng các kế quả của chương trình thường được đặt trong RAM. Các dữ liệu và chương trình muốn lưu trữ lâu dài hoặc có dung lượng lớn sẽ được đặt trong bộ nhớ ngoài.

Khối phối ghép vào/ra (I/O) tạo ra khả năng giao tiếp giữa hệ vi xử lý với thế giới bên ngoài. Các thiết bị ngoại vi như bàn phím, chuột, màn hình, máy in, chuyển đổi số/tương tự (D/A

Converter, DAC) và chuyển đổi tương tự/số (A/D Converter, ADC), ổ đĩa từ,... đều liên hệ với bộ vi xử lý qua bộ phận này. Bộ phận phối ghép cụ thể giữa bus hệ thống với thế giới bên ngoài thường được gọi là cổng. Như vậy ta sẽ có các cổng vào để lấy thông tin từ ngoài vào và các cổng ra để đưa thông tin từ trong ra. Tùy theo nhu cầu cụ thể của công việc, các mạch cổng này có thể được xây dựng từ các mạch logic đơn giản hoặc từ các vi mạch chuyên dụng lập trình được.

Bus địa chỉ (address bus) thường có từ 16, 20, 24, 32 hoặc 64 đường dây song song truyền tải thông tin của các bit địa chỉ. Khi đọc/ghi bộ nhớ CPU sẽ đưa ra trên bus này địa chỉ của ô nhớ liên quan. Khả năng phân biệt địa chỉ (số lượng địa chỉ cho ô nhớ mà CPU có thể quản lý được) phụ thuộc vào số bit của bus địa chỉ. Ví dụ nếu một CPU có số đường dây địa chỉ là $N = 16$ bit thì nó có khả năng địa chỉ hóa được 2^N = 65536 = 64 Kilobit ô nhớ khác nhau. Khi đọc/ghi với cổng vào/ra CPU cũng đưa ra trên bus địa chỉ các bit địa chỉ tương ứng của cổng. Trên sơ đồ khối ta dễ nhận ra tính một chiều của bus địa chỉ qua một chiều của mũi tên. Chỉ có CPU mới có khả năng đưa ra địa chỉ trên bus địa chỉ.

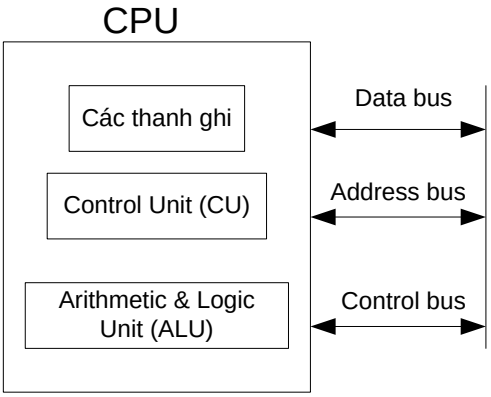
Bus dữ liệu (data bus) thường có từ 8, 16, 20, 24, 32, 64 (hoặc hơn) đường dây tùy theo các bộ vi xử lý cụ thể. Số lượng đường dây này quyết định số bit dữ liệu mà CPU có khả năng xử lý cùng một lúc. Chiều mũi tên trên bus số liệu chỉ ra rằng đây là bus 2 chiều, nghĩa là dữ liệu có thể truyền đi từ CPU (dữ liệu ra) hoặc truyền đến CPU (dữ liệu vào). Các phần tử có đầu ra nối thẳng với bus dữ liệu đều phải được trang bị đầu ra 3 trạng thái để có thể ghép vào được và hoạt động bình thường với bus này.

Bus điều khiển (control bus) thường gồm hàng chục đường dây tín hiệu khác nhau. Mỗi tín hiệu điều khiển có một chiều nhất định vì khi hoạt động CPU đưa tín hiệu điều khiển tới các khối khác trong hệ. Đồng thời CPU cũng nhận tín hiệu điều khiển từ các khối đó để phối hợp hoạt động của toàn hệ. Các tín hiệu này trên hình vẽ được thể hiện bởi các đường có mũi tên 2 chiều, điều đó không phải là để tính hai chiều của một tín hiệu mà là tính hai chiều của cả một nhóm các tín hiệu.

Mặt khác, hoạt động của hệ thống vi xử lý trên cũng có thể coi như quá trình trao đổi dữ liệu giữa các thanh ghi bên trong. Về mặt chức năng mỗi khối trong hệ thống trên tương đương với các thanh ghi trong (nằm trong CPU) hoặc các thanh ghi ngoài (nằm rải rác trong bộ nhớ ROM, bộ nhớ RAM và trong khối phối ghép vào/ra). Hoạt động của toàn hệ thực chất là sự phối hợp hoạt động của các thanh ghi trong và ngoài nói trên để thực hiện sự biến đổi dữ liệu hoặc sự trao đổi dữ liệu theo các yêu cầu đã định trước.

1.3. Các thành phần của CPU

Như đã trình bày trong phần trên, vi xử lý chính là đơn vị xử lý trung tâm CPU của máy vi tính. Như vậy sức mạnh xử lý của máy vi tính được quyết định bởi năng lực của vi xử lý. Trên nguyên tắc, vi xử lý có thể được chia thành các đơn vị chức năng chính như trong hình 1.2.



Hình 1. 2 Cấu trúc tổng quát của CPU

Các thanh ghi

Số lượng, kích cỡ và kiểu của các thanh ghi thay đổi từ vị xử lý này sang vi xử lý khác. Tuy nhiên, các thanh ghi này thực hiện các thao tác tương tự nhau. Cấu trúc các thanh ghi đóng vai trò quan trọng trong việc thiết kế kiến của vi xử lý. Đồng thời, cấu trúc thanh ghi với một loại vi xử lý cụ thể cho biết mức độ thuận lợi và dễ dùng khi lập trình cho vi xử lý đó. Dưới dây là các thanh ghi cơ bản nhất:

- Thanh ghi lệnh: lưu các lệnh. Sau khi nạp mã lệnh từ bộ nhớ, vi xử lý lưu mã lệnh trong thanh ghi lệnh. Giá trị trong thanh ghi này luôn được vi xử lý giải mã để xác định lệnh. Kích cỡ từ (word) của vi xử lý quyết định kích cỡ của thanh ghi này. Ví dụ, vi xử lý 32 bit sẽ có thanh ghi lệnh 32 bit.
- Bộ đếm chương trình: chứa địa chỉ của lệnh hay mã thực thi (op-code). Thông thường, thanh ghi này chứa địa chỉ của câu lệnh kế. Thanh ghi này có đặc điểm sau:
  - Khi khởi động lại, địa chỉ của lệnh đầu tiên được thực hiện được nạp vào thanh ghi này.
  - Để thực hiện lệnh, vi xử lý nạp vào nội dung của bộ đếm chương trình vào bus địa chỉ và đọc ô nhớ ở địa chỉ đó. Giá trị của bộ đếm chương trình tự động tăng theo bộ logic trong của vi xử lý. Như vậy, vi xử lý thực hiện các lệnh tuần tự trừ phi chương trình có các lệnh làm thay đổi trật tự tính toán.
  - Kích cỡ của bộ đếm chương trình phụ thuộc vào kích cỡ của bus địa chỉ.
  - Nhiều lệnh làm thay đổi nội dung của thanh ghi này so với trình tự thông thường. Khi đó, giá trị của thanh ghi được xác định thông qua địa chỉ chỉ định trong các lệnh này.
- Thanh ghi địa chỉ bộ nhớ: chứa địa chỉ của dữ liệu. Vi xử lý sử dụng các địa chỉ này như là các con trỏ trực tiếp tới bộ nhớ. Giá trị của các địa chỉ này chính là dữ liệu đang được trao đổi và xử lý.
- Thanh ghi dùng chung: còn được gọi là thanh ghi tích lũy (accumulator). Thanh ghi này thường là các thanh ghi 8 bit dùng để lưu các kết quả tính toán của đơn vị xử lý số học và logic ALU. Thanh ghi này còn dùng để trao đổi dữ liệu với các thiết bị vào/ra.

Đơn vị xử lý số học và logic ALU

ALU thực hiện tất cả các thao tác xử lý dữ liệu bên trong vi xử lý như là các phép toán logic, số học. Kích cỡ thanh ghi ALU tương ứng với kích cỡ từ của vi xử lý. Vi xử lý 32 bit sẽ có ALU 32 bit. Một vài chức năng tiêu biểu của ALU:

- Cộng nhị phân và các phép logic
- Tính số bù một của dữ liệu
- Dịch hoặc quay trái phải các thanh ghi dùng chung.

Đơn vị điều khiển CU

Chức năng chính của đơn vị điều khiển CU là đọc và giải mã các lệnh từ bộ nhớ chương trình. Để thực hiện lệnh, CU kích hoạt khối phù hợp trong ALU căn cứ vào mã lệnh (op-code) trong thanh ghi lệnh. Mã lệnh xác định thao tác để CU thực thi. CU thông dịch nội dung của thanh ghi lệnh và sau đó sinh ra một chuỗi các tín hiệu kích hoạt tương ứng với lệnh nhận được. Các tín hiệu này kích hoạt các khối chức năng phù hợp bên trong ALU.

CU sinh ra các tín hiệu điều khiển dẫn tới các thành phần khác của vi xử lý qua bus điều khiển. Ngoài ra, CU cũng đáp ứng lại các tín hiệu điều khiển trên bus điều khiển do các bộ phận khác gửi tới. Các tín hiệu này thay đổi theo từng loại vi xử lý. Một số tín hiệu điều khiển tiêu biểu như khởi động lại RESET, đọc ghi (R/W), tín hiệu ngắt (INT/IRQ),...

1.4. Kiến trúc CISC và RISC

Có hai kiểu kiến trúc vi xử lý: máy tính với tập lệnh rút gọn (Reduced Instruction Set Computer - RISC) và máy tính với tập lệnh phức tạp (Complex Instruction Set Computer - CISC). Vi xử lý RISC nhấn mạnh tính đơn giản và hiệu quả. Các thiết kế RISC khởi đầu với tập lệnh thiết yếu và vừa đủ. RISC tăng tốc độ xử lý bằng cách giảm số chu kỳ đồng hồ trên một lệnh. Mục đích của RISC là tăng tốc độ hiệu dụng bằng cách chuyển việc thực hiện các thao tác không thường xuyên vào phần mềm còn các thao tác phổ biến do phần cứng thực hiện. Như vậy làm tăng hiệu năng của máy tính. Các đặc trưng căn bản của vi xử lý kiểu RISC:

- Thiết kế vi xử lý RISC sử dụng điều khiển cứng (hardwired control) không hoặc rất ít sử dụng vi mã. Tất cả các lệnh RISC có định dạng cố định vì vậy việc sử dụng vi mã không cần thiết.
- Vi xử lý RISC xử lý hầu hết các lệnh trong một chu kỳ.
- Tập lệnh của vi xử lý RISC chủ yếu sử dụng các lệnh với thanh ghi, nạp và lưu. Tất cả các lệnh số học và logic sử dụng thanh ghi, còn các lệnh nạp và lưu dùng để truy nhập bộ nhớ.
- Các lệnh có một định dạng cố định và ít chế độ địa chỉ.
- Vi xử lý RISC có một số thanh ghi dùng chung.
- Vi xử lý RISC xử lý một vài lệnh đồng thời và thường áp dụng kỹ thuật đường ống (pipeline).

Vi xử lý RISC thường phù hợp với các ứng dụng nhúng. Vi xử lý hay bộ điều khiển nhúng thường được nhúng trong hệ thống chủ. Nghĩa là, các thao tác của các bộ điều khiển này thường được che dấu khỏi hệ thống chủ. Ứng dụng điều khiển tiêu biểu cho ứng dụng nhúng là hệ thống tự động hóa văn phòng như máy in laser, máy photocopy,... Vi xử lý RISC cũng rất phù hợp với các ứng dụng như xử lý ảnh, robot và đồ họa nhờ mức tiêu thụ điện thấp, thực thi nhanh chóng.

Khác với RISC, vi xử lý CISC bao gồm số lượng lớn các lệnh và nhiều chế độ địa chỉ mà nhiều kiểu rất ít được sử dụng. Với CISC hầu hết các lệnh đều có thể truy nhập bộ nhớ trong khi đó RISC chỉ có các lệnh nạp và lưu. Do tập lệnh phức tạp, CISC cần đơn vị điều khiển phức tạp và vi chương trình. Trong khi đó, RISC sử dụng bộ điều khiển kết nối cứng nên nhanh hơn. Kiến trúc CISC khó triển khai kỹ thuật đường ống.

Ưu điểm của CISC là các chương trình phức tạp có thể chỉ cần vài lệnh với vài chu trình nạp còn RISC cần một số lượng lớn các lệnh để thực hiện cùng nhiệm vụ. Tuy nhiên, RISC có thể cải thiện hiệu năng đáng kể nhờ xung nhịp nhanh hơn, kỹ thuật đường ống và tối ưu hóa quá trình biên dịch. Hiện nay, các vi xử lý CISC sử dụng phương pháp lai, với các lệnh đơn giản CISC sử dụng cách tiếp cận của RISC để thực thi xen kẽ (kỹ thuật đường ống) với các câu lệnh phức tạp sử dụng các vi chương trình để đảm bảo tính tương thích.

1.5. Tóm tắt lịch sử của vi xử lý

- **Thế hệ 1 (1971 – 1973):** Vi xử lý 4 bit
  - Độ dài từ thường là 4 bit.
  - Tốc độ thực hiện lệnh: 10 – 60 μs/lệnh với tần số đồng hồ f_{clk} = 0.1 – 0.8 MHz.
  - Tập lệnh đơn giản và phải cần nhiều vi mạch phụ trợ mới tạo nên một hệ xử lý hoàn chỉnh.
- **Thế hệ 2 (1974 – 1977):** Vi xử lý 8 bit
  - Tập lệnh phong phú hơn.
  - Có thể quản lý bộ nhớ 64KB. Một số bộ vi xử lý có thể phân biệt 256 địa chỉ thiết bị ngoại vi.
  - Sử dụng công nghệ NMOS hay CMOS.
  - Tốc độ thực hiện lệnh: 1 – 8 μs/lệnh với tần số đồng hồ f_{clk} = 1 – 5 MHz.
- **Thế hệ 3 (1978 – 1982):** Vi xử lý 16 bit
  - Tập lệnh đa dạng với các lệnh nhân, chia và xử lý chuỗi.
  - Có thể quản lý bộ nhớ lên tới 16MB. Một số bộ vi xử lý có thể phân biệt 64KB địa chỉ thiết bị ngoại vi.
  - Sử dụng công nghệ HMOS.
  - Tốc độ thực hiện lệnh: 0.1 – 1 μs/lệnh với tần số đồng hồ f_{clk} = 5 – 10 MHz.
- **Thế hệ 4:** Vi xử lý 32bit
  - Bus địa chỉ 32 bit, phân biệt 4GB bộ nhớ và có thể dùng thêm các bộ đồng xử lý
  - Có khả năng làm việc với bộ nhớ ảo.
  - Có các cơ chế pipeline, bộ nhớ cache.
  - Sử dụng công nghệ HCMOS.

- **Thế hệ 5:** Vi xử lý 64 bit
  - Bus địa chỉ và thanh ghi có độ dài 64 bit
  - Thế hệ vi xử lý 64 bit được ARM giới thiệu với tên gọi ARMv8 từ năm 2011. Hãng Intel và AMD giới thiệu vi xử lý 64 bit với tên gọi AMD64 và EMT64.

CHƯƠNG 2. BỘ VI XỬ LÝ ARM

2.1. Tổng quan về vi xử lý ARM

2.1.1. Lịch sử phát triển của vi xử lý ARM

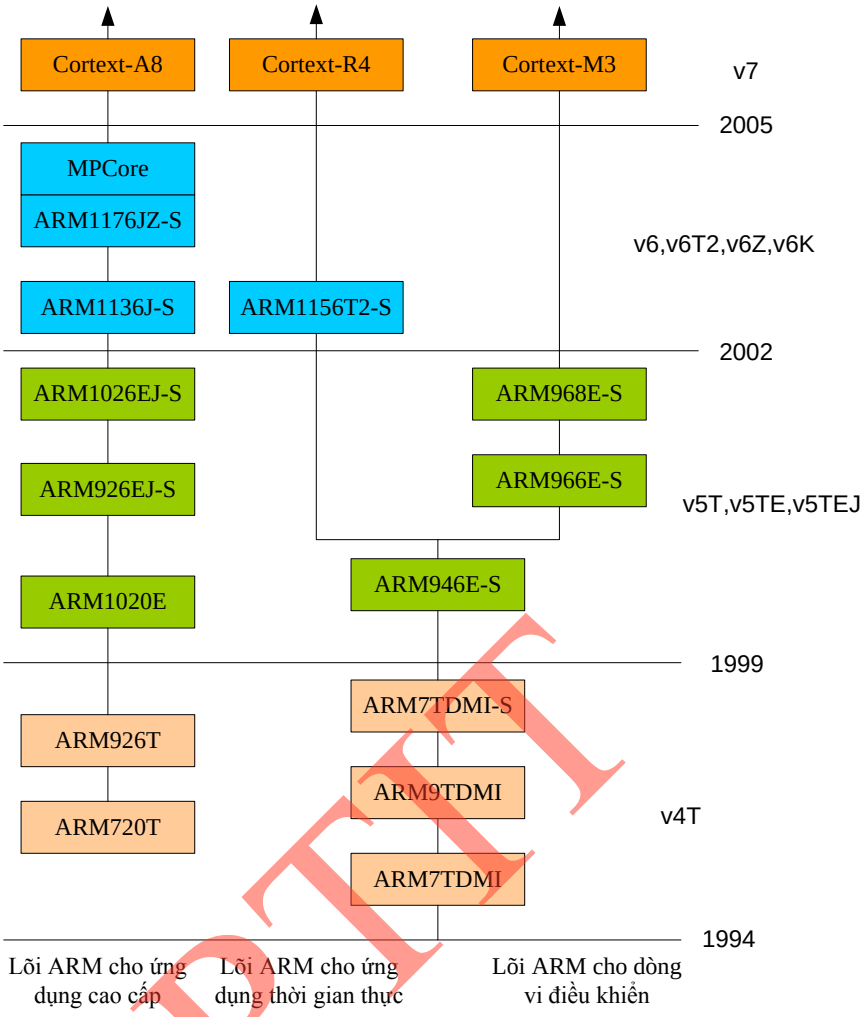
Việc thiết kế ARM được bắt đầu từ năm 1983 trong một dự án phát triển của công ty máy tính Acorn. Nhóm thiết kế, dẫn đầu bởi Roger Wilson và Steve Furber, bắt đầu phát triển một bộ vi xử lý có nhiều điểm tương đồng với Kỹ thuật MOS 6502 tiên tiến. Acorn đã từng sản xuất nhiều máy tính dựa trên 6502, vì vậy việc tạo ra một chip như vậy là một bước tiến đáng kể của công ty này.

Nhóm thiết kế hoàn thành việc phát triển mẫu gọi là ARM1 vào năm 1985, và vào năm sau, nhóm hoàn thành sản phẩm gọi là ARM2. ARM2 có bus dữ liệu 32-bit, không gian địa chỉ 26-bit tức cho phép quản lý đến 64 Mbyte địa chỉ và 16 thanh ghi 32-bit. Một trong những thanh ghi này đóng vai trò là bộ đếm chương trình với 6 bit cao nhất và 2 bit thấp nhất lưu giữ các cờ trạng thái của bộ vi xử lý. Có thể nói ARM2 là bộ vi xử lý 32-bit khả dụng đơn giản nhất trên thế giới, với chỉ gồm 30.000 transistor (so với bộ vi xử lý lâu hơn bốn năm của Motorola là 68000 với khoảng 68.000 transistor). Sự đơn giản như vậy có được nhờ ARM không có vi chương trình và cũng giống như hầu hết các CPU vào thời đó, không hề chứa cache. Sự đơn giản này đưa đến đặc điểm tiêu thụ công suất thấp của ARM, mà lại có tính năng tốt hơn cả 286. Thế hệ sau, ARM3, được tạo ra với 4KB cache và có chức năng được cải thiện tốt hơn nữa.

Vào những năm cuối thập niên 80, hãng máy tính Apple Computer bắt đầu hợp tác với Acorn để phát triển các thế hệ lõi ARM mới và ngay sau đó Acorn nâng cấp nhóm thiết kế trở thành một công ty mới gọi là Advanced RISC Machines. Vì lý do đó ARM được cho là chữ viết tắt của Advanced RISC Machines thay vì Acorn RISC Machine. Kết quả sự hợp tác này là ARM6. Mẫu đầu tiên được công bố vào năm 1991 và Apple đã sử dụng bộ vi xử lý ARM 610 dựa trên ARM6 làm cơ sở cho PDA hiệu Apple Newton. Vào năm 1994, Acorn dùng ARM 610 làm CPU trong các máy vi tính RiscPC của họ.

Trải qua nhiều thế hệ nhưng lõi ARM gần như không thay đổi kích thước. ARM2 có 30.000 transistors trong khi ARM6 chỉ tăng lên đến 35.000. Ý tưởng của nhà sản xuất lõi ARM là sao cho người sử dụng có thể ghép lõi ARM với một số bộ phận tùy chọn nào đó để tạo ra một CPU hoàn chỉnh, một loại CPU mà có thể tạo ra trên những nhà máy sản xuất bán dẫn cũ và vẫn tiếp tục tạo ra được sản phẩm với nhiều tính năng mà giá thành vẫn thấp.

Thế hệ thành công nhất có lẽ là ARM7TDMI với hàng trăm triệu lõi được sử dụng trong các máy điện thoại di động, hệ thống video game cầm tay, và Sega Dreamcast. Trong khi công ty ARM chỉ tập trung vào việc bán lõi IP, cũng có một số giấy phép tạo ra bộ vi điều khiển dựa trên lõi này.



Hình 2. 1 Các mốc lịch sử ra đời các vi xử lý ARM

Hiện nay, vi xử lý ARM mới nhất có tên gọi ARM Cortex. Lõi ARM Cortex được chia ra thành ba dòng cấu hình chính, với các ký hiệu viết tắt lần lượt là: A, R, M. Chữ A là viết tắt của Application, lõi ARM dòng này hỗ trợ cho các ứng dụng có độ phức tạp cao như: máy tính, điện thoại di động v.v... R là viết tắt của Realtime, các ứng dụng cần tính toán xử lý thời gian thực được hỗ trợ bởi cấu hình này. M là Microcontroller, dành cho các ứng dụng công nghiệp và điện tử tiêu dùng. Hình 2.1 cho thấy các kiến trúc sau v5, v6, v7 đều kế thừa từ v4T.

2.1.2. Đặc điểm của kiến trúc VT4

Điểm mới của kiến trúc v4T là hỗ trợ tập lệnh Thumb (viết tắt là T trong các ký hiệu của bộ xử lý). Hỗ trợ cùng lúc tập lệnh Thumb 16 bit và ARM 32 bit. Với tập lệnh Thumb 16 bit cho phép trình biên dịch tạo ra chương trình nhỏ hơn (tiết kiệm khoảng 35% so với khi biên dịch ở tập lệnh ARM 32 bit) mà vẫn tương thích với hệ thống 32 bit. Điển hình ở kiến trúc này là lõi ARM7TDMI được thiết kế nhằm đáp ứng các ứng dụng yêu cầu hiệu suất cao, tiêu thụ năng lượng thấp và nhỏ gọn. ARM7TDMI được cấu thành bởi các từ viết tắt: ARM7, T, D, M và I. T

có nghĩa là hỗ trợ tập lệnh Thumb 16 bit. D có nghĩa là Debug, ARM7TDMI hỗ trợ giải mã lỗi bằng khối Embedded Trace Macrocell (ETM) đây là giải pháp giải mã lỗi hoàn chỉnh dành cho lõi ARM. M có nghĩa là "Long Multiply Support" - hỗ trợ phép toán 64 bit, ngoài ra ARM7TDMI có khả năng cộng tác với các nhân khác nhằm tăng cường khả năng xử lý (coprocessor). I là viết tắt của Interface, hỗ trợ giao diện ngoại vi. ARM7TDMI có cấu trúc đường ống 3 tầng và là kiến trúc Von Neumann, bộ xử lý số học 32 bit. Hệ thống tập lệnh 16/32 bit có khả năng mở rộng thông qua giao diện đồng xử lý với nhân ngoài.

Ở phiên bản mở rộng ARM720T, bộ nhớ cache và hệ thống quản lý bộ nhớ (Memory Management Unit) được tích hợp. Tiếp đó phiên bản ARM9TDMI sử dụng cấu trúc đường ống 5 tầng và kiến trúc Harvard.

2.1.3. Đặc điểm của kiến trúc V5

Ở phiên bản v5TE, hỗ trợ khối xử lý tín hiệu số DSP. Với khối DSP này, năng lực xử lý tính toán số được tăng lên 70%. Ở phiên bản v5TE-J, khối Jazelle được thêm vào nhằm hỗ trợ trình thông dịch mã Java và máy ảo Java (Java Virtual Machine). Thời gian thực thi mã Java được tăng lên 8 lần và giảm được hơn 80% năng lượng tiêu thụ so với nhân không hỗ trợ khối Jazelle. Tính năng này cho phép lập trình viên thực thi mã Java một cách độc lập với hệ điều hành. Kiến trúc v5 được sử dụng rất nhiều ở dòng ARM10 và đặc biệt là phiên bản v5TE-J. Mặc dù không có nhiều thay đổi về kiến trúc tuy nhiên phiên bản v5 được sử dụng rất nhiều bởi vi xử lý tích hợp hệ thống (System on Chip). Tính năng cao cùng với sự linh hoạt trong giấy phép sử dụng bản quyền chính là lý do các nhà sản xuất lựa chọn lõi ARM để phát triển sản phẩm.

2.1.4. Đặc điểm của kiến trúc V6

Kế thừa các đặc điểm nổi trội của kiến trúc v4 và v5, ở kiến trúc v6 các khối 'TEJ' được tích vào lõi ARM. Để đảm bảo khả năng tương thích ngược phần bộ nhớ và xử lý ngoại lệ được kế thừa từ kiến trúc v5. Về kiến trúc, có 5 điểm chính được cải tiến ở kiến trúc v6:

- Quản lý bộ nhớ: bộ nhớ cache và khối quản lý bộ nhớ (MMU- Memory Management Unit) được cải tiến làm tăng hiệu suất thực thi của hệ thống lên 30% so với kiến trúc cũ.
- Đa nhân (Multiprocessor): đáp ứng các hệ thống mà ở đó yêu cầu khả năng tốc độ xử lý nhanh như: phương tiện giải trí cá nhân, xử lý số… Các nhân chia sẻ và đồng bộ dữ liệu với nhau thông qua vùng nhớ chung.
- Hỗ trợ xử lý đa phương tiện: tích hợp bộ tập lệnh SIMD (Single Instruction Multiple Data) làm tăng khả năng xử lý dữ liệu dạng âm thanh và hình ảnh. Hơn 60 lệnh SIMD đã được thêm vào bộ lệnh của kiến trúc v6. SIMD cũng cho phép các nhà phát triển cài đặt các ứng dụng phức tạp hơn như: giải mã dữ liệu âm thanh và hình ảnh, các bài toán nhận diện, hiển thị hình ảnh 3D hoặc hỗ trợ thiết bị sử dụng công nghệ không dây.
- Kiểu dữ liệu: là cách hệ thống sử dụng và lưu trữ dữ liệu trong bộ nhớ. Như ta đã biết, các hệ thống SoC (System on Chip), các chip vi xử lý đơn, hệ điều hành và các giao diện ngoại vi như USB hoặc PCI hay hoạt động dựa trên kiểu dữ liệu "little endian". Một số

các giao thức như TCP/IP hay MPEG lại sử dụng hệ thống "big endian". Để có thể tối ưu hóa khả năng tích hợp của hệ thống, ARMv6 hỗ trợ cùng lúc cả hai định dạng "little" và "big" endian, gọi tắt là "mixed-endian". Bên cạnh đó, ARMv6 còn cung cấp tập lệnh để xử lý dữ liệu dạng "unalignment" - có kích thước dữ liệu thay đổi. Tương tự như ARMv5, ARMv6 cũng là kiến trúc 32 bit, nên hỗ trợ đường truyền dữ liệu 64 bit hoặc cao hơn.

- Xử lý ngoại lệ và ngắt: Để thích ứng cho các hệ thống xử lý thời gian thực bảng vector ngắt được giới thiệu.

Trên thị trường, dòng ARM11 là đại diện phổ biến nhất của kiến trúc ARMv6. Với kiến trúc đường ống 8 tầng (ở ARM1156T có kiến trúc đường ống 9 tầng), hệ thống dự đoán rẽ nhánh (Branch Prediction) và kết quả trả về (Return Stack) giúp ARM11 nâng cao hiệu suất thực thi lệnh.

2.1.5. Đặc điểm của kiến trúc V7

ARM Cortex là một phiên bản khác với các phiên bản ARM thường hay được ký hiệu bởi ARMXX. ARM Cortex không có tốc độ hoạt động hay hệ thống ngoại vi nhất định, tùy thuộc vào nhà sản xuất phần cứng sẽ thiết kế hệ thống ngoại vi khác nhau, tuy nhiên tất cả đều dùng chung nhân ARM Cortex và việc lập trình và truy cập phần cứng phải tuân theo chuẩn CMSIS.

2.2. Kiến trúc và các thành phần bên trong của ARM7

2.2.1. Kiến trúc tổng quát của ARM 7

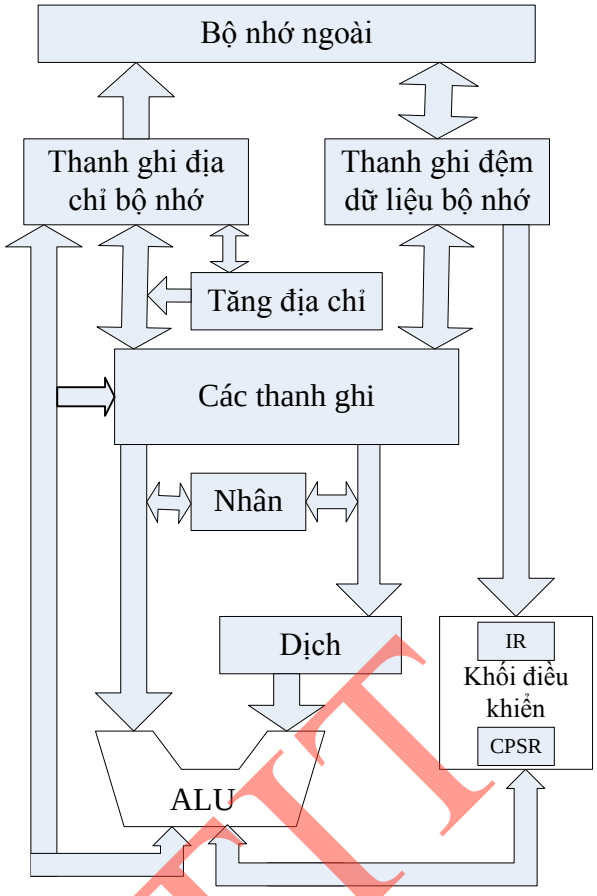
Hình 2.2 mô tả kiến trúc tổng quát của bộ vi xử lý ARM.

Vi xử lý ARM là hệ thống sử dụng tập lệnh đơn giản (Reduced Instruction Set Computer – RISC) bao gồm các thuộc tính sau:

- Chứa một mảng các thanh ghi đồng nhất
- Hỗ trợ kiến trúc nạp/lưu trữ: các dữ liệu được nạp vào thanh ghi trước khi các lệnh được thực hiện. Kết quả có thể được sử dụng cho phép toán tiếp theo hoặc lưu lại vào bộ nhớ.

Các thành phần cơ bản của ARM7 bao gồm:

- Bộ đếm chương trình (Program counter): giữ địa chỉ của lệnh hiện tại.
- Thanh ghi tích lũy (ACC): giữ giá trị dữ liệu khi đang làm việc
- Khối xử lý số học và logic (ALU): thực thi các lệnh nhị phân như cộng, trừ, tăng, giảm,…
- Khối dịch chuyển (Barrel Shifter)
- Khối nhân (Booth Multiplier)
- Thanh ghi lệnh (IR): giữ lệnh hiện tại đang thực thi.



Hình 2. 2 Sơ đồ khối kiến trúc vi xử lý ARM

ALU: Khối xử lý số học và logic bao gồm 2 đầu vào 32 bit và thực hiện các phép toán số học và logic dựa trên các tín hiệu điều khiển 5 bit từ chân ALU_control. Các phép toán logic bao gồm AND, OR, XOR. Các phép toán số học bao gồm Cộng, Cộng có nhớ, Trừ, Trừ có nhớ, Di chuyển dữ liệu giữa các thanh ghi và các cổng vào/ra. ALU có hai đầu ra: đầu ra 32 bit và các tín hiệu cờ 4 bit. Đầu ra 32 bit là các kết quả của phép tính trên các toán hạng tại đầu vào 1 và 2. Các cờ 4 bit hỗ trợ các lệnh điều kiện của ARM7. Các cờ bao gồm N, Z, C, V. Cờ N báo hiệu kết quả âm. Cờ Z báo hiệu kết quả bằng 0, Cờ C báo hiệu phép toán có nhớ. Cờ V báo hiệu các tình huống tràn.

Khối nhân: Module nhân có các đầu vào bao gồm hai toán hạng A, B (32 bit), tín hiệu Enable và xung đồng hồ hệ thống (System clock). Module nhân có chức năng thực hiện phép nhân 2 toán hạn 32 bit. Tín hiệu Enable báo cho module biết khi nào bắt đầu nhân và tín hiệu Ready được đưa ra khi phép nhân kết thúc.

Khối dịch: Khối dịch gồm một đầu vào để tiếp nhận toán hạng 32 bit cần dịch. Toán hạng được đưa đến từ các thanh ghi hoặc dữ liệu từ ô nhớ. Chức năng của khối bao gồm dịch trái, dịch phỉa, quay trái, quay phải.

Khối điều khiển: Trong vi xử lý, khối điều khiển được coi là "trái tim" của hệ thống. Khối này chịu trách nhiệm giải mã lệnh. Mã lệnh đọc vào từ bộ nhớ được đưa đến đầu vào của bộ giải

mã, các thông tin thu được từ đầu ra của nó sẽ được đưa đến mạch tạo xung điều khiển. Kết quả là ta thu được các dãy xung khác nhau (tùy theo mã lệnh) để điều khiển hoạt động của các bộ phận bên trong và bên ngoài CPU.

2.2.2. Sơ đồ chân của ARM7

Sơ đồ các chân của ARM7 được chia thành một số cụm chân tín hiệu như sau:

***MCLK** (memory clock):* Chân tín hiệu đồng hồ. Đây là tín hiệu đồng bộ cho tất cả các hoạt động của bộ vi xử lý bao gồm truy cập bộ nhớ và tính toán. Tốc độ của đồng hồ có thể được giảm để cho phép truy cập các thiết bị ngoại vi hoặc bộ nhớ có tốc độ chậm hơn.

nWAIT: Tín hiệu đồng hồ được sử dụng để thay thế cho chân MCLK.

nRESET: Chân tín hiệu được sử dụng để khởi động vi xử lý từ một địa chỉ đã biết. Khi chân tín hiệu ở mức thấp, bộ vi xử lý sẽ dừng các hoạt động đang thực hiện và khởi động lại. Tín hiệu trên chân này phải được giữ ở mức thấp trong ít nhất 2 chu kỳ đồng đồ cùng với chân nWAIT được giữ ở mức cao.

nM[0:4]: Tín hiệu tại các chân này biểu diễn trạng của các bit chế độ hoạt động của vi xử lý.

Các chân tín hiệu ngắt:

nIRQ (not Fast Interrupt Request): Khi tín hiệu chân này ở mức thấp, vi xử lý sẽ bị ngắt

nFIQ

Các chân điều khiển bus:

ALE (Address Latch Enable): Khi tín hiệu của chân ở mức thấp, các tín hiệu địa chỉ được giữ trên bus địa chỉ trong suốt chu kỳ truy nhập bộ nhớ.

