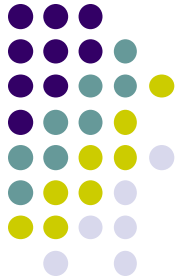
Kiến trúc tập lệnh



2.1. Biểu diễn dữ liệu trong máy tính

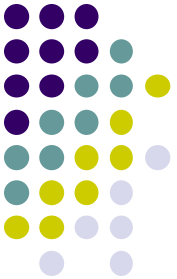
- 2.1.1. Các hệ đếm cơ bản
- 2.1.2. Biểu diễn số nguyên
- 2.1.3. Thực hiện các phép toán số học với số nguyên
- 2.1.4. Số dấu phẩy động
- 2.1.5. Biểu diễn ký tự

2.2. Các tập thanh ghi

2.3. Các kiểu lệnh

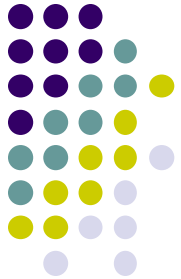
2.4. Các chế độ địa chỉ

Các hệ đếm cơ bản



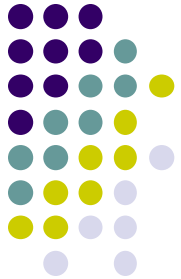
- Hệ thập phân (Decimal System)
  → con người sử dụng
- Hệ nhị phân (Binary System)
  → máy tính sử dụng
- Hệ mười sáu (Hexadecimal System)
  → dùng để viết gọn cho số nhị phân

Hệ thập phân



- Cơ số 10
- 10 chữ số: 0,1,2,3,4,5,6,7,8,9
- Dùng n chữ số thập phân có thể biểu diễn được 10^n giá trị khác nhau:
  - 00...000 = 0
  - 99...999 = 10^n - 1

Dạng tổng quát của số thập phân



$$A = a_n a_{n-1} \ldots a_0 , a_{-1} a_{-2} \ldots a_{-m}$$

$$A = a_n.10^n + a_{n-1}.10^{n-1} + \ldots + a_0.10^0 + a_{-1}.10^{-1} + \ldots + a_{-m}10^{-m}$$

$$A = \sum_{i=-m}^{n} a_i.10^i$$

Ví dụ hệ thập phân

$$123.45 = 1\times10^{2} + 2\times10^{1} + 3\times10^{0} + 4\times10^{-1} + 5\times10^{-2}$$

123:10 = 12 dư 3

12:10 = 1 dư 2

1:10 = 0 dư 1

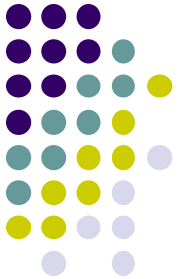
Các chữ số phần nguyên là 123

0.45 x 10 = 4.5 phần nguyên là 4

0.5 x 10 = 5.0 phần nguyên là 5

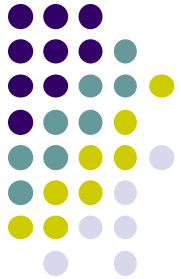
Các chữ số phần thập phân 45

Hệ nhị phân



- Cơ số 2
- 2 chữ số nhị phân: 0 và 1
- chữ số nhị phân gọi là ***bit*** (binary digit)
- Bit là đơn vị thông tin nhỏ nhất
- Dùng n bit có thể biểu diễn được 2^n giá trị khác nhau:
  - 00...000 = 0
  - 11...111 = 2^n - 1

Dạng tổng quát của số nhị phân



$$A = b_n b_{n-1} \ldots b_0 , b_{-1} b_{-2} \ldots b_{-m}$$

$$A = b_n \times 2^n + b_{n-1} \times 2^{n-1} + \ldots + b_0 \times 2^0 + b_{-1} \times 2^{-1} + b_{-2} \times 2^{-2} + \ldots + b_{-m} \times 2^{-m}$$

$$A = \sum_{i=-m}^{n} b_i \times 2^i$$

Ví dụ số nhị phân

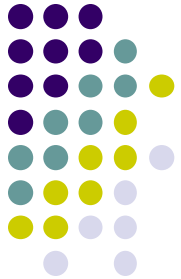
$$A = 101101.1101_{(2)}$$

$$= 1x2^5 + 0x2^4 + 1x2^3 + 1x2^2 + 0x2^1 + 1x2^0 + 1x2^{-1} + 1x2^{-2} + 0x2^{-3} + 1x2^{-4}$$

$$= 32 + 8 + 4 + 1 + 0.5 + 0.25 + 0.0625$$

$$= 45.8125$$

Chuyển từ số nguyên hệ 10 sang số 2



- Có hai cách chuyển
  - Cách 1: Chia dần cho 2 rồi lấy phần dư
  - Cách 2: Phân tích số cần chuyển thành tổng của các lũy thừa 2.

Chia dần cho 2 lấy phần dư

Ví dụ: Chuyển 87 của hệ 10 sang hệ 2

87 : 2 = 43 dư 1

43 : 2 = 21 dư 1

21 : 2 = 10 dư 1

10 : 2 = 5 dư 0

5 : 2 = 2 dư 1

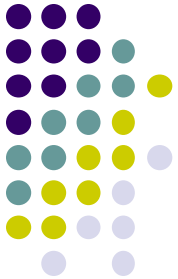
2 : 2 = 1 dư 0

1 : 2 = 0 dư 1

Vậy $87_{(10)} = 1010111_{(2)}$

Vậy 87 = 1010111b

Phân tích thành tổng của các lũy thừa 2



Dựa vào bảng lũy thừa 2

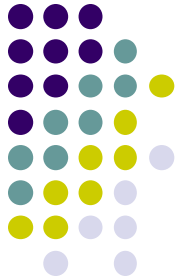
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1

Phân tích số cần chuyển thành tổng các lũy thừa 2

Ví dụ: Chuyển số 87 hệ 10 sang hệ 2

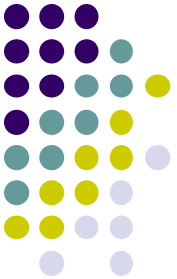
$87 = 64 + 16 + 4 + 2 + 1 = 2^6 + 2^4 + 2^2 + 2^1 + 2^0$

$= 1010111_{(2)}$



- Chuyển số 125 từ sang hệ 2 bằng 2 cách
- 125=64+32+16+8+4 +1 = 2^6+…..

Chuyển phần thập phân hệ 10 sang hệ 2

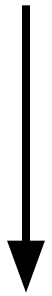


Ví dụ: Chuyển $0.875_{(10)}$ sang hệ 2

0.875 x 2 = 1.75 phần nguyên là 1

0.75 x 2 = 1.5 phần nguyên là 1

0.5 x 2 = 1.0 phần nguyên là 1



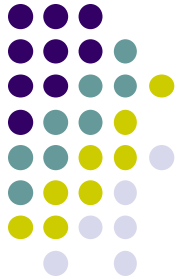
Vậy $0.875_{(10)} = 0.111_{(2)} = 111_{(2)}$

XOR: giống nhau thì kết quả là 0

Khác nhau kết quả là 1:

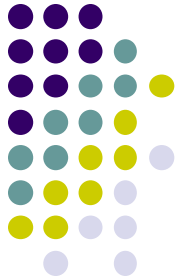
0 xor 0 = 0 1 xor 1 = 0

Hệ 16 (Hexa)



- Cơ số 16
- 16 chữ số: 0,1,2,3,4,5,6,7,8,9, A,B,C,D,E,F
- Dùng để viết gọn cho số nhị phân: cứ một nhóm 4-bit sẽ được thay bằng một chữ số Hexa

Chuyển đổi từ 10 sang hệ 16



Chia liên tiếp phần nguyên cho 16. Viết các số dư theo chiều ngược lại. Phần thập phân nhân liên tiếp với 16. Nhân đến khi thu được kết quả là số nguyên (đủ độ chính xác, vô hạn tuần hoàn).

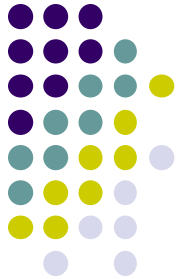
Ví dụ: 100 chuyển sang hệ 16

100 chia cho 16 = 6 dư 4

6 chia cho 16 = 0 và dư 6

Như vậy 100 = 64h

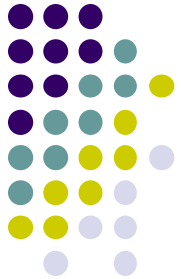
Chuyển đổi từ hệ 2 sang hệ 16



Các bước chuyển đổi từ hệ 2 sang hệ 16:

- Nhóm 4 chữ số hệ 2 thành một nhóm. Nhóm từ phải sang trái. Nhóm cuối cùng nếu không đủ 4 số thì bổ sung thêm số 0 vào trước cho đủ 4 số.
- Chuyển 4 chữ số hệ 2 sang chữ số hệ 16 tương ứng.

Ví dụ chuyển từ hệ 2 sang hệ 16



$10101111001010011110b$

Nhóm 1: $1110_{(2)} = 14_{(10)} = E_{(16)}$

Nhóm 2: $0100_{(2)} = 4_{(10)} = 4_{(16)}$

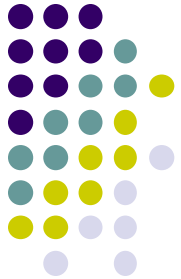
Nhóm 3: $1001_{(2)} = 9_{(10)} = 9_{(16)}$

Nhóm 4: $1011_{(2)} = 11_{(10)} = B_{(16)}$

Nhóm 5: $0010_{(2)} = 2_{(10)} = 2_{(16)}$

Vậy: $10101111001010011110_{(2)} = 2B94E_{(16)}$

Chuyển đổi từ hệ 16 sang hệ 2



Các bước chuyển đổi từ hệ 16 sang hệ 2:

- Phân tích mỗi chữ số của hệ 16 thành 4 chữ số của hệ 2.
- Ví dụ minh họa:
- AB4C5h □ hệ 2 = 1010 1011 0100 1100 0101b

Ah = 10 = 1010b

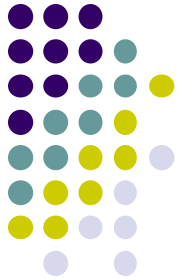
Bh = 11 = 1011b

4h = 4h = 0100b

Ch = 12 = 1100b

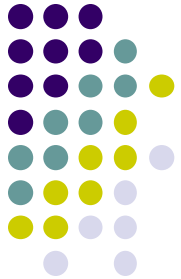
5h = 5 = 0101b

Lưu trữ dữ liệu trong máy tính



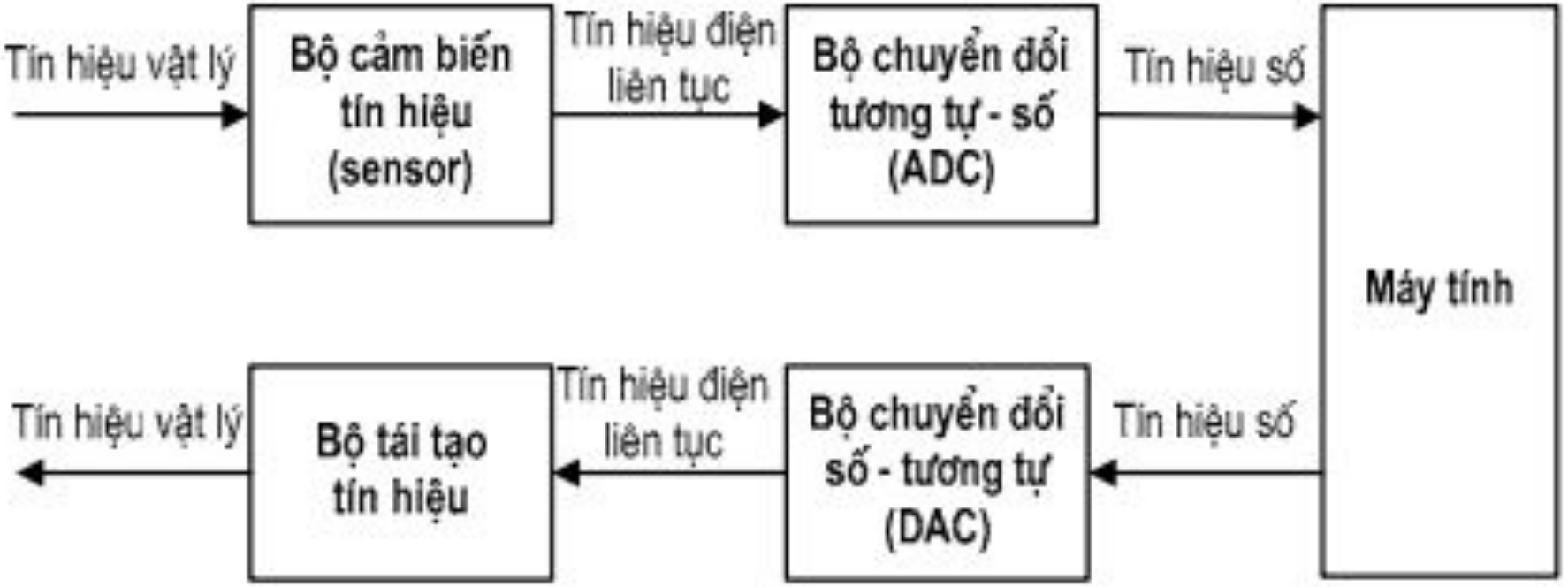
- Tất cả dữ liệu lưu trong máy tính đều phải được mã hóa sang nhị phân.
- Các loại dữ liệu:
  - Dữ liệu nhân tạo: Do con người quy ước.
  - Dữ liệu tự nhiên: Tôn tại khách quan với con người.

Mã hóa dữ liệu nhân tạo



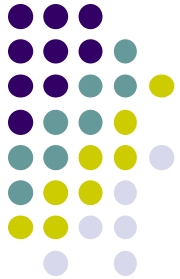
- Mã hóa theo chuẩn quy ước.
- Dữ liệu số
  - Số nguyên: Mã hóa theo một số chuẩn
  - Số thực: Mã hóa bằng số dấu phẩy động
- Dữ liệu ký tự: Mã hóa theo bảng mã ký tự

Mã hóa và tái tạo tín hiệu vật lý



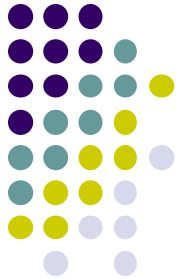
- Các tín hiệu vật lý thông dụng:
  - Âm thanh
  - Hình ảnh

Độ dài của từ dữ liệu



- Độ dài của từ dữ liệu là số bits được sử dụng để mã hóa loại dữ liệu tương ứng.
- **Độ dài của từ dữ liệu thường là bội của 8 bits.**
- Ví dụ: 8 bits (byte), 16 bits (2 byte = 1 word), 32 bits (double word), 64 bits.
- char x □ x biểu diễn 8 bit
- int I □ I biểu diễn 16 bit
- float f□ f biểu diễn 32 bit
- double d□ d biểu diễn 64 bit

Lưu trữ dữ liệu trong bộ nhớ

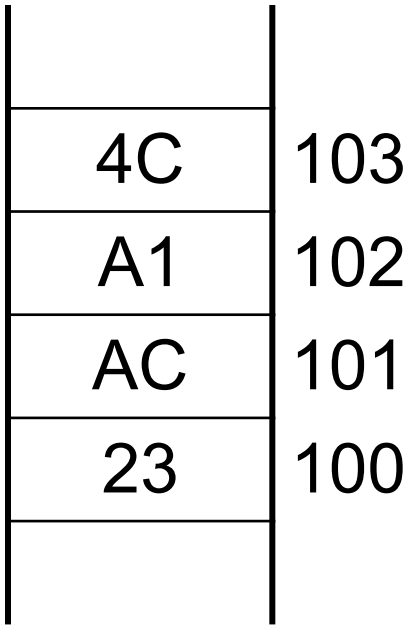


- Có hai cách lưu trữ dữ liệu nhiều byte trong máy tính.
  - Đầu nhỏ (*Little-endian*): Byte có trọng số thấp được lưu trữ ở ngăn nhớ có địa chỉ nhỏ, byte có trọng số cao được lưu trữ ở ngăn nhớ có địa chỉ lớn. 11111111 00000000
  - Đầu to (*Big-endian*): Byte có trọng số cao được lưu trữ ở ngăn nhớ có địa chỉ nhỏ, byte có trọng số thấp được lưu trữ ở ngăn nhớ có địa chỉ lớn.

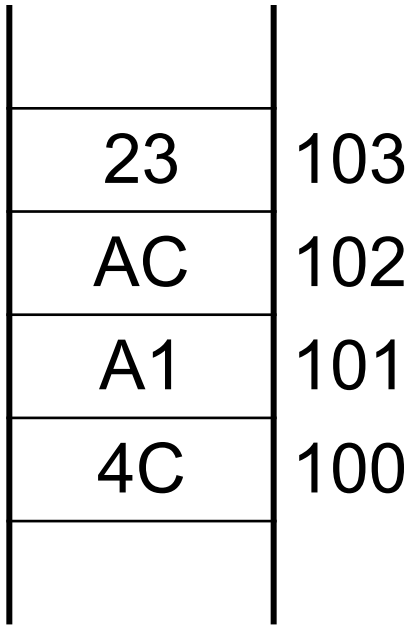
Ví dụ lưu trữ dữ liệu 32 bits

$01001100101000011010110000100011_{(2)}$

$4C\ A1\ AC\ 23_{(16)}$

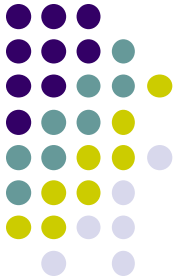


Little-endian



Big-endian

Biểu diễn số nguyên trong máy tính



- Số nguyên không dấu (unsigned integer)
- Số nguyên có dấu (signed integer)

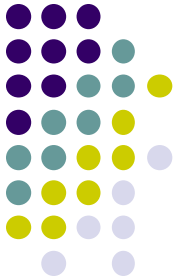
unsigned int y; □ y là số không âm >=0

int x;□ x là số có dấu □ x có thể mang giá trị âm và dương.

x = -10; //ok

x = 5; //ok

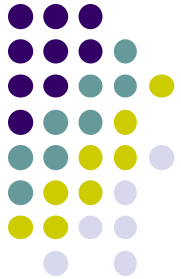
Biểu diễn số 100 trong máy tính ở dạng 8 bit



- 100:2 = 50 dư 0
- 50:2 = 25 dư 0
- 25:2 = 12 dư 1
- 12:2 = 6 dư 0
- 6:2 = 3 dư 0
- 3:2 = 1 dư 1
- 1:2 = 0 dư 1

100 = 01100100b

Biểu diễn số nguyên không dấu



- Nguyên tắc tổng quát: Dùng n bit biểu diễn số nguyên không dấu A:

$$A = b_{n-1}b_{n-2}...b_0$$

- Giá trị của A được tính như sau:

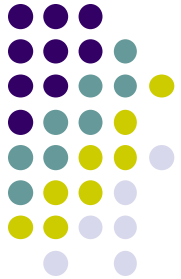
$$A = b_{n-1} \times 2^{n-1} + b_{n-2} \times 2^{n-2} + ... + b_0 \times 2^0$$

- Công thức tổng quát tính giá trị của A:

$$A = \sum_{i=0}^{n-1} b_i \times 2^i$$

- Phạm vi biểu diễn của A từ 0 đến $2^n - 1$

Ví dụ biểu diễn số nguyên không dấu trong máy tính



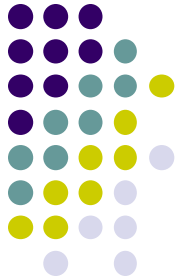
- Biểu diễn số nguyên không dấu 90 trong máy tính ở dạng 8 bits.

