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

GIÁO TRÌNH

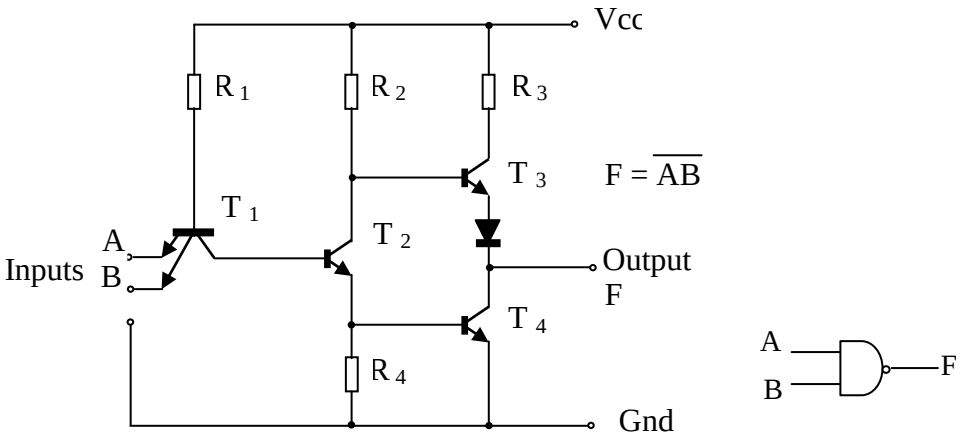
KIẾN TRÚC MÁY TÍNH

NGUYỄN TRUNG ĐỒNG

Chương I. Những kiến thức cơ sở

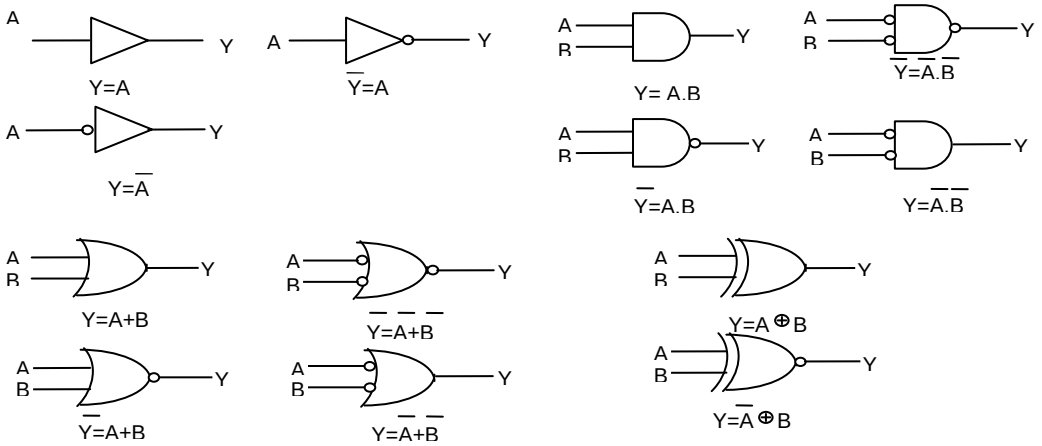
1. Một số phần tử Logic cơ bản

Các mạch logic cơ bản được tạo ra từ liên kết các phần tử điện tử thông dụng là transistor, diode, điện trở, tụ điện,… Tuỳ theo công nghệ chế tạo các phần tử đó mà chúng có những tên gọi khác nhau như logic TTL, logic CMOS, logic HMOS, logic MOSFET v.v…*Hình I.1* cho ta thấy cấu trúc mạch nguyên lý của một phần tử TTL thực hiện chức năng đảo tích logic của hai giá trị đầu vào (NAND).



Hình I.1. Sơ đồ nguyên lý mạch tạo phần tử NAND

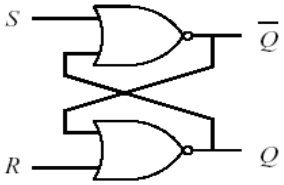
Phần tử logic cơ bản thực hiện các hàm của đại số Boole như NOT, AND, NAND, OR, XOR, v.v…Từ các phần tử này, người ta xây dựng được các mạch tổ hợp (Combinational Circuits) các mạch lật (FlipFlop) với những đặc tính chuyển đổi trạng thái khác nhau như R-S FlipFlop, D-FlipFlop, T-FlipFlop, J-K FlipFlop mà nhờ chúng, ta xây dựng được các mạch tuần tự (Sequencial Circuits) và các máy hữu hạn (Finite State Machine), những mạch tích hợp tạo nên các đơn vị chức năng cơ bản trong máy tính.



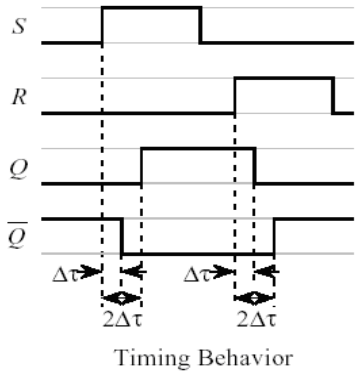
Hình I.2. Một số phần tử logic cơ bản

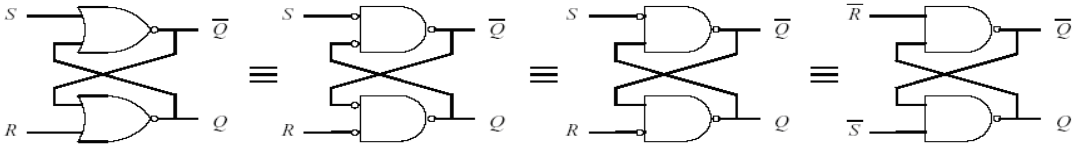
Nguyễn Trung Đồng – Tel 0983 410 866

Mạch FlipFlop RS với tác động tích cực là High "1"

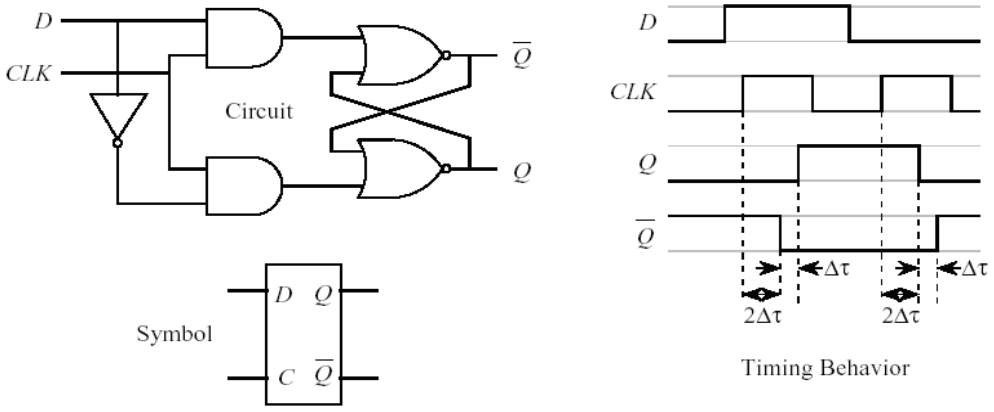


Q_t	S_t	R_t	Q_{t+1}
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	(disallowed)
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	(disallowed)





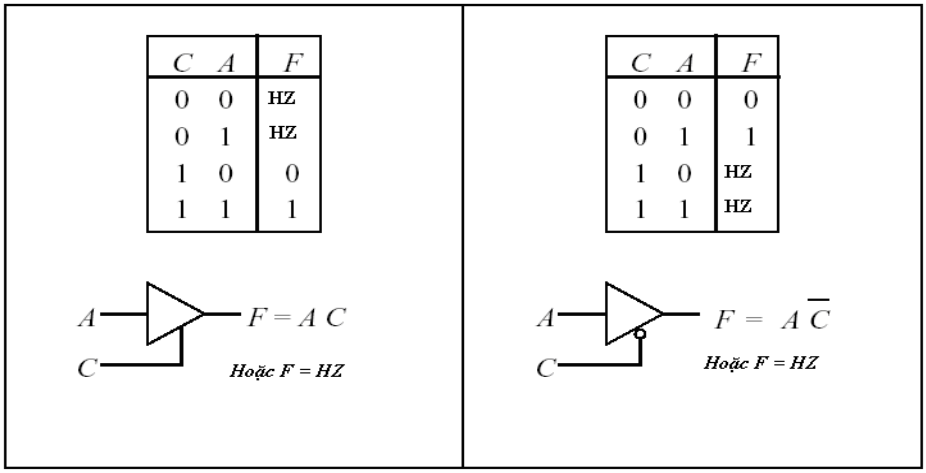
Sử dụng các phần tử logic cơ bản tạo mạch RS-FlipFlop



Hình I.3 . Các phần tử mạch lật (FlipFlop) thông dụng

Đặc biệt, mạch logic 3 trạng thái (Three-State Logic Circuit) là một mạch có ứng dụng rất quan trọng trong việc liên kết các phần tử chức năng của máy tính. Mạch logic 3 trạng thái có thể minh hoạ theo mô hình và bảng chân thực sau (*Hình I.4*), trạng thái có ký hiệu "HZ" là trạng thái thứ 3 của mạch, trạng thái trở kháng cao (High Impedance), khi mà lối vào có thể coi như được tách khỏi lối ra của mạch (không kết nối). Có hai loại mạch 3 trạng thái:, loại mạch có tín hiệu EN là tích cực cao, ứng với EN = "1" (Active High), loại thứ hai là mạch có tín hiệu EN tích cực thấp ứng với EN = "0" (Active Low).

Lối ra có thể là giá trị "0", "1" hoặc là trạng thái không kết nối (HZ)

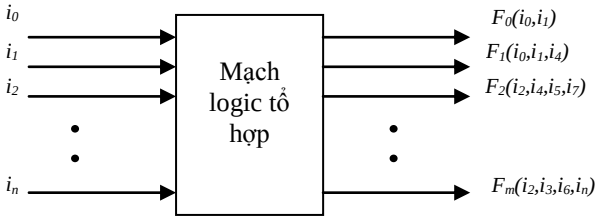


Hình I.4. Phần tử 3 trạng thái (Three-State component) và bảng chân lý

2. *Một số khái niệm cơ sở*

2.1. Mạch logic tổ hợp (Combinational Circuit)

Mạch logic tổ hợp là một mạch điện tử số mà giá trị các biến ở đầu ra chỉ phụ thuộc vào tổ hợp giá trị của các biến ở đầu vào (*Hình I.5*).



Hình I.5. Mạch logic tổ hợp

Các biến vào i_0, i_1, ..., i_n nhận giá trị là "1" hoặc "0" tương ứng với giá trị của một biến nhị phân, trong mạch điện, chúng được thể hiện bằng các trạng thái "có điện áp" hoặc "không có điện áp".

Các giá trị của đầu ra là hàm trực tiếp của các biến đầu vào, và được thay đổi gần như tức thời khi có sự thay đổi giá trị của biến đầu vào (chỉ trễ một khoảng thời gian rất nhỏ - hàng nano giây - do sự trễ của các linh kiện tạo nên mạch điện). Có thể nói *tập các giá trị đầu vào $i_0 \div i_n$ được áp vào các lối vào của mạch tổ hợp logic gây nên sự biến đổi trạng thái (giá trị) của các biến đầu ra $F_0 \div F_m$*. Các mạch tổ hợp thông dụng thường thấy là mạch mã hoá, mạch giải mã, mạch dồn kênh, v.v…

2.2. Mạch tuần tự (Sequencial Circuit)

Mạch này còn được gọi là mạch dãy. Giá trị của biến ra phụ thuộc không những vào giá trị các biến số đầu vào ở thời điểm đang xét, mà còn phụ thuộc vào trạng thái trước đó của mạch. Để duy trì được trạng thái của các biến số vào trước đó, mạch cần thêm các *phần tử nhớ*. Mô hình của mạch như sau:

$Z_i = F_i(x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_p)$;

$Y_j = G_j(x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_p)$

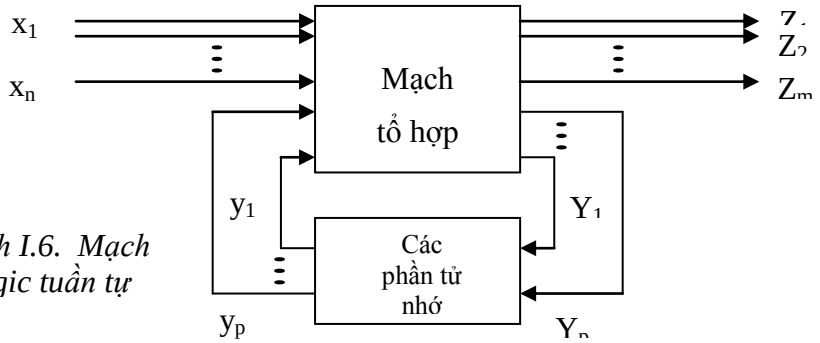
Trong đó F_i là hàm truyền đạt của mạch và G_j là hàm truyền đạt trạng thái;

x_i (i = 1, 2, …,n), Z_i (i = 1, 2, …, m) là các tín hiệu vào và tín hiệu ra của mạch;

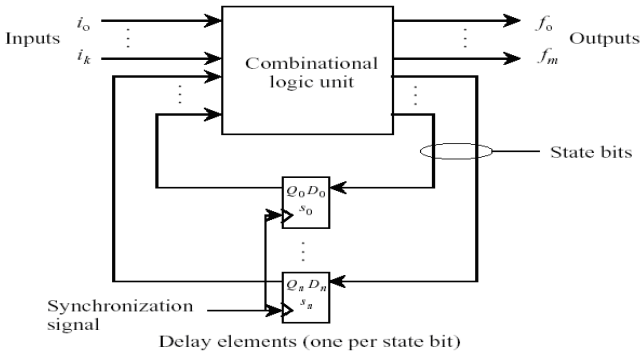
$y_1, y_2, \ldots, y_p$: trạng thái của mạch trước khi biến đổi;

$Y_1, Y_2, \ldots, Y_p$: trạng thái của mạch sau khi biến đổi.

Các phần tử nhớ là các phần tử logic có hai trạng thái ổn định ứng với các giá trị của biến nhị phân "0" và "1", thường là các mạch FlipFlop loại RS, JK hoặc D.



Hình I.6. Mạch logic tuần tự



2.3. Máy hữu hạn (Finite State Machine)

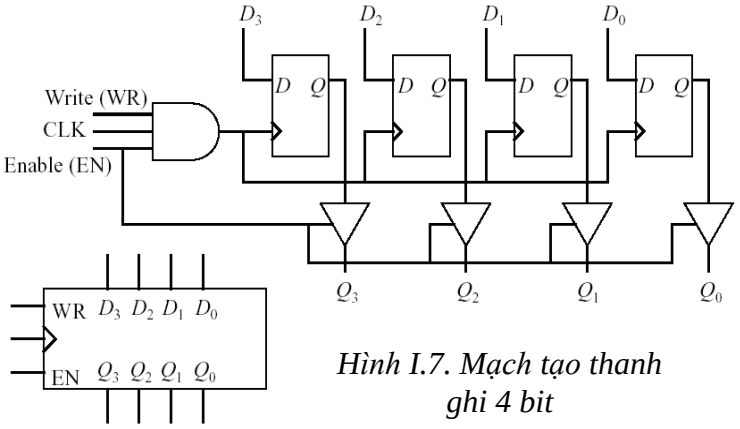
Máy hữu hạn là một loại mạch logic khác có

Nguyễn Trung Đồng – Tel 0983 410 866

trạng thái trong (internal state), đầu ra của loại mạch này là hàm của giá trị đầu vào tại thời điểm đang xét và trạng thái trong hiện tại khi có tác động của tín hiệu vào. Mạch được tạo thành từ một mạch tổ hợp logic và các phần tử trễ, thông thường là các phần tử Flip-Flop trên mạch hồi tiếp như là những phần tử lưu giữ trạng thái trong của mạch.

2.4. Thanh ghi (Register)

Thanh ghi là một mạch điện tử đặc biệt có khả năng lưu giữ các giá trị của một dữ liệu nhị phân được biểu diễn bằng trạng thái tồn tại hay không tồn tại điện áp. Phần tử cơ bản tạo nên một thanh ghi là D-FlipFlop. Trên hình vẽ mô tả, dữ liệu nhị phân 4 bit $D_3D_2D_1D_0$ (tổ hợp của hai giá trị "0" và "1" trên lối vào D tương ứng của các D-FlipFlop) sẽ được chuyển tới lối ra $Q_3Q_2Q_1Q_0$ và lưu giữ nhờ tổ hợp tín hiệu điều khiển ghi Write WR, tín hiệu xung nhịp đồng hồ CLK và tín hiệu cho phép Enable EN (*Hình 1.7*).



Hình I.7. Mạch tạo thanh ghi 4 bit

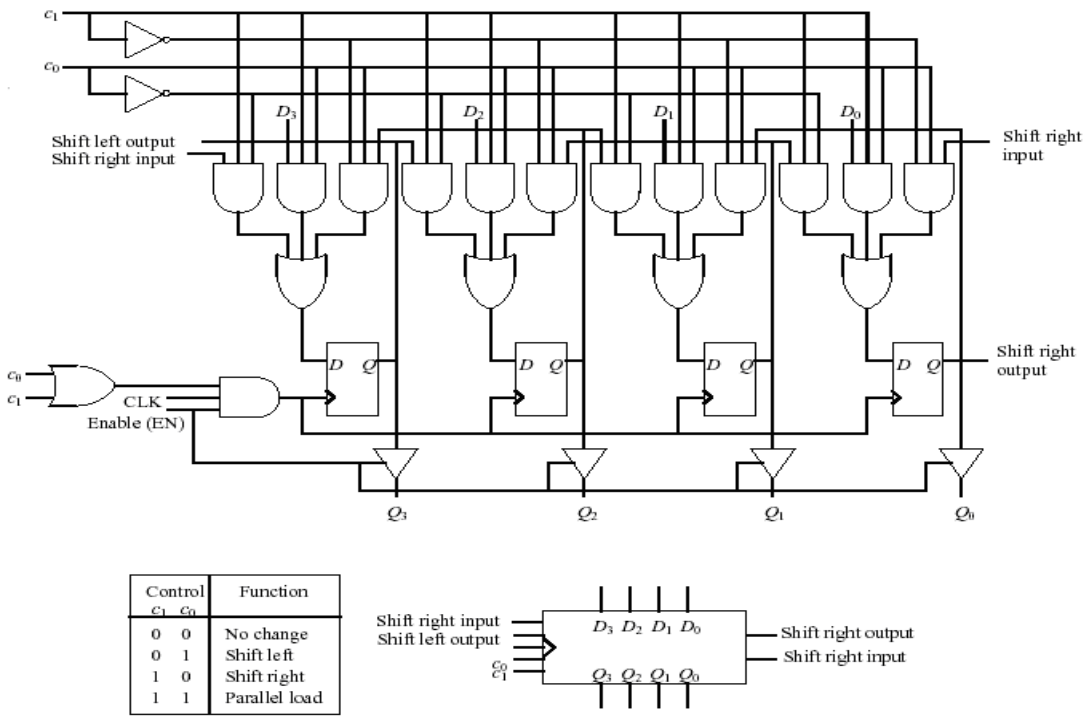
Lưu ý rằng, tín hiệu ra của thanh ghi được đưa qua phần tử 3 trạng thái để tạo khả năng kết nối với những dữ liệu ở lối ra của các thành phần khác.

Cũng cần nói thêm rằng: Thanh ghi hoàn toàn đảm nhận chức năng của một ô nhớ dữ liệu, vì mỗi khi giá trị dữ liệu nhị phân từ lối vào được ghi vào thanh ghi, dữ liệu đó không thay đổi cho đến thời điểm một dữ liệu mới được ghi vào. Dữ liệu lưu giữ trong ô nhớ có thể đọc ra được.

Hình I.9. là sơ đồ nguyên lý của một thanh ghi dịch có khả năng ghi dịch theo các hướng trái, phải hoặc lưu giữ (Load) các dữ liệu nhị phân 4 bit D3D2D1D0 song song.

2.5. Mạch cộng hai số liệu nhị phân (Binary Adder)

Mạch cộng đầy đủ 2 bit nhị phân có thể xây dựng như một mạch tổ hợp logic thực hiện phép cộng hai số nhị phân theo quy tắc trong bảng sau, trong đó Carry In là phần nhớ từ phép cộng của hàng bên phải trước đó, Operand A là giá trị của bit trong toán hạng A, Operand B là giá trị của bit trong toán hạng B. Kết quả phép cộng 2 bit cho ta tổng Sum và bit nhớ Carry Out.



Hình I.8. Sơ đồ nguyên lý mạch tạo thanh ghi dịch 4 bit

Trong ví dụ là phép cộng hai số nhị phân 0100B (giá trị bằng 4 trong hệ thập phân) với số 0110B (giá trị bằng 6 trong hệ thập phân). Hàng trên là giá trị của bit nhớ theo quy luật cộng đã nêu. Kết quả cho ta là 1010B (tức bằng 10 trong hệ thập phân).

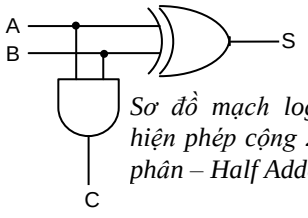
Quy tắc cộng hai bit nhị phân có nhớ và ví dụ một phép cộng hai số nhị phân 4 bit

Carry In →	0	0	0	0	1	1	1	1
Operand A →	0	0	1	1	0	0	1	1
Operand B →	+ 0	+ 1	+ 0	+ 1	+ 0	+ 1	+ 0	+ 1
	0 0	0 1	0 1	1 0	0 1	1 0	1 0	1 1

Carry Out | Sum

Example:

Carry	1	0	0	0
Operand A	0	1	0	0
Operand B +	0	1	1	0
Sum	1	0	1	0

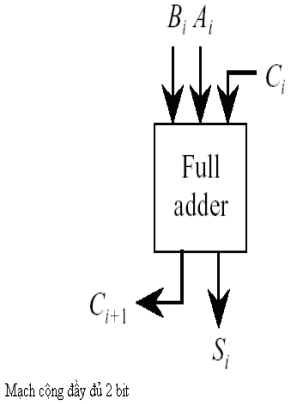


Sơ đồ mạch logic thực hiện phép cộng 2 bit nhị phân – Half Adder (HA)

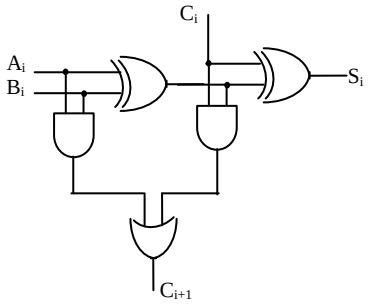
Từ quy tắc trên, giả thiết ta xây dựng được một mạch cộng đầy đủ thực hiện phép toán cộng như bảng giá trị của hàm S_i và C_i và ký hiệu là một mạch cộng đầy đủ (Full adder) với các đầu vào là A_i , B_i và C_i , đầu ra là S_i và C_{i+1}, ta có thể xây dựng mạch cộng hai dữ liệu nhị phân 4 bit bằng cách

nối nối tiếp 4 mạch cộng đầy đủ như *Hình I.11.* , hoặc mạch cộng hai số nhị phân *n* bit với *n* mạch cộng đầy đủ.

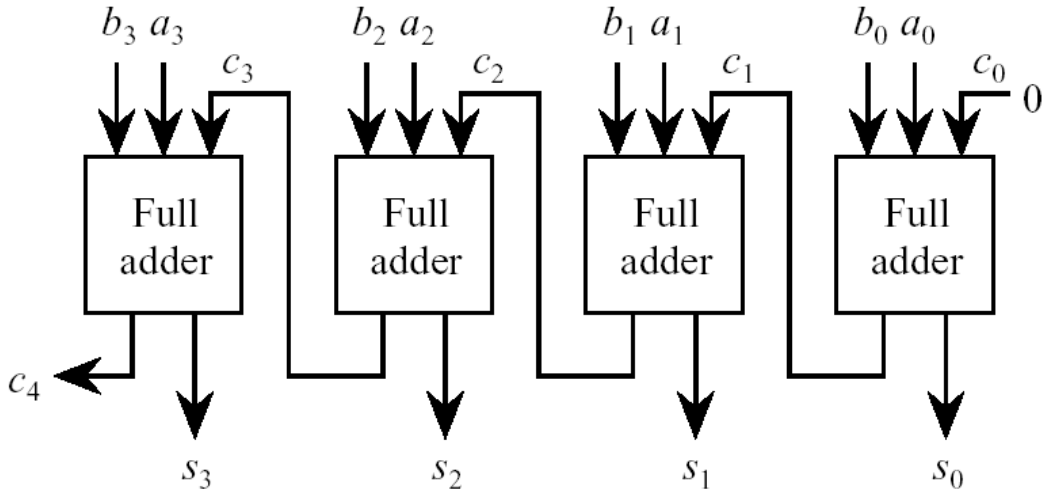
A_i	B_i	C_i	S_i	C_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



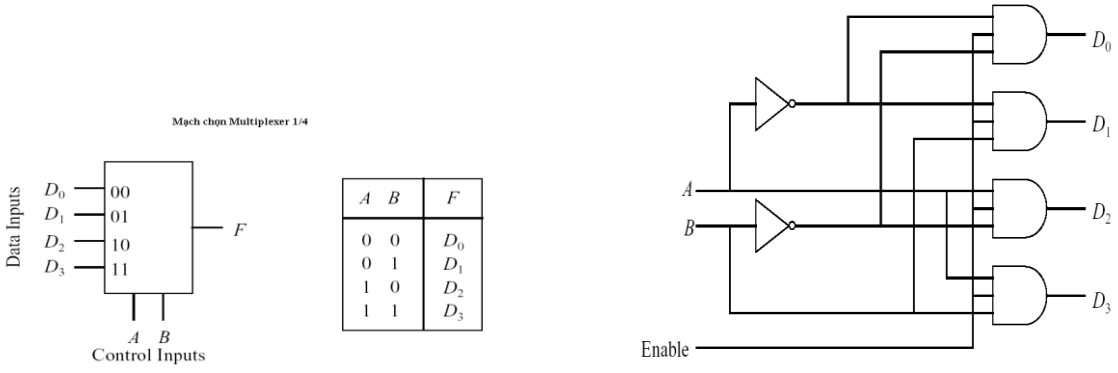
Mạch cộng đầy đủ 2 bit



Hình I.10. Sơ đồ mạch logic thực hiện phép cộng 2 bit có nhớ từ hàng trước – FullAdder (FA)



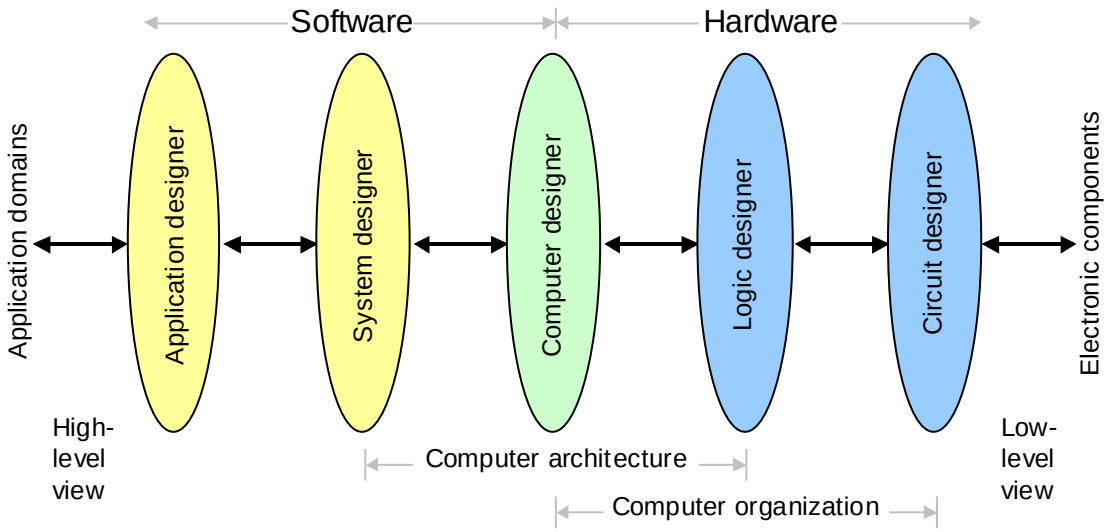
Hình I.11. Sơ đồ mạch logic thực hiện phép cộng 2 dữ liệu 4 bit



Hình I.12. Sơ đồ mạch logic thực hiện phép giải mã chọn 1 trong 4 tổ hợp

Ngoài ra, có thể tham khảo thêm các mạch dồn kênh, mạch mã hoá và giải mã trong các tài liệu Kỹ thuật điện tử số được nêu trong tài liệu tham khảo ở cuối giáo trình này.

Lưu đồ trong *Hình I.13* cho ta thấy sơ lược các bước cơ bản trong quá trình thiết kế một máy tính và phạm vi nghiên cứu về Kiến trúc và tổ chức máy tính.



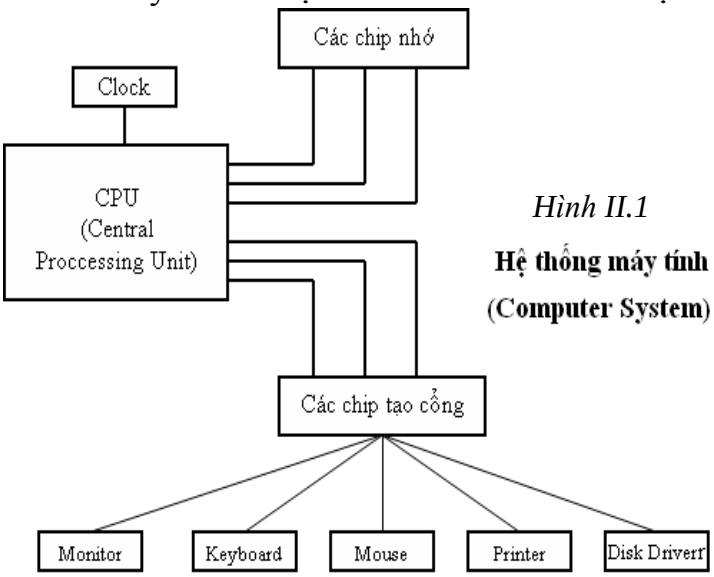
Nguyễn Trung Đồng – Tel 0983 410 866

Chương II. Giới thiệu chung

1. Máy tính và kiến trúc máy tính

1.1. Mở đầu

Máy tính được cấu thành từ các mạch điện tử tích hợp (integrated circuits – IC) rất phức tạp liên kết với nhau qua hệ thống kênh truyền dẫn được gọi là hệ thống BUS. Các khối chức năng cơ bản được xây dựng với công nghệ tích hợp mật độ lớn gồm đơn vị xử lý trung tâm (CPU – Central Proccessing Unit), khối tạo xung nhịp (Clock), bộ nhớ (Memorry) và các chip tạo các cổng (Port Chips) ghép nối thiết bị ngoại vi như minh hoạ trên *Hình II.1*

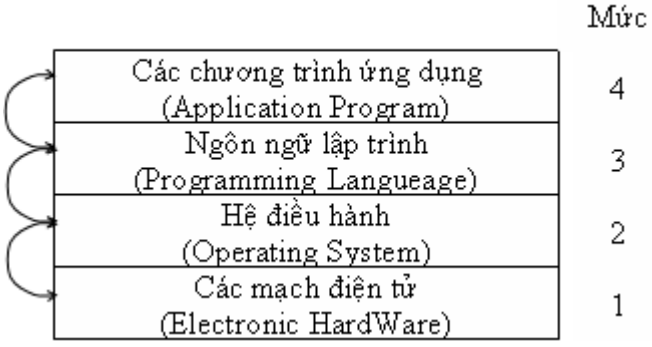


Hình II.1

Hệ thống máy tính

(Computer System)

CPU được xây dựng từ các mạch điện tử phức tạp, có khả năng thực thi tất cả các lệnh trong tập lệnh được mô phỏng trước. Bộ nhớ được xây dựng từ các chip nhớ, có khả năng lưu giữ các lệnh của chương trình và dữ liệu. Các chip tạo cổng điều khiển việc truy xuất đến các thiết bị ngoại vi như bàn phím (Keyboard), chuột (Mouse), màn hình (Monitor), máy in (Printer), các ổ đĩa (Disk Drivers). CPU chỉ truy xuất dữ liệu đến từ (input) và đi ra (output) thiết bị ngoại vi thông qua các chip tạo cổng.



Cấu trúc chức năng của máy tính được mô phỏng trên *Hình II.1*, Hệ điều hành và Ngôn ngữ lập trình bậc cao điều khiển hoạt động của các mạch điện tử trong máy tính. Khi cấp nguồn, chương trình khởi tạo hệ thống sẽ nạp hệ điều hành

Nguyễn Trung Đồng – Tel 0983 410 866

(boot hệ thống), ngôn ngữ lập trình sẽ được tải vào bộ nhớ nhờ hệ điều hành.

Ở mức trên cùng, máy tính có thể thực thi các chương trình ứng dụng. Các chương trình ứng dụng được sử dụng nhiều như tạo các bảng tính, tạo văn bản, các bản vẽ, …, được viết bằng các ngôn ngữ lập trình khác nhau như C, C++, hoặc là liên kết giữa các ngôn ngữ. Người ta sử dụng ngôn ngữ lập trình trong mối liên kết với hệ điều hành để điều khiển hoạt động chức năng của phần cứng. Ngôn ngữ máy là ngôn ngữ duy nhất bao gồm các chỉ lệnh (*Instruction*) mà phần cứng có thể hiểu và thực thi, được tạo ra từ các tổ hợp các số biểu diễn theo hệ nhị phân. Các mã nhị phân này được gọi là mã lệnh, chúng tạo nên tập lệnh của CPU, giá trị "0" hoặc "1" làm nhiệm vụ "ngắt" hoặc "đóng" dòng điện để điều khiển hoạt động của các phần tử logic trong mạch điện. Cần hiểu rằng, tất cả các CPU đều làm việc với mã máy. Một khi sử dụng ngôn ngữ lập trình bậc cao, sử dụng các phát biểu (*Statements*), các chương trình dịch (Compiler) sẽ chuyển đổi (dịch) chúng ra mã máy để CPU hiểu và thực hiện.

Mặc dù vậy, vẫn có thể nói máy tính bao giờ cũng được cấu thành từ các khối chức năng chính sau:

1. Bộ nhớ trung tâm (*Central Memory hoặc Main Memory*). Bộ nhớ trung tâm là nơi lưu giữ chương trình và dữ liệu trước khi chương trình được thực hiện.
2. Đơn vị điều khiển (*CU - Control Unit*), điều khiển mọi hoạt động của tất cả các thành phần trong hệ thống máy tính theo chương trình mà máy tính cần thực hiện.
3. Đơn vị số học và Logic (*ALU – Arithmetic & Logic Unit*), thực hiện các thao tác xử lý dữ liệu thông qua các phép toán số học và Logic theo sự điều khiển của Đơn vị điều khiển. *Đơn vị điều khiển CU và đơn vị số học-logic ALU được tích hợp trong một chip IC và được gọi là Đơn vị xử lý Trung tâm (CPU-Central Proccessing Unit)*.
4. Thiết bị vào (*Input Device*) thực hiện nhiệm vụ thu nhận các thông tin, dữ liệu từ thế giới bên ngoài, biến đổi thành dạng tương thích với phương thức biểu diễn trong máy tính, đưa vào CPU xử lý hoặc ghi vào bộ nhớ.
5. Thiết bị ra (*Output Device*) thực hiện nhiệm vụ đưa thông tin, dữ liệu từ CPU hoặc bộ nhớ ra ngoài dưới các dạng thức được người sử dụng yêu cầu. *Thiết bị vào và thiết bị ra được gọi chung là nhóm thiết bị ngoại vi (Peripherals)*.

Sau đây ta sẽ tìm hiểu nguyên lý kiến trúc và hoạt động của một máy tính thông qua một máy tính đơn giản nhất.

Máy tính, ở dạng đơn giản nhất, được cấu thành từ bốn khối chức năng cơ bản sau:

- Khối điều khiển và xử lý dữ liệu: Khối chức năng này được tích hợp trong cùng một vi mạch gọi là ***Đơn vị xử lý trung tâm*** **(CPU – Central Proccessing Unit)**.
- Khối lưu trữ dữ liệu được gọi là ***bộ nhớ*** **(Memory)**.
- Khối chức năng cung cấp dữ liệu cho máy tính xử lý, hoặc phản ánh dữ liệu đã được xử lý do máy tính cung cấp, được gọi là ***khối các thiết bị nhập xuất*** **(I/O devices)**.
- Các kênh truyền dẫn cung cấp sự liên lạc và trao đổi dữ liệu giữa các khối trên, được gọi là ***kênh liên kết hệ thống*** **(BUS)**.

Trong một máy tính, mỗi khối thực hiện các chức năng nói trên có thể tồn tại nhiều đơn vị, dưới các dạng khác nhau, trong đó CPU là quan trọng nhất.

Đơn vị xử lý trung tâm (CPU) có thể xử lý được các lệnh với khuôn dạng từ lệnh, *giả sử với độ dài 8 bit*, như sau:

B7	B6	B5	B4	B3	B2	B1	B0
Phần chứa mã lệnh				Phần chứa địa chỉ toán hạng			

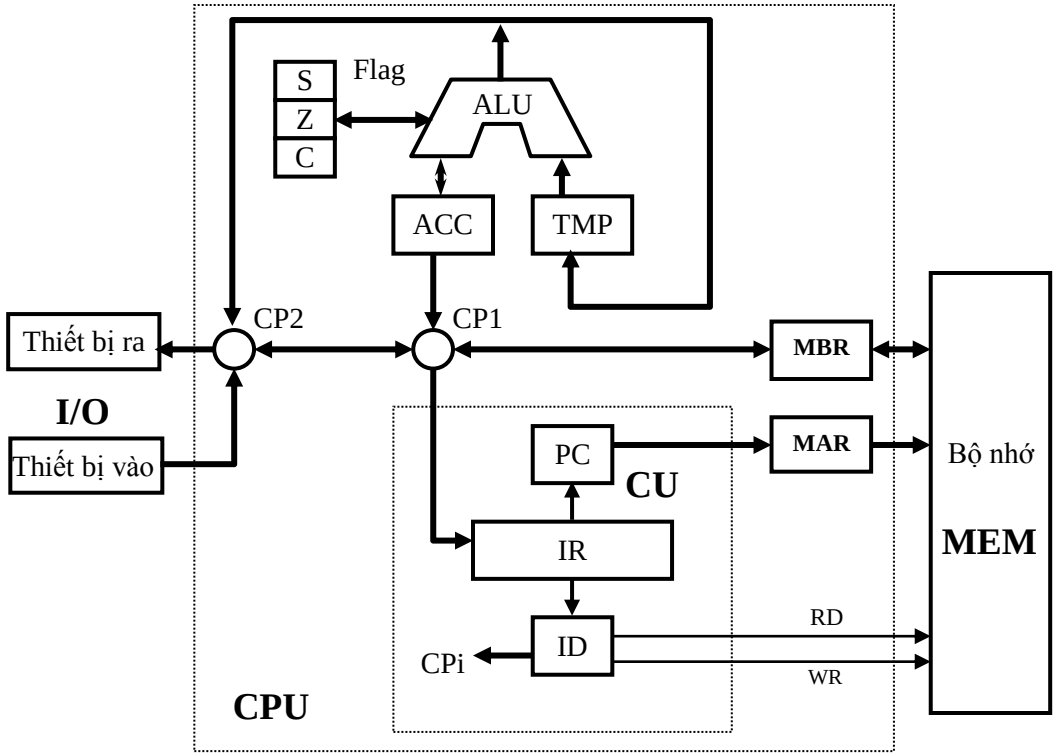
Lệnh được tạo từ hai phần: Mã lệnh và địa chỉ toán hạng

Mã lệnh là một giá trị nhị phân 4 bit, mỗi tổ hợp là một lệnh có chức năng khác nhau, phần chứa địa chỉ toán hạng cũng là một giá trị nhị phân 4 bit, xác định vị trí của ô nhớ trong bộ nhớ. Phần địa chỉ xác định toán hạng mà lệnh trực tiếp xử lý.

Đơn vị xử lý trung tâm gồm hai thành phần chức năng: Đơn vị số học-logic ALU (Arithmetic-Logic Unit) và đơn vị điều khiển CU (Control Unit) (*Hình II.2.*).

Đơn vị điều khiển CU có chức năng lấy lệnh theo tuần tự được lưu giữ từ trong bộ nhớ, giải mã lệnh và tạo các tín hiệu điều khiển hoạt động của các khối chức năng bên trong và bên ngoài CPU.

Lệnh đọc từ ô nhớ được đưa vào thanh ghi lệnh IR, được giải mã tại khối giải mã lệnh ID để xác định công việc CPU cần thực hiện.



Hình II.2. Sơ đồ cấu trúc máy tính đơn giản

Đơn vị điều khiển CU gồm thanh ghi lệnh IR (*Instruction Register*), là nơi chứa lệnh mà CPU đọc về từ ô nhớ lệnh, bao gồm cả phần mã lệnh và phần địa chỉ toán hạng, khối giải mã lệnh ID (*Instruction Decoder*), mạch giải mã này giải mã lệnh để xác định nhiệm vụ mà lệnh yêu cầu CPU xử lý, tạo các tín hiệu điều khiển các tác vụ của CPU khi thực thi lệnh và thanh đếm chương trình PC (*Program Counter*). Thanh đếm chương trình PC làm nhiệm vụ con trỏ lệnh (*Instruction Pointer*), chứa địa chỉ của ô nhớ chứa lệnh sẽ thực thi trong tuần tự thực hiện chương trình. Do vậy sau khi CPU đọc được một lệnh từ bộ nhớ chương trình, sau khi được giải mã, thông qua điều khiển của CU thì PC được tăng nội dung lên để chỉ vào ô nhớ chứa lệnh tiếp theo. Trong trường hợp gặp lệnh rẽ nhánh hay lệnh gọi chương trình con, nội dung thanh đếm PC thay đổi tuỳ theo giá trị địa chỉ mà chương trình dịch gán cho nhãn hay tên chương trình con được xác định bởi người lập trình.

CPU có các thanh ghi: thanh ghi gộp (*Acc – Accummulator*), thanh ghi tạm thời TEMP (*temporary*), thanh ghi đệm địa chỉ MAR (Memory Address Register), thanh ghi đệm bộ nhớ MBR (*Memory Buffer Register*), và thanh

ghi cờ *Flags*. Thanh ghi Acc được sử dụng để chứa nội dung một toán hạng, và thông thường là nơi chứa kết quả thực hiện phép toán, thanh ghi tạm thời chứa nội dung toán hạng thứ hai trong các phép toán hai ngôi. Nội dung thanh ghi MAR là địa chỉ của ô nhớ mà CPU đang truy xuất, còn nội dung thanh ghi MBR là dữ liệu đọc được từ bộ nhớ hoặc sẽ được ghi vào ô nhớ. Thanh ghi cờ Flags gồm các bit biểu diễn trạng thái của kết quả thực hiện phép toán xử lý dữ liệu của CPU, Trong trường hợp đơn giản, thanh ghi cờ có 3 bit, đó là bit dấu (*S – Sign*) biểu diễn giá trị dữ liệu là âm hay dương, bit *không* (*Z-Zero*) biểu diễn kết quả phép toán khác 0 hay bằng 0, bit nhớ (*C – Carry*) biểu diễn trạng thái kết quả phép toán có bit nhớ hay không có bit nhớ. Giá trị các bit cờ trạng thái được định nghĩa như sau:

Kết quả là một số âm: (S) = 1 ; dấu ngoặc thể hiện nội dung của bit

Kết quả bằng 0: (Z) = 1

Kết quả có nhớ: (C) = 1

Hoạt động thực thi một lệnh trong chương trình của máy tính có thể tóm tắt như sau:

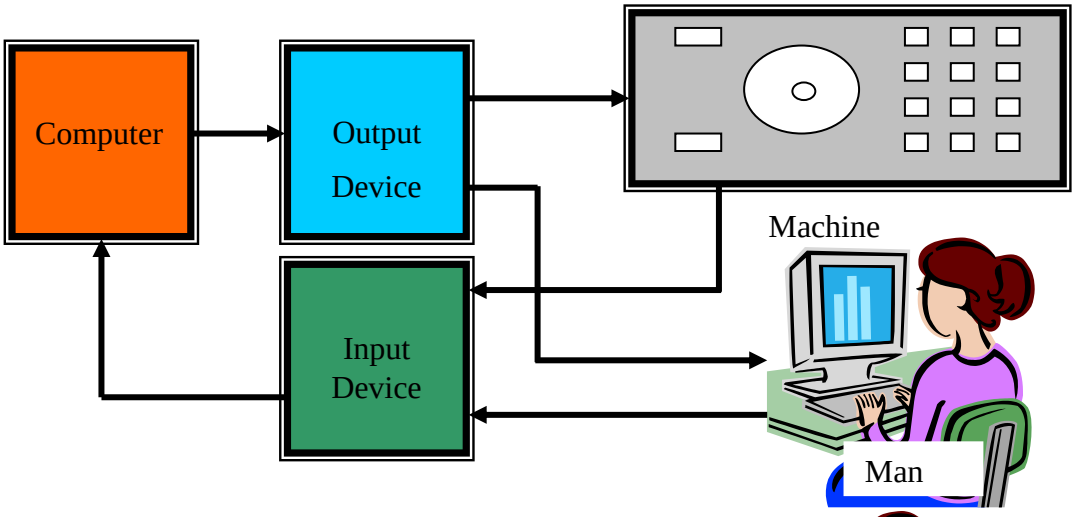
Chương trình và số liệu ban đầu được lưu giữ ở bộ nhớ trung tâm, đó là bộ nhớ ROM (Read Only Memory), RAM (Random Access Memory).

1. Bắt đầu chương trình, lệnh đầu tiên của chương trình trong vùng nhớ chương trình được đưa vào thanh ghi lệnh IR của đơn vị điều khiển (CU). Tác vụ được gọi là tác vụ *nhận lệnh (Instruction Fetch)*.
2. CU tiến hành giải mã lệnh, xác định nội dung phép toán cần xử lý là phép tính nào, trên các dữ liệu nào. Đây là tác vụ *giải mã lệnh (ID – Instruction Decoder)*.
3. Nếu lệnh đòi hỏi làm việc với các toán hạng (được xác định trong lệnh), CU xác định địa chỉ tương ứng của toán hạng trong vùng nhớ dữ liệu hoặc được nhập vào từ thiết bị ngoại vi. Tác vụ này được gọi là *tạo địa chỉ toán hạng (GOA - Generate Operand Address)*.
4. Sau khi địa chỉ toán hạng được tạo, CU phát các tín hiệu điều khiển tới các thành phần liên quan để nhận toán hạng, đặt vào các thanh ghi xác định trong CPU. Tác vụ được gọi là *nhận toán hạng (Operand Fetch)*.

5. CU phát các tín hiệu điều khiển tới Đơn vị Số học-Logic (ALU). ALU thực hiện phép toán được yêu cầu trong mã lệnh. Tác vụ này gọi là *thực hiện (Execute)*.
6. Kết quả xử lý được đặt trong thanh ghi gộp (Acc) hoặc được lưu vào bộ nhớ trong tuỳ thuộc sự xác định nơi lưu giữ thể hiện đích (destination) trong lệnh. Tác vụ được gọi là *Ghi lại kết quả (Write Back)*.

Trong trường hợp CU giải mã lệnh và đó là lệnh rẽ nhánh chương trình, CU tính địa chỉ ô nhớ chứa lệnh cần thực hiện tiếp và phát tín hiệu điều khiển để nhận lệnh về, công việc được tiến hành tuần tự như từ bước. Nếu không phải là lệnh rẽ nhánh chương trình, CPU phát các tín hiệu điều khiển để lấy về lệnh kế tiếp trong ô nhớ đứng ngay sau lệnh vừ thực hiện, hoạt động xẩy ra như từ bước 2.

Khi máy tính được sử dụng để giám sát hay điều khiển một quá trình thực, việc giao tiếp giữa máy tính và con người được mô tả đơn giản hoá như ở *Hình II.3*. Thông qua chương trình giao tiếp và các thiết bị Vào/Ra, con người làm nhiệm vụ giám sát hoặc điều khiển hoạt động của máy móc hoặc quá trình.



Hình II.3. Giao diện Máy tính - Con người - Máy móc

Ở mức độ đơn giản và phổ biến nhất, con ngu trực tiếp với máy tính thông qua các thiết bị Vào và thiết bị Ra c thiết bị này được gọi một tên chung là thiết bị ngoại vi (*Peripherals hoặc I/O Devices*). Con người gửi yêu cầu (lệnh hoặc dữ liệu) vào máy ách sử dụng thiết bị nhập dữ liệu, máy tính xử lý dữ liệu và sau kh xong gửi trả kết quả ra thiết bị xuất dữ liệu. Ở mức độ cao hơn ử dụng máy tính để điều khiển một đối tượng thứ ba (máy móc hoặ). Con người gửi tín hiệu điều khiển vào máy tính, máy tính xử lý u được cung

Ngu Tel 0983 410 866

cấp và trực tiếp gửi yêu cầu tới thiết bị để thiết bị thực hiện các thao tác đáp ứng yêu cầu của con người. Máy tính cũng có thể gửi kết quả xử lý ra thiết bị xuất để con người kiểm tra lại yêu cầu của mình.

Với một mức độ tự động hóa cao hơn, con người chỉ gửi chương trình điều khiển thiết bị vào máy tính một lần, máy tính nhận dữ liệu, xử lý dữ liệu và gửi yêu cầu tới thiết bị. Về phần mình, thiết bị, sau khi đáp ứng yêu cầu của con người sẽ gửi trả kết quả về máy tính và trên cơ sở đó máy tính sẽ xử lý và gửi các tín hiệu điều khiển tiếp theo tới thiết bị.

Như vậy máy tính là một thực thể có thể tương tác với môi trường bên ngoài. Máy tính nhận thông tin từ bên ngoài, xử lý thông tin nhận được và gửi trả lại kết quả. Đây là mối quan hệ trao đổi hai chiều, song luôn luôn xuất phát từ yêu cầu của con người. Máy tính không thể tự mình khởi đầu quá trình này.

1.2. Chức năng của máy tính

Chức năng của máy tính là ***thực hiện chương trình thông qua xử lý một tập lệnh do người lập trình cung cấp.*** *Chương trình là tập hợp các lệnh được người lập trình chọn lọc và sắp xếp theo một tuần tự chặt chẽ thông qua nguyên tắc xử lý, giải quyết một vấn đề cụ thể (hay còn gọi là thuật giải)*.

Để thực hiện chức năng này, chương trình được lưu giữ trong bộ nhớ, việc thực hiện chương trình thực chất là các tác vụ thực thi lệnh theo tuần tự đã được người lập trình quy định. Quá trình thực thi 1 lệnh, như đã trình bày ở trên, gồm các giai đoạn sau:

1. Nhận lệnh *IF-Instruction Fetch*
2. Giải mã lệnh *ID-Instruction Decoder*
3. Tạo địa chỉ toán hạng *GOA-Generate Operand Address*
4. Nhận toán hạng *OF-Operand Fetch*
5. Xử lý lệnh *EX-Execute*
6. Lưu kết quả *WB-Write Back*

Instruction Fetch	Instruction Decode	Generate Operand Address	Operand Fetch	Execute	Write Back
IF	**ID**	**GOA**	**OF**	**EX**	**WB**

Thời gian

Nguyễn Trung Đồng – Tel 0983 410 866

Việc đảm bảo thực hiện chương trình theo tuần tự, như đã nói ở trên, là do CU đảm nhận thông qua việc điều khiển sự thay đổi nội dung của thanh đếm chương trình PC. Tuần tự các lệnh trong chương trình là do người lập trình quyết định thông qua việc viết chương trình theo thuật giải.

Khi thực hiện một chương trình, thông thường máy tính thực hiện các công việc sau:

Thứ nhất, *Xử lý dữ liệu*: Xử lý các yêu cầu của con người/thiết bị trên cơ sở các dữ liệu được nhập vào. Đây là chức năng quan trọng nhất. Dữ liệu có thể ở nhiều dạng khác nhau và các yêu cầu xử lý cũng rất khác biệt. Tuy nhiên máy tính chỉ có thể thực hiện được một số lượng hữu hạn các thao tác xử lý cơ bản, người lập trình dựa trên các khả năng xử lý đó mà tạo ra những khả năng xử lý các vấn đề lớn hơn và phức tạp hơn thông qua công việc lập trình.

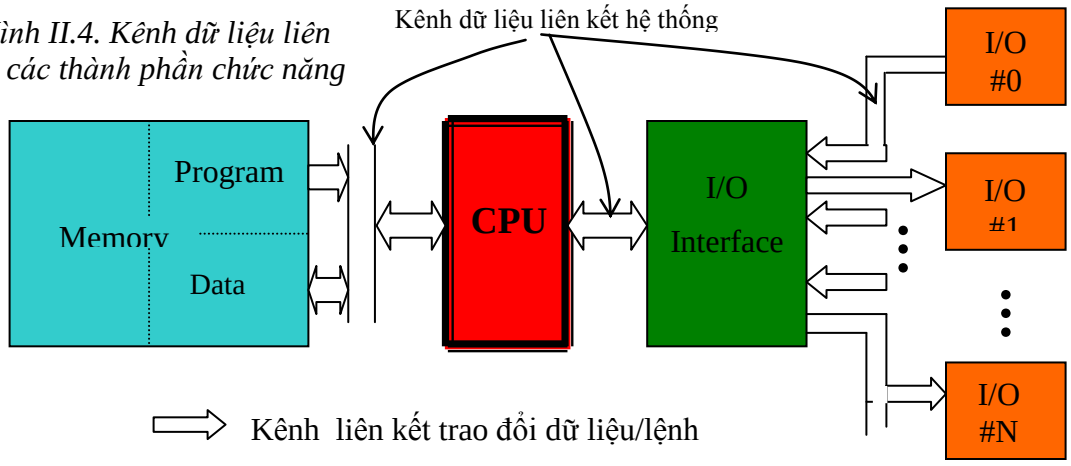
Thứ hai, *Lưu trữ dữ liệu*: Muốn công việc xử lý dữ liệu đạt hiệu quả cao, máy tính phải có khả năng lưu trữ tạm thời dữ liệu và lưu trữ dữ liệu dài hạn để tái sử dụng sau này.

Thứ ba, *Di chuyển dữ liệu*: Để phục vụ việc xử lý, dữ liệu phải có thể di chuyển từ điểm này tới điểm khác bên trong máy tính. Ngoài ra, để có dữ liệu cho xử lý và gửi kết quả ra bên ngoài, máy tính phải có khả năng trao đổi dữ liệu với môi trường bên ngoài.

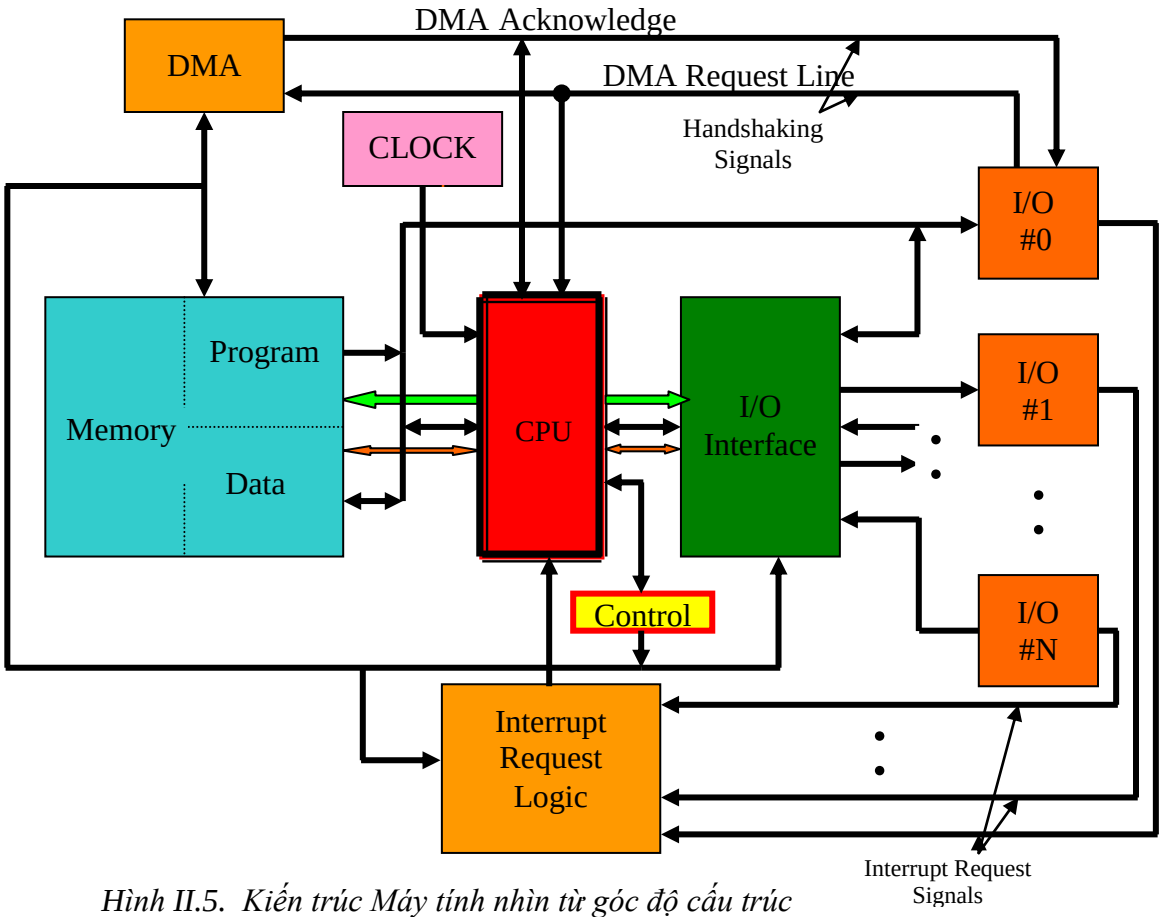
Thứ tư, *Điều khiển*: Để thực hiện có hiệu quả ba chức năng nói trên, các tác vụ máy tính thực hiện phải được điều khiển một cách đồng bộ và hợp lý. Quy trình điều khiển này sẽ được thực hiện nhờ con người cung cấp lệnh cho máy tính thi hành thông qua một đơn vị điều khiển bên trong máy tính.

Kiến trúc máy tính phải được thiết kế để máy tính có khả năng thực hiện những công việc này.

Hình II.4. Kênh dữ liệu liên kết các thành phần chức năng



Nguyễn Trung Đồng – Tel 0983 410 866



Hình II.5. Kiến trúc Máy tính nhìn từ góc độ cấu trúc

1.3. Kiến trúc máy tính và cấu trúc máy tính

Để tìm hiểu kiến trúc máy tính, cần phân biệt rõ sự khác nhau cơ bản, thuộc về nguyên lý giữa *kiến trúc (architecture)* và *tổ chức và cấu trúc (organization & structure)* của một máy tính:

- *Kiến trúc máy tính nghiên cứu những thuộc tính của một hệ thống* mà người lập trình có thể nhìn thấy được, *những thuộc tính quyết định trực tiếp đến việc thực thi một chương trình* tính toán, xử lý dữ liệu
- *Cấu trúc máy tính nghiên cứu về các thành phần chức năng và sự kết nối giữa chúng để tạo nên một máy tính*, nhằm thực hiện những chức năng và tính năng kỹ thuật của kiến trúc.

Những thuộc tính liên quan đến kiến trúc bao gồm ***tập lệnh cơ bản mà CPU có thể thực hiện, số bit được sử dụng để biểu diễn các loại dữ liệu khác nhau, cơ chế nhập/xuất dữ liệu, và các kỹ thuật đánh địa chỉ ô nhớ, v.v...*** Cấu trúc máy tính lại bao gồm các thuộc tính kỹ thuật mà người lập trình không nhận biết được như các tín hiệu điều khiển, giao diện giữa máy tính và thiết bị ngoại vi, công nghệ xây dựng bộ nhớ, v.v…

Chẳng hạn việc quyết định máy tính có cần một lệnh cơ bản để thực hiện phép nhân hay không là vấn đề về kiến trúc. Còn thể hiện lệnh nhân bằng các đơn vị vật lý cụ thể nào (chẳng hạn, một đơn vị thuộc phần cứng đặc biệt, hay thực hiện lặp nhiều phép cộng) lại là vấn đề về cấu trúc.

Để làm ví dụ minh họa sự khác biệt đó ta có thể xem các máy tính ở Trung tâm nghiên cứu nào đó. Các máy tính này có thể có kiến trúc rất giống nhau theo quan điểm của người lập trình. Chúng có cùng số thanh ghi (tức là thiết bị lưu trữ tạm thời), có cùng một tập lệnh cơ bản và dạng các toán hạng được nạp vào bộ nhớ giống nhau. Tuy nhiên các hệ thống này khác nhau về mặt cấu trúc: số bộ vi xử lý khác nhau, kích thước bộ nhớ của chúng cũng khác hẳn nhau, cách thức dữ liệu được truyền từ bộ nhớ đến bộ vi xử lý cũng không giống nhau.

Kiến trúc máy tính thường được ứng dụng trong khoảng thời gian dài, hàng chục năm; trong khi cấu trúc thường thay đổi cùng với sự phát triển của công nghệ. Trên cùng một kiến trúc, các hãng chế tạo máy tính có thể đưa ra nhiều loại máy tính khác nhau về cấu trúc, do đó các đặc trưng về hiệu suất, giá thành cũng khác nhau. Các sản phẩm của IBM là một ví dụ điển hình. Kiến trúc máy tính của IBM vẫn còn được ứng dụng cho tới ngày nay và là ngọn cờ của thương hiệu IBM.

Trong lĩnh vực máy PC, người ta thường không phân biệt rõ ràng giữa kiến trúc và cấu trúc vì sự khác biệt giữa hai khái niệm này đã rút ngắn đáng kể. Sự phát triển của công nghệ không chỉ tác động lên cấu trúc mà còn tạo điều kiện phát triển các kiến trúc mạnh hơn và nhiều tính năng hơn; và do đó tác động qua lại giữa kiến trúc và cấu trúc thường xuyên hơn.

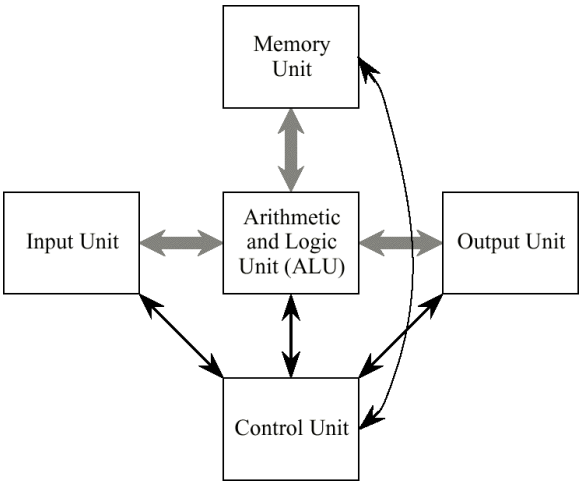
Ngoài kiến trúc máy tính và cấu trúc máy tính còn có một lĩnh vực là kỹ thuật máy tính nghiên cứu việc xây dựng cụ thể các hệ thống: chẳng hạn như độ dài dây dẫn tạo BUS, kích cỡ các vi mạch, v.v. Người lập trình thường cần đến kiến thức về kiến trúc, đôi khi cần những hiểu biết về cấu trúc, nhưng thường rất ít khi cần đến những hiểu biết về kỹ thuật máy tính.

Hiểu kiến trúc máy tính có thể giúp người lập trình nhận biết khi nào chương trình của mình tạo ra chạy chưa đạt hiệu suất tối đa của hệ thống, hiểu được các kỹ năng làm tăng hiệu suất chương trình, v.v.

1.4. Kiến trúc máy tính Von Neumann

John von Neumann (Neumann János, 28 tháng 12, 1903 – 8 tháng 2, 1957) là một nhà toán học người Hungary và là một nhà bác học thông thạo nhiều lĩnh vực, đã có nhiều đóng góp vào các chuyên ngành vật lý lượng tử, giải tích hàm, lý thuyết tập hợp, kinh tế, khoa học máy tính, giải tích số, thủy

động lực học , thống kê và nhiều lĩnh vực toán học khác. Đáng chú ý nhất, von Neumann là nhà tiên phong của *máy tính kỹ thuật số* hiện đại và áp dụng của lý thuyết toán tử (*operator theory*) vào cơ học lượng tử.