***DBE** (Data Bus Enable):* Khi tín hiệu của chân ở mức cao, dữ liệu sẽ được giữ trên bus dữ liệu.

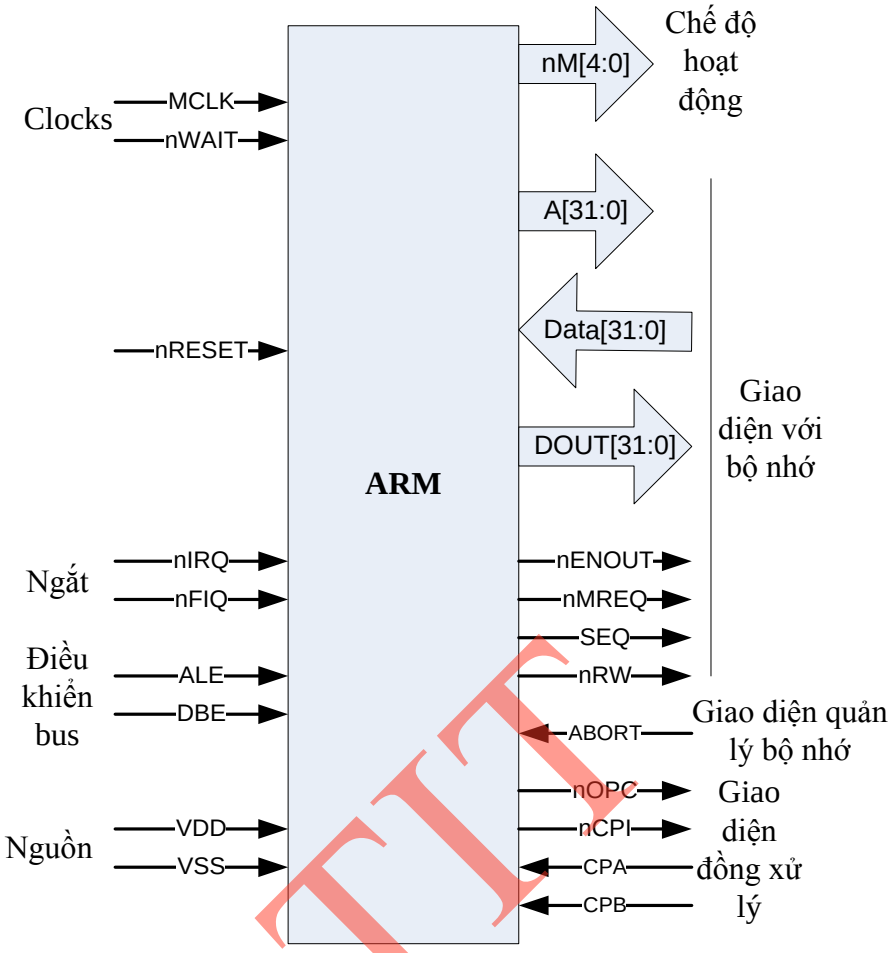
Các chân cấp nguồn:

VDD: Cung cấp nguồn cho vi xử lý.

VSS: Chân nối đất.

Giao diện với bộ nhớ:

A[31:0]: Bus địa chỉ bao gồm 32 bit địa chỉ. Các chân ALE, ABE và APE được sử dụng để điều khiển bus địa chỉ.



Hình 2. 3 Sơ đồ chân tín hiệu của ARM

D[31:0]: Bus dữ liệu 32 bit có thể truyền theo hai hướng giữa bộ vi xử lý và bộ nhớ ngoài. Trong chu kỳ đọc dữ liệu, dữ liệu phải ở trên bus tại thời điểm xuất hiện cạnh xuống của xung MCLK. Trong chu kỳ ghi, dữ liệu phải được duy trì trên bus sau thời điểm xuất hiện cạnh xuống của xung MCLK. Chân DBE được dùng để điều khiển bus dữ liệu.

nENOUT *(not Enable Output):* Trong chu kỳ ghi dữ liệu, tín hiệu ở chân này phải được đặt ở mức thấp trước khi xuất hiện cạnh lên của xung MCLK và giữ nguyên mức này trong toàn chu kỳ ghi.

mMREQ *(not Memory Request):* Khi vi xử lý yêu cầu truy nhập bộ nhớ trong chu kỳ tiếp theo, chân tín hiệu này sẽ ở mức thấp.

SEQ *(Sequence Address):* Chân tín hiệu ở mức cao chỉ ra rằng địa chỉ của ô nhớ trong chu kỳ tiếp theo sẽ là nối tiếp với địa chỉ của ô nhớ vừa được truy cập trong chu kỳ trước.

nRW *(not Read, Write):* Khi vi xử lý đang ở chu kỳ đọc, mức tín hiệu của chân sẽ ở mức thấp.

Giao diện quản lý bộ nhớ:

***ABORT** (Memory Abort):* Bộ nhớ ngoài sử dụng chân tín hiệu này để thông báo cho vi xử lý biết yêu cầu truy nhập không được phép.

Giao diện đồng xử lý:

***nOPC** (not op-code fetch):* Chân tín hiệu ở mức thấp khi vi xử lý đang nạp một lệnh từ bộ nhớ.

***nCPI** (not coprocessor instruction):* Chân tín hiệu ở mức thấp khi lệnh đồng xử lý được thực hiện. Vi xử lý sẽ đợi trả lời từ một bộ đồng xử lý trên chân CPA và CPB.

***CPA** (Coprocessor Absent):* Chân tín hiệu được đặt ở mức thấp bởi bộ đồng xử lý nếu bộ đồng xử lý có thể thực hiện phép toán được yêu cầu bởi bộ vi xử lý.

***CPB** (Coprocessor Busy):* Chân tín hiệu ở mức thấp bởi bộ đồng xử lý khi bộ đồng xử lý bắt đầu thực hiện phép toán được yêu cầu từ bộ vi xử lý.

2.3. Chu trình dữ liệu

2.3.1. Chu trình dữ liệu chung của ARM:

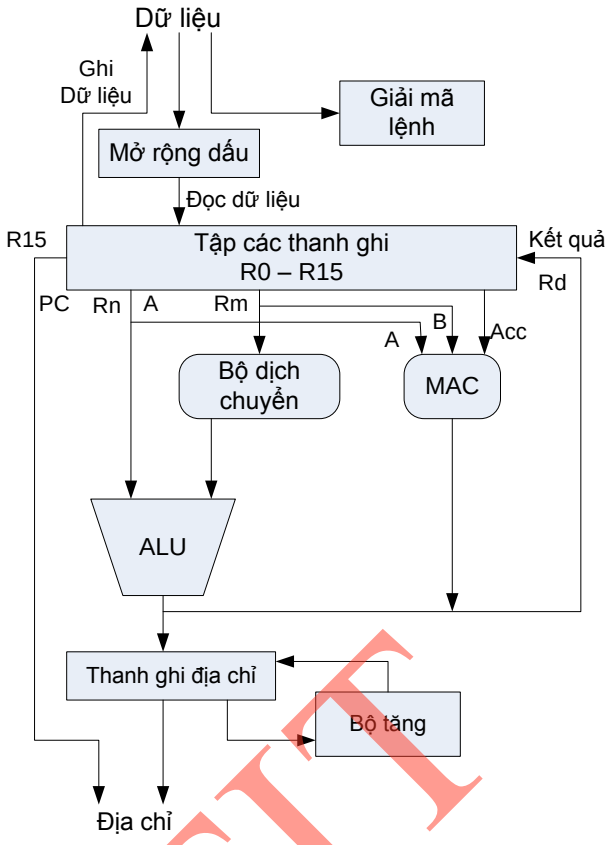
Lõi vi xử lý AMR là một khối chức năng được kết nối bởi các bus dữ liệu. Các mũi tên thể hiện cho dòng chảy của dữ liệu, các đường thể hiện cho bus dữ liệu, và các ô biểu diễn trong hình là một khối hoạt động hoặc một vùng lưu trữ. Cấu hình này cho thấy các dòng dữ liệu và các thành phần tạo nên một bộ xử lý ARM.

Các bộ giải mã sẽ định hướng dịch chuyển trước khi chúng được thực thi. Mỗi một lệnh thực hiện thuộc về một tập lệnh riêng biệt.

Bộ xử lý ARM, giống như tất cả bộ xử lý RISC, sử dụng kiến trúc load – store. Điều này có nghĩa là có hai lệnh để chuyển dữ liệu giữa bộ vi xử lý với bộ nhớ: lệnh load cho phép sao chép dữ liệu từ bộ nhớ vào thanh ghi trong lõi xử lý, và ngược lại lệnh store cho phép sao chép dữ liệu từ thanh ghi tới bộ nhớ. Không có lệnh xử lý dữ liệu trực tiếp trong bộ nhớ (ví dụ cộng, trừ, nhân, chia,… hai toán hạng cùng nằm trong bộ nhớ). Việc xử lý dữ liệu chỉ được thực hiện trong các thanh ghi.

Tất cả dữ liệu thao tác nằm trong các thanh ghi, các thanh ghi có thể là toán hạng nguồn, toán hạng đích, con trỏ bộ nhớ. Các dữ liệu 8 bit, 16 bit đều được mở rộng thành 32 bit trước khi đưa vào thanh ghi nhờ bộ mở rộng dấu.

Tập lệnh ARM nằm trong hai nguồn thanh ghi *Rn* và *Rm*, và kết quả được trả về thanh ghi đích *Rd*. Nguồn toán hạng được đọc từ thanh ghi đang sử dụng trên bus nội bộ A và B tương ứng.



Hình 2. 4 Chu trình dữ liệu chung của ARM

Như mô tả trên hình 2.4, khối số học và logic (ALU: Arithmetic Logic Unit) hay bộ tích lũy nhân (MAC: Multiply Accumulate Unit) lấy các giá trị thanh ghi *Rn* và *Rm* từ bus A và B, và tính toán kết quả (bộ tích lũy nhân có thể thực hiện phép nhân giữa hai thanh ghi và cộng kết quả với một thanh ghi khác). Các lệnh xử lý dữ liệu ghi các kết quả trực tiếp trong *Rd* rồi trả về tệp thanh ghi.

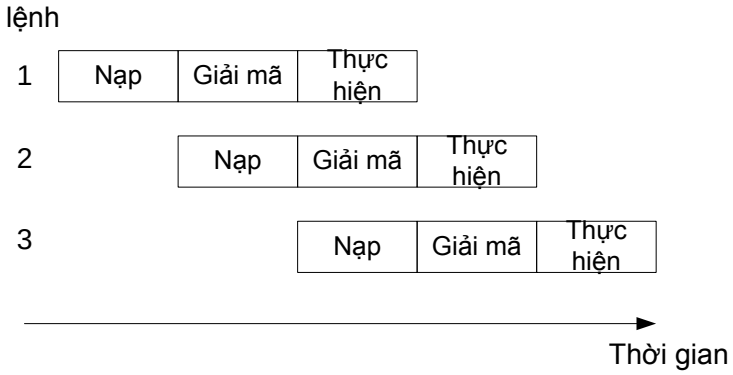
Một tính năng quan trọng của ARM là thanh ghi *Rm* còn có thể được xử lý trong Shifter (bộ dịch chuyển) trước khi nó đi vào ALU. Shifter và ALU có thể phối hợp với nhau để tính toán các biểu thức và địa chỉ.

Sau khi đi qua các khối chức năng, kết quả trong *Rd* được ghi trở lại tệp thanh ghi. Tập lệnh load – store cập nhật tăng địa chỉ thanh ghi trước khi lõi xử lý đọc hoặc ghi giá trị vào thanh ghi từ vị trí nhớ tuần tự tiếp theo. Lõi vi xử lý tiếp tục thực hiện các lệnh cho đến khi xảy ra một ngắt ngoại lệ hoặc có thay đổi dòng chảy thực hiện bình thường.

Cách tổ chức của nhân ARM không thay đổi nhiều trong khoảng 1983-1995: đến ARM7-sử dụng dòng chảy lệnh sử dụng 3 tác vụ. Từ 1995 trở về sau, đã xuất hiện một vài nhân ARM mới được giới thiệu có dòng chảy lệnh sử dụng 5 tác vụ. Các dòng ARM sau này có thể có dòng chảy lệnh 6 tác vụ (ARM10), 9 tác vụ (ARM11) hoặc 13 tác vụ (ARM Cortex).

2.3.2. Dòng chảy lệnh 3 tác vụ:

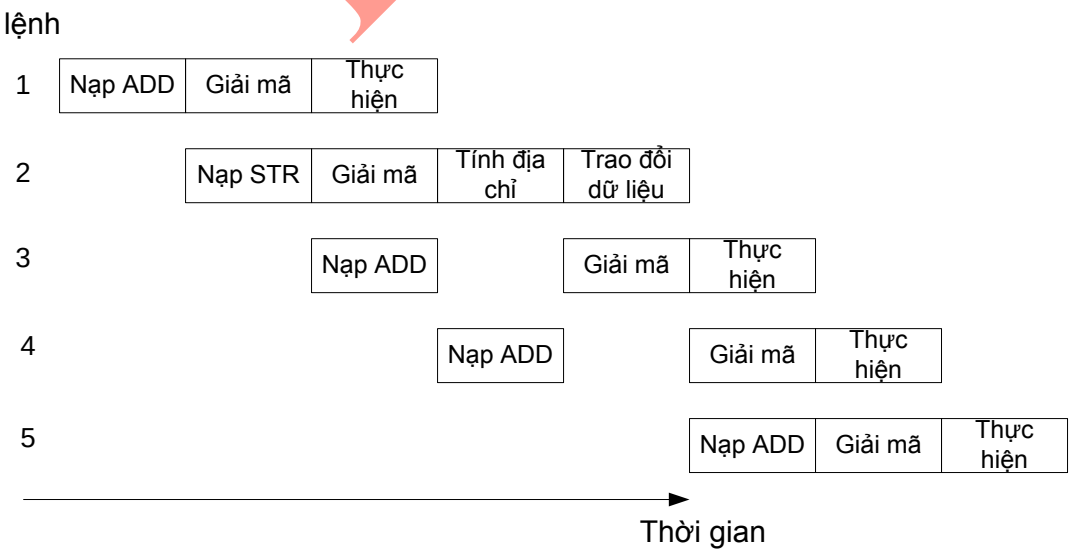
Bao gồm tác tác vụ sau: Fetch-decode-Excute (nhận lệnh, giải mã, thực thi):



Hình 2. 5 Sơ đồ dòng chảy 3 tác vụ

Tác vụ đầu tiên vi xử lý thực hiện là đọc một lệnh từ bộ nhớ và tăng giá trị của thanh ghi địa chỉ lệnh. Việc này đồng nghĩa với việc địa chỉ lệnh tiếp theo sẽ được lưu vào trong thanh ghi địa chỉ lệnh PC. Tác vụ thứ hai vi xử lý thực hiện là giải mã lệnh và chuẩn bị các tín hiệu điều khiển để thực hiện lệnh. Tác vụ thứ ba bao gồm một số công việc sau: vi xử lý đọc các toán hạng từ tệp thanh ghi (register file), thực hiện các phép toán trong ALU, đọc hoặc ghi dữ liệu vào bộ nhớ (nếu cần) và cuối cùng là ghi ngược trở lại thanh ghi. Trong trường hợp lệnh đang được thực thi là lệnh xử lý dữ liệu thì kết quả được tạo ra bởi ALU được ghi trực tiếp vào tệp thanh ghi và tác vụ thực thi được hoàn thành trong một chu kỳ. Nếu lệnh được thực thi là lệnh nạp hoặc lưu dữ liệu thì địa chỉ bộ nhớ được tính bởi ALU được đưa lên bus địa chỉ và việc truy cập vào bộ nhớ được thực hiện trong chu kỳ thứ hai của tác vụ thực thi.

2.3.3. Dòng chảy lệnh 5 tác vụ:



Hình 2. 6 Sơ đồ dòng chảy 5 tác vụ

Trong sơ đồ dòng chảy 5 tác vụ, mỗi lệnh bao gồm các tác vụ: Fetch-decode-excute-buffer/data-write back.

Mô hình dòng chảy trên hình 2.6 được xây dựng với duy nhất một cổng để ghi/đọc bộ nhớ. Điều này đồng nghĩa với việc các lệnh truyền dữ liệu gây ra hiện tượng "ngưng dòng chảy" bởi vì lệnh tiếp theo không thể được đọc trong lúc bộ nhớ đang được đọc hoặc ghi. Để giải quyết vấn đề này, từ ARM9TDMI trở đi, lệnh và dữ liệu được truyền trên hai đường độc lập.

Đầu tiên, ARM9 chuyển bước đọc thanh ghi sang tác vụ giải mã vì tác vụ này ngắn hơn tác vụ thực thi. Thứ hai, tác vụ thực thi được chia thành ba tác vụ nhỏ hơn. Tác vụ thứ nhất thực thi tính toán các lệnh số học, tác vụ thứ hai thực thi truy cập bộ nhớ (tác vụ này ở trạng thái nghỉ (idle) khi thực thi lệnh xử lý dữ liệu) và tác vụ thứ ba ghi dữ liệu vào tệp thanh ghi.

2.3.4. Dòng chảy lệnh 6 tác vụ:

- Nạp và dự đoán lệnh
- Giải mã lệnh
- Đọc thanh ghi
- Thực hiện dịch và các phép toán trong ALU
- Truy cập bộ nhớ
- Ghi vào thanh ghi

2.3.5. Dòng chảy lệnh 9 tác vụ:

- Ba tác vụ Fetch.
- Một tác vụ Decode
- Một tác vụ Issue
- Bốn tác vụ integer execution pipeline

2.4. Các thanh ghi

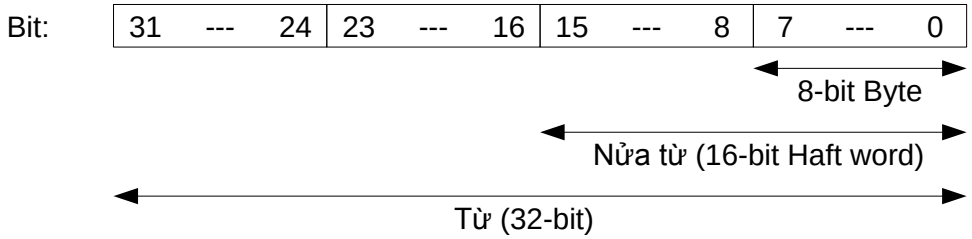
Lõi ARM có 37 thanh ghi trong đó có 30 thanh ghi đa dụng, 6 thanh ghi trạng thái và một thanh ghi bộ đếm chương trình (Hình 2.7). Các thanh ghi đa dụng có thể dùng để lưu dữ liệu hoặc địa chỉ. Các thanh ghi này được đánh dấu bằng ký hiệu R. Tất cả các thanh ghi đều là 32 bit.

Tại một thời điểm chỉ có 15 thanh ghi đa dụng (từ R0 đến R14) và 2 thanh ghi hiển thị trạng thái được sử dụng tùy theo chế độ. Các thanh ghi khác ở dạng ẩn, chỉ hiển thị ở một số chế độ hoạt động riêng. Trong 15 thanh ghi đa dụng có 8 thanh ghi (R0 – R7) luôn được sử dụng trong tất cả các chế độ. 5 thanh ghi (R8 - R12) cũng được sử dụng trong hầu hết các chế độ ngoại trừ chế độ ngắt nhanh (fast interrupt – fiq). Trong chế độ này, các thanh ghi nói trên sẽ được thay thế bằng một tập các thanh ghi khác (R8_fiq – R12_fiq).

Các chế độ						
	Các chế độ đặc quyền					
		Các chế độ ngoại lệ				
User	System	Supervisor	Abort	Undefined	Interrupt	Fast Interrupt
R0	R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7	R7
R8	R8	R8	R8	R8	R8	R8_fiq
R9	R9	R9	R9	R9	R9	R9_fiq
R10	R10	R10	R10	R10	R10	R10_fiq
R11	R11	R11	R11	R11	R11	R11_fiq
R12	R12	R12	R12	R12	R12	R12_fiq
R13	R13	R13_svc	R13_abt	R13_und	R13_irq	R13_fiq
R14	R14	R14_svc	R14_abt	R14_und	R14_irq	R14_fiq
PC	PC	PC	PC	PC	PC	PC
CPSR	CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
		SPSR_svc	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq

Hình 2. 7 Tổ chức thanh ghi của ARM7

Các thanh ghi đa dụng có thể được như các thanh ghi 8 bit, 16 bit hoặc 32 bit. Khi sử dụng như thanh ghi 8 bit hoặc 16 bit thì các bit có trọng số thấp của thanh ghi được sử dụng (hình 2.8).



Hình 2. 8 Thanh ghi 8 bit, 16 bit và 32 bit

Các thanh ghi còn lại (R13 – R15) là 3 thanh ghi có các chức năng hoặc nhiệm vụ đặc biệt riêng: R13, R14, R15:

- Thanh ghi r13 được dùng làm con trỏ ngăn xếp stack pointer (SP)
- Thanh ghi r14 được gọi là thanh ghi kết nối (LR) chứa địa chỉ quay lại của chương trình khi chương trình chạy một hàm con.

- Thanh ghi r15 là bộ đếm chương trình (PC) và chứa địa chỉ của lệnh tiếp theo.

Hai thanh ghi trạng thái bao gồm: Thanh ghi trạng thái chương trình hiện tại (CPSR) chứa thông tin về trạng thái của vi xử lý ở thời điểm hiện tại (bao gồm cả chế độ hiện tại) và Thanh ghi lưu trữ trạng thái chương trình (SPSR) dùng để lưu trữ thông tin trạng thái vi xử lý trước khi vi xử lý chuyển sang chế độ khác (các chế độ ngoại lệ).

2.4.1. Con trỏ ngăn xếp, SP – R13

Như mô tả ở hình 2.7, tại mỗi chế độ ngoại lệ, con trỏ ngăn xếp được sử dụng trên một thanh ghi khác nhau dùng để chỉ đến các ngăn xếp dành riêng cho các chế độ này.

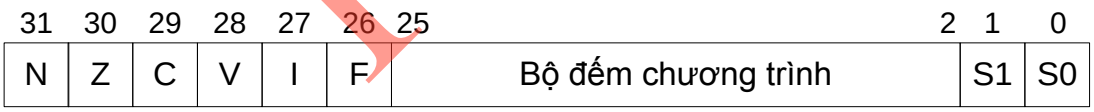
Ngăn xếp thường được dùng để lưu trữ tạm thời các giá trị của một thanh ghi bất kỳ trước khi thanh ghi tham gia vào một công việc nào đó. Điều này giúp giải phóng thanh ghi trong việc lưu trữ dữ liệu khi thực hiện công việc. Sau khi kết thúc công việc, thanh ghi có thể nhận lại giá trị từ ngăn xếp.

2.4.2. Thanh ghi kết nối

Thanh ghi kết nối được sử dụng để lưu địa chỉ quay trở về chương trình chính sau khi thực hiện chương trình con. Khi chương trình con được gọi thông qua lệnh *BL*, thanh ghi R14 sẽ lưu địa chỉ tiếp theo. Để quay trở về chương trình chính từ chương trình con, chúng ta cần phải chép dữ liệu địa chỉ từ thanh ghi R14 sang thanh ghi bộ đếm chương trình (PC) bởi lệnh *MOV PC, LR* hoặc *BAL LR*.

2.4.3. Thanh ghi bộ đếm chương trình

Thanh ghi PC lưu trữ địa chỉ của lệnh tiếp theo sẽ được thực hiện. Vì vậy, thanh ghi này cũng được gọi là thanh ghi *con trỏ lệnh*. Khi một lệnh đọc PC được thực hiện, giá trị được trả về là địa chỉ của lệnh hiện tại cộng với 8 bytes địa chỉ của 2 lệnh tiếp sau lệnh hiện tại.

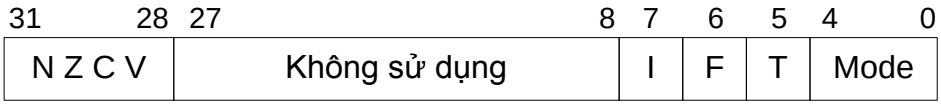


Hình 2. 9 Thanh ghi trạng thái chương trình hiện tại

Bit 2-25 dùng để lưu địa chỉ của lệnh tiếp theo. Địa chỉ lệnh có độ dài 24 bit (mặc dù địa chỉ có độ dài 26 bit vì bộ nhớ chương trình có dung lượng 64MB) bởi các lệnh được quy định là phải nằm trong giới hạn của một từ (word boundary). Vì vậy, hai bit cuối của địa chỉ lệnh luôn bằng 0 và không cần phải lưu hai bit này. Khi nội dung của R15 được đưa lên bus địa chỉ, bit 0 và bit 1 tự động được đặt bằng 0.

2.4.4. Thanh ghi trạng thái chương trình hiện tại - CPSR

CPSR có 4 trường, mỗi trường có 8 bit: cờ, trạng thái, mở rộng và điều khiển. Hiện tại phần trạng thái và mở rộng được dự trữ cho các thiết kế tương lai.



Hình 2. 10 Thanh ghi trạng thái chương trình hiện tại

Các cờ của CPSR như sau:

- N: Negative- cờ này được bật khi bit cao nhất của kết quả xử lý ALU bằng 1.
- Z: Zero- cờ này được bật khi kết quả cuối cùng trong ALU bằng 0.
- C: Carry- cờ này được bật khi kết quả cuối cùng trong ALU lớn hơn giá trị 32bit và tràn.
- V: Overflow-cờ báo tràn sang bit dấu.

Các bit điều khiển:

- Các bit ngắt:
  - I = 1: Cấm ngắt thường IRQ
  - F = 1: Cấm ngắt nhanh FIQ
- Bit trạng thái T:
  - T = 1: VXL hoạt động ở chế độ Thumb
  - T = 0: VXL hoạt động ở chế độ ARM
- Các bit chế độ:

M[4:0]	Chế độ	Các thanh ghi được phép truy nhập
10000	User	PC, R0 – R14, CPSR
10001	FIQ	PC, R0 – R7, R8_fiq – R14_fiq, CPSR, SPSR_fiq
10010	IRQ	PC, R0 – R12, R13_irq, R14_irq, CPSR, SPSR_irq
10011	SVC	PC, R0 – R12, R13_svc, R14_svc, CPSR, SPSR_svc
10111	Abort	PC, R0 – R12, R13_abt, R14_abt, CPSR, SPSR_abt
11011	Undef	PC, R0 – R12, R13_und, R14_und, CPSR, SPSR_und
11111	System	PC, R0 – R14, CPSR

Bảng 2. 1 Các chế độ hoạt động của ARM

2.5. Các chế độ hoạt động

Chế độ hoạt động của bộ vi xử lý sẽ xác định thanh ghi nào hoạt động và quyền truy cập tới thanh ghi CPSR. Mỗi chế độ hoạt động của bộ vi xử lý sẽ là chế độ đặc quyền (Priviliged) và không đặc quyền (Unpriviliged): Chế độ đặc quyền cho phép đọc và ghi tới thanh ghi CPRS. Ngược lại chế độ không đặc quyền chỉ cho phép đọc trường điều khiển của CPRS nhưng vẫn cho phép đọc và ghi tới các cờ điều kiện.

Có bảy chế độ hoạt động của bộ VXL: sáu chế độ đặc quyền (abort, fast interrupt, request, interrupt request, supervisor, system, and undefined) và một chế độ không đặc quyền (user). Đối

với từng chế độ sẽ có các thanh ghi riêng cho từng chế độ đó. Sơ đồ các thanh ghi các chế độ như mô tả trong hình 2.7.

Hoạt động của các chế độ như sau:

- Bộ VXL hoạt động ở chế độ Abort khi bộ VXL không thể truy cập bộ nhớ.
- Bộ VXL hoạt động ở chế độ interrupt request (IRQ) và fast interrupt request (FIQ) tương ứng với hai mức ngắt của chip ARM.
- Bộ VXL hoạt động ở chế độ Supervisor sau khi hệ thống khởi động (reset) hoặc khi có chương trình ngắt mềm.
- Bộ VXL hoạt động ở chế độ System khi hệ thống có thể truy cập và đọc, ghi toàn bộ thanh ghi CPRS.
- Bộ VXL chuyển sang chế độ Undefined khi bộ VXL gặp một lệnh không xác định hoặc không được hỗ trợ.
- Bộ VXL hoạt động ở chế độ User là để chạy các chương trình và các ứng dụng thông thường.

2.6. Hệ thống ngắt

2.6.1. Chế độ ngoại lệ (Exception)

Khi đang thực thi chương trình, ngoại lệ xảy ra khi hệ thống rơi vào các trường hợp đặc biệt như khởi động lại hệ thống, các phép tính chia cho 0, các lệnh sai cú pháp, không truy cập được bộ nhớ, khi có ngắt ngoài hoặc ngắt của chương trình. Mỗi tình huống ngoại lệ đều có một chương trình đi kèm gọi là *chương trình xử lý ngoại lệ* (exception handler). Khi gặp tình huống ngoại lệ, vi xử lý ARM tự động chuyển đến một địa chỉ chứa đã định sẵn cho chương trình xử lý ngoại lệ đó. Địa chỉ định sẵn đó gọi là *địa chỉ vector ngoại lệ* (Exception vector address) và nó nằm tại vị trí 32 byte cuối cùng của bộ nhớ ROM. Khoảng nhớ 32 byte này được gọi là *bảng vector ngoại lệ* (exception vector table). Bảng sau mô tả một số chế độ ngoại lệ của ARM:

Ngoại lệ	Chế độ hoạt động	Địa chỉ vector
Reset (Khởi động lại hệ thống)	Supervisor (Giám sát)	0x00000000
Undefined instruction (Lệnh bị sai)	Undefined (Không xác định)	0x00000004
Software Interrupt (Ngắt mềm)	Supervisor	0x00000008
Prefetch Abort (Không nạp được lệnh do sai địa chỉ)	Abort	0x0000000C
Data Abort (Không nạp được dữ liệu so sai địa chỉ)	Abort	0x00000010
Interrupt (IRQ) (Ngắt)	Interrupt (IRQ)	0x00000018
Fast Interrupt (FIQ) (Ngắt nhanh)	Fast Interrupt (FIQ)	0x0000001C

Bảng 2. 2 Chế độ ngoại lệ của ARM

Địa chỉ 0x00000014 chưa được sử dụng. Mỗi ngoại lệ có 4 byte để lưu 01 lệnh. Lệnh này thường là lệnh để chuyển con trỏ lệnh đến địa chỉ của chương trình xử lý ngoại lệ.

Trước khi xử lý một ngoại lệ, trạng thái hiện tại của chương trình chính phải được lưu lại. Nói cách khác, nội dung của tất cả các thanh ghi (đặc biệt là thanh ghi PC và CPSR) sau khi xử lý ngoại lệ phải giống như trước khi xử lý ngoại lệ. Việc lưu giữ trạng thái này đảm bảo không làm mất đi tính ổn định của chương trình chính.

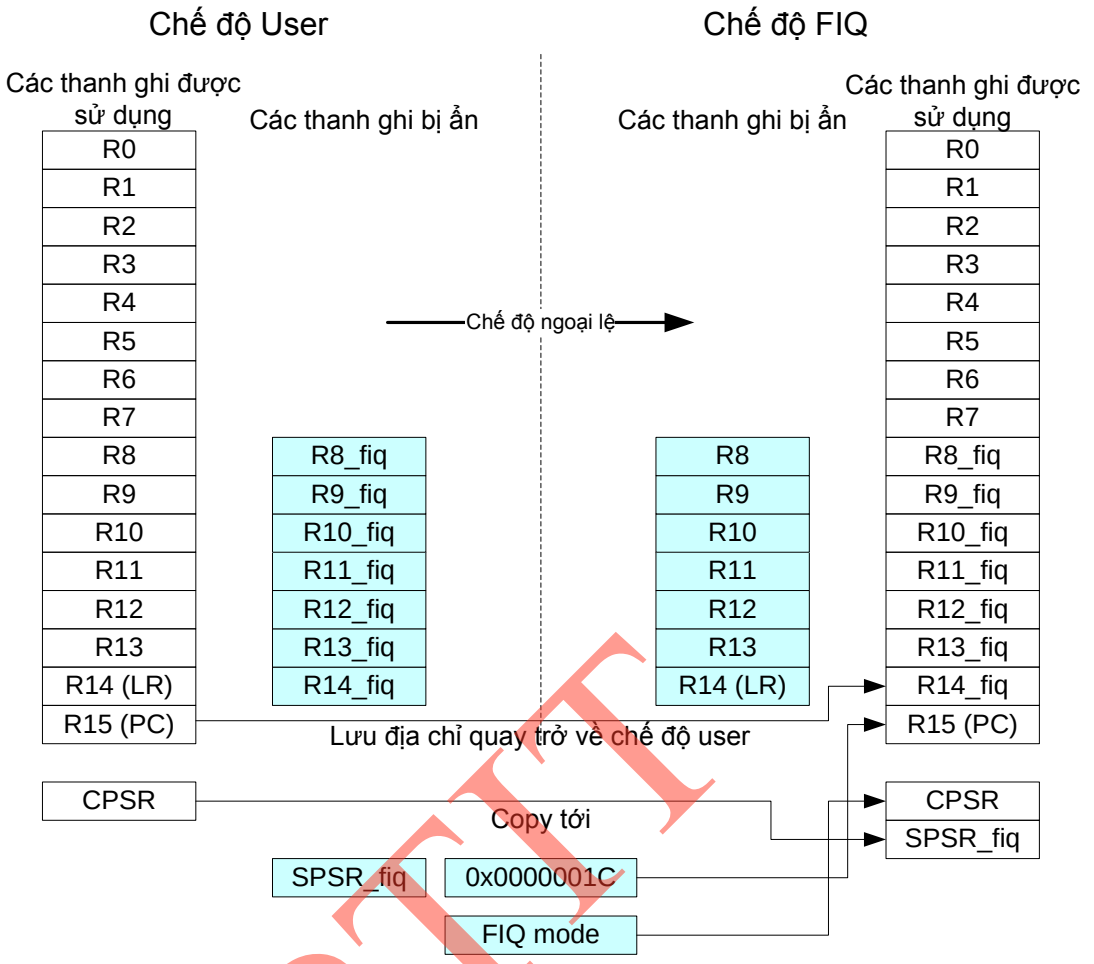
Vi xử lý Arm sử dụng các thanh ghi trống cùng với các thanh ghi chuyên dùng của mỗi chế độ để lưu giữ trạng thái. Để xử lý một ngoại lệ, vi xử lý Arm thực hiện các bước sau:

1. Lưu địa chỉ của lệnh ngay sau lệnh gọi ngoại lệ trong chương trình chính vào thanh ghi LR.
2. Lưu nội dung CPRS vào thanh ghi SPSR tương ứng với mỗi chế độ.
3. Thiết lập 5 bit chế độ của CPRS tương ứng với ngoại lệ.
4. Thiết lập trạng thái ARM bằng cách đặt bit thứ 5 (bit T) của CPSR bằng 0.
5. Cấm ngắt nhanh bằng cách đặt bit 6 (bit F) của CPSR bằng 1.
6. Cấm ngắt thường bằng cách đặt bit 7 (bit I) của CPSR bằng 1.
7. Nạp địa chỉ của vector ngoại lệ vào thanh ghi bộ đếm chương trình PC.

Sau khi thực hiện 7 bước trên, bộ vi xử Arm bắt đầu thực hiện các lệnh tại địa chỉ vector ngoại lệ. Lưu ý, trong suốt quá trình thực hiện lệnh của ngoại lệ, các thanh ghi dành riêng của chế độ đó sẽ thay thế các thanh ghi tương ứng của chế độ User. Ví dụ, trong khi ở chế độ FIQ, các thanh ghi từ R8 đến R14 sẽ không được sử dụng và thay vào đó là các thanh ghi R8_fiq – R14_fiq.

Sau khi thực hiện xong chương trình xử lý ngoại lệ, chương trình xử lý ngoại lệ thực hiện:

1. Copy nội dung thanh ghi LR trở lại vào thanh ghi PC
2. Copy SPSR trở lại CPSR.



Hình 2. 11 Chuyển từ chế độ User sang chế độ FIQ

Vì các ngoại lệ có thể xảy ra đồng thời nên bộ vi xử lý đưa ra các mức ưu tiên cho mỗi loại ngoại lệ. Trong trường hợp có nhiều ngoại lệ xảy ra cùng lúc thì bộ vi xử lý quyết định thực hiện các ngoại lệ theo thứ tự ưu tiên. Bảng sau mô tả các mức ưu tiên của các ngoại lệ:

Ngoại lệ	Mức ưu tiên
Reset	1
Data Abort	2
FIQ	3
IRQ	4
Pre-fetch abort	5
Software interrupt (SWI)	6
Underfine instruction	6

Bảng 2. 3 Thứ tự ưu tiên các ngoại lệ

2.6.2. Ngắt mềm (Software Interrupt)

Ngắt mềm là một loại ngoại lệ được gọi ra bằng câu lệnh trong chương trình. Trong vi xử lý Arm, lệnh gọi ngắt mềm là *SWI*. Khi lệnh được thực hiện, vi xử lý sẽ chuyển sang hoạt động ở chế độ Supervisor và rẽ nhánh làm việc tới địa chỉ vector ngoại lệ tương ứng là 0x00000008. Nói cách khác, lệnh *SWI* sẽ tạo ra một ngoại lệ nhưng khác với các ngoại lệ khác, ngoại lệ này có thể biết trước bởi chương trình.

