HỆ THỐNG MÁY TÍNH

02 – Biểu diễn số nguyên

Hệ cơ số q tổng quát

- Tổng quát số nguyên có n chữ số thuộc hệ cơ số q bất kỳ được biểu diễn:

$$\boxed{x_{n-1} \ldots x_1 x_0 = x_{n-1}.q^{n-1} + \ldots + x_1.q^1 + x_0.q^0}$$

(mỗi chữ số x_i lấy từ tập X có q phần tử)

- Ví dụ:
  - Hệ cơ số 10: A = 123 = 100 + 20 + 3 = **1**.10^2 + **2**.10^1 + **3**.10^0
  - q = 2, X = {0, 1}: hệ nhị phân (binary)
  - q = 8, X = {0, 1, 2,…, 7}: hệ bát phân (octal)
  - q = 10, X = {0, 1, 2,…, 9}: hệ thập phân (decimal)
  - q = 16, X = {0, 1, 2,…,9, A, B,…, F}: hệ thập lục phân (hexadecimal)
- Chuyển đổi: A = 123 d = 01111011 b = 173 o = 7B h
- Hệ cơ số thường được biển diễn trong máy tính là hệ cơ số 2

Chuyển đổi giữa các hệ cơ số

- Đặc điểm
  - Con người sử dụng hệ thập phân
  - Máy tính sử dụng hệ nhị phân, bát phân, thập lục phân
- Nhu cầu
  - Chuyển đổi qua lại giữa các hệ đếm ?
    - Hệ khác sang hệ thập phân (... → dec)
    - Hệ thập phân sang hệ khác (dec → ...)
    - Hệ nhị phân sang hệ khác và ngược lại (bin ←→ ...)
    - ...

Chuyển đổi giữa các hệ cơ số
[1] Decimal (10) → Binary (2)

- Lấy số cơ số 10 chia cho 2
  - Số dư đưa vào kết quả
  - Số nguyên đem chia tiếp cho 2
  - Quá trình lặp lại cho đến khi số nguyên = 0
- Ví dụ: A = 123
  - 123 : 2 = 61 dư 1
  - 61 : 2 = 30 dư 1
  - 30 : 2 = 15 dư 0
  - 15 : 2 = 7 dư 1
  - 7 : 2 = 3 dư 1
  - 3 : 2 = 1 dư 1
  - 1 : 2 = 0 dư 1

Kết quả: 1111011, vì 123 là số dương,

thêm 1 bit hiển dấu vào đầu là 0 vào

→ Kết quả cuối cùng: **01111011**

Chuyển đổi giữa các hệ cơ số
[2] Decimal (10) → Hexadecimal (16)

- Lấy số cơ số 10 chia cho 16
  - Số dư đưa vào kết quả
  - Số nguyên đem chia tiếp cho 16
  - Quá trình lặp lại cho đến khi số nguyên = 0
- Ví dụ: A = 123
  - 123 : 16 = 7 dư 11 (B)
  - 7 : 16 = 0 dư 7

→ Kết quả cuối cùng: **7B**

Chuyển đổi giữa các hệ cơ số
[3] Binary (2) → Decimal (10)

- Khai triển biểu diễn và tính giá trị biểu thức

$$\boxed{x_{n-1}...x_1x_0 = x_{n-1}.2^{n-1} + ... + x_1.2^1 + x_0.2^0}$$

- Ví dụ:
  - $1011_2 = 1.2^3 + 0.2^2 + 1.2^1 + 1.2^0 = 11_{10}$

Chuyển đổi giữa các hệ cơ số
[4] Binary (2) → Hexadecimal (16)

- Nhóm từng bộ 4 bit trong biểu diễn nhị phân rồi chuyển sang ký số tương ứng trong hệ thập lục phân (0000 → 0,…, 1111 → F)
- Ví dụ
  - $1001011_2 = 0100\ 1011 = 4B_{16}$

HEX	BIN	HEX	BIN	HEX	BIN	HEX	BIN
0	0000	4	0100	8	1000	C`	1100
1	0001	5	0101	9	1001	D	1101
2	0010	6	0110	A	1010	E	1110
3	0011	7	0111	B	1011	F	1111

Chuyển đổi giữa các hệ cơ số
[5] Hexadecimal (16) → Binary (2)

