KHOA CÔNG NGHỆ THÔNG TIN
BỘ MÔN MẠNG VÀ CÁC HỆ THỐNG THÔNG TIN

CHƯƠNG 2

BIỂU DIỄN THÔNG TIN TRONG MÁY TÍNH

Nội dung

2.1. Các hệ đếm cơ bản

2.2. Biểu diễn số nguyên và số thực dấu phẩy động

2.3. Các phép toán trên số nguyên và số thực

2.4. Mạch tổ hợp

2.5. Mã

2.1. 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

1. 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

$$A = a_n a_{n-1} \ldots a_1 a_0 , a_{-1} \ldots a_{-m}$$

Giá trị của *A* được hiểu như sau:

$$A = a_n 10^n + a_{n-1} 10^{n-1} + \ldots + a_1 10^1 + a_0 10^0 + a_{-1} 10^{-1} + \ldots + a_{-m} 10^{-m}$$

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

Ví dụ số thập phân

$472.38 = 4x10^2 + 7x10^1 + 2x10^0 + 3x10^{-1} + 8x10^{-2}$

- Các chữ số của phần nguyên:
  - 472 : 10 = 47 dư 2
  - 47 : 10 = 4 dư 7
  - 4 : 10 = 0 dư 4

- Các chữ số của phần lẻ:
  - 0.38 x 10 = 3.8 phần nguyên = 3
  - 0.8 x 10 = 8.0 phần nguyên = 8

2. Hệ nhị phân

- Cơ số 2
- 2 chữ số nhị phân: 0 và 1
- Chữ số nhị phân được 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 = $2^n - 1$
- Các lệnh của chương trình và dữ liệu trong máy tính đều được mã hóa bằng số nhị phân

Biểu diễn số nhị phân

Số nhị phân				Số thập phân
1-bit	2-bit	3-bit	4-bit	
0	00	000	0000	0
1	01	001	0001	1
	10	010	0010	2
	11	011	0011	3
		100	0100	4
		101	0101	5
		110	0110	6
		111	0111	7
			1000	8
			1001	9
			1010	10
			1011	11
			1100	12
			1101	13
			1110	14
			1111	15

Đơn vị dữ liệu và thông tin trong máy tính

- bit – chữ số nhị phân (binary digit): là đơn vị thông tin nhỏ nhất, cho phép nhận một trong hai giá trị: 0 hoặc 1.
- byte là một tổ hợp 8 bit: có thể biểu diễn được 256 giá trị (2^8)
- Qui ước các đơn vị dữ liệu:
  - KB (Kilobyte) = 2^{10} bytes = 1024 bytes
  - MB (Megabyte) = 2^{10} KB = 2^{20}bytes (~10^6)
  - GB (Gigabyte) = 2^{10} MB = 2^{30}bytes (~10^9)
  - TB (Terabyte) = 2^{10} GB = 2^{40}bytes (~10^{12})
  - PB (Petabyte) = 2^{10} TB = 2^{50}bytes
  - EB (Exabyte) = 2^{10} PB = 2^{60}bytes

Qui ước mới về ký hiệu đơn vị dữ liệu

Theo thập phân			Theo nhị phân		
Đơn vị	Viết tắt	Giá trị	Đơn vị	Viết tắt	Giá trị
kilobyte	KB	10^3	kibibyte	KiB	$2^{10} = 1024$
megabyte	MB	10^6	mebibyte	MiB	2^{20}
gigabyte	GB	10^9	gibibyte	GiB	2^{30}
terabyte	TB	10^{12}	tebibyte	TiB	2^{40}
petabyte	PB	10^{15}	pebibyte	PiB	2^{50}
exabyte	EB	10^{18}	exbibyte	EiB	2^{60}

Dạng tổng quát của số nhị phân

$$A = a_n a_{n-1} \ldots a_1 a_0, a_{-1} \ldots a_{-m} \qquad \text{với } a_i = 0 \text{ hoặc } 1$$

Giá trị của *A* được tính như sau:

$$A = a_n 2^n + a_{n-1} 2^{n-1} + \ldots + a_1 2^1 + a_0 2^0 + a_{-1} 2^{-1} + \ldots + a_{-m} 2^{-m}$$

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

Ví dụ số nhị phân

$1101001.1011_{(2)}$ =

6 5 4 3 2 1 0 -1 -2 -3 -4

$= 1.2^6 + 1.2^5 + 0.2^4 + 1.2^3 + 0.2^2 + 0.2^1 + 1.2^0 + 2^{-1} + 0.2^{-2} + 1.2^{-3} + 1.2^{-4}$

$= 64 + 32 + 8 + 1 + 0.5 + 0.125 + 0.0625$

$= 105.6875_{(10)}$

Chuyển đổi số nguyên thập phân sang nhị phân

- Phương pháp 1: chia dần cho 2 rồi lấy phần dư
- Phương pháp 2: Phân tích thành tổng của các số 2^i → nhanh hơn

Phương pháp chia dần cho 2

- Ví dụ: chuyển đổi $105_{(10)}$
  - 105 : 2 = 52 dư 1
  - 52 : 2 = 26 dư 0
  - 26 : 2 = 13 dư 0
  - 13 : 2 = 6 dư 1
  - 6 : 2 = 3 dư 0
  - 3 : 2 = 1 dư 1
  - 1 : 2 = 0 dư 1

biểu diễn số dư theo chiều mũi tên

- Kết quả: $105_{(10)} = 1101001_{(2)}$

Phương pháp phân tích thành tổng của các 2^i

- Ví dụ 1: chuyển đổi $105_{(10)}$
  - $105 = 64 + 32 + 8 + 1 = 2^6 + 2^5 + 2^3 + 2^0$

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

  - Kết quả: $105_{(10)} = 0110\ 1001_{(2)}$

- Ví dụ 2: $17000_{(10)} = 16384 + 512 + 64 + 32 + 8$

$$= 2^{14} + 2^9 + 2^6 + 2^5 + 2^3$$

$$17000_{(10)} = 0100\ 0010\ 0110\ 1000_{(2)}$$

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

Chuyển đổi số lẻ thập phân sang nhị phân

- Ví dụ 1: chuyển đổi $0.6875_{(10)}$
  - 0.6875 x 2 = 1.375 phần nguyên = 1
  - 0.375 x 2 = 0.75 phần nguyên = 0
  - 0.75 x 2 = 1.5 phần nguyên = 1
  - 0.5 x 2 = 1.0 phần nguyên = 1

biểu diễn theo chiều mũi tên

- Kết quả : $0.6875_{(10)} = 0.1011_{(2)}$

Chuyển đổi số lẻ thập phân sang nhị phân (tiếp)

- Ví dụ 2: chuyển đổi $0.81_{(10)}$
  - 0.81 x 2 = 1.62 phần nguyên = 1
  - 0.62 x 2 = 1.24 phần nguyên = 1
  - 0.24 x 2 = 0.48 phần nguyên = 0
  - 0.48 x 2 = 0.96 phần nguyên = 0
  - 0.96 x 2 = 1.92 phần nguyên = 1
  - 0.92 x 2 = 1.84 phần nguyên = 1
  - 0.84 x 2 = 1.68 phần nguyên = 1
- $0.81_{(10)} \approx 0.1100111_{(2)}$

3. Hệ mười sáu (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

Quan hệ giữa số nhị phân và số Hexa

Ví dụ:

- $1011\ 0011_{(2)} = B3_{(16)}$
- $0000\ 0000_{(2)} = 00_{(16)}$

- $0010\ 1101\ 1001\ 1010_{(2)} = 2D9A_{(16)}$
- $1111\ 1111\ 1111\ 1111_{(2)} = FFFF_{(16)}$

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

2.2. Biểu diễn số nguyên và số thực dấu phẩy động

2.2.1 Biểu diễn số nguyên

2.2.2 Biểu diễn số thực dấu phẩy động

2.2.1 Biểu diễn số nguyên

- Số nguyên không dấu (Unsigned Integer)
- Số nguyên có dấu (Signed Integer)

1. Biểu diễn số nguyên không dấu

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

$$a_{n-1}a_{n-2}\ldots a_2a_1a_0$$

Giá trị của A được tính như sau:

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

Dải biểu diễn của A: $[0, 2^n - 1]$

Ví dụ 1

- Biểu diễn các số nguyên không dấu sau đây bằng 8-bit:

$$A = 41 \; ; \quad B = 150$$

Giải:

$$A = 41 = 32 + 8 + 1 = 2^5 + 2^3 + 2^0$$

$$41 = 0010\ 1001$$

$$B = 150 = 128 + 16 + 4 + 2 = 2^7 + 2^4 + 2^2 + 2^1$$

$$150 = 1001\ 0110$$

Ví dụ 2

- Cho các số nguyên không dấu M, N được biểu diễn bằng 8-bit như sau:
  - M = 0001 0010
  - N = 1011 1001

  Xác định giá trị của chúng ?

Giải:

- M = 0001 0010 = $2^4 + 2^1 = 16 + 2 = 18$
- N = 1011 1001 = $2^7 + 2^5 + 2^4 + 2^3 + 2^0$

  $= 128 + 32 + 16 + 8 + 1 = 185$

Với n = 8 bit

Biểu diễn được các giá trị từ 0 đến 255 ($2^8 - 1$)

Chú ý:

1111 1111

\+ 0000 0001

1 0000 0000

có *nhớ ra ngoài*

(Carry out)

255 + 1 = 0 ???

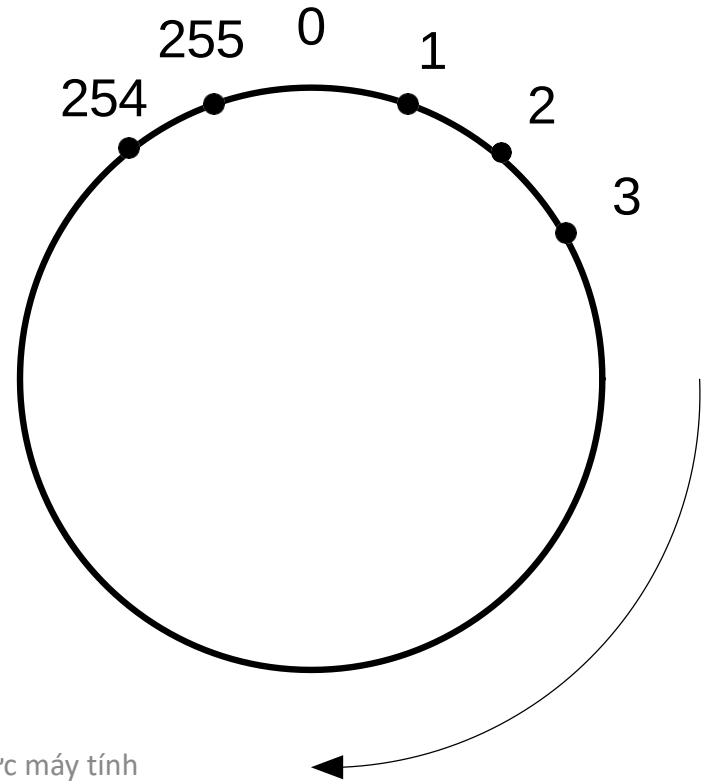
do vượt ra khỏi dải biểu diễn

Biểu diễn nhị phân	Giá trị thập phân
0000 0000	0
0000 0001	1
0000 0010	2
0000 0011	3
0000 0100	4
...	
1111 1110	254
1111 1111	255

Trục số học với n = 8 bit

Trục số học:

Trục số học máy tính:



Với n = 16 bit, 32 bit, 64 bit

- n= 16 bit: dải biểu diễn từ 0 đến 65535 ($2^{16} - 1$)
  - 0000 0000 0000 0000 = 0
  - ...
  - 0000 0000 1111 1111 = 255
  - 0000 0001 0000 0000 = 256
  - ...
  - 1111 1111 1111 1111 = 65535
- n= 32 bit: dải biểu diễn từ 0 đến $2^{32} - 1$
- n= 64 bit: dải biểu diễn từ 0 đến $2^{64} - 1$

2. Biểu diễn số nguyên có dấu

Số bù một và Số bù hai

- Định nghĩa: Cho một số nhị phân A được biểu diễn bằng n bit, ta có:
  - Số bù một của A = $(2^n-1) - A$
  - Số bù hai của A = $2^n - A$
- Số bù hai của A = (Số bù một của A) +1

Ví dụ

Với n = 8 bit, cho A = 0010 0101

- Số bù một của A được tính như sau:

$$\begin{array}{rl} 1111\ 1111 & (2^8 - 1) \\ -\ \underline{0010\ 0101} & (A) \\ 1101\ 1010 & \end{array}$$

→ đảo các bit của A

- Số bù hai của A được tính như sau:

$$\begin{array}{rl} 1\ 0000\ 0000 & (2^8) \\ -\ \underline{0010\ 0101} & (A) \\ 1101\ 1011 & \end{array}$$

→ thực hiện khó khăn

Quy tắc tìm Số bù một và Số bù hai

- Số bù một của A = đảo giá trị các bit của A
- (Số bù hai của A) = (Số bù một của A) + 1
- Ví dụ:
  - Cho A = 0010 0101
  - Số bù một của A = 1101 1010
  - \+ 1
  - Số bù hai của A = 1101 1011
- Nhận xét:

A = 0010 0101

Số bù hai của A = + 1101 1011

1 0000 0000 = 0

(bỏ qua bit nhớ ra ngoài)

→ Số bù hai của A = -A

Tìm số bù hai cách 2

Tìm bit 1 đầu tiên từ phải sang trái:

Đây là một phương pháp khác nhanh hơn, thường được sử dụng khi làm việc với các số nhị phân lớn.

1. Giữ nguyên các bit từ phải sang trái cho đến khi gặp bit 1 đầu tiên, giữ nguyên bit 1 này.
2. Đảo tất cả các bit còn lại (từ bit 1 đầu tiên đó trở về bên trái).

Ví dụ: Tìm số bù 2 của số nhị phân 10101100.

- Bước 1 (Giữ nguyên): Bắt đầu từ bit ngoài cùng bên phải. Ta giữ nguyên các bit 00 và bit 1 đầu tiên. Ta được: ...100.
- Bước 2 (Đảo bit): Bây giờ, ta đảo các bit còn lại (10101). Đảo 10101 ta được 01010.
- Kết quả: Ghép hai phần lại, ta có số bù 2 là 01010100.

