KIẾN TRÚC MÁY TÍNH

Giảng viên: Ths Phạm Thanh Bình
Bộ môn Kỹ thuật máy tính & mạng
http://vn.myblog.yahoo.com/CNTT-wru
http://ktmt.wru.googlepages.com

Nội dung:

- Tổng quan về máy tính
- Ngôn ngữ giao tiếp với máy tính
- Các phép toán trên máy tính
- Bộ vi xử lý
- Bộ nhớ
- Thiết bị ngoại vi

Chương 1:

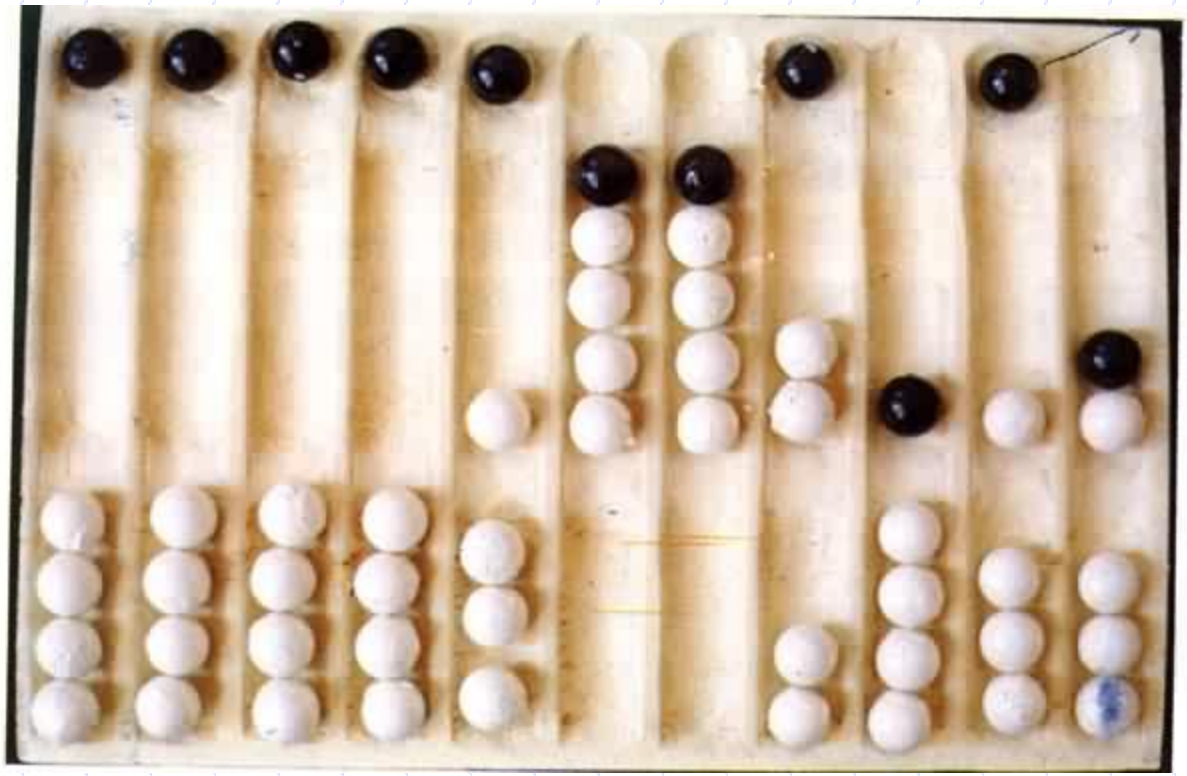
TỔNG QUAN VỀ MÁY TÍNH

- Các thế hệ máy tính
- Phân loại máy tính
- Phần mềm hệ thống
- Các thành phần của máy tính

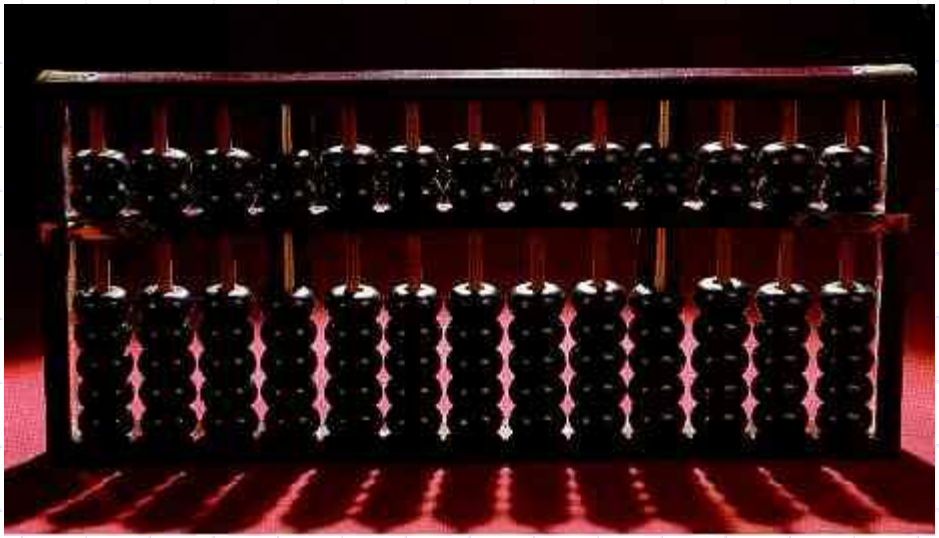
Bài 1.1 – Các thế hệ máy tính

- Các công cụ tính toán thô sơ
- Máy tính cơ học
- Máy tính cơ - điện
- Máy tính dùng bóng điện tử
- Máy tính dùng bóng bán dẫn
- Máy tính dùng mạch tổ hợp
- ...

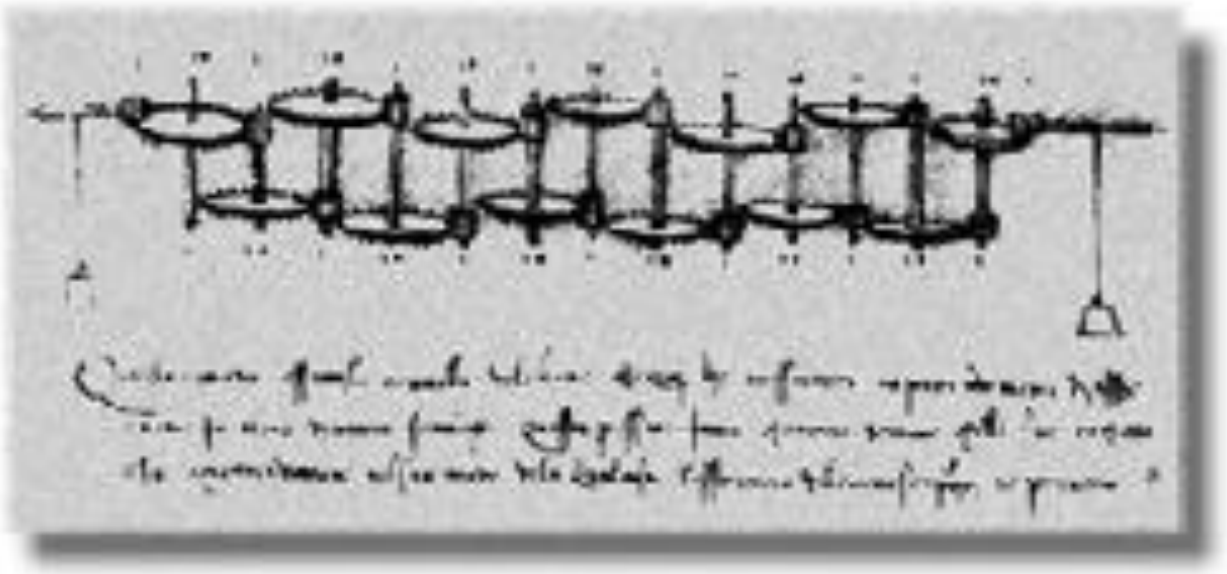
Bàn tính của người Babylon



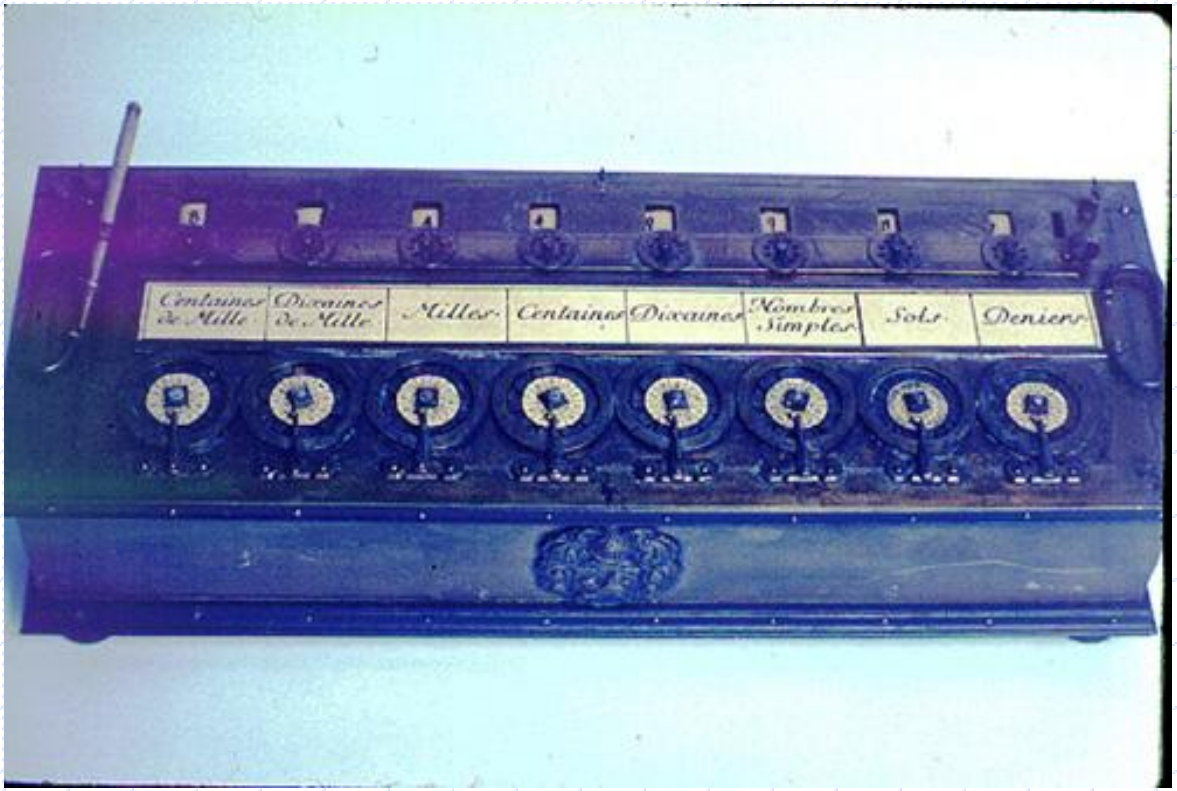
Bàn tính của người Trung Quốc



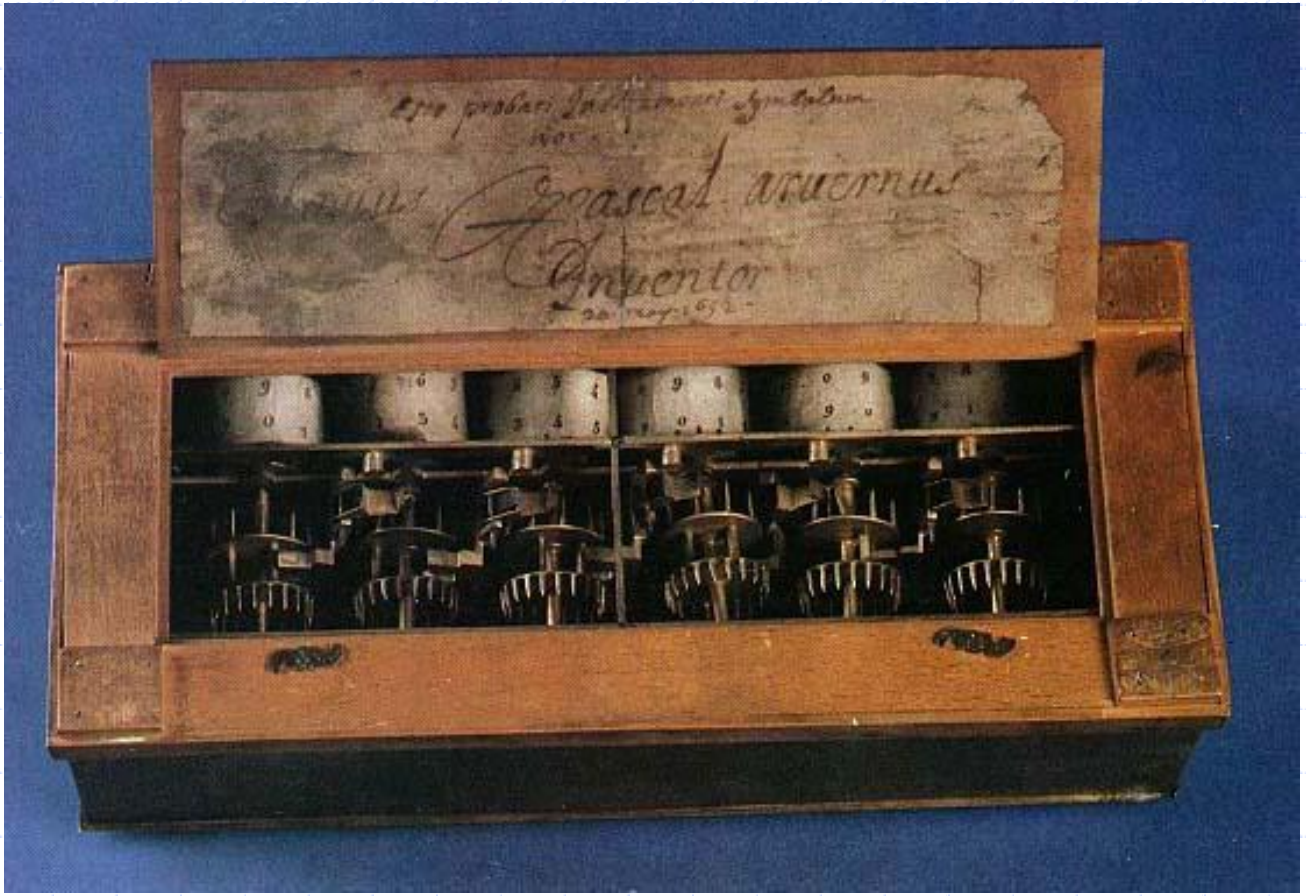
Bản vẽ của Leonardo da Vinci về chiếc máy tính cơ học



Máy tính cơ học của Pascal

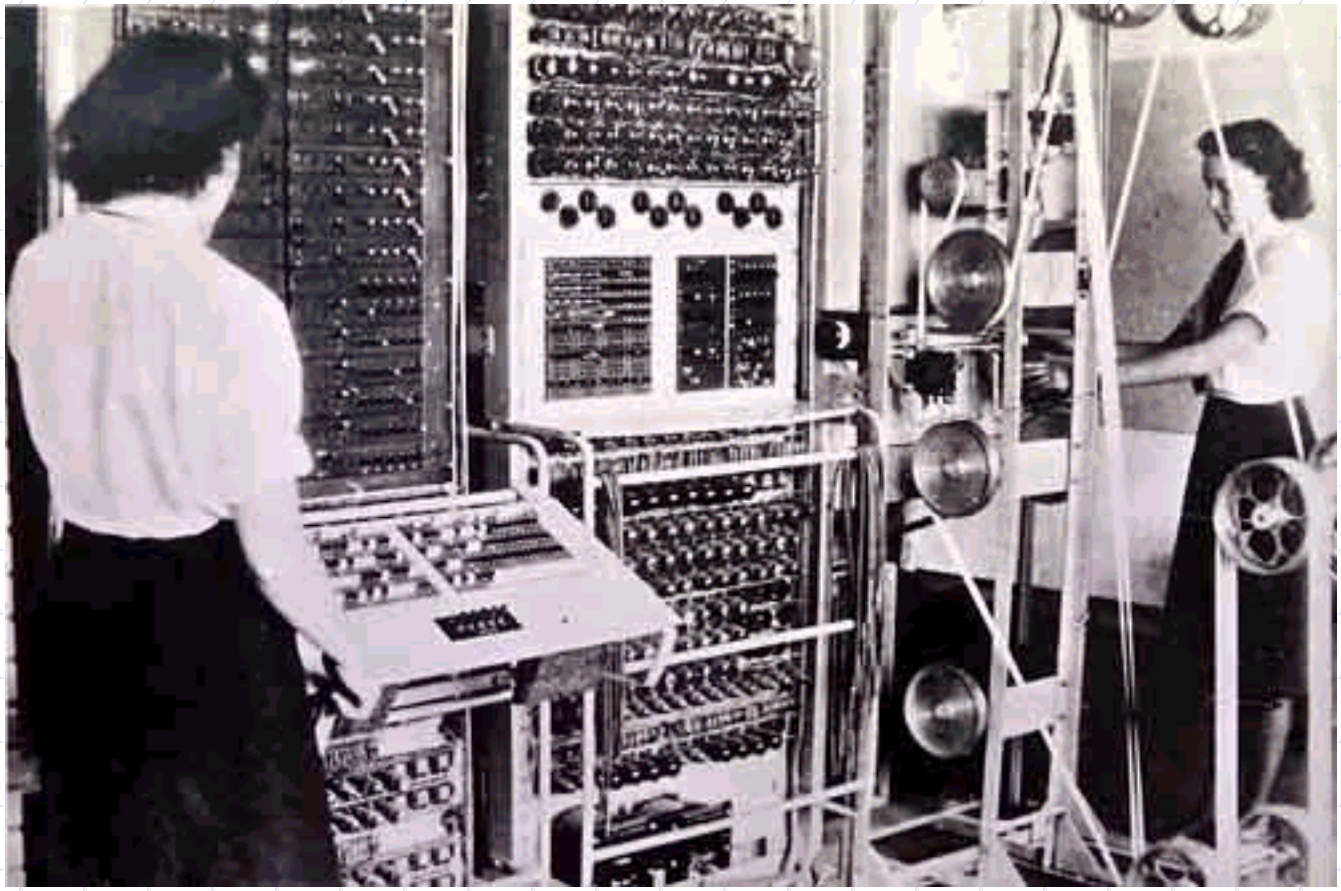


Bên trong máy tính cơ học

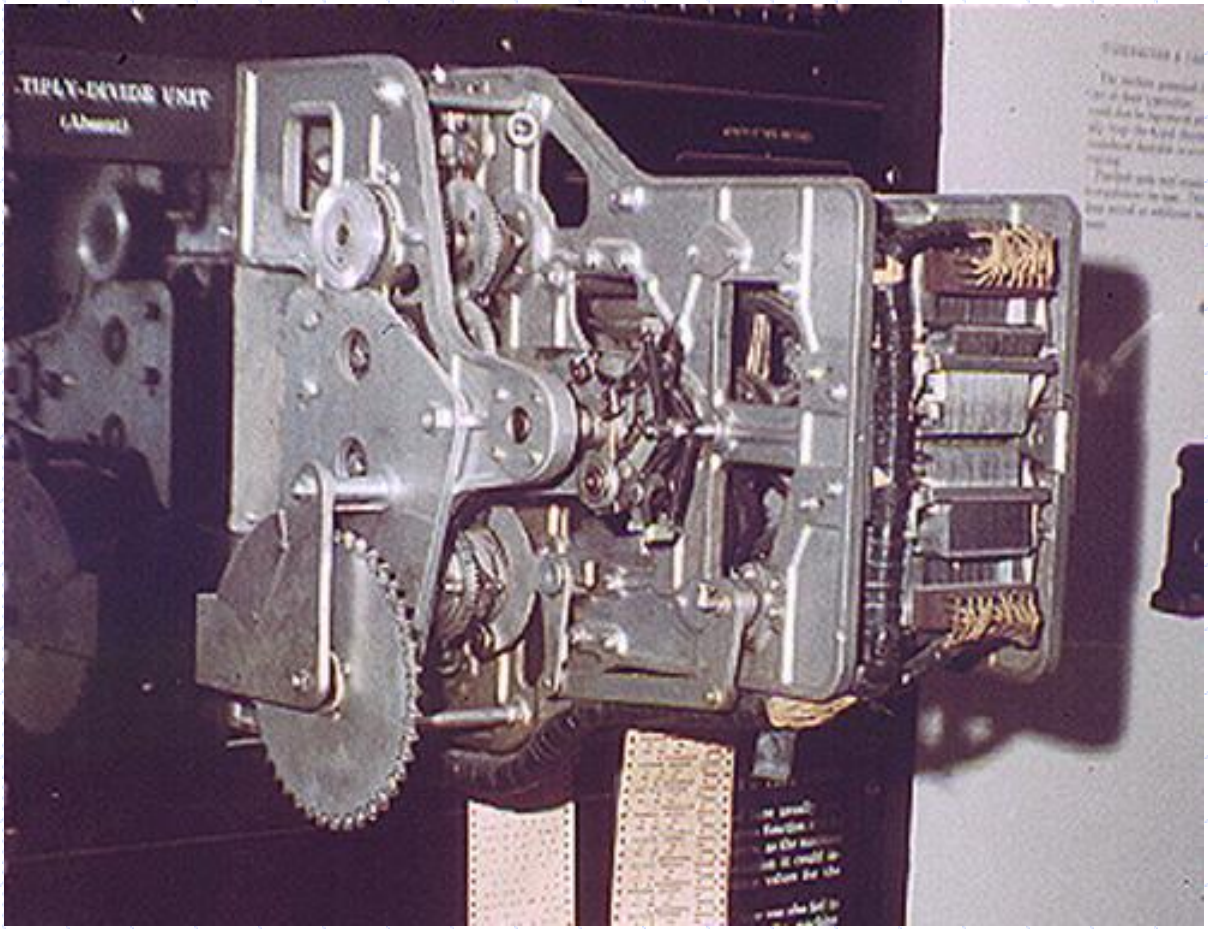


Máy tính cơ học cải tiến

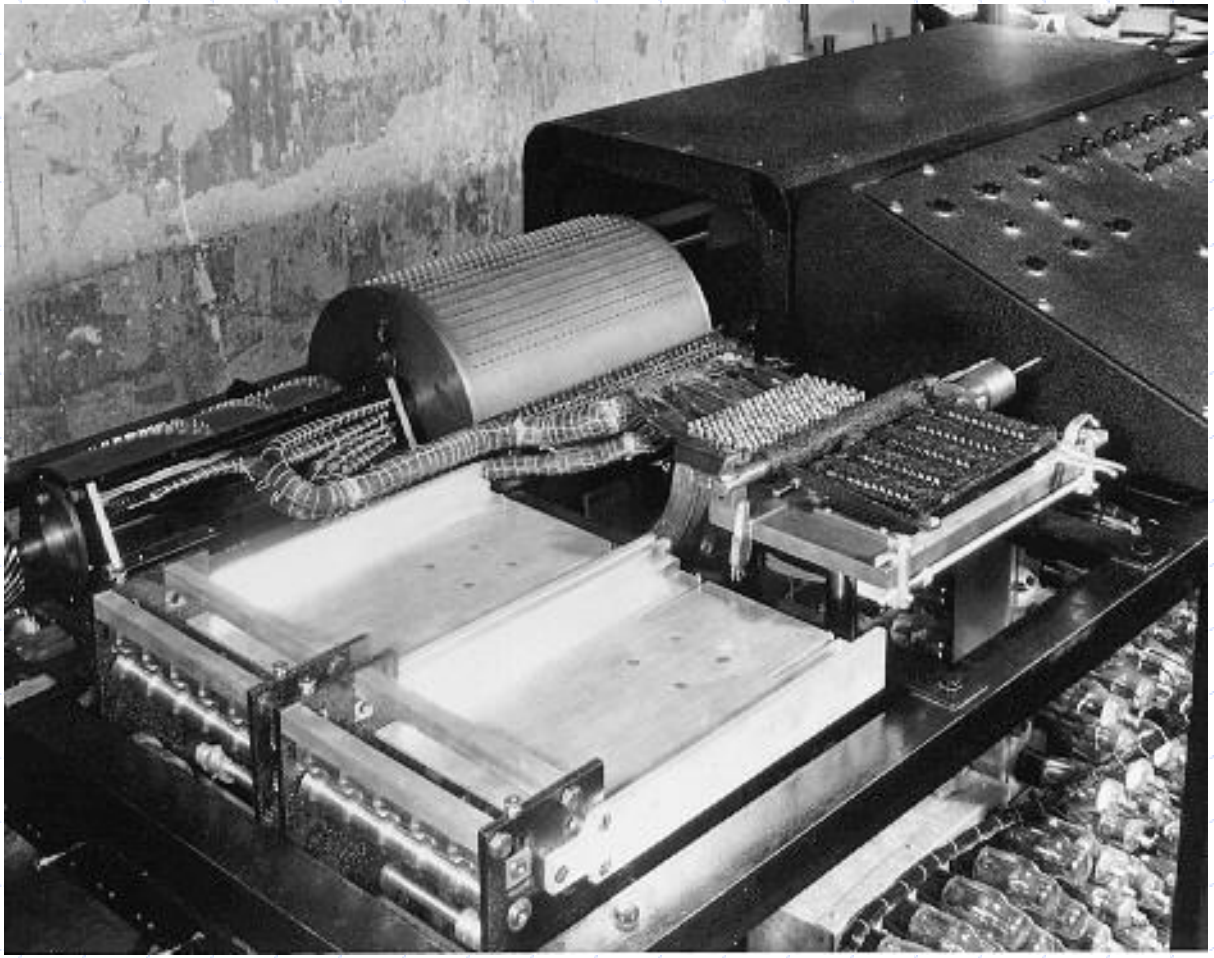
Máy tính cơ - điện



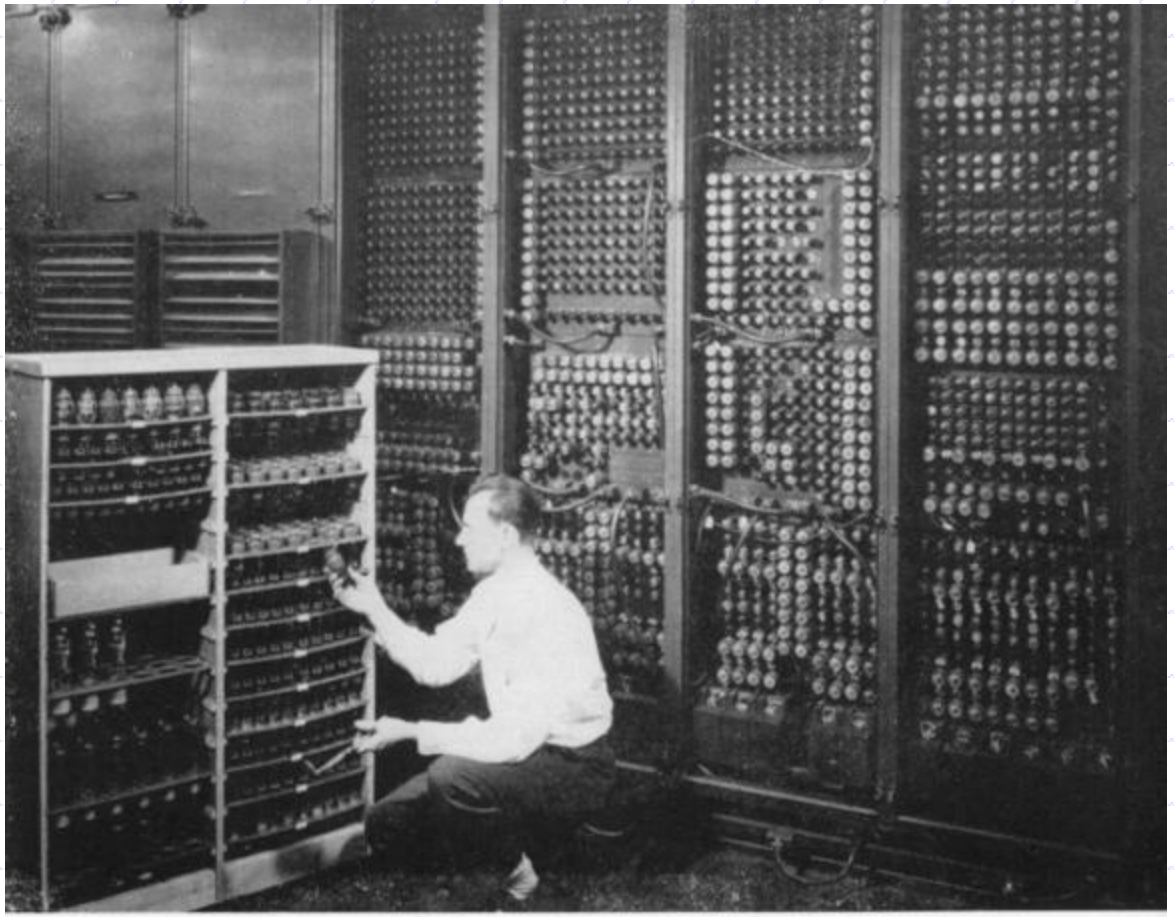
Bộ phận đọc bìa đục lỗ



Máy tính dùng bóng điện tử



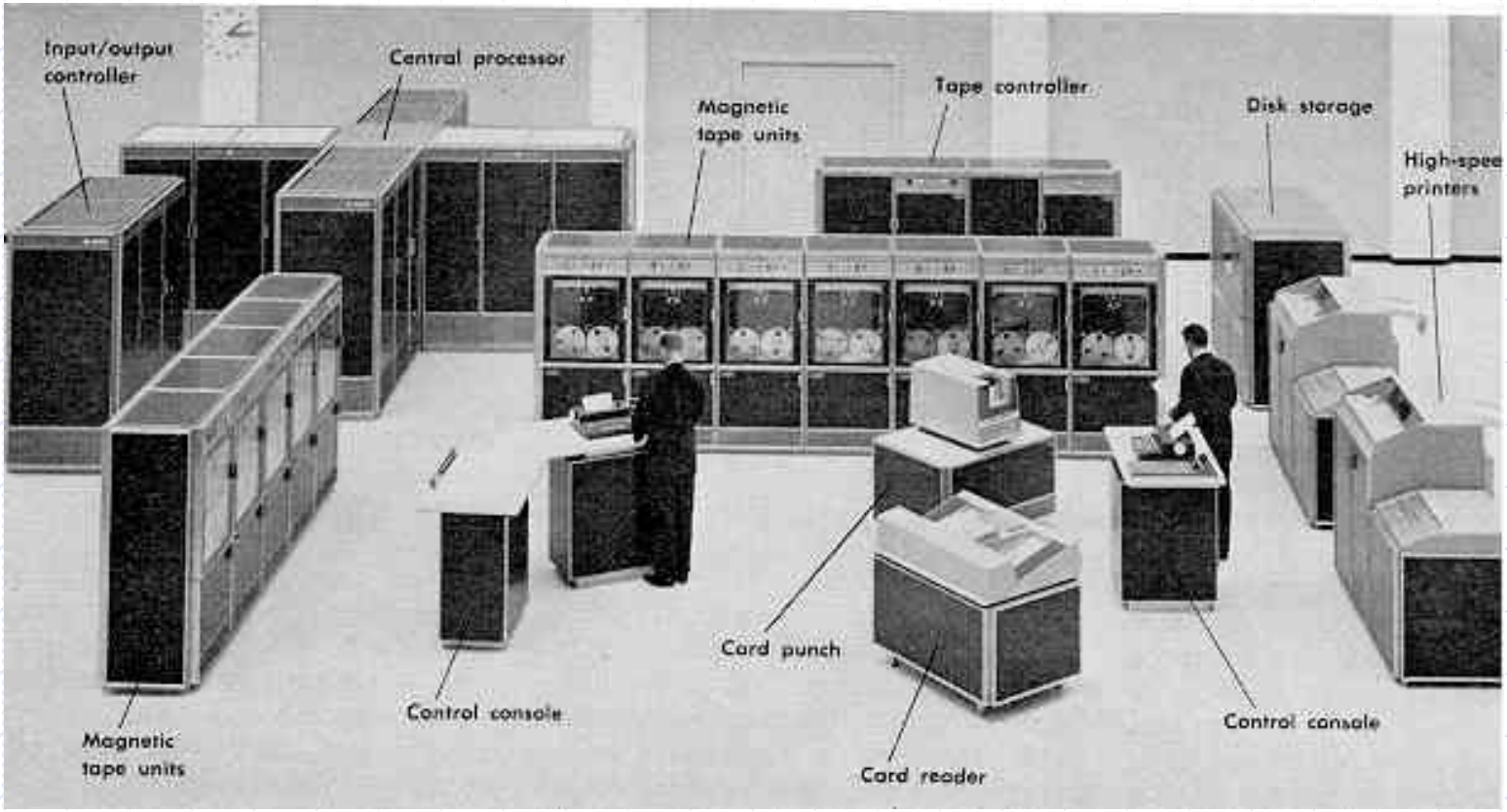
Máy tính dùng bóng điện tử



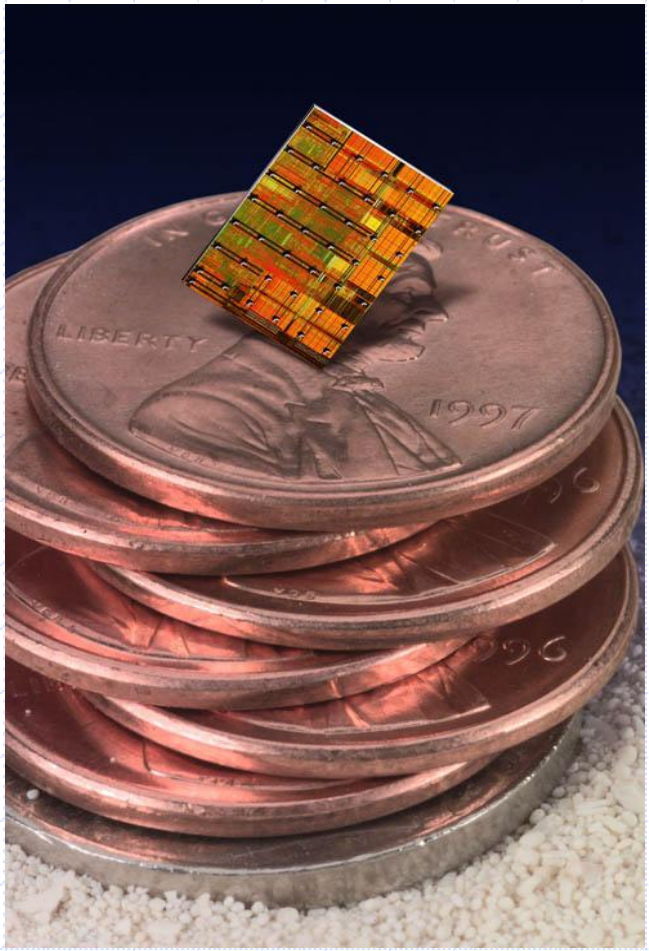
Replacing a bad tube meant checking among ENIAC's 19,000 possibilities.

Máy tính dùng bóng bán dẫn - IBM

Một trung tâm máy tính

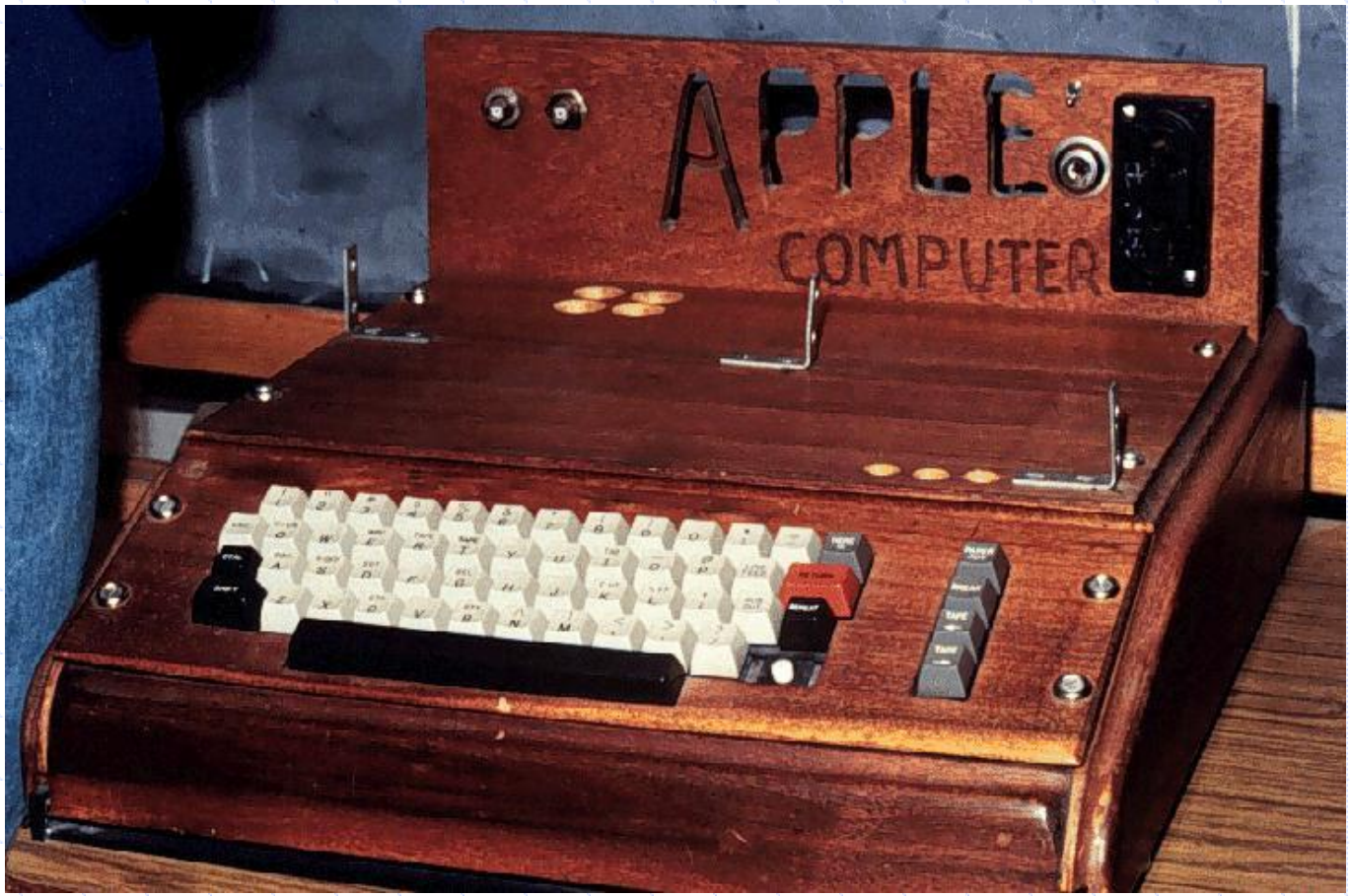


Mạch tổ hợp (IC)



Máy tính "mini"

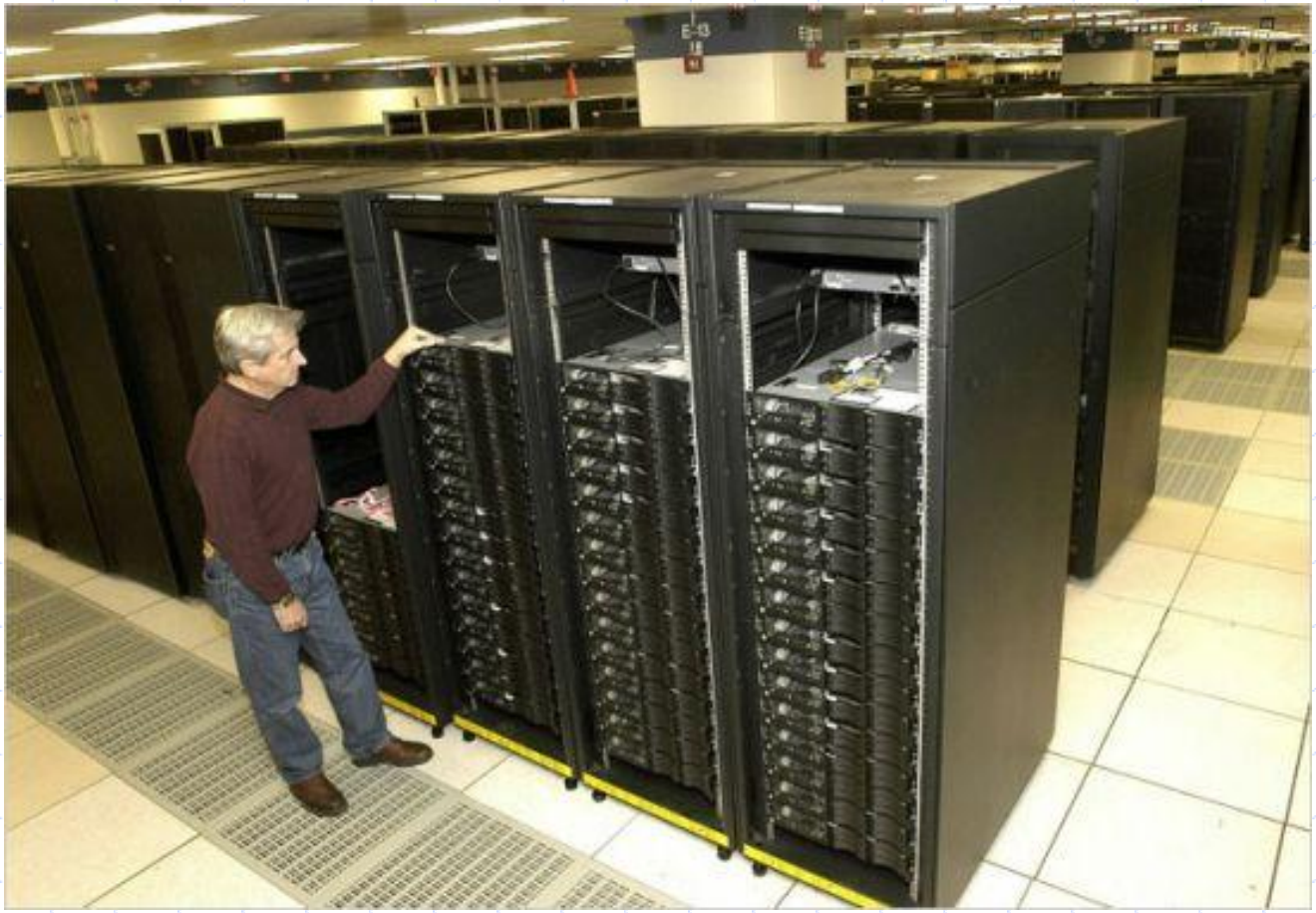
Chiếc máy Apple đầu tiên



Máy tính cá nhân IBM - PC

Máy tính xách tay

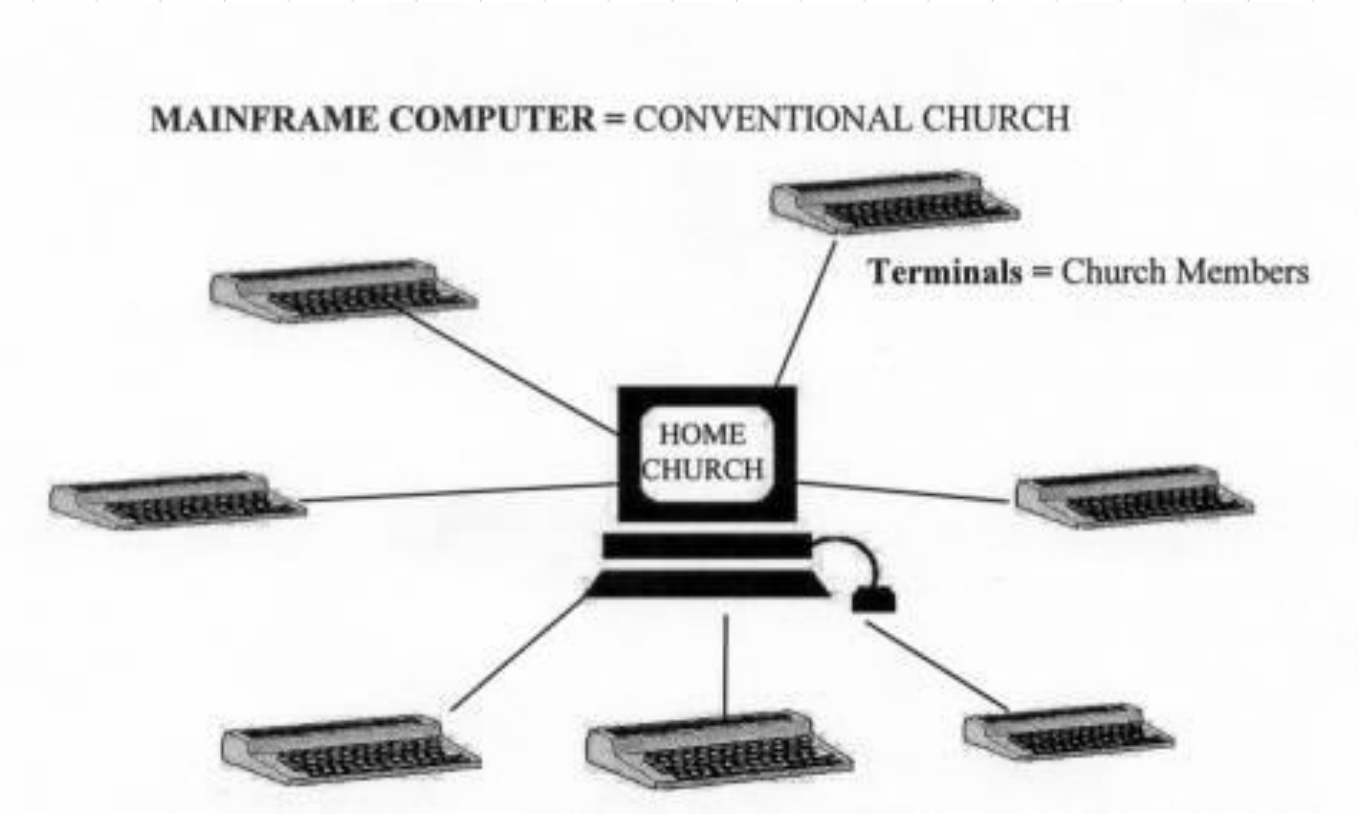
Siêu máy tính (Super Computer)



Bài 1.2 – Phân loại máy tính

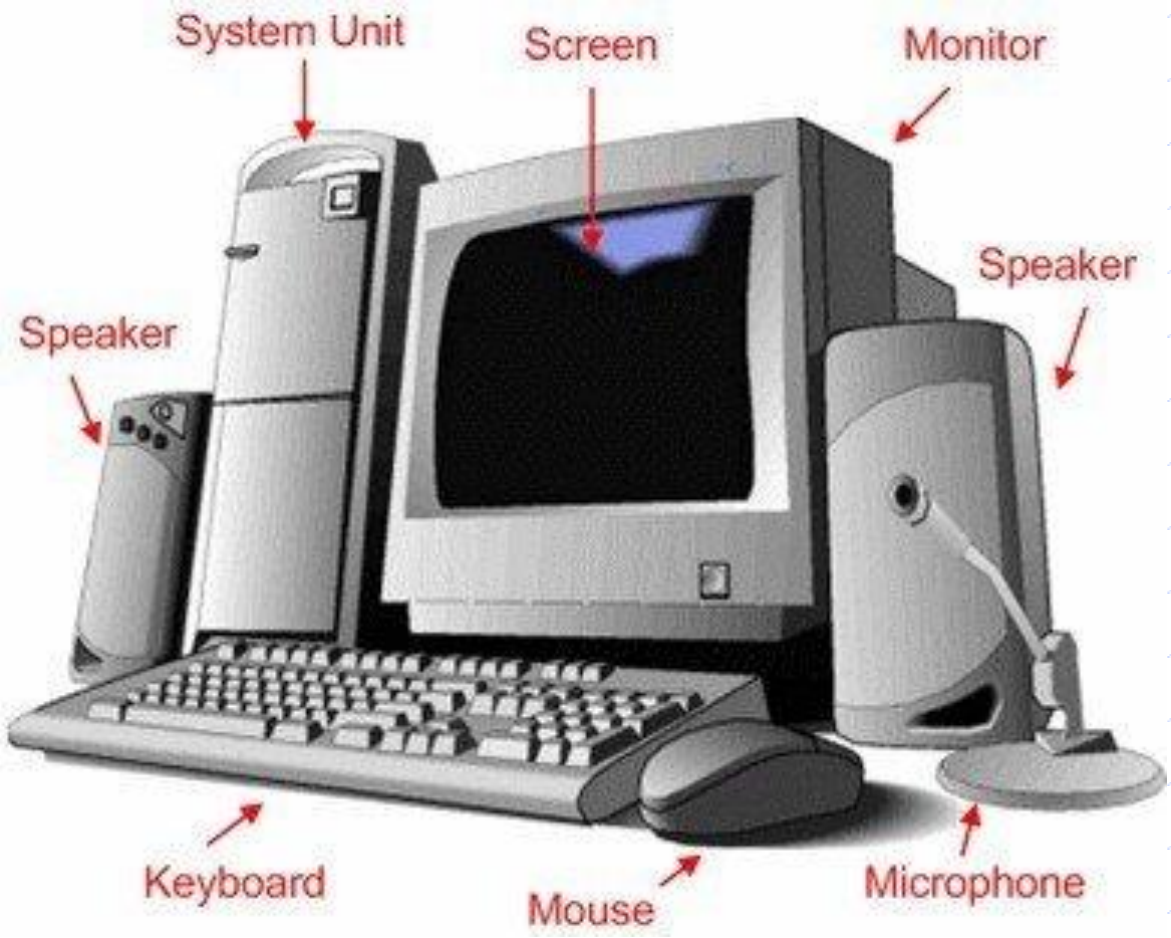
- Máy tính lớn (Mainframe Computer)
- Máy tính cá nhân (PC - Personal Computer)
- Siêu máy tính (Super Computer)
- Máy tính nhúng (Embedded Computer)
- ...