Các ngắt mềm rất hữu ích bởi nó cho phép chương trình đang chạy ở chế độ User có thể chuyển sang một chế độ đặc quyền nào đó. Giống như một ngoại lệ nói chung, trước khi thực hiện ngắt mềm, vi xử lý thực hiện một số công việc sau:

1. Lưu địa chỉ tiếp theo ngay sau lệnh SWI trong chương trình chính vào thanh ghi LR_svc (R14_svc).
2. Copy nội dung CPSR vào thanh ghi SPSR_svc.
3. Đặt các bit chế độ của CPSR tương ứng với chế độ Supervisor (10011). Việc này sẽ ẩn các thanh ghi R13, R14 và thay vào đó là thanh ghi R13_svc, R14_svc.
4. Đặt bit T = 1.
5. Đặt bit 7 (bit I) trong CPSR bằng 1. Điều này có nghĩa là các ngắt thường không thể xen vào trong quá trình ngắt mềm được gọi trừ khi bit 7 được đặt bằng 0 trong chương trình xử lý ngắt mềm. Các ngắt nhanh vẫn có thể được gọi.
6. Lưu địa chỉ vector ngoại lệ, 0x00000008, vào thanh ghi PC.

Sau khi kết thúc ngắt mềm, chương trình ngắt thực hiện các việc sau:

1. Copy nội dung thanh ghi LR_svc vào thanh ghi PC.
2. Copy nội dung SPSR_svc vào thanh ghi CPSR.

CHƯƠNG 3. LẬP TRÌNH HỢP NGỮ ARM

3.1. Tổng quan về tập lệnh ARM

Tất cả các lệnh ARM đều có chiều dài 32 bit. Các lệnh của ARM có thể chia thành các nhóm theo chức năng như sau:

- Lệnh rẽ nhánh;
- Lệnh xử lý dữ liệu;
- Lệnh nạp dữ liệu từ ô nhớ vào thanh ghi đơn và ngược lại;
- Lệnh nạp dữ liệu từ các ô nhớ vào các thanh ghi và ngược lại;
- Lệnh truy cập thanh ghi trạng thái;
- Lệnh đồng xử lý.

Tập lệnh ARM có một số khả năng sau:

- Thực hiện lệnh theo điều kiện đặt ra;
- Truy cập thanh ghi;
- Truy cập bộ dịch các bit trong thanh ghi.

3.2. Cấu trúc chung của chương trình

Cấu trúc đơn giản của một chương trình ARM viết bằng ngôn ngữ Assembly như sau:

```
        AREA Example, CODE, READONLY          ; Tên của chương trình
SWI_Exit   EQU   0x11
        ENTRY                                 ; Đánh dấu lệnh đầu tiên được
                                              thực hiện
Start
        MOV  r0, #15                          ; Thiết lập các tham số
        MOV  r1, #20
        BL     Firstfunc                      ; Gọi hàm con
        SWI    SWI_Exit                       ; Thoát khỏi chương trình
Firstfunc                                     ; Hàm con
        ADD   r0, r0, r1                      ; r0 =  r0 + r1
        MOV   pc, lr                          ; Trở lại chương trình chính

        END                                   ; Đánh dấu kết thúc chương trình
```

AREA:

Một chương trình ứng dụng có thể được tạo nên bởi nhiều đoạn mã hoặc dữ liệu. Các đoạn mã đứng độc lập và được gán tên. Trong một chương trình ứng dụng, mỗi đoạn mã được đánh dấu bởi lệnh chỉ dẫn AREA. Một chương trình ứng dụng có thể có một hoặc nhiều AREA. Đoạn

mã ở ví dụ trên chỉ bao gồm một AREA và AREA này có thuộc tính là chỉ được đọc (READONLY). CODE là phần nhỏ nhất tạo nên một ứng dụng.

Cú pháp: *AREA sectionname{,attr}{,attr}…*

Trong đó: *sectionname* là tên của đoạn chương trình. Lưu ý, nếu tên chương trình bắt đầu bằng số thì phải được đặt trong hai dấu |. Ví dụ: |1_DataArea|.

Attr là thuộc tính của đoạn chương trình. Một số thuộc tính thông dụng bao gồm:

CODE: Đoạn chương trình chứa các lệnh. Thuộc tính mặc định đi kèm là READONLY, tức là đoạn chương trình này chỉ cho phép đọc.

DATA: Đoạn dữ liệu, không có các lệnh. Thuộc tính đi kèm là READWRITE, tức là đoạn chương trình cho phép đọc và ghi.

ENTRY:

Lệnh đầu tiên được thực hiện trong một ứng dụng được đánh dấu bởi ENTRY. Một ứng dụng chỉ có duy nhất một ENTRY. Nếu ứng dụng gồm nhiều module thì chỉ có một module chứa ENTRY.

EQU: Chỉ dẫn EQU gán một tên cho một hằng số.

END: Lệnh thông báo cho trình biên dịch biết điểm kết thúc của chương trình.

Cấu trúc tổng quát của một dòng lệnh:

nhãn <dấu cách> lệnh <dấu cách> ;chú giải

Ba thành phần của dòng lệnh (*nhãn*, *lệnh* và *chú giải*) phải được cách nhau ít nhất một khoảng trống (dấu cách hoặc dấu tab). Trong trường hợp không có *nhãn* thì lưu ý *lệnh* cũng phải nằm sau ít nhất một khoảng trống. Cả ba thành phần trên đều là tùy chọn, thậm chí có những dòng trống trong chương trình cũng được trình biên dịch chấp nhận.

3.3. Biên dịch và chạy các chương trình hợp ngữ cho ARM

Để biên dịch và chạy các chương trình hợp ngữ cho ARM có một số cách sau:

a. Sử dụng bộ phát triển phần mềm cho ARM (ARM software development toolkit).
b. Sử dụng phần mềm mô phỏng Keil 5 được phát triển bởi ARM .
c. Sử dụng phần mềm mô phỏng ARMSIM#. Đây là phần mềm do các giảng viên ngành Khoa học máy tính, trường đại học University of Victory phát triển. Tài liệu hướng dẫn cài đặt và sử dụng phần mềm có thể tìm trên website:
http://armsim.cs.uvic.ca/DownloadARMSimSharp.html.
Trong phạm vi bài giảng này, phần mềm ARMSIM# được sử dụng vì một số lý do:
- Cài đặt dễ dàng trên cả hai môi trường Windows và Linux;
- Giao diện thân thiện;

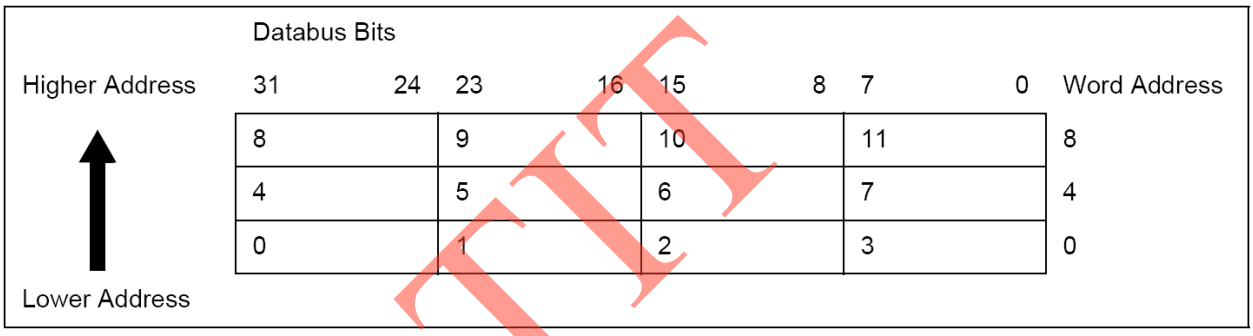
- Cho phép người sử dụng mô phỏng các chương trình viết bằng ngôn ngữ Assembly cho bộ vi xử lý ARM7TDMI.

Hạn chế của phần mềm ARMSIM# là không hỗ trợ tập lệnh Thumb.

3.4. Định dạng các ô nhớ của ARM

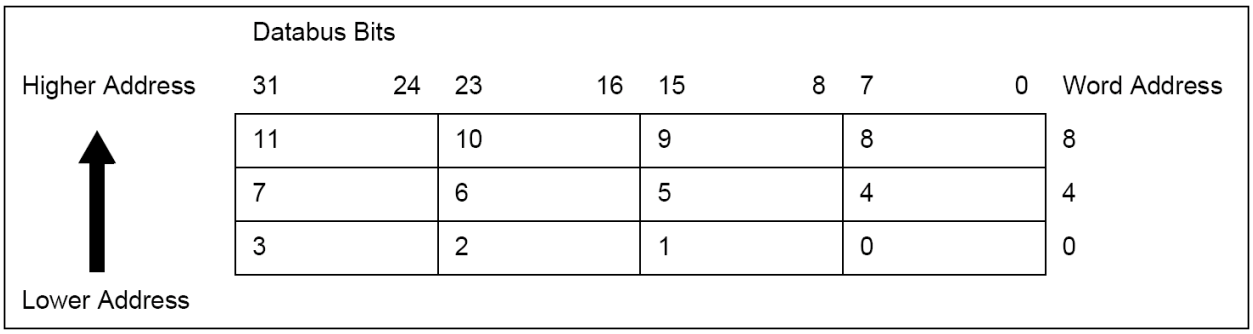
Bộ nhớ của ARM bao gồm các byte nhớ được đánh địa chỉ từ 0. Byte có địa 0 đến 3 lưu từ nhớ thứ nhất, byte có địa chỉ từ 4 đến 7 lưu từ nhớ thứ hai,… Dữ liệu trong các ô nhớ được định dạng theo hai cách: Từ trái sang phải (Big - endian) hoặc từ phải sang trái (Little - endian).

Định dạng Big - endian: Trong một từ nhớ, byte có trọng số lớn nhất của từ nhớ được lưu vào ô nhớ có địa chỉ thấp nhất và byte có trọng số nhỏ nhất được lưu vào ô nhớ có địa chỉ cao nhất. Trong hình 3.1, trong từ nhớ 0, byte có địa chỉ thấp nhất (0-7) lưu byte dữ liệu số 3 (có trọng số cao nhất) và byte có địa chỉ cao nhất (24-31) lưu byte dữ liệu số 0 (có trọng số thấp nhất).



Hình 3. 1 Định dạng Big - endian

Định dạng Little - endian: Trong một từ nhớ, byte có trọng số lớn nhất của từ nhớ được lưu vào ô nhớ có địa chỉ cao nhất và byte có trọng số nhỏ nhất được lưu vào ô nhớ có địa chỉ thấp nhất.



Hình 3. 2 Định dạng Little - endian

Trong hình 3.2, trong các từ nhớ, byte thấp nhất có địa chỉ từ 0 -7, byte có trọng số cao nhất có địa chỉ từ 24 -31.

Mặc định của vi xử lý ARM là chế độ Little - endian.

3.5. Các lệnh xử lý dữ liệu

Các lệnh xử lý dữ liệu chỉ thực thi các phép tính đối với dữ liệu trên các thanh ghi mà KHÔNG thực hiện trên ô nhớ. Nhóm lệnh xử lý dữ liệu trên thanh ghi bao gồm:

- Lệnh di chuyển dữ liệu giữa các thanh ghi: MOV, MVN
- Lệnh số học (Arithmetic): ADD, ADC, SUB, SBC, RSB, RSC
- Lệnh logic: AND, ORR, EOR, BIC
- Lệnh so sánh: CMP, CMN, TST, TEQ
- Lệnh nhân: MUL, MLA, UMULL, UMLAL, SMULL, SMLAL

Cú pháp tổng quát của lệnh:

Opcode{Cond}{S} Rd, Rn, {Operand2}

- Trong câu lệnh, các tham số trong ngoặc {} là các tham số tùy chọn. Nghĩa là có thể có hoặc không có trong câu lệnh.
- Opcode: Lệnh gợi nhớ.
- Cond: điều kiện để thực hiện lệnh. Lệnh có điều kiện chỉ được thực hiện nếu các cờ trong thanh ghi CPSR thỏa mãn điều kiện. Một số điều kiện được liệt kê trong bảng sau:

Điều kiện	Cờ bị tác động	Chú giải
EQ	Z = 1	Bằng
NE	Z = 0	Không bằng
CS/HS	C = 1	Cao hơn hoặc bằng
CC/LO	C = 0	Thấp hơn
MI	N = 1	Giá trị âm
PL	N = 0	Giá trị dương
VS	V = 1	Tràn có dấu
VC	V = 0	Không tràn có dấu
HI	C=1, Z=0	Cao hơn
LS	C=0, Z=1	Thấp hơn hoặc bằng
GE	N = V	Lớn hơn hoặc bằng
LT	N # V	Nhỏ hơn
GT	Z=0 và N#V	Lớn hơn
LE	Z=1 hoặc N#V	Nhỏ hơn hoặc bằng
AL		Thực hiện lệnh không cần điều kiện

Bảng 3. 1 Các điều kiện có thể đi kèm với lệnh

- S: Tham số được sử dụng khi muốn kết quả tác động đến các cờ trên thanh ghi CPRS. Nếu không có *{S}*, các cờ sẽ không bị thay đổi giá trị.
- Rd: Thanh ghi lưu giữ kết quả.

- Rn: Thanh ghi lưu giữ toán hạng thứ nhất.
- Operand2: Toán hạng thứ hai là toán hạng tùy chọn.

3.5.1. Lệnh di chuyển dữ liệu giữa các thanh ghi

Cú pháp:

MOV{cond}{S} Rd, Operand2: Copy giá trị của *Operand2* vào *Rd*

MVN{cond}{S} Rd, Operand2: Copy giá trị đảo của *Operand2* vào *Rd*

Ví dụ 3.1: R1 = 5, R2 = 7

```
MOV R1, R2          ; R1 = 7, R2 = 7
MVN R1, R2          ; R1 = 9 (1001B), R2 = 7(0110B)
MOVCS R0, R1        ; Nếu cờ nhớ C = 1 thì R0 = R1
MOVCC R0, R1        ; Nếu cờ nhớ C = 0 thì R0 = R1
MOVS R0, #0         ; R0 = 0, Z = 1, N = 0, cờ C và V không bị tác động.
```

3.5.2. Lệnh số học

Cú pháp: *opcode{cond}{S} Rd, Rn, Operand2*

Trong đó opcode là một trong các lệnh: ADD, ADC, SUB, SBC, RSB, RSC

Mô tả lệnh:

- ADD: Rd = Rn + Operand2
- ADC (Cộng có xét đến cờ nhớ): Rd = Rn + Operand2 + C
- SUB: Rd = Rn – Operand2
- RSB (Trừ đảo): Rd = Operand2 - Rn
- SBC (Trừ có xét đến cờ nhớ): Rd = Rn – Operand2 - $\bar{C}$
- RSC (Trừ đảo có xét đến cờ nhớ): Rd = Operand2 – Rn - $\bar{C}$

Ví dụ 3.2:

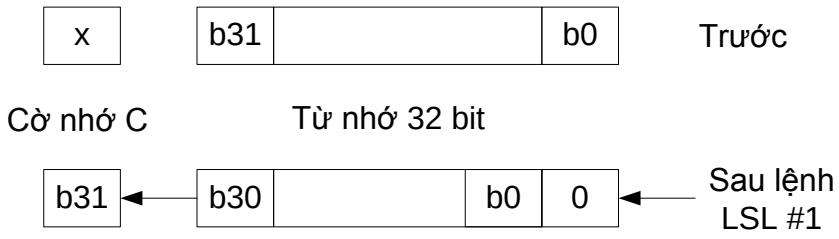
```
ADD  R0,R1,R2       ; R0 = R1 + R2
ADDNE  R0,R0,R2     ; Lệnh chỉ được thực hiện nếu Z = 0
ADDS  R0,R0,R2      ; Sau lệnh này, cờ N, Z, V, C sẽ thay đổi để phản ánh giá trị
                    ; của kết quả.
ADDNES  R0,R0,R2    ; Lệnh được thực hiện nếu Z = 0. Sau khi thực hiện các cờ
                    ; sẽ bị thay đổi giá trị để phản ánh kết quả.
SBC R0, R1, R2      ; R0 = R1 – R2 – giá trị đảo của cờ nhớ C.
```

3.5.3. Toán hạng được dịch và quay

Nếu toán hạng *Operand2* là một thanh ghi, nó có thể được dịch hoặc quay trước khi được đưa vào tính toán trong câu lệnh. Tuy nhiên, lưu ý rằng nội dung của thanh ghi sẽ không bị thay đổi, chỉ có giá trị của nó được đưa vào ALU để xử lý. Một số phép toán dịch và quay bao gồm:

LSL #n: Dịch trái logic (Logical Shift Left) n bit ($1 \leq n \leq 32$)

Sau khi dịch n lần, n bit 0 được thêm vào bên phải và cờ sẽ có giá trị bằng giá trị của bit thứ (32 – n).



Hình 3. 3 Mô tả lệnh LSL

Ví dụ 3.3:

MOV R0, R2, LSL #2

Trước: R2 = 0x00000030 (R2 được viết dưới dạng Hexa)

Sau: R0 = 0x000000C0

R2 = 0x00000030

LSR #n: Dịch phải logic (Logical Shift Right) n bit $(1 \leq n \leq 32)$

Phép dịch này tương tự với phép dịch trái logic. Tuy nhiên hướng dịch là hướng phải.

Ví dụ 3.4:

ASL #n: Dịch trái số học giá trị tức thời (Arithmetic Shift Left)

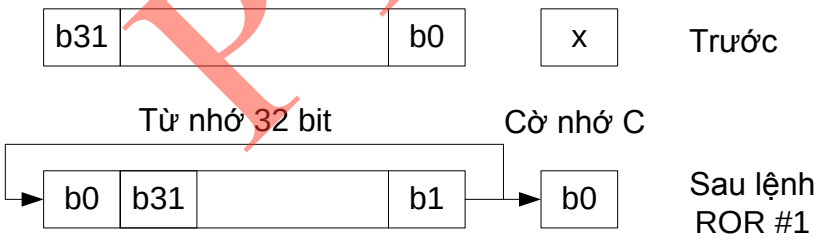
Kiểu dịch này tương đương với LSL và cũng cho kết quả tương tự.

ASR #n: Dịch phải số học giá trị tức thời

Kiểu dịch này tương tự với ASL nhưng hướng dịch chuyển sang bên phải.

Đối với số có dấu, sau khi dịch n lần, n bit dấu sẽ được thêm vào vị trí dịch và cờ dấu (C) có giá trị bằng giá trị của bit thứ (n-1) trong chuỗi ban đầu.

ROR #n: Quay phải giá trị tức thời

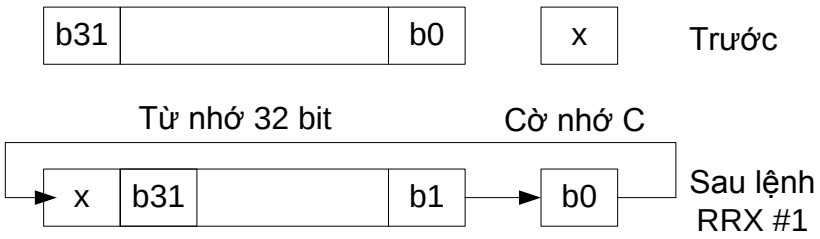


Hình 3. 4 Mô tả lệnh ROR

Sau n lần quay, bit thứ n cũ (tính từ 0) sẽ ở vị trí bit thứ 0 mới, bit thứ (n-1) cũ sẽ ở vị trí thứ 31 mới và cờ C sẽ mang giá trị bit thứ (n-1).

ROL #n: Quay trái giá trị tức thời: Cũng tương tự như ROR nhưng hướng ngược lại.

RRX: Quay phải một bit có mở rộng



Hình 3. 5 Mô tả lệnh RRX

Tương tự, ta có một số phép dịch thanh ghi. Tương tự như dịch giá trị tức thời, các giá trị của thanh ghi sẽ được dịch. Tuy nhiên, trong phép dịch thanh ghi, chỉ có 8 bit thấp của thanh ghi được dịch:

LSL Rn: Dịch trái logic thanh ghi Rn

ASL Rn: Dịch trái số học thanh ghi Rn

LSR Rn: Dịch phải logic thanh ghi Rn

ASR Rn: Dịch phải số học thanh ghi Rn

ROR Rn: Quay phải thanh ghi Rn

3.5.4. Lệnh logic

Các lệnh logic bao gồm: AND, OR, EOR (Exclusive OR) và lệnh xóa bit.

Cú pháp chung: op{cond}{S} Rd, Rn, Operand2

Op: có thể là AND, OR, EOR hoặc BIC

Cond: điều kiện tùy chọn theo bảng 3.1

S: Hậu tố tùy chọn để cập nhật cờ trạng thái sau khi thực hiện lệnh.

Rd: Thanh ghi lưu kết quả

Rn: Thanh ghi lưu toán hạng thứ nhất

Operand2: Toán hạng thứ hai (có thể là thanh ghi hoặc số nguyên).

AND Rd, Rn, Operand2: Rd = Rn AND Operand2

Rn	**Operand2**	Rd = Rn AND Operand2
0	0	0
0	1	0
1	0	0
1	1	1

OR Rd, Rn, Operand2: Rd = Rn OR Operand2

Rn	**Operand2**	Rd = Rn OR Operand2
0	0	0
0	1	1
1	0	1

1	1	1

EOR Rd, Rn, Operand2: Rd = Rn EOR Operand2

Rn	**Operand2**	Rd = Rn EOR Operand2
0	0	0
0	1	1
1	0	1
1	1	0

BIC Rd, Rn, Operand2: Rd = Rn AND NOT Operand2

Rn	**Operand2**	NOT Operand2	Rd = Rn BIC Operand2
0	0	1	0
0	1	0	0
1	0	1	1
1	1	0	0

3.5.5. Lệnh so sánh

CMP{cond} Rn, Operand : Lệnh so sánh được sử dụng để so sánh 2 số nguyên. Phép so sánh được thực hiện bằng cách lấy toán hạng Operand trừ giá trị trong thanh ghi Rn và thiết lập trạng thái cờ theo kết quả của phép trừ.

Lưu ý:

- Toán hạng Operand có thể là thanh ghi hoặc số nguyên.
- Phép so sánh thực hiện phép trừ nhưng không lưu kết quả và giá trị của các toán hạng cũng không bị thay đổi sau phép so sánh.
- Lệnh CMP có thể ghép với các điều kiện trong bảng 3.1 để thực hiện phép so sánh theo điều kiện đặt ra cho các toán hạng.

Ví dụ:

Thực hiện điều kiện sau: Nếu (R0 = R1) và (R2 = R3) thì tăng R4 lên 1 đơn vị.

```
CMP     R0, R1      ; So sánh R0 và R1
CMPEQ   R2, R3      ; So sánh R2 và R3 trong trường hợp R0 = R1
ADDEQ   R4, R4      ; R4 = R4 + 1 nếu R2 = R3.
```

CMN{cond} Rn, Operand: Ngược với lệnh CMP, trong phép so sánh CMN, toán hạng Operand sẽ được so sánh với giá trị -Rn. Điều này có nghĩa là các cờ trạng thái sẽ được thiết lập dựa trên kết quả của phép cộng (Operand + Rn).

TST{cond} Rn, Operand: Lệnh này thiết lập các cờ trạng thái dựa trên kết quả Operand AND Rn. Lệnh này tương tự lệnh AND nhưng khác ở chỗ kết quả không được lưu vào thanh ghi.

TEQ{cond} Rn, Operand: Lệnh này thiết lập các cờ trạng trái dựa trên kết quả Operand XOR Rn. Lệnh này tương tự lệnh XOR nhưng khác ở chỗ kết quả không được lưu vào thanh ghi.

3.5.6. Lệnh nhân

3.5.6.1.Lệnh nhân 32 bit

MUL, MLA: Nhân và nhân tích lũy

Cú pháp: *MUL{cond}{S} Rd, Rm, Rs* *; Rd = Rm * Rs*

MLA{cond}{S} Rd, Rm, Rs, Rn *; Rd = Rm*Rs + Rn*

Trong đó: cond: điều kiện tùy chọn.

S: Hậu tố tùy chọn để cập nhật cờ trạng thái sau khi thực hiện lệnh.

Rd: Thanh ghi lưu kết quả.

Rm, Rs, Rn: Thanh ghi lưu các toán hạng.

Lưu ý: Thanh ghi R15 không được sử dụng cho Rd, Rm, Rs hoặc Rn.

Rd và Rm không được trùng nhau.

3.5.6.2.Lệnh nhân 64 bit

UMULL, UMLAL, SMULL, và SMLAL: Lệnh nhân không dấu và có dấu

Cú pháp chung: *Op{cond}{S} RdLo, RdHi, Rm, Rs*

Trong đó Op: Lệnh UMULL, UMLAL, SMULL hoặc SMLAL.

cond: điều kiện tùy chọn.

S: Hậu tố tùy chọn để cập nhật cờ trạng thái sau khi thực hiện lệnh.

RdLo, RdHi: Các thanh ghi lưu kết quả.

Rm, Rs: Lưu các toán hạng.

Lưu ý: Thanh ghi R15 không được sử dụng cho RdHi, RdLo, Rm, Rs.

RdLo, RdHi và Rm phải là các thanh ghi khác nhau.

Cách sử dụng:

SMAL	Lệnh nhân tích lũy 64 bit có dấu (các toán hạng trong thanh ghi được coi là các số có dấu)	[RdHi, RdLo] = [RdHi, RdLo] + Rm*Rs
SMULL	Lệnh nhân 64 bit có dấu	[RdHi, RdLo] = Rm*Rs
UMAL	Lệnh nhân tích lũy 64 bit không dấu	[RdHi, RdLo] = [RdHi, RdLo] + Rm*Rs
UMULL	Lệnh nhân 64 bit không dấu	[RdHi, RdLo] = Rm*Rs

3.6. Các lệnh điều khiển chương trình

B và BL: Rẽ nhánh và rẽ nhánh có sử dụng thanh ghi liên kết.

Cú pháp: *B{cond} label*

BL{cond} label

Trong đó cond: điều kiện tùy chọn

label: nhãn được sử dụng để đánh dấu vị trí chương trình rẽ nhánh tới

Cách sử dụng:

Lệnh B: rẽ nhánh tới vị trí nhãn *label*.

Lệnh BL: Copy địa chỉ của lệnh tiếp theo vào thanh ghi R14 (thanh ghi liên kết LR) và rẽ nhánh tới vị trí nhãn *label.*

Ví dụ:

```
CMP   R0, #5
BEQ   Label        ;  Rẽ nhánh đến Label nếu R0 = 5
ADD   R1, R1, R0   ;  Nếu R0 # 5, thực hiện R1 = R1 + R0
Label: ……
```

BX: Rẽ nhánh và cho phép tùy chọn chuyển đổi giữa tập lệnh ARM và Thumb.

Cú pháp: *BX{cond} Rm*

Trong đó cond: điều kiện tùy chọn

Rm: Thanh ghi chứa địa chỉ cần rẽ nhánh tới. Lưu ý bit 0 của Rm sẽ không được sử dụng để biểu diễn địa chỉ. Nếu bit 0 được thiết lập bằng 1, cờ T trong thanh ghi CPSR sẽ tự động được thiết lập bằng 1 và do đó lệnh sẽ được chuyển sang chế độ Thumb. Ngược lại, nếu bit 0 bằng 0 thì lệnh ở chế độ ARM.

BLX: Rẽ nhánh có sử dụng thanh ghi liên kết và cho phép tùy chọn chuyển đổi giữa tập lệnh ARM và Thumb.

Cú pháp: *BLX{cond} Rm*

BLX Label

Lệnh BLX sẽ thực hiện một số bước sau:

- Copy địa chỉ của lệnh tiếp theo vào thanh ghi R14 (thanh ghi liên kết RL)
- Rẽ nhánh tới nhãn Label hoặc địa chỉ chứa trong thanh ghi Rm
- Thay đổi tập lệnh sang chế độ Thumb nếu bit 0 của Rm được thiết lập bằng 1 hoặc lệnh *BLX Label* được sử dụng.

3.7. Các lệnh trao đổi dữ liệu giữa thanh ghi và bộ nhớ

Các lệnh trao đổi dữ liệu giữa thanh ghi và bộ nhớ được chia thành 3 nhóm chính:

- Trao đổi dữ liệu giữa ô nhớ và một thanh ghi
- Trao đổi dữ liệu giữa ô nhớ và nhiều thanh ghi
- Hoán chuyển dữ liệu giữa ô nhớ và thanh ghi

3.7.1. Trao đổi dữ liệu giữa ô nhớ và một thanh ghi

LDR/STR: Load/Store một thanh ghi

Cú pháp: *LDR{cond}{B|SB|H|SH} Rd, [Rn]*

STR{cond}{B|SB|H|SH} Rd, [Rn]

Trong đó cond: điều kiện tùy chọn

B: Dữ liệu 1 byte không dấu

SB: Dữ liệu 1 byte có dấu

H: Dữ liệu nửa từ (2 byte) không dấu

SH: Dữ liệu nửa từ có dấu

Rd: Thanh ghi lưu dữ liệu

[Rn]: Ô nhớ có địa chỉ lưu trong thanh ghi Rn

Ý nghĩa các lệnh:

LDR	Nạp 1 từ từ bộ nhớ ngoài vào thanh ghi	Rd ← $[Rn]_{32}$
STR	Lưu 1 từ trong thanh ghi ra ô nhớ ngoài	$[Rn]_{32}$ → Rd
LDRB	Nạp 1 byte từ bộ nhớ ngoài vào thanh ghi	Rd ← $[Rn]_{8}$
STRB	Lưu 1 byte từ thanh ghi ra ô nhớ ngoài	$[Rn]_{8}$ → Rd
LDRH	Nạp nửa từ từ bộ nhớ ngoài vào thanh ghi	Rd ← $[Rn]_{16}$
STRH	Lưu nửa từ trong thanh ghi ra ngoài ô nhớ ngoài	$[Rn]_{16}$ → Rd
LDRSB	Nạp 1 byte có dấu từ bộ nhớ ngoài vào thanh ghi	Rd ← sign($[Rn]_{8}$)
STRSH	Lưu 1 byte có dấu từ thanh ghi ra bộ nhớ ngoài	Sign($[Rn]_{8}$) → Rd
LDRSH	Nạp nửa từ có dấu từ bộ nhớ ngoài vào thanh ghi	Rd ← sign($[Rn]_{16}$)
STRSH	Lưu nửa từ có dấu từ thanh ghi ra bộ nhớ ngoài	Sign($[Rn]_{16}$) → Rd

Ví dụ:

Rd = 0x00000000

Rn = 0x0000A5CD

Lệnh	Addr	Addr + 1	Giá trị trong thanh ghi
LDRB	11001101		00000000 00000000 00000000 11001101
LDRH	11001101	10100101	00000000 00000000 10100101 11001101
LDRSB	11001101		11111111 11111111 11111111 11001101
LDRSH	11001101	10100101	11111111 11111111 10100101 11001101
LDRSB	01001101		00000000 00000000 00000000 01001101
LDRSH	11001101	00100101	00000000 00000000 00100101 11001101

3.7.2. Chế độ địa chỉ

Địa chỉ của một ô nhớ ngoài được tạo ra bởi 2 phần: Địa chỉ cơ sở (lưu trong trong thanh ghi Rn) và địa chỉ lệch (offset). Mục đích của việc tách ra 2 phần là cho phép người lập trình có thể truy cập đến các vị trí ô nhớ trong một vùng nhớ một cách linh hoạt. Bộ vi xử lý ARM hỗ trợ 4 chế độ địa chỉ để truy cập đến ô nhớ ngoài:

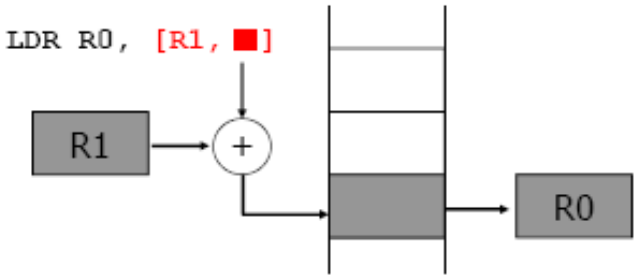
- Chế độ zero-index: Chỉ sử dụng thanh ghi Rn và không có địa chỉ lệch.
- Chế độ pre-index: Nội dung thanh ghi Rn không đổi khi thực hiện lệnh.
- Chế độ auto-index: Nội dung thanh ghi Rn thay đổi trước khi thực hiện lệnh.
- Chế độ post-index: Nội dung thanh ghi Rn thay đổi sau khi thực hiện lệnh.

Chế độ zero-index: Đây là cách đánh địa chỉ đã được trình bày ở mục 5.1

Chế độ pre-index:

Cú pháp: *LDR{cond}{B|SB|H|SH} Rd, [Rn, offset]; Rd = [Rn + offset], Rn không đổi.*
STR{cond}{B|SB|H|SH} Rd, [Rn, offset]; [Rn + offset] = Rd, Rn không đổi.

Ví dụ: LDR R0, [R1, #4] ; R0 = mem[R1 + 4]
; R1 không đổi

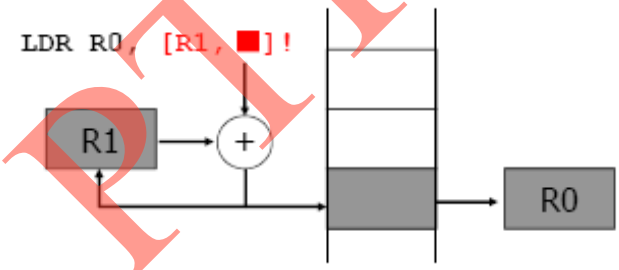


Hình 3. 6 Chế độ Pre-index

Chế độ auto - index:

Cú pháp: *LDR{cond}{B|SB|H|SH} Rd, [Rn, offset]! ; Rd = [Rn + offset], Rn = Rn+offset*
STR{cond}{B|SB|H|SH} Rd, [Rn, offset]! ; [Rn + offset] = Rd, Rn = Rn + offset

Ví dụ: LDR R0, [R1, #4]! ; R0 = mem[R1 + 4]
; R1 = R1 + 4

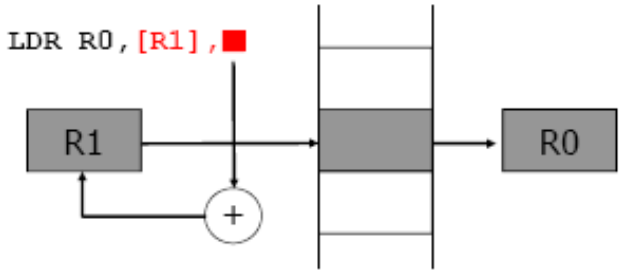


Hình 3. 7 Chế độ Auto-index

Chế độ post - index:

Cú pháp: *LDR{cond}{B|SB|H|SH} Rd, [Rn], offset ; Rd = [Rn], Rn = Rn+offset*
STR{cond}{B|SB|H|SH} Rd, [Rn], offset ; [Rn] = Rd, Rn = Rn + offset

Ví dụ 1: LDR R0, [R1],#4 ; R0 = mem[R1]
; R1 = R1 + 4



Hình 3. 8 Chế độ Post-index

Ví dụ 2: R0 = 0x00000000
R1 = 0x00000009
mem32[R1] = 0x0101010101
mem32[R1 + 4] = 0x02020202

Tìm giá trị của R0, R1 trong các trường hợp sau:

a. LDR R0, [R1,#4]
b. LDR R0, [R1,#4]!
c. LDR R0, [R1], #4

Giải:

a. R0 = 0x0202020202, R1 = 0x00000009
b. R0 = 0x02020202, R1 = 0x0000000D
c. R0 = 0x0101010101, R1 = 0x0000000D

3.7.3. Trao đổi dữ liệu giữa nhiều ô nhớ và nhiều thanh ghi

LDM|STM: Nạp hoặc lưu nhiều thanh ghi đồng thời

Cú pháp: *op{cond}mode Rn{!}, reglist*

Trong đó: op: LDM hoặc STM
cond: điều kiện tùy chọn
mode: Một trong các chế độ sau:

- IA: Tăng địa chỉ sau mỗi lần trao đổi dữ liệu
- IB: Tăng địa chỉ trước mỗi lần trao đổi dữ liệu
- DA: Giảm địa chỉ sau mỗi lần trao đổi dữ liệu
- DB: Giảm địa chỉ trước mỗi lần trao đổi dữ liệu

Ví dụ 1: Nạp các giá trị từ các ô nhỏ được trỏ bởi R0 vào các thanh ghi từ R1 đến R3. Trước khi thực hiện lệnh, R0 = 0x010.