Biểu diễn số nguyên có dấu theo mã bù hai

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

$$a_{n-1}a_{n-2}\ldots a_2a_1a_0$$

- Với A là số dương: bit $a_{n-1} = 0$, các bit còn lại biểu diễn độ lớn như số không dấu
- Với A là số âm: được biểu diễn bởi số bù hai của số dương tương ứng, vì vậy bit $a_{n-1} = 1$

Ví dụ

- Biểu diễn các số nguyên có dấu sau đây bằng 8-bit:

A = + 58 ; B = - 80

Giải:

A = + 58 = 0011 1010

B = - 80

Ta có: + 80 = 0101 0000

Số bù hai = 1011 0000

Vậy:

B = - 80 = 1011 0000

Xác định giá trị của số dương

- Dạng tổng quát của số dương:

$$0a_{n-2} \ldots a_2 a_1 a_0$$

- Giá trị của số dương:

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

- Dải biểu diễn cho số dương: [0, +(2^{n-1} - 1)]

Xác định giá trị của số âm

- Dạng tổng quát của số âm:

$$1a_{n-2}\ldots a_2a_1a_0$$

- Giá trị của số âm:

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

- Dải biểu diễn cho số âm: $[-2^{n-1}, -1]$

Công thức tổng quát cho số nguyên có dấu

- Dạng tổng quát của số nguyên có dấu A:

$$a_{n-1}a_{n-2}\ \ldots a_2 a_1 a_0$$

- Giá trị của A được xác định như sau:

$$A = -a_{n-1}2^{n-1} + \sum_{i=0}^{n-2} a_i 2^i$$

- Dải biểu diễn: $[-(2^{n-1}), +(2^{n-1}-1)]$

Ví dụ

- Hãy xác định giá trị của các số nguyên có dấu được biểu diễn theo mã bù hai với 8-bit như dưới đây:
  - P = 0110 0010
  - Q = 1101 1011

Giải:

- P = 0110 0010 = $2^6 + 2^5 + 2^1 = 64 + 32 + 2 = +98$

$$A = -a_{n-1}2^{n-1} + \sum_{i=0}^{n-2} a_i 2^i$$

- Q = 1101 1011 = $-1.2^7 + 1.2^6 + 0.2^5 + 1.2^4 + 1.2^3 + 0.2^2 + 1.2^1 + 1.2^0$
  $= -128 + 64 + 16 + 8 + 2 + 1 = -37$

Với n = 8 bit

- Biểu diễn được các giá trị từ -2^7 đến $+2^7-1$
  - -128 đến +127
  - Chỉ có một giá trị 0
  - Không biểu diễn cho giá trị +128

Chú ý:

+127 + 1 = -128

(-128)+(-1) = +127

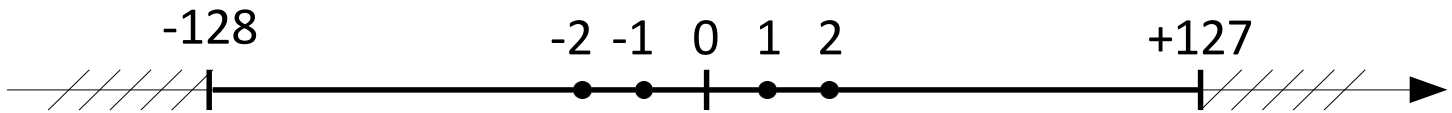
có tràn xảy ra (Overflow)

(do vượt ra khỏi dải biểu diễn)

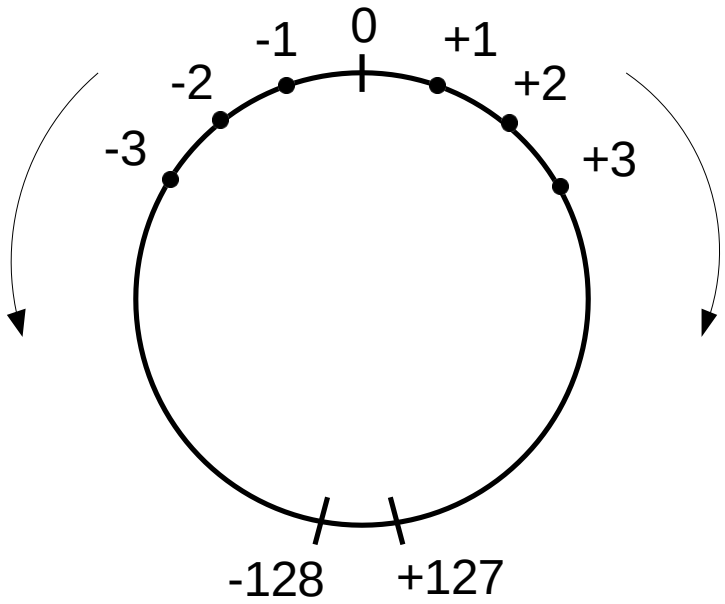
Giá trị thập phân	Biểu diễn bù hai
0	0000 0000
+1	0000 0001
+2	0000 0010
	...
+126	0111 1110
+127	0111 1111
-128	1000 0000
-127	1000 0001
	...
-2	1111 1110
-1	1111 1111

Trục số học số nguyên có dấu với n = 8 bit

- Trục số học:



- Trục số học máy tính:



Với n = 16 bit, 32 bit, 64 bit

- Với n = 16bit: biểu diễn từ -2^{15} đến $2^{15}-1$
  - 0000 0000 0000 0000 = 0
  - 0000 0000 0000 0001 = +1
  - ...
  - 0111 1111 1111 1111 = +32767 (2^{15} - 1)
  - 1000 0000 0000 0000 = -32768 (-2^{15})
  - 1000 0000 0000 0001 = -32767
  - ...
  - 1111 1111 1111 1111 = -1
- Với n = 32bit: biểu diễn từ -2^{31} đến $2^{31}-1$
- Với n = 64bit: biểu diễn từ -2^{63} đến $2^{63}-1$

Mở rộng bit cho số nguyên

- Mở rộng theo số không dấu (Zero-extended): thêm các bit 0 vào bên trái
- Mở rộng theo số có dấu (Sign-extended):
  - Số dương:

    +19 = 0001 0011 (8bit)

    +19 = 0000 0000 0001 0011 (16bit)

    → thêm các bit 0 vào bên trái
  - Số âm:

    - 19 = 1110 1101 (8bit)

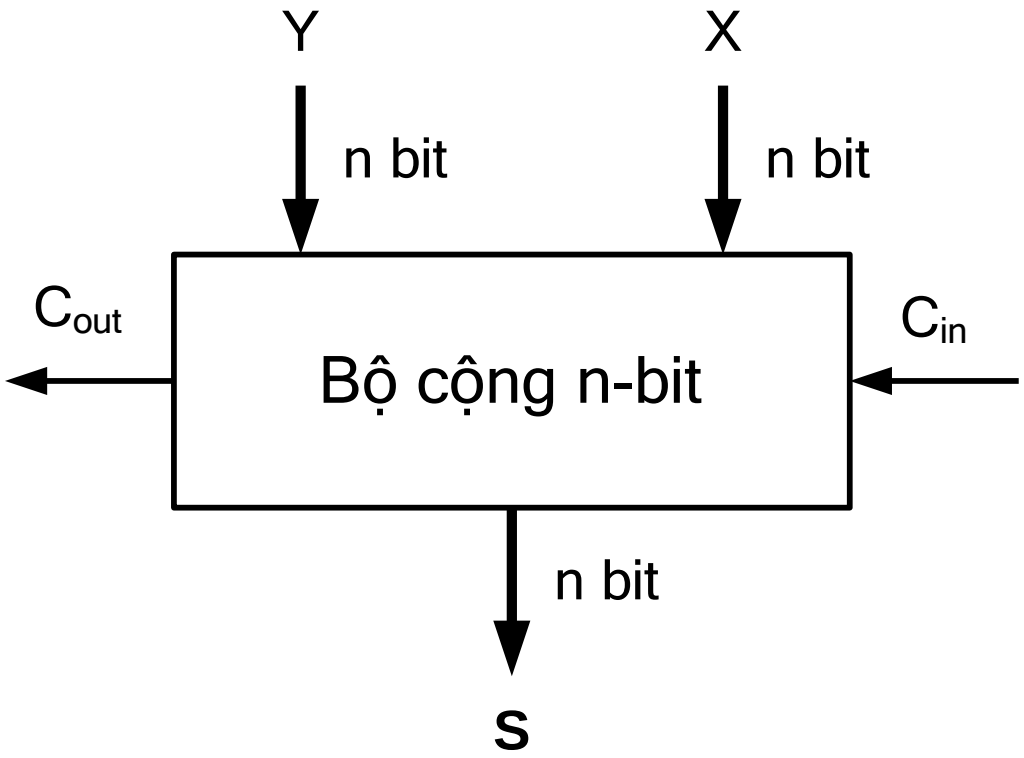
    - 19 = 1111 1111 1110 1101 (16bit)

    → thêm các bit 1 vào bên trái

1) Thực hiện phép cộng/trừ với số nguyên

1. Phép cộng số nguyên không dấu

Bộ cộng n-bit



Nguyên tắc cộng số nguyên không dấu

- Khi cộng hai số nguyên không dấu n-bit, kết quả nhận được là n-bit:
  - Nếu $C_{out} = 0$ → nhận được kết quả đúng
  - Nếu $C_{out} = 1$ → nhận được kết quả sai, do có *nhớ ra ngoài (Carry Out)*
- Hiện tượng *nhớ ra ngoài* xảy ra khi: tổng > $(2^n - 1)$

Ví dụ cộng số nguyên không dấu

- 57 = 0011 1001
  + 34 = + 0010 0010
  91 0101 1011 = 64+16+8+2+1=91 → đúng

- 209 = 1101 0001
  + 73 = + 0100 1001
  282 1 0001 1010
  kết quả = 0001 1010 =16+8+2=26 → sai
  do có *nhớ ra ngoài* (C_{out}=1)

Để có kết quả đúng, ta thực hiện cộng theo 16-bit:

209 = 0000 0000 1101 0001
+ 73 = + 0000 0000 0100 1001
0000 0001 0001 1010 = 256+16+8+2 = 282

2) Phép đảo dấu

- Ta có:

+ 37	=	0010 0101		
bù một	=	1101 1010		
		+ 1		
bù hai	=	1101 1011	=	-37

- Lấy bù hai của số âm:

- 37	=	1101 1011		
bù một	=	0010 0100		
		+ 1		
bù hai	=	0010 0101	=	+37

- Kết luận: *Phép đảo dấu số nguyên trong máy tính thực chất là lấy bù hai*

3) Cộng số nguyên có dấu

- Khi cộng hai số nguyên có dấu n-bit, kết quả nhận được là n-bit và *không cần quan tâm đến bit* C_{out}
  - Khi cộng hai số khác dấu thì kết quả luôn luôn đúng
  - Khi cộng hai số cùng dấu, nếu dấu kết quả cùng dấu với các số hạng thì kết quả là đúng
  - Khi cộng hai số cùng dấu, nếu kết quả có dấu ngược lại, khi đó có ***tràn*** (Overflow) xảy ra và kết quả bị sai
- Hiện tượng *tràn* xảy ra khi tổng nằm ngoài dải biểu diễn: $[-(2^{n-1}),+(2^{n-1}-1)]$

Ví dụ cộng số nguyên có dấu không tràn

- (+ 70) = 0100 0110
  + (+ 42) = 0010 1010
  + 112 0111 0000 = +112

- (+ 97) = 0110 0001
  + (- 52) = 1100 1100 (+52=0011 0100)
  + 45 1 0010 1101 = +45

- (- 90) = 1010 0110 (+90=0101 1010)
  + (+36) = 0010 0100
  - 54 1100 1010 = - 54

- (- 74) = 1011 0110 (+74=0100 1010)
  +(- 30) = 1110 0010 (+30=0001 1110)
  -104 1 1001 1000 = -104

Ví dụ cộng số nguyên có dấu bị tràn

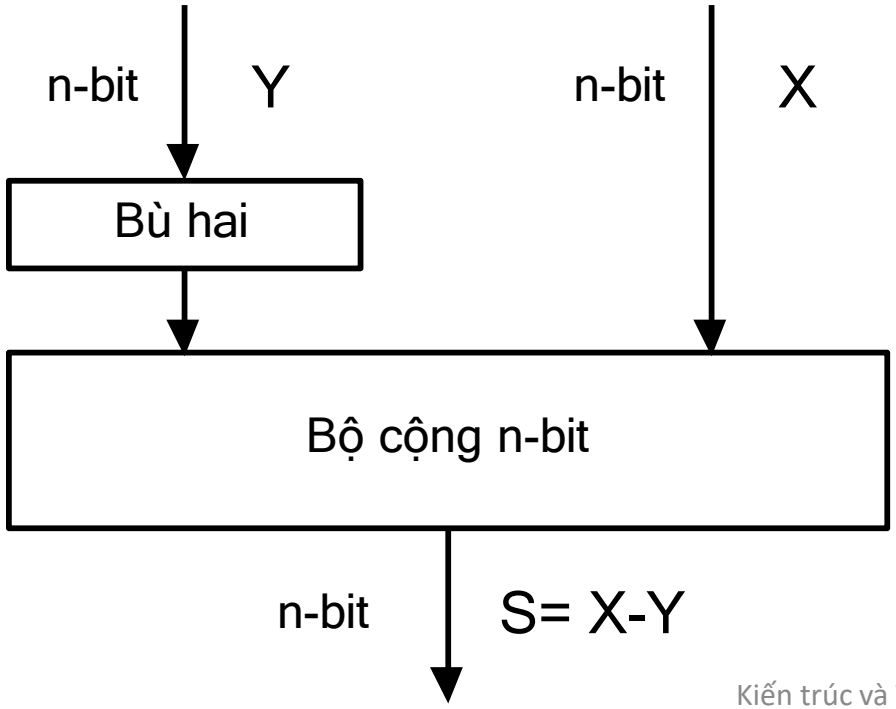
- (+ 75) = 0100 1011
  +(+ 82) = 0101 0010
  +157 1001 1101
  = - 128+16+8+4+1= -99 → sai

- (- 104) = 1001 1000 (+104=0110 1000)
  + (- 43) = 1101 0101 (+ 43 =0010 1011)
  - 147 1 0110 1101
  = 64+32+8+4+1= +109 → sai

- Cả hai ví dụ đều tràn vì tổng nằm ngoài dải biểu diễn [-128, +127]

4) Nguyên tắc thực hiện phép trừ

- Phép trừ hai số nguyên: X-Y = X+(-Y)
- Nguyên tắc: Lấy bù hai của Y để được -Y, rồi cộng với X