Mainframe Computer



Mainframe Computer – IBM 7094

Personal Computer



Super Computer

Embedded Computer

Embedded Computer

Bài 1.3 – Phần mềm hệ thống

Một máy tính muốn hoạt động được thì phải có phần mềm điều khiển nó:

Phần mềm ứng dụng– Application Software

Phần mềm hệ thống – System Software

Phần cứng - Hardware

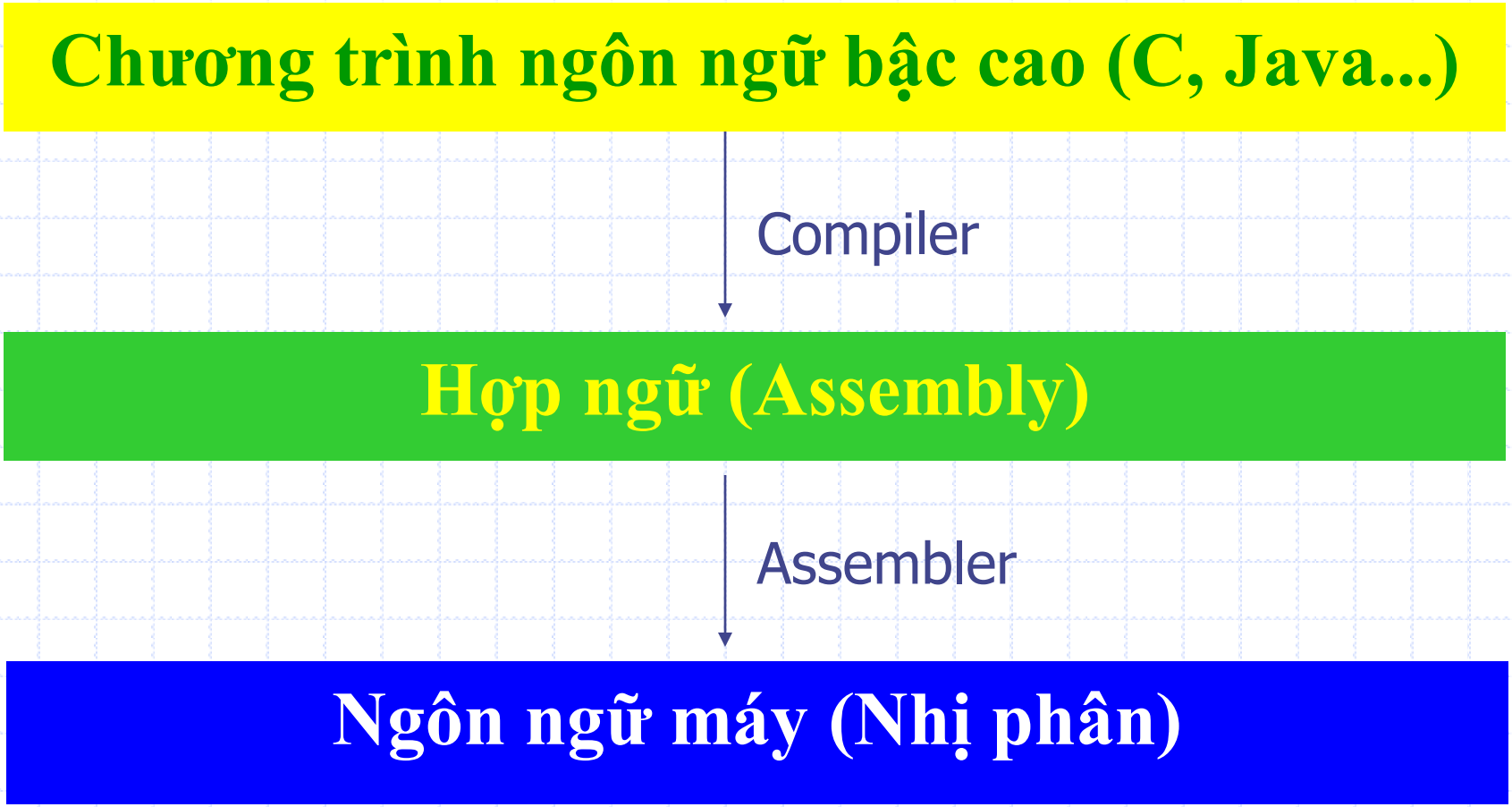
Các phần mềm hệ thống

- Hệ điều hành (Operating system)
- Chương trình dịch (Compiler, Assembler)
- ...

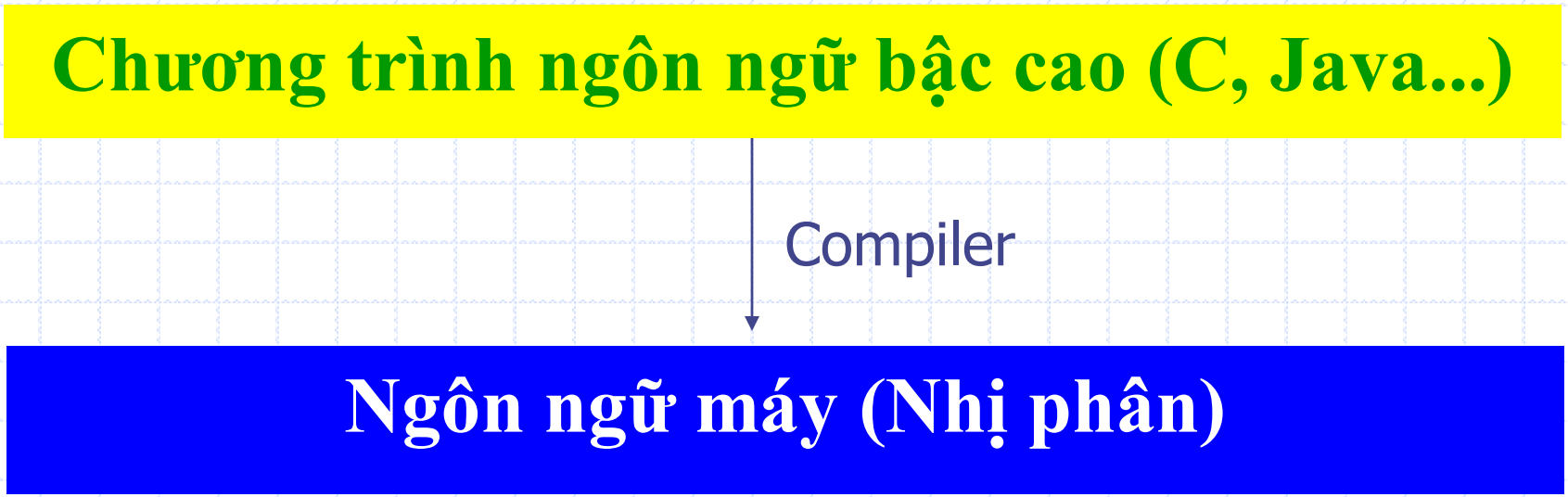
Hệ điều hành

- Quản lý và phân phối mọi tài nguyên của hệ thống như bộ nhớ, bộ vi xử lý, quản lý vào/ra, quản lý tiến trình...

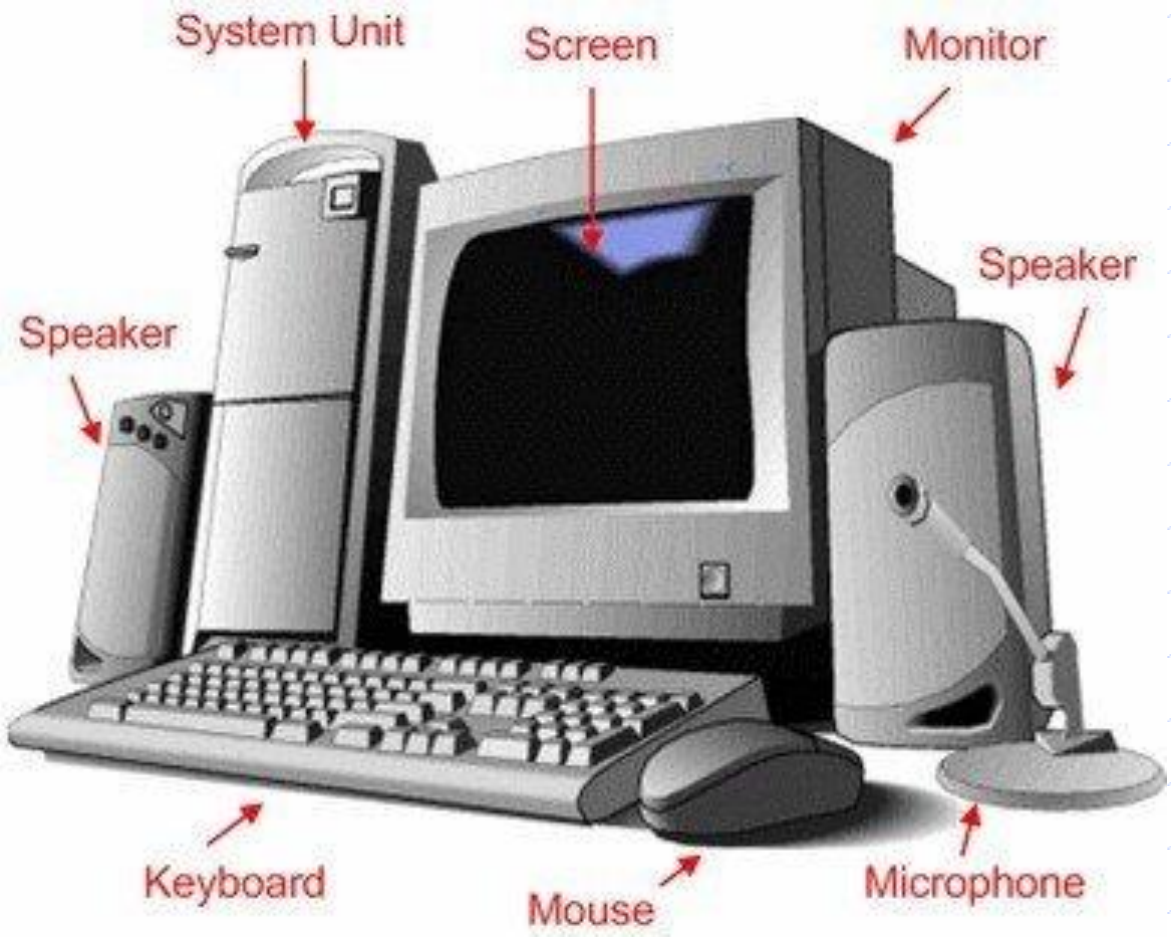
Chương trình dịch (cách 1)



Chương trình dịch (cách 2)



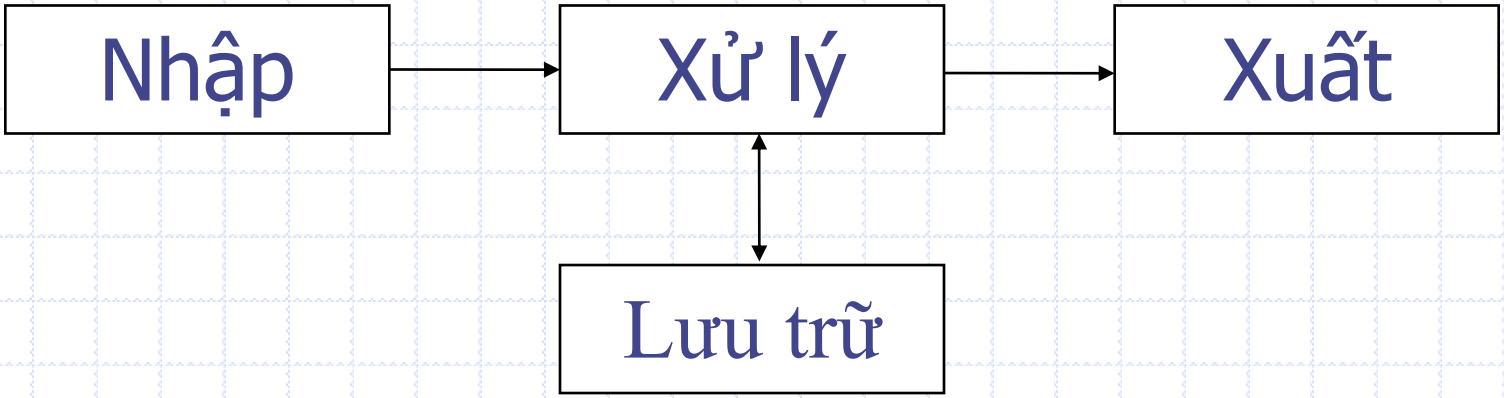
Bài 1.4 - Các thành phần của máy tính



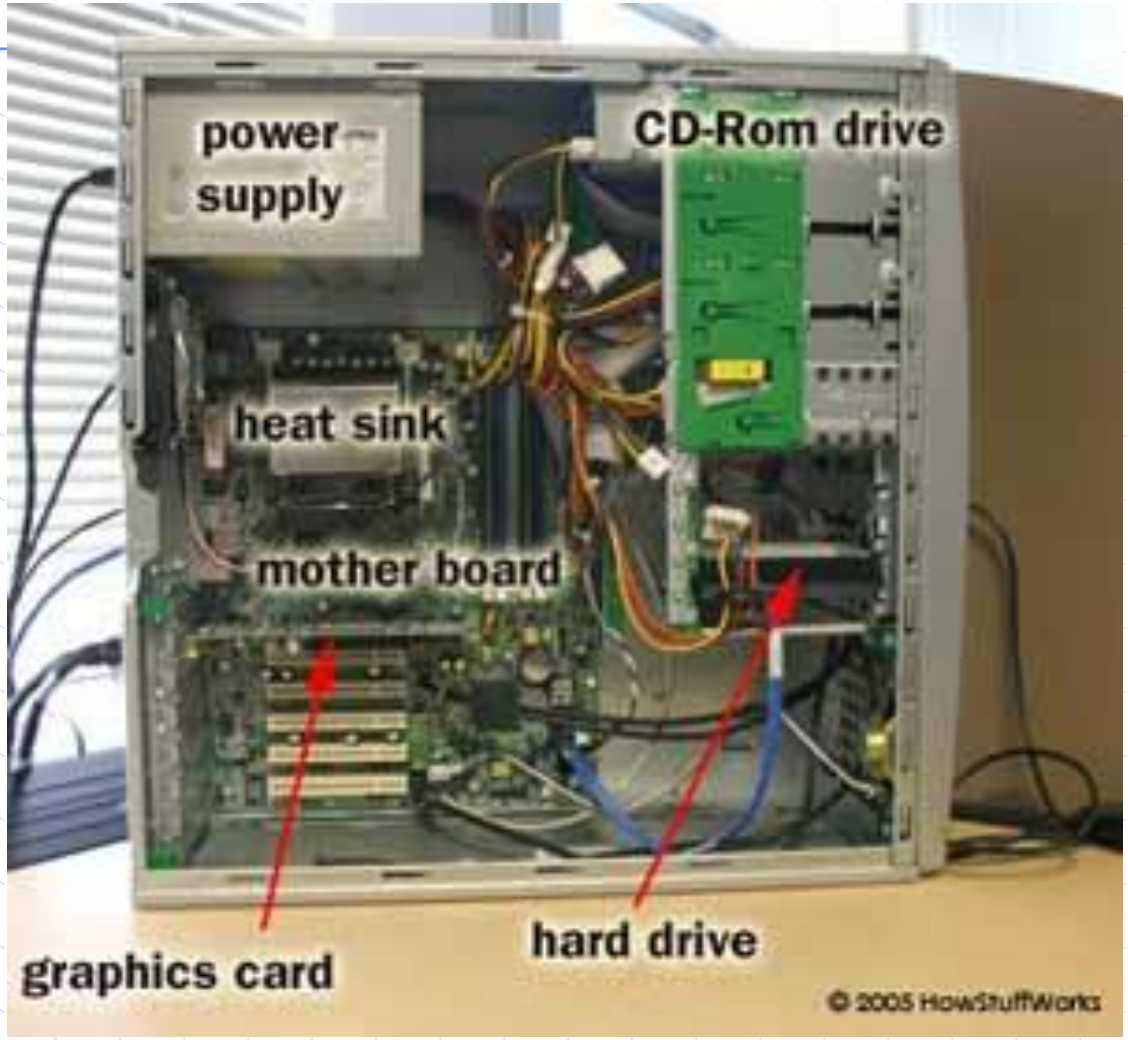
Có 4 khối chính:

- Khối xử lý: Bộ vi xử lý, bộ nhớ trong...
- Khối nhập dữ liệu: Bàn phím,chuột...
- Khối xuất dữ liệu: Màn hình, máy in...
- Khối lưu trữ: Đĩa cứng, đĩa mềm, đĩa CD...

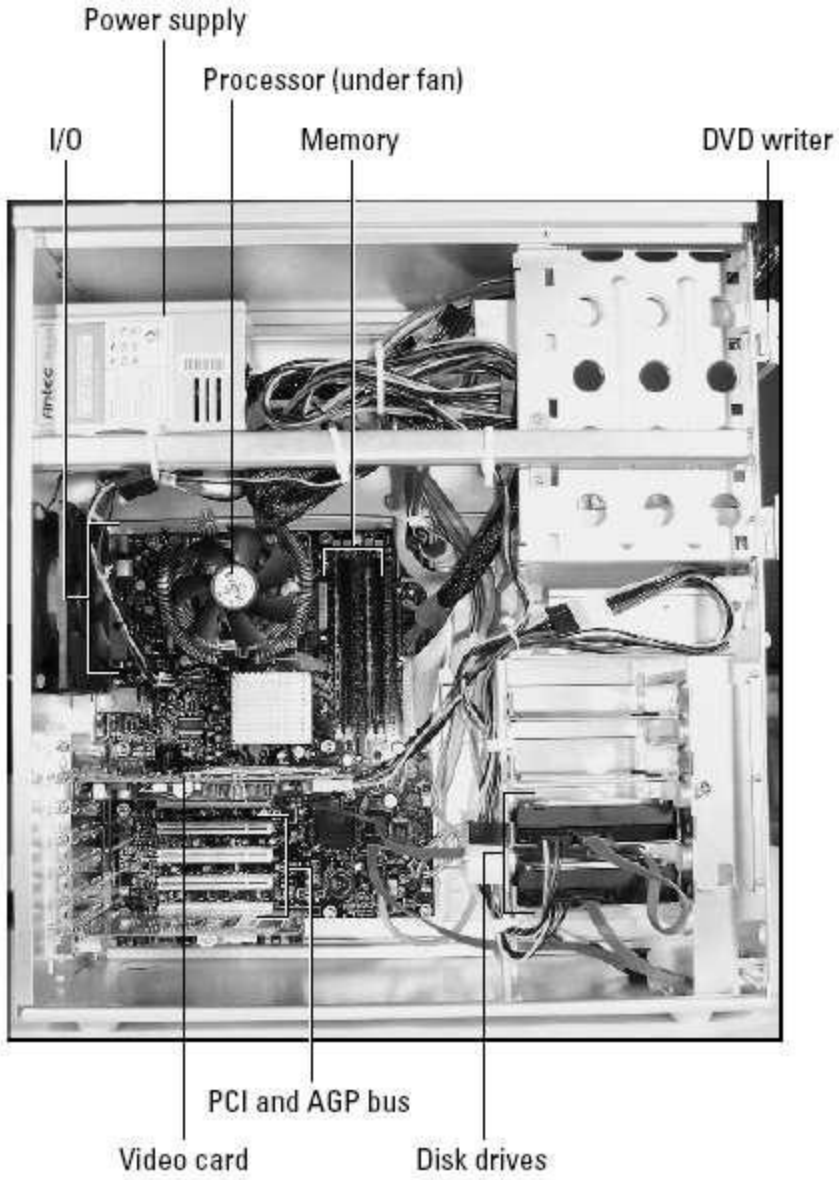
Sơ đồ khối:



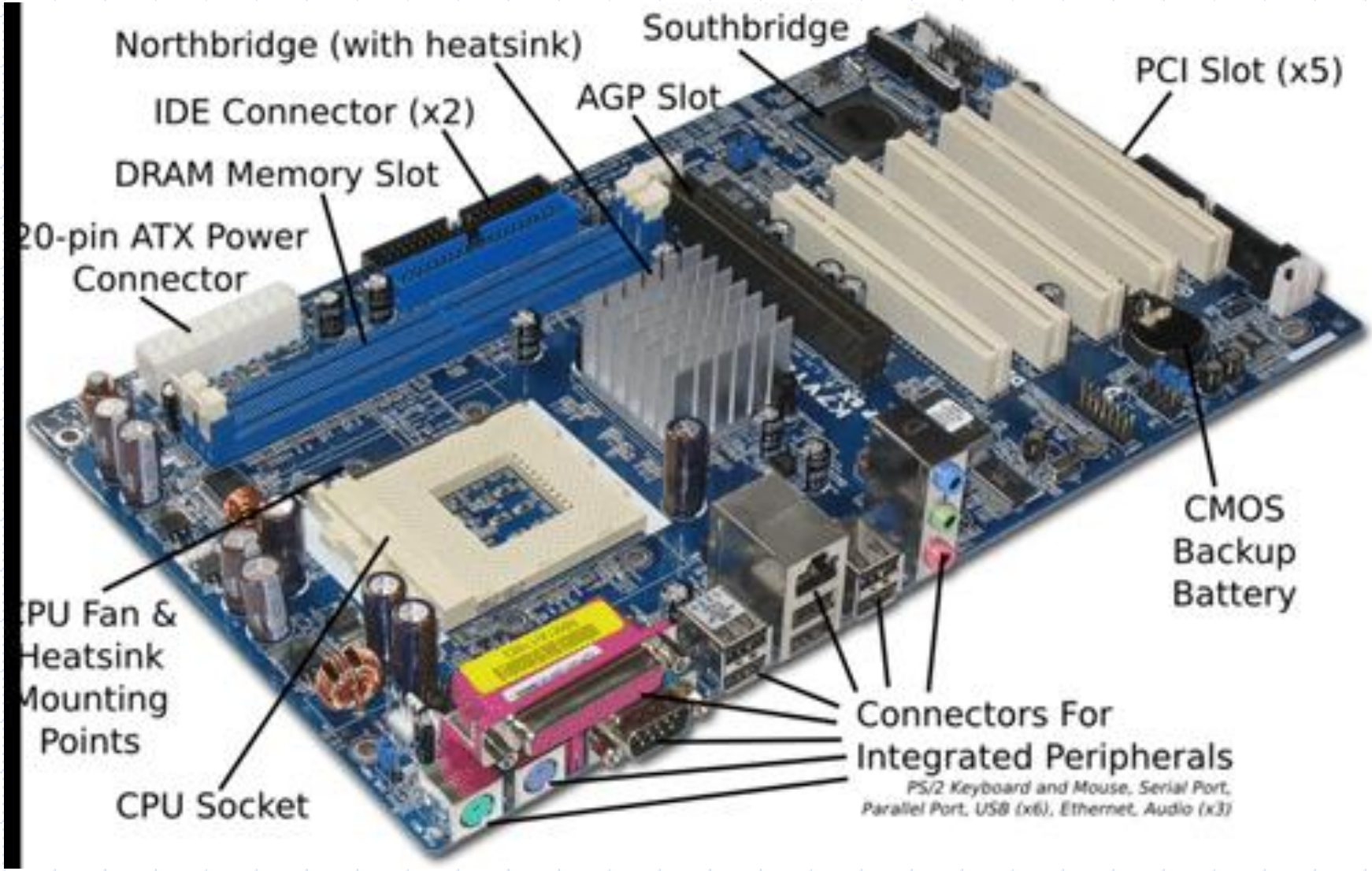
Bên trong máy tính



Bên trong máy tính



Bảng mạch chính (Mainboard)



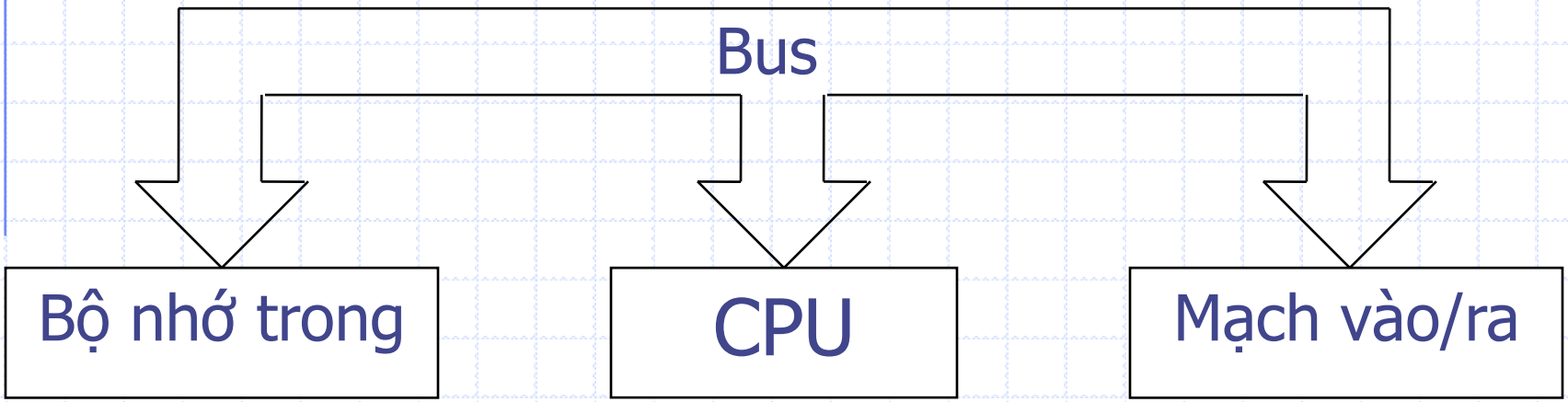
Khối xử lý

Đây là phần quan trọng nhất của một máy tính, bao gồm 3 bộ phận chính:

- Bộ vi xử lý – CPU
- Bộ nhớ trong
- Các mạch vào ra

Ngoài ra còn có hệ thống các dây dẫn, cáp nối để liên kết giữa các bộ phận trên (hệ thống Bus)

Sơ đồ khối xử lý:



Các thành phần của khối xử lý

- Bộ vi xử lý – CPU (Central Processing Unit): Là bộ não của máy tính, nó xử lý các thông tin và điều khiển mọi hoạt động của máy tính.
- Bộ nhớ trong: Là bộ nhớ có khả năng liên lạc trực tiếp với bộ vi xử lý,là nơi lưu trữ dữ liệu phục vụ cho quá trình xử lý.
- Các mạch vào ra: Để điều khiển việc giao tiếp với thiết bị ngoại vi.

Hết Phần 1

KIẾN TRÚC MÁY TÍNH

Giảng viên: Ths Phạm Thanh Bình
Bộ môn Kỹ thuật máy tính & mạng
http://vn.myblog.yahoo.com/CNTT-wru
http://ktmt.wru.googlepages.com

Chương 2:

NGÔN NGỮ CỦA MÁY TÍNH (ASSEMBLY)

- Các hệ đếm
- Biểu diễn số và kí tự trong máy tính
- Tổ chức CPU và bộ nhớ trong
- Các lệnh Assembly cơ bản

Mở đầu

- Hợp ngữ (Assembly language) là một ngôn ngữ lập trình cấp thấp, mục đích nhằm giao tiếp trực tiếp với phần cứng của máy tính.
- Máy tính chỉ có khả năng hiểu được các tín hiệu 0, 1 dưới dạng điện hoặc từ, gọi là tín hiệu nhị phân (ngôn ngữ nhị phân còn được gọi là ngôn ngữ máy).

Mở đầu

- Các lệnh Assembly thực chất là dạng kí hiệu của ngôn ngữ máy: Sử dụng các kí hiệu bằng tiếng Anh để biểu diễn các lệnh ngôn ngữ máy cho dễ nhớ hơn.

Lệnh ngôn ngữ máy	Lệnh hợp ngữ
0010 1010 1011 0100	MOV AH,2Ah
0010 0001 1100 1101	INT 21h

Bài 2.1 – Các hệ đếm

- Hệ thập phân (hệ đếm cơ số 10)
- Hệ nhị phân (hệ đếm cơ số 2)
- Hệ thập lục phân (hệ đếm cơ số 16)

Hệ thập phân (Decimal)

- Hệ thập phân sử dụng 10 kí hiệu (0, 1, 2,... 9) để biểu diễn thông tin. Các số trong hệ thập phân được biểu diễn dưới dạng tổng các luỹ thừa cơ số 10.
- Ví dụ: Số 1998 trong hệ thập phân có thể biểu diễn như sau:

$$(1998)_{10} = 1x10^3 + 9x10^2 + 9x10^1 + 8x10^0$$

Hệ thập phân (Decimal)

- Trong ngôn ngữ Assembly, người ta kí hiệu một số thập phân bằng chữ D hoặc d ở cuối (viết tắt của Decimal), cũng có thể không cần viết các chữ đó.
- Ví dụ:

$(1998)_{10}$ được kí hiệu là: 1998D, 1998d, hoặc đơn giản là 1998

Hệ nhị phân (Binary)

- Hệ nhị phân sử dụng 2 kí hiệu (0,1) để biểu diễn thông tin. Các số trong hệ nhị phân được biểu diễn dưới dạng tổng các luỹ thừa cơ số 2.
- Ví dụ: Số 1101 trong hệ nhị phân có thể biểu diễn như sau:

$$(1101)_2 = 1x2^3 + 1x2^2 + 0x2^1 + 1x2^0$$
$$= (13)_{10}$$

Hệ nhị phân (Binary)

- Trong ngôn ngữ Assembly, người ta kí hiệu một số nhị phân bằng chữ B hoặc b ở cuối (viết tắt của Binary).
- Ví dụ:

$(1101)_2$ được kí hiệu là: 1101B, hoặc 1101b

Hệ thập lục phân (Hexa Decimal)

- Hệ thập lục phân sử dụng 16 kí hiệu (0, 1, 2,...9, A, B, C, D, E, F) để biểu diễn thông tin.
- Các kí hiệu A, B, C, D, E, F lần lượt ứng với các giá trị 10, 11, 12, 13, 14, 15 trong hệ thập phân.
- Các số trong hệ thập lục phân được biểu diễn dưới dạng tổng các luỹ thừa cơ số 16.

Hệ thập lục phân (Hexa Decimal)

- Ví dụ: Số 2B trong hệ thập lục phân có thể biểu diễn như sau:

$$(2B)_{16} = 2 \times 16^1 + B \times 16^0$$

$$= (43)_{10}$$

Hệ thập lục phân (Hexa Decimal)

- Trong ngôn ngữ Assembly, người ta kí hiệu một số thập lục phân bằng chữ H hoặc h ở cuối (viết tắt của Hexa Decimal).
- Ví dụ:

  $(2B)_{16}$ được kí hiệu là: 2BH, hoặc 2Bh

Chú ý:

- Kí hiệu một số thập lục phân trong chương trình hợp Assembly phải luôn bắt đầu bằng số. Ví dụ số $(FA)_{16}$ được kí hiệu là 0FAh (chứ không kí hiệu là FAh).
- Hệ thập lục phân (gọi tắt là hệ hex) là hệ đếm được sử dụng nhiều nhất trong Assembly, do nó có thể biểu diễn những dãy bít nhị phân dài bằng những kí hiệu ngắn gọn, dễ nhớ hơn

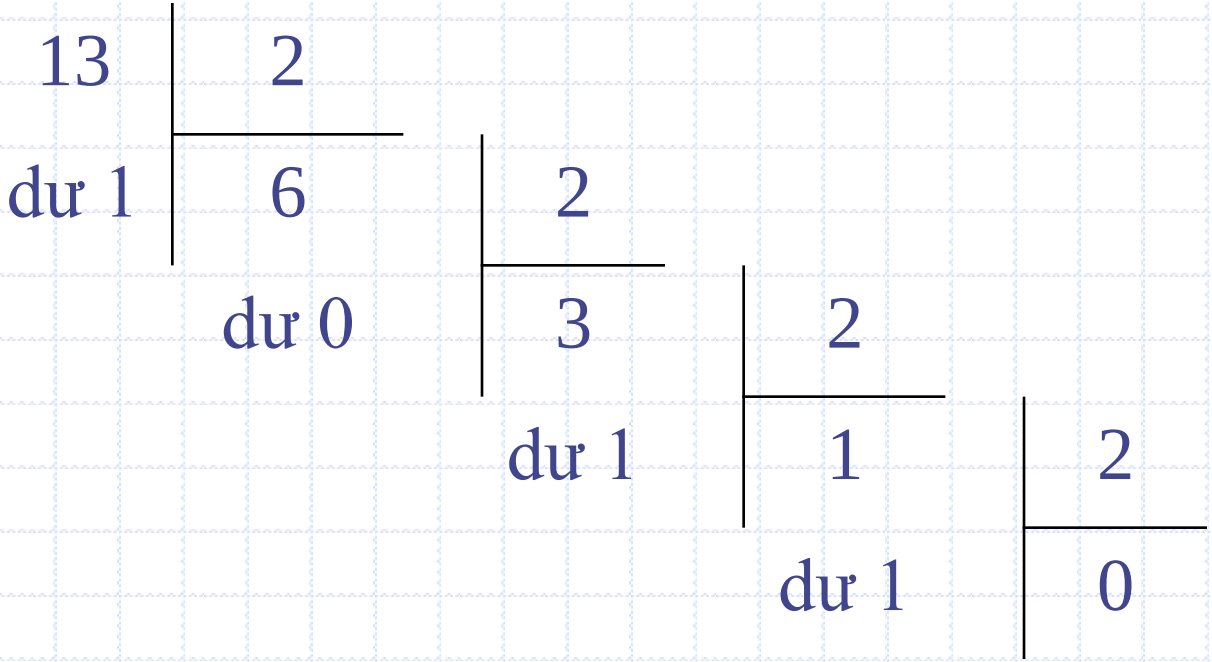
Chuyển đổi giữa các hệ đếm

- Chuyển từ hệ thập phân về hệ nhị phân:

  Đem số thập phân chia liên liếp cho 2, cho tới khi thương số bằng 0 thì dừng lại. Viết các số dư ngược từ dưới lên ta thu được số nhị phân tương ứng

Chuyển đổi giữa các hệ đếm

- Ví dụ: Chuyển số thập phân 13 sang hệ nhị phân.



Viết các số dư ngược từ dưới lên ta thu được số nhị phân 1101b

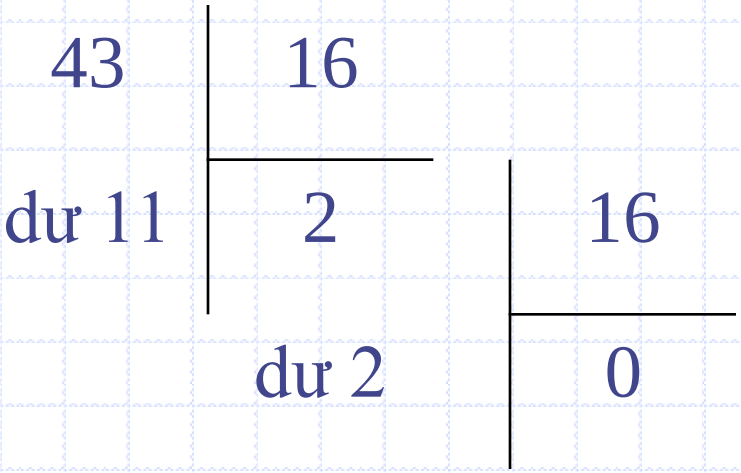
Chuyển đổi giữa các hệ đếm

- Chuyển từ hệ thập phân về hệ thập lục phân:

  Đem số thập phân chia liên liếp cho 16, cho tới khi thương số bằng 0 thì dừng lại. Viết các số dư ngược từ dưới lên ta thu được số thập lục phân tương ứng

Chuyển đổi giữa các hệ đếm

- Ví dụ: Chuyển số thập phân 43 sang hệ thập lục phân.



Viết các số dư ngược từ dưới lên ta thu được số thập lục phân 2Bh (chú ý là 11d = 0Bh).

Chuyển đổi giữa các hệ đếm

- Chuyển đổi giữa hệ nhị phân và hệ thập lục phân:

  Việc chuyển đổi giữa 2 hệ đếm này khá dễ dàng do mỗi kí hiệu trong hệ hex lại tương ứng với 4 kí hiệu nhị phân. Xem bảng chuyển đổi sau:

Bảng chuyển đổi

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

Chuyển đổi giữa các hệ đếm

- Ví dụ 1:

Chuyển đổi 2Ah sang hệ nhị phân.

Giải:

Tra bảng ta thấy: 2h = 0010b, Ah = 1010b

Vậy 2Ah = 00101010b

Chuyển đổi giữa các hệ đếm

- Ví dụ 2:

Chuyển đổi 10110110b sang hệ hex.

Giải:

Đầu tiên ta chia dãy bít nhị phân thành từng nhóm 4 bít, thu được 2 nhóm sau: 0110 và 1011.

Tra bảng ta thấy: 0110b = 6h, 1011b = Bh

Vậy 10110110b = B6h

Bài 2.2 - Biểu diễn số nguyên trong máy tính

- Dãy bít
- Số nguyên không dấu
- Số nguyên có dấu

Dãy bít

- Do giới hạn của phần cứng máy tính, dữ liệu trong máy tính thường được biểu diễn bởi các nhóm 8 bít (gọi là Byte)

  1 byte = 8 bit

  2 byte = 16 bit = 1 word

- Người ta có thể ghép nhiều byte hay nhiều word để tạo thành dãy bít dài hơn. Dãy bít càng dài thì lượng thông tin biểu diễn được càng lớn. Nếu gọi N là số bít của dãy thì số khả năng biểu diễn = 2^N

Dãy bít

- Xét một dãy bít nhị phân:

- Bít đầu tiên (bên trái) được gọi là bít nặng nhất hay bít cao nhất của dãy (Most Significant Bit).
- Bít cuối cùng (bên phải) được gọi là bít nhẹ nhất hay bít thấp nhất của dãy (Least Significant Bit).

Số nguyên không dấu

- Một dãy bít sẽ tương ứng với một số nguyên lớn hơn hoặc bằng 0
- Ví dụ: Biểu diễn số nguyên 13 trong máy tính.

  Ở phần trước ta đã biết: số nguyên 13 chuyển sang hệ nhị phân sẽ là 1101

  Trong máy tính sẽ có nhiều cách để biểu diễn số nguyên này:

+ Số nguyên dạng byte (8 bit): 00001101
+ Số nguyên dạng word (16 bit): 00000000 00001101

Số nguyên có dấu

- Một dãy bít sẽ tương ứng với một số nguyên, có thể âm hoặc dương.
- Khi biểu diễn dưới dạng nhị phân ta phải dành ra 1 bít để xác định dấu. Đó là bít đầu tiên của dãy (bít nặng nhất - Msb).

  + Msb = 0: Dấu Dương

  + Msb = 1: Dấu Âm
- Như vậy, nếu chiều dài dãy bít là 8 thì bít đầu tiên để xác định dấu, 7 bít còn lại xác định giá trị số nguyên?

Ví dụ:

- Số +13 được biểu diễn bởi dãy bít 0000 1101.

  Vậy số -13 được biểu diễn như thế nào, có phải là dãy bít 1000 1101 hay không?

- Nguyên tắc để biểu diễn số âm trong máy tính: phải thoả mãn điều kiện sau

  Số Âm (nhị phân) + Số Dương (nhị phân) = 0

- Giả sử số -13 được biểu diễn bởi dãy bít 1000 1101, ta đem nó cộng với dãy bít biểu diễn số +13 để kiểm tra:

$$\begin{array}{r} 0000\ 1101 \\ +\ 1000\ 1101 \\ \hline 1001\ 1010 \end{array} \quad \neq 0$$

- Ta thấy tổng thu được khác 0, như vậy đây không phải là dãy bít cần tìm

Quy tắc tìm số đối:

Cho 1 số nguyên A. Giả sử đã biết dãy bít biểu diễn A, khi đó muốn tìm dãy bít biểu diễn số -A ta làm như sau:

- Bước 1: Tìm số bù 1 của A bằng cách đảo tất cả các bít.
- Bước 2: Tìm số bù 2 (bằng cách lấy số bù 1 cộng với 1)

Số bù 2 tìm được chính là dãy bít biểu diễn số -A.

Ví dụ 1:

- Xét A = 13 = 0000 1101

  Khi đó số bù 1 của A là 1111 0010
- Tìm số bù 2 (bằng cách lấy số bù 1 cộng với 1)

$$\begin{array}{rr} & 1111\ 0010 \\ + & \qquad\quad 1 \\ \hline & 1111\ 0011 \end{array}$$

Như vậy -A = 1111 0011

- Kiểm tra lại bằng cách cộng 2 dãy bít:

$$\begin{array}{r} 0000\ 1101 \\ +\ 1111\ 0011 \\ \hline 1\ 0000\ 0000 \end{array}$$

- Kết quả thu được bằng 0 chứng tỏ ta đã tìm đúng

Vậy -13 = 1111 0011b

Chứng minh:

- Số -1 trong máy tính ứng với một dãy toàn bít 1:

  -1 = 1111 1111b

- Vì: 
- Nên: 
- Suy ra: 

Ví dụ 2:

- Cho một dãy bít nhị phân sau đây (16 bit):

  1110 0111 0001 1000b

  Hãy xác định xem nó biểu diễn số nguyên nào?

Giải:

Gọi số nguyên đó là N.

Có 2 trường hợp xảy ra:

- Nếu đây là số nguyên không dấu:

$$N = 1x2^{15} + 1x2^{14} + 1x2^{13} + 1x2^{10} + 1x\ 2^{9} + 1x2^{8} + 1x2^{4} + 1x2^{3}$$
$$= 32768 + 16384 + 8192 + 1024 + 512 + 256 + 16 + 8$$
$$= 59160$$

- Nếu đây là số nguyên có dấu:

  Vì Msb = 1 nên N là số âm. Để đơn giản ta sẽ xác định –N (số dương) trước, từ đó suy ra N.

