



Chương 2 : Biểu diễn thông tin trong máy tính

*

Nội dung

- * Hệ thống số
- * Số học máy tính
- * Logic số

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:
 - $00\dots000 = 0$
 - $99\dots999 = 10^n - 1$

Dạng tổng quát của số thập phân

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

Giá trị của A được hiểu như sau:

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

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

Ví dụ số thập phân

$$472.38 = 4 \times 10^2 + 7 \times 10^1 + 2 \times 10^0 + 3 \times 10^{-1} + 8 \times 10^{-2}$$

■ 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:

- $0.38 \times 10 = 3.8$ phần nguyên = 3
 - $0.8 \times 10 = 8.0$ phần nguyên = 8
- ↓



2. Hệ nhị phân

- Cơ số 2
- 2 chữ số nhị phân: 0 và 1
- chữ số nhị phân 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:
 - $00\dots000 = 0$
 - $11\dots111 = 2^n - 1$

Bits, Bytes, Nibbles...

- Bits

10010110

most significant bit least significant bit

- Bytes & Nibbles

byte

10010110

nibble

- Bytes

CEBF9AD7

most significant byte least significant byte



Lũy thừa hai

- $2^{10} = 1 \text{ kilo}$ ≈ 1000 (1024)
- $2^{20} = 1 \text{ mega}$ $\approx 1 \text{ triệu}$ (1,048,576)
- $2^{30} = 1 \text{ giga}$ $\approx 1 \text{ tỷ}$ (1,073,741,824)
- $2^{40} = 1 \text{ tera}$ $\approx 1000 \text{ tỷ}$
- $2^{50} = 1 \text{ peta}$ $\approx 1 \text{ triệu tỷ}$

Dạng tổng quát của số nhị phân

Có một số nhị phân A như sau:

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

Giá trị của A được tính như sau:

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

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

Ví dụ số nhị phân

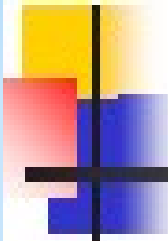
$$1101001.1011_{(2)} =$$

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

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

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

$$= 105.6875_{(10)}$$



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

- 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 $0.6875_{(10)}$

■	$0.6875 \times 2 = 1.375$	phần nguyên = 1	↓
■	$0.375 \times 2 = 0.75$	phần nguyên = 0	
■	$0.75 \times 2 = 1.5$	phần nguyên = 1	
■	$0.5 \times 2 = 1.0$	phần nguyên = 1	

- 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	$\times 2 =$	1.62	phần nguyên	$=$	1
■	0.62	$\times 2 =$	1.24	phần nguyên	$=$	1
■	0.24	$\times 2 =$	0.48	phần nguyên	$=$	0
■	0.48	$\times 2 =$	0.96	phần nguyên	$=$	0
■	0.96	$\times 2 =$	1.92	phần nguyên	$=$	1
■	0.92	$\times 2 =$	1.84	phần nguyên	$=$	1
■	0.84	$\times 2 =$	1.68	phần nguyên	$=$	1

■ $0.81_{(10)} \approx 0.1100111_{(2)}$

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

■ Ví dụ 3: chuyển đổi $0.2_{(10)}$

■	$0.2 \times 2 =$	0.4	phần nguyên	$=$	0
■	$0.4 \times 2 =$	0.8	phần nguyên	$=$	0
■	$0.8 \times 2 =$	1.6	phần nguyên	$=$	1
■	$0.6 \times 2 =$	1.2	phần nguyên	$=$	1
■	$0.2 \times 2 =$	0.4	phần nguyên	$=$	0
■	$0.4 \times 2 =$	0.8	phần nguyên	$=$	0
■	$0.8 \times 2 =$	1.6	phần nguyên	$=$	1
■	$0.6 \times 2 =$	1.2	phần nguyên	$=$	1

■ $0.2_{(10)} \approx 0.00110011_{(2)}$



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

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

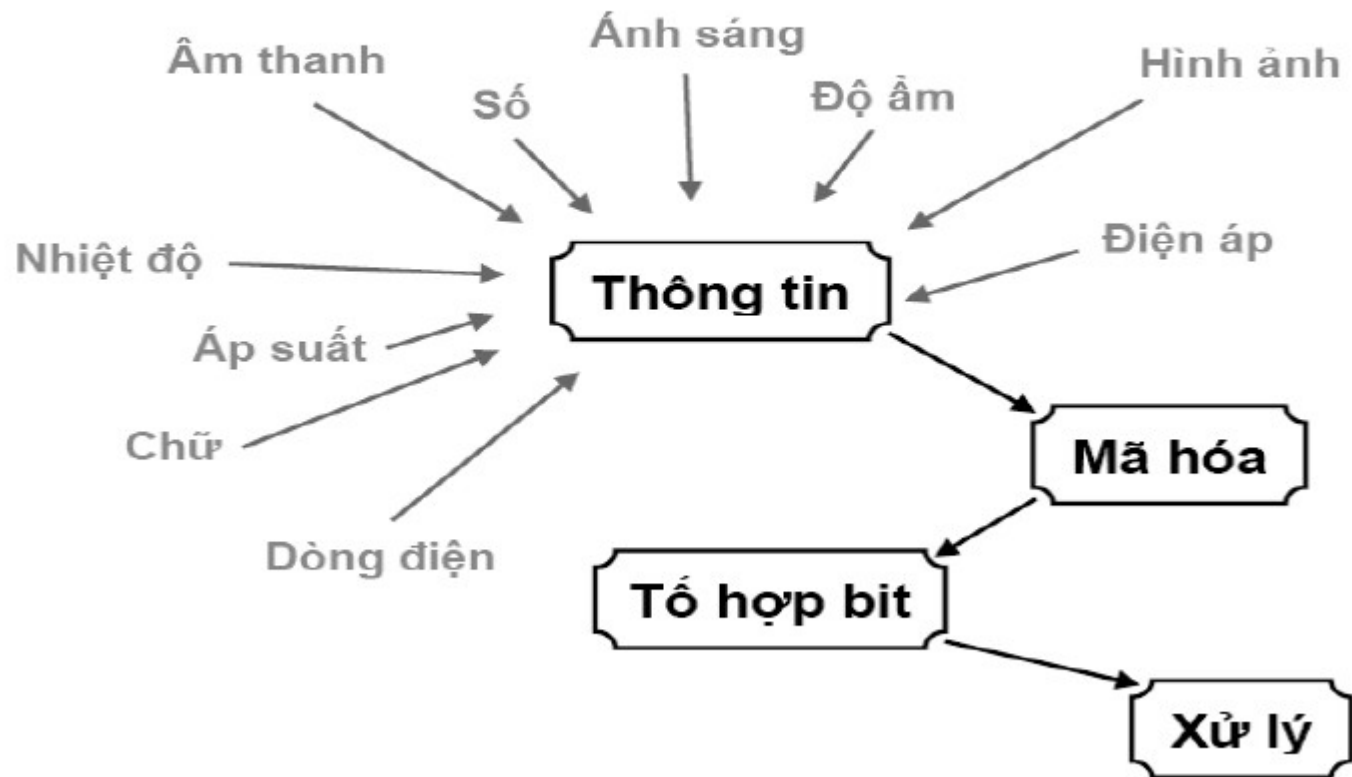
Ví dụ chuyển đổi số nhị phân \rightarrow số Hexa:

- $1011\ 0011_2 = B3_{16}$
- $0000\ 0000_2 = 00_{16}$
- $0010\ 1101\ 1001\ 1010_2 = 2D9A_{16}$
- $1111\ 1111\ 1111\ 1111_2 = FFFF_{16}$

2.1. Mã hoá và lưu trữ dữ liệu trong máy tính

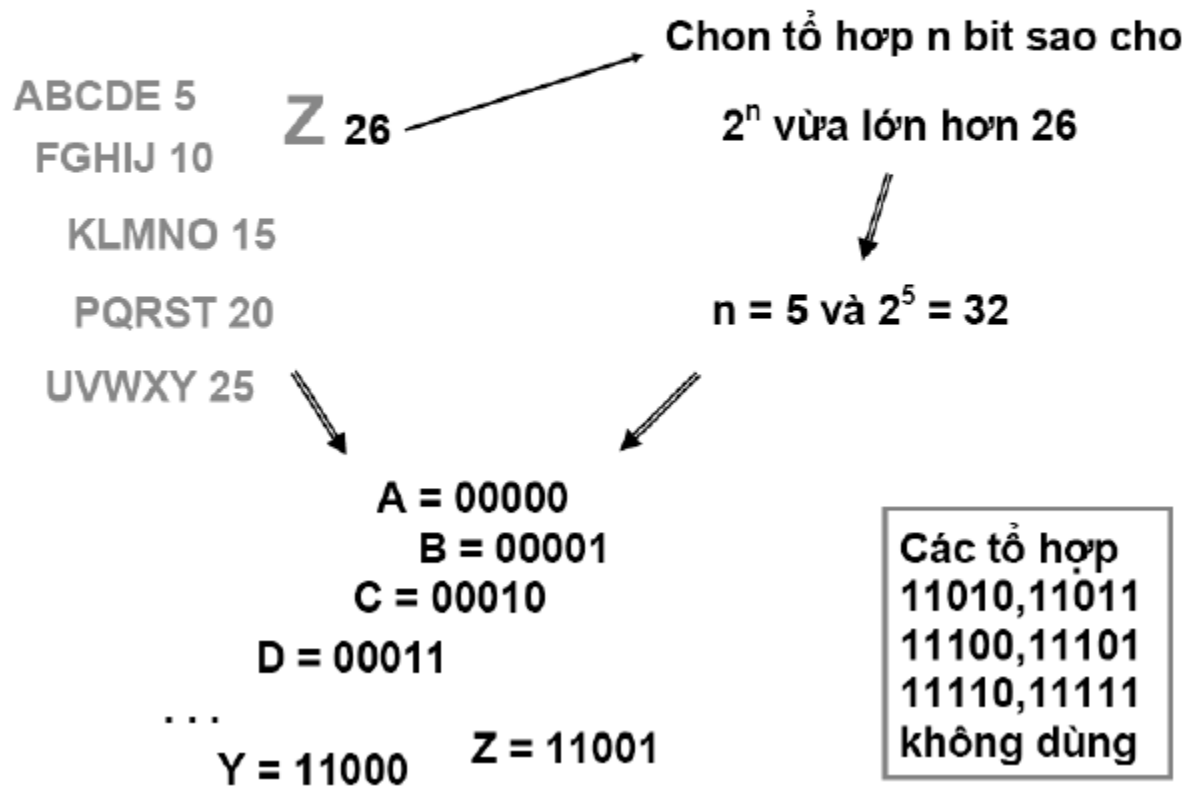
- Nguyên tắc chung về mã hoá dữ liệu
 - Mọi dữ liệu đưa vào máy tính đều được mã hoá thành số nhị phân
 - Các loại dữ liệu
 - Dữ liệu nhân tạo: do con người quy ước
 - Dữ liệu số nguyên: mã hoá theo một số chuẩn quy ước
 - Dữ liệu số thực: mã hoá bằng số dấu chấm động
 - Dữ liệu ký tự: mã hoá theo bộ mã ký tự
 - Dữ liệu tự nhiên: tồn tại khách quan với con người

Mã hóa thông tin đầu vào



Mã hóa thông tin đầu vào

Ví dụ : mã hóa chữ cái



Thứ tự lưu trữ các byte của dữ liệu

- Bộ nhớ chính thường được tổ chức theo byte
- Độ dài từ dữ liệu có thể chiếm từ một đến nhiều byte
 - cần phải biết thứ tự lưu trữ các byte trong bộ nhớ chính với các dữ liệu nhiều byte.

Thứ tự lưu trữ các byte của dữ liệu

- Có 2 cách lưu trữ:
 - Lưu trữ đầu nhỏ (Little-endian): Byte thấp được lưu trữ ở ngăn nhớ có địa chỉ nhỏ hơn, byte cao được lưu trữ ở ngăn nhớ có địa chỉ lớn hơn.
 - Lưu trữ đầu to (Big-endian): Byte cao được lưu trữ ở ngăn nhớ có địa chỉ nhỏ hơn, byte thấp được lưu trữ ở ngăn nhớ có địa chỉ lớn hơn.
- Intel 80x86 và các Pentium : Little-endian
- Motorola 680x0 và các bộ xử lý RISC : Big-endian
- Power PC và Itanium : cả hai

3. Biểu diễn số nguyên

- * Có hai loại số nguyên:
 - Số nguyên không dấu (Unsigned Integer)
 - Số nguyên có dấu (Signed Integer)
- * Biểu diễn số nguyên không dấu
 - Dùng n bit biểu diễn số nguyên không dấu A:

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

- * Giá trị của số nguyên dương A được tính như sau:
 - Dải biểu diễn của A: $0 \rightarrow 2^n-1$
- * Số 8 bit có giá trị : $0 \div 255$
- * Số 16 bit có giá trị : $0 \div 65\,535$
- * Số 32 bit có giá trị : $0 \div 4\,294\,967\,295$

1. Biểu diễn số nguyên không dấu

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

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

Giá trị của A được tính như sau:

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

Dải biểu diễn của A : từ 0 đến $2^n - 1$



Các ví dụ

- Ví dụ 1. Biểu diễn các số nguyên không dấu sau đây bằng 8-bit:

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

Giải:

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

$$41 = 0010\ 1001$$

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

$$150 = 1001\ 0110$$

Các ví dụ (tiếp)

- 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

$$0000\ 0000 = 0$$

$$0000\ 0001 = 1$$

$$0000\ 0010 = 2$$

$$0000\ 0011 = 3$$

...

$$1111\ 1111 = 255$$

Chú ý:

$$1111\ 1111$$

$$+ \underline{0000\ 0001}$$

$$1\ 0000\ 0000$$

Vậy: $255 + 1 = 0$?