5) Phép nhân và phép chia số nguyên

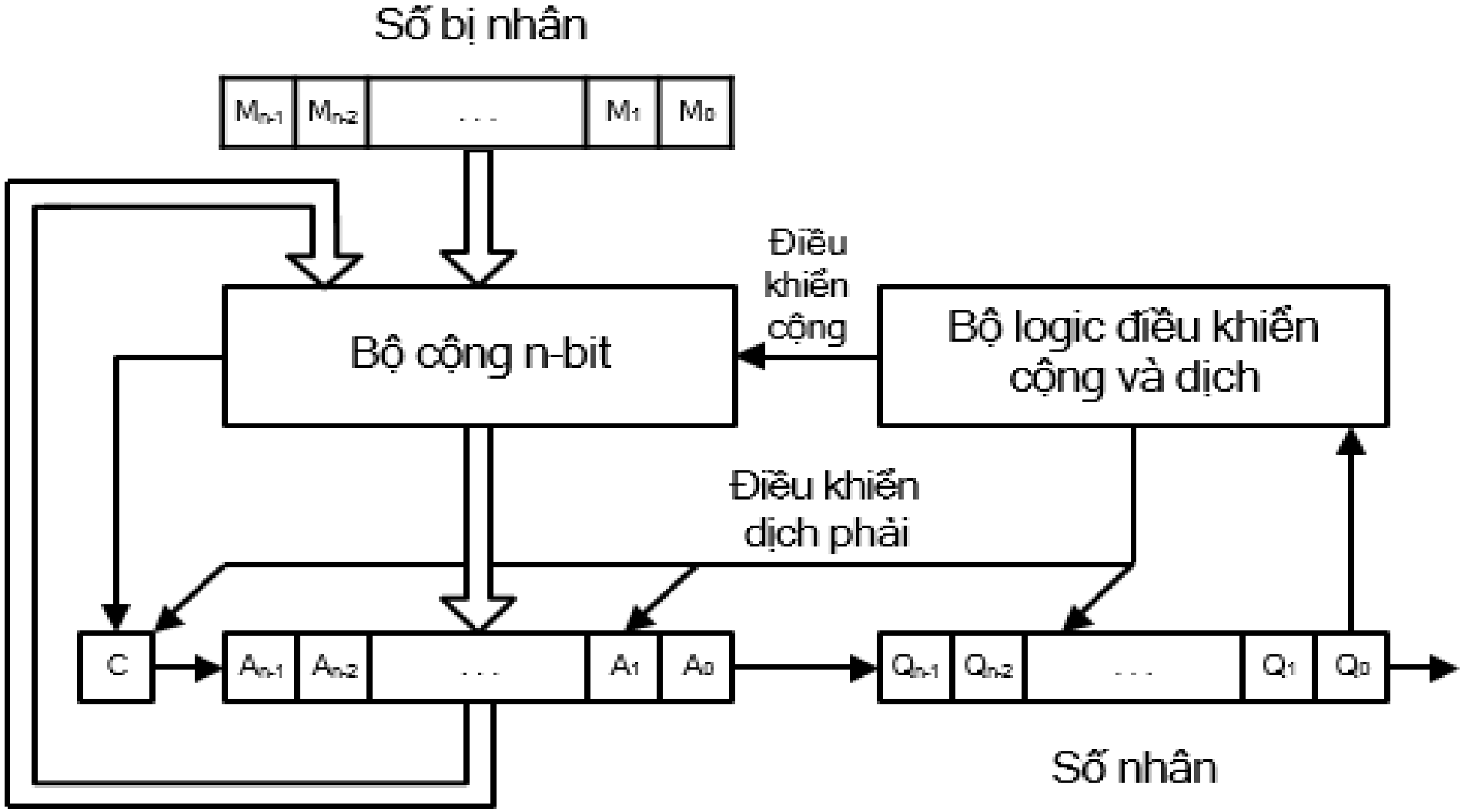
1. Nhân số nguyên không dấu

	1011	Số bị nhân (11)
x	1101	Số nhân (13)
	1011	Các tích riêng phần
	0000	
	1011	
	1011	
	10001111	Tích (143)

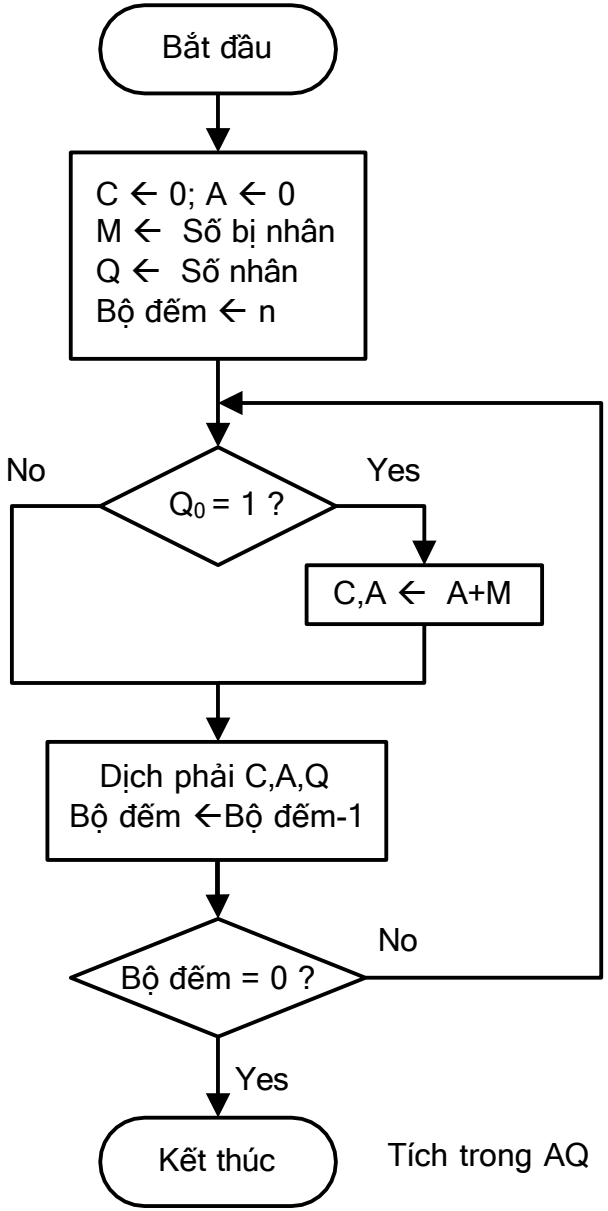
Nhân số nguyên không dấu (tiếp)

- 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)

Bộ nhân số nguyên không dấu



Lưu đồ nhân số nguyên không dấu



Ví dụ nhân số nguyên không dấu

- Số bị nhân M = 1011 (11)
- Số nhân Q = 1101 (13)
- Tích = 1000 1111 (143)

	C	A	Q	
■	0	0000	1101	Các giá trị khởi đầu
		+ 1011		
	0	1011	1101	A ← A + M
■	0	0101	1110	Dịch phải
■	0	0010	1111	Dịch phải
		+ 1011		
	0	1101	1111	A ← A + M
■	0	0110	1111	Dịch phải
		+ 1011		
	1	0001	1111	A ← A + M
■	0	1000	1111	Dịch phải

Ví dụ nhân số nguyên không dấu (tiếp)

- Số bị nhân M = 0110 (6)
- Số nhân Q = 0101 (5)
- Tích = (30)

	C	A	Q	
■	0	0000	0101	Các giá trị khởi đầu
		+ 0110		
	0	0110	0101	A ← A + M
■	0	0011	0010	Dịch phải
■	0	0001	1001	Dịch phải
		+ 0110		
	0	0111	1001	A ← A + M
■	0	0011	1100	Dịch phải
■	0	0001	1110	Dịch phải

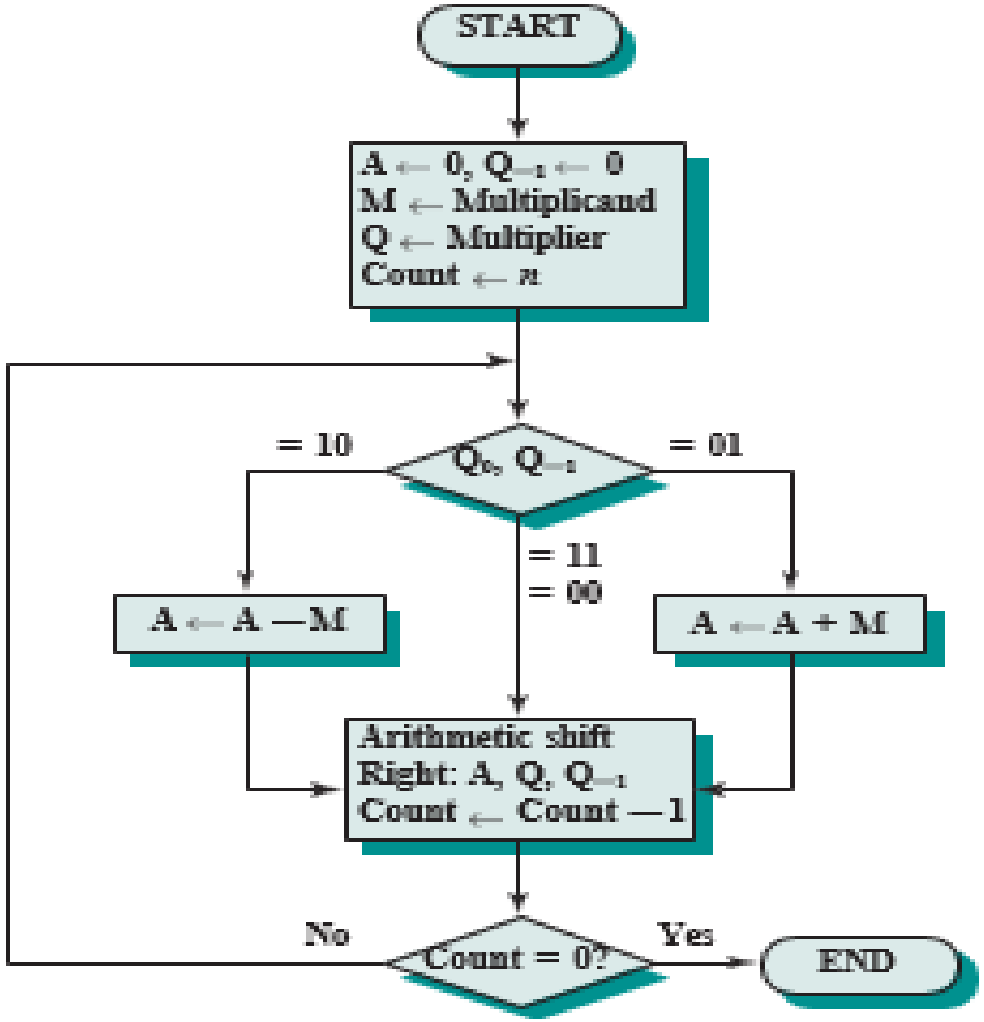
6) Nhân số nguyên có dấu

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

Sử dụng thuật giải 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 (lấy bù hai)

Thuật giải Booth



Lưu ý: Trong các trường hợp, sự dịch phải các bit trái nhất của A - bit A_{n-1} không chỉ dịch vào A_{n-2} nhưng cũng còn lại trong A_{n-1}, điều này được đòi hỏi để bảo tồn dấu của số trong A và Q

Ví dụ thuật giải Booth

Example Consider the multiplication of the two positive numbers $M = 0111$ (7) and $Q = 0011$ (3) and assuming that $n = 4$. The steps needed are tabulated below.

M	A	Q	$Q(-1)$			
0111	0000	0011	0	Initial value		c=4
0111	1001	0011	0	$A = A - M$		
0111	1100	1001	1	ASR	End cycle #1	c=3
0111	1110	0100	1	ASR	End cycle #2	c=2
0111	0101	0100	1	$A = A + M$		
0111	0010	1010	0	ASR	End cycle #3	c=1
0111	0001	0101	0	ASR	End cycle #4	c=0

+21 (correct result)

Ví dụ 2 thuật giải Booth: nhân 2 số khác dấu

Example Consider the multiplication of the two numbers $M = 0111$ (7) and $Q = 1101$ (-3) and assuming that $n = 4$. The steps needed are tabulated below.

M	A	Q	$Q(-1)$		
0111	0000	1101	0	Initial value	
0111	1001	1101	0	$A = A - M$	
0111	1100	1110	1	ASR	End cycle #1
0111	0011	1110	1	$A = A + M$	
0111	0001	1111	0	ASR	End cycle #2
0111	1010	1111	0	$A = A - M$	
0111	1101	0111	1	ASR	End cycle #3
0111	1110	1011	1	ASR	End cycle #4

-21 (correct result)