LDMIA R0, {R1 – R3}

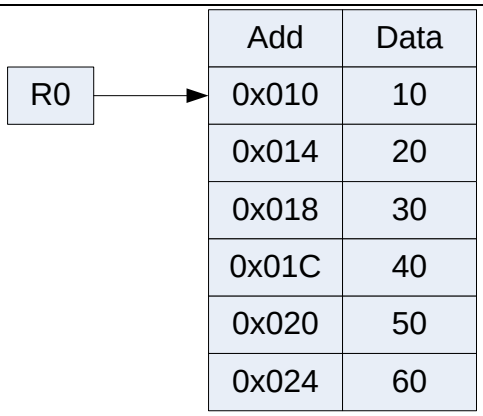
Sau khi thực hiện lệnh:

R1 = 10

R2 = 20

R3 = 30

R0 = 0x10



LDMIA R0!, {R1, R2, R3}

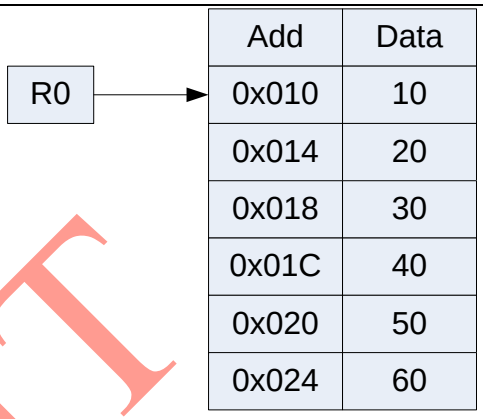
Sau khi thực hiện lệnh:

R1 = 10

R2 = 20

R3 = 30

R0 = 0x1C



LDMIB R0!, {R1, R2, R3}

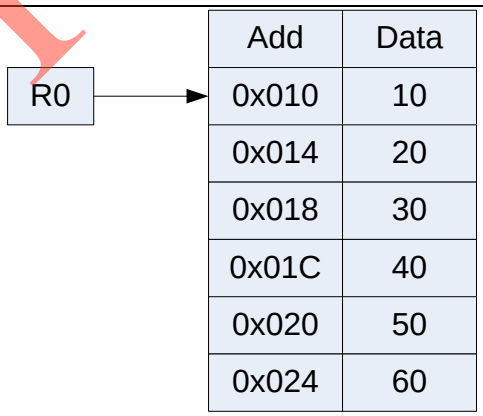
Sau khi thực hiện lệnh:

R1 = 20

R2 = 30

R3 = 40

R0 = 0x1C



LDMDA R0!, {R1, R2, R3}

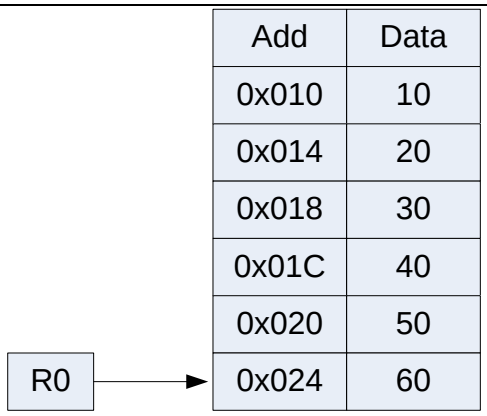
Sau khi thực hiện lệnh:

R1 = 40

R2 = 50

R3 = 60

R0 = 0x18

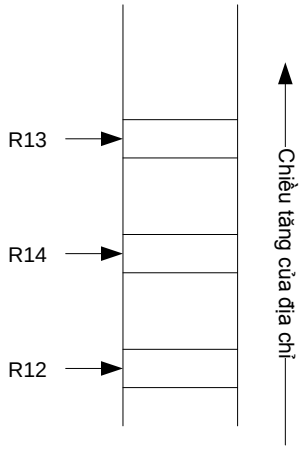


Ví dụ 2: Copy từng khối dữ liệu dài 48 byte (12 từ) từ vị trí ô nhớ được trỏ bởi R12 tới vị trí ô nhớ được trỏ bởi R13. Việc copy được dừng lại khi gặp ô nhớ được trỏ bởi R14.

Giải:

```
; R12 lưu địa chỉ bắt đầu của ô nhớ chứa dữ liệu nguồn
; R13 lưu địa chỉ bắt đầu của ô nhớ đích
; R14 chứa địa chỉ kết thúc của dữ liệu nguồn.

Loop  LDMIA R12! {R0 – R11} ; Nạp 48 byte
      STMIA R13! {R0 – R11} ; Lưu 48 byte
      CMP   R12, R14        ; Kiểm tra địa chỉ kết thúc
      BNE   Loop            ; Tiếp tục lặp
```



3.7.4. Trao đổi dữ liệu giữa ngăn xếp và nhiều thanh ghi

3.7.4.1.Hoạt động của ngăn xếp

Ngăn xếp là vùng nhớ trong RAM có thể mở rộng khi dữ liệu được thêm vào hoặc giảm đi khi dữ liệu được lấy ra. Vùng nhớ này hoạt động theo nguyên tắc LIFO (Last Input First Output – vào trước ra sau), dữ liệu nào đưa vào trước tiên sẽ được lấy ra sau cùng. Để thực hiện chức năng này, CPU sử dụng thanh ghi con trỏ ngăn xếp – SP (R13). Vào đầu chương trình, thanh ghi này sẽ được gán một giá trị trỏ tới một địa chỉ của vùng nhớ RAM. Ngăn xếp sẽ được truy cập bằng các lệnh đặc biệt là PUSH (nạp dữ liệu vào) và POP (lấy dữ liệu ra).

Vi xử lý ARM hỗ trợ một số kiểu ngăn xếp như sau:

- Tăng (Ascending) hoặc giảm (Descending) địa chỉ:

Ở kiểu ngăn xếp tăng, con trỏ SP sẽ ở địa chỉ thấp nhất và sẽ tự động tăng thêm 4 (vì mỗi ô nhớ chứa 4 byte dữ liệu) khi dữ liệu được thêm vào và giảm đi 4 khi dữ liệu được lấy ra. Ở kiểu ngăn xếp giảm, con trỏ SP ban đầu ở địa chỉ cao nhất. SP sẽ tự động giảm đi 4 khi dữ liệu thêm vào và tăng thêm 4 khi dữ liệu được lấy ra.

- Rỗng (empty) hoặc đầy (full):

Ở kiểu ngăn xếp rỗng, con trỏ ngăn xếp sẽ trỏ vào vị trí ô nhớ tiếp theo ô nhớ cuối cùng chứa dữ liệu và do đó con trỏ SP sẽ tăng sau khi thực hiện lệnh PUSH. Ở kiểu ngăn xếp đầy, con trỏ ngăn xếp sẽ trỏ vào vị trí ô nhớ cuối cùng chứa dữ liệu và do đó con trỏ SP sẽ tăng trước khi thực hiện lệnh PUSH.

3.7.4.2.Trao đổi dữ liệu giữa ngăn xếp và thanh ghi

Vi xử lý ARM hỗ trợ một số lệnh sau để trao đổi dữ liệu giữa ngăn xếp và các thanh ghi:

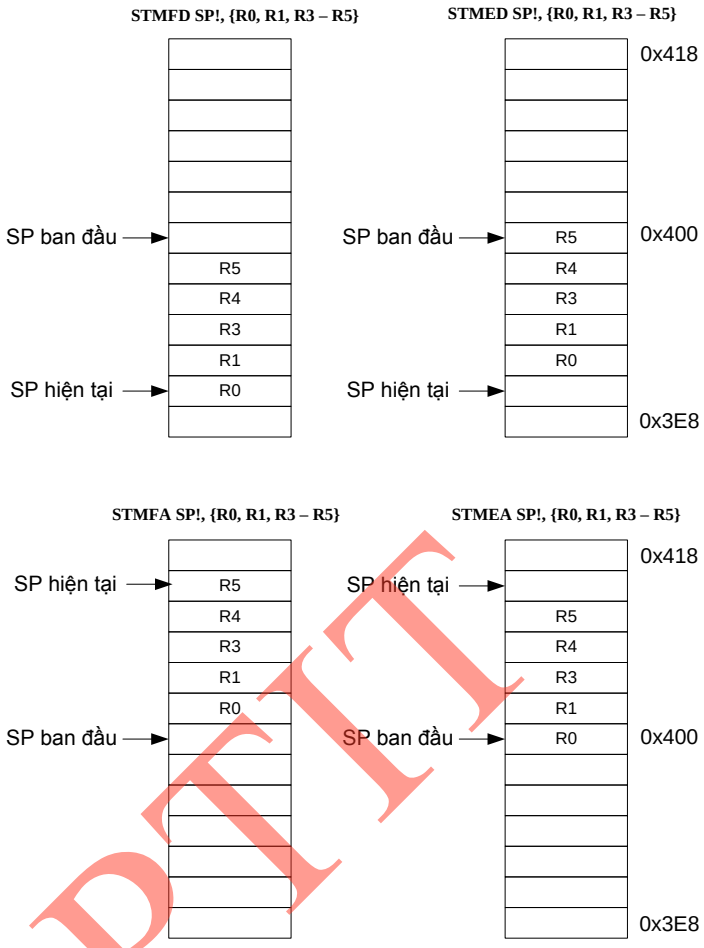
STMFD/LDMFD (Full Descending stack): Sử dụng kiểu ngăn xếp đầy và có địa chỉ giảm dần.

STMFA/LDMFA (Full Ascending stack): Sử dụng kiểu ngăn xếp đầy và có địa chỉ tăng dần.

STMED/LDMED (Empty Descending stack): Sử dụng kiểu ngăn xếp rỗng và có địa chỉ giảm.

STMEA/LDMEA (Empty Ascending stack): Sử dụng kiểu ngăn xếp rỗng và có địa chỉ tăng.

Trên hình 3.7 minh họa hoạt động của con trỏ SP trong 4 trường hợp FD, ED, FA và EA:

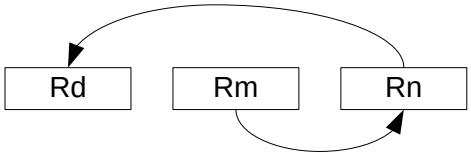


Hình 3. 9 Hoạt động của con trỏ SP

3.7.4.3. Hoán chuyển dữ liệu giữa ô nhớ và thanh ghi

SWP: Hoán chuyển dữ liệu giữa ô nhớ và thanh ghi

Cú pháp: *SWP {cond}{B} Rd, Rm, [Rn]*



Hình 3. 10 Mô tả lệnh SWP

Trong đó *cond:* điều kiện tùy chọn

B: Dữ liệu 1 byte không dấu. Trong trường hợp không sử dụng B thì dữ liệu là 4 byte

Rd: Dữ liệu từ ô nhớ được nạp vào thanh ghi.

Rn: Địa chỉ ô nhớ lưu dữ liệu được chuyển vào thanh ghi Rm. Thanh ghi Rn phải khác với thanh ghi Rd và Rm.

Rm: Nội dung của thanh ghi Rm được lưu vào ô nhớ. Rm có thể trùng với Rd. Trong trường hợp này, lệnh sẽ thực hiện hoán chuyển nội dung giữa Rd và ô nhớ.

3.8. Các lệnh chỉ dẫn trong chương trình

Align: Lệnh chỉ dẫn việc căn chỉnh vị trí ô nhớ hiện tại theo một kích thước nhất định bằng cách chèn thêm các bit 0 vào ô nhớ.

Cú pháp: *Align {expr{,offset}}*

Trong đó: expr là số byte được căn chỉnh. Nếu *expr* không được chỉ ra, Align sẽ căn chỉnh vị trí lệnh tại từ nhớ tiếp theo.

Offset chỉ ra độ lệch từ vị trí được chỉ ra bởi *expr*.

Lệnh Align cho phép căn chỉnh các địa chỉ của lệnh tiếp theo tới vị trí ô nhớ (n*expr + offset), n là các giá trị 0, 1,…. Nếu expr không được chỉ ra, lệnh Align sẽ đặt lệnh tiếp theo ở vị trí từ nhớ tiếp theo tính từ lệnh trước đó.

Ví dụ 1: Đoạn mã sau đây sử dụng 2 ô nhớ 1 byte gồm ô nhớ thứ nhất và ô nhớ thứ 4 trong 1 từ nhớ 4 byte để lưu dữ liệu:

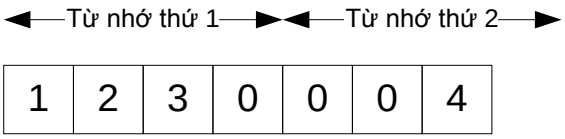
```
        AREA    OffsetExample, CODE
Label   DCB     0x5
        ALIGN   4,3         ; ô nhớ tiếp theo là: 0*4 + 3 = 3
        DCB     0x6
```

Nhãn Label là một mảng ô nhớ trong đó ô nhớ thứ nhất có giá trị là 5. Lệnh ALIGN 4,3 chỉ dẫn rằng lệnh tiếp theo được căn chỉnh 4 byte và cách ô nhớ trước đó một đoạn 3 byte. Kết quả là lệnh DCB thứ hai đặt giá trị 0x6 vào byte cuối cùng của từ nhớ và giá trị 0 được chèn vào 2 byte giữa.

Ví dụ 2:

```
        AREA    OffsetExample1, CODE
        DCB     0x1
        DCB     0x2
        DCB     0x3
        ALIGN   4,2         ; 1*4 + 2
        DCB     0x4
```

Trong ví dụ 2, lệnh ALIGN chỉ dẫn lệnh tiếp theo được căn chỉnh 4 byte và lệch một đoạn 2 byte. Trong trường hợp này, n = 1 vì 3 byte đầu của từ nhớ đã được sử dụng. Ngoài ra, giá trị 0 sẽ được chèn vào 3 byte tiếp theo lệnh DCB thứ 3.



Hình 3. 11 Căn chỉnh ô nhớ

Việc sử dụng Align có mục đích:

- Đảm bảo các địa chỉ dữ liệu Thumb được căn chỉnh theo đúng kích thước của từ nhớ. Ví dụ, lệnh Thumb *ADR label* chỉ nạp dữ liệu có kích thước 4 byte. Tuy nhiên có thể nhãn label đặt tại vùng nhớ không được căn chỉnh 4 byte. Do đó, lệnh *Align 4* phải được sử dụng để đảm bảo dữ liệu được nạp đúng.
- Một số vi xử lý ARM như ARM940T có bộ nhớ đệm dữ liệu là 16 byte. Khi đó lệnh *Align 16* được sử dụng để đảm bảo dữ liệu được nạp đúng và tăng hiệu quả của bộ nhớ đệm.
- Lệnh LDRD và STRD dùng để trao đổi dữ liệu có kích thước 8 byte. Do đó, lệnh *Align 8* phải được sử dụng trước lệnh DCQ để đảm bảo dữ liệu được truy cập khi sử dụng lệnh LDRD và STRD.

DCB: Lệnh chỉ dẫn vi xử lý phân bổ một hoặc nhiều byte của bộ nhớ để lưu dữ liệu.

Cú pháp: {label} DCB expr{ ,expr}…

Trong đó: expr là số nguyên trong khoảng từ -128 đến 255 hoặc chuỗi ký tự. Các ký tự trong chuỗi được nạp vào các byte liên tiếp trong bộ nhớ.

Trong trường hợp DCB được đặt trước một lệnh, chỉ dẫn Align được sử dụng để đảm bảo lệnh đó được căn chỉnh đúng.

DCD: Lệnh chỉ dẫn vi xử lý phân bổ một hoặc nhiều từ nhớ (được căn chỉnh 4 byte) để lưu dữ liệu.

DCW: Lệnh chỉ dẫn vi xử lý phân bổ một hoặc nhiều nửa từ nhớ (được căn chỉnh 2 byte) để lưu dữ liệu.

3.9. Lập trình với ngắt mềm

Các chương trình con xử lý ngắt mềm có số hiệu nằm trong khoảng từ 0 đến 255. Bảng dưới đây liệt kê một số chương trình con xử ngắt mềm:

Số hiệu ngắt	Mô tả chức năng	Đầu vào	Đầu ra
SWI 0x00	Hiển thị một ký tự	R0 lưu ký tự	Hiển thị trên màn hình ký tự
SWI 0x02	Hiển thị một chuỗi ký tự ra màn hình	R0 lưu địa chỉ của chuỗi ký tự	Hiển thị trên màn hình chuỗi ký tự
SWI 0x11	Dừng chương trình		

SWI 0x12	Đăng ký sử dụng vùng nhớ trên Heap	R0 lưu kích thước (đơn vị là byte)	R0: địa chỉ của vùng nhớ
SWI 0x13	Giải phóng vùng nhớ trên Heap		
SWI 0x66	Mở một file dữ liệu. Chế độ mở được lưu trên R1: - 0: Mở để đọc - 1: Mở để ghi - 2: Mở để ghi tiếp	R0: lưu tên file R1: lưu chế độ mở file	R0: Kết quả quá trình mở file. Nếu file không mở được, R0 = -1.
SWI 0x68	Đóng file	R0: lưu tên file	
SWI 0x69	Viết ghi chuỗi ký ra file hoặc ra màn hình	R0: tên file hoặc Stdout R1: Địa chỉ chuỗi ký tự	
SWI 0x6a	Đọc một chuỗi ký tự từ một file	R0: tên file R1: địa chỉ đích R2: số byte lớn nhất cần đọc ra	R0: số byte được lưu trữ
SWI 0x6b	Viết số nguyên ra file	R0: tên file R1: số nguyên	
SWI 0x6c	Đọc số nguyên từ file	R0: tên file	R0: lưu số nguyên
SWI 0x6d	Lấy thời gian của hệ thống		R0: thời gian (tính theo milligiây)

Bảng 3. 2 Một số số hiệu ngắt thông dụng

Ví dụ 1: Viết chương trình hiển thị chuỗi ký tự "Hello world" ra màn hình.

Giải:

```
.equ swi_stdout,0x02  ; gán số hiệu ngắt với tên ngắt
ldr r0,=TextString    ; nạp địa chỉ chuỗi ký tự vào thanh ghi
swi swi_stdout        ; gọi ngắt
TextString: .asciz "Hello world\n"
```

Ví dụ 2: Mở file và ghi chuỗi ký tự vào file.

Giải:

```
InFileName:   .asciz "Infile1.txt"
Message:      .asciz "Hello there \n"
```

```
.equ SWI_Open,0x66
ldr     r0,=InFileName
mov     r1,#2           @ input mode
swi     SWI_Open

ldr     r1,=Message
swi     0x69
swi     0x68
```

3.10. Lập trình trong chế độ Thumb

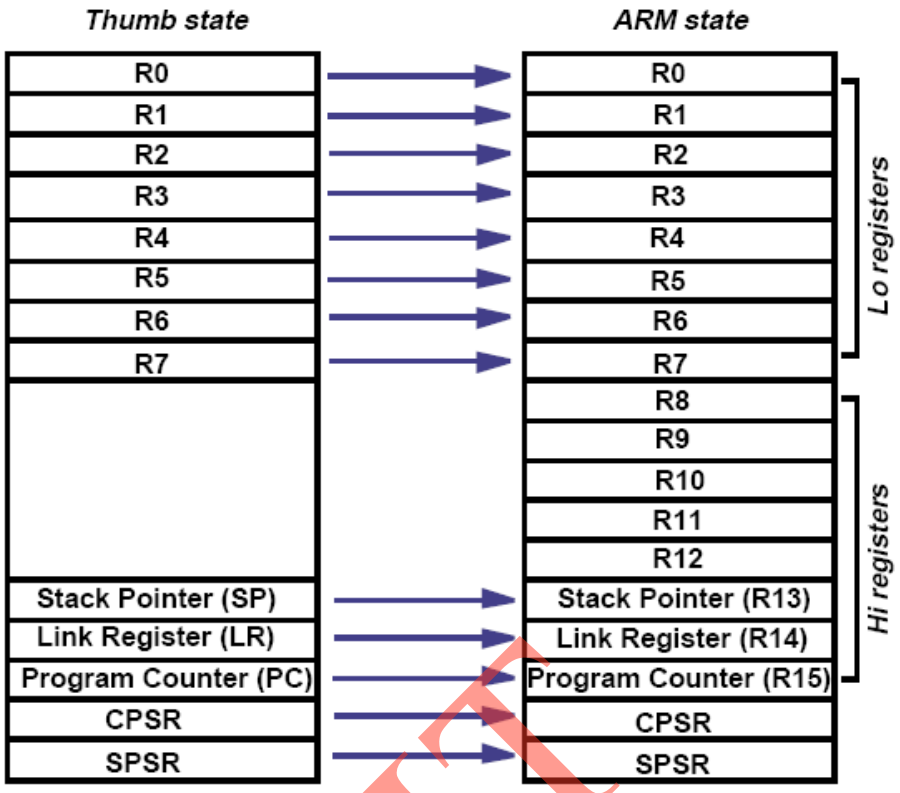
Hầu hết các vi xử lý 32 bit hiện nay đều sử dụng công nghệ RISC. Khác với các bộ vi xử lý sử dụng CISC, bộ vi xử lý RISC thực hiện mỗi lệnh trong một chu kỳ lệnh và do đó tốc độ thực hiện lệnh của RISC nhanh hơn CISC. Tuy nhiên, RISC có một số nhược điểm: Bộ vi xử lý RISC cần bộ nhớ lớn hơn CISC để lưu chương trình. Mặc dù CISC thực hiện 1 lệnh chậm hơn RISC nhưng mỗi lệnh của CISC là ghép của nhiều lệnh nhỏ đơn giản hơn các lệnh của RISC.

Để giảm dung lượng bộ nhớ lưu trữ chương trình, ARM đã tạo ra tập lệnh Thumb 16 bit bên cạnh tập lệnh ARM 32 bit. Vi xử lý thông dụng nhất hiện nay được hỗ trợ tập lệnh Thumb là vi xử lý ARM7TDMI. Chữ cái "T" chính là chữ viết tắt của Thumb.

Tập lệnh Thumb 16 bit hoạt động như một tập lệnh con, rút gọn của tập lệnh ARM 32 bit. Tất cả các lệnh Thumb có chức năng tương đương với các lệnh ARM 32 bit. Tuy nhiên, không phải tất cả các lệnh Arm 32 bit đều có mặt trong tập lệnh Thumb. Ví dụ, tập lệnh Thumb không hỗ trợ truy cập thanh ghi trạng thái hoặc thanh ghi đồng xử lý. Ngoài ra, một số phép toán có thể thực hiện trong một lệnh đơn của ARM nhưng đối với Thumb phải thực hiện trong một vài lệnh. Mặc dù các lệnh Thumb 16 bit nhưng sau khi nạp và biên dịch, bộ vi xử lý sẽ tự động chèn thêm vào mã lệnh các bit để đủ 32 bit. Do vậy, tốc độ xử lý lệnh Thumb và ARM hầu như không khác nhau nhiều. Tuy nhiên, trong một số trường hợp, chế độ Thumb có thể đạt hiệu suất cao hơn do ít sai sót hơn trong khi nạp lệnh vào bộ đệm lệnh (I - cache) so với chế độ ARM.

3.10.1. Tập thanh ghi của Thumb

Khi hoạt động ở chế độ Thumb, các ứng dụng chỉ được sử dụng một số thanh ghi giới hạn cho chế độ này. Hình sau so sánh tập thanh ghi của 2 chế độ ARM và Thumb:



Hình 3. 12 Ánh xạ các thanh ghi của Thumb sang thanh ghi của ARM

Các thanh ghi của Thumb liên quan đến các thanh ghi của ARM như sau:

- Các thanh ghi từ R0 – R7 của Thumb và ARM có chức năng giống nhau;
- Các thanh ghi CPSR và SPSR của Thumb và Arm có chức năng giống nhau;
- Thanh ghi SP của Thumb giống thanh ghi R13 của ARM;
- Thanh ghi LR của Thumb giống thanh ghi R14 của ARM;
- Thanh ghi PC của Thumb giống thanh ghi R15 của ARM;
- Các thanh ghi từ R8 đến R12 bị ẩn trong chế độ Thumb.

3.10.2. Chuyển từ chế độ ARM sang Thumb và ngược lại

Khi vi xử lý đang ở chế độ ARM thì không thể thực hiện các lệnh Thumb và ngược lại, khi vi xử lý ở chế độ Thumb thì không thể thực hiện các lệnh ARM. Do vậy, người lập trình phải lưu ý chuyển sang các chế độ để sử dụng lệnh phù hợp. Có một số cách chuyển đổi từ ARM sang Thumb và ngược lại bằng cách sử dụng lệnh BX hoặc BLX. Khi thực hiện các lệnh này, CPU kiểm tra bit thấp nhất của địa chỉ đích để xác định chế độ làm việc. Nếu bit LSB bằng 1, CPU chuyển từ chế độ ARM sang Thumb trước khi thực hiện các lệnh tại địa chỉ mới. Nếu bit LSB bằng 0, CPU chuyển từ chế độ Thumb về chế độ ARM.

Ví dụ: Chuyển từ chế độ ARM sang Thumb và ngược lại:

```
AREA Vidu, CODE, READONLY
CODE32                  ; Lệnh chỉ dẫn rằng đoạn code tiếp theo ở chế độ ARM
```

```
        ……..                ; Các lệnh thực hiện ở chế độ ARM
        LDR   R0, =Thumb_mode+1; Nạp địa chỉ của nhãn Thumb_mode đồng thời
                                chuyển sang chế độ Thumb.
        BX    R0            ; Chuyển đến nhãn Start
        CODE16              ; Lệnh chỉ dẫn rằng đoạn code tiếp theo ở chế độ Thumb
Thumb_mode
        MOV   R1,#10        ; Các lệnh Thumb
        …….
        ADR   R5, Arm_mode
        BX    R5
        Align
        CODE32
Arm_mode
        ……..                ; Các lệnh ở chế độ ARM
```

3.10.3. Các lệnh trong chế độ Thumb

3.10.3.1. Đặc điểm của tập lệnh Thumb

Về chức năng, tập lệnh Thumb giống như một tập con của tập lệnh ARM. Tất cả các lệnh Thumb có độ dài 16 bit và được căn chỉnh theo định dạng nửa từ (halfword – aligned) trong bộ nhớ. Do vậy, bit có trọng số thấp nhất của địa chỉ lệnh luôn có giá trị 0. Một số lệnh sử dụng bit có trọng số thấp nhất để xác định chế độ làm việc là ARM hay Thumb.

Tất cả các lệnh xử lý dữ liệu của Thumb đều thao tác trên các thanh ghi 32 bit. Ngoài ra, các lệnh này cũng xử lý các địa chỉ 32 bit để truy cập dữ liệu.

Giống như ARM, tập lệnh Thumb có một số khả năng sau:

- Thực hiện lệnh theo điều kiện đặt ra;
- Truy cập thanh ghi;
- Truy cập bộ dịch.

3.10.3.2. Các lệnh trong chế độ Thumb

Lệnh	Mô tả lệnh
ADC	Cộng có nhớ
ADD	Cộng
ADR	Nạp địa chỉ
AND	Phép toán logic AND
ASR	Dịch phải số học
B	Rẽ nhánh
BIC	Xóa bit
BKPT	Điểm ngắt chương trình
BL	Rẽ nhánh và có sử dụng thanh ghi LR
BLX	Rẽ nhánh

CMN, CMP	So sánh âm, so sánh
EOR	Phép toán XOR
LDMIA	Tải dữ liệu vào thanh ghi, tăng địa chỉ sau khi tải dữ liệu
LDR	Tải dữ liệu từ địa chỉ ô nhớ vào thanh ghi
LSL, LSR	Phép toán dịch trái, phải
MOV	Chuyển dữ liệu giữa các thanh ghi
MUL	Nhân
MVN, NEG	Chuyển dữ liệu
ORR	Phép toán logic OR
POP, PUSH	Chuyển dữ liệu giữa ngăn xếp và thanh ghi
ROR	Phép toán quay
SBC	Phép trừ có xét đến cờ nhớ
STMIA	Lưu dữ liệu vào ô nhớ, giá trị ô nhớ tăng sau khi chuyển
STR	Lưu giữ liệu vào ô nhớ
SUB	Phép trừ
SWI	Ngắt mềm
TST	Kiểm tra bit

Bảng 3. 3 Các lệnh trong chế độ Thumb

CHƯƠNG 4. VI ĐIỀU KHIỂN 8051

4.1. Tổng quan về họ vi điều khiển 8051

Vào năm 1976, Intel đã giới thiệu bộ vi điều khiển 8748, một chip tương tự như các bộ vi điều khiển và là chip đầu tiên trong họ vi điều khiển MCS-48. 8748 là vi mạch chứa trên 17.000 transitor bao gồm một CPU, 1K byte EPROM, 64 byte RAM, 27 chân xuất nhập và một bộ định thời 8-bit. IC này và các IC khác tiếp theo của họ MCS-48 đã nhanh chóng trở thành chuẩn công nghiệp trong các ứng dụng hướng điều khiển. Việc thay thế các thành phần cơ điện trong các sản phẩm như các máy giặt và các bộ điều khiển trong xe ôtô, thiết bị công nghiệp, các sản phẩm tiêu dùng và các thiết bị ngoại vi của máy tính,…

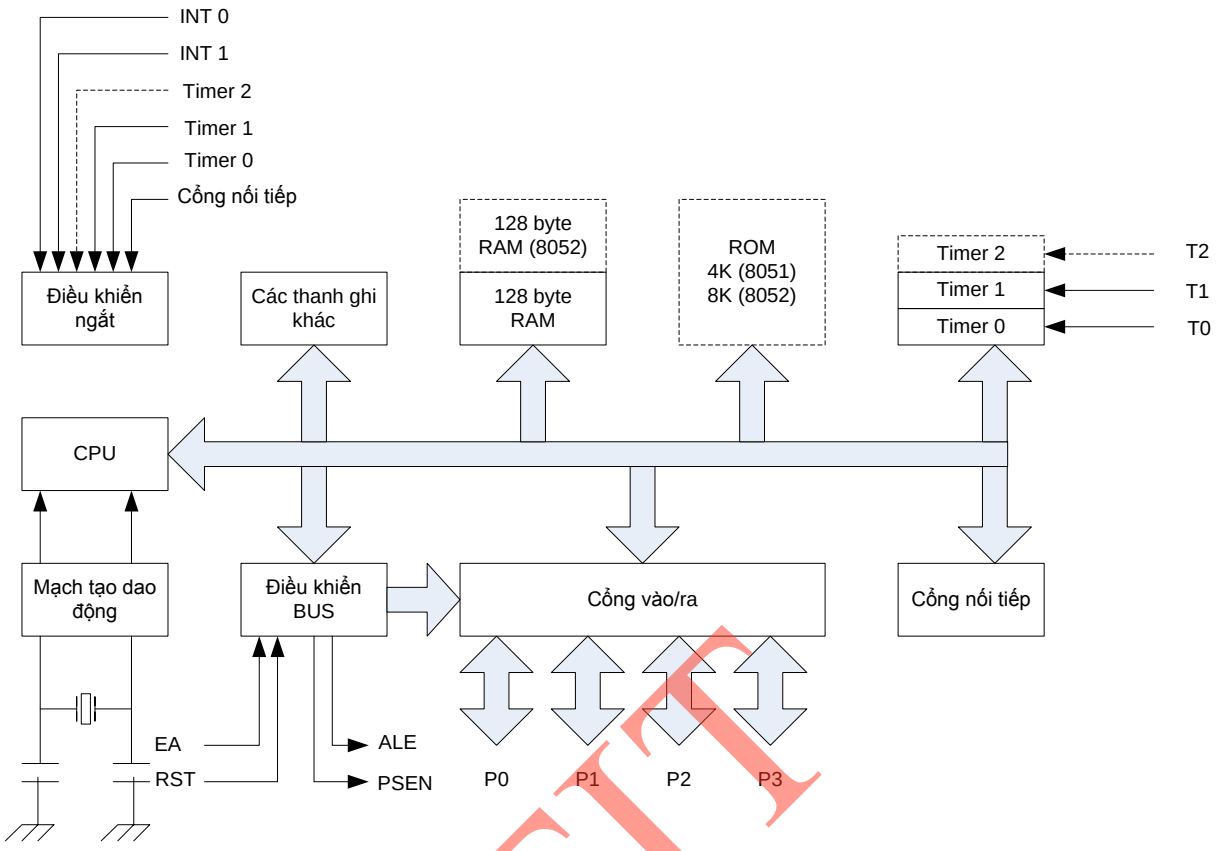
Độ phức tạp, kích thước và khả năng của các bộ vi điều khiển được tăng thêm một bậc quan trọng vào năm 1990 khi Intel công bố chip 8051. So với 8048, chip 8051 chứa trên 60.000 transitor bao gồm 4K byte ROM, 128 byte RAM, 32 đường xuất nhập, 1 port nối tiếp và 2 bộ định thời 16 bit. Tập đoàn Siemens, nguồn sản xuất thứ hai các bộ vi điều khiển thuộc họ MCS-51 cung cấp chip SAB80515, một cải tiến của 8051 chứa trong một vỏ 68 chân, có 6 port xuất nhập 8 bit, 13 nguồn tạo ra ngắt và một bộ biến đổi A/D 8 bit với 8 kênh ngõ vào. Bộ 8051 là một trong những bộ vi điều khiển 8 bit mạnh và linh hoạt nhất, đã trở thành bộ vi điều khiển hàng đầu trong những năm gần đây.

4.2. Cấu trúc tổng quát của vi điều khiển

Sơ đồ khối của một vi điều khiển 8051 có thể được mô tả tổng quát như hình 4.1.

Chức năng của các khối:

1. Khối xử lý trung tâm – CPU (Center Processing Unit) là bộ phận chính của một vi điều khiển. Khối này có các thành phần chính:
   a. Thanh ghi tích lũy (ký hiệu là A)
   b. Thanh ghi tích lũy phụ (ký hiệu là B) thường được dùng cho phép nhân và phép chia.
   c. Khối Logic số học ALU (Arithmetic Logical Unit) thực hiện các thao tác tính toán.
   d. Thanh ghi Từ trạng thái chương trình PSW (Program status word)
   e. Bốn băng thanh ghi
   f. Con trỏ ngăn xếp (Stack point) cũng như con trỏ dữ liệu để định địa chỉ cho bộ nhớ dữ liệu ở bên ngoài.
   g. Thanh ghi đếm chương trình
   h. Bộ giải mã lệnh
   i. Bộ điều khiển thời gian và logic: Sau khi Reset, CPU bắt đầu làm việc tại địa chỉ 0000h, là địa chỉ đầu được ghi trong thanh ghi chứa chương trình (PC). Sau đó, thanh ghi này sẽ tăng lên 1 đơn vị và chỉ đến các lệnh tiếp theo của chương trình.