- Tìm –N băng cách tìm số bù 2 của N
- Bước 1: đảo bít
- Số bù 1 = 0001 1000 1110 0111
- Bước 2: đem cộng với 1

$$\begin{array}{r} 0001\ 1000\ 1110\ 0111 \\ +\qquad\qquad\qquad\qquad\quad 1 \\ \hline 0001\ 1000\ 1110\ 1000 \end{array}$$

- Ta tìm được –N = 0001 1000 1110 1000b

$$-N = 1x2^{12} + 1x2^{11} + 1x\ 2^{7} + 1x2^{6} + 1x2^{5} + 1x2^{3}$$
$$= 4096 + 2048 + 128 + 64 + 32 + 8$$
$$= 6376$$

Vậy N = -6376

Bài 2.3 - Biểu diễn kí tự trong máy tính

- Các kí tự cũng được biểu diễn bởi các dãy bít nhị phân
- Cần phải có một hệ thống quy ước chung mang tính quốc tế
- Cách thức mã hoá kí tự theo bảng mã ASCII (American Standard Code for Information Interchange) được sử dụng phổ biến nhất. Hệ thống này sử dụng 8 bít để biểu diễn 1 kí tự

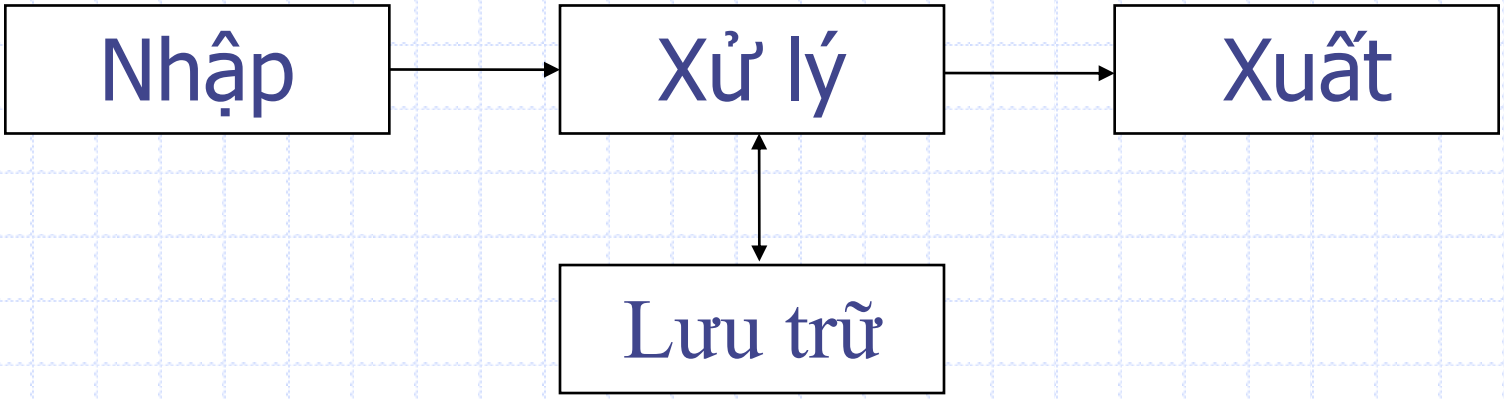
Bảng mã ASCII

Kí tự	Mã nhị phân	Mã hex
A	0100 0001	41h
B	0100 0010	42h
...	...	...
a	0110 0001	61h
b	0110 0010	62h
...	...	...
1	0011 0001	31h
2	0011 0010	32h
...	...	...
*	0010 1010	2Ah
+	0010 1011	2Bh
...	...	...

Bài 2.4 - Tổ chức CPU

- Các thành phần chính của bộ vi xử lý
- Họ vi xử lý Intel x86
- Bộ vi xử lý 8086

Nhắc lại về khối xử lý



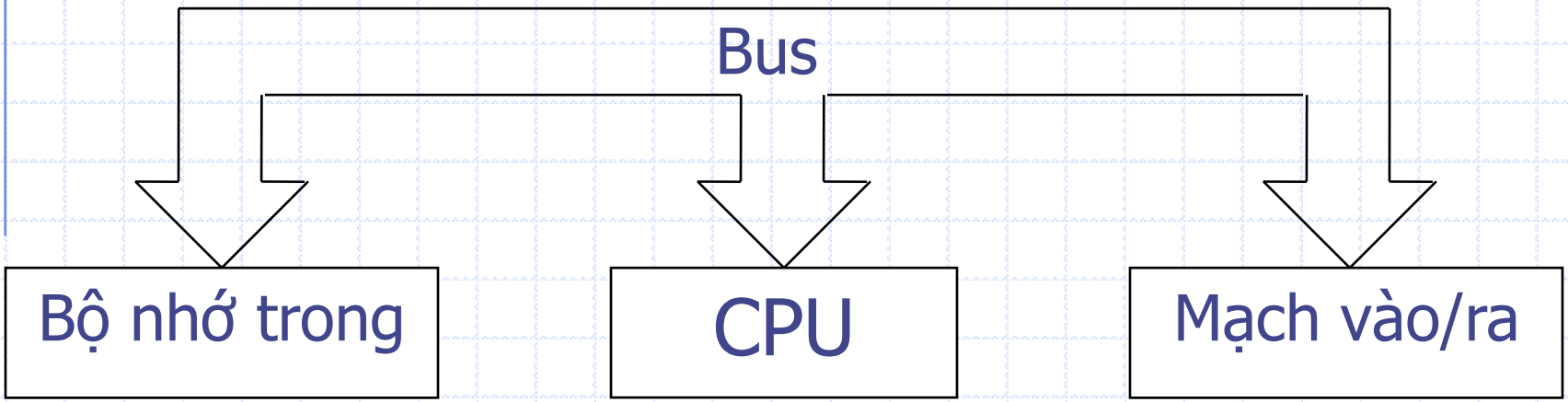
Khối xử lý

Đây là phần quan trọng nhất của một máy tính, bao gồm 3 bộ phận chính:

- Bộ vi xử lý – CPU
- Bộ nhớ trong
- Các mạch vào ra

Ngoài ra còn có hệ thống các dây dẫn, cáp nối để liên kết giữa các bộ phận trên (hệ thống Bus)

Sơ đồ khối xử lý:



Các thành phần của khối xử lý

- Bộ vi xử lý – CPU (Central Processing Unit): Là bộ não của máy tính, nó xử lý các thông tin và điều khiển mọi hoạt động của máy tính.
- Bộ nhớ trong: Là bộ nhớ có khả năng liên lạc trực tiếp với bộ vi xử lý,là nơi lưu trữ dữ liệu phục vụ cho quá trình xử lý.
- Các mạch vào ra: Để điều khiển việc giao tiếp với thiết bị ngoại vi.

Các thành phần chính của bộ vi xử lý

- ALU (Arithmetic & Logic Unit): Khối số học và logic. Đây là nơi thực hiện các phép tính số học (cộng, trừ, nhân, chia...) và các phép logic (Not, And, Or...).
- Các thanh ghi: Cung cấp khả năng nhớ bên trong CPU. Mỗi thanh ghi có khả năng chứa được một dãy các bít dữ liệu (độ dài còn phụ thuộc vao từng loại CPU).
- Hệ thống nối ghép bên trong CPU (Bus nội bộ): Cho phép liên lạc giữa các bộ phận bên trong CPU.

Họ vi xử lý Intel x86

- Bộ vi xử lý đầu tiên thuộc dòng này là 8086, ra đời năm 1978, là bộ vi xử lý 16 bit đầu tiên của Intel.
- 8088 ra đời sau 8086, về cơ bản nó cũng giống như 8086, nhưng có giá thành rẻ hơn vì chỉ có bus dữ liệu 8 bít, và tốc độ cũng thấp hơn
- Tiếp theo là các bộ vi xử lý 80186, 80286, 80386, 80486, 80586 (Pentium), PII, PIII, P4, Core Duo...
- Các bộ vi xử lý ngày càng trở nên mạnh mẽ hơn với độ dài các thanh ghi lớn hơn, tốc độ đồng hồ cao hơn, bề rộng bus lớn hơn...

Bộ vi xử lý 8086

- 8086 có cấu trúc đơn giản, dễ tìm hiểu
- Hầu hết các lệnh của nó đều được các bộ vi xử lý sau này kế thừa
- Các chương trình viết cho 8086 vẫn có thể chạy trên các bộ vi xử lý hiện đại hơn

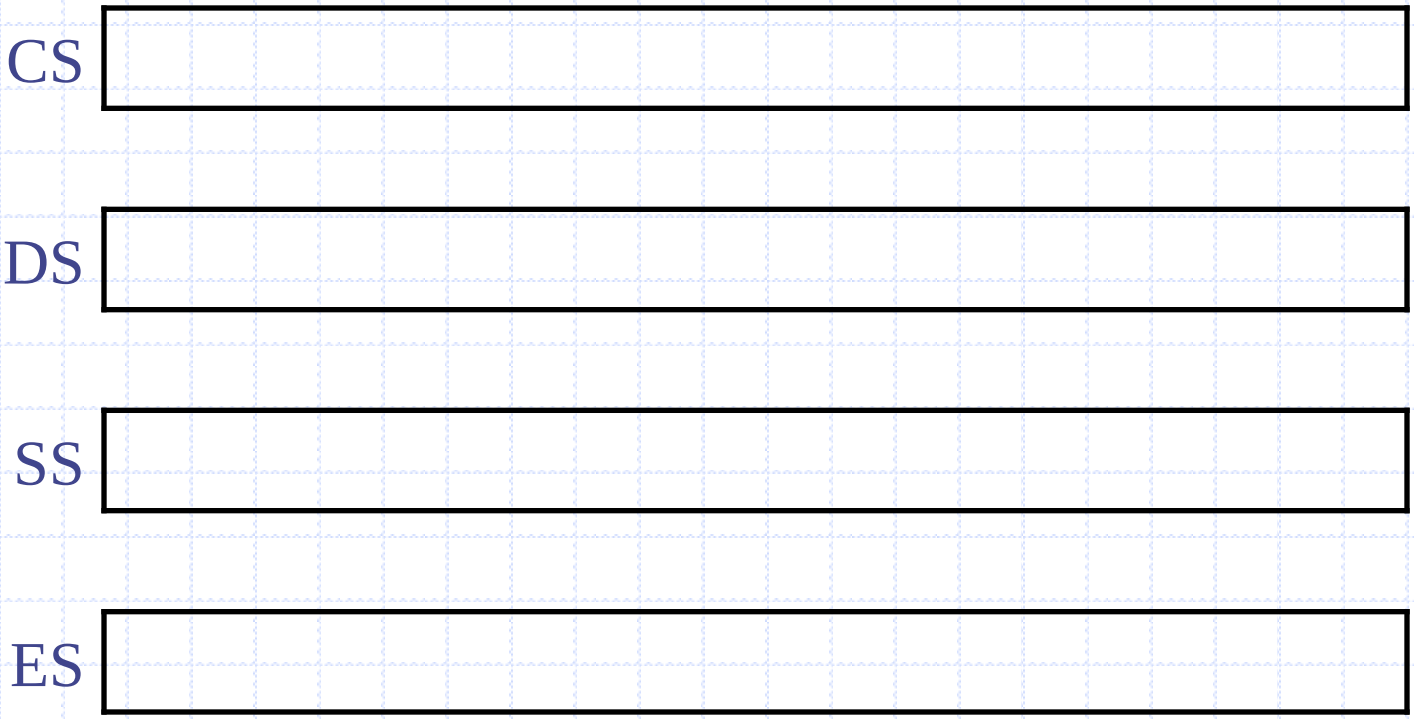
14 thanh ghi cơ bản của 8086

(Mỗi thanh ghi dài 16 bít)

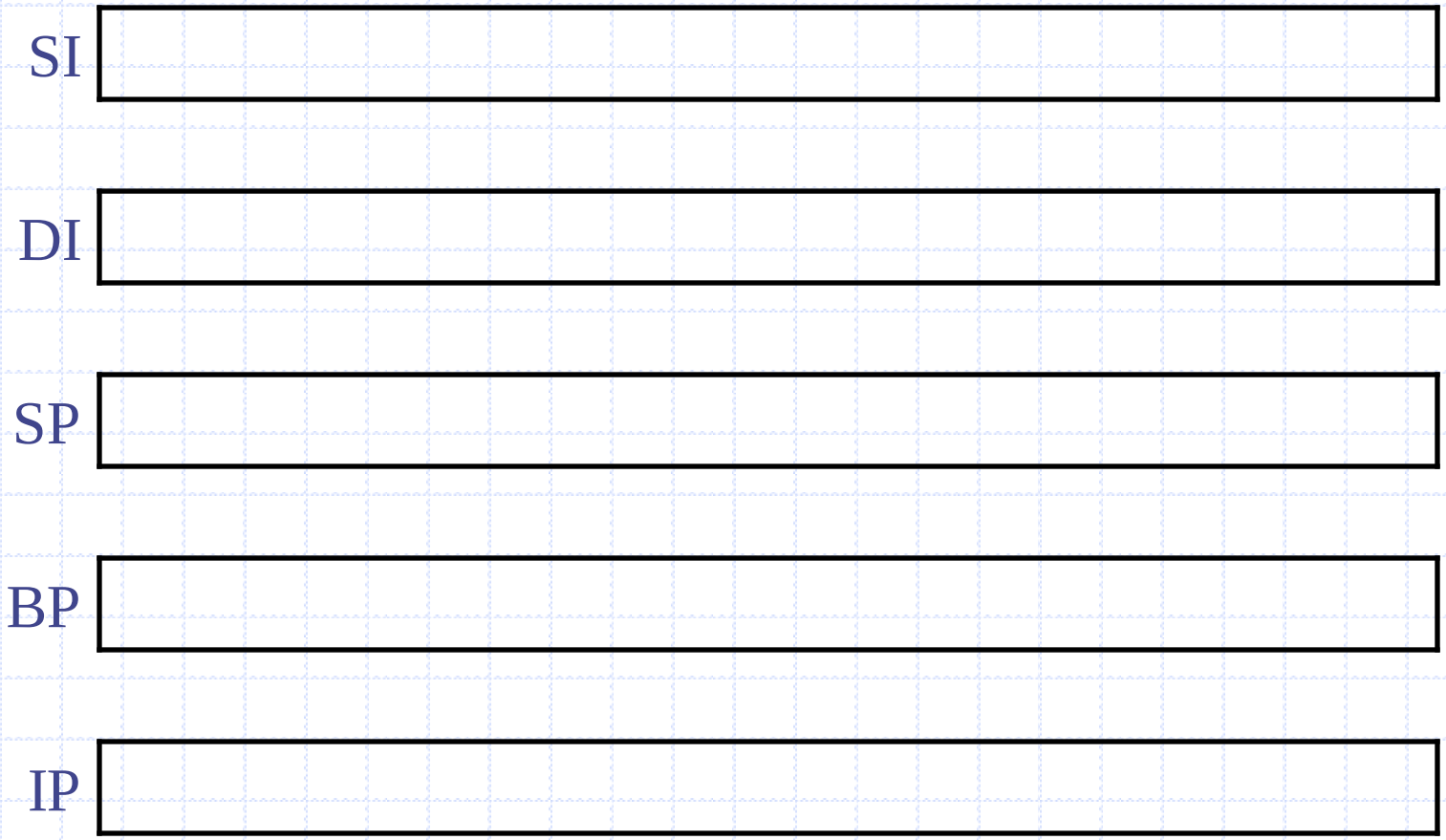
- Nhóm các thanh ghi dữ liệu (Thanh ghi công dụng chung):

AX	AH	AL
BX	BH	BL
CX	CH	CL
DX	DH	DL

- Nhóm các thanh ghi đoạn:



- Nhóm các thanh ghi con trỏ và chỉ số:



Thanh ghi trạng thái (Thanh ghi cờ - Flag):

Thanh ghi này dùng để xác định trạng thái của bộ vi xử lý. Mỗi bít trên thanh ghi cờ có một tên riêng, có một công dụng riêng trong việc phản ánh trạng thái

Bài 2.5 - Tổ chức Bộ nhớ trong

- ROM và RAM
- Địa chỉ vật lý
- Địa chỉ logic
- Sự phân chia không gian nhớ

Bộ nhớ trong có 2 loại: ROM và RAM

- RAM (Random Access Memory): Bộ nhớ truy cập ngẫu nhiên, có thể đọc và ghi dữ liệu lên đó. Dữ liệu trên RAM sẽ mất đi khi tắt máy.
- ROM (Read Only Memory): Bộ nhớ chỉ đọc, không thể thay đổi nội dung của nó. Khi tắt máy thì dữ liệu trên ROM vẫn được giữ nguyên.

Địa chỉ vật lý

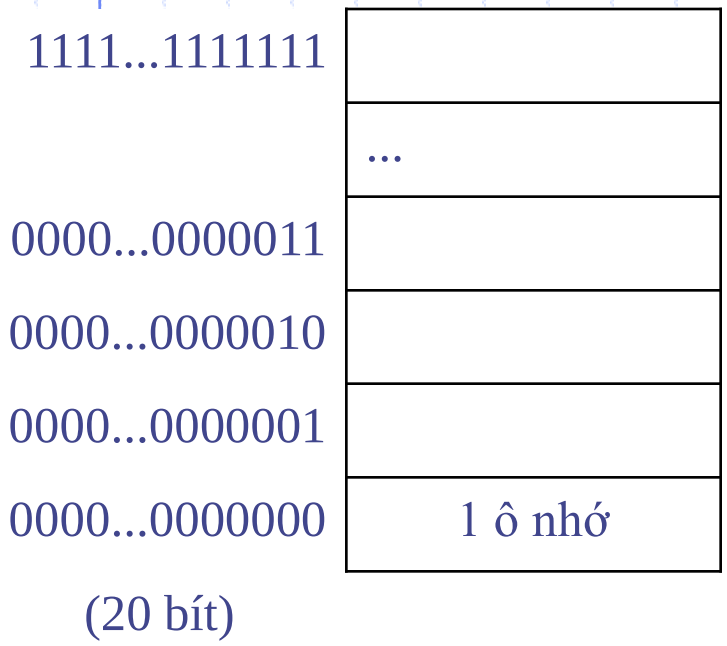
- Bộ nhớ của máy tính được cấu tạo bởi các phần tử nhớ 1 bit.
- Cứ 8 phần tử nhớ tạo thành một ô nhớ (1 byte).
- Các ô nhớ được sắp xếp tuần tự trong bộ nhớ và được đánh số lần lượt từ 0, 1, 2... Số hiệu các ô nhớ như trên được gọi là địa chỉ vật lý của ô nhớ.

Tổ chức bộ nhớ trong hệ thống 8086

- Bộ vi xử lý 8086 sử dụng 20 đường dây địa chỉ (bus địa chỉ 20 bít) để liên lạc với bộ nhớ

  (địa chỉ các ô nhớ là một dãy dài 20 bít)
- Số lượng cực đại các ô nhớ có thể đánh địa chỉ là 2^{20} (= 1MB)

Địa chỉ vật lý của các ô nhớ



Địa chỉ dạng hex:

Địa chỉ logic

- Các thanh ghi của 8086 chỉ dài 16 bít, không thể chứa được địa chỉ dài 20 bít. Do đó người ta phải sử dụng một phương pháp đánh địa chỉ khác, gọi là địa chỉ logic
- Bộ nhớ được chia thành từng đoạn, mỗi đoạn có chiều dài tối đa 64 KB, các đoạn được đánh số lần lượt là 0, 1, 2,...
- Các ô nhớ trong đoạn cũng được đánh số lần lượt là 0, 1, 2,... (cực đại là 65535)

Địa chỉ logic

- Địa chỉ logic của một ô nhớ sẽ gồm 2 phần:

  Số hiệu đoạn (segment) và vị trí của ô nhớ trong đoạn (offset)
- Người ta sử dụng 16 bít để đánh số các đoạn, như vậy địa chỉ segment sẽ nằm trong phạm vi từ 0000h đến FFFFh
- Người ta cũng sử dụng 16 bít để đánh số các ô nhớ trong đoạn đoạn, như vậy địa chỉ offset sẽ nằm trong phạm vi từ 0000h đến FFFFh,

Ví dụ:

- Một ô nhớ có địa chỉ segment:offset = 10A2:34B4h, hãy xác định địa chỉ vật lý của nó

Giải:

- Bước 1: Dịch địa chỉ segment về bên trái 4 bít (tương đương với dịch 1 kí hiệu hex).

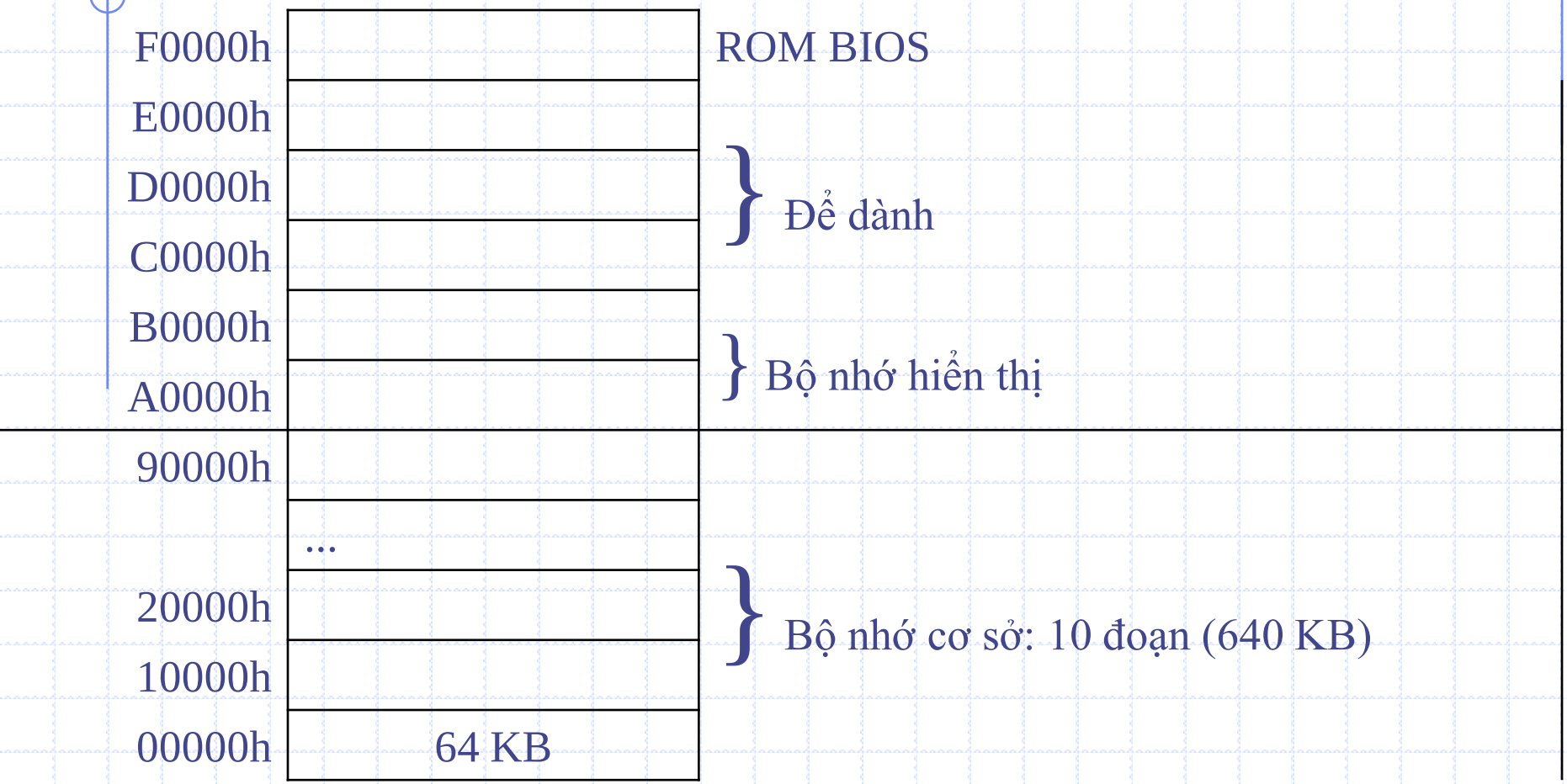
  Ta thu được: 10A20h

- Bước 2: Lấy giá trị thu được ở bước 1 đem cộng với địa chỉ offset:

$$\begin{array}{r} 10A20h \\ +\ 34B4h \\ \hline 13ED4h \end{array}$$

- Vậy địa chỉ vật lý của ô nhớ đó là 13ED4h

Sự phân chia không gian nhớ



Hết Phần 2.1

KIẾN TRÚC MÁY TÍNH

Giảng viên: Ths Phạm Thanh Bình
Bộ môn Kỹ thuật máy tính & mạng
http://vn.myblog.yahoo.com/CNTT-wru
http://ktmt.wru.googlepages.com

Bài 2.6 – Các lệnh Assembly cơ bản

- Cấu trúc chương trình
- Một số lệnh thường dùng
- Nhập/xuất dữ liệu
- Cách chạy chương trình Assembly
- Các lệnh nhảy, cấu trúc rẽ nhánh và lặp
- Các lệnh logic, dịch, và quay
- Ngăn xếp và thủ tục
- Mảng và các chế độ địa chỉ

2.6.1 Cấu trúc chương trình

```
TITLE  VI DU 1
.MODEL   SMALL
.STACK   100H
.DATA
   A   DB   4
   B   DB   6
   C   DB   ?
.CODE
   MAIN   PROC
      MOV   AX, @DATA
      MOV   DS, AX
      MOV   AL, A
      ADD   AL, B
      MOV   C, AL
      MOV   AX, 4C00H
      INT   21H
   MAIN ENDP
END MAIN
```

Giải thích:

- ***Phần tên***

  Tên chương trình được viết sau từ khoá TITLE ở đầu chương trình. Tên có thể chứa dấu cách và các kí tự đặc biệt khác. Thông thường phần tên sẽ cho ta biết mục đích, nhiệm vụ hoặc nội dung tóm tắt của chương trình.

  Ví dụ: TITLE VI DU 1

- ***Phần khai báo***

  Trong hợp ngữ có nhiều nội dung cần phải khái báo như kiểu bộ nhớ, ngăn xếp, biến, hằng...

- Khai báo kiểu bộ nhớ

  Kiểu bộ nhớ được viết sau từ .MODEL. Kiểu bộ nhớ sẽ quy định kích thước của đoạn mã và dữ liệu trong chương trình
- Trong ví dụ trên, kiểu bộ nhớ là SMALL, nghĩa là kiểu bộ nhớ nhỏ, mã lệnh sẽ nằm trong 1 đoạn nhớ, dữ liệu nằm trong 1 đoạn nhớ. Ngoài kiểu SMALL còn có nhiều kiểu bộ nhớ khác

Một số kiểu bộ nhớ

MEDIUM	Mã lệnh chiếm nhiều hơn 1 đoạn Dữ liệu trong 1 đoạn
COMPACT	Mã lệnh trong 1 đoạn Dữ liệu chiếm nhiều hơn 1 đoạn
LARGE	Mã lệnh chiếm nhiều hơn 1 đoạn Dữ liệu chiếm nhiều hơn 1 đoạn Không có mảng nào lớn hơn 64 KB
HUGE	Mã lệnh chiếm nhiều hơn 1 đoạn Dữ liệu chiếm nhiều hơn 1 đoạn Các mảng có thể lớn hơn 64 KB

Khai báo kích thước ngăn xếp

Kích thước ngăn xếp được viết sau từ .STACK

Ví dụ: .STACK 100H

Khi đó kích thước vùng bộ nhớ dùng làm ngăn xếp là 100H Bytes

Khai báo dữ liệu

Khai báo dữ liệu được viết sau từ .DATA. Các biến của chương trình sẽ được khai báo ở phần này.

Ví dụ 1: A DB 4

Trong đó: A là tên biến, DB là kiểu dữ liệu (biến kiểu DB sẽ có kích thước 1 byte), 4 là giá trị ban đầu của biến.

Ví dụ 2: B DW ?

Trong ví dụ này, biến B sẽ có kiểu là DW (1 Word) và không có giá trị khởi tạo.

Khai báo hằng:

Ví dụ: H EQU 2Bh

Hằng H sẽ nhận giá trị bằng 2Bh

Quy tắc đặt tên (Biến, hằng, nhãn, thủ tục):

- Tên có chiều dài tối đa 31 kí tự.
- Có thể chứa chữ cái, chữ số và các kí tự đặc biệt (? . @ _ $ %).
- Không được bắt đầu bằng số.
- Nếu dùng dấu chấm thì nó phải đứng đầu tiên.
- Tên không được chứa dấu cách.

- ***Phần mã lệnh***

  Phần này bao gồm các thủ tục được viết sau từ .CODE. Trong số các thủ tục này phải chọn một thủ tục làm chương trình chính, tên của thủ tục đó được viết sau từ END ở cuối chương trình. Tên chương trình chính thường đặt là MAIN, cũng có thể chọn một tên khác.

  Cấu trúc một thủ tục:

  <Tên thủ tục> PROC

  Lệnh 1

  Lệnh 2

  Lệnh 3

  ...

  <Tên thủ tục> ENDP

Cấu trúc chung của phần mã lệnh:

```
.CODE
<Tên chương trình chính>    PROC
        Lệnh 1
        Lệnh 2
        Lệnh 3
        ...
<Tên chương trình chính>    ENDP
...Các thủ tục khác
END   <Tên chương trình chính>
```

2.6.2 Một số lệnh thường dùng

- Cấu trúc câu lệnh Assembly
- Lệnh MOV (Move)
- Lệnh XCHG (Exchange)
- Lệnh ADD và SUB (Subtract)
- Lệnh INC (Increment) và DEC (Decrement)
- Lệnh NEG (Negartive)

Cấu trúc câu lệnh Assembly

- Một câu lệnh hợp ngữ gồm 3 phần: Tên lệnh, các toán hạng, phần chú thích.
- Nếu có nhiều toán hạng thì chúng được phân cách với nhau bằng dấu phảy (,).

<Tên lệnh> <Toán hạng 1> [, <Toán hạng 2>...] [;Lời chú thích]

- Ví du:

MOV DS, AX ;Chuyển nội dung của thanh ghi AX vào thanh ghi DS

Lệnh MOV (Move)

- Lệnh này được sử dụng để chuyển dữ liệu giữa các thanh ghi hay ô nhớ.
- Cú pháp lệnh:

MOV <Đích>, <Nguồn>

<Đích>: là một thanh ghi hay một ô nhớ

<Nguồn>: là một thanh ghi, một ô nhớ, hoặc một hằng số

- Dữ liệu sẽ được chuyển từ Nguồn vào Đích (nội dung của Nguồn không thay đổi sau khi chuyển)

- Ví dụ 1:

```
MOV   AX, 4C00h
```

Lệnh trên chuyển giá trị 4C00h vào thanh ghi AX.

- Ví dụ 2:

```
MOV   AL, A
```

Lệnh trên chuyển giá trị của biến A vào thanh ghi AL.

Chú ý:

- Không được chuyển trực tiếp nội dung của hai thanh ghi đoạn cho nhau
- Không được chuyển trực tiếp nội dung của hai biến cho nhau
- Không được chuyển trực tiếp một hằng số vào một thanh ghi đoạn

Ví dụ:

- Các lệnh sau đều sai:

```
MOV   CS, DS
MOV   A, B
MOV   DS, 2000h
```

- Để khắc phục: sử dụng trung gian:

```
MOV   AX, DS        ; dùng AX làm trung gian
MOV   CS, AX
```

Lệnh XCHG (Exchange)

- Lệnh này dùng để hoán đổi dữ liệu dữ liệu giữa hai toán hạng.
- Cú pháp lệnh:

  XCHG <Toán hạng 1>, <Toán hạng 2>

- Các toán hạng có thể là thanh ghi công dụng chung, hoặc một thanh ghi công dụng chung và một ô nhớ.

- Ví dụ:

```
XCHG   AX, BX      ;hoán đổi nội dung của
                   ; AX và BX
```

Lệnh ADD và SUB (Subtract)

- Cú pháp lệnh:

  ADD <Đích>, <Nguồn>

  SUB <Đích>, <Nguồn>

  <Đích>: là một thanh ghi hay một ô nhớ

  <Nguồn>: là một thanh ghi, một ô nhớ, hoặc một hằng số

  <Đích>,<Nguồn> không đồng thời là hai ô nhớ.
- Lệnh ADD sẽ thực hiện phép cộng Đích với Nguồn, kết quả chứa trong Đích.
- Lệnh SUB sẽ lấy Đích trừ đi Nguồn, kết quả chứa trong Đích.

Ví dụ:

```
ADD   AX, 10     ;Tăng nội dung của
                 ; thanh ghi AX lên 10
ADD   BX, AX     ;Cộng nội dung 2 thanh ghi
                 ; AX và BX, tổng cất vào BX
SUB   AX, B      ;Trừ nội dung thanh ghi AX
                 ; cho biến B
```

Lệnh INC (Increment) và DEC (Decrement)

- Cú pháp lệnh:

  INC <Đích>

  DEC <Đích>

  <Đích>: là một thanh ghi hay một ô nhớ

- Lệnh INC sẽ tăng Đích lên 1 (cộng Đích với 1).
- Lệnh DEC sẽ giảm Đích đi 1 (trừ Đích cho 1).

- Ví dụ:

```
INC   AH      ;Cộng nội dung của
              ; thanh ghi AH với 1
DEC   B       ;Trừ giá trị của biến B
              ; cho 1
```

Lệnh NEG (Negartive)

- Cú pháp lệnh:

NEG <Đích>

<Đích>: là một thanh ghi hay một ô nhớ.

- Lệnh này có tác dụng đổi dấu toán hạng Đích.
- Ví dụ:

NEG AL

2.6.3 Nhập/xuất dữ liệu

- Chương trình ngắt
- Chức năng nhập/xuất của ngắt 21h

Chương trình ngắt

- Chương trình ngắt là những chương trình con đã được viết sẵn nhằm thực hiện những chức năng cơ bản khi thao tác với máy tính.
- Các chương trình con này được phân phối kèm theo các phần mềm điều khiển hệ thống như BIOS, Hệ điều hành.
- Mỗi chương trình có một số hiệu riêng (0, 1, 2, ...). Khi lập trình ta có thể sử dụng các chương trình con có sẵn này bằng cách dùng lệnh INT (interrupt).
- Cú pháp lệnh:

INT <Số hiệu ngắt>

- Ví dụ 1:

  INT 21h

  Lệnh trên sẽ gọi thực hiện chương trình ngắt số 21h (đây là số hiệu ngắt hay sử dụng nhất của DOS).

- Ví dụ 2:

  INT 13h

  Lệnh trên sẽ gọi thực hiện chương trình ngắt số 13h (đây là số hiệu ngắt của BIOS, dùng để thao tác với đĩa từ).

Chức năng nhập/xuất của ngắt 21h

- Ngắt 21h của DOS cung cấp rất nhiều chức năng khác nhau, mỗi chức năng cũng có một số hiệu riêng (0, 1, 2, ...). Trong phần này ta chỉ quan tâm tới chức năng nhập - xuất dữ liệu.
- Trước khi gọi ngắt cần xác định được số hiệu chức năng, số hiệu đó được đặt vào thanh ghi AH. Ngoài ra cũng cần quan tâm tới các tham số khác (chương trình ngắt sẽ sử dụng thanh ghi nào? Giá trị của chúng bằng bao nhiêu?...)

Hiện một kí tự ra màn hình

- Đây là chức năng số 2 của ngắt 21h. Các tham số cần thiết để gọi ngắt như sau:

Vào: AH = 2
DL = Mã ASCII của kí tự cần hiển thị

Ra: AL chứa mã ASCII của kí tự hiển thị

Ví dụ 2: Hiện kí tự ‘M’ ra màn hình

```
MOV   AH, 2
MOV   DL, ‘M’
INT   21h
```

Chương trình đầy đủ

```
TITLE  HIEN KI TU
.MODEL  SMALL
.STACK  100H
.CODE
MAIN   PROC
    MOV   AH, 2       ;Chức năng số 2
    MOV   DL, 'M'     ;Kí tự cần hiển thị
    INT   21h         ;Gọi ngắt
    MOV   AH, 4Ch     ;Kết thúc
    INT   21h
MAIN ENDP
END MAIN
```

Chú ý:

- Chương trình trên có sử dụng chức năng số 4Ch của ngắt 21h. Chức năng này có tác dụng kết thúc chương trình và trả lại quyền điều khiển cho hệ điều hành DOS.

Hiện một chuỗi kí tự ra màn hình

- Đây là chức năng số 9 của ngắt 21h. Các tham số cần thiết để gọi ngắt như sau:

Vào: AH = 9

DX = Địa chỉ offset của vùng nhớ chứa chuỗi kí tự

- Ví dụ:

  Hiện ra màn hình dòng chữ:

  KHOA CONG NGHE THONG TIN

```
TITLE  HIEN CHUOI KT
.MODEL  SMALL
.STACK  100H
.DATA
    ChuoiKT   DB   'KHOA CONG NGHE THONG TIN$'
.CODE
MAIN   PROC
    MOV   AX, @DATA
    MOV   DS, AX
    MOV   AH, 9          ;Chức năng số 9
    LEA   DX, ChuoiKT  ;Lấy địa chỉ chuỗi kí tự đặt vào DX
    INT   21h            ;Gọi ngắt
    MOV   AH, 4Ch       ;Kết thúc
    INT   21h
MAIN ENDP
END MAIN
```

Chú ý:

- Chuỗi kí tự cần hiển thị phải được kết thúc bằng dấu $.
- Nếu trong chương trình có sử dụng khai báo dữ liệu .DATA thì ở đầu của chương trình chính phải có các lệnh:

```
MOV   AX, @DATA
MOV   DS, AX
```

Mục đích là để đặt địa chỉ segment của đoạn dữ liệu vào thanh ghi DS.

Chú ý:

- Chương trình trên có sử dụng lệnh LEA (Load Effective Address). Cú pháp lệnh như sau:

  LEA <Đích>, <Nguồn>

  <Đích>: là một thanh ghi công dụng chung hoặc thanh ghi con trỏ - chỉ số.

  <Nguồn>: là một ô nhớ.

- Lệnh này sẽ lấy địa chỉ offset của Nguồn đặt vào Đích.

Nhập một kí tự từ bàn phím

- Đây là chức năng số 1 của ngắt 21h. Các tham số cần thiết để gọi ngắt như sau:

  Vào: AH = 1

  Ra: AL chứa mã ASCII của kí tự

- Các lệnh cụ thể như sau:

```
MOV   AH, 1
INT   21h
```

- Khi gặp các lệnh trên, chương trình sẽ dừng lại chờ ta gõ một kí tự từ bàn phím, mã ASCII của kí tự đó sẽ được cất trong thanh ghi AL.

Ví dụ:

- Nhập một kí tự thường từ bàn phím, đổi nó thành kí tự in hoa rồi hiện ra màn hình.

```
TITLE  DOI KI TU
.MODEL  SMALL
.STACK  100H
.CODE
MAIN   PROC
    MOV   AH, 1      ;Chức năng số 1: Nhập một kí tự
    INT   21h
    SUB   AL, 20h    ;Đổi kí tự sang in hoa
    MOV   AH, 2      ;Chức năng số 2: Hiện kí tự
    MOV   DL, AL
    INT   21h
    MOV   AH, 4Ch     ;Kết thúc
    INT 21h
MAIN ENDP
END MAIN
```

Giải thích:

Mã ASCII của kí tự thường lớn hơn kí tự in hoa tương ứng là 20h. Muốn chuyển từ kí tự thường thành in hoa thì chỉ việc lấy mã ASCII của nó trừ đi 20h.

2.6.4 Cách chạy chương trình Assembly

- Để có thể chạy một chương trình hợp ngữ thì trước hết phải biên dịch nó thành file thi hành (EXE, COM).
- Có nhiều công cụ biên dịch khác nhau do nhiều hãng phần mềm sản xuất. Ta sẽ sử dụng bộ công cụ MASM (Microsoft Macro Assembler) của hãng MicroSoft vì nó khá nhỏ gọn và dễ dùng.
- Để bắt đầu thì ta chỉ cần tới hai file: MASM.EXE và LINK.EXE.

Các bước thực hiện:

- Bước 1:

  Soạn thảo nội dung chương trình bằng một công cụ soạn thảo text bất kì, cất vào file với phần mở rộng là ASM.

  Ví dụ: tên file là Baitap.asm

- Bước 2:

  Dịch file ASM thành file OBJ bằng công cụ MASM.EXE. Gõ lệnh như sau:

  MASM Baitap; (Enter)

  (file MASM.EXE và file Baitap.asm nên để cùng một thư mục)
- Nếu dịch thành công (chương trình không có lỗi) thì ta sẽ thu được file Baitap.obj.
- Nếu chương trình bị lỗi thì phải sửa, sau đó tiến hành dịch lại.

- Bước 3:

  Sử dụng công cụ LINK.EXE để liên kết các file OBJ thu được ở bước 2 thành file thi hành được (EXE). Vì trong ví dụ này chỉ có 1 file OBJ nên cách gõ lệnh như sau:

  LINK Baitap; (Enter)

  (file LINK.EXE cũng ở cùng thư mục nói trên)

- Ta sẽ thu được file Baitap.exe. Để chạy file này chỉ việc gõ lệnh:

Baitap (Enter)

2.6.5 Các lệnh nhảy

- Thanh ghi cờ và các cờ trạng thái
- Các lệnh nhảy có điều kiện
- Lệnh nhảy không điều kiện JMP
- Cấu trúc rẽ nhánh
- Cấu trúc lặp

Thanh ghi cờ và các cờ trạng thái

- Thanh ghi cờ dài 16 bít, mỗi bít được gọi là một cờ và có công dụng riêng. Dưới đây là vị trí của các cờ:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
				OF	DF	IF	TF	SF	ZF		AF		PF		CF

Tên và chức năng của các cờ

Bit	Tên cờ	Kí hiệu
0	Cờ nhớ (Carry Flag)	CF
2	Cờ chẵn lẻ (Parity Flag)	PF
4	Cờ nhớ phụ (Auxiliary Flag)	AF
6	Cờ Zero (Zero Flag)	ZF
7	Cờ dấu (Sign Flag)	SF
11	Cờ tràn (OverFlow Flag)	OF
8	Cờ bẫy (Trap Flag)	TF
9	Cờ ngắt (Interrupt Flag)	IF
10	Cờ định hướng (Direction Flag)	DF

Các cờ chia làm hai nhóm khác nhau:

- Nhóm cờ trạng thái (gồm 6 cờ: CF, PF, AF, ZF, SF, OF)
- Nhóm cờ điều khiển (gồm 3 cờ: TF, IF, DF)

Dưới đây ta sẽ tìm hiểu một số cờ hay dùng: CF, ZF, và OF

Cờ nhớ CF:

- Xét các lệnh sau đây:

```
MOV   AX, 0FFFFh
ADD   AX, 1
```

Trước khi thực hiện lệnh ADD thì AX = FFFFh = 1111 1111 1111 1111b = 65535

- Sau khi thực hiện phép cộng với 1 thì AX bằng bao nhiêu?

$$\begin{array}{r} 1111\ 1111\ 1111\ 1111b \\ + \qquad\qquad\qquad\qquad\quad 1 \\ \hline 1\ 0000\ 0000\ 0000\ 0000b \end{array}$$

- Thanh ghi AX dài 16 bit nên sau lệnh ADD thì AX = 0! Phép cộng đã không còn chính xác do kết quả vượt quá phạm vi chứa của AX (gọi là hiện tượng tràn khi cộng số không dấu). Khi đó cờ CF được thiết lập bằng 1.
- Như vậy, cờ CF sẽ được thiết lập khi thực hiện phép cộng có nhớ ở bít Msb hoặc khi thực hiện phép trừ có vay ở bít Msb.

Cờ Zero ZF:

- Xét các lệnh sau đây:

```
MOV   CX, 2Ah
SUB   CX, 2Ah
```

Sau khi thực hiện lệnh SUB thì CX = 0, cờ ZF được thiết lập bằng 1.

- Như vậy, cờ ZF sẽ được thiết lập khi kết quả của lệnh vừa thực hiện bằng 0.

Cờ tràn OF:

- Xét các lệnh sau đây:

```
MOV   AX, 7FFFh
ADD   AX, 7FFFh
```

Trước khi thực hiện lệnh ADD thì AX = 7FFFh = 0111 1111 1111 1111b = 32767

- Sau khi thực hiện phép cộng thì AX bằng bao nhiêu?

$$\begin{array}{r} 0111\ 1111\ 1111\ 1111b \\ +\ 0111\ 1111\ 1111\ 1111b \\ \hline 1111\ 1111\ 1111\ 1110b \end{array}$$

- Sau lệnh ADD thì AX = FFFEh. Nếu coi đây là số không dấu thì AX = 65534, không có hiện tượng tràn, cờ CF = 0. Nhưng nếu coi đây là số có dấu thì AX = -2 (32767 + 32767 = -2!), phép cộng đã không còn chính xác do kết quả vượt quá phạm vi chứa của AX (gọi là hiện tượng tràn khi cộng số có dấu). Khi đó cờ OF được thiết lập bằng 1.
- Như vậy, cờ OF sẽ được thiết lập khi xuất hiện hiện tượng tràn trong phép tính với số có dấu.

Các lệnh nhảy có điều kiện

Ví dụ:

```
TITLE Lenh nhay
.MODEL  SMALL
.STACK  100H
.CODE
MAIN   PROC
NHAPLAI:
    MOV   AH, 1       ;Chức năng số 1: Nhập 1 kí tự
    INT   21h

    CMP   AL, ' '     ;Kiểm tra kí tự vừa nhập
    JZ    NHAPLAI
    ...
    MOV   AH, 4Ch     ;Kết thúc
    INT   21h
MAIN ENDP
END MAIN
```

Giải thích:

- Chương trình trên sẽ nhập một kí tự từ bàn phím, kiểm tra xem đó có phải là kí tự khoảng trống ‘ ’ hay không, nếu đúng thì tiến hành nhập lại. Quá trình đó được thực hiện nhờ lệnh so sánh CMP và lệnh nhảy JZ.

Lệnh CMP (Compare)

- Cú pháp lệnh:

  CMP <Đích>, <Nguồn>

  Lệnh này có tác dụng tương tự lệnh SUB, nó thực hiện phép trừ giữa Đích và Nguồn. Sự khác biệt là ở chỗ: Đích không bị thay đổi sau phép trừ, chỉ có các cờ là thay đổi.

- Ví dụ:

  CMP AL, ‘ ’

  Lệnh trên sẽ lấy nội dung của AL trừ cho 20h (mã ASCII của kí tự khoảng trống). Nếu kết quả mà bằng 0, tức là AL = 20h (AL = ‘ ‘), thì cờ ZF sẽ được thiết lập bằng 1.

- Trạng thái của các cờ sẽ được sử dụng làm điều kiện cho các lệnh nhảy

Lệnh nhảy JZ

- Lệnh JZ là lệnh nhảy khi cờ ZF = 1 (Jump if Zero).
- Cú pháp lệnh:

JZ <Nhãn>

- Trong chương trình trên, lệnh JZ sẽ kiểm tra cờ ZF, nếu ZF = 1 thì sẽ nhảy tới nhãn NHAPLAI, nghĩa là thực hiện lại các lệnh nhập dữ liệu.

Một số lệnh nhảy

- Có nhiều lệnh nhảy khác nhau ứng với trạng thái khác nhau của các cờ:

Lệnh	Chức năng	Điều kiện nhảy
JC	Nhảy nếu có nhớ (Jump if Carry)	CF = 1
JNC	Nhảy nếu không nhớ (Jump if Not Carry)	CF = 0
JO	Nhảy nếu tràn có dấu (Jump if OverFlow)	OF = 1
JNO	Nhảy nếu không tràn (Jump if Not OverFlow)	OF = 0
JS	Nhảy nếu dấu âm (Jump if Sign)	SF = 1
JNS	Nhảy nếu dấu dương (Jump if Not Sign)	SF = 0
...	...	...