6) Chia số nguyên không dấu

Số bị chia 10010011 | 1011 Số chia

- 1011 00001101 Thương

001110

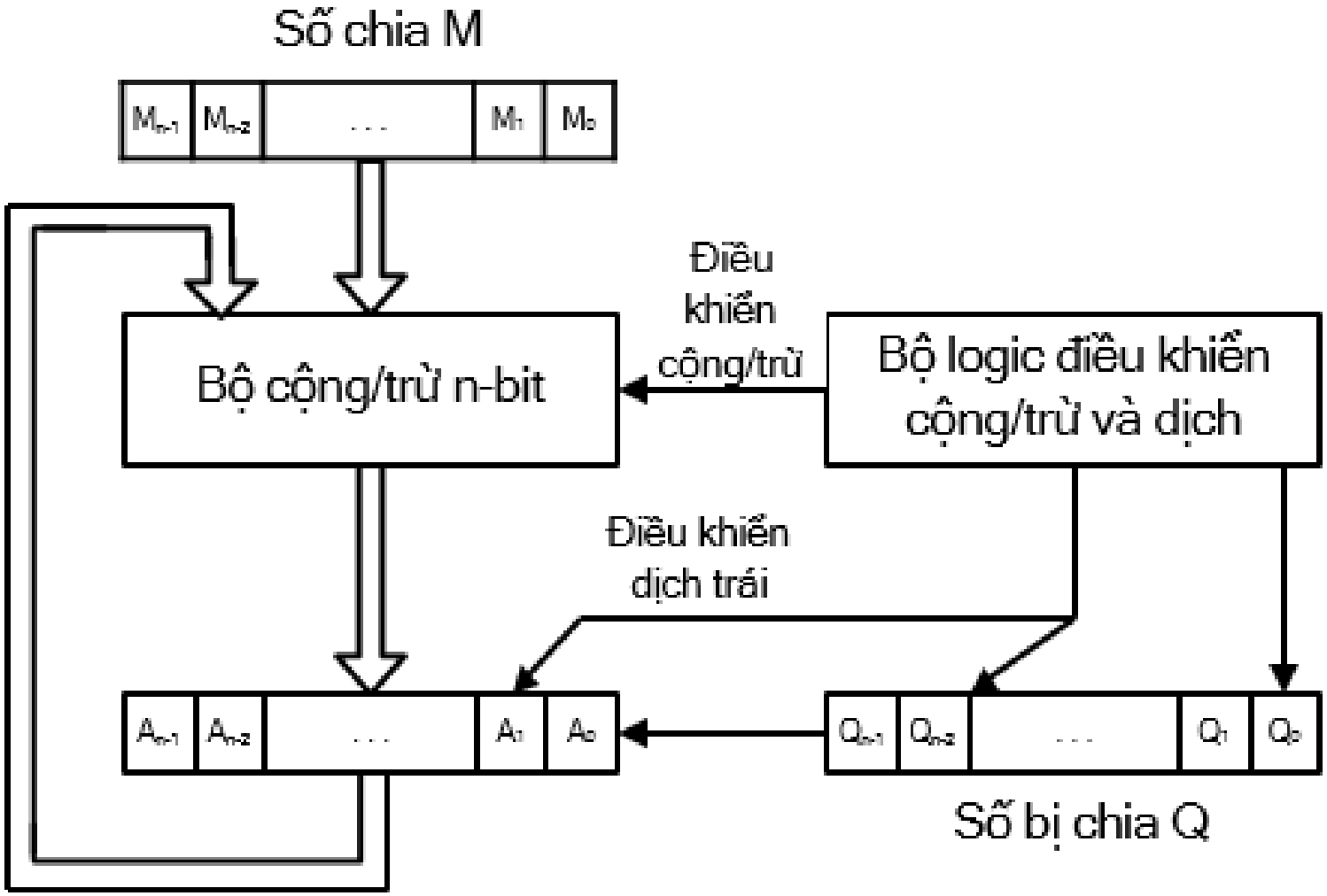
- 1011

001111

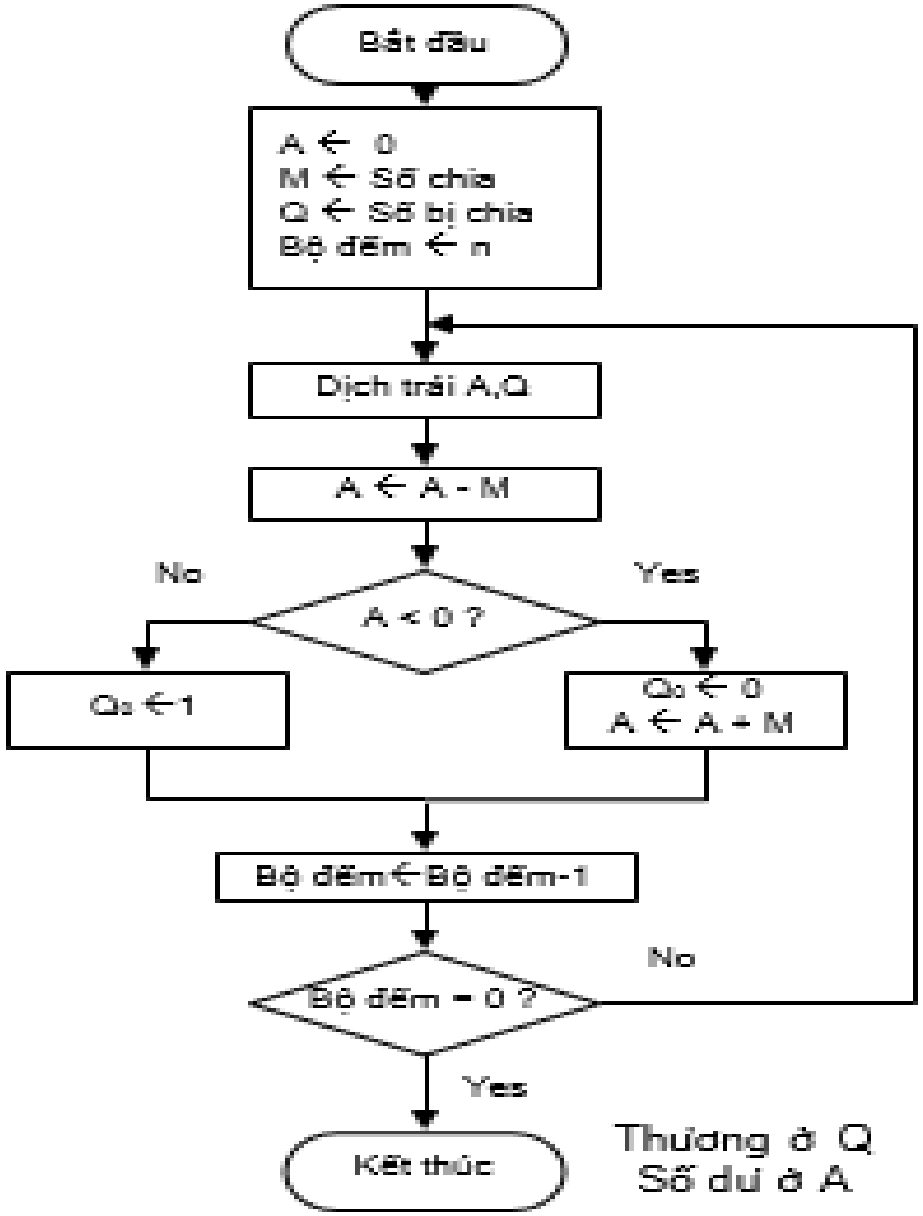
- 1011

100 Phần dư

Bộ chia số nguyên không dấu



Lưu đồ chia số nguyên không dấu



Ví dụ: Q = 1011 (11) M = 0011 (3) → -M = 1101

A	Q		
0000	1011		BĐ = 4
0001	0110	dịch trái	
1101			
1110		A = A - M < 0	
0011			
0001	0110	A = A + M	BĐ = 3
0010	1100	dịch trái	
1101			
1111		A = A - M < 0	
0011			
0010	1100	A = A + M	BĐ = 2
0101	1000	dịch trái	
1101			
0010		A = A - M > 0	
0010	1001		BĐ = 1
0101	0010	dịch trái	
1101			
0010		A = A - M > 0	
0010	0011		BĐ = 0
2	3		

7) 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:

  (Lưu ý: phép đảo dấu thực chất là thực hiện phép lấy bù hai)

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

2.2.2 Biểu diễn số thực dấu phẩy động

1. Nguyên tắc chung

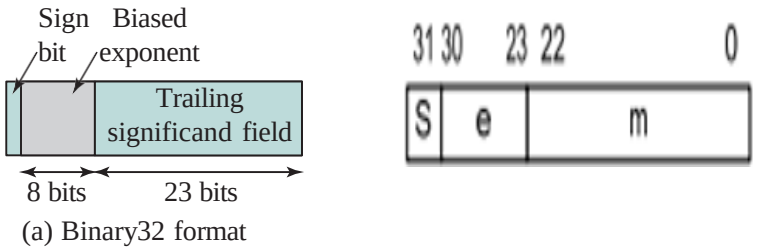
- Floating Point Number → biểu diễn cho số thực
- Tổng quát: một số thực X được biểu diễn theo kiểu số dấu phẩy động như sau:

$$X = \pm M * R^E$$

  - M là phần định trị (Mantissa),
  - R là cơ số (Radix),
  - E là phần mũ (Exponent).

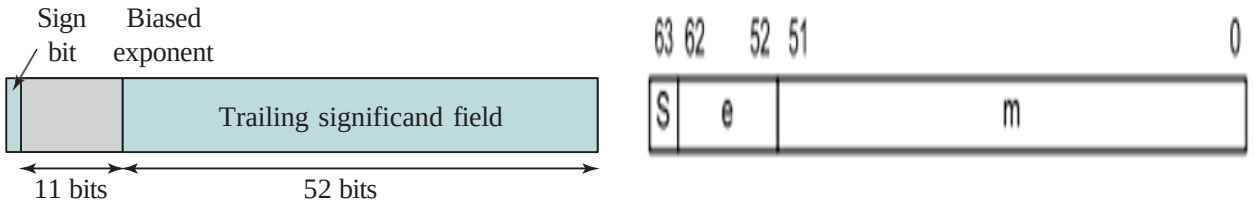
2. Chuẩn IEEE754-2008

- Cơ số R = 2
- Các dạng:
  - Dạng 32-bit



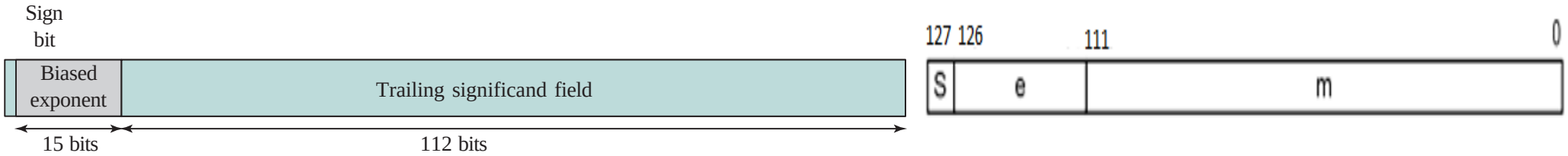
(a) Binary32 format

  - Dạng 64-bit



(b) Binary64 format

  - Dạng 128-bit



(c) Binary128 format

Dạng 32-bit

- S là bit dấu:
  - S = 0 → số dương
  - S = 1 → số âm
- e (8 bit) là giá trị dịch chuyển của phần mũ E:
  - e = E + 127 → phần mũ E = e - 127
- m (23 bit) là phần lẻ của phần định trị M:
  - M = 1.m
- Công thức xác định giá trị của số thực:

$$X = (-1)^S * 1.m * 2^{e-127}$$

$X = (-1)^S * 1.m * 2^{e-127}$

Ví dụ 1

Xác định giá trị của các số thực được biểu diễn bằng 32-bit sau đây:

- 1100 00010 101 0110 0000 0000 0000 0000
  - S = 1 → số âm
  - e = $1000\ 0010_{(2)} = 130_{(10)}$ → E = 130 - 127 = 3

  Vậy

  $X = -1.10101100_{(2)} * 2^3 = -1101.011_{(2)} = -13.375_{(10)}$

- 0011 1111 1 000 0000 0000 0000 0000 0000 = ?

Ví dụ 2

Biểu diễn số thực $X = 83.75_{(10)}$ 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 → e = 127 + 6 = $133_{(10)}$ = $1000\ 0101_{(2)}$
- Vậy:

X = 0100 00101 010 0111 1000 0000 0000 0000

Các qui ước đặc biệt

- Các bit của e bằng 0, các bit của m bằng 0, thì $X = \pm 0$

  x000 0000 0000 0000 0000 0000 0000 0000 → $X = \pm 0$

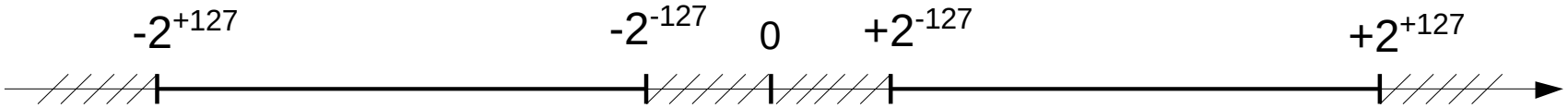
- Các bit của e bằng 1, các bit của m bằng 0, thì $X = \pm \infty$

  x111 1111 1000 0000 0000 0000 0000 0000 → $X = \pm \infty$

- 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)

Dải giá trị biểu diễn

- 2^{-127} đến 2^{+127}
- 10^{-38} đến 10^{+38}



Dạng 64-bit

- S là bit dấu
- e (11 bit) là giá trị dịch chuyển của phần mũ E:
  - e = E + 1023 → phần mũ E = e - 1023
- m (52 bit): phần lẻ của phần định trị M
- Giá trị số thực:

$$X = (-1)^S * 1.m * 2^{e-1023}$$

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

Dạng 128-bit

- S là bit dấu
- e (15 bit) là giá trị dịch chuyển của phần mũ E:
  - e = E + 16383 → phần mũ E = e - 16383
- m (112 bit): phần lẻ của phần định trị M
- Giá trị số thực:

$$X = (-1)^S * 1.m * 2^{e-16383}$$

- Dải giá trị biểu diễn: 10^{-4932} đến 10^{+4932}

Thực hiện phép toán số dấu phẩy động

- $X1 = M1 * R^{E1}$
- $X2 = M2 * R^{E2}$
- Ta có
  - $X1 * X2 = (M1 * M2) * R^{E1+E2}$
  - $X1 / X2 = (M1 / M2) * R^{E1-E2}$
  - $X1 \pm X2 = (M1 * R^{E1-E2} \pm M2) * R^{E2}$, với $E2 \geq E1$

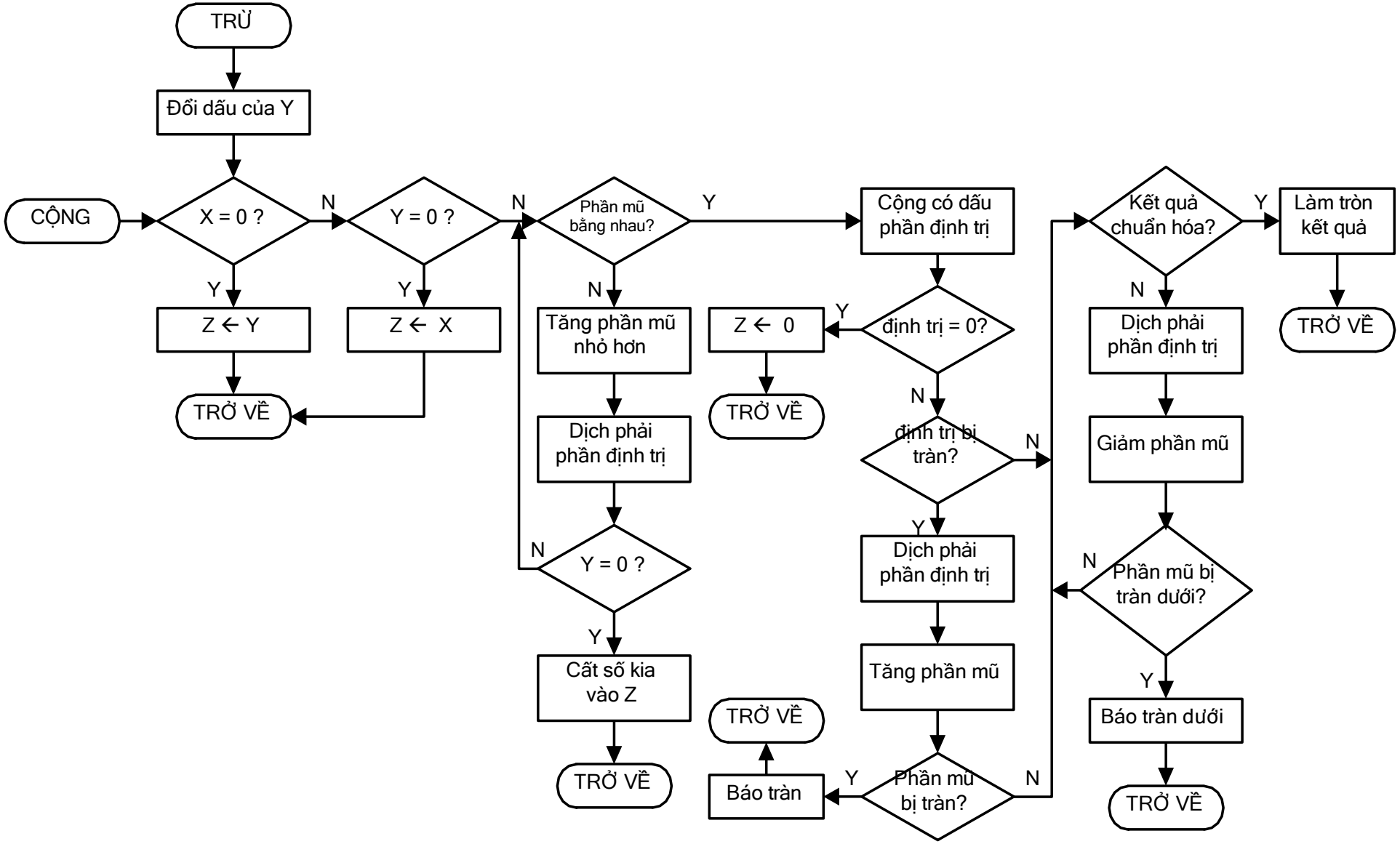
Các khả năng tràn số

- Tràn trên số mũ (Exponent Overflow): mũ dương vượt ra khỏi giá trị cực đại của số mũ dương có thể (→ ∞)
- Tràn dưới số mũ (Exponent Underflow): mũ âm vượt ra khỏi giá trị cực đại của số mũ âm có thể (→ 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