Năm 1945, ông đã đưa ra một đề nghị về kiến trúc máy tính như sau:

- *Lệnh (Instruction) và dữ liệu (Data) phải được lưu giữ trong một bộ nhớ ghi/đọc được.*
- *Từng ô nhớ trong bộ nhớ phải được định vị bằng địa chỉ. Sự định địa chỉ là tuần tự và không phụ thuộc vào nội dung của từng ô nhớ.*
- *Chương trình xử lý, giải bài toán phải thực hiện tuần tự từ lệnh này đến lệnh tiếp theo, từ lệnh bắt đầu đến lệnh cuối cùng.*

2. Tổng quan về kiến trúc máy tính

Trên cơ sở nguyên lý kiến trúc Von Neumann, máy tính là một hệ thống bao gồm đơn vị xử lý trung tâm, bộ nhớ và các thiết bị vào/ra được kết nối với nhau. Hệ thống đường truyền dẫn liên kết các khối là một trong những vấn đề mà các hãng chế tạo máy tính gặp nhiều nan giải nhất. Để nắm được những kiến thức cơ bản về kiến trúc, chúng ta sẽ bắt đầu bằng việc tìm hiểu về hướng phát triển kiến trúc thông qua liên kết CPU với những khối chức năng cơ bản nhất trong hệ thống: bộ nhớ, thiết bị vào/ra, đơn vị điều khiển truy cập trực tiếp bộ nhớ, đơn vị điều khiển ngắt, đơn vị tạo xung nhịp, đơn vị diều khiển, v.v…

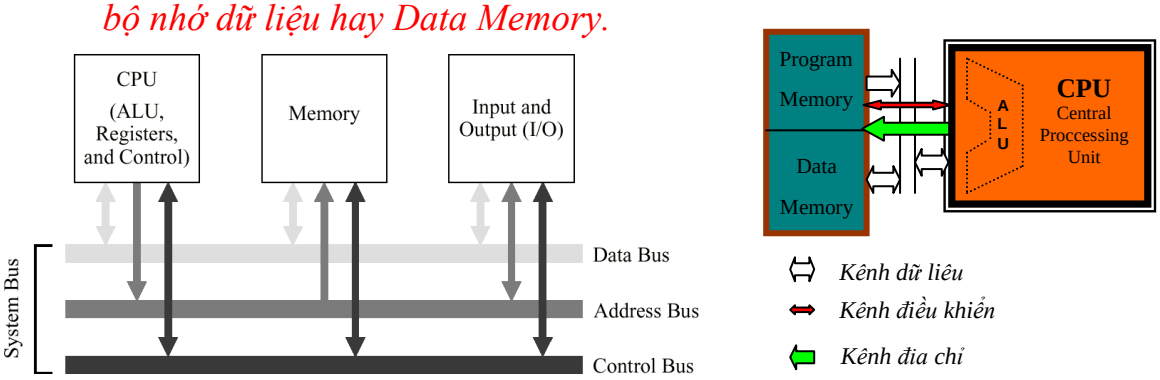
2.1. Liên kết các khối khối chức năng

2.1.1. Bộ xử lý trung tâm (CPU) và bộ nhớ

Nói đến máy tính tức là bàn luận *về sự phối hợp giữa thực hiện xử lý (processing) dữ liệu và đưa ra kết luận (making decisions).* Việc xử lý và đưa ra kết luận được thực hiện bởi *Đơn vị xử lý trung tâm* hay còn gọi là CPU của máy tính. Vậy thì "bộ não" của máy tính chính là CPU. CPU không phải là một bộ phận chức năng biết suy nghĩ và biết thực hiện công việc, song nó có khả năng thực hiện những ý đồ và công việc của người sử

dụng thông qua các lệnh. Những ý tưởng của người sử dụng được thể hiện qua chương trình. *Chương trình là tập hợp các lệnh được chọn lọc và sắp xếp theo một tuần tự chặt chẽ thông qua nguyên tắc xử lý, giải quyết một vấn đề cụ thể (hay còn gọi là thuật giải)*. Chương trình và dữ liệu tương ứng được lưu giữ trong bộ nhớ của máy tính. Bộ nhớ được chia ra thành 2 phần:

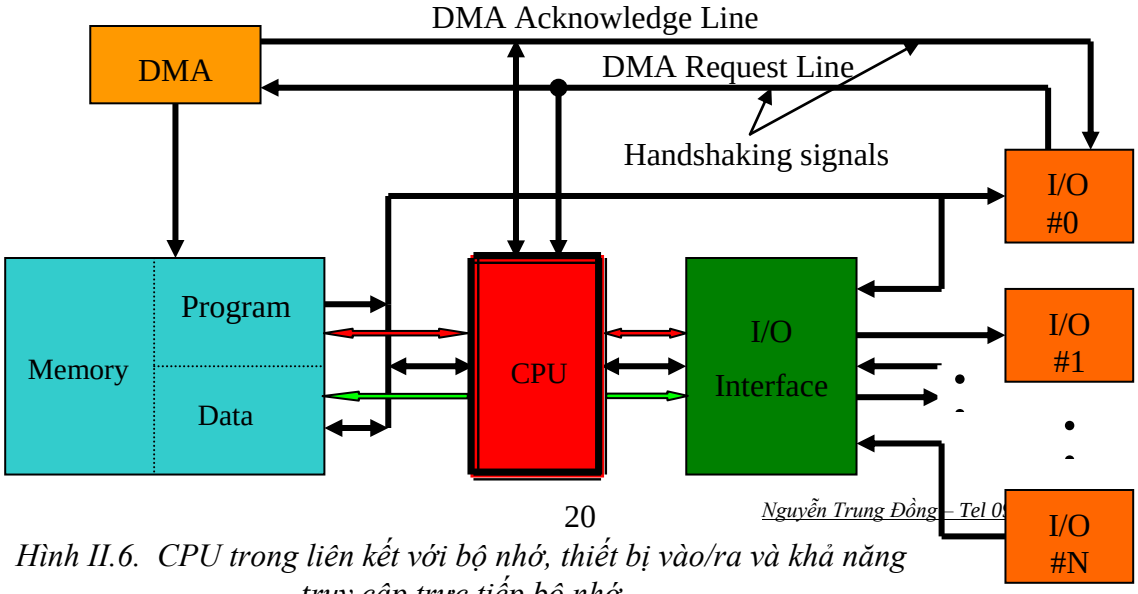
- *Phần lưu giữ chương trình được gọi là bộ nhớ chương trình hay Program Memory.*
- *Phần lưu giữ dữ liệu (dữ liệu để xử lý và dữ liệu kết quả) được gọi là bộ nhớ dữ liệu hay Data Memory.*



Như vậy, CPU cần có bộ nhớ để lưu giữ chương trình và dữ liệu. Theo hình vẽ, thấy rằng CPU chỉ đọc chương trình, song với dữ liệu, nó phải đọc ra để xử lý và phải ghi lại kết quả, tương ứng với các mũi tên một chiều và mũi tên hai chiều trên hình.

2.1.2. CPU, bộ nhớ và thiết bị vào/ra

CPU liên lạc với các thiết bị bên ngoài (hay còn gọi là *thiết bị ngoại vi – peripherals*) để đọc dữ liệu, lệnh và đưa ra kết quả đã được xử lý, ví dụ như bàn phím, máy in, màn hình... Chức năng của các thiết bị này là giao diện giữa người sử dụng và máy tính. Các thiết bị này được gọi chung là *thiết bị vào/ra, hay thiết bị nhập/xuất,* (*I/O device*). Một máy tính có thể có rất nhiều thiết bị vào/ra.



Hình II.6. CPU trong liên kết với bộ nhớ, thiết bị vào/ra và khả năng truy cập trực tiếp bộ nhớ

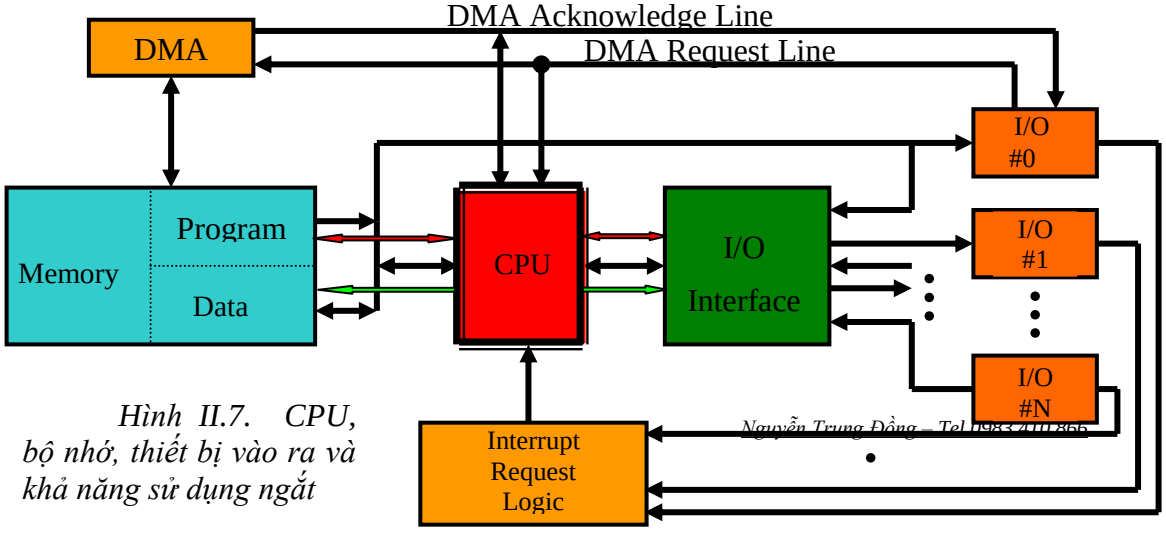
Thông thường, các thiết bị vào/ra không tương thích được với CPU về mặt mức tín hiệu (điện áp thể hiện mức logic "0" hoặc "1") và tốc độ v.v…, do đó ta cần phải bổ sung vào giữa chúng các khối phối ghép (hay còn gọi là giao diện – *I/O interface*).

2.1.3. CPU, bộ nhớ, thiết bị vào ra và khả năng truy cập trực tiếp bộ nhớ

Rõ ràng, việc trao đổi dữ liệu giữa bộ nhớ và thiết bị ngoại vi đều phải thông qua CPU. Mặc dù có lúc nào đó CPU không có yêu cầu dữ liệu, nhưng nó điều khiển quá trình trao đổi dữ liệu của mọi thành phần trong máy tính. Điều đó làm cho CPU tham gia vào mọi hoạt động và tốc độ xử lý của CPU chậm đi rất nhiều.

Để giải quyết vấn đề này, kiến trúc máy tính đưa ra giải pháp thiết bị vào/ra được phép truy cập trực tiếp bộ nhớ (DMA-Direct Memory Access). Để thay thế CPU trong việc truy cập trực tiếp vào bộ nhớ, thiết bị vào/ra được ghép thêm đơn vị điều khiển truy cập trực tiếp bộ nhớ DMAC (DMA Controler). Cơ chế này thực sự mang lại hiệu quả lớn trong các hệ thống máy tính thu thập và xử lý những khối dữ liệu phức tạp và được thực hiện như sau:

- Khi thiết bị vào ra yêu cầu truy cập vào bộ nhớ, thay vì CPU tham gia vào toàn bộ quá trình này, thiết bị vào/ra đưa ra yêu cầu thực hiện truy cập trực tiếp bộ nhớ tới CPU thông qua DMA Request Line.
- CPU nhận yêu cầu, thực hiện việc trao quyền sử dụng đường truyền dẫn dữ liệu cho thiết bị vào/ra (tức là treo dường truyền dẫn dữ liệu giữa bộ nhớ, thiết bị vào/ra và CPU), sau đó gửi thông báo nhận biết và đồng ý cho thiết bị vào/ra qua DMA Acknowledge Line). Những tín hiệu trao đổi này được gọi là tín hiệu bắt tay (Handshaking)
- Thiết bị vào/ra thực hiện việc Ghi hoặc Đọc bộ nhớ trực tiếp qua đường truyền dẫn dữ liệu không thông qua CPU. Như vậy CPU có thể tiếp tục thực hiện các thao tác xử lý khác.



Hình II.7. CPU, bộ nhớ, thiết bị vào ra và khả năng sử dụng ngắt

2.1.4. CPU, bộ nhớ, thiết bị vào ra và khả năng sử dụng ngắt

Khối chức năng tiếp theo thực hiện việc đáp ứng yêu cầu phục vụ của CPU đối với các thiết bị vào/ra là thiết bị điều khiển ngắt (*Interrupt Controller*). Để hiểu ngắt cần thiết như thế nào, ta xét những khả năng sau:

- Thiết bị vào/ra chỉ cần đến CPU khi có sự trao đổi dữ liệu giữa CPU với thiết bị vào/ra.
- Một số thiết bị vào/ra hoạt động rất chậm so với khả năng xử lý của CPU, do vậy, việc CPU phải chờ đợi để trao đổi dữ liệu làm mất rất nhiều thời gian.

Dựa trên thực tế này, kiến trúc máy tính đề nghị một giải pháp hữu hiệu nhằm làm tăng hiệu suất hoạt động xử lý dữ liệu của CPU cũng như của máy tính nói chung. Giải pháp dựa trên quy trình sau:

- Thiết bị ngoại vi muốn làm việc với CPU phải gửi yêu cầu ngắt đến CPU thông qua tín hiệu yêu cầu ngắt (*Interrupt Request Signal*).
- CPU tạm dừng tiến trình đang thực hiện, gửi tín hiệu chấp nhận phục vụ ngắt cho thiết bị vào/ra.
- CPU tiến hành phục vụ thiết bị vào/ra thực chất là thực hiện việc trao đổi dữ liệu, khi thực hiện xong thì quay về tiếp tục công việc đang bỏ dở.

2.1.5. Khối xung nhịp (Clock) và khối điều khiển (Control)

Đã có thể nhìn thấy rằng, chỉ cần thêm vào 2 khối cơ bản nữa là ta có một cấu trúc máy tính hoàn chỉnh: Khối xung nhịp *(Clock)* và khối điều khiển *(Control)*. Có thể nói khối điều khiển là sợi chỉ xuyên suốt, chỉ đạo mọi hoạt động của tất cả các khối chức năng nhằm đảm bảo hoạt động ổn định cho một máy tính, không bao giờ xảy ra bất cứ sự tranh chấp nào. Khối tạo nhịp Clock thực hiện việc định thời cho mọi hoạt động trong máy tính được đồng bộ hoá.

2.2. Kiến trúc máy tính nhìn từ góc độ cấu trúc cơ bản

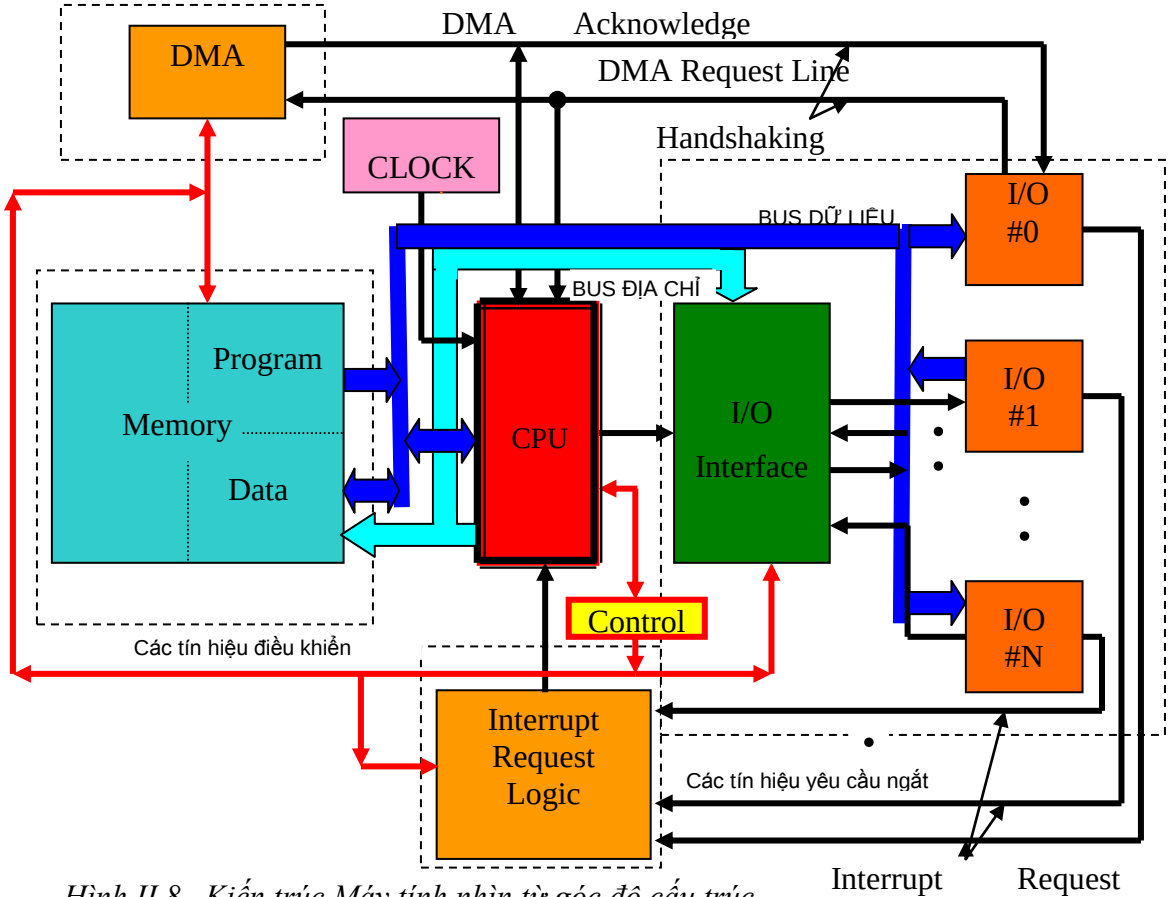
Một hệ thống máy tính phải có các tính năng cơ bản sau:

1. Khả năng thực hiện *Vào/Ra* - là khâu nối hay giao diện giữa người sử dụng và máy tính

2. Khả năng *Ngắt* - cho phép máy tính hoạt động hiệu suất hơn và mềm dẻo hơn.
3. Khả năng *Truy Cập Trực tiếp Bộ Nhớ* - cho phép các thiết bị vào/ra làm việc với bộ nhớ mà không ảnh hưởng đến tiến trình thực hiện chương trình của CPU

Các khối chức năng chủ yếu gồm:

1. *CPU* - đó là bộ não của máy tính
2. *Bộ nhớ* - nơi lưu giữ mọi dữ liệu của máy tính
3. *Khối điều khiển* - Điều khiển sự lưu thông của các dòng dữ liệu trong tiến trình, đảm bảo không xảy ra tranh chấp giữa các khối chức năng
4. *Logic yêu cầu ngắt* thiết lập quan hệ phục vụ hợp lý của CPU với các thiết bị vào/ra
5. *Khối phối ghép vào/ra* thực hiện việc đồng nhất hoá các loại tín hiệu, định thời ... giữa CPU và các thiết bị vào/ra
6. *Tạo nhịp Clock* thực hiện việc định thời cho toàn bộ hệ thống.



Hình II.8. Kiến trúc Máy tính nhìn từ góc độ cấu trúc

Nguyễn Trung Đồng – Tel 0983 410 866

Thuật ngữ *Kiến trúc máy tính* được dùng để mô tả sơ đồ tổ chức nguyên lý. Thuật ngữ được dùng nhằm minh hoạ những khối chức năng cần thiết và các kênh truyền dẫn liên kết chúng để cấu thành một máy tính.

Trong các chương tiếp theo, chúng ta sẽ tìm hiểu phần kiến trúc các BUS hệ thống, kiến trúc CPU, kiến trúc khối điều khiển. ***"Kiến trúc"*** nôm na có nghĩa là các sơ đồ ***các tổ hợp liên kết các khối-chức năng và cấu hình hệ thống.***

3. Biểu diễn thông tin trong máy tính

Các máy tính xử lý các thông tin số và chữ. Các thông tin được biểu diễn dưới dạng mã nhất định. Bản chất vật lý của việc biểu diễn thông tin là điện áp ("0" ứng với không có điện áp, "1" ứng với điện áp ở mức quy chuẩn trong mạch điện tử) và việc mã hoá các thông tin số và chữ được tuân theo chuẩn quốc tế. *Mã hiệu để mã hoá các thông tin cho máy tính xử lý là các giá trị của biến nhị phân "0" hoặc "1", tương ứng với biến logic "False" hoặc "True"*. Một biến chỉ nhận một trong hai giá trị duy nhất là "0" hoặc "1" được gọi là một bit (Binary Digit). Hai trạng thái này của bit, thực chất là các giá trị tương ứng với "False" hoặc "True", hay trạng thái "tắt" hoặc "đóng" của một công tắc điện, được sử dụng để mã hoá cho tất cả các ký tự (gồm số, chữ và các ký tự đặc biệt khác). Các bit được ghép lại thành các đơn vị mang thông tin đầy đủ - các mã tự - cho các ký tự biểu diễn các số, các ký tự chữ và các ký tự đặc biệt khác.

Bit (BInary digiT) là đơn vị cơ bản mang thông tin trong hệ đếm nhị phân. Các mạch điện tử trong máy tính phát hiện sự khác nhau giữa hai trạng thái (điện áp mức "1" và điện áp mức "0") và biểu diễn hai trạng thái đó dưới dạng một trong hai số nhị phân "1" hoặc "0".

Nhóm 8 bit ghép kề liền nhau, tạo thành đơn vị dữ liệu cơ sở của máy tính được gọi là 1 Byte. Do được lưu giữ tương đương với một ký tự (số, chữ hoặc ký tự đặc biệt) nên Byte cũng là đơn vị cơ sở để đo các khả năng lưu giữ, xử lý của các máy tính. Các thuật ngữ như KiloByte, MegaByte hay GigaByte thường được dùng làm bội số trong việc đếm Byte, dĩ nhiên theo hệ đếm nhị phân, nghĩa là:

1KiloByte = 1024 Bytes,
1MegaByte = 1024 KiloBytes,
1GigaByte = 1024 MegaBytes.

Các đơn vị này được viết tắt tương ứng là KB, MB và GB.

3.1. Mã hoá các thông tin không số

Có hai loại mã phổ cập nhất được sử dụng là mã ASCII và EBCDIC.

- Mã ASCII (American Standard Code for Information Interchange) dùng 7 bits để mã hoá các ký tự
- Mã EBCDIC (Extended Binary Coded Decimal Interchange Code) dùng cả 8 bits (1 Byte) để mã hoá thông tin
- Loại mã trước đây được dùng nhiều trong ngành bưu điện, trong các máy teletype là mã BAUDOT, chỉ sử dụng 5 bits để mã hoá thông tin.
- Tồn tại nhiều loại mã khác nhau, nhưng hầu như không được sử dụng trong các máy tính thông dụng…

3.2. Hệ đếm thập phân

Từ xa xưa con người đã sử dụng công cụ tự nhiên và sẵn có nhất của mình để đếm các vật, đó là các ngón tay trên hai bàn tay của mình. Vì hai bàn tay chỉ có 10 ngón nên *hệ đếm thập phân* mà chúng ta sử dụng ngày nay là kết quả tự nhiên của cách đếm đó.

Hệ đếm thập phân sử dụng 10 kí hiệu khác nhau để biểu diễn các số: 0, 1, 2, ..., 9. Để biểu diễn các số lớn hơn 10, hệ đếm thập phân sử dụng *kí pháp vị trí*. Theo kí pháp này giá trị mà kí hiệu biểu diễn phụ thuộc vào vị trí của nó trong dãy kí hiệu. Ví dụ 2 trong số 29 biểu diễn số 20, nhưng trong 92 biểu diễn số 2. Cũng cần nhắc lại rằng: tất cả các hệ đếm đều tuân thủ ký pháp vị trí.

Với các số 0, 1, 2, ..., 9 và kí pháp vị trí chúng ta có thể biểu diễn mọi số tự nhiên lớn tùy ý. Quy tắc biểu diễn tổng quát một số tự nhiên trong hệ thập phân là

$$a_n a_{n-1}...a_1 a_0 = a_n \times 10^n + a_{n-1} \times 10^{n-1} + ... + a_1 \times 10 + a_0$$

Ví dụ

$$2158 = 2 \times 10^3 + 1 \times 10^2 + 5 \times 10 + 8$$

Với các số thập phân chúng ta sử dụng lũy thừa âm của 10 với quy tắc tương tự:

$$0.a_1 a_2...a_{n-1} a_n = a_1 \times 10^{-1} + a_2 \times 10^{-2} + ... + a_{n-1} \times 10^{n-1} + a_1 \times 10^{-n}$$

$$0.2158 = 2 \times 10^{-1} + 1 \times 10^{-2} + 5 \times 10^{-3} + 8 \times 10^{-4}$$

$$126.5 = 1 \times 10^2 + 2 \times 10 + 6 + 5 \times 10^{-1}$$

Ở đây 10 được gọi là cơ số của hệ đếm và các số 10^{-n}, ..., 10^{-1},10^{1}, 10^{2}, ..., 10^{n} được gọi là các trọng số.

Lưu ý rằng thay cho 10 chúng ta có thể sử dụng số tự nhiên *a* bất kỳ và kí pháp vị trí để biểu diễn mọi số cũng với quy tắc trên. Khi đó hệ đếm được gọi là hệ đếm cơ số *a*. Ví dụ như hệ đếm cơ số 12 dùng để biểu diễn thời gian.

3.3. Hệ đếm nhị phân

Máy tính số được xây dựng từ các mạch điện tử. Các mạch điện tử trong máy tính phân biệt được sự khác nhau giữa hai trạng thái (có dòng điện hay không, điệp áp cao hay thấp, v.v.) và biểu diễn các trạng thái đó dưới dạng một trong hai số 1 hoặc 0. Vì vậy các số 0 và 1 rất thích hợp và đủ để biểu diễn các số tùy ý trong máy tính. Việc chế tạo một mạch điện tin cậy có thể phân biệt được sự khác nhau giữa 1 và 0 tương đối dễ và rẻ, hơn nữa máy tính có khả năng xử lý nội bộ các số 0 và 1 rất chính xác.

Các số 0 và 1 được gọi là các số nhị phân. Hệ đếm nhị phân là hệ đếm chỉ dùng các số 0 và 1 và kí pháp vị trí để biểu diễn các số. Trong hệ đếm này 2 là cơ số, các số ..., 2^{-2}, 2^{-1}, 2^{0}, 2^{1}, 2^{2}, v.v. là trọng số. Ví dụ số thập phân 126.5 được biểu diễn trong hệ nhị phân với dạng:

$$26.5_{10} = 11010.1_{2} = 1\times 2^{4} + 1\times 2^{3} + 0\times 2^{2} + 1\times 2^{1} + 0\times 2^{0} + 1\times 2^{-1}$$

(chỉ số dưới biểu diễn cơ số của hệ đếm).

4. Chuyển đổi giữa các hệ đếm

4.1. Chuyển đổi hệ thập phân sang hệ nhị phân

Để chuyển đổi một số ở hệ thập phân sang hệ nhị phân cần thực hiện các bước sau:

1. Tách phần nguyên và phần thập phân;
2. Chuyển đổi phần nguyên và phần thập phân sang hệ nhị phân;
3. Ghép lại thành kết quả.

4.1.1 Chuyển đổi phần nguyên

Lấy phần nguyên chia cho 2, được thương số và phần dư. Phần dư dùng làm kết quả chuyển đổi, phần thương số được tiếp tục chia cho 2. Thực

hiện quá trình này cho đến khi thương số bằng 0 thì dừng. Các phần dư được viết ghép lại từ dưới lên theo chiều mũi tên.

Ví dụ: Đổi số 173_D ra số nhị phân

173	2	dư 1	k_0
86	2	dư 0	k_1
43	2	dư 1	k_2
21	2	dư 1	k_3
10	2	dư 0	k_4
5	2	dư 1	k_5
2	2	dư 0	k_6
1	2	dư 1	k_7
0			

Vậy $173_D = 10101101_B$

4.1.2. Chuyển đổi phần thập phân

Lấy phần thập phân nhân với 2. Nếu tích lớn hơn 1 thì lấy 1 làm kết quả và chỉ giữ lại phần thập phân; nếu tích nhỏ hơn 1 thì lấy 0 làm kết quả. Thực hiện quá trình này cho đến khi tích bằng 1 thì dừng. Các kết quả trong từng bước được viết ghép vào bên phải.

Ví dụ: Chuyển đổi 19.625_{10} sang hệ nhị phân:

Bước 1: Tách 19.625_{10} thành phần nguyên và phần thập phân ta được 19 và 0.625

Bước 2: Chuyển đổi phần nguyên 19 được 10011_2 và phần thập phân 0.625 được 0.101_2.

Bước 3: Ghép lại ta được kết quả $19.625_{10} = 10011.101_2$.

Ví dụ: Chuyển đổi số 0.8128 thành số nhị phân.Thực hiện phép nhân liên tiếp với 2, phần nguyên của tích bao giờ cũng là các giá trị hoặc bằng "0" hoặc bằng "1", thu được kết quả sau:

0.8128	x 2	= 1.6256	= 1 +	0.6256
0.6256	x 2	= 1.2512	= 1 +	0.2512
0.2512	x 2	= 0.5024	= 0 +	0.5024
0.5024	x 2	= 1.0048	= 1 +	0.0048
0.0196	x 2	= 0.0392	= 0 +	0.0392
0.0392	x 2	= 0.0784 Quá nhỏ có thể bỏ qua		

Như vậy, $0.8128_D \cong 0.11010_B$

Nguyễn Trung Đồng – Tel 0983 410 866

4.2 Chuyển đổi hệ nhị phân sang các hệ Hexa, Octal

Dữ liệu số trong máy tính được biểu diễn theo hệ đếm nhị phân. Từ nhớ cơ bản của các loại máy tính đều tuân thủ chuẩn của IBM, nghĩa là theo từng Byte. Mỗi Byte gồm 8 bit nhị phân. Tuy nhiên, cách viết một số liệu nhị phân không thích hợp với chách nhìn và nhận biết độ lớn, nhất là khi dữ liệu là một số có độ dài nhiều Byte. Do vậy, ta hay sử dụng các phương pháp biểu diễn theo hệ đếm Hexa (hệ đếm cơ số 16). Một thuận lợi cơ bản ở đây là một ký tự Hexa có thể đại diện cho một dữ liệu nhị phân 4 bit. Như vậy mỗi byte có thể biểu diễn một số nhị phân hoặc có thể tách ra hai nhóm, mỗi nhóm 4 bit và dữ liệu lưu trong mỗi byte có thể biểu diễn dưới dạng hai ký tự số hexa. Vì trong hệ hexa cần 16 kí hiệu khác nhau, ngoài các số 0, 1, ..., 9, người ta bổ sung thêm các chữ A, B, C, D, E và F với các trọng số tương ứng trong hệ đếm thập phân là:

AH = 10D, BH = 11D, CH = 12D, DH = 13D, EH = 14D, FH = 15D.

Ví dụ: số BBH = 187D, C7H = 199D.

Sự tiện dụng của hệ đếm hexa là ở chỗ ta dễ dàng chuyển đổi số nhị phân thành hexa và ngược lại theo quy tắc sau:

Giả sử cho số nhị phân 10001011_B. Để đổi thành số hexa ta chia dãy này thành từng nhóm 4 số: 1000 và 1011. Thay các nhóm 4 số bằng số hexa tương ứng cho trong bảng sau, ta được $8B_H$. Ngược lại, nếu có số $C7_H$, ta chỉ đơn giản ghép các nhóm số nhị phân tương ứng 1100 và 0111 sẽ được biểu diễn nhị phân 11000111_B.

Nhị phân	Hexa	Thập phân	Nhị phân	Hexa	Thập phân
0000	0	0	1000	8	8
0001	1	1	1001	9	9
0010	2	2	1010	A	10
0011	3	3	1011	B	11
0100	4	4	1100	C	12
0101	5	5	1101	D	13
0110	6	6	1110	E	14
0111	7	7	1111	F	15

5. Các phép tính với số nhị phân

Trong hệ đếm thập phân chúng ta đã quen thuộc với bốn phép tính cơ bản: cộng, trừ nhân và chia. Trong hệ nhị phân, chúng ta cũng dễ dàng thực hiện các phép tính này.

5.1. Phép cộng

Phép cộng được thực hiện với quy tắc:

0 + 0 =0

1 + 0 =1

0 + 1 = 1

1 + 1 = 0 và nhớ 1

Giá trị 1 + 1 = 2, nhưng vì trong hệ nhị phân chỉ có hai số 0 và 1, mặt khác biểu diễn của 2 trong hệ nhị phân là $2 = 1\times2^2 + 0 = 10_2$ nên kết quả chính là 0 và nhớ 1.

Ví dụ:

1 1 0

\+ 0 1 1

= 1 0 0 1

5.2. Phép trừ

Phép trừ là phép toán ngược với phép cộng và được thực hiện với quy tắc:

0 - 0 =0

1 - 0 =1

1 - 1 = 0

0 - 1 = 1 và mượn 1

Trong quy tắc cuối cùng 0 - 1 = 1, vì số bị trừ nhỏ hơn số trừ nên phải mượn 1 từ số ở vị trí cao hơn (bên trái)

Ví dụ:

1 1 0

\- 0 1 1

= 0 1 1

5.3. Phép nhân

Cũng giống như trong hệ thập phân, phép nhân được thực hiện với quy tắc:

$0 \times 0 = 0$ $1 \times 0 = 0$

$0 \times 1 = 0$ $1 \times 1 = 1$

Ví dụ:

1 1 0 0

× 1 0 1

= 1 1 0 0

\+ 0 0 0 0

1 1 0 0

= 1 1 1 1 0 0

5.4. Phép chia

Phép chia là phép ngược của phép nhân. Phép chia cho 0 không có nghĩa và:

$0 : 1 = 0$

$1 : 1 = 1$

Ví dụ:

1 1 0 0 1 : 1 0 1 = 1 0 1

6. *Biểu diễn dữ liệu số trong máy tính*

6.1. Biểu diễn số và số âm

Trong máy tính các số được biểu diễn bằng dãy các số nhị phân, ví dụ $19_{10} = 10011_2$. Mỗi số nhị phân trong dãy biểu diễn này được gọi là *bit*. Như vậy bit chỉ có thể nhận một trong hai giá trị "0" hoặc "1". Bit là đơn vị cơ bản nhất, nhỏ nhất để biểu diễn dữ liệu trong máy tính.

Với 7 bit ta chỉ biểu diễn được các số từ $0 = 0000000_2$ đến 1111111_2 ($= 127_{10}$). Để *biểu diễn được các giá trị âm* từ -127 đến 0, tức từ -1111111_2 đến 0000000_2, ta cần sử dụng thêm 1 bit để đánh dấu. Đây là bit thứ 8, gọi là *bit dấu* và nằm ở vị trí cao nhất (bên trái) trong số nhị phân. Theo quy ước chung số dương có bit dấu là 0, số âm có bit dấu là 1.

Bit dấu	2^6	2^5	2^4	2^3	2^2	2^1	2^0
S	D_6	D_5	D_4	D_3	D_2	D_1	D_0

Trên thực tế người ta còn sử dụng *số bù* để biểu diễn số âm. Kỹ thuật này cho phép máy tính có thể thực hiện phép trừ như là phép cộng.

Trước hết ta minh họa phương pháp biểu diễn này trong hệ thập phân. Trong hệ thập phân số bù chính là số bù 10. Số bù 10 của một số được tạo bằng cách lấy 9 trừ đi mỗi chữ số và sau đó cộng 1 vào chữ số có ý nghĩa thấp nhất. Ví dụ số bù 10 của 88 là 12, của 23 là 77.

Việc thực hiện phép trừ một số dương (số bị trừ) đi một số dương (số trừ) có thể được thực hiện thông qua phép cộng như sau:

(1) Tạo số bù 10 của số trừ

(2) Cộng số bị trừ với số bù 10 của số trừ

(3) Nếu có nhớ ở chữ số cao nhất thì chữ số đó bị bỏ đi và kết quả là số dương. Nếu không kết quả là âm và kết quả thật là số bù 10 của số nhận được và thêm dấu – vào đằng trước.

Ví dụ 1. 97 – 88 = 09

(1) Tạo số bù 10 của 88 là 12

(2) Cộng 97 với 12 được 109

(3) Có nhớ ở chữ số cao nhất: Kết quả là số dương và bằng 09.

Ví dụ 2. 53 – 67 = -14

(1) Tạo số bù 10 của 67 là 33

(2) Cộng 53 với 33 được 86

(3) Không nhớ ở số cao nhất, kết quả là âm.

(4) Lấy số bù 10 của 86 được 14. Kết quả thật là – 14.

Trong hệ nhị phân, số bù được hiểu là số bù 2. Số bù 2 là số được tạo ra bằng cách lấy 1 trừ đi mỗi chữ số và cộng 1 vào số có giá trị thấp nhất, hoặc tương đương, đổi số 0 thành số 1, số 1 thành số 0 và cộng 1 vào số có giá trị thấp nhất.

Ví dụ số bù 2 của 11000 là (00111 + 1) = 01000.

Trong hệ nhị phân, sử dụng số bù 2 phép trừ cũng được thực hiện thông qua phép cộng.

Ví dụ 1. Cả hai số là số dương: cộng từng số từ phải qua trái, kể cả bit dấu.

Ký pháp bình thường 1000 + 0101 = 1101,

Dạng dữ liệu trong máy tính 01000 + 00101 = 01101

Ví dụ 2. Một số dương và một số âm có trị tuyệt đối nhỏ hơn: Tạo bù 2 của của số âm, thêm bit dấu vào vị trí cao nhất. Bỏ bit nhớ trong kết quả và kết quả là số dương.

Ký pháp bình thường 1000 + (-0101) = 0011,

Dạng dữ liệu trong máy tính 01000 + 11011 =1 00011. Kết quả 00011

Ví dụ 3. Một số dương và một số âm có trị tuyệt đối lớn hơn: Thực hiện phép cộng, bit dấu không có nhớ. Kết quả là số âm dạng bù 2. Cần bỏ bit dấu để tạo số bù 2 để có kết quả thật

Ký pháp bình thường 0101 + (-1000) = -0011,

Dạng dữ liệu trong máy tính 00101 + 11000 = 11101. Kết quả -0011

Ví dụ 4. Hai số âm: Khi thực hiện phép công, có nhớ ở bit dấu. Cần bỏ bit nhớ và kết quả là số âm dạng bù 2.

Ký pháp bình thường -0101 + (-1000) = -1101,

Dạng dữ liệu trong máy tính 11011 + 11000 = 11000. Kết quả -1101

Trong máy tính hai số nhị phân 0 và 1 được biểu diễn và lưu giữ bằng các *thiết bị hai trạng thái* (có dòng điện/không có dòng điện, điện áp cao/điện áp thấp, v.v.). Một số, sau khi biểu diễn dưới dạng nhị phân, được biểu diễn và lưu trong máy tính bằng dãy các thiết bị hai trạng thái (được liên kết với nhau theo cách nào đấy). Dãy các thiết bị hai trạng thái đó được gọi là *thanh ghi*. Các thanh ghi được gán các tên khác nhau để phân biệt, ví dụ thanh ghi A, thanh ghi B.

6.2. Biểu diễn số dấu phẩy động (Floatting Point Number)

Khi biểu diễn các số rất lớn hoặc rất nhỏ người ta thường dùng ký pháp dấu chấm động. Ví dụ $190000=0.49*10^{6}$, $0.00023=0.23*10^{-3}$. Dạng biểu diễn tổng quát:

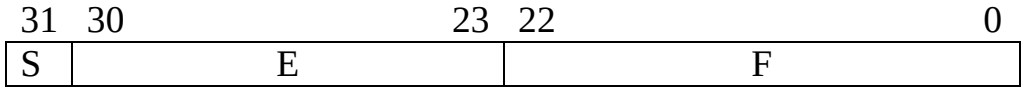
$$\pm \text{Định trị} * \text{Cơ số}^{\pm \text{Số mũ}}$$

Chẳng hạn, trong hai số trên định trị là 0.19 và 0.23, cơ số = 10, số mũ là 6 và –3.

Trong máy tính số biểu diễn theo dấu chấm động có hai dạng:

6.2.1. Dạng đơn giản

Số dấu chấm động được lưu dưới dạng 32 bit:

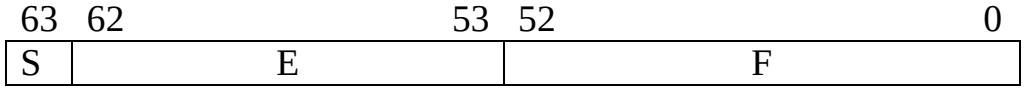


$$N = (-1)^S * 2^{E-127} * 1.F$$

Bit đầu tiên là bit dấu, 8 bit tiếp theo là số nguyên nhị phân biểu diễn số mũ, 23 bit cuối cùng là số nhị phân biểu diễn phần sau dấu chấm nhị phân. Cơ số ở đây là 2 và phần định trị có dạng 1.F

6.2.2. Dạng chính xác gấp đôi

Số dấu chấm động được lưu dưới dạng 64 bit:



$$N = (-1)^S * 2^{E-1023} * 1.F$$

Chương III. Kiến trúc Trung tâm xử lý (CPU)

1. Kiến trúc CPU

Để có hiểu biết về kiến trúc một CPU, cần thiết phải định nghĩa các chức năng cơ bản của nó. Càng nhiều chức năng thì CPU càng phức tạp, càng có giá thành cao. Đây chính là lý do CPU được phân loại thành CPU chuyên dụng. Chúng ta sẽ bắt đầu từ những yêu cầu cơ bản nhất đối với CPU:

- *Có kiến trúc hợp lý nhất để thoả mãn những yêu cầu tối thiểu trong chức năng xử lý dữ liệu*
- *Có đầy đủ các tính năng để xây dựng một máy tính đáp ứng yêu cầu sử dụng*
- *Có khả năng ứng dụng được trong thực tế.*

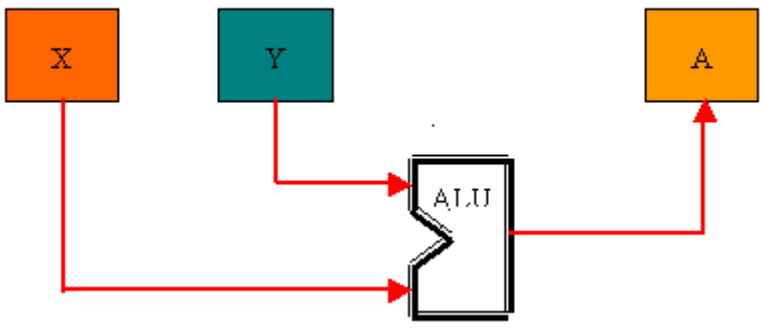
1.1. Chức năng và kiến trúc của CPU

Trước hết, CPU phải thực hiện được các phép tính số học và phép tính lôgic cơ bản: *cộng (addition), trừ (subtraction), VÀ logic (AND), đảo giá trị (NOT) và HOẶC logic (OR) và các lệnh vào/ra dữ liệu (INP, OUT)*. Kiến trúc CPU phải đáp ứng yêu cầu tối thiểu này, phải có các khối chức năng hiện thực hoá các phép tính trên.

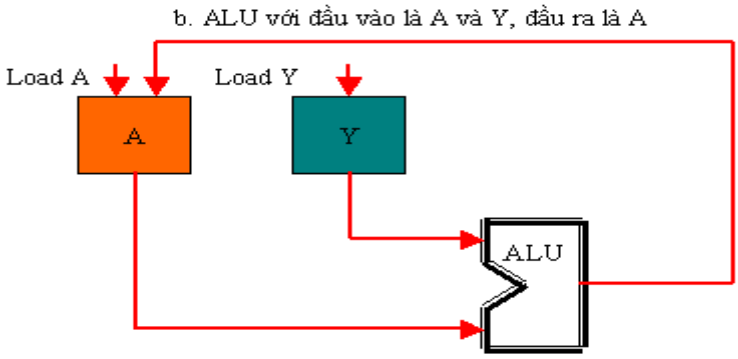
Khối chức năng đầu tiên là **ALU** *(Arithmetic – Logic Unit*) thực hiện các phép tính ADD, SUB, AND, NOT và OR).

Như vậy, kiến trúc đòi hỏi CPU, ngoài ALU, phải có các thanh ghi toán hạng X, Y và thanh ghi kết quả tính toán A.

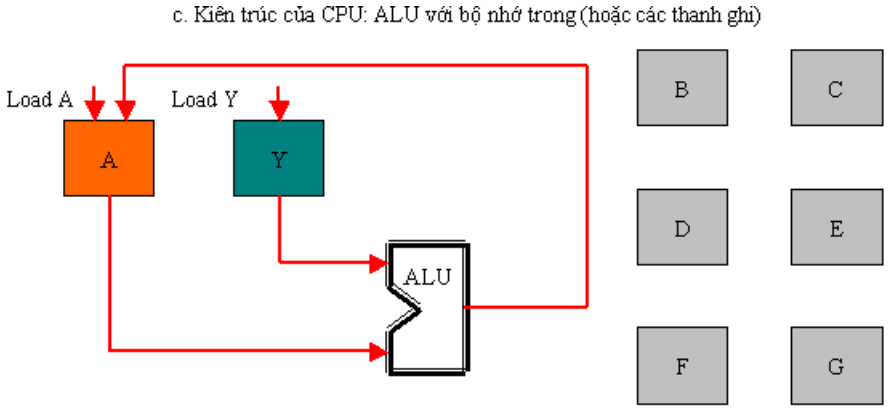
a. ALU với đầu vào là X và Y, đầu ra là A



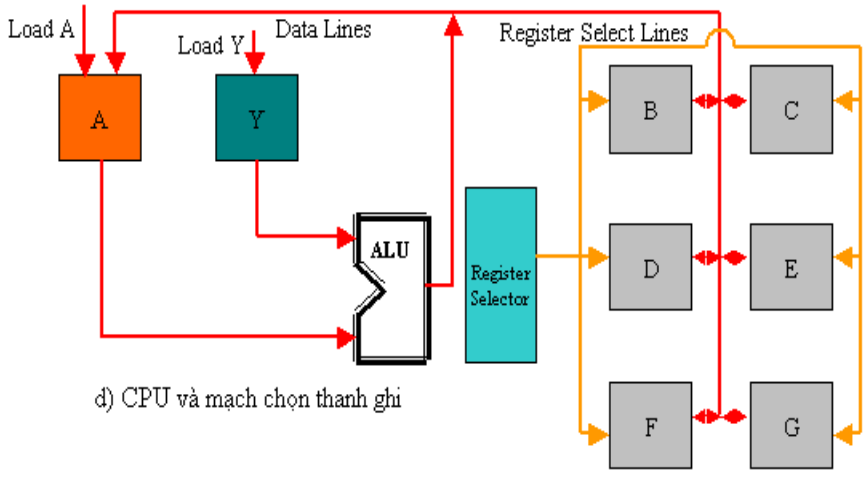
Nếu cho rằng để đơn giản hoá cấu trúc CPU, ta sử dụng thanh ghi A làm thanh ghi cho một toán hạng, đồng thời là thanh ghi kết quả xử lý, xây dựng lại đường truyền dữ liệu ta có CPU đơn giản hơn như hình sau:



Sau cùng, cần hiểu rằng khi xử lý dữ liệu, tính toán, có những *kết quả trung gian (hoặc các toán hạng)* cần được lưu giữ vì nhiều lý do khác, CPU cần thêm một vùng nhớ phụ với tốc độ truy nhập thật cao, hoặc còn gọi là *các thanh ghi* (đa dụng), cấu trúc sẽ được mở rộng thêm như sau:

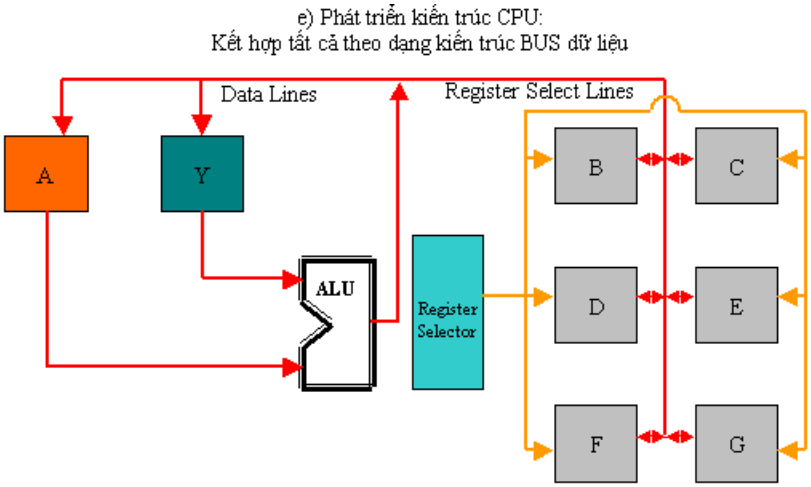


Với các thanh ghi (ô nhớ phụ) B, C, D, E, F, G cần thiết phải có *mạch chọn thanh ghi (Register Selector)* để chọn thanh ghi, tức là chọn đúng ô lưu giữ số liệu cần thiết cho ALU xử lý.

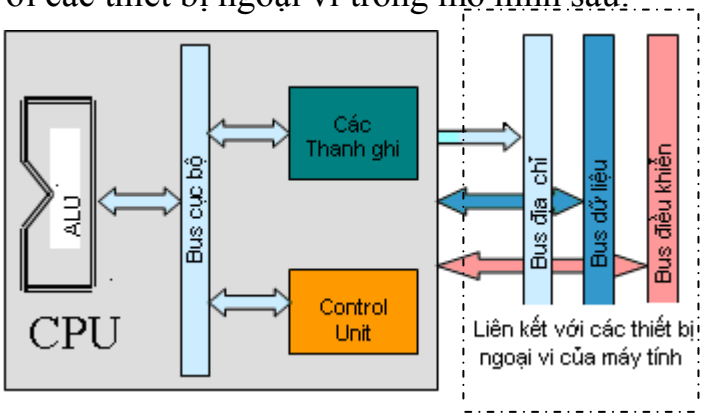


Nguyễn Trung Đồng – Tel 0983 410 866

Và cuối cùng, một thanh ghi bất kỳ đều có thể chứa một trong các toán hạng ALU cần xử lý, hoặc là lưu giữ kết quả xử lý, chúng sẽ được liên kết với thanh ghi A và ALU như sau:



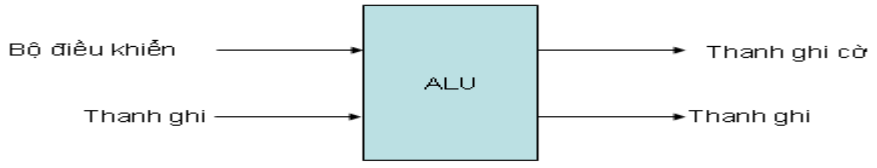
Như vậy có thể tóm lược kiến trúc của CPU và khả năng trao đổi thông tin dữ liệu với các thiết bị ngoại vi trong mô hình sau:



1.2. Kiến trúc ALU

ALU là thành phần của máy tính chịu trách nhiệm thực hiện các phép toán số học và logic trên các dữ liệu được cung cấp. Mọi thành phần khác như bộ điều khiển, các thanh ghi, bộ nhớ, thiết bị vào ra cung cấp dữ liệu để ALU xử lý và nhận dữ liệu đã xử lý từ đó.

ALU thực chất là tổ hợp các vi mạch được thiết kế để lưu trữ dữ liệu và thực hiện các tính toán logic đơn giản. Hình sau đây minh họa sự kết nối của ALU với các thành phần khác của CPU:

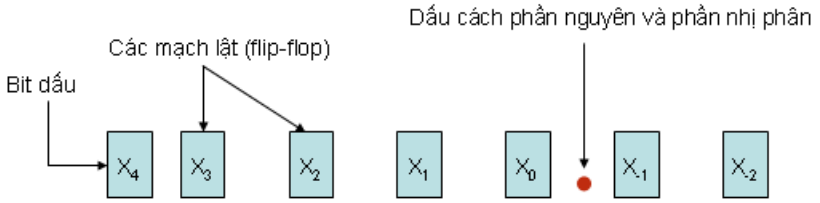


Nguyễn Trung Đồng – Tel 0983 410 866

Các tín hiệu điều khiển từ đơn vị điều khiển xác định tác vụ mà ALU phải thực hiện, dữ liệu cần xử lý có thể được nạp từ nội dung của các thanh ghi. Kết quả xử lý sẽ được lưu vào thanh ghi hoặc bộ nhớ, hoặc đưa ra thiết bị ngoại vi. Kết quả xử lý của ALU cũng tác động lên các cờ trạng thái kết quả phép toán.

Như vậy, dữ liệu được cung cấp cho ALU từ các thanh ghi, và kết quả xử lý cũng được lưu trong các thanh ghi. Các thanh ghi này là các thiết bị lưu trữ tạm thời bên trong CPU và được kết nối với ALU. ALU cũng cho các tín hiệu cờ để ghi lại trạng thái của một hoạt động xử lý, ví dụ như tín hiệu cờ Overflow được đặt bằng 1 khi kết quả tính toán phải biểu diễn dưới dạng có độ dài vượt quá độ dài của thanh ghi được sử dụng để lưu kết quả. Các giá trị cờ cũng được lưu bên trong CPU. Cuối cùng, bộ điều khiển cung cấp các tín hiệu điều khiển các hoạt động xử lý và di chuyển dữ liệu của ALU.

Số được biểu diễn dưới dạng nhị phân và được lưu trong thanh ghi-dãy các mạch lật (FlipFlop). Hình sau đây minh họa một biểu diễn đơn giản:



$$A = \pm \quad X_3.2^3 + X_2.2^2 + X_1.2^1 + X_0 + X_{-1}.2^{-1} + X_{-2}.2^{-2}$$

Thành phần quan trọng nhất của ALU là các bộ cộng số học và bộ cộng logic. Như đã biết, phép trừ có thể thực hiện bằng cách sử dụng phép cộng với phần bù (complement). Như vậy, để hỗ trợ thực hiện các phép tính số học, kiến trúc của ALU cần có vi mạch chuyển một số nhị phân sang phần bù 2 của nó.

Mặt khác, nếu để ý phép nhân hai số nhị phân, ví dụ

$$\begin{array}{rr} & 1\,0\,0\,1 \\ \times & 1\,1\,0\,1 \end{array}$$

với quá trình thực hiện như sau:

```
                1 0 0 1
×               1 1 0 1
-----------------------
                1 0 0 1
              0 0 0 0
+           1 0 0 1 <----dịch trái số bị nhân 2 bit
          1 0 0 1 <------dịch trái số bị nhân 3 bit
-----------------------
          1 1 1 0 1 0 1
```

ta thấy rằng có hai thao tác cần thực hiện:

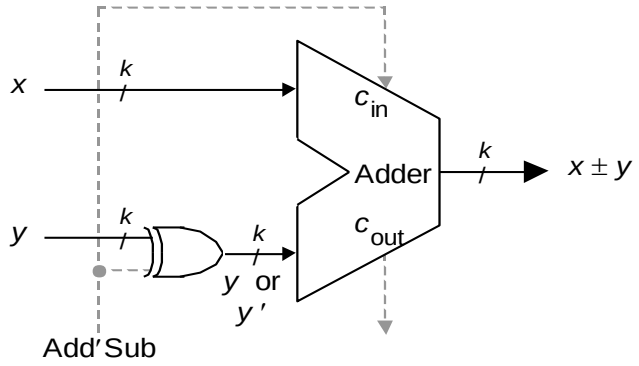
- Sao dữ liệu cần nhân vào thanh ghi, nếu nhân với 1; hoặc nạp 0 nếu nhân với 0;
- Dịch chuyển dữ liệu sang trái một số bit, tuỳ theo vị trí của giá trị "1" trong số nhân
- cộng dồn các dữ liệu trong từng bước để nhận được kết quả cuối cùng. Như vậy, để thực hiện phép nhân, ngoài mạch cộng hai số nhị phân chỉ cần thêm vi mạch dịch chuyển số nhị phân trong thanh ghi sang trái một hoặc nhiều vị trí.

Tóm lại, kiến trúc bên trong của ALU cần có các thành phần chính sau đây:

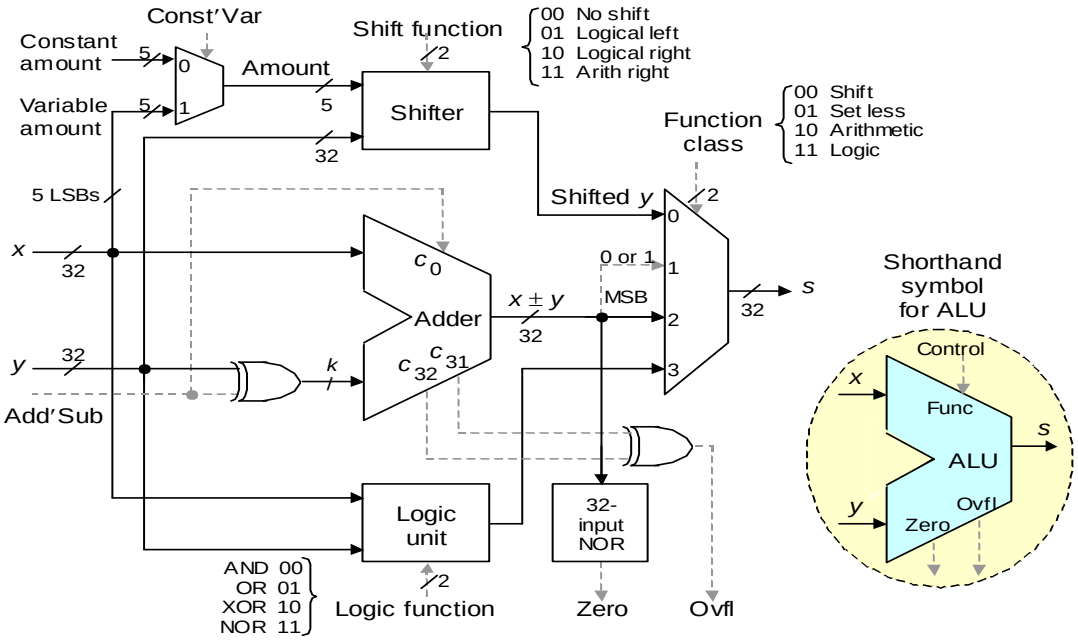
- Các mạch số học và logic: Thực hiện các phép toán số học và logic trên dữ liệu được cung cấp.
- Mạch lấy phần bù (Complemeter): Lấy phần bù 2 của một số được lưu trong một thanh ghi để hỗ trợ cho việc tính toán.
- Mạch dịch (Shifter): Dịch chuyển các bit được lưu trong một thanh ghi sang trái hoặc sang phải một số bit xác định trước để hỗ trợ cho việc tính toán.
- Thanh ghi cờ: (Status Flag): Lưu các trạng thái xử lý dữ liệu (thành công hay không thành công, lý do...) để thông báo cho đơn vị điều khiển.

Đối với các thanh ghi, hai thao tác cơ bản cần thực hiện được là:

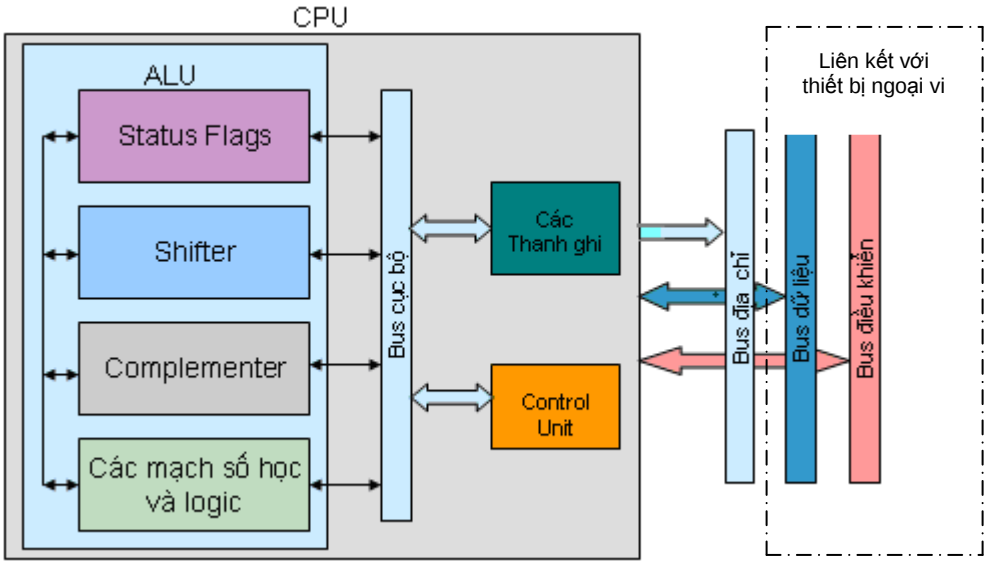
- Di chuyển dữ liệu từ một thanh ghi sang một thanh ghi khác.



Binary adder used as 2's-complement adder/subtractor



A multifunction ALU with 8 control signals (2 for function class, 1 arithmetic, 3 shift, 2 logic) specifying the operation



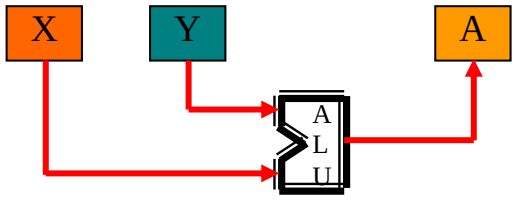
– Cộng hoặc trừ nội dung một thanh ghi vào hoặc từ nội dung của một thanh ghi khác.

Các phép toán số học được ALU thực hiện thường là dãy các thao tác thuộc hai loại trên. Các phép toán phức tạp hơn, chẳng hạn phép nhân, có thể bao gồm rất nhiều các thao tác cũng thuộc hai loại nói trên.

Bộ cộng trong ALU thường được thiết kế từ các đơn vị sau: mạch bán cộng (half-adder) và mạch cộng đầy đủ (full-adder), thanh ghi dịch, mạch lấy phần bù…. Cấu trúc của half-adder (HA) và full-adder (FA) được xây dựng từ các phần tử logic cơ bản và đã trình bày trong phần các kiến thức cơ bản, mục 2.5 của chương I : Mạch cộng hai số liệu nhị phân.

2. Phát triển kiến trúc CPU

2.1. Khái quát



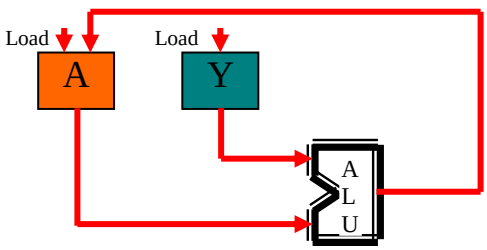
Giả thiết một CPU với chỉ 3 thanh ghi có độ dài (số bit) như nhau là X,Y và A. Thanh ghi X chứa một toán hạng, thanh ghi Y chứa toán hạng thứ 2 và thanh ghi A chứa kết quả phép toán thực hiện trên hai toán hạng X và Y. Đơn vị số học ALU có khả năng thực hiện các phép toán số học *cộng, trừ,* các phép toán logic *AND, OR.* Thanh ghi A gọi là thanh ghi gộp (*Accumulator- Acc).* Các phép toán sẽ được thực hiện theo biểu thức:

Phép cộng: $X + Y = A$

Phép trừ: $X - Y = A$

Phép AND: X *and* $Y = A$

Phép OR: X *or* $Y = A$



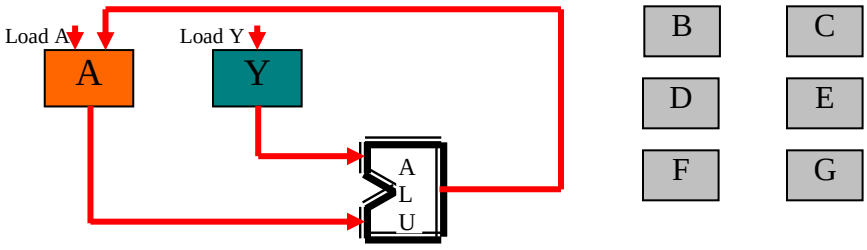
Vấn đề đặt ra là có thể tổ hợp hai thanh ghi X và A thành chỉ một thanh ghi, có thể sử dụng để chứa một toán hang, giả sử toán hạng X, và sau khi thực hiện phép toán, sẽ là nơi chứa kết quả. Xét về mặt cấu trúc, ta đã bớt đi được một thanh ghi, song phép toán sẽ được thực hiện như sau: Toán hạng X sẽ được nạp vào thanh gi A, toán hạng Y nạp vào thanh ghi Y, ALU thực hiện phép toán theo yêu cầu. Phép toán có thể mô hình như sau:

Phép cộng: $A \leftarrow A + Y$

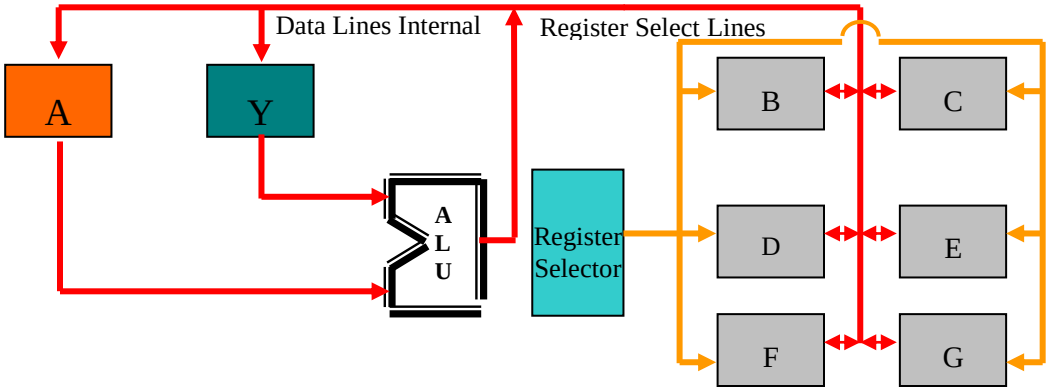
Phép trừ: $A \leftarrow A - Y$

Phép AND: $A \leftarrow A \wedge Y$

Phép OR: $A \leftarrow A \vee Y$

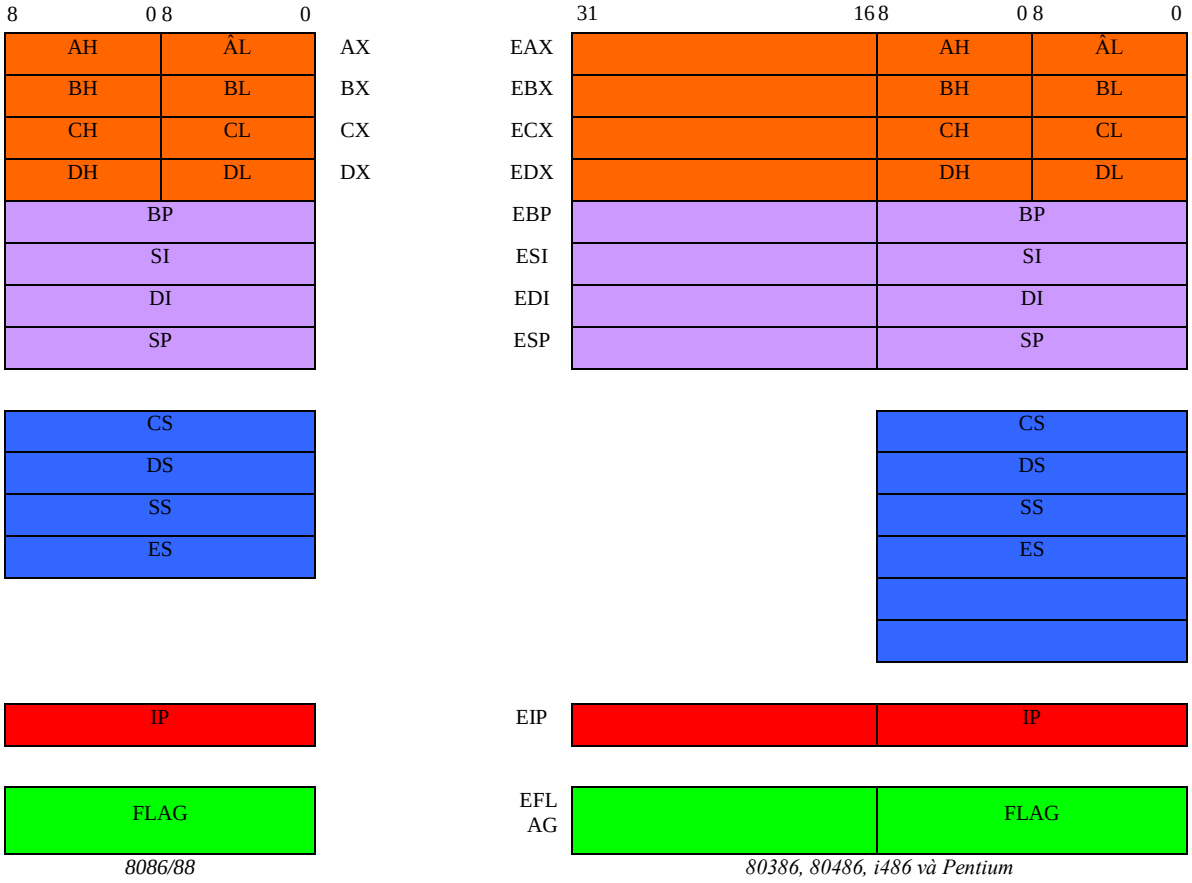


Một ý tưởng có thể thực hiện là: Sẽ là rất thuận lợi, cho dù có thể không cần thiết, nếu CPU có thêm một vùng nhớ với tốc độ truy xuất thật nhanh để làm nơi lưu giữ các kết quả trung gian trong những phép toán phức tạp hơn. Ví dụ ta phải thực hiện tính kết quả của một biểu thức trong đó có 6 phép toán, kết quả của phép thứ nhất cần được lưu giữ tạm thời để sử dụng trong phép toán thứ sáu. Như vậy một *bộ nhớ trong (Internal Memory)* là cần thiết để khả năng trên được thực hiện. Coi rằng, các thanh ghi làm nhiệm vụ bộ nhớ trong đó là những thanh ghi gộp phụ, giả sử là B, C, D, E, F và G chẳng hạn, ta được một cấu trúc CPU như hình trên.