  $90_{(10)} = 64 + 16 + 8 + 2 = 2^6 + 2^4 + 2^3 + 2^1$

  $= 1011010_{(2)}$

- Vậy số 90 được lưu trong máy tính ở dạng 8 bits như sau: 01011010.

Biểu diễn số nguyên có dấu

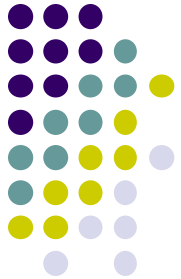


- Máy tính sử dụng bit có trọng số cao nhất (bên trái nhất) để làm bit dấu. Quy ước số âm bit dấu bằng 1. Số dương bit dấu bằng 0.
- Máy tính sử dụng số bù 2 để biểu diễn cho số nguyên âm.
- 10101010b số âm ? Âm?
- C1: -1*2^7+0*2^6+1*2^5+1*2^3+1*2^1
- =-128+32+8+2 = -86
- Số bù 2 như thế nào?

Số bù 2

- **Số bù 1 của một số là phủ định của chính nó**.
- **Số bù 2 bằng số bù 1 cộng thêm 1**.
- X muốn tìm bù 1 của x (y) x + y = 11111..b
- X = 1010 tìm bù 1 của X: 0101
  - 0 + 0 = 0; 0 + 1 = 1; 1 + 0 = 1; 1 + 1 = 10b
- Ví dụ tìm số bù 2 của: 10101010b
  - Số bù 1 là: 01010101b
  - Số bù 2 là: 01010101 + 1 = 01010110b

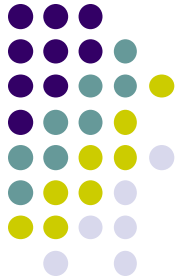
Các bước để biểu diễn số nguyên âm trong máy tính



- Bước 1: Biểu diễn số nguyên dương tương ứng ở dạng số bits yêu cầu.
- Bước 2: Tìm số bù 2 của số tìm được ở bước 1.

bù 2 của A sẽ là -A

Ví dụ biểu diễn số nguyên âm trong máy tính



- Biểu diễn số -90 trong máy tính ở dạng 8 bits
- char c = -90;
- Int c = -90; bieu c o dang 32
- **Bước 1**: Chuyển số 90 sang dạng nhị phân 8 bits

  $90 = 64 + 16 + 8 + 2 = 2^6 + 2^4 + 2^3 + 2^1$

  = 01011010b
- **Bước 2:** Tìm số bù 2 của số tìm được ở bước 1

  Số bù 1 của 01011010b = 10100101b

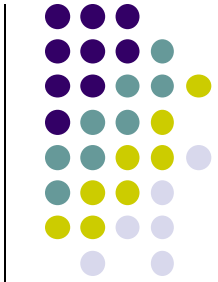
  Số bù 2 của 01011010b = 10100101b

  \+ 1

  =1010/0110b = A6h

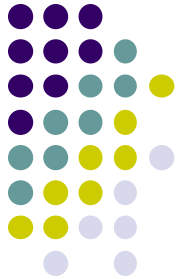
=-1*2^7+2^5+2^2+2^1 = -128 + 32 + 4 + 2 = -128 + 38 = -90

A = 10100110b



- Nếu A không dấu: (unsigned)
  - A = 2^7 + 2^5+2^2+2^1 = 128 + 32 + 4 + 2 = 166
- Nếu A có dấu: Bít dấu của A là 1-> A là số âm. Bù 2 của A sẽ là –A.
- Bù 2 của A.
  - Bù 1 của A là: 01011001b
  - Bù 2 của A là: bu1 + 1 = 01011010b
  - -A = 01011010b = 2^6 + 2^4 +2^3 + 2^1 = 64+16+8+2 = 90-> A = -90
  - Char c = 10100110b thì c = -90
  - Unsigned char c = 10100110b thì c = 166

Tính giá trị của số có dấu sau



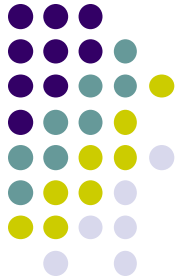
- A = 10110011b □? Đây là số âm □ A là bù 2 của một số dương X nào đó.

Bù 2 của X = bù 1 của X + 1□Bù 1 của X= Bù 2 – 1 = 10110011 – 1 = 10110010b □ X = 01001101 = 2^6+13 = 77□ A = -77.

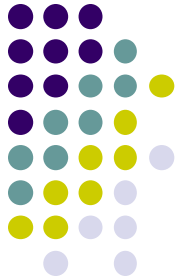
Bù 2 của số A là –A. –A = Bù 2 của A = Bù 1 của A + 1 = 01001100 + 1 = 01001101b = 77. □ -A = 77□ A = -77.

A = -1.2^7 + 32 + 16 + 3 = -128+51 = -77

Đổi các số nguyên thập phân sau ra số hex 16 bit: - 234



A = 10100110b



Cách 3: Bù 2 = Bù 1 + 1 Bù 1 = Bù 2 – 1.

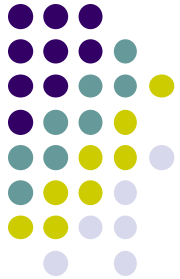
Bù 1 rồi phủ định sẽ tìm được lại số dương tương ứng lấy đối sẽ thu được số cần tính.

A chính là bù 2 của số dương tương ứng B (-A)

Số bù 1 = A – 1 = 10100110b – 1 = 10100101b

B = 01011010b = 64 + 16 + 8 + 2 = 90 A = -90

Biểu diễn



- Đổi các số nguyên thập phân sau ra số hex 16 bit: (biểu diễn -234 trong máy tính ở dạng 16 bit chuyển 16 bit sang hệ hex)
- - 234

Tính số có dấu sau: 10101011b ?

- Đổi các số nguyên thập phân sau ra số hex 16 bit: –16

Biểu diễn

- - 234

B1. Biểu diễn 234 ở dạng 16 bits

234 = 2^7 + 2^6 + 2^5 + 2^3 + 2^1 = 128 + 64 +32+10 = 0000000011101010b

B2. Tìm số bù 2

- Bù 1: 1111 1111 00010101b
- Bù 2 = bù 1 + 1 = 1111 1111 0001 0110b = FF16h

Kết luận -234 = FF16h int x = -234;

printf("%x",x); 0xff16

Tính?

Tính số có dấu sau: A = 10101011b □?

- C1: A= -1*2^7+32+11 = -128 + 43 = -85
- C2: Tìm bù 2 của A là –A

Bù 1 của A là: 01010100b

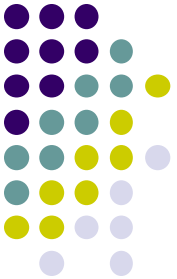
Bù 2 của A là: Bù 1 + 1 = 01010101b

-A = 64 + 16 + 5 = 85 □ A = -85

C3:

Theo định nghĩa Bù 1 của B = A – 1 = 10101010b□B = 01010101b = 85□A = -B = -85

Dạng tổng quát của số nguyên dương

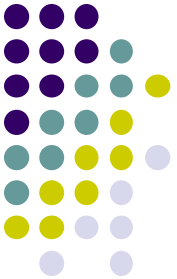


$$A = 0b_{n-2}b_{n-3}\ldots b_1 b_0$$

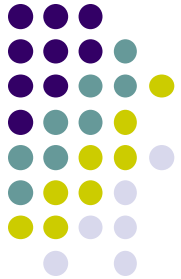
$$A = \sum_{i=0}^{n-2} b_i \times 2^i$$

$$A = 01101011 = 1\times 2^6 + 1\times 2^5 + 1\times 2^3 + 1\times 2^1 + 1\times 2^0 = 64 + 32 + 8 + 2 + 1 = 107$$

Biểu diễn số nguyên có dấu 129 trong máy tính



Dạng tổng quát của số nguyên âm

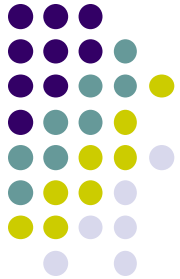


$$A = 1b_{n-2}b_{n-3}\ldots b_1 b_0$$

$$A = -2^{n-1} + \sum_{i=0}^{n-2} b_i \times 2^i$$

$$A = 11101011 = -2^7 + 1\times 2^6 + 1\times 2^5 + 1\times 2^3 + 1\times 2^1 + 1\times 2^0 = -128 + 64 + 32 + 8 + 2 + 1 = -21$$

Biểu diễn số nguyên theo mã BCD (Binary Code Decimal)

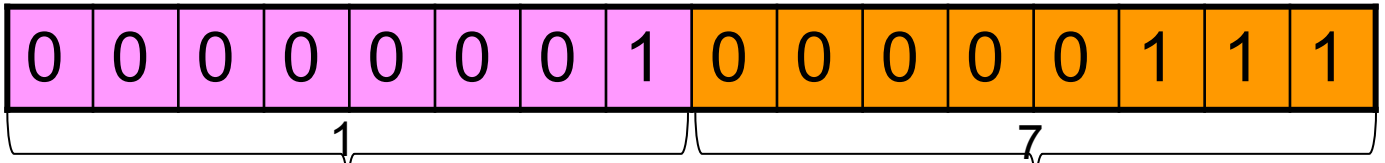


- Dùng 4 bit để mã hóa cho các chữ số thập phân từ 0 đến 9.

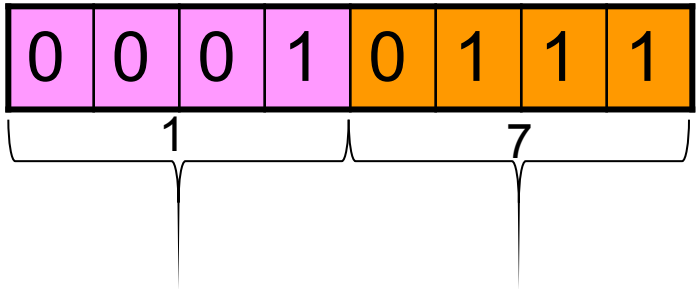
Số thập phân	Mã BCD	Số thập phân	Mã BCD
0	0000	5	0101
1	0001	6	0110
2	0010	7	0111
3	0011	8	1000
4	0100	9	1001

Biểu diễn số BCD

- BCD không đóng gói (unpacked BCD): Sử dụng 1 byte để lưu trữ cho một chữ số BCD (4 bit cao của byte bằng 0).
  - Ví dụ: biểu diễn số 17 ở dạng BCD không đóng gói:

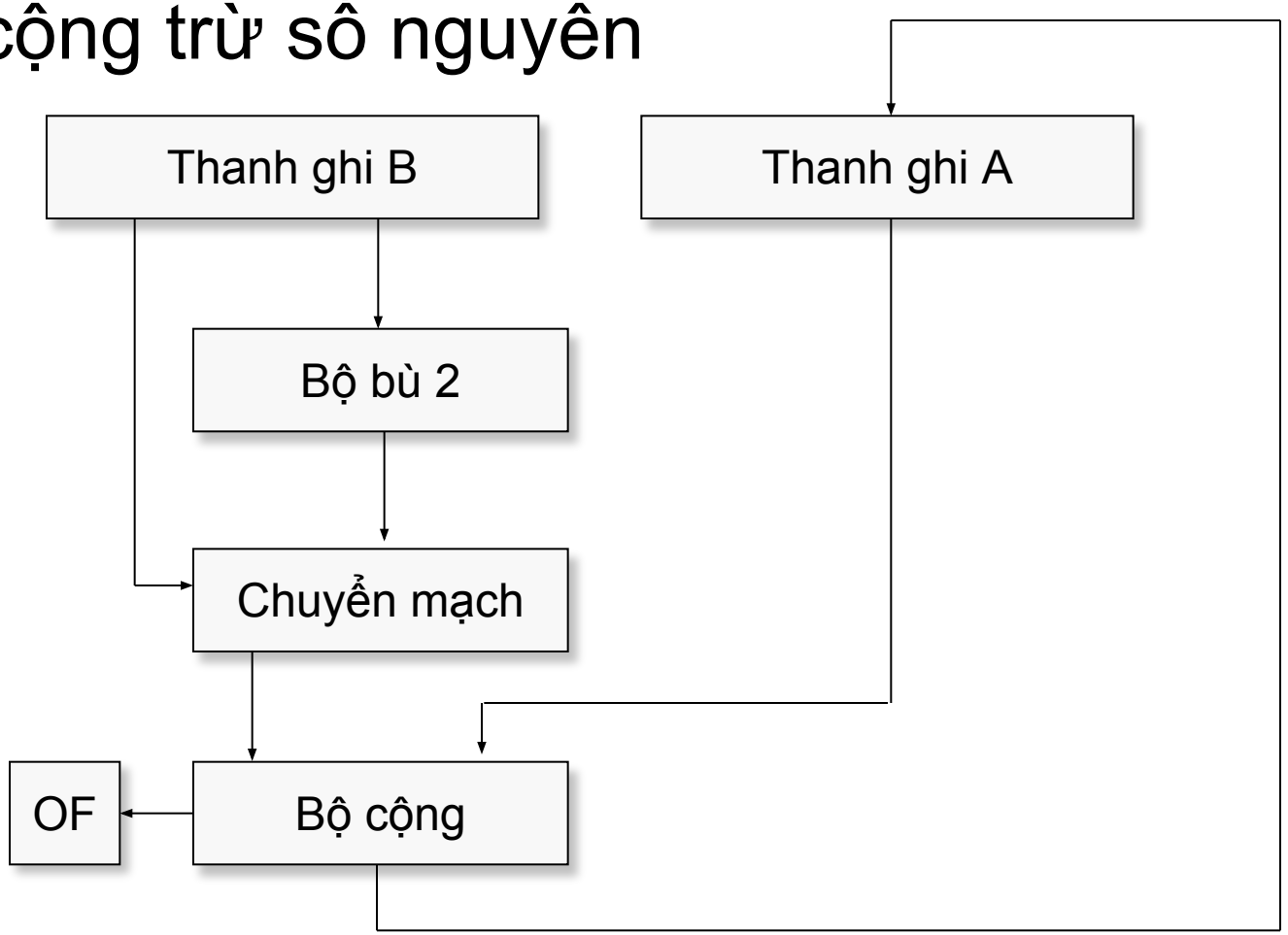


- BCD đóng gói (packed BCD): Hai số BCD được lưu trữ trong 1 byte

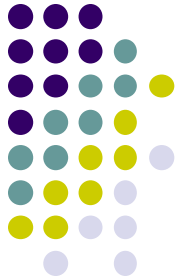


Các phép tính số học với số nguyên

Bộ cộng trừ số nguyên



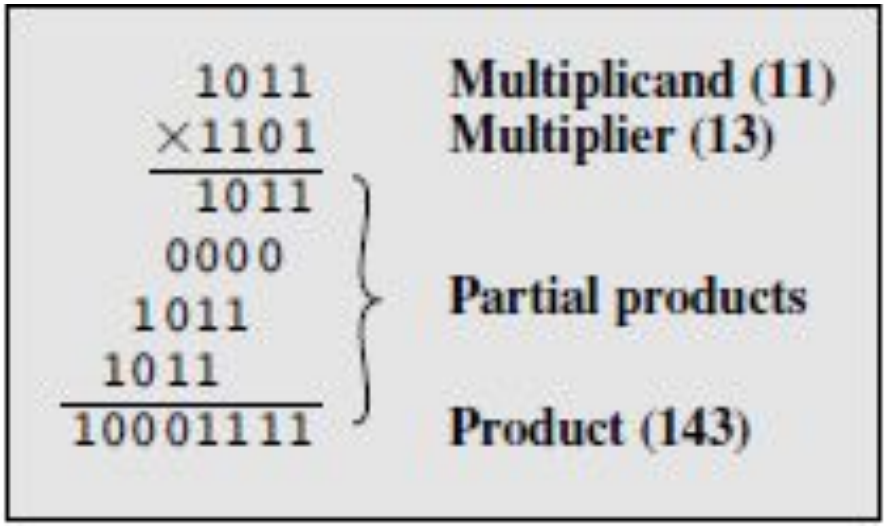
Phép cộng trừ số nguyên



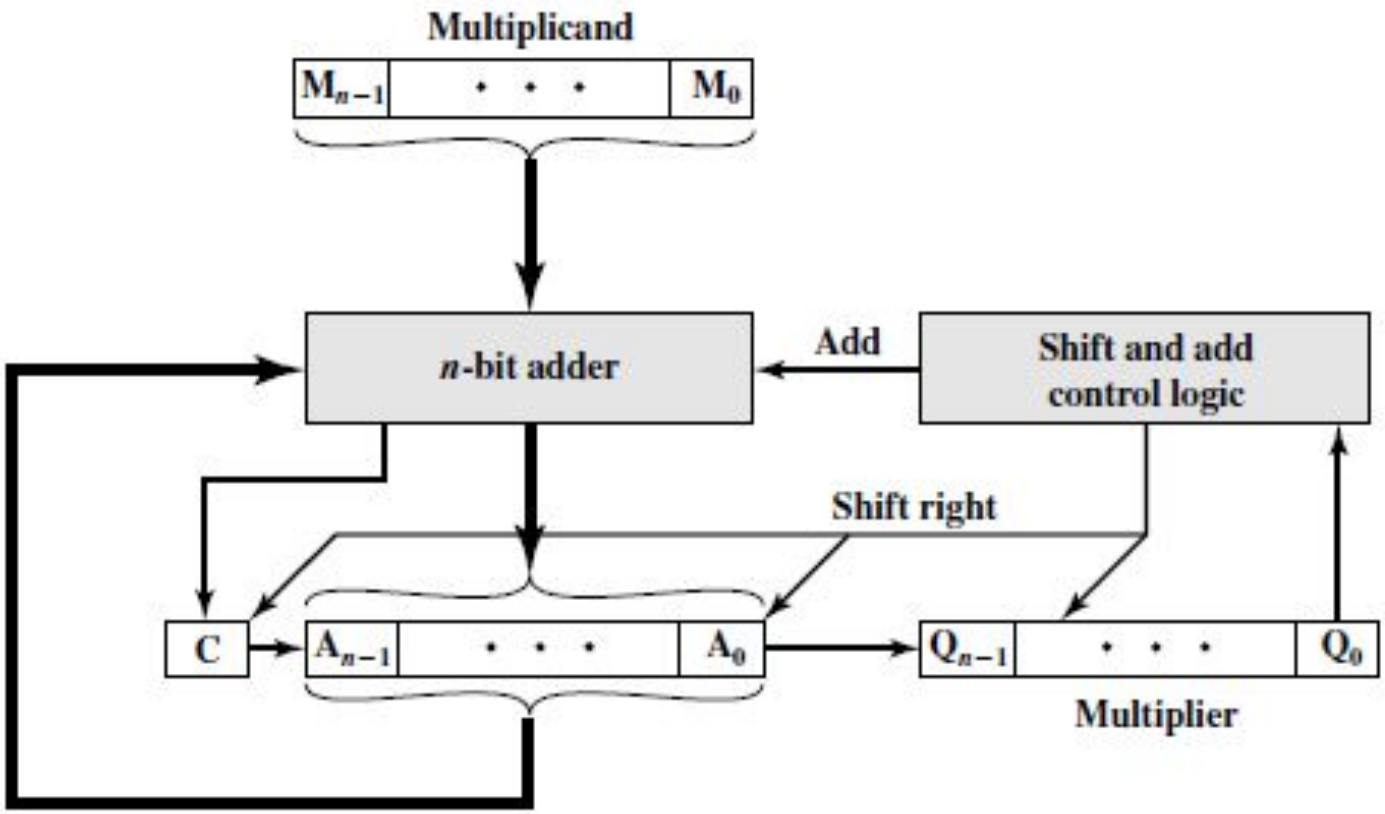
- Khi phép cộng có tràn thì kết quả bị sai. Muốn kết quả đúng thì phải tăng kích thước biểu diễn cho các số.
- Phép trừ chính là phép cộng với số bù 2.