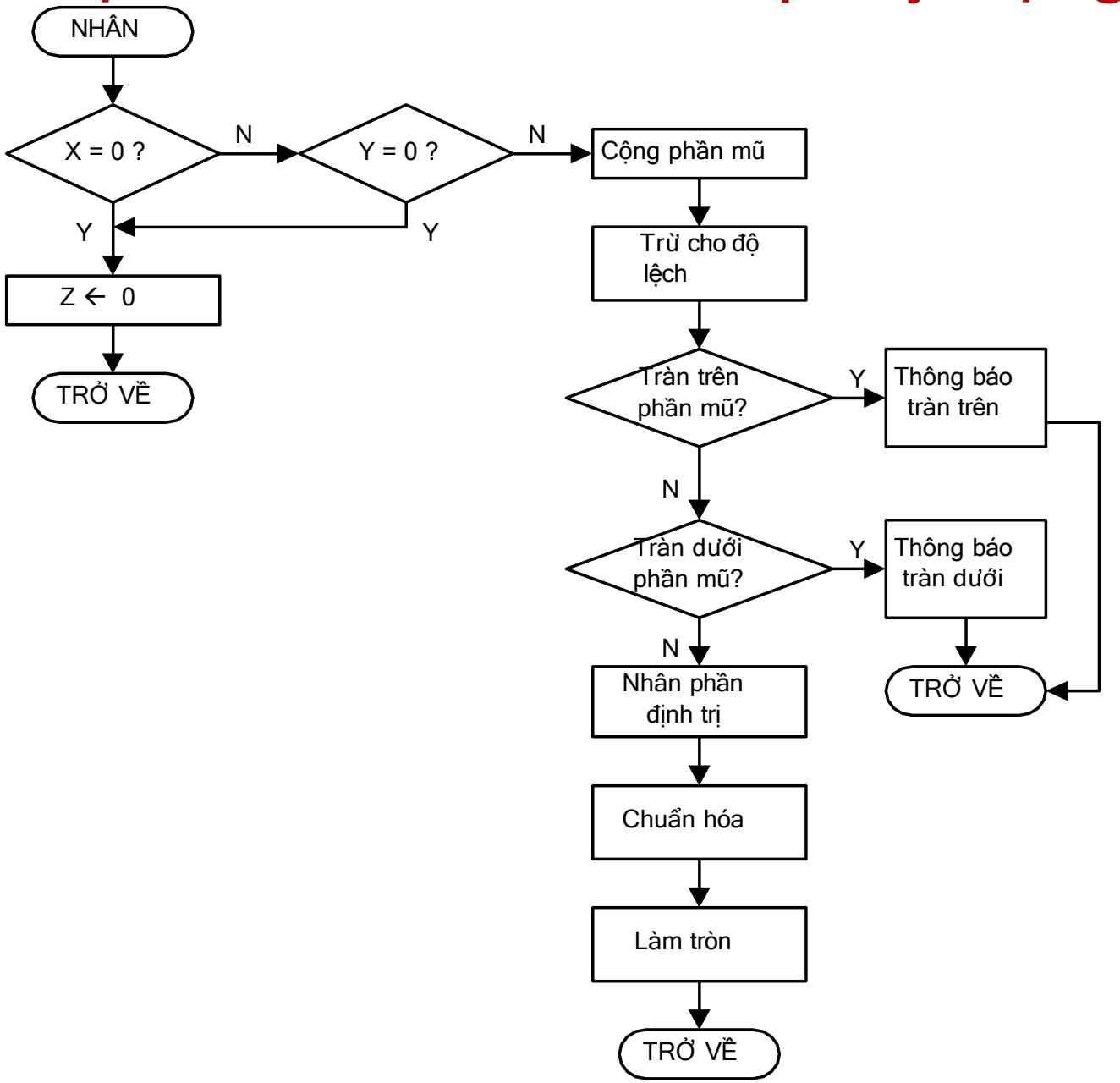
Phép cộng và phép trừ

- Kiểm tra các số hạng có bằng 0 hay không
- Hiệu chỉnh phần định trị
- Cộng hoặc trừ phần định trị
- Chuẩn hoá kết quả

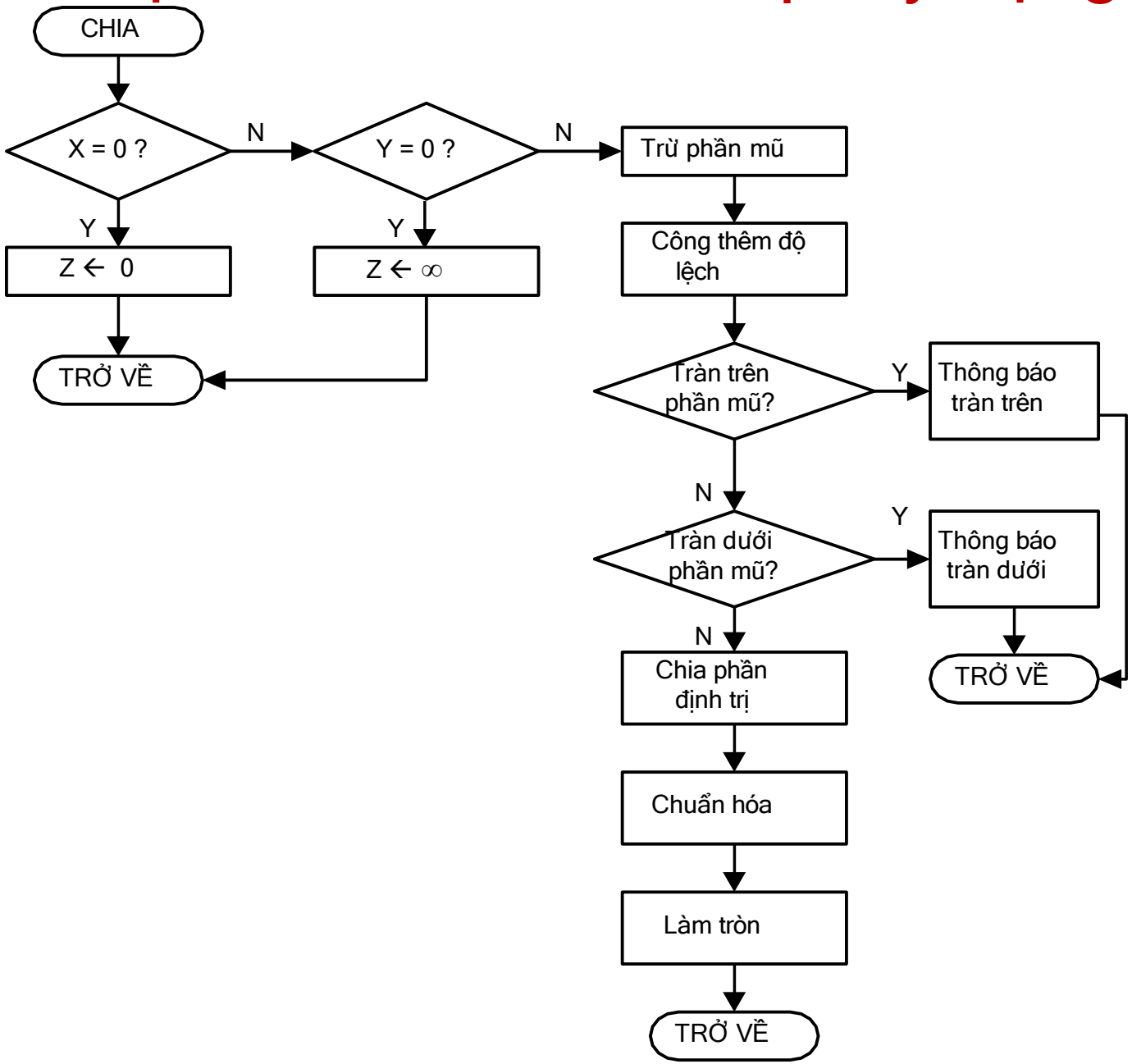
Thuật toán cộng/trừ số dấu phẩy động



Thuật toán nhân số dấu phẩy động



Thuật toán chia số dấu phẩy động



Đại số Boole

- Đại số Boole sử dụng các biến luận lý và phép toán luận lý
- Biến luận lý có thể nhận giá trị 1 (TRUE) hoặc 0 (FALSE)
- Các phép toán luận lý cơ bản: AND, OR và NOT
  - A AND B : $A \bullet B$ hay AB
  - A OR B : $A + B$
  - NOT A : $\overline{A}$
  - Thứ tự ưu tiên: NOT > AND > OR
- Thêm các phép toán luận lý: NAND, NOR, XOR
  - A NAND B: $\overline{A \bullet B}$
  - A NOR B : $\overline{A + B}$
  - A XOR B: $A \oplus B = A \bullet \overline{B} + \overline{A} \bullet B$

Phép toán đại số Boole với hai biến

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

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

A	NOT A $\overline{A}$
0	1
1	0

NOT là phép toán 1 biến

A	B	A NAND B $\overline{A \bullet B}$
0	0	1
0	1	1
1	0	1
1	1	0

A	B	A NOR B $\overline{A + B}$
0	0	1
0	1	0
1	0	0
1	1	0

A	B	A XOR B $A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

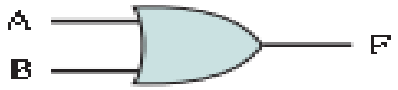
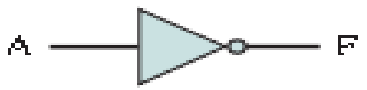
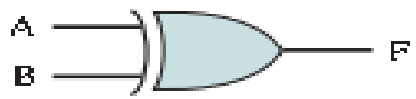
Các đồng nhất thức của đại số Boole

$A \cdot B = B \cdot A$	$A + B = B + A$
$A \cdot (B + C) = (A \cdot B) + (A \cdot C)$	$A + (B \cdot C) = (A + B) \cdot (A + C)$
$1 \cdot A = A$	$0 + A = A$
$A \cdot \overline{A} = 0$	$A + \overline{A} = 1$
$0 \cdot A = 0$	$1 + A = 1$
$A \cdot A = A$	$A + A = A$
$A \cdot (B \cdot C) = (A \cdot B) \cdot C$	$A + (B + C) = (A + B) + C$
$\overline{A \cdot B} = \overline{A} + \overline{B}$ (Định lý De Morgan)	$\overline{A + B} = \overline{A} \cdot \overline{B}$ (Định lý De Morgan)

Các cổng luận lý (logic gates)

- Thực hiện các hàm luận lý:
  - NOT, AND, OR, NAND, NOR, XOR
- Cổng luận lý một đầu vào:
  - Cổng NOT
- Cổng hai đầu vào:
  - AND, OR, XOR, NAND, NOR
- Cổng nhiều đầu vào

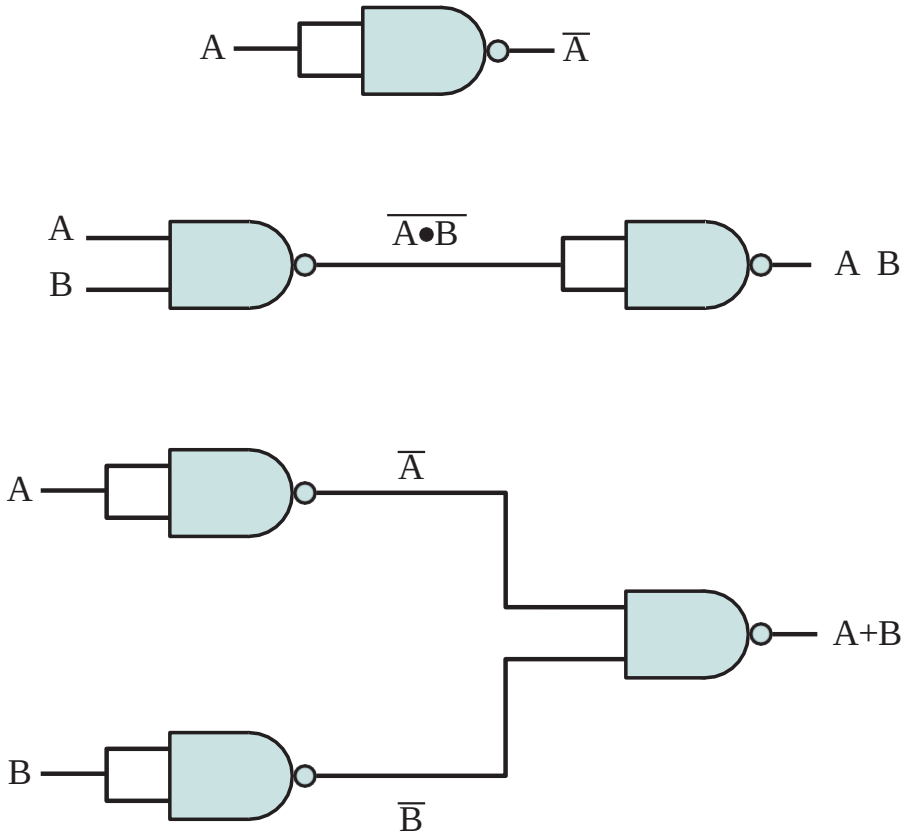
Ký hiệu các cổng luận lý

Name	Graphical Symbol	Algebraic Function	Truth Table
AND		$F = A \cdot B$ or $F = AB$	A B | F 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1
OR		$F = A + B$	A B | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1
NOT		$F = \overline{A}$ or $F = A'$	A | F 0 | 1 1 | 0
NAND		$F = \overline{AB}$	A B | F 0 0 | 1 0 1 | 1 1 0 | 1 1 1 | 0
NOR		$F = \overline{A + B}$	A B | F 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 0
XOR		$F = A \oplus B$	A B | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 0

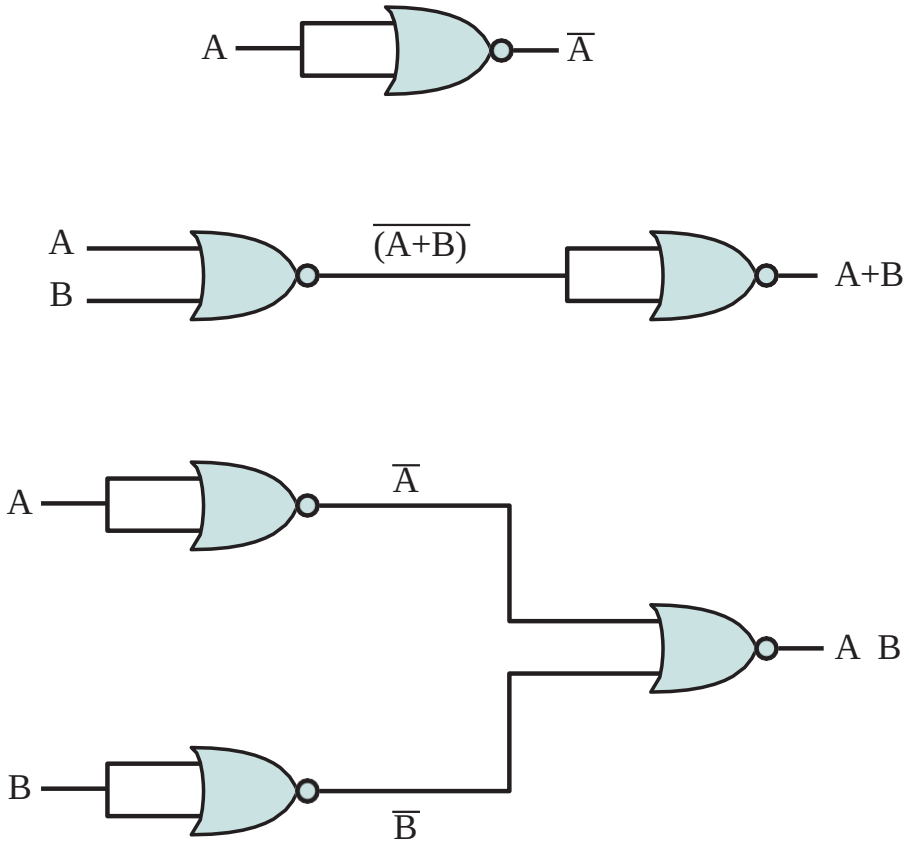
Tập đầy đủ

- Là tập các cổng có thể thực hiện được bất kỳ hàm logic nào từ các cổng của tập đó
- Một số ví dụ về tập đầy đủ:
  - {AND, OR, NOT}
  - {AND, NOT}
  - {OR, NOT}
  - {NAND}
  - {NOR}