Giả sử các thanh ghi A, B, C, D, E, F, và G có thể sử dụng như những thanh ghi gộp phụ, bằng cách liên kết chúng vợi các thanh ghi A và thanh ghi Y qua một BUS dữ liệu nội bộ (Data BUS Internal), thêm vào một mạch xác định thanh ghi nào sẽ được chọn để Ghi hoặc Đọc dữ liệu bằng khối Register Selector, ta sẽ được một cấu trúc CPU với các thanh ghi hoàn chỉnh hơn như ở hình trên.

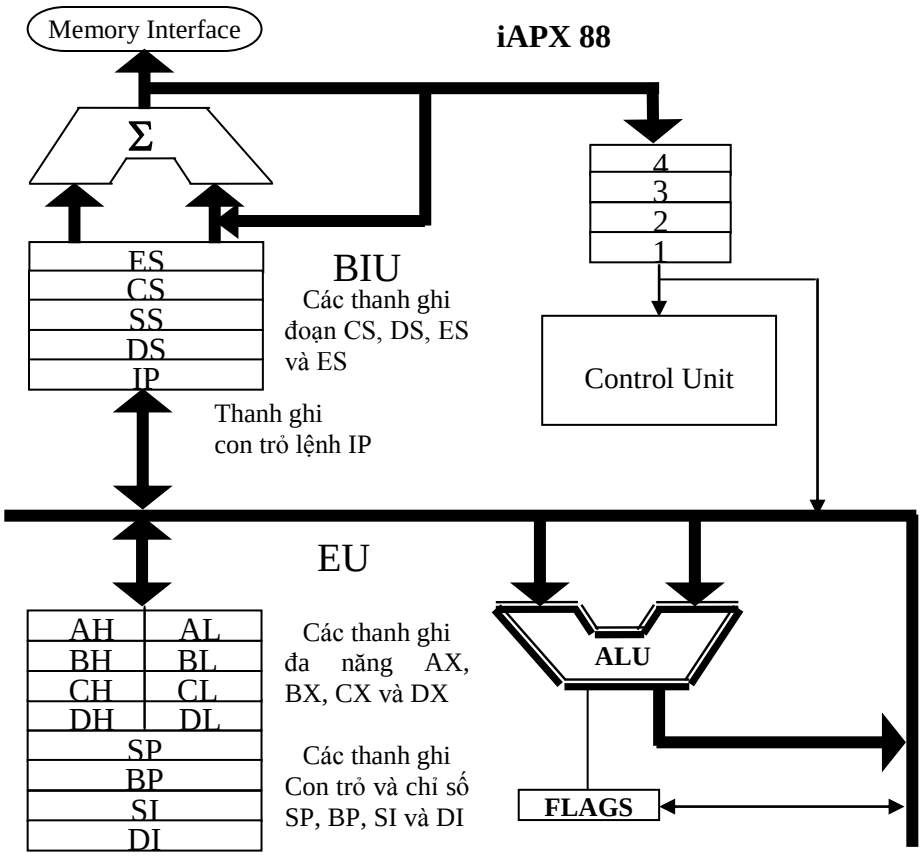
2.2. Tổ chức thanh ghi trong CPU họ x86



Hình III.1. Tổ chức các thanh ghi trong các CPU họ x86

Nguyễn Trung Dũng – Tel: 0983 410 866

Có thể coi các thanh ghi của các CPU họ x86 như một bộ nhớ được đặt ngay bên trong CPU, có tốc độ truy cập cực kỳ nhanh, được dùng để chứa các toán hạng hoặc lưu giữ các dữ liệu và các kết quả tạm thời của các quá trình tính toán, xử lý. Các thanh ghi trong họ CPU 80x86 có độ dài khác nhau, 16 bits với các trung tâm xử lý 8088/86, 80188/86 và 80286, 32 bits với các trung tâm 80386/486 trở đi và được mô tả trên *Hình III.1*.



Hình III.1. Cấu trúc CPU 8088 của Intel

1. Các thanh ghi đa năng:

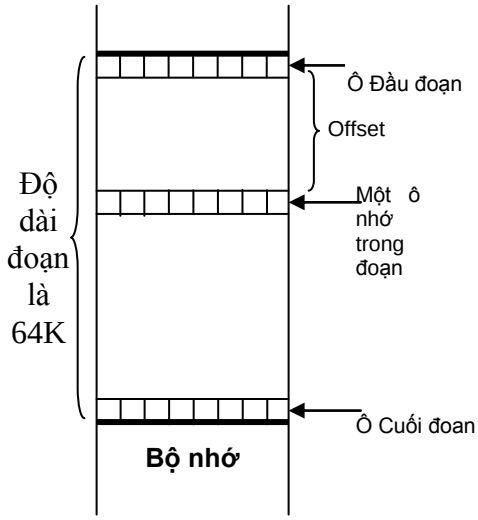
CPU 8086 có 8 thanh ghi đa năng với tên gọi là AH, AL, BH, BL, CH, CL, DH, DL. Những thanh ghi này có thể sử dụng riêng rẽ cho việc lưu giữ các dữ liệu nhị phân 8 bits. Cũng có thể sử dụng chúng thành từng cặp thanh ghi có tên gọi là AX (AH-AL), BX (BH-BL), CX (CH-CL), và DX (DH-DL) để lưu giữ các dữ liệu nhị phân 16 bits.

Ưu điểm của việc sử dụng các thanh ghi này để lưu giữ tạm thời các dữ liệu là tốc độ truy cập của CPU với chúng nhanh hơn rất nhiều so với việc sử dụng các ô nhớ.

2. Các thanh ghi đoạn:

CPU đưa ra BUS địa chỉ 20 bits để quản lý một không gian nhớ 1Mbyte (1.048.576 Bytes) bộ nhớ vật lý. Tuy nhiên, các thanh ghi trong CPU lại chỉ có độ dài 16 bits, do vậy, không gian nhớ được chia thành *từng*

đoạn (*segment*), mỗi đoạn dài 64Kbytes, địa chỉ của Byte đầu tiên được lấy làm *địa chỉ đoạn*. Hai đoạn nhớ kề cận cách nhau tối thiểu là 16 Bytes. Mỗi Byte nhớ trong đoạn sẽ được xác định bởi *độ lệch* (offset), tức là khoảng cách tính từ Byte nhớ đó đến đầu đoạn.



Hình II. 17 Về khái niệm địa chỉ đoạn và địa chỉ offset

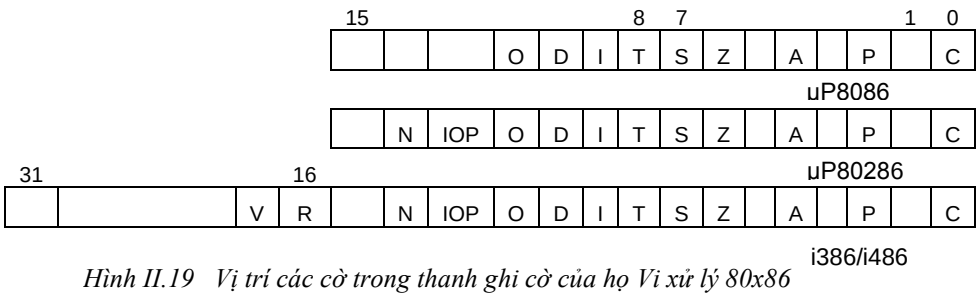
Địa chỉ vật lý = (Segment) x 10H + (offset)

µP8086 sử dụng 4 thanh ghi đoạn riêng biệt là: *Thanh ghi đoạn mã lệnh CS* (*Code Segment*), *thanh ghi đoạn ngăn xếp SS* (*Stack Segment*), *thanh ghi đoạn mở rộng ES* (*Extra Segment*) và *thanh ghi đoạn dữ liệu DS* (*Data Segment*).

- *Thanh ghi đoạn mã lệnh* CS là thanh ghi chứa địa chỉ bắt đầu của đoạn chương trình hiện hành trong bộ nhớ
- *Thanh ghi đoạn dữ liệu* DS là thanh ghi địa chứa chỉ bắt đầu của đoạn chứa số liệu hiện hành trong bộ nhớ, hay còn gọi là nơi chứa các biến của chương trình
- *Thanh ghi đoạn ngăn xếp* SS là thanh ghi địa chứa chỉ bắt đầu của đoạn ngăn xếp (Stack) trong bộ nhớ (ô nhớ do thanh ghi này chỉ đến còn được gọi là *đáy ngăn xếp*), nơi lưu giữ địa chỉ và dữ liệu khi thực hiện các chương trình con, lệnh gọi chương trình con hoặc thủ tục
- *Thanh ghi đoạn mở rộng* ES là thanh ghi địa chứa chỉ bắt đầu của đoạn chứa các dữ liệu chuỗi, xâu ký tự
- Ngoài ra, trong các trung tâm i386/i486 còn có hai thanh ghi đoạn FS và GS.

3. *Thanh ghi cờ FLAG:*

Chỉ có 9 trong số 16 bits của thanh ghi cờ trong các bộ vi xử lý µP8086 - µP80286 và 11 trong số 32 bits của thanh ghi cờ trong các bộ xử lý i386/i486 được sử dụng. Mỗi cờ có thể được lập (= "1") hay xoá (= "0") để biểu thị trạng thái kết quả của một phép xử lý trước đó hoặc trạng thái hiện tại của CPU. Các cờ IOP, N, R và V liên quan đến chế độ bảo vệ trong các bộ xử lý 80286 và i386/i486. Chín cờ còn lại gồm 6 cờ chỉ trạng thái và 3 cờ điều khiển.



Hình II.19 Vị trí các cờ trong thanh ghi cờ của họ Vi xử lý 80x86

Các cờ trạng thái gồm:

- *Cờ nhớ* CF (carry flag) được lập nếu một thao tác xẩy ra hiện tượng *carry* hoặc *borrow* đối với toán hạng đích. CF có thể lập bởi lệnh STC và xoá bởi lệnh CLC.
- *Cờ chẵn lẻ* PF (parity flag) được lập nếu kết quả của một phép xử lý có số bit bằng "1" là số chẵn.
- *Cờ mang phụ* AF (auxiliary flag) được dùng cho xử lý các mã BCD và được lập nếu thao tác xử lý gây hiện tượng carry hoặc borrow cho 4 bits thấp của toán hạng
- *Cờ zero* ZF (zero flag) được lập nếu kết quả xử lý số liệu có kết quả bằng 0
- *Cờ dấu* SF (Sign flag) dấu tương ứng với MSB của kết quả phép toán, được lập với kết quả dương và xoá với kết quả âm
- *Cờ tràn* OF (Overflow flag) nếu kết quả phép toán là quá lớn cho toán hạng đích.

Các cờ điều khiển gồm:

- *Cờ hướng* DF (direction flag) xác định hướng của phép toán xử lý xâu, chuỗi ký tự, nếu được lập, xâu, chuỗi sè được xử lý từ địa chỉ cao tới địa chỉ thấp và ngược lại. Cờ được lập bởi lệnh STD và xoá bằng lệnh CLD
- *Cờ ngắt* IF (Interrupt enable flag) nếu được lập, CPU sẽ chấp nhận yêu cầu ngắt cứng và phục vụ ngắt. Được lập bởi lệnh STI và xoá bằng lệnh CLI
- *Cờ bẫy* TF (Trap flag) Dùng trong gỡ rối chương trình (Debuger) Không thể lập hay xoá trực tiếp bởi lệnh của máy.

4. Thanh ghi con trỏ lệnh IP

Thanh ghi con trỏ lệnh IP (Instruction Pointer) – thanh ghi 16 bits dùng để lưu giữ phần offset của địa chỉ lệnh kế tiếp sẽ được thực hiện trong tuần tự thực hiện chương trình. Kết hợp với CS, IP giống như thanh đếm chương trình PC, mỗi lần từ lệnh được đọc ra từ bộ nhớ, BIU sẽ thay đổi giá trị IP tuỳ theo độ dài của từ lệnh (số bytes của từ lệnh) sao cho nó chỉ đến từ lệnh kế tiếp trong bộ nhớ chương trình. Cũng cần nói thêm rằng khi gặp các lệnh rẽ nhánh hoặc lệnh gọi chương trình con, gọi thủ tục …, các giá trị của CS:IP sẽ thay đổi đột ngột không theo quy luật trên. Các giá trị mới của CS:IP do người lập trình cung cấp thông qua địa chỉ của các nhãn (Label) trong chương trình hoặc giá trị cụ thể.

5. *Các thanh ghi dữ liệu*

Có 4 thanh ghi dữ liệu:

- *Thanh ghi tích luỹ* AX (Accummulator register) thường dùng để lưu giữ các kết quả xử lý
- *Thanh ghi cơ sở* BX (Base register) thường dùng chỉ địa chỉ cơ sở (đáy) của một vùng nhớ trong bộ nhớ
- *Thanh ghi đếm* CX (Counter register) thường dùng để khai báo số lần một thao tác nào đó phải được thực hiện trong các vòng lặp, phép dịch, phép quay…, Giá trị của nội dung thanh ghi CX sẽ giảm đi một sau mỗi thao tác
- *Thanh ghi số liệu* DX (Data register) thường dùng để lưu giữ số liệu dùng làm thông số chuyển giao cho một chương trình. DX là thanh ghi duy nhất được dùng để chứa địa chỉ của các thiết bị vào/ra.

Tất cả 4 thanh ghi trên đều được tạo thành từ cặp các thanh ghi 8bit, được gọi là byte thấp và byte cao: AH, AL, BH, BL, CH, CL và DH, DL.

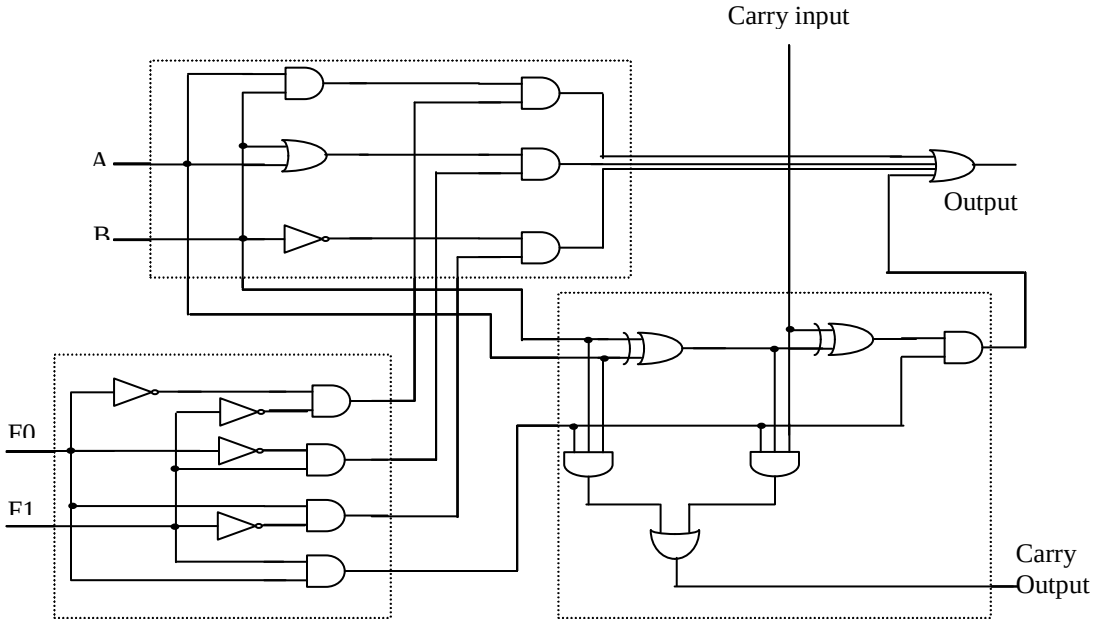
6. *Các thanh ghi con trỏ và chỉ số*

Có 2 thanh ghi con trỏ và 2 thanh ghi chỉ số:

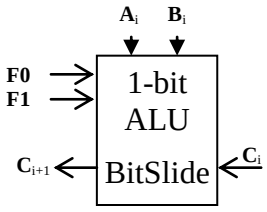
- *Thanh ghi con trỏ ngăn xếp* SP (Stack Pointer) chứa địa chỉ đỉnh ngăn xếp (vùng nhớ đặc biệt, hoạt động theo nguyên tắc LIFO – Last In First Out – vào sau ra trước) sử dụng cho việc lưu giữ tạm thời các dữ liệu hay địa chỉ khi gọi chương trình con, khi phục vụ ngắt v.v…giá trị nội dung của SP luôn luôn là phần offset của địa chỉ ngăn xếp kế tiếp

- *Thanh ghi con trỏ cơ sở* BP (Base Pointer) có chức năng chứa giá trị offset tính từ địa chỉ SS nhưng còn được sử dụng để truy cập dữ liệu bên trong ngăn xếp
- *Các thanh ghi chỉ số nguồn* DI và *thanh ghi chỉ số đích* SI (Destination Index và Source Index) được dùng để lưu giữ các thành phần offset đối với những vùng dữ liệu được cất trong đoạn dữ liệu. Hai nội dung của hai thanh ghi này liên kết với nội dung thanh ghi đoạn DS để tạo ra địa chỉ nguồn và địa chỉ đích của vùng nhớ.

Để hiểu rõ hơn về cấu trúc của ALU, ta có thể tham khảo sơ đồ nguyên lý mạch ALU xử lý 1 bit. Sơ đồ mạch ALU 1 bit thực hiện các phép Cộng, AND, OR và NOT (BitSlide), A, B là các bit đầu vào, F0 và F1 là tổ hợp bit xác định chức năng phép toán.

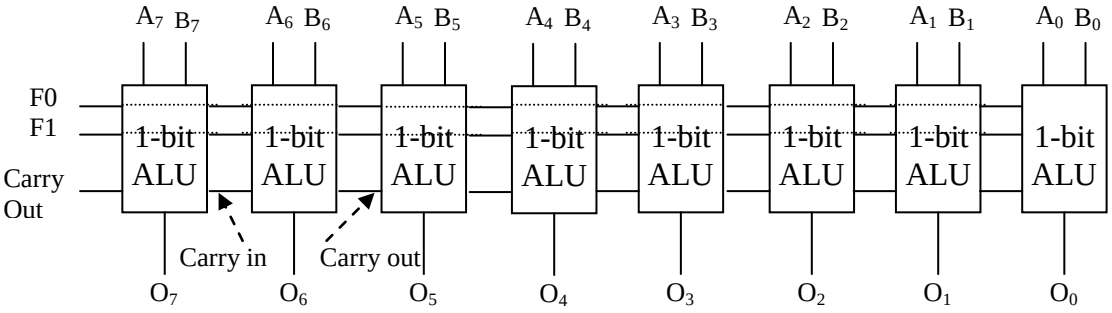


BitSlide thực hiện được những phép toán số học và logic theo bảng sau:



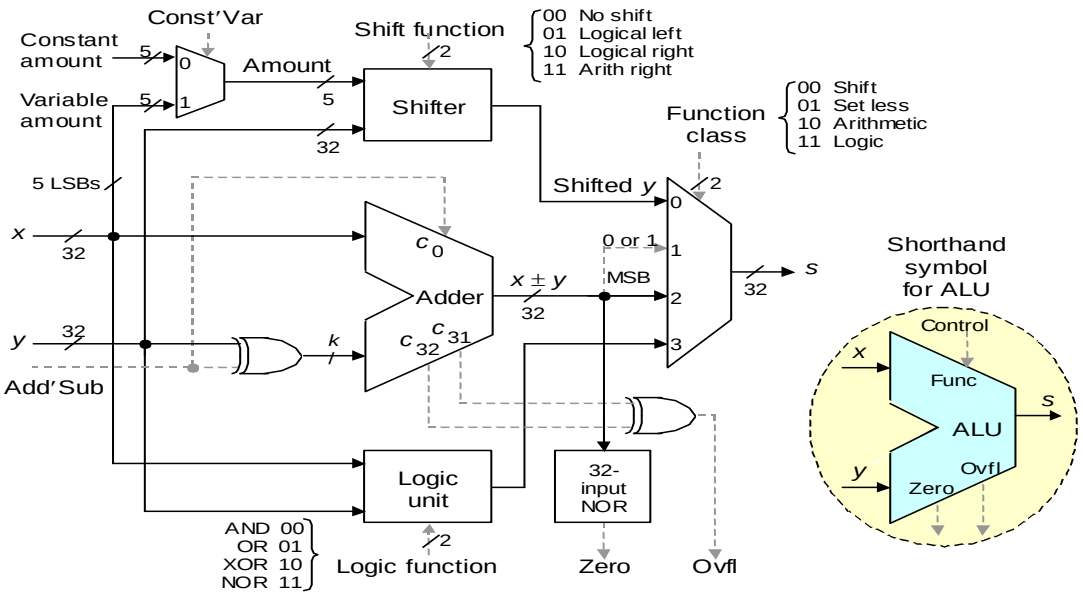
F1F0	Phép toán	
00	Ai ∩Bi	Phép AND
01	$\overline{Bi}$	Phép NOT B
10	Ai∪Bi	Phép OR
11	Ai+Bi+Ci	Phép cộng

Giả sử cần xây dựng một đơn vị ALU thực hiện các phép toán với dữ liệu 8 bit, ta có thể ghép các BitSlide nối tiếp nhau, mỗi Slide thực hiện phép toán trên một bit, bit Carry Out sẽ dược chuyển lên Slide ké tiếp bên trái như hình vẽ sau, thực hiện các phép tính cộng số học, phép VÀ (Logic AND), phép HOẶC (Logic OR) trên hai dữ liệu nhị phân 8 bit A B. và phép ĐẢO (Logic NOT) đối với riêng toán hạng B.

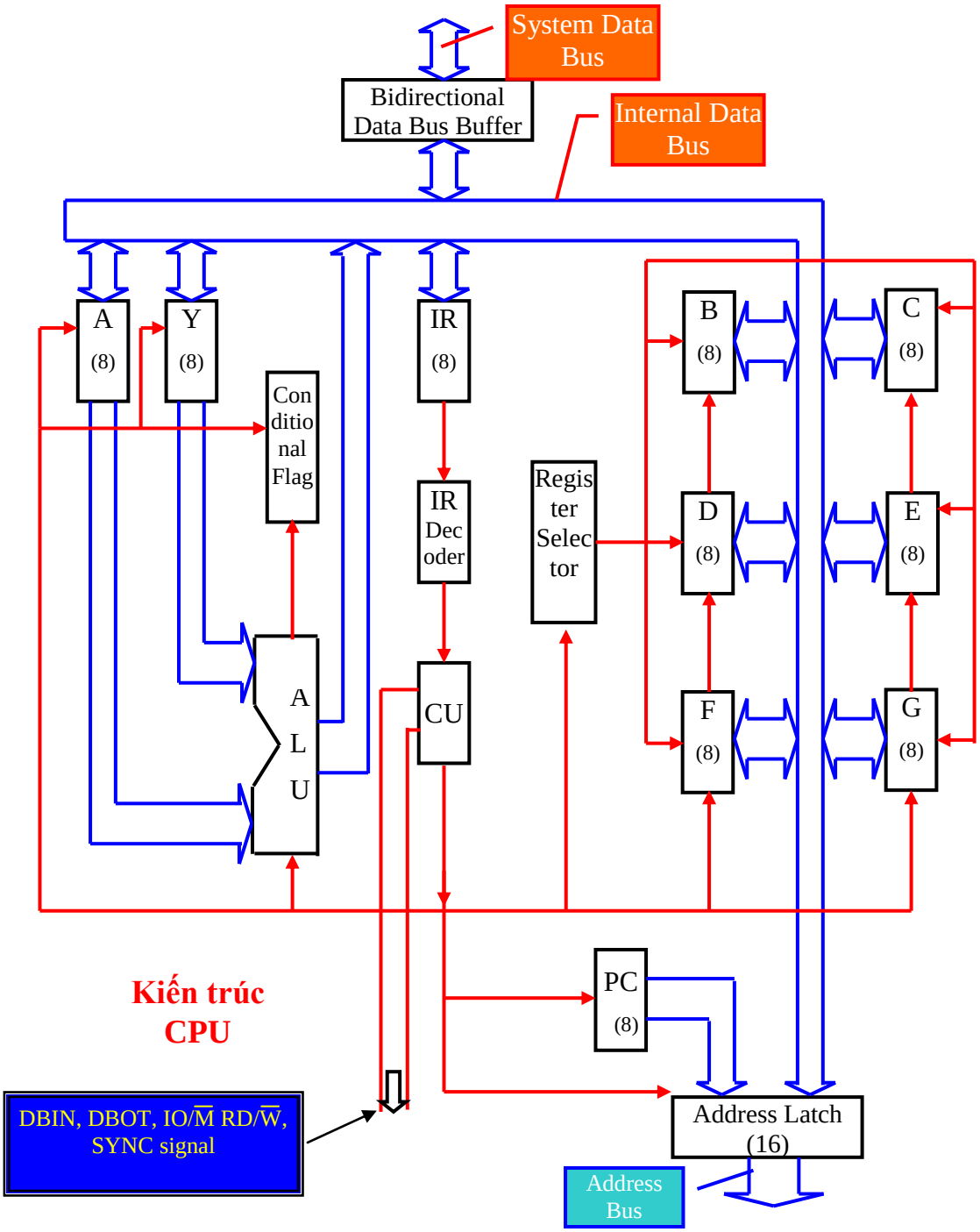


Xây dựng ALU 8-bit từ 8 "mảnh" BitSlide

Hình dưới là sơ đồ nguyên lý một đơn vị số học-logic của CPU xử lý các dữ liệu 32 bit trong các mối liên kết với các tín hiệu điều khiển các chức năng thực thi phép toán.



Nguyễn Trung Đồng – Tel 0983 410 866



3. Kiến trúc CU – Control Unit

3.1. Khái quát

CU-Control Unit là đơn vị điều khiển, điều phối mọi hoạt động của các bộ phận chức năng trong CPU thông qua Control BUS. Có thể coi CU là khối dịch lệnh của CPU, nó tạo ra các tín hiệu tương ứng làm đầu vào cho

Controller để điều khiển hoạt động của các khối chức năng. Các tín hiệu do CU tạo ra có thể phân thành 2 loại: Tín hiệu định thời và tín hiệu điều hành hoạt động của CPU. Các tín hiệu định thời do CU tạo ra xác định trạng thái của CPU làm việc:

- *Đang ở chế độ đọc dữ liệu vào (Input mode)*
- *Đang đưa dữ liệu ra (Output mode)*
- *Đang bắt đầu mộttác vụ khác (Beginning another operation).*

Các tín hiệu trạng thái của CPU xác định CPU đang:

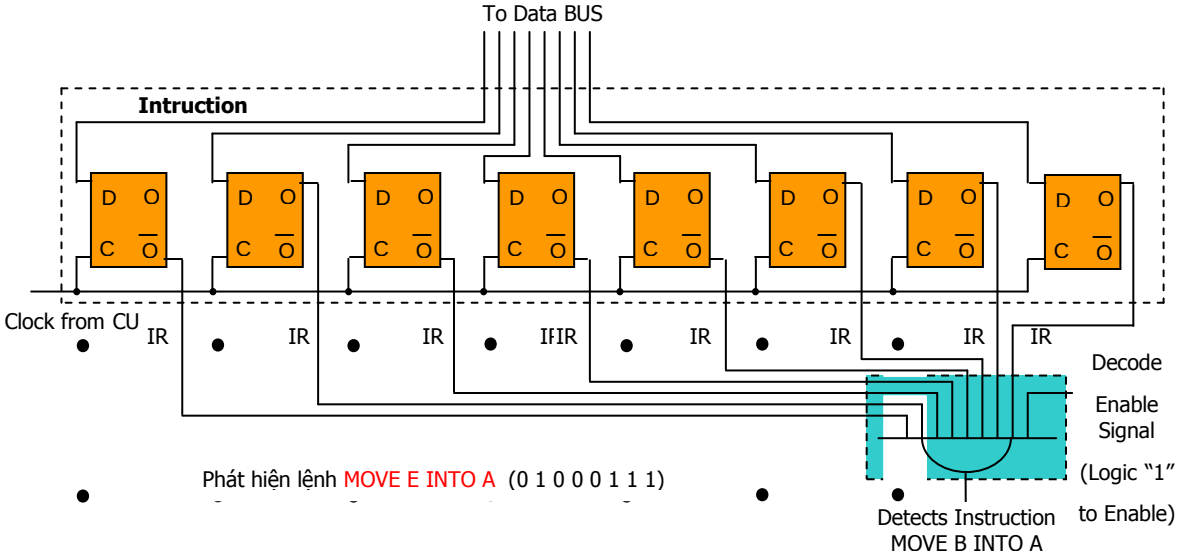
- *Đọc dữ liệu từ bộ nhớ (Memory Read)*
- *Ghi dữ liệu vào bộ nhớ (Memory Write)*
- *Nhận lệnh (Instruction Fetch)*
- *Đọc dữ liệu từ I/O (I/O Read)*
- *Đưa dữ liệu ra I/O (I/O Write)*

Cũng có thể có những thao tác không được nêu ở đây, nhưng chỉ các thao tác trên là quan trọng nhất. Các tín hiệu để phân biệt các tác vụ trên gồm: IO/M, RD/W và DBIN, DBOT.

Cần hiểu rằng việc sử dụng mạch Logic Controller tạo các tín hiệu điều khiển dựa vào các tín hiệu trạng thái của CPU và tín hiệu định thời, có nghĩa là tạo tín hiệu gì và vào thời điểm nào.

Để hiểu được kiến trúc khối CU, hãy tìm lời giải đáp cho câu hỏi: *Sau khi nhận lệnh, CPU làm sao " biết" phải thực hiện những thao tác nào để thực hiện lệnh?*

Tất cả các lệnh đều được biểu diễn dưới dạng mã nhị phân. Giả sử lệnh được biểu diễn bằng một mã 8 bits 01000111B (chuyển nội dung thanh ghi B sang thanh ghi A, ký hiệu là [A] ⇐ [B]).

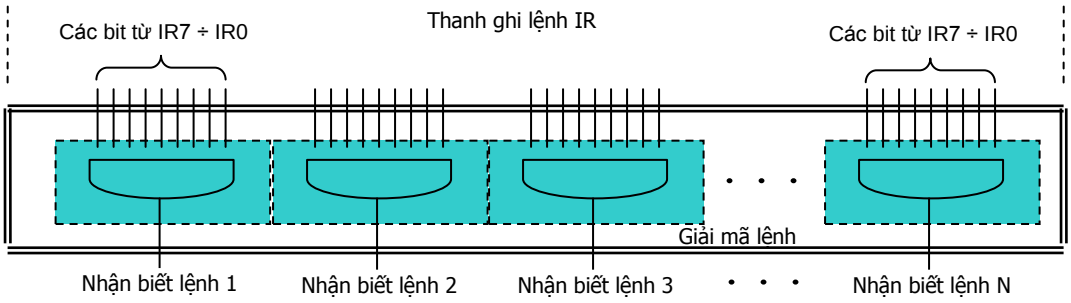


Nguyễn Trung Đồng – Tel 0983 410 866

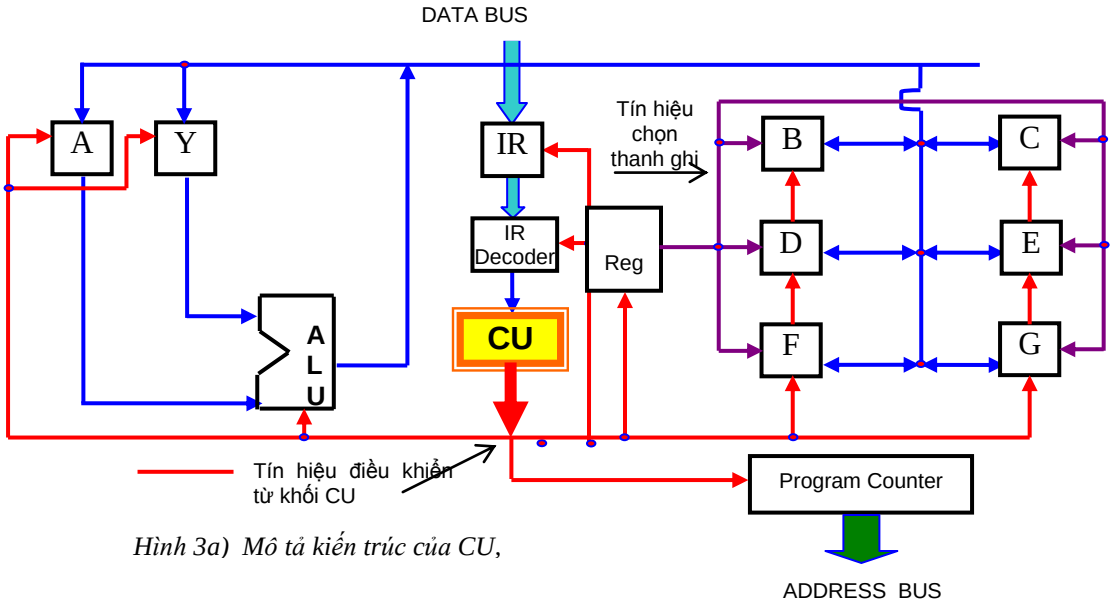
Trước hết, lệnh phải được *giải mã*. Một mạch AND có thể sử dụng để tạo ra tín hiệu nhận biết lệnh (Hình 1) Đầu vào của mạch AND này được nối với đầu ra của thanh ghi lệnh. Đầu ra ở mức "1" xác nhận sự hiện diện của lệnh MOVE B TO A theo công thức

$$(\text{MOVE B TO A}) = \overline{IR}_7.IR_6.\overline{IR}_5.\overline{IR}_4.\overline{IR}_3.IR_2.IR_1.IR_0.$$

trong đó IR_n là đầu ra của các flip-flop tương ứng với các giá trị nhị phân của mã lênh MOVE B TO A. Mạch AND nhận biết mã lệnh được gọi là mạch giải mã lệnh. Như vậy, nếu CPU sử dụng 8 bit để mã hoá các lệnh, có thể có 256 lệnh, và mạch giải mã lệnh cũng sẽ cần đến 256 mạch AND tương tự, tuy nhiên đầu vào của mỗi mạch là một tổ hợp duy nhất trong 256 tổ hợp có thể.



Để thực hiện lệnh, khối điều khiển CU xúc tiến mọi thao tác ngay bên trong CPU bằng cách tạo ra các tín hiệu điều khiển và các xung nhịp để định thời cho các khối chức năng thực hiện các thao tác.



Hình 3a) Mô tả kiến trúc của CU,

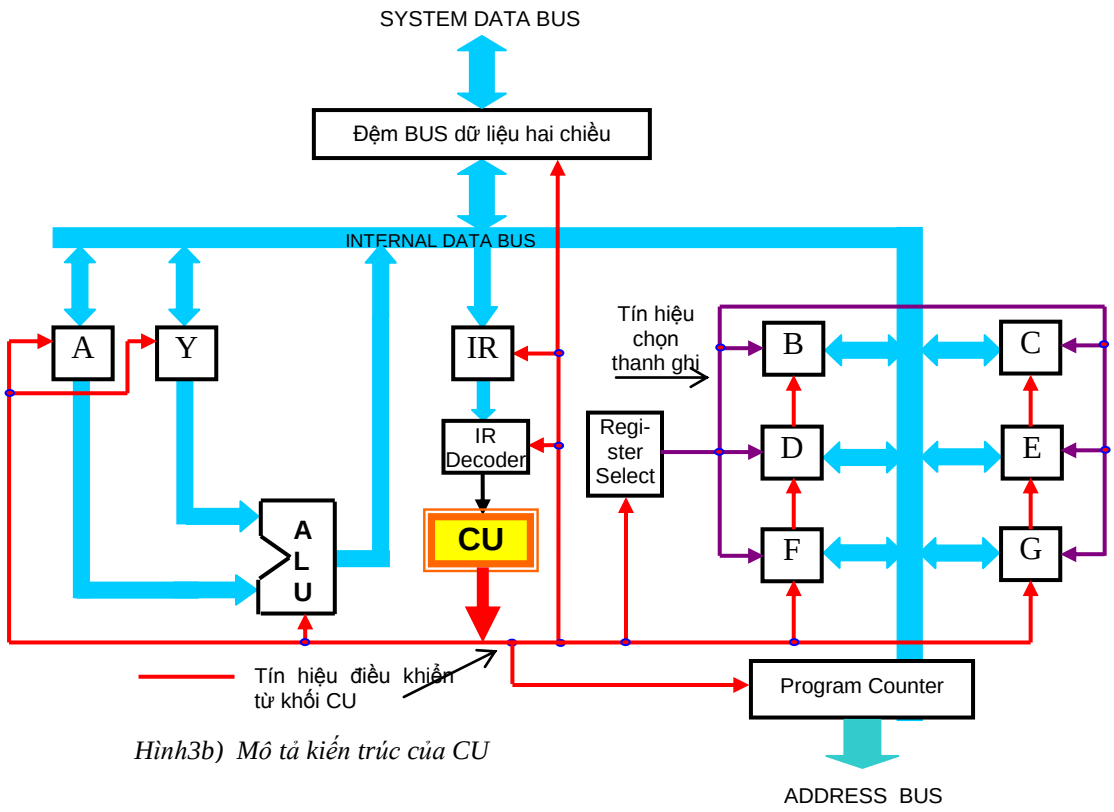
Nguyễn Trung Đồng – Tel 0983 410 866

Sau khi nhận tín hiệu từ khối giải mã lệnh (Instruction Decoder), CU sẽ tạo ra các tín hiệu điều khiển và các xung nhịp. Tín hiệu điều khiển sẽ cho phép (Enable) khối chọn thanh ghi (Register Select) chọn thanh ghi B và thiết lập hệ thống đường truyền thông suốt giữa hai thanh ghi B và A. tiếp theo CU sẽ tạo các tín hiệu tương ứng để việc truyền dữ liệu giữa hai thanh ghi được thực hiện.

Tiếp theo, CU điều khiển thanh đếm chương trình PC tăng lên 1 để nhận tiếp lệnh từ bộ nhớ. Vì CU có nhiệm vụ giám sát và điều khiển mọi thao tác của các thành phần chức năng trong CPU, nên các dây điều khiển phải được nối trực tiếp từ CU tới mọi khối chức năng trong CPU như trên hình 3a.

Cũng cần nhận thức rằng, lệnh được CPU lấy từ bộ nhớ. Trong thực tế, dữ liệu để xử lý cũng có thể xuất phát từ bộ nhớ, và các thanh ghi cũng có thể được chọn bất kỳ ngoại trừ thanh ghi lệnh IR và thanh ddeems chương trình PC. Như vậy, lại cần thêm một thanh ghi liên lạc với BUS dữ liệu có nhiệm vụ truy nhập được vào bộ nhớ. Thanh ghi này làm trung gian giữa BUS dữ liệu bên ngoài và các thanh ghi đa năng khác, và nó được liên lạc với nhau thông qua BUS dữ liệu nội bộ (Internal Data BUS), một BUS mà các thanh ghi được truy xuất trực tiếp. CU phải làm nhiệm vụ xác định thanh ghi nào được truy xuất qua BUS dữ liệu nội bộ tại thời điểm đó. Cũng vì BUS dữ liệu nội bộ của CPU truy xuất đến BUS dữ liệu hệ thống, nên cần phải có một cách thức để hoặc cách ly chúng khi cần thiết, hoặc cho phép ghép nối, nên cần thiết phải có thêm *thanh ghi đệm dữ liệu hai chiều*. Và như vậy, CU phải làm nhiệm vụ *điều khiển hướng di chuyển* của dữ liệu khi đi qua thanh ghi đệm (xem hình 3b).

Giả thiết rằng lối ra của khối CU (*tạo các tín hiệu điều khiển*) phải tạo ra 12 tín hiệu tại các cửa $G_1 - G_{12}$, 2 tín hiệu điều khiển bộ nhớ và 5 tín hiệu xung nhịp kích hoạt các thanh ghi PC (thanh đếm chương trình), MAR (thanh ghi đệm địa chỉ, MSR (thanh ghi đệm bộ nhớ), DO (thanh ghi dữ liệu) và IR (thanh ghi lệnh) để điều khiển quá trình nhận và thực hiện lệnh ADD. Các tín hiệu này được gửi tới để điều khiển hoạt động của các thành phần khác nhau trong CPU. Một chu trình thực hiện lệnh trên sẽ được thi hành.



Hình3b) Mô tả kiến trúc của CU

Số bước	Các tín hiệu điều khiển cửa												Điều khiển bộ nhớ		Các xung nhịp kích hoạt thanh ghi				
	G_1	G_2	G_3	G_4	G_5	G_6	G_7	G_8	G_9	G_{10}	G_{11}	G_{12}	EN	E/W	PC	MAR	MBR	DO	IR
1	1	0	0	0	0	0	0	0	0	0	0	0	0	x	0	1	0	0	0
2	0	1	0	0	0	0	0	0	0	0	0	0	0	x	1	0	0	0	0
3	0	0	0	0	0	1	0	0	0	0	1	0	1	1	0	0	1	0	0
4	0	0	0	0	0	0	1	0	0	0	0	0	0	x	0	0	0	0	1
5	0	0	1	0	0	0	0	0	0	0	0	0	0	x	0	1	0	0	0
6	0	0	0	0	0	1	0	0	0	0	1	0	1	1	0	0	1	0	0
7	0	0	0	0	0	0	1	0	0	1	0	0	0	x	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	1	0	x	0	0	0	1	0

Thực tế trong CPU của máy tính có từ 64 đến hơn 200 các tín hiệu điều khiển như thế. Sự khác nhau quan trọng giữa các lệnh và vi lệnh là ở chỗ vi lệnh có nhiều trường hơn. Tám bước trong bảng trên là một *vi chương trình* dịch một giai đoạn nhận lệnh (OPCODE FETCH) được thực thi sau lệnh cộng ADD. Như vậy một lệnh được dịch thành một chuỗi các vi lệnh, hay nói cách khác, mỗi mã lệnh có một vi chương trình.

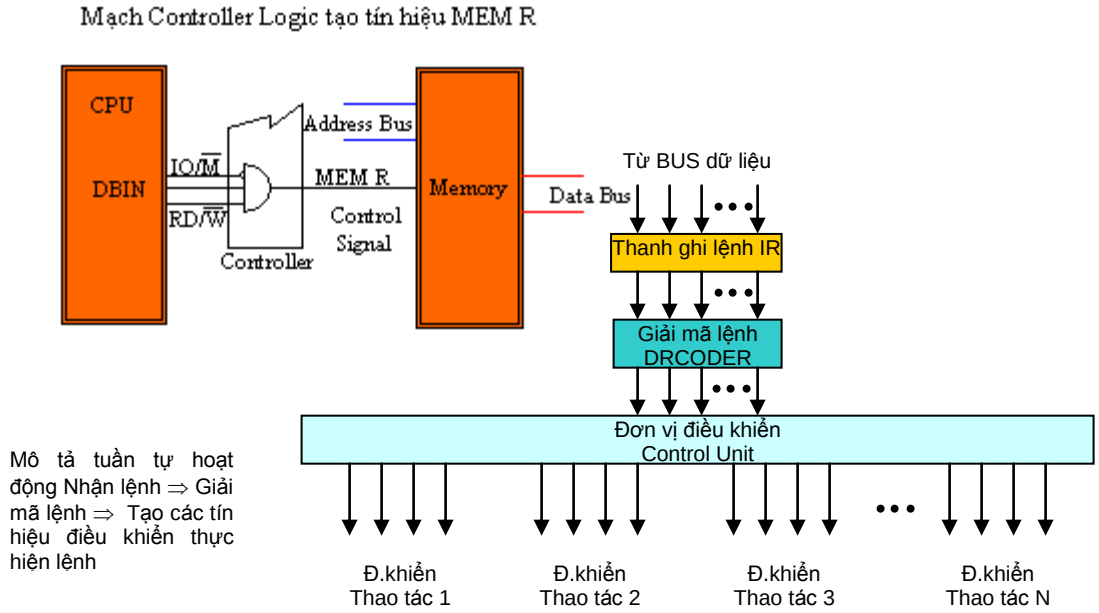
Có thể thấy rằng, *khối tạo các tín hiệu điều khiển*:

– "Biết" phải thực hiện lệnh "như thế nào", một khi lệnh từ IR (Instruction Register) được giải mã và chuyển sang cho nó

– "Giải quyết" việc thực hiện một lệnh bằng cách điều khiển các khối chức năng liên quan thực hiện các phần việc.

Từ cách nhìn nhận trên, dễ dàng nhận ra rằng CU - khối tạo các tín hiệu điều khiển là bộ não thực thụ của CPU. Có thể coi khối này là một máy tính đặc dụng (Special-purpose Computer)(*) bên trong CPU. Nó là hạt nhân cơ bản nhất dành riêng cho việc thực hiện một lệnh. Để thiết kế và xây dựng được khối này, cần phải có một "chương trình" (program)(*) thật chi tiết. Chương trình dùng để xây dựng nên khối này cần phải có những thủ tục tuyệt đối chính xác nhằm mục đích thực hiện các lệnh. Chương trình đó được gọi là Vi chương trình (MicroProgram) và được chế tạo như là một phần tích hợp cứng bên trong CPU, người lập trình không thể thay thế cũng như không thể truy nhập vào được.

Dễ thấy rằng để thực hiện một lệnh, CPU chia "công việc" ra thành một *chuỗi các thao tác cơ bản*. Mỗi một khối chức năng sẽ thao tác một hoặc một số chuỗi thao tác cơ bản đó theo một tuần tự nhất định (khái niệm Vi chương trình). Tín hiệu điều khiển các khối chức năng đó và xung nhịp để đảm bảo sự chính xác tuần tự và đồng bộ các thao tác đó là do CU tạo ra.



(*) Thuật ngữ được dùng trong tài liệu gốc (Manual) của hãng Intel®

3.3. Chức năng của CU

CU - Control Unit là đơn vị điều khiển, điều phối mọi hoạt động của các bộ phận chức năng trong CPU thông qua Bus điều khiển. Có thể xem CU là khối dịch lệnh của CPU, nó tạo ra các tín hiệu tương ứng làm đầu vào cho Controller để điều khiển hoạt động của các khối chức năng.

Các tín hiệu do CU tạo ra có thể phân thành 2 loại: Tín hiệu định thời và tín hiệu điều hành hoạt động của CPU.

Các tín hiệu định thời do CU tạo ra xác định trạng thái của CPU làm việc:

- Đang ở chế độ đọc dữ liệu vào (Input mode)
- Đang đưa dữ liệu ra (Output mode)
- Đang bắt đầu một hoạt động khác (Beginning another operation).

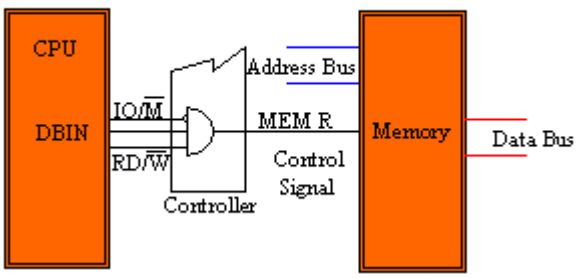
Các tín hiệu trạng thái của CPU xác định CPU đang:

- Đọc dữ liệu từ bộ nhớ (Memory Read)
- Ghi dữ liệu vào bộ nhớ (Memory Write)
- Nạp lệnh (Instruction Fetch)
- Đọc dữ liệu từ I/O (I/O Read)
- Đưa dữ liệu ra I/O (I/O Write)

Cũng có thể có những hoạt động không được nêu ở đây, nhưng các thao tác trên là quan trọng nhất. Các tín hiệu để phân biệt các hoạt động trên gồm: IO/M, RD/W và DBIN, DBOT.

Cần hiểu rằng việc sử dụng mạch Controller Logic tạo các tín hiệu điều khiển dựa vào các tín hiệu trạng thái của CPU và tín hiệu định thời, có nghĩa là tạo tín hiệu gì và vào thời điểm nào.

Mạch Controller Logic tạo tín hiệu MEM R



Nguyễn Trung Đồng – Tel 0983 410 866

3.4. Kiến trúc CU

CU chỉ chấp nhận các lệnh có khả năng thực thi được thông qua vi chương trình. Các lệnh này tạo nên hệ lệnh của CPU, từ đó người lập trình xây dựng được các chương trình. Để hiểu về khái niệm hệ lệnh, hãy bắt đầu bằng một hệ lệnh được giả thiết cho cấu trúc CPU. Từ các từ nhớ có độ dài 8 bits, ta có thể thấy rằng tạo được 2^8 tổ hợp, hay nói cách khác có thể xây dựng 256 lệnh.

Tất nhiên, có thể có số lượng lệnh ít hơn, song mỗi lệnh phải thực thi được một công việc cụ thể. Một tổ hợp 8 bits, tuỳ theo mạch CU được thiết kế, sẽ tạo ra các tín hiệu dựa trên các tín hiệu do thanh giải mã lệnh IR Decoder tạo ra. Ta có thể phân loại các lệnh thành các nhóm sau:

1. Nhóm lệnh điều khiển rẽ nhánh
2. Nhóm lệnh di chuyển dữ liệu
3. Nhóm lệnh số học và logic
4. Nhóm lệnh vào/ra dữ liệu.

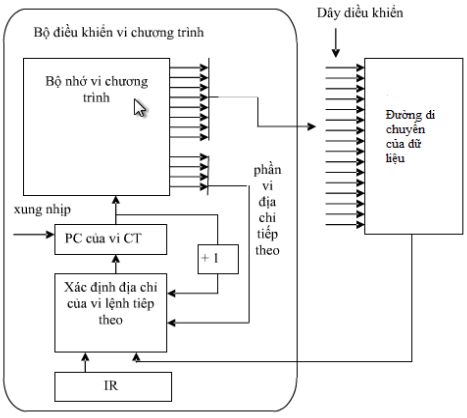
Ký hiệu 8 bits tạo nên lệnh là IR0 đến IR7, để quản lý 4 nhóm lệnh, ta cần 2 bits theo bảng sau:

IR_7	IR_6	IR_5	IR_4	IR_3	IR_2	IR_1	IR_0

IR_7	IR_6	Loại lệnh
0	0	Lệnh rẽ nhánh
0	1	Lệnh di chuyển dữ liệu
1	0	Lệnh số học và logic
1	1	Lệnh I/O và thực hiện theo điều kiện

Bảng mã các thanh ghi như sau:

Mã nhị phân			**Thanh ghi**
0	0	0	**B**
0	0	1	**C**
0	1	0	**D**
0	1	1	**E**
1	0	0	**F**
1	0	1	**G**
1	1	0	**Y**
1	1	1	**A**



Nguyễn Trung Đồng – Tel 0983 410 866

Kết hợp hai bảng trên, ta đã có thể biết được lệnh chuyển nội dung thanh ghi B sang thanh ghi A (MOVE B TO A) sẽ có mã nhị phân là:

0 1 | 0 0 0 | 1 1 1

Chuyển dữ liệu | Thanh ghi B | Thanh ghi A

Đây là cách thức đơn giản minh hoạ kiến trúc hệ lệnh của CPU.

Giả sử phải thực hiện công việc sau: Nếu giá trị cờ Zero (Z) = 1, chuyển nội dung thanh ghi A sang thanh ghi E, nếu không thì tăng nội dung cặp thanh ghi B, C

IF Z = 1 MOVE A TO E OTHERWISE INCREMENT (B, C) AND CONTINUE.

Có thể phân tích như sau:

IR_7	IR_6	IR_5	IR_4	IR_3	IR_2	IR_1	IR_0
1	1						

Theo bảng | I/O=1 | Cờ điều kiện | Increment RP

- 000 – S
- 001 – C
- 010 – P
- 011 – Z

- 00 – BC
- 01 – DE
- 10 – FG

Theo các giá trị được thể hiện ở trên, mã lệnh sẽ có giá trị nhị phân cụ thể như sau:

IR_7	IR_6	IR_5	IR_4	IR_3	IR_2	IR_1	IR_0
1	1	0	0	1	1	0	0

- IR_7 IR_6: Lệnh được thực hiện theo điều kiện
- IR_5: Làm việc với thanh ghi
- IR_4 IR_3 IR_2: Mã địa chỉ của các ô nhớ giá trị cờ Z
- IR_1 IR_0: Mã tăng nội dung của cặp thanh ghi B và C

IF Z = 1 MOVE A TO E OTHERWISE INCREMENT (B, C) AND CONTINUE

Có thể thấy rằng mã lệnh trên sẽ có thể thành lập được. Sử dụng hai bảng mã đã nêu, ta thấy rằng không thể sử dụng một lệnh 8 bits để biểu diễn

lệnh này. Và CPU biết rằng cần phải lấy lệnh tiếp theo theo các điều kiện là trạng thái của các thanh ghi...

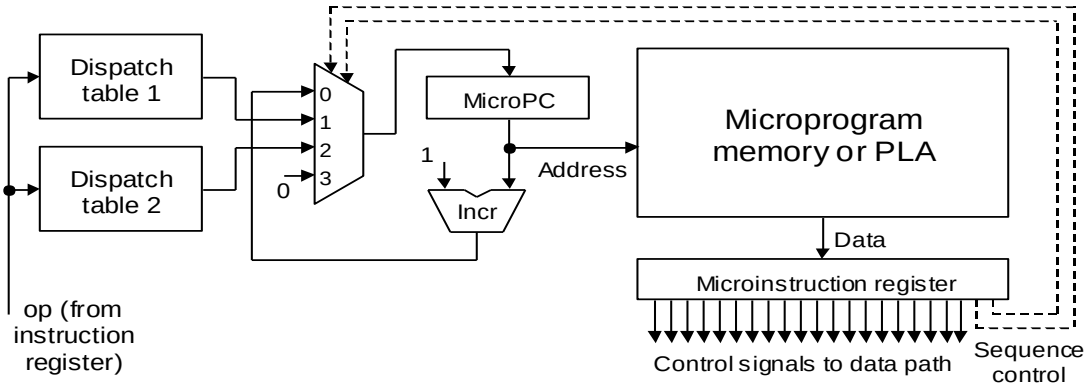
Hai tổ hợp 8 bits liên tiếp biểu diễn yêu cầu

IF Z = 1 MOVE A TO E OTHERWISE INCREMENT (B, C) AND CONTINUE được thành lập dựa trên hai bảng mã trên là:

1 1 0 0 1 1 0 0 (1)

0 1 1 1 1 0 1 1 (2)

Khi IR chuyển tổ hợp (1) cho CU, CU giải mã tổ hợp này và thông qua (IR7 IR6 = 11) và vi chương trình, "biết" rằng, đây là lệnh được thực hiện theo điều kiện. Mã 011 (IR4 IR3 IR2) cho CU biết điều kiện chính là nội dung cờ Z. Nếu {Z}=1, thực hiện lệnh (2), nếu {Z}= 0, bỏ qua lệnh (2) và thực hiện lệnh tăng nội dung cặp thanh ghi BC, (IR1 IR0=00).



4. Vài nét về Kiến trúc CPU Pentium của Intel®

Để tìm hiểu kiến trúc của các CPU Pen tium, trước hết cần nhắc lại tuần tự thực hiện một chương trình trong máy tính.

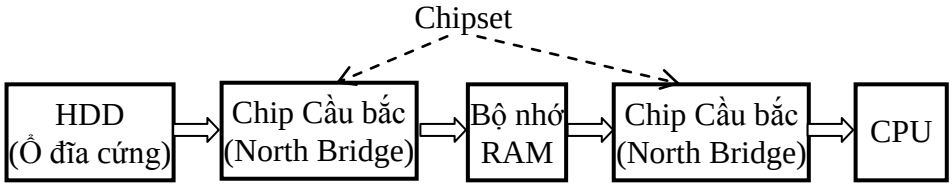
Khi sử dụng các hệ điều hành MS Windows, kích đúp vào một biểu tượng nào đó để chạy chương trình thì những gì sẽ xảy ra là:

Chương trình đã lưu bên trong ổ đĩa cứng (HDD – Hard Disk) sẽ được đưa vào bộ nhớ RAM. Ở đây chương trình chính là một chuỗi các lệnh mà CPU sẽ thực thi. CPU sử dụng một mạch phần cứng được gọi là đơn vị điều

khiển bộ nhớ (*memory controller unit*) để tải lệnh và dữ liệu từ bộ nhớ RAM vào CPU để xử lý. Những gì diễn ra tiếp theo phụ thuộc vào nội dung chương trình vừa được nạp. CPU có thể tiếp tục tải và thực thi chương trình hoặc có thể thực hiện một công việc nào đó với dữ liệu đã được xử lý, ví dụ hiển thị kết quả xử lý lên màn hình.

Đối với các máy tính các thế hệ cũ, CPU điều khiển việc trao đổi dữ liệu giữa ổ đĩa cứng (HDD) và bộ nhớ RAM. Do tốc độ truy cập HDD thấp hơn rất nhiều so với bộ nhớ RAM, nên hiệu suất làm việc của CPU bị thấp đi rất nhiều. Đó là cách thực hiện trao đổi giữa HDD và bộ nhớ theo phương thức PIO (Proccessor I/O hay Programed I/O).

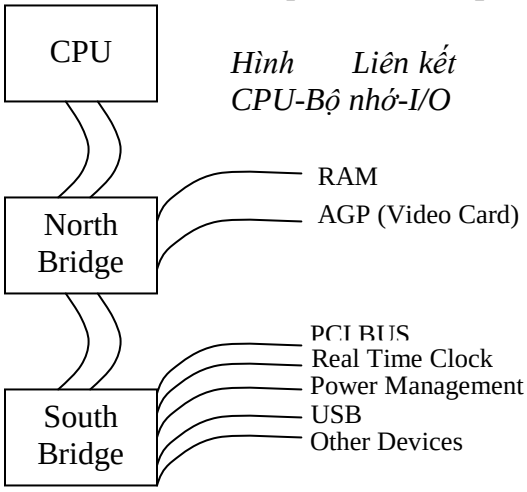
Trong máy tính hiện nay, công việc chuyển tải đó được thực hiện theo phương thức DMA (Direct Memorry Access) nhờ một thiết bị gọi là DMAC (*Direct Memorry Access Controler*). Thực thi công việc này là các chip cầu nối: Chip Cầu bắc và Chip Cầu nam trong các mạch tích hợp (IC) được gọi là Chipset. Tuy nhiên, các bộ xử lý của hãng AMD dựa trên các sockets 754, 939 và 940 sử dụng một *Memory Controller* được nhúng bên trong, do vậy CPU truy xuất trực tiếp bộ nhớ RAM không qua chipset.



Hình Lệnh và Dữ liệu được chuyển vào CPU

4.1. Vai trò của chipset trong máy tính

Chipset là một nhóm các mạch tích hợp được thiết kế để làm việc cùng nhau như một sản phẩm độc lập. Trong máy tính, thuật ngữ chipset nói về các chip đặc biệt trên bo mạch chủ hoặc trên các card mở rộng. Với máy tính dùng Pentium CPU, chipset là hai chip trong bo mạch chủ: Chip Cầu bắc và Chip Cầu nam. Các hãng sản xuất các chipset cho bo mạch chủ máy PC thông dụng là nVIDIA, ATI, VIA Technologies, SiS và Intel. Các máy tính trước đây thường dùng chung một loại chipset giá thành thấp sử dụng giao diện SCSI cho các thiết bị lưu trữ, các



Hình Liên kết CPU-Bộ nhớ-I/O

thiết bị nhúng và các máy tính cá nhân.

Chip Cầu bắc, hay còn gọi là *Memory Controller Hub (MCH)*, là một trong hai chipset trên một bo mạch chủ, chip còn lại là *Chip Cầu nam*. Mặc dù có tên gọi như vậy, nhưng đôi khi, cả hai chip này được tích hợp trong một chip.

Chip Cầu bắc làm nhiệm vụ liên lạc giữa CPU, RAM và thiết bị AGP hoặc PCI Express, một vài loại còn chứa cả chương trình điều khiển video tích hợp, gọi là *Graphics and Memory Controller Hub* (GMCH). Vì các CPU và bộ nhớ RAM khác nhau yêu cầu các tín hiệu khác nhau, nên một Chip Cầu bắc chỉ làm việc với một vài loại CPU, và thông thường chỉ với một loại RAM. Ví dụ chipset Intel i875 chỉ làm việc với hệ thống Pentium IV hoặc Celeron có tốc độ lớn hơn 1,3GHz và DDR SDRAM, còn chipset Intel i915 làm việc với Pentium IV và Intel Celeron sử dụng bộ nhớ DDR hoặc DDR2.

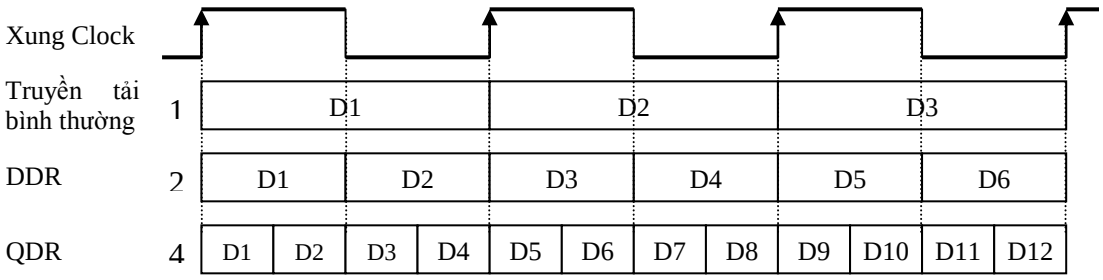
Cũng cần nhắc lại rằng, Chip Cầu bắc trong bo mạch chủ là nhân tố quyết định số lượng, tốc độ và loại CPU cũng như dung lượng, tốc độ và loại RAM có thể sử dụng được. Các máy tính với CPU Pentium thường chỉ quản lý được bộ nhớ tối đa là 128MB, trong khi với CPU Pentium IV, do sử dụng đến 36 bit địa chỉ, nên có thể quản lý được đến 64GB bộ nhớ. Tuy nhiên, các máy tính, do còn bị hạn chế bởi hệ điều hành, nên chúng chỉ hỗ trợ quản lý được một dung lượng RAM nhỏ hơn.

Chip Cầu nam, hay còn gọi là *I/O Controller Hub* (ICH), đảm nhiệm việc giao tiếp với các thiết bị có tốc độ thấp hơn. Chip Cầu nam không được kết nối trực tiếp với CPU, hay nói đúng hơn, Chip Cầu bắc kết nối Chip Cầu nam với CPU. Thông thường, Chip Cầu nam được đặt xa CPU hơn, và thường làm việc với một vài Chip Cầu bắc khác, và mỗi cặp thường phải có thiết kế phù hợp mới có thể làm việc được với nhau. Giao tiếp chung giữa Chip Cầu bắc và Chip Cầu nam là BUS PCI, do vậy xuất hiện hiệu ứng cổ chai.

4.2. Vấn đề xung nhịp (Clock)

Xung nhịp được dùng để đồng bộ hoá hoạt động của các khối chức năng trong máy tính. CPU "biết" được với mỗi lệnh cụ thể, nó cần bao nhiêu xung nhịp để thực thi thông qua một bảng liệt kê các thông tin này (trong vi mã - Microcode). Như vậy nếu 1 lệnh cần *n* xung nhịp để thực thi, thì xung nghịp thứ *n+1* CPU sẽ thực thi lệnh tiếp theo. Nếu CPU có một số khối thực thi song song, chúng có thể thực thi lệnh thứ hai tại cùng thời điểm thực thi lệnh thứ nhất. Đây là nguyên lý của kiến trúc *superscalar*. Nếu so sánh 2 CPU cùng loại với nhau, CPU nào chạy với tần số xung nhịp cao hơn dĩ

nhiên sẽ nhanh hơn. Có nhiều vấn đề phải xem xét về hiệu suất làm việc của CPU, nếu cho rằng, các CPU có số lượng các khối thực thi khác nhau, và kích thước bộ nhớ *cache* khác nhau, thì cách truyền tải dữ liệu bên trong CPU cũng khác nhau. Cũng cần hiểu rằng, bo mạch chủ không thể làm việc với cùng tần số xung nhịp của CPU. Khái niệm nhân tần số xung nhịp được bắt đầu sử dụng ở CPU 486DX2. Với cơ chế nhân xung nhịp, CPU có một bộ tạo xung nhịp ngoài (*External Clock*) để sử dụng khi truyền tải dữ liệu vào/ra bộ nhớ RAM và một xung nhịp bên trong (*Internal Clock*) cao hơn. Ví dụ với Pentium IV 3,4GHz, tần số xung nhịp trong là tần số xung nhịp ngoài 200MHz nhân với hệ số 17. Như vậy, khi đọc dữ liệu từ RAM, CPU Pentium IV 3,4GHz phải giảm tốc độ 17 lần, và nó làm việc như một CPU với tốc độ tương ứng với xung nhịp 200MHz. Một số kỹ thuật được sử dụng để tối thiểu hoá ảnh hưởng của sự khác nhau về xung nhịp. Một là sử dụng bộ nhớ cache, hai là truyền tải nhiều dữ liệu trong một chu kỳ xung nhịp, ví dụ như CPU của hãng AMD chỉ truyền tải hai dữ liệu trong một chu kỳ xung nhịp, còn CPU của hãng Intel tuyền tải bốn dữ liệu.



Kỹ thuật truyền tải hai dữ liệu trên một chu kỳ xung nhịp được gọi là DDR (*Dual Data Rate*), kỹ thuật truyền tải bốn dữ liệu trong một chu kỳ xung nhịp gọi là QDR (*Quat Data Rate*).

Một điểm cần biết nữa là: BUS dữ liệu giữa bộ nhớ RAM và CPU thường là 64bit, hoặc 128bit nếu sử dụng cấu hình bộ nhớ kênh *dual*. Số lượng bit truyền tải và tốc độ xung nhịp được kết hợp trong một giá trị gọi là tốc độ truyền tải, tính theo MB/s, được tính bằng số bit nhân với tần số xung nhịp chia cho 8. Ví dụ bộ nhớ DDR400 có tốc độ truyền tải là 3200MB/s nếu sử dụng cấu hình kênh đơn 64bit, và là 6400MB/ nếu sử dụng cấu hình kênh dual 128bit.