Hình 4. 1 Cấu trúc của vi điều khiển 8051

2. Bộ tạo dao động: Khối xử lý trung tâm nhận trực tiếp xung nhịp từ bộ tạo dao động được lắp thêm vào. Linh kiện phụ trợ có thể là một khung dao động làm bằng tụ gốm hoặc thạch anh. Ngoài ra, còn có thể đưa một tín hiệu giữ nhịp từ bên ngoài vào.
3. Khối điều khiển ngắt: Chương trình đang chạy có thể cho dựng lại nhờ một khối logic ngắt bên trong. Các nguồn ngắt có thể là các sự kiện ở bên ngoài, sự kiện tràn bộ đếm/bộ định thời hay có thể là truyền thông nối tiếp. Tất cả các ngắt đều có thể được thiết lập chế độ làm việc thông qua hai thanh ghi IE (Interrup Enable) và IP (Interrupt Priority).
4. Khối điều khiển và quản lý bus: Các khối trong vi điều khiển liên lạc với nhau thông qua hệ thống Bus nội bộ được điều khiển bởi khối điều khiển quản lý Bus.
5. Các bộ đếm/định thời: Vi điều khiển 8051 có chứa hai bộ đếm tiến 16 bit có thể hoạt động như một bộ định thời hay bộ đếm sự kiện bên ngoài hoặc như bộ phát tốc độ Baud dùng cho giao diện nối tiếp. Trạng thái tràn bộ đếm có thể được kiểm tra trực tiếp hoặc được xóa đi bằng một ngắt.
6. Các cổng vào ra: Vi điều khiển 8051 có bốn cổng vào/ra (P0 – P3). Mỗi cổng chứa 8 bit độc lập với nhau. Các cổng này có thể được sử dụng cho những mục đích điều khiển rất đa dạng. Ngoài chức năng chung, một số cổng còn đảm nhận thêm một số chức năng đặc biệt khác.

7. Giao diện nối tiếp: Giao diện nối tiếp có chứa một bộ truyền và một bộ nhận không đồng bộ làm việc độc lập với nhau. Bằng cách đấu nối các bộ đệm thích hợp, có thể hình thành một cổng nối tiếp RS-232 đơn giản. Tốc độ truyền qua cổng nối tiếp có thể được thiết lập nhờ bộ định thời và tần số dao động riêng của thạch anh.
8. Bộ nhớ chương trình: Bộ nhớ chương trình thường là bộ nhớ ROM (Read Only Memory). Bộ nhớ chương trình được sử dụng để cất giữ chương trình điều khiển hoạt động của vi điều khiển.
9. Bộ nhớ số liệu: Bộ nhớ số liệu thường là bộ nhớ RAM (Random Access Memory). Bộ nhớ số liệu dùng để cất giữ các thông tin tạm thời trong quá trình vi điều khiển làm việc.

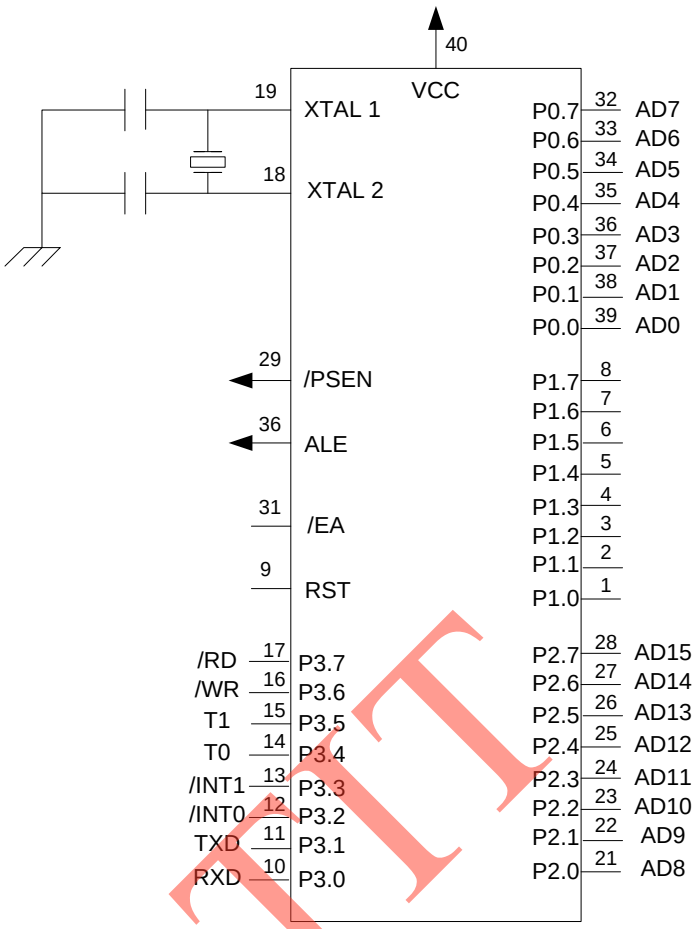
4.3. Sơ đồ và chức năng các chân tín hiệu của VĐK8051

Phần lớn các bộ vi điều khiển 8051 được đóng vào vỏ theo kiểu hai hàng với số chân ra là 40 chân. Một số ít còn lại được đóng vỏ theo kiểu hình vuông với 44 chân và loại này dùng cho những hệ thống cần tiết kiệm diện tích.

- **Port 0:** Port 0 (P0.0 – P0.7) là port hai chức năng trên các chân từ 32 đến 39. Trong các thiết kế cỡ nhỏ (không dùng bộ nhớ ngoài) nó có chức năng như các đường I/O. Đối với các thiết kế lớn với bộ nhớ ngoài, port 0 được dồn kênh giữa bus dữ liệu (D0 – D7) và byte thấp của bus địa chỉ (A0 – A7).
- **Port 1:** Port 1(P1.0 – P1.7) là một port I/O trên các chân từ 1 đến 8. Port 1 không có chức năng khác, nó chỉ được dùng cho giao tiếp với thiết bị ngoài.
- **Port 2:** Port 2 (P2.0 – P2.7) là port có công dụng kép trên các chân từ 21 đến 28. Nó được dùng như các đường I/O hoặc là byte cao của bus địa chỉ (A8 – A15) đối với các thiết kế dùng bộ nhớ ngoài.
- **Port 3:** Port 3 (P3.0 – P3.7) là một port công dụng kép trên các chân từ 10 đến 17. Các chân của port này vừa có chức năng là các đường I/O vừa có chức năng riêng khác tùy từng chân. Bảng sau mô tả các chức năng riêng của từng chân:

Bit	Tên	Chức năng
P3.0	RXD	Dữ liệu nhận cho port nối tiếp
P3.1	TXD	Dữ liệu phát cho port nối tiếp
P3.2	$\overline{INT0}$	Ngắt ngoài 0
P3.3	$\overline{INT1}$	Ngắt ngoài 1
P3.4	T0	Ngõ vào timer/counter 0
P3.5	T1	Ngõ vào timer/counter 1
P3.6	$\overline{WR}$	Xung ghi bộ nhớ dữ liệu ngoài
P3.7	$\overline{RD}$	Xung đọc bộ nhớ dữ liệu ngoài

Bảng 4. 1 Chức năng các chân của Port 3



Hình 4. 2 Sơ đồ chân của vi mạch 8051

- **$\overline{\text{PSEN}}$ (Program Store Enable):**
  - **$\overline{\text{PSEN}}$** là tín hiệu ra trên chân 29. Nó là tín hiệu điều khiển để cho phép đọc bộ nhớ ngoài và thường được nối đến chân $\overline{\text{OE}}$ (Output Enable) của bộ nhớ này.
  - **$\overline{\text{PSEN}}$** sẽ ở mức 0 (mức tích cực) trong thời gian lấy lệnh. Các mã nhị phân của chương trình được đọc từ bộ nhớ qua bus dữ liệu và được chốt vào thanh ghi lệnh của 8051 để giải mã. Khi thực thi chương trình trong ROM nội, **$\overline{\text{PSEN}}$** được duy trì ở mức 1 (mức không tích cực).
- **ALE (Address Latch Enable)**
  - ALE là tín hiệu ra trên chân 30. Nó là tín hiệu ra cho phép chốt địa chỉ để phân kênh cho bus dữ liệu (D0 – D7) và byte thấp của bus địa chỉ (A0 – A7) trên port 0: trong nửa đầu của chu kỳ bộ nhớ, xung ALE cho phép chốt địa chỉ vào một thanh ghi bên ngoài, trong nửa sau của chu kỳ bộ nhớ, các đường port 0 như là các đường xuất/nhập dữ liệu.
  - Các xung tín hiệu ALE có tốc độ bằng 1.6 lần tần số của mạch dao động trên chip và có thể được dùng làm nguồn xung nhịp cho các thành phần khác của hệ thống. Ví dụ,

nếu xung nhịp mạch dao động trên 8051 là 12 MHz thì ALE có tần số là 2 MHz. Chỉ trừ khi thi hành lệnh MOVX, một xung của ALE sẽ bị mất đi. Chân này cũng được làm ngõ vào của xung lập trình cho EPROM trong 8051.

- **$\overline{EA}$ (External Access):** $\overline{EA}$ là tín hiệu vào trên chân 31. Nó thường được nối với +5V (mức 1) hay GND (mức 0). Nếu ở mức 1, 8051 thực thi chương trình từ ROM nội trong khoảng địa chỉ thấp (4 KB). Nếu ở mức 0, 8051 chỉ thực thi chương trình từ bộ nhớ chương trình ngoài.
- **RST (Reset):** RST là ngõ vào trên chân 9. Khi tín hiệu này được đưa lên mức cao (trong ít nhất là 2 chu kỳ máy), hệ thống sẽ khởi động lại.
- **XTAL1 và XTAL2:** XTAL1 và XTAL2 là ngõ vào và ngõ ra của mạch dao động trên chip ở chân 18 và 19. Chúng thường được nối với một thạch anh ngoài và các tụ như hình 4.3 để tạo xung clock. Tần số thạch anh thông thường là 12 MHz.

$$Chu\ kỳ\ máy\ (T_M) = 12/f_{OSC}$$

  Nếu tần số thạch anh là 12 MHz thì chu kỳ máy bằng $1\mu s$.
- **VCC và VSS:** Là các chân nguồn trên chân 40 và 30. V_{CC} =+5V, V_{SS} nối đất.

4.4. Tổ chức bộ nhớ

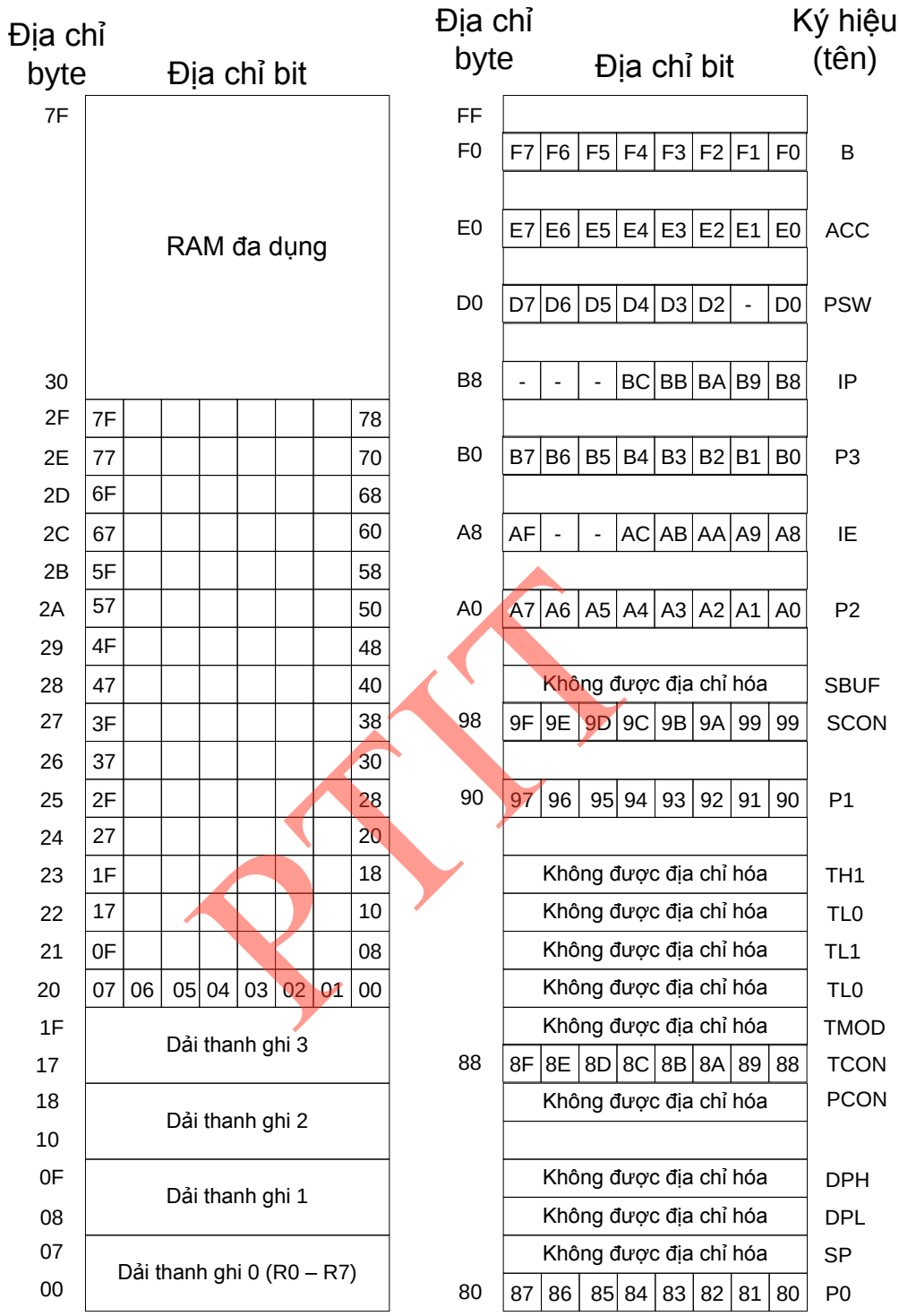
8051 có không gian bộ nhớ riêng bên trong cho chương trình (ROM) và dữ liệu (RAM). Tổ chức bộ nhớ của 8051 cho phép mở rộng bộ nhớ chương trình ngoài đến 64 K và mở rộng bộ nhớ dữ liệu ngoài đến 64 K (tương ứng với 16 bit địa chỉ).

4.4.1. Tổ chức bộ nhớ RAM nội

Bộ nhớ RAM nội của 8051 có dung lượng 256 byte (địa chỉ 00H – FF H) và được chia làm hai phần:

- 128 byte RAM từ địa chỉ 00H đến 7FH là các dãy (bank) thanh ghi, vùng RAM định vị bit và vùng RAM đa dụng.
- 128 byte RAM từ địa chỉ 80H đến FFH là các thanh ghi chức năng đặc biệt.

Cấu trúc của RAM được mô tả trong hình vẽ 4.3 dưới đây:



Hình 4. 3 Tổ chức bên trong RAM nội của 8051

a. Các bank thanh ghi

Có 4 bank thanh ghi 0, 1, 2, 4 nằm tại địa chỉ từ 00H đến 1FH. Mỗi bank có 8 thanh ghi từ R0 – R7. Tại mỗi thời điểm chỉ có một bank thanh ghi tích cực (thông qua việc thiết lập các bit chọn bank thanh ghi trong PSW). Bank thanh ghi tích cực mặc định sau khi reset hệ thống là bank 0.

b. Vùng RAM định vị bit

Vùng RAM định vị bit nằm trong dải địa chỉ 20H – 2FH. Vùng RAM này gồm 128 bit được đánh địa chỉ từ 00H đến 7FH. 8051 cho phép truy xuất vùng RAM này theo từng bit hoặc từng byte.

Ý tưởng truy xuất từng bit riêng rẽ bằng phần mềm là một đặc tính mạnh của các vi điều khiển, đặc biệt đối với các ứng dụng điều khiển. Các bit có thể được đặt, xóa, AND, OR,… với một lệnh đơn.

c. Vùng RAM đa dụng

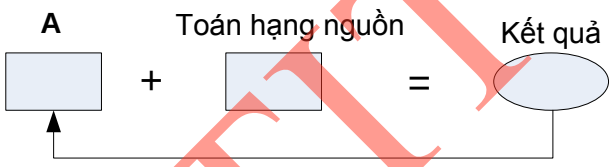
Vùng RAM đa dụng nằm trong dải địa chỉ từ 30H – 7FH và được sử dụng tùy mục đích của người dùng.

4.4.2. Các thanh ghi chức năng đặc biệt

a. Thanh ghi tích lũy A (Accummulator)

Thanh ghi A có địa chỉ E0H và được định địa chỉ từng bít. Thanh ghi có chức năng chứa toán hạng đích trong các lệnh số học và logic, kết quả của lệnh.

Ví dụ:



Hình 4. 4 Chức năng thanh ghi A

b. Thanh ghi trạng thái chương trình PSW (Program Status Word)

Thanh ghi PSW có địa chỉ là D0H và chứa các bit trạng thái như sau:

CY	AC	F0	RS1	RS0	OV	-	P

Hình 4. 5 Thanh ghi trạng thái chương trình

- Cờ nhớ CY (C) có hai chức năng:
  - Được đặt bằng 1 nếu có một số nhớ sinh ra bởi phép cộng hoặc có một số mượn bởi phép trừ. Ví dụ, nếu thanh ghi tích lũy chứa FFH thì lệnh *ADD A, #1* sẽ trả về thanh ghi A kết quả là 00H và đặt cờ nhớ C = 1.
  - Là "thanh ghi tích lũy" 1 bit trong các lệnh logic thao tác trên bit. Ví dụ lệnh *ANL C, 25H* sẽ AND bit 25H với cờ nhớ và đặt kết quả vào cớ nhớ.
- Cờ nhớ phụ AC (Auxiliary Carry):
  - Cờ nhớ AC được đặt bằng 1 nếu có số nhớ sinh ra từ bit 3 sang bit 4 trong phép cộng.

    Ví dụ:

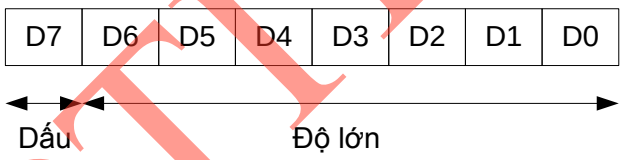
```
  78          0111 1000
+ 69        + 0110 1001
  E1          1110 0001
```

- Được dùng khi cộng các giá trị BCD.

- Cờ F0: là một bit cờ đa dụng, được dùng tùy mục đích của người sử dụng.
- Cờ RS1 và RS0: Là các bit dùng để chọn bank thanh ghi tích cực. Chúng được xóa sau khi reset hệ thống và được thay đổi bằng phần mềm nếu cần.

RS1	RS0	Bank
0	0	0: địa chỉ 00H – 07H
0	1	1: địa chỉ 08H – 0FH
1	0	2: địa chỉ 10H – 17H
1	1	3: địa chỉ 18H – 1FH

- Cờ tràn OV (Overflow):
  - Để biểu diễn số âm, bit cao nhất MSB được sử dụng để biểu diễn bit dấu (+) hoặc (-), còn các bit còn lại được dùng để biểu diễn độ lớn. Dấu được biểu diễn bởi 0 đối với các số dương và biểu diễn bởi 1 đối với số âm (-).



Hình 4. 6 Dữ liệu có dấu

  - Trong các phép toán với số có dấu 8 bit thì cờ OV được bật lên 1 khi xuất hiện một trong hai điều kiện sau:
    - Có nhớ từ D6 sang D7 nhưng không có nhớ ra từ D7 (cờ CY = 0).
    - Có nhớ ra từ D7 (cờ CY = 1) nhưng không có nhớ từ D6 sang D7.

Ví dụ:

Tìm giá trị của cờ OV trong đoạn chương trình sau:

```
MOV   A, #-128      ; A = 1000 0000
MOV   R4, #-2       ; R4 = 1111 1110
ADD   A, R4         ; A = 0111 1110
```

Giải:

```
  -128          1000 0000
+   -2          1111 1110
= -130          0111 1110     và OV = 1
```

Kết quả là +126 là kết quả sai nên cờ OV = 1

Ví dụ:

Tìm giá trị của cờ OV trong đoạn chương trình sau:

```
MOV   A, #7          ; A = 0000 0111
MOV   R1, #18        ; R1 = 0001 0010
ADD   A, R1          ; A = 1111 1001
```

Giải:

```
   7          0000 0111
+  18         0001 0010
=  25         0001 1001      và OV = 0
```

Kết quả là 25 là kết quả đúng nên cờ OV = 0

Từ các ví dụ trên ta có thể kết luận rằng trong bất kỳ phép cộng số có dấu nào, cờ OV đều báo kết quả là đúng hoặc sai. Nếu cờ OV = 1 thì kết quả sai, còn nếu OV = 0 thì kết quả là đúng. Chúng ta có thể nhấn mạnh rằng, trong phép cộng các số không dấu ta phải hiện thị trạng thái của cờ CY (cờ nhớ) và trong phép cộng các số có dấu thì cờ tràn OV phải được quan tâm. Trong 8051 thì các lệnh như JNC và JC cho phép chương trình rẽ nhánh ngay sau phép cộng các số không dấu. Đối với phép cộng có dấu thì sử dụng lệnh JB PSW.2 hoặc JNB PSW.2.

- Cờ kiểm tra chẵn lẻ P (Parity): Được tự động tạo ra theo phương pháp kiểm tra chẵn đối với dữ liệu trong thanh ghi A.
  - Kiểm tra chẵn: P = 0 khi số các bit 1 trong thanh ghi A là chẵn.
  - Kiểm tra lẻ: P = 1 khi số các bit 1 trong thanh ghi A là lẻ.

c. Thanh ghi B

Thanh ghi B ở địa chỉ F0H được dùng chung với thanh ghi A trong các phép toán nhân, chia. Lệnh MUL AB nhân 2 số 8 bit không dấu chứa trong A và B và chứa kết quả 16 bit vào cặp thanh ghi B : A (Thanh ghi A chứa byte thấp và thanh ghi B chứa byte cao).

Lệnh chia DIV AB chia A bởi B, thương số cất trong thanh ghi A và số dư cất trong thanh ghi B. Thành ghi B còn được xử lý như một thanh ghi nháp. Các bit được định địa chỉ của thanh ghi B có địa chỉ từ F0H đến F7H.

d. Con trỏ ngăn xếp SP (Stack Pointer)

Ngăn xếp là vùng nhớ trên RAM mà 8051 dùng để lưu thông tin tạm thời.

Con trỏ ngăn xếp SP có địa chỉ 81H là thanh ghi chứa địa chỉ của byte dữ liệu trên đỉnh của ngăn xếp.

Các lệnh trên stack bao gồm các thao tác cất dữ liệu vào stack và lấy dữ liệu ra khỏi vùng stack. Lệnh cất dữ liệu vào stact (PUSH) sẽ làm tăng nội dung SP trước khi ghi dữ liệu và lệnh

lấy dữ liệu ra khỏi vùng stack (POP) sẽ đọc dữ liệu và giảm nội dung SP. Việc cất và lấy dữ liệu trên vùng stack tuân theo nguyên tắc First In Last Out (FILO).

Vùng stack của 8051 được giữ trong RAM nội. Khi khởi động, nội dung mặc định của SP là 07H. Do đó vùng stack là 08H – 7FH. Trong trường hợp này, bank thanh ghi 1(có thể cả 2 và 3) sẽ không dùng được vì vùng RAM này đã được dùng làm stack. Vì vậy, cần khởi động lại giá trị cho SP, thường lấy vùng nhớ phía trên từ 60H – 7FH.

Trước khi thực hiện một chương trình con hoặc chương trình phục vụ ngắt, địa chỉ của lệnh kế tiếp của lệnh hiện hành sẽ được tự động lưu vào vùng stack. Sau khi thực thi xong và thoát khỏi chương trình con hoặc chương trình phục vụ ngắt, địa chỉ của lệnh kế tiếp được lưu trong vùng stack sẽ được đưa vào PC và chương trình sẽ tiếp tục thực thi tại điểm mà nó đã tạm dừng trước đó. Trong trường hợp có nhiều chương trình con lồng nhau hoặc ngắt trong ngắt, quy tắc FILO sẽ đảm bảo cho việc thực thi chương trình theo đúng trình tự.

e. Thanh ghi con trỏ dữ liệu DPTR (Data Pointer)

Là thanh ghi 16 bit, gồm hai thanh ghi 8 bit là DPL (byte thấp) và DPH (byte cao).

Được dùng để xác định địa chỉ bộ nhớ ngoài (bộ nhớ chương trình ngoài hay bộ nhớ dữ liệu ngoài).

f. Các thanh ghi port

Gồm 4 thanh ghi tương ứng với 4 port:

P0 ⇔ Port 0: địa chỉ 80H

P1 ⇔ Port 1: địa chỉ 90H

P2 ⇔ Port 2: địa chỉ A0H

P3 ⇔ Port 3: địa chỉ B0H

Để truy xuất port, ta truy xuất các thanh ghi port tương ứng. Các thanh ghi này được định địa chỉ từng bít.

Ví dụ:

P1.0: bit 0 của thanh ghi P1

P2.7: bit 7 của thanh ghi P2

g. Các thanh ghi bộ định thời (Timer)

8051 có hai bộ định thời/đếm 16 bit được dùng cho việc định thời hoặc đếm sự kiện:

- Timer 0 gồm TL0 (byte thấp) ở địa chỉ 8AH và TH0 (byte cao) ở địa chỉ 8BH.
- Timer 1 gồm TL1 (byte thấp) ở địa chỉ 8CH và TH1 (byte cao) ở địa chỉ 8DH.

Việc vận hành timer được điều khiển bởi thanh ghi chế độ timer TMOD (địa chỉ 89H) và thanh ghi điều khiển timer TCON (địa chỉ 88H).

h. Các thanh ghi port nối tiếp (Serial port)

8051 chứa một port nối tiếp trên chip dành cho việc trao đổi thông tin với các thiết bị nối tiếp như máy tính, modem hoặc cho việc giao tiếp với các IC khác có giao tiếp nối tiếp (các bộ chuyển đổi A/D, các thanh ghi dịch,…).

Thanh ghi SBUF (Serial Buffer) ở địa chỉ 99H là bộ đệm nhập/xuất nối tiếp. Khi xuất dữ liệu thì ghi lên SBUF, khi nhập dữ liệu thì đọc từ SBUF.

Các chế độ hoạt động khác nhau của port nối tiếp được lập trình thông qua thanh ghi điều khiển port nối tiếp SCON (Serial Control) ở địa chỉ 98H. Đây là thanh ghi được định địa chỉ từng bit.

i. Các thanh ghi ngắt (Interrupt)

8051 có 5 nguồn ngắt, 2 mức ưu tiên ngắt.

Các ngắt bị cấm sau khi reset hệ thống và sẽ được cho phép bằng cách lập trình cho thanh ghi cho phép ngắt IE (Interrupt Enable) ở địa chỉ A8H. Đây là thanh ghi định địa chỉ từng bit.

Việc xác lập chế độ ưu tiên ngắt được lập trình thông qua thanh ghi ưu tiên ngắt IP (Interrupt Priority).

j. Thanh ghi điều khiển nguồn PCON (Power Control)

Không được định địa chỉ từng bit.

Chứa các bit điều khiển như sau:

SMOD	-	-	-	GF1	GF2	PD	IDL

Hình 4. 7 Thanh ghi PCON

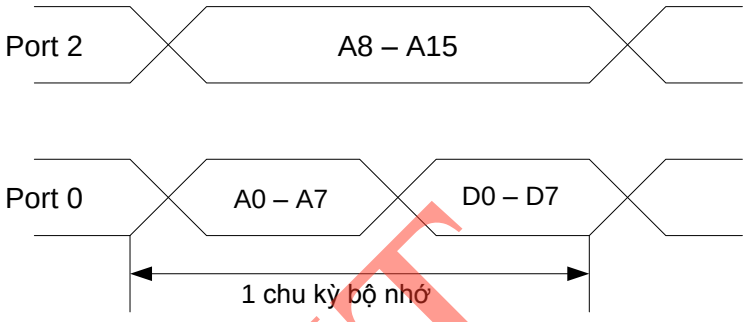
- SMOD: bit tăng gấp đôi tốc độ baud của port nối tiếp nếu được thiết lập (SMOD = 1)
- GF1, GF0: các bit cờ đa dụng.
- PD: Thiết lập chế độ nguồn giảm khi được đặt, chỉ thoát khi reset.
- IDL: Thiết lập chế độ nguồn nghỉ khi được đặt, chỉ thoát nếu có ngắt hoặc reset.
  - Chế độ nguồn giảm (PD = 1): mức điện áp 2V
    - Mạch dao động trên chip ngừng hoạt động.
    - Mọi chức năng ngừng hoạt động.
    - Nội dung các RAM trên chip được duy trì.
    - Các chân port được duy trìn ở mức logic của chúng.
    - ALE và $\overline{PSEN}$ được giữ ở mức thấp.
  - Chế độ nghỉ (IDL = 1)
    - Tín hiệu clock nội khóa không cho đến CPU nhưng không khóa đối với các chức năng ngắt, định thời và port nối tiếp.
    - Nội dung của tất cả các thanh ghi được duy trì.
    - ALE và $\overline{PSEN}$ được giữ ở mức thấp.

4.4.3. Truy xuất bộ nhớ ngoài

8051 có khả năng mở rộng bộ nhớ lên đến 64K cho bộ nhớ chương trình ngoài và 64 K cho bộ nhớ dữ liệu ngoài. Do đó 8051 có thể được ghép nối thêm với ROM, RAM và các IC giao tiếp ngoại vi như 74573, 74244, 74245,…

Khi dùng bộ nhớ ngoài, port 0 được dồn kênh giữa bus địa chỉ (A0 – A7) và bus dữ liệu (D0 – D7), port 2 thường dùng làm byte cao của bus địa chỉ (A8 – A15).

Ngõ ra ALE chốt byte thấp của địa chỉ ở mỗi nửa đầu chu kỳ bộ nhớ (nửa sau chu kỳ bộ nhớ port 0 được dùng làm bus dữ liệu).

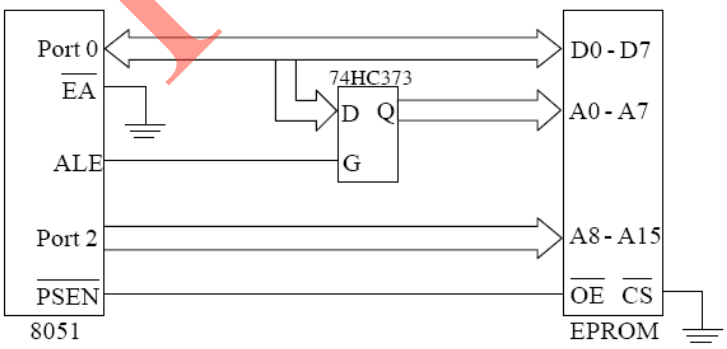


Hình 4. 8 Hoán chuyển chức năng của cổng P0

a. Truy xuất bộ nhớ chương trình ngoài

- Tín hiệu $\overline{EA}$ được tích cực ($\overline{EA} = 0$)
- Tín hiệu $\overline{PSEN}$ nối với $\overline{OE}$ để cho phép đọc bộ nhớ chương trình ngoài.

Ví dụ: Kết nối phần cứng 8051 với bộ nhớ ngoài EPROM 64 K như sau:



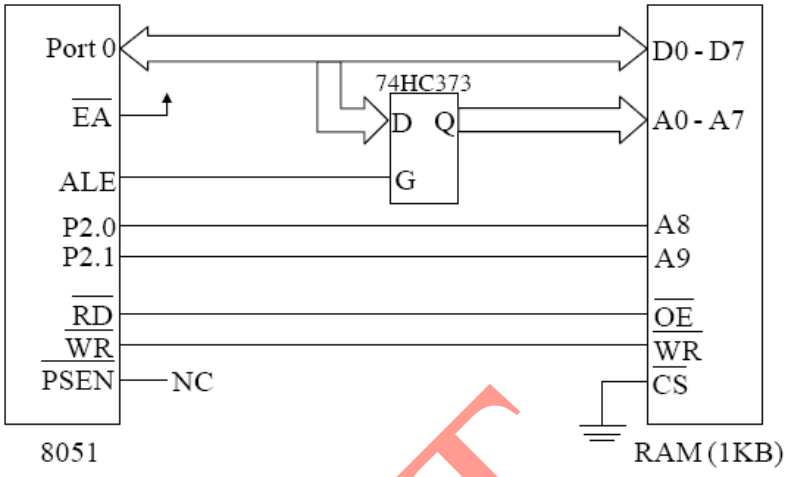
Hình 4. 9 Truy xuất bộ nhớ chương trình ngoài

b. Truy xuất bộ nhớ dữ liệu ngoài

- Cho phép đọc/ghi bởi các tín hiệu $\overline{RD}/\overline{WR}$
- Lệnh dùng để truy xuất bộ nhớ dữ liệu ngoài là MOVX, sử dụng DPTR hay R0/R1 để chứa địa chỉ dữ liệu.

- RAM ngoài có thể giao tiếp với 8051 theo cùng cách như EPROM ngoài trừ đường $\overline{RD}$ nối với đường cho phép xuất $\overline{OE}$ và đường $\overline{WR}$ nới với đường ghi $\overline{WR}$ của RAM.
- Trong trường hợp chỉ có một lượng nhỏ bộ nhớ dữ liệu ngoài (không có bộ nhớ chương trình ngoài) thì có thể dùng địa chỉ 8 bit để tạo trang bộ nhớ 256 byte.

  Ví dụ: Giao tiếp giữa 8051 và RAM 1K ngoài được kết nối như sau:



Hình 4. 10 Truy xuất bộ nhớ dữ liệu ngoài

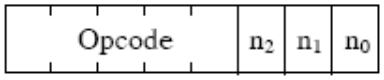
4.5. Các chế độ định địa chỉ của VĐK 8051

Kiểu định địa chỉ là phần cần thiết cho toàn bộ tập lệnh của mỗi một bộ vi xử lý hay vi điều khiển. Các kiểu định địa chỉ cho phép xác định rõ nguồn và đích của dữ liệu theo nhiều cách khác nhau mà vi xử lý hay vi điều khiển sử dụng trong quá trình thực thi lệnh. Có 5 kiểu định địa chỉ đối với 8051:

- Định địa chỉ thanh ghi (register)
- Định địa chỉ trực tiếp (direct)
- Định địa chỉ gián tiếp (indirect)
- Định địa chỉ tức thời (immediate)
- Định địa chỉ chỉ số (index)

4.5.1. Định địa chỉ thanh ghi

- 8051 cho phép truy xuất 8 thanh ghi được đánh số từ R0 đến R7. Các lệnh này được mã hóa dài 1 byte, trong đó dùng 3 bit thấp nhất để chỉ thanh ghi được truy xuất:



Hình 4. 11 Các bit của thanh ghi

Ngoài ra, trong lệnh cũng có thể truy xuất đến các thanh ghi đặc biệt như: thanh ghi tích lũy (A), con trỏ dữ liệu (DPTR), bộ đếm chương trình (PC), cờ nhớ (C) và cặp thanh ghi

AB. Các lệnh này không cần các bit địa chỉ, ban thân opcode của lệnh đã chỉ ra thanh ghi được dùng.

Ví dụ: INC R1 : Tăng nội dung thanh ghi R1 lên 1
INC A : Tăng nội dung thanh ghi A lên 1
INC DPTR : Tăng nội dung thanh ghi DPTR lên 1

4.5.2. Định địa chỉ tức thời

- Khi toán hạng nguồn là một hằng số thay vì là một biến, hằng số này có thể đưa vào lệnh và đây là byte dữ liệu tức thời.
- Trong hợp ngữ, các toán hạng tức thời được nhận biết nhờ vào ký tự # đặt trước chúng. Toán hạng này có thể là một hằng số học, một biến hoặc một biểu thức số học sử dụng các hằng số, các ký hiệu và các toán tử. Trình dịch hợp ngữ tính giá trị và thay thế dữ liệu tức thời vào trong lệnh.

  Ví dụ: MOV A, #12 : nạp giá trị 12 (OCH) vào thanh ghi A.
- Tất cả các lệnh sử dụng kiểu định địa chỉ tức thời đều sử dụng hằng dữ liệu 8 bit làm dữ liệu tức thời. Có một ngoại lệ khi ta khởi động con trỏ dữ liệu 16 bit DPTR, hằng địa chỉ 16 bit được cần đến.

  Ví dụ: MOV DPTR, #8000H: Nạp hằng địa chỉ 8000H vào con trỏ dữ liệu DPTR.

4.5.3. Định địa chỉ trực tiếp

- Kiểu định địa chỉ trực tiếp được sử dụng để truy xuất các ô nhớ trong RAM nội (địa chỉ từ 00H – 7FH) hoặc các thanh ghi chức năng đặc biệt (địa chỉ từ 80H – FFH) thông qua địa địa chỉ trực tiếp của ô nhớ hoặc thanh ghi.
- Ví dụ:
  - MOV P0, A : Chuyển nội dung của thanh ghi A vào cổng Port 0. P0 có địa chỉ 80H nên lệnh này tương đương với lệnh MOV 80H, A.
  - INC 30H : Tăng nội dung ô nhớ 30H lên 1. Giả sử nội dung ô nhớ 30H ban đầu là 06H, sau lệnh trên nội dung ô nhớ là 07H.