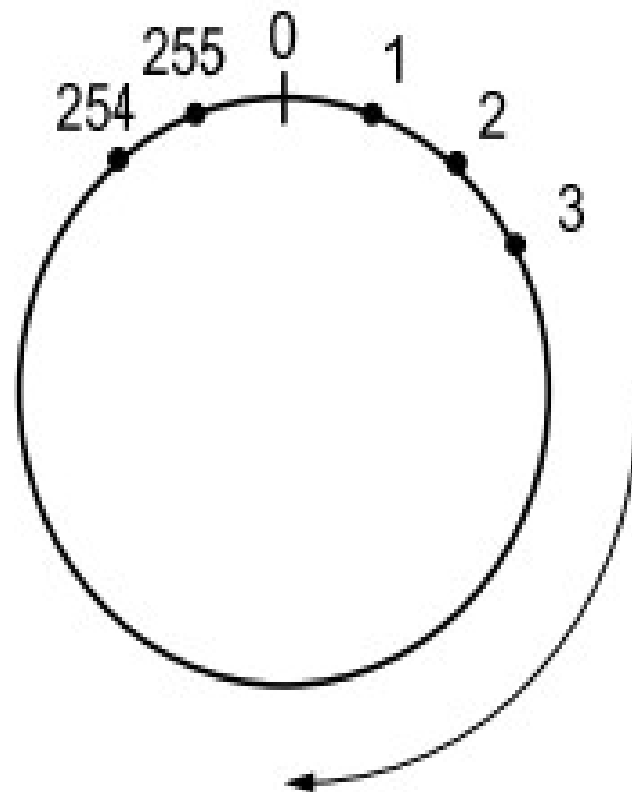
→ do tràn nhớ ra ngoài

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 chín và Số bù mười

- Cho một số thập phân A được biểu diễn bằng n chữ số thập phân, ta có:
 - Số bù chín của $A = (10^n - 1) - A$
 - Số bù mười của $A = 10^n - A$
- Số bù mười của $A = (\text{Số bù chín của } A) + 1$

Số bù chín và Số bù mười (tiếp)

- Ví dụ: với $n=4$, cho $A = 3265$

- Số bù chín của A :

$$\begin{array}{r} 9999 \quad (10^4-1) \\ - \underline{3265} \quad (A) \\ 6734 \end{array}$$

- Số bù mười của A :

$$\begin{array}{r} 10000 \quad (10^4) \\ - \underline{3265} \quad (A) \\ 6735 \end{array}$$



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 $A = (\text{Số bù một của } A) + 1$

Số bù một và Số bù hai (tiếp)

Ví dụ: với $n = 8$ bit, cho $A = 0010\ 0101$

- Số bù một của A được tính như sau:

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

→ đảo các bit của A

- Số bù hai của A được tính như sau:

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

→ 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		=	1101 1010
			<u> 1</u>
■ Số bù hai		=	1101 1011

- Nhận xét:

A	=	0010 0101
Số bù hai	= +	<u>1101 1011</u>
		1 0000 0000 = 0

(bỏ qua bit nhớ ra ngoài)

→ Số bù hai của A = -A

Biểu diễn số nguyên có dấu bằng mã bù hai

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

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

- 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 bằng số bù hai của số dương tương ứng, vì vậy bit $a_{n-1} = 1$

Biểu diễn số dương

- Dạng tổng quát của số dương A:

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

- Giá trị của số dương A:

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

- Dải biểu diễn cho số dương: 0 đến $2^{n-1}-1$

Biểu diễn số âm

- Dạng tổng quát của số âm A:

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

- Giá trị của số âm A:

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

- Dải biểu diễn cho số âm: -1 đến -2^{n-1}

Biểu diễn tổng quát cho số nguyên có dấu

- Dạng tổng quát của số nguyên A:

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

- 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: từ $-(2^{n-1})$ đến $+(2^{n-1}-1)$

Các ví dụ

- Ví dụ 1. Biểu diễn các số nguyên có dấu sau đây bằng 8-bit:

$$A = +58 ; \quad B = -80$$

Giải:

$$A = +58 = 0011\ 1010$$

$$B = -80$$

$$\text{Ta có: } +80 = 0101\ 0000$$

$$\text{Số bù một} = 1010\ 1111$$

$$+ \quad \underline{\quad\quad\quad 1}$$

$$\text{Số bù hai} = 1011\ 0000$$

$$\text{Vậy: } B = -80 = 1011\ 0000$$



Các ví dụ

- Ví dụ 2. Hãy xác định giá trị của các số nguyên có dấu được biểu diễn dưới đây:

- $P = 0110\ 0010$

- $Q = 1101\ 1011$

Giải:

- $P = 0110\ 0010 = 64 + 32 + 2 = +98$

- $Q = 1101\ 1011 = -128 + 64 + 16 + 8 + 2 + 1 = -37$

Với $n = 8$ bit

Biểu diễn được các giá trị từ -128 đến +127

$$0000\ 0000 = 0$$

$$0000\ 0001 = +1$$

$$0000\ 0010 = +2$$

$$0000\ 0011 = +3$$

...

$$0111\ 1111 = +127$$

$$1000\ 0000 = -128$$

$$1000\ 0001 = -127$$

...

$$1111\ 1110 = -2$$

$$1111\ 1111 = -1$$

Chú ý:

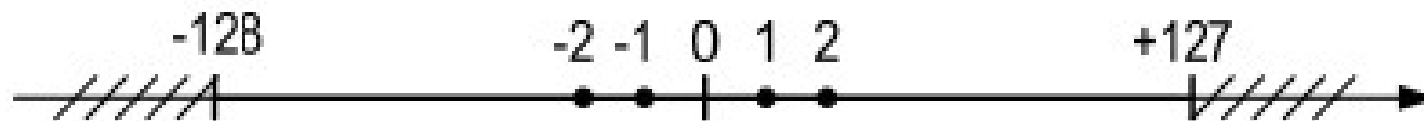
$$+127 + 1 = -128$$

$$-128 - 1 = +127$$

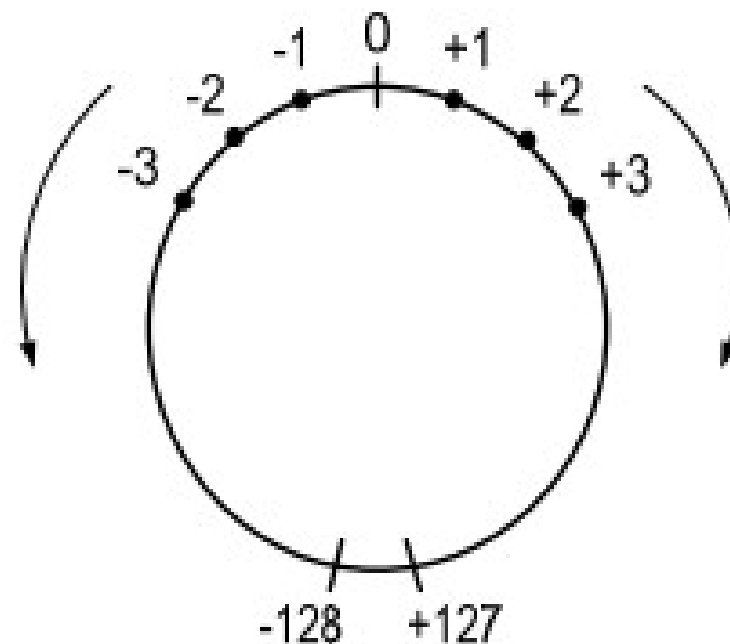
→ do tràn xảy ra

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 $n=16$ bit: biểu diễn từ -32768 đến +32767

- 0000 0000 0000 0000 = 0
- 0000 0000 0000 0001 = +1
- ...
- 0111 1111 1111 1111 = +32767
- 1000 0000 0000 0000 = -32768
- ...
- 1111 1111 1111 1111 = -1

- Với $n=32$ bit: biểu diễn từ -2^{31} đến $2^{31}-1$

- Với $n=64$ bit: biểu diễn từ -2^{63} đến $2^{63}-1$

Chuyển đổi từ 8 bit thành 16 bit

- Đối với số dương:

+19 = 0001 0011 (8bit)

+19 = 0000 0000 0001 0011 (16bit)

→ thêm 8 bit 0 bên trái

- Đối với số âm:

- 19 = 1110 1101 (8bit)

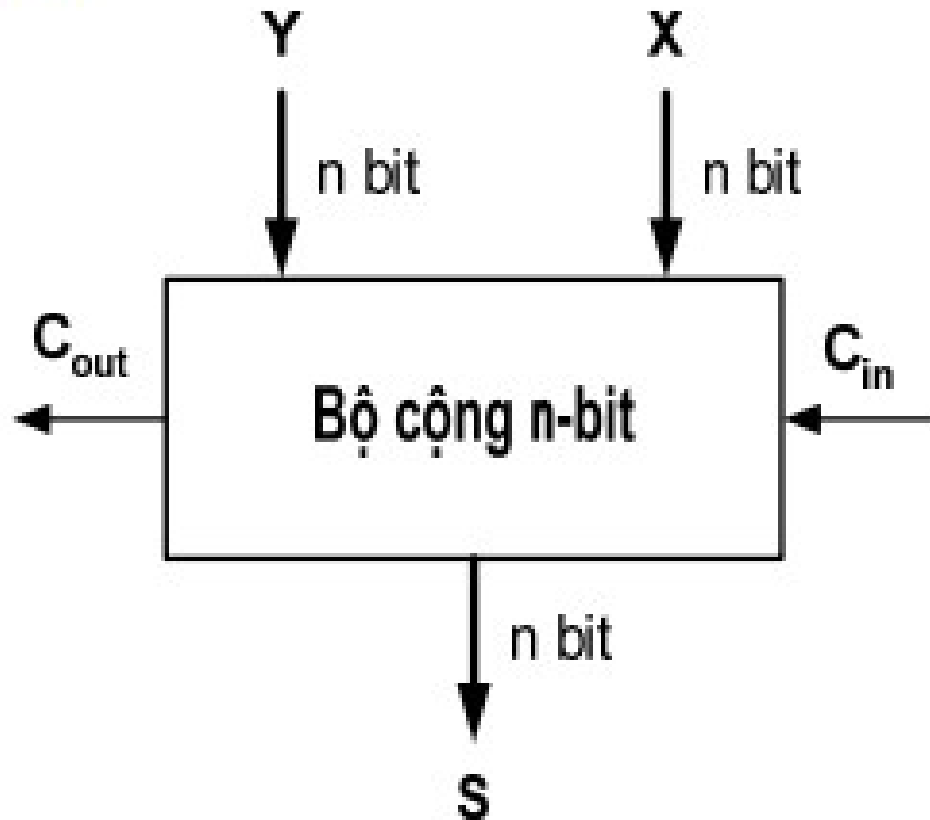
- 19 = 1111 1111 1110 1101 (16bit)

→ thêm 8 bit 1 bên trái

* 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 $C_{out}=0 \rightarrow$ nhận được kết quả đúng.
- Nếu $C_{out}=1 \rightarrow$ nhận được kết quả sai, do tràn nhớ ra ngoài (*Carry Out*).
- Tràn nhớ ra ngoài khi: tổng $> (2^n - 1)$

Ví dụ cộng số nguyên không dấu

$$\begin{array}{rcl} \blacksquare & 57 & = \quad 0011 \ 1001 \\ & + \ 34 & = + \ \underline{0010 \ 0010} \\ & 91 & \quad 0101 \ 1011 = 64+16+8+2+1=91 \rightarrow \text{đúng} \end{array}$$

$$\begin{array}{rcl} \blacksquare & 209 & = \quad 1101 \ 0001 \\ & + \ 73 & = + \ \underline{0100 \ 1001} \\ & 282 & \quad \textcolor{red}{1} \ 0001 \ 1010 \\ & & \quad 0001 \ 1010 = 16+8+2=26 \rightarrow \text{sai} \\ & & \rightarrow \text{có tràn nhớ ra ngoài (C}_{\text{out}}\text{=1)} \end{array}$$

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

$$\begin{array}{rcl} 209 & = & 0000 \ 0000 \ 1101 \ 0001 \\ + \ 73 & = + & \underline{0000 \ 0000 \ 0100 \ 1001} \\ & & 0000 \ 0001 \ 0001 \ 1010 = 256+16+8+2 = 282 \end{array}$$

2. Phép đảo dấu

- Ta có:

$$\begin{array}{rcll} + 37 & = & 0010\ 0101 & \\ \text{bù một} & = & 1101\ 1010 & \\ & & + \quad \quad \quad 1 & \\ \text{bù hai} & = & 1101\ 1011 & = -37 \end{array}$$

- Lấy bù hai của số âm:

$$\begin{array}{rcll} - 37 & = & 1101\ 1011 & \\ \text{bù một} & = & 0010\ 0100 & \\ & & + \quad \quad \quad 1 & \\ \text{bù hai} & = & 0010\ 0101 & = +37 \end{array}$$

- 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} .

- Cộng hai số khác dấu: kết quả luôn luôn đúng.
- Cộng hai số cùng dấu:
 - nếu dấu kết quả cùng dấu với các số hạng thì kết quả là đúng.
 - nếu kết quả có dấu ngược lại, khi đó có tràn xảy ra (Overflow) và kết quả bị sai.
- 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

$$\begin{array}{rcl} \blacksquare & (+70) & = 0100\ 0110 \\ & + \underline{(+42)} & = \underline{0010\ 1010} \\ & +112 & 0111\ 0000 = +112 \end{array}$$

$$\begin{array}{rcl} \blacksquare & (+97) & = 0110\ 0001 \\ & + \underline{(-52)} & = \underline{1100\ 1100} \quad (+52=0011\ 0100) \\ & +45 & 1\ 0010\ 1101 = +45 \end{array}$$

$$\begin{array}{rcl} \blacksquare & (-90) & = 1010\ 0110 \quad (+90=0101\ 1010) \\ & + \underline{(+36)} & = \underline{0010\ 0100} \\ & -54 & 1100\ 1010 = -54 \end{array}$$

$$\begin{array}{rcl} \blacksquare & (-74) & = 1011\ 0110 \quad (+74=0100\ 1010) \\ & + \underline{(-30)} & = \underline{1110\ 0010} \quad (+30=0001\ 1110) \\ & -104 & 1\ 1001\ 1000 = -104 \end{array}$$