Phép nhân số nguyên không dấu

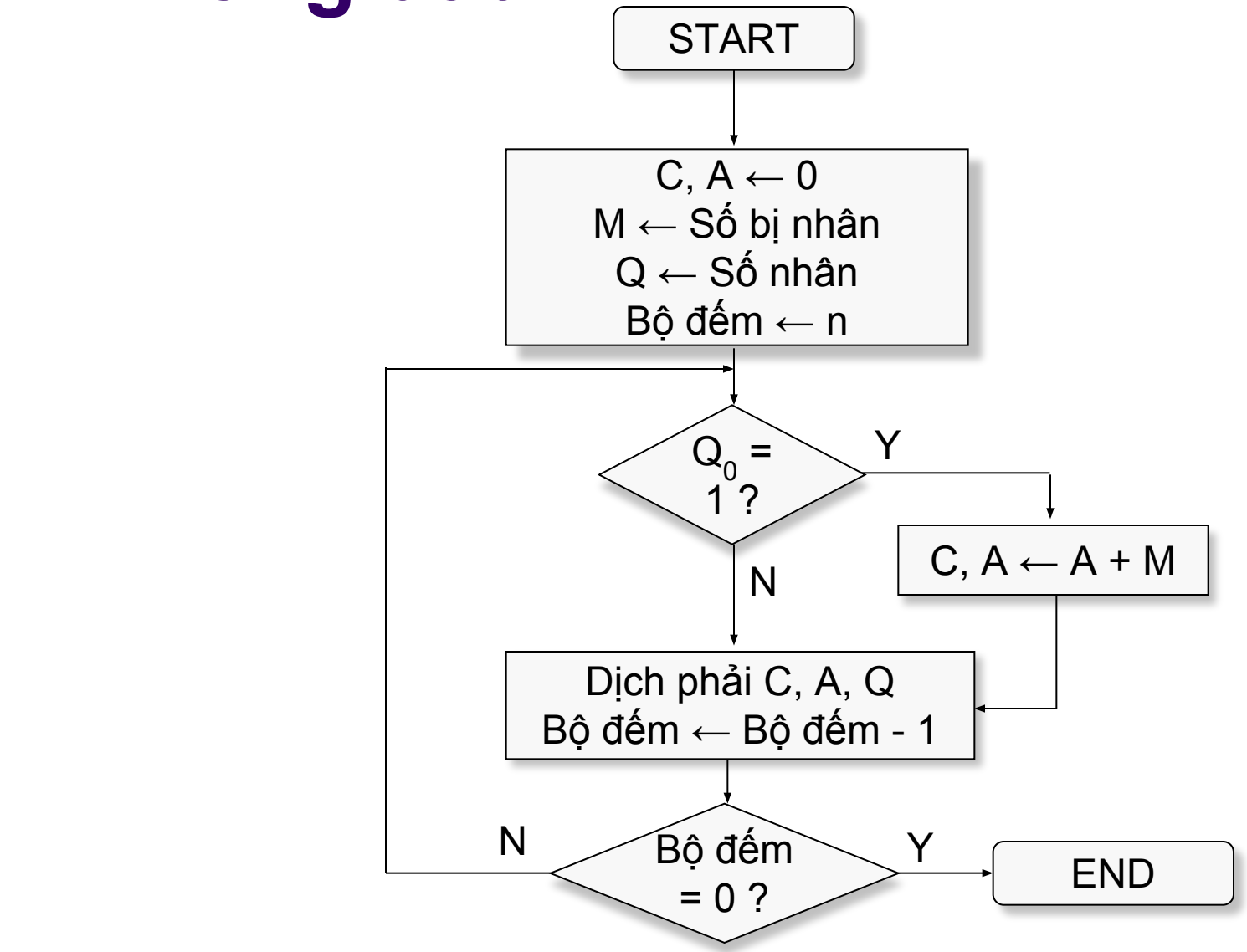
- Các tích riêng phần được xác định như sau:
  - Nếu bit của số nhân bằng 0 → tích riêng phần bằng 0.
  - Nếu bit của số nhân bằng 1 → tích riêng phần bằng số bị nhân.
- Tích riêng phần tiếp theo được dịch trái một bit so với tích riêng phần trước đó.
- Tích bằng tổng các tích riêng phần
- Nhân hai số nguyên n-bit, tích có độ dài 2n bit (không bao giờ tràn).



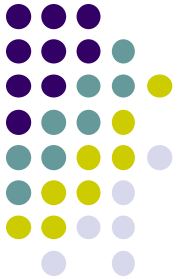
Sơ đồ bộ nhân số nguyên không dấu



Giải thuật nhân số nguyên không dấu

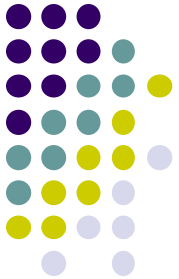


Minh họa nhân số nguyên không dấu



C	A	Q	M		
0	0000	1101	1011	Các giá trị khởi tạo	
0	1011	1101	1011	Cộng	Lần lặp thứ nhất
0	0101	1110	1011	Dịch	
0	0010	1111	1011	Dịch	Lần lặp thứ hai
0	1101	1111	1011	Cộng	Lần lặp thứ ba
0	0110	1111	1011	Dịch	
1	0001	1111	1011	Cộng	Lần lặp thứ tư
0	1000	1111	1011	Dịch	

Kết quả lưu trong AQ = $10001111_{(2)}$=128 + 15 = 143

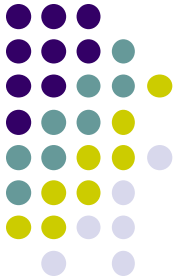


M = 10 = 1010b (số bị nhân)

Q = 12 = 1100b (số nhân)

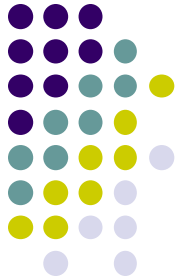
M*Q = ?

Nhân số nguyên có dấu



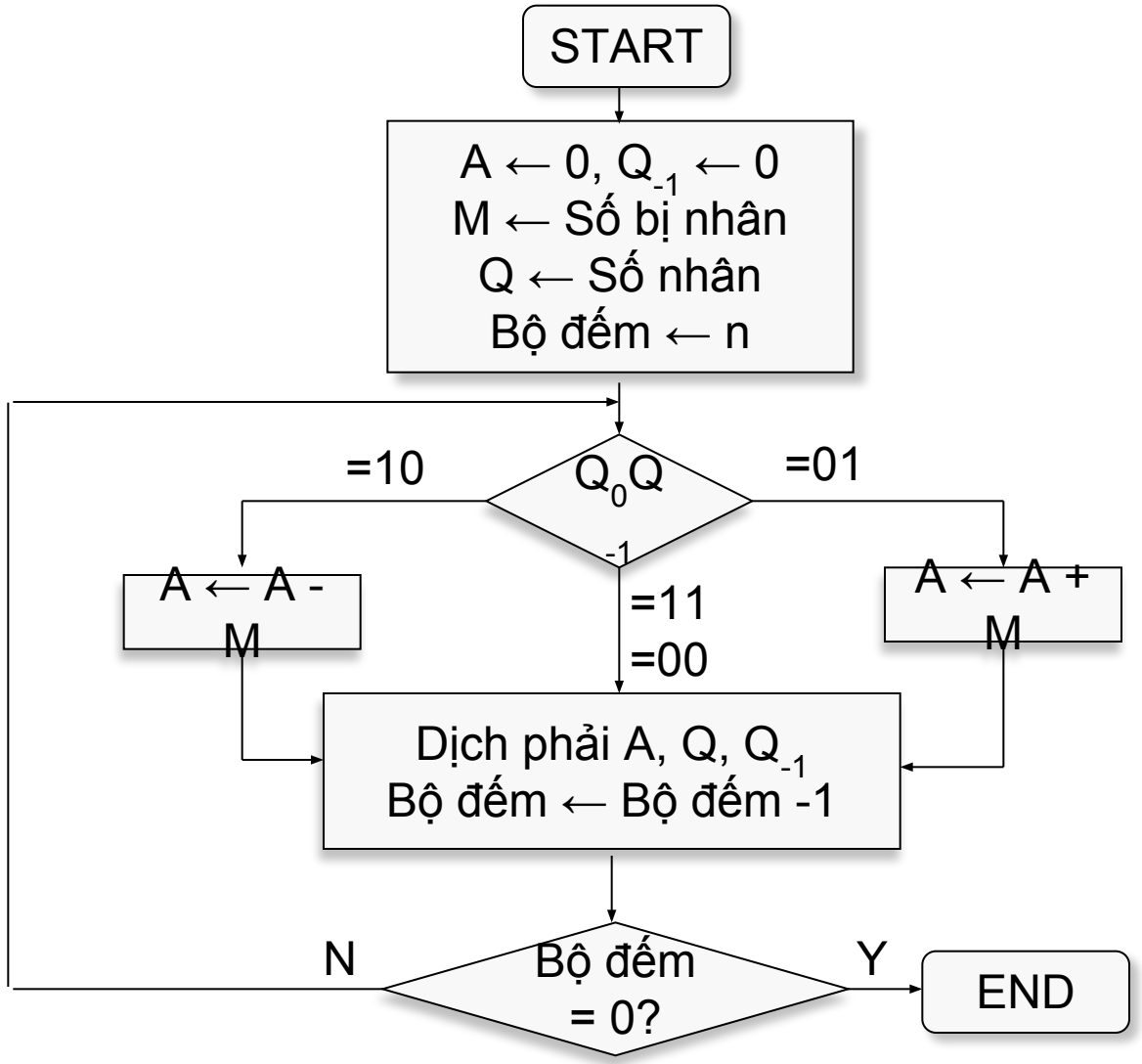
- Sử dụng thuật giải nhân không dấu
- Sử dụng thuật giải Booth

Sử dụng giải thuật nhân không dấu



- Bước 1. Chuyển đổi số bị nhân và số nhân thành số dương tương ứng
- Bước 2. Nhân hai số dương bằng thuật giải nhân số nguyên không dấu, được tích của hai số dương.
- Bước 3. Hiệu chỉnh dấu của tích:
  - Nếu hai thừa số ban đầu cùng dấu thì giữ nguyên kết quả ở bước 2.
  - Nếu hai thừa số ban đầu là khác dấu thì đảo dấu kết quả của bước 2.

Sử dụng giải thuật Booth



Ví dụ minh họa

A	Q	Q_{-1}	M	
0000	0011	0	0111	Các giá trị khởi tạo
1001	0011	0	0111	A ← A - M
1100	1001	1	0111	Dịch phải
1110	0100	1	0111	Dịch phải
0101	0100	1	0111	A ← A + M
0010	1010	0	0111	Dịch phải
0001	0101	0	0111	Dịch phải

Kết quả lưu trong AQ = 00010101 = 16 + 4 + 1 = 21

Chú ý: Khi dịch bit dấu của A được bảo tồn.

Chia số nguyên không dấu

Số chia

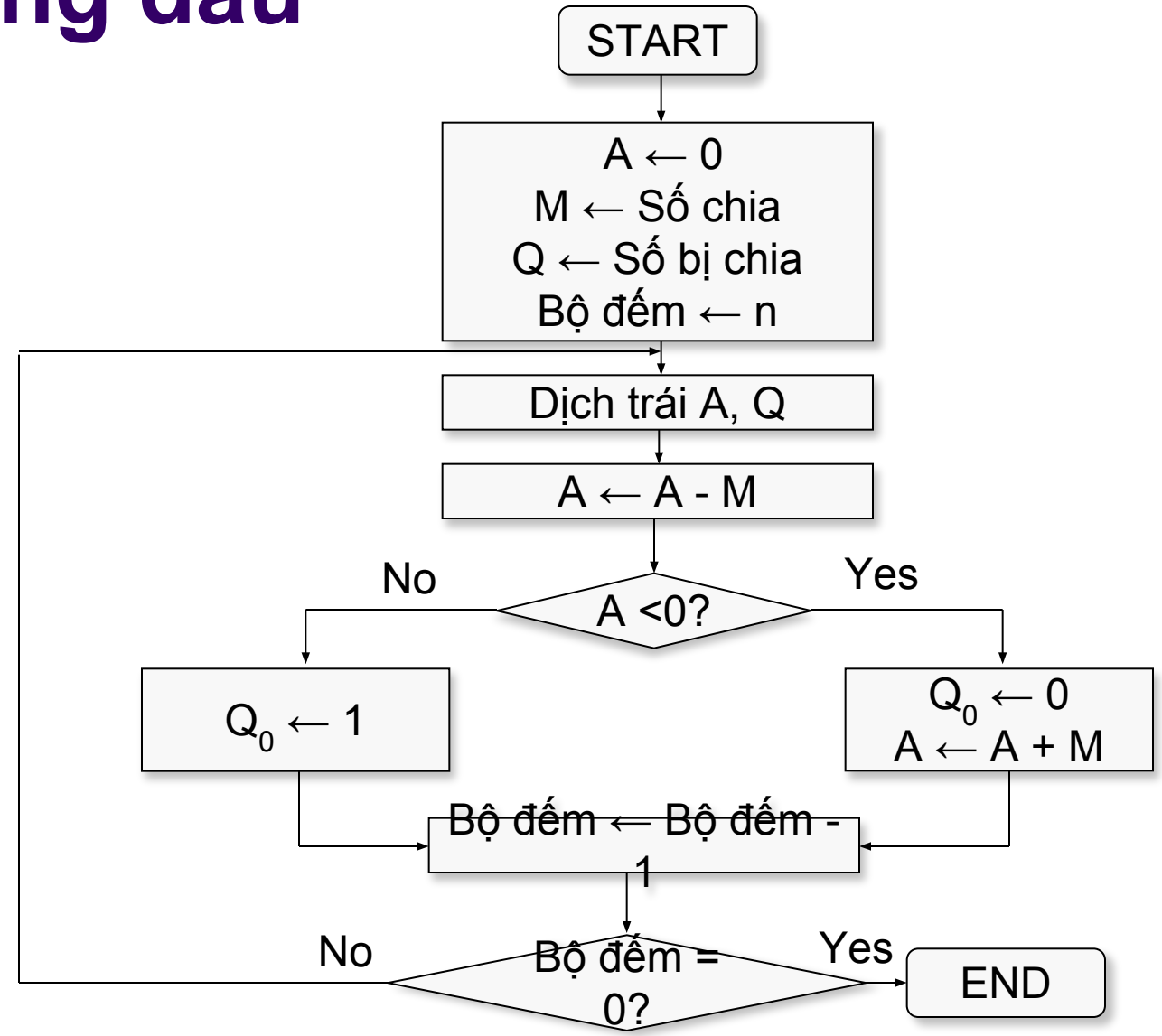
Số bị chia

```
10010011 | 1011
-  1011  | 00001101
001110
 -  1011
  001111
   -  1011
    0100
```

Thương số

Số dư

Giải thuật chia hai số nguyên không dấu



Ví dụ minh họa

A	Q	
0000	0111	Initial value
0000	1110	Shift
1101		Use twos complement of 0011 for subtraction
1101		Subtract
0000	1110	Restore, set $Q_0 = 0$
0001	1100	Shift
1101		
1110		Subtract
0001	1100	Restore, set $Q_0 = 0$
0011	1000	Shift
1101		
0000	1001	Subtract, set $Q_0 = 1$
0001	0010	Shift
1101		
1110		Subtract
0001	0010	Restore, set $Q_0 = 0$

Chia số nguyên có dấu

- Bước 1. Chuyển đổi số bị chia và số chia về thành số dương tương ứng.
- Bước 2. Sử dụng thuật giải chia số nguyên không dấu để chia hai số dương, kết quả nhận được là thương Q và phần dư R đều là dương
- Bước 3. Hiệu chỉnh dấu của kết quả như sau:

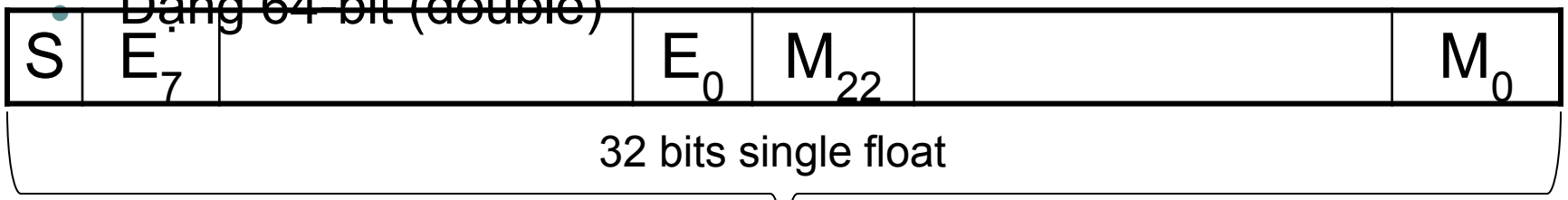
Số bị chia	Số chia	Thương	Số dư
Dương	Dương	Giữ nguyên	Giữ nguyên
Dương	Âm	Đảo dấu	Giữ nguyên
Âm	Dương	Đảo dấu	Đảo dấu
Âm	Âm	Giữ nguyên	Đảo dấu

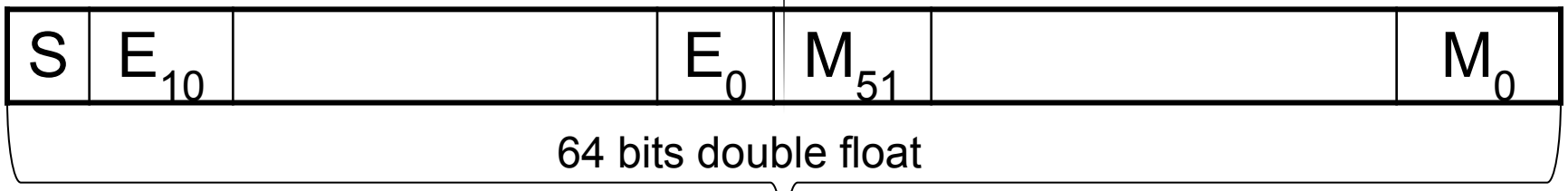
Số dấu phẩy động

- Nguyên tắc chung
  - Floating Point Number → biểu diễn cho số thực
  - Tổng quát: một số thực X được biểu diễn theo kiểu số dấu phẩy động như sau:
    - $X = (-1)^S * M * R^E$
      - S là dấu
      - M là phần định trị (Mantissa),
      - R là cơ số (Radix),
      - E là phần mũ (Exponent).