- Sử dụng bảng dưới đây để chuyển đổi:

HEX	BIN	HEX	BIN	HEX	BIN	HEX	BIN
0	0000	4	0100	8	1000	C`	1100
1	0001	5	0101	9	1001	D	1101
2	0010	6	0110	A	1010	E	1110
3	0011	7	0111	B	1011	F	1111

- Ví dụ:
  - $4B_{16} = 1001011_2$

Chuyển đổi giữa các hệ cơ số
[6] Hexadecimal (16) → Decimal (10)

- Khai triển biểu diễn và tính giá trị biểu thức

$$\boxed{x_{n-1}\ldots x_1 x_0 = x_{n-1}.16^{n-1} + \ldots + x_1.16^1 + x_0.16^0}$$

- Ví dụ:
  - $7B_{16} = 7.16^1 + 12\ (B).16^0 = 123_{10}$

Hệ nhị phân

$$\boxed{x_{n-1}...x_1x_0 = x_{n-1}.2^{n-1} + ... + x_1.2^1 + x_0.2^0}$$

- Được dùng nhiều trong máy tính để biểu diện các giá trị lưu trong các thanh ghi hoặc trong các ô nhớ. Thanh ghi hoặc ô nhớ có kích thước 1 byte (8 bit) hoặc 1 word (16 bit).
- n được gọi là chiều dài bit của số đó
- Bit trái nhất x_{n-1} là bit có giá trị (nặng) nhất MSB (Most Significant Bit)
- Bit phải nhất x_0 là bit ít giá trị (nhẹ) nhất LSB (Less Significant Bit)

Ý tưởng nhị phân

- Số nhị phân có thể dùng để biểu diễn bất kỳ việc gì mà bạn muốn!
- Một số ví dụ:
  - Giá trị logic: 0 → False; 1 → True
  - Ký tự:
    - 26 ký tự (A → Z): 5 bits (2^5 = 32)
    - Tính cả trường hợp viết hoa/thường + ký tự lạ → 7 bits (ASCII)
    - Tất cả các ký tự ngôn ngữ trên thế giới → 8, 16, 32 bits (Unicode)
  - Màu sắc: Red (00), Green (01), Blue (11)
  - Vị trí / Địa chỉ: (0, 0, 1)…
  - Bộ nhớ: N bits → Lưu được tối đa 2^N đối tượng

Số nguyên không dấu

- Đặc điểm
  - Biểu diễn các đại lương luôn dương
    - Ví dụ: chiều cao, cân nặng, mã ASCII…
  - Tất cả bit đều được sử dụng để biểu diễn giá trị (không quan tâm đến dấu âm, dương)
  - Số nguyên không dấu 1 byte lớn nhất là $1111\ 1111_2 = 2^8 - 1 = 255_{10}$
  - Số nguyên không dấu 1 word lớn nhất là $1111\ 1111\ 1111\ 1111_2 = 2^{16} - 1 = 65535_{10}$
  - Tùy nhu cầu có thể sử dụng số 2, 3… word.
  - LSB = 1 thì số đó là số đó là số lẻ

Số nguyên có dấu

- Lưu các số dương hoặc âm (số có dấu)
- Có 4 cách phổ biến:
  - [1] Dấu lượng
  - [2] Bù 1
  - [3] Bù 2
  - [4] Số quá (thừa) K
- Số có dấu trong máy tính được biểu diễn ở dạng số bù 2

Số nguyên có dấu
[1] Dấu lượng

- Bit trái nhất (MSB): bit đánh dấu âm / dương
  - 0: số dương
  - 1: số âm
- Các bit còn lại: biểu diễn độ lớn của số (hay giá trị tuyệt đối của số)
- **Ví dụ:**
  - Một byte 8 bit: sẽ có 7 bit (trừ đi bit dấu) dùng để biểu diễn giá trị tuyệt đối cho các số có giá trị từ 0000000 (0_{10}) đến 1111111 (127_{10})

  → Ta có thể biểu diễn các số từ -127_{10} đến $+127_{10}$
  - -N và N chỉ khác giá trị bit MSB (bit dấu), phần độ lớn (giá trị tuyệt đối) hoàn toàn giống nhau