Ví dụ cộng số nguyên có dấu bị tràn

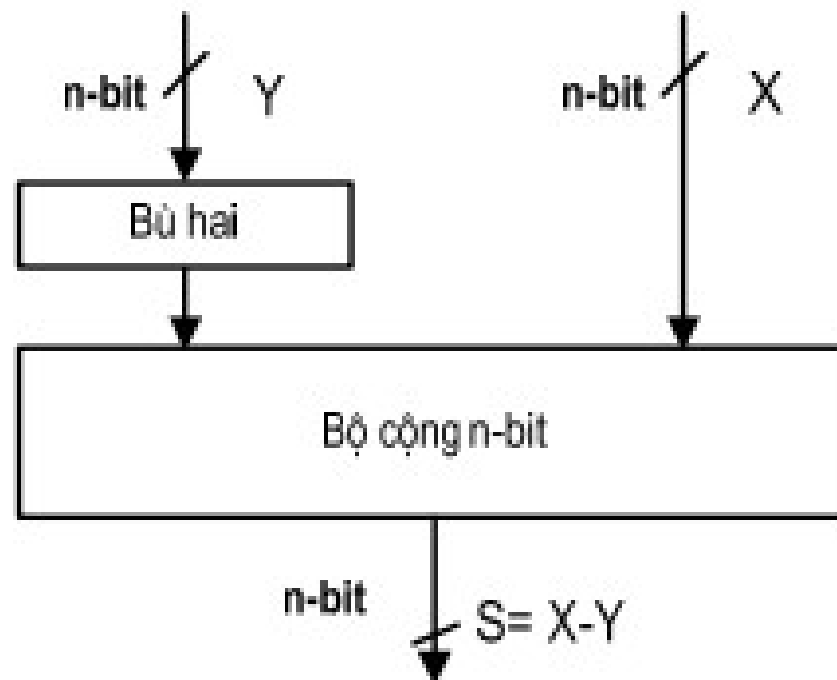
- $(+75) = 0100\ 1011$
 $+ (+82) = \underline{0101\ 0010}$
 $+157 \qquad 1001\ 1101$
 $= -128 + 16 + 8 + 4 + 1 = -99 \rightarrow \text{sai}$

- $(-104) = 1001\ 1000 \quad (+104 = 0110\ 1000)$
 $+ (-43) = \underline{1101\ 0101} \quad (+43 = 0010\ 1011)$
 $-147 \quad 1\ 0110\ 1101$
 $= 64 + 32 + 8 + 4 + 1 = +109 \rightarrow \text{sai}$

- 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



Phép nhân và phép chia số nguyên

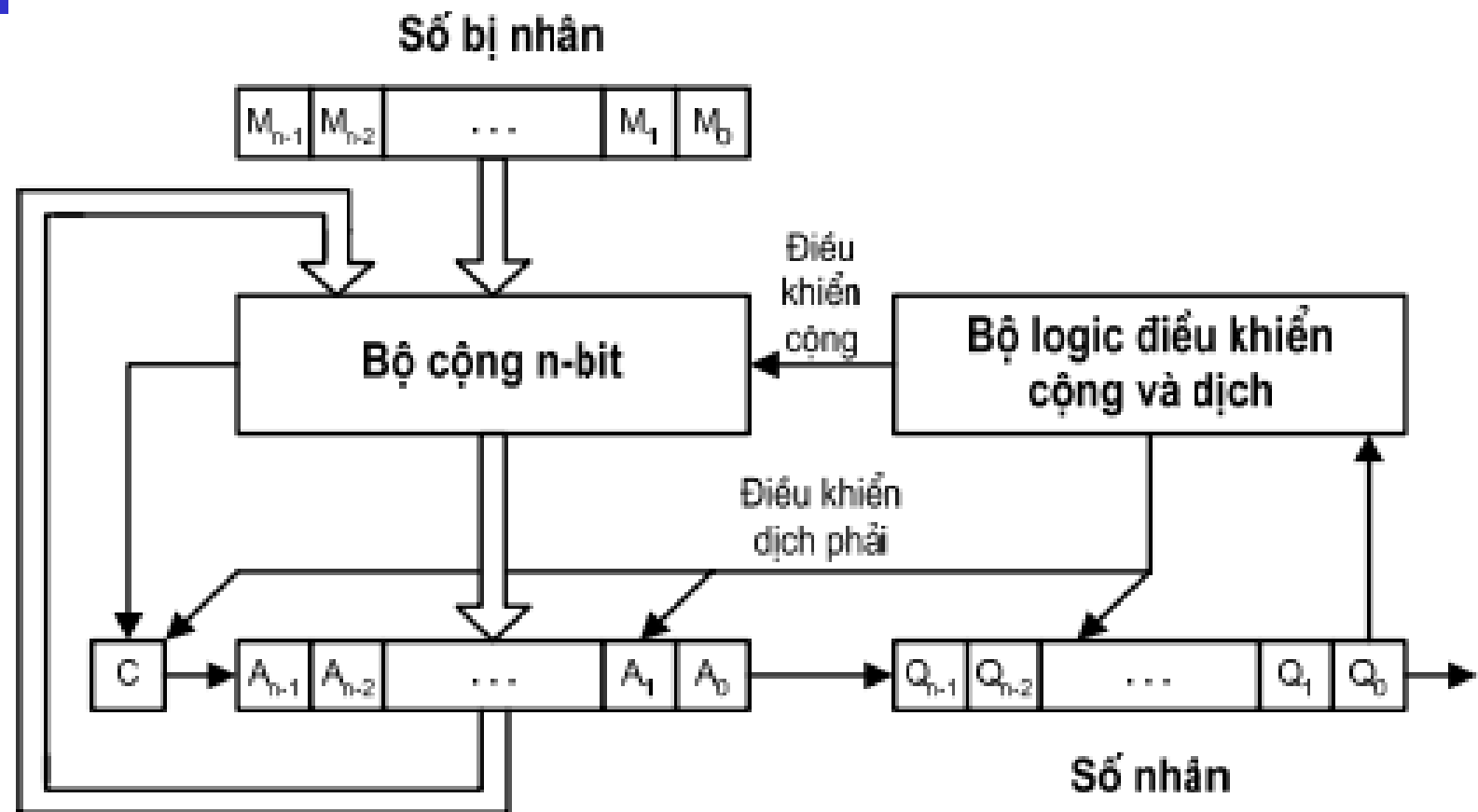
1. Nhân số nguyên không dấu

1011	Số bị nhân (11)
x 1101	Số nhân (13)
1011	} Các tích riêng phần
0000	
1011	
1011	
<hr/> 10001111	Tích (143)

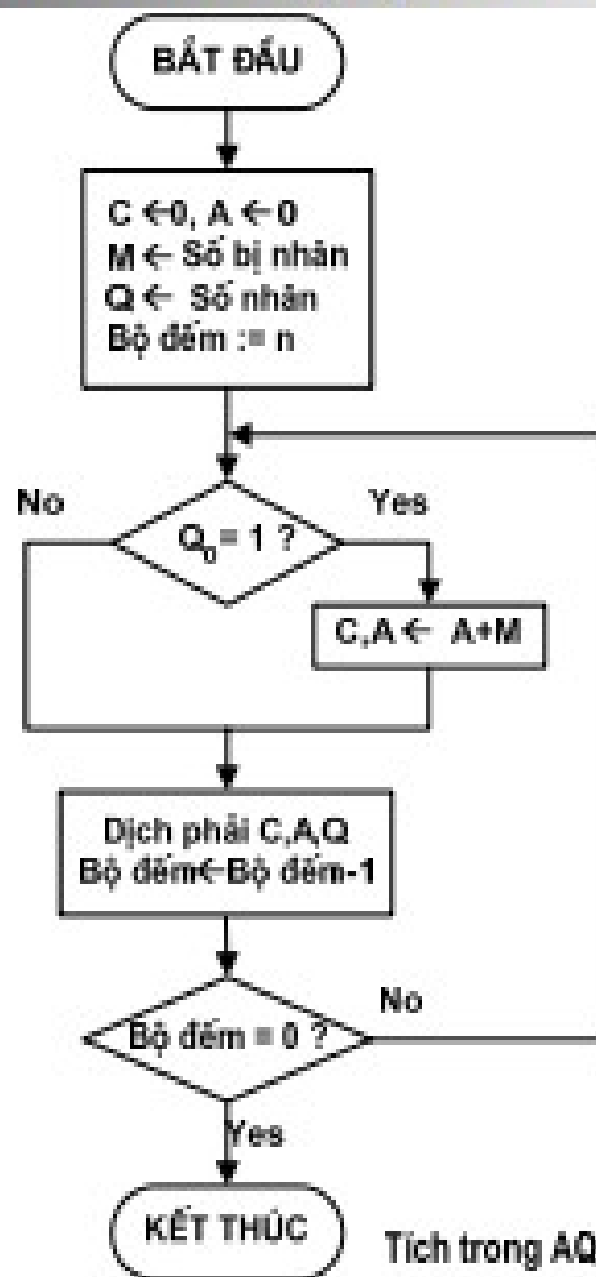
Nhân số nguyên không dấu (tiếp)

- Các tích riêng phần được xác định như sau:
 - Nếu bit của số nhân bằng 0 \rightarrow tích riêng phần bằng 0.
 - Nếu bit của số nhân bằng 1 \rightarrow 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



Tích trong AQ

Ví dụ nhân số nguyên không dấu

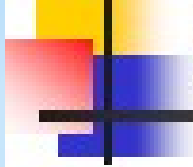
- Số bị nhân M = 1011 (11)
- Số nhân Q = 1101 (13)
- Tích = 1000 1111 (143)

- | | C | A | Q | |
|---|---|--------|--------------|----------------------|
| ■ | 0 | 0000 | 110 1 | Các giá trị khởi đầu |
| | | + 1011 | | |
| | 0 | 1011 | 1101 | A ← A + M |
| ■ | 0 | 0101 | 111 0 | Dịch phải |
| ■ | 0 | 0010 | 111 1 | Dịch phải |
| | | + 1011 | | |
| | 0 | 1101 | 1111 | A ← A + M |
| ■ | 0 | 0110 | 111 1 | Dịch phải |
| | | + 1011 | | |
| | 1 | 0001 | 1111 | A ← A + M |
| ■ | 0 | 1000 | 1111 | Dịch phải |

Ví dụ nhân số nguyên không dấu

- Số bị nhân M = 0010 (2)
- Số nhân Q = 0011 (3)
- Tích = 0000 0110 (6)

- | | C | A | Q | |
|---|---|-------------|------|----------------------|
| ■ | 0 | 0000 | 0011 | Các giá trị khởi đầu |
| | + | <u>0010</u> | | |
| | 0 | 0010 | 0011 | A ← A + M |
| ■ | 0 | 0001 | 0001 | Dịch phải |
| | + | <u>0010</u> | | |
| | 0 | 0011 | 0001 | |
| ■ | 0 | 0001 | 1000 | Dịch phải |
| ■ | 0 | 0000 | 1100 | Dịch phải |
| ■ | 0 | 0000 | 0110 | Dịch phải |



2. 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).

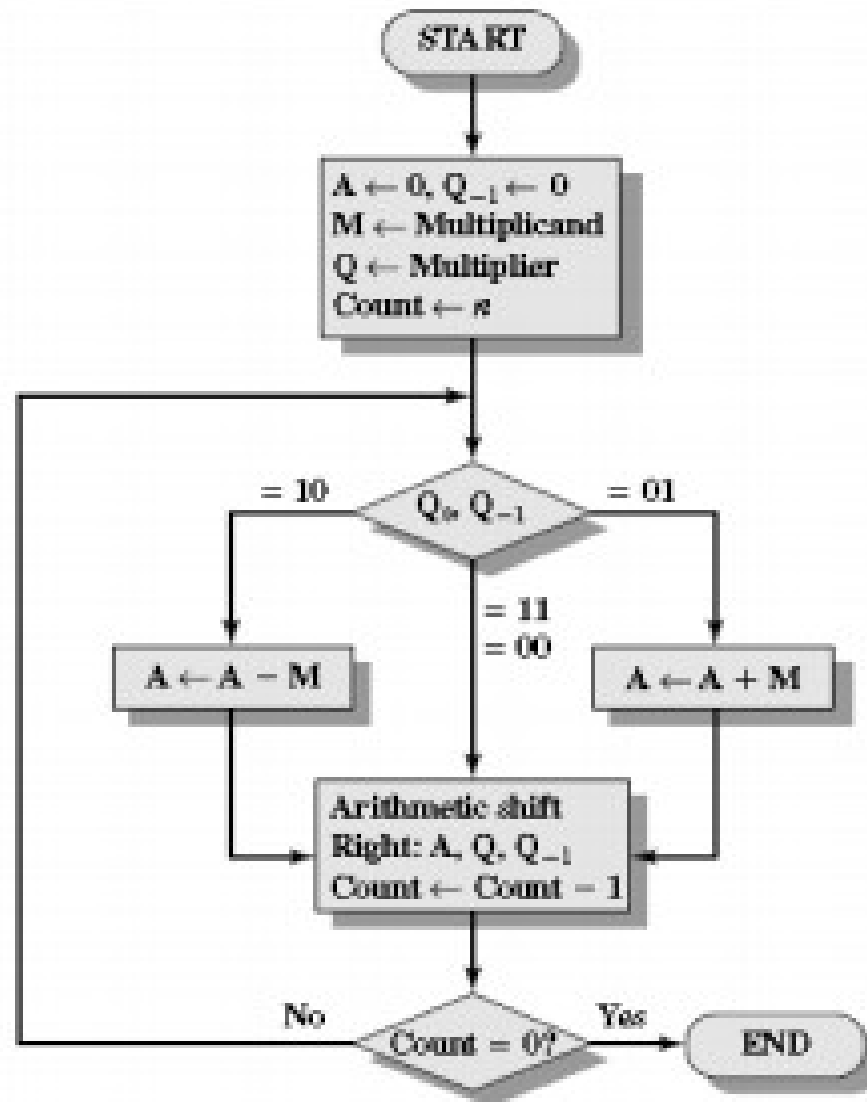
Ví dụ nhân 10×13 Số bị nhân $10_{10} = 1010_2$ và Số nhân $13_{10} = 1101_2$

Bước	Bộ đếm	Số nhân	Nhớ	Tích thành phần	Vòng lặp	Ghi chú
(a) và (b)	4	1101	0	0000 0000		Khởi trị, LSB=1
(c)	4	1101	0	1010 0000	1	Cộng
(d)	4	0110	0	0101 0000	1	Dịch phải
(e)	3	0110	0	0101 0000	1	Giảm n
(c)	3	0110	0	0101 0000	2	LSB=0
(d)	3	0011	0	0010 1000	2	Dịch phải
(e)	2	0011	0	0010 1000	2	Giảm n, LSB=1
(c)	2	0011	0	1100 1000	3	Cộng
(d)	2	0001	0	0110 0100	3	Dịch phải
(e)	1	0001	0	0110 0100	3	Giảm n, LSB=1
(c)	1	0001	1	0000 0100	4	Cộng
(d)	1	0000	0	1000 0010	4	Dịch phải
(e)	0	0000	0	1000 0010	4	Giảm n. Dừng

* **Phép nhân các số có dấu**

- * Phương pháp kinh điển nhân các số có dấu dạng bù hai là giải thuật Booth, có thể áp dụng cho 2 số dương, một số âm và một dương hoặc cả 2 số đều âm