Đơn giản hoá cách dùng lệnh nhảy

- Kết hợp lệnh nhảy và lệnh CMP theo quy tắc sau:

CMP <Đích>, <Nguồn>

Điều kiện nhảy	Lệnh nhảy không dấu	Lệnh nhảy có dấu
Đích > Nguồn	JA/ JNBE	JG/ JNLE
Đích < Nguồn	JB/ JNAE	JL/ JNGE
Đích = Nguồn	JE/ JZ	JE/ JZ
Đích ≥ Nguồn	JAE/ JNB	JGE/ JNL
Đích ≤ Nguồn	JBE/ JNA	JLE/ JNG
Đích ≠ Nguồn	JNE/ JNZ	JNE/ JNZ

Một số từ viết tắt:

- A: Above (lớn hơn) = G: Greater than

  B: Below (nhỏ hơn) = L: Less than

  E: Equal (bằng)

  N: Not (không)

- Ví dụ:

  JNA: Jump if Not Above = JBE: Jump if Below - Equal

Giải thích:

- Trước mỗi lệnh nhảy cần dùng một lệnh CMP để tạo điều kiện nhảy. Người lập trình sẽ căn cứ vào quan hệ giữa <Đích> và <Nguồn> để lựa chọn lệnh nhảy thích hợp.
- Ví dụ:

```
CMP AL, 5Ah
JA    KetThuc      ;Nếu AL > 5Ah thì nhảy
                   ; tới nhãn KetThuc
```

- Nếu Đích > Nguồn: Ta có thể sử dụng lệnh nhảy JA hoặc JNBE (trong trường hợp Đích và Nguồn là số không dấu). Hai lệnh này có tác dụng giống hệt nhau.
- Nếu coi đích và nguồn là các số có dấu thì phải sử dụng lệnh JG hoặc JNLE.

Lệnh nhảy không điều kiện JMP

- Các lệnh nhảy có điều kiện mà ta đã nghiên cứu có một nhược điểm là không thể nhảy quá xa. Các lệnh đó chỉ có thể nhảy tới một nhãn đứng trước nó không quá 126 byte hoặc đứng sau không quá 127 byte.
- Để khắc phục điều này có thể sử dụng lệnh nhảy không điều kiện JMP. Cú pháp lệnh như sau:

JMP <Nhãn>

Vị trí của <Nhãn> phải nằm cùng một đoạn nhớ với lệnh nhảy JMP.

Ví dụ:

Xét đoạn lệnh sau:

```
    MOV   AH, 1        ;Nhập một kí tự
    INT    21h
    CMP    AL, 'Z'     ;So sánh kí tự vừa nhập với 'Z'
    JA   KetThuc       ;Nếu AL > 'Z' thì nhảy tới nhãn KetThuc
    ...                ;Các lệnh khác
KetThuc:
    MOV   AH, 4Ch
    INT 21h
```

- Đoạn lệnh trên chỉ thực hiện được khi khoảng cách giữa lệnh JA và vị trí đặt nhãn KetThuc không quá 127 byte.

Sử dụng phương pháp "nhảy hai bước" :

```
MOV   AH, 1                    ;Nhập một kí tự
   INT     21h
   CMP    AL, 'Z'              ;So sánh kí tự vừa nhập với 'Z'
   JA      NhanTrungGian       ;Nếu AL > 'Z' thì nhảy tới NhanTrungGian
   JMP    TiepTuc
NhanTrungGian:
   JMP    KetThuc
TiepTuc:
   ...                         ;Các lệnh khác
KetThuc:
   MOV   AH, 4Ch
   INT 21h
```

Cấu trúc rẽ nhánh IF

- Đối với cấu trúc rẽ nhánh thì vị trí của nhãn sẽ đứng ở sau lệnh nhảy:

<Lệnh nhảy>

....

<Nhãn>

- Ví dụ:

  Nhập một kí tự từ bàn phím, nếu là kí tự in thường thì đổi sang in hoa. Hiện kí tự ra màn hình.

- Thuật toán như sau:

  + Nhập một kí tự KT

  + IF (KT ≤ ‘z’) AND (KT ≥ ‘a’) THEN Đổi KT sang in hoa

  + Hiện KT ra màn hình

```
TITLE  DOI KI TU
.MODEL  SMALL
.STACK  100H
.CODE
MAIN   PROC
    MOV   AH, 1   ;Nhập một kí tự
    INT   21h
    CMP   AL, 'z'
    JA    HienChu ;Nếu AL > 'z' thì hiện kí tự ra màn hình
    CMP   AL, 'a'
    JB    HienChu ;Nếu AL < 'a' thì hiện kí tự ra màn hình
    SUB   AL, 20h ;Đổi kí tự sang in hoa
HienChu:
    MOV   AH, 2   ;Chức năng số 2: Hiện kí tự
    MOV   DL, AL
    INT   21h
    MOV   AH, 4Ch        ;Kết thúc
    INT   21h
MAIN ENDP
END MAIN
```

Cấu trúc lặp

- *Lặp không biết trước số lần lặp (While, Repeat)*
- *Lặp với số lần lặp biết trước (For)*
- *Lệnh lặp LOOP*

- Đối với các cấu trúc lặp nói chung thì vị trí của nhãn sẽ đứng ở trước lệnh nhảy:

  <Nhãn>

  ...

  <Lệnh nhảy>

Lặp không biết trước số lần lặp (While, Repeat)

- Ví dụ:

  Nhập một kí tự số từ bàn phím (‘0’, ‘1’, ... , ‘9’), đổi nó sang số thập phân tương ứng.

  Nhân đôi số đó.

- Thuật toán như sau:

```
REPEAT
    Nhập một kí tự KT
UNTIL   (KT ≥ ‘0’) AND (KT ≤ ‘9’)
Đổi KT sang số thập phân
Nhân đôi số
```

```
TITLE  VI DU LAP
.MODEL  SMALL
.STACK  100H
.CODE
MAIN   PROC
NhapLai:
    MOV   AH, 1       ;Nhập một kí tự
    INT   21h
    CMP   AL, '0'
    JB    NhapLai     ;Nếu AL < '0' thì nhập lại
    CMP   AL, '9'
    JA    NhapLai     ;Nếu AL > '9' thì nhập lại
    SUB   AL, 30h     ;Đổi sang số thập phân tương ứng
    ...                 ;Các lệnh khác
    MOV   AH, 4Ch     ;Kết thúc
    INT   21h
MAIN ENDP
END MAIN
```

Giải thích:

Kí tự ‘0’ có mã ASCII bằng 30h

Kí tự ‘1’ có mã ASCII bằng 31h

...

Kí tự ‘9’ có mã ASCII bằng 39h

Để đổi kí tự số sang số thập phân tương ứng, ta lấy mã ASCII của nó đem trừ cho 30h.

Lặp với số lần lặp biết trước (For)

- Ví dụ:

  Hiện ra màn hình 10 số nguyên theo thứ tự: 0, 1, 2, 3, ..., 9.

- Thuật toán như sau:

FOR I = 0 TO 9 DO <Hiện I ra màn hình>

```
TITLE  VI DU LAP FOR
.MODEL  SMALL
.STACK  100H
.DATA
    I  DB  0          ;Khởi tạo giá trị biến I bằng 0
.CODE
MAIN   PROC
    MOV   AX, @DATA
    MOV   DS, AX
Lap:
    MOV   DL, I
    ADD   DL,30h     ;Đổi số nguyên sang kí tự số tương ứng
    MOV   AH, 2      ;Chức năng số 2: Hiện kí tự
    INT   21h
    INC   I          ;Tăng biến I lên 1
    CMP   I, 10
    JNZ   Lap        ;Nếu I ≠ 10 thì lặp lại
    MOV   AH, 4Ch          ;Kết thúc
    INT   21h
MAIN ENDP
END MAIN
```

Lệnh lặp LOOP

- Đây là cấu trúc lệnh có sẵn trong hợp ngữ để thực hiện các vòng lặp biết trước số lần lặp. Cách viết lệnh như sau:

```
MOV   CX,  <Số lần lặp>
NHANLAP:
...                  ;Các lệnh cần lặp
LOOP   NHANLAP
```

- Số lần lặp được đặt vào thanh ghi CX, sau mỗi lần lặp thì CX được tự động giảm đi 1.

```
TITLE  VI DU LAP LOOP
.MODEL  SMALL
.STACK  100H
.DATA
    I  DB  0          ;Khởi tạo giá trị biến I bằng 0
.CODE
MAIN   PROC
    MOV   AX, @DATA
    MOV   DS, AX
    MOV   CX, 10     ;Số lần lặp
Lap:
    MOV   DL, I
    ADD   DL,30h     ;Đổi số nguyên sang kí tự số tương ứng
    MOV   AH, 2      ;Chức năng số 2: Hiện kí tự
    INT   21h
    INC   I          ;Tăng biến I lên 1
    LOOP  Lap
    MOV   AH, 4Ch    ;Kết thúc
    INT   21h
MAIN ENDP
END MAIN
```

Hết Phần 2.2

KIẾN TRÚC MÁY TÍNH

Giảng viên: Ths Phạm Thanh Bình
Bộ môn Kỹ thuật máy tính & mạng
http://vn.myblog.yahoo.com/CNTT-wru
http://ktmt.wru.googlepages.com

2.6.6 Các lệnh logic, dịch, và quay

- Các phép logic
- Các phép dịch
- Các phép quay

Các phép logic

- *Phép toán AND (và)*
- *Phép toán OR (hoặc)*
- *Phép toán NOT (phủ định)*
- *Phép toán XOR (hoặc - phủ định)*
- *Các lệnh logic trong Assembly*

Phép toán AND (và)

- Quy tắc thực hiện phép toán AND giữa hai số nhị phân A và B được trình bày trong bảng sau:

A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

- Ví dụ:

  Cho M = 16h, N = 0Dh, hãy tính M AND N = ?

- Giải:

$$\begin{aligned} M &= 0001\ 0110b \quad (16h) \\ N &= 0000\ 1101b \quad (0Dh) \\ \hline M \text{ AND } N &= 0000\ 0100b = 04h \end{aligned}$$

Phép toán OR (hoặc)

- Quy tắc thực hiện phép toán OR giữa hai số nhị phân A và B được trình bày trong bảng sau:

A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

- Ví dụ:

  Cho M = 16h, N = 0Dh, hãy tính M OR N = ?

- Giải:

$$M = \ 0001\ 0110b \quad (16h)$$

$$N = \ 0000\ 1101b \quad (0Dh)$$

$$M \text{ OR } N = \ 0001\ 1111b \quad = 1Fh$$

Phép toán NOT (phủ định)

- Quy tắc thực hiện phép toán NOT giữa hai số nhị phân A và B được trình bày trong bảng sau:

A	NOT A
0	1
1	0

- Ví dụ:

  Cho M = 16h, hãy tính NOT M = ?

- Giải:

$$M = 0001\ 0110b \quad (16h)$$

$$\text{NOT } M = 1110\ 1001b = E9h$$

Phép toán XOR (hoặc - phủ định)

- Quy tắc thực hiện phép toán XOR giữa hai số nhị phân A và B được trình bày trong bảng sau:

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

- Ví dụ:

  Cho M = 16h, N = 0Dh, hãy tính M XOR N = ?

- Giải:

$$\begin{aligned} M &= 0001\ 0110b \quad (16h) \\ N &= 0000\ 1101b \quad (0Dh) \\ \hline M \text{ XOR } N &= 0001\ 1011b = 1Bh \end{aligned}$$

Các lệnh logic trong Assembly

- Tương ứng với các phép toán logic trên, hợp ngữ có các lệnh sau đây:

```
AND   <Đích>, <Nguồn>
OR    <Đích>, <Nguồn>
XOR   <Đích>, <Nguồn>
NOT   <Đích>
```

<Đích>: là một thanh ghi hay một ô nhớ

<Nguồn>: là một thanh ghi, một ô nhớ, hoặc một hằng số

<Đích>,<Nguồn> không đồng thời là hai ô nhớ.

- Ví dụ:

```
AND   AX, 002Ah
OR    AL, 3Dh
NOT   BX
```

Ứng dụng các lệnh logic:

- Ví dụ 1:

  Hãy thay đổi bit dấu trong thanh ghi AX.

  ```
  XOR   AX, 8000h
  ```

- Ví dụ 2:

  Hãy xoá bit LSB trong thanh ghi BH.

  ```
  AND   BH, 0FEh
  ```

- Ví dụ 3:

  Nhập một kí tự số từ bàn phím (‘0’, ‘1’, ... , ‘9’), đổi nó sang số thập phân tương ứng.

- Giải:

  Ta sẽ sử dụng các lệnh logic để chuyển đổi kí tự sang số.

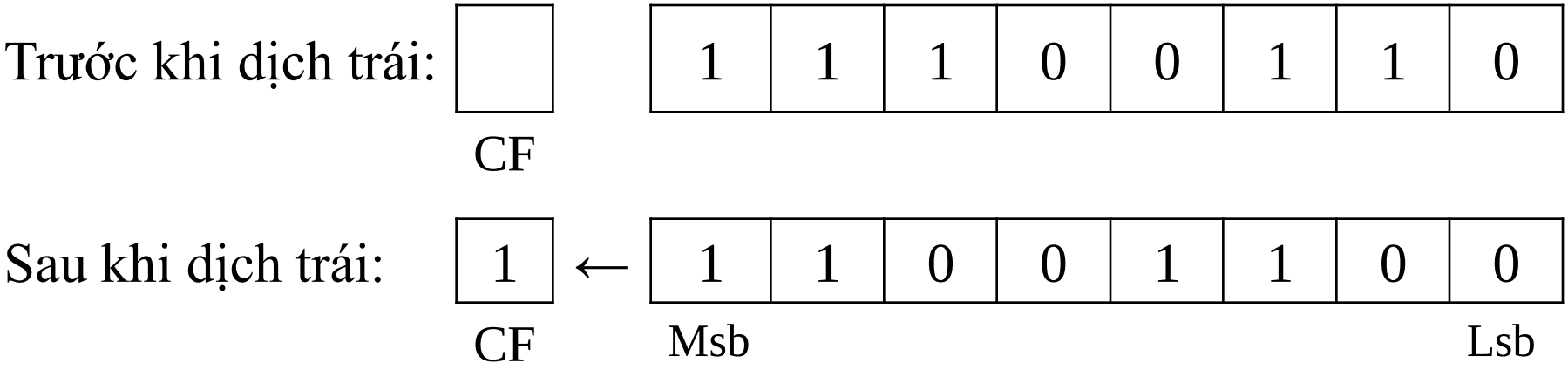
```
TITLE  VI DU 3
.MODEL  SMALL
.STACK  100H
.CODE
MAIN   PROC
NhapLai:
    MOV   AH, 1       ;Nhập một kí tự
    INT   21h
    CMP   AL, '0'
    JB    NhapLai     ;Nếu AL < '0' thì nhập lại
    CMP   AL, '9'
    JA    NhapLai     ;Nếu AL > '9' thì nhập lại
    AND   AL, 0Fh     ;Đổi sang số thập phân tương ứng
                      ;          (xoá 4 bit cao của AL)
    ...               ;Các lệnh khác
    MOV   AH, 4Ch     ;Kết thúc
    INT   21h
MAIN ENDP
END MAIN
```

Các phép dịch

- *Dịch trái*
- *Dịch phải*

Dịch trái

- Xét một dãy bit trong một thanh ghi hoặc một ô nhớ: phép dịch trái sẽ dịch chuyển toàn bộ các bít trong dãy về bên trái, giá trị của Msb được đưa vào cờ CF, thêm bit 0 vào vị trí Lsb



Lệnh dịch trái

- Có thể sử dụng lệnh SHL (Shift Left) hoặc lệnh SAL (Shift Arithmetic Left), hai lệnh này tạo ra cùng một mã máy.
- Cú pháp lệnh:

  - Dạng 1: SHL <Đích>, 1
  - Dạng 2: SHL <Đích>, CL

  <Đích>: là một thanh ghi hay một ô nhớ
- Dạng 1 sẽ dịch các bít của toán hạng đích sang trái 1 lần, dạng 2 sẽ dịch các bít của toán hạng đích sang trái nhiều lần, số lần dịch chứa trong thanh ghi CL

- Ví dụ 1:

```
SHL   AX, 1     ;Dịch các bít của thanh
                ; ghi AX sang trái 1 lần
```

- Ví dụ 2:

```
MOV   CL, 3
SHL   AX, CL    ;Dịch các bít của thanh
                ;ghi AX sang trái 3 lần
```

Ứng dụng của lệnh dịch trái

- Một trong số các ứng dụng của lệnh dịch trái là thực hiện phép nhân với 2.
- Giả sử AH = 0001 1010b = $1x2^4 + 1x2^3 + 1x2^1 = 26$
- Sau khi dịch trái 1 lần thì AH = 0011 0100b = $1x2^5 + 1x2^4 + 1x2^2 = 2x(1x2^4 + 1x2^3 + 1x2^1) = 52$
- Như vậy, phép dịch trái 1 lần tương đương phép nhân toán hạng đích với 2.
- Tổng quát: Phép dịch trái N lần tương đương phép nhân toán hạng đích với 2^N.

Chú ý:

- Kết luận trên chỉ đúng khi không có hiện tượng tràn xảy ra.
- Ví dụ:

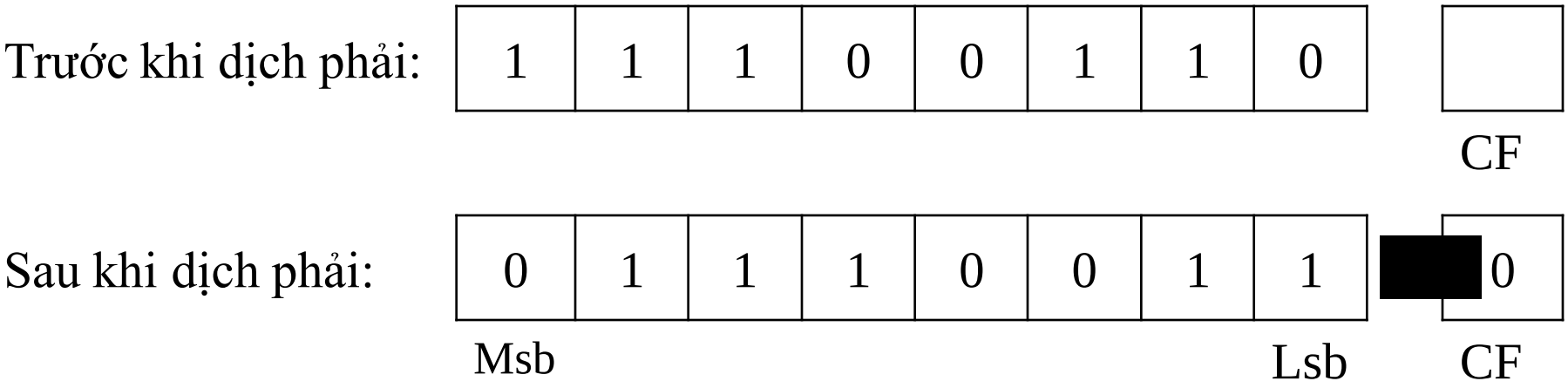
  Giả sử AH = 1000 0001b = 129

  Sau khi dịch trái 1 lần thì AH = 0000 0010b = 2 *Không phải là phép nhân với 2*.

  Bít Msb của AH được chuyển vào cờ CF: cờ CF = 1 báo hiệu hiện tượng tràn xảy ra, kết quả không còn đúng nữa

Dịch phải

- Xét một dãy bit trong một thanh ghi hoặc một ô nhớ: phép dịch phải sẽ dịch chuyển toàn bộ các bít trong dãy về bên phải, giá trị của Lsb được đưa vào cờ CF, thêm bit 0 vào vị trí Msb



Lệnh dịch phải

- Có hai loại lệnh dịch phải: SHR (Shift Right) và SAR (Shift Arithmetic Right), tác dụng của chúng không hoàn toàn giống nhau.
- Cú pháp lệnh SHR:

  - Dạng 1: SHR <Đích>, 1
  - Dạng 2: SHR <Đích>, CL

  <Đích>: là một thanh ghi hay một ô nhớ
- Dạng 1 sẽ dịch các bít của toán hạng đích sang phải 1 lần, dạng 2 sẽ dịch các bít của toán hạng đích sang phải nhiều lần, số lần dịch chứa trong thanh ghi CL

- Ví dụ 1:

```
SHR   BX, 1   ;Dịch các bít của thanh
              ;ghi BX sang phải 1 lần
```

- Ví dụ 2:

```
MOV   CL, 3
SHR   BX, CL  ;Dịch các bít của thanh
              ;ghi BX sang phải 3 lần
```

Ứng dụng của lệnh dịch phải

- Một trong số các ứng dụng của lệnh dịch phải là thực hiện phép chia cho 2.
- Giả sử AL = 0001 0000b = $1 \times 2^4 = 16$
- Sau khi dịch phải 1 lần thì AL = 0000 1000b = $1 \times 2^3 = 8$
- Như vậy, phép dịch phải 1 lần tương đương phép chia toán hạng đích cho 2.
- Tổng quát: Phép dịch phải N lần tương đương phép chia toán hạng đích cho 2^N

Chú ý:

- Kết luận trên chỉ đúng nếu không làm thay đổi bít dấu khi dịch phải.
- Ví dụ:

Giả sử AL = 1000 0001b = -127

Sau khi dịch phải 1 lần thì AL = 0100 0000b = 64: Không phải là phép chia cho 2

Chú ý:

- Do đó, đối với các số có dấu ta không được sử dụng lệnh SHR mà phải sử dụng lệnh SAR. Lệnh SAR sẽ giữ nguyên bít dấu khi dịch phải.
- Ví dụ:

  Ban đầu AL = 1000 0001b = -127

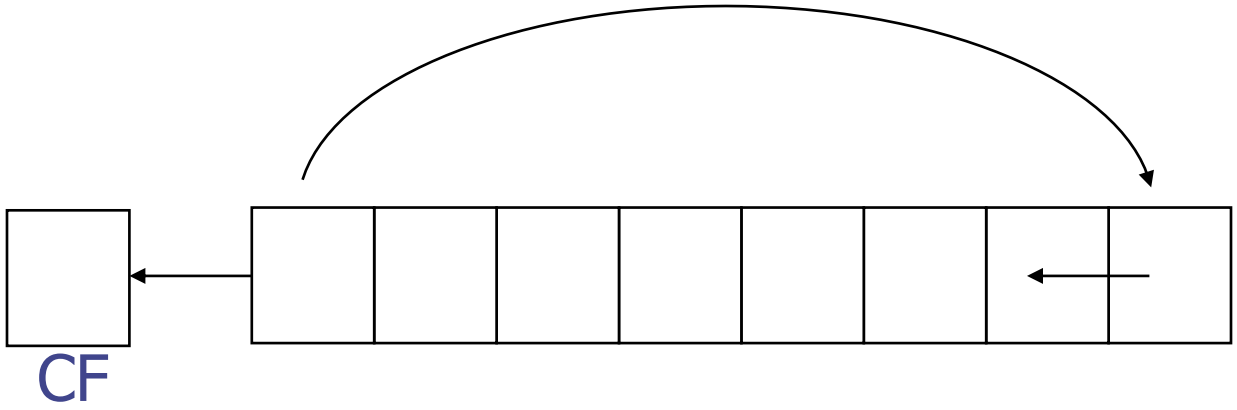
  Sau lệnh SAR AL, 1 thì AL = 1100 0000b = -64 (bít dấu vẫn được giữ nguyên

Các phép quay

- *Lệnh quay trái (ROL - Rotate Left)*
- *Lệnh quay phải (ROR - Rotate Right)*
- *Lệnh quay trái qua cờ CF (RCL – Rotate Carry Left)*
- *Lệnh quay phải qua cờ CF (RCR – Rotate Carry Right)*

Lệnh quay trái (ROL - Rotate Left)

- Lệnh quay trái cũng gần giống với lệnh dịch trái, chỉ khác ở chỗ bít Msb vừa được đưa vào cờ CF, vừa được đưa trở lại vị trí Lsb



- Cú pháp lệnh:
  - Dạng 1: ROL <Đích>, 1

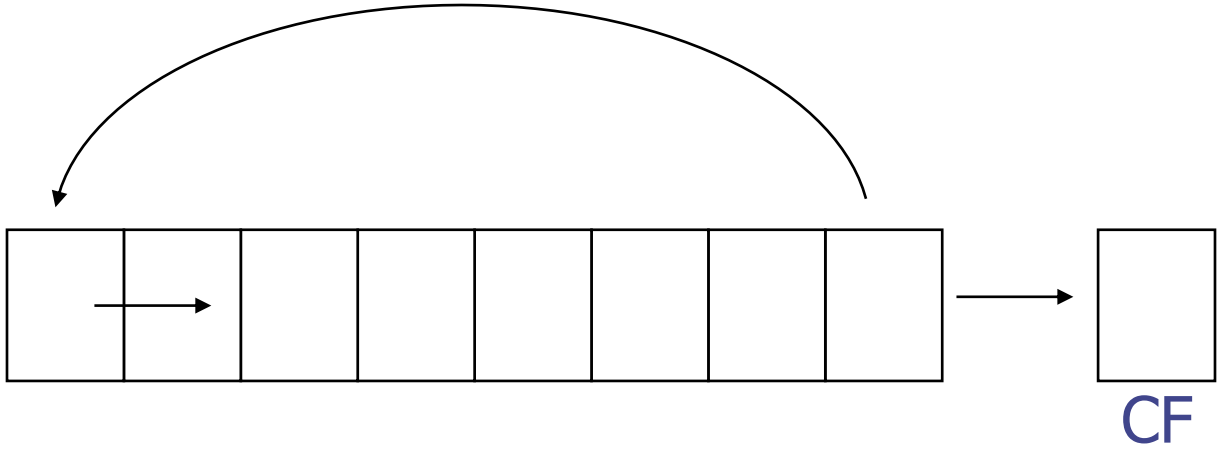
    (Quay trái một lần)
  - Dạng 2: ROL <Đích>, CL

    (Quay trái nhiều lần, CL chứa số lần quay)

    <Đích>: là một thanh ghi hay một ô nhớ

Lệnh quay phải (ROR - Rotate Right)

- Lệnh quay phải cũng gần giống với lệnh dịch phải, chỉ khác ở chỗ bít Lsb vừa được đưa vào cờ CF, vừa được đưa trở lại vị trí Msb



- Cú pháp lệnh:
  - Dạng 1: ROR <Đích>, 1

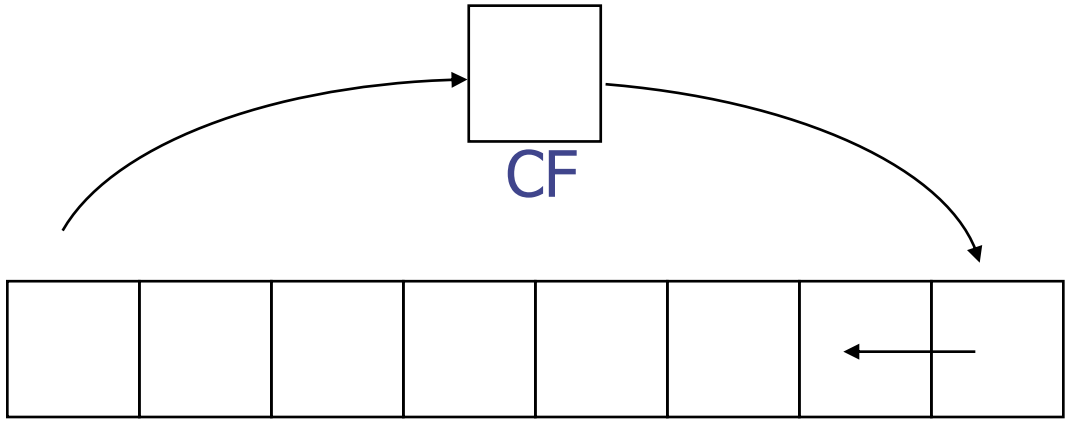
    (Quay phải một lần)
  - Dạng 2: ROR <Đích>, CL

    (Quay phải nhiều lần, CL chứa số lần quay)

    <Đích>: là một thanh ghi hay một ô nhớ

Lệnh quay trái qua cờ CF (RCL – Rotate Carry Left)

- Lệnh này cũng gần giống với lệnh dịch trái, chỉ khác ở chỗ bít Msb được đưa vào cờ CF, còn nội dung cờ CF lại được đưa vào vị trí Lsb



- Cú pháp lệnh:
  - Dạng 1: RCL <Đích>, 1

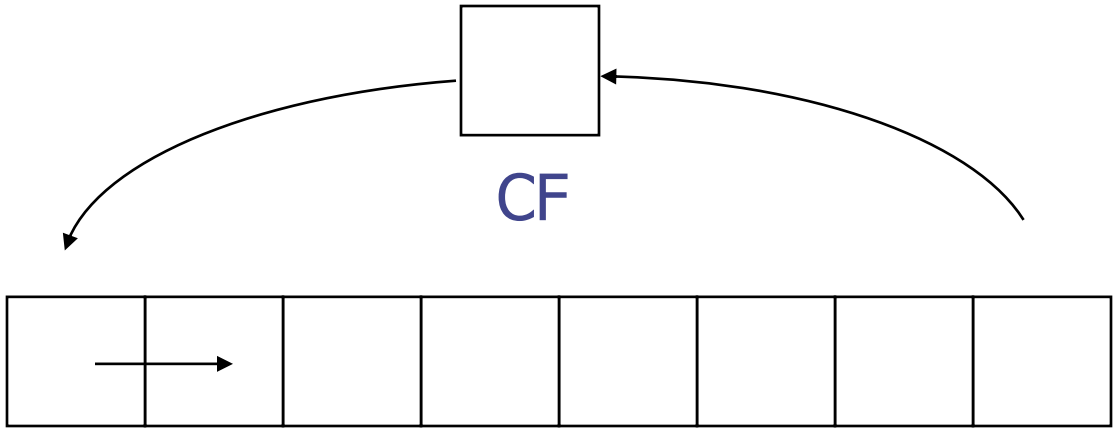
    (Quay một lần)
  - Dạng 2: RCL <Đích>, CL

    (Quay nhiều lần, CL chứa số lần quay)

<Đích>: là một thanh ghi hay một ô nhớ

Lệnh quay phải qua cờ CF (RCR – Rotate Carry Right)

- Lệnh này cũng gần giống với lệnh dịch phải, chỉ khác ở chỗ bít Lsb được đưa vào cờ CF, còn nội dung cờ CF lại được đưa vào vị trí Msb



- Cú pháp lệnh:
  - Dạng 1: RCR <Đích>, 1

    (Quay một lần)
  - Dạng 2: RCR <Đích>, CL

    (Quay nhiều lần, CL chứa số lần quay)

<Đích>: là một thanh ghi hay một ô nhớ

Ứng dụng của lệnh quay

- Ví dụ: Đếm số bít 1 trong thanh ghi AX
- Giải:

  Ta sẽ thực hiện lệnh quay AX 16 lần (quay trái hay quay phải đều được). Mỗi lần quay thì một bít sẽ được đưa vào cờ CF, nếu bít đó bằng 1 thì tăng BX lên 1 (BX chứa giá trị đếm được). Sau 16 lần quay thì thanh ghi AX sẽ trở lại giá trị ban đầu.

```
MOV   CX, 16      ;CX chứa số lần lặp
XOR   BX          ;Xoá BX để chuẩn bị chứa số lượng bít 1
Lap:
    ROL   AX, 1         ;Quay trái AX 1 lần
    JNC   TiepTuc       ;Nếu CF = 0 (gặp bít 0) thì nhảy
    INC    BX           ;Nếu gặp bít 1 thì tăng BX
    TiepTuc:
LOOP Lap
```

2.6.7 Ngăn xếp và thủ tục

- Các thành phần của chương trình
- Cách sử dụng ngăn xếp
- Thủ tục

Các thành phần của chương trình

```
TITLE   <Tên chương trình>
.MODEL  <Kiểu bộ nhớ>
.STACK  <Kích thước ngăn xếp>
.DATA
        <Khai báo dữ liệu>
.CODE
        <Phần mã lệnh>
```

- Ngăn xếp là vùng nhớ đặc biệt của chương trình. Có thể sử dụng vùng nhớ này để lưu trữ dữ liệu và giải phóng nó khi không dùng đến. Như vậy, việc sử dụng ngăn xếp một cách hợp lý sẽ giúp tiết kiệm bộ nhớ. Trong hợp ngữ, kích thước ngăn xếp của chương trình được khai báo sau từ khoá .Stack

Một chương trình bao gồm ba phần cơ bản: Mã lệnh, Dữ liệu, Ngăn xếp. Khi chương trình được nạp vào bộ nhớ thì ba phần trên được nạp vào các đoạn nhớ khác nhau:

- Đoạn nhớ chứa phần mã lệnh được gọi là Đoạn mã (Code Segment), địa chỉ của nó được lưu giữ trong thanh ghi đoạn CS.
- Đoạn nhớ chứa phần dữ liệu được gọi là Đoạn dữ liệu (Data Segment), địa chỉ của nó được lưu giữ trong thanh ghi đoạn DS.
- Đoạn nhớ chứa phần ngăn xếp được gọi là Đoạn ngăn xếp (Stack Segment), địa chỉ của nó được lưu giữ trong thanh ghi đoạn SS.

Cách sử dụng ngăn xếp

- *Cất dữ liệu vào ngăn xếp*
- *Lấy dữ liệu khỏi ngăn xếp*
- *Ứng dụng của ngăn xếp*
- *Cách thức làm việc của ngăn xếp*

Cất dữ liệu vào ngăn xếp

- Để cất dữ liệu vào ngăn xếp ta sử dụng lệnh Push, cách viết lệnh như sau:

  PUSH <Nguồn>

  <Nguồn>: là một thanh ghi hay một biến có kích thước 16 bít (1 word).

- Sau lệnh Push thì giá trị của toán hạng Nguồn vẫn giữ nguyên

- Ví dụ 1:

```
PUSH   AX
```

Lệnh trên cất nội dung thanh ghi AX vào ngăn xếp.

- Ví dụ 2:

```
PUSH   A
```

Lệnh trên cất nội dung biến A vào ngăn xếp *(A phải là biến kiểu Word)*

Lấy dữ liệu khỏi ngăn xếp

- Để lấy dữ liệu khỏi ngăn xếp ta sử dụng lệnh Pop, cách viết lệnh như sau:

POP <Đích>

<Đích>: là một thanh ghi hay một biến có kích thước 16 bít (1 word).

- Việc lấy dữ liệu khỏi ngăn xếp sẽ đồng thời giải phóng ô nhớ đang chứa dữ liệu (tức là có thể dùng nó để chứa dữ liệu khác).

- Ví dụ 1:

```
POP   AX
```

Lệnh trên lấy dữ liệu từ ngăn xếp đặt vào thanh ghi AX.

- Ví dụ 2:

```
POP   A
```

Lệnh trên lấy dữ liệu từ ngăn xếp đặt vào biến A *(A phải là biến kiểu Word)*.

Ứng dụng của ngăn xếp

- *Ví dụ 1:*

  Hãy chuyển nội dung của thanh ghi đoạn DS vào thanh ghi đoạn ES

Giải:

Do không thể chuyển trực tiếp nội dung của hai thanh ghi đoạn cho nhau (xem lại phần lệnh MOV) nên ta sẽ sử dụng ngăn xếp làm trung gian: dữ liệu được chuyển từ DS vào ngăn xếp, sau đó lấy từ ngăn xếp chuyển vào ES:

```
PUSH   DS      ;Cất DS vào ngăn xếp
POP    ES      ;Lấy dữ liệu từ ngăn xếp đặt
               ;  vào ES
```

Ví dụ 2:

Viết chương trình nhập một kí tự từ bàn phím rồi hiện nó ở đầu dòng tiếp theo

Giải:

```
TITLE  Vi du 2
.MODEL  SMALL
.STACK  100H
.CODE
MAIN   PROC
    MOV   AH, 1       ;Chức năng số 1: Nhập một kí tự
    INT   21h
    PUSH  AX          ;Cất kí tự vào ngăn xếp
    MOV   AH, 2       ;Đưa con trỏ về đầu dòng tiếp theo
    MOV   DL, 0Dh
    INT   21h
    MOV   DL, 0Ah
    INT   21h
    POP   DX          ;Lấy kí tự từ ngăn xếp đặt vào DL
    INT   21h         ;Hiển thị kí tự
    MOV   AH, 4Ch             ;Kết thúc
    INT   21h
MAIN ENDP
END MAIN
```

- *Giải thích*:

  Kí tự nhập vào được cất ở thanh ghi AL. Để đưa con trỏ xuống đầu dòng tiếp theo thì phải hiển thị hai kí tự có mã ASCII là 0Dh (CR: về đầu dòng) và 0Ah (LF: xuống dòng). Quá trình hiển thị hai kí tự này sẽ làm thanh ghi AL bị thay đổi (xem lại chức năng số 2 của ngắt 21h). Do đó cần phải lưu kí tự ban đầu vào ngăn xếp trước khi xuống dòng, khi nào muốn hiển thị kí tự này thì lại lấy nó ra từ ngăn xếp

Ví dụ 3:

Viết lệnh thực hiện các công việc sau:

+ Lưu nội dung thanh ghi cờ vào AX.

+ Xoá thanh ghi cờ.

- *Giải*:

  Ta không thể tác động tới thanh ghi cờ bằng các lệnh thông thường đã học như MOV, ADD, SUB, AND, OR…Bộ vi xử lý 8086 cung cấp hai lệnh sau để thao tác với thanh ghi cờ (cả hai lệnh đều liên quan tới ngăn xếp):

```
PUSHF    ;Cất nội dung thanh ghi cờ vào ngăn xếp
POPF     ;Lấy dữ liệu từ ngăn xếp đặt vào thanh ghi cờ
```

- Sử dụng hai lệnh này ta có thể giải quyết yêu cầu đặt ra ở trên

```
PUSHF
POP    AX        ;chuyển nội dung thanh ghi cờ từ ngăn xếp vào AX
XOR    BX, BX    ;xóa BX (BX = 0)
PUSH   BX        ;Đặt giá trị 0 vào ngăn xếp
POPF             ;Chuyển giá trị 0 từ ngăn xếp vào thanh ghi cờ
                                                    (xoá các cờ)
```

Cách thức làm việc của ngăn xếp

- Kích thước của ngăn xếp được khai báo ở đầu chương trình hợp ngữ sau từ khoá .Stack
- Ví dụ:

  .Stack 100h

  Khi đó ngăn xếp có kích thước bằng 100h byte (256 byte hay 128 word)

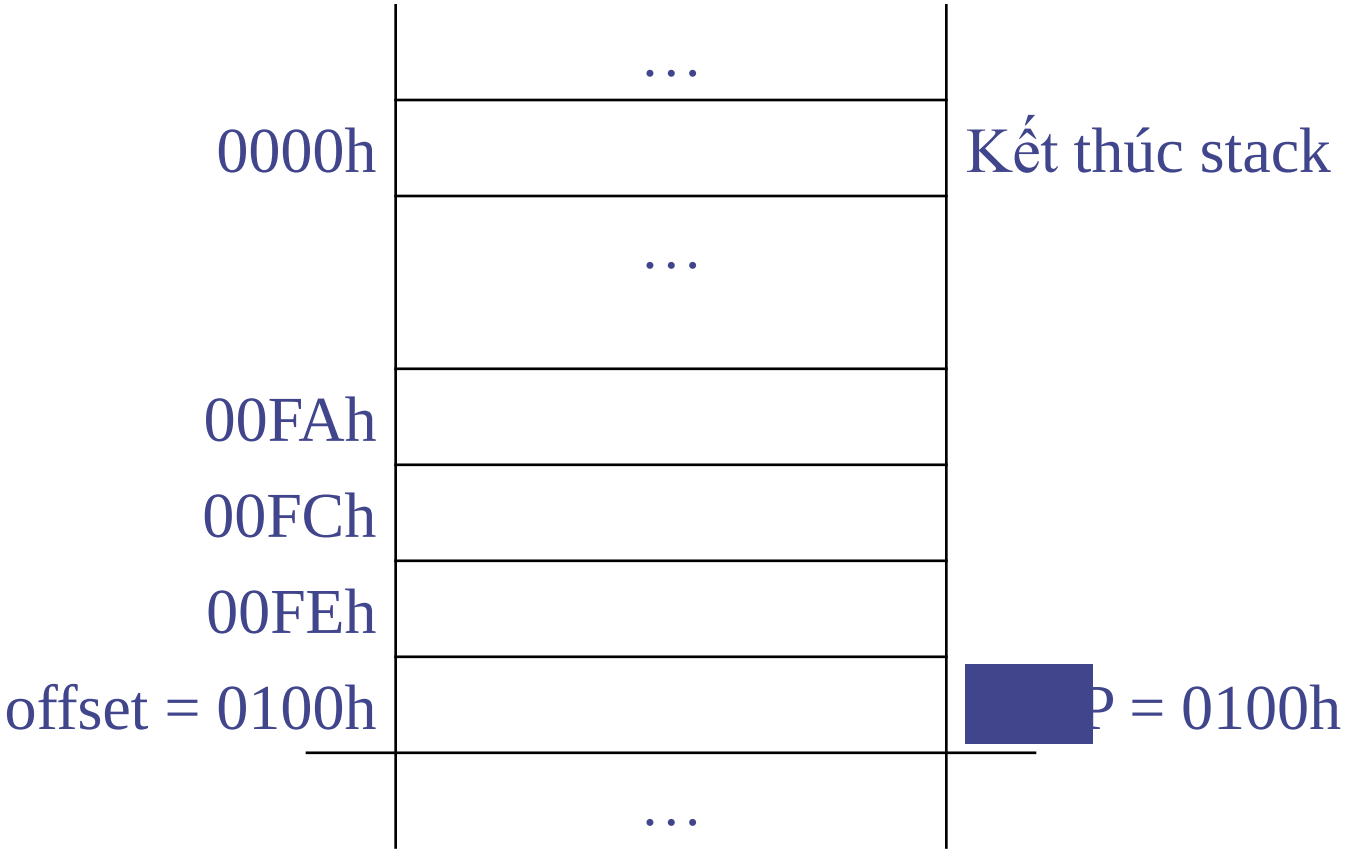
- Mỗi lệnh Push sẽ chiếm dụng 1 word của ngăn xếp, như vậy ngăn xếp khai báo như trên sẽ cho phép cất tối đa 128 lần.
- Người lập trình sẽ phải tính toán để khai báo ngăn xếp có kích thước hợp lý nhất (không quá thừa hay thiếu)

Cấu trúc của ngăn xếp:

- Dữ liệu được lấy ra khỏi ngăn xếp theo trình tự ngược lại so với khi cất vào, nghĩa là cất vào sau thì sẽ được lấy ra trước (LIFO - Last In First Out).
- Bộ vi xử lý 8086 sử dụng hai thanh ghi chuyên dụng cho các thao tác với ngăn xếp là SS (Stack Segment) và SP (Stack Pointer).
- SS chứa địa chỉ segment còn SP chứa địa chỉ offset của ô nhớ trong ngăn xếp

- Dữ liệu được cất vào ngăn xếp theo trật tự ngược lại so với các đoạn nhớ khác (từ địa chỉ cao xuống địa chỉ thấp).
- Giả sử khai báo ngăn xếp là .Stack 100h thì ngăn xếp sẽ bắt đầu tại địa chỉ offset = 0100h và kết thúc tại offset = 0000h

(xem hình trang sau)


...
0000h
Kết thúc stack
...
00FAh
00FCh
00FEh
offset = 0100h
P = 0100h
...