Chuẩn IEEE754/85

- Cơ số R = 2 $X = (-1)^S * 1.M*(2)^{E-Bias}$
- **Bias = 2^ số bít biểu diễn cho số mũ/2 – 1 = 2^8/2-1 = 127, 2^11/2-1 = 1023**
- Các dạng chính:
  - Dạng 32-bit (single)
  - Dạng 64-bit (double)



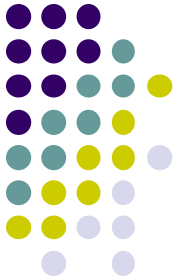


Dạng 32 bits

- S là bít dấu với quy ước sau:
  - 0: là số dương
  - 1: là số âm
- E (8 bit) là phần mũ
- M (23 bit) là phần lẻ của phần định trị:
  - 1.M
- Công thức xác định giá trị của số thực:

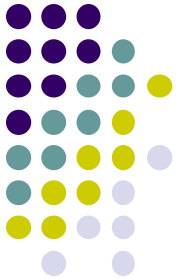
$$X = (-1)^S \times 1.M \times 2^{E-127}$$

Các bước biểu diễn số thực theo chuẩn 32 hoặc 64 bits



- B1: Viết công thức: $\mathbf{R = (-1)^S * 1.M * 2^{E-bias}}$
  Bias = $2^{\text{số bít biểu diễn số mũ}}/2 - 1$
- B2: Viết khuôn dạng: S E M (S là 1 bits S= 1 số âm, S = 0 là số dương, E 8 bit cho mũ, M 23 bits)
- B3: Chuyển số thập phân sang số nhị phân: Chuyển phần nguyên và chuyển phần thập phân sang nhị phân. Ghép phần nguyên và phần thập phân lại.
- B4: Chuẩn hóa: Đưa về dạng công thức
- B5: Xác định: S, E và M
- B6: Lắp vào khuôn dạng.

Ví dụ biểu diễn số 100.125 ở dạng 32 bits



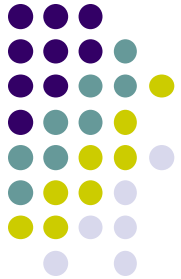
- B1: Viết công thức: $R = (-1)^S * 1.M * 2^{E-bias}$

  Bias = 2^ số bit biểu diễn E/2 -1 = 2^8/2 – 1 = 127

  Số bít E là 8 với số 32 bit, số bit của E là 11 nếu số thực 64 bits
- B2: Viết khuôn dạng: S E M

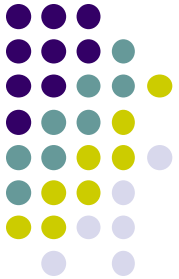
S	E_7…….. E_0	M_{22}……M_0

- B3: Chuyển số cần biểu diễn
  - Phần nguyên: 100 = 2^6+ 2^5+2^2 = 1100100b
  - Phần thập phân:0.125 x 2 = 0.25
  - 0.25 x 2 = 0.5
  - 0.5 x 2 = 1.0
  - 0.125 = 0.001b
  - Kết hợp lại: **100.125 = 1100100.001b**



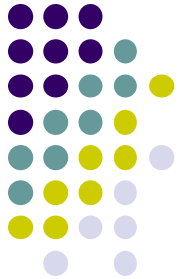
- **Biểu diễn số sau ở dạng dấu chấm động trong máy tính theo chuẩn IEEE 32 bit: –0.125**
- Biểu diễn số sau ở dạng dấu chấm động trong máy tính theo chuẩn IEEE 64 bit:1032.0625

Ví dụ biểu diễn số 100.125 ở dạng 32 bits



- B4: Chuẩn hóa:1100100.001b = $(-1)^0*1.100100001*2^6$
- B5: Xác định S, E và M
  - S = 0
  - E – bias = 6 □ E = 127 + 6 = 133 = 128 + 4 + 1 = 2^7+2^2+2^0 = 10000101b
  - M = 10010000100000000000000b
- B6: Lắp khuôn
- X = 0 100/0010/1 100/1000/0100/0000/0000/0000b
- =42C84000h
- 0000b = 0h, 0001 = 1h,0010 = 2h, 0011 = 3h, 0100 = 4h, 0101 = 5h, 0110 = 6h, 0111 = 7h, 1000 = 8h, 1001 = 9h, 1010 = Ah 1011 = Bh 1100 = C, 1101 = D, 1110 = E 1111 = F

Biểu diễn số dấu chấm động 64 bits



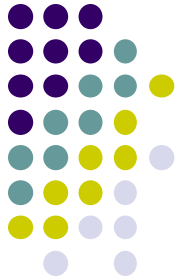
Biểu diễn số sau: 154.75

S: 1 bit

E: 11 bits

M: 52 bits

Biểu diễn số dấu chấm động 64 bits



Biểu diễn số sau: 154.75

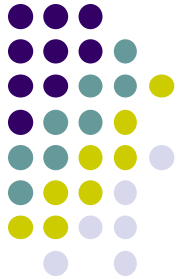
B1: Công thức: $R = (-1)^{S} * 1.M * 2^{E-bias}$

$Bias = 2^{\text{số bít biểu diễn cho E}} / 2 - 1$

B2: Khuôn dạng: S E M (S là 1 bít, E là 11 bít, M là 52 bít).

B3: Biểu diễn số thực sang dạng nhị phân.

Biểu diễn số dấu chấm động 64 bits



Biểu diễn số sau: 154.75

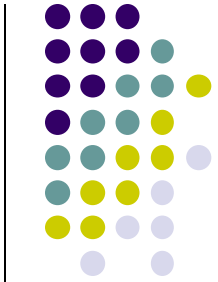
B3: Biểu diễn số thực sang dạng nhị phân.

- Biểu diễn phần nguyên: 154 = ?

154 = 2^7 + 2^4 + 2^3 + 2^1 = 10011010b

- Biểu diễn phần thập phân

Biểu diễn số dấu chấm động 64 bits



Biểu diễn số sau: 154.75

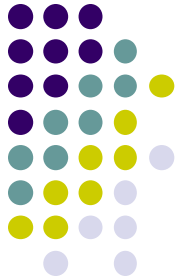
B3:

- Biểu diễn phần thập phân
- 0.75 x 2 = 1.5
- 0.5 x 2 = 1.0

0.75 = 0.11b

Ghép lại: R = 10011010.11b

Biểu diễn số dấu chấm động 64 bits



B4. Đưa R về dạng công thức chuẩn hóa:

$R = (-1)^0 * 1.0011101011 * 2^7$

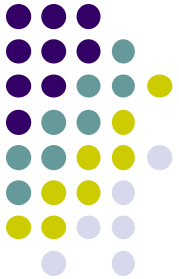
B5: Xác định S, E và M

S = 0

E-bias = 7 E = bias + 7 = 1023 + 7 = 1030 = 1024 + 4 + 2 = 2^10+2^2+2^1 = 10000000110b

M = 0011010110...0

Biểu diễn số dấu chấm động 64 bits

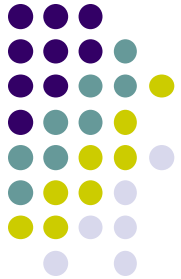


B6: Ghép vào khuôn dạng

R = 0 100/0000/0110/
0011/0101/1000…0b

= 4063580000000000h

Biểu diễn số dấu chấm động 64 bits



B6: Ghép vào khuôn dạng

S E M

1 01111100 **0000000000000000…0**

R = 0 100/0000/0110/
0011/0101/1000…0b

= 4063580000000000h

Ví dụ

- Xác định giá trị số thực được biểu diễn bằng 32 bits sau:
  - X = 1100 0001 0101 0110 0000 0000 0000 0000b
  - Bit dấu S = 1 nên X là số âm.
  - E = $10000010_{(2)}$ = 128 + 2 = 130

→ E -127 = 130 – 127 = 3

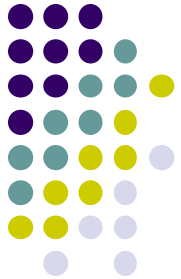
  - 1.M = 1.101011

→ X = - 1.101011×2^3 = -1101.011 = -(13+0*2^-1+1*2^-2+1*2^-3)=-(13+0.25+0.125)=-13.375

Ví dụ (continue)

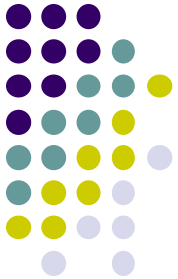
- Biểu diễn X= 83.75 về dạng số dấu phẩy động IEEE754 32-bit
- Giải:
- $X = 83.75_{(10)} = 1010011.11_{(2)} = 1.01001111 \times 2^6$
- Ta có:
  - S = 0 vì đây là số dương
  - $E = e-127 = 6 \rightarrow e = 127 + 6 = 133_{(10)} = 1000\ 0101_{(2)}$
- Vậy:

X = 0100 0010 1010 0111 1000 0000 0000 0000



- Biểu diễn -1012,8 ở dạng 64 bít theo chuẩn IEEE
- Cho biết 409CCCCCH theo chuẩn IEEE 32 bít bằng bao nhiêu hệ 10.

Ví dụ (continue)



- Biểu diễn số thực X= -0,2 về dạng số dấu phẩy động IEEE754 32-bit
- Giải:
- Xác định công thức: $X = (-1)^s \times 1.M \times 2^{E-Bias}$

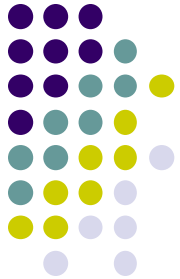
Bias = 2^ số bít biểu diễn E/2 – 1 = 127

Khuông dạng S E M

  - $X = -0,2_{(10)} = -0.00110011...0011..._{(2)} =$
    $= -1.\mathbf{100110011..0011}... \times 2^{-3}$
- Ta có:
  - S = 1 vì đây là số âm
  - $E - 127 = -3 \rightarrow E = 127 - 3 = 124_{(10)} = 0111\ 1100_{(2)}$
  - M= 10011001100110011001100b
- Vậy:

  X = 1011 1110 0**100 1100 1100 1100 1100 1100**

Các quy ước đặc biệt



- Các bit của e bằng 0, các bit của m bằng 0, thì X = ± 0

  S000 0000 0000 0000 0000 0000 0000 0000 → X = ± 0
- Các bit của e bằng 1, các bit của m bằng 0, thì X = ± ∞

  S111 1111 1000 0000 0000 0000 0000 0000 → X =± ∞
- Các bit của e bằng 1, còn m có ít nhất một bit bằng 1, thì nó không biểu diễn cho số nào cả (NaN - not a number)

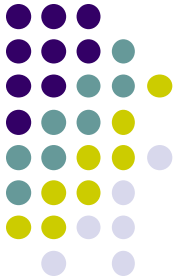
Số thực 64 bits

- S là bit dấu
- e (11 bit): mã *excess-1023* của phần mũ E $\rightarrow$ e = E – 1023 $\rightarrow$ E = 1023 + e
- M (52 bit): phần lẻ của phần định trị ở dạng 1.M
- Giá trị số thực:

$$X = (-1)^S \times 1.M \times 2^{E-1023}$$

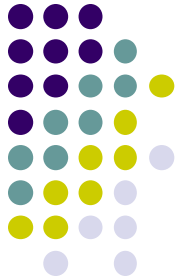
- Dải giá trị biểu diễn: 10^{-308} đến 10^{+308}

Các phép tính số học với dấu phẩy động



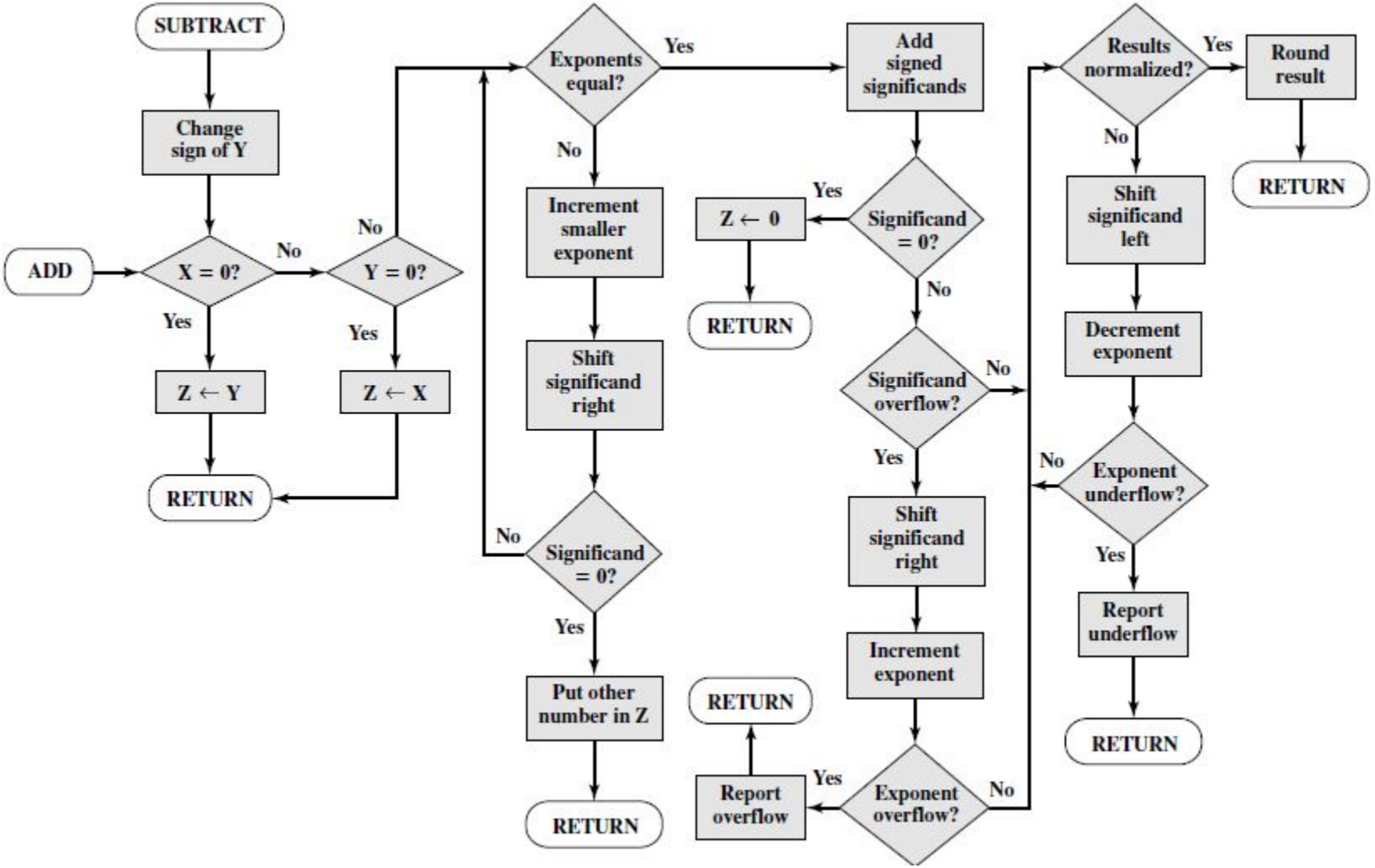
- Giả sử cho hai số dấu phẩy động
  - $X = M1 * R^{E1}$
  - $Y = M2 * R^{E2}$
- Ta có
  - $Z = X * Y = (M1 * M2) * R^{E1+E2}$
  - $Z = X / Y = (M1 / M2) * R^{E1-E2}$
  - $Z = X \pm Y = (M1 * R^{E1-E2} \pm M2) * R^{E2}$, với $E2 \geq E1$