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

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	
0111	1001	0011	0	$A = A - M$	
0111	1 100	1001	1	ASR	End cycle #1

0111	1 110	0100	1	ASR	End cycle #2

0111	0101	0100	1	$A = A + M$	
0111	0010	1010	0	ASR	End cycle #3

0111	0001	0101	1	ASR	End cycle #4
					
+21 (correct result)					


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	1 100	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	1 101	0111	1	ASR	End cycle #3

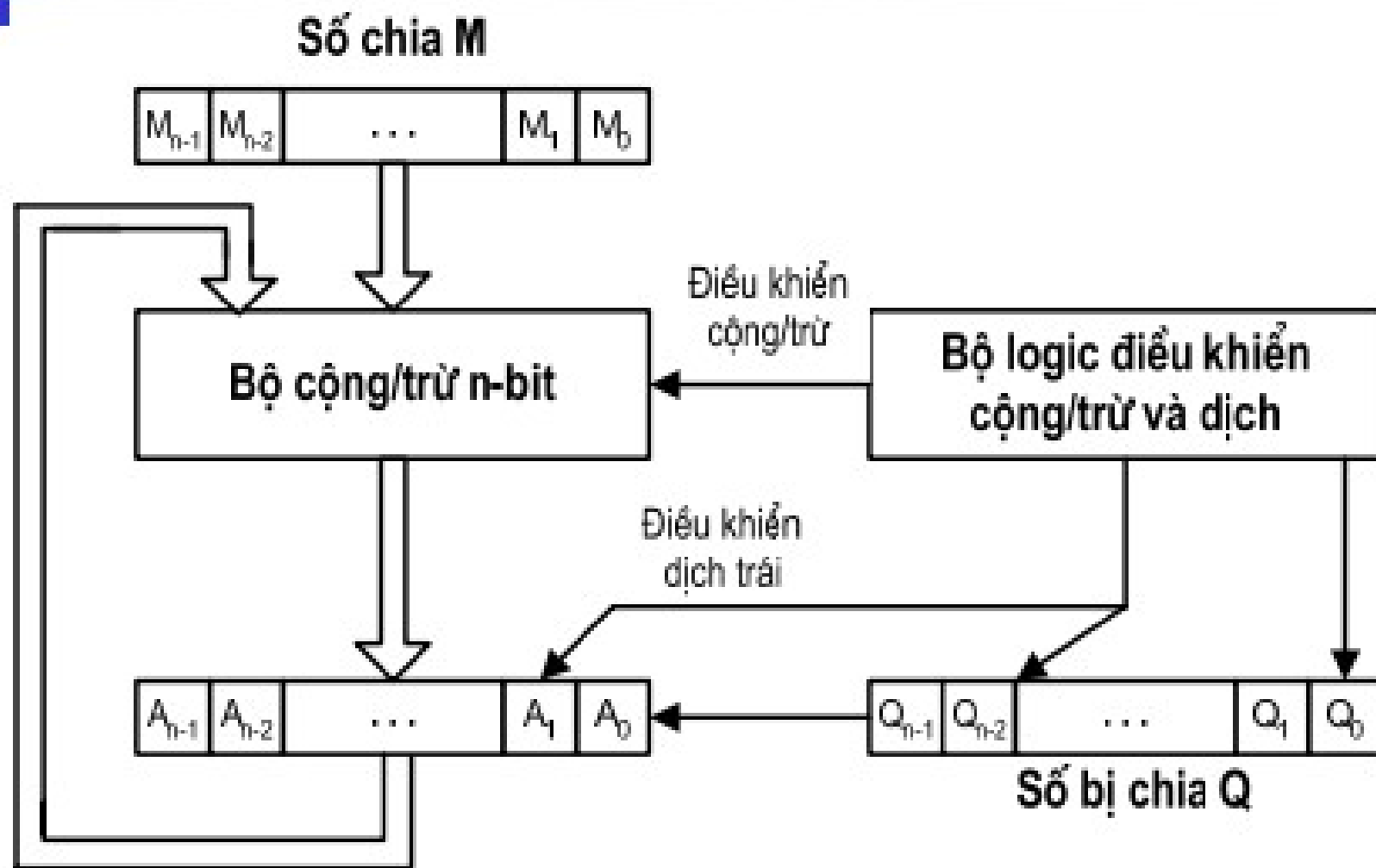
0111	1110	1011	1	ASR	End cycle #4


 -21 (correct result)

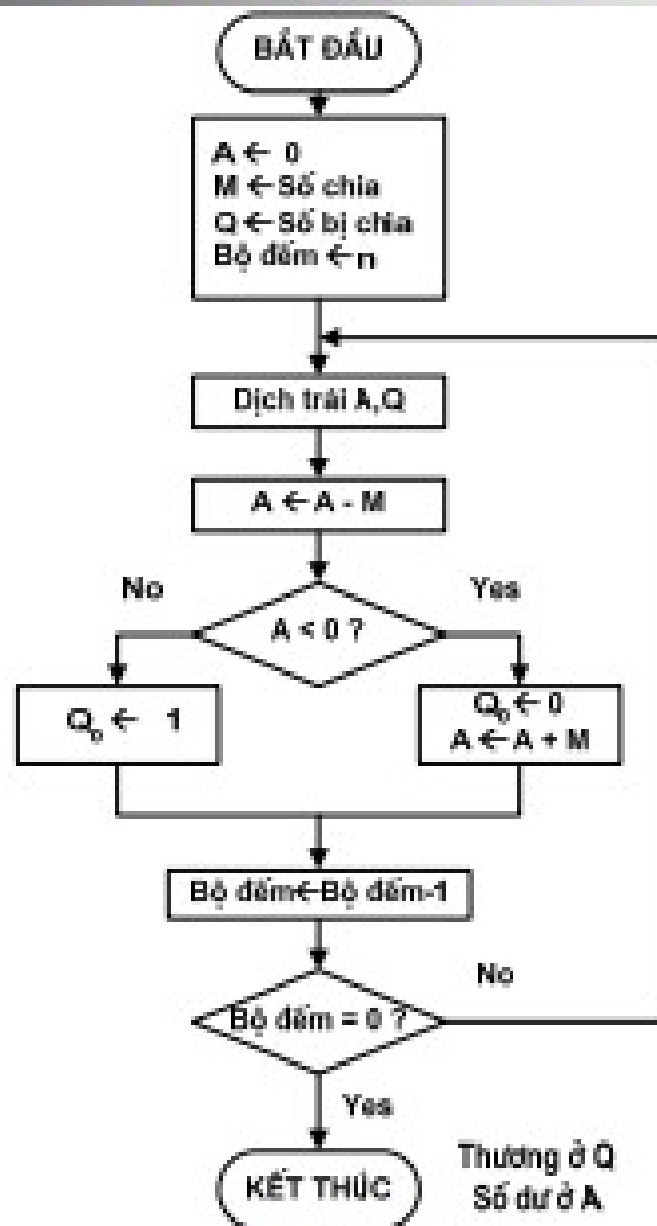
3. Chia số nguyên không dấu

Số bị chia	$\begin{array}{r} 10010011 \\ \underline{1011} \\ 001110 \\ \underline{1011} \\ 001111 \\ \underline{1011} \\ 100 \end{array}$	$\begin{array}{r} 1011 \\ \hline 00001101 \end{array}$	Số chia Thương
			Phần dư

Bộ chia số nguyên không dấu



Lưu đồ chia số nguyên không dấu



* Ví dụ phép chia $7/3$ ($0111_2/0011_2$)

A	Q	M = 0011
0000	0111	Khởi trị. $n=4$
0000	1110	Dịch trái A, Q (vòng 1)
1101		Trừ ($A \leftarrow A-M$), không thành công do ($A < 0$)
0000	1110	$A=0$, $Q_0 \leftarrow 0$, khôi phục A ($A \leftarrow A+M$), $n=3$
0001	1100	Dịch trái (vòng 2)
1110		Trừ ($A \leftarrow A-M$); không thành công ($A < 0$)
0001	1100	Khôi phục A, $n=2$
0011	1000	Dịch trái (vòng 3)
0000		Trừ, thành công ($A \geq 0$)
0000	1001	Đặt $Q_0 \leftarrow 1$, $n=1$
0001	0010	Dịch trái (vòng 4)
1110		Trừ
0001	0010	Khôi phục A, $n=0$. Dừng.

Vậy kết quả thương $Q = 0010_2 = 2_{10}$ và dư $A = 0001_2 = 1_{10}$

4. 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

Thuật toán chia 2 số nguyên tổng quát

1. Nạp số chia vào thanh ghi M và số bị chia vào các thanh ghi A, Q. Số bị chia phải ở dạng $2n$ bit, vì vậy, ví dụ số 4 bit 0111 trở thành 00000111 và 1001 trở thành 11111001. $\text{Count}=n$
2. Dịch trái A, Q đi 1 bit
3. Nếu M và A có cùng dấu, thực hiện gán $A \leftarrow A-M$, ngược lại gán $A \leftarrow A+M$
4. Thao tác trên thành công nếu dấu của A là như nhau trước và sau thao tác
 - a. Nếu thao tác thành công hoặc ($A=0$ hoặc $Q=0$) thì đặt $Q_0 \leftarrow 1$
 - b. Nếu thao tác không thành công và ($A \neq 0$ hoặc $Q \neq 0$) thì đặt $Q_0 \leftarrow 0$ và khôi phục giá trị trước đó của A. $\text{Count} \leftarrow \text{Count}-1$
5. Lặp lại bước 2 đến bước 4 số lần lặp bằng số bit của Q ($\text{Count}=0$)
6. Phần dư là A. Nếu dấu của số chia và số bị chia giống nhau thì thương là Q, ngược lại thương là số bù 2 của Q.

* Bài tập

Bài 1:

Số nhị phân 8 bit (11001100), số này tương ứng với số nguyên thập phân có dấu là bao nhiêu nếu số đang được biểu diễn trong cách biểu diễn:

b. Số bù 1.

c. Số bù 2.

Bài 2: Đổi các số sau đây:

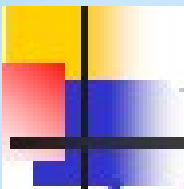
a. $(011011)_2$ ra số thập phân.

b. $(-2005)_{10}$ ra số nhị phân 16 bits.

c. $(55.875)_{10}$ ra số nhị phân.

Bài 3: Thực hiện các phép tính

$$7+8 \quad 8*9 \quad 7/4 \quad -7/3 \quad --6/(-5)$$



4. Số 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:

$$X = M * R^E$$

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



2. Chuẩn IEEE754/85

- Cơ số $R = 2$
- Các dạng:
 - Dạng 32-bit
 - Dạng 44-bit
 - Dạng 64-bit
 - Dạng 80-bit

Các dạng biểu diễn chính



Dạng 32 bit



- **S** là bit dấu:
 - $S = 0 \rightarrow$ số dương
 - $S = 1 \rightarrow$ số âm
- **e** (8 bit) là mã *excess-127* của phần mũ E:
 - $e = E + 127 \rightarrow E = e - 127$
 - giá trị 127 gọi là độ lệch (bias)
- **m** (23 bit) là phần lẻ của phần định trị M:
 - $M = 1.m$
- Công thức xác định giá trị của số thực:

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

Ví dụ 1

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

■ 1100 0001 0101 0110 0000 0000 0000 0000

■ $S = 1 \rightarrow$ số âm

■ $e = 1000\ 0010_2 = 130 \rightarrow E = 130 - 127 = 3$

Vậy

$$X = -1.10101100 \cdot 2^3 = -1101.011 = -13.375$$

■ 0011 1111 1000 0000 0000 0000 0000 0000 = ?
= +1.0

Ví dụ 2

Biểu diễn số thực $X = 83.75$ 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:
 $X = 0100\ 0010\ 1010\ 0111\ 1000\ 0000\ 0000\ 0000$

Ví dụ 3

Biểu diễn số thực $X = -0,2$ về dạng số dấu phẩy động IEEE754 32-bit

Giải:

- $X = -0,2_{(10)} = -0.00110011...0011..._{(2)} =$
 $= -1.100110011...0011... \times 2^{-3}$
- Ta có:
 - $S = 1$ vì đây là số âm
 - $E = e - 127 = -3 \rightarrow e = 127 - 3 = 124_{(10)} = 0111\ 1100_{(2)}$
- Vậy:
 $X = 1011\ 1110\ 0100\ 1100\ 1100\ 1100\ 1100\ 1100$



Bài tập

Biểu diễn các số thực sau đây về dạng số dấu phẩy động IEEE754 32-bit:

$$X = -27.0625; \quad Y = 1/32$$

* Phương pháp đổi số thực sang số dấu phẩy động 32 bit:

- Đổi số thập phân thành số nhị phân.
- Biểu diễn số nhị phân dưới dạng $\pm 1.xxxBy$ (B: cơ số 2).
- Bit cao nhất 31: lấy giá trị 0 với số dương, 1 với số âm.
- Phần mũ y đổi sang mã excess -127 của y , được xác định $= y + 127$
- Phần xxx là phần định trị, được đưa vào từ bit 22..0.

Ví dụ: Biểu diễn số thực $(9,75)_{10}$ dưới dạng dấu phẩy động.

Ta đổi sang dạng nhị phân: $(9,75)_{10} = (1001.11)_2 = 1.00111B3$.

Bit dấu: bit 31 = 0.

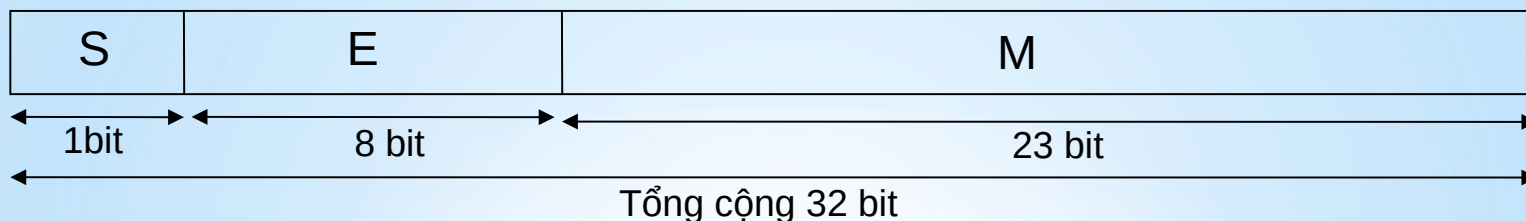
Mã excess - 127 của 3 là: $127 + 3 = 130_{10} = (10000010)_2$. Được đưa vào các bit tiếp theo: từ bit 30 đến bit 23.