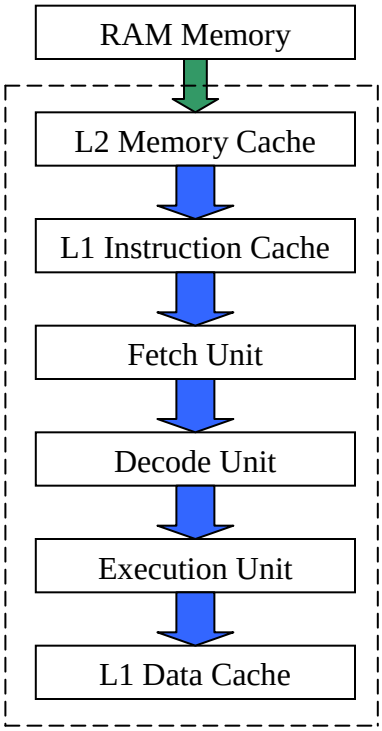
$$T = \frac{Di * fclock}{8}$$

T tốc độ truyền tải dữ liệu
Di số bit của 1 dữ liệu
fclock tần số xung nhịp

Nguyễn Trung Đồng – Tel 0983 410 866

4.3. Về bộ nhớ cache

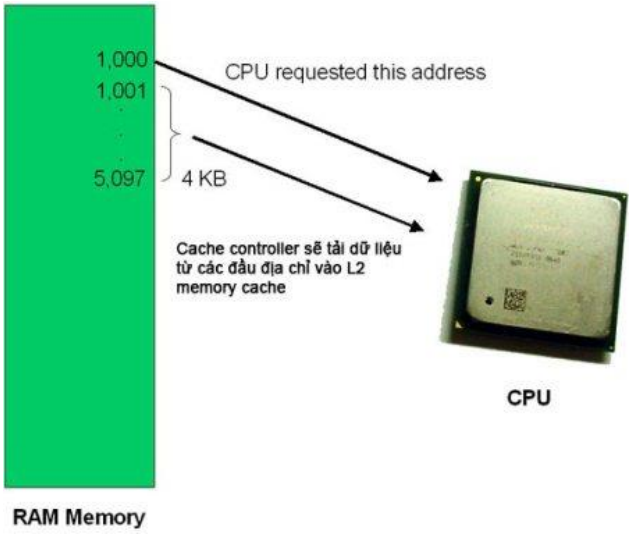
Cơ cấu bộ nhớ cache L2 của bộ nhớ (*L2 memory cache*), L1 của lệnh (*L1 instruction cache*) và L1 của dữ liệu (*L1 data cache*), khối nhận lệnh, giải mã lệnh và thực thi lệnh trong CPU Pentium IV đều được liên kết bằng BUS dữ liệu có độ rộng lên tới 256bit. Về tổ chức, có thể hình dung liên lạc giữa CPU và bộ nhớ RAM như hình dưới đây. BUS dữ liệu liên kết bộ nhớ RAM với CPU có độ rộng là 64 bit, hoặc 128 bit. Liên lạc giữa bộ nhớ cache L2 và bộ nhớ cache lệnh trong Pentium IV là 256 bit. Bộ nhớ cache được sử dụng là RAM tĩnh, có thể hoạt động nhanh như tốc độ của CPU, song tiêu tốn năng lượng và giá thành rất cao. Kỹ thuật cache được sử dụng nhằm mục đích không "bắt" CPU phải lấy dữ liệu từ bộ nhớ RAM với tốc độ thấp, mà khối điều khiển bộ nhớ cache (*memory cache controller*) sẽ đảm nhận công việc nạp sẵn vào bộ nhớ cache một khối lệnh và dữ liệu lấy từ bộ nhớ RAM theo thứ tự liên tiếp nhau. Thay vì phải đọc từ bộ nhớ RAM, CPU sẽ đọc lệnh hoặc dữ liệu có sẵn trong bộ nhớ cache với tốc độ cao hơn rất nhiều. Rõ ràng là, nếu bộ nhớ Cache càng lớn, cơ hội lấy lệnh và dữ liệu theo yêu cầu của CPU ở đây càng lớn, thay vì phải truy xuất trực tiếp vào bộ nhớ RAM, hiệu suất làm việc do vậy được nâng lên rất cao.



Hiện tượng CPU lấy được dữ liệu mong muốn từ bộ nhớ cache gọi là *cache hit*, nếu không có ở bộ nhớ cache thì gọi là *cache miss*, CPU phải truy xuất tại bộ nhớ RAM. Trên hình vẽ, có thể coi *L1 memory cache* như là *input cache*, còn *L1 data cache* là *output cache*.

Khối tìm nạp chịu trách nhiệm nạp lệnh vào cache từ bộ nhớ RAM. Đầu tiên, nó tìm trong *L1 instruction cache*, nếu không có, nó sẽ tìm ở *L2 memory cache*, nếu lệnh mà CPU yêu cầu vẫn chưa có ở đây, nó sẽ nạp trực tiếp từ bộ nhớ RAM.

Memory Cache là một kiểu bộ nhớ hiệu suất cao, cũng được gọi là bộ nhớ tĩnh. Kiểu bộ nhớ đã sử dụng trên bộ nhớ RAM chính của máy tính được gọi là bộ nhớ động. Bộ nhớ tĩnh tiêu tốn nhiều năng lượng điện hơn, đắt hơn và có kích thước vật lý lớn hơn so với bộ nhớ động, tuy nhiên nó lại chạy nhanh hơn. Nó có thể làm việc với cùng tốc độ clock của CPU, điều mà bộ nhớ động không thể thực hiện được. Vào "thế giới bên ngoài" để tìm nạp dữ liệu làm cho CPU phải làm việc ở tốc độ clock thấp hơn do vậy mà kỹ thuật cache nhớ được sử dụng ở đây để khắc phục nhược điểm này. Khi CPU nạp dữ liệu từ một vị trí nhớ nào đó thì mạch có tên gọi là memory cache controller (mạch này không được vẽ trong hình) nạp vào cache nhớ một khối dữ liệu bên dưới vị trí hiện hành mà CPU đã nạp. Vì các chương trình được thực hiện theo thứ tự nên vị trí nhớ tiếp theo mà CPU sẽ yêu cầu có thể là vị trí ngay dưới vị trí nhớ mà nó đã nạp. Do memory cache controller đã nạp rất nhiều dữ liệu dưới vị trí nhớ đầu tiên được đọc bởi CPU nên dữ liệu kế tiếp sẽ ở bên trong cache nhớ, chính vì vậy CPU không cần phải thực hiện thao tác lấy dữ liệu bên ngoài: nó đã được nạp vào bên trong cache nhớ nhúng trong CPU, chính vì nhúng trong CPU mà chúng có thể truy cập bằng tốc độ clock trong. Cache controller luôn luôn quan sát các vị trí nhớ đã và đang được nạp dữ liệu từ một vài vị trí nhớ sau khi vị trí nhớ vừa được đọc. Một ví dụ thực tế, nếu một CPU đã nạp dữ liệu được lưu tại địa chỉ 1.000 thì cache controller sẽ nạp dữ liệu từ "*n*" địa chỉ sau địa chỉ 1.000. Số "*n*" được gọi là trang; nếu một bộ vi xử lý này làm việc với 4KB trang (giá trị điển hình) thì nó sẽ nạp dữ liệu từ các địa chỉ 4.096 dưới vị trí nhớ hiện hành đang được nạp (địa chỉ 1.000 trong ví dụ). 1KB bằng 1.024 byte, do đó là 4,096 chứ không phải 4,000. Memory cache càng lớn thì cơ hội cho dữ liệu yêu cầu bởi CPU ở đây càng cao, chính vì vậy CPU sẽ giảm sự truy cập trực tiếp vào bộ nhớ RAM, do đó hiệu suất hệ thống tăng (hãy nên nhớ rằng khi CPU cần truy cập trực tiếp vào bộ nhớ RAM thì nó phải thực hiện ở tốc độ clock thấp hơn nên giảm hiệu suất của toàn hệ thống). Trên trang chi tiết kỹ thuật của một CPU, L1 cache có thể được thể hiện bằng một hình ảnh hoàn toàn khác. Một số nhà máy sản xuất liệt kê hai L1 cache riêng biệt (đôi khi gọi cache chỉ lệnh là "I" và cache dữ liệu là "D"), một số hãng ghi số



lượng của cả hai là 128 KB nhưng điều đó có nghĩa là 64 KB cho cache chỉ lệnh và 64 KB cho cache dữ liệu. Mặc dù vậy đối với các CPU Pentium 4 và Celeronn đời mới dựa trên socket 478 và 775 thì không có hiện tượng này. Các bộ vi xử lý Pentium 4 (và các bộ vi xử lý Celeron sử dụng socket 478 và 775) không có L1 instruction cache mà thay vào đó chúng có một *trace execution cache*, đây là cache được đặt giữa khối giải mã và khối thực thi. Chính vì vậy đây là L1 instruction cache nhưng tên đã được thay đổi và ở một vị trí cũng khác. Lỗi rất thường xảy ra khi nghĩ rằng các bộ vi xử lý Pentium 4 không có L1 instruction cache. Vậy khi so sánh Pentium 4 với các CPU khác mọi người hãy nghĩ rằng L1 cache của nó nhỏ hơn nhiều.

4.4. Trường hợp gặp lệnh rẽ nhánh

Hiện tượng "cache miss" xẩy ra rất thường xuyên, khối tìm nạp phải truy xuất bộ nhớ RAM làm hoạt động của máy tính chậm hơn. Trong trường hợp gặp lệnh rẽ nhánh, ví dụ lệnh JMP ("jump" hoặc "goto"), tuần tự thực hiện lệnh liền sau đó bị phá vỡ, lệnh theo yêu cầu ở vị trí mới trong RAM và chưa có trong *L2 memory cache*, do vậy trong các CPU hiện đại, có khối chức năng *suy đoán trước lệnh rẽ nhánh* sẽ phân tích khối nhớ đã nạp, bất cứ khi nào tìm thấy lệnh JMP, nó sẽ nạp nội dung khối nhớ này vào *L2 memory cache* trước khi CPU xử lý lệnh JMP đó. Vấn đề phát sinh khi lệnh rẽ nhánh là *có điều kiện*. Do điều kiện chưa được xác định chương trình có rẽ nhánh hay không, *cache controller* sẽ nạp cả hai vùng nhớ có địa chỉ lệnh CPU phải nhảy tới vào bộ nhớ cache. Khi xử lý lệnh rẽ nhánh có điều kiện, CPU sẽ chỉ xử lý như đối với lệnh rẽ nhánh bình thường bằng cách loại bỏ trường hợp không được chọn theo điều kiện rẽ nhánh. Dù sao, việc nạp vào bộ nhớ các dữ liệu không cần thiết vẫn tốt hơn việc truy xuất vào bộ nhớ RAM.

4.5. Kỹ thuật đường ống (Pipeline) và xử lý song song mức lệnh

Kỹ thuật đường ống làm tăng đáng kể tốc độ xử lý của CPU. Chu kỳ thực hiện lệnh sẽ được thực hiện ở từng khối chức năng riêng, do vậy khối nhận lệnh sau khi nhận lệnh sẽ tiếp tục nhận lệnh khác để chuẩn bị chuyển cho khối giải mã, khi lệnh thứ nhất đã bước sang giai đoạn 3 của việc thực thi thì khối nhận lệnh nhận lệnh thứ 3 v.v… CPU hiện đại có 11 tầng, có thể nhận 11 lệnh tại cùng một thời điểm. Thực tế, hầu như tất cả các CPU hiện nay đều theo kiến trúc superscalar, số lệnh đồng thời được nạp vào trong CPU sẽ cao hơn nhiều. Với đường ống của CPU có 11 tầng, một lệnh được thực thi hoàn toàn sẽ phải chuyển qua cả 11 khối. Lệnh đầu tien có thể sẽ mất 11 bước để được thực hiện xong, song lệnh thứ hai đi ngay sau đó sẽ

được hoàn thành sớm hơn, chứ không phải là qua đến 11 bước như lệnh đầu…

Pentium là loại đơn vị xử lý trung tâm 32 bit và là loại đơn vị xử lý trung tâm đầu tiên sử dụng kỹ thuật ILP (Instruction Level Pararellism), kỹ thuật xử lý song song mức lệnh.

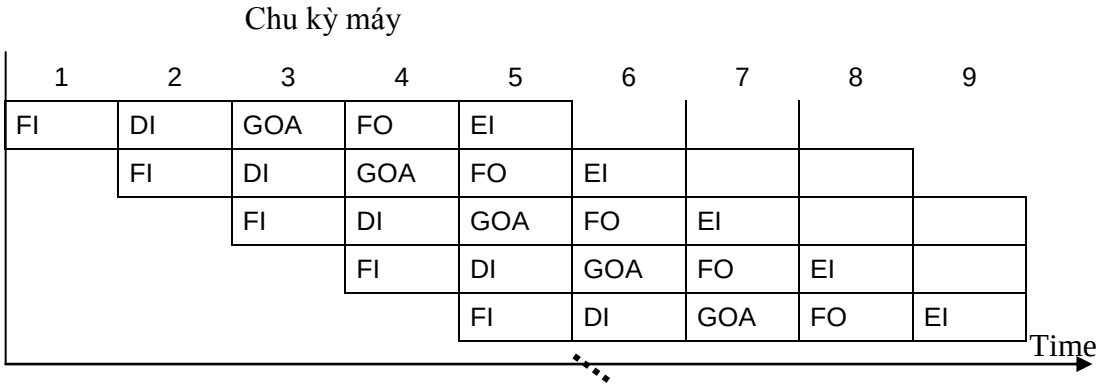
Nếu coi rằng một lệnh thường được xử lý qua năm giai đoạn:

1. Nhập lệnh (FI Fetch the Instruction)
2. Giải mã lệnh (DI Decode the Instruction)
3. Tạo địa chỉ toán hạng (GOA Generate Operand Address)
4. Nhập toán hạng (FO Fetch Operands)
5. Thực hiện lệnh (EI Execute Instruction)

Với kỹ thuật xử lý lệnh thông thường, đơn vị xử lý trung tâm phải tuần tự thực hiện xong tất cả các giai đoạn thực hiện một lệnh, thường là sau 5 chu kỳ máy, rồi mới chuyển sang nhập và thực hiện lệnh tiếp theo. Để tăng tốc độ xử lý lệnh mà không nhất thiết phải tăng tần số nhịp của máy, người ta sử dụng các kỹ thuật khác như kỹ thuật xử lý lệnh kiểu đường ống (Pipeline) và kỹ thuật xử lý lệnh song song (ILP).

- Kỹ thuật xử lý lệnh kiểu đường ống (Pipeline)

Kỹ thuật xử lý lệnh kiểu đường ống được sử dụng trong các đơn vị xử lý trung tâm từ đời đơn vị xử lý trung tâm Intel 8086.

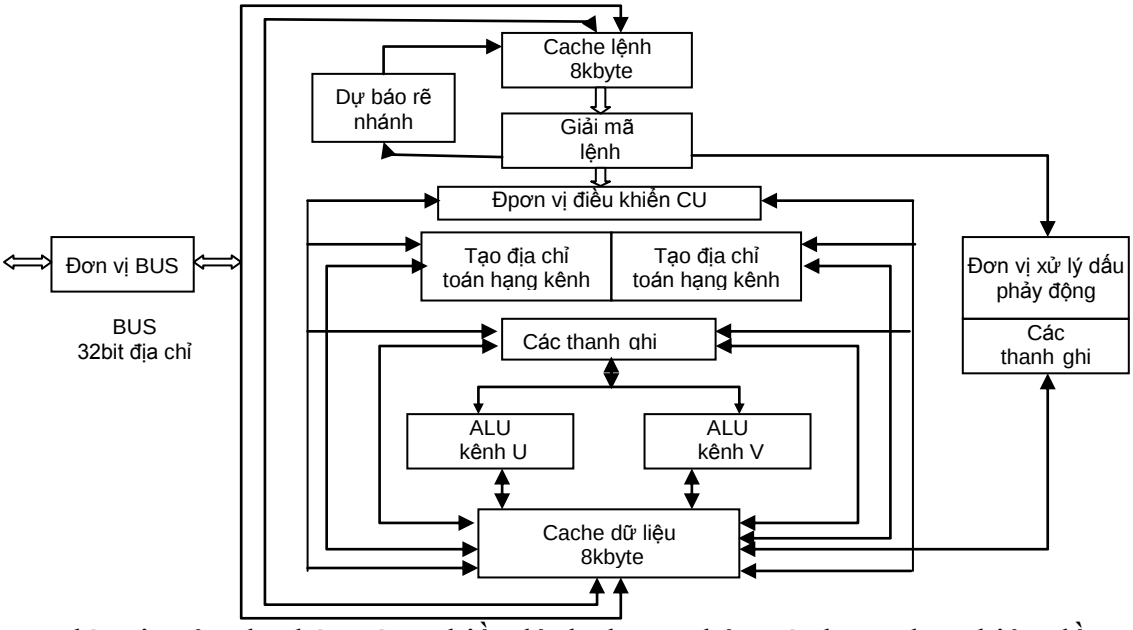


Đường ống tương tự như dây chuyền sản xuất nhiều công đoạn. ở dây chuyền sản xuất, mỗi công đoạn thực hiện một thao tác sản xuất. Sản phẩm được chuyển và hình thành dần sau mỗi công đoạn sản xuất, cho đến khi được hoàn thành ở công đoạn cuối cùng. Trong kỹ thuật xử lý lệnh theo kiểu đường ống, việc thực hiện lệnh cũng được thực hiện qua 5 thao tác, mỗi thao tác được thực hiện trong một giai đoạn, giai đoạn nọ tiếp sau giai đoạn kia,

cho đến khi thực hiện xong lệnh. Với một đường ống 5-giai đoạn, tại mỗi chu kỳ máy có 5 bộ dữ liệu thuộc 5 giai đoạn xử lý được gửi vào đường ống và 5 thao tác được thực hiện đồng thời, nhờ vậy mà sau mỗi chu kỳ máy lại có một lệnh được hoàn thành và một lệnh mới được nhập. Kỹ thuật đường ống với đường ống 5-giai đoạn cho phép tăng tốc độ thực hiện lệnh lên gấp 5 lần.

- Kỹ thuật ILP (xử lý song song mức lệnh)

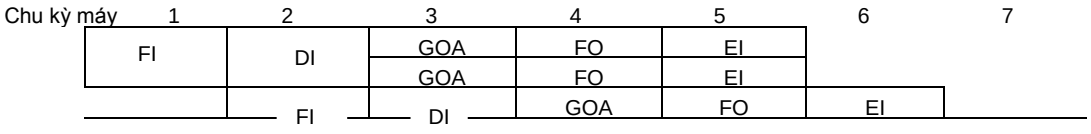
Kỹ thuật ILP là kỹ thuật thiết kế đơn vị xử lý trung tâm và chương trình dịch nhằm làm tăng tốc độ các thao tác máy (như ghi-đọc bộ nhớ) và thực hiện các phép tính. Trong các kỹ thuật ILP có kỹ thuật *superscalar*,



trong đó tại một chu kỳ máy, nhiều lệnh được nhập và được thực hiện đồng thời trên nhiều đường ống khác nhau.

Cấu trúc đơn vị xử lý trung tâm Pentium (CPU)

Pentium là loại đơn vị xử lý trung tâm được thiết kế theo kỹ thuật superscalar, trong đó hai lệnh được nhập và giải mã đồng thời. Pentium có hai đường ống thực hiện lệnh song song U và V. Quá trình thực hiện lệnh được mô tả như sau :



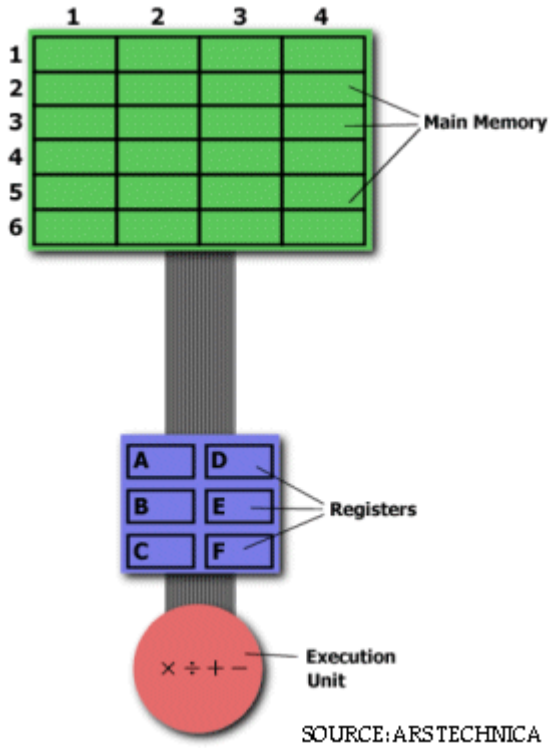
Nguyễn Trung Đồng – Tel 0983 410 866

			GOA	FO	EI	
		FI	DI	GOA	FO	EI
				GOA	FO	EI

4.6. Về kiến trúc RISC, CISC

RISC: (Reduced Instruction-Set Computer)

CISC: (Complete Instruction-Set Computer)



Cách đơn giản nhất để có thể khảo sát những ưu nhược điểm của kiến trúc RISC là so sánh việc thực hiện một phép toán đối với loại kiến trúc CISC trước đây. Giả sử ta phải thực hiện một lệnh nhân hai toán hạng được lưu giữ trong bộ nhớ.

Hình II. … mô tả tổ chức của một máy tính. Bộ nhớ được tạo từ các ô nhớ từ 1:1 (hàng 1: cột 1) đến 6:4 (hàng 6 : cột 4). Khối thực hiện lệnh có nhiệm vụ thực hiện các lệnh tính toán ∗ (nhân), ÷ (chia), + (cộng) và – (trừ). Tất nhiên, khối thục hiện tính toán chỉ có thể làm việc với các dữ liệu (toán hạng) đã được chứa sẵn ở một trong các thanh ghi A,B,C,D,E hoặc F. Giả sử ta phải tìm tích 2 số, số thứ nhất được chứa ở ô 2:3 và số thứ hai ở ô 5:2, kết quả sẽ được lưu lại vào ô 2:3. Bây giờ ta sẽ tiếp cận cách giải quyết vấn đề trên hai loại CPU, CISC và RISC.

Trên CPU CISC: ưu tiên hàng đầu của loại CPU này là hoàn thiện một công việc với ít lệnh nhất có thể. Điều này có thể thực hiện nhờ vào việc xây dựng một phần cứng CPU có khả năng *hiểu được và thực hiện được* một chuỗi các tác nghiệp. Trong trường hợp cụ thể này, CISC sẽ có một lệnh xác định duy nhất, tạm gọi là MULT, mà khi thực hiện, lệnh sẽ nạp hai giá trị toán hạng vào 2 thanh ghi sau đó thực hiện phép nhân rồi ghi kết quả vào một thanh ghi tương ứng. Như vậy công việc sẽ được thể hiện bằng một lệnh như sau:

MULT 2:3,5:2

Lệnh MULT là một lệnh hoàn thiện (complex). Lệnh làm việc trực tiếp trên băng nhớ của máy tính, chứ không cần người lập trình phải dùng lệnh gọi hay nạp nội dung, ghi nội dung vào ô nhớ. Lệnh rất gần với ngôn ngữ bậc cao. Giả sử ta gọi "*a*" là giá trị của toán hạng trong ô nhớ 2:3 và "*b*" là giá trị toán hạng trong ô nhớ 5:2, thì lệnh tương ứng trong ngôn ngữ C là "*a = a*b"*.

Ưu điểm lớn nhất của hệ thống CISC là chương trình dịch phải làm rất ít việc khi dịch một chương trình, hay một lệnh của ngôn ngữ bậc cao sang ngôn ngữ máy. Vì độ dài của mã lệnh rất nhỏ, nên hệ thống cũng cần ít RAM hơn để ghi nhớ lệnh. Dĩ nhiên, việc thiết kế cấu trúc loại CISC đặc biệt sẽ phải tích hợp các lệnh hoàn thiện bằng phần cứng.

Trên CPU RISC: Các CPU loại RISC chỉ sử dụng các lệnh (Instruction) có thể thực hiện được trong một chu kỳ xung nhịp. Như vậy lệnh MULT được mô tả ở phần trên phải được chia thành 3 lệnh nhỏ hơn: "LOAD" chuyển dữ liệu (toán hạng) từ ô nhớ vào thanh ghi; "PROD" thực hiện phép nhân hai toán hạng được lưu giữ trong các thanh ghi, và lệnh "STORE" sẽ thực hiện việc chuyển kết quả tính toán ghi vào ô nhớ. Để thực hiện được phép nhân hai toán hạng, người lập trình phải mã hoá thành 4 lệnh như sau:

LOAD A, 2:3

LOAD B, 5:2

PROD A, B

STORE 2:3, A

Có thể thấy rằng với cấu trúc RISC, không thuận lợi lắm cho hoàn thành phép toán nhân hai số vì phải viết nhiều dòng lệnh hơn, cần nhiều RAM hơn để lưu giữ các lệnh mức assembly. Chương trình dịch cũng phải trhực hiện nhiều việc hơn để chuyển đổi các lệnh của ngôn ngữ bậc cao sang mã máy.

Thế nhưng, chiến lược RISC đã mang đến nhiều thuận lợi quan trọng. Vì mỗi lệnh chỉ cần một chu kỳ xung nhịp để thực hiện, toàn bộ chương trình cũng sẽ chỉ cần số chu kỳ xung nhịp như khi thực hiện lệnh MULT ở hệ thống CISC. Nhưng kiến trúc RISC với hệ lệnh rút gọn cần ít linh kiện và không gian cho mạch tích hợp, bỏ qua được các thanh ghi đa năng. Hơn nữa, mỗi lệnh chỉ thực thi trong một chu kỳ xung nhịp nên việc tổ chức đường ống cũng đơn giản hơn nhiều.

Việc tách lệnh "LOAD" và lệnh "STORE" đã đơn giản hoá đáng kể khối lượng công việc CPU phải thực hiện. Sau khi thực hiện lệnh MULT ở cấu trúc CISC, CPU tự động xoá nội dung các thanh ghi. Nếu một toán hạng

nào đó còn tếp tục được sử dụng cho lệnh tiếp theo, CPU phải nạp lại. Ở cấu trúc RISC, nội dung của toán hạng vẫn được giữ lại cho đến khi một giá trị mới được nạp vào.

Cuối cùng, để so sánh một cách toàn diện hơn, công thức sau được dùng để đánh giá khả năng tính toán, xử lý của các loại CPU:

$$\frac{\text{time}}{\text{program}} = \frac{\text{time}}{\text{cycle}} \text{ x } \frac{\text{cycles}}{\text{instruction}} \text{ x } \frac{\text{instructions}}{\text{program}}$$

CISC cố gắng giảm số lệnh trong một chương trình, hy sinh số chu kỳ thực hiện một lệnh trong khi RISC theo chiến lược ngược lại.

Một số thông tin thú vị cho độc giả: Chỉ các chip họ x86 vẫn trung thành với kiến trúc CISC, dĩ nhiên không được thông dụng lắm vì những lý do khác: Trước hết do sự phát triển vũ bão của công nghệ tích hợp mạch, trong công nghệ sản xuất linh kiện điện tử. Sự giảm giá đến mức khó hiểu của bộ nhớ RAM cũng làm đảo lộn cách nhìn nhận những nhược điểm của các CPU theo kiến trúc RISC. Giá 1Mbyte RAM năm 1977 là khoảng 5000USD, nhưng đến năm 1994 là khoảng 6USD, còn tại thời điểm này (2005) là khoảng hơn 0,2 USD! Công nghệ chương trình dịch (compiler technology) cũng trở nên hoàn thiện hơn nên CPU loại RISC cùng với bộ nhớ RAM dung lượng lớn và công nghệ phần mềm đã trở thành lý tưởng hơn nhiều đối với các hãng sản xuất máy tính.

Chương IV. Chương trình và thực hiện chương trình

1. Tổng quan về lập chương trình cho máy tính

Máy tính không tự mình đưa ra lời giải cho các bài toán. Để máy tính cho các lời giải, chúng ta phải thông báo chính xác cho máy tính biết rằng cần phải giải quyết bài toán như thế nào, cũng như đưa ra các quyết định trên cơ sở nào. Nói ngắn gọn, con người phải ra lệnh chính xác cho máy tính về trình tự và cách thức tính toán và xử lý dữ liệu.

Tập hợp các lệnh được tổ chức theo một trật tự nhất định theo yêu cầu của thuật giải được gọi là chương trình và dùng để thông báo cho máy tính thực hiện và xử lý một vấn đề cụ thể.

Thông thường việc viết một chương trình gồm hai bước cơ bản:

- Phân tích - thiết kế và
- Lập trình (còn gọi là viết mã, coding).

Lập trình là viết các bước đã được xác định trong giai đoạn thiết kế bằng ngôn ngữ đặc biệt mà máy tính có thể hiểu được.

Cần ghi nhớ rằng chương trình phải được viết trên ngôn ngữ có thể dịch được sang *ngôn ngữ máy* (machine language), vì đó chính là ngôn ngữ hoạt động của phần cứng.

Ngôn ngữ máy rất khó sử dụng, ngay cả để giải quyết một vấn đề đơn giản. Vì vậy việc phát triển các *ngôn ngữ lập trình* và *bộ dịch* có tầm quan trọng đặc biệt.

Có nhiều loại ngôn ngữ máy tính, song phổ biến nhất là các ngôn ngữ assembly và ngôn ngữ bậc cao.

Các ngôn ngữ bậc cao chẳng hạn như Pascal, Fortran, C, C++ hoặc VB,...

Các ngôn ngữ assembly được xây dựng cho một lớp máy tính cụ thể, ví dụ cho bộ vi xử lý 80486 hay DEC VAX, v.v. Tập lệnh Assembly của một máy tính cụ thể gồm các lệnh làm việc dựa trên kiến trúc cụ thể của các thanh ghi, phương thức tổ chức và quản lý bộ nhớ của Đơn vị xử lý trung tâm được dùng trong máy tính đó.

Ngược lại, các ngôn ngữ bậc cao được xây dựng để chương trình có thể chạy trên nhiều máy tính khác nhau.

Chương trình dịch ngôn ngữ assembly sang ngôn ngữ máy gọi là ***assembler***, còn chương trình dịch ngôn ngữ bậc cao được gọi là ***compiler***.

Thông thường compiler dịch ngôn ngữ bậc cao sang một dạng ngôn ngữ assembly rồi cuối cùng mới được chuyển sang ngôn ngữ máy.

Có nhiều chương trình máy tính được phát triển nhằm hỗ trợ cho việc lập trình và quản lý máy tính hoạt động một cách hiệu quả hơn. Các chương trình này được gọi là chương trình hệ thống, trong số đó có các assembler, compliler và hệ điều hành.

2. Lệnh và thực thi lệnh

2.1. Kiến trúc của lệnh

Đơn vị xử lý trung tâm của máy tính có khả năng thực hiện được một *tập hữu hạn* các lệnh. *Mỗi lệnh được lưu trữ trong bộ nhớ máy tính dưới dạng một dãy số nhị phân*. Các lệnh thường được phân theo các nhóm chức năng như nhóm lệnh số học-logic, nhóm lệnh điều khiển rẽ nhánh, nhóm lệnh vào-ra dữ liệu, v.v.

Cấu trúc của lệnh thường gồm hai phần:

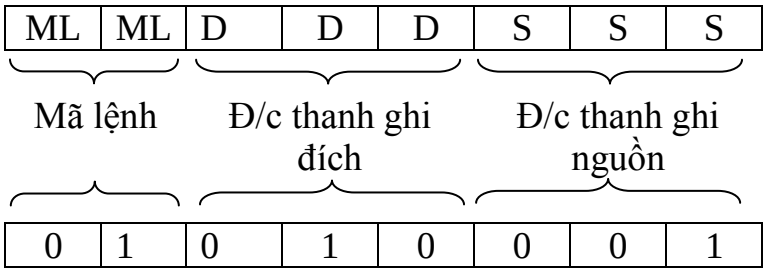
(1) Mã lệnh (tác vụ CPU cần thực hiện: sao chép dữ liệu, phép cộng, phép trừ, v.v.) và

(2) Toán hạng hoặc là Địa chỉ của toán hạng.

Mặc dù mã lệnh và toán hạng là những mã tự nhị phân, song ta thường không dùng các chúng trong để viết chương trình vì thói quen nhận biết các mã tự. Để thuận lợi, lệnh thường được viết dưới dạng *mã ngữ đặc tả gợi nhớ* (Mnemonics) mô tả thao tác cần thực hiện và địa chỉ lưu trữ toán hạng sử dụng trong thao tác. Sau đó phần mã ngữ sẽ được dịch sang dạng mã tự nhị phân.

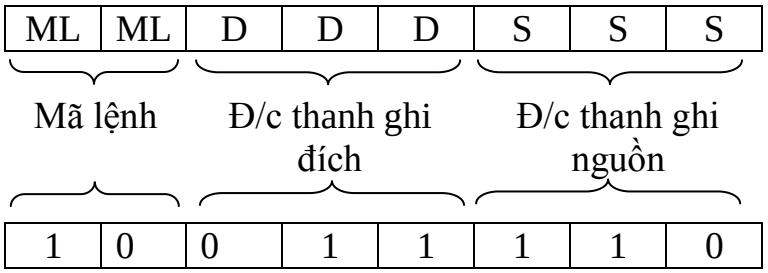
Ví dụ sau đây minh họa một lệnh có độ dài 8 bit, 2 bit mã lệnh và 6 bit địa chỉ. Trong dạng mã ngữ, mã lệnh MOV, có nghĩa là di chuyển (sao chép) dữ liệu. Ở phần thứ hai là địa chỉ, nơi chứa dữ liệu cần sao chép và vị trí sao chép. Ở dưới lệnh được chuyển sang dạng nhị phân.

> **MOV r1,r2** Sao chép dữ liệu là nội dung thanh ghi r2 sang thanh ghi r1, thanh ghi r1 gọi là **đích** (Destination) đến của dữ liệu , thanh ghi r2 là **nguồn** (Source) chứa dữ liệu (trong hình dưới, các bit mã địa chỉ thanh ghi r1 được ký hiệu là D, các bit mã địa chỉ thanh ghi r2 được ký hiệu là S.



Tổ chức các Bit trong ô nhớ lệnh MOV r1,r2

SUB M Lệnh trừ, nội dung thanh ghi gộp (ACC - Accummulator) được trừ đi nội dung của ô nhớ có địa chỉ là nội dung thanh ghi 16 bit (địa chỉ vật lý của ô nhớ chứa nội dung toán hạng trừ).



Tổ chức các Bit trong ô nhớ lệnh SUB

Trong dạng mã ngữ, mã lệnh là SUB, nghĩa là thực hiện phép trừ. Phần thứ hai, 011 là phần địa chỉ của toán hạng đích, 110 là địa chỉ của toán hạng nguồn. Chuyển sang dạng nhị phân, phần mã lệnh SUB dạng mã ngữ tương ứng với 10 dạng nhị phân và phần địa chỉ 011 nhị phân, tức là 3 thập phân là địa chỉ của thanh ghi gộp. 110_B là địa chỉ của thanh ghi 16 bit chứa địa chỉ của ô nhớ toán hạng thứ hai, tương ứng với 6 trong hệ thập phân. Cần nhấn mạnh rằng phần thứ hai của lệnh chỉ cho biết địa chỉ của các toán hạng chứ không phải bản thân toán hạng. Số 3 hay số 6 chỉ nói rằng cần tìm các toán hạng ở địa chỉ nào.

Toán hạng ở địa chỉ 3 được trừ số nào? Nó được trừ đi nội dung đã có sẵn ở thiết bị lưu trữ, trong một ô nhớ, mà địa chỉ ô nhớ được xác định là nội dung thanh ghi 16 bit có địa chỉ là 6.

Như vậy thanh ghi gộp là thiết bị không thể thiếu của CPU và nó được dùng để lưu (tạm thời) kết quả xử lý.

2.1. Tập lệnh cơ bản của máy tính

Các lệnh cơ bản trong máy tính có thể chia ra thành các nhóm nhỏ sau:

1. Nhóm các lệnh di chuyển dữ liệu (Data Movement) cho phép sao chép (Copy) dữ liệu từ nơi này sang nơi khác, từ thanh ghi sang thanh ghi, từ ô nhớ sang thanh ghi hay ngược lại.
2. Nhóm các lệnh thực hiện phép toán:
   - Các phép toán một ngôi: lệnh chỉ làm việc trên một toán hạng, tạo ra một kết quả, thông thường, đó là các phép toán logic
   - Các phép toán hai ngôi: lệnh làm việc với hai toán hạng tạo ra kết quả, ví dụ các phép toán số học, các phép toán AND, OR, XOR v.v…
3. Nhóm các lệnh so sánh và điều khiển rẽ nhánh không điều kiện và có điều kiện: lệnh điều khiển rẽ nhánh chương trình theo tuần tự thực hiện thuật giải, hoặc tuỳ theo kết quả phép toán hoặc tuỳ theo tuần tự chương trình.
4. Nhóm các lệnh gọi chương trình con: chương trình con là một nhóm các lệnh thực hiện một nhiệm vụ nào đó và có thể gọi ra từ một số nơi trong chương trình chính. Khi thực hiện xong, điều khiển phải trở về thực hiện lệnh tiếp sau lệnh gọi chương trình con, thông thường, địa chỉ trở về được cất trong vùng nhớ ngăn xếp (Stack).
5. Nhóm các lệnh vòng lặp: Khi cần thực hiện 1 nhóm lệnh trong chương trình một số lần, có thể sử dụng lệnh vòng lặp. Thông thường số vòng lặp liên quan đến một thanh đếm, số vòng lặp có thể là cố dịnh, hoặc kết thúc khi thoả mãn một điều kiện nào đó.
6. Nhóm các lệnh Vào/Ra (I/O): lệnh dược sử dụng để nhập dữ liệu cho máy tính hoặc hiển thị kết quả xử lý dữ liệu, có các phương pháp Vào/Ra thường sử dụng là:
   - Vào/Ra theo định trình
   - Vào/Ra điều khiển bởi cơ chế ngắt
   - Vào/Ra theo truy cập trực tiếp bộ nhớ

Mỗi kiến trúc máy tính có một tập lệnh cơ bản riêng. Đây là các lệnh mà máy tính có thể hiểu và thực hiện được. Sau đây chúng ta minh họa một số lệnh trong tập lệnh cơ bản của một máy tính đơn giản.

Lệnh LOAD: Lệnh dùng để nạp nội dung của một ô nhớ tại một địa chỉ xác định vào thanh ghi gộp. Ví dụ ở đây minh họa lệnh LOAD xxxx thực hiện thao tác nạp nội dung ở địa chỉ xxxx vào thanh ghi gộp.

Lệnh STORE: Lệnh dùng để nạp nội dung thanh ghi gộp vào một ô nhớ có địa chỉ xác định trong bộ nhớ hay vào một thanh ghi.

Lệnh ADD: Lệnh cộng nội dung thanh ghi gộp với nội dung một ô nhớ, kết quả đặt tại thanh ghi gộp.

Lệnh SUB: Lệnh trừ nội dung của thanh ghi gộp một đại lượng là nội dung của một thanh ghi khác, hay nội dung ô nhớ tại một địa chỉ xác định trong bộ nhớ .

Lệnh AND: Lệnh thực hiện phép VÀ logic hai dữ liệu, một ở thanh ghi gộp, một ở một thanh ghi khác hoặc nội dung một ô nhớ có địa chỉ xác định trong bộ nhớ.

Lệnh OR: Lệnh thực hiện phép HOẶC logic hai dữ liệu, một ở thanh ghi gộp, một ở một thanh ghi khác hoặc nội dung một ô nhớ có địa chỉ xác định trong bộ nhớ.

Lệnh IN: Lệnh lấy dữ liệu từ một thiết bị ngoại vi đưa vào thanh ghi gộp.

Lệnh OUT: Lệnh đưa nội dung thanh ghi gộp ra một thiết bị ngoại vi.

Lệnh JUMP: Lệnh điều khiển CPU thực hiện lệnh được lưu tại một ô nhớ có địa chỉ xác định trong bộ nhớ.

Lệnh CALL: Lệnh gọi một chương trình con từ một vị trí nhất định trong chương trình chính.

Tất cả các lệnh đều được viết dưới dạng mã gợi nhớ (hợp ngữ).

Trong máy tính *lệnh và dữ liệu đều được lưu trữ dưới dạng một dữ liệu nhị phân*. Vì vậy chúng thường được lưu ở các khu vực tách biệt trong bộ nhớ. Máy tính xử lý lệnh trước khi truy xuất dữ liệu từ bộ nhớ.

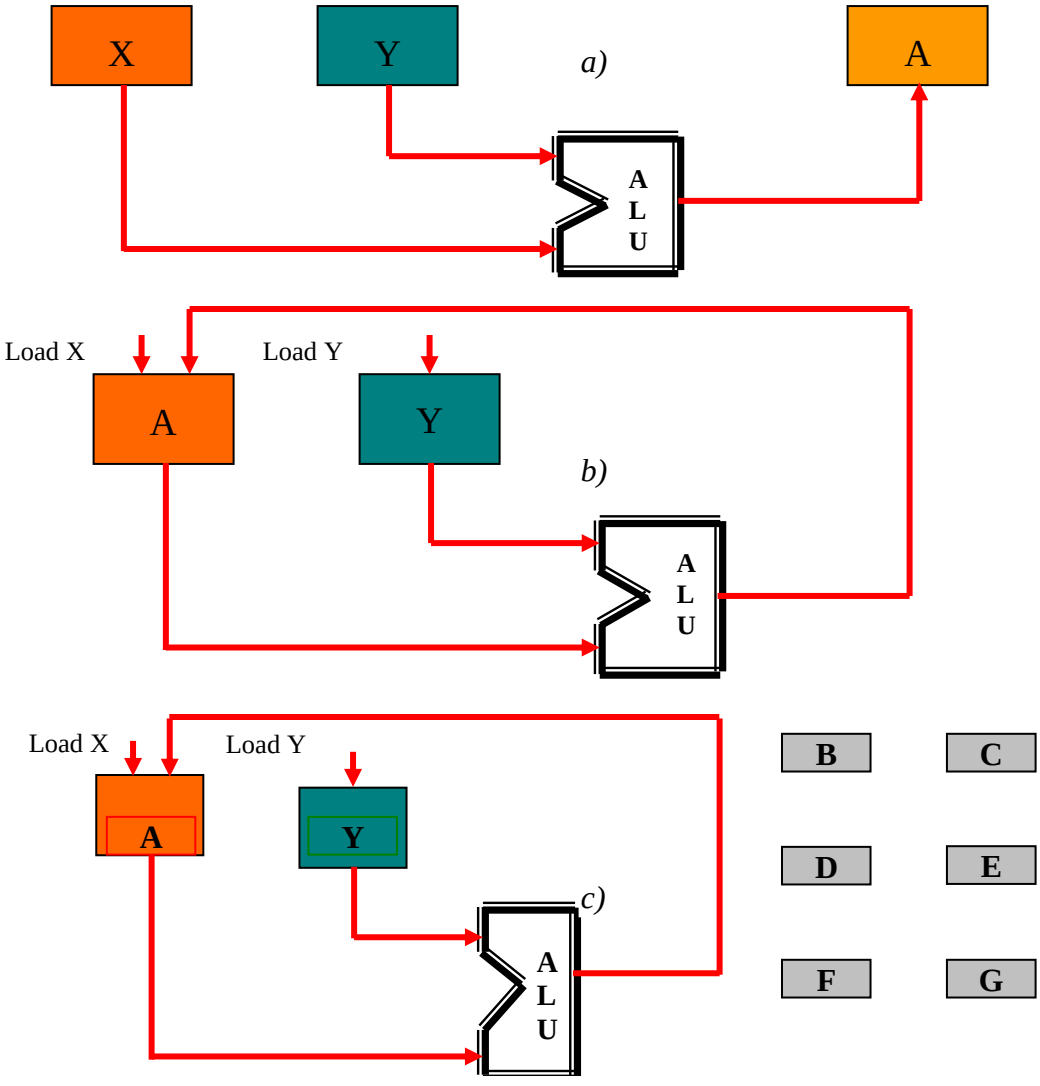
3. Kiến trúc thanh ghi của CPU

Thanh ghi trong CPU giữ vai trò rất quan trọng trong xử lý dữ liệu. Theo quan điểm kiến trúc, thanh ghi trong CPU là những ô nhớ có địa chỉ xác định trực tiếp. Hình vẽ sau cho ta biết những ý đồ phát triển các thanh ghi trong CPU: Xuất phát từ lệnh thực hiện phép cộng hai toán hạng X và Y, kết quả cất ở A.

Thoạt tiên, giả sử ta có thanh ghi X để chứa nội dung toán hạng X, thanh ghi Y chứa nội dung toán hạng Y, ALU thực hiện phép cộng hai nội dung trên và đưa kết quả vào thanh ghi A (*Hình IV.1. a)*.

Trong trường hợp có thể, bằng cách tổ chức hợp lý hơn, ta có thể sử dụng A như là nơi chứa một toán hạng, giả sử nội dung toán hạng X, và sau khi ALU thực hiện phép cộng kết quả lại được chứa vào thanh ghi A, như ở *Hình IV.1.b)*.

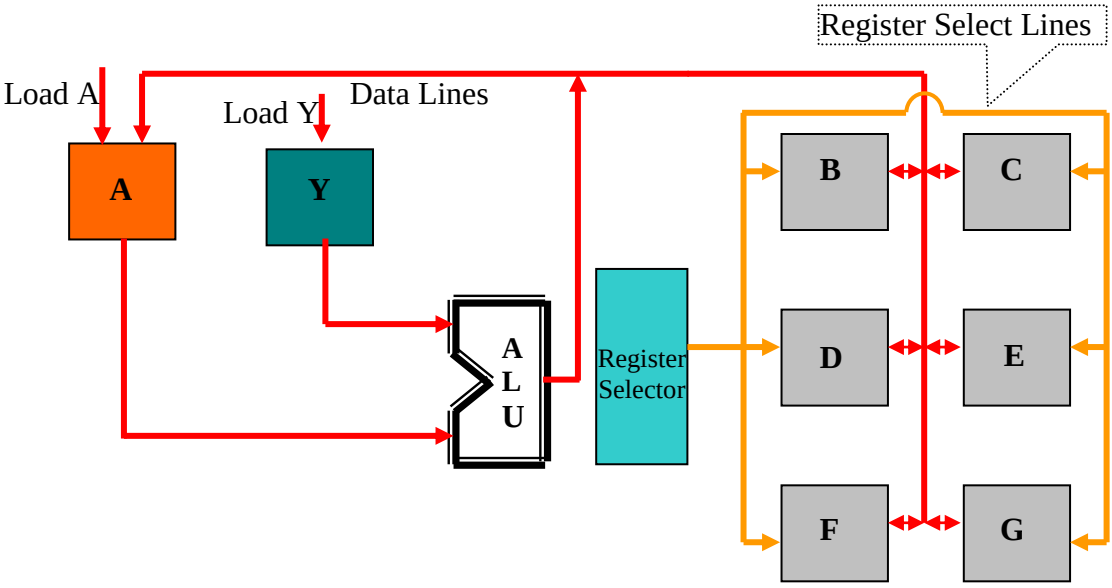
Không những thế, CPU còn cần một só thanh ghi phụ trợ để có thể chứa tạm thời nội dung các toán hạng hay kết quả trung gian, với khả năng truy xuất đơn giản hơn và nhanh hơn, ta càn ghép thêm các thanh ghi B, C, D, E, F, và G, như ở Hình IV.1.c).



Hình IV.1. Minh hoạ phát triển tổ chức các thanh ghi trong CPU

Nguyễn Trung Đồng – Tel 0983 410 866

Việc đưa thêm các thanh ghi này cho phép sử dụng phương thức địa chỉ trực tiếp thanh ghi, cho phép CPU truy xuất rất nhanh, nhanh gấp nhiều lần so với truy xuất bộ nhớ, sử dụng đơn giản nên tạo hiệu suất làm việc của CPU rất cao. Thanh ghi trong CPU có thể được xác định cho một chức năng cụ thể, hoặc có thể sử dụng như những thanh ghi đa năng, theo nhu cầu của người lập trình. Xét minh hoạ phép cộng hai số liệu được thực hiện trong ALU, dễ dàng thấy được kiến trúc CPU cần tổ chức và quản lý các thanh ghi như thế nào, giả sử ta cần thực hiện phép cộng số X với số Y.



Hình trên cho ta thấy một cách trực quan các thanh ghi chuyên dụng và các thanh ghi đa năng trong tổ chức của các CPU họ x86 của hãng Int$_e$l

4.1. Các bước thực thi một chương trình

Chức năng cơ bản của máy tính là thực hiện chương trình. Chương trình được thực thi gồm một số lệnh được lưu trữ trong bộ nhớ. CPU thực thi chương trình bằng cách thực hiện tuần tự các lệnh trong chương trình theo đúng nguyên lý cơ bản của Von Neumann.

Việc thực thi một chương trình được diễn ra theo hai bước:

(1) CPU lần lượt đọc các lệnh từ bộ nhớ

(2) Thực thi từng lệnh của chương trình

4.2. Thực thi lệnh và thực hiện chương trình

Quá trình thực thi chương trình gồm việc thực hiện từng lệnh. Khoảng thời gian từ khi nhận lệnh cho đến khi thực hiện xong một lệnh được gọi là một *chu kỳ lệnh*. Độ dài của các chu kỳ này không nhất thiết như nhau và tùy thuộc vào bản chất của từng lệnh. Một chu kỳ lệnh xảy ra trên nhiều chu kỳ nhịp đồng hồ của CPU.

Chu kỳ lệnh được chia làm hai chu kỳ nhỏ: Đọc lệnh và thực thi lệnh.

Đọc lệnh là một hoạt động của CPU đối với từng lệnh và là việc "nhặt (Fetching)" một lệnh từ một vị trí được xác định trước trong bộ nhớ, địa chỉ của ô nhớ chứa lệnh chính là nội dung thanh đếm chương trình PC. Thực thi lệnh có thể bao gồm nhiều hoạt động và tùy thuộc vào nội dung cụ thể của lệnh.

Việc thực thi chương trình chỉ dừng lại khi máy bị tắt, có một lỗi xảy ra hay một lệnh của chương trình làm ngừng hoạt động của máy tính, v.v.

Như vậy ta thấy rằng bên cạnh đơn vị số học-logic và đơn vị điều khiển, CPU phải lưu trữ tạm thời ở đâu đó các lệnh được đọc vào từ bộ nhớ. Các lệnh này sau đó phải được tách ra thành mã lệnh và địa chỉ của toán hạng, cuối cùng mã lệnh và địa chỉ của toán hạng cũng phải được lưu lại trong quá trình thực thi lệnh.

Mặt khác, sau khi thực hiện xong một lệnh, CPU cũng cần được thông báo rằng lệnh tiếp theo cần phải thực hiện được lưu ở địa chỉ nào trong bộ nhớ. Thông thường lệnh tiếp theo đó được lưu tại địa chỉ ngay sau địa chỉ của lệnh vừa được thực hiện. Điều này phù hợp hoàn toàn với nguyên lý kiến trúc do Von Neumann đề xuất. Tuy nhiên có những lệnh rẽ nhánh (BRANCH) hoặc lệnh nhảy (JUMP) lại chỉ dẫn CPU đến một địa chỉ khác. Do đó cũng cần phải có nơi lưu địa chỉ của lệnh tiếp theo mà CPU cần thực hiện.

Rõ ràng là để thực thi lệnh và chương trình, các máy tính số không thể thiếu được các thành phần sau đây:

- *Thanh ghi lệnh* (Instruction Register – IR): *Thanh ghi lệnh* được sử dụng để nạp các lệnh được lấy về từ bộ nhớ.
- *Con trỏ lệnh* (Instruction Pointer – IP): *Con trỏ lệnh* là thanh ghi chứa dữ liệu cần thiết để tạo nên địa chỉ của lệnh tiếp theo cần thực hiện theo tuần tự chương trình.

Trong kiến trúc của các máy tính trước kia, con trỏ lệnh được gọi là *thanh đếm chương trình* (Program Counter – PC). Thanh đếm chương trình

trước kia chứa phần địa chỉ của lệnh vừa được nạp vào thanh ghi lệnh, do đó nó phải có độ dài (tức là số bit) bằng độ dài của phần địa chỉ trong một lệnh. Tương tự, thanh ghi lệnh thông thường có độ dài bằng độ dài của một từ lệnh trong bộ nhớ. Trong các máy tính ngày nay con trỏ lệnh có thể không chứa đầy đủ phần địa chỉ của lệnh vừa được nạp mà có thể chỉ chứa những thông tin căn bản làm cơ sở để tính toán địa chỉ lệnh tiếp theo.

Bộ đếm chương trình có thể được đặt lại giá trị 0 (nghĩa là thực hiện lệnh đầu tiên của chương trình) hoặc tăng lên một đơn vị (tức là trỏ vào lệnh tiếp theo nằm ở địa chỉ ngay sau đó). Trong trường hợp lệnh rẽ nhánh bộ đếm chương trình sẽ được nạp phần địa chỉ của lệnh.

4.3. Chu kỳ đọc lệnh

Một cách sơ lược, khởi đầu chu kỳ điển hình đọc một lệnh, con trỏ lệnh (IP) nạp thông tin cần thiết để tạo ra địa chỉ của lệnh, đơn vị điều khiển (CU) đưa ra tín hiệu RD đọc dữ liệu từ bộ nhớ ở địa chỉ do con trỏ lệnh tạo ra, dữ liệu tại địa chỉ đó được nạp vào thanh ghi lệnh. Cuối cùng nội dung của con trỏ lệnh tự động cộng thêm một giá trị nhất định để trỏ vào địa chỉ của lệnh tiếp theo trong bộ nhớ.

Quá trình đọc lệnh sẽ diễn ra như sau:

(1) Đơn vị điều khiển điều khiển con trỏ lệnh cung cấp dữ liệu để tạo địa chỉ của ô nhớ chứa lệnh cần thực hiện.

(2) Đơn vị điều khiển (CU) ra tín hiệu RD để nhận dữ liệu ở ô nhớ có địa chỉ do con trỏ lệnh tạo ra.

(3) Nội dung của ô nhớ có địa chỉ tương ứng được nạp vào thanh ghi lệnh.

(4) Nội dung của con trỏ lệnh tăng lên một giá trị tuỳ thuộc độ dài từ lệnh vừa được truy nhập để trỏ sang địa chỉ lệnh tiếp theo.

4.4. Thanh ghi đệm dữ liệu (MBR) và thanh ghi địa chỉ bộ nhớ (MAR)

Để thực thi lệnh, ngoài thanh ghi lệnh và con trỏ lệnh, CPU còn có:

- Thanh ghi đệm dữ liệu (MBR) có chức năng chứa dữ liệu CPU đọc từ bộ nhớ hoặc sẽ ghi ra bộ nhớ.
- Thanh ghi địa chỉ bộ nhớ (MAR) dùng để chứa và xác định địa chỉ của ô nhớ hiện thời CPU đang truy cập.

Một cách chi tiết hơn, quá trình đọc lệnh sẽ diễn ra như sau:

Bước 1. *Đơn vị điều khiển điều khiển con trỏ lệnh cung cấp dữ liệu để tạo địa chỉ của ô nhớ chứa lệnh cần thực hiện.*

Bước 2. *Địa chỉ ô nhớ vừa được tạo ra được nạp vào thanh ghi địa chỉ bộ nhớ (MAR).*

Bước 3. *Đơn vị điều khiển CU ra tín hiệu RD yêu cầu đọc dữ liệu ô nhớ ở địa chỉ lưu trong thanh ghi địa chỉ bộ nhớ MAR.*

Bước 4. *Nội dung của ô nhớ có địa chỉ tương ứng được nạp vào thanh ghi đệm bộ nhớ (MBR).*

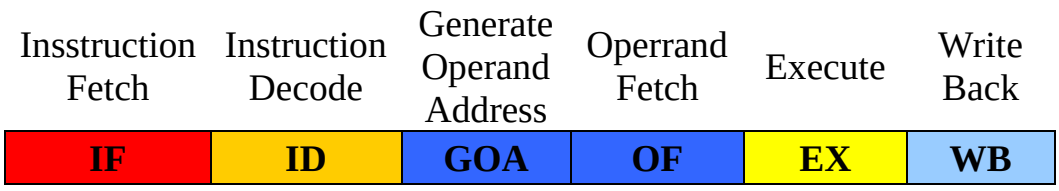
Bước 5. *Nội dung của con trỏ lệnh tăng lên một giá trị tuỳ thuộc vào độ dài từ lệnh vừa đọc để chuẩn bị lấy lệnh theo tuần tự ở địa chỉ tiếp theo trong bộ nhớ chương trình.*

Bước 6. *Nội dung của thanh ghi đệm bộ nhớ (MBR) được ghi vào thanh ghi lệnh (IR).*

4.5. Thực hiện lệnh

Sau chu kỳ đọc lệnh, chu kỳ thực hê lệnh sẽ tiếp theo và gồm các bước sau:

7. Giải mã lệnh
8. Tạo địa chỉ toán hạng



9. Nhập toán hạng
10. Thực hiện xử lý lệnh
11. Lưu kết quả

4.6. Bộ giải mã lệnh (ID)

Để giải mã lệnh, CPU sử dụng *khối giải mã lệnh* (Instruction Decoder – ID).

Khối giải mã lệnh nhận các bit từ thanh ghi lệnh IR và thực hiện các thao tác sau

- Giải mã phần mã lệnh trong lệnh

- Nạp phần địa chỉ của toán hạng trong lệnh vào thanh ghi địa chỉ bộ nhớ (MAR).

Đến đây CPU biết cần thực hiện:

(1) Phép toán số học hay logic nào?

(2) Đọc và nạp toán hạng của phép toán từ ô nhớ có địa chỉ đã nạp trong thanh ghi địa chỉ bộ nhớ (MAR).

4.7. Giải mã lệnh

Quá trình giải mã lệnh như sau:

Bộ giải mã lệnh nhận mã lệnh từ thanh ghi lệnh IR và sử dụng các vi mạch để thực hiện giải mã. Trước hết, phần mã lệnh sẽ cho biết "công việc" mà CPU cần giải quyết bằng cách xác định công việc theo một sơ đồ như ở hình AAAA (trang 24). Sau đó bộ giải mã lệnh gửi tín hiệu về phép toán cần thực hiện cho Đơn vị số học và logic thông qua Đơn vị điều khiển CU.

Ví dụ khi nhận các bit 0001 bộ giải mã lệnh thực hiện các phép so sánh cần thiết và nhận biết đó là thao tác nạp (LOAD) dữ liệu từ bộ nhớ vào thanh ghi gộp ACC. Tín hiệu này sẽ được truyền tới Đơn vị số học và logic. Tương tự, khi nhận các bit 0011, bộ giải mã lệnh sẽ giải mã thành phép cộng (ADD) và gửi tín hiệu yêu cầu thực hiện phép cộng tới Đơn vị số học và logic.

4.8. Nhập toán hạng, xử lý và lưu dữ liệu

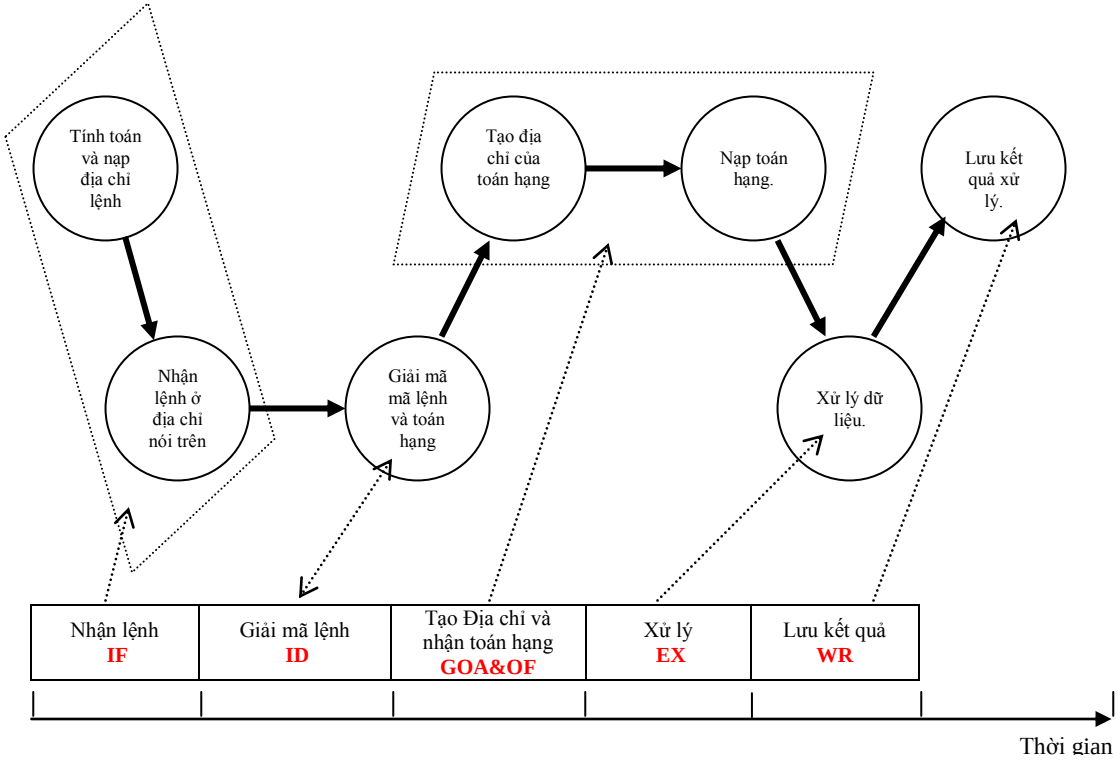
Trong các bước còn lại, đơn vị điều khiển CU gửi tín hiệu đã giải mã của mã lệnh cho Đơn vị số học và logic, đồng thời gửi tín hiệu RD yêu cầu đọc dữ liệu từ bộ nhớ tại địa chỉ đã nạp trong thanh ghi địa chỉ bộ nhớ (MAR).

Dữ liệu được đọc cũng sẽ được gửi về Đơn vị số học và logic, Đơn vị số học và logic thực hiện các thao tác xử lý dữ liệu cần thiết và lưu kết quả trong thanh ghi gộp.

Tóm lại, một chu kỳ lệnh bao gồm các bước sau đây:

(1) *Tính địa chỉ lệnh*: Xác định địa chỉ của lệnh sẽ được thực thi. Thông thường điều này được thực hiện bằng cách cộng 1 vào địa chỉ của lệnh trước. Tuy nhiên, tùy thuộc vào độ dài của lệnh và tổ chức từ trong bộ nhớ thì có thể cộng một hằng số khác. Trong trường hợp lệnh vừa được thực thi là lệnh rẽ nhánh thì địa chỉ lệnh được cho (tuỳ theo các định địa chỉ) bởi phần địa chỉ của lệnh trước đó.

(2) *Đọc lệnh*: Bước này đọc lệnh từ ô nhớ có địa chỉ đã được xác định ở bước 1 vào CPU.



(3) *Giải mã lệnh*: Giải mã phần mã lệnh trong lệnh để xác định loại tác vụ sẽ được thực hiện và xác định địa chỉ các toán hạng sẽ được sử dụng.

(4) *Tạo địa chỉ toán hạng*: Nếu hoạt động thực hiện lệnh liên quan đến toán hạng trong bộ nhớ thì xác định địa chỉ của toán hạng đó.

(5) *Nạp toán hạng*: Đọc toán hạng từ thanh ghi hoặc từ ô nhớ đãđược xác định từ bộ nhớ vào thanh ghi đệm bộ nhớ.

(6) *Xử lý dữ liệu*: Thực hiện hoạt động xử lý dữ liệu được xác định trong lệnh.

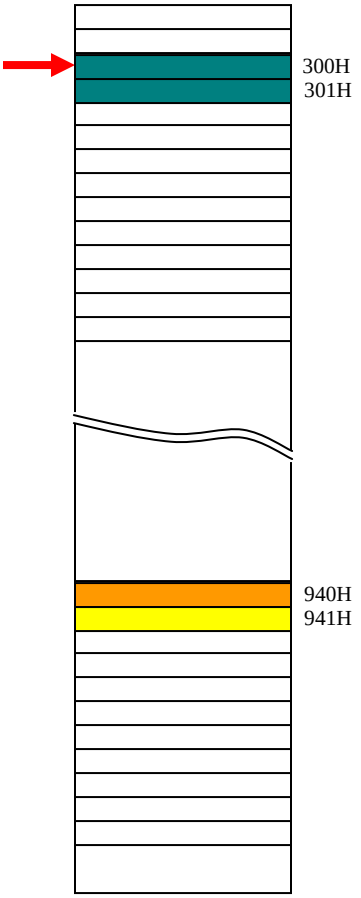
(7) *Lưu kết quả*: Lưu kết quả vào thanh ghi chứa (Acc) hoặc vào 1 vị trí nhớ đã được xác định trong bộ nhớ hay các thiết bị khác.

Sơ đồ sau đây minh họa các trạng thái của một chu kỳ lệnh căn bản:

Nhận lệnh **IF**	Giải mã lệnh **ID**	Tạo Địa chỉ và nhận toán hạng **GOA&OF**	Xử lý **EX**	Lưu kết quả **WR**

Tóm gọn, ta thấy có năm bước cơ bản để CPU thực hiện được một lệnh.

Sau năm bước trên, CPU quay lại bước 1 để thực hiện tác vụ nhận lệnh tiếp theo.



Lưu ý rằng các lệnh của một số máy tính cho phép sử dụng nhiều toán hạng, do đó có thể có nhiều trạng thái *Tạo địa chỉ toán hạng* và *Nạp toán hạng* trước khi xử lý dữ liệu.

Để minh họa, ta hãy khảo sát ví dụ thực hiện đoạn chương trình sau đây, được đặt tại địa chỉ 300H

LOAD @940H; *Lấy nội dung ô nhớ có địa chỉ là 940H nạp vào Acc*

ADD @941H; *Cộng nội dung thanh ghi gộp Acc với nội dung ô nhớ có địa chỉ là 941H, kết quả đặt tại thanh ghi Acc*

Giả sử nội dung ô nhớ có địa chỉ là (940H) là 0302H, nội dung ô nhớ có địa chỉ (941H) là 0022H Giả sử rằng con trỏ lệnh bắt đầu tại vị trí 300H.

Để thực hiện, các lệnh sẽ được đọc tuần tự từ vị trí 300H, 301H, v.v.

Đoạn chương trình này thực hiện việc cộng các nội dung các ô nhớ trong bộ nhớ tại các địa chỉ 940H và 941H.

Tại chu kỳ đọc lệnh thứ nhất, thanh đếm chương trình PC chứa nội dung là 300H, đó là địa chỉ của lệnh đầu tiên. Địa chỉ này được nạp vào thanh ghi địa chỉ bộ nhớ MAR. Nội dung ô nhớ có địa chỉ 300H được đọc và nạp vào thanh ghi đệm bộ nhớ và sau đó được đưa vào thanh ghi lệnh. Nội dung thanh đếm chương trình PC tự động tăng lên 1 để có nội dung là 301H. Thanh ghi gộp có nội dung 0000H.