4.5.4. Định địa chỉ gián tiếp

- Trong kiểu định địa chỉ này, các thanh ghi R0 và R1 hoạt động như các con trỏ (pointer) và nội dung của chúng chỉ ra địa chỉ trong RAM, nơi dữ liệu được đọc hay được ghi. Trong hợp ngữ của 8051, kiểu định địa chỉ gián tiếp sử dụng ký tự @ đặt trước R0 hoặc R1.

  Ví dụ: R1 chứa 40H và địa chỉ 40H của RAM nội chứa 55H.
  Lệnh MOV A, @R1 : nạp 55H cho thanh ghi A.
- Kiểu định địa chỉ này thường được sử dụng khi duyệt các vị trí liên tiếp trong bộ nhớ.

  Ví dụ: Thực hiện xóa RAM nội tuần tự từ địa chỉ 60H đến 7FH:

```
        MOV R0, #60H      : Khởi động R0 với nội dung là 60H.
LOOP:   MOV @R0, #0       : Xóa nội dung tại địa chỉ con trỏ R0 chỉ tới.
```

INC R0 : Tăng nội dung R0 lên 1.

CJNR R0, #80H, LOOP : Lặp lại bước 2 nếu R0 < 80H.

4.5.5. Định địa chỉ chỉ số

- Dùng một địa chỉ cơ sở (chứa trong thanh ghi PC hoặc DPTR) và một offset (chứa trong thanh ghi A) để tạo địa chỉ được tác động cho các lệnh JMP hoặc MOVC

  Địa chỉ được tác động = (PC) hoặc (DPTR) + (A)
- Thường dùng khi truy xuất dữ liệu trong một bảng dữ liệu đã được định nghĩa trước. Khi đó, thanh ghi PC hay DPTR sẽ lưu địa chỉ đầu bảng và thanh ghi A lưu địa chỉ offset của dữ liệu cần truy xuất trong bảng.

4.6. Khung chương trình hợp ngữ 8051

4.6.1. Khuôn dạng của chương trình hợp ngữ

Một chương trình hợp ngữ có thể bao gồm:

- Các lệnh (instruction) của vi điều khiển
- Các chỉ dẫn (direction) của trình dịch hợp ngữ
- Các điều khiển (control) của trình dịch hợp ngữ
- Các chú thích (comment)

Các lệnh là các mã gợi nhớ quen thuộc và sẽ được dịch ra mã máy tương ứng với vi điều khiển. Các chỉ dẫn của trình dịch hợp ngữ là các lệnh của trình dịch hợp ngữ dùng để định nghĩa cấu trúc chương trình, các ký hiệu, dữ liệu, các hằng số,… Các điều khiển của trình dịch hợp ngữ thiết lập các chế độ trình dịch hợp ngữ và các luồng hợp dịch trực tiếp. Các chú thích giúp chương trình dễ đọc bằng cách đưa ra các giải thích về mục đích và hoạt động của ác chuỗi lệnh.

a. Khuôn dạng của dòng lệnh:

Các dòng chứa các lệnh và các chỉ dẫn phải được viết theo các qui luật mà trình dịch hợp ngữ hiểu được. Mỗi dòng được chia thành các trường cách biệt nhau bởi khoảng trắng hay khoảng tab. Khuôn dạng tổng quát của mỗi dòng như sau:

Tên (nhãn) ***Mã gợi nhớ*** ***Các toán hạng*** ***Chú thích***

Với trình dịch hợp ngữ, trường mã gợi nhớ không cần ở trên cùng một dòng với trường nhãn. Tuy nhiên, trường toán hạng phải ở trên cùng một dòng với trường mã gợi nhớ. Có thể viết các dòng này bằng chữ hoa hoặc chữ thường.

Trường tên:

- Trường này có thể chứa các nhãn, tên biến, hay tên chương trình con. Các tên và nhãn này sẽ được trình dịch hợp ngữ gán bằng các địa chỉ cụ thể của lệnh (hoặc dữ liệu) theo sau.
- Tên và nhãn có thể có độ dài từ 1 đến 31 ký tự, không chứa khoảng trắng, phải bắt đầu bằng ký tự, dấu “?” hay dấu “_” và tiếp theo phải là các ký tự chữ, ký tự số, dấu “?” hoặc dấu “_”.Tên nhãn kết thúc bằng dấu “ : ”
- Các tên và nhãn không được trùng với các từ khóa (các mã gợi nhớ, các chỉ dẫn, các toán tử hay các kiểu định nghĩa trước).

Trường mã gợi nhớ:

- Trường mã gợi nhớ (mnemonic) cho biết chức năng của lệnh (ví dụ như ADD, MOV, DIV, MUL, INC,…) hay chỉ dẫn của trình dịch hợp ngữ (ví dụ như ORG, END, EQU, DB,…).
- Lưu ý: các chỉ dẫn không được dịch ra mã máy.

Trường toán hạng (operand)

- Trường này chứa địa chỉ hay dữ liệu mà lệnh sẽ sử dụng. Tùy theo từng loại lệnh mà có thể có 0, 1, 2 hay 3 toán hạng.
- Các toán hạng cách nhau bởi dấu phảy.

Trường chú thích (comment):

- Các chú thích để làm rõ chương trình được đặt ở cuối dòng lệnh. Điều này giúp cho người đọc chương trình dễ hiểu các thao tác của chương trình hơn.
- Các chú thích cần phải được bắt đầu bằng dấu " ; ". Các chú thích có thể chiếm nhiều dòng riêng và cũng phải bắt đầu bằng dấu " ; ".

b. Một số chỉ dẫn của trình dịch hợp ngữ:

- **ORG**

  Dạng: ORG *biểu thức*

  Chỉ dẫn ORG thay đổi nội dung bộ đếm chương trình theo giá trị của *biểu thức* để thiết lập nơi bắt đầu mới của chương trình cho các phát biểu theo sau nó.

- **END**

  Dạng: END

  END là phát biểu cuối cùng của chương trình nguồn.

- **EQU**

  Dạng: *kí hiệu* *EQU* *biểu thức*

  Chỉ dẫn EQU gán giá trị của *biểu thức* cho *kí hiệu*. *Kí hiệu* phải là tên hợp lệ.

- **DB:**

  Dạng: *nhãn:* *DB* *biểu thức*

  Chỉ dẫn DB thường được dùng để định nghĩa các giá trị byte tương ứng với các *biểu thức* trong bộ nhớ chương trình bắt đầu từ địa chỉ tương ứng với *nhãn*.

4.6.2. Biên dịch chương trình

Chương trình phải được chuyển sang thành dạng object code trước khi vi điều khiển có thể thực hiện chương trình. Quá trình chuyển từ chương trình dạng source code sang object code được gọi là biên dịch/hợp dịch (assembling). Sau đó ta nạp object code này vào bộ nhớ vi điều khiển và vi điều khiển chạy chương trình.

Việc chuyển đổi mnemonic sang object code thường thực hiện bằng máy tính. Trước hết ta dùng một chương trình soạn thảo quen thuộc của Window như Notepad để viết chương trình. Sau đó ta chạy chương trình biên dịch gọi là assembler để biên dịch chương trình sang tập tin dưới

dạng object code. Cuối cùng, ta dùng một chương trình khác để nạp object code từ bộ nhớ của máy tính vào bộ nhớ của vi điều khiển.

4.6.3. Cấu trúc một chương trình hợp ngữ

ORG *địa chỉ bắt đầu của chương trình*

…..

<Đoạn chương trình chính>

….

<Các chương trình con>

….

END

Ví dụ:

```
ORG 00H
MOV R0, #20;
ACALL LOOP1          ; Gọi chương trình con

LOOP1:
    DJNZ R1,LOOP1  ; Nhảy đến nhãn LOOP1 cho đến khi R1 = 0;
RET                          ; Quay trở lại chương trình chính.

END
```


4.7. Tập lệnh của vi điều khiển 8051

Tập lệnh của 8051 được chia thành 5 nhóm:

- Nhóm lệnh chuyển dữ liệu
- Nhóm lệnh số học
- Nhóm lệnh logic
- Nhóm lệnh rẽ nhánh
- Nhóm lệnh xử lý bit

4.7.1. Nhóm lệnh chuyển số liệu

MOV

- Lệnh di chuyển dữ liệu có nhiều dạng phụ thuộc vào nguồn và đích của dữ liệu. Lệnh di chuyển dữ liệu không làm thay đổi dữ liệu mà chỉ copy dữ liệu từ nguồn tới đích. Các ví dụ về lệnh MOV đã được đề cập đến trong mục 4.5

MOVC: Lệnh truy xuất dữ liệu từ ROM nội

Như đã nói ở mục 4.5.4, R0 và R1 là các thanh ghi duy nhất có thể được dùng làm con trỏ trong chế độ đánh địa chỉ gián tiếp thanh ghi. Vì R0 và R1 là các thanh ghi 8 bit nên việc sử dụng

của chúng bị hạn chế ở việc truy cập mọi thông tin trong ngăn nhớ RAM nội (các ngăn nhớ từ 30H đến 7FH và các thanh ghi đặc biệt). Tuy nhiên, nhiều khi ta cần truy cập dữ liệu trong RAM ngoài hoặc trong không gian mã lệnh của ROM nội thì ta cần sử dụng thanh ghi 16 bit đó là DPTR. Lệnh được dùng cho mục đích này là "MOVC A,@A+DPTR" (chữ C ở cuối chỉ mã lệnh Code).

- Cú pháp: MovC A,@A+DPTR
- Công dụng: Chuyển dữ liệu từ bộ nhớ ROM có địa chỉ bằng giá trị của A cộng với DPTR vào thanh ghi A

XCH: Lệnh trao đổi dữ liệu

- Cú pháp: XCH A,direct
- Công dụng: Trao đổi dữ liệu của thanh ghi A với ô nhớ có địa chỉ direct, tức là sau khi thực hiện lệnh ô nhớ có địa chỉ direct mang dữ liệu của thanh ghi A trước đó và thanh ghi A mang dữ liệu của ô nhớ có địa chỉ direct.

Ví dụ:
```
Mov  A,#0FAH
Mov  50H,#60H
XCH A,50H
```
Kết quả:
A mang giá trị là 60H
50H mang giá trị là 0FAH

XCHD: Lệnh trao đổi dữ liệu 4 bit

- Cú pháp: XCHD A,@Ri
- Công dụng: Trao đổi dữ liệu của 4 bit thấp ở thanh ghi A với dữ liệu của 4 bit thấp ở ô nhớ có địa chỉ bằng giá trị lưu giữ trong thanh ghi Ri

4.7.2. Nhóm lệnh số học

ADD: Lệnh cộng

- Cú pháp: Add A,Rn
- Công dụng: Cộng giá trị dữ liệu trên thanh ghi A với giá trị dữ liệu trên thanh ghi Rn, sau khi thực hiện lệnh kết quả được lưu ở thanh ghi A. Lệnh này có ảnh hưởng đến thanh thanh trạng thái PSW

Ví dụ 1:
```
Mov  A,#20H
Mov  R1,#08H
Add  A,R1
```
Kết quả:
A có giá trị là 28H
R1 vẫn giữ nguyên giá trị là 08H
Cờ C = 0

Ví dụ 2:
```
Mov  A,#0E9H
```

Mov R6,#0BAH

Add A,R6

Kết quả: A = #0A3h

R6 = #0BAh

Cờ C = 1

ADDC: Lệnh cộng có xét đến cờ nhớ C

- Cú pháp: AddC A,Rn
- Công dụng: Cộng giá trị dữ liệu trên thanh ghi A với giá trị dữ liệu trên thanh ghi Rn và cộng thêm giá trị của số nhớ trên cờ C, sau khi thực hiện lệnh kết quả được lưu ở thanh ghi A. Lệnh này có ảnh hưởng đến thanh thanh trạng thái PSW

Ví dụ: C = 1

Mov A,#08h

Mov R1,#10h

Addc A,R1

Kết quả: A = #19h ;cộng cả cờ C

R1 = #10h

C = 0

SUBB: Lệnh trừ có xét đến cờ nhớ C

- Cú pháp: SubB A,Rn
- Công dụng: Trừ giá trị dữ liệu trên thanh ghi A với giá trị dữ liệu trên thanh ghi Rn và trừ cho giá trị nhớ trên cờ C, sau khi thực hiện lệnh kết quả được lưu ở thanh ghi A. Lệnh này có ảnh hưởng đến thanh thanh trạng thái PSW

Ví dụ: C = 1

Mov A,#0E5h

Mov R3,#9Fh

Subb A,R3

Kết quả: A = 45h

C = 0

INC: Lệnh tăng

- Cú pháp: Inc A
- Công dụng: Tăng giá trị dữ liệu lưu giữ trên thanh ghi A lên 1 đơn vị, không ảnh hưởng đến các cờ nhớ trên PSW

Ví dụ: Mov A,#05h

Inc A

Kết quả: A = #06h

DEC: Lệnh giảm

- Cú pháp: Dec A

- Công dụng: Giảm giá trị dữ liệu lưu giữ trên thanh ghi A xuống 1 đơn vị, không ảnh hưởng đến các cờ nhớ trên PSW

Ví dụ: Mov A,#05h
Dec A

Kết quả: A = #04h

MUL: Lệnh nhân

- Cú pháp: Mul AB
- Công dụng: Nhân hai dữ liệu là số nguyên không dấu ở thanh ghi A với thanh ghi B, kết quả là một dữ liệu 16 bit. Byte thấp của kết quả lưu ở thanh ghi A và byte cao của kết quả lưu ở thanh ghi B. Nếu tích số lớn hơn 255(0FFH), cờ tràn OV ở thanh trạng thái PSW được thiết lập lên 1, ngược lại nếu tích số nhỏ hơn 255(0FFH), cờ tràn OV được thiết lấp về 0. Cờ nhớ C luôn ở giá trị 0.

Ví dụ: Mov A,#0B9h
Mov B,#F7h
Mul AB

Kết quả: A = #7Fh
B = #0B2h

DIV: Lệnh chia

- Cú pháp: Div AB
- Công dụng: Chia hai dữ liệu là số nguyên không dấu ở thanh ghi A với thanh ghi B, dữ liệu ở thanh ghi A là số chia còn ở thanh ghi B là số bị chia, kết quả là một dữ liệu 8 bit được lưu ở thanh ghi A.số dư lưu trữ trong thanh ghi B Cờ nhớ C luôn ở giá trị 0. Cờ tràn OV được thiết lập giá trị 1 khi thanh ghi B mang giá trị là 00H-phép chia không thể thực hiện.

Ví dụ: Mov A,#50h
Mov B,#10h
DIV AB

Kết quả: A = #5h
B = #0h

4.7.3. Nhóm lệnh logic

ANL: Lệnh And logic

- Cú pháp: ANL A,Rn
- Công dụng: thực hiện phép logic AND dữ liệu ở thanh ghi A với dữ liệu ở thanh ghi Rn, kết quả được lưu trữ ở thanh ghi A

Ví dụ:

mov A,#0Fh
mov R1,#0F0h

ANL A,R1

Kết quả: A = #0H

ORL: Lệnh OR logic

- Cú pháp: ORL A,Rn
- Công dụng: thực hiện phép logic OR dữ liệu ở thanh ghi A với dữ liệu ở thanh ghi Rn, kết quả được lưu trữ ở thanh ghi A
- *Ví dụ:*

```
mov A,#0Fh
mov R1,#0F0h
ORL A,R1
```

Kết quả: A = #0FFh

XRL: Lệnh OR tuyệt đối

- Cú pháp: XRL A,Rn
- Công dụng: thực hiện phép logic EX-OR dữ liệu ở thanh ghi A với dữ liệu ở thanh ghi Rn, kết quả được lưu trữ ở thanh ghi A
- *Ví dụ:*

```
mov A,#0F2h
mov R3,#0E0h
XRL A,R3
```

Kết quả: A = #12h

CPL: Lệnh bù logic

- Cú pháp: CPL A
- Công dụng: lấy bù giá trị lưu giữ ở thanh ghi A, các bit có giá trị là 1 chuyển thành 0 và ngược lại các bit có giá trị là 0 chuyển thành 1.

Ví dụ:

```
mov A,#01100111b ;(tương đương 67h)
CPL A
```

Kết quả: A = #10011000b (tương đương 98h)

CLR: Lệnh xóa dữ liệu

- Cú pháp: CLR A
- Công dụng: tất cả các bit của thanh ghi A đều được xác lập giá trị 0.

Ví dụ:

```
mov A,#01100111b
CLR A
```

Kết quả: A = #0

RL: Lệnh xoay trái dữ liệu

- Cú pháp: RL A
- Công dụng: thanh ghi A gồm tám bit A7 A6 A5 A4 A3 A2 A1 A0. Khi thực hiện lệnh xoay trái RL A giá trị của các bit được chuyển trang bit ở bên trái nó, giá trị của bit A0 chuyển sang bit A1, giá trị của bit A1 chuyển sang bit A2, tương tự với các bit còn lại, và

giá trị của bit A7 chuyển sang bit A0. Minh họa các bit trong thanh ghi A khi thực hiện lệnh như trong hình dưới:

Các bit của thanh ghi A:	A7	A6	A5	A4	A3	A2	A1	A0
Quá trình xoay dữ liệu:		←	←	←	←	←	←	←
Giá trị từ A7 chuyển sang A0:	A7	→						A0

Ví dụ: Mov A,#01001001b
RL A

Kết quả sau khi các lệnh được thực hiện A mang giá trị là 10010010b

RLC: Lệnh xoay trái dữ liệu cùng với cờ nhớ

- Cú pháp: RLC A
- Công dụng: thanh ghi A gồm tám bit A7 A6 A5 A4 A3 A2 A1 A0. Khi thực hiện lệnh xoay trái A với cờ nhớ RLC A giá trị của các bit được chuyển trang bit ở bên trái nó, giá trị của bit A0 chuyển sang bit A1, giá trị của bit A1 chuyển sang bit A2, tương tự với các bit còn lại, và giá trị của bit A7 chuyển sang cờ nhớ C, giá trị ở cờ nhớ C chuyển sang bit A0.

Các bit của thanh ghi A & C:	C	A7	A6	A5	A4	A3	A2	A1	A0
Quá trình xoay dữ liệu:		←	←	←	←	←	←	←	←
Giá trị từ cờ chuyển sang A0:	C	→							A0

Ví dụ: giả sử cờ nhớ C đang mang giá trị 1
Mov A,#11001001b
RLC A

Kết quả sau khi các lệnh được thực hiện A mang giá trị là 10010011b và C mang giá trị 1

RR: Lệnh xoay phải dữ liệu

- Cú pháp: RR A
- Công dụng: thanh ghi A gồm tám bit A7 A6 A5 A4 A3 A2 A1 A0. Khi thực hiện lệnh xoay phải RR A giá trị của các bit được chuyển trang bit ở bên phải nó, giá trị của bit A7 chuyển sang bit A6, giá trị của bit A6 chuyển sang bit A5, tương tự với các bit còn lại, và giá trị của bit A0 chuyển sang bit A7. Minh họa các bit trong thanh ghi A khi thực hiện lệnh như trong hình dưới.

Các bit của thanh ghi A:	A7	A6	A5	A4	A3	A2	A1	A0
Quá trình xoay dữ liệu:	→	→	→	→	→	→	→	
Giá trị từ A0 chuyển sang A7:	A7	←						A0

Ví dụ: Mov A,#01001001b
RL A

Kết quả sau khi các lệnh được thực hiện A mang giá trị là 10100100b

RRC: Lệnh xoay phải dữ liệu cùng với cờ nhớ

- Cú pháp: RRC A
- Công dụng: thanh ghi A gồm tám bit A7 A6 A5 A4 A3 A2 A1 A0. Khi thực hiện lệnh xoay phải A với cờ nhớ -RRC A -giá trị của các bit được chuyển trang bit ở bên phải nó, giá trị của bit A7 chuyển sang bit A6, giá trị của bit A6 chuyển sang bit A5, tương tự với các bit còn lại, và giá trị của bit A0 chuyển sang cờ nhớ C, giá trị ở cờ nhớ C chuyển sang bit A7

Các bit của thanh ghi A & C:	C	A7	A6	A5	A4	A3	A2	A1	A0
Quá trình xoay dữ liệu:	→	→	→	→	→	→	→	→	
Giá trị từ A0 chuyển sang cờ:	C	←							A0

Ví dụ: giả sử cờ nhớ C đang mang giá trị 1

```
Mov   A,#11001001b
RLC   A
```

Kết quả sau khi các lệnh được thực hiện A mang giá trị là 11100100b và C mang giá trị 1

SWAP: Lệnh xoay 4 bit dữ liệu

- Cú pháp: SWAP A
- Công dụng: hoán chuyển dữ liệu ở 4 bit thấp lên 4 bit cao và 4 bit cao xuống 4 bit thấp

Các bit của thanh ghi A:	A7	A6	A5	A4	A3	A2	A1	A0
Dữ liệu trước khi xoay:	X7	X6	X5	X4	X3	X2	X1	X0
Dữ liệu sau khi xoay:	X3	X2	X1	X0	X7	X6	X5	X4

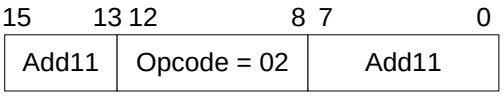
Ví dụ:
```
mov   A,#0E7h
SWAP  A
```
Kết quả : A = # 7Eh

4.7.4. Nhóm lệnh rẽ nhánh

- **Các lệnh gọi hàm**

ACALL (Absolute Call): Lệnh gọi tuyệt đối

- Cú pháp: ACALL addr11
- Công dụng: Khi lệnh được thực hiện, vi điều khiển chuyển về thực hiện các câu lệnh của chương trình con bắt đầu từ địa chỉ *addr11* trên ROM. Địa chỉ *addr11* có thể thay bằng nhãn bắt đầu của một chương trình con. Câu lệnh được thực hiện khi địa chỉ *addr11* cách lệnh gọi không quá 2 KB.
- Mã lệnh:



Hình 4. 12 Mã lệnh của ACALL

- Khi thực hiện lệnh ACALL, vi điều khiển thực hiện các bước sau:

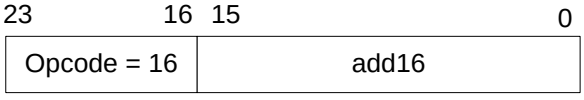
```
PC = PC + 3              ; Con trỏ lệnh trỏ tới lệnh tiếp theo
SP = SP + 1              ; Con trỏ ngăn xếp trỏ tới ô nhớ tiếp theo
(SP) = PC[7-0]           ; Byte thấp của PC được lưu vào ngăn xếp
SP = SP + 1              ; Con trỏ ngăn xếp tăng lên 1 đơn vị
(SP) = PC[15 – 8]        ; Ba bit cao của byte cao được lưu vào ngăn xếp
PC[10-0] = add11[10-0]   ; 11 bit thấp của PC có giá trị bằng add11.
```

Ví dụ:

Dòng	Giá trị PC	Mã lệnh (Opcode)	Chương trình
04	0004	7100	ACALL DELAY
05	0006	74AA	MOV A, #0AAH
			…
11	0300		ORG 300H
12	0300	7DFF	DELAY: MOV R5,#0FFH
			…
15	0304	22	RET

LCALL (Long Call): Lệnh gọi xa

- Cú pháp: LCALL addr16
- Công dụng: Khi lệnh được thực hiện, vi điều khiển chuyển về thực hiện các câu lệnh của chương trình con bắt đầu từ địa chỉ *addr16* trên ROM. Địa chỉ *addr16* có thể thay bằng nhãn bắt đầu của một chương trình con. Câu lệnh được thực hiện khi địa chỉ *addr16* cách lệnh gọi không quá 64 KByte .
- Mã lệnh:



Hình 4. 13 Mã lệnh của lệnh LCALL

- Khi thực hiện lệnh LCALL, vi điều khiển thực hiện các bước sau:

```
PC = PC + 3              ; Con trỏ PC trỏ tới lệnh ngay sau lệnh LCALL
SP = SP + 1              ; Con trỏ ngăn xếp trỏ tới ô nhớ tiếp theo
(SP) = PC [7-0]          ; Byte thấp của PC được lưu vào ngăn xếp
SP = SP + 1              ; Con trỏ ngăn xếp trỏ tới ô nhớ tiếp theo
```

(SP) = PC[15 -8] ; Byte cao của PC được lưu vào ngăn xếp

PC = addr16 ; Con trỏ PC nhảy tới địa chỉ addr16

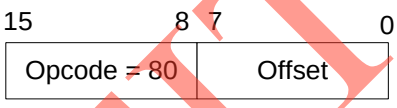
RET: Lệnh kết thúc chương trình con

- Cú pháp: RET
- Công dụng: Lệnh này dùng kết thúc chương trình con, khi gặp lệnh này vi điều khiển quay về thực hiện lệnh ở chương trình chính.

- **Các lệnh nhảy không điều kiện:**

SJMP (Short Jump): Lệnh nhảy ngắn

- Cú pháp: SJMP offset
- Công dụng: Lệnh SJMP thực hiện nhảy đến địa chỉ của offset. Địa chỉ của offset được tính bằng cách cộng byte cuối của lệnh (lệnh SJMP dài 2 byte) với địa chỉ của tiếp theo lệnh SJMP. Như vậy lệnh này cho phép nhảy lùi lại 128 byte hoặc nhảy tiến lên 127 byte kể từ địa chỉ của lệnh tiếp theo lệnh SJMP.
- Mã lệnh:



Hình 4. 14 Mã lệnh của lệnh SJMP

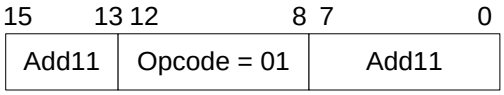
- Khi thực hiện lệnh SJMP, vi điều khiển thực hiện một số thao tác sau:

PC = PC + 2 ; Con trỏ lệnh trỏ tới lệnh ngay sau lệnh SJMP

PC = PC + offset ; Địa chỉ nhảy tới bằng địa chỉ của lệnh tiếp theo cộng với khoảng cách nhảy.

AJMP (Absolute Jump): Lệnh nhảy tuyệt đối

- Cú pháp: AJMP add11
- Công dụng: Tương tự lệnh ACALL
- Mã lệnh:



Hình 4. 15 Mã lệnh của lệnh AJMP

- Khi thực hiện lệnh AJMP, vi điều khiển thực hiện một số thao tác sau:

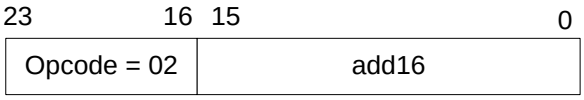
PC = PC + 2

PC[10-0] = Add11[10-0]

LJMP(Long Jump): Lệnh nhảy xa

- Cú pháp: LJMP add16

- Công dụng: Lệnh thực hiện nhảy tới địa chỉ dài 16 bit. Mã của lệnh dài 3 byte trong đó byte cao nhất chứa mã lệnh, 2 byte thấp chứa địa chỉ của lệnh được nhảy tới. Như vậy, khoảng cách nhảy của lệnh trong phạm vi 64KB.
- Mã lệnh:



Hình 4. 16 Mã lệnh của lệnh LJMP

- Khi thực hiện lệnh LJMP, vi điều khiển thực hiện:

  PC = add16

- **Các lệnh nhảy có điều kiện:**

JZ: Lệnh nhảy thuận với cờ Zero

- Cú pháp: JZ nhãn
- Công dụng:
  - Nếu cờ Zero có giá trị 1(tức thanh ghi A có giá trị 0), vi điều khiển sẽ nhảy đến thực hiện chương trình tại địa chỉ mà nhãn được đặt
  - Nếu cờ Zero có giá trị 0 (tức thanh ghi A có giá trị khác 0), vi điều khiển thực hiện lệnh kế tiếp (không thực hiện lệnh nhảy)

  Ví dụ:

```
        MOV  A,R0      ; Nạp giá trị của R0 vào A
        JZ   OVER      ; Nhảy đến OVER nếu A = 0
        MOV  A,R1      ; Nạp giá trị R1 vào A
OVER:   …..            ; Nhãn OVER
```

  Trong chương trình này, nếu R0 = 0 thì nó nhảy đến địa chỉ có nhãn OVER. Nếu A khác 0 thì thực hiện lệnh tiếp theo. Lưu ý rằng lệnh JZ chỉ có thể được sử dụng đối với thanh ghi A.

JNZ: Lệnh nhảy nghịch với cờ Zero

- Cú pháp: JNZ nhãn
- Công dụng:
  - Nếu cờ Zero có giá trị 0 (tức thanh ghi A có giá trị 0), vi điều khiển sẽ nhảy đến thực hiện chương trình tại địa chỉ mà nhãn được đặt.
  - Nếu cờ Zero có giá trị 1 (tức thanh ghi A có giá trị khác 0), vi điều khiển thực hiện lệnh kế tiếp (không thực hiện lệnh nhảy).

JC: Lệnh nhảy thuận với cờ nhớ CY

- Cú pháp: JC nhãn
- Công dụng:

- Nếu cờ C có giá trị 1, vi điều khiển sẽ nhảy đến thực hiện chương trình tại địa chỉ mà nhãn được đặt.
- Nếu cờ C có giá trị 0, vi điều khiển thực hiện lệnh kế tiếp (không thực hiện lệnh nhảy).

JNC: Lệnh nhảy nghịch với cờ nhớ CY

- Cú pháp: JNC nhãn
- Công dụng:
  - Nếu cờ C có giá trị 0, vi điều khiển sẽ nhảy đến thực hiện chương trình tại địa chỉ mà nhãn được đặt.
  - Nếu cờ C có giá trị 1, vi điều khiển thực hiện lệnh kế tiếp (không thực hiện lệnh nhảy).

Ví dụ: Tính tổng của các giá trị 79H, F5H và E2H. Đặt giá trị byte thấp của kết quả vào R0, byte cao vào R2.

Giải:

```
        MOV A,#0        ;Xóa thanh ghi A
        MOV R2,A        ;Xóa R2
        ADD  A,#79H     ;Cộng 79H vào A (A = 0 + 79H = 79H)
        ADD  A,#0F5H    ;A=79H + F5H = 6EH và CY = 1
        JNC  N1         ;Nếu không tràn thì nhảy đến nhãn N1
        INC  R2         ;Nếu tràn thì tăng R2
N1:     ADD  A,#E2H     ;A = 6EH + E2H = 50H và CY =1
        JNC  N2         ;Nếu không tràn thì nhảy đến nhãn N2
        INC  R2         ;Nếu tràn thì tăng R2
N2:     MOV  R0,A       ;Lưu byte thấp vào R0, R2 = 2
```

JB: Kiểm tra giá trị của bit

- Cú pháp: JB bit,nhãn
- Công dụng:
  - Nếu bit có giá trị 1, vi điều khiển sẽ nhảy đến thực hiện chương trình tại địa chỉ mà nhãn được đặt.
  - Nếu bit có giá trị 0, vi điều khiển thực hiện lệnh kế tiếp (không thực hiện lệnh nhảy).

Ví dụ: JB P3.5, LABEL ;Nhảy đến nhãn LABEL nếu chân P3.5 = 1

JNB: Kiểm tra giá trị của bit

- Cú pháp: JNB bit,nhãn
- Công dụng:
  - Nếu bit có giá trị 0, vi điều khiển sẽ nhảy đến thực hiện chương trình tại địa chỉ mà nhãn được đặt.
  - Nếu bit có giá trị 1, vi điều khiển thực hiện lệnh kế tiếp (không thực hiện lệnh nhảy).

Ví dụ: Viết chương trình kiểm tra xem thanh ghi A có chứa số chẵn không? Nếu có thì chia nó cho 2, nếu không thì tăng thêm 1 đơn vị sau đó chia cho 2

Giải:

```
                MOV   B,#2              ;Gán B = 2
                JNB   ACC.0, LABEL      ;Nhảy đến nhãn nếu bit cuối bằng 0
                INC   ACC
        LABEL:  DIV   AB                ;Chia A cho 2
```

JBC: Kiểm tra giá trị của bit và xóa nó

- Cú pháp: JBC bit,nhãn
- Công dụng:
  - Nếu bit có giá trị 1, vi điều khiển sẽ nhảy đến thực hiện chương trình tại địa chỉ mà nhãn được đặt, đồng thời xóa giá trị chứa trong bit đó tức là đưa bit đó về giá trị 0.
  - Nếu bit có giá trị 0, vi điều khiển thực hiện lệnh kế tiếp (không thực hiện lệnh nhảy).

Các lệnh so sánh và nhảy đến nhãn:

CJNE: So sánh và nhảy nếu không bằng

- Cú pháp: CJNE toán hạng 1,toán hạng 2,nhãn
- Công dụng:
  - Vi điều khiển nhảy đến thực hiện chương trình tại địa chỉ mà nhãn được đặt nếu toán hạng 1 khác toán hạng 2, nếu bằng nhau vi điều khiển không nhảy và thực hiện lệnh kế.
  - Ảnh hưởng của lệnh đến cờ nhớ C:
    Nếu giá trị của toán hạng 1 > toán hạng 2 thì bit C có giá trị 0.
    Nếu giá trị của toán hạng 1 < toán hạng 2 thì bit C có giá trị 1.

Ví dụ: So sánh R1 với 10:

Nếu R1 = 10: Xóa chân P1.1
Nếu R1 <10: Xóa chân P1.2
Nếu R1>10: Xóa chân P1.3

Giải:

```
                CJNE  R1,#10, Not_equal   ; Kiểm tra R1 có bằng 10 hay không?
                CLR   P1.1                ; Nếu R1 = 10: xóa chân P1.1
                SJMP  Exit
 Not_equal:     JNC   Greater             ; R1 > 10  -> C=0
                CLR   P1.2                ; Nếu R1 < 10: xóa chân P1.2
                SJMP  Exit
 Greater:       CLR   P1.3                ; Nếu R1 > 10: xóa chân P1.3
 Exit:          END
```

DJNZ: Giảm và nhảy nếu A = 0

- Cú pháp: DJNZ toán hạng,nhãn
- Công dụng:
  - Giảm giá trị của toán hạng xuống 1 đơn vị, và
    Nếu giá trị của toán hạng khác 0, vi điều khiển nhảy đến thực hiện chương trình tại địa chỉ mà nhãn được đặt.
    Nếu giá trị toán hạng bằng 0, vi điều khiển thực hiện lệnh kế tiếp.
  - Toán hạng có thể là thanh ghi (R0 – R7) hoặc địa chỉ ô nhớ. Trong trường hợp toán hạng là địa chỉ ô nhớ thì giá trị của ô nhớ sẽ được giảm đi 1 đơn vị khi thực hiện lệnh.

Ví dụ: Viết chương trình tính tổng của các giá trị được lưu trong các ô nhớ có địa chỉ liên tiếp từ 40H đến 44H. Byte thấp của kết quả lưu vào thanh ghi A, byte cao lưu vào R7.

Giải:

```
        MOV   A,#0        ; Xóa thanh ghi A
        MOV   R7,A        ; Xóa thanh ghi R7
        MOV   R1,#5       ; Lưu số lần lặp vào R1
        MOV R2,#40H       ; Lưu địa chỉ ô nhớ đầu tiên vào R2
LOOP:   ADD   A,@R2       ; A = A + @40H
        JNC   LABEL       ; Nếu không tràn thì nhảy đến nhãn
        INC   R7          ; Nếu tràn thì lưu giá trị nhớ vào R7.
LABEL:  INC   R2          ; Tăng địa chỉ của ô nhớ
        DJNZ  R1,LOOP     ; Nếu R1 khác 0 thì nhảy đến nhãn LOOP
```

4.7.5. Nhóm lệnh xử lý bit

CLR: Lệnh xóa bit

- Cú pháp: CLR bit
- Công dụng: Xóa giá trị của bit có địa chỉ xác định – tức là đưa giá trị bit đó về 0.

SETB: Lệnh thiết lập bit

- Cú pháp: SETB bit
- Công dụng: Thiết lập giá trị của bit có địa chỉ xác định – tức là đưa giá trị bit đó lên 1.

CPL: Lệnh bù bit

- Cú pháp: CPL bit
- Công dụng: đổi giá trị của bit có địa chỉ xác định, nếu trước đó bit đó có giá trị 0 chuyển thành 1, và ngược lại nếu trước đó bit đó có giá trị 1 chuyển thành 0.