Số nguyên có dấu
[2] Bù 1

- Tương tự như phương pháp [1], bit MSB dùng làm bit dấu
  - 0: Số dương
  - 1: Số âm
- Các bit còn lại (*) dùng làm độ lớn
- Số âm: Thực hiện phép đảo bit tất cả các bit của (*)
- **Ví dụ:**
  - Dạng bù 1 của 00101011 (43) là 11010100 (−43)
  - Một byte 8 bit: biểu diễn từ -127_{10} đến $+127_{10}$
  - Bù 1 có hai dạng biểu diễn cho số 0, bao gồm: 00000000 (+0) và 11111111 (−0) (mẫu 8 bit, giống phương pháp [1])
  - Khi thực hiện phép cộng, cũng thực hiện theo quy tắc cộng nhị phân thông thường, tuy nhiên, nếu còn phát sinh bit nhớ thì phải tiếp tục cộng bit nhớ này vào kết quả vừa thu được

Số nguyên có dấu
[3] Bù 2

- Biểu diễn giống như số bù 1 + ta phải cộng thêm số 1 vào kết quả (dạng nhị phân)
- Số bù 2 ra đời khi người ta gặp vấn đề với hai phương pháp dấu lượng [1] và bù 1 [2], đó là:
  - Có hai cách biểu diễn cho số 0 (+0 và -0) → không đồng nhất
  - Bit nhớ phát sinh sau khi đã thực hiện phép tính phải được cộng tiếp vào kết quả → dễ gây nhầm lẫn

  → Phương pháp số bù 2 khắc phục hoàn toàn 2 vấn đề đó
- **Ví dụ:**
  - Một byte 8 bit: biểu diễn từ -128_{10} đến $+127_{10}$ (được lợi 1 số vì chỉ có 1 cách biểu diễn số 0)

Số bù 1 và Số bù 2

Số 5 (8 bit)		0	0	0	0	0	1	0	1
Số bù 1 của 5		1	1	1	1	1	0	1	0
+									1
Số bù 2 của 5		1	1	1	1	1	0	1	1
+ Số 5		0	0	0	0	0	1	0	1
Kết quả	1	0	0	0	0	0	0	0	0

Nhận xét số bù 2

- (Số bù 2 của x) + x = một dãy toàn bit 0 (không tính bit 1 cao nhất do vượt quá phạm vi lưu trữ)

→ Do đó số bù 2 của x chính là giá trị âm của x hay – x (Còn gọi là phép lấy đối)

- Đổi số thập phân âm –5 sang nhị phân?

→ Đổi 5 sang nhị phân rồi lấy số bù 2 của nó

- Thực hiện phép toán a – b?

→ a – b = a + (–b) → Cộng với số bù 2 của b.

Số nguyên có dấu
[4] Số quá (thừa) K (biased)

- Còn gọi là biểu diễn số dịch (*biased representation*)
- Chọn một số nguyên dương K cho trước làm giá trị dịch
- Biểu diễn số N:
  - **+N (dương):** có được bằng cách lấy K + N, với K được chọn sao cho tổng của K và một số âm bất kỳ trong miền giá trị luôn luôn dương
  - **-N (âm):** có được bằng cáck lấy K - N (hay lấy bù hai của số vừa xác định)
- **Ví dụ:**
  - Dùng 1 Byte (8 bit): biểu diễn từ -128_{10} đến $+127_{10}$
  - Trong hệ 8 bit, biểu diễn N = 25, chọn số thừa k = 128, :
    - $+25_{10} = 10011001_2$
    - $-25_{10} = 01100111_2$
  - Chỉ có một giá trị 0: $+0 = 10000000_2$, $-0 = 10000000_2$

Nhận xét

- Số bù 2 [3] → lưu trữ số có dấu và các phép tính của chúng trên máy tính (thường dùng nhất)
  - Không cần thuật toán đặc biệt nào cho các phép tính cộng và tính trừ
  - Giúp phát hiện dễ dàng các trường hợp bị tràn.
- Dấu lượng [1] / số bù 1 [2] → dùng các thuật toán phức tạp và bất lợi vì luôn có hai cách biểu diễn của số 0 (+0 và -0)
- Dấu lượng [1] → phép nhân của số có dấu chấm động
- Số thừa K [4] → dùng cho số mũ của các số có dấu chấm động