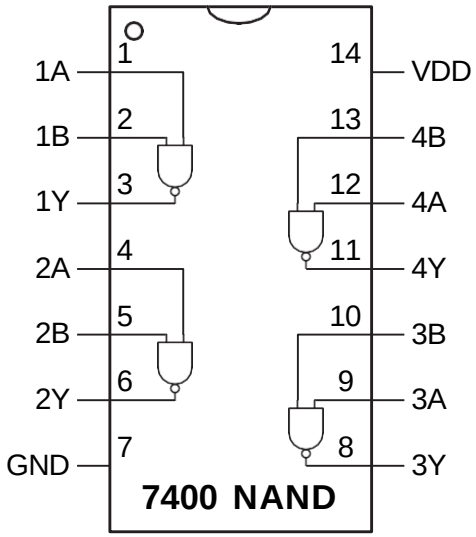
Sử dụng cổng NAND

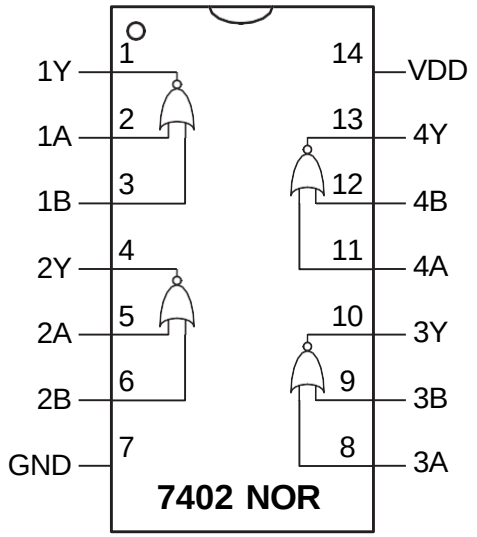


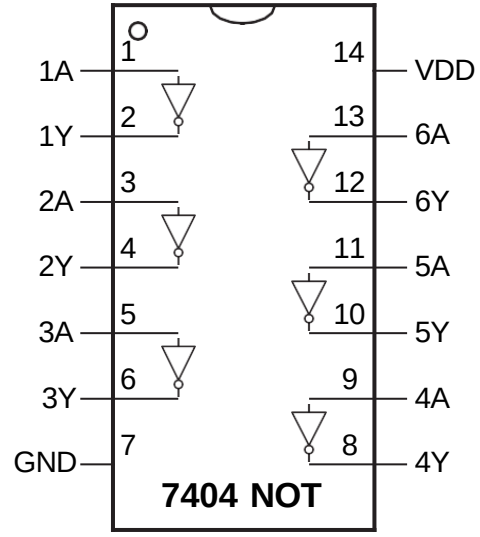
Sử dụng cổng NOR

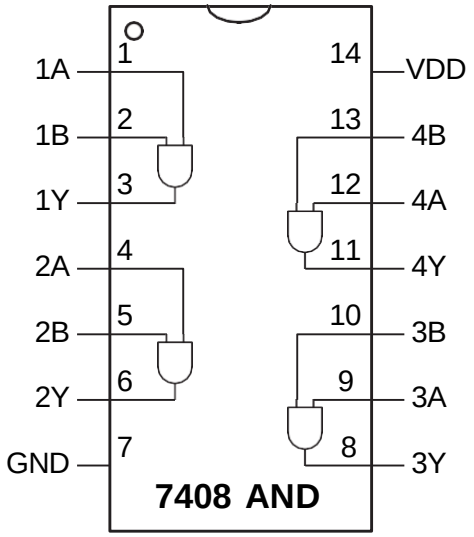


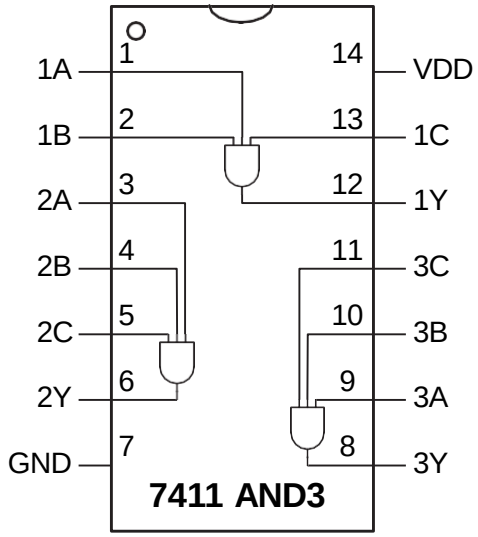
Một số vi mạch luận lý

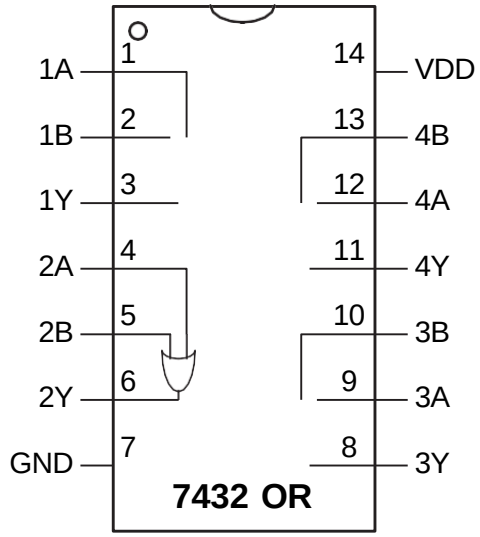












2.4. Mạch tổ hợp

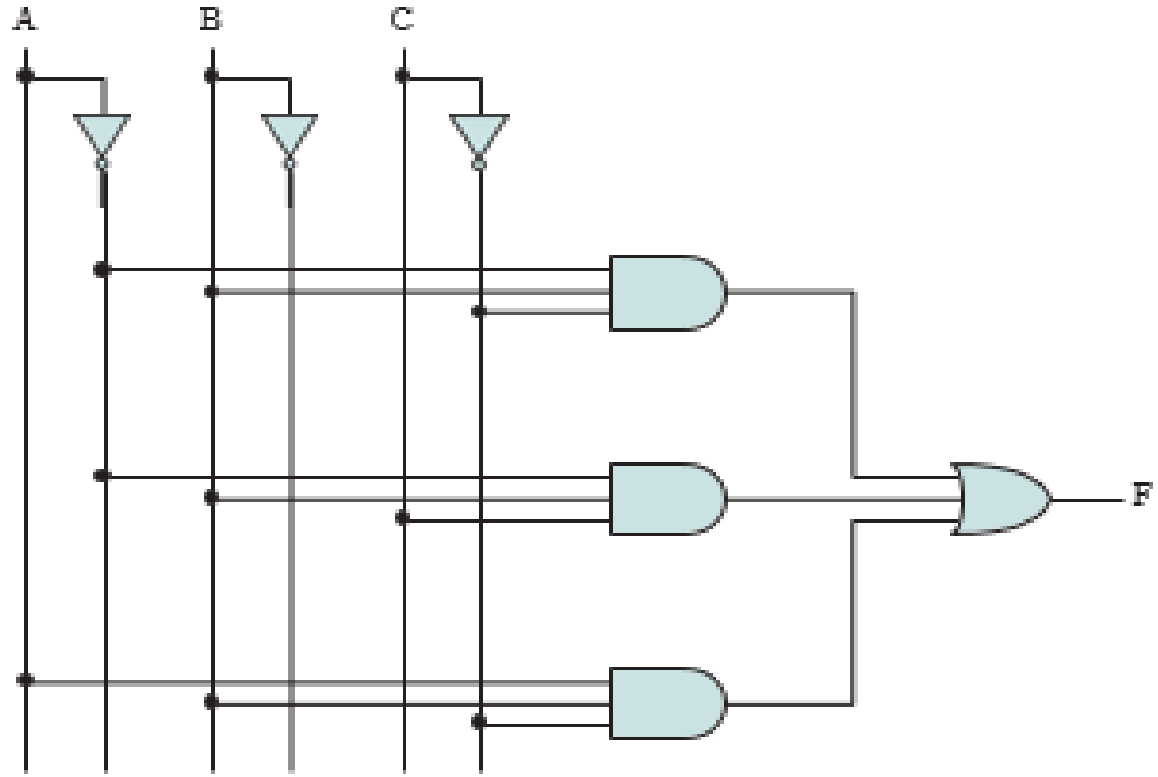
- Mạch luận lý (logic circuit) là mạch bao gồm:
  - Các đầu vào (Inputs)
  - Các đầu ra (Outputs)
  - Đặc tả chức năng (Functional specification)
  - Đặc tả thời gian (Timing specification)
- Các kiểu mạch luận lý:
  - Mạch tổ hợp (Combinational Circuits)
    - Mạch không nhớ
    - Đầu ra được xác định bởi các giá trị hiện tại của đầu vào
  - Mạch dãy (Sequential Circuits)
    - Mạch có nhớ
    - Đầu ra được xác định bởi các giá trị trước đó và giá trị hiện tại của đầu vào

Mạch tổ hợp

- Mạch tổ hợp là mạch logic trong đó đầu ra chỉ phụ thuộc đầu vào ở thời điểm hiện tại
- Là mạch không nhớ và được thực hiện bằng các cổng logic
- Mạch tổ hợp có thể được định nghĩa theo ba cách:
  - Bảng thật (True Table)
  - Dạng sơ đồ
  - Phương trình Boole

Ví dụ

Đầu vào			Đầu ra
A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

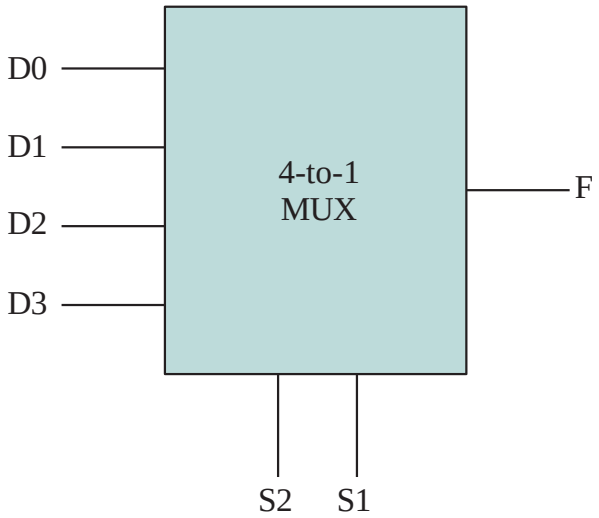


$$F = \overline{A}B\overline{C} + \overline{A}BC + AB\overline{C}$$

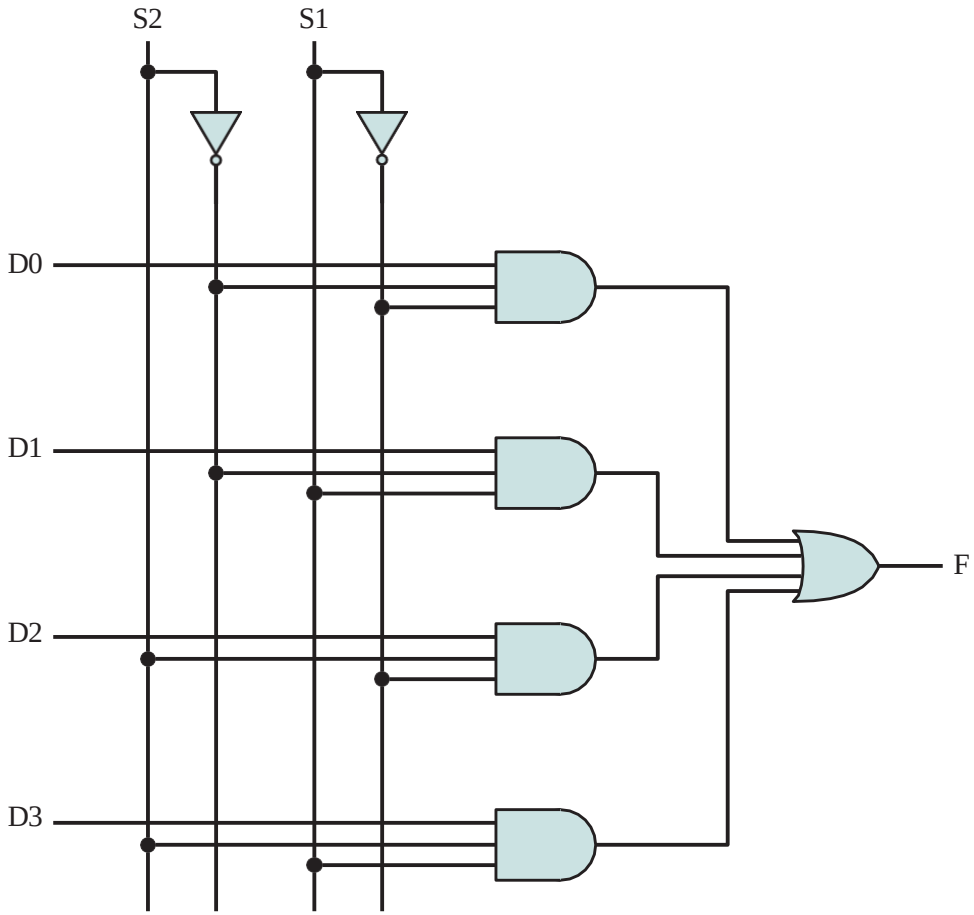
Bộ ghép kênh (Multiplexer - MUX)

- 2^n đầu vào dữ liệu
- n đầu vào chọn
- 1 đầu ra dữ liệu
- Mỗi tổ hợp đầu vào chọn (S) xác định đầu vào dữ liệu nào (D) sẽ được nối với đầu ra (F)

Bộ ghép kênh 4 đầu vào



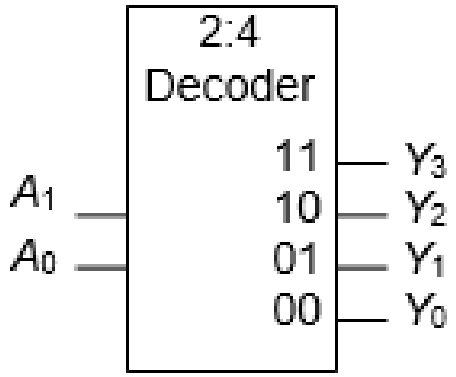
Đầu vào chọn		Đầu ra
S2	S1	F
0	0	D0
0	1	D1
1	0	D2
1	1	D3



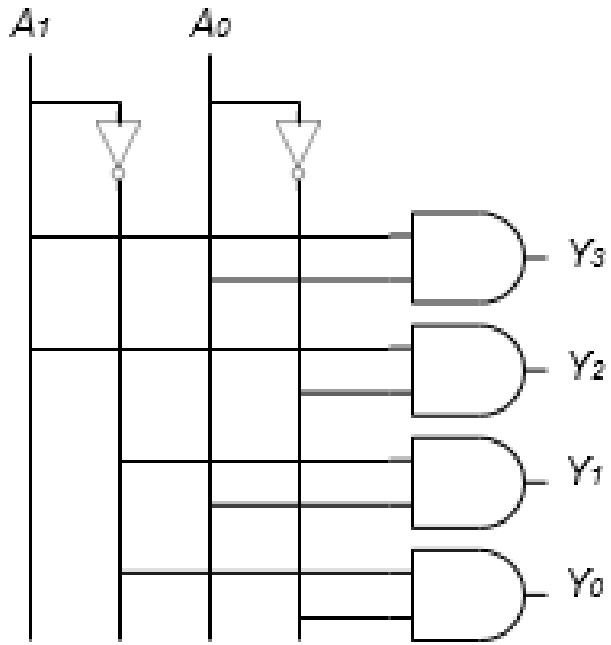
$$F = D0 \bullet \overline{S2} \bullet \overline{S1} + D1 \bullet \overline{S2} \bullet S1 + D2 \bullet S2 \bullet \overline{S1} + D3 \bullet S2 \bullet S1$$

Bộ giải mã (Decoder)