ANL: Thực hiện phép AND với bit và lưu vào CY

- Cú pháp: ANL C,bit
- Công dụng: Thực hiện phép And cờ nhớ C và bit có địa chỉ xác định, kết quả lưu ở CY.

ORL: Thực hiện phép OR với bit và lưu vào CY

- Cú pháp: ORL C,bit
- Công dụng: Thực hiện phép And cờ nhớ C và bit có địa chỉ xác định, kết quả lưu ở CY.

MOV: Sao chép trạng thái bit vào cờ nhớ CY

- Cú pháp: MOV C,bit
- Công dụng: Thực hiện chuyển giá trị của bit có địa chỉ xác định vào cờ nhớ CY.

CHƯƠNG 5. BỘ ĐẾM/ĐỊNH THỜI VÀ UART TRONG 8051

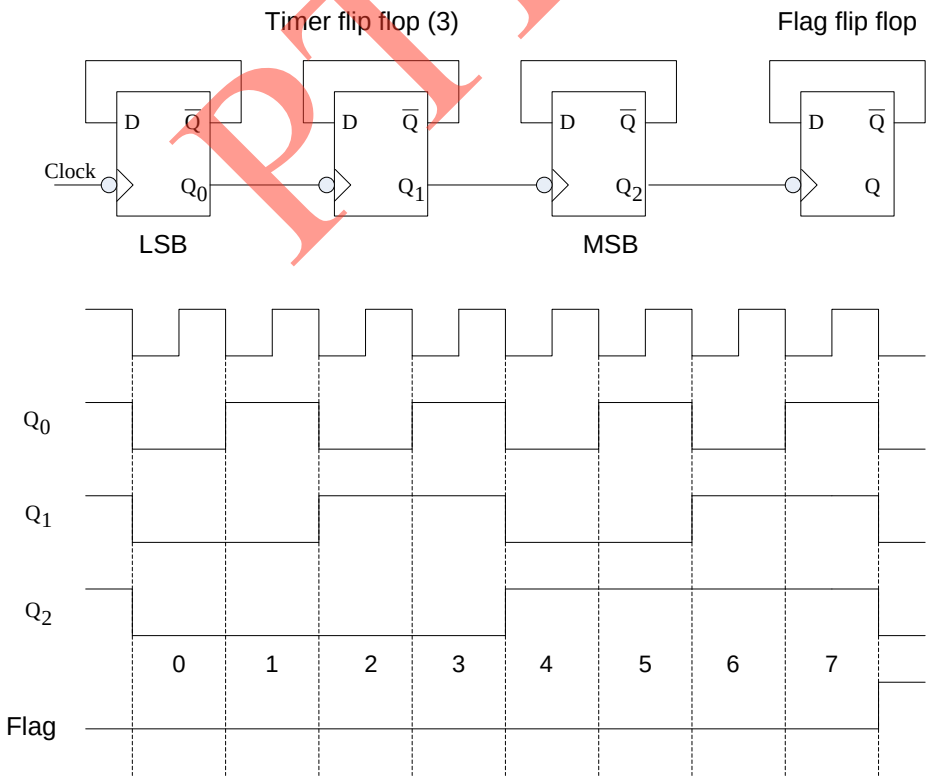
5.1. Giới thiệu

Bộ đếm/Bộ định thời: Đây là các ngoại vi được thiết kế để thực hiện một nhiệm vụ đơn giản: đếm các xung nhịp. Mỗi khi có thêm một xung nhịp tại đầu vào đếm thì giá trị của bộ đếm sẽ được tăng lên 01 đơn vị (trong chế độ đếm tiến/đếm lên) hay giảm đi 01 đơn vị (trong chế độ đếm lùi/đếm xuống).

Xung nhịp đưa vào đếm có thể là một trong hai loại:

- Xung nhịp bên trong IC: Đó là xung nhịp được tạo ra nhờ kết hợp mạch dao động bên trong IC và các linh kiện phụ bên ngoài nối với IC. Trong trường hợp sử dụng xung nhịp loại này, người ta gọi là các bộ định thời (timers). Do xung nhịp bên loại này thường đều đặn nên ta có thể dùng để đếm thời gian một cách khá chính xác.
- Xung nhịp bên ngoài IC: Đó là các tín hiệu logic thay đổi liên tục giữa 02 mức 0-1 và không nhất thiết phải là đều đặn. Trong trường hợp này người ta gọi là các bộ đếm (counters). Ứng dụng phổ biến của các bộ đếm là đếm các sự kiện bên ngoài như đếm các sản phẩm chạy trên băng chuyền, đếm xe ra/vào kho bãi…

8051 có 02 bộ đếm/bộ định thời. Chúng có thể được dùng như các bộ định thời để tạo một bộ trễ thời gian hoặc như các bộ đếm để đếm các sự kiện xảy ra bên ngoài bộ VĐK. Trong chương này chúng ta sẽ tìm hiểu về cách lập trình cho chúng và sử dụng chúng như thế nào.



Hình 5. 1 Cấu tạo bộ đếm/định thời

5.2. Nguyên lý hoạt động cơ bản của bộ định thời

Một bộ định thời là một chuỗi các flipflop với mỗi flipflop là một mạch chia 2. Chuỗi này nhận một tín hiệu ngõ vào làm nguồn xung clock. Xung clock đặt vào flipflop thứ nhất, flipflop này chia đôi tần số xung clock. Ngõ ra của flipflop thứ nhất trở thành nguồn xung clock cho flipflop thứ hai, nguồn xung clock này cũng được chia cho 2, v.v… Vì mỗi một tầng kế tiếp nhau đều chia cho 2 nên một bộ định thời có n tầng sẽ chia tần số xung clock ở ngõ vào của bộ này cho 2^n.

Ngõ ra của tầng cuối cùng làm xung clock cho một flipflop báo tràn bộ định thời hay còn gọi là cờ tràn (overflow flag). Cờ tràn này được kiểm tra bởi phần mềm hoặc tạo ra một ngắt. Giá trị nhị phân trong các flipflop của bộ định thời là số đếm của các xung clock từ khi bộ định thời bắt đầu đếm. Ví dụ, một bộ định thời 16 bit sẽ đếm từ 0000H đến FFFFH. Cờ tràn được thiết lập bằng 1 khi xảy ra tràn số đếm.

Hoạt động của bộ định thời đơn giản được minh họa trong hình 5.1, bộ định thời 3 bit. Mỗi tầng là một DFF (Delay flipflop) hoạt động như một mạch chia cho 2 do ta nối ngõ ra $\bar{Q}$ với ngõ vào D. Flipflop cờ đơn giản là một mạch chốt D được set bằng 1 bởi tầng cuối bộ định thời. Giản đồ thời gian ở hình 5.2 cho thấy tầng thứ nhất (Q_0) chia đôi tần số xung clock, tầng thứ hai chia 4 tần số xung clock,… Số đếm được ghi ở dạng thập phân và được kiểm tra bằng cách khảo sát trạng thái của 3 flipflop. Ví dụ số đếm là 4 xuất hiện khi $Q_2 = 1$, $Q_1 = 0$, $Q_0 = 0$. Các flipflop ở hình 5.1 là các flipflop tác động cạnh âm (nghĩa là ngõ ra Q của các flipflop đổi trạng thái theo cạnh âm của xung clock). Khi số đếm tràn từ 111_2 xuống 000_2, ngõ ra Q_2 có cạnh âm (1 -> 0) làm cho trạng thái của flipflop cờ đổi từ 0 lên 1 (ngõ vào D của flipflop này luôn luôn ở logic 1).

Các bộ định thời được dùng để:

- Định thời trong một khoảng thời gian;
- Đếm sự kiện;
- Tạo tốc độ baud cho port nối tiếp của chip 8051.

Trong các ứng dụng định thời trong một khoảng thời gian, bộ định thời được lập trình sao cho sẽ tràn sau một khoảng thời gian quy định và set cờ tràn của bộ định thời bằng 1. Cờ tràn được sử dụng để đồng bộ chương trình nhằm thực hiện một công việc như là kiểm tra trạng thái của các ngõ nhập hoặc gởi dữ liệu đến các ngõ xuất. Các ứng dụng khác có thể sử dụng xung clock quy định của bộ định thời để đo khoảng thời gian giữa hai sự kiện (ví dụ đo độ rộng xung).

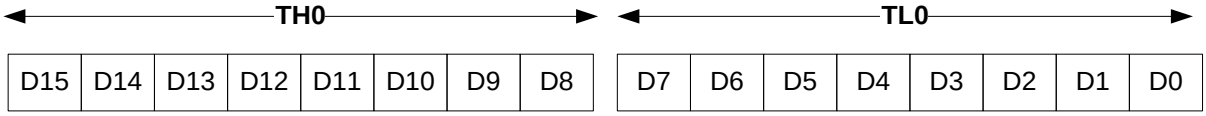
Việc đếm sự kiện được dùng để xác định số lần xuất hiện của một sự kiện hơn là đo thời gian giữa các sự kiện. Từ “sự kiện” là một kích thích bên ngoài cung cấp sự chuyển trạng thái từ 1 xuống 0 tới một chân của 8051. Các bộ định thời cũng có thể cung cấp xung clock tốc độ baud cho port nối tiếp bên trong 8051.

5.3. Các thanh ghi dùng cho bộ đếm/định thời

8051 có hai bộ định thời là Timer 0 và Timer 1. Cả hai bộ định thời đều có độ dài 16 bit được truy cập như hai thanh ghi tách biệt byte thấp và byte cao.

5.3.1. Các thanh ghi của bộ Timer 0

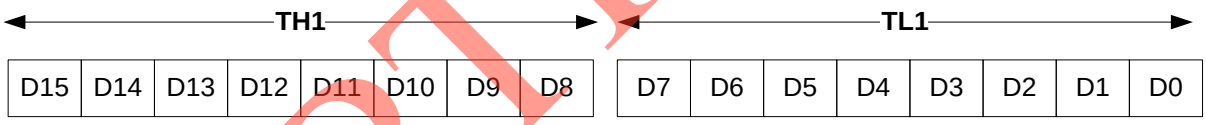
Thanh ghi 16 bit của bộ Timer 0 được truy cập như byte thấp và byte cao. Thanh ghi byte thấp được gọi là TL0 (Timer 0 low byte) và thanh ghi byte cao là TH0 (Timer 0 high byte). Các thanh ghi này có thể được truy cập như các thanh ghi A, B, R0, R1,… Ví dụ, lệnh "MOV TL0,#4FH" là chuyển giá trị 4FH vào TL0. Các thanh ghi này cũng có thể được đọc như các thanh ghi khác. Ví dụ "MOV R5, TH0" là lưu byte cao TH0 của TIMER0 vào R5.



Hình 5. 2 Thanh ghi Timer0

5.3.2. Các thanh ghi của bộ Timer 1

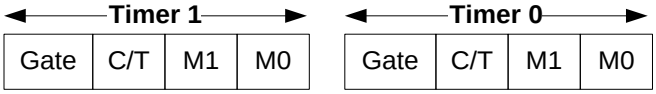
Bộ định thời Timer 1 dài 16 bit và cũng được chia thành hai byte TL1 và TH1. Các thanh ghi này được truy cập và đọc giống như các thanh ghi của bộ Timer 0.



Hình 5. 3 Thanh ghi Timer1

5.3.3. Thanh ghi chế độ định thời (TMOD)

Thanh ghi TMO (timer mode register) chứa hai nhóm 4 bit dùng để thiết lập chế độ hoạt động cho bộ định thời 0 và bộ định thời 1. TMOD không được định địa chỉ từng bit mà được nạp một lần tại thời điểm bắt đầu của chương trình để khởi động chế độ hoạt động của bộ định thời. Sau đó, bộ định thời có thể được bắt đầu, dừng,… bằng cách truy xuất các thanh ghi chức năng đặc biệt khác của bộ định thời.



Hình 5. 4 Thanh ghi TMOD

Chức năng của các bit trong thanh ghi TMOD:

Bit	Tên	Bộ định thời	Mô tả
0	M0	0	Bit chọn chế độ
1	M1	0	Bit chọn chế độ

2	C/$\overline{T}$	0	Bit chọn chực năng đếm hoặc định thời: 1: đếm sự kiện 0: định thời trong một khoảng thời gian
3	GATE	0	Bit điều khiển cổng cho bộ định thời 0. Khi được đặt bằng 1, bộ định thời chỉ hoạt động khi $\overline{INT1}$ ở mức cao.
4	M0	1	Bit chọn chế độ
5	M1	1	Bit chọn chế độ
6	C/$\overline{T}$	1	Bit chọn chức năng đếm hoặc định thời cho Timer 1
7	GATE	1	Bit điều khiển cổng cho Timer 1.

Bảng 5. 1 Các bit trong thanh ghi TMOD

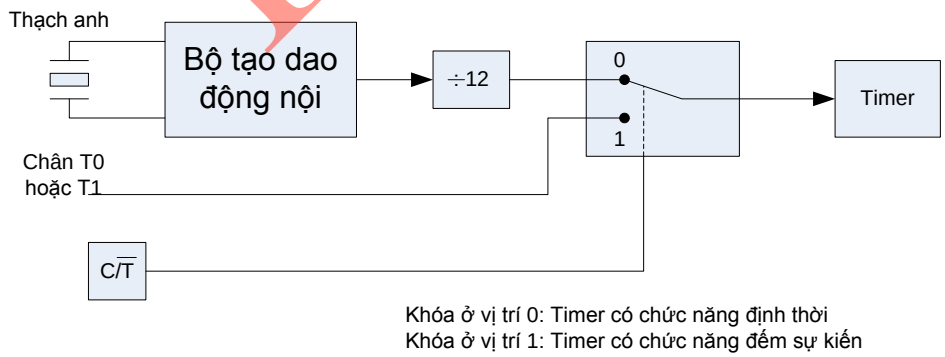
5.3.3.1. Các chế độ của bộ định thời

M0	M1	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 tự động nạp lại 8 bit
1	1	3	Chế độ định thời chia xẻ

Bảng 5. 2 Các chế độ của bộ định thời

5.3.3.2. Nguồn đồng hồ cho bộ định thời

Bit C/$\overline{T}$ trong thanh ghi TMOD được dùng để quyết định xem bộ định thời được dùng như một máy tạo độ trễ hay bộ đếm sự kiện.



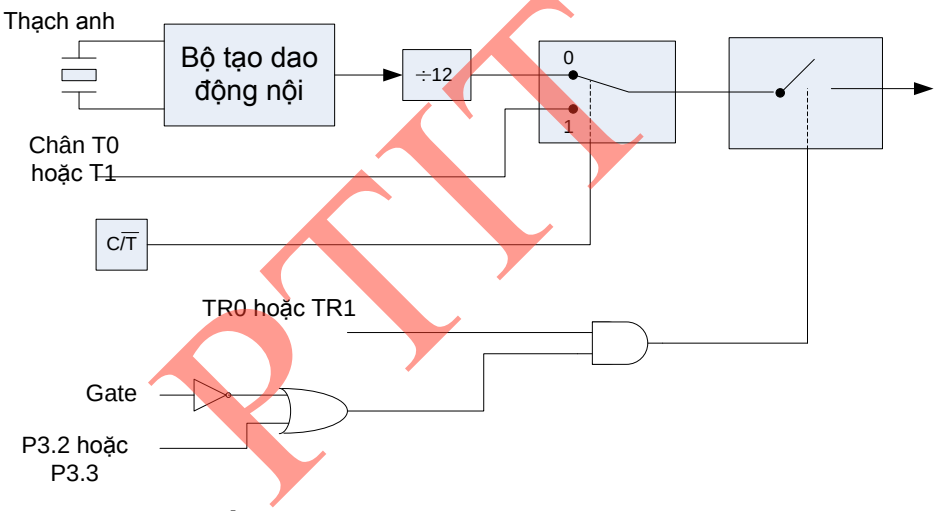
Hình 5. 5 Nguồn đồng hồ cho bộ định thời

Nếu bit C/$\overline{T}$ = 0 thì nó được dùng như một bộ định thời tạo độ trễ thời gian. Nguồn đồng hồ cho chế độ này là tần số thạch anh của 8051 và tần số của bộ định thời luôn bằng 1/12 tần số của thạch anh.

Nếu C/$\overline{T}$ = 1, bộ định thời được dùng như một bộ đếm sự kiện và được cung cấp xung clock từ 1 nguồn tạo xung bên ngoài. Trong các ứng dụng, nguồn xung clock này cung cấp cho bộ định thời một xung dựa trên việc xảy ra một sự kiện. Số các sự kiện được xác định trong phần mềm bằng cách độc các thanh ghi định thời (TLx/THx), giá trị 16 bit trong các thanh ghi này tăng theo mỗi sự kiện. Hai chân của port 3 (P3.4 và P3.5) bây giờ trở thành ngõ vào xung clock cho các bộ định thời. Chân P3.4 là ngõ vào xung clock cho bộ định thời 0 (T0) và chân P3.4 (còn gọi là T1) là ngõ vào xung clock cho bộ định thời 1.

5.3.3.3. Bit cổng GATE

Vi điều khiển 8051 cho phép khởi động và dừng bộ định thời được thực hiện bằng cả phần cứng và phần mềm. Việc khởi động và dừng bằng phần mềm được thực hiện nhờ các lệnh "SETB TRx" và "CLR TRx". Lệnh SETB khởi động bộ định thời và lệnh CLR dùng để dừng nó. Các lệnh này khởi động và dừng các bộ định thời khi bit GATE = 0. Trong trường hợp khởi động và dừng bộ định thời bằng phần cứng từ nguồn ngoài thì bit GATE = 1.



Hình 5. 6 Chức năng của bit GATE

5.3.4. Thanh ghi điều khiển định thời (TCON)

Thanh ghi TCON chứa các bit điều khiển và trạng thái của bộ định thời 0 và bộ định thời 1. Bốn bit cao trong TCON (TCON.4 – TCON.7) được dùng để điều khiển các bộ định thời hoạt động hoặc dừng (TR0, TR1) hoặc để báo các bộ định thời tràn (TF0, TF1). Bốn bit thấp của TCON (TCON.0 – TCON.3) không dùng để điều khiển các bộ định thời, chúng được dùng để phát hiện và khởi động các ngắt ngoài.

Bit	Ký hiệu	Địa chỉ bit	Mô tả
TCON.7	TF1	8FH	Cờ tràn của bộ định thời 1. Cờ này được set bởi phần cứng khi có tràn, được xóa bởi phần mềm hoặc bởi phần cứng khi bộ vi xử lý trỏ đến trình

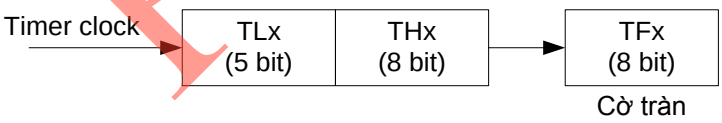
			phục vụ ngắt.
TCON.6	TR1	8EH	Bit điều khiển hoạt động của bộ định thời 1. Bit này được set hoặc được xóa bởi phần mềm để điều khiển bộ định thời hoạt động hay dừng hoạt động.
TCON.5	TR0	8DH	Cờ tràn của bộ định thời 0
TCON.4	TF0	8CH	Bit điều khiển hoạt động của bộ định thời 0.
TCON.3	IE1	8BH	Cờ ngắt bên ngoài 1 (kích bởi cạnh). Cờ này được set bởi phần cứng khi có cạnh âm xuất hiện trên chân $\overline{INT1}$, được xóa bởi phần mềm, hoặc phần cứng khi CPU trỏ đến trình phục vụ ngắt.
TCON.2	IT1	8AH	Cờ ngắt bên ngoài 1 (kích bởi cạnh hoặc mức). Cờ này được set hoặc xóa bởi phần mềm khi xảy ra cạnh âm hoặc mức thấp tại chân ngắt ngoài.
TCON.1	IE0	89H	Cờ ngắt bên ngoài 0
TCON.0	IT0	88H	Cờ ngắt bên ngoài 0

Bảng 5. 3 Các bit trong thanh ghi TCON

5.4. Các chế độ định thời

5.4.1. Chế độ định thời 0

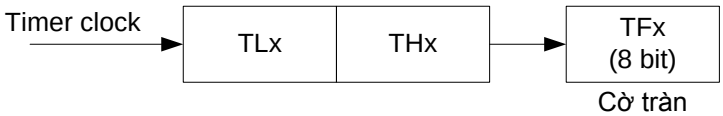
Chế độ 0 là chế độ định thời 13 bit. Trong chế độ này, byte cao của bộ định thời THx ("x" là ký hiệu của 0 hoặc 1) được ghép 5 bit thấp của byte thấp của bộ định thời TLx để tạo thành một bộ định thời 13 bit. Ba bit cao của TLx không sử dụng.



Hình 5. 7 Chế độ 0

5.4.2. Chế độ định thời 1

Chế độ 1 là chế độ định thời 16 bit và có cấu hình giống chế độ định thời 13 bit, chỉ khác nhau ở chỗ bây giờ là bộ định thời 16 bit. Xung clock được đặt vào các thanh ghi định thời cao và thấp kết hợp (TLx/THx). Khi có xung clock đến, bộ định thời đếm lên từ 0000H đến FFFFH. Sự kiện tràn xuất hiện khi có sự chuyển số đếm từ FFFFH xuống 0000H và set cờ tràn bằng 1. Cờ tràn là bit TFx trong thanh ghi điều khiển định thời TCON. Bit này được đọc hoặc ghi bởi phần mềm.



Hình 5. 8 Chế độ 1

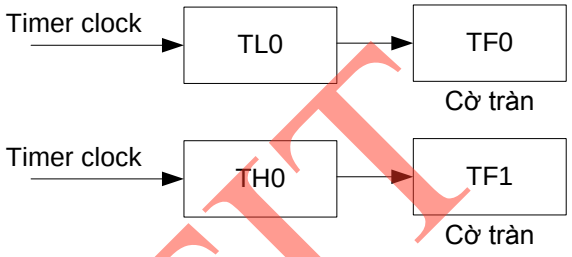
5.4.3. Chế độ định thời 2

Chế độ 2 là chế độ tự nạp lại 8 bit. Byte thấp của bộ định thời (TLx) hoạt động động định thời 8 bit trong khi byte cao của bộ định thời lưu giữ giá trị nạp lại. Khi số đếm tràn từ FFH xuống 00H, ngoài việc cờ tràn được set lên 1 thì giá trị trong THx được nạp vào TLx. Việc đếm sẽ tiếp tục từ giá trị này cho đến khi xảy ra tràn.

5.4.4. Chế độ định thời 3

Chế độ 3 là chế độ định thời chia sẻ và có hoạt động khác nhau cho từng bộ định thời. Bộ định thời 0 ở chế độ 3 được chia thành 2 bộ định thời 8 bit hoạt động riêng lẻ TL0 và TH0, mỗi bộ định thời sẽ set các cờ tràn tương tứng TF0 và TF1 khi xảy ra tràn.

Bộ định thời 1 không hoạt động ở chế độ 3 nhưng có thể được khởi động bằng cách chuyển bộ định thời này vào một trong các chế độ khác. Khi đó cờ tràn TF1 của bộ định thời 1 không bị ảnh hưởng bởi bộ định thời này khi xảy ra tràn vì TF1 được nối với bộ định thời 8 bit TH0.



Hình 5. 9 Chế độ 3

5.5. Lập trình cho bộ đếm/định thời

5.5.1. Lập trình ở chế độ 1

Đặc điểm của chế độ 1:

- Là bộ định thời 16 bit nên cho phép các giá trị từ 0000H đến FFFFH được nạp vào các thanh ghi TL và TH của bộ định thời.
- Sau khi TL và TH được nạp một giá trị khởi tạo 16 bit thì bộ định thời phải được khởi động. Điều này được thực hiện bởi "SETB TR0" đối với Timer 0 và "SETB TR1" đối với Timer 1.
- Sau khi bộ định thời được khởi động sẽ bắt đầu đếm lên cho đến khi đạt giới hạn FFFFH. Khi chuyển từ FFFFH về 0000H thì cờ tràn TF được bật lên. Khi cờ tràn được thiết lập, để dừng bộ định thời ta sử dụng lệnh "CLR TR0" đối với Timer 0 hoặc "CLR TR1" đối với Timer 1.
- Sau khi bộ định thời đạt giá trị giới hạn và quay lại giá trị 0000H, muốn lặp lại quá trình đếm thì các thanh ghi TH và TL phải được được nạp lại giá trị ban đầu và TF phải được đưa về giá trị 0.

Các bước lập trình ở chế độ 1:

1. Nạp giá trị cho thanh ghi TMOD để báo cho vi điều khiển biết sử dụng bộ định thời nào và chế độ nào được chọn.
2. Nạp thanh ghi TL và TH với các giá trị đếm ban đầu.
3. Khởi động bộ định thời.
4. Kiểm tra cờ bộ định thời TF bằng lệnh "JNB TFx, đích" để xem nó được bật không. Thoát khỏi vòng lặp khi TF có giá trị bằng 1.
5. Dừng bộ định thời.
6. Xóa cờ TF cho lần đếm kế tiếp
7. Quay trở lại bước 2 để nạp TL và TH

Ví dụ: Giả sử tần số XTAL là 11,0592MHz. Hãy viết chương trình tạo ra xung vuông tần số 2KHz trên chân P2.5 với độ rộng phần cao và thấp bằng nhau.

Giải:

- Chu kỳ của xung vuông: $T = \frac{1}{f} = \frac{1}{2KHz} = 500\mu s$
- Độ rộng của phần cao và thấp: $T_{cao} = T_{thấp} = \frac{1}{2}T = 250\mu s$
- Vì tần số XTAL = 11.0592MHz nên tần số của bộ đếm là $\frac{11.0592}{12} = 0{,}9216MHz$. Như vậy, bộ đếm tăng lên sau mỗi $1.085\mu s$.
- Số lần đếm trong khoảng thời gian 250 μs là: $\frac{250}{1.085} = 230$.
- Giá trị cần nạp vào các thanh ghi TH và TL là: 65536 – 230 = 65306 = FF1AH.

Chương trình:

```
            MOV  TMOD,#10H  ; Chọn bộ định thời Timer 1, chế độ 1
AGAIN:      MOV  TL1,#1AH   ; Gán giá trị byte thấp TL1 = 1AH
            MOV  TH1,#0FFH  ; Gán giá trị byte cao TH1 = FFH
            SETB TR1        ; Khởi động Timer 1
BACK:       JNB  TF1,BACK   ; Lặp cho đến khi TF1=1
            CLR  TR1        ; Dừng bộ định thời
            CPL  P1.5       ; Bù bit P1.5 để chuyển từ 1 sang 0 và ngược lại
            CLR  TF1        ; Xóa cờ TF1
            SJMP AGAIN      ; Nạp lại bộ định thời
```

5.5.2. Lập trình ở chế độ 0

Chế độ 0 hoàn toàn giống chế độ 1 chỉ khác là bộ định thời 16 bit được thay bằng 13 bit. Bộ đếm 13 bit có thể giữ các giá trị giữa 0000H đến 1FFFH trong TH và TL. Do vậy khi bộ định thời đạt giá trị cực đại là 1FFFH thì nó sẽ quay trở về 0000H và cờ TF được bật lên.

5.5.3. Lập trình ở chế độ 2

Đặc điểm của chế độ 2:

- Là bộ định thời 8 bit nên chỉ phép nạp các giá trị từ 00H đến FFH vào thanh ghi TH của bộ định thời.
- Sau khi TH được nạp, vi điều khiển copy giá trị này vào thanh ghi TL.
- Sau khi bộ định thời được khởi động bằng lệnh "SETB TR0" hoặc "SETB TR1", bộ đếm bắt đầu đếm tăng lên bằng cách tăng giá trị trong thanh ghi TL. Khi TL đạt giá trị FFH thì sẽ quay về giá trị 00H và thiết lập cờ tràn TF.
- Khi thanh ghi TL quay trở về 00H từ FFH thì TF được bật lên 1 và thanh ghi TL được tự động nạp lại giá trị ban đầu bởi thanh ghi TH. Khác với chế độ 1, trong chế độ 2 người lập trình không phải nạp lại giá trị cho TH mà quá trình này được thực hiện bởi vi điều khiển.

Các bước lập trình cho chế độ 2:

1. Nạp giá trị cho thanh ghi TMOD để báo cho vi điều khiển biết sử dụng bộ định thời nào và chế độ nào được chọn.
2. Nạp thanh ghi TH với giá trị đếm ban đầu.
3. Khởi động bộ định thời.
4. Kiểm tra cờ bộ định thời TF bằng lệnh "JNB TFx, đích" để xem nó được bật không. Thoát khỏi vòng lặp khi TF có giá trị bằng 1.
5. Xóa cờ TF
6. Quay trở lại bước 4 vì chế độ 2 là chế độ tự nạp lại.

Ví dụ: Giả sử tần số XTAL = 11.0592 MHz. Hãy tìm

- Tần số của sóng vuông được tạo ra trên chân P1.0 trong chương trình sau.
- Tần số nhỏ nhất có thể có được bằng chương trình này và giá trị TH để đạt được điều đó.

```
        MOV  TMOD,#20H      ;Chọn Timer 1/chế độ 2
        MOV  TH1,#5         ; TH1=5
        SETB TR1            ;Khởi động Timer1
BACK:   JNB  TF1,BACK       ;Giữ nguyên cho đến khi TF1=1
        CPL  P1.0           ;Dừng bộ định thời
        CLR  TF1            ;Xóa cờ TF1
        SJMP BACK
```

Giải:

- Độ rộng phần cao của xung: (256-5)*1.085=272.33μs

  Tần số của xung là:$\frac{1}{2*272.33} = 1.83597KHz$
- Để nhận được tần số nhỏ nhất thì ta cần tạo chu kỳ T lớn nhất có thể. Nghĩa là TH = 00H. Trong trường hợp này, $T = 2 * 265 * 1.085 = 555.52\mu s$ và tần số là $\frac{1}{T} = 1.8KHz$

5.5.4. Lập trình bộ đếm

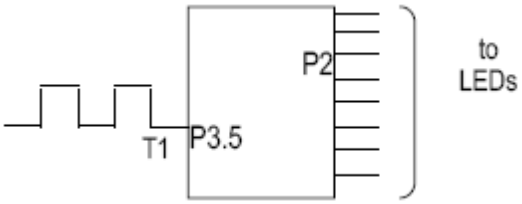
Ở phần trên ta đã sử dụng các bộ định thời của 8051 để tạo ra các độ trễ thời gian. Các bộ định thời này cũng có thể được dùng như các bộ đếm các sự kiện xảy ra bên ngoài 8051. Đối với bộ định thời thì nguồn tần số là tần số thạch anh của 8051. Tuy nhiên, khi nó được dùng như một bộ đếm thì nguồn xung để tăng nội dung thanh ghi TH và TL là từ bên ngoài 8051. Ở chế độ bộ đếm, các thanh ghi TMOD, TH và TL cũng có các chế độ giống như ở bộ định thời ngoại trừ bit $C/\overline{T}$. Nếu bit $C/\overline{T}$ = 0 thì bộ định thời nhận các xung đồng hồ từ bộ dao động thạch anh của 8051. Ngược lại, khi $C/\overline{T}$ = 1 thì bộ định thời nhận các xung từ nguồn bên ngoài được đưa đến chân P3.4, P3.5 để tăng bộ đếm. Các chân này có tên là T0 (đầu vào của bộ đếm Timer 0) và T1 (đầu vào của bộ Timer 1).

Ví dụ: Giả sử xung đồng hồ được cấp tới chân T1, hãy viết chương trình cho bộ đếm 1 ở chế độ 2 để đếm các xung và hiển thị trạng thái của số đếm TL1 trên cổng P2.

Giải:

```
        MOV   TMOD,#01100000B  ;chọn bộ đếm 1, chế độ 2, C/T = 1
        MOV   TH1,#0           ; xóa TH1
        SETB  P3.5             ; Lấy đầu vào từ T1
AGAIN:  SETB  TR1              ; Khởi động bộ đếm
BACK:   MOV   A,TL1            ; Lấy số đếm từ TL1
        MOV   P2,A             ; Đưa số đếm ra cổng P2
        JNB   TF1,BACK         ; Nếu TF1 = 0 thì nhảy đến nhãn
        CLR   TR1              ; Nếu TF1 = 1 thì dừng bộ đếm
        CLR   TF1              ; Xóa cờ TF
        SJMP  AGAIN            ; Quay trở lại đếm từ đầu.
```

Trong chương trình trên, vì các cổng mặc định là đầu ra khi 8051 được cấp nguồn nên khi ta muốn P3.5 trở thành đầu vào thì phải dùng lệnh SETB P3.5 để bật nó lên. Nói cách khác, ta phải cấu hình (đưa lên cao) chân T1 (P3.5) để cho phép các xung được cấp vào nó.



5.6. Tốc độ baud cho cổng nối tiếp

Trong phần này chúng ta sẽ nghiên cứu về các thanh ghi truyền thông nối tiếp của 8051 và cách lập trình chúng để truyền và nhận dữ liệu nối tiếp với cổng COM của PC. Để cho phép truyền dữ liệu giữa máy tính PC và hệ thống 8051 mà không có bất kỳ lỗi nào thì chúng ta phải biết chắc rằng tốc độ baud của 8051 phải phù hợp với tốc độ baud của cổng COM.

8051 truyền và nhận dữ liệu nối tiếp theo nhiều tốc độ khác nhau. Tốc độ truyền của nó có thể lập trình được. Điều này được thực hiện nhờ sự trợ giúp của bộ định thời Timer1. Trước khi ta đi vào bàn cách làm điều đó như thế nào thì ta sẽ xét quan hệ giữa tần số thạch anh và tốc độ baud trong 8051.

Như đã đề cập ở trên, tần số của thạch anh được chia cho 12 để tính tần số máy. Trong trường hợp XTAL = 11.0592MHz thì tần số máy là 921.6KHz. Mạch điện truyền thông nối tiếp UART chia tiếp tần số máy cho 32 một lần nữa trước khi nó được dùng bộ định thời Timer1 để tạo ra tốc độ baud (28.8KHz). Muốn Timer1 đặt tốc độ baud thì nó phải được lập trình về chế độ mode 2 (chế độ thanh ghi 8 bit tự động nạp lại). Để có tốc độ baud tương thích với PC ta phải nạp TH1 theo các giá trị trong bảng dưới đây:

Tốc độ baud	TH1 (thập phân)	TH1 (Hexa)
9600	-3	FD
4800	-6	FA
2400	-12	F4
1200	-24	F8

Bảng 5. 4 Bảng tốc độ baud

5.6.1. Thanh ghi điều khiển và các chế độ hoạt động của cổng nối tiếp

5.6.1.1. Thanh ghi SBUF

SBUF là thanh ghi 8 bit được dùng riêng cho truyền thông nối tiếp trong 8051. Đối với một byte dữ liệu cần truyền qua đường TxD thì nó phải được đặt trong thanh ghi SBUF. Tương tự như vậy, SBUF giữ một byte dữ liệu khi nó được nhận bởi đường RxD của 8051. SBUF có thể được truy cập bởi mọi thanh ghi bất kỳ trong 8051

```
Ví dụ:      MOV  SBUF,#"D"   ; Nạp vào SBUF giá trị 44H là mã ACSII của D
            MOV  SBUF,A      ; Chép nội dung của A vào SBUF
            MOV  A,SBUF      ; Chép nội dung của SBUF vào A
```

Khi một byte được ghi vào SBUF nó sẽ được đóng khung với các bit Start, Stop và được truyền nối tiếp qua chân TxD. Tương tự như vậy, khi các bit được nhận nối tiếp từ RxD thì 8051 mở khung để loại bỏ các bit Start, Stop để lấy ra một byte dữ liệu đặt vào thanh ghi SBUF.

5.6.1.2. Thanh ghi điều khiển nối tiếp SCON

Thanh ghi SCON là thanh ghi 8 bit được dùng để lập trình việc đóng khung bit bắt đầu Start, bit dừng Stop và các bit dữ liệu.

Mô tả các bit của thanh ghi SCON:

SM0	SM1	SM2	REN	TB8	RB8	T1	R1

Hình 5. 10 Thanh ghi SCON

SM0, SM1:

M0 và M1 được dùng để xác định đóng khung dữ liệu bằng cách xác định số bit của một ký tự và các bit Start, Stop. Các tổ hợp của chúng là:

SM0	SM1	Serial Mode	Explanation	Baud Rate
0	0	0	8-bit Shift Register	Oscillator / 12
0	1	1	8-bit UART	Set by Timer 1
1	0	2	9-bit UART	Oscillator/ 32
1	1	3	9-bit UART	Set by Timer 1

Bảng 5. 5 Các chế độ truyền nối tiếp

SM2: Cho phép khả năng đa xử lý. Trong phạm vi bài giảng này, SM2 được đặt bằng 0 vì không sử dụng 8051 trong môi trường đa xử lý.