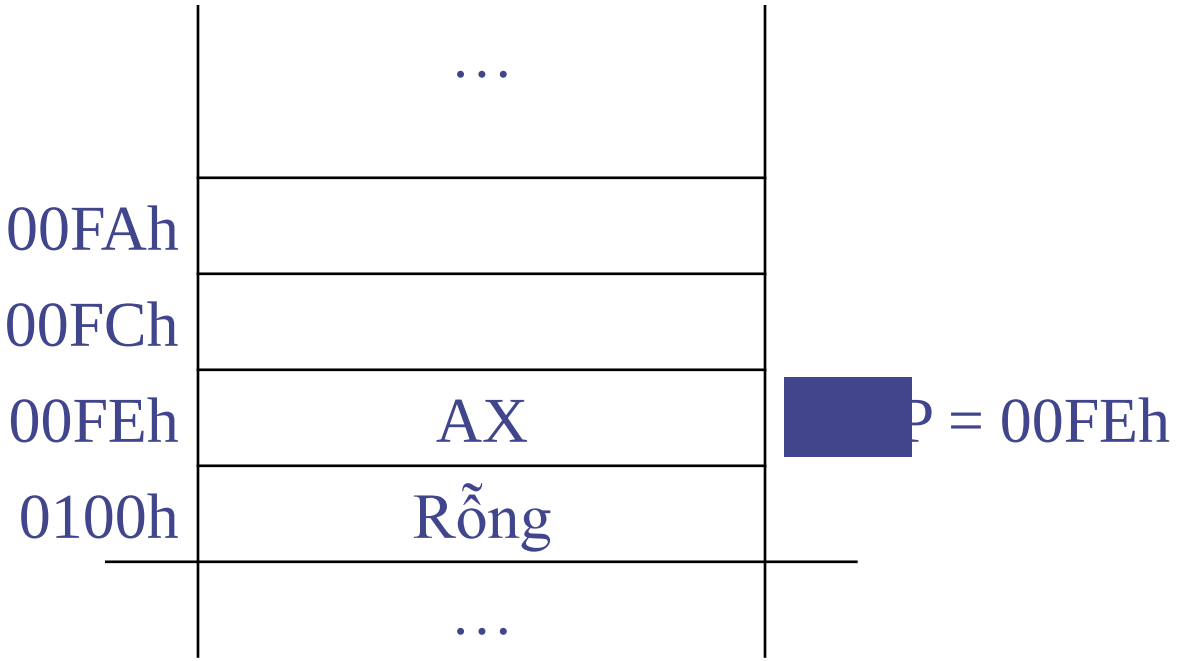
- Khi ngăn xếp chưa có dữ liệu thì SP trỏ tới ô nhớ có địa chỉ cao nhất trong ngăn xếp.
- Sau mỗi lệnh Push thì SP sẽ giảm đi 2 để trỏ tới ô tiếp theo của ngăn xếp, dữ liệu sẽ được cất vào ô nhớ do SP trỏ tới

Ví dụ:

- Cất ba thanh ghi AX, BX, CX vào ngăn xếp:

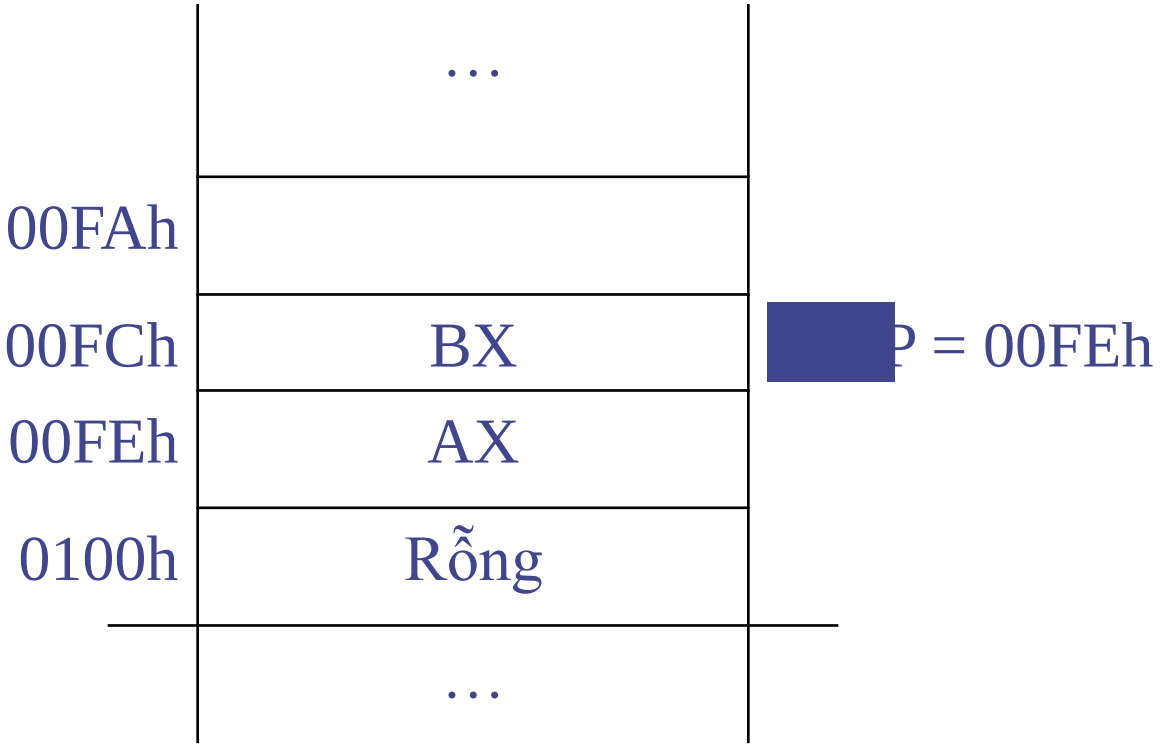
```
PUSH   AX
PUSH   BX
PUSH   CX
```

Sau lệnh PUSH AX:

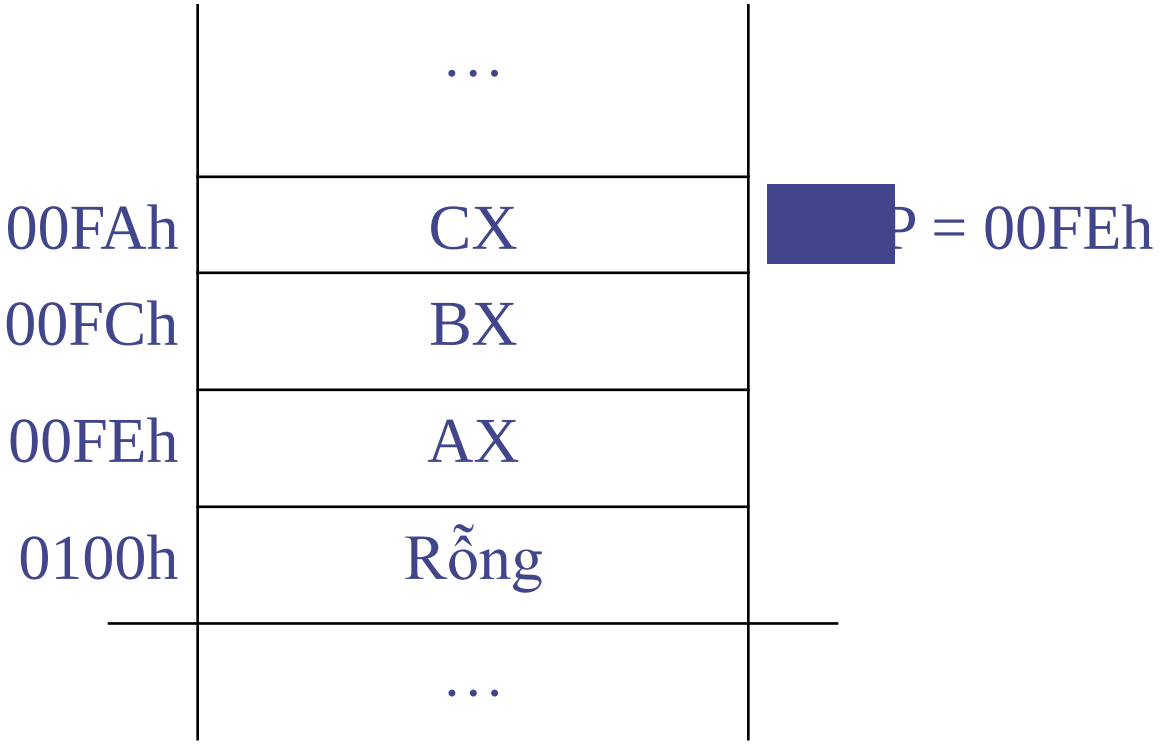


- Do giảm SP đi 2 rồi mới cất thanh ghi AX vào ngăn xếp nên sẽ tạo ra một ô rỗng ở địa chỉ cao nhất

Sau lệnh PUSH BX:



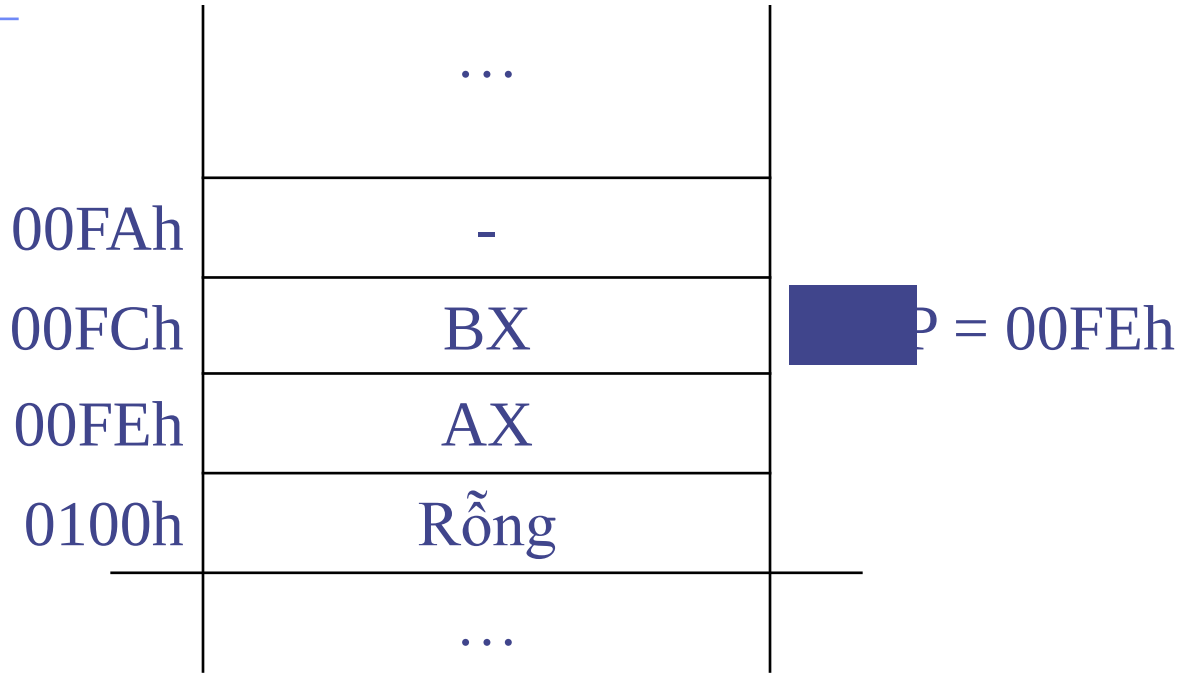
Sau lệnh PUSH CX:



- Muốn lấy nội dung của ba thanh ghi ra khỏi ngăn xếp thì phải tiến hành theo trình tự ngược lại:

```
POP   CX
POP   BX
POP   AX
```

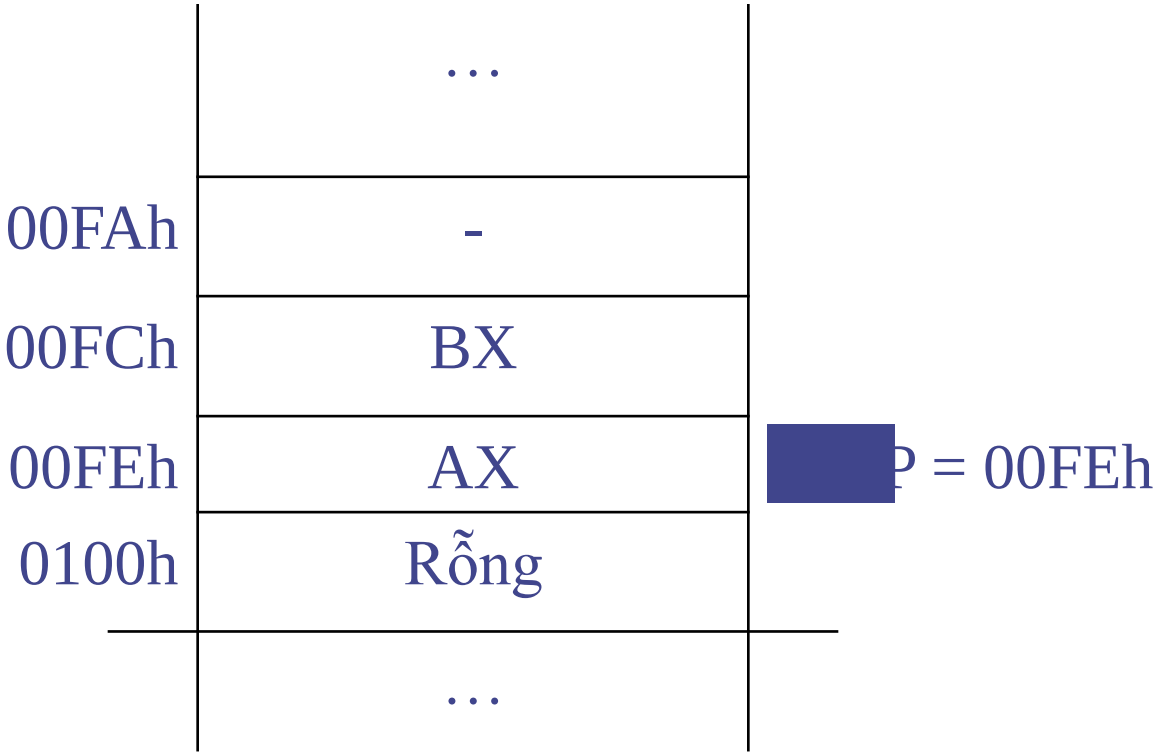
Sau lệnh POP CX:



- Dữ liệu tại ô nhớ do SP trỏ tới (offset = 00FAh) sẽ được nạp vào thanh ghi CX, sau đó SP tăng lên 2 để trỏ tới ô cao hơn.
- Ô nhớ chứa CX đã được giải phóng nên ta không cần quan tâm tới nội dung bên trong nó nữa

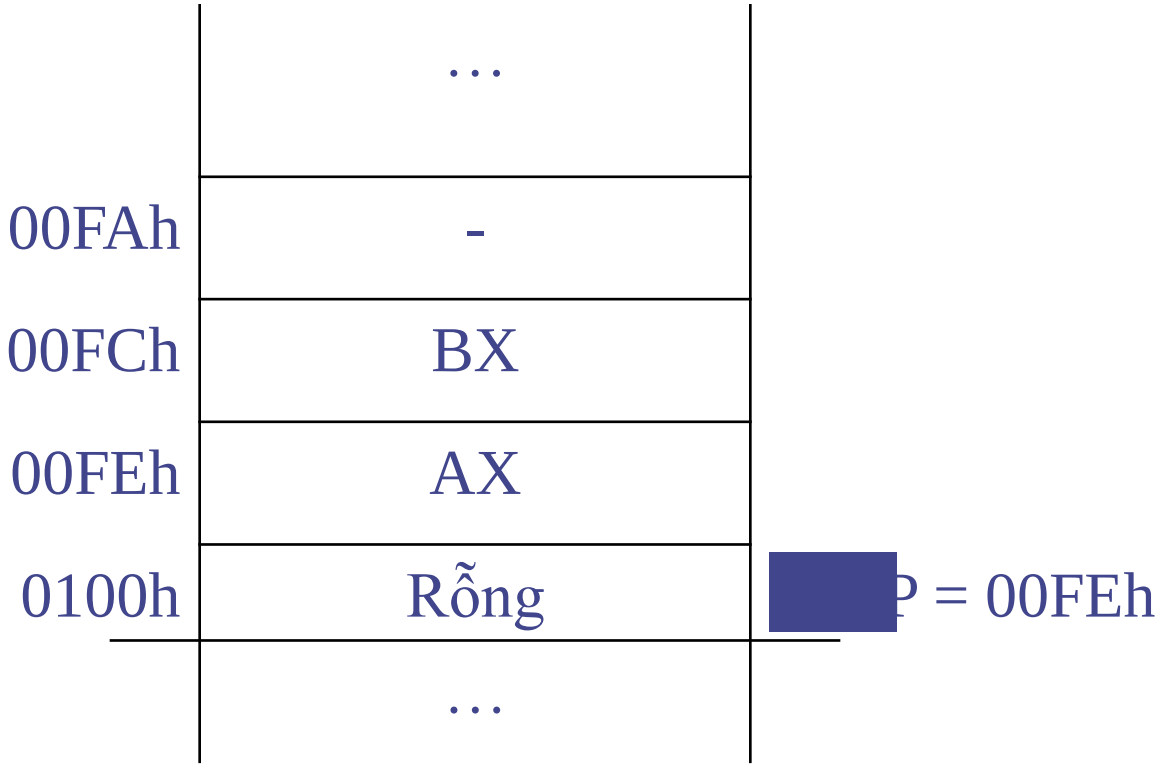
Sau lệnh POP BX:

Dữ liệu tại ô nhớ có offset = 00FCh được nạp vào thanh ghi BX



Sau lệnh POP AX:

Dữ liệu tại ô nhớ có offset = 00FEh được nạp vào thanh ghi AX.



Thủ tục

- *Cấu trúc thủ tục*
- *Sử dụng thủ tục trong chương trình*
- *Quan hệ giữa thủ tục và ngăn xếp*

Cấu trúc thủ tục

```
<Tên thủ tục>    PROC     ;Bắt đầu thủ tục
        Lệnh 1
        Lệnh 2
        Lệnh 3
        ...
        RET                ;Trở về chương trình chính
<Tên thủ tục>    ENDP     ;Kết thúc thủ tục
```

- Đây là cấu trúc của một thủ tục thông thường (không là chương trình chính), nó phải được kết thúc bởi lệnh RET để trở về chương trình chính.
- Lệnh RET thường nằm ở cuối thủ tục, nhưng nó cũng có thể nằm ở một vị trí khác

- Ví dụ:

  Viết một thủ tục đưa con trỏ màn hình xuống đầu dòng tiếp theo.

- Giải:

  Để đưa con trỏ xuống đầu dòng tiếp theo cần hiển thị các kí tự CR (0Dh) và LF (0Ah). Ta đặt tên thủ tục này là Writeln

```
Writeln     PROC
    MOV    AH, 2        ;Chức năng số 2 của
                        ;   ngắt 21h để hiện kí tự
    MOV    DL, 0Dh
    INT     21h
    MOV    DL, 0Ah
    INT     21h
    RET
Writeln     ENDP
```

Sử dụng thủ tục trong chương trình

- Để gọi một thủ tục từ chương trình chính ta sử dụng lệnh Call, cú pháp lệnh như sau:

CALL <Tên thủ tục>

- Cấu trúc của một chương trình assembly có sử dụng thủ tục như sau:

```
TITLE  <Tên chương trình>
.MODEL  <Kiểu bộ nhớ>
.STACK  <Kích thước ngăn xếp>
.DATA
    <Khai báo dữ liệu>
.CODE
<Chương trình chính>   PROC
        Lệnh 1
        Lệnh 2
        Lệnh 3
        ...
    CALL   <Tên thủ tục>            ;Gọi  thủ tục
    ...
<Chương trình chính>   ENDP
    <Tên thủ tục>   PROC
        Lệnh 1
        Lệnh 2
        Lệnh 3
        ...
        RET                         ;Trở về chương trình chính
    <Tên thủ tục>   ENDP
    ...Các thủ tục khác
END   <Chương trình chính>
```

- Ví dụ:

  Viết chương trình nhập một kí tự từ bàn phím rồi hiện nó ở đầu dòng tiếp theo (có sử dụng thủ tục).

```
TITLE  Vi du
.MODEL  SMALL
.STACK  100H
.CODE
MAIN   PROC
   MOV   AH, 1          ;Nhập một kí tự
   INT   21h
   PUSH  AX             ;Cất kí tự vào ngăn xếp
   CALL  Writeln        ;Đưa con trỏ về đầu dòng tiếp theo
   POP   DX             ;Lấy kí tự từ ngăn xếp đặt vào DL
   MOV   AH, 2          ;Hiển thị kí tự
   INT   21h
   MOV   AH, 4Ch        ;Kết thúc
   INT   21h
MAIN ENDP
 Writeln  PROC          ;Thủ tục đưa con trỏ về đầu dòng tiếp theo
   MOV   AH, 2
   MOV   DL, 0Dh
   INT   21h
   MOV   DL, 0Ah
   INT   21h
   RET
 Writeln  ENDP
END MAIN
```

Quan hệ giữa thủ tục và ngăn xếp

- Khi lệnh Call gọi một thủ tục thì các lệnh của thủ tục đó sẽ thi hành. Vậy làm cách nào để quay trở về chương trình chính sau khi thủ tục thi hành xong?

Trình tự thực hiện lệnh của bộ vi xử lý 8086:

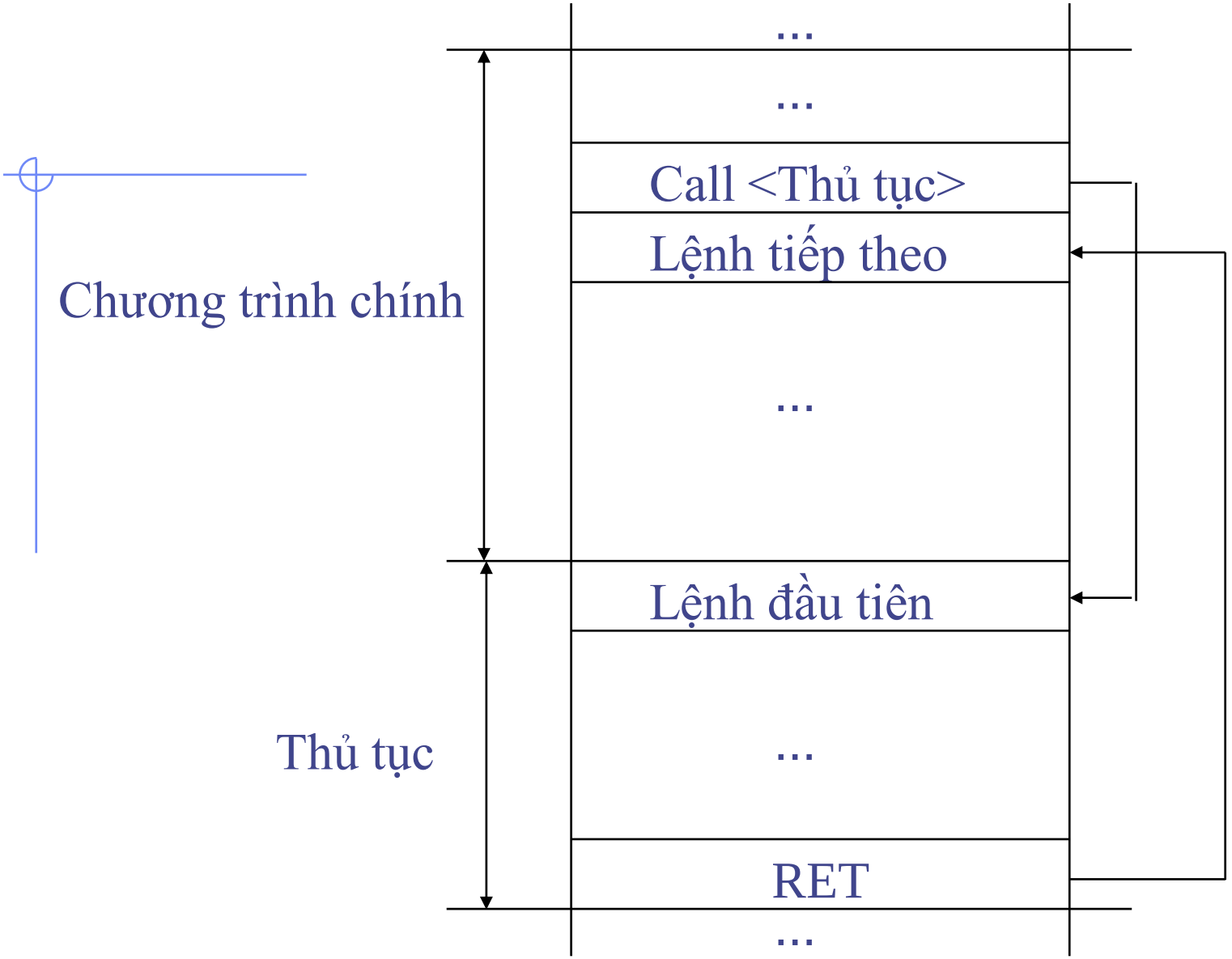
- Đoạn mã lệnh có địa chỉ segment nằm trong thanh ghi CS, còn offset của các lệnh sẽ được đặt vào thanh ghi con trỏ lệnh IP (Instruction Pointer).
- Như vậy cặp thanh ghi CS:IP chứa địa chỉ của ô nhớ nào thì lệnh tại ô nhớ đó sẽ được thi hành.

Khi sử dụng lệnh Call thì các công việc sau đây được thực hiện:

- Cất địa chỉ của lệnh đứng sau lệnh Call (trong chương trình chính) vào ngăn xếp.
- Nạp địa chỉ lệnh đầu tiên của thủ tục vào cặp thanh ghi CS:IP (tức là thi hành lệnh này).

- Lần lượt các lệnh trong thủ tục sẽ được thi hành cho tới khi gặp lệnh RET.
- Lệnh RET sẽ lấy địa chỉ lệnh từ ngăn xếp (do lệnh Call cất trước đó) rồi nạp vào các thanh ghi CS:IP. Như vậy quyền điều khiển đã được trả về chương trình chính

(xem sơ đồ trang sau)


...
...
Call <Thủ tục>
Lệnh tiếp theo
...
Chương trình chính
Lệnh đầu tiên
...
Thủ tục
RET
...

Hết Phần 2.3

KIẾN TRÚC MÁY TÍNH

Giảng viên: Ths Phạm Thanh Bình
Bộ môn Kỹ thuật máy tính & mạng
http://vn.myblog.yahoo.com/CNTT-wru
http://ktmt.wru.googlepages.com

2.6.8 Mảng và các chế độ địa chỉ

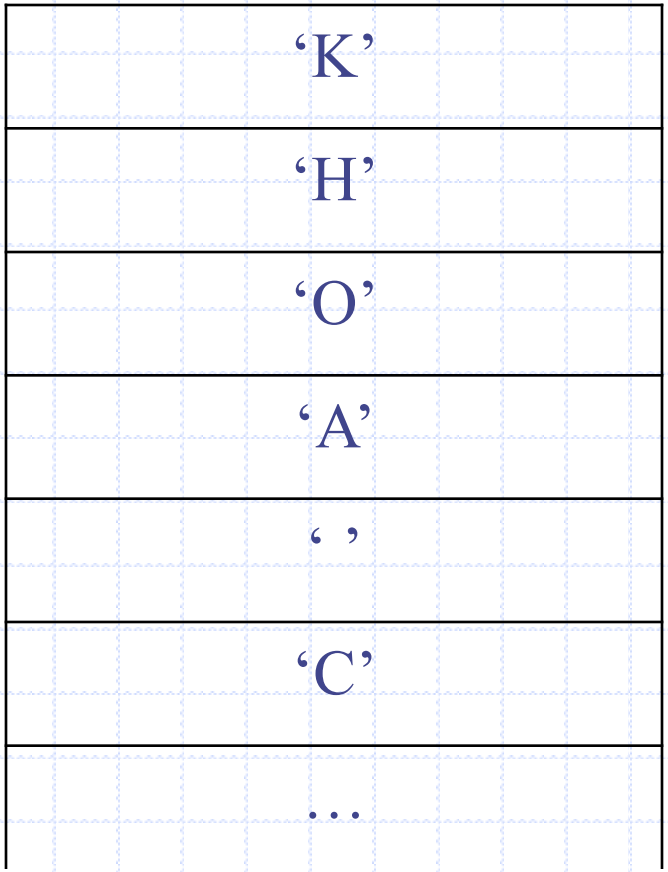
- Khai báo mảng một chiều
- Các phần tử của mảng một chiều
- Các chế độ địa chỉ
- Các lệnh thao tác với chuỗi

Khai báo mảng một chiều

- Mảng một chiều gồm một chuỗi liên tiếp các byte hay word trong bộ nhớ.
- Ở Chương 2 ta đã từng sử dụng khai báo:

```
.DATA
    ChuoiKT   DB   'KHOA CONG NGHE THONG TIN$'
```

- Thực chất khai báo này sẽ chiếm một vùng 25 ô nhớ trong đoạn dữ liệu và đặt vào đó các kí tự tương ứng:


‘K’
‘H’
‘O’
‘A’
‘ ’
‘C’
. . .

- Cách khai báo như trên tương đương với cách khai báo sau đây:

```
.DATA
    ChuoiKT   DB   'K', 'H', 'O', 'A', 'CONG', 'NGHE', 'THONG TIN$'
```

- Và cũng tương đương với:

```
.DATA
    ChuoiKT   DB   4Bh, 48h, 4Fh, 41h, 'CONG', 'NGHE', 'THONG TIN$'
```

- Các khai báo đó được gọi là khai báo liệt kê, tức là sẽ tạo ra trong bộ nhớ một mảng có số lượng phần tử xác định, đồng thời khởi tạo luôn giá trị cho từng phần tử

Các phương pháp khai báo tổng quát

Khai báo mảng Byte:

Mảng Byte là mảng mà mỗi phần tử có kích thước 1 byte.

- Cách 1:

<Tên mảng> DB <liệt kê các phần tử của mảng>

- Ví dụ:

A DB 10h, 12h, 30, 40

Khai báo trên tạo ra mảng A có 4 phần tử, mỗi phần tử chiếm 1 byte nhớ

- Cách 2:

<Tên mảng> DB <Số phần tử của mảng> DUP (Giá trị khởi tạo)

- Ví dụ 1:

A DB 50 DUP (0)

Khai báo trên tạo ra mảng A có 50 phần tử, giá trị ban đầu của các phần tử bằng 0.

- Ví dụ 2:

B DB 100 DUP (?)

Khai báo trên tạo ra mảng B có 100 phần tử, không khởi tạo giá trị ban đầu cho các phần tử

Khai báo mảng Word:

Mảng Word là mảng mà mỗi phần tử có kích thước 1 word.

- Cách 1:

<Tên mảng> DW <liệt kê các phần tử của mảng>

- Cách 2:

<Tên mảng> DW <Số phần tử của mảng> DUP (Giá trị khởi tạo)

- Ví dụ 1:

```
A    DW   10h, 12h, 30, 40
```

Khai báo trên tạo ra mảng A có 4 phần tử, mỗi phần tử dài 16 bít.

- Ví dụ 2:

```
B    DW   50 DUP (?)
```

Các phần tử của mảng một chiều

- Tên mảng chính là một biến ứng với phần tử đầu tiên của mảng. Các phần tử tiếp theo có thể được xác định bằng cách lấy địa chỉ phần tử đứng trước cộng với kích thước của nó.

Ví dụ 1:

```
M     DB   10,  20,  30,  40
```

Các phần tử của mảng có thể kí hiệu như sau (chú ý: kích thước của mỗi phần tử trong mảng này là 1 byte):

	Kí hiệu	Giá trị
Phần tử 1	M	10
Phần tử 2	M + 1	20
Phần tử 3	M + 2	30
Phần tử 4	M + 3	40

Ví dụ 2:

N DW 1, 6, 20, 10, 15

Các phần tử của mảng có thể kí hiệu như sau (chú ý: kích thước của mỗi phần tử trong mảng này là 2 byte):

	Kí hiệu	Giá trị
Phần tử 1	N	1
Phần tử 2	N + 2	6
Phần tử 3	N + 4	20
Phần tử 4	N + 6	10
Phần tử 5	N + 8	15

- Ví dụ 3:

  Cho mảng A gồm 12 phần tử, các phần tử có kiểu là Byte. Hãy đổi chỗ phần tử đầu tiên và phần tử cuối cùng của mảng cho nhau.

- Giải:

  Phần tử đầu tiên là: A

  Phần tử cuối cùng là: A + 11

```
MOV   AL, A
MOV   BL, A + 11
MOV   A, BL
MOV   A+11, AL
```

Các chế độ địa chỉ

- Việc truy nhập trực tiếp tới các phần tử của mảng thông qua cách viết: <Tên mảng> + <Khoảng cách> gây rất nhiều bất tiện trong lập trình.
- Một phương pháp khác, mềm dẻo hơn, là sử dụng các thanh ghi để chứa <Khoảng cách> hoặc chứa địa chỉ của từng phần tử. Bằng việc thay đổi nội dung các thanh ghi → có thể truy nhập vào các phần tử khác nhau của mảng.
- Các thanh ghi có thể được sử dụng là BX, BP, SI, DI

Dùng thanh ghi chứa địa chỉ của phần tử:

- Giả sử thanh ghi SI đang chứa địa chỉ offset của một ô nhớ nào đó, cách viết: [SI] sẽ trả về nội dung của ô nhớ đó.
- Nếu sử dụng các thanh ghi BX, DI và SI để chứa địa chỉ offset thì địa chỉ segment sẽ được chứa trong DS. Còn nếu sử dụng thanh ghi BP thì SS sẽ chứa segment

- Ví dụ:

Cho mảng sau:

```
A    DB   10, 12, 3, 4, 9, 5, 7, 6
```

Hãy tính tổng các phần tử của mảng (cất tổng vào AL).

- Giải:

Ta sẽ sử dụng thanh ghi SI lần lượt trỏ tới từng phần tử của mảng để thực hiện phép tính tổng.

```
XOR   AL, AL        ;Xoá AL để chuẩn bị chứa tổng
LEA   SI, A         ;SI chứa địa chỉ offset phần tử đầu
                    ;  tiên của mảng
MOV   CX, 8         ;Số lần lặp (mảng có 8 phần tử)
Lap:
  ADD   AL, [SI]    ;Cộng phần tử của mảng vào AL
   INC   SI         ;SI trỏ tới phần tử tiếp theo
LOOP  Lap
```

- Cách viết như trên được gọi là *Chế độ địa chỉ gián tiếp thanh ghi*.

Dùng thanh ghi để chứa <Khoảng cách>

- Trong phương pháp này, muốn truy nhập vào một phần tử của mảng thì cần phải biết được <Khoảng cách> từ phần tử đó tới đầu mảng.
- Các phần tử của mảng sẽ được kí hiệu như sau:

  <Tên mảng> [Thanh ghi]

  Trong đó Thanh ghi sẽ chứa <Khoảng cách> của phần tử tính từ đầu mảng

- Ví dụ 1:

  Kí hiệu: A[BX]

  A: là tên mảng

  BX: là thanh ghi chứa <Khoảng cách>

  Nếu BX = 0 thì A[BX] chính là phần tử đầu tiên của mảng.

Ví dụ 2:

Viết lại đoạn chương trình tính tổng các phần tử của mảng bằng một cách khác.

Cho mảng sau:

```
A    DB   10, 12, 3, 4, 9, 5, 7, 6
```

Hãy tính tổng các phần tử của mảng (cất tổng vào AL).

Giải:

```
XOR   AL, AL            ;Xoá AL để chuẩn bị chứa tổng
XOR   BX, BX            ;<Khoảng cách> = 0: phần tử đầu tiên của mảng
MOV   CX, 8             ;Số lần lặp (mảng có 8 phần tử)
Lap:
   ADD   AL, A[BX]      ;Cộng phần tử của mảng vào AL
    INC    BX           ;tăng <Khoảng cách> để trỏ tới phần tử tiếp theo
LOOP  Lap
```

- Ngoài cách kí hiệu A[BX] còn có thể sử dụng các kí hiệu khác tương đương như [A + BX], [BX + A], A + [BX], [BX] + A.
- Nếu sử dụng các thanh ghi BX (Base Register) hay BP (Base Pointer) trong cách viết trên thì gọi là *Chế độ địa chỉ cơ sở*, còn nếu sử dụng SI (Source Index) hay DI (Destination Index) thì gọi là *Chế độ địa chỉ chỉ số*.

Các lệnh thao tác với chuỗi

- *Lệnh chuyển chuỗi (Moving a String)*
- *Lệnh chuyển dữ liệu từ thanh ghi vào chuỗi (Store a String)*
- *Lệnh chuyển dữ liệu từ chuỗi vào thanh ghi (Load a String)*

- Trong các lệnh thao tác với chuỗi, có hai cặp thanh ghi hay được sử dụng là DS:SI và ES:DI.
- Nếu dùng DS để chứa segment thì SI sẽ chứa offset, và nếu ES chứa segment thì DI sẽ chứa offset.
- Để ES cũng chứa địa chỉ của đoạn dữ liệu giống như DS thì ở đầu của chương trình chính phải có các lệnh:

```
MOV   AX, @DATA
MOV   DS, AX
MOV   ES, AX
```

Lệnh chuyển chuỗi (Moving a String)

Lệnh này còn được gọi là lệnh sao chép chuỗi.

a) Chuyển một lần:

- Dạng 1: MOVSB

  Lệnh trên sao chép 1 byte dữ liệu từ ô nhớ có địa chỉ DS:SI sang ô nhớ có địa chỉ ES:DI.

- Dạng 2: MOVSW

  Lệnh trên sao chép 1 word dữ liệu từ ô nhớ có địa chỉ DS:SI sang ô nhớ có địa chỉ ES:DI.

Ví dụ 1:

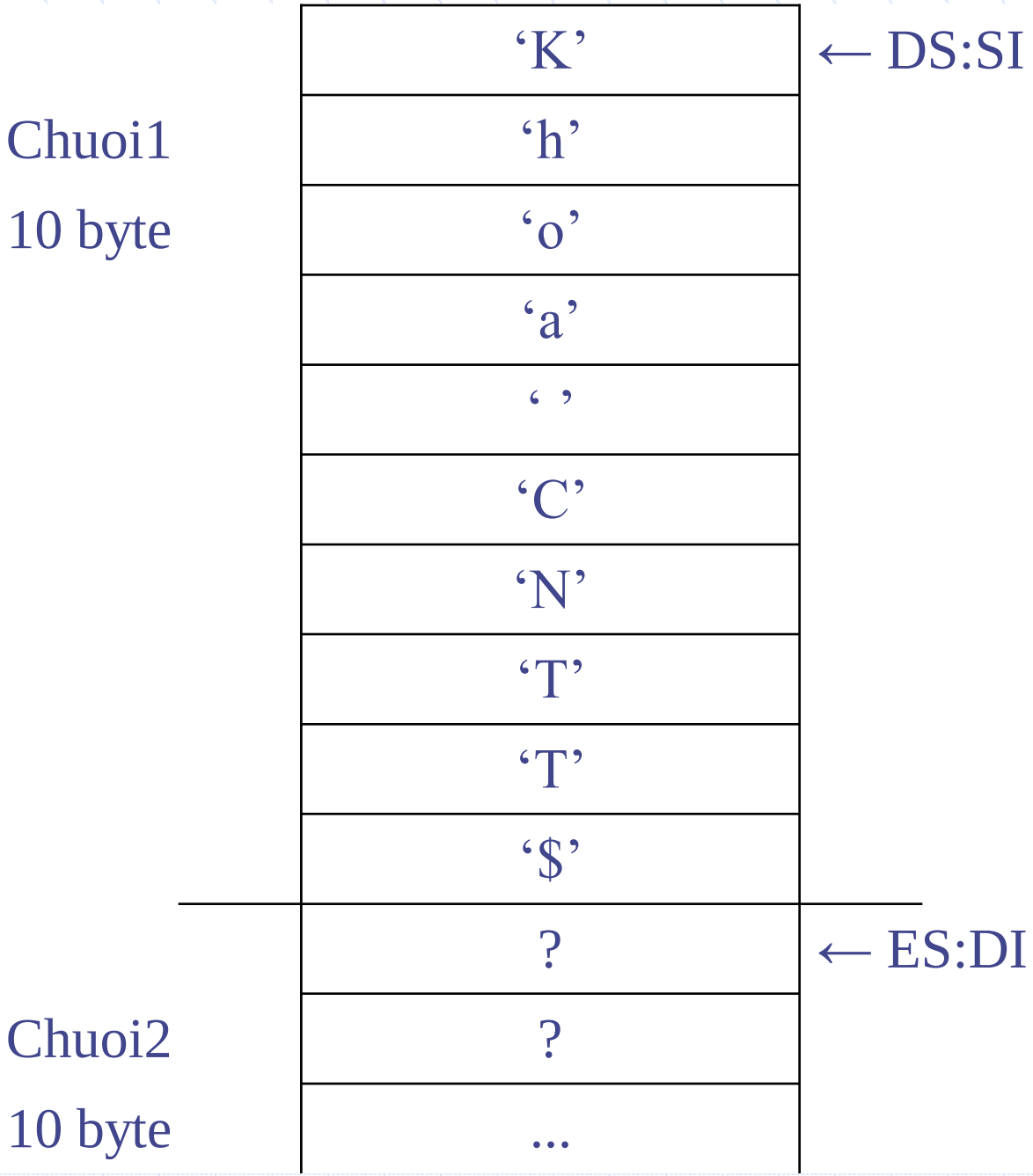
Xét hai chuỗi được khai báo như sau:

```
.DATA
    Chuoi1   DB   'Khoa CNTT$'
    Chuoi2   DB  10  DUP  (?)
```

Hãy sao chép nội dung của Chuoi1 sang Chuoi2.

- Giải:

Để thực hiện yêu cầu trên ta sẽ lần lượt sao chép từng byte của Chuoi1 sang Chuoi2.


'K'
← DS:SI
Chuoi1
'h'
10 byte
'o'
'a'
' '
'C'
'N'
'T'
'T'
'$'
?
← ES:DI
Chuoi2
?
10 byte
...

- Muốn sao chép byte đầu tiên (kí tự 'K') thì DS:SI phải chứa địa chỉ đầu của Chuoi1, ES:DI phải chứa địa chỉ đầu của Chuoi2. Điều này được thực hiện bởi các lệnh sau:

```
LEA   SI, Chuoi1      ;SI chứa offset của Chuoi1
LEA   DI, Chuoi2      ;DI chứa offset của Chuoi2
MOVSB                 ;Chuyển 1 byte
```

- Mỗi chuỗi có độ dài 10 byte nên phải lặp lại quá trình trên 10 lần thì mới sao chép xong.

Chú ý:

- Mỗi khi sao chép xong 1 byte thì phải tăng SI và DI lên 1 để nó trỏ tới ô nhớ tiếp theo.
- Sau mỗi lệnh MOVSB thì SI và DI sẽ được tự động tăng lên 1 nếu cờ DF = 0 (SI & DI sẽ tự động giảm đi 1 nếu cờ DF=1). Như vậy vấn đề là phải xoá được cờ DF trước khi thi hành lệnh MOVSB. Điều này được thực hiện nhờ lệnh CLD (Clear Direction Flag):

```
LEA   SI, Chuoi1        ;SI chứa offset của Chuoi1
LEA   DI, Chuoi2        ;DI chứa offset của Chuoi2
CLD                     ;Xoá cờ định hướng: DF = 0
MOVSB                   ;Chuyển 1 byte
```

- Ngược lại với lệnh CLD là lệnh STD (Set Direction Flag), lệnh này sẽ thiết lập cờ DF=1. Ta có thể sử dụng lệnh STD để chuyển các byte dữ liệu theo chiều ngược lại.

```
TITLE  Vi du Chuoi
.MODEL  SMALL
.STACK  100H
.DATA
   Chuoi1   DB   'Khoa CNTT$'
   Chuoi2   DB  10  DUP  (?)
.CODE
MAIN   PROC
   MOV   AX, @DATA
   MOV   DS, AX
   MOV   ES, AX   ;DS và ES chứa segment của đoạn dữ liệu
    MOV   CX, 10          ;Số lần lặp
    LEA   SI, Chuoi1      ;SI chứa offset của Chuoi1
    LEA   DI, Chuoi2      ;DI chứa offset của Chuoi2
    CLD                   ;Xoá cờ định hướng: DF = 0
    Lap:
      MOVSB               ;Thực hiện lặp 10 lần
    LOOP  Lap
   MOV   AH, 9h     ;Hiển thị chuỗi 2 để kiểm tra kết quả
   LEA   DX, Chuoi2
   INT   21h
   MOV   AH, 4Ch         ;Kết thúc
   INT   21h
MAIN ENDP
END MAIN
```

Ví dụ 2:

Làm lại ví dụ 1 bằng cách dùng lệnh MOVSW.

Giải:

Do lệnh MOVSW mỗi lần sao chép được 2 byte nên chỉ phải thực hiện lặp 5 lần, các lệnh cụ thể như sau:

```
MOV   CX, 5              ;Số lần lặp
LEA   SI, Chuoi1         ;SI chứa offset của Chuoi1
LEA   DI, Chuoi2         ;DI chứa offset của Chuoi2
CLD                      ;Xoá cờ định hướng: DF = 0
Lap:
   MOVSW                 ;Thực hiện lặp 5 lần
LOOP Lap
```

b) Chuyển nhiều lần:

- Dạng 1: REP MOVSB

  Lệnh trên sao chép nhiều byte dữ liệu từ ô nhớ có địa chỉ DS:SI sang ô nhớ có địa chỉ ES:DI, số byte cần chuyển chứa trong thanh ghi CX.

- Dạng 2: REP MOVSW

  Lệnh trên sao chép nhiều word dữ liệu từ ô nhớ có địa chỉ DS:SI sang ô nhớ có địa chỉ ES:DI, số word cần chuyển chứa trong thanh ghi CX.

- Ví dụ:

  Để thực hiện việc sao chép nội dung của Chuoi1 sang Chuoi2 trong ví dụ trước, ta có thể viết lại các lệnh như sau:

```
MOV   CX, 10          ;Số byte cần chuyển
LEA   SI, Chuoi1      ;SI chứa offset của Chuoi1
LEA   DI, Chuoi2      ;DI chứa offset của Chuoi2
CLD                   ;Xoá cờ định hướng: DF = 0
REP   MOVSB
```

- hoặc dùng các lệnh:

```
MOV   CX, 5          ;Số word cần chuyển
LEA   SI, Chuoi1     ;SI chứa offset của Chuoi1
LEA   DI, Chuoi2     ;DI chứa offset của Chuoi2
CLD                  ;Xoá cờ định hướng: DF = 0
REP   MOVSW
```

Lệnh chuyển dữ liệu từ thanh ghi vào chuỗi (Store a String)

Lệnh này còn được gọi là lệnh lưu chuỗi.

- Dạng 1: STOSB

  Lệnh trên chuyển nội dung của thanh ghi AL (1 byte) tới ô nhớ có địa chỉ ES:DI.

- Dạng 2: STOSW

  Lệnh trên chuyển nội dung của thanh ghi AX (2 byte) tới ô nhớ có địa chỉ ES:DI.

- Ví dụ 1:

  Xét chuỗi sau đây:

```
.DATA
    ChuoiKT   DB   6  DUP  (?)
```

  Hãy nhập một kí tự từ bàn phím rồi đặt kí tự đó vào phần tử đầu tiên của chuỗi.

Giải:

- Ta sẽ sử dụng chức năng số 1 của ngắt 21h để nhập kí tự, thanh ghi AL sẽ chứa mã ASCII của kí tự đó.
- Muốn chuyển kí tự từ AL vào phần tử đầu tiên của chuỗi bằng lệnh STOSB thì ES:DI phải chứa địa chỉ đầu của ChuoiKT, điều đó được thực hiện nhờ lệnh sau:

```
LEA   DI, ChuoiKT
```

```
TITLE  Vi du Chuoi
.MODEL  SMALL
.STACK  100H
.DATA
   ChuoiKT   DB   6  DUP  (?)
.CODE
MAIN   PROC
   MOV   AX, @DATA
   MOV   DS, AX
   MOV   ES, AX   ;DS và ES chứa segment của đoạn dữ liệu
   MOV   AH, 1    ;Chức năng nhập kí tự của ngắt 21h
   INT   21h
   LEA   DI, ChuoiKT    ;DI chứa offset của ChuoiKT
   STOSB                ;Chuyển kí tự từ AL vào đầu chuỗi
   MOV   AH, 4Ch        ;Kết thúc
   INT   21h
MAIN ENDP
END MAIN
```

- Ví dụ 2:

  Nhập một chuỗi 10 kí tự từ bàn phím.

- Giải:

  Để nhập 10 kí tự và cất nó vào một chuỗi trong bộ nhớ ta vẫn sử dụng phương pháp như ví dụ 1. Có một số điểm khác biệt như sau:

  + Phải có một vòng lặp với số lần lặp bằng 10.

  + Sau mỗi lệnh lưu chuỗi (STOSB) thì DI phải được tăng lên 1 để trỏ tới ô nhớ tiếp theo. Điều này được thực hiện nhờ lệnh xoá cờ định hướng (CLD).

```
TITLE  Vi du 2
.MODEL  SMALL
.STACK  100H
.DATA
   ChuoiKT   DB   10  DUP  (?)
.CODE
MAIN   PROC
   MOV   AX, @DATA
   MOV   DS, AX
   MOV   ES, AX          ;DS và ES chứa segment của đoạn dữ liệu
   MOV   CX, 10          ;Số lần lặp bằng 10
   LEA   DI, ChuoiKT     ;DI chứa offset của ChuoiKT
   CLD                   ;Xoá cờ định hướng: DF = 0
   Lap:
      MOV   AH, 1        ;Chức năng nhập kí tự của ngắt 21h
      INT   21h

      STOSB              ;Chuyển kí tự từ AL vào đầu chuỗi
                         ;DI được tự động tăng lên 1
   LOOP   Lap
   MOV   AH, 4Ch      ;Kết thúc
   INT   21h
MAIN ENDP
END MAIN
```

Lệnh chuyển dữ liệu từ chuỗi vào thanh ghi (Load a String)

Lệnh này còn được gọi là lệnh nạp chuỗi.