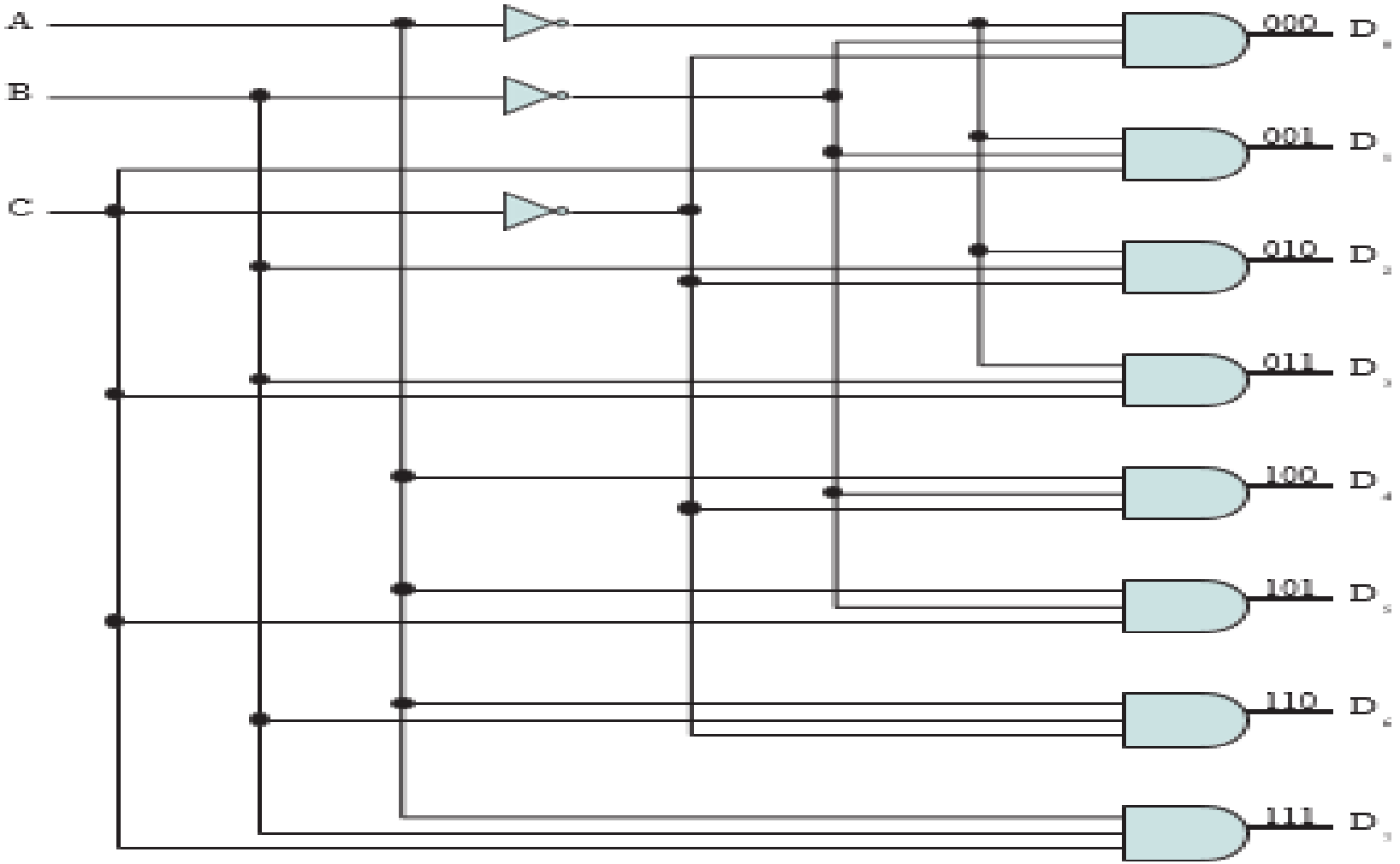
- *N* đầu vào, 2^N đầu ra
- Với một tổ hợp của N đầu vào, chỉ có một đầu ra tích cực (khác với các đầu ra còn lại)
- Ví dụ: Bộ giải mã 2 ra 4



A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0



Thực hiện bộ giải mã 3 ra 8

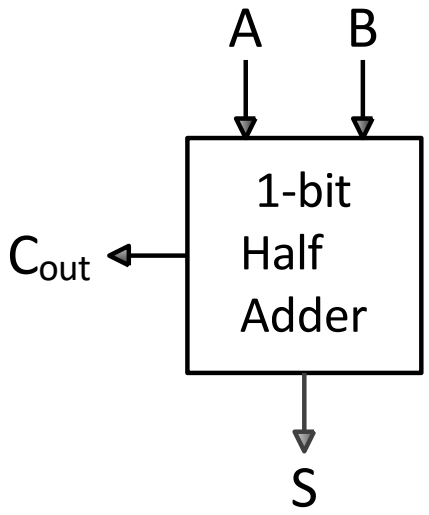


Bộ cộng

- Bộ cộng bán phần 1-bit (Half-adder)
  - Cộng hai bit tạo ra bit tổng và bit nhớ ra
- Bộ cộng toàn phần 1-bit (Full-adder)
  - Cộng 3 bit
  - Cho phép xây dựng bộ cộng N-bit

Bộ cộng bán phần 1-bit

0 + 0 0	0 + 1 1	1 + 0 1	1 + 1 10

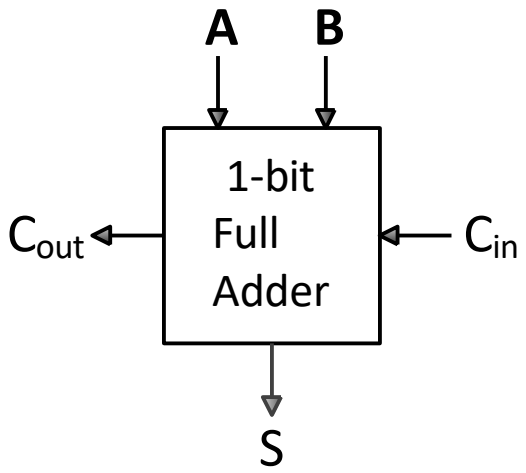


Đầu vào		Đầu ra	
A	B	S	C_{out}
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$$S = A \oplus B$$

$$C_{out} = AB$$

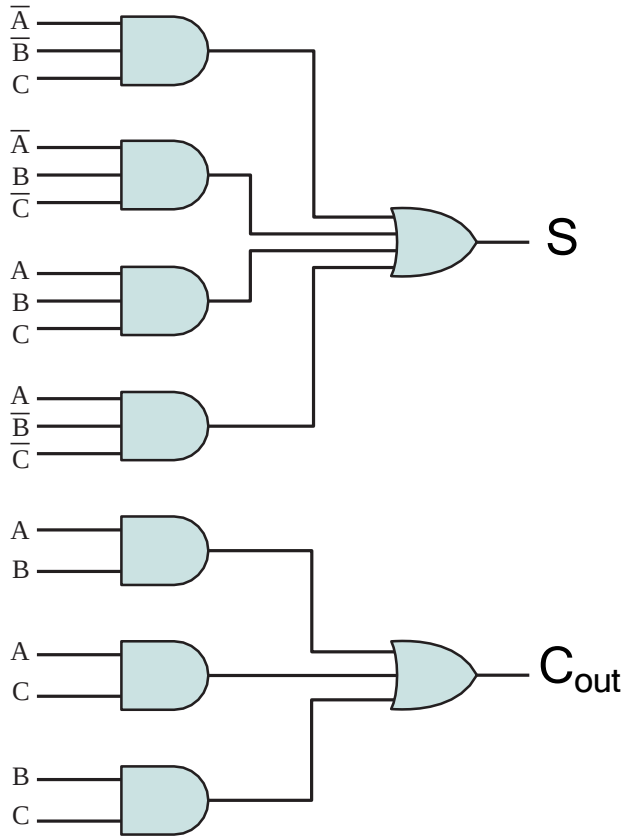
Bộ cộng toàn phần 1-bit



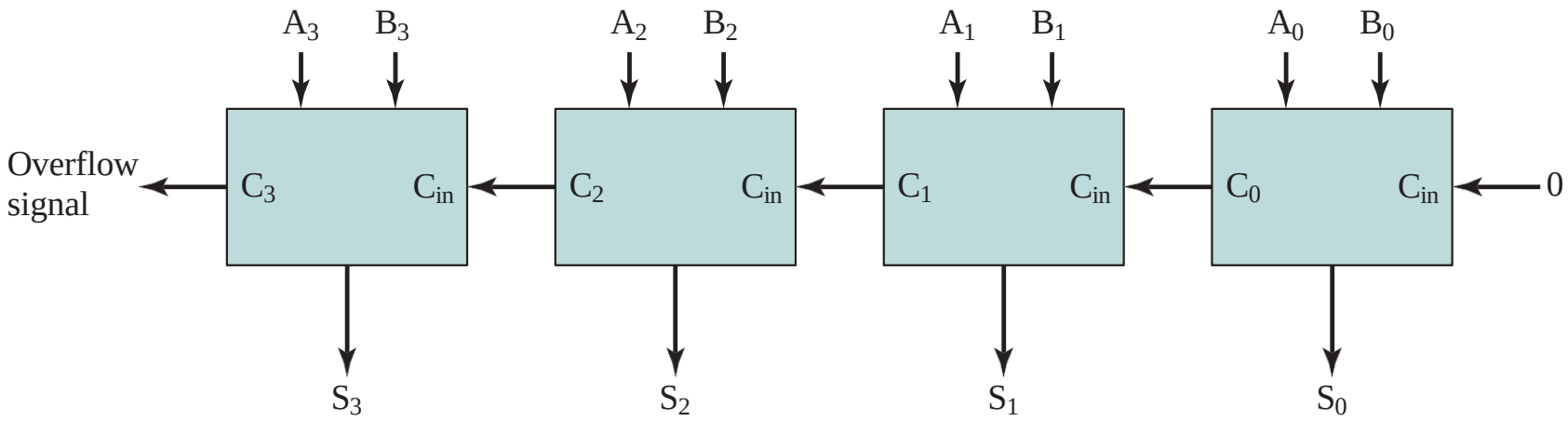
Đầu vào			Đầu ra	
C_{in}	A	B	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

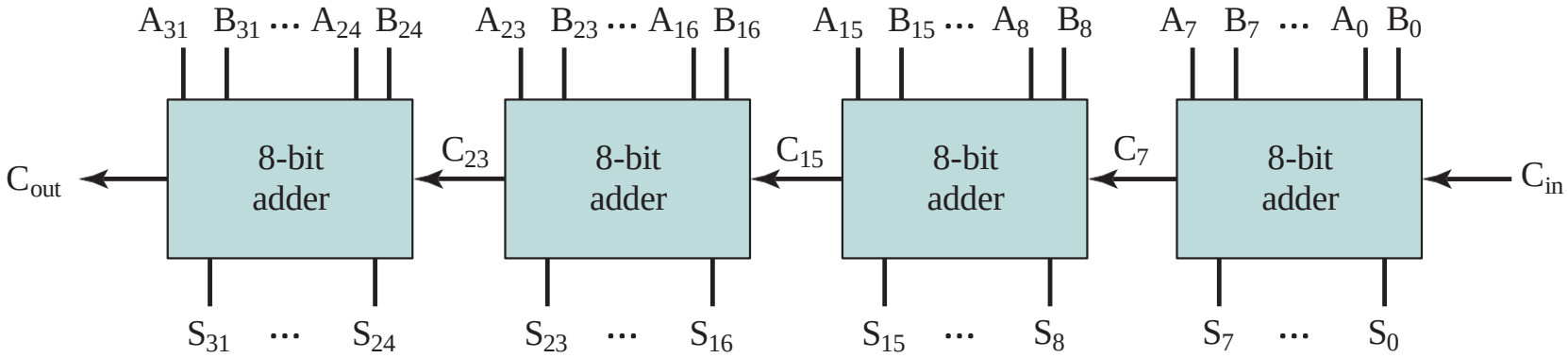
$$S = \overline{A}\overline{B}C + \overline{A}B\overline{C} + ABC + A\overline{B}\overline{C}$$

$$C_{out} = AB + AC + BC$$



Bộ cộng 4-bit và bộ cộng 32-bit

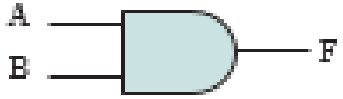
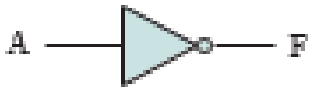
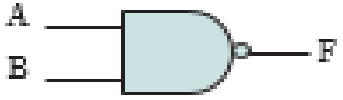
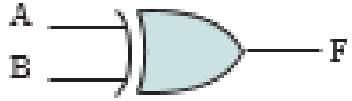




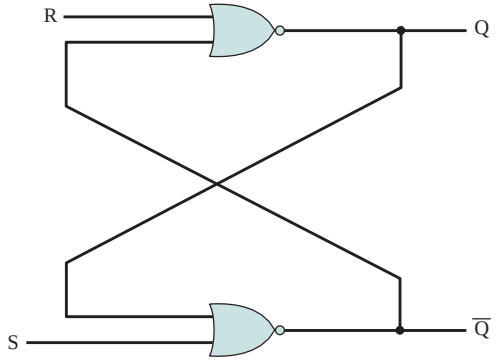
Mạch dãy

- Mạch dãy là mạch logic trong đó đầu ra phụ thuộc giá trị đầu vào ở thời điểm hiện tại và đầu vào ở thời điểm quá khứ
- Là mạch có nhớ, được thực hiện bằng phần tử nhớ (Latch, Flip-Flop) và có thể kết hợp với các cổng logic

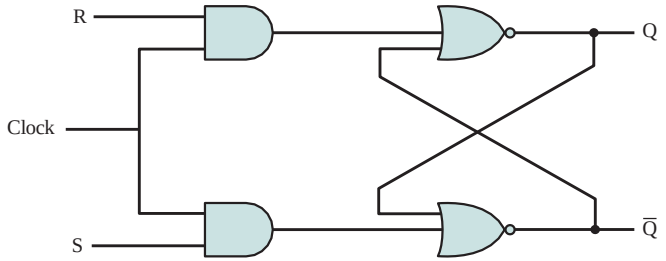
Các Flip-Flop cơ bản

Name	Graphical Symbol	Algebraic Function	Truth Table
AND		F = A • B or F = AB	A B | F 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1
OR		F = A + B	A B | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1
NOT		$F = \overline{A}$ or F = A'	A | F 0 | 1 1 | 0
NAND		$F = \overline{AB}$	A B | F 0 0 | 1 0 1 | 1 1 0 | 1 1 1 | 0
NOR		$F = \overline{A + B}$	A B | F 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 0
XOR		F = A ⊕ B	A B | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 0