Cuối cùng số thực $(9,75)_{10}$ được biểu diễn dưới dạng dấu phẩy động 32 bit như sau:

0 1000 0010 001 1100 0000 0000 0000 0000
bit 31|30 23|22 0|

*

Ví dụ 2:
Một số chính xác đơn dài 32 bit, độ dịch là 127 và phần phân số của định trị là 23 bit có định dạng như sau:



* Số -2345,125 hệ 10 ở trong máy tính sẽ là số 16 bit
 $-2345,125 = -1001\ 0010\ 1001.001$ (số nhị phân bình thường)
 $= -1.001\ 0010\ 1001001 \times 2^{11}$ (dạng chuẩn hoá)

Phần định trị âm, do vậy bit dấu S=1

Số mũ có độ lệch được cho là
 $+11+127=138=10001010$

Phần phân số của định trị là .001 0010
 0010010000000000 (số 23 bit)

Do đó, dạng IEEE của số này là:

110001010 001 0010 1001001000000000. Nó được
 lưu trữ 2 từ 16 bit là:

1100010100010010 và 1001001000000000

Các quy 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 $\rightarrow X = \pm 0$
- Các bit của e bằng 1, các bit của m bằng 0, thì $X = \pm \infty$
 - x111 1111 1000 0000 0000 0000 0000 0000 $\square X = \pm \infty$
- Các bit của e bằng 1, còn m có ít nhất 1 bit bằng 1, thì nó không biểu diễn cho số nào cả (NaN – not a number)

2.2. Đại số Boole

- Đại số Boole sử dụng các biến logic và phép toán logic
- Biến logic có thể nhận giá trị 1 (TRUE) hoặc 0 (FALSE)
- Phép toán logic cơ bản là AND, OR và NOT với ký hiệu như sau:
 - $A \text{ AND } B : A \cdot B$
 - $A \text{ OR } B : A + B$
 - $\text{NOT } A : \bar{A}$
- Thứ tự ưu tiên: NOT > AND > OR

Các phép toán logic (tiếp)

- Các phép toán 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

P	Q	\overline{P}	P AND Q $P \cdot Q$	P OR Q $P + Q$	P NAND Q $\overline{P \cdot Q}$	P NOR Q $\overline{P + Q}$	P XOR Q $P \oplus Q$
0	0	1	0	0	1	1	0
0	1	1	0	1	1	0	1
1	0	0	0	1	1	0	1
1	1	0	1	1	0	0	0

Các đồng nhất thức của đại số Boole

$$A \cdot B = B \cdot A$$

$$A \cdot (B + C) = (A \cdot B) + (A \cdot C)$$

$$1 \cdot A = A$$

$$A \cdot \bar{A} = 0$$

$$A + B = B + A$$

$$A + (B \cdot C) = (A + B) \cdot (A + C)$$

$$0 + A = A$$

$$A + \bar{A} = 1$$

$$0 \cdot A = 0$$

$$A \cdot A = A$$

$$A \cdot (B \cdot C) = (A \cdot B) \cdot C$$

$$\overline{A \cdot B} = \bar{A} + \bar{B} \text{ (Định lý De Morgan)}$$

$$1 + A = 1$$

$$A + A = A$$

$$A + (B + C) = (A + B) + C$$

$$\overline{A + B} = \bar{A} \cdot \bar{B} \text{ (Định lý De Morgan)}$$

Các phép toán logic (tiếp)

- Các phép toán 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

P	Q	\overline{P}	P AND Q $P \cdot Q$	P OR Q $P + Q$	P NAND Q $\overline{P \cdot Q}$	P NOR Q $\overline{P + Q}$	P XOR Q $P \oplus Q$
0	0	1	0	0	1	1	0
0	1	1	0	1	1	0	1
1	0	0	0	1	1	0	1
1	1	0	1	1	0	0	0

Các đồng nhất thức của đại số Boole

$$A \cdot B = B \cdot A$$

$$A \cdot (B + C) = (A \cdot B) + (A \cdot C)$$

$$1 \cdot A = A$$

$$A \cdot \bar{A} = 0$$

$$A + B = B + A$$

$$A + (B \cdot C) = (A + B) \cdot (A + C)$$

$$0 + A = A$$

$$A + \bar{A} = 1$$

$$0 \cdot A = 0$$

$$A \cdot A = A$$

$$A \cdot (B \cdot C) = (A \cdot B) \cdot C$$

$$\overline{A \cdot B} = \bar{A} + \bar{B} \text{ (Định lý De Morgan)}$$

$$1 + A = 1$$

$$A + A = A$$

$$A + (B + C) = (A + B) + C$$



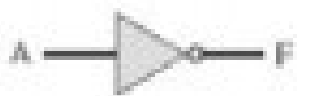



$$\overline{A + B} = \bar{A} \cdot \bar{B} \text{ (Định lý De Morgan)}$$

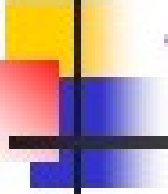


2.3. Các cổng logic (Logic Gates)

- Thực hiện các hàm logic:
 - NOT, AND, OR, NAND, NOR, etc.
- Cổng logic một đầu vào:
 - Cổng NOT
- Cổng hai đầu vào:
 - AND, OR, XOR, NAND, NOR, XNOR
- Cổng nhiều đầu vào

Các cổng logic

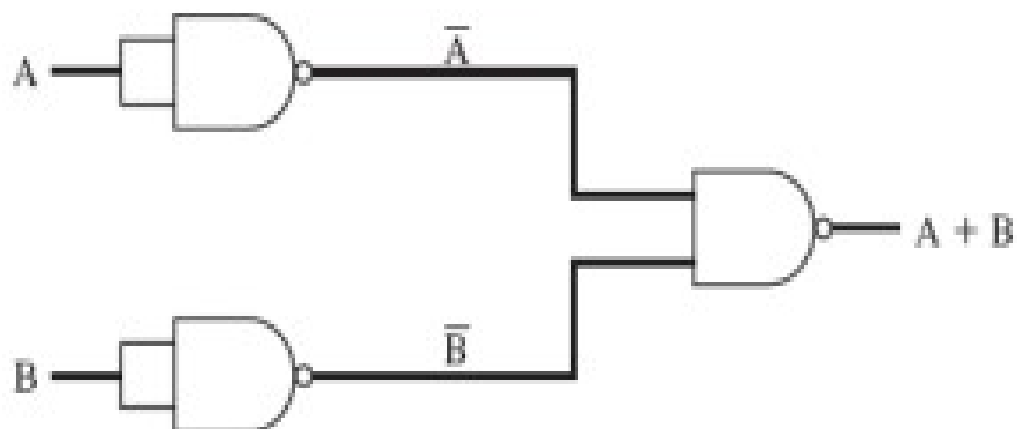
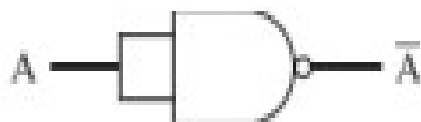
Name	Graphical Symbol	Algebraic Function	Truth Table															
AND		$F = A \cdot B$ or $F = AB$	<table><tr><th>A</th><th>B</th><th>F</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	F	0	0	0	0	1	0	1	0	0	1	1	1
A	B	F																
0	0	0																
0	1	0																
1	0	0																
1	1	1																
OR		$F = A + B$	<table><tr><th>A</th><th>B</th><th>F</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	F	0	0	0	0	1	1	1	0	1	1	1	1
A	B	F																
0	0	0																
0	1	1																
1	0	1																
1	1	1																
NOT		$F = \overline{A}$ or $F = A'$	<table><tr><th>A</th><th>F</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	A	F	0	1	1	0									
A	F																	
0	1																	
1	0																	
NAND		$F = \overline{AB}$	<table><tr><th>A</th><th>B</th><th>F</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	F	0	0	1	0	1	1	1	0	1	1	1	0
A	B	F																
0	0	1																
0	1	1																
1	0	1																
1	1	0																
NOR		$F = \overline{A + B}$	<table><tr><th>A</th><th>B</th><th>F</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	F	0	0	1	0	1	0	1	0	0	1	1	0
A	B	F																
0	0	1																
0	1	0																
1	0	0																
1	1	0																
XOR		$F = A \oplus B$	<table><tr><th>A</th><th>B</th><th>F</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	F	0	0	0	0	1	1	1	0	1	1	1	0
A	B	F																
0	0	0																
0	1	1																
1	0	1																
1	1	0																



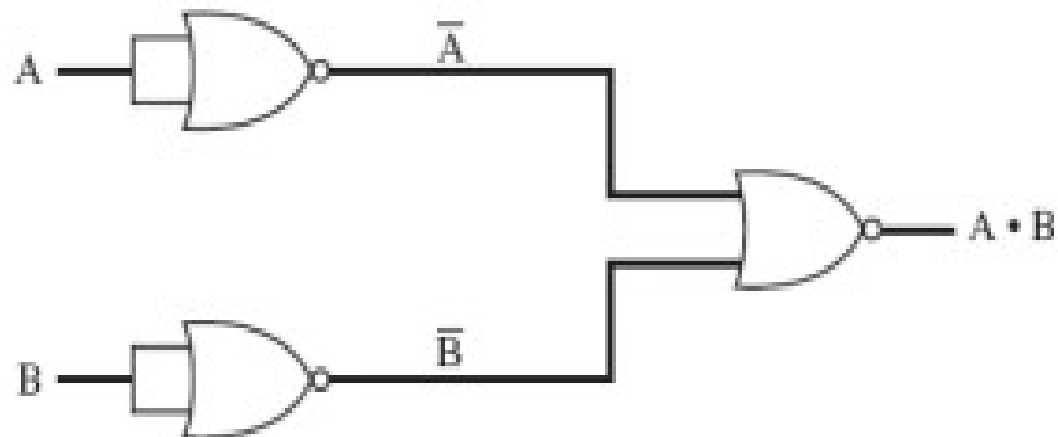
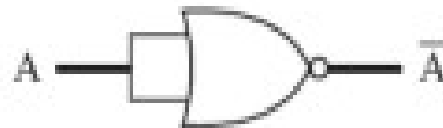
Tập đầy đủ

- 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}

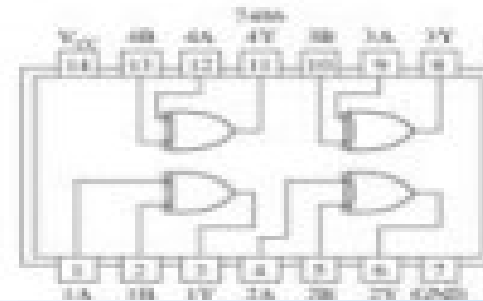
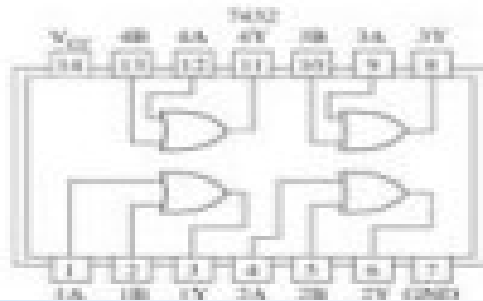
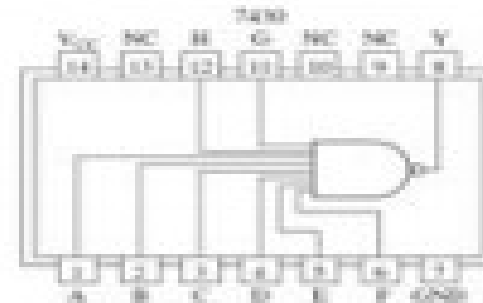
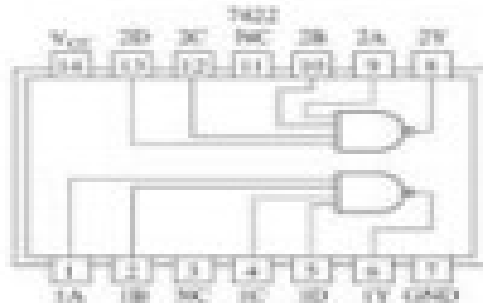
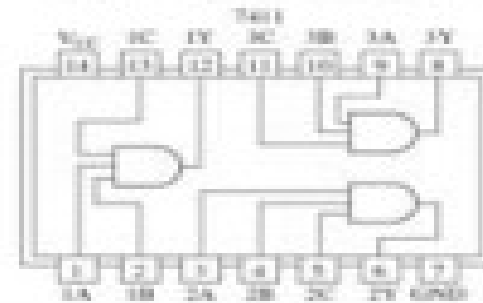
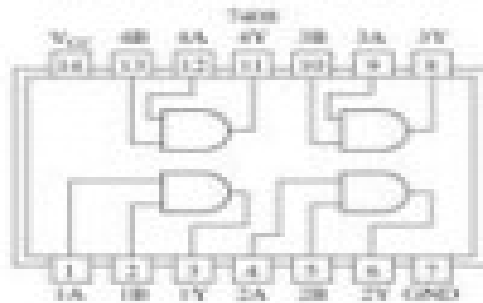
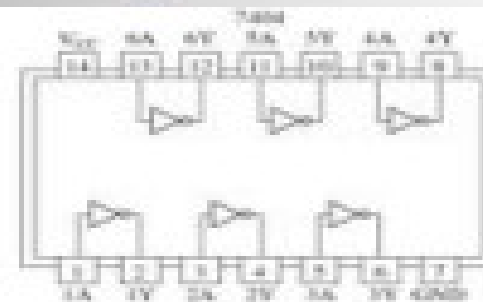
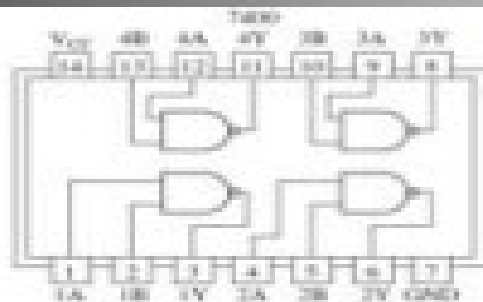
Sử dụng cổng NAND



Sử dụng cổng NOR



Một số ví dụ vi mạch logic





2.4. Mạch tổ hợp

- Mạch logic 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 logic:
 - Mạch logic tổ hợp (Combinational Logic)
 - Mạch không nhớ
 - Đầu ra được xác định bởi các giá trị hiện tại của đầu vào
 - Mạch logic dãy (Sequential Logic)
 - Mạch có nhớ
 - Đầu ra được xác định bởi các giá trị trước đó và giá trị hiện tại của đầu vào