Trong chu kỳ thực thi lệnh đầu tiên, lệnh được giải mã và xác định rằng cần thực hiện thao tác nhận dữ liệu (LOAD) từ địa chỉ 940H. Thanh ghi địa chỉ bộ nhớ MAR được nạp nội dung 940H. Nội dung 0302H tại địa chỉ này của bộ nhớ được nạp vào thanh ghi đệm MBR và sau đó được nạp vào thanh ghi gộp Acc.

Trong chu kỳ đọc lệnh thứ hai, vì con trỏ lệnh đang có nội dung 301H, nội dung này được nạp vào thanh ghi địa chỉ bộ nhớ MAR. Nội dung của ô nhớ có địa chỉ 301H sau đó được nạp vào thanh ghi đệm bộ nhớ (MBR) và thanh ghi lệnh. Con trỏ lệnh tăng lên 1 để có nội dung 302H. Thanh ghi gộp có nội dung 0302H.

Trong chu kỳ thực thi lệnh thứ hai, lệnh được giải mã và xác định rằng cần thực hiện thao tác cộng dữ liệu (ADD) từ địa chỉ 941H vào nội dung của thanh ghi gộp ACC. Con trỏ lệnh chứa nội dung 302H, thanh ghi địa chỉ bộ nhớ được nạp nội dung 941H. Nội dung 0022H tại địa chỉ 941H của bộ nhớ được nạp vào thanh ghi đệm. Nội dung này được chuyển tới bộ số học và logic để cộng với nội dung của thanh ghi gộp, thành 0342H. Kết quả cuối cùng được lưu ở thanh ghi gộp.

Ghi chú

Trong quá trình thực thi chương trình, CPU tuần tự giải mã các lệnh và thực hiện những thao tác được yêu cầu. Trong phần trên chúng ta đã minh họa quá trình thực hiện lệnh với sự truyền dữ liệu từ bộ nhớ vào thanh ghi lệnh trong CPU. Ngoài bộ nhớ, CPU còn có thể trao đổi dữ liệu với các thiết bị nhập và xuất. Nói chung, những thao tác yêu cầu CPU thực hiện được chia làm bốn dạng

- Trao đổi dữ liệu giữa CPU và bộ nhớ: Dữ liệu được truyền từ CPU đến bộ nhớ hoặc ngược lại.
- Trao đổi dữ liệu giữa CPU và thiết bị nhập/xuất: Dữ liệu có thể truyền ra ngoài hoặc từ ngoài vào bằng cách truyền giữa CPU và các mô-đun nhập/xuất.
- Xử lý dữ liệu: CPU thực hiện các tính toán số học và logic trên dữ liệu.
- Điều khiển: Một lệnh có thể xác định trật tự các lệnh sẽ được thực thi có thay đổi hay không. Ví dụ, CPU có thể đọc lệnh tại địa chỉ 149, lệnh lưu ở địa chỉ này có thể cho biết thêm thông tin là lệnh tiếp theo sẽ được đọc tại địa chỉ 182. CPU ghi nhớ thông tin này bằng cách nạp địa chỉ 182 vào con trỏ lệnh. Nhờ đó chu kỳ đọc tiếp theo sẽ được thực hiện tại địa chỉ 182 chứ không phải tại địa chỉ tiếp theo là 150.

5. Ngắt và cơ chế ngắt (Interupt)

Trong thực tế, tốc độ xử lý dữ liệu của CPU cao hơn rất nhiều so với "sự chế biến dữ liệu" của các thiết bị I/O. Vì vậy cần tạo ra một cơ chế vào/ra hợp lý để tăng hiệu suất làm việc của CPU. Ngắt trong hệ thống máy tính nhằm mục đích giải quyết sự bất hợp lý do CPU phải chờ đợi thiết bị ngoại vi. Thiết bị ngoại vi chỉ yêu cầu CPU phục vụ việc nhận hay chuyển giao dữ liệu khi bản thân nó đã sẵn sàng. Để thực hiện tốt yêu cầu này, cơ chế phục vụ ngắt là hợp lý nhất.

Ngắt nghĩa là yêu cầu CPU tạm thời dừng công việc hiện tại để trao đổi hay xử lý dữ liệu không thuộc tuần tự của chương trình đang thực hiện. Ngắt là một hiện tượng xuất hiện ngẫu nhiên về phương diện thời điểm nhưng được dự đoán trước.

Ngắt là hiện tượng thiết bị có yêu cầu ngắt gửi một tín hiệu báo với CPU rằng một sự kiện đã xảy ra, yêu cầu CPU phải xử lý. Quá trình đang được CPU xử lý sẽ bị tạm thời dừng lại để thực hiện một thao tác khác phục vụ sự kiện có yêu cầu. Khi thao tác này kết thúc, quá trình xử lý vừa bị tạm dừng sẽ được tiếp tục ngay tại nơi nó đã bị tạm dừng. Bản thân sự kiện thông thường là yêu cầu phục vụ của thiết bị ngoại vi đối với CPU.

Trong thực tế, ngắt được sử dụng chủ yếu khi các thiết bị ngoại vi (thường rất chậm so với tốc độ xử lý của CPU) cần trao đổi thông tin, dữ liệu với CPU.

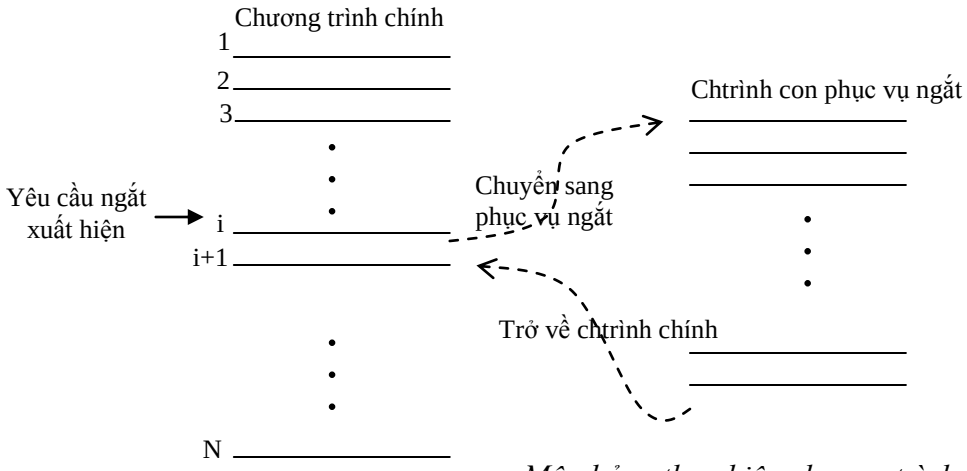
Khi cần trao đổi thông tin, dữ liệu, thiết bị ngoại vi gửi tín hiệu yêu cầu ngắt (Interrupt Request) tới CPU. CPU sẽ thực hiện nốt lệnh hiện tại và trả lời bằng tín hiệu nhận biết yêu cầu ngắt ($\overline{\text{INTA}}$). Chương trình chính lúc này bị tạm dừng (ngắt) và CPU chuyển sang thực hiện chương trình con phục vụ ngắt, tức là chương trình con trao đổi thông tin, dữ liệu với thiết bị ngoại vi yêu cầu ngắt. Sau khi xong công việc phục vụ ngắt, CPU quay về thực hiện tiếp chương trình chính kể từ lệnh tiếp theo sau khi bị ngắt.

Các tín hiệu yêu cầu phục vụ ngắt từ một thiết bị ngoại vi bất kỳ được gửi tới chân nhận yêu cầu ngắt của CPU thông qua một khối điều khiển ngắt. Tuỳ theo người lập trình mà yêu cầu ngắt đó có được chuyển tới CPU hay không. Trong trường hợp yêu cầu ngắt được gửi tới CPU, xử lý của CPU gồm các bước sau:

1. Thực hiện nốt lệnh đang được xử lý

2. Phát tín hiệu nhận biết yêu cầu ngắt gửi cho thiết bị yêu cầu phục vụ ngắt qua chân $\overline{\text{INTA}}$
3. Cất các cờ trạng thái hiện tại vào ngăn xếp
4. Xoá các cờ IF (Interrupt Flag) và cờ TF (Trap Flag)
5. Cất địa chỉ lệnh tiếp theo trong tuần tự chương trình đang thực hiện vào vùng nhớ ngăn xếp
6. Lấy địa chỉ của chương trình con phục vụ ngắt trong bảng vector ngắt theo số hiệu ngắt do thiết bị điều khiển ngắt gửi tới
7. Thực hiện chương trình con phục vụ ngắt.

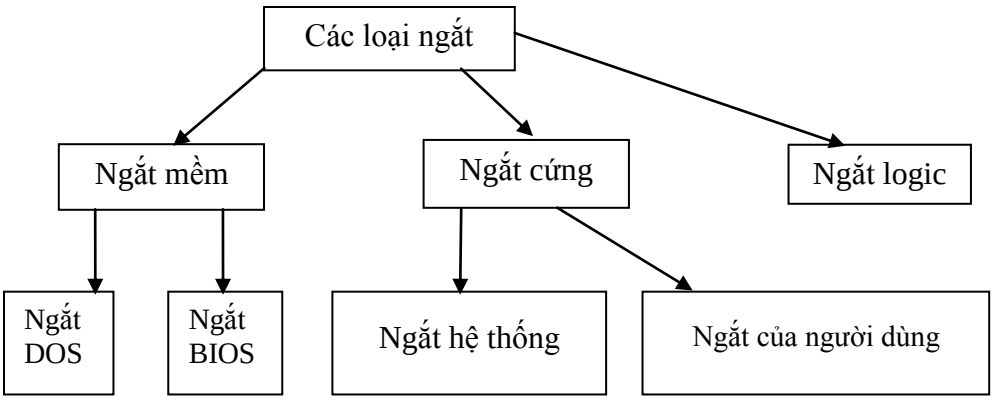
Ngắt là sự kiện CPU bị tạm dừng việc thực hiện quá trình chính và chuyển sang thực hiện quá trình phục vụ ngắt. Ngắt cứng được sử dụng trong phương pháp vào/ra dữ liệu, trong đó thiết bị vào/ra (thiết bị vật lý: bàn phím, máy in, đồng hồ nhịp thời gian v.v.) chủ động khởi động quá trình vào/ra. Quá trình phục vụ ngắt cứng được kích thích bằng một tín hiệu vật lý từ bên ngoài.



Mô phỏng thực hiện chương trình khi gặp ngắt

5.1. Phân loại ngắt

Thuật ngữ "ngắt" xuất phát từ kỹ thuật ngắt cứng. Khi nói đến ngắt cứng, ngắt mềm hoặc ngắt logic (ngoại lệ) là *hàm ý nói đến* các chương trình con phục vụ hoạt động của hệ thống máy tính và nói đến cách kích hoạt các chương trình con này. Tất cả các chương trình phục vụ ngắt đều có chung đặc điểm:



Thứ nhất là hầu hết các chương trình con phục vụ ngắt đã được viết sẵn (là các chương trình của hệ điều hành) và được phép sử dụng; thứ hai là địa chỉ của các chương trình con này phải được đặt ở một vùng xác định là *Bảng véc tơ ngắt*, nằm trong bộ nhớ chính. Các chương trình con phục vụ ngắt cứng thường được dùng để điều khiển quá trình vào/ra với các thiết bị vào-ra chuẩn ở mức vật lý. Các chương trình con phục vụ ngắt cứng được kích hoạt bởi các *tín hiệu vật lý IRQ* (Interrupt Request) đến từ thiết bị vào-ra. Các chương trình con phục vụ ngắt mềm là các chương trình hệ thống thực hiện các thao tác vào-ra cơ bản ở mức logic và các hoạt động khác của hệ thống. Các chương trình con phục vụ ngắt mềm được kích hoạt bởi *lệnh INT* trong tập lệnh của CPU. Các chương trình con phục vụ ngắt logic cũng phục vụ cho hoạt động của hệ thống, nhưng chúng chỉ được kích hoạt khi CPU thực hiện lệnh và do *phát sinh một ngoại lệ* nào đó.

5.2. Bảng véc tơ ngắt

Bảng véc tơ ngắt là bảng chứa địa chỉ của các chương trình con phục vụ ngắt, do chương trình Hệ điều hành quy định. Bảng này có 256 ô, các ô được đánh số thứ tự lần lượt từ 00H, 01H, .. , 08H, .. , 0FH, 10H, .. , FFH. Số thứ tự của từng ô trong bảng được gọi là *số hiệu ngắt,* và được gán cho các yêu cầu ngắt đã được dự đoán trước. Chương trình con phục vụ các yêu cầu ngắt này được nạp sẵn trong bộ nhớ tại những địa chỉ xác định. Mỗi ô chứa địa chỉ logic của một chương trình con phục vụ ngắt xác định, các địa chỉ này còn được gọi là *véc tơ ngắt*. Có thể tham khảo bảng vector của một số ngắt sau

Nội dung của ô nhớ	Số TT.	Số ngắt	Chức năng của chương trình
Địa chỉ đoạn Địa chỉ offset	00H	00H	Xử lý chia cho 0
Địa chỉ đoạn Địa chỉ offset	01H	01H	Thực hiện gỡ rối (debug)
Địa chỉ đoạn Địa chỉ offset	08H	08H	Đồng hồ hệ thống
Địa chỉ đoạn Địa chỉ offset	09H	09H	Phục vụ bàn phím
Địa chỉ đoạn Địa chỉ offset	21H	21H	Thực hiện các dịch vụ của hệ điều hành
Địa chỉ đoạn Địa chỉ offset	FFH	FFH	Dự phòng

5.2. Cơ chế gọi chương trình con

Một trong những chức năng chính của máy tính (của đơn vị xử lý trung tâm CPU) là thực hiện chương trình, trong đó có việc thực hiện chương trình con. Chương trình con là một modul mã lệnh (modul chương trình) độc lập có thể được gọi (kích hoạt) từ chương trình chính và việc này có thể được thực hiện nhiều lần, xuất phát từ các vị trí khác nhau trong chương trình chính.

Việc thực hiện chương trình con xảy ra khi CPU thực hiện lệnh CALL (lệnh gọi chương trình con). Để thực hiện chương trình con thì bộ đếm chương trình (con trỏ lệnh) PC phải trỏ đến (phải chứa địa chỉ) ô nhớ đầu tiên của chương trình con đó. Sau khi thực hiện xong chương trình con, CPU cần trở về chương trình chính tại nơi vừa gọi chương trình con này. Việc này được thực hiện khi CPU gặp lệnh RET. Vùng nhớ ngăn xếp là nơi được sử dụng để chứa địa chỉ trở về chương trình chính khi CPU gọi chương trình con.

Đơn vị xử lý trung tâm thực hiện chương trình con khi gặp lệnh CALL (gọi chương trình con):

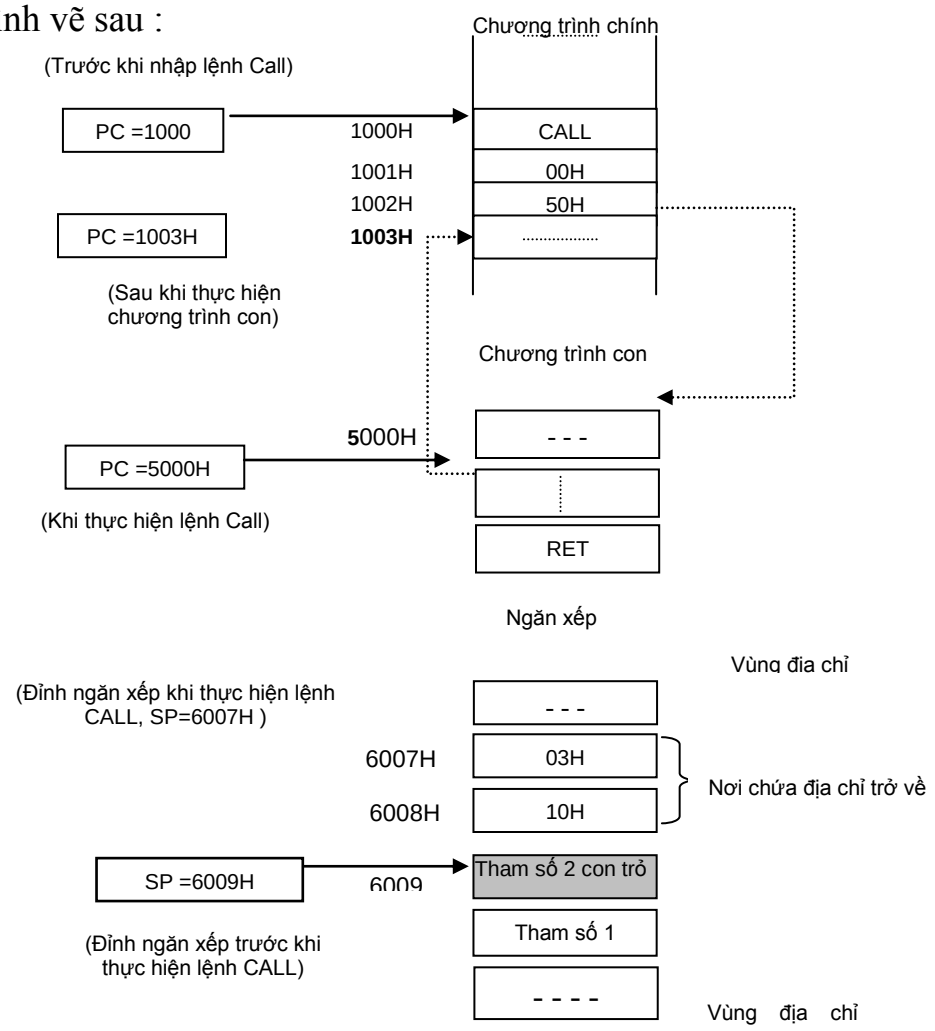
CALL xxxx

trong đó xxxx là địa chỉ của chương trình con.

Khi thực hiện lệnh CALL, CPU thực hiện hai thao tác cơ bản :

– Bảo vệ địa chỉ của câu lệnh tiếp theo ngay sau lệnh CALL bằng cách cất vào ngăn xếp. Địa chỉ này được gọi là *địa chỉ trở về* chương trình chính. Thanh ghi PC 16 bit được cấu thành từ hai phần 8 bit PCH và PCL. Giá trị của địa chỉ trở về thực chất là nội dung của con trỏ lệnh PC khi nó đang trỏ đến câu lệnh nằm sau lệnh CALL.
– Nạp địa chỉ của chương trình con vào bộ đếm chương trình (con trỏ lệnh) PC và thực hiện chương trình con từ vị trí PC trỏ đến.

Cơ chế thực hiện chương trình con được thể hiện qua ký pháp và hình vẽ sau :



Nguyễn Trung Đồng – Tel 0983 410 866

(SP - 1) ⇦ PCH

(SP - 2) ⇦ PCL

SP ⇦ SP-2

PC ⇦ Địa chỉ chương trình con

Ví dụ: CALL 5000H

trong đó 5000H là địa chỉ đầu của chương trình con.

Giả sử trong chương trình chính mã lệnh CALL 5000H nằm ở các ô 1000H, 1001H, 1002H.

Địa chỉ của lệnh tiếp theo (địa chỉ trở về) là 1003H.

Lệnh RET

Khi thực hiện lệnh RET thì CPU đọc nội dung ô đỉnh ngăn xếp (ô chứa địa chỉ trở về chương trình chính) và nạp vào PC, nội dung của SP tự động tăng thêm 2. Sau khi thực hiện lệnh RET thì PC trỏ trở lại vào ô nhớ của lệnh tiếp theo trong chương trình chính, chương trình chính tiếp tục được thực hiện.

RET

PCL ⇦ (SP)

PCH ⇦ (SP + 1)

SP ⇦ SP + 2

6. Lệnh hai địa chỉ

Ở phần trên chúng ta đã xét các lệnh trong đó chỉ có một thành phần là địa chỉ. Các lệnh này được gọi là các *lệnh một địa chỉ*. Các máy tính hiện nay có thể hiểu và thực hiện được các *lệnh hai địa chỉ*.

Lệnh hai địa chỉ gồm ba phần:

1. Mã lệnh
2. Địa chỉ thứ 1
3. Địa chỉ thứ 2

trong đó mã lệnh là phép tính hay thao tác cần thực hiện, địa chỉ thứ nhất là địa chỉ chứa toán hạng, còn địa chỉ thứ hai là địa chỉ lưu kết quả xử lý.

Trong máy tính, ngoài thanh ghi gộp thường có nhiều thanh ghi, được gọi là các thanh ghi đa năng. Các thanh ghi này cũng được sử dụng để lưu dữ liệu phục vụ cho quá trình tính toán và xử lý.

Các thanh ghi đa năng được gán tên để phân biệt. Số bit của các thanh ghi trong các bộ vi xử lý thường là 16 hoặc 32.

6.1. Các chế độ thực hiện lệnh hai địa chỉ

Các lệnh hai địa chỉ có thể được thực hiện trong các chế độ sau:

Trao đổi dữ liệu giữa các ô nhớ trong bộ nhớ (Bộ nhớ - bộ nhớ)

Ví dụ 1: Máy tính thực hiện thao tác: Cộng nội dung ô nhớ có địa chỉ 963H vào nội dung ô nhớ có địa chỉ 492H

- Trao đổi dữ liệu giữa thanh ghi đa năng và bộ nhớ (Thanh ghi đa năng - bộ nhớ)
- Trao đổi dữ liệu giữa bộ nhớ và các thanh ghi đa năng (Bộ nhớ - thanh ghi đa năng)
- Cuối cùng là trao đổi dữ liệu giữa thanh ghi đa năng và thanh ghi đa năng

6.2. Kiến trúc RISC và CISC

Có thể phân kiến trúc máy tính ra hai lớp cơ bản

- Máy tính với tập hợp lệnh tối thiểu (RISC – Reduced Instruction Set Computer) và
- Máy tính với tập hợp lệnh phức hợp (CISC – Complex Instruction Set Computer)

Các máy tính trước kia thường có một số ít các lệnh đơn giản vì công nghệ ống chân không và bán dẫn lúc bấy giờ quá đắt, kềnh càng và phát nhiều nhiệt, chưa cho phép thiết kế các lệnh phức hợp.

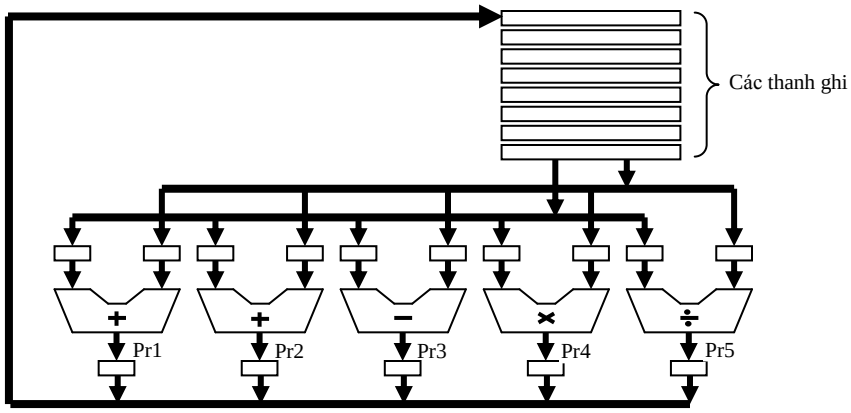
Khi công nghệ đạt được những tiến bộ nhất định, máy tính với kiến trúc tập hợp lệnh phức hợp (CISC) được thiết kế để có thêm nhiều lệnh và các lệnh càng ngày càng phức tạp. Chúng được thiết kế để hỗ trợ hệ điều hành và các trình biên dịch. Một lệnh thậm chí có thể thực hiện cả một chương trình con.

Ý tưởng cơ bản để thiết kế các máy tính này là nếu lệnh càng phức hợp thì tập hợp các lệnh cơ bản càng ít và như vậy ít lệnh sẽ được đọc từ bộ nhớ hơn và do đó làm tăng tốc độ xử lý của CPU. Tuy nhiên các lệnh phức hợp lại đòi hỏi thêm linh kiện điện tử, làm cho bộ vi xử lý kềnh càng hơn và cuối cùng lại làm chậm tốc độ của CPU. Trong khi đó nhiều lệnh phức hợp lại hầu như không được sử dụng trong thực tế.

Vì thế lớp các máy tính có kiến trúc với tập hợp lệnh tối thiểu RISC xuất hiện với tư tưởng: chứa tập hợp một số ít các lệnh được chọn lọc kỹ lưỡng và các lệnh được thể hiện vật lý một cách đơn giản nhất có thể. Chương trình có thể sử dụng nhiều lệnh, nhưng hiệu suất tổng thể của máy tính tăng lên.

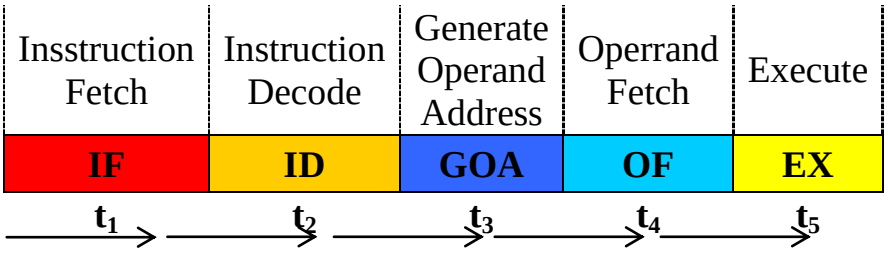
6.3. Kiến trúc xử lý song song

Ở một chừng mực nào đó, có thể tổ chức việc thực hiện lệnh nhờ vào kiến trúc có nhiều đơn vị số học và Logic trong một CPU. Mỗi ALU có thể thực hiện một thao tác đơn với tốc đọ cao. Ví dụ, ALU thực hiện phép cộng, ALU thực hiện phép trừ, ALU thực hiện phép nhân và ALU thực hiện phép chia. Trong trường hợp này, CU sau khi giải mã lệnh sẽ điều khiển một trong các ALU thực hiện lệnh, đồng thời CU nhận về lệnh tiếp theo để chuyển cho các ALU khác thực hiện. Cách thức này sẽ đạt hiệu quả cao khi thời gian thực hiện lệnh dài hơn nhiều so với thao tác lấy lệnh từ bộ nhớ. Trong hình vẽ, giả sử ta có 5 ALU, hai ALU thực hiện phép cộng, các ALU còn lại thực hiện các phép trừ, nhân và chia.

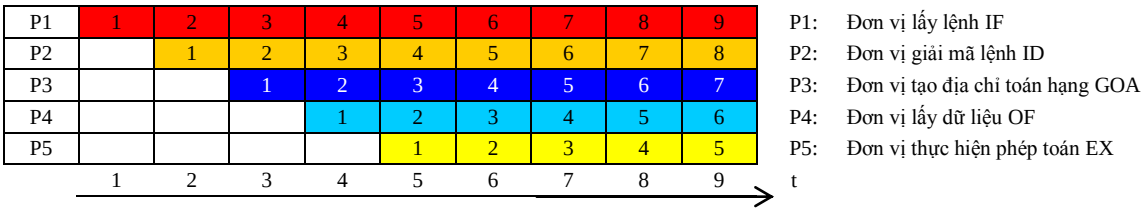


Giả sử một CPU có 5 đơn vị xử lý thực hiện các thao tác như hình vẽ dưới, ký hiệu là ĐVXL1 đến ĐVXL5. Khởi đầu, ĐVXL1 lấy về lệnh thứ nhất từ bộ nhớ và công việc chiếm một khoảng thời gian là t_1. Trong khoảng thời gian t_2 llệnh đó được chuyển cho ĐVXL2 để giải mã, trong lúc ĐVXL1

lấy về lệnh kế tiếp. trong mỗi khoảng thời gian tiếp theo, ĐVXL1 sẽ lấy về một lệnh mới, các lệnh đã nhận sẽ chuyển cho các đơn vị xử lý kế tiếp. Kiểu kiến trúc này được gọi là Kiến trúc đường ống (Pipeline Machine). Thấy rằng, để thực hiện 1 lệnh cần một thời gian là $T = t_1 + t_2 + t_3 + t_4 + t_5$. Tuy nhiên, vớ cách kiến trúc này, trong cùng thời gian đó, CPU thực hiện được 5 lệnh, và có thể coi là tốc độ xử lý đã tăng nhanh gấp 5 lần.



Có thể mô tả phương thức hoạt động đường ống trong hình vẽ sau:



7. Các phương pháp đánh địa chỉ ô nhớ

7.1. Quản lý bộ nhớ

Bộ nhớ máy tính được quản lý theo phương thức phân đoạn (Segmentation). Bộ nhớ vật lý được quản lý và giám sát theo đoạn. Chương trình trong bộ nhớ không được ghi liên tục như là một chuỗi mã lệnh và dữ liệu, mà được chia thành từng module riêng biệt: module mã lệnh và module dữ liệu, module ngăn xếp v.v… Mỗi module được gọi là một đoạn (segment) và có một địa chỉ đầu đoạn xác định. Trong máy tính 16bit, mỗi đoạn có kích thước là 64Kbytes. Có 4 loại đoạn khác nhau:

– Đoạn mã lệnh (Code segment) chứa mã lệnh của chương trình;

– Đoạn dữ liệu (Data Segment) chứa dữ liệu của chương trình;

– Đoạn ngăn xếp (Stack Segment) chứa các thông tin và dữ liệu phục vụ việc thực hiện các chương trình con, phục vụ ngắt;

– Đoạn mở rộng (Extra Segment) chứa các dữ liệu mở rộng, xâu, chuỗi.

Mỗi đoạn kể trên đều có một địa chỉ xác định gọi là địa chỉ đoạn, do hệ điều hành phân phát. CPU của các máy tính có các thanh ghi chuyên dụng lưu giữ các địa chỉ này để quản lý các tác vụ truy xuất và được gọi là thanh ghi đoạn.

B_{15} ... 0	
CS	Thanh ghi đoạn mã (Code Segment Register)
DS	Thanh ghi đoạn dữ liệu (Data Segment Register)
SS	Thanh ghi đoạn ngăn xép (Stack Segment Register)
ES	Thanh ghi đoạn mở rộng (Extra Segment Register)

Thanh ghi đoạn chứa địa chỉ nền của đoạn, nếu ta dịch trái tất cả các bit của thanh ghi này và điền thêm 4 bit 0 phía phải của nội dung đó, thì có được địa chỉ của ô nhớ đầu tiên của đoạn. Về mặt toán học, nó đồng nghĩa với công thức tính địa chỉ đầu đoạn như sau:

Địa chỉ vật lý đầu đoạn = (Nội dung thanh ghi đoạn) x 10H

Mỗi ô nhớ trong đoạn được định vị bằng một cặp giá trị: Địa chỉ đoạn và Địa chỉ offset (địa chỉ lệch). Địa chỉ offset xác định số thứ tự của ô nhớ đó kể từ địa chỉ nền của đoạn. Cách biểu diễn cặp giá trị này dạng ***(Địa chỉ đoạn):(Địa chỉ offset)*** được gọi là ***địa chỉ logic*** của ô nhớ.

Thanh ghi offset có hai loại: thang chuyên dụng giữ giá trị offset và các thanh ghi đa năng khác. Có thể coi thanh ghi offset như là "con trỏ" chỉ tới một vị trí nhớ trong đoạn tương ứng. Các thanh ghi chuyên dụng giữ địa chỉ offset được mô tả trong hình vẽ sau gồm:

Các thanh ghi con trỏ và chỉ số đều là những thanh ghi 16bit, tạo khả năng đánh địa chỉ một vùng nhớ gồm 2^{16} = 65536 ô nhớ (64Kbytes). Điều này lý giải tại sao mỗi đoạn được chỉ ra bởi thanh ghi đoạn, bắt đầu từ địa chỉ đầu đoạn, lại có độ lớn là 64Kbytes. Nguyên lý dùng thanh ghi đoạn và thanh ghi offset để xác định một vị trí nhớ cho thấy rằng: *Nội dung thanh ghi đoạn là 16 bit cao, còn nội dung thanh ghi offset là 16 bit thấp của các bit trong BUS địa chỉ*

B

B_{15} … 0	
IP	Thanh ghi con trỏ lệnh (Instruction Pointer)
SP	Thanh ghi con trỏ ngăn xếp (Stack Pointer)
BP	Thanh ghi con trỏ cơ sở (Base Pointer)
SI	Thanh ghi chỉ số nguồn (Source Index)
DI	Thanh ghi chỉ số đích (Destination Index)

Dễ dàng thấy rằng, với một BUS địa chỉ có 20 bit địa chỉ, thì để tính được địa chỉ vật lý của ô nhớ được xác định bởi nội dung thanh ghi đoạn và nội dung thanh ghi offset, ta thực hiện phép dịch trái nội dung thanh ghi đoạn 4 bit rồi tiến hành cộng với nội dung thanh ghi offset như được biểu diễn trong hình vẽ dưới đây, và phép tính được biểu diễn theo biểu thức sau:

Địa chỉ vật lý = (Nội dung thanh ghi đoạn) x 10H + (Nội dung thanh ghi offset)

Mỗi thanh ghi con trỏ và chỉ số đều được gán một chức năng cụ thể, và cùng với một thanh ghi đoạn tương ứng, tạo thành địa chỉ logic của những đoạn nhớ theo chức năng trong bộ nhớ.

Thanh ghi chỉ số nguồn SI và thanh ghi chỉ số đích DI là những thanh

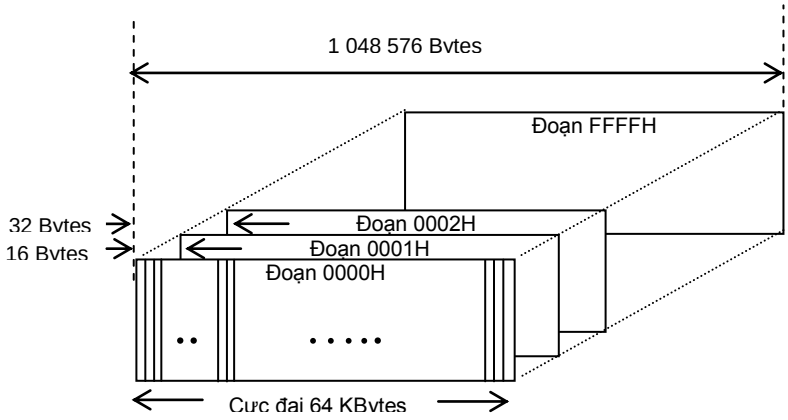
B_{15} … B_0	
CS	Cặp thanh ghi đoạn mã lệnh CS và con trỏ lệnh IP chứa địa chỉ logic CS:IP, làm chức năng của thanh đếm chương trình
IP	
SS	Cặp thanh ghi đoạn ngăn xếp SS và con trỏ ngăn xếp chứa địa chỉ logic SS:SP, làm chức năng con trỏ đỉnh ngăn xếp
SP	

ghi được sử dụng trong các thao tác xử lý xâu, chuỗi .

7.2. Quản lý bộ nhớ, các mode địa chỉ trong CPU 8086

a. Phương thức quản lý bộ nhớ trong CPU8086:

BUS địa chỉ của µP8086 có độ dài 20 bits, do vậy có thể quản lý được 2^{20} = 1M ô nhớ (Mỗi tổ hợp "0" hoặc "1" của các bit trong 20 bits địa chỉ xác định vị trí của một ô nhớ). Vì một ô nhớ trong hệ Vi xử lý là 1 Byte, nên nói cách khác, *không gian nhớ* mà µP8086 quản lý được là 1Mbyte.



Hình II.20 Cách chia đoạn nhớ trong µP8086

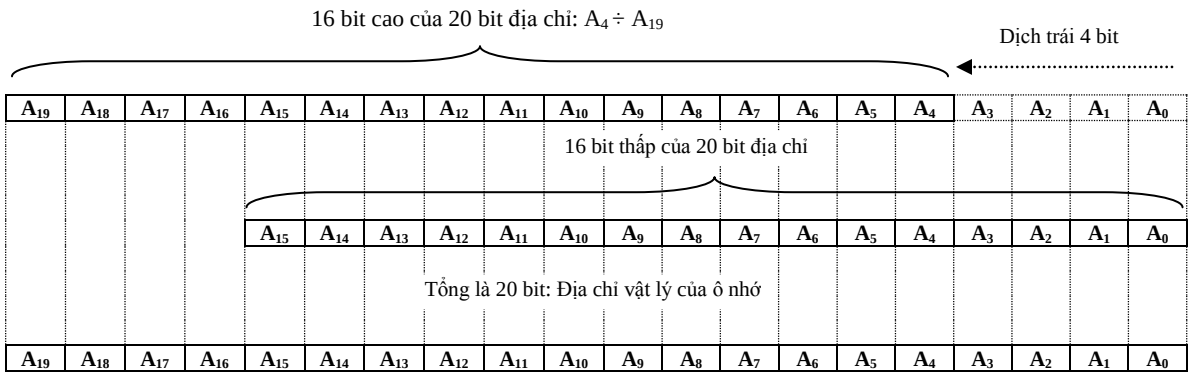
Nguyễn Trung Đồng – Tel 0983 410 866

Các thanh ghi của µP8086 chỉ có độ dài 16 bits, nên nếu dùng một thanh ghi để đánh địa chỉ thì chỉ quản lý được 2^{16} ô nhớ, tức là 64KB. Để giải quyết vấn đề quản lý 1MByte, tức là 1.048.576 Bytes, µP8086 sử dụng BUS địa chỉ có độ rộng 20 bits thông qua nội dung của hai thanh ghi 16 bits để đánh địa chỉ của bộ nhớ theo phương thức sau:

Bằng cách lập chương trình, không gian địa chỉ được chia thành các *đoạn* (segment) nhớ với kích thước cố định là 64Kbytes gọi là một *đơn vị logic* của bộ nhớ. Mỗi đoạn gồm các ô nhớ liên tiếp, độc lập và được định vị tách rời nhau. Mỗi đoạn được người lập trình gán cho một *địa chỉ đoạn*, là địa chỉ ô nhớ đầu tiên của đoạn đó, còn được gọi là *địa chỉ nền*.Giá trị của các địa chỉ đoạn liền kề cách nhau tối thiểu là 16 Bytes. Các đoạn có thể kế cận, tách rời, phủ lấp nhau. Bên trong đoạn sẽ sử dụng các *giá trị lệch* (offset), tức là khoảng cách từ địa chỉ đoạn đến ô nhớ nằm trong đoạn. Một cặp giá trị địa chỉ đoạn và giá trị lệch, [*segment*]:[*offset*], được gọi là *địa chỉ logic*. Địa chỉ logic cho phép định vị chính xác một Byte nhớ trong không gian địa chỉ. Địa chỉ đoạn được chứa trong các thanh ghi đoạn, giá trị dịch chuyển được chứa trong các thanh ghi đa năng, con trỏ hoặc chỉ số.

Về bản chất, thanh ghi đoạn chứa 16 bits cao của 20 bits dịa chỉ, giá trị dịch chuyển là 16 bit thấp, và sự lệch nhau 4 bits đã được đơn vị địa chỉ của BIU giải quyết như trình bày trong hình II. 18: Dịch trái thanh ghi đoạn 4 bits (tương đương phép nhân với 16, cộng với giá trị dịch chuyển offset trong thanh ghi đa năng để tính địa chỉ vật lý của ô nhớ. Như đã trình bày ở trên, công thức tương ứng phép "dịch trái và cộng" có thể trình bày như trên hình vẽ sau, việc tính toán này do đơn vị giao diện BUS đảm nhận (BIU – *BUS Interface Unit*)

Địa chỉ vật lý = 10H x (***segment***) + (***offset***)



Nguyễn Trung Đồng – Tel 0983 410 866

Thanh ghi đoạn là một thanh ghi 16 bits, có nhiệm vụ xác định đoạn của ô nhớ, còn thanh ghi đa năng cũng là một thanh ghi 16 bits. Vậy thanh ghi đoạn có thể định được 2^{16} = 65.536 đơn vị (64K) đoạn nhớ và mỗi đoạn có 64Kbytes. Vậy Vi xử lý µP8086 có thể định địa chỉ tới 64K x 64Kbytes = 4Gbytes nhớ.

Thanh ghi đoạn mã CS xác định đoạn nhớ chương trình mà lệnh kế tiếp sẽ được lấy để thực hiện, thanh ghi con trỏ IP chứa địa chỉ offset của lệnh kế tiếp. Cặp CS:IP tạo nên địa chỉ logic của lệnh kế tiếp trong tuần tự thực hiện chương trình. Các từ lệnh của họ 80x86 có thể có độ dài từ 1 byte đến tối đa là 15 bytes. Khi lệnh được thực hiện, giá trị của con trỏ IP do vậy sẽ tăng lên đúng bằng số Bytes của từ lệnh. Cần nhớ rằng nội dung của thanh ghi *con trỏ lệnh* IP cùng với nội dung thanh ghi đoạn CS xác định địa chỉ của ô nhớ lệnh tiếp theo trong tuần tự thực hiện chương trình.

b. Mode địa chỉ trực tiếp

1. Mode định vị thanh ghi (register addressing): Toán hạng được truy xuất là nội dung thanh ghi của CPU.

Thí dụ MOV AX,BX ; chuyển nội dung của toán hạng nguồn (nội dung của thanh ghi) BX vào toán hạng đích AX. Nội dung thanh ghi BX vẫn được giữ nguyên.

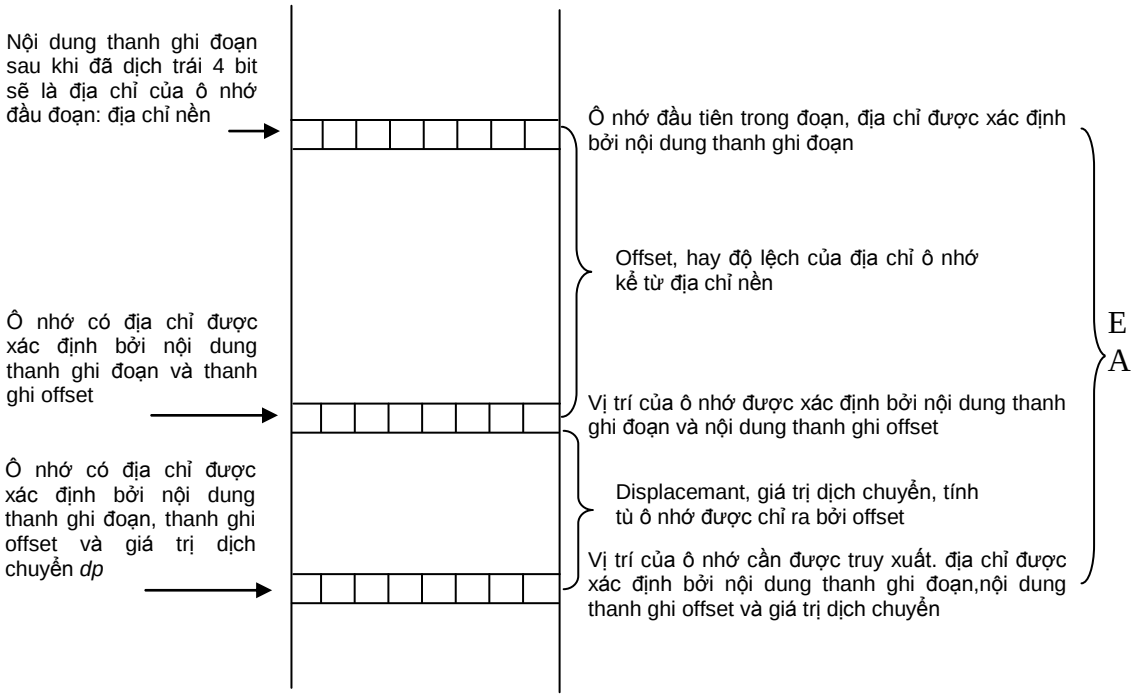
2. Mode định vị tức thời (immediate addressing): Toán hạng tức thời là dữ liệu 8 hay 16 bits nằm ngay trong lệnh, có thể dùng làm toán hạng nguồn hay hằng số. Toán hạng tức thời được lưu giữ ngay trong đoạn mã của bộ nhớ, ngay sau mã lệnh, nó được lấy ra cùng với lệnh và ghi vào hàng đợi lệnh PQ, do vậy được truy xuất nhanh hơn so với truy xuất toán hạng từ bộ nhớ.

Thí dụ MOV AL, 12H ; nạp số 12H vào thanh ghi AL

c. Các mode định vị gián tiếp

Khác với hai kiểu định vị trên, toán hạng trong đoạn nhớ dữ liệu được CPU *truy xuất qua BUS dữ liệu*. Biết rằng, địa chỉ vật lý của ô nhớ được tính từ nội dung thanh ghi đoạn và offset theo cách trình bày trong Hình II. 18. Giá trị offset mà đơn vị thực hiện lệnh EU tính cho một toán hạng trong đoạn nhớ được gọi là *địa chỉ hiệu dụng* EA (effective address) của toán hạng. Đơn vị thực hiện lệnh có thể tính EA dựa vào cách mô tả địa chỉ trong phần toán hạng nguồn của lệnh. Ngoài giá trị trực tiếp, hoặc nội dung thanh ghi cơ sở

hay thanh ghi chỉ số, khi cần còn có thể có một giá trị số có độ dài 8 bits hay 16 bits được cộng thêm vào gọi là *giá trị dịch chuyển dp* (displacement).



Hình II. 21 Mô tả cách xác định địa chỉ vật lý của ô nhớ càn truy xuất

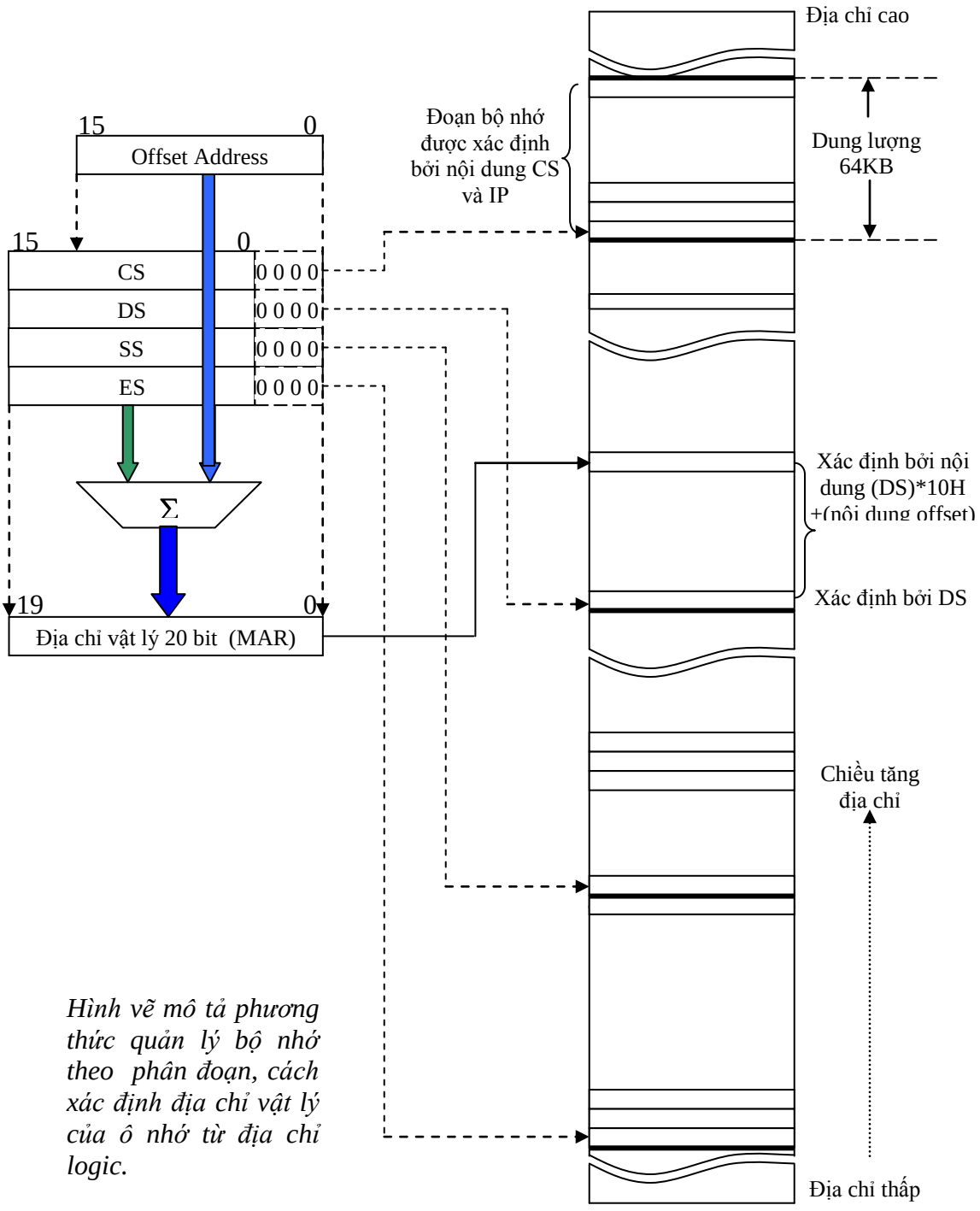
Cụ thể như sau:

– *Định vị trực tiếp* (direct addressir7;): Toán hạn¡0 chứa địa chỉ là một số nằm ngay trong lệnh. Địa chỉ đoạn hiện tại nằm trong thanh ghi đoạn DS

Thí dụ MOV CX,[1435H] ;
chuyển nội dung ô nhớ có địa chỉ offset bằng 1435H trong đoạn số liệu hiện tại vào thanh ghi CX

– *Định vị gián tiếp thanh ghi* (register indirect): địa chỉ hiệu dụng EA là nội dung của một trong các thanh ghi BX, BP, SI hoặc DI

Thí dụ MOV AX, [SI] ;
chuyển nội dung của ô nhớ trong đoạn số liệu hiện tại có địa chỉ offset là nội dung thanh ghi SI



Hình vẽ mô tả phương thức quản lý bộ nhớ theo phân đoạn, cách xác định địa chỉ vật lý của ô nhớ từ địa chỉ logic.

– *Định vị cơ sở* (based addressing): EA là tổng của nội dung thanh ghi BX hoặc BP và giá trị dịch chuyển *dp* nếu có

Thí dụ MOV [BX] + dp, AL ; chuyển nội dung thanh ghi AL và ô nhớ có địa chỉ offset bằng tổng của nội dung thanh ghi BX và giá trị dịch chuyển dp*Định vị*

chỉ số (indexed addressing): EA là tổng của nội dung thanh ghi SI hoặc DI và giá trị dịch chuyển *dp* nếu có

Thí dụ MOV AL,[SI] + dp ; chuyển nội dung ô nhớ có địa chỉ offset bằng tổng của nội dung thanh ghi SI và giá trị dịch chuyển dp vào thanh ghi AL

– *Định vị chỉ số và cơ sở* (indexed addressing): EA là tổng của nội dung các thanh ghi cơ sở, thanh ghi chỉ số và giá trị dịch chuyển *dp* nếu có

Thí dụ MOV AH,[BX][SI] + dp ; chuyển nội dung ô nhớ có địa chỉ offset bằng tổng của nội dung thanh ghi BX, thanh ghi SI và giá trị dịch chuyển dp vào thanh ghi AH

– *Định vị chuỗi* (string addressing): dùng riêng cho xử lý chuỗi. CPU sẽ tự động sử dụng các thanh ghi chỉ số nguồn SI và thanh ghi chỉ số đích DI để chỉ đến các byte kế tiếp

Thí dụ MOVS ; di chuyển chuỗi, nguồn tại vùng nhớ có địa chỉ đầu là DS : SI, đích là vùng nhớ có địa chỉ đầu DS : DI.

Phương pháp đánh địa chỉ là cách thức kiến trúc máy tính chỉ ra phương thức xác định địa chỉ của một đối tượng mà các thành phần khác muốn truy cập.

7.3. Biểu diễn lệnh và dữ liệu

Máy tính hiện nay thường có

- Bộ nhớ được chia thành các từ 8 bit, mỗi từ ứng với một địa chỉ trong bộ nhớ.
- Các lệnh có độ dài (số bit) thay đổi nhưng là bội số của 8.
- Các từ chứa dữ liệu cũng có độ dài (số bit) thay đổi và cũng là bội số của 8.

Trong kiến trúc của nhiều máy tính các từ chứa dữ liệu có độ dài 16, 32, thậm chí tới 64 bit.

7.4. Yêu cầu đối với các phương pháp đánh địa chỉ trong lệnh

Để mô tả địa chỉ trong một lệnh, phương pháp hiển nhiên và đơn giản nhất là cho địa chỉ trực tiếp dạng một số liệu nhị phân. Phương pháp này được gọi là phương pháp *đánh địa chỉ trực tiếp*. Ngoài ra người ta còn sử dụng nhiều phương pháp đánh địa chỉ khác nhau. Các phương pháp đánh địa chỉ phải thỏa mãn các yêu cầu sau đây:

- Rút ngắn phần địa chỉ trong lệnh.
- Thuận tiện cho người lập trình.
- Hỗ trợ hệ thống.

Ví dụ như nếu máy tính nạp đồng thời nhiều chương trình vào bộ nhớ và thường xuyên chuyển đổi việc thực thi các chương trình đó thì để nạp và giải phóng bộ nhớ một cách hiệu quả từ nhiều vị trí khác nhau, phương pháp đánh địa chỉ có thể cho phép chương trình nạp và thực thi trên nhiều vùng khác nhau của bộ nhớ.

Các phương pháp đánh địa chỉ

Chúng ta xét một số phương pháp đánh địa chỉ thông dụng sau đây:

- Phương pháp đánh địa chỉ trực tiếp
- Phương pháp đánh địa chỉ tức thời
- Phương pháp đánh địa chỉ tương đối
- Phương pháp đánh địa chỉ gián tiếp

7.5. Phương pháp đánh địa chỉ trực tiếp

Phương pháp đánh địa chỉ trực tiếp nạp địa chỉ trong lệnh tại ô nhớ ngay sau ô nhớ lưu mã lệnh (tức là ô nhớ có địa chỉ lớn hơn 1).

Ta xét ví dụ sau đây: Giả sử bộ nhớ được tổ chức thành từng ô 8 bit. Mã lệnh chiếm 8 bit, tức trọn vẹn một ô nhớ. Các bit địa chỉ của lệnh chiếm các ô nhớ tiếp theo. Nếu CPU có thể sử dụng 2^{16} từ trong bộ nhớ thì sẽ cần 2 byte để đánh địa chỉ các ô nhớ. Như vậy để nạp một lệnh cần sử dụng 3 byte, tức 3 ô nhớ.

Nếu mã lệnh 3AH (lệnh LOAD), được nạp trong 8 bit ở ô nhớ 0245H, địa chỉ trong lệnh sẽ được nạp trong 16 bit ở hai ô nhớ tiếp theo ngay sau đó với các địa chỉ tương ứng là 0246H và 0247H.

Khi thực hiện lệnh CPU đọc mã lệnh trong ô nhớ có địa chỉ 0245H . Sau đó nó sẽ đọc các byte tiếp theo từ ô nhớ 0246H và 0247H, ráp các nội

dung đọc được thành một lệnh nguyên vẹn (các bit có thứ tự thấp chứa ở byte đầu tiên) và thực thi lệnh này, tức là nạp nôi dung của ô nhớ có địa chỉ là nội dung hai ô nhớ 0246H và 0247H vào thanh ghi gộp.

7.6. Phương pháp đánh địa chỉ tức thời

Phương pháp đánh địa chỉ tức thời thực chất là nạp nội dung của toán hạng (chứ không phải địa chỉ ô nhớ chứa toán hạng) vào ngay sau các ô nhớ chứa mã lệnh.

Trở lại ví dụ trên, nếu mã lệnh 3AH (LOAD), được nạp trong 8 bit ở ô nhớ 0245H, toán hạng của lệnh LOAD sẽ được nạp trong các bit ở ô nhớ tiếp theo ngay sau đó với địa chỉ tương ứng là 0246H hoặc có thể ở cả các ô nhớ tiếp theo.

Khi thực hiện lệnh CPU đọc mã lệnh trong ô nhớ có địa chỉ 0245H và biết lệnh cần thực hiện là nạp nội dung vào thanh ghi gộp. Sau đó nó sẽ đọc các byte tiếp theo từ ô nhớ 0246H hoặc có thể ở cả các ô nhớ tiếp theo và cuối cùng nạp nội dung đó vào thanh ghi.

Để làm ví dụ minh họa, ta xét lệnh như trên (ADD0000010100010000). Giả sử mã lệnh ADD (00000011) được lưu trong ô nhớ có địa chỉ 300H. Với cách đánh địa chỉ trực tiếp, hai ô nhớ tiếp theo 301H và 302H sẽ lưu các bit 00000101 và 00010000. Các dãy bit này được ghép lại thành địa chỉ 0510H và đây là địa chỉ của ô nhớ chứa toán hạng. Còn với cách đánh địa chỉ tức thời thì 0000010100010000 và đây chính là toán hạng

7.7. Phương pháp đánh địa chỉ tương đối

Phương pháp đánh địa chỉ tương đối không cho địa chỉ của toán hạng trong lệnh mà cho gia số của địa chỉ ô nhớ chứa toán hạng và địa chỉ của ô nhớ chứa mã lệnh. Với phương pháp này ta có thể giảm bớt số bít cần sử dụng để cho địa chỉ trong lệnh.

Ví dụ nếu bộ nhớ quản lý được 2^{16} ô nhớ (từ), chúng ta phải cần 2 byte để đánh địa chỉ các ô nhớ. Khi đó với cách đánh địa chỉ trực tiếp, ngoài 1 byte để cho mã lệnh, ta cần thêm 2 byte trong lệnh để cho địa chỉ của toán hạng.

Còn nếu sử dụng phương pháp đánh địa chỉ tương đối, gia số của địa chỉ chỉ cần 8 bit để biểu diễn.

Để hiểu rõ nhất cách đánh địa chỉ này ta so sánh với cách đánh địa chỉ trực tiếp. Với cách đánh địa chỉ trực tiếp, các bit tiếp theo trong lệnh là địa chỉ của toán hạng.

Còn với cách đánh địa chỉ tương đối, chúng là hiệu số của địa chỉ của toán hạng và địa chỉ của mã lệnh.

Ví dụ như với các máy tính 6800, sử dụng cách đánh địa chỉ tương đối lệnh sẽ gồm mã lệnh và gia số của địa chỉ gồm 8 bit (mã lệnh sẽ cho biết sử dụng cách đánh địa chỉ nào). Như vậy để nạp hết một lệnh chỉ cần 2 byte. Địa chỉ trong byte thứ hai của lệnh sẽ được cộng với địa chỉ của ô nhớ chứa mã lệnh và kết quả sẽ là địa chỉ của toán hạng. Địa chỉ trong byte thứ hai của lệnh được hiểu như là số bù 2 có dấu, do đó nó có thể trỏ tới một địa chỉ cao hơn hoặc thấp hơn so với địa chỉ chứa mã lệnh.

Một vấn đề quan trọng cần phải xác định đối với phương pháp đánh địa chỉ tương đối là độ lớn tối đa của gia số. Việc lựa chọn độ lớn này có tác động trực tiếp tới độ dài của lệnh cũng như vùng nhớ có thể truy cập được.

7.8. Phương pháp đánh địa chỉ gián tiếp

Một cách đánh địa chỉ khác cũng rất thông dụng là *đánh địa chỉ gián tiếp*. Với cách đánh địa chỉ này, phần địa chỉ trong lệnh không phải là địa chỉ của toán hạng mà là địa chỉ trỏ tới địa chỉ của toán hạng.

Ví dụ trong ô nhớ địa chỉ 245 (hexa) ta có lệnh ADD 302H. Phần thứ hai của lệnh, địa chỉ 302H trỏ tới ô nhớ có địa chỉ 495 (hexa). Đây mới là cho địa chỉ của toán hạng. Giả sử ô nhớ này chứa dữ liệu 194H. Kết quả là CPU sẽ cộng 194H với nội dung của thanh ghi gộp và lưu kết quả nhận được ở thanh ghi gộp.

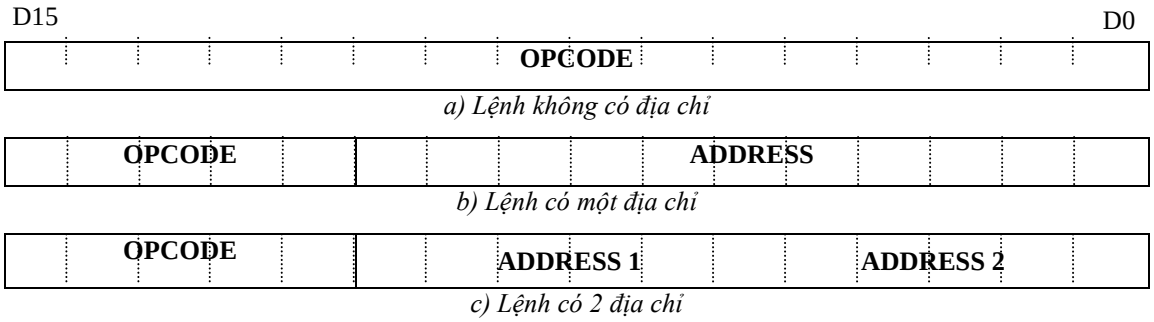
Ngoài các phương pháp đánh địa chỉ nói trên nhiều kiến trúc máy tính còn sử dụng các cách đánh địa chỉ khác. Các phương pháp đánh địa chỉ tức thời và đánh địa chỉ tương đối thường được sử dụng phổ biến nhất.

7.9. Mã hóa các phương pháp đánh địa chỉ

Một kiến trúc máy tính có thể sử dụng đồng thời nhiều phương pháp đánh địa chỉ khác nhau. Vì vậy lệnh phải chưa thông tin cho biết nó sử dụng phương pháp đánh địa chỉ gì. Việc mã hóa cách đánh địa chỉ trong lệnh phụ thuộc vào hai yếu tố: Số phương pháp đánh địa chỉ được sử dụng và mức độ độc lập giữa các mã lệnh và phương pháp đánh địa chỉ.

- Nếu sử dụng ít phương pháp đánh địa chỉ hoặc số tổ hợp mã lệnh/phương pháp đánh địa chỉ ít, cách đánh địa chỉ thường được mã hóa vào trong mã lệnh.
- Nếu số lượng các tổ hợp nhiều, người ta thường sử dụng các bit chỉ định cách đánh địa chỉ cho mỗi toán hạng.

Sau đây là cấu trúc của lệnh với từng kiểu địa chỉ khác nhau : Trong số 16 bit của lệnh có các bit chứa Opcode là phần mã lệnh, và các bit chứa Address là phần địa chỉ.



a) Lệnh không có địa chỉ

b) Lệnh có một địa chỉ

c) Lệnh có 2 địa chỉ

Chương V. Liên kết các thành phần chức năng - bus

1. Khái niệm BUS trong máy tính

Để tạo thành một máy tính, các thành phần (các khối) chức năng của máy tính phải được kết nối với nhau. Cách thức kết nối và trao đổi dữ liệu của các thành phần ảnh hưởng lớn đến hiệu quả của hệ thống.

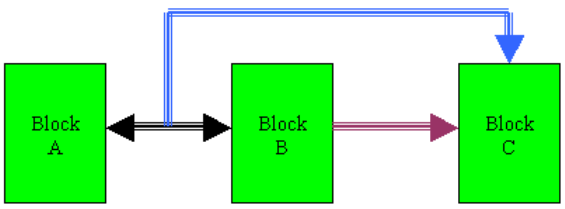
Việc liên kết các thành phần (các khối) chức năng được thực hiện nhờ một *hệ thống BUS*. BUS là các *đường dẫn liên kết các khối chức năng* trong hệ thống máy tính. Một trong những đặc điểm chính của BUS là dùng chung môi trường truyền dẫn. Các loại dữ liệu và lệnh trong máy tính đều được biểu diễn thông qua tổ hợp các dãy số nhị phân (các bit "0" và "1") và được thể hiện vật lý qua hiện tượng "không có" hoặc "có" điện áp tương ứng, do vậy môi trường truyền dẫn ở đây chính là các đường dây dẫn diện. Đặc điểm của môi trường truyền dẫn điện tạo nên BUS trong máy tính là có độ truyền dẫn gần như lý tưởng (trở kháng suất là thấp nhất có thể).

Đơn vị số học và logic thường được đặt cùng với đơn vị điều khiển và tạo thành CPU. Nhắc lại rằng CPU chịu trách nhiệm về mọi hoạt động của máy tính, điều khiển hoạt động của mọi thành phần khác. Do đó CPU phải được kết nối với mọi thành phần đó.

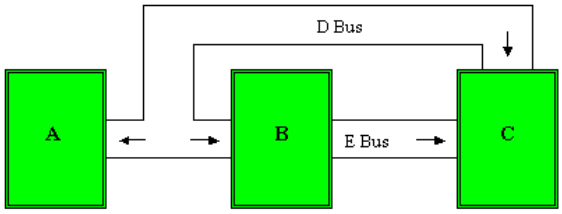
Trong các máy tính trước đây, mỗi thành phần như bộ nhớ, thiết bị vào/ra thực sự được kết nối với CPU bằng đường kết nối riêng. Với kiến trúc như thế CPU tham gia vào mọi tiến trình diễn ra trong máy tính, hệ thống có quá nhiều đường kết nối với bộ phối ghép (giao diện) riêng.

Để hạ giá thành và tiêu chuẩn hóa logic của giao diện, giải pháp thông dụng là sử dụng một hệ thống kết nối duy nhất để kết nối mọi thành phần của máy tính.

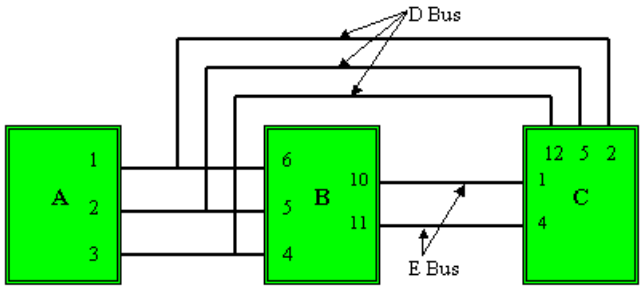
Giả thiết một kiến trúc cơ bản như Hình . Hình a) và b) là vẽ theo kiến trúc, song trên hình b) việc kết nối được thể hiện bằng BUS, đầy đủ hơn là cách vẽ theo nhận thức ở a). Tất nhiên, để các BUS liên kết có thể thực hiện được, phải có được cách thể hiện đầy đủ qua sơ đồ nối chân như ở c)



Ha. Kiến trúc hệ thống



b. Kiến trúc hệ thống nhìn từ góc độ Bus



c. Sơ đồ nguyên lý kết nối các khối chức năng

Như vậy bus là đường truyền thông tin trong máy tính, kết nối hai hoặc nhiều thiết bị của máy tính. Đặc điểm quan trọng của bus là môi trường truyền dẫn thông tin chung giữa các thiết bị.