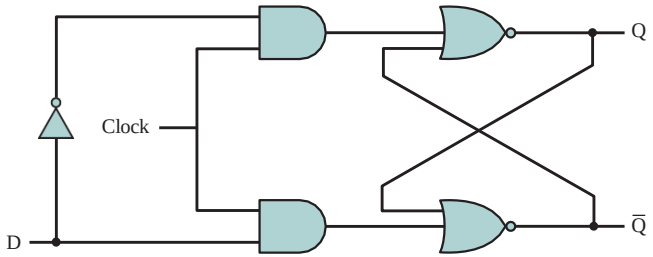
S-R Latch và các Flip-Flop



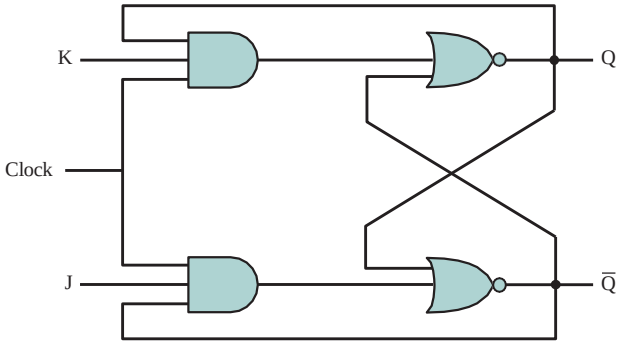
S-R Latch



S-R Flip-Flop

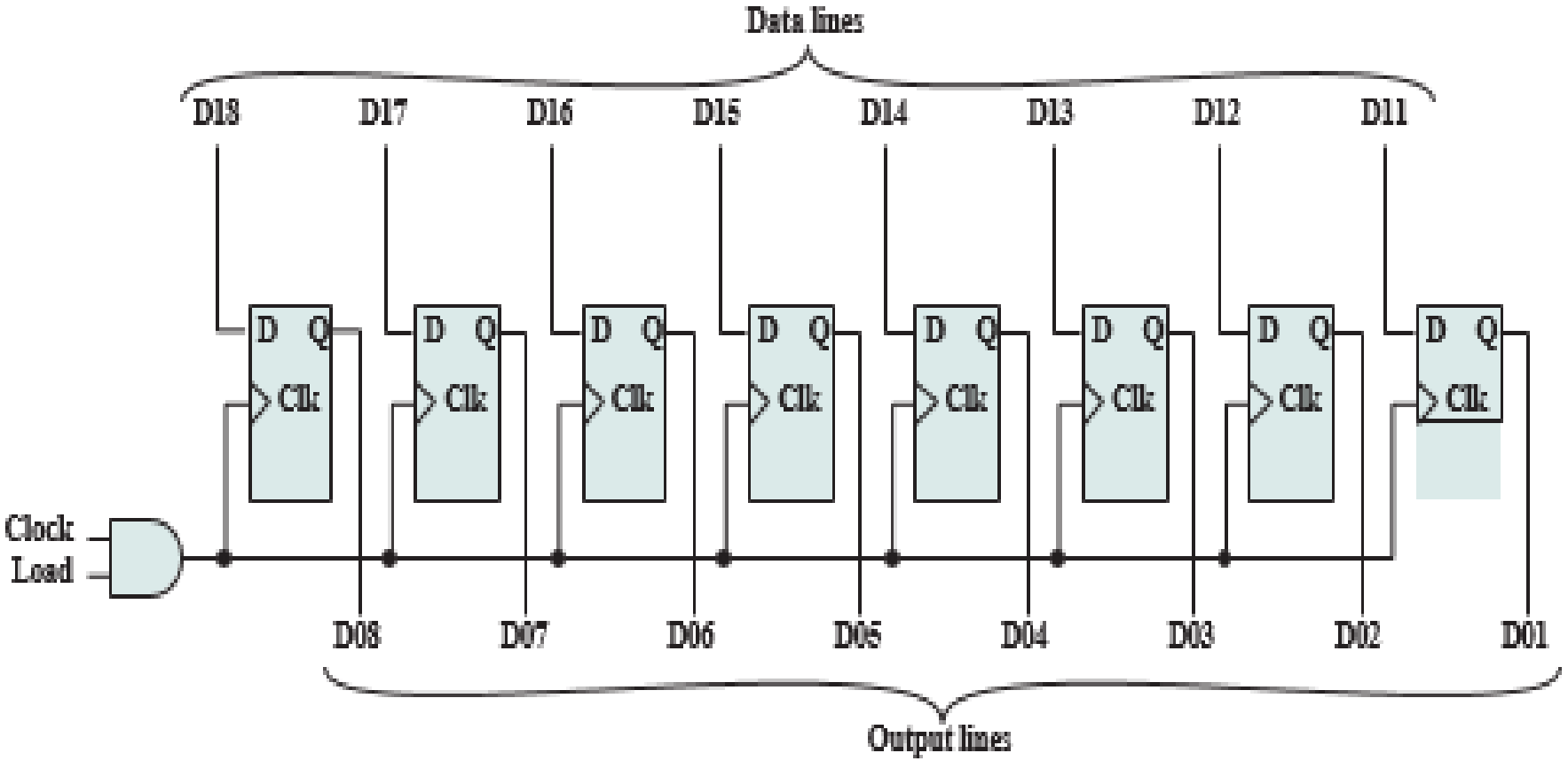


D Flip Flop

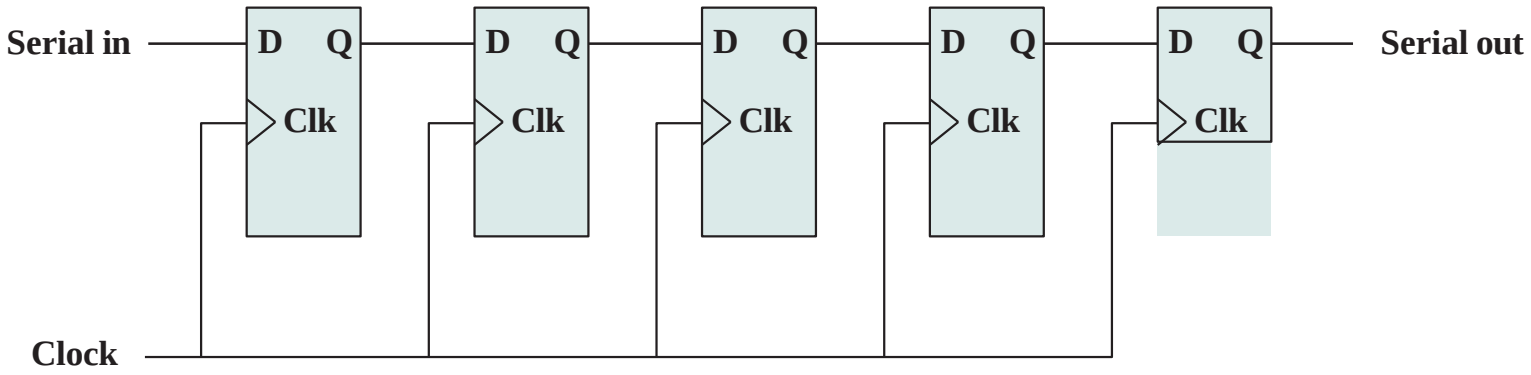


J-K Flip-Flop

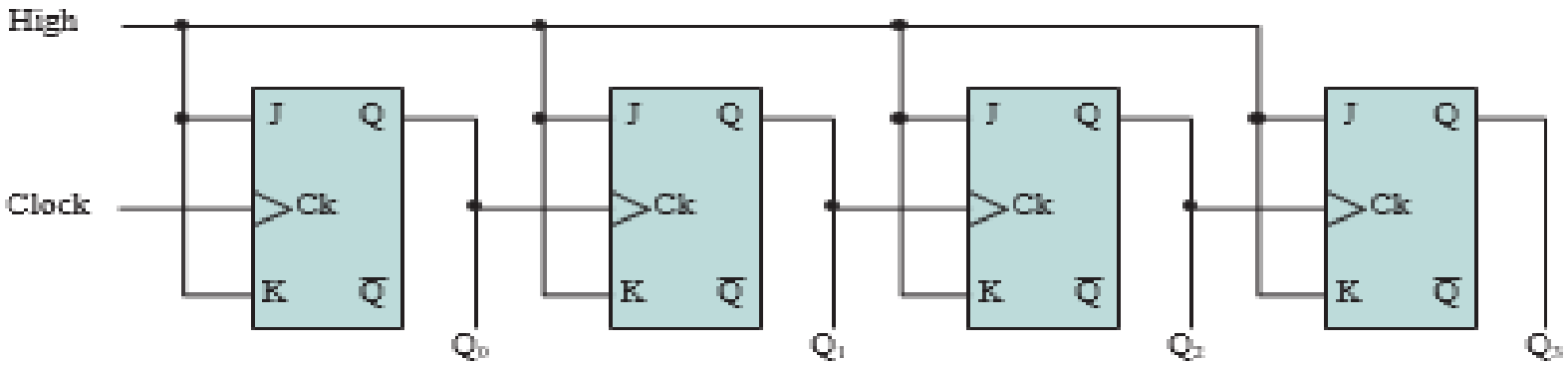
Thanh ghi 8-bit song song



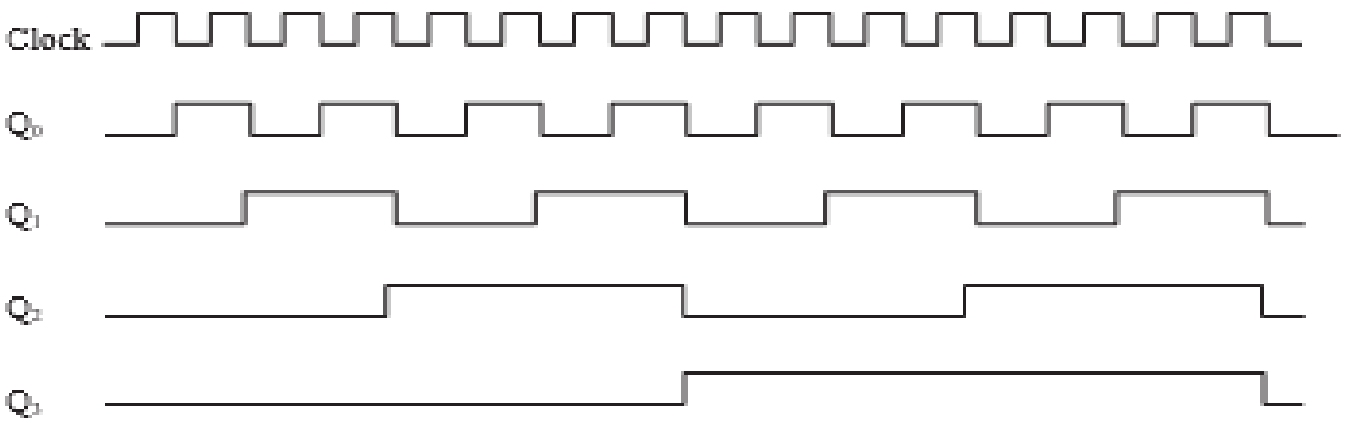
Thanh ghi dịch 5-bit



Bộ đếm 4-bit



(a) Sequential circuit



(b) Timing diagram

2.5. Mã

Các bảng mã như ASCII hay Unicode được dùng để mã hóa ký tự thành mã nhị phân, giúp máy tính hiểu, lưu trữ và xử lý văn bản.

Bảng mã (character encoding table): là quy tắc ánh xạ giữa ký tự (chữ cái, số, ký hiệu) và giá trị nhị phân (số) mà máy tính có thể hiểu được.

Ví dụ:Ký tự 'A' → trong ASCII là 65 → máy tính lưu dưới dạng nhị phân: 01000001.

Vai trò của bảng mã:

Mục đích chính	Giải thích
Lưu trữ	Chuyển ký tự thành mã số để lưu trữ trong bộ nhớ.
Truyền tải	Đảm bảo văn bản có thể gửi qua mạng, lưu file mà không bị lỗi ký tự.
Tương thích quốc tế	Cho phép thể hiện nhiều ngôn ngữ khác nhau (như tiếng Việt, Trung, Nhật...).

BẢNG MÃ ASCII 7 BIT

Dec	Hex	Oct	Chr	Dec	Hex	Oct	HTML	Chr	Dec	Hex	Oct	HTML	Chr	Dec	Hex	Oct	HTML	Chr
0	0	000	NULL	32	20	040	&#032;	Space	64	40	100	&#064;	@	96	60	140	&#096;	`
1	1	001	Start of Header	33	21	041	&#033;	!	65	41	101	&#065;	A	97	61	141	&#097;	a
2	2	002	Start of Text	34	22	042	&#034;	"	66	42	102	&#066;	B	98	62	142	&#098;	b
3	3	003	End of Text	35	23	043	&#035;	#	67	43	103	&#067;	C	99	63	143	&#099;	c
4	4	004	End of Transmission	36	24	044	&#036;	$	68	44	104	&#068;	D	100	64	144	&#100;	d
5	5	005	Enquiry	37	25	045	&#037;	%	69	45	105	&#069;	E	101	65	145	&#101;	e
6	6	006	Acknowledgment	38	26	046	&#038;	&	70	46	106	&#070;	F	102	66	146	&#102;	f
7	7	007	Bell	39	27	047	&#039;	'	71	47	107	&#071;	G	103	67	147	&#103;	g
8	8	010	Backspace	40	28	050	&#040;	(	72	48	110	&#072;	H	104	68	150	&#104;	h
9	9	011	Horizontal Tab	41	29	051	&#041;	)	73	49	111	&#073;	I	105	69	151	&#105;	i
10	A	012	Line feed	42	2A	052	&#042;	*	74	4A	112	&#074;	J	106	6A	152	&#106;	j
11	B	013	Vertical Tab	43	2B	053	&#043;	+	75	4B	113	&#075;	K	107	6B	153	&#107;	k
12	C	014	Form feed	44	2C	054	&#044;	,	76	4C	114	&#076;	L	108	6C	154	&#108;	l
13	D	015	Carriage return	45	2D	055	&#045;	-	77	4D	115	&#077;	M	109	6D	155	&#109;	m
14	E	016	Shift Out	46	2E	056	&#046;	.	78	4E	116	&#078;	N	110	6E	156	&#110;	n
15	F	017	Shift In	47	2F	057	&#047;	/	79	4F	117	&#079;	O	111	6F	157	&#111;	o
16	10	020	Data Link Escape	48	30	060	&#048;	0	80	50	120	&#080;	P	112	70	160	&#112;	p
17	11	021	Device Control 1	49	31	061	&#049;	1	81	51	121	&#081;	Q	113	71	161	&#113;	q
18	12	022	Device Control 2	50	32	062	&#050;	2	82	52	122	&#082;	R	114	72	162	&#114;	r
19	13	023	Device Control 3	51	33	063	&#051;	3	83	53	123	&#083;	S	115	73	163	&#115;	s
20	14	024	Device Control 4	52	34	064	&#052;	4	84	54	124	&#084;	T	116	74	164	&#116;	t
21	15	025	Negative Ack.	53	35	065	&#053;	5	85	55	125	&#085;	U	117	75	165	&#117;	u
22	16	026	Synchronous idle	54	36	066	&#054;	6	86	56	126	&#086;	V	118	76	166	&#118;	v
23	17	027	End of Trans. Block	55	37	067	&#055;	7	87	57	127	&#087;	W	119	77	167	&#119;	w
24	18	030	Cancel	56	38	070	&#056;	8	88	58	130	&#088;	X	120	78	170	&#120;	x
25	19	031	End of Medium	57	39	071	&#057;	9	89	59	131	&#089;	Y	121	79	171	&#121;	y
26	1A	032	Substitute	58	3A	072	&#058;	:	90	5A	132	&#090;	Z	122	7A	172	&#122;	z
27	1B	033	Escape	59	3B	073	&#059;	;	91	5B	133	&#091;	[	123	7B	173	&#123;	{
28	1C	034	File Separator	60	3C	074	&#060;	<	92	5C	134	&#092;	\	124	7C	174	&#124;	|
29	1D	035	Group Separator	61	3D	075	&#061;	=	93	5D	135	&#093;	]	125	7D	175	&#125;	}
30	1E	036	Record Separator	62	3E	076	&#062;	>	94	5E	136	&#094;	^	126	7E	176	&#126;	~
31	1F	037	Unit Separator	63	3F	077	&#063;	?	95	5F	137	&#095;	_	127	7F	177	&#127;	Del

asciicharstable.com

VÍ DỤ BẢNG MÃ UNICODE

Hex		00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
	Dec	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
00	0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
10	16	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
20	32		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
30	48	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
40	64	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
50	80	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
60	96	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
70	112	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	
80	128																
90	144																
A0	160		¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	-	®	¯
B0	176	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
C0	192	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
D0	208	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
E0	224	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
F0	240	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