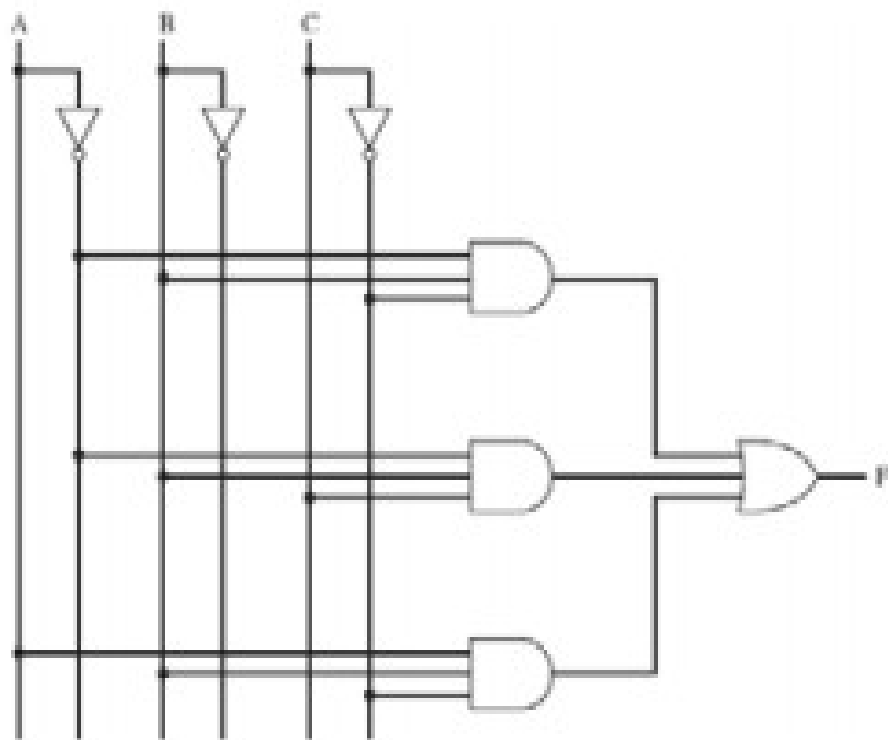


Mạch tổ hợp

- 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 cơ bản
- Mạch tổ hợp có thể được định nghĩa theo ba cách:
 - Bảng thật
 - Dạng sơ đồ
 - Phương trình Boole

Ví dụ

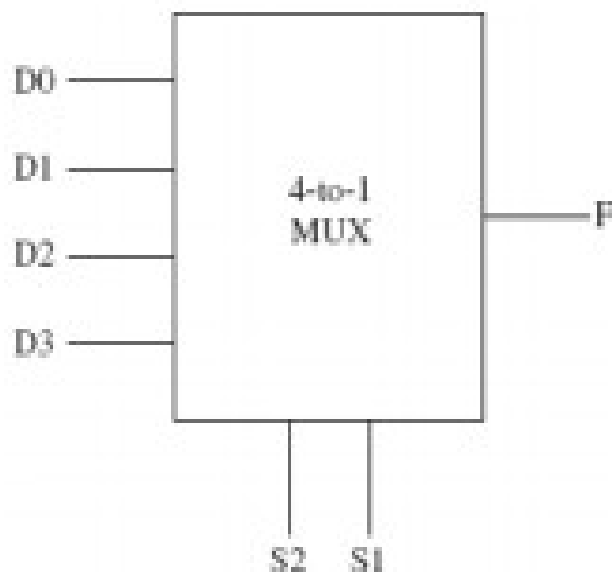
A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0



$$F = \bar{A}\bar{B}\bar{C} + \bar{A}BC + ABC\bar{C}$$

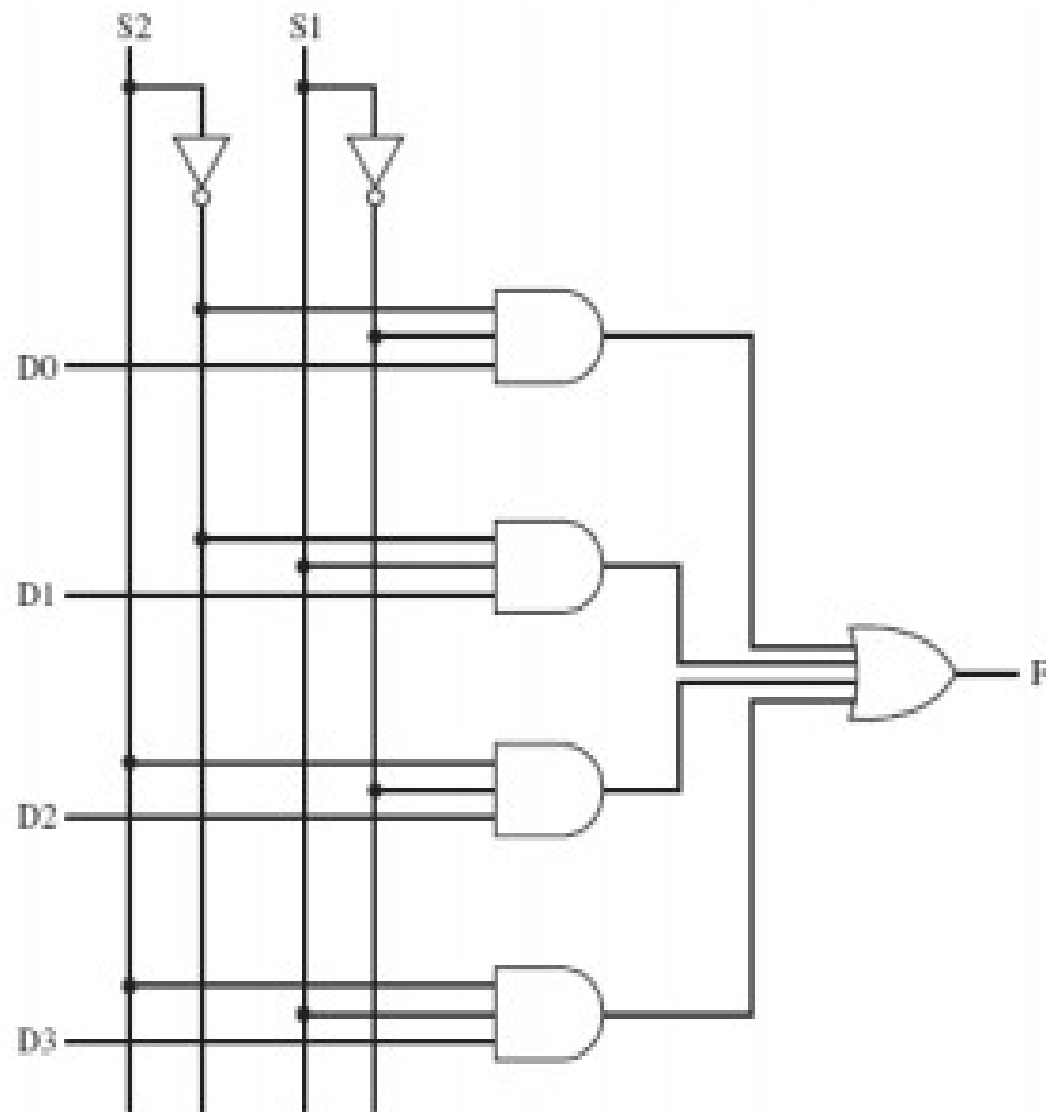
Bộ dồn kênh (Multiplexer-MUX)

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



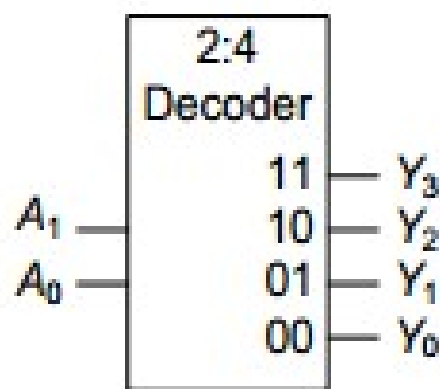
S2	S1	F
0	0	D0
0	1	D1
1	0	D2
1	1	D3

Thực hiện MUX bốn đầu vào

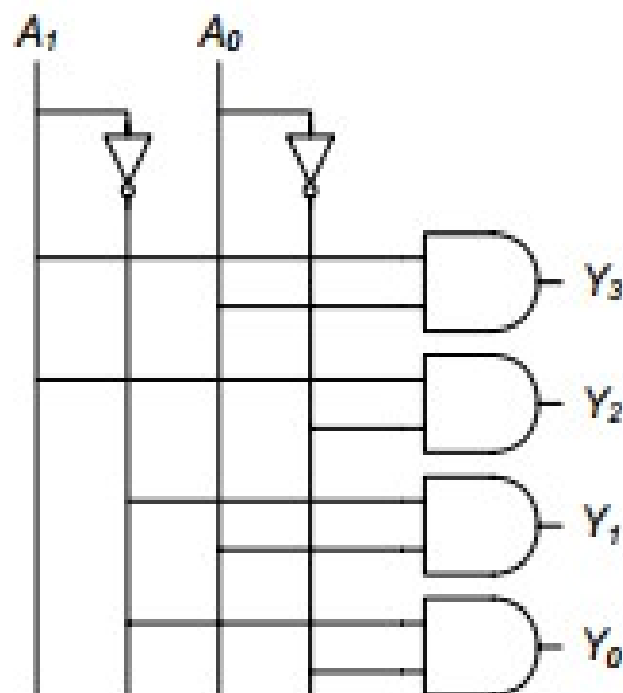


Bộ giải mã (Decoder)

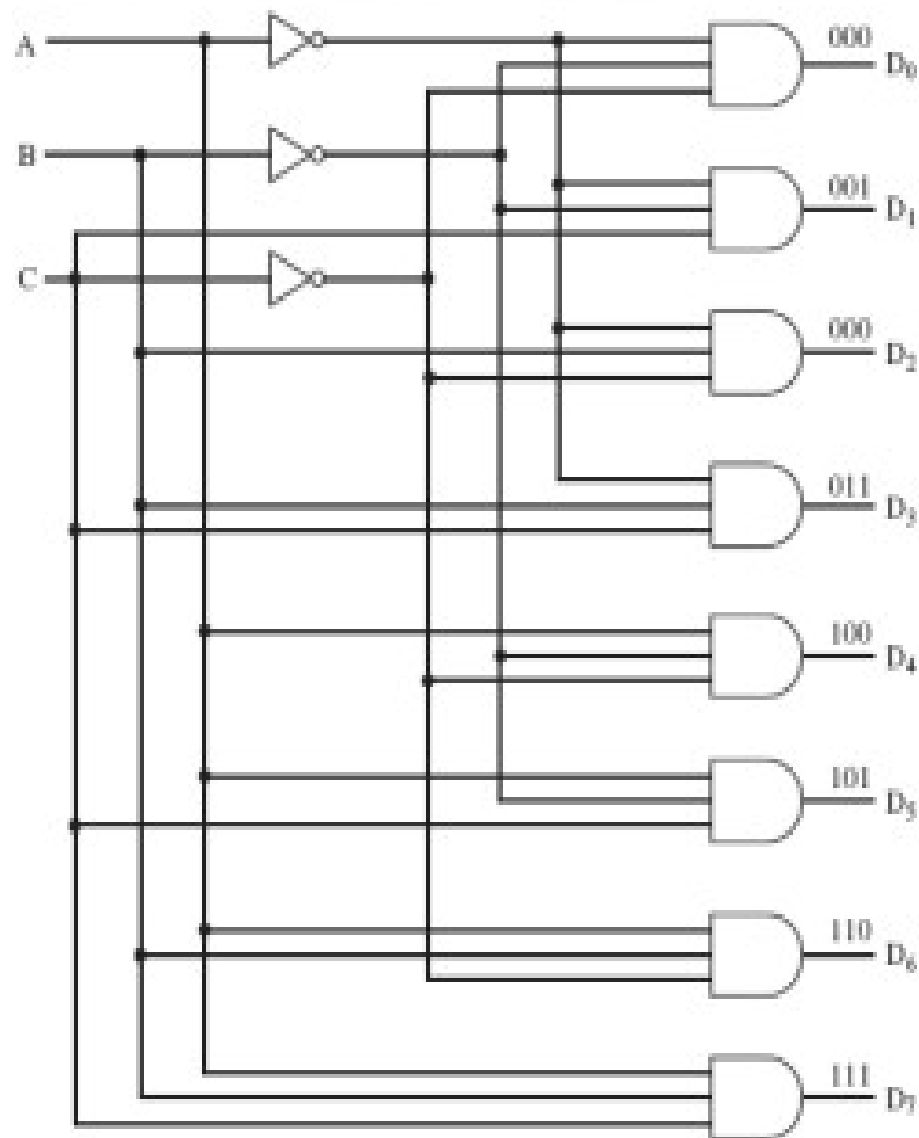
- N đầu vào, 2^N đầu ra
- Chỉ có một đầu ra tích cực (được chọn) tương ứng với một tổ hợp của N đầu vào.