Về mặt vật lý, bus là tập hợp các đường dây dẫn truyền tín hiệu điện, mỗi đường có khả năng truyền một bit thông tin (0 hoặc 1) tại một thời điểm. Các thiết bị được kết nối lên bus và tín hiệu do một thiết bị phát ra có thể được nhận bởi mọi thiết bị khác đang được kết nối (về mặt điện) lên bus. Nhiều đường truyền gộp lại có thể truyền đồng thời các dãy số nhị phân. Ví dụ nhóm dữ liệu 8 bit có thể truyền bằng bus gồm 8 đường truyền.

Nếu hai thiết bị cùng phát tín hiệu lên bus tại một thời điểm, các tín hiệu này sẽ bị chồng lên nhau và gây tín hiệu sai lệch. Như vậy, tại mỗi thời điểm chỉ một thiết bị có thể truyền tín hiệu thành công.

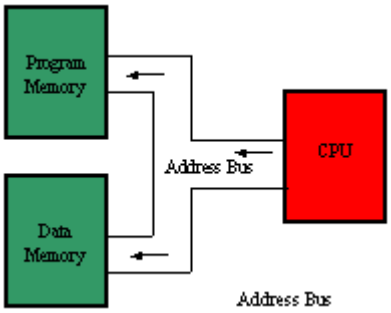
Sự kết nối các thiết bị của máy tính có thể được thực hiện bằng nhiều hệ thống bus khác nhau. Bus kết nối các thành phần chính của máy tính (CPU, bộ nhớ, các thiết bị vào/ra) được gọi là bus hệ thống.

2. Bus hệ thống

Mỗi bus hệ thống thường gồm hàng chục đến hàng trăm đường riêng biệt, mỗi đường có chức năng riêng. Nhưng tựu trung, có thể phân loại các đường trong bus thành ba nhóm theo chức năng sau: Bus địa chỉ, bus dữ liệu và bus điều khiển.

2.1. Bus địa chỉ

Bộ nhớ được CPU truy cập đến từng ô nhớ cụ thể qua bus địa chỉ (*Address Bus*). Bus địa chỉ là bus được dùng để truyền địa chỉ của ô nhớ hoặc thiết bị do CPU lựa chọn và muốn truy nhập. Để lựa chọn ô nhớ hoặc thiết bị, CPU phát ra địa chỉ tương ứng và địa chỉ này được truyền trên bus địa chỉ đến nơi CPU cần truy nhập. Bus địa chỉ là loại bus một chiều. Độ rộng của bus địa chỉ (hay số lượng đường truyền) xác định kích thước tối đa của bộ nhớ trong máy tính.



2.2. Bus dữ liệu

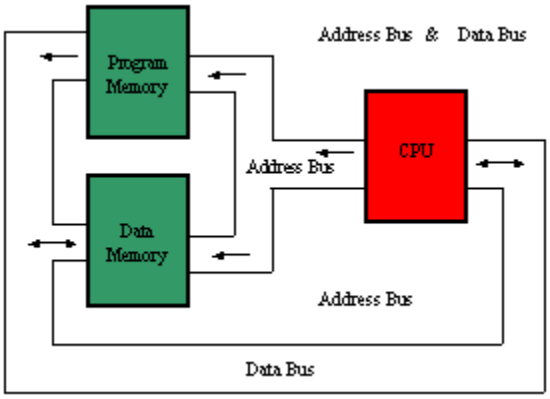
Dữ liệu từ bộ nhớ được CPU đọc ra hay được CPU lưu giữ vào phải được truyền qua một kênh khác được gọi là bus dữ liệu (*Data Bus*). Như vậy bus dữ liệu được dùng để truyền dữ liệu. Bus dữ liệu có thể gồm 8, 16 hoặc 32 đường, do vậy tại mỗi thời điểm có thể truyền dữ liệu có kích thước 8, 16 hoặc 32 bit. Bus địa chỉ là bus hai chiều, dữ liệu có thể do CPU phát ra hay nhận về từ bộ nhớ hoặc các thiết bị.

Tại mõi thời điểm, CPU chỉ làm việc với bộ nhớ hoặc với một thiết bị. Khi CPU muốn trao đổi thông tin với đối tượng nào, CPU sẽ phát ra địa chỉ của nó lên bus địa chỉ. Đối tượng có địa chỉ tương ứng sẽ được kết nối lên bus dữ liệu để thực hiện quá trình truyền dữ liệu.

Trong đa số các hệ thống, các bus địa chỉ hoàn toàn do bus master điều khiển. Thông thường, bus master chính là CPU. Nếu chỉ có một bus master, các thành phần khác kết nối lên bus được gọi là slave. Mỗi slave có một địa chỉ tương ứng và bus master sử dụng các bus địa chỉ để điều khiển thành phần nào được sử dụng bus. Trong một số hệ thống khác, ngoài CPU còn có các thành phần khác có thể có quyền điều khiển hệ thống trong những thời điểm khác nhau. Khi đó thành phần điều khiển bus chỉ được gọi là bus master tại thời điểm nó thực sự có quyền điều khiển bus.

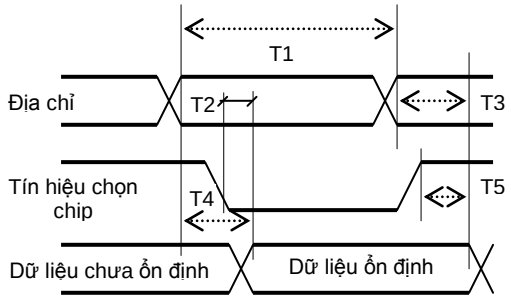
Lưu ý rằng các thiết bị vào/ra cũng sẽ được liên kết với CPU thông qua Address bus và Data bus để thực hiện các thao tác đưa dữ liệu vào CPU hoặc từ CPU ra. Như vậy, sự kết hợp giữa hai đường truyền cơ bản này (bus

địa chỉ và bus dữ liệu) tạo điều kiện cho CPU cơ chế ĐỌC ra từ hoặc GHI vào một ô nhớ (hoặc thiết bị) xác định thông qua địa chỉ cụ thể của ô nhớ (hoặc thiết bị) đó. Một lần nữa, lưu ý rằng *Address bus là bus một chiều* (chỉ do CPU đưa ra). Còn *Data bus là bus hai chiều* để CPU có thể đọc dữ liệu từ ô nhớ ra hoặc ghi một dữ liệu vào ô nhớ (hoặc thiết bị).

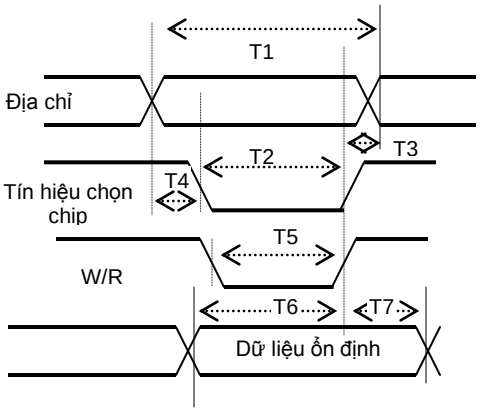


2.3. Định thời hoạt động Ghi/Đọc trong giao tiếp CPU với bộ nhớ

Hoạt động xử lý dữ liệu của CPU trong máy tính, ngoài việc thực hiện các tác vụ số học và logic, thì các tác vụ trao đổi thông tin, dữ liệu giữa CPU và bộ nhớ chiếm rất nhiều thời gian, và xảy ra liên tục. Theo nguyên lý Von Neumann, lệnh và dữ liệu được lưu giữ trong bộ nhớ, nên các tác vụ trao đổi thông tin, dữ liệu bao gồm đọc lệnh và ghi/đọc dữ liệu đến và từ các ô nhớ phải đảm bảo đạt tốc độ cao và chính xác. Trong bất kỳ một tác vụ Ghi/Đọc nào, việc xác định nguồn hoặc đích của dữ liệu đều phải được xác định trước, sau đó mới đến nội dung dữ liệu. Do vậy, địa chỉ của ô nhớ cần phải được CPU đưa ra trước, và phải ổn định trên BUS trước khi đưa ra tín hiệu Ghi/Đọc. Trên biểu đồ thời gian, thấy rằng quá trình Ghi/Đọc chỉ xảy ra khi *địa chỉ đã ổn định và dữ liệu ổn định.*



Chu kỳ đọc dữ liệu ROM



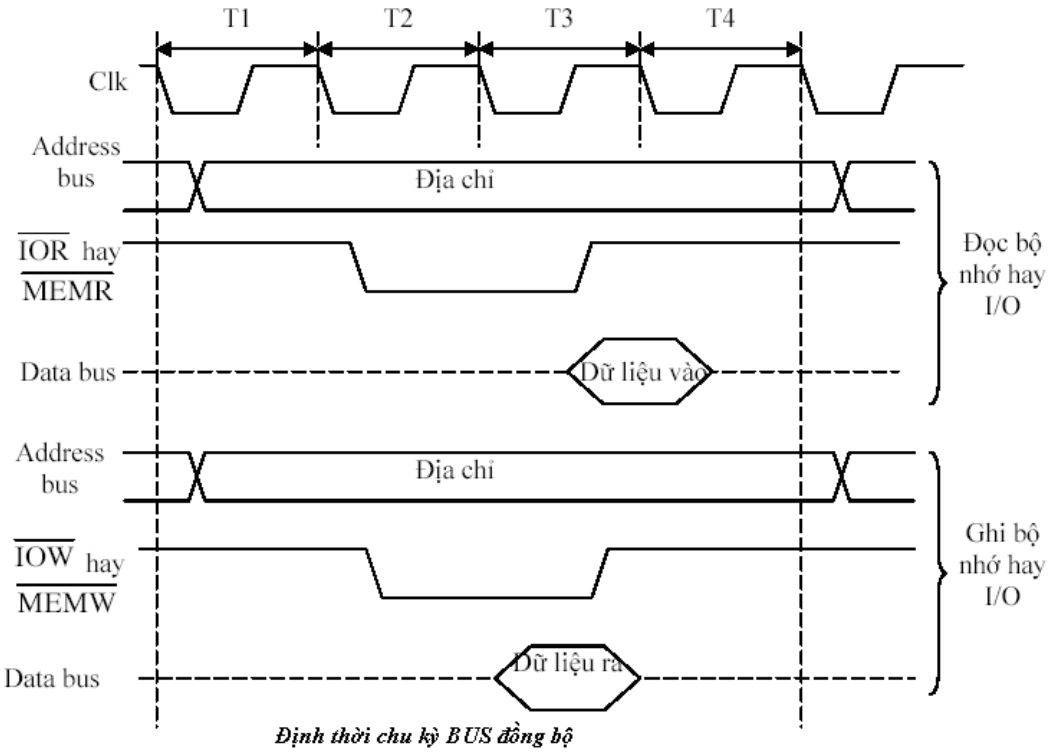
Chu kỳ Ghi/Đọc dữ liệu RAM

Nguyễn Trung Đồng – Tel 0983 410 866

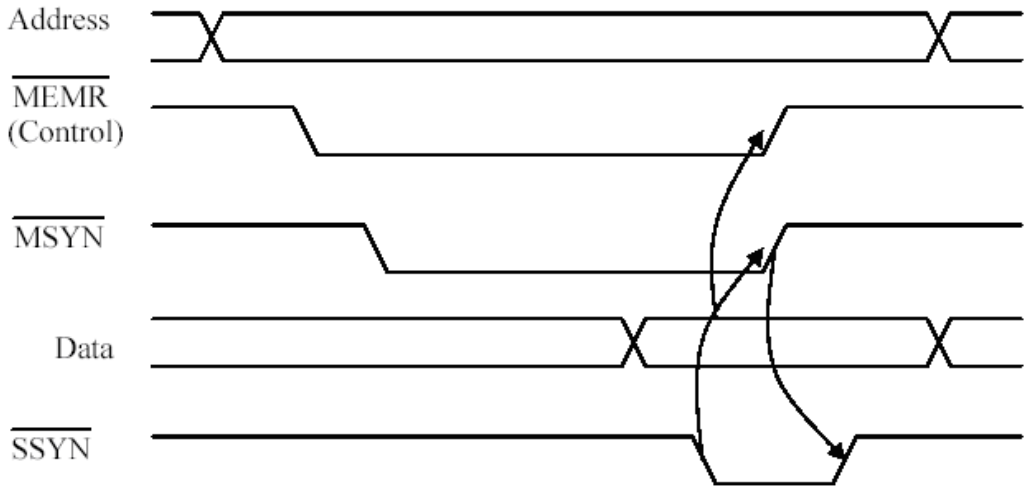
Trong hai biểu đồ thời gian trên, thấy rằng:

- Trong chu kỳ đọc dữ liệu từ bộ nhớ ROM, hoặc trong chu kỳ Ghi/Đọc bộ nhớ RAM, khoảng thời gian T4 là cần thiết trước khi xuất hiện tín hiệu chọn chip là rất quan trọng.

- Tín hiệu Ghi/Đọc W/R tồn tại trong khoảng dữ liệu đã ổn định trên BUS đảm bảo dữ liệu Ghi/Đọc được không bị lỗi.



Định thời chu kỳ BUS đồng bộ



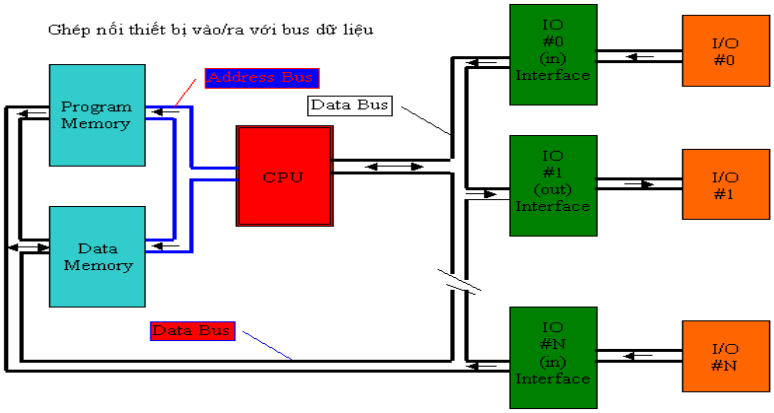
Định thời chu kỳ BUS không đồng bộ

Nguyễn Trung Đồng – Tel 0983 410 866

2.3. Giao tiếp CPU với thiết bị ngoại vi

Như đã biết, các thiết bị vào/ra được kết nối với CPU thông qua các bộ phối ghép (hay còn gọi là giao diện - *interface*). Ta cũng đã nói rằng các thiết bị vào/ra cũng sẽ được liên kết với CPU bằng Address bus và Data bus để thực hiện các thao tác đưa dữ liệu vào CPU hoặc từ CPU ra.

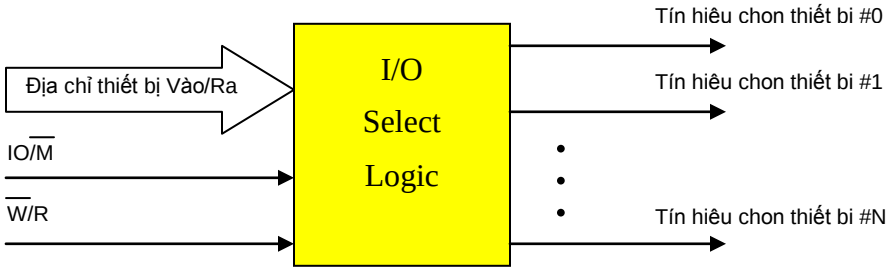
Điều đáng lưu tâm ở đây là: *Các thiết bị vào/ra chỉ có thể thực hiện duy nhất một nhiệm vụ: Hoặc cung cấp dữ liệu cho CPU, hoặc lấy dữ liệu do CPU cung cấp*, nên với thiết bị vào, Data bus là một chiều từ đó vào CPU, đối với thiết bị ra, Data bus có chiều ngược lại. Chiều của Data BUS được quyết định bởi bộ phối ghép thiết bị vào/ra (*IO Interface*).



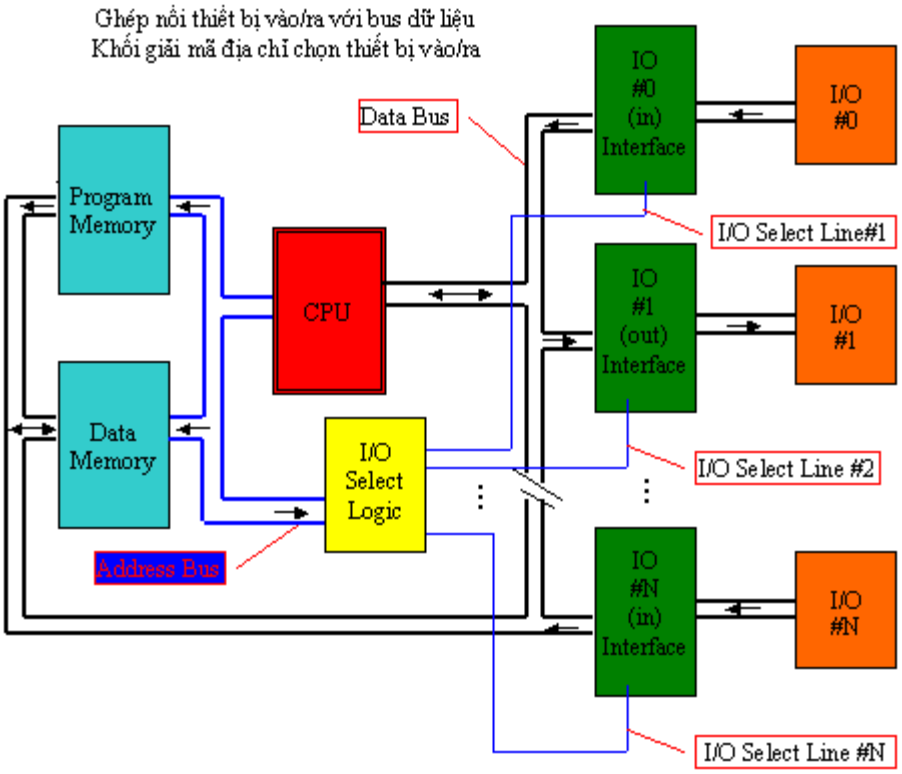
Từ những khái niệm trên, suy ra cần phải có một khối chức năng quan trọng để đảm bảo việc truyền dữ liệu giữa CPU với các thiết bị vào/ra là *mạch logic chọn thiết bị vào/ra* (*I/O Select Logic*).

Mạch này làm việc theo nguyên tắc chọn đúng thiết bị vào/ra thông qua địa chỉ của thiết bị vào/ra do CPU cung cấp để tạo ra các tín hiệu chọn thiết bị vào/ra. Các tín hiệu này được đưa đến thiết bị vào/ra cụ thể thông qua dây chọn (*I/O Select Line*). Dĩ nhiên, lối vào của mạch chọn này là các tín hiệu địa chỉ do CPU cung cấp, nó được *nối với Address Bus*.

Thực chất, mạch chọn thiết bị vào/ra chính là mạch giải mã địa chỉ thiết bị vào ra, tạo tín hiệu chọn thiết bị vào/ra từ địa chỉ của thiết bị và tín hiệu xác nhận làm việc với thiết bị vào ra IO/M.

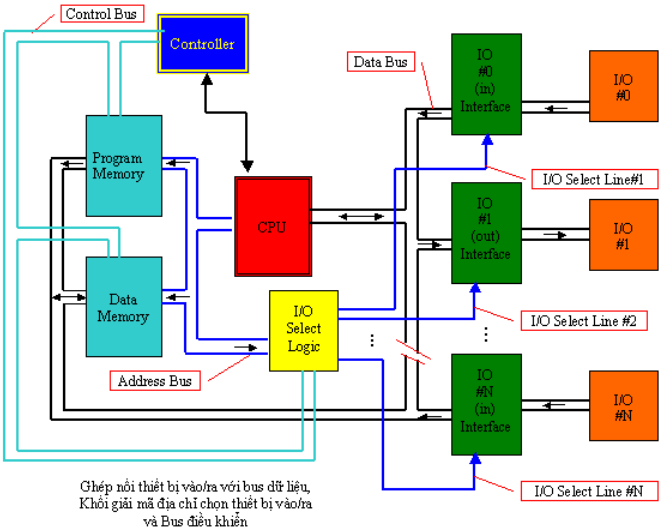


Nguyễn Trung Đồng – Tel 0983 410 866



2.4. Bus điều khiển

Hoạt động *trao đổi dữ liệu giữa CPU với bộ nhớ và thiết bị vào/ra diễn ra trên Data Bus*, vị trí cụ thể, tức là đích đến của dữ liệu, hoặc nơi cung cấp dữ liệu theo yêu cầu của CPU *do Address Bus* đảm nhận.



Vì mọi thành phần của máy tính đều sử dụng chung các bus địa chỉ và bus dữ liệu để truyền thông tin, để tránh xung đột cần có khối chức năng điều khiển việc sử dụng các bus này. Việc điều phối các hoạt động trên do một khối chức năng đặc biệt đảm nhận, đó là *khối điều khiển (Controller)*,

(hay còn gọi là *đơn vị điều khiển – CU – Control Unit)*. Dữ liệu vào của khối này là mã đã được khối giải mã lệnh tạo ra từ việc phân tích các lệnh CPU cần thực thi. Các tín hiệu do khối này tạo ra được đưa đến cho các khối chức năng theo các đường truyền dẫn riêng, gọi là bus điều khiển (*Control Bus*). Các tín hiệu điều khiển trên Control Bus điều phối mọi hoạt động chức năng của các thiết bị liên quan (bộ nhớ, thiết bị vào/ra...).

Các đường truyền của bus điều khiển bao gồm:

– Ghi bộ nhớ (Memory Write): Cho phép ghi dữ liệu trên bus dữ liệu vào ô nhớ có địa chỉ trên bus địa chỉ;

– Đọc bộ nhớ (Memory Read): Cho phép đọc dữ liệu của ô nhớ có địa chỉ trên bus địa chỉ lên bus dữ liệu;

– Ghi vào/ra (I/O Write): Cho phép truyền dữ liệu trên bus dữ liệu đến cổng vào/ra có địa chỉ trên bus địa chỉ;

– Đọc vào/ra (I/O Read): Cho phép truyền dữ liệu từ cổng vào/ra có địa chỉ trên bus địa chỉ lên bus dữ liệu;

– Xác nhận trao đổi (Transsfer ACK): Cho biết dữ liệu đã được đưa lên bus hay đã được đọc từ bus;

– Yêu cầu bus (Bus Request): Cho biết một thành phần yêu cầu sử dụng bus;

– Cấp bus (Bus Grant): Cho biết thành phần yêu cầu sử dụng bus được quyền điều khiển bus;

– Yêu cầu ngắt (Interrupt Request): Cho biết có yêu cầu ngắt;

– Xác nhận ngắt (Interrupt ACK): Xác nhận yêu cầu ngắt đã được nhận;

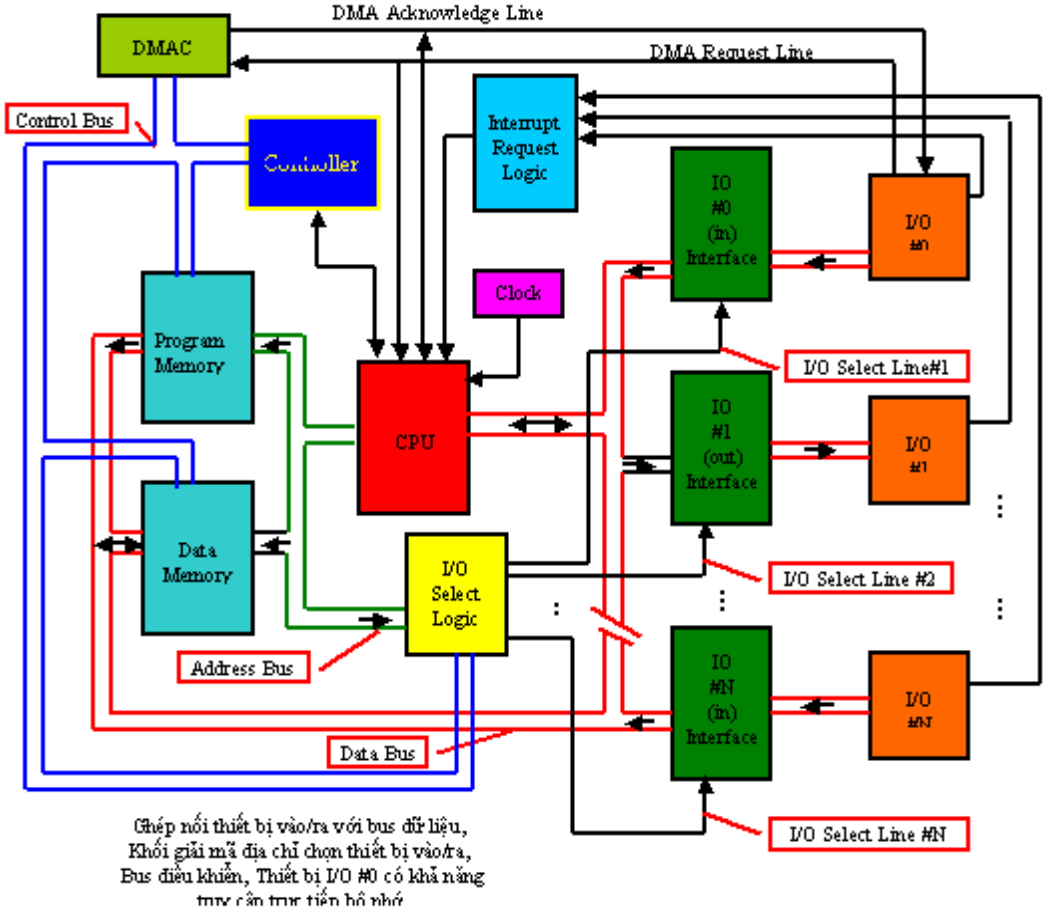
– Xung nhịp (Clock): Được sử dụng để đồng bộ hóa các hoạt động;

– Khởi động lại (Reset): Khởi động lại tất cả các module.

2.5. Truy nhập trực tiếp bộ nhớ và ngắt

Một vấn đề thuộc phạm vi nâng cao hiệu quả hoạt động của CPU là: Hoạt động của các thiết bị vào ra bao giờ cũng chậm hơn rất nhiều so với tốc độ xử lý của CPU. Để giải quyết vấn đề, các nhà thiết kế đưa ra giải pháp *"truy nhập trực tiếp bộ nhớ (Direct Memory Access)"*. Do sử dụng BUS chung trong trao đổi dữ liệu, hoạt động truy nhập trực tiếp bộ nhớ của các thiết bị vào ra được tổ chức và thực hiện một cách chặt chẽ nhờ *mạch điều*

khiển truy nhập trực tiếp bộ nhớ DMAC. Giả sử thiết bị nhập dữ liệu #0 có nhu cầu chuyển một khối dữ liệu vào bộ nhớ, nó gửi yêu cầu thực thi DMA bằng tín hiệu Request DMA qua đường DMA Request Line. CPU phân tích tín hiệu này và tự tách ra khỏi các BUS chung, gửi tín hiệu nhận biết và chấp thuận cho #0 qua đường DMA Acknowledge Line, trao quyền sử dụng BUS chung cho thiết bị #0 và thiết bị #0 thực hiện việc truy nhập trực tiếp bộ nhớ.



Một điều quan trọng nữa trong việc vào/ra dữ liệu là vấn đề kịp thời phục vụ yêu cầu cung cấp dữ liệu của thiết bị vào/ra đối với CPU (hoặc nhập vào hoặc đưa ra). Những yêu cầu này thường xuất hiện bất ngờ, không có "hẹn" trước. Giải quyết vấn đề này, kiến trúc máy tính đưa ra giải pháp *NGẮT* đối với CPU. Khối xử lý yêu cầu ngắt CPU gửi tín hiệu yêu cầu qua *Interrupt Request Logic* tới chân yêu cầu ngắt của CPU. CPU thực hiện nốt lệnh đang thực hiện và gác tiến trình thực hiện chương trình lại để quay ra thực hiện chương trình phục vụ ngắt; Hoặc thu thập, xử lý dữ liệu do thiết bị vào/ra cung cấp, hoặc cung cấp dữ liệu đã xử lý cho thiết bị...

Như vậy một kiến trúc máy tính, tuỳ theo mục đích sử dụng, có thể được thực hiện theo kiến trúc như ở hình 10. Khối chức năng CLOCK là bộ phận tạo xung nhịp để đồng bộ hoá mọi hoạt động của các khối chức năng.

3. Hoạt động của bus

3.1. Hoạt động của bus

Hoạt động của bus xảy ra như sau:

a) Nếu một thiết bị muốn gửi dữ liệu đến một thiết bị khác, thiết bị đó phải

(i) Lấy quyền sử dụng bus

(ii) Truyền dữ liệu qua bus tới thiết bị nhận dữ liệu.

b) Nếu một thiết bị yêu cầu dữ liệu từ một thiết bị khác, thiết bị đó phải

(i) Lấy quyền sử dụng bus

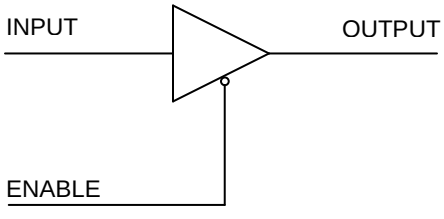
(ii) Truyền yêu cầu dữ liệu đến thiết bị có dữ liệu thông qua các bus điều khiển và bus địa chỉ tương ứng.

(iii) Chờ thiết bị kia gửi dữ liệu.

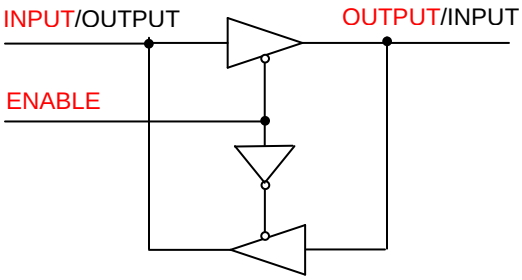
3.2. Kết nối các thiết bị lên bus

Việc chỉ dùng một bus để kết nối và truyền thông tin giữa các thiết bị khác nhau đòi hỏi tại một thời điểm chỉ một thiết bị được phép phát tín hiệu lên bus. Nếu tại một thời điểm có nhiều thiết bị được kết nói và cùng phát tín hiệu lên bus thì sẽ gây ra xung đột.

Để làm chủ việc kết nối các thiết bị lên bus người ta sử dụng thiết bị điện tử có tên là *thiết bị ba trạng thái*. Dạng logic cuả thiết bị này như sau:



Sơ đồ logic phần tử 3 trạng thái



Ghép nối hai phần tử 3 trạng thái để tạo liên kết hai chiều có điều khiển

Nguyễn Trung Đồng – Tel 0983 410 866

Thiết bị ba trạng thái có hai đầu vào là INPUT và ENABLE, và một tín hiệu ra OUTPUT. Tín hiệu ra có thể có ba trạng thái: 0, 1, và trạng thái *Không kết nối* (High Impedance State – trạng thái trở kháng cao). Khi đầu vào ENABLE = 1, ở đầu ra OUTPUT là *Không kết nối* với INPUT, còn khi ENABLE = 0 thì đầu ra OUTPUT có cùng giá trị như đầu vào INPUT. Phối ghép 2 phần tử 3 trạng thái ngược chiều nhau, tín hiệu ENABLE đảo giá trị cho nhau, ta được phần tử liên kết hai chiều có điều khiển, được sử dụng trong các liên kết lên BUS dữ liệu.

Một điều quan trọng cần hiểu ró là khi nhiều thiết bị ba trạng thái cùng được kết nối lên một đường truyền và chỉ một thiết bị này có các đầu vào 1 và ENABLE, khi đó thiết bị này sẽ điều khiển trạng thái của đường truyền.

Thiết bị ba trạng thái cho phép CPU và các bus master khác điều khiển việc kết nối các đối tượng khác nhau lên bus. Khi cần chọn đối tượng nào, CPU phát ra địa chỉ của đối tượng đó và tín hiệu điều khiển tương ứng. Các đối tượng thực hiện giải mã địa chỉ này, kết hợp với tín hiệu điều khiển để tạo ra tín hiệu cho phép ENABLE. Tại đối tượng có địa chỉ phù hợp, tín hiệu ENABLE = 0 cho phép đối tượng này kết nối lên bus.

Các đối tượng khác sẽ có tín hiệu ENABLE = 1 nên không được kết nối.

Về mặt vật lý, BUS là các dây dẫn song song chạy suốt qua các thành phần chức năng của máy tính, mỗi thành phần được kết nói với toàn bộ hay chỉ một phần trong số các dây Cách kết nối thông dụng: Các thành phần của máy tính được kết nối từng khoảng trên bus thông qua các rãnh cắm (slot) có sẵn và các bảng mạch. Cách bố trí này có lợi thế có thể mở rộng hay thay thế các thành phần của máy tính một cách dễ dàng.

3.3. Phân cấp bus

Một hệ thống máy tính thường sử dụng nhiều bus. Bố trí đặc trưng của bus được minh họa trên hình sau:

Bus cục bộ được dùng để nối bộ xử lý với bộ nhớ cache. Bus này có thể được kết nối với các bộ điều khiển I/O cục bộ.

Bộ điều khiển bộ nhớ cache được kết nối với bộ nhớ chính bằng *bus hệ thống*. Ở đây, cấu trúc cache được sử dụng để tách quá trình truy xuất bộ nhớ thường xuyên từ các thành phần khác, do đó bộ nhớ được gắn với bus hệ

thống. Do đó các thiết bị vào/ra có thể truy xuất bộ nhớ thông qua bus hệ thống mà không ảnh hưởng tới hoạt động của CPU.

Có thể kết nối các bộ phối ghép thiết bị vào/ra lên bus hệ thống. Tuy nhiên, thiết kế hiệu quả hơn là sử dụng một hay nhiều *bus ngoại vi* để kết nối chúng và sau đó kết nối *bus ngoại vi* với bus hệ thống thông qua bộ phối ghép bus ngoại vi (expansion bus interface). (hình...)

Kiến truc này đảm bảo hiệu quả hợp lý, song trong trường hợp các thiết bị vào/ra có tốc độ cao, chúng sẽ làm giảm hiệu suất. Người ta thường sử dụng một bus có tốc độ cao kết nối trực tiếp với bus cục bộ và bus hệ thống thông qua cầu nối cache để giải quyết vấn đề đối với các thiết bị vào/ra có tốc độ cao. (hình...)

3.4. Các đặc trưng thiết kế bus

Các đặc trưng của bus bao gồm:

- Kiểu bus
- Điều khiển
- Chu kỳ bus

3.4.1. Kiểu bus

Bus được chia làm hai kiểu chính: bus dành riêng và bus dùng chung. Bus dành riêng là bus được dùng để phục vụ một mục đích cụ thể hoặc kết nối một số thành phần nhất định.

Ví dụ, bus có thể dành riêng để truyền địa chỉ hoặc dữ liệu; đây là cách thiết kế phổ biến nhất. Tuy nhiên địa chỉ và dữ liệu có thể sử dụng một bus chung bằng cách kích hoạt một đường truyền xác định đó là quá trình truyền dữ liệu hay địa chỉ.. Khi truyền dữ liệu, địa chỉ được đưa lên bus và đường truyền này được kích hoạt. Mỗi thành phần của máy tính sẽ nhận địa chỉ này trong khoảng thời gian thích hợp và thiết bị có địa chỉ tương ứng được xác định. Sau đó địa chỉ được xóa khỏi bus và quá trình truyền dữ liệu bắt đầu trên cùng bus đó. Phương pháp này có ưu điểm là giảm không gian và hạ giá thành, tuy nhiên việc thiết kế các mạch logic phức tạp hơn và hiệu suất truyền thông tin bị giảm.

Bus dành riêng về mặt vật lý là bus chỉ được kết nối với những thành phần nhất định. Một ví dụ đặc trưng là sử dụng I/O bus để kết nối các thiết bị vào ra, sau đó bus I/O được kết nối chung với bus chung. Thiết kế này cho hiệu suất cao.

3.4.2. Điều khiển

Trong đại đa số các trường hợp, nhiều thành phần có yêu cầu điều khiển bus, ví dụ như thiết bị vào/ra yêu cầu kết nối trực tiếp tới bộ nhớ không qua CPU. Vì tại mỗi thời điểm chỉ có một thành phần có thể truyền thông tin thành công qua bus, cần phải có một cơ chế điều khiển bus nào đó. Các phương pháp điều khiển được chia ra hai loại chính: tập trung và phân tán.

Với cách quản lý tập trung, *bộ điều khiển bus* chịu trách nhiệm phân phát thời gian sử dụng bus cho mỗi thành phần.

Cách điều khiển phân tán không cần bộ điều khiển bus mà mỗi thành phần có một mạch logic điều khiển truy cập bus.

3.4.3. Chu kỳ bus

Hai phương pháp truyền thông tin trên bus là

- Đồng bộ;
- Không đồng bộ.

Với phương pháp đồng bộ, CPU điều khiển toàn bộ các quá trình truyền thông tin thông qua các tín hiệu điều khiển ghi/đọc. Đây là phương pháp được sử dụng trong hầu hết các hệ thống.

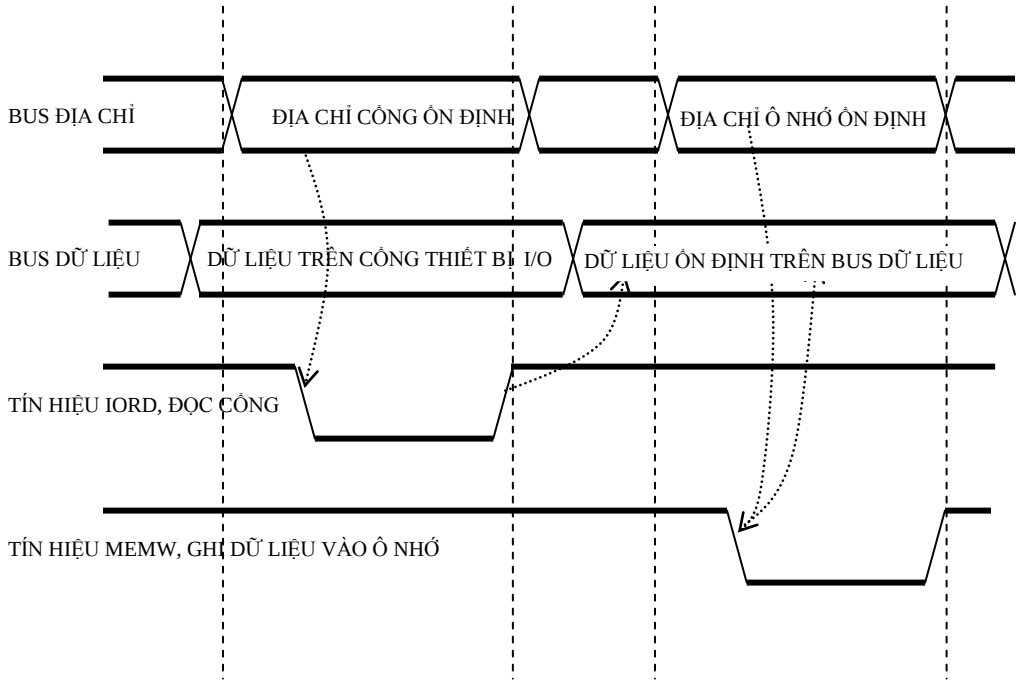
Chu kỳ bus là khoảng thời gian được CPU dùng để thực hiện một thao tác truyền thông tin nhất định với một đối tượng nhất định. Mỗi chu kỳ bus kéo dài trên nhiều chu kỳ nhịp đồng hồ của máy tính.

Có 6 loại chu kỳ bus cơ bản:

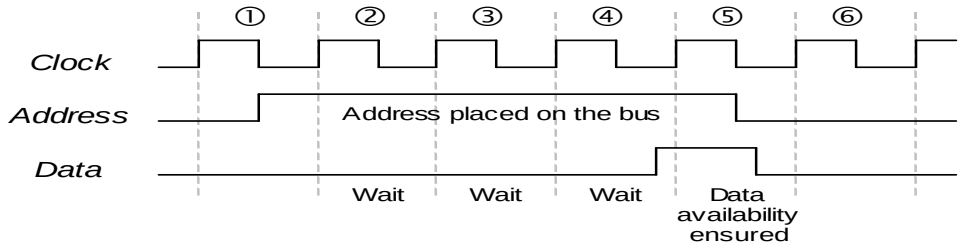
1. Nhập lệnh;
2. Đọc bộ nhớ;
3. Ghi bộ nhớ;
4. Đọc cổng vào/ra;
5. Ghi cổng vào ra;
6. Trả lời ngắt

Để hiểu được quá trình truyền thông tin trên hệ thống bus theo kỹ thuật đồng bộ, ta khảo sát quá trình CPU nhập dữ liệu từ thiết bị vào bộ nhớ (Đọc cổng vào/ra). Trong ví dụ này, chu kỳ bus gồm hai chu kỳ nhỏ: Chu kỳ đọc cổng và chu kỳ ghi bộ nhớ. Trong chu kỳ đọc cổng, CPU đưa địa chỉ của cổng vào/ra được chọn lên bus địa chỉ. Khi địa chỉ đã ổn định, CPU phát ra tín hiệu điều khiển đọc cổng I/O lên bus điều khiển. Thiết bị vào/ra được

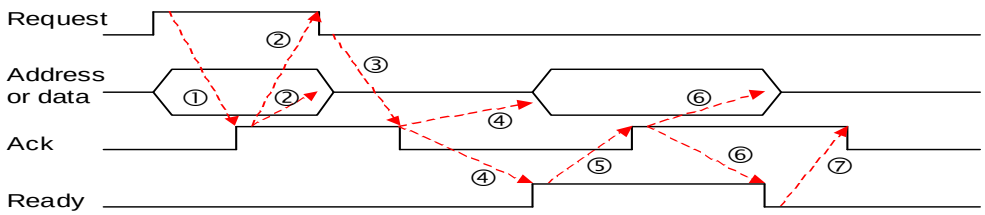
chọn đưa dữ liệu lên bus dữ liệu, khi dữ liệu ổn định, CPU nhập dữ liệu này. Trong chu kỳ ghi bộ nhớ, CPU đưa địa chỉ của bộ nhớ lên bus địa chỉ. Khi địa chỉ ổn định, CPU phát ra tín hiệu điều khiển ghi cổng I/O lên bus điều khiển. Dữ liệu từ CPU được ghi vào vị trí tương ứng trong bộ nhớ.



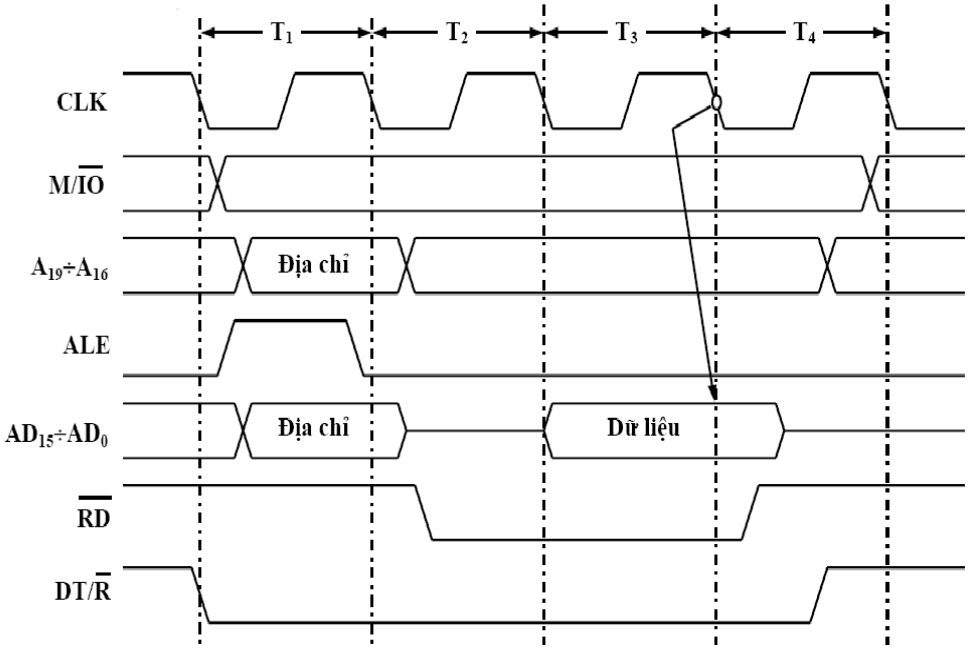
BUS đồng bộ và Định thời đọc dữ liệu



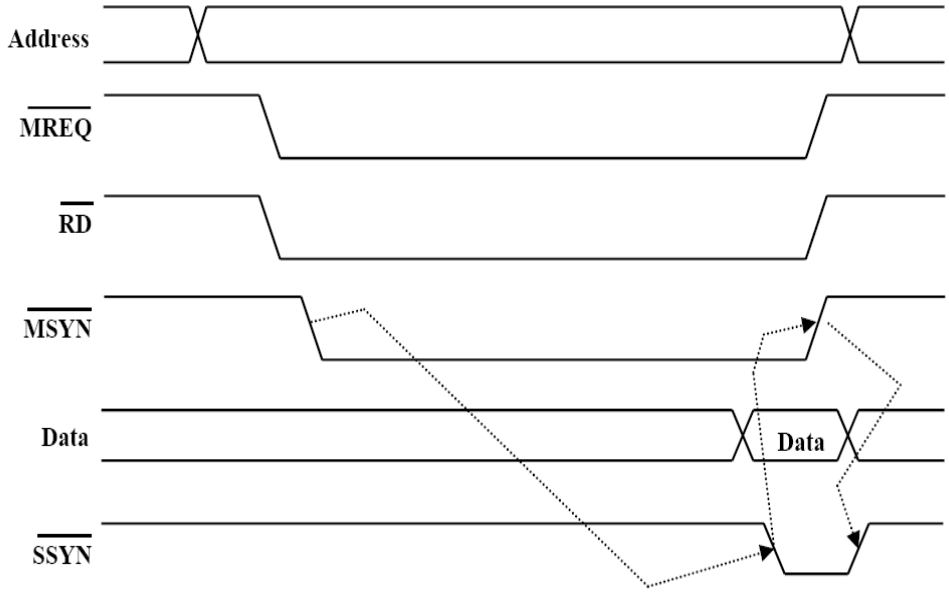
Phương thức sử dụng Handshaking trong trao đổi dữ liệu với bộ nhớ



Nguyễn Trung Đồng – Tel 0983 410 866



BUS không đồng bộ và Định thời đọc dữ liệu



Chương VI. Kiến trúc bộ nhớ (8)

1. Bộ nhớ trong của máy tính

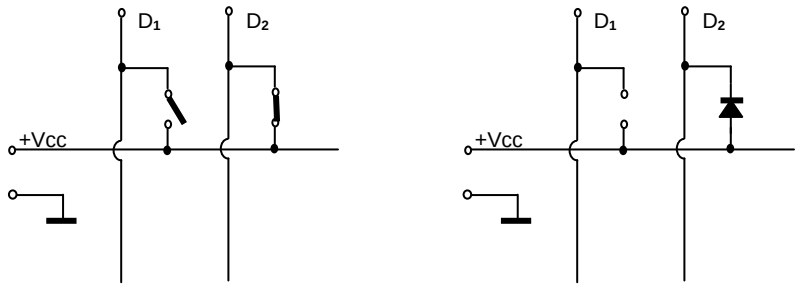
Bộ nhớ được sử dụng để lưu giữ mã lệnh của chương trình và dữ liệu cần xử lý. Bộ nhớ được ghép nối trực tiếp với CPU qua BUS hệ thống và là nơi đầu tiên CPU truy xuất tới để lấy thông tin khi khởi động hệ thống. Yêu cầu đặt ra cho bộ nhớ là phải cho phép truy xuất với tốc độ cao để đáp ứng kịp thời các đòi hỏi của CPU. Chỉ có bộ nhớ bán dẫn mới đáp ứng được yêu cầu cao về tốc độ truy xuất cao (hàng trăm đến hàng chục nsec).

Bộ nhớ bán dẫn được chia ra hai loại: Bộ nhớ chỉ đọc ROM (Read Only Memory) và bộ nhớ truy xuất ngẫu nhiên RAM (Random Access Memory).

1.1. Phần tử nhớ, vi mạch nhớ, từ nhớ và dung lượng bộ nhớ

Phần tử nhớ

Phần tử nhớ thông thường là một mạch điện có thể ghi lại và lưu giữ một trong hai giá trị của một biến nhị phân, hoặc "0" hoặc "1", tương ứng với *không có điện áp* hoặc *có điện áp,* được gọi là **bit**. Trên mạch điện dưới đây (Hình III.1), trên dây D_1 sẽ không có điện áp (do công tắc mở), trong khi dây D_2 có điện áp (vì công tắc đóng, hay thông qua diode mắc theo chiều thuận), gần bằng giá trị nguồn nuôi Vcc, tương ứng với bit D_1 = "0" và bit D_2 = "1".



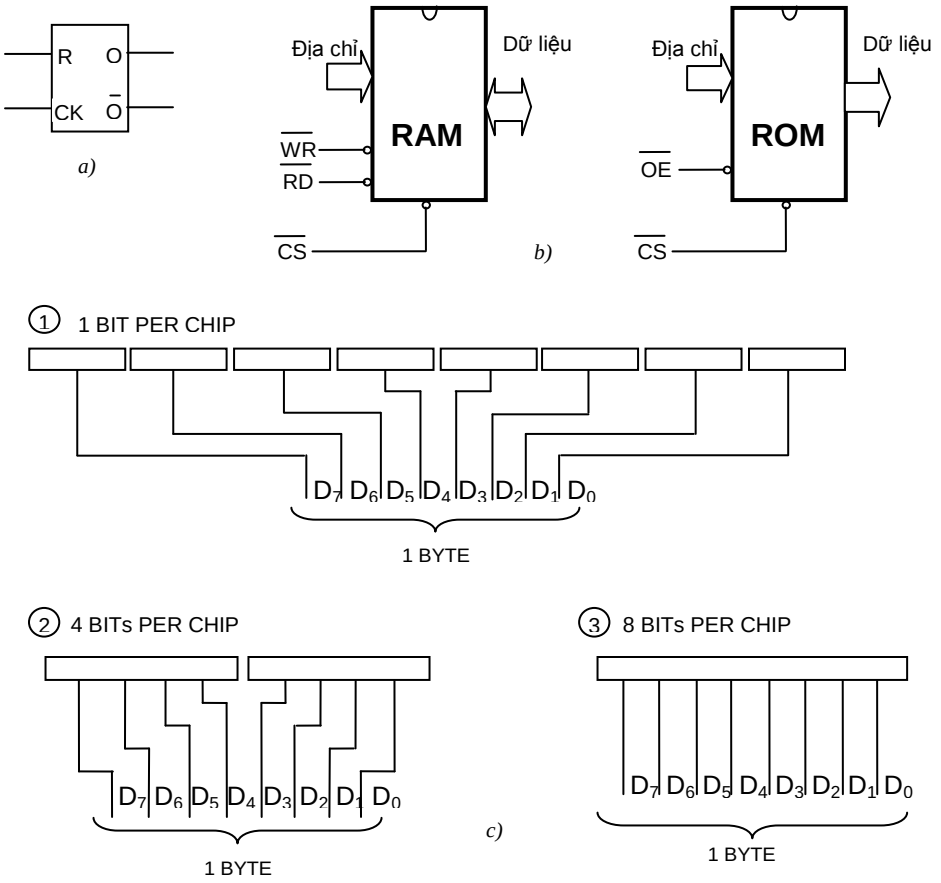
Phương pháp tạo phần tử nhớ $D_1 = 0$ và $D_2 = 1$ bằng mạch điện đơn giản

Hình III.1 Mô phỏng phần tử nhớ

Mạch flip-flop RS (còn gọi là triger RS) đồng bộ là một mạch có khả năng lưu giữ các giá trị "0" hoặc "1" ở lối ra. Có thể dùng RS flip-flop làm một mạch lưu giữ tín hiệu vào R bằng cách chốt dữ liệu đó lại tại đầu ra Q (hình III.2a). Các hãng chế tạo thực hiện mạch này bằng công nghệ cao, nên kích thước vô cùng nhỏ, có thể có hàng nhiều triệu phần tử nhớ trên một diện tích 1mm^2. Các vi mạch nhớ thông thường được chế tạo với độ dài từ nhớ và

số lượng từ nhớ cố định. *Số bit nhớ được liên kết tại một vị trí nhớ (có cùng địa chỉ) trong một chip nhớ* được gọi là *từ nhớ* của chip nhớ, thường được chọn là 1, 4, hoặc 8bit. Để tạo được một *từ nhớ của bộ nhớ*, tức là từ nhớ có độ dài (số bit trong một từ) chuẩn (theo chuẩn IBM là 8 bits), trong một số trường hợp nhất định cần phải tiến hành ghép các chip nhớ lại với nhau.

Hình III.2 a), b) và c) cho ta khái niệm về khả năng tạo một từ nhớ cơ bản (byte) khi từ nhớ của chip nhớ là 1bit, 2bits và 4 bits. Trong trường hợp độ dài từ nhớ của chip nhớ là 8 bits, việc liên kết là không cần thiết.



Hình III.2 a) Mạch Flip-flop RS như một phần tử nhớ giá trị nhị phân
b)Chip nhớ RAM và chip nhớ ROM

Do ưu điểm tương thích tuyệt đối về kích thước, tiêu thụ năng lượng thấp và mức logic, đặc biệt là tốc độ truy nhập, nên bộ nhớ bán dẫn được sử dụng làm bộ nhớ chính (Main Memory) trong các hệ Vi xử lý cũng như trong các máy tính PC, nhiều khi được ghép nối ngay trong bo mạch chính, hoặc được thiết kế như những vỉ nhỏ cắm vào khe cắm riêng trên bo mạch chính.

Nhờ những tiến bộ vượt bậc của công nghệ vi mạch, đặc biệt là công nghệ cao (High Technology) các chip nhớ được chế tạo ngày càng nhỏ và có dung lượng tương đối lớn, tốc độ truy nhập rất cao và giá thành thấp. Hiện đã có các chip nhớ có dung lượng hàng trăm triệu từ nhớ, được cấu thành từ hàng chục tỷ transistor trên một một cấu trúc cỡ $1mm^2$.

Bộ nhớ trong của một hệ Vi xử lý gồm hai loại chính:

– Bộ nhớ ROM – là bộ nhớ chỉ đọc (Read Only Memory), thông thường chứa các chương trình giám sát (monitoring) các hoạt động chức năng của hệ Vi xử lý: chương trình thiết lập hệ thống, chương trình vào/ra dữ liệu, quản lý và phân phát bộ nhớ, quản lý các thiết bị vào/ra v.v…Đối với máy tính PC, đó là chương trình hệ thống vào/ra cơ sở (BIOS – Basic Input Output System). Đặc điểm cơ bản nhất của bộ nhớ này là sự bảo toàn dữ liệu khi không có nguồn nuôi.

– Bộ nhớ RAM – là bộ nhớ ghi/đọc tuỳ tiện (Random Access Memory). Vì có khả năng ghi/đọc tuỳ theo người dùng, nên bộ nhớ này được sử dụng để chứa dữ liệu, các chương trình ứng dụng nhất thời của người dùng v.v… Trong máy tính PC, bộ nhớ này là nơi chương trình hệ điều hành được nạp khi khởi động máy, hay nơi chứa các chương trình ứng dụng lúc nó được thực thi. Bộ nhớ này bị mất dữ liệu khi bị mất nguồn nuôi.

Trong các hệ Vi xử lý đơn giản, hai bộ nhớ này thường được thiết kế và lắp ráp từ các chip nhớ riêng biệt thành một vỉ nhớ. Địa chỉ được *giải mã cho từng chip nhớ* nhờ khối giải mã, thông thường là một vi mạch giải mã hay được xây dựng từ các mạch tổ hợp logic. Các tín hiệu điều khiển việc ghi/đọc bộ nhớ do CPU cung cấp. Mạch triger RS đồng bộ là một mạch có khả năng lưu giữ các giá trị "0" hoặc "1" ở lối ra. Có thể dùng RS flip-flop làm một mạch lưu giữ tín hiệu vào R bằng cách chốt dữ liệu đó lại tại đầu ra Q (hình III.2)

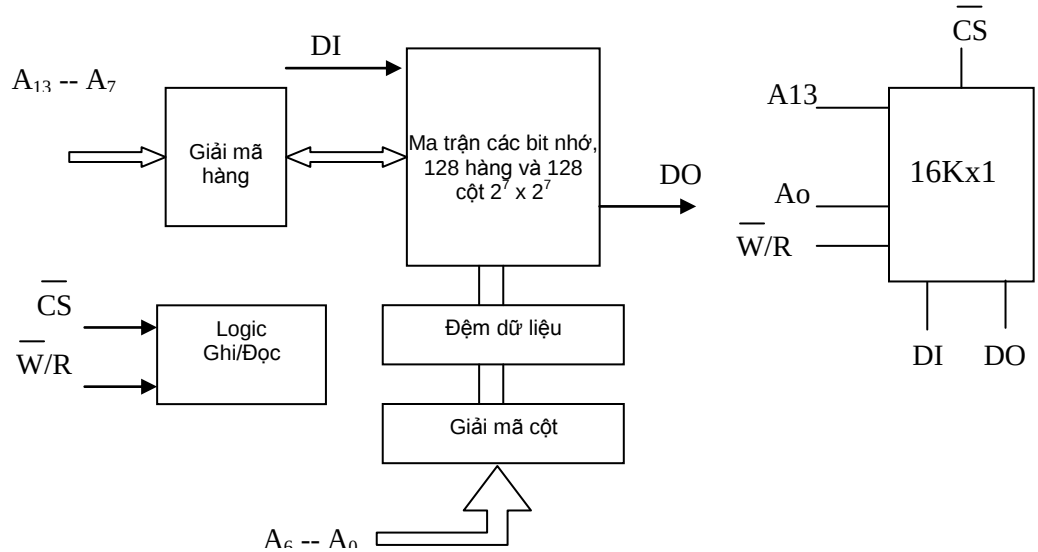
Bộ nhớ được xây dựng từ các chip nhớ. Các chip nhớ RAM (SRAM hoặc DRAM) thường có các từ nhớ có độ dài 1 bit, 4 bits hoặc 8 bits. Từ các

chíp nhớ loại này có thể xây dựng được bộ nhớ với mỗi ô nhớ chứa được byte dữ liệu (8 bits).

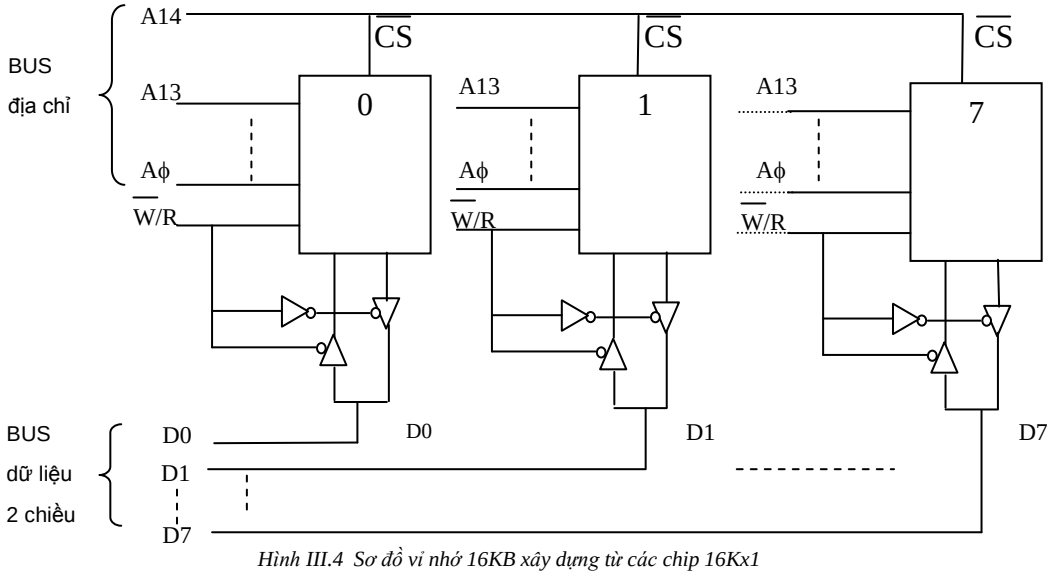
1.2. Xây dựng bộ nhớ với các chip SRAM

Giả sử cần xây dựng một bộ nhớ kích thước 16Kbyte trên cơ sở các chíp SRAM loại 16Kx1bit.

Băng nhớ SRAM 16Kbyte được xây dựng trên cơ sở 8 chip SRAM loại 16K x 1bit, để có được ô nhớ có độ dài 8 bits (từ nhớ cơ bản). Để làm được điều này người ta sắp đặt 8 chip SRAM loại 16K x 1bit sao cho mỗi chip tại một vị trí xác định sẽ đảm nhiệm lưu trữ bit dữ liệu có trọng số tương ứng trong byte dữ liệu.



Hình III.3 Chip nhớ RAM 64K bit (64K x 1)



Hình III.4 Sơ đồ vi nhớ 16KB xây dựng từ các chip 16Kx1

Các đường tín hiệu :

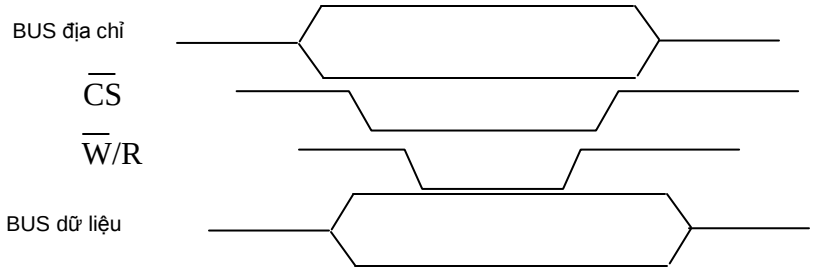
A13 - A0 BUS địa chỉ

-CS: Tín hiệu chọn chip. Nếu CS = 0 thì truy nhập được chip

-W/R: Tín hiệu điều khiển ghi/đọc. W=0 điều khiển ghi

D0 - D7: Các đường dây truyền các bit dữ liệu từ D0 đến D7.

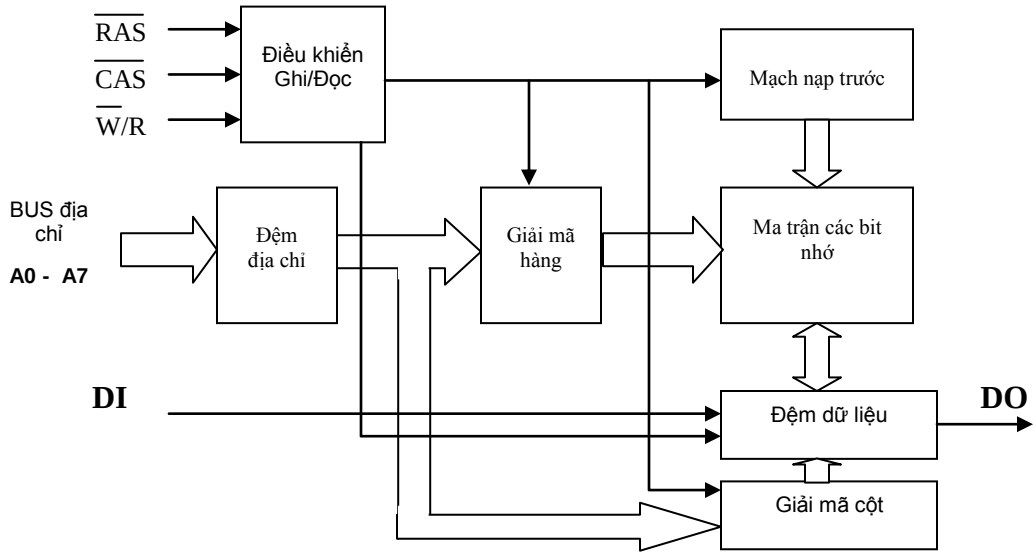
Chu kỳ ghi bộ nhớ SRAM :



Hình III.5 - Biểu đồ thời gian ghi đọc bộ nhớ

1.2.1. Tổ chức bộ nhớ với DRAM

Cấu trúc của chip DRAM:



Hình III.6 - Cấu trúc bên trong chip DRAM

Nguyễn Trung Đồng – Tel 0983 410 866

DRAM dùng phương pháp dồn kênh để nạp lần lượt (2 lần) địa chỉ hàng và địa chỉ cột vào đệm địa chỉ.

Tín hiệu điều khiển :

- $\overline{RAS}$: khi $\overline{RAS}$ (Row Access Strobe) tích cực thì địa chỉ hàng được nạp (chốt lại).
- $\overline{CAS}$: khi $\overline{CAS}$ (Column Access Strobe) tích cực thì địa chỉ cột được nạp (chốt lại).
- $\overline{W}/R$: $\overline{W}/R \equiv$ "0" điều khiển ghi chip, $\overline{W}/R \equiv$ "1" điều khiển đọc chip.

Việc xây dựng bộ nhớ từ các chip DRAM được thực hiện gần tương tự như với SRAM.