Biểu diễn số âm (số bù 2)

$$x_{n-1}...x_1x_0 = x_{n-1}.(-2^{n-1}) + x_{n-2}.2^{n-2}... + x_1.2^1 + x_0.2^0$$

N bits

-2^{n-1}	2^{n-2}	...	...	...	2^3	2^2	2^1	2^0

Phạm vi lưu trữ: $[-2^{n-1}, 2^{n-1} - 1]$

- Ví dụ:
  - $1101\ 0110_2 = -2^7 + 2^6 + 2^4 + 2^3 + 2^2 + 2^1$
    $= -128 + 64 + 16 + 4 + 2 =$
    $= -42_{10}$

Ví dụ (số bù 2)


```
+123 = 01111011b
-123 = 10000101b
   0 = 00000000b
  -1 = 11111111b
  -2 = 11111110b
  -3 = 11111101b
-127 = 10000001b
-128 = 10000000b
```

Tính giá trị không dấu và có dấu

- Tính giá trị không dấu và có dấu của 1 số?
  - Ví dụ số word (16 bit): 1100 1100 1111 0000
  - Số nguyên không dấu ?
    - Tất cả 16 bit lưu giá trị → giá trị là 52464
  - Số nguyên có dấu ?
    - Bit MSB = 1 do đó số này là số âm
    - Áp dụng công thức → giá trị là –13072

Tính giá trị không dấu và có dấu

- Nhận xét
  - Bit MSB = 0 thì giá trị có dấu bằng giá trị không dấu.
  - Bit MSB = 1 thì giá trị có dấu bằng giá trị không dấu trừ đi 256 (2^8 nếu tính theo byte) hay 65536 (2^{16} nếu tính theo word).
- Tính giá trị không dấu và có dấu của 1 số?
  - Ví dụ số word (16 bit): 1100 1100 1111 0000
  - Giá trị không dấu = 52464
  - Giá trị có dấu: vì bit MSB = 1 nên giá trị có dấu = 52464 – 65536 = –13072

Phép dịch bit và phép xoay

- Shift left (SHL): 1100 1010 → 1001 0100
  - Chuyển tất cả các bit sang trái, bỏ bit trái nhất, thêm 0 ở bit phải nhất
- Shift right (SHR): 1001 0101 → 0100 1010
  - Chuyển tất cả các bit sang phải, bỏ bit phải nhất, thêm 0 ở bit trái nhất
- Rotate left (ROL): 1100 1010 → 1001 0101
  - Chuyển tất cả các bit sang trái, bit trái nhất thành bit phải nhất
- Rotate right (ROR): 1001 0101 → 1100 1010
  - Chuyển tất cả các bit sang phải, bit phải nhất thành bit trái nhất

Phép toán Logic
AND, OR, NOT, XOR

AND	0	1
0	0	0
1	0	1

"Phép nhân"

OR	0	1
0	0	1
1	1	1

"Phép cộng"

XOR	0	1
0	0	1
1	1	0

"Phép so sánh khác"

NOT	0	1
	1	0

"Phép phủ định"

```
AND  11010011      OR  00000011      XOR  01100011
     00001111          01100000           01100011
     --------          --------           --------
     00000011          01100011           00000000

NOT  11010011
     --------
  =  00101100
```

Ví dụ

- X = 0000 1000b = 8d
- → X shl 2 = 0010 0000b = 32d = 8 . 2^2
- → (X shl 2) or X = 0010 1000b = 40d = 32 + 8

- Y = 0100 1010b = 74d
- → ((Y and 0Fh) shl 4) = 1010 0000

  OR OR

- → ((Y and F0h) shr 4) = 0000 0100

= 1010 0100 = 164d (không dấu)

= (164 – 2^8) = -92d (có dấu)

Một số nhận xét

- x SHL y = x . 2^y
- x SHR y = x / 2^y
- AND dùng để tắt bit (AND với 0 luôn = 0)
- OR dùng để bật bit (OR với 1 luôn = 1)
- XOR, NOT dùng để đảo bit (XOR với 1 = đảo bit đó)
- x AND 0 = 0
- x XOR x = 0