Các khả năng tràn số

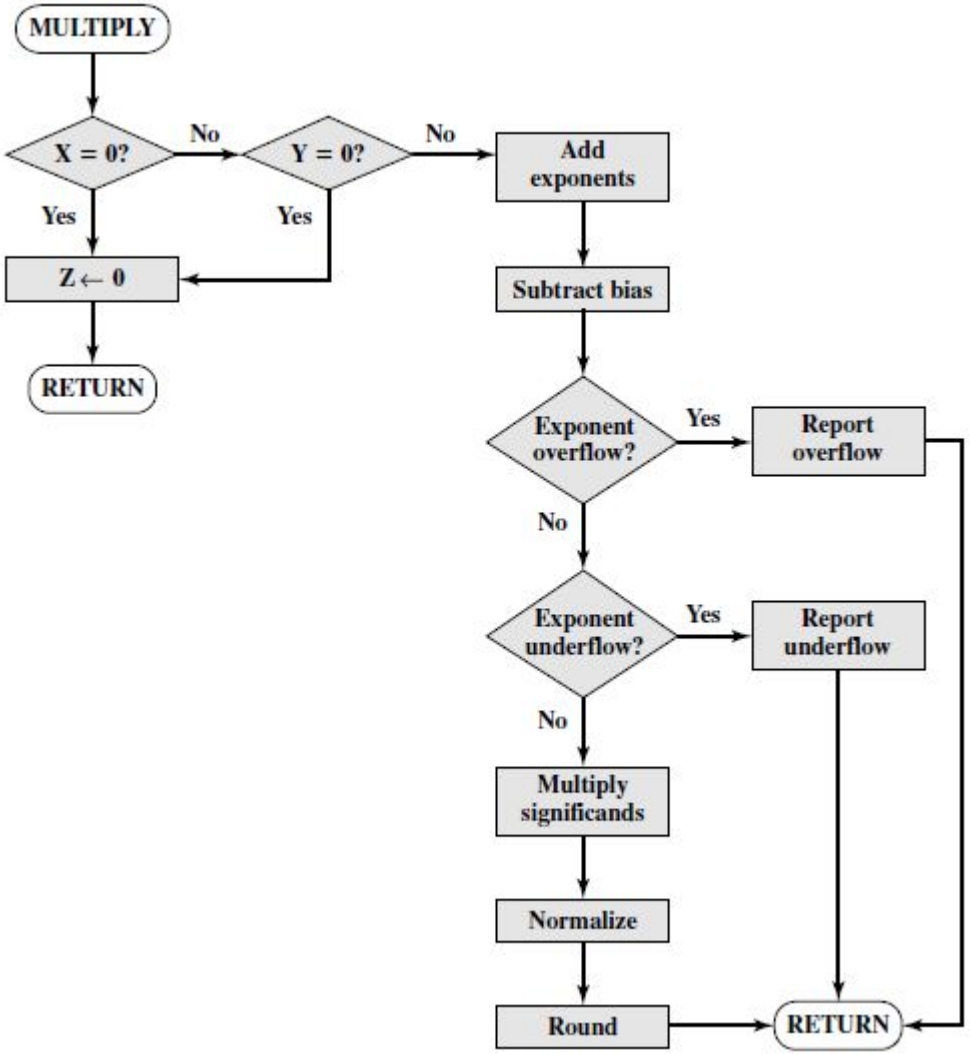


- Tràn trên số mũ (Exponent Overflow): mũ dương vượt ra khỏi giá trị cực đại của số mũ dương có thể. ($\rightarrow \infty$)
- Tràn dưới số mũ (Exponent Underflow): mũ âm vượt ra khỏi giá trị cực đại của số mũ âm có thể ($\rightarrow 0$).
- Tràn trên phần định trị (Mantissa Overflow): cộng hai phần định trị có cùng dấu, kết quả bị nhớ ra ngoài bit cao nhất.
- Tràn dưới phần định trị (Mantissa Underflow): Khi hiệu chỉnh phần định trị, các số bị mất ở bên phải phần định trị.

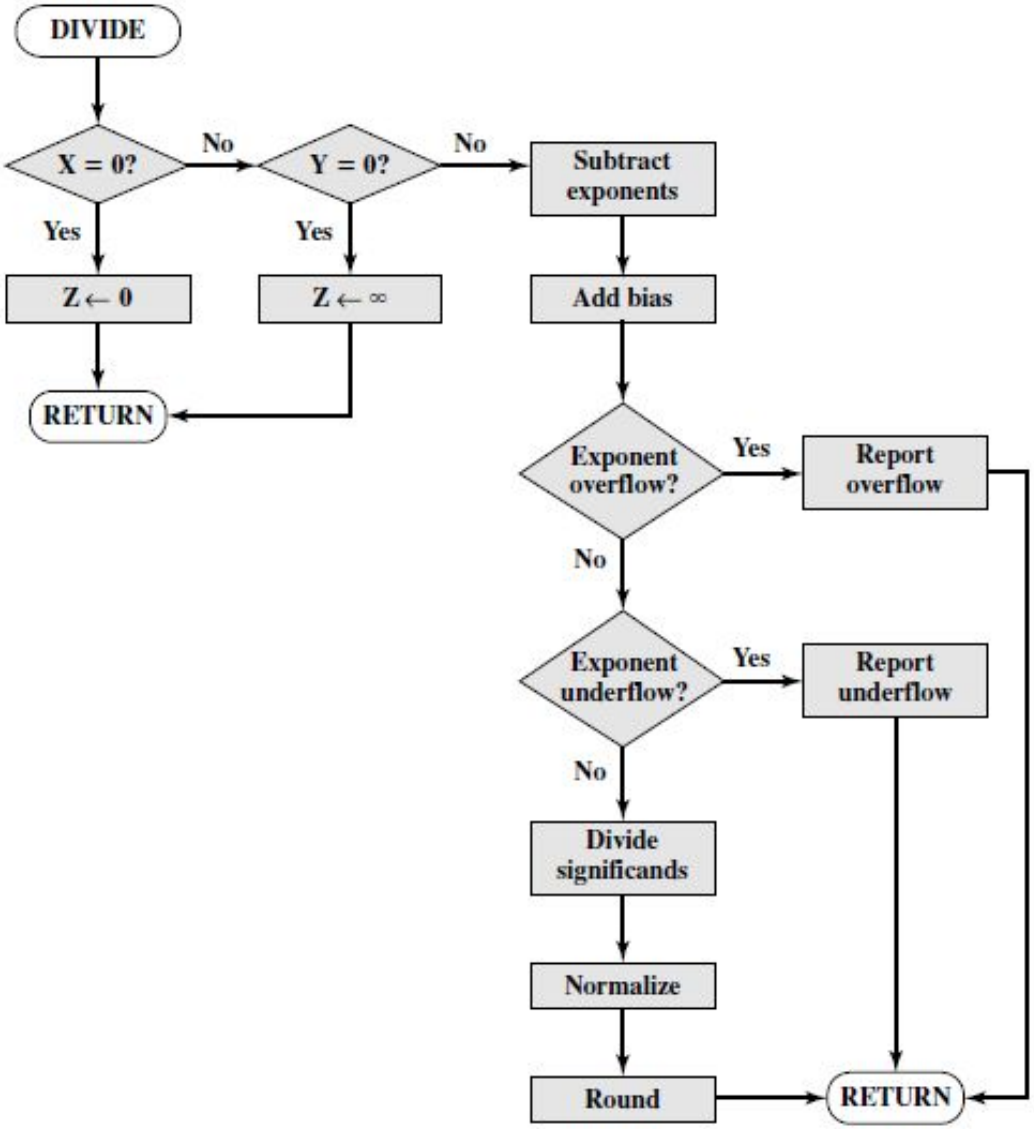
Thuật toán cộng/trừ hai số dấu phẩy động



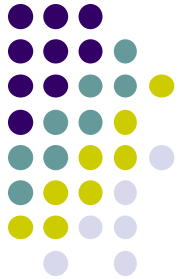
Thuật toán nhân hai số dấu phẩy động



Thuật toán chia hai số dấu phẩy động



Biểu diễn ký tự

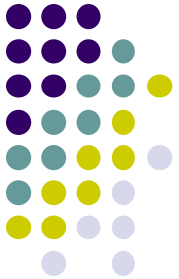


- Bộ mã ASCII (American Standard Code for Information Interchange)
- Bộ mã Unicode

Bộ mã ACSII

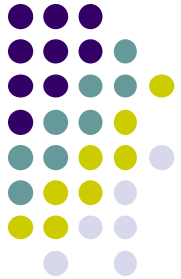
- Do ANSI (American National Standard Institute) thiết kế
- Bộ mã 8-bit → có thể mã hóa được 2^8 ký tự, có mã từ: $00_{16} \div FF_{16}$, trong đó:
  - 128 ký tự chuẩn có mã từ $00_{16} \div 7F_{16}$
  - 128 ký tự mở rộng có mã từ $80_{16} \div FF_{16}$

Bộ mã Unicode



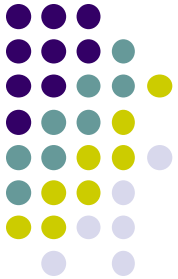
- Do các hãng máy tính hàng đầu thiết kế
- Bộ mã 16-bit
- Bộ mã đa ngôn ngữ
- Có hỗ trợ các ký tự tiếng Việt

Giới thiệu chung về tập lệnh



- Hoạt động của CPU được xác định bởi các lệnh mà nó thi hành được, gọi là các lệnh mã máy (machine instructions).
- Tập hợp các lệnh khác nhau mà CPU có thể thi hành được, gọi là tập lệnh của bộ vi xử lý (instruction set).

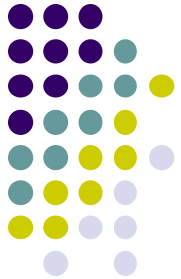
Các thành phần của một câu lệnh mã máy



Mã thao tác	Địa chỉ toán hạng đích	Địa chỉ toán hạng nguồn

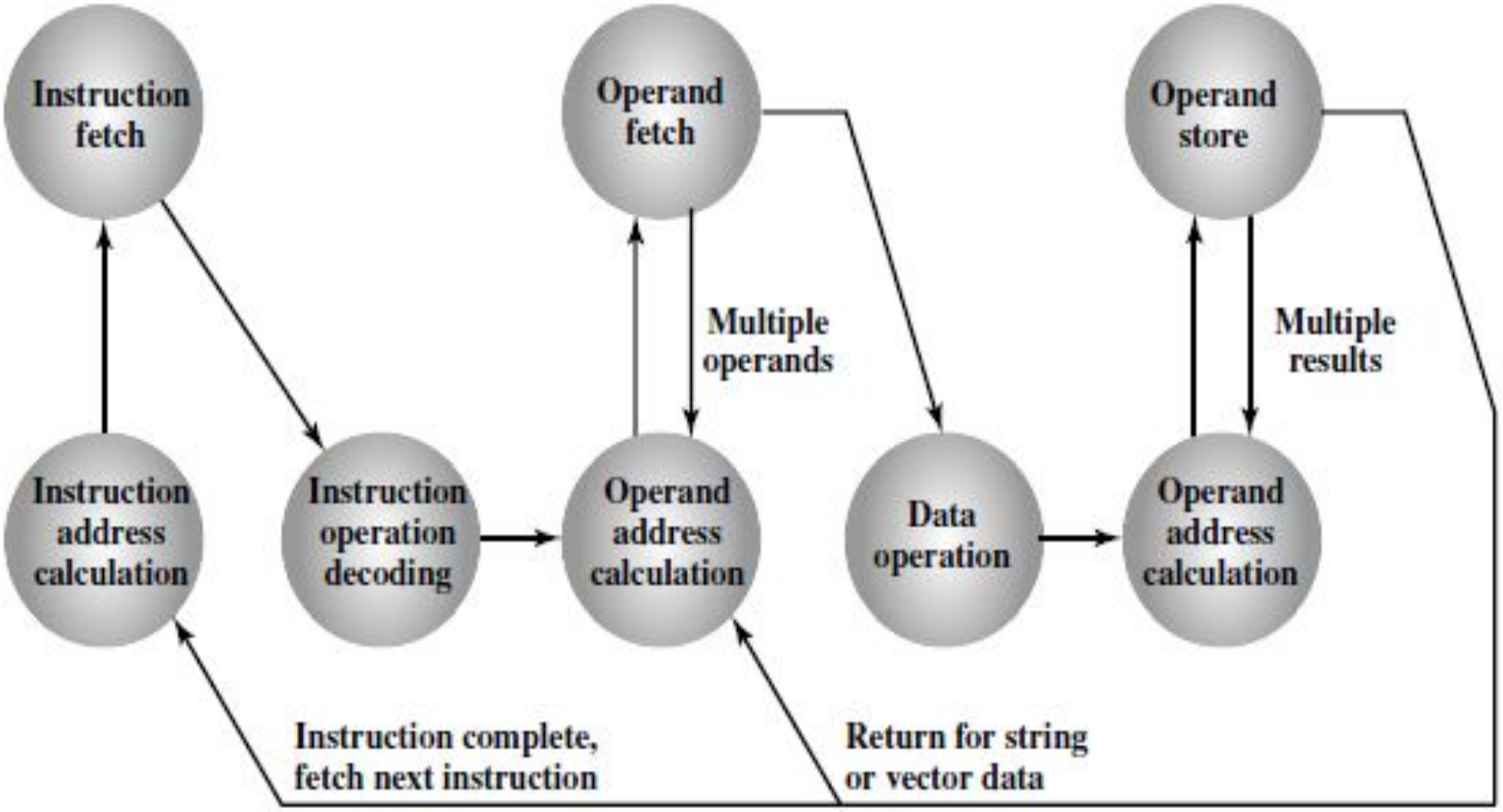
- Mã thao tác (operation code → opcode) là một chuỗi bits mã hóa thao tác mà CPU sẽ phải thực hiện.
- Địa chỉ toán hạng nguồn: Một câu lệnh có thể chứa 1 hoặc một số toán hạng nguồn để làm dữ liệu đầu vào cho thao tác.
- Địa chỉ toán hạng đích: Chỉ ra nơi cất kết quả của thao tác.

Địa chỉ toán hạng nguồn và đích

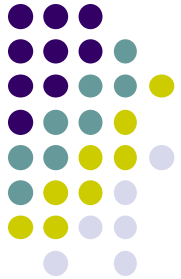


- Địa chỉ ô nhớ của bộ nhớ chính hoặc bộ nhớ ảo.
- Tên thanh ghi
- Địa chỉ trực tiếp: Toán tử của lệnh là hằng số.
- Địa chỉ thiết bị vào/ra.

Chu trình thực hiện một lệnh của CPU

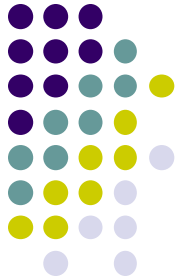


Biểu diễn lệnh



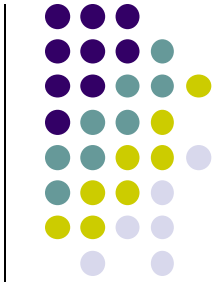
- Lệnh mã máy khó nhớ → sử dụng các ký hiệu gợi nhớ (mnemonic) cho thao tác và các ký hiệu cho toán hạng (ngôn ngữ Assembly).
- Một số ký hiệu gợi nhớ cho thao tác:
  - ADD công
  - SUB trừ
  - MUL nhân
  - DIV chia
  - NEG lấy số đối

Phân loại lệnh



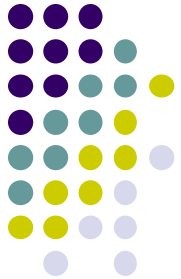
- Xử lý dữ liệu (Data processing): Các lệnh số học và logic.
- Lưu dữ liệu (Data storage): Lấy hoặc cất dữ liệu vào thanh ghi hoặc ô nhớ.
- Di chuyển dữ liệu (Data movement): Các lệnh vào/ra dữ liệu.
- Điều khiển (control): Các lệnh kiểm tra và rẽ nhánh.

Số lượng địa chỉ toán hạng trong một lệnh



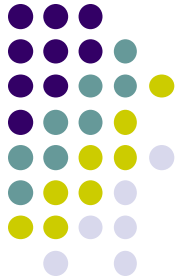
- 3 địa chỉ toán hạng
- 2 địa chỉ toán hạng
- 1 địa chỉ toán hạng

Ba địa chỉ toán hạng



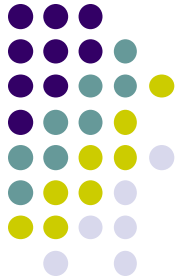
- Hai địa chỉ toán hạng nguồn và một địa chỉ toán hạng đích.
- Từ lệnh dài vì phải mã hóa cho cả 3 toán hạng.

Hai địa chỉ toán hạng



- Một toán hạng nguồn và một toán hạng đích.
- Nếu toán hạng nguồn vừa đóng vai trò là toán hạng đích thì dữ liệu cũ của toán hạng đích sẽ bị mất.
- Ví dụ: ADD AX,BX

Một địa chỉ toán hạng



- Thao tác một toán hạng

  Ví dụ: NEG BX

- Đối với các phép toán hai ngôi thì toán hạng đích ngầm định là thanh ghi chứa.

  Ví dụ: LOAD D AC ← D

- Thường được sử dụng trên các máy tính thế hệ cũ.

Minh họa

Thực hiển biểu thức: $Y = (A - B)/[C + D \times E]$

Instruction		Comment
SUB	Y, A, B	Y ← A − B
MPY	T, D, E	T ← D × E
ADD	T, T, C	T ← T + C
DIV	Y, Y, T	Y ← Y ÷ T

(a) Three-address instructions

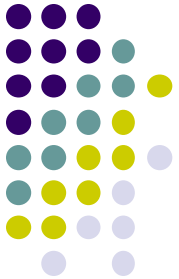
Instruction		Comment
MOVE	Y, A	Y ← A
SUB	Y, B	Y ← Y − B
MOVE	T, D	T ← D
MPY	T, E	T ← T × E
ADD	T, C	T ← T + C
DIV	Y, T	Y ← Y ÷ T

(b) Two-address instructions

Instruction		Comment
LOAD	D	AC ← D
MPY	E	AC ← AC × E
ADD	C	AC ← AC + C
STOR	Y	Y ← AC
LOAD	A	AC ← A
SUB	B	AC ← AC − B
DIV	Y	AC ← AC ÷ Y
STOR	Y	Y ← AC

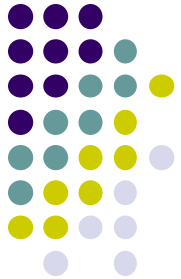
(c) One-address instructions

Thiết kế tập lệnh



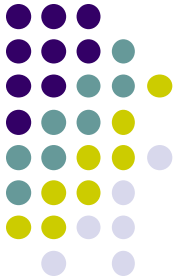
- Về thao tác
  - Bao nhiêu thao tác?
  - Các thao tác nào?
  - Mức độ phức tạp của các thao tác ?
- Các kiểu dữ liệu mà lệnh có thể xử lý
- Các khuôn dạng lệnh
  - Độ dài của trường mã thao tác
  - Số lượng địa chỉ toán hạng

Thiết kế tập lệnh (tiếp)



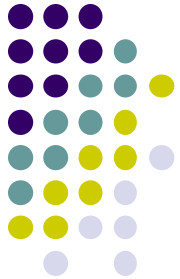
- Thanh ghi: Số lượng thanh ghi của CPU và chức năng của chúng.
- Các chế độ định địa chỉ: Cách xác định địa chỉ của toán hạng.