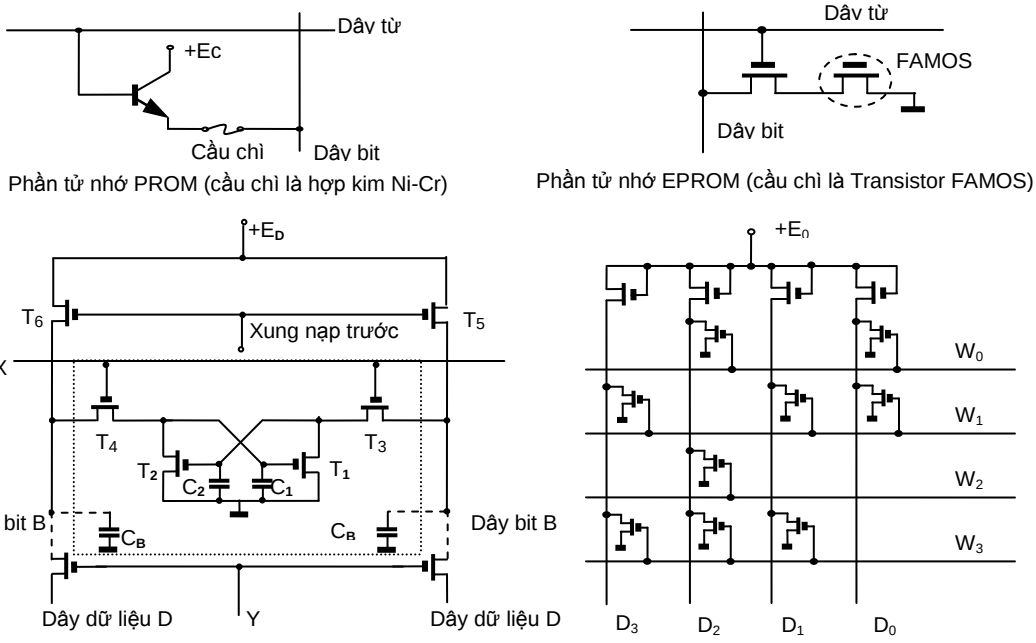
1.2.2. Phân loại các chip nhớ ROM, RAM

Các chip nhớ ROM (Read Only Memory) được phân loại theo khả năng ghi đọc như sau:

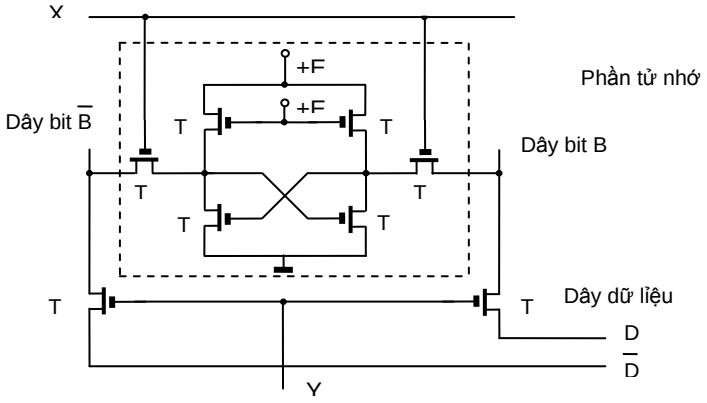
- ROM, nhớ chỉ đọc, dữ liệu trong chip nhớ loại này được ghi ngay tại hãng sản xuất chip nhớ theo đơn đặt hàng của các nhà sản xuất thiết bị cần sử dụng nó.
- EPROM, chip nhớ ROM có khả năng xoá nội dung và ghi lại nội dung. Nội dung được xoá bằng tia cực tím nhờ một thiết bị chuyên dùng.
- EEPROM, chip nhớ ROM có khả năng xoá, ghi lại nhờ sử dụng xung điện

Các chip nhớ RAM chủ yếu được chia thành 2 loại chủ yếu sau:

- RAM tĩnh (SRAM), mỗi phần tử nhớ là một mạch flip-flop, trong quá trình sử dụng không cần quan tâm đến việc dữ liệu được lưu giữ nếu không bị mất nguồn nuôi
- RAM động (DRAM), phần tử nhớ dùng công nghệ nạp điện tích lên tụ điện. Trong quá trình sử dụng cần thiết một chế độ *làm tươi.*



Hình III.7b – Sơ đồ cấu trúc các phần tử nhớ



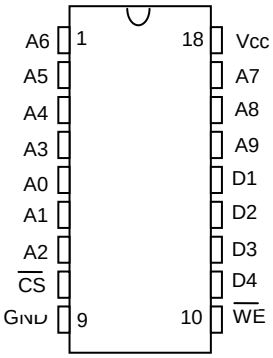
Hình III.7a – Sơ đồ cấu trúc các phần tử nhớ cơ bản

1.2.3. Tổ chức bộ nhớ vật lý

Tổ chức bộ nhớ cho một hệ Vi xử lý (máy vi tính) phụ thuộc không chỉ vào một hệ Vi xử lý cụ thể, mà còn phụ thuộc vào cách bố trí thuận lợi bên trong hệ thống. Trước hết, hãy làm quen với các khái niệm chip nhớ và từ nhớ để phân tích vấn đề tổ chức vật lý một bộ nhớ, sau đó mở rộng khái niệm tổ chức theo quan điểm của người lập trình (tổ chức logic).

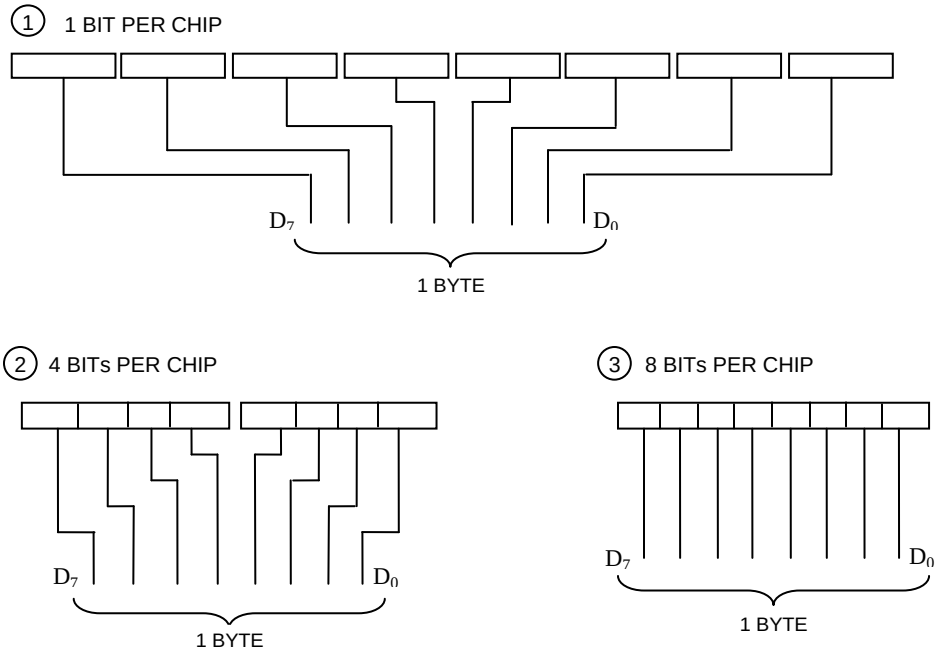
Các chip nhớ được sản xuất dưới nhiều kích cỡ khác nhau, phụ thuộc vào công nghệ chế tạo. Chip nhớ là một vi mạch cụ thể, được bố trí các chân cơ bản như Hình III.8 Các chân của một chip nhớ thông thường gồm các lối vào của BUS địa chỉ, lối dữ liệu, các chân điều khiển chọn chip, ghi/đọc và các chân nguồn.

A0 ÷ A9 Các chân địa chỉ
D1 ÷ D4 Các chân dữ liệu
CS Chân chọn chip
WE Điều khiển Ghi/Đọc
Vcc Chân nguồn nuôi +5V



Hình III.8 Sơ đồ nối chân một vi mạch nhớ RAM 1Kx4

Tuỳ theo từng chip, số lượng chân địa chỉ và số lượng chân dữ liệu có thể khác nhau phụ thuộc vào độ dài *từ nhớ của chip* và *dung lượng của chip nhớ*. Độ dài từ nhớ của chip nhớ có thể là 1bit, 4 bits hoặc 8 bits, trong khi số chân địa chỉ có thể từ 10 trở lên tuỳ thuộc vào dung lượng của chip nhớ. Trong trường hợp độ dài từ nhớ của chip là 1 bit, ta cần phải ghép liên tiếp 8 chip để tạo thành 1 byte, ghép liên tiếp 16 chip để tạo một từ word – 2 bytes). Cần lưu ý việc gán trọng cho các bit trong byte được tạo.



Hình III.8 Tạo từ nhớ 8 bit từ các chíp nhớ có độ dài từ nhớ nhỏ hơn 8 bit

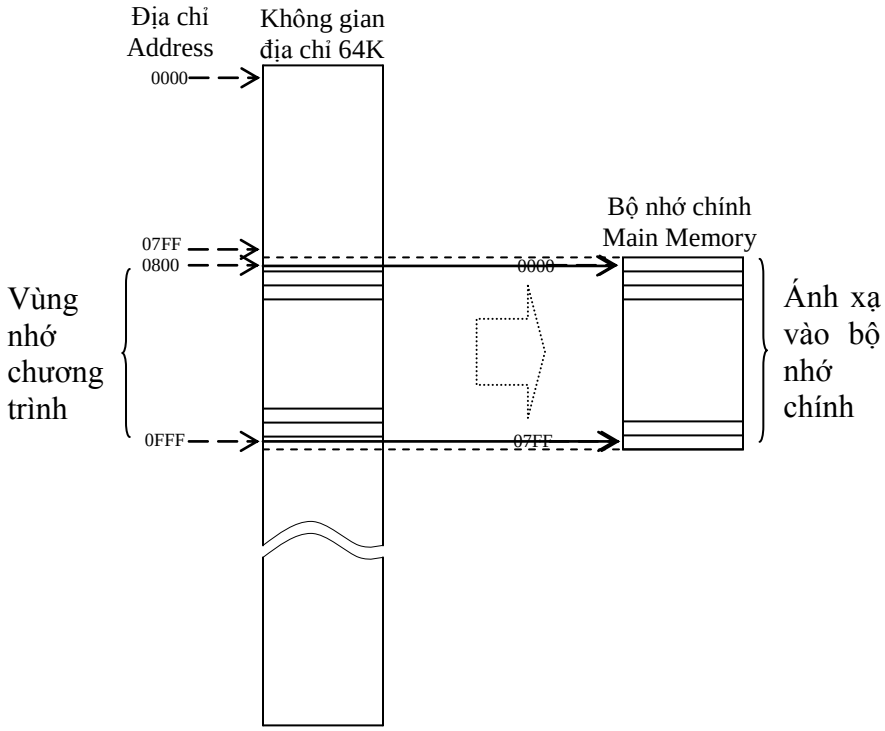
2. Vấn đề quản lý bộ nhớ

Bộ nhớ ngoài của máy tính được dùng để lưu trữ các chương trình và dữ liệu không sử dụng ngay trong quá trình hoạt động. Nội dung các dữ liệu này không bị mất khi tắt nguồn điện. Bộ nhớ ngoài đóng vai trò vô cùng quan trọng, là một bộ phận không thể thiếu trong máy tính.

Các thiết bị nhớ ngoài thông dụng hiện nay là đĩa cứng, đĩa quang, bộ nhớ Flash…

2.1. Chiến lược phân trang (Paging)

Chương trình muốn thực hiện bao giờ cũng được nạp vào bộ nhớ trong của máy tính. Các không gian miền địa chỉ không khả dụng phải được truy xuất thông qua địa chỉ bộ nhớ thực tế. Việc này thực hiện được nhờ phương thức *ánh xạ bộ nhớ từ các địa chỉ của không gian địa chỉ vào các vị trí nhớ thực* như trong hình vẽ sau:



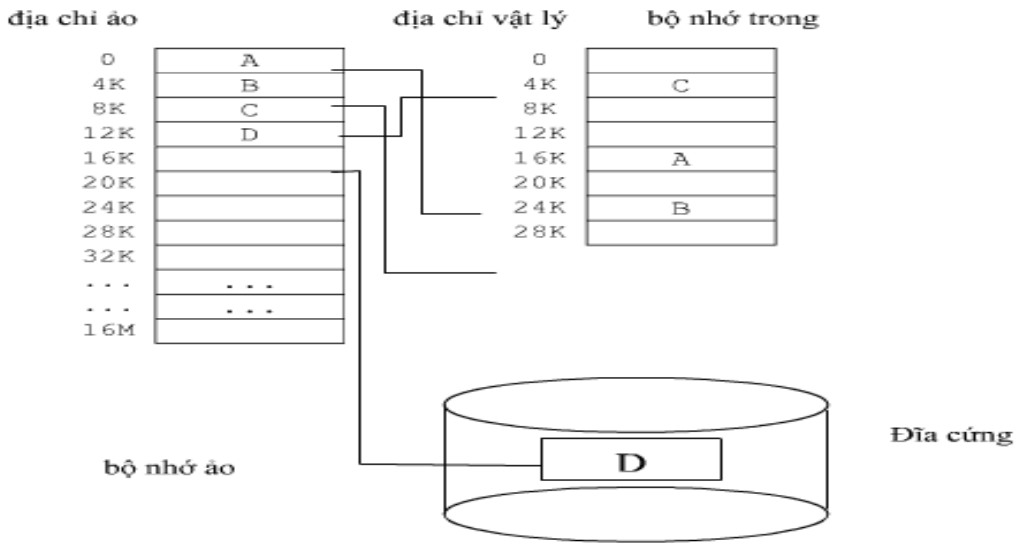
Có thể suy diễn như sau: Vào bất kỳ thời điểm nào, các ô nhớ trong khoảng 4Kbytes đều có thể được truy nhập trực tiếp, nhưng không phải tương ứng với các địa chỉ từ 0000 đến 07FF. Thí dụ từ một thời điểm nhất

định, mỗi khi *truy nhập ô nhớ 0800* thì *từ nhớ ở ô 0000* của bộ nhớ chính được truy nhập, đáng lẽ truy nhập ô nhớ 0801 thì sẽ sử dụng ô nhớ 0001 v.v…Nói cách khác, ta định nghĩa một sự ánh xạ từ không gian địa chỉ vào các địa chỉ bộ nhớ thực, như minh hoạ trên hình vẽ.

Bằng phuơng thức này, nếu không có bộ nhớ ảo (Virtual Memory), một máy với 4Kbytes chỉ có một ánh xạ cố định từ địa chỉ 0000 đến 07FF vào 4096 từ nhớ. Vấn đề xẩy ra là trong trường hợp có một lệnh nhảy tới địa chỉ vượt quá 4Kbytes này, ví dụ tới địa chỉ trong vùng từ 0800 tới 0FFF. Đối với máy có bộ nhớ ảo, các bước sau sẽ được thực hiện:

- Nội dung bộ nhớ chính được cất vào bộ nhớ phụ
- Các nội dung trong vùng 0800 đến 0FFF đang ở trong bộ nhớ phụ được nạp vào bộ nhớ chính.
- Ánh xạ địa chỉ sẽ thay đổi để ánh xạ các địa chỉ từ 0800 tới 0FFF vào các vị trí nhớ thừ 0000 đến 07FF.
- Chương trình được tiếp tục .

Kỹ thuật thực hiện tự động các việc trên được gọi là *kỹ thuật phân trang – Paging.* Các đoạn chương trình được đọc vào bộ nhớ chính từ bộ nhớ phụ được gọi là các *trang*. Miền địa chỉ mà chương trình có thể truy cập là *không gian địa chỉ ảo (Virtual Address Space)*, còn các địa chỉ bộ nhớ thực, được gọi là *không gian địa chỉ vật lý (Physical Address Space)*.

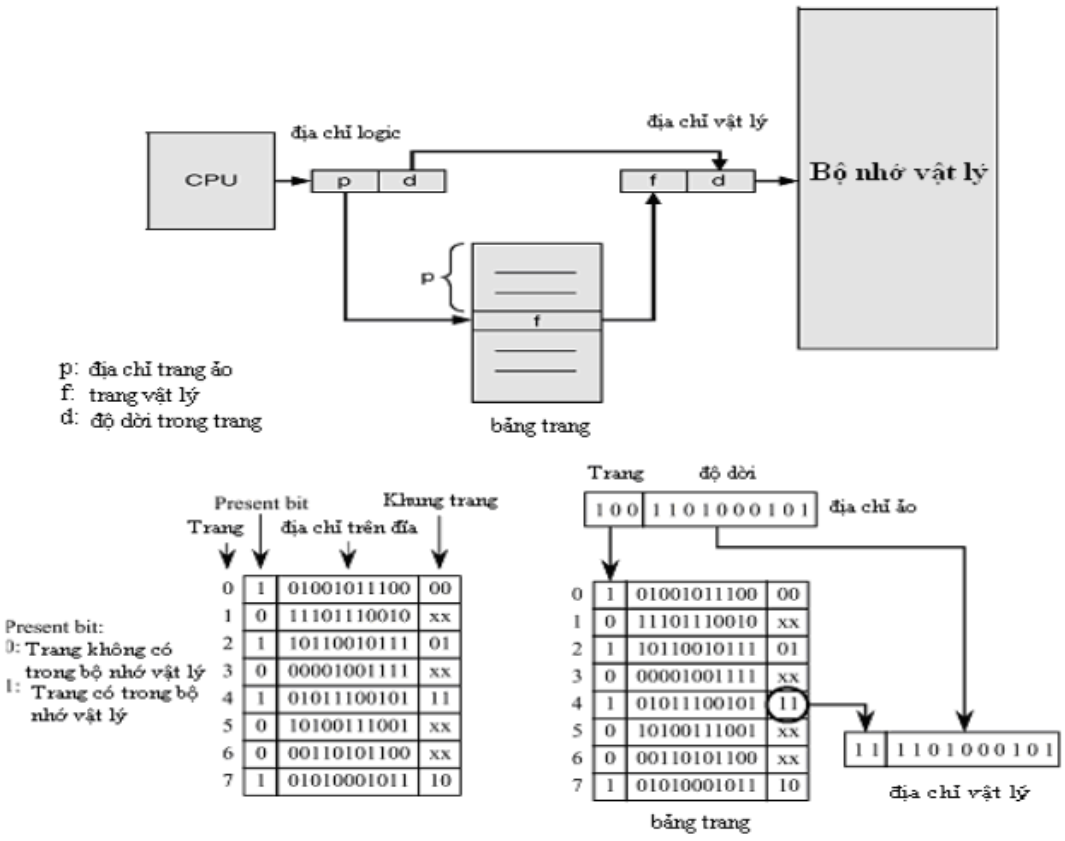


Ví dụ một chương trình gồm có 4 trang A, B, C, D trong đó trang D nằm trong ổ đĩa

Từ thực tế trên, thấy rằng kỹ thuật phân trang khác hẳn với phương thức quản lý bộ nhớ theo phân đoạn (Segmentation) đã trình bày ở chương trước. Lập trình viên luôn luôn quan tâm đến các đoạn trong bộ nhớ (mức lập trình Assembler) nhưng không hề có ý thức về sự tồn tại của bộ nhớ ảo. Lập trình viên thực hiện công việc của mình mà không hề quan tâm dung lượng thực tế của bộ nhớ trong máy tính, mặc dù chúng nhỏ hơn rất nhiều so với không gian nhớ mà CPU có thể quản lý được.

Bộ nhớ ảo đòi hỏi sự tồn tại của bộ nhớ phụ có khả năng lưu được toàn bộ chương trình. Nếu coi bản sao của chương trình trong bộ nhớ phụ là bản gốc, còn các phần của chương trình được tải vào bộ nhớ chính là bản sao, ta dễ dàng nắm bắt được những khái niệm về thực hiện phân trang. Điều quan trọng ở đây là những thay đổi xảy ra với bản sao phải được cập nhật vào bản gốc.

Không gian bộ nhớ ảo được chia thành các trang có kích thước bằng nhau (thông thường là trong khoảng 512 bytes đến 4096 bytes (luỹ thừa của 2).Tương tự, không gian địa chỉ vật lý cũng được chia thành các mảnh, mỗi mảnh có kích thước bằng kích thước một trang. Các mảnh của bộ nhớ chính mà các trang sẽ được chuyển vào gọi là *khung trang – page frame*. Bộ nhớ chính của máy tính thường có rất nhiều khung trang.

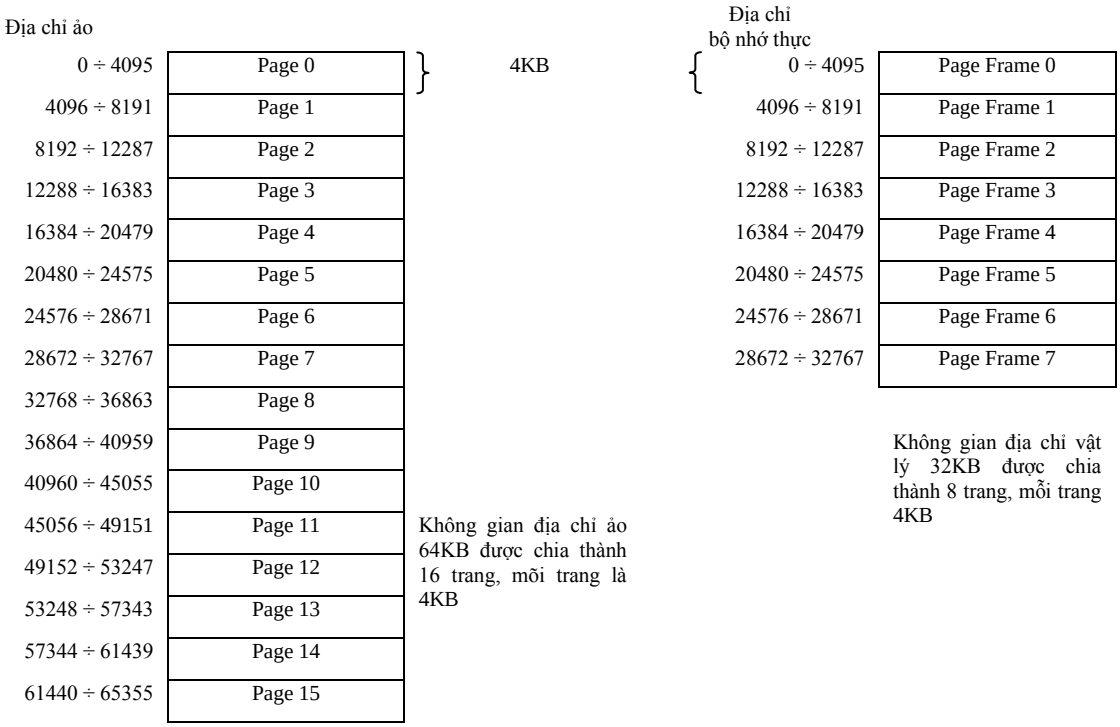


Nguyễn Trung Đồng – Tel 0983 410 866

Ví dụ bộ nhớ ảo 64KB được chia thành 16 trang, mỗi trang 4KB và bộ nhớ thực có dung lượng lầ 32KB được chia thành 8 trang như trên hình sau (*Hình…….)*. Ta cần một *bảng phân trang – page table* gồm 16 từ (word). Như vậy, địa chỉ ảo sẽ được tạo từ 16 bit như sau:

A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
0	0	1	1	0	0	0	0	0	0	1	0	1	1	0	0
Địa chỉ trang của bộ nhớ ảo				Địa chỉ 12 bit cho trang ảo đã chọn											

Với các giá trị như trên, ta tính được địa chỉ ô nhớ đó là 02CH của trang 3, tương ứng là ô nhớ có địa chỉ là 302CH. Với lập luận này, ta có bảng phân trang có 3 trường như sau:



a) Bit cao nhất là "0" hoặc "1" để xác định trang đó có tồn tại trong bộ nhớ chính hay không,

b) Nội dung của 12 bit tiếp theo là địa chỉ của ô nhớ phụ,

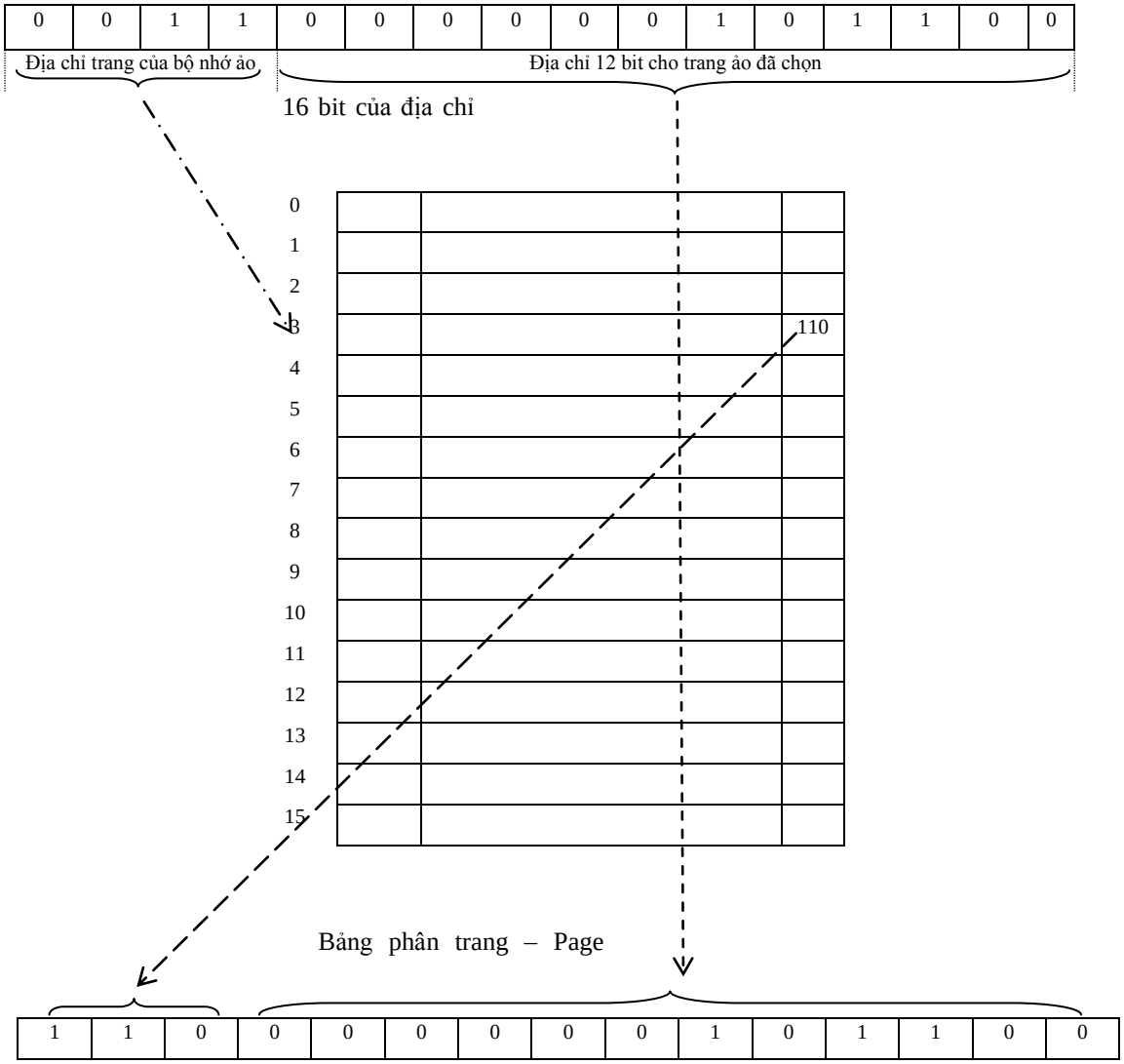
c) 3 bit trẻ nhất là số khung trang.

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
1	0	0	0	0	0	0	1	0	1	1	0	0	1	1	0
	Các bit xác định địa chỉ của ô nhớ phụ												Khung trang		

Nguyễn Trung Đồng – Tel 0983 410 866

Địa chỉ của bộ nhớ chính sẽ được tạo ra từ địa chỉ ảo như sau:

Giả sử trang ảo nằm trông bộ nhớ chính, trường khung tang 3 bit sẽ chỉ cho ta trang đó nằm ở đâu. Nội dung của 3 bit này sẽ được nạp vào 3 bit cao nhất cuả thanh ghi địa chỉ của ô nhớ nằm trong vùng 32KB, 12 bit còn lại sẽ là nội dung 12 bit địa chỉ trong trang ảo, tạo thành một địa chỉ mới gồm 15 bit. Cách tạo ra địa chỉ được mô tả tronh *Hình III*…



2.2. Chế độ bảo vệ (Protected Mode) và quản lý bộ nhớ trong chế độ bảo vệ

Chế độ bảo vệ được thiết kế để hỗ trợ hệ điều hành đa nhiệm, cách ly và bảo vệ hệ điều hành khỏi những truy nhập trái phép của các chương trình ứng dụng, cách ly và bảo vệ chương trình ứng dụng này khỏi sự truy nhập trái phép của chương trình ứng dụng khác.

2.2.1. Các mức đặc quyền và luật về quyền truy nhập

Trong chế độ bảo vệ thì mỗi đoạn nhớ được gắn một mức đặc quyền và được bảo vệ nhờ cơ chế về quyền truy nhập.

Các mức đặc quyền được thiết kế để hỗ trợ hoạt động của hệ điều hành đa nhiệm nhằm :

- Cách ly và bảo vệ hệ điều hành khỏi các truy nhập trái phép của chương trình ứng dụng.
- Cách ly và bảo vệ chương trình ứng dụng này khỏi sự truy nhập trái phép của chương trình ứng dụng khác.

Dựa vào mức đặc quyền và luật về quyền truy nhập mà CPU sẽ quyết định cho phép hay không cho phép truy nhập đoạn nhớ yêu cầu.

Các mức đặc quyền (ký hiệu là PL – Privilege Level) nằm trong một hệ thống các mức đặc quyền đặc quyền có 4 cấp:

- Đặc quyền mức PL = 0, mức đặc quyền cao nhất: các chương trình quản lý thiết bị và quản lý bộ nhớ có mức đặc quyền PL = 0 .
- Đặc quyền mức PL = 1: các chương trình thiết lập mức ưu tiên giữa các nhiệm vụ, chương trình hoán đổi dữ liệu giữa các bộ nhớ chính và bộ nhớ thứ cấp (đĩa từ), chương trình quản lý các cổng vào/ra và các dịch vụ hệ thống khác có mức đặc quyền PL = 1.
- Đặc quyền mức PL = 2: các chương trình quản lý tệp, thư mục và các chức năng mở rộng của hệ điều hành có mức đặc quyền PL = 2.
- Đặc quyền mức PL = 3, mức thấp nhất: các chương trình ứng dụng có mức đặc quyền PL = 3.

Các luật về quyền truy nhập: luật về quyền truy nhập xác định quy tắc truy nhập đoạn nhớ .

Luật 1:

Dữ liệu được lưu trữ trong đoạn nhớ có mức đặc quyền PL = P chỉ có thể bị truy nhập bởi mã lệnh có mức đặc quyền bằng hoặc cao hơn P (CPL ≤ DPL, CPL là mức đặc quyền của nhiệm vụ đang thực hiện, DPL là mức đặc quyền của đoạn dữ liệu bị truy nhập).

Luật 2:

Đoạn mã lệnh có mức đặc quyền PL = P có thể bị gọi hoặc truy nhập bởi nhiệm vụ có mức đặc quyền bằng hoặc thấp hơn P. Đoạn mã lệnh có mức đặc quyền thấp có thể gọi hoặc truy nhập đoạn mã lệnh có mức đặc quyền cao hơn thông qua cửa gọi (CPL ≥ DPL, CPL là mức đặc quyền của

nhiệm vụ đang thực hiện, DPL là mức đặc quyền của đoạn mã lệnh bị truy nhập).

Theo các luật về quyền truy nhập thì chương trình đang thực hiện có thể truy nhập tự do vào các đoạn mã lệnh và đoạn dữ liệu có cùng mức đặc quyền. Một chương trình có thể truy nhập và một đoạn dữ liệu có mức đặc quyền thấp hơn, nhưng nếu truy nhập hoặc gọi đoạn mã lệnh có mức đặc quyền cao hơn thì phải thông qua cổng gọi.

2.2.2. Quản lý bộ nhớ theo phân đoạn trong chế độ bảo vệ

Các đoạn nhớ trong chế độ bảo vệ được quản lý theo 3 thông số:

- Địa chỉ nền
- Giới hạn đoạn
- Quyền truy nhập

Do thông tin về các đoạn khá lớn nên không thể chứa trong thanh ghi đoạn mà được chứa trong các *Bộ mô tả đoạn*. Các bộ mô tả nằm trong *Bảng bộ mô tả*.

Có ba loại Bảng bộ mô tả:

- Bảng bộ mô tả toàn cục GDT (bảng GDT - *Global Descriptor Table*). Bảng GDT quản lý các đoạn (các vùng nhớ) chứa các chương trình của hệ điều hành và dữ liệu của hệ thống (các vùng nhớ chứa các thông tin có tính chất toàn cục, thuộc không gian nhớ toàn cục). Các chương trình ứng dụng có thể truy nhập vùng nhớ này.
- Bảng bộ mô tả cục bộ LDT (bảng LDT - *Local Descriptor Table*). Mỗi Bảng LDT quản lý các vùng nhớ thuộc một nhiệm vụ (các vùng nhớ chứa các thông tin có tính chất cục bộ, thuộc không gian nhớ cục bộ). Mã lệnh và dữ liệu của một chương trình ứng dụng đang chạy (một nhiệm vụ) sẽ được bảo vệ trước sự truy nhập trái phép của các nhiệm vụ khác. Các Bảng LDT thuộc không gian nhớ toàn cục.
- Bảng bộ mô tả ngắt (bảng IDT - *Interrupt Descriptor Table*). Bảng IDT chứa các bộ mô tả trỏ đến 256 chương trình phục vụ ngắt. Bảng IDT đóng vai trò bảng véc tơ ngắt,trong đó mỗi véc tơ ngắt là một bộ mô tả.

Tất cả các Bảng bộ mô tả đều nằm trong bộ nhớ chính.

a. Bộ chọn đoạn 16 bit

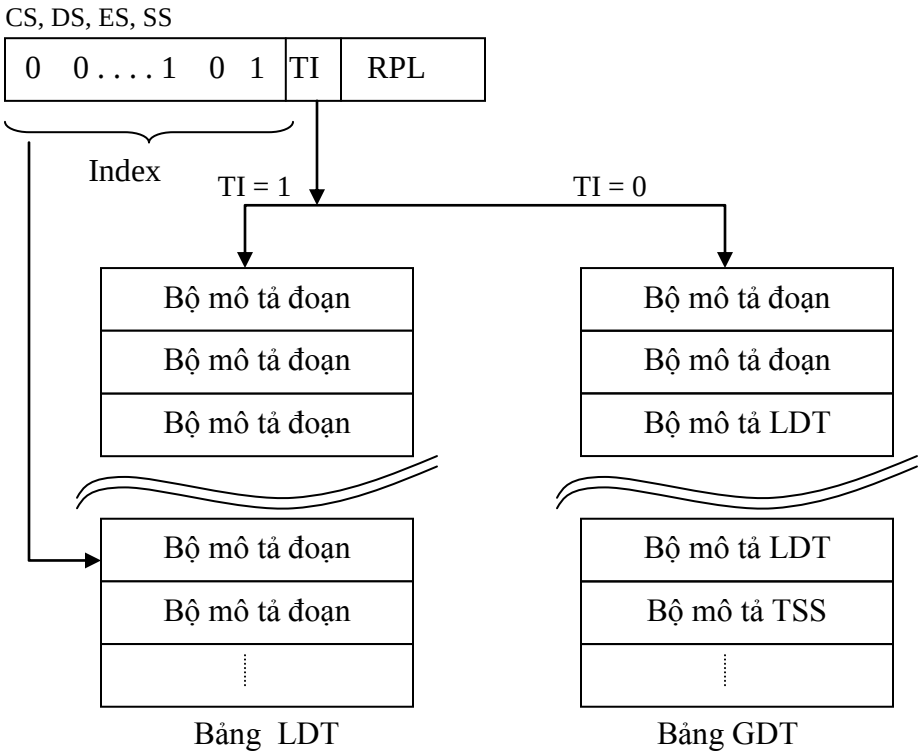
Trong chế độ bảo vệ các thanh ghi đoạn CS, DS, ES, SS không được dùng để xác định địa chỉ nền đoạn như trong chế độ thực, mà được dùng để

chọn Bộ mô tả đoạn trong Bảng bộ mô tả , thực hiện chức năng *Bộ chọn đoạn.*

Bộ chọn đoạn được dùng để xác định vị trí của Bộ mô tả đoạn trong Bảng bộ mô tả. Người lập trình phải nạp Bộ chọn đoạn vào thanh ghi đoạn tương ứng khi muốn truy nhập một đoạn nào đó.

Bộ chọn đoạn có 3 phần :

- Phần Index: 13 bit, dùng để xác định vị trí của Bộ mô tả đoạn, tính từ nền của Bảng bộ mô tả.
- TI: xác định loại Bảng bộ mô tả cần truy nhập.
  TI = 1 Truy nhập các bảng LDT
  TI = 0 Truy nhập bảng GDT
- PL (Requested Privilege Level): mức đặc quyền yêu cầu. Mức đặc quyền RPL được sinh ra bởi người nạp bộ chọn đoạn.

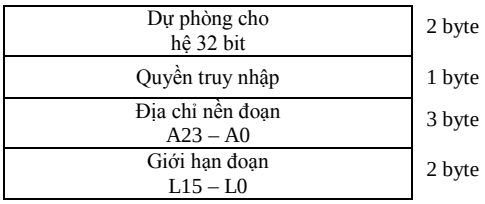


b- Bộ mô tả đoạn

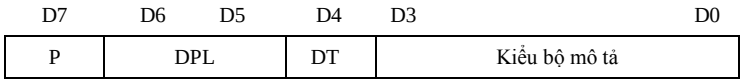
Bộ mô tả đoạn chứa các thông tin quản lý một đoạn : địa chỉ nền đoạn, kích thước (giới hạn) đoạn và quyền truy nhập đoạn.

Bộ mô tả đoạn được hệ điều hành, trình biên dịch hoặc trình nạp bộ nhớ tạo ra.

Bộ mô tả đoạn gồm 8 byte:



- Trường Địa chỉ nền đoạn (24 bit: A23 – A0) xác định địa chỉ nền của đoạn. ở hệ 16 bit thì địa chỉ này cũng là địa chỉ vật lý nền của đoạn.
- Trường Giới hạn đoạn (16 bit: L15 – L0) xác định kích thước của đoạn từ 1 byte đến 64 Kb.
- Trường Quyền truy nhập (8 bit) xác định mức đặc quyền và các thuộc tính khác của đoạn:



P - (Present) : Nếu P = 1 đoạn đang tồn tại trong bộ nhớ.

Nếu P = 0 CPU sẽ tạo ra ngoại lệ "không tồn tại đoạn" khi người yêu cầu chọn đoạn này.

DPL - (Descriptor Privilege Level) : xác định mức đặc quyền của bộ mô tả (mức đặc quyền của đoạn)

DT - (Descriptor Type) : xác định loại bộ mô tả.

DT=1 Bộ mô tả đoạn mã lệnh hoặc dữ liệu

DT=0 Bộ mô tả đoạn hệ thống hoặc cổng giao dịch

Kiểu bộ mô tả : cấu trúc của trường này phụ thuộc vào loại bộ mô tả : Bộ mô tả đoạn dữ liệu, Bộ mô tả đoạn mã lệnh, Bộ mô tả đoạn hệ thống. Bộ mô tả đoạn hệ thống (DT=0) có 2 loại : bộ mô tả LDT, bộ mô tả TSS . Bộ mô tả cổng giao dịch (cổng giao dịch) được dùng để truy nhập vào các đoạn mã lệnh. Các bộ mô tả này sẽ được trình bày ở các phần sau.

- Cấu trúc của byte quyền truy nhập trong Bộ mô tả đoạn *dữ liệu* :

P	DPL	1	0	ED	W/R	A

ED (Expansion Direction) : xác định hướng truy nhập đoạn (hướng tiến triển của địa chỉ)

ED = 1: hướng địa chỉ giảm, đoạn dữ liệu là loại ngăn xếp

ED = 0: hướng địa chỉ tăng

W/R (Write/Read): xác định quyền ghi/đọc

W/R = 1: cho đọc/ghi đoạn dữ liệu

W/R = 0: cấm ghi đoạn dữ liệu

A (Accessed):

A = 1 đoạn đã bị truy nhập

– Cấu trúc của byte quyền truy nhập trong Bộ mô tả đoạn *mã lệnh* :

P	DPL	1	1	C	R	A

C (Conforming) :

C = 0 chương trình con sẽ thực hiện với mức đặc quyền PL = DPL

C = 1 chương trình sẽ thực hiện với mức đặc quyền PL bằng mức đặc quyền của đoạn chứa chương trình gọi chương trình con này.

R (Read):

R = 0 : Đoạn mã lệnh thực hiện được

R = 1 : Đoạn mã lệnh thực hiện được và đọc được

A (Accessed):

A = 1 đoạn mã lệnh đã bị truy nhập

– Cấu trúc của byte quyền truy nhập trong Bộ mô tả đoạn *hệ thống* :

Bộ mô tả đoạn hệ thống (Bộ mô tả đoạn TSS, Bộ mô tả đoạn LDT) quy chiếu (trỏ đến) các đoạn chứa thông tin hệ thống.

P	DPL	0	0	0	Kiểu đoạn

Kiểu đoạn:

– Kiểu=1 : Bộ mô tả quy chiếu đến đoạn trạng thái nhiệm vụ TSS, nhiệm vụ này không ở trạng thái đang thực hiện

– Kiểu=2 : Bộ mô tả quy chiếu đến đoạn chứa bảng LDT

– Kiểu=3 : Bộ mô tả quy chiếu đến đoạn trạng thái nhiệm vụ TSS của nhiệm vụ đang thực hiện.

– Hai byte dự phòng cho hệ 32 bit có dạng :

Địa chỉ nền A31 – A24	
Đặc tính đoạn	Giới hạn đoạn L19 – L16

Đối với hệ 16 bit thì hai byte này phải có giá trị là 0000H.

a- Bộ mô tả cổng giao dịch (cổng giao dịch) :

Bộ mô tả cổng giao dịch (cổng giao dịch) được dùng để truy nhập vào các đoạn mã lệnh. Cổng giao dịch cung cấp phương tiện chuyển giao điều khiển giữa chương trình nguồn và chương trình đích, ví dụ các lệnh CALL và có thể truy nhập vào các đoạn có mức đặc quyền cao hơn thông qua một cổng giao dịch là cổng gọi.

Cổng giao dịch có cấu trúc như sau :

Dự phòng cho hệ 32 bit				2 byte
Quyền truy nhập				1 byte
0	0	0	Bộ đếm	1 byte
Bộ chọn đoạn				2 byte
Địa chỉ offset bắt đầu chương trình				2 byte

Bộ đếm (WC - word count) : xác định số từ cần sao chép từ ngăn xếp của chương trình gọi sang chương trình được gọi. Thông số WC chỉ có ở cổng giao dịch kiểu gọi (cổng gọi).

Byte quyền truy nhập trong Bộ mô tả cổng giao dịch (cổng giao dịch) :

P	DPL	0	Kiểu cổng giao dịch

Kiểu cổng giao dịch: có 4 loại cổng giao dịch.

Kiểu = 4 : cổng giao dịch kiểu gọi (cổng gọi)

Kiểu = 5 : cổng giao dịch kiểu nhiệm vụ (cổng nhiệm vụ)

Kiểu = 6 : cổng giao dịch kiểu ngắt (cổng ngắt)

Kiểu = 7 : cổng giao dịch kiểu bẫy (cổng bẫy)

Bộ mô tả cổng giao dịch kiểu gọi (cổng gọi) thường được dùng để chương trình nguồn có mức đặc quyền thấp hơn gọi chương trình đích có mức đặc quyền cao hơn .

Bộ mô tả cổng giao dịch kiểu nhiệm vụ (cổng nhiệm vụ) được sử dụng khi có sự thay đổi nhiệm vụ trong nhiệm vụ hiện hành. Bộ mô tả cửa giao dịch kiểu nhiệm vụ quy chiếu (trỏ tới) bảng TSS.

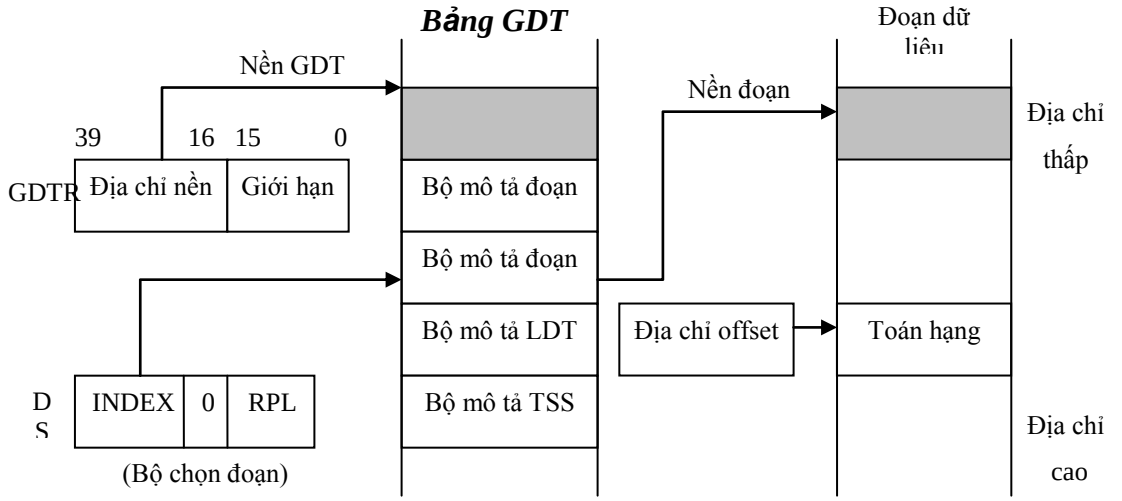
Bộ mô tả cổng giao dịch kiểu ngắt và kiểu bẫy (cổng ngắt và cổng bẫy) cung cấp bộ chọn và địa chỉ offset xác định vị trí của chương trình con phục vụ ngắt bên trong đoạn mã lệnh đó.

d- Lược đồ truy nhập đoạn nhớ nhờ Bộ chọn đoạn và Bộ mô tả đoạn:

Trong chế độ bảo vệ CPU 80286 do bộ chọn đoạn cho khả năng trỏ tới được 2^{13} Bộ mô tả đoạn và mỗi Bộ mô tả trỏ đến một đoạn có kích thước cực đại 2^{16} byte nên CPU có thể quản lý được bộ nhớ kích thước

$$2*2^{13}*2^{16} = 2^{30} = 1\text{Gbyte}$$

e. Cơ chế truy nhập bộ nhớ (ô nhớ) qua bảng GDT



Bảng GDT được hệ điều hành tạo ra khi khởi động hệ thống. CPU quản lý bảng GDT qua thanh ghi GDTR. Thanh ghi GDTR chứa hai thông tin về bảng GDT : địa chỉ nền bảng và kích thước (giới hạn) bảng.

Khi có yêu cầu truy nhập đoạn, người yêu cầu cung cấp Bộ chọn đoạn. CPU thực hiện thao tác kiểm tra quyền truy nhập đoạn trước khi cho truy nhập.

Đối với việc truy nhập đoạn dữ liệu, quá trình kiểm tra được tiến hành theo quy tắc:

$$EPL = \max(CPL, RPL) \leq DPL$$

trong đó:

- CPL là mức đặc quyền của nhiệm vụ đang thực hiện. Thông thường CPL có giá trị bằng mức đặc quyền của đoạn chứa mã lệnh đang chạy. Bộ xử lý trung tâm có thể thay đổi giá trị của CPL khi điều khiển chương trình chuyển đến một đoạn mã có mức đặc quyền cao hơn.
- RPL là mức đặc quyền yêu cầu và là mức đặc quyền của bộ chọn. Mức đặc quyền RPL được sinh ra bởi người nạp bộ chọn đoạn.
  + EPL là mức đặc quyền hiệu dụng.
  + DPL là mức đặc quyền của đoạn bị truy nhập.

Nếu điều kiện trên không được thoả mãn thì sẽ sinh ra một ngoại lệ và CPU không cho truy nhập đoạn. Nếu điều kiện về quyền truy nhập được thoả mãn thì CPU cho truy nhập đoạn. Việc truy nhập từng ô nhớ trong đoạn được thực hiện thông qua địa chỉ nền đoạn (có được từ Bộ mô tả đoạn vừa được chọn) và địa chỉ offset của ô nhớ đó.

Đối với việc truy nhập đoạn mã lệnh, quá trình kiểm tra được tiến hành theo quy tắc:

$$EPL = \max(CPL, RPL) \geq DPL$$

trong đó việc truy nhập một đoạn mã lệnh có mức đặc quyền cao hơn (EPL > DPL) phải thực hiện thông qua cổng gọi.

f. Cơ chế truy nhập bộ nhớ (ô nhớ) qua bảng LDT

LDT được hệ điều hành tạo ra khi nạp một chương trình ứng dụng vào bộ nhớ. Mỗi bảng LDT quản lý các đoạn của một chương trình ứng dụng (không gian nhớ cục bộ). Việc quản lý các đoạn (các vùng nhớ) thuộc một chương trình ứng dụng (một nhiệm vụ) được tổ chức như sau :

Mỗi một đoạn nhớ được quản lý bởi một Bộ mô tả đoạn.

Các Bộ mô tả đoạn của một nhiệm vụ được chứa trong một bảng LDT riêng biệt. Nói cách khác, mỗi bảng LDT quản lý các đoạn nhớ của một nhiệm vụ.

Mỗi bảng LDT được quản lý bởi một Bộ mô tả LDT. Bộ mô tả LDT chứa địa chỉ nền bảng LDT, kích thước bảng, quyền truy nhập bảng (quyền truy nhập nhiệm vụ).

Các Bộ mô tả LDT của các nhiệm vụ được chứa trong bảng GDT. Bảng GDT được quản lý bởi thanh ghi hệ thống GDTR.

Khi một nhiệm vụ được thực hiện, hệ điều hành sẽ nạp Bộ chọn LDT vào thanh ghi hệ thống LDTR. Thanh ghi LDTR trỏ đến Bộ mô tả LDT trong bảng GDT, từ đây CPU thông qua bảng LDT quản lý được các đoạn của nhiệm vụ đó và bắt đầu (hoặc tiếp tục) thực hiện nhiệm vụ này. Để truy nhập các đoạn trong nhiệm vụ, người yêu cầu cần nạp Bộ chọn đoạn vào thanh ghi đoạn tương ứng.

CPU thực hiện thao tác kiểm tra quyền truy nhập đoạn. Nếu điều kiện về quyền truy nhập được thoả mãn thì CPU cho truy nhập đoạn. Nếu điều kiện trên không được thoả mãn thì sẽ sinh ra một ngoại lệ và CPU không cho truy nhập đoạn. Việc truy nhập từng ô nhớ trong đoạn được thực hiện thông qua địa chỉ nền đoạn (có được từ Bộ mô tả đoạn vừa được chọn) và địa chỉ offset của ô nhớ đó.

g- Cơ chế chuyển điều khiển và gọi chương trình con trong chế độ bảo vệ

Việc chuyển điều khiển xẩy ra khi thực hiện các lệnh nhẩy (lệnh JMP) hoặc lệnh gọi chương trình con (lệnh CALL).

Trường hợp thực hiện lệnh nhẩy hoặc lệnh gọi *trong cùng đoạn mã lệnh* của nhiệm vụ đang chạy (*lệnh nhẩy gần, lệnh gọi gần*) xẩy ra như sau :

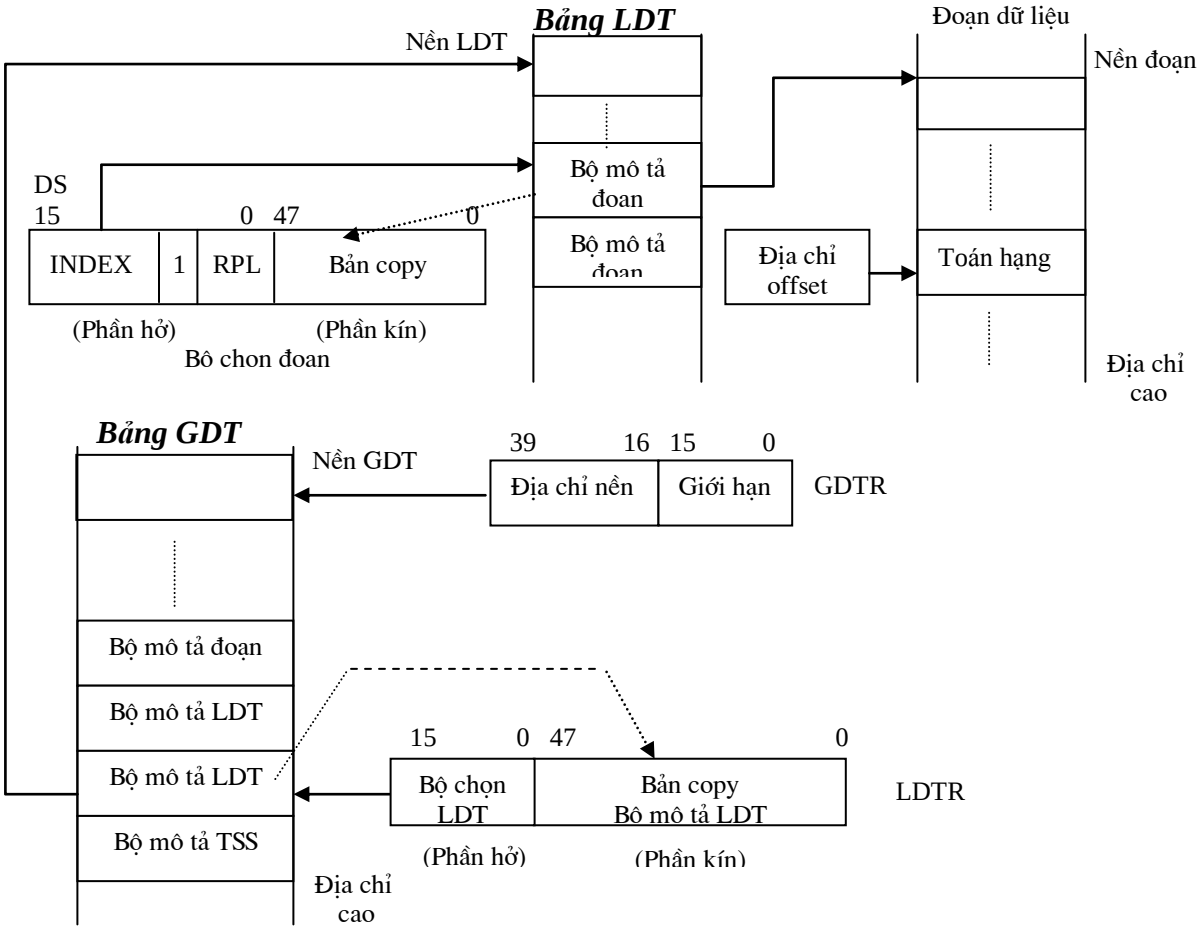
Khi thực hiện lệnh nhẩy gần (near jump), con trỏ lệnh IP được nạp giá trị mới. Chương trình tiếp tục được thực hiện từ vị trí mới do IP trỏ đến.

Khi thực hiện lệnh gọi gần (near call), bộ xử lý trung tâm thực hiện các thao tác sau :

- Cất giá trị hiện thời của IP vào ngăn xếp
- Nạp địa chỉ offset của chương trình con được gọi (đích) vào IP
- Thực hiện chương trình con (đích).

Trường hợp chuyển điều khiển đến những đoạn mã lệnh khác khi thực hiện lệnh gọi xa (FAR CALL) thì có hai tình huống :

a) Đoạn mã lệnh đích có mức đặc quyền thấp hơn hoặc bằng mức đặc quyền của đoạn mã lệnh nguồn hiện tại. Khi đó bộ xử lý trung tâm thực hiện các thao tác sau :



- Cất giá trị hiện thời của CS và IP vào ngăn xếp
- Nạp bộ chọn đoạn mã lệnh chứa chương trình con (đích) vào CS
- Nạp địa chỉ offset của chương trình con (đích) vào IP
- Thực hiện chương trình đích

Lệnh RET cho phép rời khỏi chương trình con để trở về chương trình gọi nó. Lệnh này khôi phục lại nội dung bộ chọn đoạn mã lệnh nguồn (nội dung thanh ghi CS), nội dung con trỏ lệnh (nội dung thanh ghi IP) và tiếp tục thực hiện chương trình đã gọi chương trình con này.

b) Đoạn mã lệnh chương trình con (đích) có mức đặc quyền cao hơn mức đặc quyền của đoạn mã lệnh nguồn hiện tại. Khi đó việc gọi chương trình con (đích) phải thực hiện qua cổng gọi. Bộ chọn đoạn lúc này không trỏ đến bộ mô tả đoạn mã lệnh chứa chương trình con (đích), mà trỏ đến cổng gọi (bộ mô tả cổng gọi). Cổng gọi trỏ đến bộ mô tả đoạn mã lệnh của chương trình con (đích) và chứa địa chỉ offset bắt đầu chương trình con

(đích), qua đó gọi được chương trình con (đích). Bộ xử lý trung tâm thực hiện quá trình này như sau:

- Tạm lưu giữ nội dung CS, IP, SS, SP hiện thời (thuộc chương trình nguồn)
- Nạp bộ chọn cổng gọi và kiểm tra quyền truy nhập
- Cất giữ giá trị tạm lưu của SS và SP nguồn vào ngăn xếp đích
- Chuyển các tham số từ ngăn xếp nguồn sang ngăn xếp đích
- Cất giữ giá trị tạm lưu của CS và IP nguồn vào ngăn xếp đích
- Nạp bộ chọn bộ mô tả đoạn mã lệnh đích và địa chỉ offset (lấy từ cổng gọi), qua đó nạp bộ mô tả đoạn mã lệnh đích
- Thực hiện chương trình con (đích)

Khi bộ xử lý trung tâm gặp lệnh RET thì việc trở về chương trình nguồn được thực hiện bắt đầu bằng việc kiểm tra quyền truy nhập, sau đó là khôi phục nội dung các thanh ghi CS, IP, SS, SP theo một trình tự ngược lại.

2.3. Cơ chế hoạt động đa nhiệm

Nhiệm vụ được định nghĩa như là sự thực hiện một chương trình nào đó. Mỗi một nhiệm vụ có một đoạn trạng thái nhiệm vụ (đoạn TSS – Task State Segment) chứa toàn bộ trạng thái của nhiệm vụ đó. Mỗi đoạn TSS được quản lý (trỏ) bởi một Bộ mô tả TSS nằm trong bảng GDT.

CPU x86 có phần cứng hỗ trợ thao tác chuyển nhiệm vụ. Thao tác chuyển nhiệm vụ thực hiện lưu và bảo vệ toàn bộ trạng thái hoạt động của nhiệm vụ đang thực hiện (bao gồm nội dung toàn bộ các thanh ghi của CPU, không gian địa chỉ có liên quan và Bộ chọn LDT của nhiệm vụ đang chạy) vào đoạn TSS, sau đó nạp trạng thái của nhiệm vụ tiếp theo từ đoạn TSS tương ứng vào CPU, kiểm tra quyền truy nhập và bắt đầu thực hiện nhiệm vụ mới. Thanh ghi nhiệm vụ TR (Task Register) trỏ đến Bộ mô tả TSS quản lý nhiệm vụ hiện thời.

Thao tác chuyển nhiệm vụ được tiến hành theo các bước sau :

- Lưu toàn bộ trạng thái hoạt động của nhiệm vụ đang thực hiện (bao gồm nội dung toàn bộ các thanh ghi của CPU, các địa chỉ có liên quan và Bộ chọn LDT của nhiệm vụ hiện thời) vào đoạn trạng thái nhiệm vụ TSS của nhiệm vụ này.
- Nạp Bộ chọn nhiệm vụ tiếp theo vào thanh ghi TR. Thanh ghi TR trỏ đến Bộ mô tả TSS quản lý đoạn TSS của nhiệm vụ tiếp theo.
- Qua Bộ mô tả TSS truy nhập đoạn TSS của nhiệm vụ tiếp theo, nạp trạng thái nhiệm vụ tiếp theo vào các thanh ghi của CPU, trong

đó có thanh ghi LDTR. Bộ mô tả TSS được nạp vào phần kín của TR.

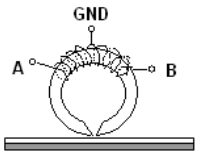
- Thực hiện kiểm tra quyền truy nhập.
- Thực hiện nhiệm vụ tiếp theo.

3. Bộ nhớ ngoài của máy tính

Bộ nhớ ngoài của máy tính là một thiết bị vô cùng quan trọng và không thể thiếu đối với các máy tính hiện nay. Chúng được dùng để lưu giữ các chương trình và dữ liệu không được sử dụng ngay trong quá trình hoạt động của máy, các nội dung được lưu giữ không bị mất khi không có điện hoặc khi tắ máy. Đĩa từ được đưa vào sử dụng từ bắt đầu những năm 1970, và hiện nay, đang được sử dụng rộng rãi với những ưu điểm vượt trội so với các thiết bị nhớ ngoài khác nhờ dung lượng nhớ rất lớn và tốc độ truy xuất rất nhanh. Đĩa quang (đĩa CD) cũng được sử dụng nhiều nhờ dung lượng lớn, bảo quản dễ dàng và có độ tin cậy cao. Phương hướng phát triển chủ yếu đối với các loại thiết bị này luôn luôn nhằm vào khả năng nâng cao dung lượng và tốc độ truy xuất. Sự xuất hiện gần đây của các thiết bị nhớ Flash cũng đang là hướng nghiên cứu được coi trọng trong việc phát triển các thiết bị nhớ dễ dàng bảo quản và di chuyển, dễ kết nối với máy tính để sử dụng khi cần thiết.

3.1. Đĩa từ

Nguyên lý được sử dụng để lưu giữ và đọc dữ liệu trên ổ đĩa từ là ứng dụng tính chất nhiễm từ, duy trì từ tính của vật liệu sắt từ có độ thẩm từ cao. Đầu từ để Ghi/Đọc có nguyên lý cấu tạo như một nam châm điện có độ thẩm từ cao, nhưng không có tính duy trì từ tính. Dòng điện đi qua cuộn dây AB có cường độ tương ứng với giá trị của bit thông tin cần ghi, tạo ra một từ trường trong lõi hình khuyên. Qua khe hở, từ thông đi xuyên đến lớp sắt từ phủ trên mặt đĩa và sắp xếp (hướng từ hoá) các phần tử có khả năng nhiễm từ và duy trì từ tính. Do dòng điện trong cuộn dây thay đổi theo quy luật cần ghi, nên các phần tử nhiễm từ cũng được sắp xếp theo quy luật tương ứng. Hay nói cách khác: *Từ trường dọc theo đường ghi thay đổi theo quy luật của dòng điện mang thông tin đi qua cuộn dây AB*.



Do khả năng duy trì từ tính, thông tin được ghi lên mặt đĩa sẽ được lưu giữ lại.

Ngược lại với quá trình ghi, khi đọc, sự thay đổi chiều sắp xếp của các phần tử đã bị nhiễm từ (do quá trình ghi) sẽ tạo nên sự thay đổi từ thông trong lõi, điện áp do cảm ứng sinh ra trong cuộn dây AB sẽ được xử lý và

biến đổi thành các thông tin tương ứng. Các thông tin không bị xoá trong quá trình đọc.

3.2. Đĩa quang

Đĩa quang (đĩa CD – Compact Disc) là thiết bị lưu giữ dữ liệu làm việc theo nguyên lý biến đổi Quang-Điện. Các "hốc" được tạo ra trên một mặt đĩa sẽ phản xạ một lượng năng lượng trở lại đầu đọc, năng lượng quang học được biến thành điện năng và biến đổi thành các thông tin tương ứng.

Để ghi thông tin lên đĩa, ta dùng tia lazer tạo thành các "hốc" có đường kính cực kỳ nhỏ (thường là hàng chục đến hàng trăm nanometre), gọi là pit. Vùng xung quanh hốc bị đốt nóng, tạo ra khả năng phản xạ quang học khác nhau, gọi là land. Thông tin trên đĩa CD được ghi theo một đường xuắn ốc duy nhất, và được ghi theo từng khối (block), mỗi khối có độ lớn là 2KB dữ liệu. Tuy cách ghi phức tạp nhưng do độ tin cậy cao của phương thức lưu giữ thông tin, nên đĩa CD dược sử dụng rất rộng rãi.

3.3. Bộ nhớ Flash

Bộ nhớ flash là một loại bộ nhớ máy tính không khả biến có thể xóa và ghi lại bằng điện. Đây là công nghệ đã được sử dụng trong các thẻ nhớ, ổ USB flash để lưu trữ và truyền dữ liệu giữa các máy tính và các thiết bị kĩ thuật số khác. Không như EEPROM, nó được xóa và ghi lại theo khối gồm nhiều vị trí (ban đầu bộ nhớ flash chỉ có thể xóa toàn bộ). Bộ nhớ flash rẻ hơn nhiều so với EEPROM. Bộ nhớ flash được sử dụng trong máy tính xách tay, máy nghe nhạc kĩ thuật số, máy ảnh kĩ thuật số và điện thoại di động. Nó cũng được sử dụng trên các máy trò chơi, thay thế cho EEPROM hoặc RAM tĩnh nuôi bằng pin để lưu dữ liệu của trò chơi.

Ổ USB Flash, ổ cứng di động USB, ổ cứng flash USB (còn được gọi sai là cái USB) là thiết bị lưu trữ dữ liệu sử dụng bộ nhớ flash tích hợp với giao tiếp USB (Universal Serial Bus). Chúng có kích thước nhỏ, nhẹ, có thể tháo lắp và ghi lại được. Dung lượng của các ổ USB flash trên thị trường có thể từ hàng trăm đến gàng nghìn MB trở lên.
"Ổ USB" là loại thiết bị nhớ không mất dữ liệu khi ngừng cung cấp điện.

So sánh các bộ nhớ bán dẫn

Loại	Mất dữ liệu khi mất điện?	Khả năng ghi ?	Cỡ xoá ?	Xoá nhiều lần ?	Tốc độ ?	Giá thành (theo byte)
SRAM	Có	Có	Byte	Không giới hạn	Nhanh	Đắt
DRAM	Có	Có	Byte	Không giới	Vừa phải	Vừa phải

				hạn		
Masked ROM	Không	Không	Không sẵn sàng	Không sẵn sàng	Nhanh	Không đắt
PROM	Không	Một lần, yêu cầu thiết bị chuyên dụng	Không sẵn sàng	Không sẵn sàng	Nhanh	Vừa phải
EPROM	Không	Có, nhưng cần thiết bị chuyên dụng	Toàn bộ	Giới hạn	Nhanh	Vừa phải
EEPROM	Không	Có	Byte	Giới hạn	Nhanh cho đọc, chậm cho xoá và ghi	Đắt
Flash	Không	Có	Sector	Giới hạn	Nhanh cho đọc, chậm cho xoá/ghi	Vừa phải
NVRAM	Không	Có	Byte	Không giới hạn	Nhanh	

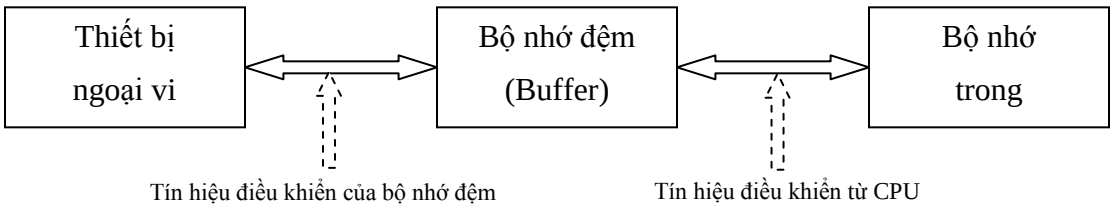
Chương VII. Thiết bị ngoại vi của máy tính

Thiết bị vào/ra của máy tính là một phần rất quan trọng trong khối các thiết bị ngoại vi. Thiết bị ngoại vi (Peripherals) bao gồm các thiết bị, từ các vi mạch phụ trợ như USART, PPI, PIT, v.v…, cho đến các hệ thống hoàn chỉnh như ổ đĩa mềm, ở đĩa cứng, bàn phím, màn hình, máy in, modem, v.v… Thiết bị ngoại vi liên lạc và trao đổi thông tin với CPU thông qua các cổng vào/ra (I/O port) và thường tạo cho hệ thống máy tính những khả năng logic phụ, hoặc là nối ghép máy tính với những đối tượng phi điện tử khác. Chức năng của thiết bị ngoại vi có thể được thực hiện bằng phần mềm, hoặc kết hợp giữa phần mềm và phần cứng. Các chức năng và yêu cầu cơ bản đối với thiết bị ngoại vi có thể liệt kê như sau:

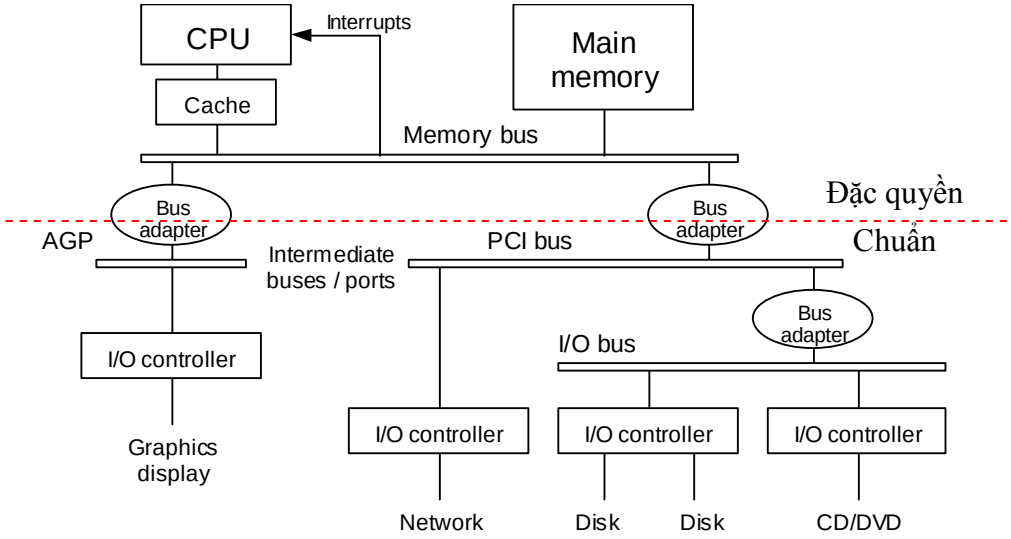
- Là phương tiện giao tiếp, trao đổi thông tin giữa con người và máy tính
- Làm bộ nhớ trung gian
- Phải thuận lợi nhất cho người dùng
- Tốc độ vào/ra phải cao
- Lỗi phải được xử lý dễ dàng.