- Dạng 1: LODSB

  Lệnh trên chuyển 1 byte dữ liệu từ ô nhớ có địa chỉ DS:SI vào thanh ghi AL.

- Dạng 2: LODSW

  Lệnh trên chuyển 1 word dữ liệu từ ô nhớ có địa chỉ DS:SI vào thanh ghi AX.

- Ví dụ:

  Xét chuỗi sau đây:

  .DATA

  ChuoiKT DB ‘Viet Nam’

  Hãy hiển thị chuỗi ra màn hình.

Giải:

- Vì chuỗi không kết thúc bằng dấu '$' nên không thể hiện chuỗi bằng chức năng số 9 của ngắt 21h.
- Ta sẽ cho hiện lần lượt các kí tự của chuỗi bằng chức năng số 2 của ngắt 21h (các tham số: AH = 2, DL = Mã ASCII của kí tự cần hiển thị). Chuỗi có 8 kí tự nên cần 8 lần lặp.
- Đầu tiên cần chuyển từng kí tự từ chuỗi vào thanh ghi AL bằng lệnh LODSB, sau đó chuyển từ AL sang DL, rồi gọi chức năng số 2 của ngắt 21h.

```
TITLE  Vi du
.MODEL  SMALL
.STACK  100H
.DATA
   ChuoiKT   DB   'Viet Nam'
.CODE
MAIN   PROC
   MOV   AX, @DATA
   MOV   DS, AX              ;DS chứa segment của đoạn dữ liệu
   MOV   CX, 8               ;Số lần lặp bằng 8
   LEA   SI, ChuoiKT         ;SI chứa offset của ChuoiKT
   CLD                       ;Xoá cờ định hướng: DF = 0
   Lap:
      LODSB                 ;Chuyển kí tự từ chuỗi vào AL
                            ;SI được tự động tăng lên 1 (để trỏ tới kí tự tiếp theo)
      MOV   DL, AL          ;Chuyển kí tự vào DL
      MOV   AH, 2           ;Hiển thị kí tự
      INT   21h
   LOOP   Lap
   MOV   AH, 4Ch            ;Kết thúc
   INT   21h
MAIN ENDP
END MAIN
```

Hết Phần 2.4

KIẾN TRÚC MÁY TÍNH

Giảng viên: Ths Phạm Thanh Bình
Bộ môn Kỹ thuật máy tính & mạng
http://vn.myblog.yahoo.com/CNTT-wru
http://ktmt.wru.googlepages.com

Chương 3:

CÁC PHÉP TOÁN TRÊN MÁY TÍNH

- Nhắc lại về phép cộng và phép trừ
- Phép nhân
- Phép chia
- Số thực dấu phảy động

Bài 3.1 - Nhắc lại về phép cộng và phép trừ

A	B	A + B
0	0	0
0	1	1
1	0	1
1	1	10

Ví dụ 1:

- Thực hiện phép cộng sau trong máy tính: 7 + 6

$$\begin{array}{r} 0000\ 0111 \\ +\ 0000\ 0110 \\ \hline 0000\ 1101 \end{array} = (13)_{10}$$

- Ta thấy tổng thu được bằng 13, giống như cộng số thập phân thông thường.

Ví dụ 2:

- Thực hiện phép trừ sau trong máy tính: 7 - 6

$$\begin{array}{rl} & 0000\ 0111 \\ - & 0000\ 0110 \\ \hline & 0000\ 0001 \end{array} = (1)_{10}$$

- Ta thấy hiệu thu được bằng 1, giống như trừ số thập phân thông thường

Ví dụ 3:

- So sánh kết quả Ví dụ 2 với phép cộng sau: 7 + (-6)

$$\begin{array}{r} 0000\ 0111 \\ +\ 1111\ 1010 \\ \hline 0000\ 0001 \end{array} = (1)_{10}$$

- Như vậy: 7 – 6 = 7 + (-6) = 1

Bài 3.2 - Phép nhân

- Nhân số nhị phân
- Giải thuật nhân
- Phần cứng thực hiện
- Các lệnh nhân của VXL 8086

Nhân số nhị phân

A	B	A * B
0	0	0
0	1	0
1	0	0
1	1	1

Ví dụ:

- Thực hiện phép nhân sau: 8 * 9

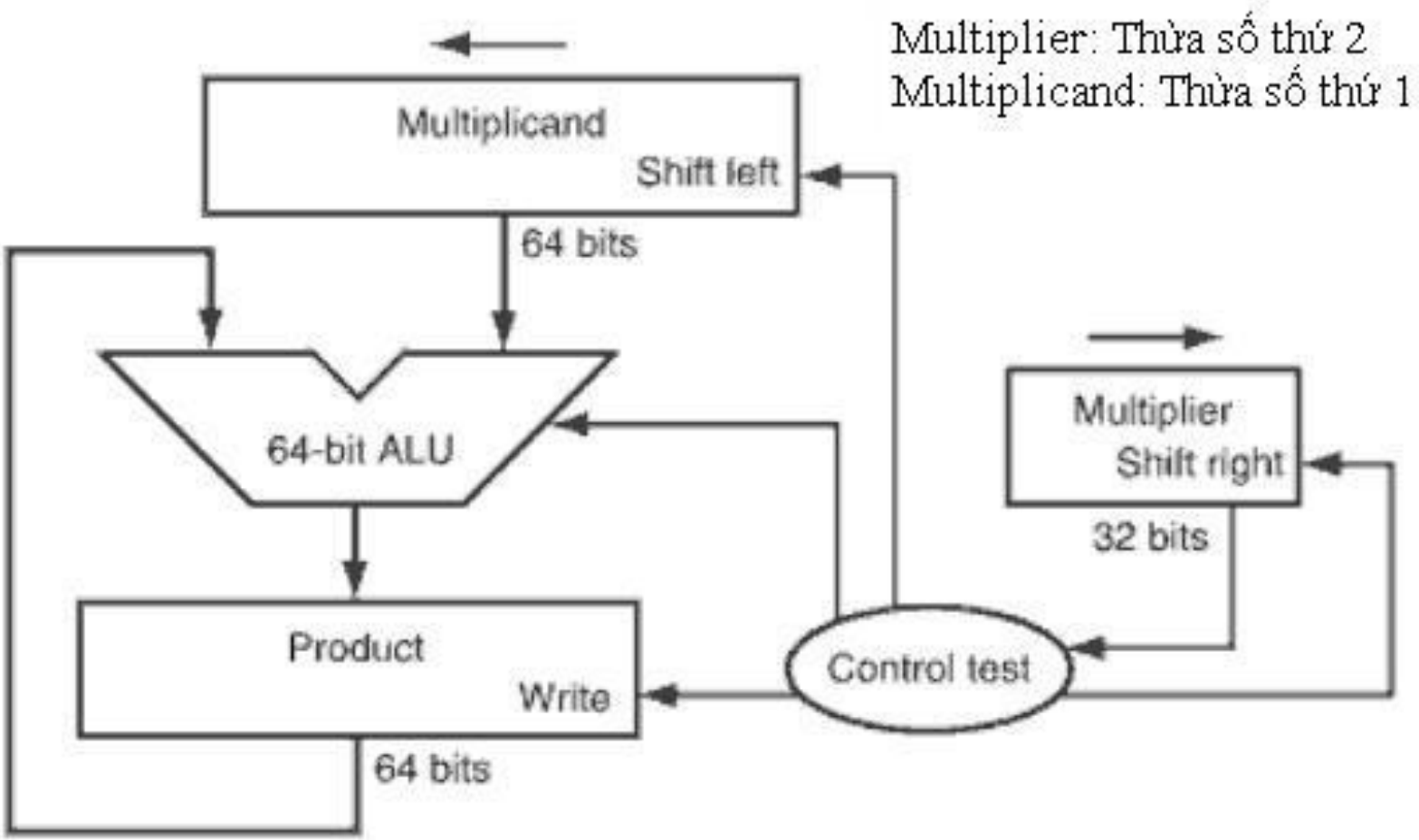
$$
\begin{array}{rr}
 & 1000 \\
+ & 1001 \\
\hline
 & 1000 \\
 & 0000 \\
 & 0000 \\
 & 1000 \\
\hline
 & 1001000
\end{array}
\quad = \ (72)_{10}
$$

- Ta thấy tích thu được bằng 72, giống như nhân số thập phân thông thường.

Giải thuật nhân:

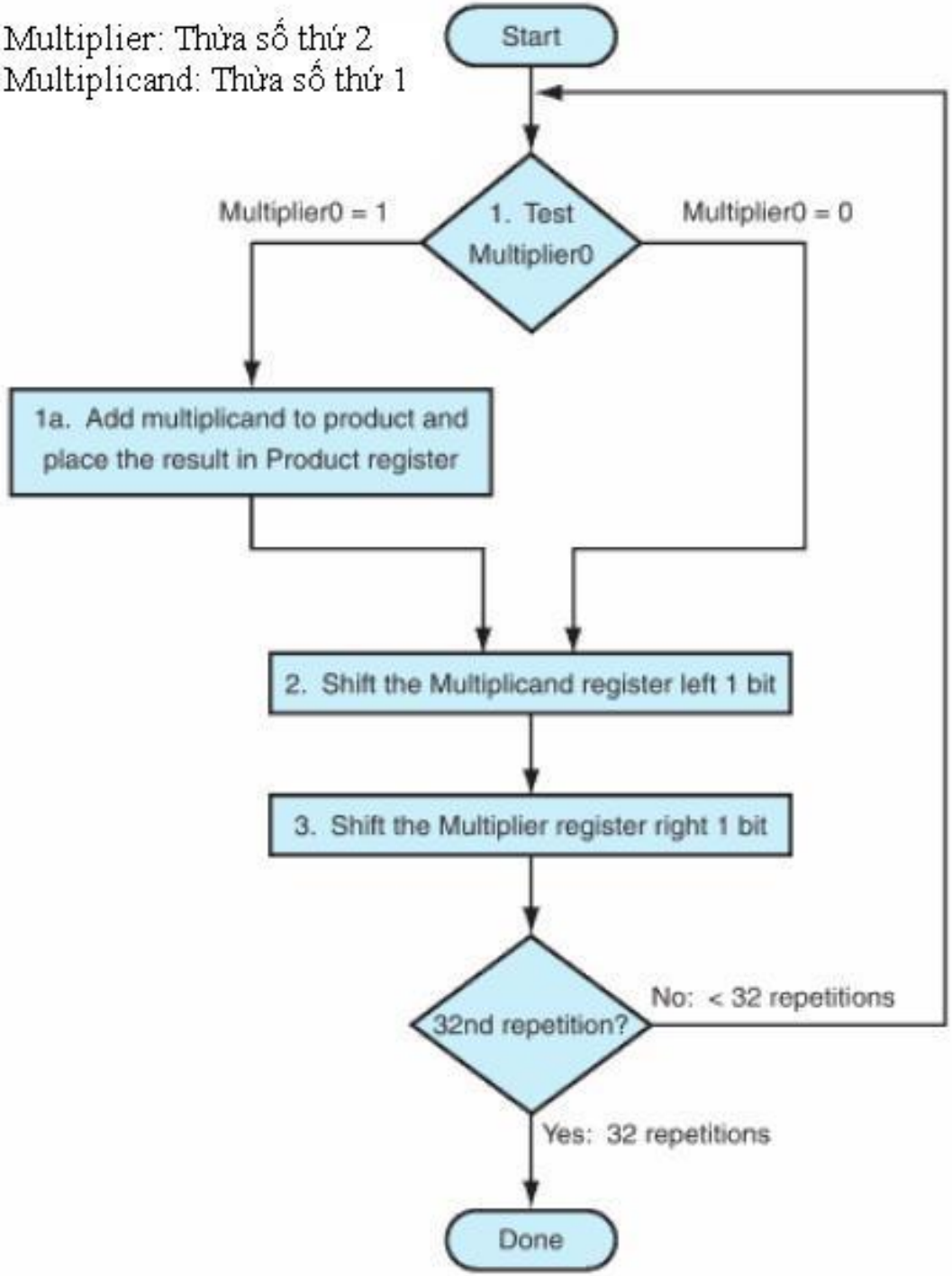
- Lần lượt nhân các bít của thừa số thứ hai với thừa số thứ nhất.
- Nếu gặp bít 1 thì chỉ việc giữ nguyên thừa số thứ nhất và đặt nó vào vị trí thích hợp.
- Nếu gặp bít 0 thì chỉ việc đặt một dãy toàn bít 0 vào vị trí thích hợp.
- Cộng các kết quả lại.

Phần cứng thực hiện



Giải thích:

- Thừa số 1 và Thừa số 2 dài 32 bít
- Thanh ghi chứa Thừa số 1 dài 64 bít
- Thanh ghi chứa Thừa số 2 dài 32 bít
- Thanh ghi chứa kết quả (Product) dài 64 bít
- Bộ cộng ALU dài 64 bít


Multiplier: Thừa số thứ 2
Multiplicand: Thừa số thứ 1
Start
1. Test Multiplier0
Multiplier0 = 1
Multiplier0 = 0
1a. Add multiplicand to product and place the result in Product register
2. Shift the Multiplicand register left 1 bit
3. Shift the Multiplier register right 1 bit
32nd repetition?
No: < 32 repetitions
Yes: 32 repetitions
Done

Ví dụ:

- Sử dụng phần cứng trên để thực hiện phép nhân: 2 x 3 (hay 0010b x 0011b)

Iteration	Step	Multiplier	Multiplicand	Product
0	Initial values	001(1)	0000 0010	0000 0000
1	1a: 1 ⇒ Prod = Prod + Mcand	0011	0000 0010	0000 0010
	2: Shift left Multiplicand	0011	0000 0100	0000 0010
	3: Shift right Multiplier	000(1)	0000 0100	0000 0010
2	1a: 1 ⇒ Prod = Prod + Mcand	0001	0000 0100	0000 0110
	2: Shift left Multiplicand	0001	0000 1000	0000 0110
	3: Shift right Multiplier	000(0)	0000 1000	0000 0110
3	1: 0 ⇒ no operation	0000	0000 1000	0000 0110
	2: Shift left Multiplicand	0000	0001 0000	0000 0110
	3: Shift right Multiplier	000(0)	0001 0000	0000 0110
4	1: 0 ⇒ no operation	0000	0001 0000	0000 0110
	2: Shift left Multiplicand	0000	0010 0000	0000 0110
	3: Shift right Multiplier	0000	0010 0000	0000 0110

- Vậy kết quả phép nhân là:

Tích = 0000 0110b = 6

Các lệnh nhân của VXL 8086

- *Lệnh MUL (Multiply)*
- *Lệnh IMUL (Integer Multiply)*

Lệnh MUL (Multiply)

- Lệnh này dùng để thực hiện phép nhân đối với các số không dấu.
- Cú pháp lệnh:

  MUL <Thừa số 1>

  <Thừa số 1>: là một thanh ghi hay một biến
- Nếu <Thừa số 1> có kích thước 1 byte thì <Thừa số 2> sẽ là thanh ghi AL. Lệnh trên sẽ thực hiện phép nhân giữa <Thừa số 1> và <Thừa số 2>, kết quả phép nhân được chứa trong thanh ghi AX (16 bit).
- Nếu <Thừa số 1> có kích thước 1 word thì <Thừa số 2> sẽ là thanh ghi AX. Kết quả phép nhân được chứa trong hai thanh ghi DX:AX (32 bit).

- Ví dụ:

  Hãy thực hiện phép nhân hai số: 51 và 5

- Giải:

Cách 1:

```
MOV   AL, 51
MOV   BL, 5
MUL   BL
```

Kết quả: Tích = AX = 255 = 00FFh (16 bit)

Cách 2:

```
MOV   AX, 51
MOV   BX, 5
MUL   BX
```

Kết quả: Tích = DX:AX = 255 = 0000 00FFh

(32 bit)

Nhận xét:

- Cả hai cách trên đều cho cùng một kết quả: Tích = 255 (8 bit: 1111 1111b). Tuy nhiên cách 1 vẫn phải dùng một thanh ghi 16 bít để chứa kết quả này. Cách 2 quá lãng phí tài nguyên vì nó phải dùng tới 2 thanh ghi 16 bít để chứa một giá trị dài 8 bít!
- Vấn đề đặt ra là phải xác định được độ dài của kết quả phép nhân nhằm tránh sự lãng phí tài nguyên trong các thao tác tiếp theo. Việc này được thực hiện bằng cách kiểm tra các cờ CF và OF.

- Trường hợp 1: <Thừa số 1> dài 8 bít:
  + Nếu sau phép nhân 2 cờ CF/OF = 0: Tích được chứa trong AL (8 bít)
  + Nếu sau phép nhân 2 cờ CF/OF = 1: Tích được chứa trong AX (16 bít)
- Trường hợp 2: <Thừa số 1> dài 16 bít:
  + Nếu sau phép nhân 2 cờ CF/OF = 0: Tích được chứa trong AX (16 bít)
  + Nếu sau phép nhân 2 cờ CF/OF = 1: Tích được chứa trong DX:AX (32 bít)

Bảng tổng hợp:

Kích thước <Thừa số 1>	Trạng thái cờ CF/OF	Nơi chứa kết quả nhân
8 bít	0	AL
	1	AX
16 bít	0	AX
	1	DX:AX

Lệnh IMUL (Integer Multiply)

- Lệnh này dùng để thực hiện phép nhân đối với các số có dấu.
- Cú pháp lệnh:

IMUL <Thừa số 1>

<Thừa số 1>: là một thanh ghi hay một biến

- Các vấn đề đã trình bày với lệnh MUL ở trên đều có thể áp dụng cho lệnh IMUL.

- Ví dụ:

  Hãy thực hiện phép nhân hai số: -64 và 2

- Giải:

```
MOV   AL, 2
MOV   BL, -64
IMUL   BL
```

Kết quả: Tích = AX = -128 (thực chất chỉ chứa trong AL vì kết quả dài 8 bit = 80h)

Bài 3.3 - Phép chia

- Chia số nhị phân
- Giải thuật chia
- Phần cứng thực hiện
- Các lệnh chia của VXL 8086

Chia số nhị phân

- Ví dụ:

  Thực hiện phép chia 74:8

  74 = 1001010 b

  8d = 1000 b

Giải:

1001010	1000
-1000	1001
10	
101	
1010	
- 1000	
dư 10	

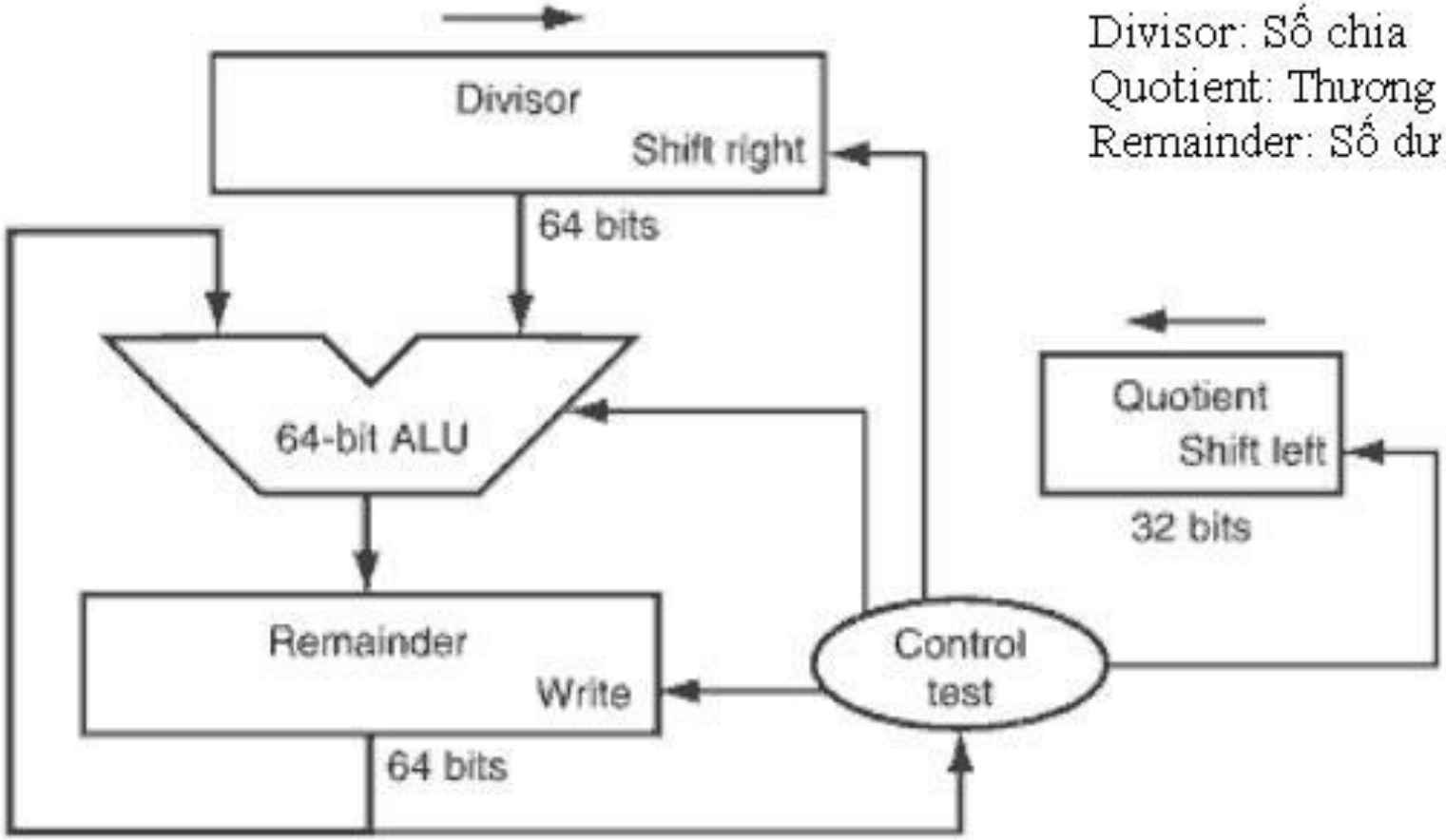
Như vậy Thương = 1001 b = 9 d, Dư = 10 b = 2, phù hợp với kết quả chia ở hệ thập phân.

Giải thuật chia

- Bước 1: Lấy các bít đầu của Số bị chia trừ đi Số chia
- Bước 2:

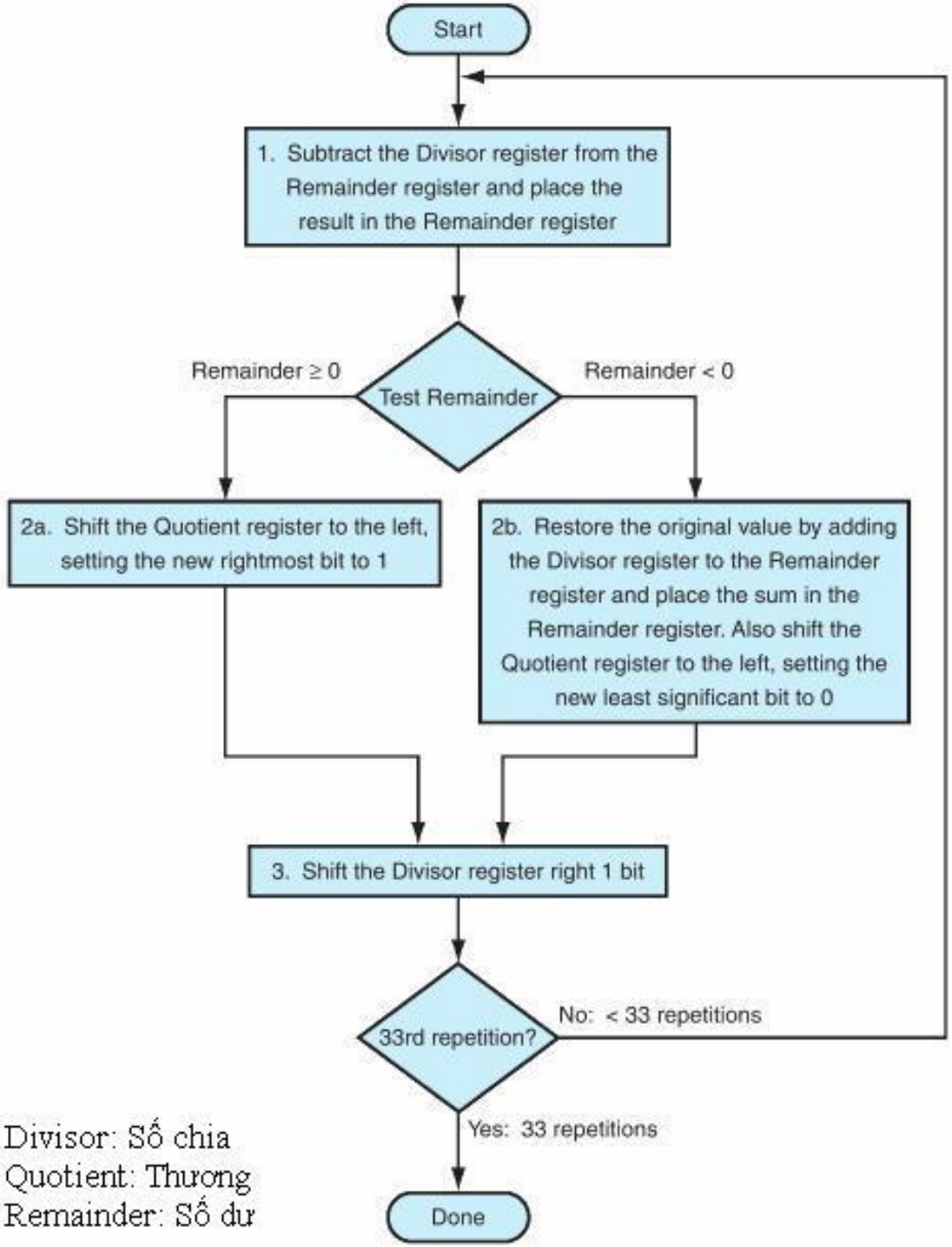
  Nếu Kết quả ≥ 0 thì thêm bít 1 vào Thương,
  Nếu Kết quả < 0 thì thêm bít 0 vào Thương.
- Bước 3: Ghép bít tiếp theo của Số bị chia vào Kết quả, rồi lặp lại Bước 1.
- Lặp lại quá trình trên cho tới hết các bít của Số bị chia

Phần cứng thực hiện



Giải thích:

- Ban đầu Số bị chia được đặt vào thanh ghi Remainder dài 64 bít
- Thanh ghi chứa Số chia (Divisor) dài 64 bít (Số chia chiếm 32 bít cao)
- Thanh ghi chứa Thương (Quotient) dài 32 bít
- Bộ trừ ALU dài 64 bít
- Kết thúc phép chia thì thanh ghi Remainder sẽ chứa số dư



Divisor: Số chia
Quotient: Thương
Remainder: Số dư

Ví dụ:

- Sử dụng phần cứng trên để thực hiện phép chia 7 : 2 (hay 0000 0111b : 0010b)

Iteration	Step	Quotient	Divisor	Remainder
0	Initial values	0000	0010 0000	0000 0111
1	1: Rem = Rem – Div	0000	0010 0000	1110 0111
	2b: Rem < 0 ⇒ +Div, sll Q, Q0 = 0	0000	0010 0000	0000 0111
	3: Shift Div right	0000	0001 0000	0000 0111
2	1: Rem = Rem – Div	0000	0001 0000	1111 0111
	2b: Rem < 0 ⇒ +Div, sll Q, Q0 = 0	0000	0001 0000	0000 0111
	3: Shift Div right	0000	0000 1000	0000 0111
3	1: Rem = Rem – Div	0000	0000 1000	1111 1111
	2b: Rem < 0 ⇒ +Div, sll Q, Q0 = 0	0000	0000 1000	0000 0111
	3: Shift Div right	0000	0000 0100	0000 0111
4	1: Rem = Rem – Div	0000	0000 0100	0000 0011
	2a: Rem ≥ 0 ⇒ sll Q, Q0 = 1	0001	0000 0100	0000 0011
	3: Shift Div right	0001	0000 0010	0000 0011
5	1: Rem = Rem – Div	0001	0000 0010	0000 0001
	2a: Rem ≥ 0 ⇒ sll Q, Q0 = 1	0011	0000 0010	0000 0001
	3: Shift Div right	0011	0000 0001	0000 0001

- Vậy kết quả phép chia là:

Thương = 0011b = 3

Số dư = 0001b = 1

Các lệnh chia của VXL 8086

- *Cú pháp lệnh*
- *Hiện tượng tràn trong phép chia*

Cú pháp lệnh

DIV <Số chia> ;Dùng cho số không dấu

IDIV <Số chia> ;Dùng cho số có dấu

<Số chia>: là một thanh ghi hay một biến

- Nếu <Số chia> có kích thước 1 byte thì Số bị chia sẽ được chứa trong AX (2 byte).

  Kết quả: Thương số chứa trong AL, Số dư chứa trong AH.

- Nếu <Số chia> có kích thước 2 byte thì Số bị chia sẽ được chứa trong DX:AX (4 byte).

  Kết quả: Thương số chứa trong AX, Số dư chứa trong DX.

- Ví dụ 1:

  Hãy thực hiện phép chia 65 cho 2.

- Giải:

```
MOV   AX, 65
MOV   BL, 2
DIV   BL
```

Kết quả: Thương = AL = 32

Số dư = AH = 1

- Ví dụ 2:

  Hãy thực hiện phép chia -128 cho 4.

- Giải:

```
MOV   AX, -128
MOV   BL, 4
IDIV  BL
```

Kết quả: Thương = AL = -32

Số dư = AH = 0

- Ví dụ 3:

  Hãy thực hiện phép chia -1024 cho 256.

- Giải:

+ Vì số chia bằng 256 (là một số 16 bit) nên số bị chia phải chứa trong DX:AX, tức là DX:AX = -1024.

+ Nhưng -1024 có thể chứa trọn vẹn trong thanh ghi AX (16 bit), muốn chuyển nó thành số 32 bit cần sử dụng lệnh CWD (Convert Word to Double Word). Lệnh này sẽ chuyển dữ lệu có dấu dạng Word trong AX thành dữ liệu có dấu dài 2 Word trong DX:AX.

```
MOV   AX, -1024
CWD
MOV   BX, 256
IDIV  BX
```

Kết quả: Thương = AX = -4

Số dư = DX = 0

Hiện tượng tràn trong phép chia

- Ví dụ:

  Hãy thực hiện phép chia 512 cho 2.

- Giải:

```
MOV   AX, 512
MOV   BL, 2
DIV   BL
```

Kết quả phép chia bằng 256, không thể chứa trong thanh ghi AL: Hiện tượng tràn xảy ra. Khi đó hệ thống sẽ đưa ra thông báo: "Divide OverFlow".

Bài 3.4 - Số thực dấu phảy động

- Số thực dạng nhị phân
- Biểu diễn số thực trên máy tính
- Phép cộng số thực
- Phép nhân số thực

Số thực dạng nhị phân

- Ví dụ 1:

  Chuyển số 0,75 sang dạng nhị phân.

- Các bước tiến hành như sau:

  + Bước 1: Lấy phần thập phân nhân với 2:

    0,75 x 2 = 1,5 → Thu được phần nguyên = 1, phần thập phân = 0,5

  + Bước 2: Lấy phần thập phân của kết quả bước trên nhân với 2:

    0,5 x 2 = 1,0 → Thu được phần nguyên = 1, phần thập phân = 0

  Quá trình trên được lặp đi lặp lại cho tới khi phần thập phân = 0 thì dừng lại.

- Kết quả:

  Viết lần lượt các phần nguyên thu được ở trên vào sau dấu phảy của số nhị phân:

$$(0,75)_{10} = (0,11)_2$$

- Ví dụ 2:

  Chuyển số 4,9 sang dạng nhị phân.

Giải:

Số trên có phần nguyên = 4 và phần thập phân = 0,9. Các bước tiến hành như sau:

+ Chuyển phần nguyên sang dạng nhị phân:

$$(4)_{10} = (100)_2$$

+ Chuyển phần thập phân sang dạng nhị phân (giống như ví dụ 1):

Bước 1: Lấy phần thập phân nhân với 2:

0,9 x 2 = 1,8 → Phần nguyên = 1, phần thập phân = 0,8

Bước 2: 0,8 x 2 = 1,6 → Phần nguyên = 1, phần thập phân = 0,6

Bước 3: 0,6 x 2 = 1,2 → Phần nguyên = 1, phần thập phân = 0,2

Bước 4: 0,2 x 2 = 0,4 → Phần nguyên = 0, phần thập phân = 0,4

Bước 5: 0,4 x 2 = 0,8 → Phần nguyên = 0, phần thập phân = 0,8

Bước 6: 0,8 x 2 = 1,6 → Phần nguyên = 1, phần thập phân = 0,6

...

Quá trình này sẽ không bao giờ kết thúc vì phần thập phân không thể bằng 0. Do đó ta chỉ có thể thu được một số nhị phân xấp xỉ với số thập phân đã cho.

Kết quả: $(4,9)_{10} = (100,11100\ 1100\ 1100...)_2$

Biểu diễn số thực trên máy tính

- Ví dụ 1:

  Xét số thực dạng nhị phân sau: $M = (10,1)_2$

  (bằng 2,5 trong hệ thập phân)

  + Số M cũng có thể được viết là:

  $$M = 10,1 \times 2^0 = 1,01 \times 2^1$$

  $$= \langle\text{Phần định trị}\rangle \times 2^{\langle\text{Phần mũ}\rangle}$$

  + Như vậy ta có thể thay đổi vị trí của dấu phảy mà giá trị của số thực vẫn giữ nguyên bằng cách điều chỉnh phần mũ và phần định trị.

Chuẩn hoá số thực:

- Phần định trị (Fraction):

  Để thống nhất trong cách biểu diễn, phần định trị của các số khác 0 thường phải thoả mãn điều kiện sau:

  $$1 \leq \text{Fraction} < 2$$

  Phần định trị thoả mãn điều kiện trên được gọi là phần định trị đã được chuẩn hoá.

Phần mũ (Exponent):

Trên thực tế phần mũ có thể là số âm hoặc số dương. Việc biểu diễn số mũ âm là phức tạp nên người ta thường cộng thêm một giá trị (gọi là Bias) vào phần mũ để nó luôn là số dương.

Bias được lựa chọn theo quy tắc sau:

Nếu độ dài dãy bít biểu diễn phần mũ là N thì Bias $= 2^{N-1} - 1$

(Ví dụ: nếu N = 8 thì Bias = $2^7 - 1 = 127$)

Ví dụ 2:

Xét số thực dạng nhị phân sau: M = 0,0001b

Hãy xác định phần định trị và phần mũ của M sau khi chuẩn hoá.

Giải:

$M = 0,0001b = 1,0 \times 2^{-4}$

Fraction = 1,0

Giả sử dùng 8 bít để biểu diễn phần mũ: Bias = 127

→ Exponent = -4 + 127 =123

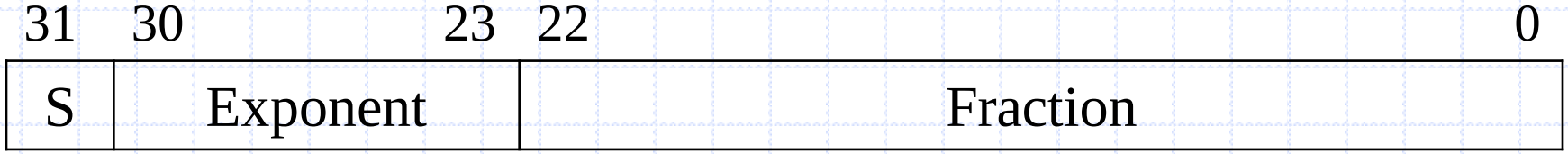
Số thực 32 bít (Short Real):

Có nhiều kiểu số thực trong máy tính, ta sẽ tìm hiểu cách thức để biểu diễn số thực bởi một dãy 32 bít. Dãy bít đó được chia làm 3 phần:

Phần dấu (Sign): Chiếm 1 bít, bằng 1 nếu dấu âm, bằng 0 nếu dấu dương.

Phần mũ: Dài 8 bít

Phần định trị: Dài 23 bít (phần định trị chỉ lấy các bít nằm sau dấu phảy, vì trước dấu phảy luôn là 1).



Ví dụ 1:

Chuyển số 4,9 sang dạng ShortReal.

Giải:

$$(4,9)_{10} = (100,11100\ 1100\ 1100...)_2$$

Chuẩn hoá:

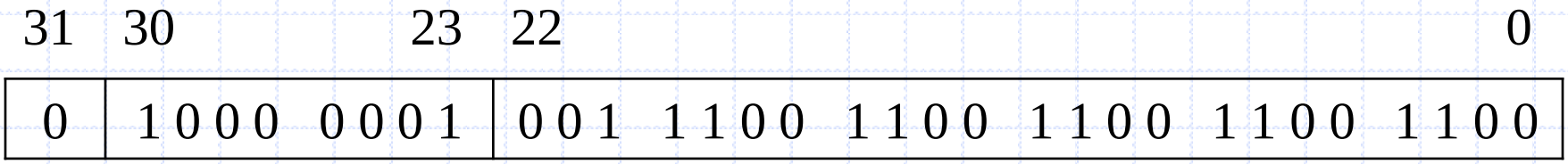
$$4,9 = 1,0011100\ 1100\ 1100...x\ 2^2$$

S = 0 (dấu dương)

Fraction = 0011100 1100 1100... (23 bít)

Exponent = 2 + 127 = 129 = 1000 0001b

→ Số thực dạng ShortReal:



Hay viết dưới dạng Hex: 4,9 ≈ 40 9C CC CCh

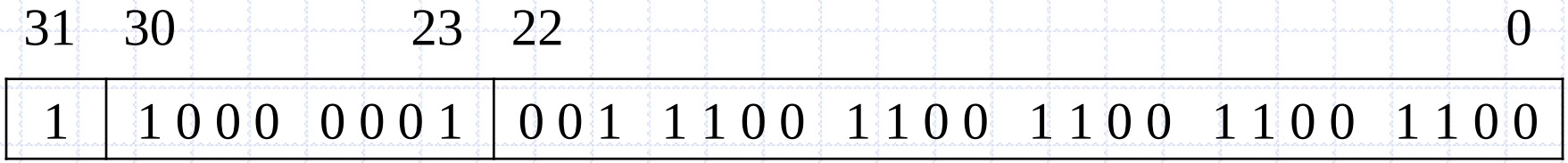
Ví dụ 2:

Chuyển số - 4,9 sang dạng ShortReal.

Giải:

Cách làm tương tự ví dụ 1, chỉ khác ở bít dấu: S = 1 (dấu âm)

→ Số thực dạng ShortReal:



Hay viết dưới dạng Hex: - 4,9 ≈ C0 9C CC CCh

Phép cộng số thực

Các bước để cộng 2 số thực:

Bước 1: Dịch chuyển dấu phảy của số thực có phần mũ nhỏ hơn cho tới khi phần mũ của hai số bằng nhau.

Bước 2: Cộng phần định trị của hai số.

Bước 3: Chuẩn hoá kết quả thu được.

Bước 4: Làm tròn kết quả (để phù hợp với độ dài dãy bít)

Ví dụ:

Thực hiện phép cộng giữa 0.5 và -0.4375

(hay giữa 1.000×2^{-1} và -1.110×2^{-2} trong hệ nhị phân)

Giải:

Bước 1: $-1.110 \times 2^{-2} = -0.111 \times 2^{-1}$

Bước 2: $1.000 \times 2^{-1} + (-0.111 \times 2^{-1}) = 0.001 \times 2^{-1}$

Bước 3: $0.001 \times 2^{-1} = 0.010 \times 2^{-2} = 0.100 \times 2^{-3}$

$= 1.000 \times 2^{-4}$

Bước 4: Vì kết quả dài đúng 4 bít, nên không cần làm tròn nữa: Tổng = 1.000×2^{-4}

Kiểm tra kết quả:

$1.000 \times 2^{-4} = 0.0001b = (1/2^4)_{10} = 0.0625$

⇒ Phù hợp với kết quả phép cộng:

$0.5 + (-0.4375) = 0.0625$

Giải thuật và phần cứng phép cộng số thực:

(Sinh viên tự nghiên cứu trong sách *Computer Organization and Design* – *trang 200, 201*)

Phép nhân số thực

Các bước để nhân 2 số thực:

Bước 1: Cộng phần mũ của hai số (Chú ý: trên máy tính thì kết quả thu được phải trừ đi Bias).

Bước 2: Nhân phần định trị của hai số với nhau.

Bước 3: Chuẩn hoá kết quả thu được.

Bước 4: Làm tròn kết quả (để phù hợp với độ dài dãy bít).

Bước 5: Xác định dấu của kết quả.

Ví dụ:

Thực hiện phép nhân giữa 0.5 và -0.4375

(hay giữa 1.000×2^{-1} và -1.110×2^{-2} trong hệ nhị phân)

Giải:

Bước 1: Cộng phần mũ: -1 + (-2) = -3

Nếu sử dụng Bias thì phần mũ mới sẽ là:

(-1 + 127) + (-2+127) – 127 = -3 + 127 = 124

Bước 2: Nhân phần định trị:

$$
\begin{array}{r}
1.000 \\
\times\ 1.110 \\
\hline
0000 \\
1000 \\
1000 \\
1000 \\
\hline
1110000
\end{array}
\quad \Rightarrow \text{ Kết quả } = 1.110000 \times 2^{-3}
$$

Giải:

Bước 3: Kết quả của Bước 2 đã ở dạng chuẩn:

$$\text{Kết quả} = 1.110000 \times 2^{-3}$$

Bước 4: Làm tròn: $\text{Kết quả} = 1.110 \times 2^{-3}$

Bước 5: Do hai thừa số trái dấu nhau, nên Tích sẽ có dấu âm:

$$\text{Tích} = -1.110 \times 2^{-3}$$

Kiểm tra kết quả:

$-1.110 \times 2^{-3} = -0.001110b = -0.00111b$

$= (-7/2^5)_{10} = -0.21875$

⇒ Phù hợp với kết quả phép nhân:

$0.5 \times (-0.4375) = -0.21875$

Giải thuật và phần cứng phép nhân số thực:

(Sinh viên tự nghiên cứu trong sách *Computer Organization and Design – trang 205*)

Hết Phần 3

KIẾN TRÚC MÁY TÍNH

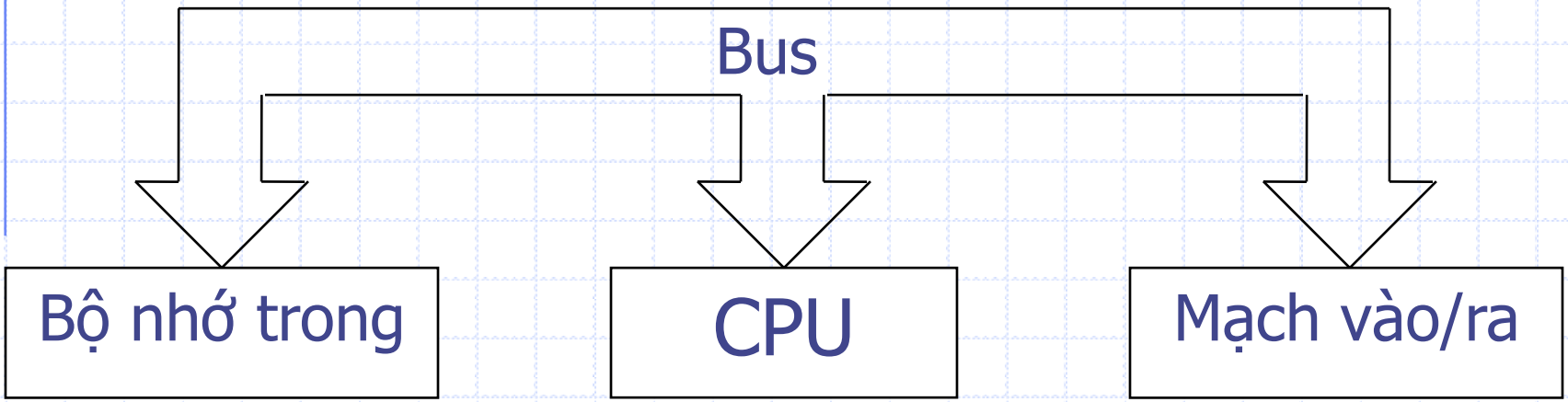
Giảng viên: Ths Phạm Thanh Bình
Bộ môn Kỹ thuật máy tính & mạng
http://vn.myblog.yahoo.com/CNTT-wru
http://ktmt.wru.googlepages.com

Chương 4:

BỘ VI XỬ LÝ- CPU

- Sơ đồ hệ thống xử lý
- Kiến trúc bộ vi xử lý
- Quá trình thi hành lệnh
- Các chân tín hiệu cơ bản của CPU
- Họ VXL Intel 8x86

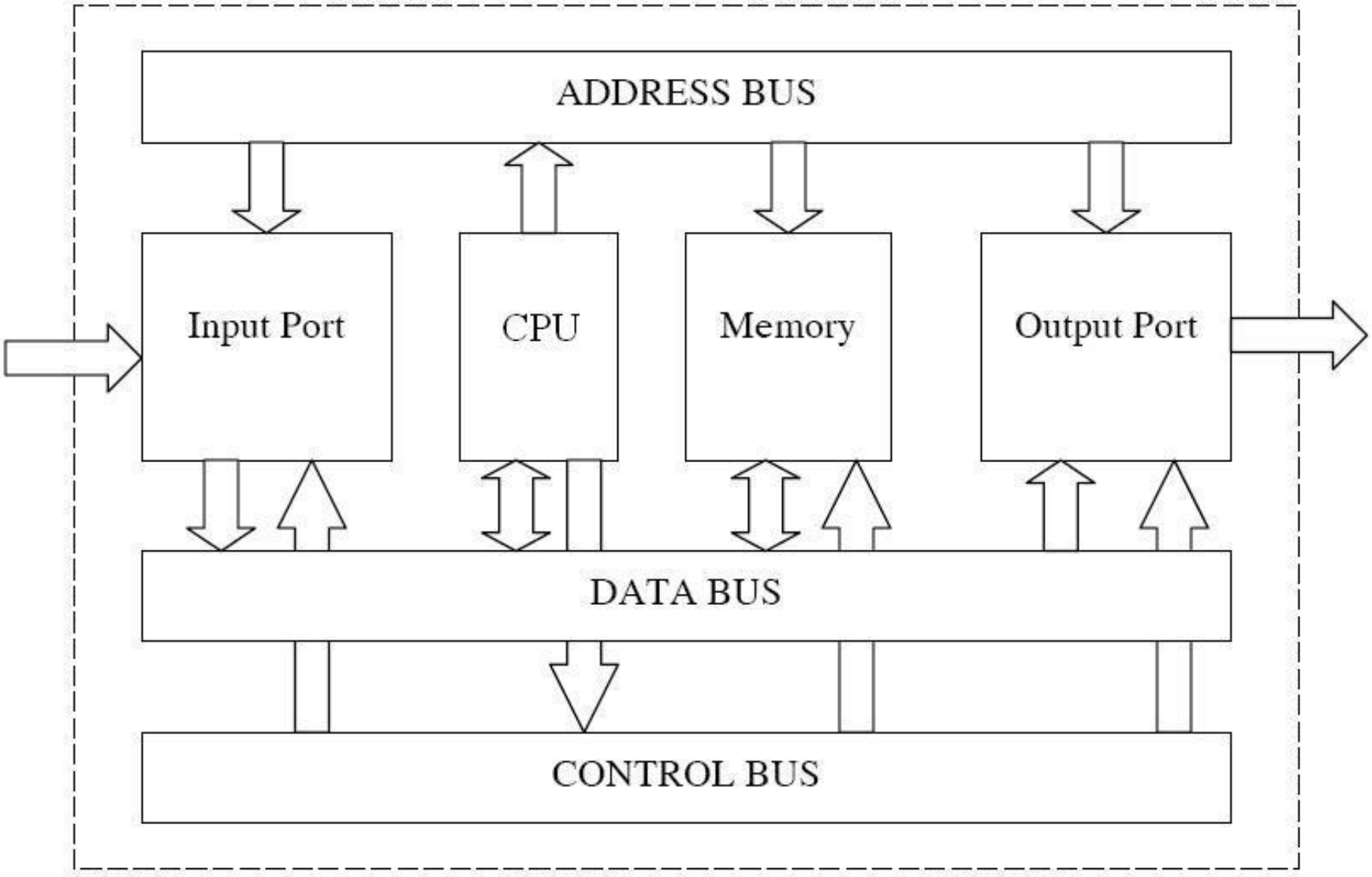
Sơ đồ hệ thống xử lý:



Các thành phần của khối xử lý

- Bộ vi xử lý – CPU (Central Processing Unit): Là bộ não của máy tính, nó xử lý các thông tin và điều khiển mọi hoạt động của máy tính.
- Bộ nhớ trong: Là bộ nhớ có khả năng liên lạc trực tiếp với bộ vi xử lý,là nơi lưu trữ dữ liệu phục vụ cho quá trình xử lý.
- Các mạch vào ra: Để điều khiển việc giao tiếp với thiết bị ngoại vi.
- Hệ thống Bus: Kết nối các bộ phận trên lại với nhau.