Các kiểu thao tác cơ bản



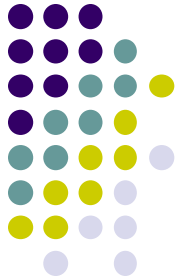
- Di chuyển dữ liệu
- Phép toán số học
- Phép toán logic
- Điều khiển vào/ra
- Chuyển điều khiển
- Điều khiển hệ thống

Các lệnh di chuyển dữ liệu



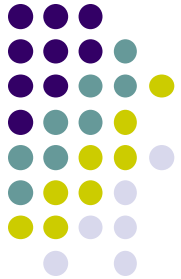
- MOVE di chuyển dữ liệu từ nguồn đến đích
- STORE cất dữ liệu từ thanh ghi của CPU ra bộ nhớ.
- LOAD (FETCH) nạp dữ liệu từ bộ nhớ vào thanh ghi của CPU.
- EXCHANGE hoán đổi dữ liệu giữa nguồn và đích.
- CLEAR chuyển các bit 0 vào toán hạng đích.
- SET chuyển các bit 1 vào toán hạng đích.
- PUSH cất toán hạng nguồn vào đỉnh ngăn xếp
- POP nạp đỉnh ngăn xếp vào toán hạng đích.

Các thao tác số học



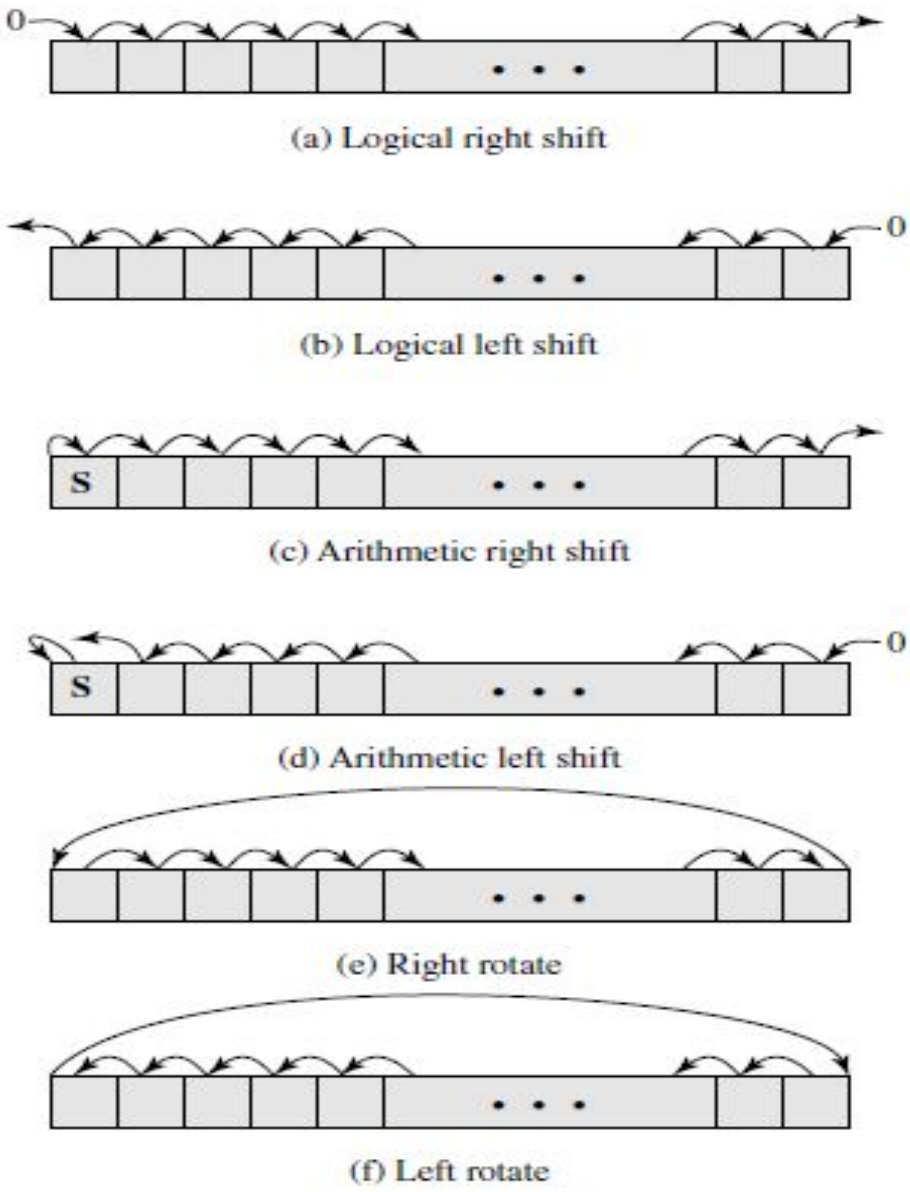
- ADD cộng hai toán hạng
- SUBTRACT trừ hai toán hạng
- MULTIPLY nhân hai toán hạng
- DIVIDE chia hai toán hạng
- ABSOLUTE tính trị tuyệt đối của toán hạng
- NEGATIVE đảo dấu toán hạng
- INCREMENT tăng toán hạng lên 1
- DECREMENT giảm toán hạng đi 1

Các thao tác logic, dịch và quay

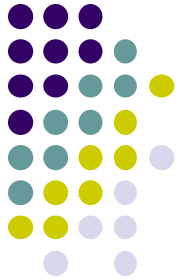


- **AND** thực hiện phép AND logic hai toán hạng.
- **OR** thực hiện phép OR logic hai toán hạng.
- **NOT** thực hiện phép tính phủ định.
- **XOR** thực hiện phép XOR hai toán hạng.
- **TEST** thực hiện phép AND để thiết lập các cờ.
- **COMPARE** thực hiện phép trừ để thiết lập các cờ.
- **SHIFT** dịch trái hoặc dịch phải toán hạng đích
- **ROTATE** quay trái hoặc quay phải toán hạng đích.

Minh họa toán hạng dịch và quay

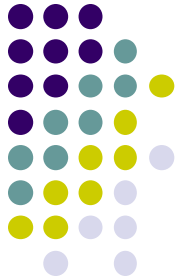


Các thao tác vào/ra



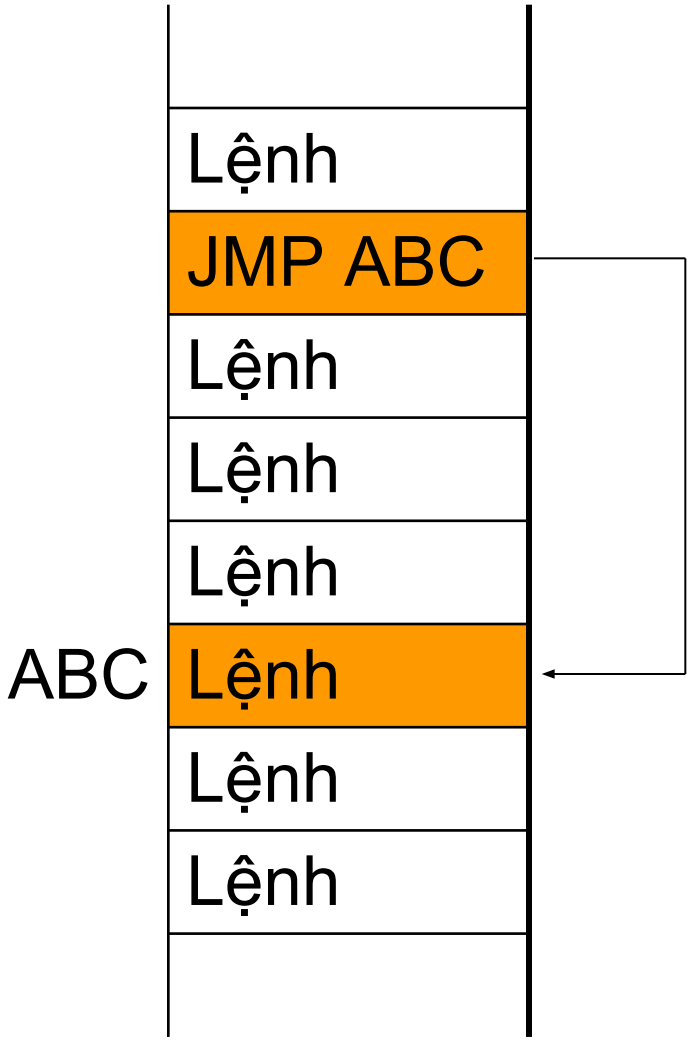
- INPUT di chuyển dữ liệu từ cổng vào/ra vào bộ nhớ hoặc thanh ghi của CPU.
- OUT di chuyển dữ liệu từ nguồn đến cổng vào/ra.

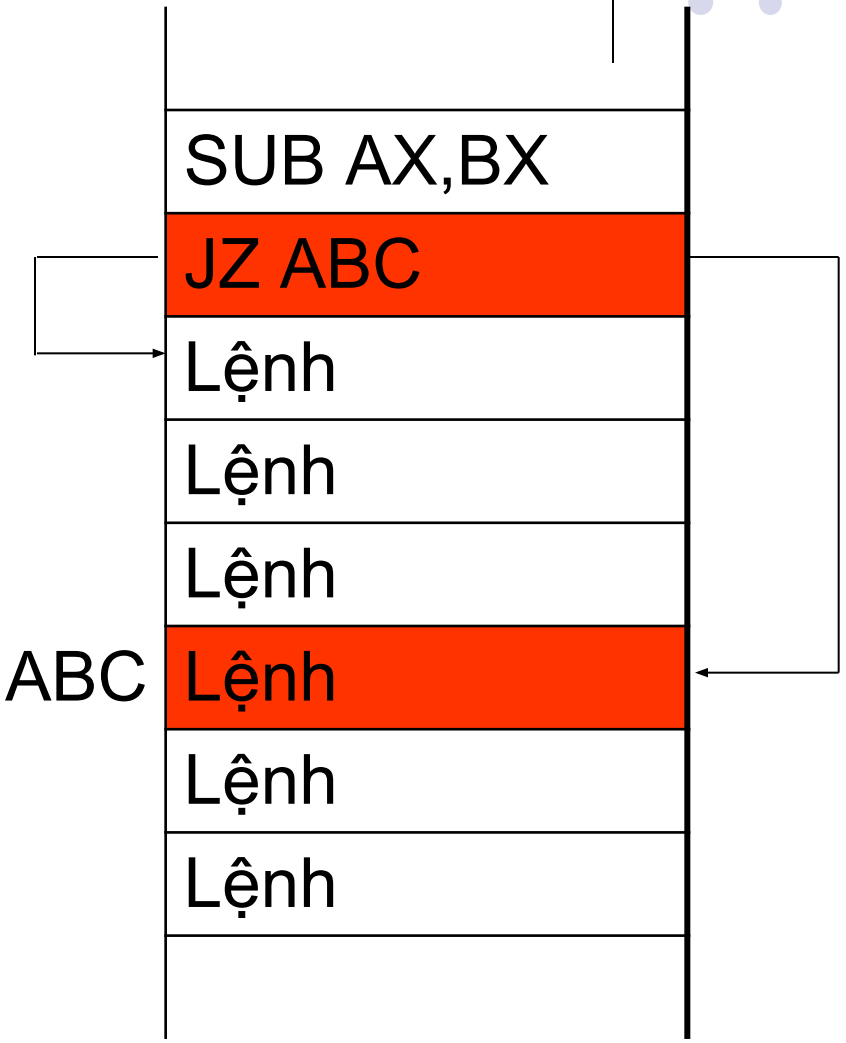
Các thao tác chuyển điều khiển



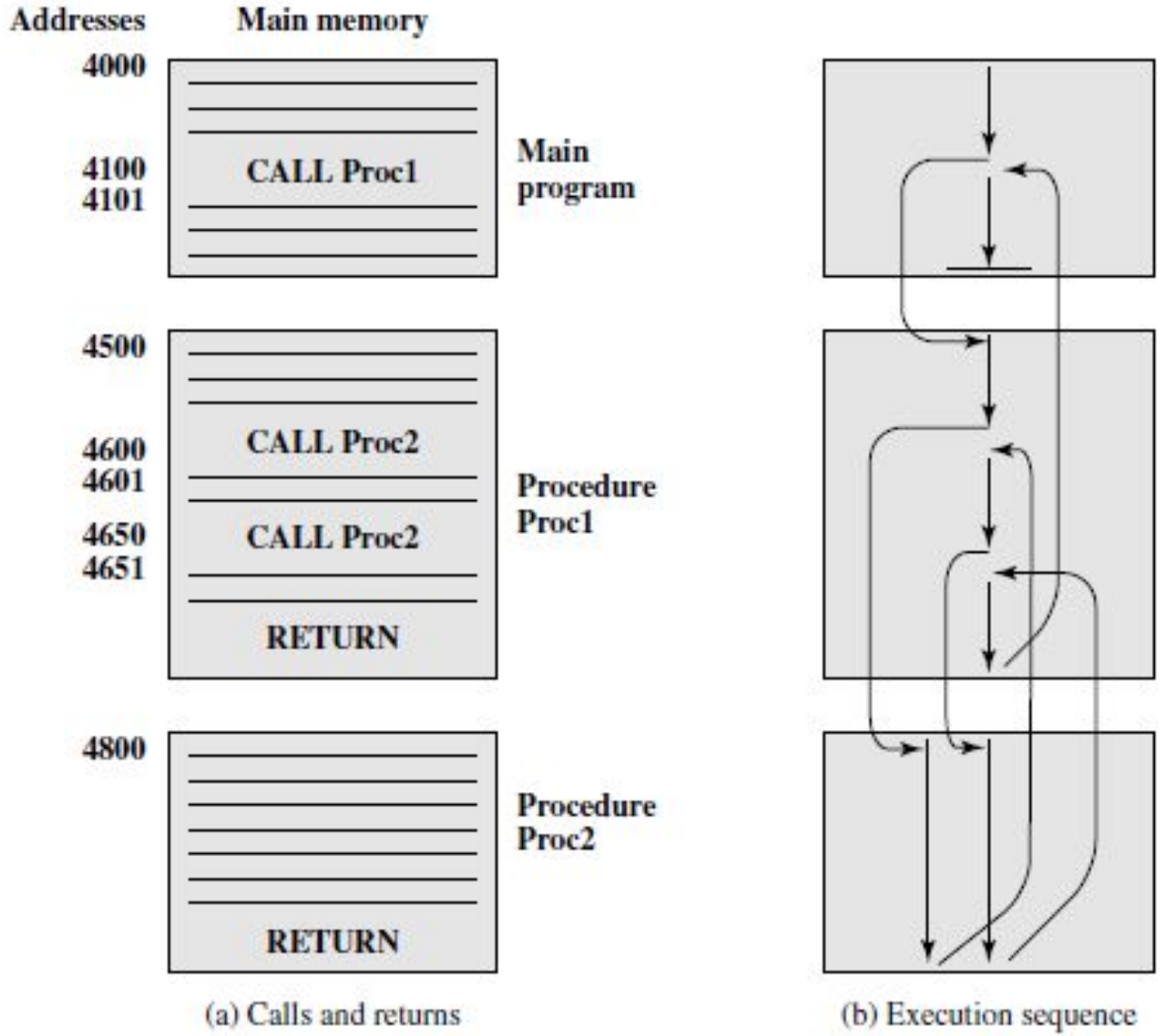
- JUMP (BRANCH) Lệnh nhảy không điều kiện:
  - nạp vào PC một địa chỉ xác định
- JUMP CONDITIONAL Lệnh nhảy có điều kiện:
  - điều kiện đúng thì nạp vào PC một địa chỉ xác định
  - điều kiện sai thì thực hiện lệnh kế tiếp
- CALL Lệnh gọi chương trình con:
  - Cất nội dung của PC (địa chỉ trở về) ra một vị trí xác định (thường ở Stack)
  - Nạp vào PC địa chỉ của lệnh đầu tiên của chương trình con
- RETURN Lệnh trở về từ chương trình con:
  - Khôi phục địa chỉ trở về trả lại cho PC để trở về chương trình chính.

Minh họa các lệnh chuyển điều khiển

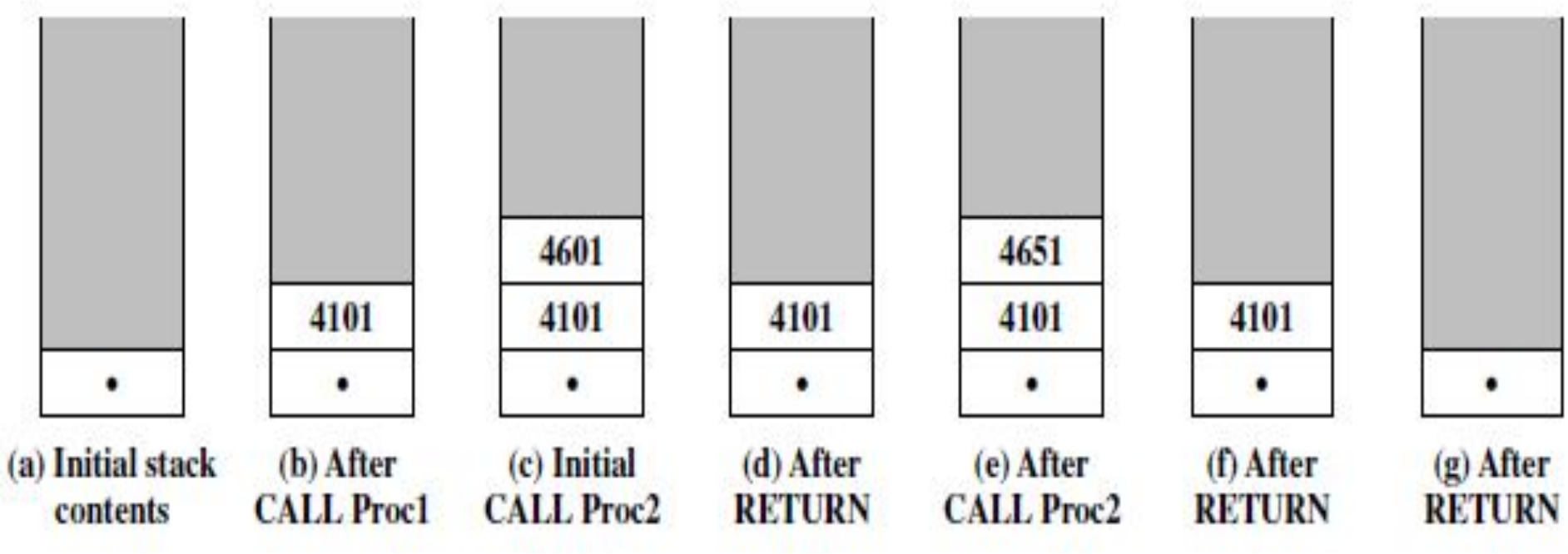




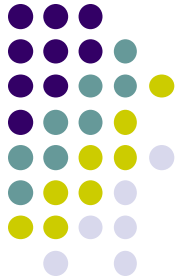
Minh họa thao tác gọi chương trình con (CTC) và trở về từ CTC



Mô tả ngăn xếp cho thao tác gọi và trở về từ CTC

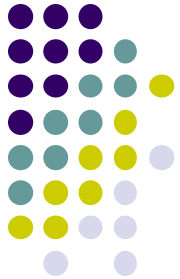


Các thao tác điều khiển hệ thống



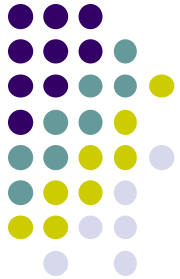
- **HALT** Dừng thực hiện chương trình.
- **WAIT** Tạm dừng thực hiện chương trình, lặp kiểm tra điều kiện cho đến khi thoả mãn thì tiếp tục thực hiện.
- **NO OPERATION** Không thực hiện gì cả.
- **LOCK** Cấm không cho xin chuyển nhượng bus.
- **UNLOCK** Cho phép xin chuyển nhượng bus.