- Mở rộng:
  - Lấy giá trị tại bit thứ i của x: (x SHR i) AND 1
  - Gán giá trị 1 tại bit thứ i của x: (1 SHL i) OR x
  - Gán giá trị 0 tại bit thứ i của x: NOT(1 SHL i) AND x
  - Đảo bit thứ i của x: (1 SHL i) XOR x

Các phép toán tử

- Phép Cộng (+)
- Phép Trừ (-)
- Phép Nhân (*)
- Phép Chia (/)

Phép cộng

- Nguyên tắc cơ bản:

+	0	1
0	0	1
1	1	10

- Ví dụ:

				1	
	1	1	1	0	1
+	1	0	0	0	1
1	0	1	1	1	0

Phép cộng

1001 = −7 +0101 = 5 1110 = −2 (a) (−7) + (+5)	1100 = −4 +0100 = 4 10000 = 0 (b) (−4) + (+4)
0011 = 3 +0100 = 4 0111 = 7 (c) (+3) + (+4)	1100 = −4 +1111 = −1 11011 = −5 (d) (−4) + (−1)
0101 = 5 +0100 = 4 1001 = Overflow (e) (+5) + (+4)	1001 = −7 +1010 = −6 10011 = Overflow (f) (−7) + (−6)

Phép trừ

- Nguyên tắc cơ bản: Đưa về phép cộng

  A – B = A + (-B) = A + (số bù 2 của B)

- Ví dụ: 11101 – 10011 = 11101 + 01101

```
          1
    1 1 1 0 1
+   0 1 1 0 1
  -----------
  1 0 1 0 1 0
```

Phép trừ

0010 = 2 +1001 = −7 1011 = −5 (a) M = 2 = 0010 S = 7 = 0111 −S = 1001	0101 = 5 +1110 = −2 10011 = 3 (b) M = 5 = 0101 S = 2 = 0010 −S = 1110
1011 = −5 +1110 = −2 11001 = −7 (c) M = −5 = 1011 S = 2 = 0010 −S = 1110	0101 = 5 +0010 = 2 0111 = 7 (d) M = 5 = 0101 S = −2 = 1110 −S = 0010
0111 = 7 +0111 = 7 1110 = Overflow (e) M = 7 = 0111 S = −7 = 1001 −S = 0111	1010 = −6 +1100 = −4 10110 = Overflow (f) M = −6 = 1010 S = 4 = 0100 −S = 1100

Phép nhân

- Nguyên tắc cơ bản:

×	0	1
0	0	0
1	0	1

Phép nhân

			1	0	0	0	1
×					1	1	0
			0	0	0	0	0
+		1	0	0	0	1	
	1	0	0	0	1		
	1	1	0	0	1	1	0

Phép nhân


```
      1011     = 11
  x   1101     = 13
  ----------
      1011
     0000
    1011
   1011
  ----------
  10001111     = 143
```