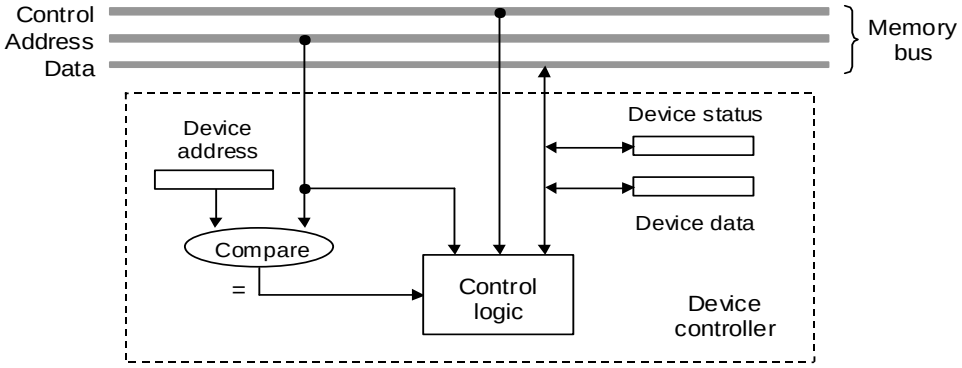
Thiết bị ngoại vi thường phải sử dụng các nguyên lý kết hợp điện-cơkhí nên thường hoạt động rất chậm. Để nâng cao hiệu suất làm việc của CPU và của máy tính, ta thường dùng bộ nhớ đệm – Buffer. Trao đổi dữ liệu giữa thiết bị ngoại vi và CPU được tiến hành theo 2 bước:

1. Trao đổi dữ liệu giữa CPU và bộ nhớ đệm, thực hiện nhờ các lệnh của CPU
2. Trao đổi dữ liệu giữa bộ nhớ đệm và thiết bị ngoại vi. Tốc độ trao đổi được quyết dịnh bởi tốc độ xử lý của thiết bị ngoại vi. Quá trình trao đổi do bộ nhớ đệm điều khiển thông qua các tín hiệu điều khiển từ CPU.



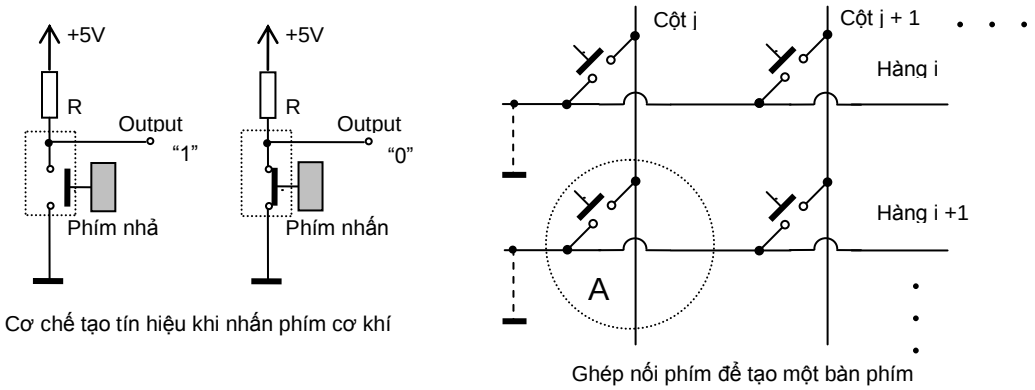
Trong toàn bộ khoảng thời gian của bước 2, bộ nhớ trong không liên kết trực tiếp với bộ nhớ đệm, do vậy, CPU vẫn có thể thực hiện các chương trình khác, hoặc trao đổi dữ liệu với các thiết bị ngoại vi khác. Cơ chế làm việc này đòi hỏi bộ nhớ đệm phải có thông tin trạng thái (thông thường là báo trạng thái EMPTY (rỗng) hay FULL (đầy), để cho CPU biết và thực hiện các lệnh trao đổi dữ liệu với bộ nhớ đệm.



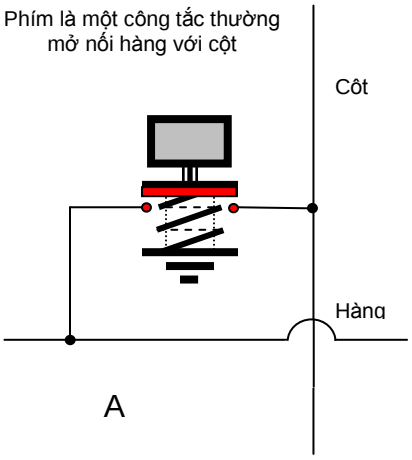


Nguyễn Trung Đồng – Tel 0983 410 866

1. Bàn phím Hex Keyboard



Hình V.1. Phím tiếp xúc và cách tạo bàn phím



Bàn phím được tổ chức theo kiểu ma trận các hàng và các cột, tại vị trí giao nhau không tiếp xúc được ghép một công tắc thường mở nối hàng với cột, chỉ tiếp xúc khi được nhấn. Để xác định có một phím bị nhấn, ta nối đất tất cả các hàng và đọc nội dung các cột. Nếu trên cột nào đó ta đọc được giá trị là "0", tương ứng với trường hợp có một phím trên cột đó bị nhấn. Dễ dàng thấy rằng, nếu các hàng i và i + 1 nối đất, bất cứ phím nào trên cột j (hay j + 1) bị nhấn, ta đều đọc được giá trị "0" trên cột j (hay j + 1).

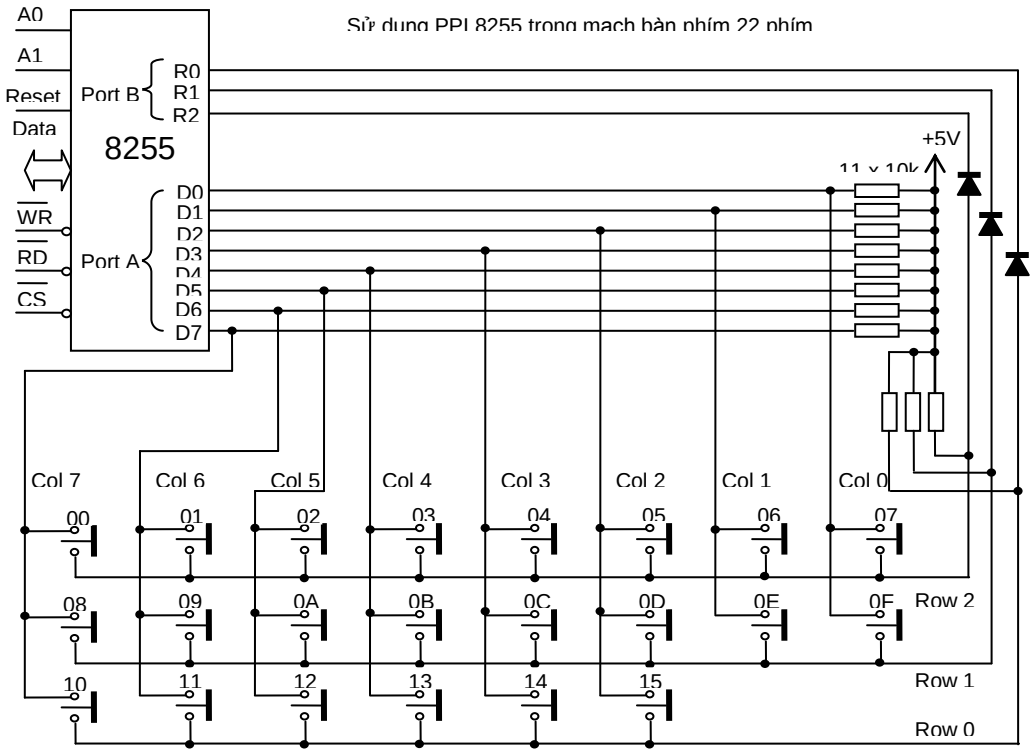
Hình V.2 là một bàn phímH gồm 22 phím được tạo từ một ma trận 3 hàng và 8 cột. Giả sử rằng ta dùng vi mạch vào ra song song PPI-8255 để xây dựng nên bàn phím như trên Hình V.2. Ba lối ra của port B gồm R0, R1, R2 (tương ứng với các dây PB0, PB1 và PB2) được dùng ở chế độ Output, 8 lối vào của port A dùng D0 ÷ D7 (tương ứng với các dây PA0 ÷ PA7) ở chế độ Input. Như vậy chu trình đọc phím theo chế độ dò tìm (polling) được thực hiện như sau:

1. Để đảm bảo phím nhấn trước đó đã được nhả ra, các giá trị "0" cùng lúc được áp lên tất cả các hàng và đọc các giá trị trên các cột. Nếu các cột đều ở mức "1", chương trình tiếp tục đọc giá trị các cột

2. Quét các cột, tức là đọc giá trị tại các cột để phát hiện có phím bị nhấn. Để tăng độ tin cậy khi đọc phím, tránh tác động của nhiễu cơ học khi phím bị nhấn và các loại nhiễu khác, sau khi phát hiện có phím bị nhấn, chương trình chờ khoảng 20msec rồi đọc tiếp giá trị tại các cột. Giá trị "0" đọc dược ở cột nào sẽ được ghi nhớ để sử dụng cho việc xác định phím ở vị trí nào bị nhấn

3. Quét hàng để xác định vị trí của phím bị nhấn. Số vòng lặp này là không cố định, nhưng nhiều nhất là bằng số hàng có trong cấu trúc của bàn phím

4. Gán mã cho phím. Mã cho phím là do thiết kế phần cứng quy định, tuỳ theo chức năng và yêu cầu của người dùng.



Hình V.2 – Bàn phím 22 phím sử dụng giao tiếp qua PPI8255

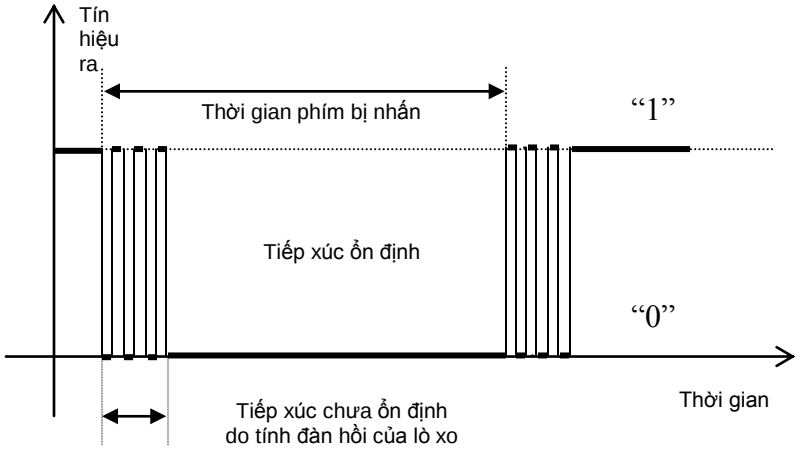
Trong ví dụ này giả sử rằng các phím được gán mã như sau:

- Từ phím 00 đến phím 0F (toàn bộ các phím trong Row 1 và Row 2) được gán mãH từ "0_H" đến "F_H"
- Các phím ở Row 0 có thể gán các chức năng sau:
  + Phím 10 là phím chức năng "GO" - thực hiện chương trình

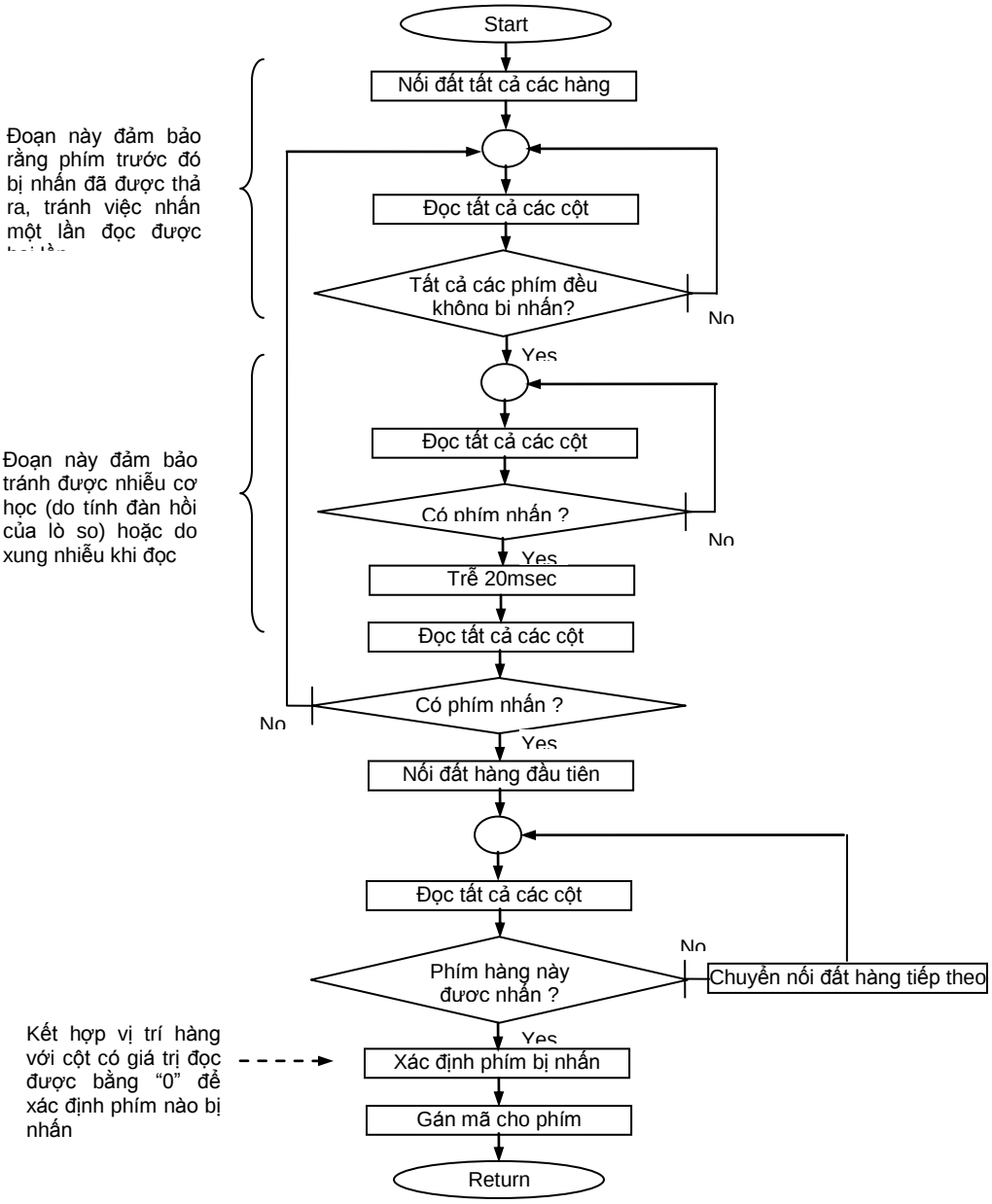
+ Phím 11 là phím chức năng "INS" - thực hiện chức năng thay đổi nội dung các thanh ghi của CPU
+ Phím 12 là phím "REP" - thực hiện chức năng sửa nội dung thanh ghi của CPU
+ Phím 13 là phím "DISP" - thực hiện chức năng hiển thị nội dung các thanh ghi của CPU
+ Phím 14 là phím "STEP" - thực hiện chức năng chạy chương trình theo từng lệnh
+ Phím 14 là phím "ENTER" - thực hiện chức năng kết thúc nhập dữ liệu hoặc lệnh từ bàn phím

Lưu đồ chương trình đọc và xác định phím bị nhấn được thể hiện trên Hình V.3 Chương trình có thể được viết dưới dạng một chương trình con.

Do tính đàn hồi của lò xo trong phím nên sự tiếp xúc của phím sau khi bị nhấn có thể mô tả như hình sau:



Nếu không phím nào bị nhấn, dữ liệu đọc vào từ Port A của PPI8255 có giá trị là 1 byte nhị phân 11111111 ứng với các bit từ D_0 đến D_7. Đưa giá trị 000 nhị phân ra các bit tương ứng R_0 R_1 và R_2 của Port B, khi có một phím bất kỳ bị nhấn, giả sử đó là phím được gán ký hiệu 05, ta sẽ đọc được từ Port A một byte nhị phân có giá trị tương ứng là 1101111. Giá trị này cũng sẽ đọc được từ Port A của PPI8255 nếu như phím được gán ký hiệu là 0D hoặc phím được gán ký hiệu 15 bị nhấn. Dùng phương pháp "quét" – scan bằng cách đưa theo thứ tự giá trị "0" ra bit R_0, R_1 hoặc R_2 cuae Port B, đọc lại giá trị vào từ Port A ta sẽ xác định được phím nào vừa bị nhấn.



Hình V.3 Lưu đồ chương trình đọc bàn phìm

2. Ghép nối bàn phím với máy tính

Bàn phím là thiết bị ngoại vi cho phép đưa thông tin vào máy tính dưới dạng mã ký tự. Bàn phím thực hiện chức năng chuyển thông tin dạng lực nhấn phím và vị trí của phím được nhấn thành mã phím và chuyển cho máy tính. Bàn phím gồm hai bộ phận chính là ma trận phím và mạch điện tử quét phím. Ma trận phím là tổ hợp các phím nhấn được sắp xếp theo các hàng và cột.

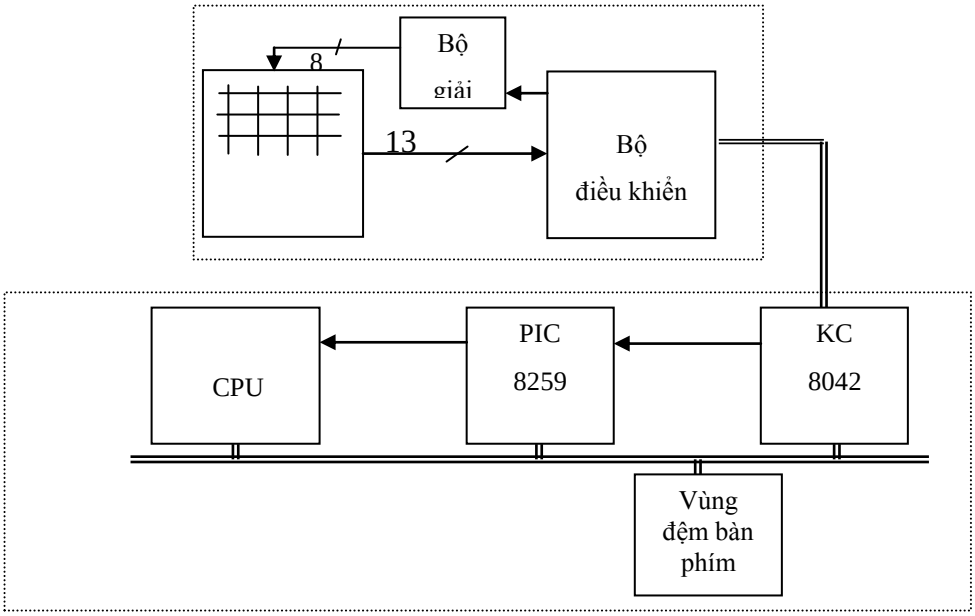
Bình thường phím luôn ở trạng thái nhả, khi phím nhả thì hai tiếp điểm không được nối với nhau, đầu ra có mức điện áp dương tương ứng với mức logic "1". Khi phím được nhấn thì hai tiếp điểm được nối với nhau qua công tắc phím và đầu ra có mức điện áp bằng 0V tương ứng mức logic "0".

Để mỗi lần nhấn phím có một mã phím tương ứng được tạo ra, cần sắp xếp hệ thống phím dưới dạng ma trận phím.

Ma trận phím gồm các dây hàng và các dây cột giao nhau nhưng không tiếp xúc với nhau. Các công tắc phím được đặt ở chỗ giao của hàng và cột. Hai tiếp điểm của công tắc nằm ở trên hàng và cột tại chỗ giao nhau đó. Mỗi khi phím được nhấn thì hai dây hàng và cột được nối với nhau qua hai tiếp điểm của công tắc tại chỗ giao nhau .

2.1. Hệ thống bàn phím của máy vi tính

Hệ thống bàn phím của máy vi tính gồm hai phần bàn phím và thiết bị giao diện bàn phím, được kết nối và trao đổi thông tin theo kiểu "chủ" "thợ".



Hình V.4 – Sơ đồ ghép nối bàn phím (keyboard) với hệ thống máy tính

Bàn phím là tổ hợp của ma trận 8x13 phím và mạch vi điều khiển µP8048. Mạch µC8048 là một hệ vi xử lý nhỏ được tích hợp trên một đơn chip. Mạch 8048 bao gồm CPU, bộ nhớ ROM chứa chương trình điều khiển quét và tạo mã phím, RAM chứa dữ liệu của chương trình điều khiển, hai cổng vào/ra P1và P2, một cổng dữ liệu 8 bit. Mạch 8048 tuần tự đưa mã nhị phân 3 bit ra tại cổng P2, qua bộ giải mã 3/8 tạo ra tín hiệu quét bàn phím. Tại thời điểm mã 3 bit được đưa ra, mạch µP8048 thực hiện đọc tín hiệu 13 bit từ ma trận phím vào cổng P1, từ đây tạo ra mã phím (mã quét) của phím được nhấn. Khi phím được nhả một mã phím (mã quét) cũng được tạo ra bằng cách cộng mã phím nhấn với 80H.

Mạch µP8048, được nuôi bằng nguồn từ máy tính, thực hiện trao đổi thông tin với thiết bị giao diện bàn phím KC 8042 theo kiểu nối tiếp đồng bộ. KC 8042 có cấu trúc tương tự mạch µP8048. KC 8042 đóng vai trò "chủ", 8048 đóng vai trò "thợ" trong các quá trình truyền tin thông qua hai dây tín hiệu: dây "DATA" và dây "CLOCK".

Dây "DATA" truyền tín hiệu dữ liệu nối tiếp giữa µP8048 và KC 8042 . Tín hiệu nối tiếp bao gồm: bit START, 8 bit dữ liệu, 1 bit PARITY, 1 bit STOP. Quá trình trao đổi thông tin giữa µP8048 và KC 8042 được đồng bộ bởi tín hiệu trên dây "CLOCK".

2.2. Quá trình truyền dữ liệu từ bàn phím cho CPU

Mạch µP8048 luôn phải kiểm tra trạng thái truyền tin qua hai dây "DATA" và "CLOCK" trước khi phát đi mã phím . Khi KC 8042 đặt "DATA" = 0 và "CLOCK"=1 thì 8048 phải nhận các chỉ lệnh từ KC 8042. Khi KC 8042 đặt "DATA " = 1 và "CLOCK" = 1 thì µP8048 được quyền truyền mã phím cho máy tính. Quá trình truyền dữ liệu được đồng bộ bằng dãy xung đồng bộ do µP8048 phát ra trên dây "CLOCK".

Khi KC 8042 nhận được mã phím dạng nối tiếp, nó loại bỏ các bit tạo khung dữ liệu truyền, chuyển mã phím vào thanh ghi tạm và phát ra yêu cầu ngắt IRQ1 cho hệ thống ngắt cứng. Hệ thống ngắt cứng sẽ kích hoạt chương trình phục vụ bàn phím 09H (chương trình phục vụ ngắt 09H) nằm ở BIOS. Chương trình phục vụ bàn phím 09H có chức năng dịch mã phím thành mã hai byte và chứa vào vùng đệm bàn phím.

Chương trình phục vụ bàn phím 09H trước hết kiểm tra (mã) các phím trượt (Shift, Alt, Ctrl) và các phím đặc biệt (ScrollLock, NumLock , CapsLock, Insert) trước khi dịch mã phím sang mã hai byte.

Mã hai byte được chương trình phục vụ bàn phím 09H tạo ra có cấu trúc tuỳ thuộc mã phím hoặc tổ hợp mã phím nhận được. Nếu nhận được mã của phím ký tự thì byte thấp của mã hai byte chứa mã ASCII của ký tự tương ứng, byte cao chứa mã phím (mã quét phím). Khi chương trình phục vụ bàn phím 09H nhận được mã các phím không phải là ký tự thì byte thấp của mã hai byte có giá trị 0, byte cao chứa mã phím mở rộng.

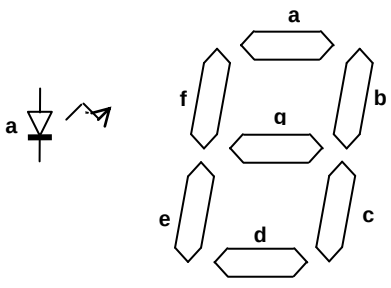
Vùng đệm bàn phím có kích thước 32 byte nằm trên bộ nhớ chính tại địa chỉ 0000H:041EH . Trạng thái của các phím trượt và các phím đặc biệt được chứa ở hai ô nhớ 0000H:0417H và 0000H:0418H . Có thể truy nhập vùng đệm bàn phím để đọc thông tin về bàn phím nhờ chương trình ngắt 16H của BIOS.

Chương trình phục vụ bàn phím 09H cũng xử lý các trường hợp đặc biệt như:

- Khi phím được nhấn quá lâu (ví dụ quá 0.5 giây) và KC 8042 không nhận được mã phím nhả, nó sẽ gửi ra cho đơn vị xử lý trung tâm mã của phím được nhấn.
- Khi nhận được tổ hợp các phím Ctrl+Alt+Del nó sẽ khởi động lại máy tính.
- Khi nhận được mã phím PrintScreen nó sẽ kích hoạt ngắt 05H của BIOS.
- Khi nhận được mã phím Ctrl+Break nó sẽ kích hoạt ngắt 1BH của BIOS.

3. Mạch điều khiển và lập trình chỉ thị 7-segments

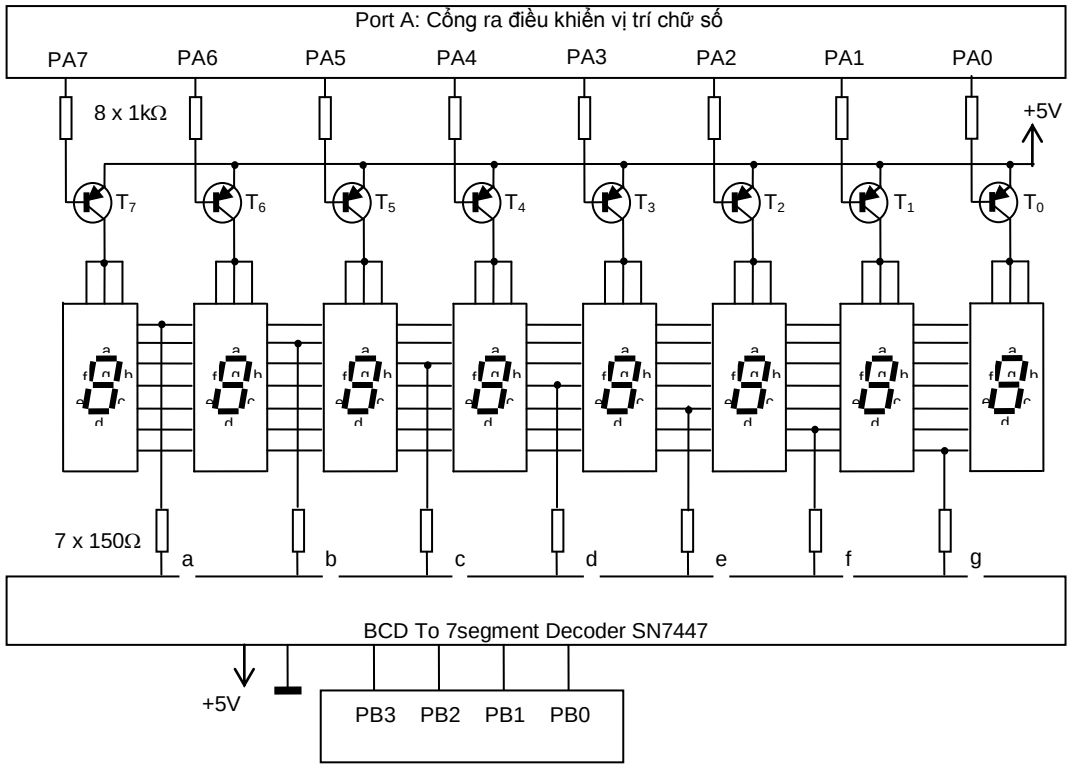
Hiển thị 7 thanh (7-segment Light Emitting Diode – LED Display) là loại đơn giản nhất nhận tín hiệu ra và hiển thị dưới dạng phát sáng. Có thể sử dụng vi mạch này để hiển thị các ký tự số từ 0 đến 9. Khi có dòng điện chạy qua, diode sẽ phát sáng.



	a	b	c	d	e	f	g
0	1	1	1	1	1	1	0
1	0	1	1	0	0	0	0
2	1	1	0	1	1	0	1
3	1	1	1	1	0	0	1
4	0	1	1	0	0	1	1
5	1	0	1	1	0	1	1
6	1	0	1	1	1	1	1
7	1	1	1	0	0	0	0
8	1	1	1	1	1	1	1
9	1	1	1	0	0	1	1
A	1	1	1	0	1	1	1
B	0	0	1	1	1	1	1
C	1	0	0	1	1	1	0
D	0	1	1	1	1	0	1
E	1	0	0	1	1	1	1
F	1	0	0	0	1	1	1

Hình V.5 là sơ đồ mạch hiển thị 8 digits sử dụng các vi mạch hiển thị 7 segment sử dụng 2 cổng của PPI-8255 theo phương pháp điều khiển hiển thị đa công (Multiplexing) đồng bộ. Các thanh sáng a, b, c, …, g của các mạch hiển thị 7 thanh được nối song song với nhau và nối với đầu ra của giải mã BCD-7segment SN7447. Việc cấp nguồn nuôi cho mạch hiển thị (1 digit) được đóng ngắt bởi một transistor PNP làm việc ở chế độ khoá đóng mở nhờ xung điều khiển từ một lối ra của cổng A của PPI-8255. Như vậy, tại một thời điểm, bằng cách lập trình cho PPI-8255, ta sẽ điều khiển để duy nhất một mạch hiển thị phát sáng. Nếu tần số của quá trình phát sáng đạt đến khoảng 15 đến 20 lần/sec, không xảy ra hiện tượng nhấp nháy khi theo dõi.

Dữ liệu cần hiển thị ở dạng mã BCD (4-bit) được đưa ra mạch giả mã hiển thị 7 thanh SN7447 qua 4 dây tương ứng của cổng B, đồng thời vị trí của digit cần hiển thị sẽ được điều khiển phát sáng bằng cách đưa điện áp mức "0" lên lối ra tương ứng trên cổng A để làm thông Transistor cấp nguồn cho mạch 7 segment tương ứng. Như vậy bằng cách lập trình "quét" lần lượt vòng qua tất cả các digit, có thể điều khiển hiển thị một dữ liệu gồm tối đa 8 chữ số.



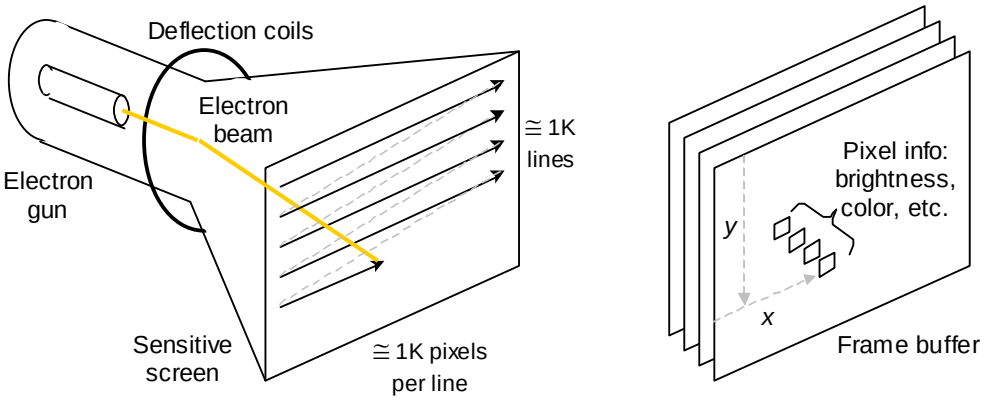
Hình V.5 – Sơ đồ nguyên lý mạch điều khiển bảng hiển thị 8 ký tự số sử dụng PPI 8255 theo phương pháp quét động

4. Màn hình (Monitor)

4.1. Màn hình ống tia âm cực CRT (Cathode Ray Tube)

Màn hình ống tia âm cực CRT là thiết bị hiển thị thông dụng nhất hiện nay. Màn hình CRT có cấu tạo như sau :

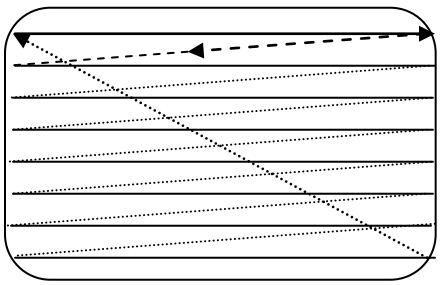
Màn hình CRT là một ống thủy tinh chân không với các bộ phận: cathode phát xạ điện tử, ống phóng tia điện tử, cuộn lái tia và màn hiển thị.



(a) Image formation on a CRT (b) Data defining the image

Hình V.6 – Màn hình CRT

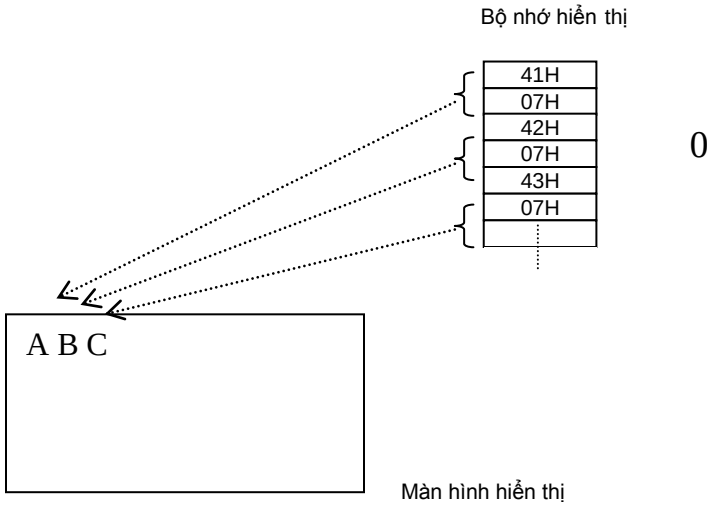
Cathode bằng kim loại được nối với điện áp âm , được đốt nóng và tạo ra các điện tử tự do . Màn hiển thị được phủ một lớp chất liệu phát quang và dẫn điện, được nối với điện áp dương và đóng vai trò một anode. Dưới tác dụng của điện trường cường độ cao trong ống phóng, điện tử rời khỏi cathode, được hội tụ thành chùm tia hướng về phía màn hiển thị. Cuộn lái tia có tác dụng lái chùm tia điện tử dịch chuyển theo hai chiều dọc và ngang màn hình. Khi chùm tia điện tử đập vào màn hiển thị sẽ tạo nên một điểm phát sáng. Cường độ điểm sáng phụ thuộc vào cường độ chùm tia và chất liệu phát sáng. Khi chùm tia mất đi hoặc chuyển hướng thì điểm vẫn còn lưu sáng một khoảng thời gian ngắn sau đó, thời gian lưu sáng phụ thuộc vào chất liệu phát sáng và cường độ chùm tia.



Ảnh trên màn hình CRT được tạo từ các điểm ảnh. Điểm ảnh được tạo ra khi cường độ chùm tia điện tử được tăng lên, điểm ảnh không xuất hiện khi chùm tia bị tắt đi. Các điểm ảnh được tạo theo từng dòng, từ trên xuống dưới . Một ảnh hoàn chỉnh được tạo ra trên màn hiển thị bởi các dòng chứa các điểm ảnh. Các điểm ảnh chỉ tồn tại trong một thời gian rất ngắn. Để có thể quan sát được ảnh cần làm tươi các điểm ảnh theo một chu kỳ xác định. Các điểm ảnh được làm tươi theo từng dòng, bắt đầu từ dòng thứ nhất. Các dòng được làm tươi tuần tự từ trên xuống dưới. Khi dòng cuối cùng được quét xong, quá trình làm tươi được bắt đầu lại từ dòng đầu tiên (hình vẽ).

4.2. Ghép nối màn hình với máy tính

Các thiết bị hiển thị được sử dụng ở máy vi tính PC đều là loại ánh xạ bộ nhớ. Bộ nhớ này được cả đơn vị xử lý trung tâm và thiết bị điều khiển màn hình cùng truy nhập và được gọi là bộ nhớ hiển thị. Thông tin cần hiển thị được đưa ra bộ nhớ hiển thị, thiết bị điều khiển màn hình CRTC liên tục đọc bộ nhớ này để đưa ra màn hình. Hình vẽ sau đây minh họa nguyên tắc ánh xạ từ bộ nhớ hiển thị ra màn hình trong chế độ văn bản :



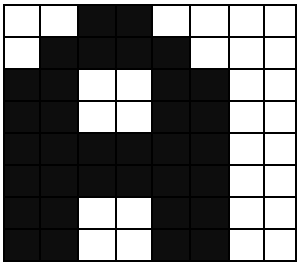
Hình V.7 – Hiển thị ký tự trên màn hình CRT theo nguyên tắc ánh xạ bộ nhớ

Mỗi một ký tự trên màn hình là một ánh xạ của một ô nhớ hai byte trong bộ nhớ hiển thị. Byte đầu chứa mã ASCII của ký tự, byte thứ hai chứa thuộc tính (màu nền, màu chữ, có/không nhấp nháy) của ký tự. Vị trí của mã ký tự trong bộ nhớ xác định vị trí ký tự trên màn hình. Mã ký tự đầu tiên

trong bộ nhớ hiển thị (ví dụ : mã 41H) được ánh xạ thành ký tự (ký tự A) lên góc trái trên của màn hiển thị, mã ký tự tiếp theo được ánh xạ thành ký tự tiếp theo v.v.

Phương pháp ánh xạ bộ nhớ cho phép chương trình máy tính có thể dễ dàng thay đổi nội dung màn hiển thị bằng cách thay đổi nội dung của bộ nhớ hiển thị.

Mỗi ký tự được hiển thị trên màn hình dưới dạng một ma trận 8x8(*) điểm ảnh sáng/tối như trên hình vẽ: *Cũng có những trường hợp sử dụng ma trận 5x7, 7x9, 7x12 và 9x14 điểm*

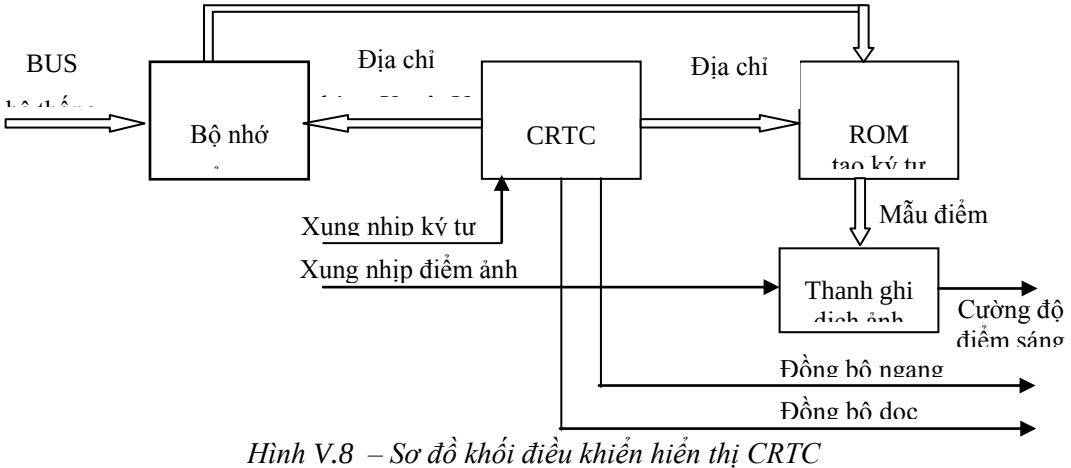


Phương pháp hiển thị ánh xạ bộ nhớ không hoàn toàn phù hợp với việc hiển thị các đối tượng có hình dạng không bình thường và chuyển động nhanh, đáp ứng thời gian thực bị chậm vì cần phải thao tác nhiều điểm ảnh để dịch chuyển đối tượng.

4.3. Bộ điều khiển màn hình CRTC

Thiết bị giao diện màn hình (bộ điều khiển màn hình) CRTC thực hiện việc chuyển mã ký tự trong bộ nhớ hiển thị thành ký tự hiện trên màn hình. ở chế độ văn bản các mẫu ký tự chỉ được hiển thị ở các vị trí hàng và cột cố định (25 hàng x 80 cột).

Sơ đồ nguyên lý của thiết bị giao diện màn hình ở chế độ văn bản như sau:



Hình V.8 – Sơ đồ khối điều khiển hiển thị CRTC

Mỗi một ký tự trên màn hình chứa nhiều hàng điểm ảnh. CRTC có nhiệm vụ chuyển mỗi mã ASCII trong bộ nhớ hiển thị thành chuỗi các mẫu điểm ảnh, đưa mỗi mẫu nằm lên một dòng màn hình. Điều này được thực hiện nhờ bộ ROM tạo ký tự. ROM tạo ký tự chứa các hộp mẫu ký tự, mỗi hộp mẫu ký tự có kích thước 8 byte mang thông tin về ma trận điểm ảnh của một ký tự. Ví dụ hộp mẫu ký tự A có dạng sau :

00110000
01111000
11001100
11001100
11111100
11111100
11001100
11001100

Nếu cần hiển thị 256 ký tự ASCII cần một ROM 2Kbyte, đủ chứa 256 hộp mẫu ký tự, mỗi hộp mẫu chiếm 8 ô nhớ liền nhau. Các hộp mẫu ký tự trong bộ ROM tạo ký tự được định vị bằng địa chỉ 11 bit, trong đó 8 bit địa chỉ cao xác định vị trí của hộp trong ROM, 3 bit địa chỉ thấp xác định vị trí của từng byte mẫu điểm ảnh trong hộp đó. Các mẫu ký tự được đặt trong ROM theo trật tự của bảng mã ASCII.

Nguyên lý hoạt động của thiết bị giao diện màn hình trong chế độ văn bản như sau: Giả sử cần hiển thị hai ký tự A và B tại các vị trí hàng 0 - cột 0 và hàng 0 - cột 1 trên màn hình. Mã ASCII của hai ký tự được đặt tại hai vị trí tương ứng trong bộ nhớ hiển thị.

CRTC gửi địa chỉ hàng và cột màn hình cho bộ nhớ hiển thị (hàng=0, cột=0). Bộ nhớ hiển thị gửi mã ASCII của ký tự (ký tự A) cho ROM, mã ASCII của ký tự mang thông tin về địa chỉ của hộp mẫu ký tự trong ROM (8 bit địa chỉ cao). Tại cùng thời điểm này CRTC gửi địa chỉ của dòng mẫu điểm ảnh (dòng mẫu điểm 0) cho ROM (3 bit địa chỉ thấp). Hai địa chỉ này được kết hợp lại tạo thành địa chỉ (11 bit) cho phép truy nhập vào dòng mẫu điểm ảnh đầu tiên của ký tự (ký tự A) trong ROM và xuất nó ra thanh ghi dịch ảnh. Từ thanh ghi dịch ảnh, từng bit mẫu ảnh tuần tự được đưa ra màn hình.

Khi tất cả các bit mẫu ảnh từ thanh ghi dịch được đẩy ra màn hình, CRTC tiếp tục gửi địa chỉ hàng-cột (hàng=0, cột=1) cho bộ nhớ hiển thị và gửi địa chỉ dòng mẫu điểm ảnh (dòng mẫu điểm 0) cho ROM, bộ nhớ hiển

thị gửi mã ASCII của ký tự (ký tự B) cho ROM. Dòng mẫu điểm ảnh đầu tiên của ký tự (ký tự B) được xuất ra thanh ghi dịch ảnh. Tương tự như thế các dòng mẫu điểm đầu tiên của tất cả các ký tự trên cùng một hàng màn hình được hiển thị, cho đến ký tự cuối cùng trên hàng.

CRTC tiếp tục gửi địa chỉ hàng-cột (hàng=0, cột=0) đến bộ nhớ hiển thị, nhưng địa chỉ dòng mẫu điểm ảnh bây giờ là 1(dòng mẫu điểm 1) cho ROM. Bộ nhớ hiển thị gửi mã ASCII của ký tự A cho ROM, ROM xuất ra dòng mẫu điểm ảnh 1 của ký tự A. Dòng 1 của ký tự B được xuất ra theo cách tương tự. Các dòng điểm ảnh tiếp theo của ký tự lần lượt được hiển thị lên màn hình cho đến khi tất cả các dòng điểm ảnh của hàng văn bản đầu tiên (hàng 0) được hiển thị trên màn hình.

Các hàng văn bản tiếp theo cũng được hiển thị theo phương pháp nói trên.

Trên thực tế hoạt động của CRTC phức tạp hơn. CRTC phải có khả năng hiển thị ở chế độ đồ họa. CRTC phải theo dõi thông tin về thuộc tính của ký tự hiển thị, phải tạo ra điểm nháy. CRTC cũng phải tạo ra hai tín hiệu đồng bộ ảnh ngang - dọc và làm tươi màn hình. Tần số làm tươi tối thiểu là 50 Hz.

Chương VIII. Kỹ thuật và công cụ phát triển phần mềm máy tính

Trong hầu hết các ứng dụng của máy tính, dù là máy Vi tính, máy tính Mini hay các MainFrame, phần mềm đóng vai trò vô cùng quan trọng. Đối tượng của phần mềm là sự đòi hỏi phần cứng thực thi một công việc (ứng dụng) cụ thể. Ứng dụng càng lớn,càng phức tạp, càng hoàn thiện thì phần mềm càng đòi hỏi sự đầu tư cao về trí tuệ và công sức. Do vậy, càng ngày càng đòi hỏi "những công cụ" phát triển phần mềm với những tính năng cao nhất và tiện dụng nhất.

Dĩ nhiên, không thể sử dụng hết những khả năng mà các công cụ phát triển phần mềm tạo ra nếu như không nắm vững các kỹ thuật lập trình thích hợp. Điều này cũng có thể ví như một người sử dụng ô tô, xe máy mà không biết hệ thống điện trong đó hoạt động như thế nào. Khai thác triệt để các tiềm năng của công cụ phát triển phần mềm, ta có thể xây dựng được những chương trình ứng dụng tuyệt vời hơn, quản lý và khai thác các tính năng và tài nguyên trong hệ thống máy tín một cách hữu hiệu hơn nhiều.

Quá trình phát triển một phần mềm từ khi bài toán được xác định cho đến khi một phần mềm được cài dặt và hoạt đọng có thể chia ra thành 5 giai đoạn như sau:

a) ***Đặt vấn đề (xác nhận vấn đề):*** Trước khi giải quyết vấn đề, người lập trình cần xác định xem, liệu vấn đề có thể được giải quyết nhờ một chương trình trong một máy tính hay không. Phải thấy rằng không phải máy tính "vạn năng" đến mức có thể giải quyết tất cả mọi vấn đề nảy sinh trong thực tiễn, thậm chí đôi khi còn làm cho sự việc càng thêm phức tạp.

b) ***Xác định phương pháp giải quyết vấn đề:*** Đây chính là bước tìm thuật giải (Algorithm) tối ưu cho vấn đề được đặt ra. Người lập trình phải tìm và lựa chọn được từ nhiều giải pháp một giải pháp tốt nhất, nhưng kinh tế nhất để thực hiện. Không chỉ tìm giải thuật tốt nhất mà còn phải tìm ngôn ngữ lập trình phù hợp nhất để giải quyết vấn đề.

c) ***Thực hiện giải pháp:*** Phương pháp giải quyết vấn đề thường được xác nhận qua từng bước theo một lưu đồ. Lưu đồ là cách thể hiện tường minh các bước thực hiện chương trình trong hệ thống máy tính, đồng thời nó giúp người lập trình định hướng tốt khi viết chương trình.

d) ***Viết chương trình :*** Bản thân lưu đồ đã cho thấy rõ giải pháp giải quyết vấn đề theo quan điểm lập trình. Việc chuyển từ lưu đồ sang ngôn ngữ chương trình là bước dễ dàng hơn rất nhiều so với cách viết chương trình không có lưu đồ. Đây chỉ là bước cụ thể hóa lưu đồ nhờ tuần tự thực hiện các lệnh, và là bước thực tế hóa giải pháp thực hiện vấn đề.

e) ***Kiểm tra và gỡ rối:*** Sau khi cài đặt, việc kiểm tra tính chính xác là vô cùng quan trọng. Những sai sót phải được phát hiện và hiệu chỉnh, đôi khi là từ chính thuật giải. Việc gỡ rối chương trình tức là thực hiện từng bước chương trình, phát hiện các sai sót ẩn, hiệu chỉnh các sai sót này.

Người lập trình cần tuân thủ năm bước trên khi phát triển một chương trình. Kỹ thuật và các công cụ lập trình được trình bày sau đây là những gợi ý ban đầu. Các khái niệm thuật giải (Algoritnm), lưu đồ (Flowchart) và cấu trúc chương trình (Program Structure) là những khái niệm quan trọng nhất và là cơ sở để phát triển một chương trình cho máy tính. Các công cụ trợ giúp như hợp ngữ (Assembler), chương trình dịch (Compiler), chương trình thông dịch (Interpreter), chương trình giám sát (Monitor), chương trình mô phỏng (Simulator), và hệ điều hành (Operating System) giúp người lập trình "giao tiếp" với những tài nguyên của máy tính phục vụ công việc lập trình.

1. Thuật giải và lưu đồ

Thuật giải là một thuật ngữ mô tả tập hợp các thủ tục cần thiết, hay còn gọi là phương pháp để giải quyết bài toán. Lưu đồ là biểu diễn đồ thị của các định nghĩa, phương pháp phân tích, và tuần tự giải bài toán. Các ký hiệu được sử dụng thường là các phép toán, phương thức xử lý, yêu cầu nhập xuất, quyết định, các ngắt, v.v… Thiết kế thuật giải hay lưu đồ là bước đầu tiên trong quá trình xây dựng một chương trình.

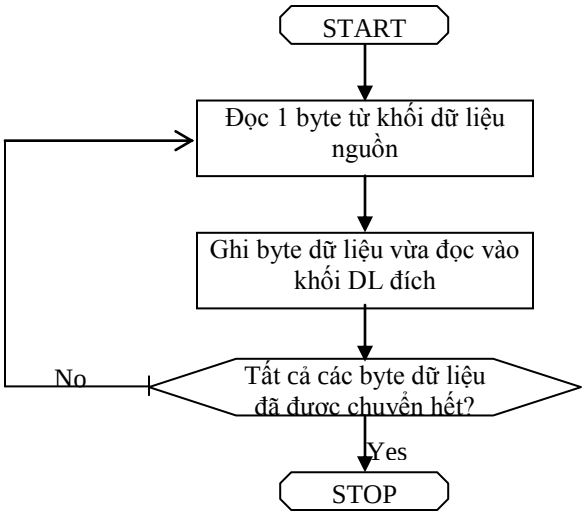
Ví dụ: Viết chương trình chuyển một khối dữ liệu trong bộ nhớ sang một vị trí mới.

Thuật giải tổng quát sẽ gồm 3 bước sau:

Bước 1. Đọc một byte dữ liệu từ khối dữ liệu nguồn

Bước 2. Ghi byte dữ liệu vừa đọc vào vị trí tương ứng ở khối dữ liệu đích

Bước 3. Kiểm tra xem đã chuyển hết khối chưa? Nếu đã hết, dừng chương trình; nếu chưa hết, quay lại bước 1.



Lưu đồ chương trình được thể hiện trên *Hình VIII.1.* Thực tế, thuật giải vừa nêu là quá sơ sài, chưa thể hiện được những yêu cầu cụ thể khi viết chương trình cho máy tính.

Chúng ta có thể tham khảo thêm thuật giải đầy đủ bao gồm 9 bước sau đây:

Bước 1. Đặt giá trị khởi đầu cho con trỏ vào địa chỉ đầu của khối dữ liệu nguồn (Pointer1)

Bước 2. Đặt giá trị khởi đầu cho con trỏ vào địa chỉ đầu của khối dữ liệu đích (Pointer2)

Bước 3. Đặt giá trị thanh đếm số byte cần di chuyển

Bước 4 Đọc một byte từ khối dữ liệu nguồn tại vị trí con trỏ (Poniter1)

Bước 5. Ghi byte vừa đọc được vào khối dữ liệu đích vào vị trí con trỏ (Pointer2)

Bước 6. Tăng giá trị con trỏ Pointer1 lên 1, (chỉ vào byte kế tiếp)

Bước 7. Tăng giá trị con trỏ Pointer2 lên 1, (chỉ vào vị trí kế tiếp)

Bước 8. Giảm giá trị thanh đếm số lượng byte cần di chuyển đi 1

Bước 9. Kiểm tra nội dung thanh đếm, nếu bằng 0, dừng lại, nếu khác 0, quay lại thực hiện tiếp từ bước 1.

Trong thực tế, tồn tại cách thể hiện lưu đồ chương trình dưới 3 dạng sau:

- *Lưu đồ tổng quát (general flowchart)*: được sử dụng để mô tả thuật giải một cách tổng quan, sơ lược
- *Lưu đồ mức thuật giải (Algorithmic-level flowchart)*: mô tả từng bước cụ thể trong thuật giải, dưới dạng lời giải thích tác vụ cụ thể cần thực hiện trong mỗi bước
- *Lưu đồ mức lệnh (Instruction-level flowchart)*: mô tả từng bước cụ thể trong thuật giải, mỗi khối trong lưu đồ được sử dụng các lệnh cụ thể thay cho lời giải thích tác vụ.

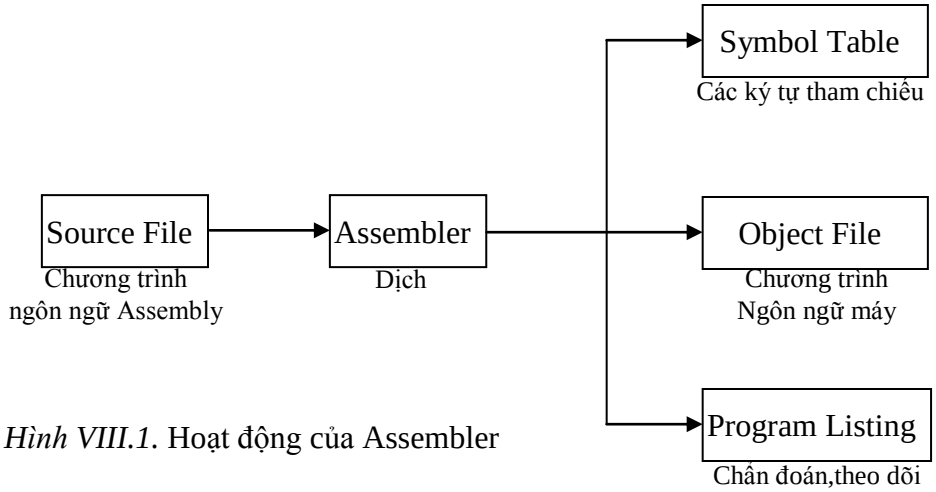
2. Lập trình hợp ngữ (Assemblers)

Hợp ngữ (Assembler) là một trong những công cụ mạnh nhất thường được sử dụng để phát triển các chương trình dạng mã máy. Hợp ngữ là chương trình dịch các mã lệnh gợi nhớ (mnemonics) và các ký hiệu (symbols) thành dạng mã nhị phân mà máy tính có thể thực thi được. Hợp ngữ là một phần mềm (SoftWare) của máy tính.

Có thể hình dung đầu vào (input) của hợp ngữ là một tập hợp các lệnh của chương trình dùng mã gợi nhớ, đầu ra (output) là chuỗi các byte dữ liệu

nhị phân biểu diễn mã lệnh của máy (người lập trình có thể đọc được dưới dạng các chữ số thập phân, hexa hoặc octal). Mã lệnh và dữ liệu cần xử lý được sắp xếp theo tuần tự xác định kể cả địa chỉ cụ thể của vùng nhớ mà nhóm các byte mã lệnh hay dữ liệu được lưu giữ.

Đầu vào của hợp ngữ được gọi là mã nguồn (source code), còn đầu ra của hợp ngữ là mã đối tượng (object code). Quá trình dịch từ mã nguồn sang mã đối tượng được gọi là trình dịch hợp ngữ (assembly).



Hình VIII.1. Hoạt động của Assembler

Tác vụ đầu tiên của Assembler là dịch chương trình từ File nguồn được viết bằng các mã lệnh gợi nhớ (assembly language program) thành tập mã thích hợp cho CPU thực thi (Machine language program). Quá trình dịch được gọi là Assembly. Cũng cần nhắc lại rằng: các lệnh gợi nhớ (mnemonics) là tập lệnh của từng loại CPU, do hãng chế tạo CPU cung cấp. Các lệnh này được tạo ra phù hợp với cấu trúc và tính năng của từng loại CPU. Tuỳ theo kiến trúc RICS hay CICS của CPU mà tập lệnh này khác nhau về ký tự mnemonics cũng như phép toán mà lệnh có khả năng xử lý.

Có hai thể loại Assembler là assembler tuyệt đối (absolute assembler) và assembler tái định vị (relocatable assembler). Assembler tuyệt đối tạo tệp mã đối tượng có tham chiếu địa chỉ tuyệt đối, nghĩa là khi tệp mã đối tượng được tạo, có thể nạp ngay vào máy tính và thực hiện chương trình bắt đầu từ địa chỉ đã được cài. Assembler tái định vị tạo tệp mã đối tượng có địa chỉ tương đối, và chỉ tạo địa chỉ tuyệt đối bằng một chương trình khác được gọi là chương trình định vị (locator). Một trong những ưu điểm của các chương trình assembler là khả năng tạo các macro. Có thể coi Macro như là khả năng thêm lệnh mới cho tập lệnh của một CPU, vì sau khi đã xây dựng, có thể sử dụng nó như một lệnh mnemonic.

3. Chương trình dịnh (Compilers)

Chương trình dịch Compiler có thể nói là cùng họ với Assembler về chức năng. Chúng đều là những chương trình dịch. Mã đối tượng do chương trình Assembler và Compiler tạo ra là hoàn toàn đồng nhất và không có sự khác biệt. Sự khác nhau chủ yếu và rõ nhất là ở File nguồn.

File nguồn của Assembler là các mã lệnh dạng mnemonics, đòi hỏi người lập trình nắm vững cấu trúc của CPU cũng như các phương thức định địa chỉ trong lệnh. Chương trình thường được gọi là chương trình mức thấp (Low-level program). Khi người lập trình sử dụng một ngôn ngữ lập trình bậc cao hơn, thông thường, họ không cần biết gì về phần cứng như thanh ghi, bản đồ bộ nhớ, bản đồ thiết bị vào/ra, v.v… Chương trình được viết giống như sử dụng Anh ngữ (English-like) thông qua các phát biểu (Statements) và là đầu vào của chương trình dịch Compiler.

Trong khi các chương trình mức Assembler là hướng thiết bị (machine oriented) thì ngôn ngữ mức compiler (compiler-level) là hướng nhiệm vụ (task-oriented). Một lập trình viên chỉ có thể viết được chương trình Assembler khi hiểu rõ về cấu trúc và tập lệnh của CPU, trong khi đối với mức compiler, có thể hiểu được mục đích của công việc thông qua cấu trúc chương trình. Cũng giống như hợp ngữ, có hai dạng compiler là thường trú (resident compiler) và *cross* compiler. Sự khác nhau của chúng cũng giống như giữa hai loại hợp ngữ đã nêu trên.

4. Liên kết và định vị (Linkers and Locators)

Như đã nói ở trên, các chương trình khi dịch thường cho ta một tệp mã đối tượng chưa có địa chỉ tuyệt đối. Địa chỉ trong mã thường được tham chiếu tới một vị trí nhớ thông thường có địa chỉ bằng 0, nên được gọi là tương đối. Ví tính chất này, mã đối tượng trở thành loại mã có khả năng tái định vị. Dĩ nhiên, nếu địa chỉ được gán vào mã đối tượng là tương đối, chương trình có thể cài vào bất kỳ vị trí nào của bộ nhớ để thực hiện. *Locators* làm nhiệm vụ coi mã đối tượng tái định vị như là đầu vào, nó gán địa chỉ tuyệt đối tương ứng với vị trí mà người dùng xác định để tẫom đối tượng tuyệt đối.

Liên kết (Linkers) tạo một thuận lợi lớn cho người lập trình: cho phép trong một chương trình vùa sử dụng các ngôn ngữ bậc cao với hợp ngữ, nhằm đạt được tốc độ cao khi thực hiện chương trình. Linkers làm nhiệm vụ liên kết các mã đối tượng tương đối do assembler và compiler tạo ra thành một mã đối tượng tương đối có khả năng tái định vị tham chiếu về một vị trí nhớ theo yêu cầu của người dùng. Trong tực tế, Linkers được dùng để tạo

khả năng sử dụng các thư viện chương trình con và Macro có sẵn dạng thư viện tiện ích (Utility Library) khi viết chương trình bằng hợp ngữ.

5. Chương trình thông dịch (Interpreters)

Cũng tương tự như compilers, chương trình thông dịch (interpreters) sử dụng ngôn ngữ bậc cao để viết chương trình. Điều khác nhau là ở chỗ interpreters không tạo ra mã đối tượng. Chương trình interpreters cùng tồn tại với chương trình người dùng, mỗi phát biểu (statement) trong chương trình người dùng được thông dịch trực tiếp thành mã máy và lệnh sẽ được CPU thực hiện tức thời. Dễ thấy rằng chương trình không tốn nhiều bộ nhớ và được thực hiện chậm hơn rất nhiều khi sử dụng Compilers.

Quá trình phát tiển phần mềm bao gồm giải quyết vấn đề và gỡ rối thường sủ dụng interpreters. Khi "phần rối " đã được gỡ, chương trình sẽ được dịch thành mã đối tượng.

6. Hệ điều hành (Operating Systems)

Hệ điều hành là công cụ đầu tiên tạo khả năng cho người dùng truy xuất vào các tài nguyên phần cứng và phần mềm của một máy tính. Thông thường hệ điều hành là một tập hợp các môđun phần mềm tạo điều kiện cho người dùng máy tính truy xuất vào tập hợp các lệnh (comands) và tiện ích. Sự tiện dụng của các hệ điều hành, từ đơn giản nhất như các chương trình giám sát (Monitors) đến các phiên bản DOS và Windows của Microsoft đã quá quen thuộc đối với người dùng.

Có thể coi Monitor là một hệ điều hành dạng đơn giản nhất, độ lớn của nó chỉ khoảng 1 đến 2KB và được đặt trong ROM. Nó cho phép người dùng có thể nạp chương trình vào RAM, truy xuất vào các thanh ghi của CPU,… và thực hiện chương trình, sửa chữa chương trình khi cần thiết, và thường được cài đặt trong các Design Kit

TÀI LIỆU THAM KHẢO

[1] *David Hergert, Nancy Thibeault* PC Architecture from Assembly langueage to C – Prentice-Hall, Inc. New Jersey 1997

[2] *Victor M. Roooney, Amin R. Issmail* Microproccessors and Microcomputer – Macmillan Publisshing Company – New York 1984

[3] *Kai Hwang* Advanced Computer Architecture. Parallelism – Scalability –Programmability – McGraw-Hill International Editions – 1993

[4] *Christopher L. Morgan & Mitchell Waite* 8086/8088 16-Bit Microproccessor Primer – McGraw-Hill, Inc. 1982

[5] *James L., Turley* Advanced 80386 Programming Techniques - Osborne McGraw-Hill – Berkeley, California 1988

[6] *Andrew S. Tanenbaun* Structured Computer Organization – Prentice Hall 1990

[7] *James M. Feldman, Charles T. Retter* Computer Architecture – MITPress & McGraw Hill, Singapore 1994

[8] *Vũ Chấn Hưng* Giáo trình Kiến trúc máy tính – NXB Giao thông vận tải - Hà Nội 2002

[9] *Đặng Thành Phu* Turbo Assembler & ứng dụng – NXB Khoa học và kỹ thuật – Hà Nội 2007

[10] *Nguyễn Thuý Vân* Kỹ thuật số - NXB Khoa học và kỹ thuật – Hà Nội 1999

[11] *Nguyễn Trung Đồng* Giáo trình Kỹ thuật Vi xử lý - Tập bài giảng

[12] *Nguyễn Đình Việt* Kiến trúc máy tính – NXB Đại học Quốc gia Hà Nội 2006

[13] Giáo trình Kỹ thuật Vi xử lý (2 tập) – Biên soạn *TS. Hồ Khánh Lâm* – NXB Bưu điện – Hà Nội 2007

[14] *Parham Behrooz* Computer Architecture: From Microprocessors to Supercomputers, Oxford University Press, 556 + xx pp., February 2005, rev. 2007 (ISBN 0-19-515455-X).

MỤC LỤC