Các chế độ địa chỉ của CPU



- Chế độ địa chỉ (Addressing mode) là các cách khác nhau để xác định ví trị của toán hạng trong các câu lệnh.
  - Chế độ địa chỉ tức thì (Immediate)
  - Chế độ địa chỉ trực tiếp (Direct)
  - Chế độ địa chỉ gián tiếp (Indirect)
  - Chế độ địa chỉ thanh ghi (Register)
  - Chế độ địa chỉ gián tiếp thanh ghi (Register indirect)
  - Chế độ địa chỉ dịch chuyển (Displacement)

Chế độ địa chỉ trực tiếp

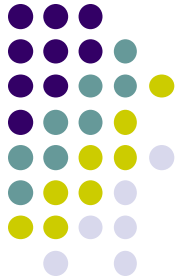


- Giá trị của toán hạng được chỉ ra ngay bên trong câu lệnh (hằng số).
- Toán hạng được ngầm định trong câu lệnh.
- Ví dụ: Add 5 → Acc ← Acc + 5

opcode	operand

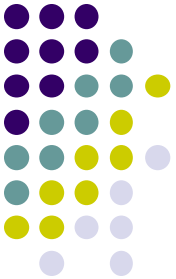
- Đặc điểm:
  - Nhanh vì toán hạng có sẵn trong lệnh
  - Phạm vi giá trị của toán hạng bị hạn chế vì độ dài dành cho toán hạng là hạn chế.

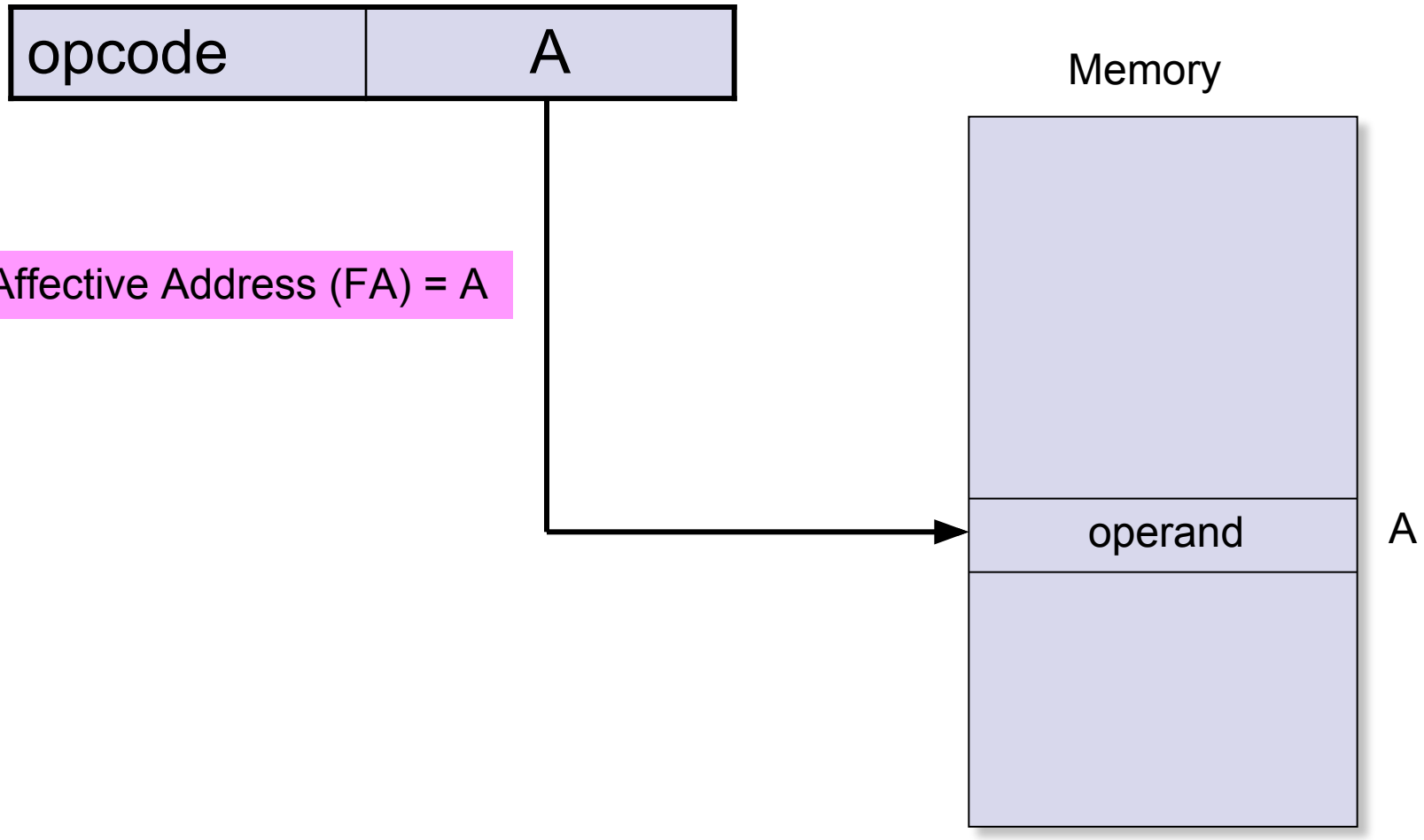
Chế độ địa chỉ trực tiếp



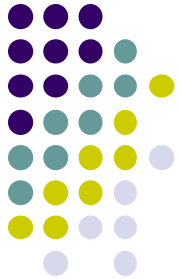
- Trường địa chỉ trong lệnh chứa địa chỉ của toán hạng nằm trong bộ nhớ.
- Ví dụ: Add A Acc ← Acc + [A]
- Đặc điểm:
  - CPU cần tham chiếu bộ nhớ một lần để truy cập dữ liệu.
  - Không cần tính toán địa chỉ của toán hạng.
  - Hạn chế không gian địa chỉ.

Minh họa chế độ địa chỉ trực tiếp



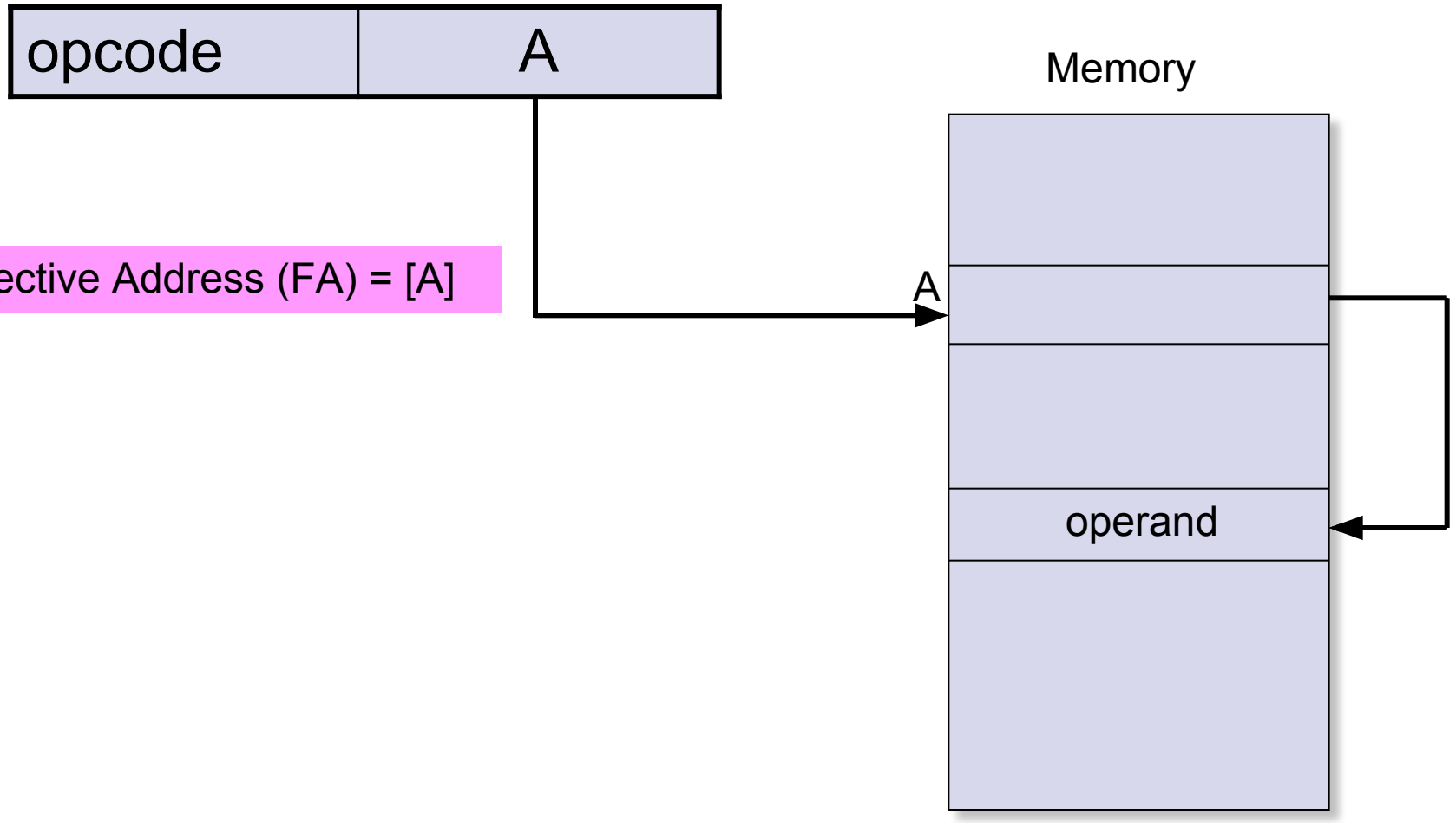


Chế độ địa chỉ gián tiếp

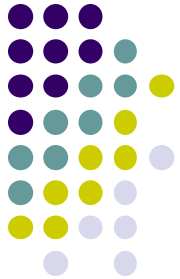


- Trường địa chỉ của lệnh chứa địa chỉ của ô nhớ mà nội dung của ô nhớ này mới là địa chỉ của toán hạng.
- Đặc điểm:
  - CPU phải tham chiếu bộ nhớ nhiều lần mới truy cập được toán hạng → chậm
  - Vùng nhớ được tham chiếu là lớn.

Minh họa chế độ địa chỉ gián tiếp bộ nhớ

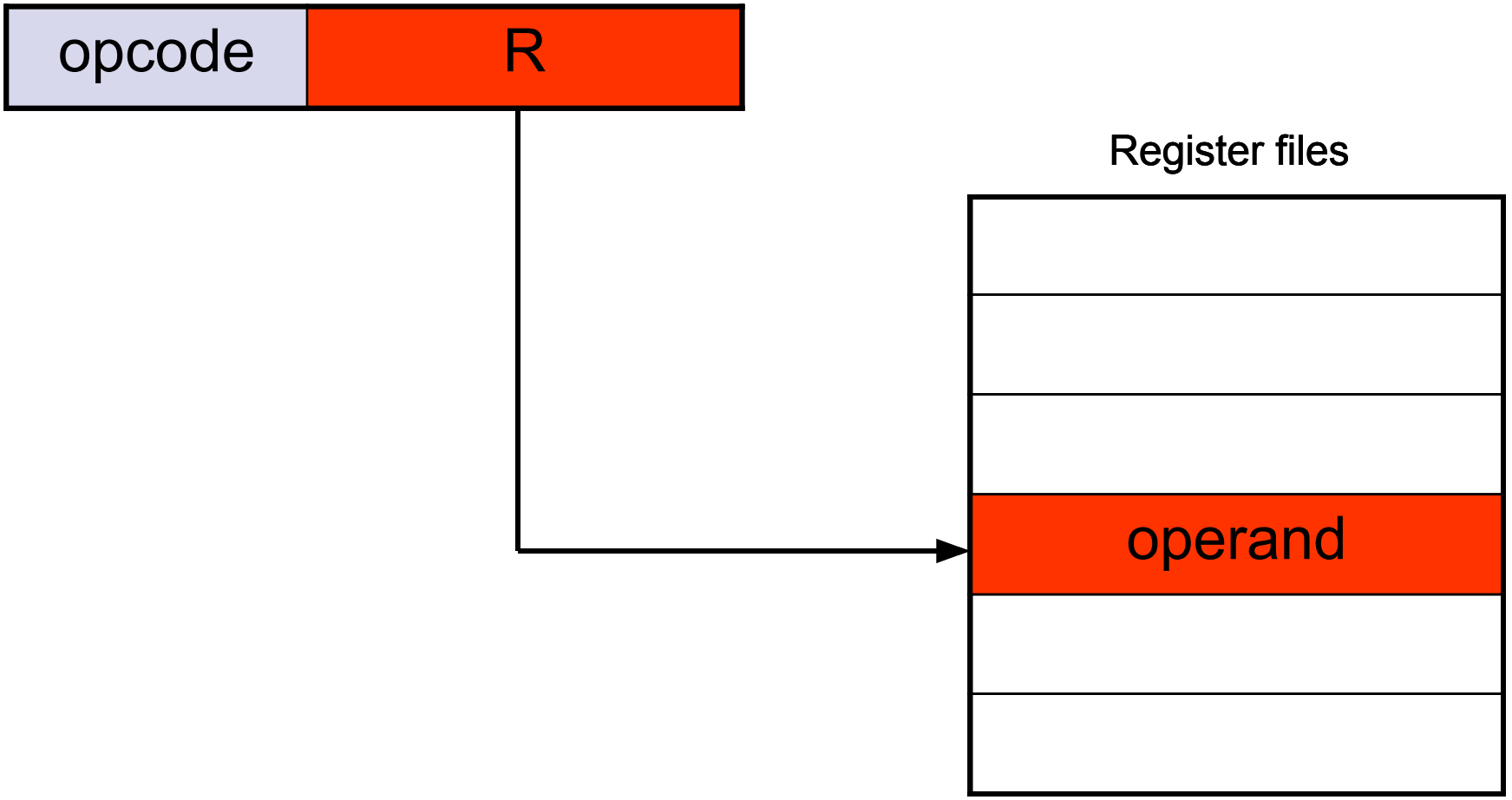


Chế độ địa chỉ thanh ghi

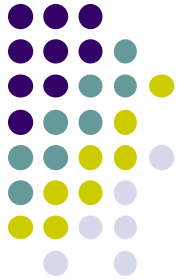


- Toán hạng được lưu trong thanh ghi mà tên của thanh ghi này được chỉ ra bởi trường địa chỉ trong lệnh.
- EA = R
- Số lượng thanh ghi hạn chế → cần it bit cho trường địa chỉ → chiều dài của câu lệnh giảm → nạp lệnh nhanh.
- Không cần truy cập bộ nhớ
- Thực hiện lệnh nhanh
- Không gian địa chỉ rất hạn chế

Minh họa chế độ địa chỉ thanh ghi

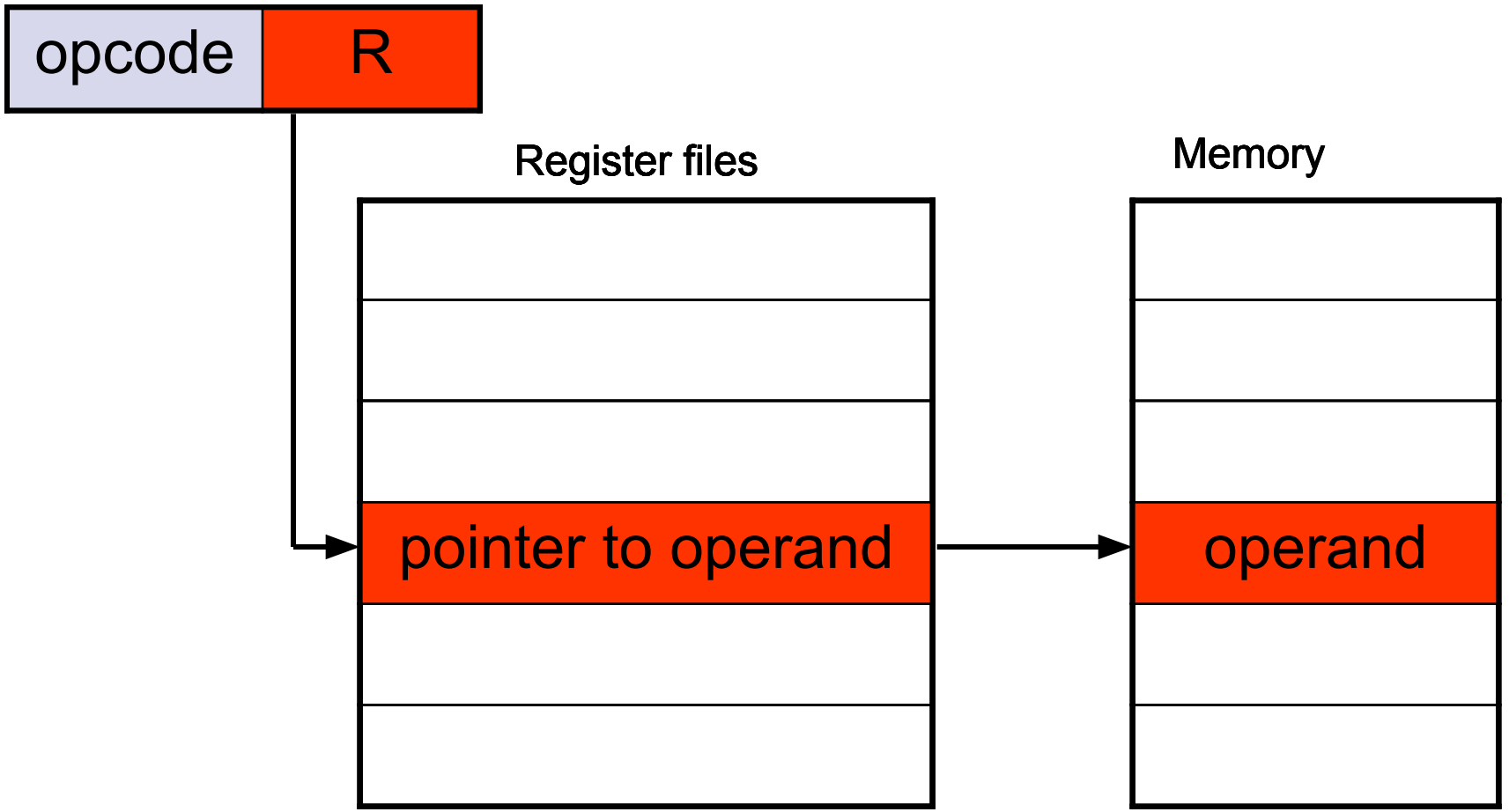


Chế độ địa chỉ gián tiếp thanh ghi

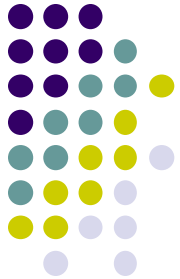


- Địa chỉ của toán hạng trong bộ nhớ được trỏ bởi thanh ghi R.
- EA = [R]
- Không gian địa chỉ lớn (2^n)
- Số lần tham chiếu đến bộ nhớ nhỏ hơn 1 lần so với chế độ địa chỉ gián tiếp bộ nhớ.