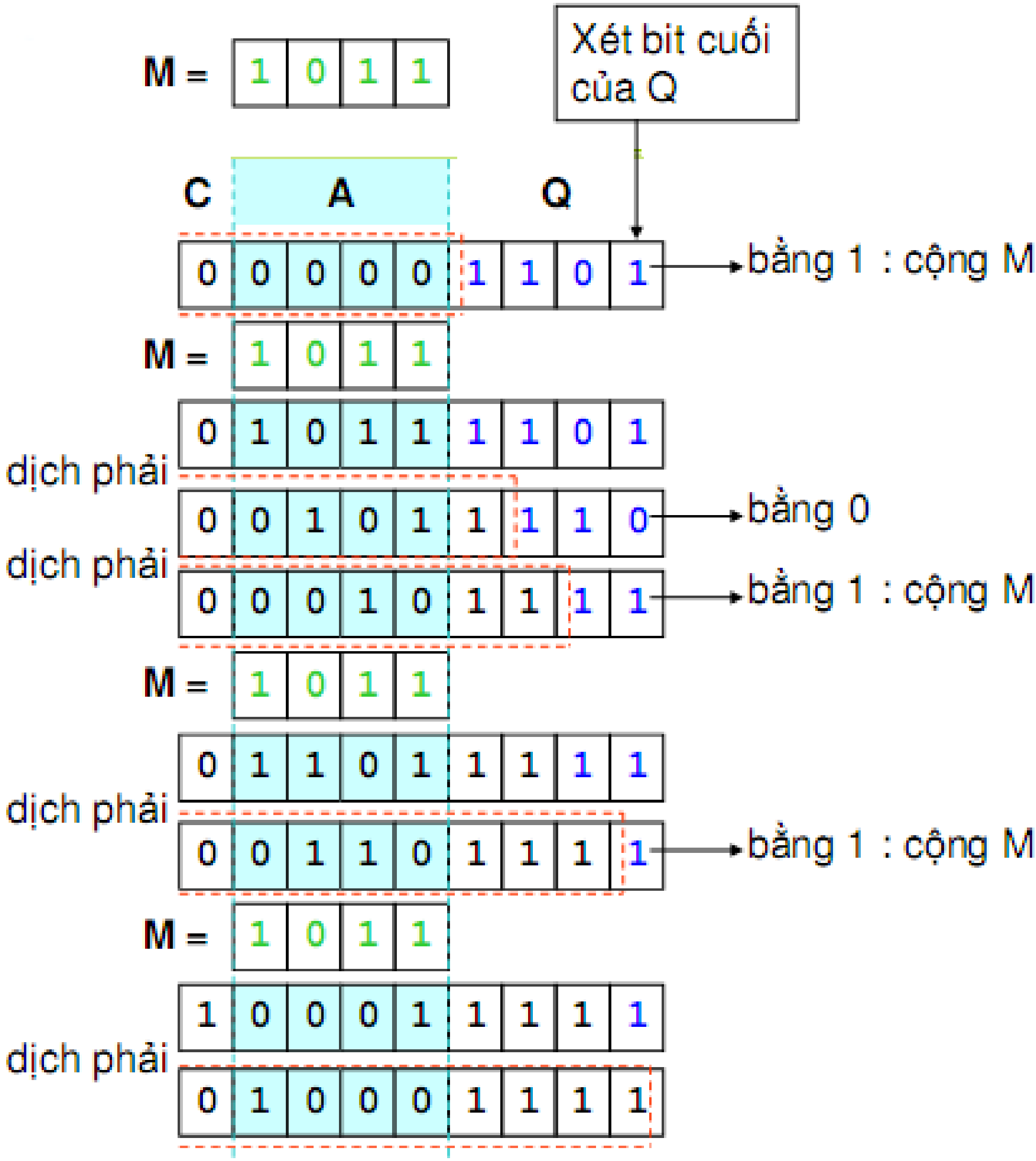
```
      1011
  x   1101
  ----------
  00000000
+     1011
  ----------
  00001011
+    0000
  ----------
  00001011
+   1011
  ----------
  00110111
+  1011
  ----------
  10001111
```

Thuật toán nhân

- Giả sử ta muốn thực hiện phép nhân M x Q với
  - Q có n bit
- Ta định nghĩa các biến:
  - C (1 bit): đóng vai trò bit nhớ
  - A (n bit): đóng vai trò 1 phần kết quả nhân ([C, A, Q]: kết quả nhân)
  - [C, A] (n + 1 bit) ; [C, A, Q] (2n + 1 bit): coi như các thanh ghi ghép
- Thuật toán:

```
Khởi tạo: [C, A] = 0; k = n
Lặp khi k > 0
{
      Nếu bit cuối của Q = 1 thì
            Lấy (A + M) → [C, A]

      Shift right [C, A, Q]
      k = k – 1
}
```



Xét bit cuối của Q
M = 1 0 1 1
C
A
Q
0 0 0 0 0 1 1 0 1
bằng 1 : cộng M
M = 1 0 1 1
0 1 0 1 1 1 1 0 1
dịch phải
0 0 1 0 1 1 1 1 0
bằng 0
dịch phải
0 0 0 1 0 1 1 1 1
bằng 1 : cộng M
M = 1 0 1 1
0 1 1 0 1 1 1 1 1
dịch phải
0 0 1 1 0 1 1 1 1
bằng 1 : cộng M
M = 1 0 1 1
1 0 0 0 1 1 1 1 1
dịch phải
0 1 0 0 0 1 1 1 1

Thuật toán nhân cải tiến booth (số không/có dấu)

Khởi tạo: A = 0; k = n; Q_{-1} = 0 (thêm 1 bit = 0 vào cuối Q)