REN: Đây là bit cho phép thu (Receive Enable). Bit REN cũng được tham chiếu bằng SCON.4 vì SCON là thanh ghi đánh địa chỉ theo bit. Khi bit REN = 1 thì 8051 thu dữ liệu trên chân RxD.

TB8: TB8 được dùng cho chế độ độ nối tiếp 2 và 3. Trong các chế độ này, 9 bit dữ liệu được truyền đi trong đó 8 bit dữ liệu và bit thứ 9 được lấy từ TB8. Nếu TB8 = 1, các bit dữ liệu sẽ lần lượt được ghi ra cổng nối tiếp và được theo sau cùng bởi bit TB8. Nếu TB8 = 0, bit thứ 9 sẽ không được sử dụng.

RB8: RB8 cũng hoạt động trong chế độ 2 và 3 tương tự như TB8 nhưng ở chiều nhận. Khi 9 bit dữ liệu nhận trong chế độ 2 và 3, 8 bit đầu tiên được đặt vào SBUF, bit thứ 9 được lưu trong RB8.

TI và RI: Các bit ngắt truyền TI và ngắt thu RI là các bit D1 và D0 của SCON. Khi 8051 kết thúc truyền một ký tự 8 bit thì nó bật TI để báo rằng nó sẵn sàng truyền một byte khác. Bit TI được bật lên trước bit Stop. Khi 8051 nhận dữ liệu nối tiếp qua chân RxD thì nó tách các bit Start và Stop để lấy ra 8 bit dữ liệu để đặt vào SBUF. Sau khi hoàn tất, nó bật cờ RI để báo 8051 cần đọc nhanh giá trị trước khi một byte khác đi tới.

5.6.1.3. Khởi động và truy xuất các thanh ghi

Khi lập trình 8051 để truyền các byte ký tự nối tiếp thì cần thực hiện các bước sau đây:

- Nạp thanh ghi TMOD giá trị 20H để báo sử dụng Timer1 ở chế độ 2 thiết lập chế độ baud
- Nạp thanh ghi TH1 các giá trị để thiết lập tốc độ baud.
- Nạp thanh ghi SCON giá trị 50H báo chế độ nối tiếp 1 để đóng khung 8 bit dữ liệu, 1 bit Start và 1 bit Stop.
- Bật TR1 = 1 để khởi động Timer 1.
- Xóa bit TI bằng lệnh “CLR TI”.
- Byte ký tự cần phải truyền được ghi vào SBUF.

- Bit cờ TI được hiển thị bằng lệnh "JNB TI, xx" để báo ký tự đã được truyền hoàn tất chưa.
- Để truyền ký tự tiếp theo quay trở về bước 5.

Ví dụ: Viết chương trình cho 8051 để truyền nối tiếp ký tự "A" với tốc độ 4800 baud liên tục.

Giải:

```
            MOV   TMOD,#20H      ; Chọn Timer1, chế độ 2 (tự động nạp lại)
            MOV   TH1,#-6        ; Chọn tốc độ baud 4800
            MOV   SCON,#50H      ; Truyền 8 bit dữ liệu, 1 bit Stop
            SETB  TR1            ; Khởi động Timer1
AGAIN:      MOV   SBUF,#"A"      ; Nạp ký tự A
HERE:       JNB   TI,HERE        ; Chờ đến bit cuối cùng
            CLR   TI             ; Xóa bit TI cho ký tự kế tiếp
            SJMP  AGAIN          ; Tiếp tục gửi lại ký tự A
```

Ví dụ: Viết chương trình để truyền chữ "YES" nối tiếp liên tục với tốc độ 9600 baud (8 bit dữ liệu, 1 bit Stop)

Giải:

```
            MOV   TMOD,#20H      ; Chọn bộ Timer1, chế độ 2
            MOV   TH1,#-3        ; Chọn tốc độ 9600 baud
            MOV   SCON,#50H      ; Truyền 8 bit dữ liệu, 1 bit Stop
            SETB  TR1            ; Khởi động Timer1
AGAIN:      MOV   A,#"Y"         ; Truyền ký tự "Y"
            ACALL TRANS
            MOV   A,#"E"         ; Truyền ký tự "E"
            ACALL TRANS
            MOV   A,#"S"         ; Truyền ký tự "S"
            ACALL TRANS
            SJMP  AGAIN
; Chương trình con truyền dữ liệu nối tiếp
TRANS:      MOV   SBUF,A         ; Nạp SBUF
HERE:       JNB   TI,HERE        ; Chờ cho đến khi truyền bit cuối
            CLR   TI
            RET
```

Trong 8051, để nhận các byte ký tự nối tiếp ta thực hiện các bước sau đây:

- Nạp giá trị 20H và thanh ghi TMOD để báo sử dụng bộ Timer 1, chế độ 2 (8 bit, tự động nạp lại) để thiết lập chế độ baud.
- Nạp TH1 giá trị tương ứng với tốc độ baud.

- Nạp giá trị 50H vào thanh ghi SCON để báo sử dụng chế độ truyền nối tiếp.
- Bật TR1 = 1 để khởi động Timer 1.
- Xóa cờ ngắt RI bằng lệnh "JNB RI, xx" để xem toàn bộ ký tự đã được nhận chưa
- Khi RI được thiết lập thì trong SBUF đã có 1 byte.
- Để nhận ký tự tiếp theo ta quay trở về bước 5.

Ví dụ: Lập trình cho 8051 để nhận các byte dữ liệu nối tiếp và đặt chúng vào cổng P1. Đặt tốc độ baud là 4800, 8 bit dữ liệu và 1 bit Stop.

Giải:

```
        MOV   TMOD,#20H     ; Chọn bộ Timer1, chế độ 2
        MOV   TH1,-6        ; Chọn tốc độ 4800 baud
        MOV   SCON,#50H     ; Chọn khung dữ liệu 8 bit
        SETB  TR1           ; Khởi động bộ Timer1
HERE:   JNB   R1,HERE       ; Đợi nhận toàn bộ ký tự
        MOV   A,SBUF        ; Lưu ký tự vào thanh ghi A
        MOV   P1,A          ; Gửi ra cổng P1
        CLR   RI            ; Sẵn sàng nhận byte kế tiếp
        SJMP  HERE          ; Tiếp tục nhận dữ liệu
```

CHƯƠNG 6. LẬP TRÌNH NGẮT TRONG 8051

6.1. Các ngắt của 8051

Trong phương pháp sử dụng các ngắt thì mỗi khi có một thiết bị bất kỳ cần đến dịch vụ của nó 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 ngắt tất cả những gì nó đang thực hiện để chuyển sang phục vụ thiết bị. Chương trình đi cùng với ngắt được gọi là trình dịch vụ ngắt ISR (Interrupt Service Routine) hay còn gọi là trình quản lý ngắt (Interrupt handler).

Đặc điểm của phương pháp ngắt là bộ vi điều khiển có thể phục vụ được rất nhiều thiết bị (tất nhiên là không tại cùng một thời điểm). Mỗi thiết bị có thể nhận được sự chú ý của bộ vi điều khiển dựa trên mức ưu tiên được gán cho nó.

Đối với mỗi ngắt thì phải có một trình phục vụ ngắt ISR hay trình quản lý ngắt. Trình quản lý ngắt nằm tại một địa chỉ xác định trong ROM. Nhóm các vị trí nhớ được dành riêng để gửi các địa chỉ của các ISR được gọi là bảng vector ngắt.

Ngắt	Địa chỉ ROM	Chân
Bật lại nguồn (RESET)	0000	9
Ngắt phần cứng ngoài (INT0)	0003	12 (P3.2)
Ngắt Timer0 (TF0)	000B	
Ngắt phần cứng ngoài 1 (INT1)	0013	13 (P3.3)
Ngắt bộ Timer 1 (TF1)	001B	
Ngắt COM nối tiếp (RI và TI)	0023	

Bảng 6. 1 Bảng vector ngắt

- RESET: Khi chân RESET được kích hoạt từ 8051 nhảy về địa chỉ 0000. Đây là địa chỉ bật lại nguồn.
- TF0, TF1: Hai ngắt dành cho bộ định thời Timer0 và Timer1 tương ứng.
- INT0, INT1: Hai ngắt dành cho các ngắt phần cứng bên ngoài chân 12 (P3.2) và 13 (P3.3) của cổng P3. Các ngắt ngoài cũng còn được coi như EX1 và EX2.
- Truyền thông nối tiếp có một ngắt thuộc về cả thu và phát.

Cho phép và cấm ngắt

Khi bật lại nguồn thì tất cả mọi ngắt đầu bị cấm (bị che). Điều này có nghĩa là không có ngắt nào sẽ được bộ vi điều khiển đáp ứng nếu chúng được kích hoạt. Các ngắt phải được kích hoạt bằng phần mềm để bộ vi điều khiển đáp ứng chúng. Có một thanh ghi được gọi là thanh ghi cho phép ngắt IE (Interrupt Enable) chịu trách nhiệm về việc cho phép (không che) và cấm (che) các ngắt.

D7							D0
EA	---	ET2	ES	ET1	EX1	ET0	EX0

Hình 6. 1 Thanh ghi cho phép ngắt

EA: Đây là bit cho phép tất cả các ngắt EA (Enable ALL).

- EA = 1: Tất cả mọi ngắt đều được phép và sẽ được đáp ứng nếu các bit của trong trong IE có mức cao
- EA = 0: Không có ngắt nào sẽ được đáp ứng cho dù bit tương ứng của nó trong IE có giá trị cao

--: Bit IE.6 là bit dự phòng cho tương lai

ET2: Bit IE.5 cho phép hoặc cấm ngắt tràn hoặc ngắt thu của Timer2

ES: Bit IE.4 cho phép hoặc cấm ngắt cổng nối tiếp

ET1: Bit IE.3 cho phép hoặc cấm ngắt tràn của Timer1

EX1: Bit IE.2 cho phép hoặc cấm ngắt ngoài 1

ET0: Bit IE.1 cho phép hoặc cấm ngắt tràn của Timer0

EX0: Bit IE.0 cho phép hoặc cấm ngắt ngoài 0.

Ví dụ: Sử dụng lệnh để

a. Cho phép ngắt nối tiếp, ngắt Timer0 và ngắt phần cứng ngoài 1
b. Che ngắt Timer0
c. Che tất cả mọi ngắt chỉ bằng một lệnh duy nhất

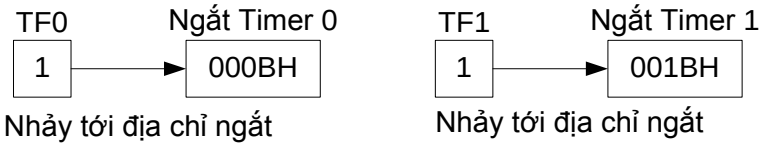
Giải:

a. MOV IE,#10010110B
b. CLR IE.1
c. CLR IE.7

6.2. Lập trình các ngắt của bộ định thời

Trong chương 5, cách sử dụng các bộ định thời Timer0 và Timer1 bằng phương pháp thăm dò đã được đề cập. Trong phần này, ta sẽ sử dụng các ngắt để lập trình cho các bộ định thời.

Trong phương pháp thăm dò, lệnh "JNB TF, nhãn" được sử dụng để chương trình liên tục thăm dò trạng thái của cờ TF. Khi cờ TF được bật lên, chương trình mới thực hiện lệnh tiếp theo. Vấn đề với phương pháp này là bộ vi điều khiển bị trói buộc khi cờ TF chưa được bật lên và không thể làm được bất kỳ việc gì khác. Sử dụng các ngắt có thể giải quyết vấn đề này. Nếu bit cho phép ngắt định thời trong thanh ghi IE được bật lên thì mỗi khi cờ TF được bật lên, bộ vi điều khiển bị ngắt tại bất kỳ thời điểm nó đang thực hiện việc nào đó và nhảy tới bảng vector ngắt để phục vụ ISR. Bằng cách này thì bộ vi điều khiển có thể làm những công việc khác cho đến khi nào nó nhận được thông báo bộ định thời đã quay về 0.



Hình 6. 2 Ngắt định thời TF0 và TF1

Các chương trình phục vụ ngắt kích thước nhỏ

Các chương trình phục vụ ngắt phải được bắt đầu ở gần đáy của bộ nhớ tại các địa chỉ cho ở bảng 6.1. Mặc dù chỉ có 8 byte ở giữa các điểm nhập của các trình phục vụ ngắt, dung lượng này thường đủ để thực hiện các công việc được yêu cầu và quay trở về chương trình chính từ một trình phục vụ ngắt. Điều này có nghĩa là trình phục vụ ngắt thường không dài quá 8 byte. Cấu trúc chương trình sử dụng ngắt bộ định thời có kích thước nhỏ như sau:

```
                ORG   0000H       ; reset
                LJMP  MAIN
                ORG   000BH       ; Địa chỉ của ngắt do bộ định thời 0
T0ISR:----                        ; Bắt đầu ISR của bộ định thời 0
                ----
                RETI              ; Trở về chương trình chính
MAIN:           ----
```

Nếu có nhiều ngắt được sử dụng, ta phải cẩn thận để đảm bảo các ISR được bắt đầu đúng vị trị và không tràn sang ISR kế. Vì chỉ có một ngắt được sử dụng trong cấu trúc trên nên chương trình chính được bắt đầu ngay sau lệnh RETI.

Các chương trình phục vụ ngắt kích thước lớn

Nếu một trình phục vụ ngắt dài hơn 8 byte được cần đến, ta phải di chuyển chương trình này đến một nơi khác trong bộ nhớ chương trình hoặc ta có thể cho lấn qua điểm nhập của ISR kế. Cấu trúc chương trình sử dụng ngắt có kích thước lớn:

```
                ORG   0000H       ; điểm nhập reset
                LJMP  MAIN
                ORG   000BH       ; điểm nhập của bộ định thời 0
                LJMP  TOISR
                ORG   0030H       ; phía trên vector ngắt
MAIN:           -----
                -----
T0ISR:-----                       ; ISR của bộ định thời 0
                RETI              ; Quay về chương trình chính.
```

Ví dụ: Viết chương trình sử dụng bộ định thời 0 và các ngắt để tạo ra một sóng vuông tần số 10KHz trên chân P1.0

Giải:

```
                ORG   0000H          ; điểm nhập reset
                LJMP  MAIN           ; Nhảy qua khỏi các vector ngắt
                ORG   000BH          ; vector ngắt của timer0
T0ISR:          CPL   P1.0           ; lấy bù
                RETI
                ORG   0030H          ; điểm nhập của chương trình chính
MAIN:           MOV   TMOD, #02H     ; chế độ 2 của timer0
                MOV   TH0, #-50      ; đếm trong 50 microgiây
                SETB  TR0            ; khởi động timer0
                MOV   IE, #82H       ; cho phép ngắt của timer0
                SJMP  $              ; không làm gì
                END
```

Đây là một chương trình hoàn chỉnh và có thể nạp cho EPROM sau khi dịch sang mã máy. Ngay sau khi reset hệ thống, bộ đếm chương trình PC được nạp 0000H. Lệnh đầu tiên được thực thi là LJMP MAIN để rẽ nhánh đến chương trình chính ở địa chỉ 0030H trong ROM. Ba lệnh đầu tiên của chương trình chính khởi động bộ định thời 0 ở chế độ tự nạp lại 8 bit sao cho sẽ tràn sau mỗi 50 μs. Lệnh MOV IE, #82H cho phép các ngắt do bộ định thời 0 tạo ra. Mỗi một lần tràn, bộ định thời sẽ tạo ra một ngắt. Khi ngắt xuất hiện, chương trình chính bị ngắt và trình phục vụ ngắt cho bộ định thời 0 được thực thi. Ở ví dụ này, ISR chỉ đơn giản lấy bù bit của port và quay trở về chương trình chính nơi vòng lặp "không làm gì" được thực thi để chờ một ngắt mới sau 50 μs.

Lưu ý là ta không phải sử dụng lệnh xóa cờ tràn TF0 do cờ này tự động được xóa bởi phần cứng khi CPU trỏ tới trình phục vụ ngắt.

Hiển nhiên địa chỉ quay về trong chương trình là địa chỉ của lệnh SJMP. Địa chỉ này được cất vào vùng stack nội của 8051 trước khi CPU trỏ tới trình phục vụ ngắt và được lấy lại từ stack khi lệnh RETI ở cuối trình phục vụ ngắt được thực thi. Vùng stack được mặc định ở địa chỉ 07H trong RAM nội. Địa chỉ quay về được đặt vào địa chỉ 08H và 09H trong RAM nội.

Ví dụ: Viết một chương trình sử dụng các ngắt để tạo đồng thời các dạng sóng vuông có tần số là 20KHz và 50KHz trên các chân P1.7 và P1.6

Giải:

```
                ORG   0000H
                LJMP  MAIN
                ORG   000BH          ; Chương trình phục vụ ngắt Timer0
                CPL   P1.6
                RETI
                ORG   001BH          ; Chương trình phục vụ ngắt Timer1
```

```
        CPL   P1.7
        RETI

        ORG   0030H
MAIN:   MOV   TMOD, #22H
        MOV   TH0, #E7H     ; Timer0, chế độ 2, chu kỳ tràn: 25microgiây
        MOV   TH1, #F6H     ; Timer1, chế độ 2, chu kỳ tràn: 10microgiây
        MOV   IE, #8AH
        SETB  TR0           ; Khởi động Timer0
        SETB  TR1           ; Khởi động Timer1
HERE:   SJMP  HERE
        END
```

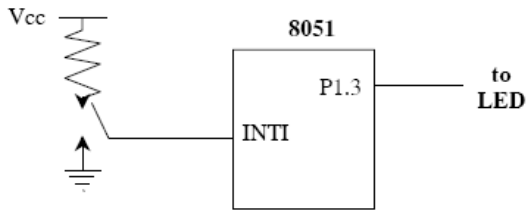
6.3. Lập trình các ngắt phần cứng bên ngoài

Bộ vi điều khiển 8051 có hai ngắt phần cứng bên ngoài là chân 12 (P3.2) và chân 13 (P3.3) dùng cho ngắt INT0 và INT1. Khi kích hoạt những chân này thì 8051 bị ngắt tại bất kỳ công việc nào mà nó đang thực hiện và nó nhảy đến bảng vector ngắt để thực hiện chương trình ngắt. Có hai mức kích hoạt cho các ngắt phần cứng ngoài: ngắt theo mức và ngắt theo sườn.

6.4. Ngắt theo mức

Ở trạng bình thường, các chân $\overline{INT0}$ và $\overline{INT1}$ ở mức cao. Khi có một tín hiệu ở mức thấp được cấp tới chúng thì vi điều khiển dừng tất cả mọi công việc nó đang thực hiện và nhảy đến bảng vector ngắt để phục vụ ngắt. Điều này được gọi là ngắt được kích hoạt theo mức và là chế độ ngắt mặc định khi cấp nguồn lại cho 8051. Tín hiệu mức thấp tại chân $\overline{INT}$ phải được lấy đi trước khi thực hiện lệnh cuối cùng của trình phục vụ ngắt RETI. Nếu không một ngắt khác sẽ lại được tạo ra.

Ví dụ: Cho mạch điện sau:



Ở trạng thái không được kích hoạt thì công tắc ở mức cao. Viết chương trình để khi ấn công tắc xuống sẽ làm cho đèn LED sáng trong một khoảng thời gian nhất định. Nếu công tắc được giữ ở trạng thái kích hoạt thì đèn LED sáng liên tục.

Giải:

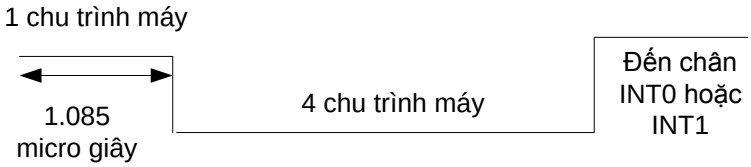
```
        ORG   0000H
        LJMP  MAIN
```

```
                ORG   0013H          ; Nhảy đến vector ngắt ngoài 1
                SETB  P1.3           ; Bật đèn LED
                MOV   R3, #255       ;
BACK:           DJNZ  R3, BACK       ; Giữ đèn LED sáng một lúc
                CLR   P1.3           ; Tắt đèn LED
                RETI
; Bắt đầu chương trình chính
                ORG   30H
MAIN:           MOV   IE, #84H               ; Cho phép ngắt ngoài 1
                SJMP  $                      ; Lặp liên tục đến khi được ngắt
                END
```

Trong chương trình này, vi điều khiển quay vòng liên tục trong vòng lặp "SJMP $". Mỗi khi công tắc trên chân P3.3 được kích hoạt thì vi điều khiển thoát khỏi vòng lặp và nhảy đến bảng vector ngắt tại địa chỉ 0013H. Trình ISR cho đèn LED bật lên và giữ trong một khoảng thời gian nhất định ((2+255x2) x 1.085μs = 555.52μs) và tắt nó trước khi trở về chương trình chính. Nếu trong lúc ISR quay trở về chương trình chính mà chân INT1 vẫn còn ở mức thấp thì vi điều khiển khởi tạo lại ngắt. Do vậy, nếu chỉ để đèn LED sáng một khoảng thời gian nhất định thì công tắc phải được đưa lên cao trước thời điển lệnh RETI được thực hiện.

6.4.1. Lấy mẫu ngắt theo mức

Các chân P3.2 và P3.3 bình thường được dùng cho vào – ra dữ nếu các bit $\overline{INT0}$ và $\overline{INT1}$ trong thanh ghi IE không được kích hoạt. Sau khi các bit này được kích hoạt thì vi điều khiển duy trì lấy mẫu trạng thái trên các chân này đối với tín hiệu thấp một lần trong một chu kỳ máy. Theo quy định của nhà sản xuất thì "chân ngắt phải được giữ ở mức thấp cho đến khi bắt đầu thực hiện trình phục vụ ngắt ISR. Nếu chân ngắt được đưa trở lại mức cao trước khi bắt đầu thực hiện ISR thì sẽ chẳng có ngắt nào xảy ra". Tuy nhiên, trong quá trình kích hoạt ngắt theo mức thấp thì nó phải được đưa lên mức cao trước khi lệnh RETI được thực hiện. Nếu chân INT vẫn ở mức thấp sau lệnh RETI của trình phục vụ ngắt thì một ngắt khác sẽ được kích hoạt sau lệnh khi lệnh RETI được thực hiện. Do vậy, để đảm bảo việc kích hoạt ngắt phần cứng tại các chân INT thì tín hiệu ở mức thấp phải tồn tại khoảng 4 chu kỳ máy và không được hơn. Điều này do một thực tế là ngắt theo mức không được chốt. Do đó chân ngắt phải được giữ ở mức thấp cho đến khi bắt đầu thực hiện ISR.



Ghi chú: Khi bật lại nguồn (RESET) thì cả hai chân INT0 và INT1 đều ở mức thấp tạo các ngắt ngoài theo mức.

Hình 6. 3 Thời gian tối thiểu dành cho ngắt mức thấp

6.4.2. Ngắt theo sườn

Theo mặc định, các ngắt $\overline{INT0}$ và $\overline{INT1}$ luôn ở mức thấp. Để biến các chân này trở thành các ngắt theo sườn thì chúng ta phải viết chương trình cho các bit của thanh ghi TCON. Trong thanh ghi TCON, hai bit IT0 và IT1 xác định chế độ ngắt theo mức hay ngắt theo sườn của các ngắt phần cứng IT0 và IT1. Bằng việc sử dụng các lệnh "SETB TCON.0" và "SETB TCON.2" thì các ngắt phần cứng ngoài trở thành các ngắt theo sườn. Tuy nhiên, lưu ý rằng các bit ngắt cũng phải được cho phép trong thanh ghi IE.

Ví dụ: Giả thiết chân P3.3 được nối với một máy tạo xung. Hãy viết chương trình trong đó sườn xuống của xung sẽ gửi một tín hiệu cao đến chân P1.3 để làm LED sáng. Nói cách khác, đèn LED được bật và tắt cùng tần số với các xung được cấp tới chân $\overline{INT1}$.

Giải:

```
                ORG   0000H
                LJMP  MAIN
                ORG   0013H          ; Nhảy đến vector ngắt ngoài 1
                SETB  P1.3           ; Bật đèn LED
                MOV   R3, #255       ;
BACK:           DJNZ  R3,BACK        ; Giữ đèn LED sáng một lúc
                CLR   P1.3           ; Tắt đèn LED
                RETI
; Bắt đầu chương trình chính
                ORG   30H
                SETB  TCON.2
MAIN:           MOV   IE,#84H              ; Cho phép ngắt ngoài 1
                SJMP  $                    ; Lặp liên tục đến khi được ngắt
                END
```

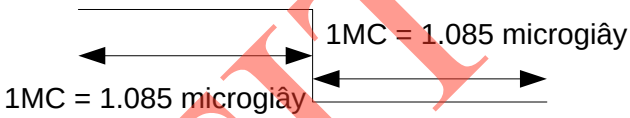
Ví dụ này tương tự ví dụ ở phần ngắt theo mức. Tuy nhiên, trong chương trình chính ta phải sử dụng lệnh "SETB TCON.2" để chuyển ngắt $\overline{INT1}$ về kiểu ngắt theo sườn. Khi sườn xuống của tín hiệu được cấp đến chân INT1 thì đèn LED được bật lên một khoảng thời gian. Để bật lại đèn LED thì cần phải có một sườn xung xuống khác được cấp tới chân P3.3. Trong ví dụ trước,

do bản chất ngắt theo mức nên đèn LED còn sáng chừng nào tín hiệu chân $\overline{INT1}$ vẫn còn ở mức thấp. Trong ví dụ này, để bật lại đèn LED thì xung ở chân $\overline{INT1}$ phải được đưa lên cao rồi sau đó bị hạ xuống thấp để tạo ra một sườn xuống làm kích hoạt ngắt.

6.4.3. Lấy mẫu ngắt theo sườn

Một câu hỏi đặt ra là ngắt theo sườn được vi điều khiển phát hiện như thế nào? Như trên đã đề cập, vi điều khiển lấy mẫu trạng thái chân INT một lần ở mỗi chu kỳ máy. Ở chế độ ngắt theo cạnh, nếu mẫu ở chân $\overline{INTx}$ (x = 0 hoặc 1) cho thấy chân này ở mức cao trong một chu kỳ và ở mức thấp trong chu kỳ kế tiếp thì cờ ngắt IE trong thanh ghi TCON được đặt bằng 1. Kế đến, IE yêu cầu một ngắt.

Vì các chân ngắt ngoài được lấy mẫu một lần ở mỗi chu kỳ máy nên các ngõ vào này phải được duy trì tối thiểu 12 chu kỳ dao động để đảm bảo việc lấy mẫu là đúng. Nếu ngắt ngoài thuộc loại tác động cạnh, nguyên nhân ngắt ngoài phải được duy trì tại chân yêu cầu ở mức cao tối thiểu một chu kỳ và sau đó ở mức thấp tối thiểu một chu kỳ nữa để đảm bảo rằng sự chuyển trạng thái được phát hiện.



Hình 6. 4 Thời gian xung tối thiểu để phát hiện ra ngắt theo sườn với XTAL = 11.0592MHz

Sườn xuống của xung được chốt bởi 8051 và được giữ bởi thanh ghi TCON. Các bit TCON.1 (IE0) và TCON.3 (IE1) hoạt động như các cờ "ngắt đang được phục vụ" (Interrupt – in – servive). Khi một cờ "ngắt đang được phục vụ" bật lên thì sẽ không có ngắt nào được đáp ứng trên chân INT này chừng nào ngắt này chưa được phục vụ xong.

Khi các chương trình phục vụ ngắt ISR kết thúc (nghĩa là trong thanh ghi thực hiện lệnh RETI), các bit IE0 và IE1 được xóa để báo rằng ngắt được hoàn tất và 8051 sẵn sàng đáp ứng ngắt khác trên chân đó. Để ngắt khác được nhận thì tín hiệu trên chân đó phải trở lại mức cao và sau đó nhảy xuoogns mức thấp để được phát hiện như một ngắt theo sườn.

Trong thời gian trình phục vụ ngắt đang được thực hiện thì chân $\overline{INTx}$ bị làm ngơ và vi điều khiển không quan tâm nó có bao nhiêu lần chuyển dịch từ cao xuống thấp. Trong thực tế, một trong các chức năng của lệnh RETI là xóa bit IE0 và IE1 trong thanh ghi TCON. Nó báo cho ta rằng trình phục vụ ngắt sắp kết thúc. Vì lý do này mà các bit IE0 và IE1 được gọi là các bit báo "ngắt đang được phục vụ" Cờ này sẽ lên cao khi một sườn xuống được phát hiện trên chân $\overline{INTx}$ và dừng ở mức cao trong toàn bộ quá trình thực hiện ISR. Nó chỉ bị xóa bởi lệnh RETI là lệnh cuối cùng của ISR. Do vậy, sẽ không cần đến các lệnh xóa các bit như "CLR TCON.1" hay "CLR TCON.3" trước lệnh RETI trong trình phục vụ ngắt ngoài.

6.5. Lập trình ngắt truyền thông nối tiếp

Như đã đề cập ở chương 5, cờ ngắt truyền TI được bật lên khi bit cuối cùng của khung dữ liệu (bit Stop) được truyền đi báo rằng thanh ghi SBUF sẵn sàng truyền byte kế tiếp. Cờ RI được bật lên khi toàn bộ khung dữ liệu (kể cả bit Stop) đã được nhận. Cờ RI bật lên để báo rằng byte dữ liệu nhận được trong SBUF cần được cất vào nơi an toàn trước khi nó bị ghi đè bởi dữ liệu mới nhận được.

Cơ chế truyền và nhận thông tin trong hai phương pháp thăm dò và ngắt hoàn toàn giống nhau. Sự khác nhau duy nhất giữa hai phương pháp này là ở cách phục vụ quá trình truyền thông nối tiếp như thế nào. Trong phương pháp thăm dò thì chúng ta phải đợi cho cờ (TI hay RI) bật lên và trong lúc chờ đợi thì ta không thể làm gì khác. Trong phương pháp ngắt thì ta được báo khi 8051 đã nhận được một byte hoặc nó sẵn sàng truyền byte kế tiếp và ta có thể làm các công việc khác trong khi truyền thông nối tiếp đang được phục vụ.

Trong 8051 chỉ có một ngắt dành riêng cho truyền thông nối tiếp. Ngắt này được dùng cho cả truyền và nhận dữ liệu. Khi RI và TI bật lên thì bit ngắt trong thanh ghi IE (bit IE.4) được bật lên báo cho 8051 nhảy đến địa chỉ trình phục vụ ngắt dành cho truyền thông nối tiếp 0023H trong bảng vector ngắt. Trong trình ISR này chúng ta phải kiểm tra các cờ TI và RI để xem cờ nào gây ra ngắt để đáp ứng một cách phù hợp.

Ví dụ: Viết chương trình để 8051 đọc dữ liệu từ cổng P1 và ghi nó tới cổng P2 liên tục trong khi đưa một bản sao dữ liệu tới cổng COM nối tiếp để thwucj hiện truyền nối tiếp. Giả thiết tần số XTAL là 11.0592 MHz và tốc độ baud là 9600.

Giải:

```
                ORG   0000H
                LJMP  MAIN
; Trình phục vụ ngắt cổng nối tiếp
                ORG   0023H
                LJMP  SERIAL          ; Nhảy đến chương trình ngắt
MAIN:           MOV   P1,#FFH         ; Xóa cổng P1
                MOV   TMOD,#20H       ; Timer1, chế độ 2 tự nạp lại
                MOV   TH1,#FDH        ; Tốc độ baud = 9600
                MOV   SCON,#50H       ; Khung dữ liệu 8 bit, 1 bit Stop
                MOV   IE,#10010000B   ; Cho phép ngắt nối tiếp
                SETB  TR1
BACK:           MOV   A,P1
                MOV   SBUF,A
                MOV   P2,A
                SJMP  BACK
```

```
; Trình phục vụ ngắt
            ORG   100H
SERIAL:     JB    TI,TRANS        ; Nhảy đến TRANS nếu TI = 1
            MOV   A,SBUF          ; Nếu không tiếp tục nhận dữ liệu
            CLR   RI              ; Xóa RI vì CPU không làm điều này
            RETI
TRANS:      CLR   TI
            RETI
            END
```

Lưu ý rằng trước lệnh RETI là lệnh xóa các cờ RI và TI. Đây là điều cần thiết bởi vì đó là ngắt duy nhất dành cho nhận và truyền. 8051 không biết được nguồn gây ra ngắt là nguồn nào. Do vậy trình phục vụ ngắt phải được xóa các cờ này để cho phép các ngắt sau đó được đáp ứng. Điều này khác với ngắt ngoài và ngắt bộ định thời đều được 8051 xóa các cờ khi kết thúc chương trình phục vụ ngắt.

6.6. Các mức ưu tiên ngắt trong 8051

6.6.1. Các mức ưu tiên mặc định

Khi 8051 được cấp nguồn thì các mức ưu tiên ngắt được gán theo thứ tự từ cao xuống thấp như sau:

Ngắt ngoài 0	INT0
Ngắt bộ định thời 0	TF0
Ngắt ngoài 1	INT1
Ngắt bộ định thời 1	TF1
Ngắt truyền thông nối tiếp	RI, TI

6.6.2. Thiết lập mức ưu tiên ngắt với thanh ghi IP

Chúng ta có thể thay đổi trình tự ưu tiên ngắt bằng cách gán mức ưu tiên cao hơn cho bất kỳ ngắt nào. Điều này được thực hiện bằng cách lập trình thanh ghi ưu tiên ngắt IP (Interrupt Priority). Hình 6.5 là các bit của thanh ghi này. Khi bật lại nguồn thì các bit của thanh ghi IP đều bằng 0 để tạo ra trình tự ưu tiên mặc định Để một ngắt nào đó mức ưu tiên cao hơn thì ta thực hiện đưa bit tương ứng lên cao. Trong hợp nhiều bit ngắt trong thanh ghi IP được đưa lên cao thì thứ tự ưu tiên của các ngắt này lại theo thứ tự mặc định ban đầu.

D7							D0
---	---	PT2	PS	PT1	PX1	PT0	PX0

Hình 6. 5 Thanh ghi ưu tiên ngắt IP

Các bit trong thanh ghi:

- Bit D7 và D6: Chưa dùng.
- Bit D5: Ưu tiên ngắt Timer2
- Bit D4: Ưu tiên ngắt cổng nối tiếp
- Bit D3: Ưu tiên ngắt Timer1
- Bit D4: Ưu tiên ngắt ngoài 1
- Bit D1: Ưu tiên ngắt Timer0
- Bit D0: Ưu tiên ngắt ngoài 0

Ví dụ:

a. Lập trình thanh ghi IP để gán mức ưu tiên cao nhất cho ngắt ngoài 1
b. Phân tích thứ tự ưu tiên ngắt khi INT0, INT1 và TF được kích hoạt cùng lúc.

Giải:

a. MOV IP,#00000100B
b. Lệnh trong bước a gán mức ưu tiên cao hơn INT1 so với các ngắt khác. Do vậy khi INT0, INT1 và TF0 được kích hoạt cùng lúc thì trước hết INT1 được phục vụ, sau đó đến INT0 và cuối cùng là TF0.

TÀI LIỆU THAM KHẢO

1. Văn Thế Minh, "Kỹ thuật vi xử lý", Nhà xuất bản Giáo dục, 1997.
2. Tống Văn On, Hoàng Đức Hải, "Họ vi điều khiển 8051", Nhà xuất bản Lao động - Xã hội, 2001.
3. "ARM Developer Suite – Assembler guide", ARM Limited, 2001.
4. "ARM Software Development Toolkit Version 2.0 - Program Techniques", ARM DUI 0021A, Advanced RISC Machines Ltd, 1995.
5. "ARM v7-M Architecture Application Level Reference Manual", ARM Limited, 2006.
6. "ARM Assembly Language Programming", Pete Cockerell, Computer Concepts Ltd, England.
7. http://www.arm.com/index.php
8. http://www2.keil.com/mdk5/
9. http://armsim.cs.uvic.ca/DownloadARMSimSharp.html.