Chi tiết:


ADDRESS BUS
Input Port
CPU
Memory
Output Port
DATA BUS
CONTROL BUS

BUS:

- BUS: Là hệ thống dây dẫn và cáp nối để liên lạc giữa CPU với bộ nhớ và các vi mạch vào ra.
- Có ba loại BUS: Bus địa chỉ (Address bus), Bus dữ liệu (Data bus), Bus điều khiển (Control bus).

- Bus Địa chỉ: Dùng để truyền địa chỉ của ô nhớ mà CPU cần liên lạc.
- Bus Dữ liệu: Dùng để truyền dữ liệu.
- Bus Điều khiển: Dùng để truyền các tín hiệu điều khiển trong quá trình liên lạc (Ví dụ: tín hiệu xác định quá trình truy nhập bộ nhớ là Đọc hay Ghi)

Các mạch vào ra

- CPU không thể liên lạc trực tiếp với các thiết bị ngoại vi mà phải thông qua các vi mạch vào/ra.
- Mỗi vi mạch này chứa một vài thanh ghi gọi là cổng vào/ra (Input/Output Port).
- Tương tự như bộ nhớ, các cổng vào/ra cũng được đánh địa chỉ.

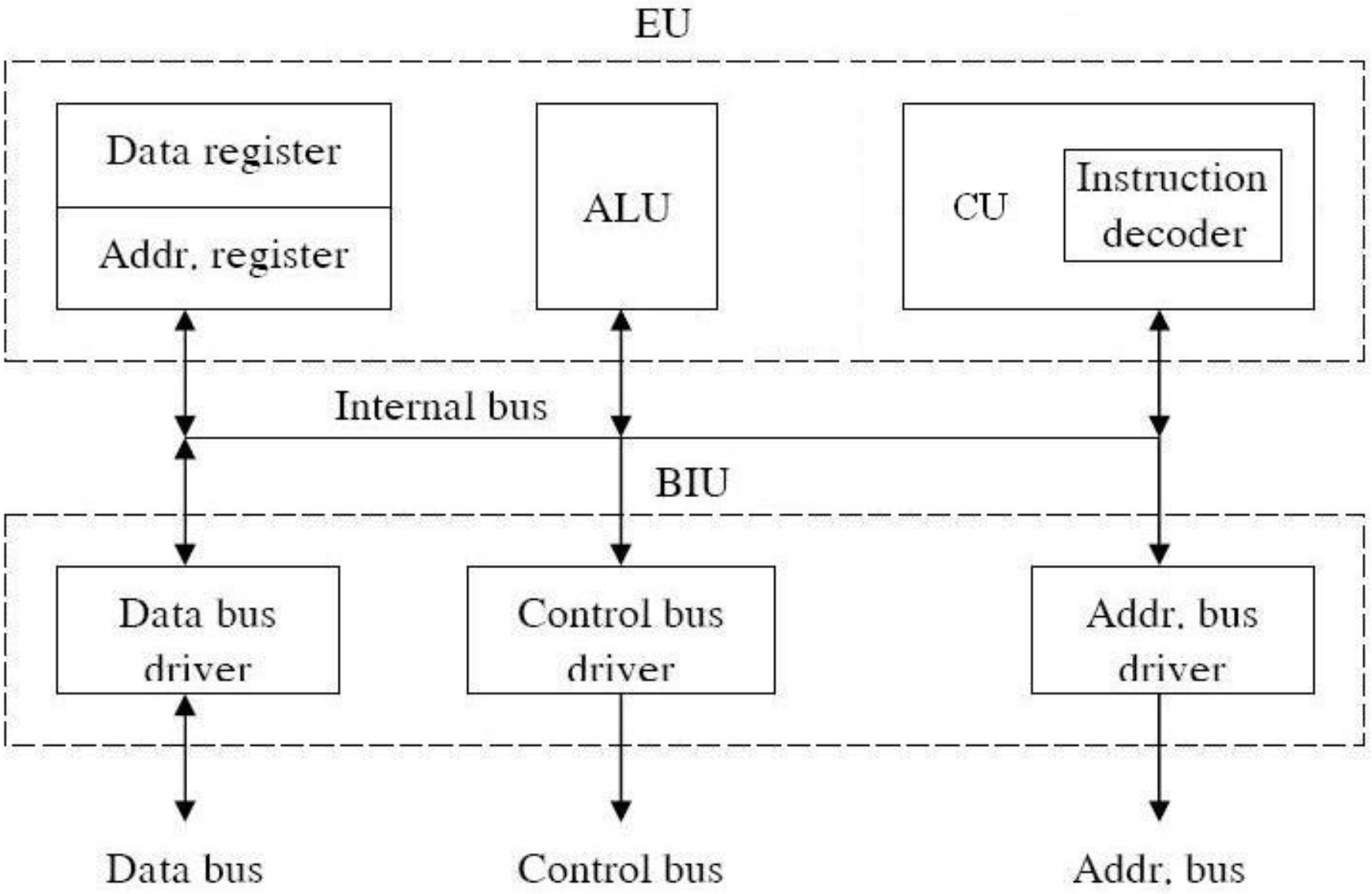
Phân loại vi xử lý

- ***Multi chip***: dùng 2 hay nhiều chip LSI (Large Scale Intergration: tích hợp từ 1000 ÷ 10000 transistor) cho ALU và control.
- ***Microprocessor***: dùng 1 chip LSI/VLSI (Very Large Scale Intergration: tích hợp ÷ 10000 transistor) cho ALU và control.
- ***Single chip microprocessor*** (còn gọi là microcomputer / microcontroller): là 1 chip LSI/VLSI chứa toàn bộ các khối như ở slide trước.


Intel® 8088 Chip

Kiến trúc bộ vi xử lý

- Xem sơ đồ:


EU
Data register
Addr. register
ALU
CU
Instruction decoder
Internal bus
BIU
Data bus driver
Control bus driver
Addr. bus driver
Data bus
Control bus
Addr. bus

Bộ vi xử lý có 2 khối chức năng:

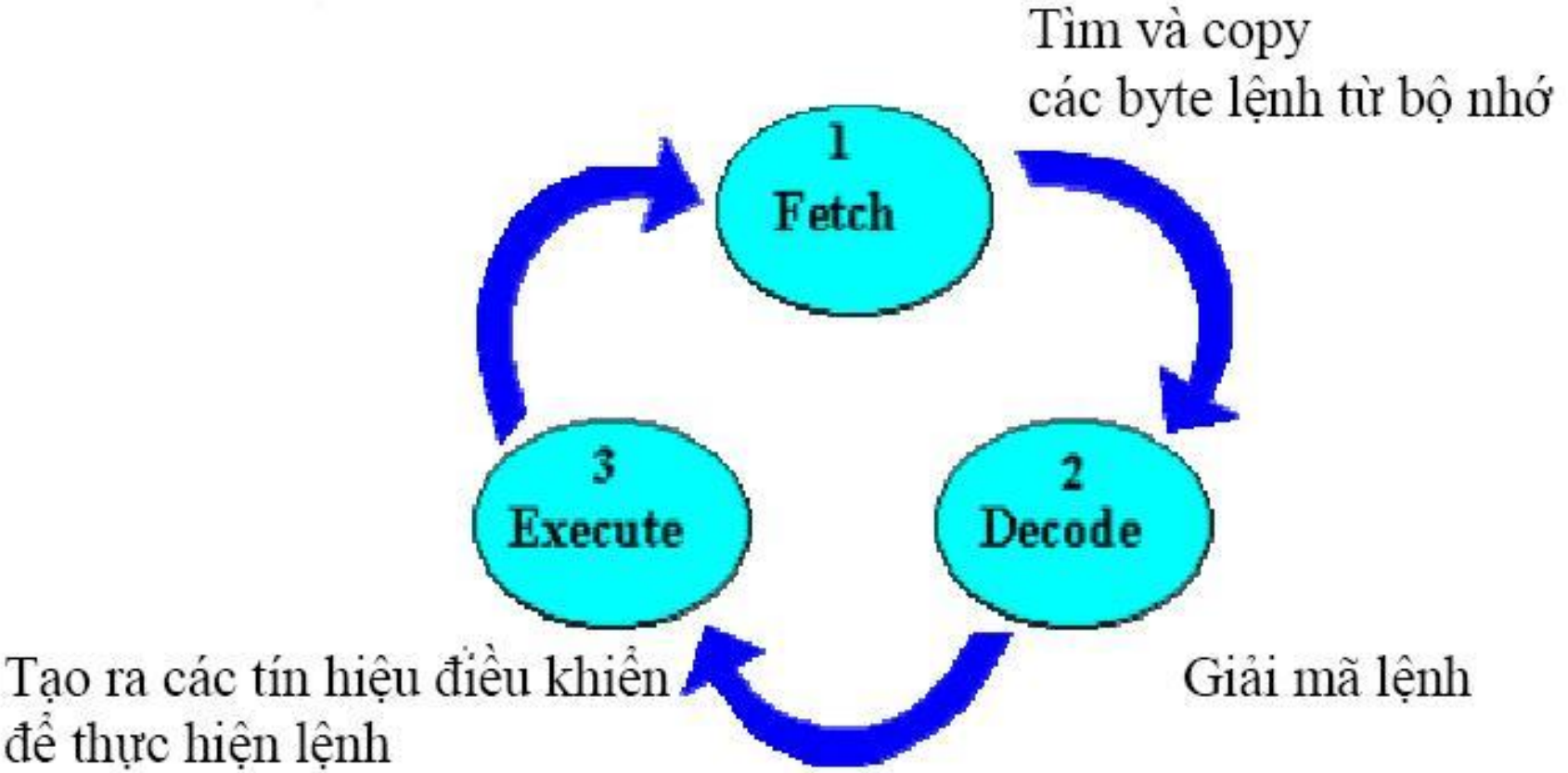
- Khối thực thi (EU - Execution unit): Đảm nhiệm việc thực hiện lệnh.
- Khối giao tiếp bus (BIU – Bus interface unit): Liên lạc giữa EU với Bus ngoài.

Khối thực thi (EU)

- ALU: Thực hiện các lệnh số học và logic.
- Các toán hạng được chứa trong các thanh ghi dữ liệu (data register) hay thanh ghi địa chỉ (address register), hay từ bus nội (internal bus).
- CU (Control Unit): Khối điều khiển hoạt động của EU. Trong khối này có mạch giải mã lệnh (Instruction decoder).

Quá trình thi hành lệnh:

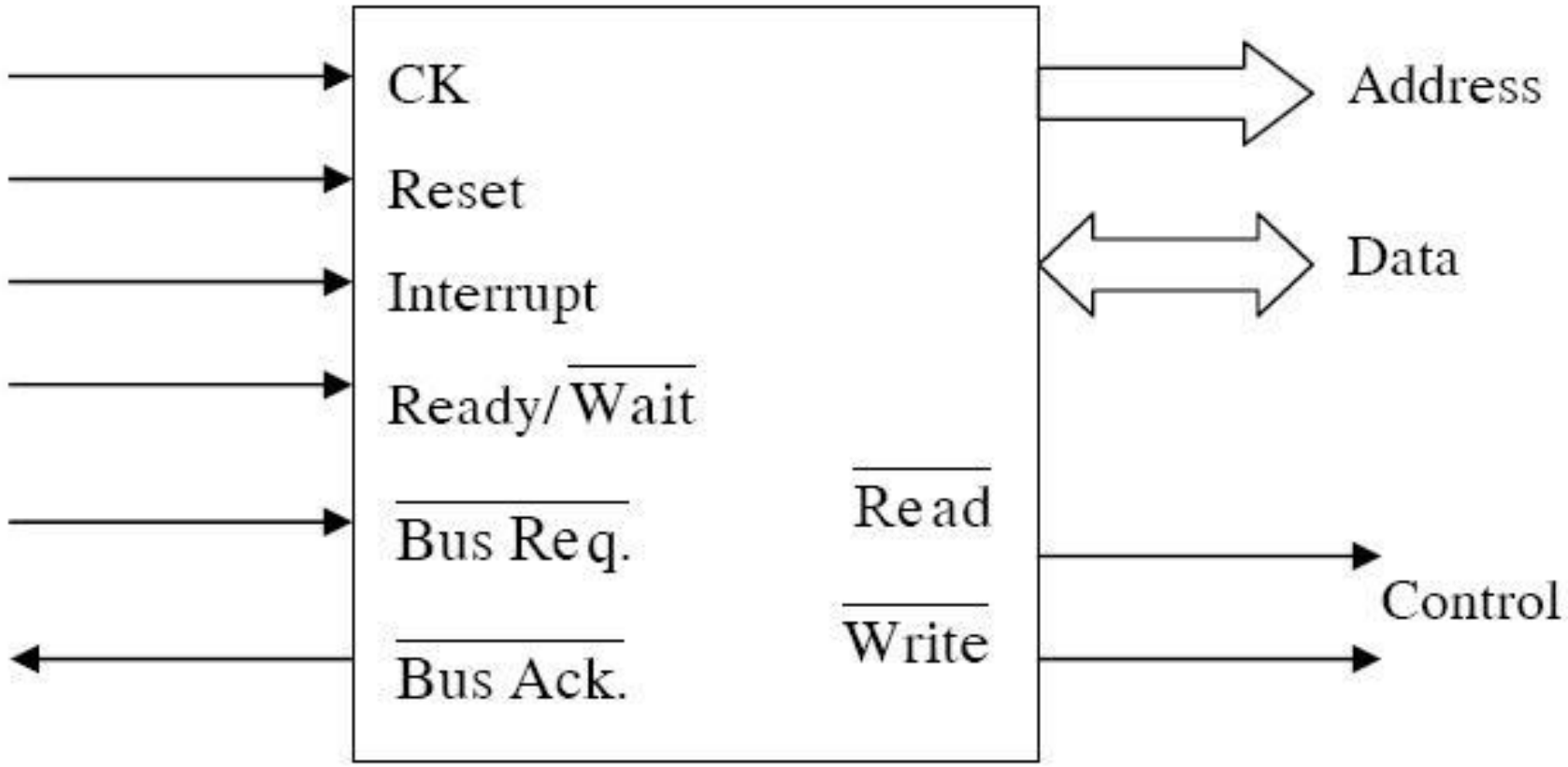
Lấy lệnh - Giải mã – Thực hiện



Quá trình thi hành lệnh:

- Khi chương trình bắt đầu, Thanh ghi con trỏ lệnh sẽ ở địa chỉ đầu chương trình. Địa chỉ này được chuyển qua bộ nhớ thông qua Address bus.
- Khi tín hiệu Read đưa vào Control bus, nội dung bộ nhớ liên quan sẽ đưa vào bộ giải mã lệnh.
- Bộ giải mã lệnh sẽ khởi động các phép toán cần thiết để thực thi lệnh. Quá trình này đòi hỏi một số chu kỳ máy (machine cycle) tuỳ theo lệnh.
- Sau khi lệnh đã thực thi, bộ giải mã lệnh sẽ đặt Con trỏ lệnh đến địa chỉ của lệnh kế.

Các chân tín hiệu cơ bản của CPU



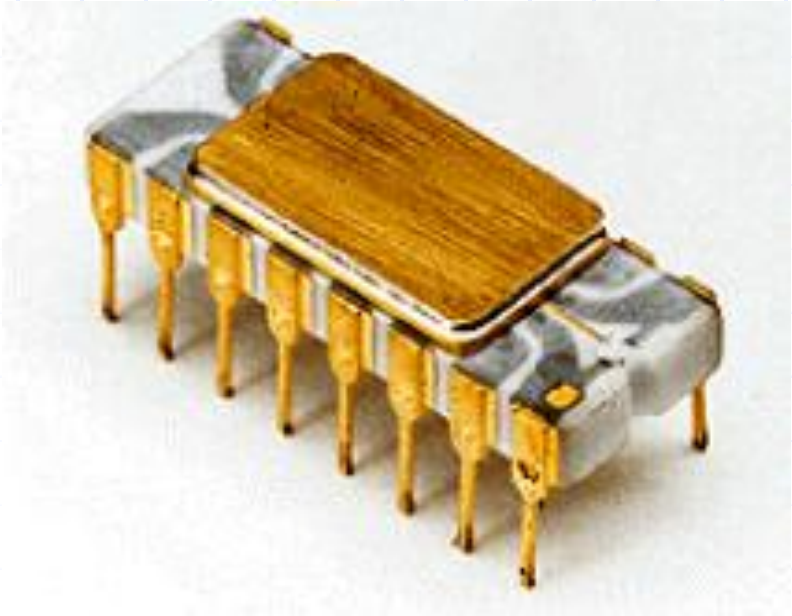
- CK: Chân nhận tín hiệu xung nhịp
- Reset: Chân khởi động lại
- Ready/Wait: Xác định trạng thái sẵn sàng phục vụ hay phải chờ.
- Bus Req: Tín hiệu yêu cầu được sử dụng Bus
- Bus Ack: Xác nhận yêu cầu về Bus

- Read: Điều khiển đọc
- Write: Điều khiển ghi
- Address: Các chân xác định địa chỉ khi cần giao tiếp với bộ nhớ
- Data: Các chân dành cho dữ liệu.

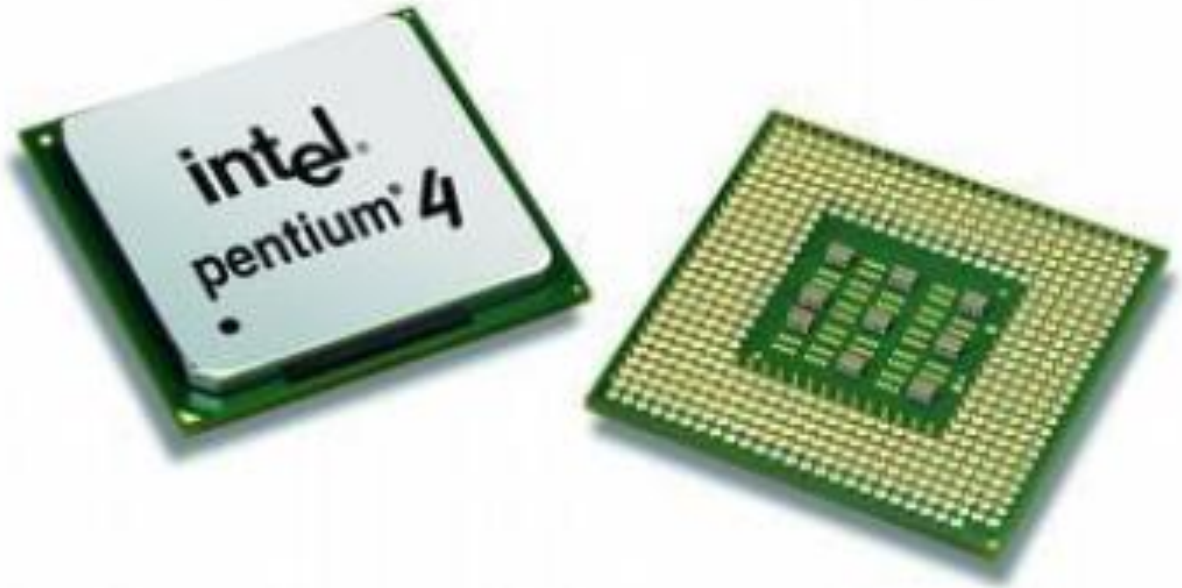
...

Họ vi xử lý Intel 8x86

- 4004 là bộ VXL đầu tiên của Intel, ra đời năm 1971. Nó chứa 2300 transitor:



- Bộ VXL Pentium 4 hiện nay chứa 55 triệu transistor:



Họ vi xử lý x86 của Intel

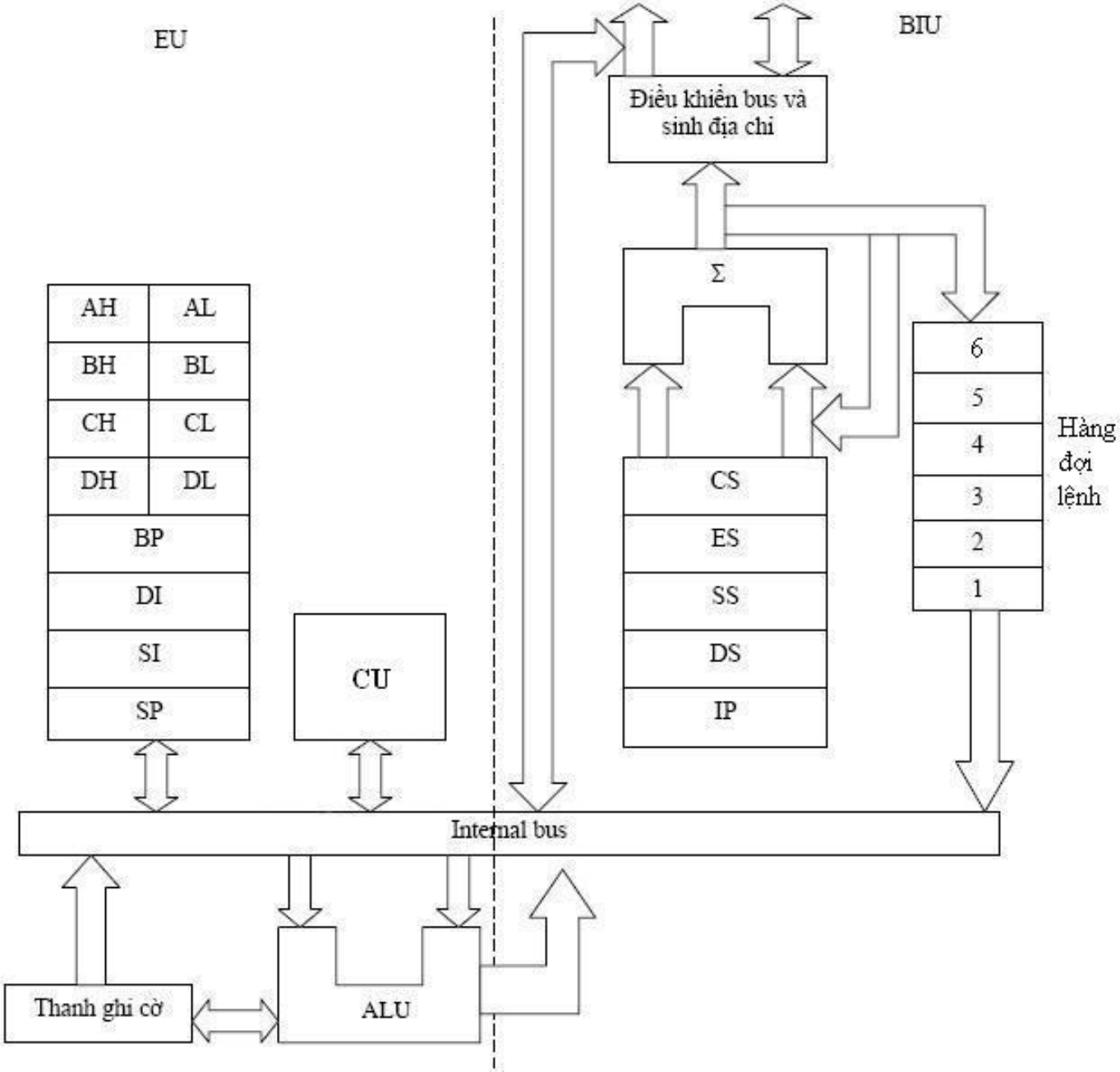
Model	Năm sản xuất	Số lượng Transistor
4004	1971	2,300
8008	1972	2,500
8080	1974	5,000
8086	1978	29,000
80286	1982	120,000
80386™ processor	1985	275,000
80486™ DX processor	1989	1,180,000
Pentium® processor	1993	3,100,000
Pentium II processor	1997	7,500,000
Pentium III processor	1999	24,000,000
Pentium 4 processor	2000	55,000,000

	Tốc độ	Bus	Số transistor	Dung lượng bộ nhớ tối đa	Bộ nhớ ảo
4004	108 KHz	4 bits	2,300 (10 microns)	640 bytes	
8008	108 KHz	8 bits	3,500	16 KBytes	
8080	2 MHz	8 bits	6,000 (6 microns)	64 KBytes	
8086	5 MHz 8 MHz 10 MHz	16 bits	29,000 (3 microns)	1 Megabyte	
8088	5 MHz 8 MHz	8 bits	29,000 (3 microns)		
80286	8 MHz 10 MHz 12 MHz	16 bits	134,000 (1.5 microns)	16 Megabytes	1 gigabyte
Intel386(TM)DX Microprocessor	16 MHz 20 MHz 25 MHz 33 MHz	32 bits	275,000 (1 micron)	4 gigabytes	64 terabytes
Intel386(TM)SX Microprocessor	16 MHz 20 MHz	16 bits	275,000 (1 micron)	4 gigabytes	64 terabytes
Intel486(TM)DX Microprocessor	25 MHz 33 MHz 50 MHz	32 bits	1,200,000 (1 micron, .8 micron with 50 MHz)	4 gigabytes	64 terabytes

	Tốc độ	Bus	Số transistor	Dung lượng bộ nhớ tối đa	Bộ nhớ ảo
Intel486(TM)SX Microprocessor	16 MHz 20 MHz 25 MHz 33 MHz	32 bits	1,185,000 (.8 micron)	4 gigabytes	64 terabytes
Pentium® Processor	60MHz 66MHz 75MHz 90MHz 100MHz 120MHz 133MHz 150MHz 166MHz	32 bits	3.1 million (.8 micron)	4 gigabytes	64 terabytes
Pentium® Pro Processor	150MHz 180MHz 200MHz	32 bits	5.5 million (.32 micron)	4 gigabytes	64 terabytes

Kiến trúc của Intel 8086

- Sơ đồ khối
- Quá trình thi hành lệnh
- Các chân của VXL 8086
- Lập trình ngắt


EU
BIU
Điều khiển bus và sinh địa chỉ
Σ
AH
AL
BH
BL
CH
CL
DH
DL
BP
DI
SI
SP
CU
CS
ES
SS
DS
IP
6
5
4
3
2
1
Hàng đợi lệnh
Internal bus
Thanh ghi cờ
ALU

- 8086 có 2 thành phần: BIU (Khối giao tiếp Bus) và EU (Khối thực thi).
- BIU cung cấp các chức năng phần cứng, bao gồm tạo các địa chỉ bộ nhớ và I/O để chuyển dữ liệu giữa EU với bên ngoài VXL.
- EU nhận các mã lệnh chương trình và dữ liệu từ BIU, thực thi các lệnh này và chứa các kết quả trong các thanh ghi.
- Chú ý rằng EU không có bus hệ thống nên phải thực hiện nhận và xuất tất cả các dữ liệu của nó thông qua BIU.

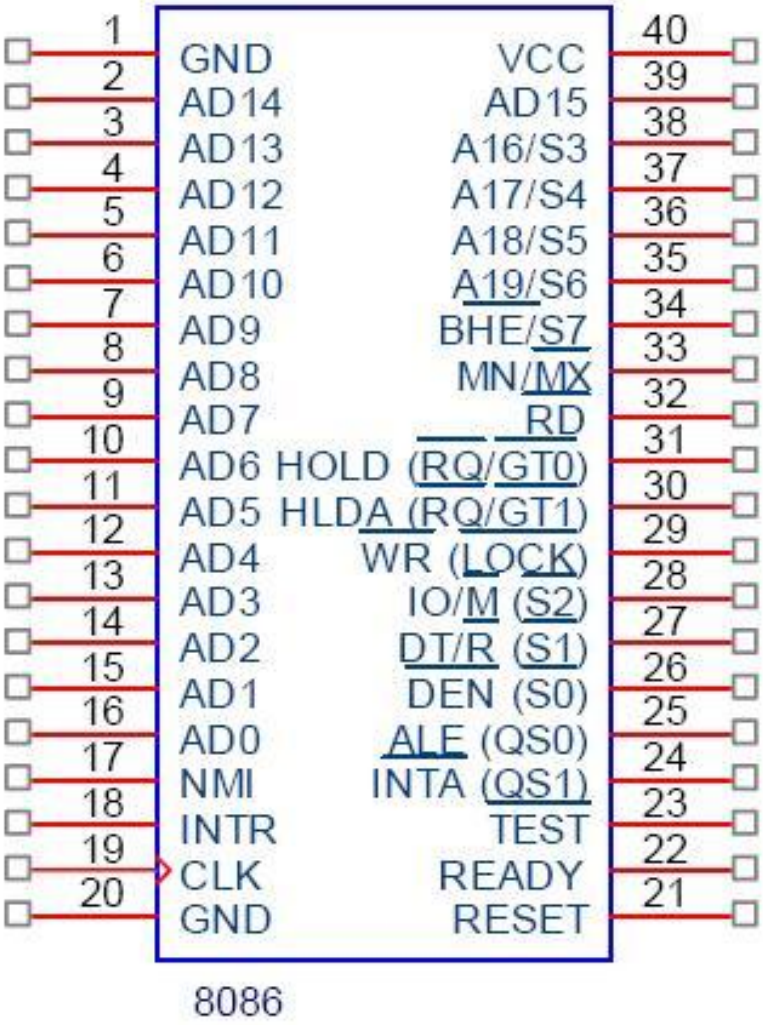
Quá trình thi hành lệnh:

- BIU nhận các mã lệnh từ bộ nhớ và đặt chúng vào hàng chờ lệnh.
- EU (Execute Unit – đơn vị thực thi) sẽ giải mã và thực hiện các lệnh trong hàng.
- Chú ý rằng các đơn vị EU và BIU làm việc độc lập với nhau nên BIU có khả năng nhận một lệnh mới trong khi EU đang thực thi lệnh trước đó.
- Khi EU thực hiện xong lệnh, nó sẽ lấy mã lệnh kế tiếp trong hàng đợi lệnh.

So sánh 8086 và 8088:

- Sự khác biệt giữa VXL 8086 và 8088 là ở BIU. 8088 sử dụng bus dữ liệu rộng 8 bit, còn của 8086 là 16 bit. Ngoài ra hàng đợi lệnh của 8088 dài 4 byte trong khi của 8086 là 6 byte.
- Tuy nhiên do EU của hai bộ VXL này giống nhau nên các chương trình viết cho 8086 có thể chạy được trên 8088 mà không cần thay đổi gì cả.

Các chân của VXL 8086



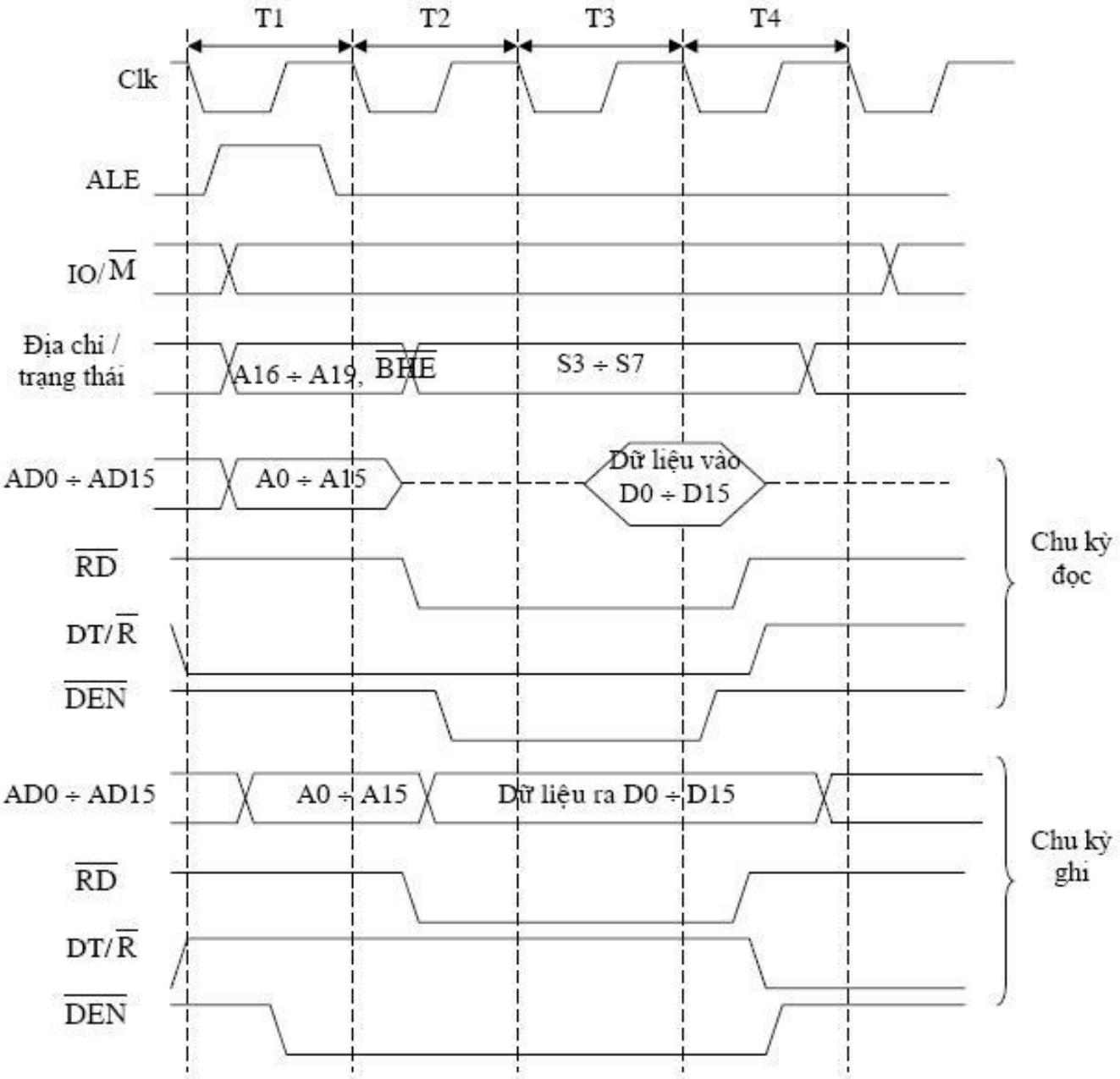
8086

Giải thích:

- 8086 có bus địa chỉ 20 bit, bus dữ liệu 16 bit, 3 chân nguồn và 17 chân dùng cho các chức năng điều khiển.
- Tuy nhiên một chân có thể đảm nhiệm nhiều chức năng ta (nhờ kỹ thuật phân kênh theo thời gian - time multiplexing).

Cụ thể:

- 16 chân (AD0 ÷ AD15): vừa là chân dữ liệu, vừa là chân địa chỉ. Các chân này sẽ là các đường địa chỉ trong trạng thái T1 và dữ liệu trong các trạng thái T2 – T4.
- 4 chân địa chỉ và trạng thái (A16/S3 ÷ A19/S6)
- 3 chân nguồn (VCC, GND1, GND20)
- 17 chân định thời và điều khiển


T1
T2
T3
T4
Clk
ALE
IO/M
Địa chỉ / trạng thái
A16 ÷ A19, BHE
S3 ÷ S7
AD0 ÷ AD15
A0 ÷ A15
Dữ liệu vào D0 ÷ D15
RD
DT/R
DEN
Chu kỳ đọc
AD0 ÷ AD15
A0 ÷ A15
Dữ liệu ra D0 ÷ D15
RD
DT/R
DEN
Chu kỳ ghi

Chức năng của các chân

- 8086 có thể hoạt động ở 2 chế độ: tối thiểu (minimum mode) và tối đa (maximum mode).
- Chế độ tối thiểu chỉ dùng cho các hệ thống VXL đơn giản.
- Chế độ tối đa dùng cho các hệ thống phức tạp hơn, giao tiếp với các bộ nhớ và I/O riêng.

Các chân dùng chung cho cả hai chế độ tối đa và tối thiểu:

Chân	Chức năng	Loại
AD15 ÷ AD0	Bus dữ liệu / địa chỉ	2 chiều, 3 trạng thái
A19/S6 ÷ A16/S3	Địa chỉ / trạng thái	Ngõ ra 3 trạng thái
$\overline{MX}$	Điều khiển chế độ	Ngõ vào
$\overline{RD}$	Điều khiển đọc	Ngõ ra 3 trạng thái
$\overline{TEST}$	Chờ kiểm tra điều khiển	Ngõ vào
READY	Chờ trạng thái điều khiển	Ngõ vào
RESET	Reset hệ thống	Ngõ vào
NMI	Yêu cầu ngắt không thể che	Ngõ vào
INTR	Yêu cầu ngắt	Ngõ vào
CLK	Xung nhịp hệ thống	Ngõ vào
VCC	+5V	Ngõ vào
GND	GND	Ngõ vào

Các chân chỉ dùng cho chế độ tối thiểu:

Chân	Chức năng	Loại
HOLD	Yêu cầu giữ	Ngõ vào
HLDA	Ghi nhận giữ	Ngõ vào
$\overline{WR}$	Điều khiển ghi	Ngõ ra 3 trạng thái
IO/$\overline{M}$	Điều khiển I/O và bộ nhớ	Ngõ ra 3 trạng thái
DT/$\overline{R}$	Truyền / nhận dữ liệu	Ngõ ra 3 trạng thái
$\overline{DEN}$	Cho phép dữ liệu	Ngõ ra 3 trạng thái
$\overline{BHE}$/S7	Đường trạng thái	Ngõ ra 3 trạng thái
ALE	Cho phép chốt địa chỉ	Ngõ ra
$\overline{INTA}$	Ghi nhận ngắt	Ngõ ra

Các chân chỉ dùng cho chế độ tối đa:

Chân	Chức năng	Loại
$\overline{RQ}/\overline{GT1,0}$	Yêu cầu / cấp bus	2 chiều
$\overline{LOCK}$	Điều khiển khóa ưu tiên bus	Ngõ ra 3 trạng thái
$\overline{S2} \div \overline{S0}$	Trạng thái chu kỳ bus	Ngõ ra 3 trạng thái
QS1, QS2	Trạng thái hàng lệnh	Ngõ ra

Các chân Nguồn:

- 8086 sử dụng nguồn cấp điện +5V (VCC) và có 2 chân đất (GND).
- Dòng điện cực đại là 340 mA (10 mA cho loại CMOS)

Xung nhịp:

- 8086 sử dụng xung nhịp dạng xung chữ nhật có chu kỳ với thời gian cạnh lên và xuống nhỏ hơn 10 ns (đi vào chân CLK).

Các chân trạng thái bus (S0, S1, S2):

Ngõ vào trạng thái			Chu kỳ CPU
$\overline{S2}$	$\overline{S1}$	$\overline{S0}$	
0	0	0	Ghi nhận ngắt
0	0	1	Đọc I/O port
0	1	0	Ghi I/O port
0	1	1	Ngừng
1	0	0	Nhận lệnh
1	0	1	Đọc bộ nhớ
1	1	0	Ghi bộ nhớ
1	1	1	Thụ động

Các chân điều khiển bus (HOLD, HLDA, RQ/GT0 , RQ/GT1, LOCK):

Chế độ tối thiểu:

- HOLD (giữ): Ngõ vào tác động mức cao làm cho VXL hở mạch tất cả các bus của nó, tách VXL khỏi bộ nhớ và I/O để cho phép thiết bị khác xử lý bus hệ thống. Quá trình này gọi là truy xuất bộ nhớ trực tiếp (DMA – Direct Memory Access).
- HLDA (Hold acknowledge): Ghi nhận yêu cầu DMA đối với bộ điều khiển DMA.

Chế độ tối đa:

- RQ/GT0, RQ/GT1 (Request / Grant): Các chân này dùng cả hai chức năng vào (nhận yêu cầu) và ra (chấp nhận yêu cầu).
  + Khi một thiết bị muốn lấy quyền điều khiển bus cục bộ, nó sẽ phát yêu cầu bằng cách đưa tín hiệu mức thấp vào chân yêu cầu.
  + Sau khi nhận yêu cầu, 8086 sẽ ở trạng thái HOLD và gửi tín hiệu chấp nhận ra chân này. Ở đây, chân RQ/GT0 có độ ưu tiên cao hơn chân RQ/GT1.
- LOCK: báo cho các thiết bị khác biết không thể sử dụng bus cục bộ.

Chân RESET:

- Hoạt động khi có xung tác động mức cao, dùng để khởi động lại hệ thống.
- Chân RESET thường được sử dụng khi hệ thống có sự cố.

Các chân điều khiển bus (READY, IO/M, RD, WR, DEN, DT/R...):

- READY: Chân này ở mức thấp sẽ ứng với trạng thái không sẵn sàng, còn mức cao thì ứng với trạng thái sẵn sàng.
- Chân IO/M (IO/Memory – Xuất nhập /Bộ nhớ): xác định chu kỳ bus hiện hành đang làm việc với bộ nhớ (mức thấp) hay I/O (mức cao).

- Chân RD (Read): Xác định chiều truyền dữ liệu từ bộ nhớ hay I/O đến VXL.
- Chân WR (Write): tín hiệu này ngược với RD, nó xác định chiều truyền dữ liệu từ VXL đến I/O hay bộ nhớ.

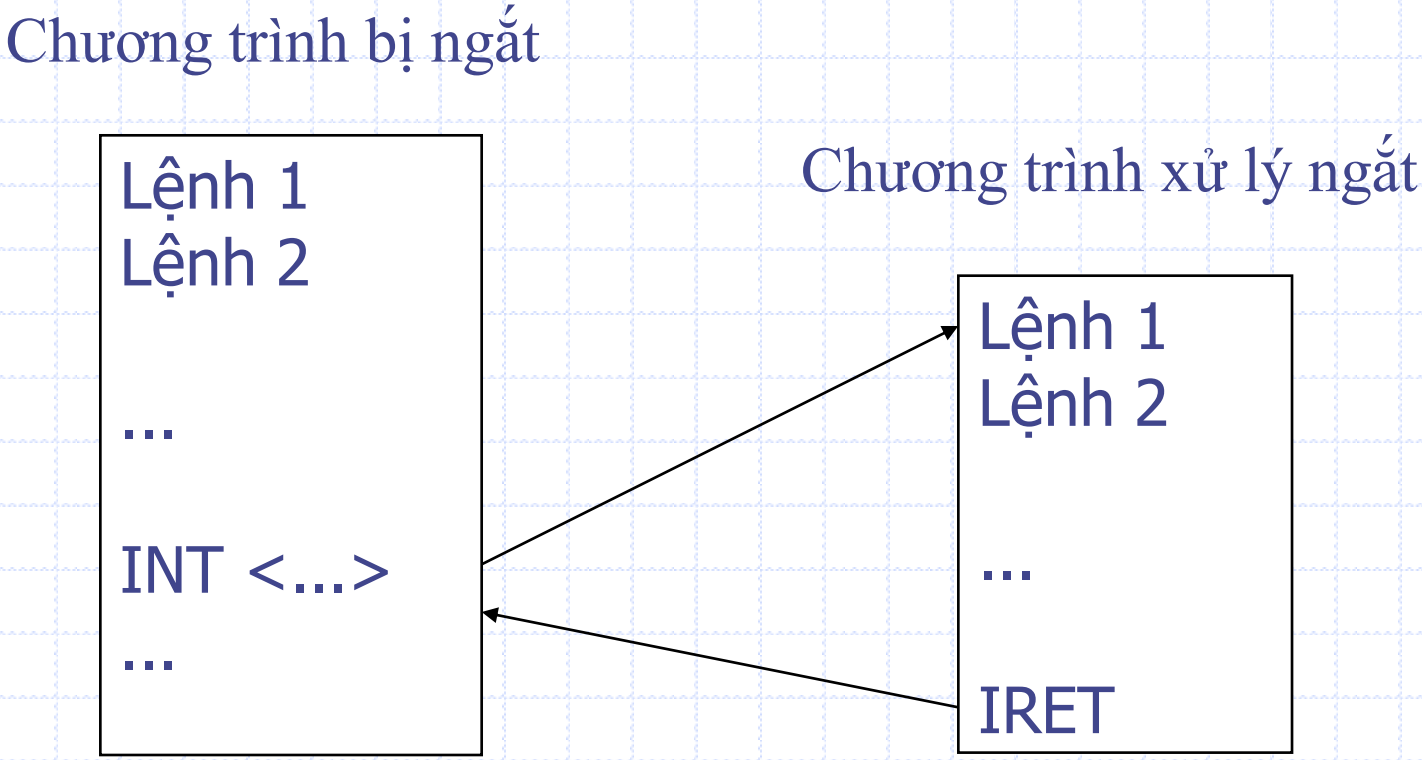
- DEN (Data Enable – cho phép dữ liệu): Chân này được dùng với DT/R để cho phép nối các bộ đệm hai chiều vào data bus. Nó ngăn ngừa sự tranh chấp bus bằng cách cấm các bộ đệm dữ liệu cho đến trạng thái T2, khi các đường dữ liệu / địa chỉ không còn lưu trữ địa chỉ của bộ nhớ hay I/O nữa.
- Chân DT/R (Data transmit/receive – truyền/nhận dữ liệu): dùng để điều khiển chiều của luồng dữ liệu qua các bộ đệm (nếu có) vào bus dữ liệu của hệ thống. Khi ở mức thấp, nó chỉ thực hiện tác vụ đọc và khi ở mức cao nó chỉ thực hiện tác vụ ghi.