Lặp khi k > 0

{

Nếu 2 bit cuối của Q_0Q_{-1}

{

= 10 thì A – M → A

= 01 thì A + M → A

= 00, 11 thì A không thay đổi

}

Shift right [A, Q, Q_{-1}]

k = k – 1

}

Kết quả: [A, Q]

Ví dụ
M = 7, Q = -3, n = 4

		A	Q	Q_{-1}	M
Khởi đầu		0000	1101	0	0111
Bước 0:	A=A-M	1001	1101	0	0111
	shift	1100	1110	1	0111
Bước 1:	A=A+M	0011	1110	1	0111
	shift	0001	1111	0	0111
Bước 2:	A=A-M	1010	1111	0	0111
	shift	1101	0111	1	0111
Bước 3:	shift	1110	1011	1	0111

Kết quả 11101011 = -21

Phép chia

- Giả sử ta muốn thực hiện Q / M với

Khởi tạo: A = n bit 0; k = n

Lặp khi k > 0

{

Shift left (SHL) [A, Q]

A – M → A

\# Nếu A < 0: Q_0 = 0 và A + M → A

\# Ngược lại: Q_0 = 1

k = k – 1

}

Kết quả: Q là thương, A là số dư

Ví dụ phép chia

A	Q	M = 0011
0000	0111	Initial value
0000	1110	Shift
1101		Subtract
0000	1110	Restore
0001	1100	Shift
1110		Subtract
0001	1100	Restore
0011	1000	Shift
0000		Subtract
0000	1001	Set $Q_0 = 1$
0001	0010	Shift
1110		Subtract
0001	0010	Restore

(a) (7)/(3)

A	Q	M = 1101
0000	0111	Initial value
0000	1110	Shift
1101		Add
0000	1110	Restore
0001	1100	Shift
1110		Add
0001	1100	Restore
0011	1000	Shift
0000		Add
0000	1001	Set $Q_0 = 1$
0001	0010	Shift
1110		Add
0001	0010	Restore

(b) (7)/(−3)

Prefix in byte (Chuẩn IEC)

- International Electrotechnical Commission (IEC)

Name	Abbr	Factor
kibi	Ki	2^{10} = 1,024
mebi	Mi	2^{20} = 1,048,576
gibi	Gi	2^{30} = 1,073,741,824
tebi	Ti	2^{40} = 1,099,511,627,776
pebi	Pi	2^{50} = 1,125,899,906,842,624
exbi	Ei	2^{60} = 1,152,921,504,606,846,976
zebi	Zi	2^{70} = 1,180,591,620,717,411,303,424
yobi	Yi	2^{80} = 1,208,925,819,614,629,174,706,176

Prefix in byte (Chuẩn SI)

- International System of Units (SI)

Name	Abbr	Factor	SI size
Kilo	K	2^{10} = 1,024	10^3 = 1,000
Mega	M	2^{20} = 1,048,576	10^6 = 1,000,000
Giga	G	2^{30} = 1,073,741,824	10^9 = 1,000,000,000
Tera	T	2^{40} = 1,099,511,627,776	10^{12} = 1,000,000,000,000
Peta	P	2^{50} = 1,125,899,906,842,624	10^{15} = 1,000,000,000,000,000
Exa	E	2^{60} = 1,152,921,504,606,846,976	10^{18} = 1,000,000,000,000,000,000
Zetta	Z	2^{70} = 1,180,591,620,717,411,303,424	10^{21} = 1,000,000,000,000,000,000,000
Yotta	Y	2^{80} = 1,208,925,819,614,629,174,706,176	10^{24} = 1,000,000,000,000,000,000,000,000

- Chú ý: khi nói "kilobyte" chúng ta nghĩ là 1024 byte nhưng thực ra nó là 1000 bytes theo chuẩn SI, 1024 bytes là kibibyte (IEC)
- Hiện nay chỉ có các nhà sản xuất đĩa cứng và viễn thông mới dùng chuẩn SI
  - **30** GB → $30 * 10^9$ ~ **28** * 2^{30} bytes
  - 1 Mbit/s → 10^6 b/s

Homework

- Đọc chương 9, sách của W.Stalling
- Đọc trước slide bài giảng số thực