Minh họa chế độ địa chỉ gián tiếp thanh ghi

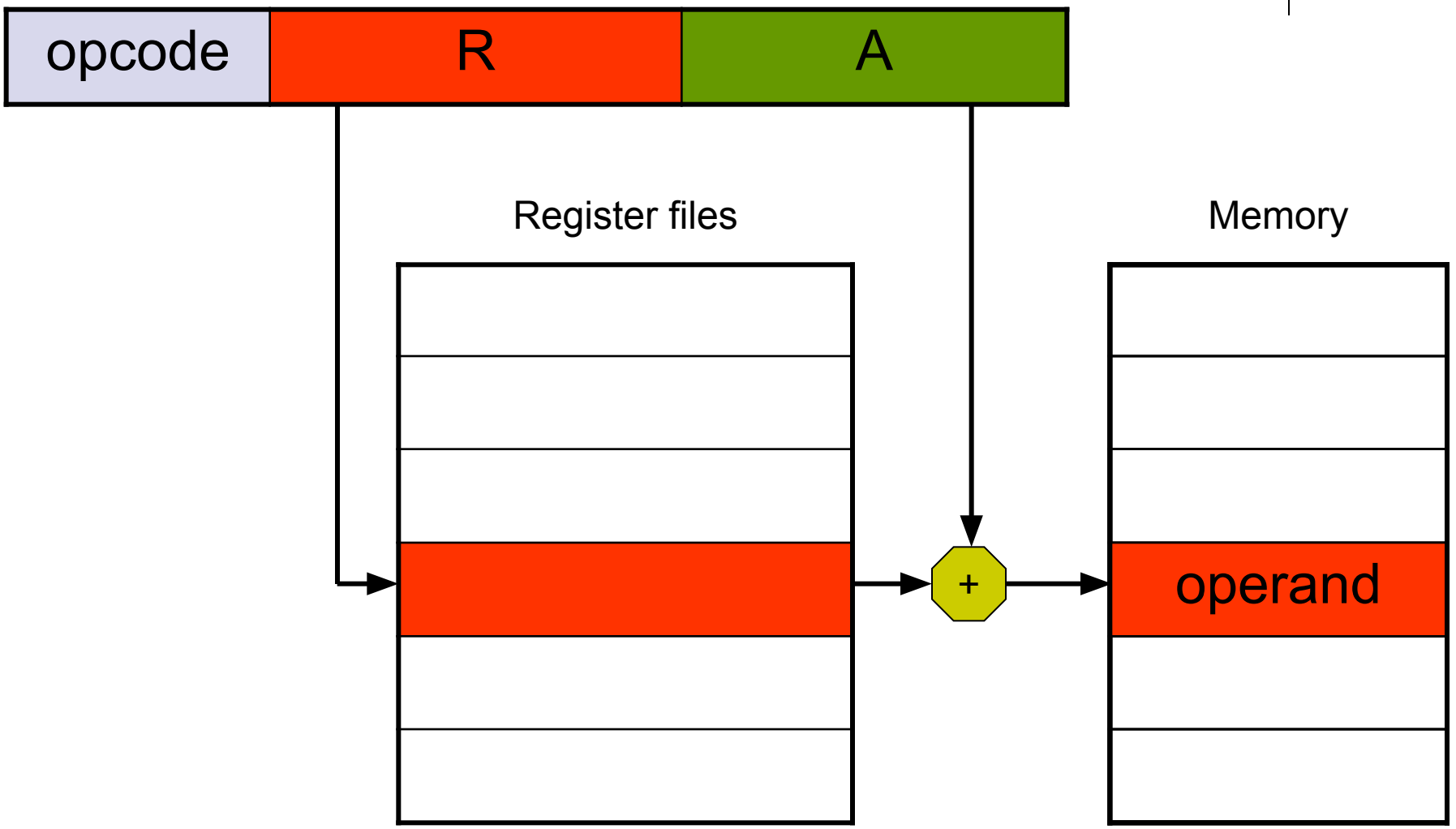


Chế độ địa chỉ dịch chuyển

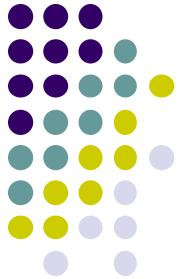


- Địa chỉ của toán hạng được tính bằng nội dung của thanh ghi cộng với một hằng số.
- EA = [R] + A
- Trường địa chỉ của lệnh phải có 2 thành phần
  - Tên thanh ghi
  - Hằng số
- Có 3 dạng địa chỉ dịch chuyển thông dụng là:
  - Địa chỉ tương đối (relative addressing)
  - Địa chỉ thanh ghi cơ sở (base register addressing)
  - Địa chỉ chỉ số (index addressing)

Minh họa chế độ địa chỉ dịch chuyển



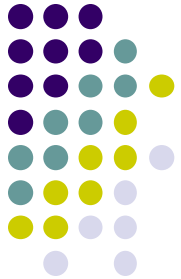
Chế độ địa chỉ tương đối



- Sử dụng ngầm định thanh ghi đếm chương trình PC (Program Counter) trong chế độ địa chỉ dịch.

→ EA = [PC] + A

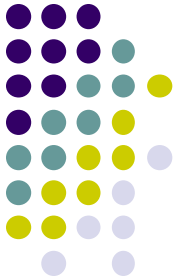
Chế độ địa chỉ cơ sở, chỉ số



- Địa chỉ cơ sở: Thanh ghi chứa địa chỉ đầu đoạn (segment) – địa chỉ cơ sở trong bộ nhớ chính. Hằng số chính là độ dịch (displacement) tính từ địa chỉ đầu đoạn.
- Địa chỉ chỉ số: Hằng số là địa chỉ đầu đoạn. Thanh ghi chứa đị chỉ dịch.

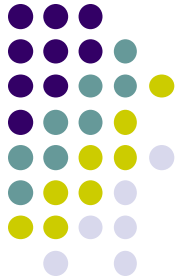
$$EA = A + [R]$$

Tập thanh ghi của CPU



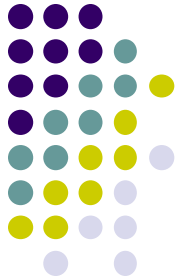
- **User-visible registers:** Thanh ghi mà người lập trình bằng ngôn ngữ máy hoặc hợp ngữ có thể sử dụng để giảm tối đa việc tham chiếu bộ nhớ thông qua việc sử dụng tối ưu các thanh ghi này.
- **Control and status registers:** Được sử dụng bởi đơn vị điều khiển để điều khiển hoạt động của CPU.

User-Visible Registers



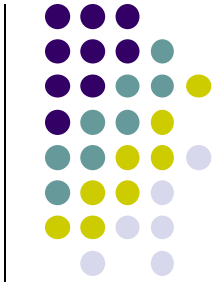
- Thanh ghi công dụng chung (general purpose register)
- Thanh ghi dữ liệu (data register)
- Thanh ghi địa chỉ (address register)
- Thanh ghi trạng thái (status register)

Thanh ghi công dụng chung



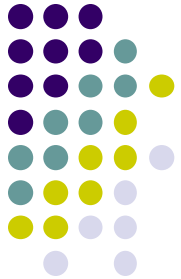
- Có thể dụng để chứa toán tử cho các phép toán số học và logic và cũng có thể dùng để chứa địa chỉ của ô nhớ trong một số chế độ địa chỉ.
- Ví dụ thanh ghi BX, SI, DI

Thanh ghi địa chỉ



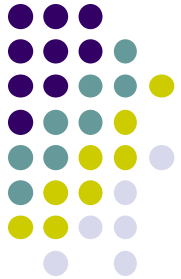
- Thanh ghi đoạn (segment registers)
- Thanh ghi chỉ số (index register)
- Thanh ghi con trỏ (pointer register)

Các thanh ghi điều khiển và trạng thái của CPU



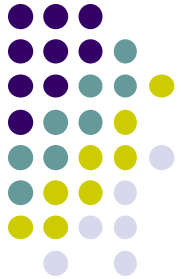
- **Program counter (PC):** Chứa địa chỉ của lệnh sẽ được nạp vào thanh ghi lệnh.
- **Instruction register (IR):** Chứa lệnh vừa được nạp.
- **Memory address register (MAR):** Chứa địa chỉ của ô nhớ sẽ được đọc hoặc ghi.
- **Memory buffer register (MBR):** Chữa dữ liệu sẽ được ghi ra bộ nhớ hoặc dữ liệu vừa được đọc vào từ bộ nhớ.

Một số bit điển hình của thanh ghi trạng thái (thanh ghi cờ)



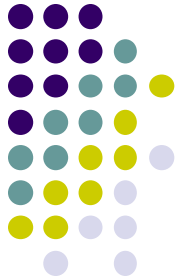
- Cờ Zero (cờ rỗng): được thiết lập lên 1 khi kết quả của phép toán bằng 0.
- Cờ Sign (cờ dấu): được thiết lập lên 1 khi kết quả phép toán nhỏ hơn 0
- Cờ Carry (cờ nhớ): được thiết lập lên 1 nếu phép toán có nhớ ra ngoài bit cao nhất → cờ báo tràn với số không dấu.
- Cờ Overflow (cờ tràn): được thiết lập lên 1 nếu cộng hai số nguyên cùng dấu mà kết quả có dấu ngược lại → cờ báo tràn với số có dấu .

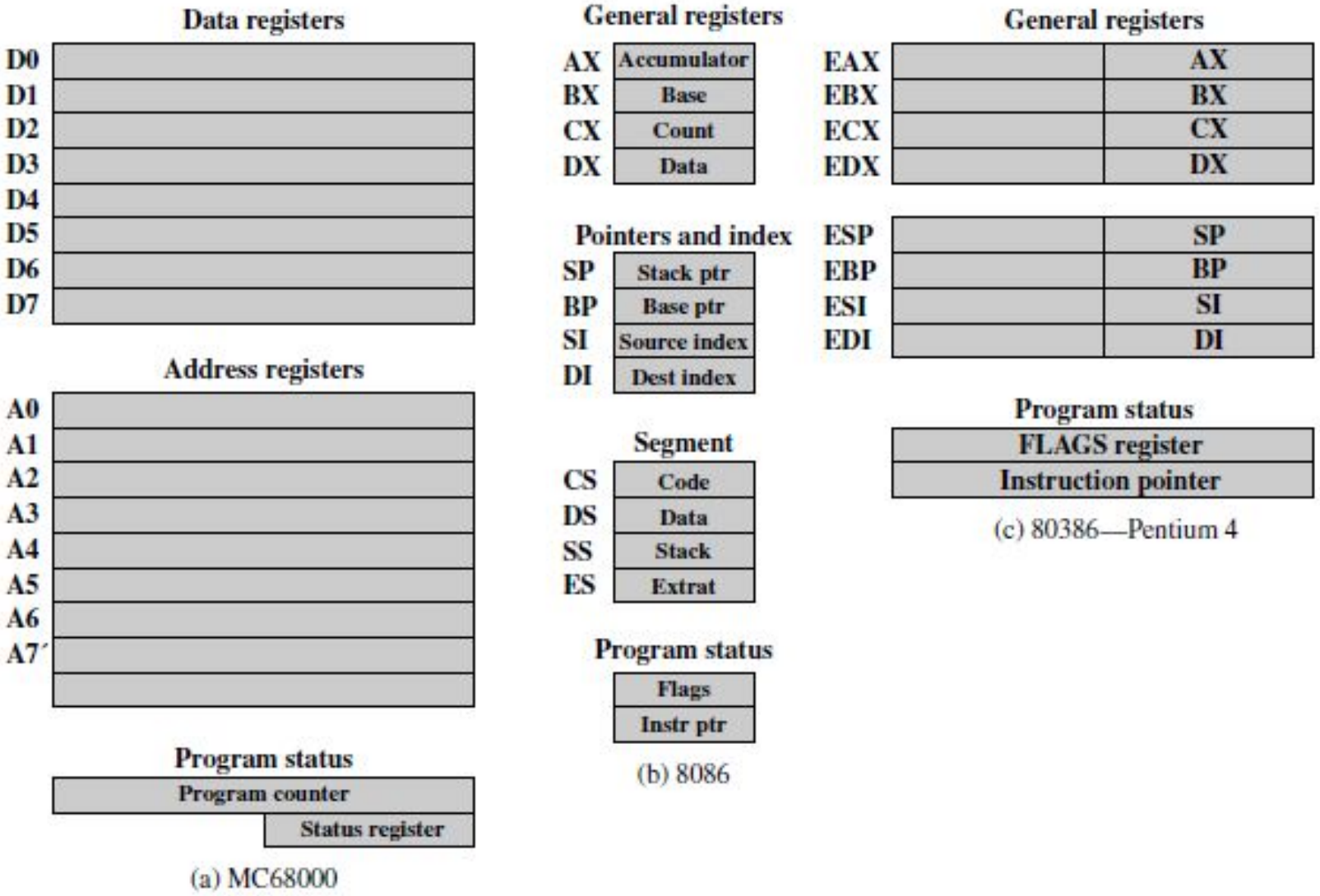
Cờ điều khiển



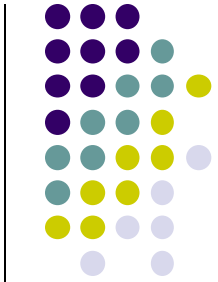
- Cờ Interrupt (Cờ cho phép ngắt):
  - Nếu IF = 1 → CPU ở trạng thái cho phép ngắt với tín hiệu yêu cầu ngắt từ bên ngoài gửi tới.
  - Nếu IF = 0 → CPU ở trạng thái cấm ngắt với tín hiệu yêu cầu ngắt từ bên ngoài gửi tới.

Minh họa các thanh ghi của CPU



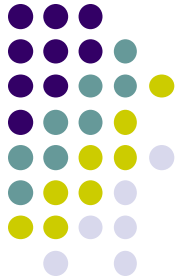


-0.125



- Công thức: **R = $(-1)^S$*1.M*2^{E-bias}**
- **Bias = 2^8/2-1 = 127**
- **Khuôn dạng: S E M**
  - S 1 bít. S = 0 số dương, S = 1 số âm
  - E là 8 bit
  - M là 23 bit
  - 0.125 x 2 = 0.25
  - 0.25 x 2 = 0.5
  - 0.5 x 2 = 1.0
  - 0.125 = 0.001b
- Chuẩn hóa: -0.125 =-0.001b= (-1)^1*1.0*2^-3
- Xác đinh: S = 1; E-bias = -3 -> E = bias-3 = 124

Chuyển 124 sang nhị phân

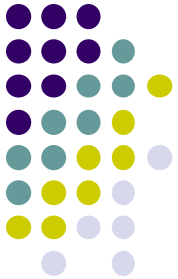


- 124/2 = 62 dư 0
- 62/2 = 31 dư 0
- 31/2 = 15 dư 1
- 15/2 = 7 dư 1
- 7/2 = 3 dư 1
- 3/2 = 1 dư 1
- ½ = 0 dư 1
- 124 = 1111100b-> E=01111100b
- M = 00000000000000000000000b
- -0.125 = 1011/1110/0000/0000/0000/0000/0000/0000b
- = BE000000h

Phân tích 124 thành tổng của các lũy thừa 2.

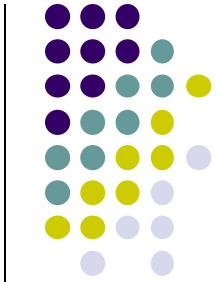
124 = 64 + 32 + 16 + 8 + 4 = 2^6 + 2^5 + 2^4 + 2^3 + 2^2 = 1111100b

97.2



- Công thức: R = (-1)^S*1.M*2^E-bias
- Bias = 2^ số bít của E/2-1
- Khuôn dạng: SEM
- Chuyển:
- + Phần nguyên
- 97 = 64 + 32 + 1 = 2^6 + 2^5 + 2^0 =1100001b
- Phần thập phân: 0.2x2 = 0.4
- 0.4x2 = 0.8 0.8x2 = 1.6 0.6x 2 = 1.2 0.2x2 = 0.4 -> 0.2 = 0.001100110011…
- 97.2 = 1100001.001100110011…b
- Chuẩn hóa: 97.2 = (-1)^**0***1.**10000100110011…***2^6

97.2



- Chuẩn hóa: 97.2 = (-1)^**0***1.**10000100110011…***2^6
- S = 0
- E-bias = 6->E = 6+127 = 133 = 128 + 4 + 1
- = 2^7 + 2^2 + 2^0 = 10000101b
- M = 10000100110011001100110b
- 0 100/0010/1 100/0010/0110/0110/0110/0110b
- 42C26666h

- “S12.75” Tra lần lượt kí tự in ASCII
- S có mã là 53h = 01010011b
- 1 có mã là 31h = 00110001b
- 2 có mã là 32h = 00110010b
- . Có mã là 2Eh = 00101110b
- 7 có mã là 37h = 00110111b
- 5 có mã là 35h = 00110101b

00110101	5
00110111	4
00101110	3
00110010	2
00110001	1
01010011	0