...

Lập trình ngắt

- *Ngắt là hành động dừng chương trình đang chạy để thực hiện một chương trình khác* (chương trình này được gọi là *chương trình xử lý ngắt*).
- Bộ VXL sẽ dừng các công việc đang thực hiện khi chân INTR hay chân NMI của nó nhận được tín hiệu yêu cầu ngắt. Sau đó nó trao quyền điều khiển lại cho chương trình xử lý ngắt.
- Tín hiệu yêu cầu ngắt có thể do một thiết bị phần cứng hoặc do một lệnh INT trong chương trình sinh ra.

- Quá trình ngắt được mô tả trong hình dưới đây:



- Chương trình xử lý ngắt cần được kết thúc bằng lệnh IRET để sau khi thực hiện xong có thể quay trở về thực hiện tiếp chương trình bị ngắt trước đó.
- Có nhiều loại tín hiệu ngắt khác nhau, để phân biệt các ngắt cần dựa vào số hiệu của chúng. Bộ vi xử lý 8086 có thể quản lý 256 ngắt, được đánh số lần lượt từ 0, 1, 2,..., FFh.
- Dưới đây là bảng danh sách các ngắt:

Số hiệu ngắt	Chức năng
0 – 1Fh	Ngắt của BIOS
20h – 3Fh	Ngắt của DOS
40h – 7Fh	Dự trữ
80h – F0h	Dùng cho chương trình BASIC trong ROM
F1h – FFh	Không sử dụng

Giải thích:

- Chương trình xử lý ngắt có thể là một bộ phận của BIOS hay của DOS, cũng có thể do người sử dụng tự viết.
- Ta cần phân biệt rõ hai khái niệm: "Ngắt" và "Chương trình xử lý ngắt". Không phải số hiệu ngắt nào cũng có có chương trình xử lý ngắt tương ứng.
- Khi một ngắt có số hiệu từ 0 – 1Fh xuất hiện thì chúng sẽ được xử lý bởi các chương trình viết sẵn nằm trong ROM BIOS (chủ yếu là giải quyết các yêu cầu vào/ ra cơ bản).
- Còn nếu ngắt có số hiệu từ 20h – 3Fh thì sẽ do hệ điều hành DOS xử lý.

Phân loại ngắt:

Để phân loại cần dựa trên một tiêu chí nào đó, ở đây ta sẽ phân loại ngắt dựa trên cách thức phát sinh ngắt, tạm chia làm hai loại sau:

- Ngắt mềm
- Ngắt cứng

Ngắt mềm

- Ta gọi một ngắt là ngắt mềm nếu nó được phát sinh khi có lời gọi ngắt bằng lệnh INT trong chương trình.
- Cú pháp của lệnh INT là:

  INT <Số hiệu ngắt>
- Ví dụ:

```
INT   21h        ;Gọi ngắt 21h của DOS
INT   13h        ;Gọi ngắt 13h của BIOS
```

Ngắt cứng

- Một ngắt cứng phát sinh khi có một thiết bị phần cứng gửi tín hiệu yêu cầu ngắt tới bộ vi xử lý.
- Ví dụ:

  Khi ta gõ một phím trên bàn phím hay bấm chuột, sẽ có tín hiệu ngắt gửi tới bộ vi xử lý để yêu cầu xử lý hành động vừa thực hiện.
- Các ngắt được kích hoạt từ thiết bị ngoài (bàn phím, chuột...) giống như ví dụ trên được gọi là *Ngắt cứng ngoài*. Còn nếu ngắt phát sinh bởi các kinh kiện bổ trợ nằm trên mainboard thì được gọi là *Ngắt cứng trong* (hay *ngắt cứng nội bộ*).

Ngắt 17h – Vào/ra máy in:

Đây là ngắt do BIOS quản lý, nó có ba chức năng:

- *Chức năng số 0: Đưa một kí tự ra máy in*
- *Chức năng số 1: Khởi tạo cổng máy in*
- *Chức năng số 2: Kiểm tra trạng thái máy in*

Chức năng số 0: Đưa một kí tự ra máy in

Các tham số: AH = 0
AL = Mã ASCII của kí tự cần in
DX = Số hiệu máy in

- Ví dụ:

  Gửi kí tự ‘A’ ra máy in.

Giải:

```
MOV   AH, 0
MOV   AL, ‘A’     ;Kí tự cần in
MOV   DX, 0       ;Máy in số 0
INT   17h
MOV   AH, 0
MOV   AL, 0Ah     ;in tiếp kí tự xuống dòng
INT   17h
```

Ngắt 21h

Đây là ngắt hay dùng nhất của DOS, nó có rất nhiều chức năng. Ở phần trước ta đã sử dụng bốn chức năng của ngắt này (chức năng số 1, 2, 9 và 4Ch). Trong phần này ta sẽ tìm hiểu thêm một số chức năng khác:

- *Chức năng số 39h: Tạo một thư mục trên ổ đĩa*
- *Chức năng số 2Ch: Lấy thời gian từ đồng hồ hệ thống*
- ...

Chức năng số 39h: Tạo một thư mục trên ổ đĩa

Các tham số: AH = 39h

DS:DX = Địa chỉ của chuỗi chứa đường dẫn (chuỗi này phải có tận cùng là 0)

- Ví dụ:

Tạo thư mục ASM trong ổ đĩa C.

```
TITLE  Tao thu muc
.MODEL  SMALL
.STACK  100H
.DATA
    DuongDan   DB   'C:\ASM', 0
.CODE
MAIN   PROC
    MOV   AX, @DATA
    MOV   DS, AX
    MOV   AH, 39h          ;Chức năng số 39h: tạo thư mục
    LEA   DX, DuongDan     ;Lấy địa chỉ offset của chuỗi đặt
                           ;vào DX
    INT   21h              ;Gọi ngắt
    MOV   AH, 4Ch          ;Kết thúc
    INT   21h
MAIN ENDP
END MAIN
```

Chức năng số 2Ch: Lấy thời gian từ đồng hồ hệ thống

Vào: AH = 2Ch

Ra: CH = giờ $(0 \leq CH \leq 23)$

CL = phút $(0 \leq CL \leq 59)$

DH = giây $(0 \leq DH \leq 59)$

DL = % giây $(0 \leq DL \leq 99)$

- Ví dụ:

  Viết chương trình hiện ra màn hình giờ hiện tại của hệ thống.

```
TITLE   Hien thoi gian
.MODEL  SMALL
.STACK  100H
.DATA
    Time_Buf   DB   '00:00:00$'
.CODE
MAIN   PROC
    MOV   AX, @DATA
    MOV   DS, AX

    MOV   AH, 2Ch        ;Chức năng số 2Ch: đọc thời
gian
    INT   21h            ;Gọi ngắt
```

```
;Đổi Giờ (trong CH) từ số thập phân sang mã ASCII rồi
    ;cất vào Time_Buf
    MOV   AL, CH          ;Chuyển Giờ vào AX
    MOV   AH, 0
    MOV   DL, 10          ;Chia AX cho 10
    DIV   DL              ;AL = Thương = Số hàng chục
                          ;AH = Số dư = Số hàng đơn vị
    ADD   AL, 30h         ;Đổi số hàng chục sang mã ASCII
    ADD   AH, 30h         ;Đổi số hàng đơn vị sang mã ASCII
    MOV   Time_Buf, AL    ;Cất vào chuỗi
    MOV   Time_Buf+1, AH
```

```
;Đổi Phút (trong CL) từ số thập phân sang mã ASCII rồi
    ;cất vào Time_Buf
    MOV   AL, CL          ;AX chứa Phút
    MOV   AH, 0
    MOV   DL, 10          ;Chia AX cho 10
    DIV   DL              ;AL chứa số hàng chục của Phút
                          ;AH chứa số hàng đơn vị của Phút
    ADD   AL, 30h         ;Đổi sang mã ASCII
    ADD   AH, 30h
    MOV   Time_Buf+3, AL ;Cất vào chuỗi (sau dấu hai chấm)
    MOV   Time_Buf+4, AH
```

```
;Đổi Giây (trong DH) từ số thập phân sang mã ASCII rồi
    ;cất vào Time_Buf
    MOV   AL, DH          ;AX chứa Giây
    MOV   AH, 0
    MOV   DL, 10          ;Chia AX cho 10
    DIV   DL              ;AL chứa số hàng chục của Giây
                          ;AH chứa số hàng đơn vị của Giây
    OR    AX, 3030h       ;Đổi sang mã ASCII
    MOV   Time_Buf+6, AL
    MOV   Time_Buf+7, AH
```

```
;Hiện chuỗi chứa thời gian (Time_Buf) ra màn hình.
    MOV   AH, 9              ;Chức năng số 9
    LEA   DX, Time_Buf       ;Lấy địa chỉ chuỗi kí tự đặt vào
                             ;thanh ghi DX
    INT   21h                ;Gọi ngắt
    MOV   AH, 4Ch            ;Kết thúc
    INT   21h
MAIN ENDP
END MAIN
```

Hết Phần 4

KIẾN TRÚC MÁY TÍNH

Giảng viên: Ths Phạm Thanh Bình
Bộ môn Kỹ thuật máy tính & mạng
http://vn.myblog.yahoo.com/CNTT-wru
http://ktmt.wru.googlepages.com

Chương 5:

BỘ NHỚ

- ROM (Read Only Memory)
- RAM (Random Access Memory)
- Cache
- CMOS RAM

ROM (Read Only Memory)

- Đặc tính chung của ROM là dữ liệu lưu trữ sẽ không bị mất đi dù cho không còn nguồn cung cấp cho ROM.
- ROM có tốc độ cao, giá thành rẻ.
- Dữ liệu trên ROM thường do nhà sản xuất ghi vào, ta chỉ có thể đọc dữ liệu từ ROM.
- Trên máy tính, ROM thường dùng để chứa chương trình khởi động máy và các chương trình điều khiển vào/ra cấp thấp.

Một số loại ROM:

- PROM (Programmable ROM): ROM lập trình được.
- EPROM (Erasable Programmable ROM): ROM lập trình được bằng xung điện, có thể xoá bằng tia cực tím.
- EEPROM (Electrically Erasable Programmable ROM): ROM có thể lập trình và xoá bằng điện.

Đặc điểm: Tốc độ ghi dữ liệu vào ROM chậm hơn RAM rất nhiều.

RAM (Random Access Memory)

- RAM là bộ nhớ có thể đọc/ghi. Thông tin trong RAM sẽ bị mất đi khi không còn nguồn cung cấp .
- RAM có tốc độ rất nhanh.
- Trên máy tính, RAM được dùng làm bộ nhớ chính.

Một số loại RAM:

- **SRAM (Static RAM – RAM tĩnh):** Mỗi phần tử nhớ là một mạch lật hai trạng thái (Flip-Flop), tính ổn định cao.
- **DRAM (Dynamic RAM – RAM động):** Mỗi phần tử nhớ là một tụ điện rất nhỏ được chế tạo bằng công nghệ MOS. Do hiện tượng rò rỉ điện tích theo thời gian, ta phải thực hiện nạp điện lại. Quá trình này gọi là làm tươi (refreshing) bộ nhớ.
- Thuận lợi của DRAM là một số lượng lớn transistor có thể được đặt trên một chip nhớ nên nó có dung lượng cao hơn và nhanh hơn SRAM.

Cache

- Cache là bộ nhớ trung gian giữa CPU và bộ nhớ chính (RAM). Có thể đọc/ghi vào Cache.
- CPU có thể truy nhập cache mà không cần thông qua bus, nhờ đó mà tốc độ truy nhập sẽ rất nhanh (nhanh hơn truy nhập vào RAM). Cache có giá thành rất đắt.
- Những thông tin hay dùng nhất trên RAM sẽ được đưa vào Cache.
- Cache có thể được đặt bên trong CPU (Internal Cache), hoặc bên ngoài CPU (External Cache).

CMOS RAM

- CMOS là loại bộ nhớ có thể đọc và ghi.
- Thông tin trên CMOS được nuôi bằng Pin, nó tiêu tốn rất ít năng lượng.
- Trên máy tính, CMOS được dùng để lưu trữ thông tin về ngày, giờ, và các cấu hình hệ thống.
- Khi hết Pin, thông tin trên CMOS vẫn có thể tồn tại được nhiều giờ, rồi mới mất hẳn.

Hết Phần 5

KIẾN TRÚC MÁY TÍNH

Giảng viên: Ths Phạm Thanh Bình
Bộ môn Kỹ thuật máy tính & mạng
http://vn.myblog.yahoo.com/CNTT-wru
http://ktmt.wru.googlepages.com

Chương 6:

MỘT SỐ THIẾT BỊ NGOẠI VI

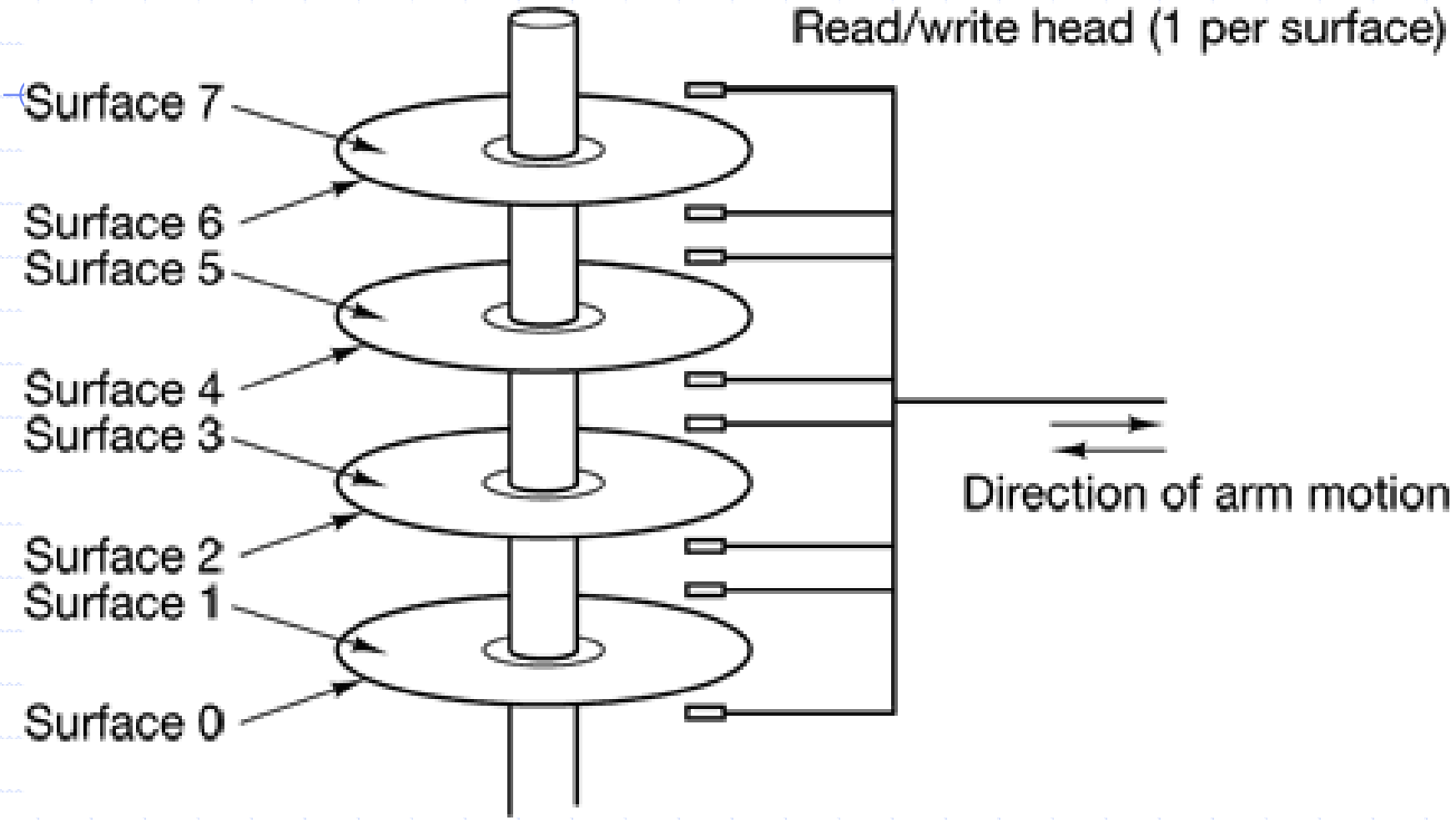
- Đĩa từ
- RAID
- Đĩa CD ROM
- ...

Đĩa từ

- Đĩa từ là thiết bị có dung lượng lưu trữ lớn, tốc độ đọc/ghi nhanh, là thiết bị lưu trữ chính của máy tính.
- Đĩa từ có giá thành rẻ hơn RAM, nhưng tốc độ truy nhập dữ liệu thì chậm hơn, vì nó là một thiết bị cơ khí .


gettyimages

- Một đĩa có thể chứa một hoặc nhiều đĩa kim loại, với tốc độ quay là 5400, 7200, hoặc 10800 vòng/phút.
- Một cánh tay cơ khí được gắn ở góc để đầu đọc (head) có thể chuyển động trên các bề mặt đĩa


Read/write head (1 per surface)
Surface 7
Surface 6
Surface 5
Surface 4
Surface 3
Surface 2
Surface 1
Surface 0
Direction of arm motion

- Mỗi khi cánh tay dịch chuyển, các đầu đọc có thể đọc được dữ liệu ở một vòng tròn mới, goi là **rãnh (track)**
- Tất các các rãnh ứng với cùng một vị trí của cánh tay tạo thành một **trụ (cylinder)**
- Mỗi rãnh được chia thành nhiều **cung từ (sector)**, thường có 512 byte trên mỗi cung từ.

- Thời gian chuyển động của cánh tay giữa hai trụ kế tiếp vào khoảng 1ms. Di chuyển cánh tay tới một trụ bất kỳ mất từ 5 tới 10 ms, tuỳ từng thiết bị.
- Khi cánh tay được đưa tới vị trí của rãnh, thiết bị sẽ phải chờ để cung từ quay tới vị trí đầu đọc, thời gian chờ khoảng 5 đến 10 ms, tuỳ vào tốc độ quay của đĩa.
- Đầu đọc sẽ thực hiện đọc (hoặc ghi) dữ liệu lên cung từ với tốc độ từ 5 MB/s tới 160 MB/s (tuỳ loại đĩa).

RAID

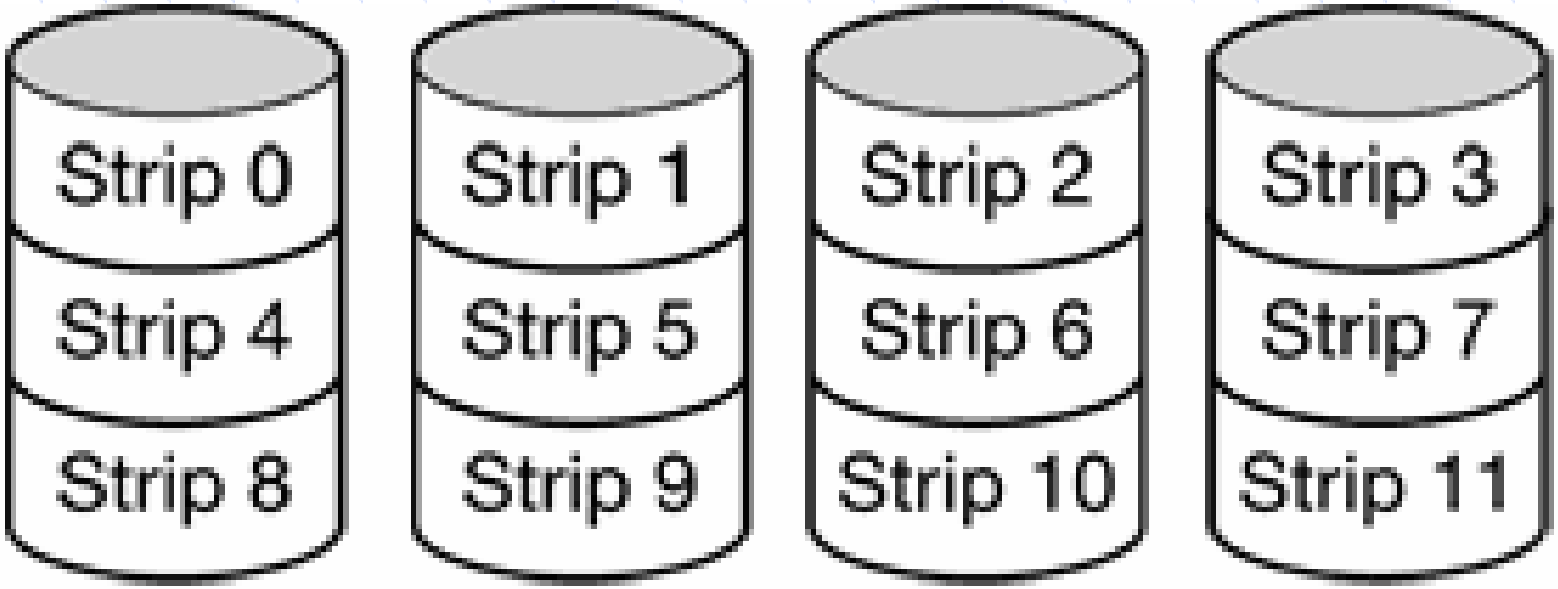
- RAID (Redundant Array of Independent Disks) là một kỹ thuật để tăng hiệu suất và độ tin cậy của đĩa, do Patterson đưa ra năm 1988.
- Ý tưởng cơ bản của RAID là ghép nhiều ổ đĩa riêng thành một hệ thống đĩa, hoạt động giống như một ổ đĩa lớn (gọi là đĩa đơn ảo).

- Patterson đưa ra 5 giải pháp, gọi là RAID mức 0, mức 1,... mức 5.
- Các hệ thống RAID đều có đặc tính phân phối dữ liệu trên các ổ đĩa, nhằm cho phép hoạt động song song. Ví dụ: Một file có thể nằm trải ra trên nhiều ổ đĩa
- Mỗi thao tác đọc/ghi vào đĩa được thực hiện song song trên nhiều đĩa thành phần, nhờ đó tốc độ đọc/ghi sẽ tăng lên rất nhiều.

RAID mức 0

- Một đĩa đơn ảo được chia thành nhiều phần, mỗi phần được gọi là 1 strip.
- Mỗi strip có thể gồm một hoặc nhiều sector.
- RAID mức 0 sắp xếp các strip liên tiếp nhau trên các đĩa khác nhau.

Minh hoạ RAID mức 0 với 4 ổ đĩa:



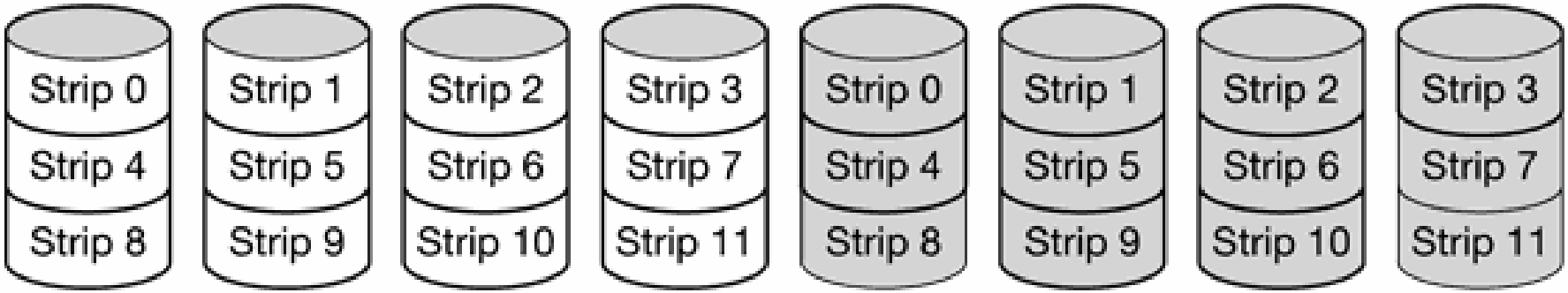
- Nếu phần mềm phát ra một lệnh để đọc một khối dữ liệu gồm 4 strip liên tiếp, bộ điều khiển RAID sẽ chia nhỏ lệnh này thành 4 lệnh riêng rẽ.
- Mỗi lệnh sẽ đọc dữ liệu trên một đĩa, và được thực hiện song song đồng thời. Phần mềm sẽ không biết gì về quá trình vào/ra song song này cả.
- Như vậy tốc độ đọc sẽ được tăng lên 4 lần!

- RAID mức 0 hoạt động tốt nhất khi có các yêu cầu lớn về dữ liệu, càng lớn càng tốt.
- RAID mức 0 hoạt động tệ nhất với các hệ điều hành chỉ đọc ghi dữ liệu theo từng sector. Kết quả vẫn chính xác, nhưng không có sự thực hiện song song, và do đó không cải thiện được hiệu suất.

RAID mức 1

- RAID mức 1 sẽ tăng gấp đôi số đĩa, như vậy sẽ có bốn đĩa chính thức và bốn đĩa dùng để dự phòng.

Minh hoạ RAID mức 1 với 4 ổ đĩa chính và 4 ổ dự phòng:



- Khi ghi dữ liệu, tất cả các strip được ghi làm hai bản.
- Còn khi đọc thì sử dụng bản nào cũng được, cũng có thể đọc song song trên cả hai bản.
- Do đó hiệu suất ghi dữ liệu sẽ không cao bằng sử dụng đĩa đơn, nhưng hiệu suất đọc có thể tăng gấp đôi.
- Khả năng chống lỗi thì tuyệt vời: nếu có một ổ đĩa bị hỏng thì chỉ việc sử dụng bản sao còn lại để thay thế. Việc khắc phục hệ thống cũng rất đơn giản, chỉ việc thay một ổ đĩa mới, rồi sao chép toàn bộ dữ liệu từ ổ dự phòng vào đó.

RAID mức 2

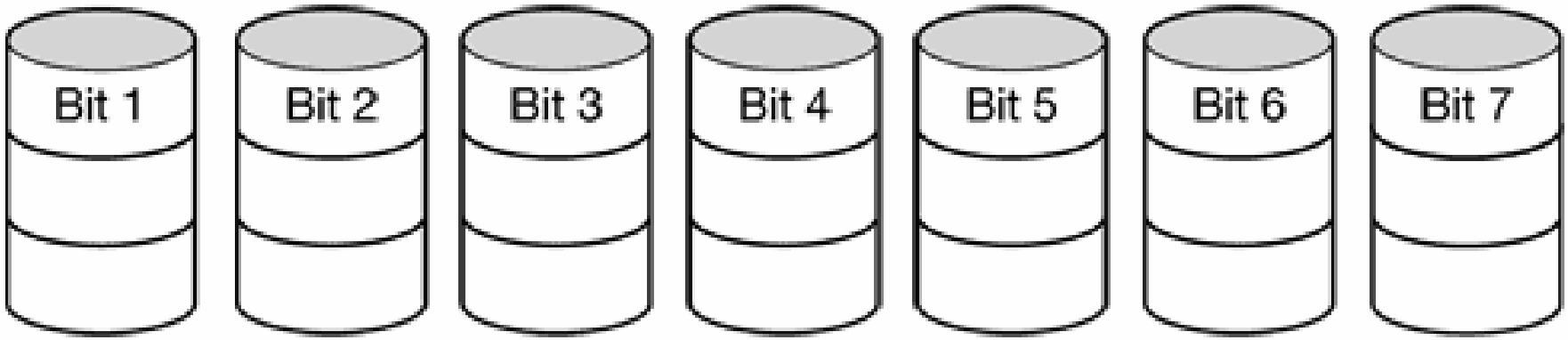
- RAID mức 2 hoạt động dựa trên các word dữ liệu (mỗi word gồm nhiều bít).
- Ví dụ:

+ Chia đôi mỗi byte của đĩa đơn ảo thành từng cặp 4 bít, rồi lắp thêm 3 bít mã Hamming vào để tạo thành word 7 bít, trong đó các bít 1, 2, và 4 là các bít chẵn lẻ.

+ Khi ghi Word dữ liệu lên đĩa, 7 bít sẽ được đồng thời ghi trên 7 đĩa khác nhau.

+ Bảy ổ đĩa phải được đồng bộ về vị trí của cánh tay đĩa và chiều quay.

Minh hoạ RAID mức 2 với 7 ổ đĩa:



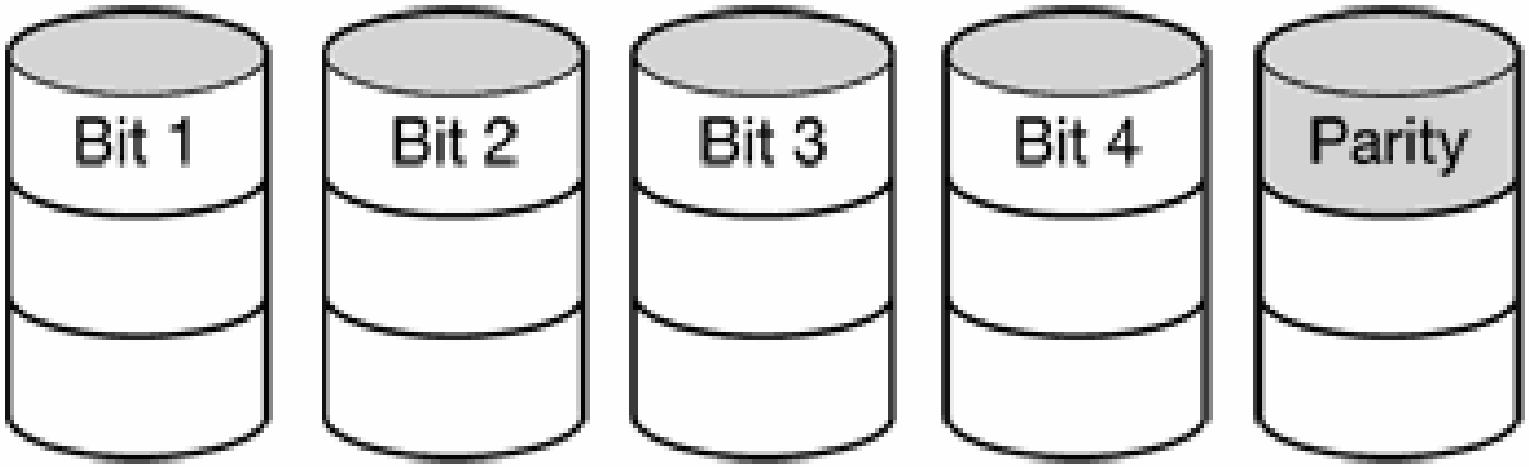
- Giải pháp này đòi hỏi phải đồng bộ được sự quay của tất cả các ổ đĩa, và nó chỉ có ý nghĩa khi sử dụng một số lượng lớn các đĩa (Với 32 đĩa chứa dữ liệu và 6 đĩa chứa bít chẵn lẻ, chi phí lên tới 19%).
- Nó cũng đòi hỏi phải có nhiều bộ điều khiển, do nó phải thực hiện tính toán các mã Hamming với từng bít.

- Nếu có một ổ đĩa bị hỏng thì cũng không gây ra điều gì nghiêm trọng, vì cũng chỉ là mất một bít trong tổng số nhiều bít, đôi khi có thể sử dụng các mã Hamming để khôi phục bít này ngay lập tức.

RAID mức 3

- RAID mức 3 là một phiên bản đơn giản của RAID mức 2. Chỉ có 1 bít chẵn lẻ được tính toán cho mỗi word dữ liệu, rồi ghi vào ổ đĩa chẵn lẻ.
- Giống như RAID mức 2, các ổ đĩa phải được đồng bộ chính xác, do mỗi word dữ liệu được cắt rải rác trên nhiều ổ đĩa.

Minh hoạ RAID mức 3 với 4 ổ chứa dữ liệu và 1 ổ chứa bít chẵn lẻ:

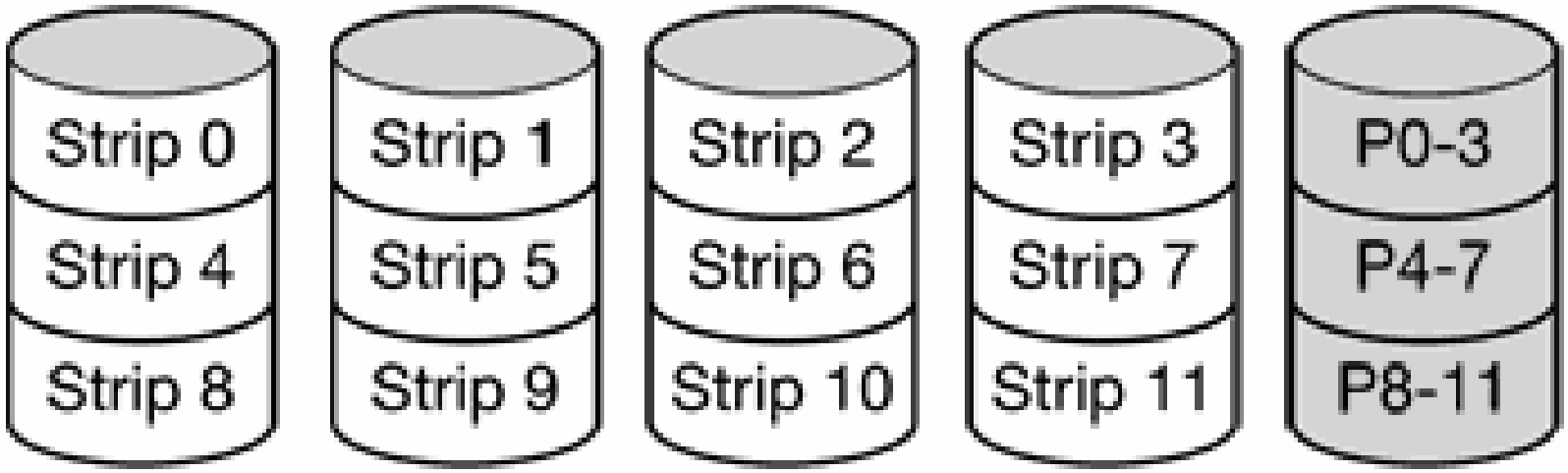


- Trong trường hợp có một ổ đĩa bị hỏng, thì 1 bít chẵn lẻ này có thể giúp khắc phục lỗi.
- Khi một ổ đĩa hỏng, bộ điều khiển chỉ việc giả thiết tất cả các bít của nó bằng 0. Nếu word thu được có bít chẵn lẻ bị sai, thì chứng tỏ bít trên ổ đĩa hỏng có giá trị bằng 1.

RAID mức 4

- RAID mức 4 cũng giống như RAID mức 0, nhưng có thêm một ổ đĩa để chứa strip chẵn lẻ của các strip dữ liệu.
- Nhờ vậy độ an toàn được tăng lên vì có thể khôi phục lại dữ liệu bị hỏng.

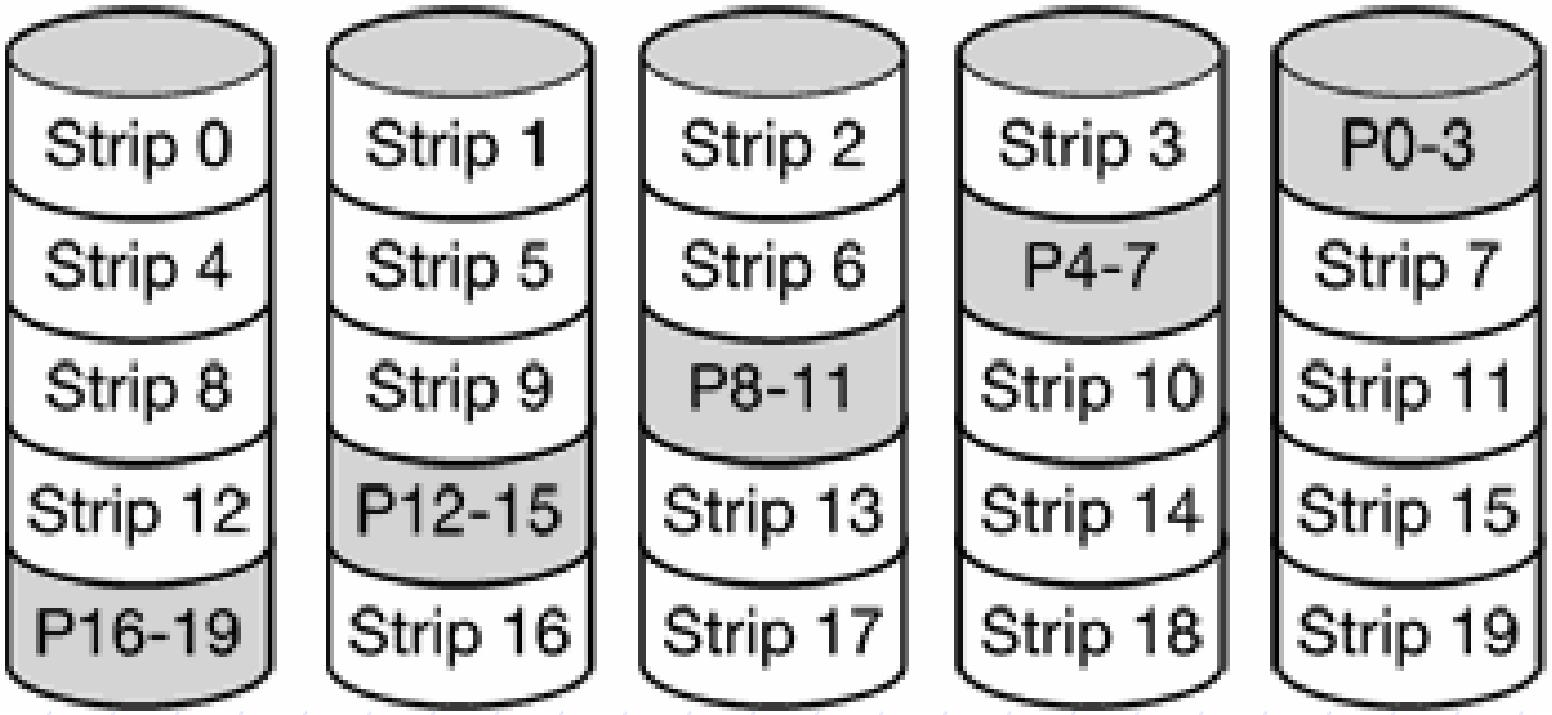
Minh hoạ RAID mức 4 với 4 ổ chứa dữ liệu và 1 ổ chứa các bít chẵn lẻ:



- Ví dụ, nếu mỗi strip có *k* byte, thì tất cả các strip dữ liệu liên quan sẽ được XOR với nhau, kết quả thu được là một strip chẵn lẻ dài *k* byte.
- Nếu một ổ đĩa bị hỏng, byte bị mất có thể được khôi phục lại nhờ thông tin trên ổ đĩa chẵn lẻ.

- Thiết kế này giúp tăng cường sự an toàn nếu có ổ đĩa bị hỏng, nhưng lại rất phiền phức khi cần thực hiện những thay đổi nhỏ. Nếu muốn thay đổi nội dung một sector, sẽ phải đọc tất cả các ổ đĩa để tính toán lại các bít chẵn lẻ, và sau đó phải ghi lại chúng.
- Kết quả là ổ đĩa chẵn lẻ có thể bị quá tải, và có thể dẫn tới tắc nghẽn. Sự tắc nghẽn này có thể được giải quyết nhờ RAID mức 5.

RAID mức 5

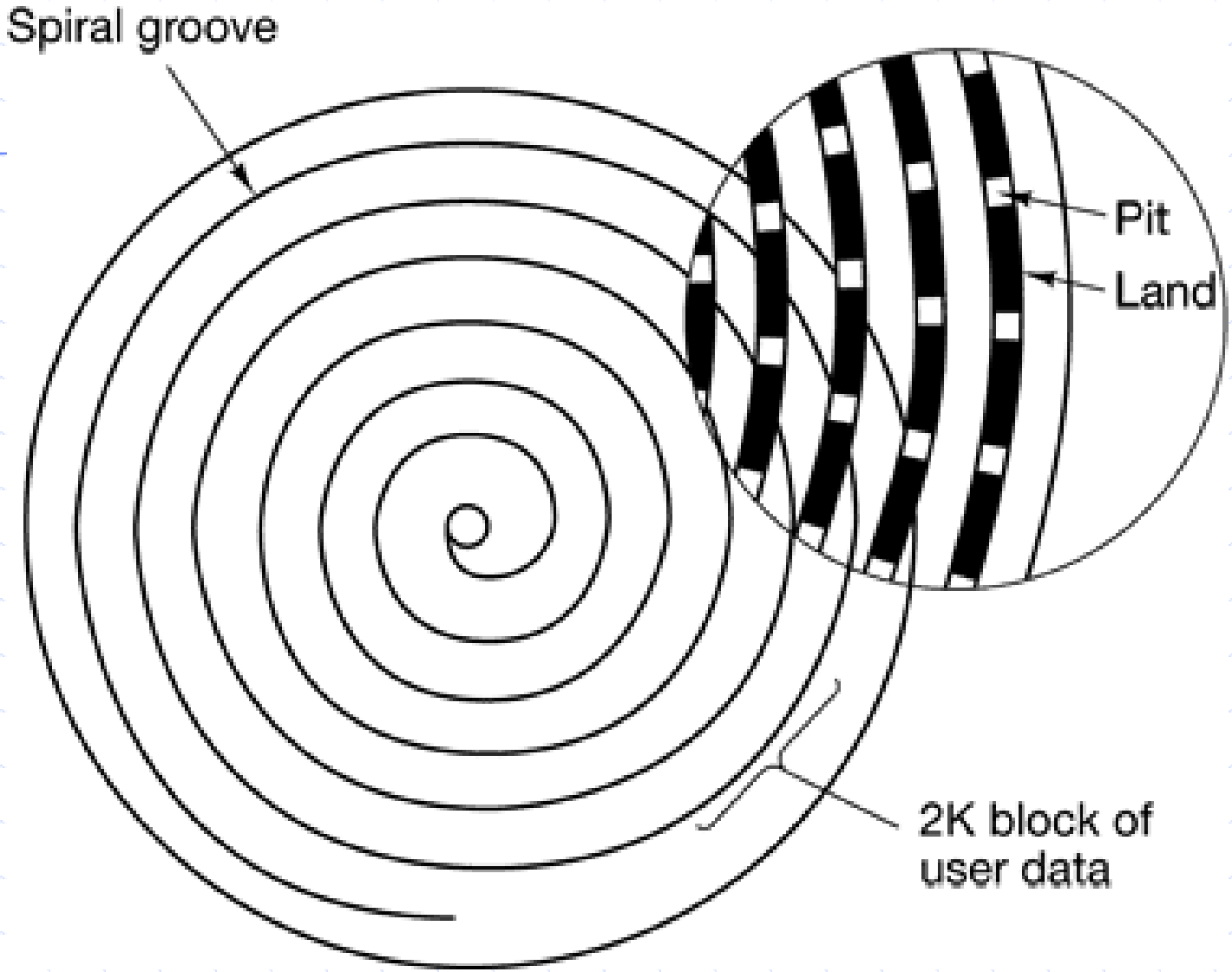

Strip 0
Strip 4
Strip 8
Strip 12
P16-19
Strip 1
Strip 5
Strip 9
P12-15
Strip 16
Strip 2
Strip 6
P8-11
Strip 13
Strip 17
Strip 3
P4-7
Strip 10
Strip 14
Strip 18
P0-3
Strip 7
Strip 11
Strip 15
Strip 19

- RAID mức 5 không dành riêng một ổ đĩa để chứa các bít chẵn lẻ, mà phân phối chúng trên các ổ đĩa thành phần.
- Tuy nhiên, nếu xảy ra biến cố hỏng đĩa, thì việc khôi phục lại dữ liệu liệu trên ổ đĩa hỏng sẽ phức tạp hơn.

CD ROM

- Lúc đầu, các đĩa quang được phát triển để ghi các chương trình truyền hình, nhưng về sau chúng được sử dụng như một thiết bị lưu trữ của máy tính.
- Các đĩa CD ROM tiêu chuẩn có đường kính 120 mm và dày 1.2 mm, với một lỗ thủng ở giữa rộng 15 mm. Chúng được coi là có thể bền tới 100 năm!

- Đĩa CD được sản xuất bằng cách dùng tia laser công suất lớn tạo các hố có đường kính 0.8 micrô mét trên trên một đĩa chủ bọc thuỷ tinh. \
- Từ đĩa chủ này người ta sẽ tạo ra một khuôn đúc, nó có các điểm gồ lên tại vị trí của các hố do tia laser tạo ra. Bên trong khuôn người ta đổ đầy nhựa polycarbonate lỏng để tạo thành đĩa CD, đĩa này sẽ có các hố lõm giống hệt như trên đĩa chủ.
- Sau đó một lớp nhôm rất mỏng sẽ được phủ lên trên bề mặt polycarbonate, rồi quét lên đó một lớp sơn bảo vệ, trên cùng người ta sẽ dán một cái nhãn. Các hố lõm trên lớp polycarbonate được gọi là các **pit** (lõm), còn các vùng không bị khoét lõm được gọi là **land** (phẳng).


Spiral groove
Pit
Land
2K block of
user data

- Các pit và land được ghi theo một đường xoắn ốc, bắt đầu từ vị trí gần với lỗ thủng ở giữa đĩa, rồi mở rộng ra phía rìa đĩa.
- Đường xoắn ốc cuốn quanh đĩa khoảng 22188 vòng (khoảng 600 vòng trên 1 mm). Nếu trải ra nó có thể dài tới 5.6 km.

- Các ổ CD-ROM tốc độ đơn hoạt động ở tốc độ 75 sector/s, tức là tốc độ dữ liệu sẽ bằng 153600 byte/s hoặc 175200 byte/s (tuỳ theo cách tổ chức dữ liệu trên đĩa). Các ổ đĩa tốc độ kép có tốc độ nhanh gấp đôi…
- Cứ như vậy, một ổ 40x có thể đạt được tốc độ 40 x 153600 byte/s (giả sử giao diện ổ, bus, và hệ điều hành có thể xử lý dữ liệu với tốc độ này).
- Một đĩa audio CD chuẩn có thể chứa 74 phút âm nhạc, với dung lượng bằng 681 984 000 bytes. Con số này thường được ghi là 650 MB vì 1 MB bằng 2^{20} byte (1 048 576 bytes).

Hết Phần 6