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



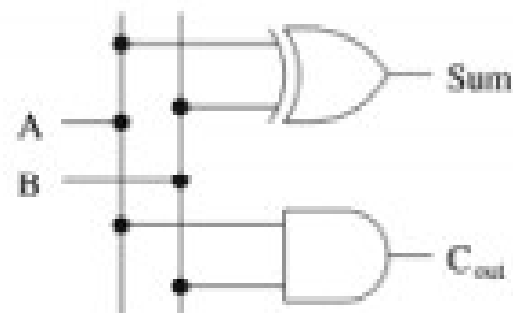


Bộ cộng (Adder)

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

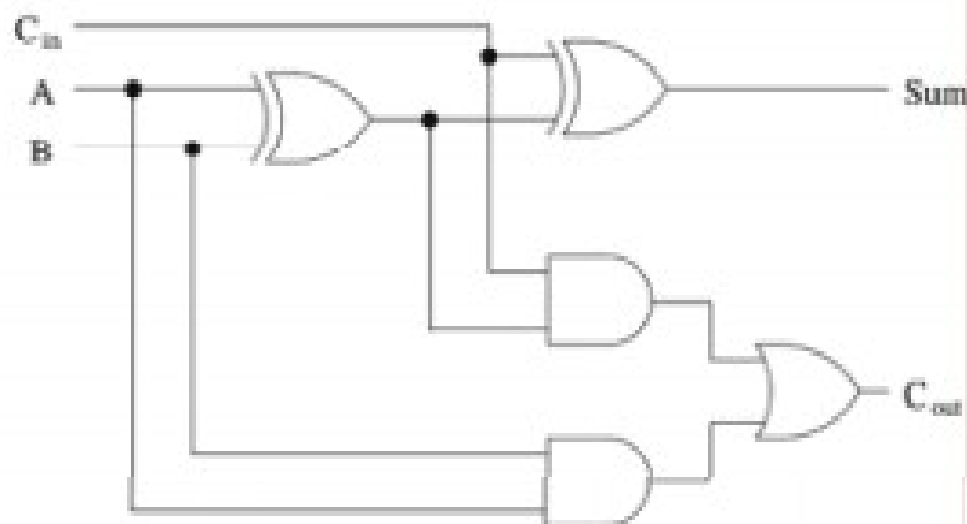
Bộ cộng (tiếp)

A	B	Sum	C_{out}
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



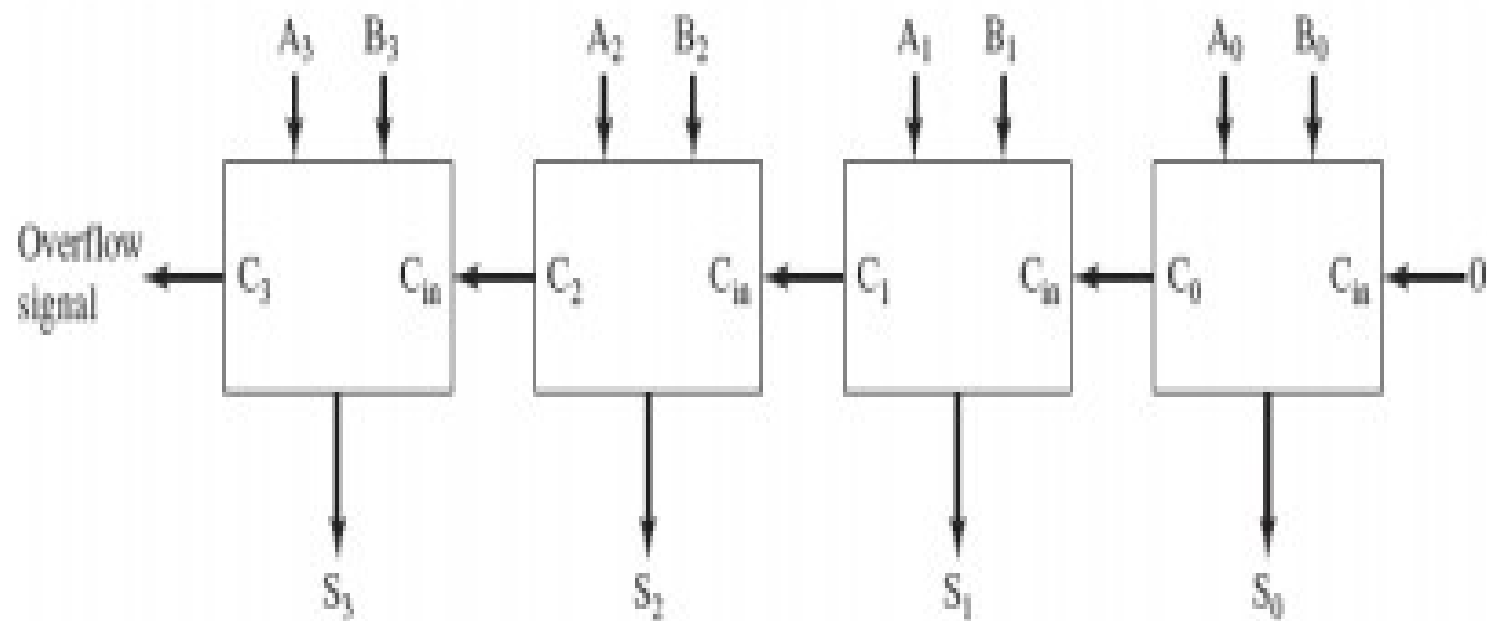
(a) Half-adder truth table and implementation

A	B	C_{in}	Sum	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



(b) Full-adder truth table and implementation

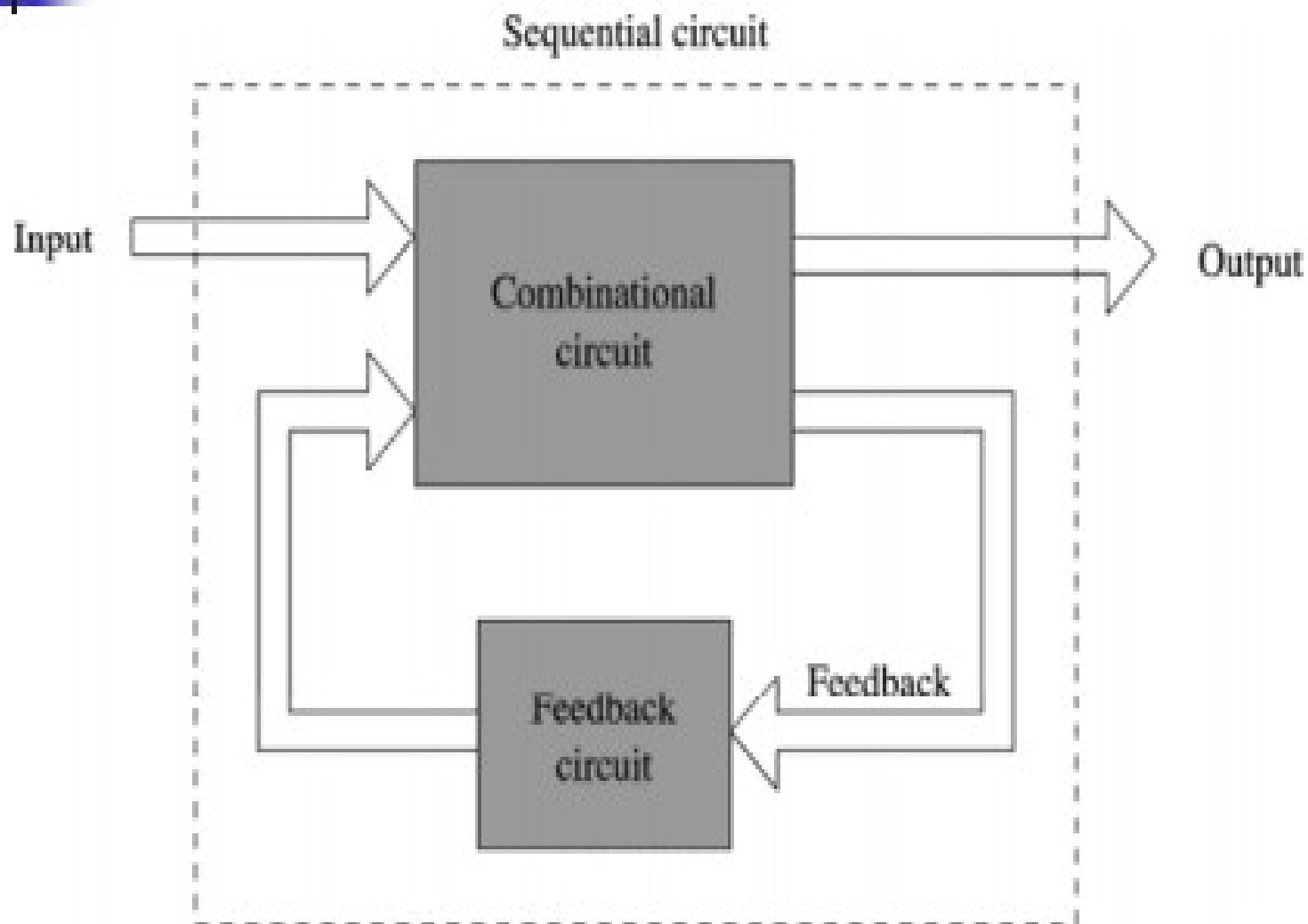
Bộ cộng 4-bit



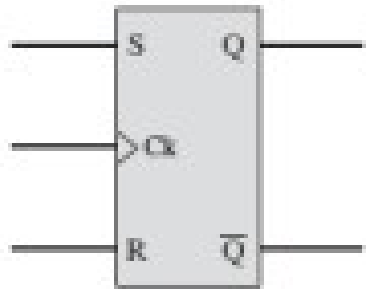
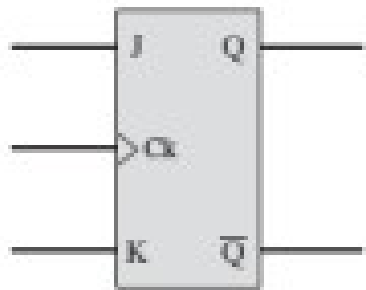
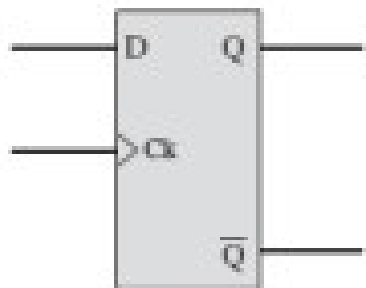
2.5. Mạch dẫy

- Mạch dẫy là mạch logic trong đó đầu ra phụ thuộc giá trị đầu vào ở thời điểm hiện tại và 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 cơ bản
- Mạch dẫy bao gồm:
 - Mạch tổ hợp
 - Mạch hồi tiếp

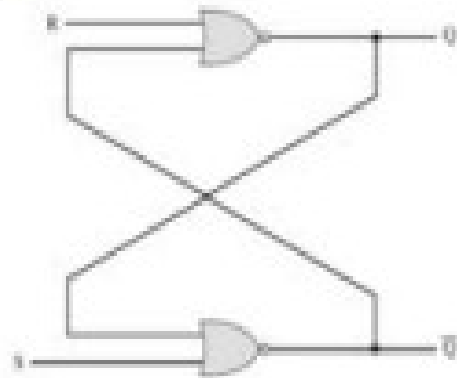
Các thành phần chính của mạch dãy



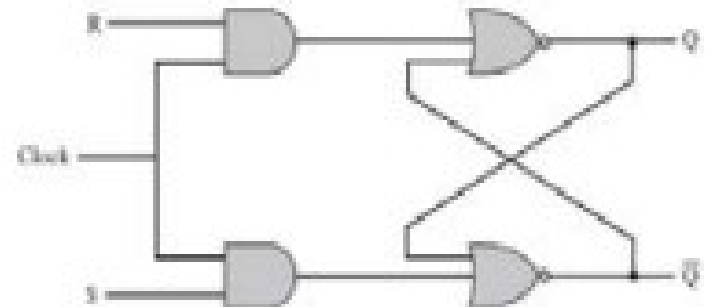
Các Flip-Flop cơ bản

Name	Graphical Symbol	Truth Table															
S-R		<table><tr><th>S</th><th>R</th><th>Q_{n+1}</th></tr><tr><td>0</td><td>0</td><td>Q_n</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>-</td></tr></table>	S	R	Q_{n+1}	0	0	Q_n	0	1	0	1	0	1	1	1	-
S	R	Q_{n+1}															
0	0	Q_n															
0	1	0															
1	0	1															
1	1	-															
J-K		<table><tr><th>J</th><th>K</th><th>Q_{n+1}</th></tr><tr><td>0</td><td>0</td><td>Q_n</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>$\overline{Q_n}$</td></tr></table>	J	K	Q_{n+1}	0	0	Q_n	0	1	0	1	0	1	1	1	$\overline{Q_n}$
J	K	Q_{n+1}															
0	0	Q_n															
0	1	0															
1	0	1															
1	1	$\overline{Q_n}$															
D		<table><tr><th>D</th><th>Q_{n+1}</th></tr><tr><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td></tr></table>	D	Q_{n+1}	0	0	1	1									
D	Q_{n+1}																
0	0																
1	1																

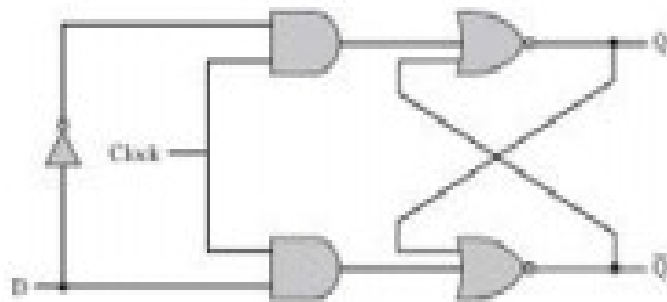
R-S Latch và các Flip-Flop



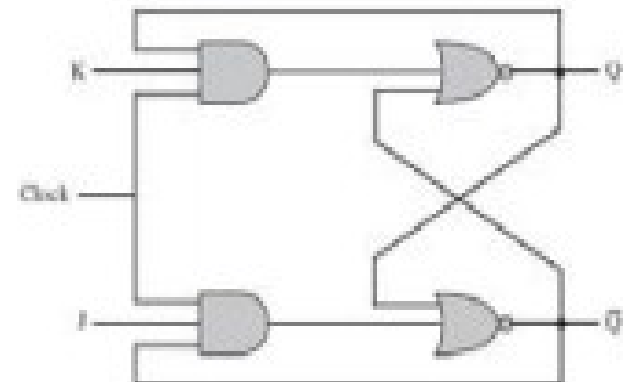
R-S Latch



R-S Flip Flop

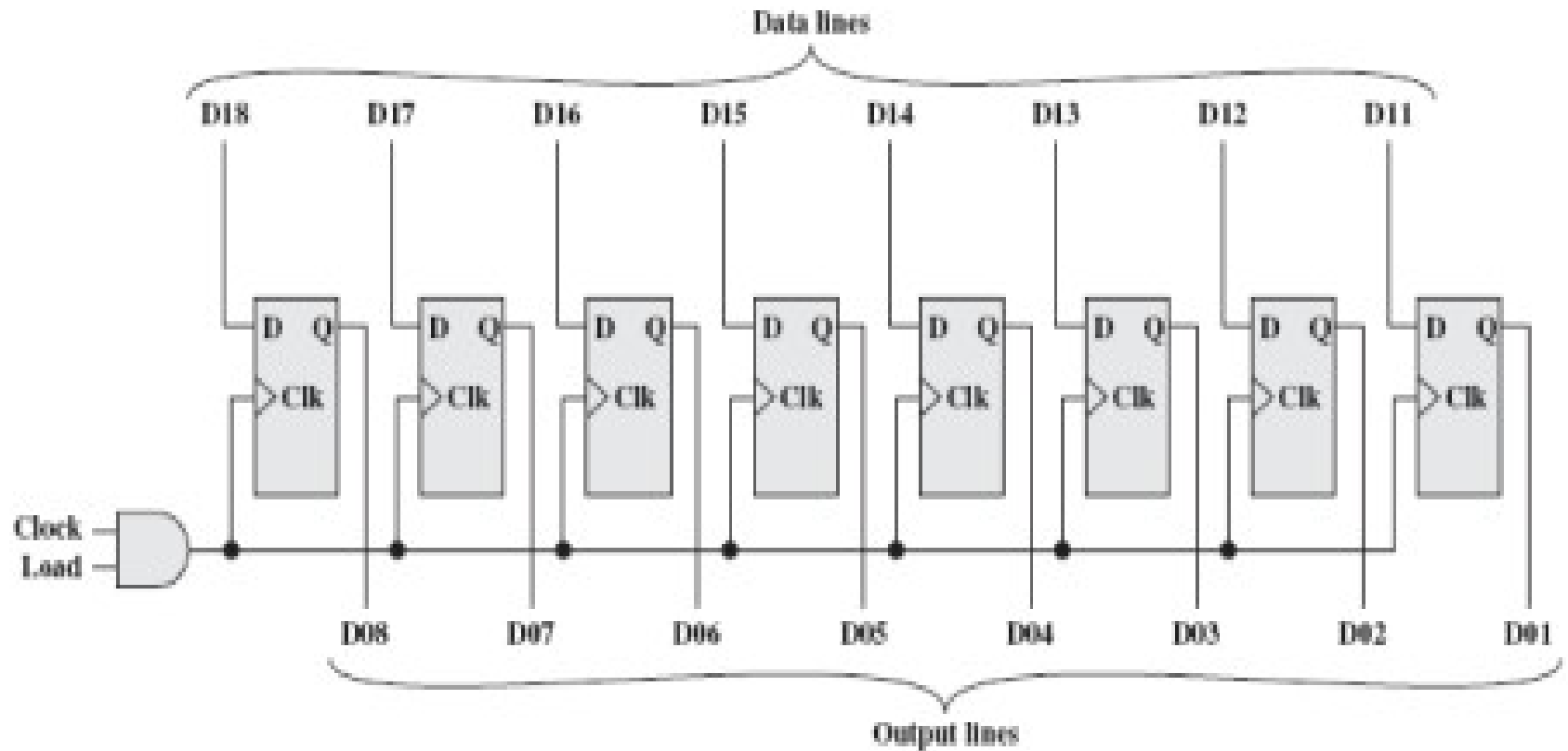


D Flip Flop

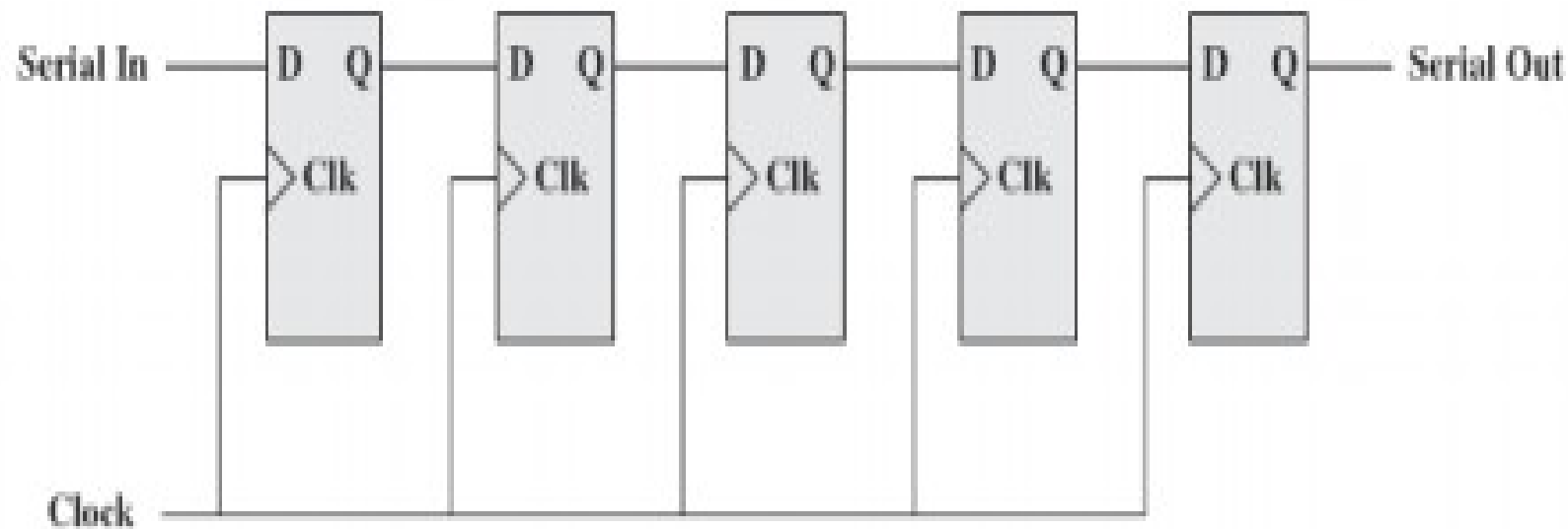


J-K Flip Flop

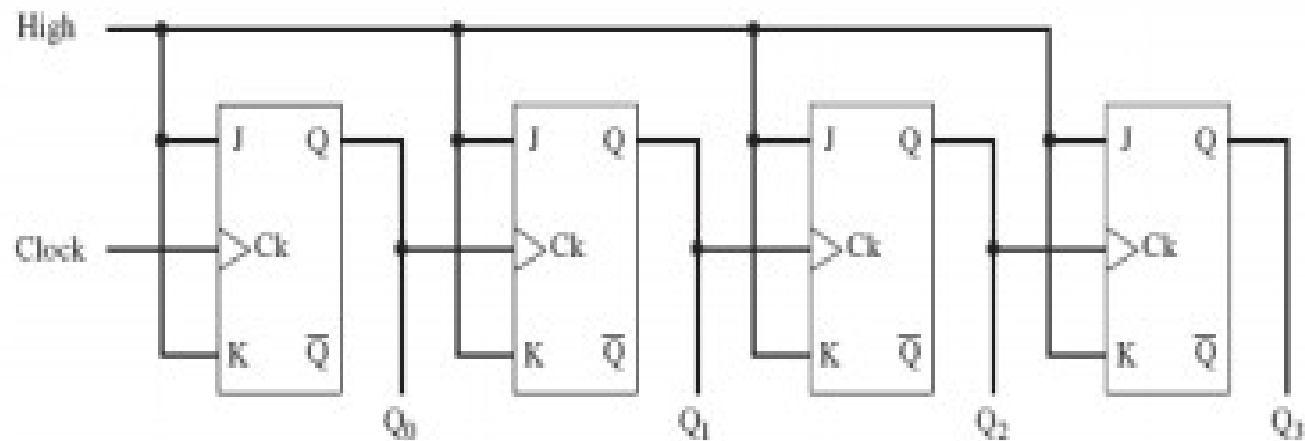
Thanh ghi 8-bit song song



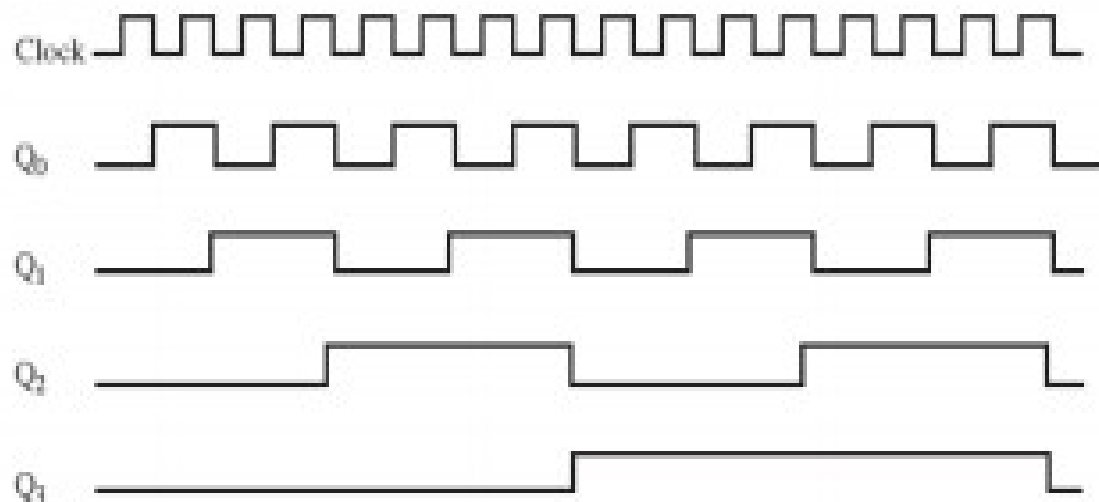
Thanh ghi dịch 5-bit



Bộ đếm 4-bit



(a) Sequential circuit



(b) Timing diagram

Biểu diễn ký tự

- Bộ mã ASCII (American Standard Code for Information Interchange)
 - Do ANSI (American National Standard Institute) thiết kế
 - Bộ mã 8 bit có thể mã hóa được $2^8 = 256$ 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

Bộ mã ASCII

- Các ký tự chuẩn
26 chữ cái hoa 'A' đến 'Z' có mã từ 41_{16} đến $5A_{16}$ (65 đến 90)
 - 'A' \square $0100\ 0001 = 41_{16}$
 - 'B' \square $0100\ 0010 = 42_{16}$
 - ...
 - 'Z' \square $0101\ 1010 = 5A_{16}$
- 26 chữ cái thường 'a' đến 'z' có mã từ 61_{16} đến $7A_{16}$ (97 đến 122)
 - 'a' \square $0110\ 0001 = 61_{16}$
 - 'b' \square $0110\ 0010 = 62_{16}$
 - ...
 - 'z' \square $0111\ 1010 = 7A_{16}$
- 10 chữ số thập phân từ 0 đến 9 có mã từ 30_{16} đến 39_{16} (48 đến 57)
 - '0' \square $0011\ 0000 = 30_{16}$
 - '1' \square $0011\ 0001 = 31_{16}$
 - ...
 - '9' \square $0011\ 1001 = 39_{16}$

Bộ mã hợp nhất 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

CÁC MÃ PHÁT HIỆN LỖI

Ta đã biết đôi khi các dữ liệu trong bộ nhớ hoặc các phần khác của máy tính cũng bị lỗi do nhiều nguyên nhân khác nhau mà chúng ta không thể kiểm soát được hoàn toàn .Như sai hỏng các thiết bị thu phát hay nhiễu lúc này khi dữ liệu được truyền đi đến một địa chỉ nào đó thì một số bit sẽ bị thay đổi so với giá trị thực của nó

- Để tránh việc gây lỗi như vậy chúng ta cần có một số chương trình nhỏ để kiểm tra và sửa lỗi hay còn gọi là các mã phát hiện ECD (error detecting code), mã sửa lỗi ECC (error correcting code) nhằm mục đích làm cho công việc truyền dữ liệu được thực hiện một cách hiệu quả và chính xác nhất .

* Có khá nhiều mã dùng để phát hiện lỗi nhưng ở đây ta chỉ xét 3 loại mã phát hiện lỗi thông thường đó là :

- + Mã chẵn lẻ (parity code)
- + Mã CRC (Cyclic Redundancy Check)
- + Mã Hamming

Mã chẵn lẻ là một trong những mã đơn giản nhất trong 3 loại mã ta xét .

Nguyên lý: _ Nếu chúng ta có m bit cần truyền đi thì ta sẽ truyền đi thêm 1 bit nữa vào xâu dữ liệu (thông thường $m=7$ hoặc $m=8$ bit)

*** Mã chẵn lẻ (parity code)**

Có 2 loại mã chẵn lẻ:

Mã chẵn-lẻ-chẵn: $=1$ khi có $2n$ bit 1 trong m bit thông tin .

$=0$ khi có $2n+1$ bit 1

trong m bit thông tin .

Mã chẵn-lẻ-lẻ: $=0$ khi có $2n$ bit 1 trong m bit thông tin .

$=1$ khi có $2n+1$ bit 1

trong m bit thông tin .

* Ví dụ: khi ta truyền một chuỗi bit dữ liệu: 10100110. Như vậy giá trị của bit chẵn-lẻ-chẵn phải là 1 hoặc giá trị của bit chẵn-lẻ-lẻ phải là 0. Nếu như bit được truyền đi là giá trị khác thì mã kiểm tra lỗi sẽ nhận biết được. Tuy nhiên Mã chẵn lẻ chỉ phát hiện được các lỗi đơn còn lỗi kép thì không. Ví dụ: vẫn chuỗi bit dữ liệu trên nhưng khi truyền chuỗi : 10100000 thì mã lại không phát hiện ra lỗi này. Đây là hạn chế lớn nhất của Mã chẵn lẻ

* Mã CRC (Cycle Redundancy Check). Mã CRC dùng để phát hiện các lỗi khi đọc số liệu từ đĩa mềm
Từ các dữ liệu được ghép thêm vào các byte kiểm soát lỗi CRC khi phát .

Nguyên lý tạo mã CRC là:

Ta coi n bit cần truyền đi là các hệ số của đa thức $P(x)$
VD như byte 10100101 ứng với

$$P(x) = 1.X^7 + 1.X^5 + 1.X^2 + 1.X^0 = X^7 + X^5 + X^2 + 1 \quad \text{Sau}$$

đó sử dụng một đa thức sinh $Q(x)$ (Generation Polynomial). Đa thức này do tổ chức viễn thông quốc tế CCITT quy định

$$Q(x) = X^{16} + X^{12} + X^5 + 1$$

Lấy phần dư $R(x)$ của phép chia $P(x)/Q(x)$

$$R(x) = P(x) \% Q(x) .$$

Sau đó truyền đi $P(x)$ và $R(x)$ khi đó mã CRC sẽ kiểm tra được và phát hiện được các lỗi nhóm

* Mã sửa lỗi Hamming

- Đây là loại mã sửa lỗi khá đơn giản nhưng cũng khá hiệu quả. Từ một mã Harming gồm m bit dữ liệu ta sẽ truyền thêm r bit kiểm tra lỗi
- Các bit khi kiểm tra được chọn để đưa vào vị trí nào đó sao cho hợp lý nhất để phát hiện lỗi tốt nhất và chính xác tại vị trí truyền dữ liệu sai - Mã Harming được ký hiệu $H_{n,m}$ với $n=m+r$

1. Đánh dấu tất cả các vị trí bit là lũy thừa hai là các bit chẵn lẻ. (Vị trí 1, 2, 4, 8, 16, 32, 64, v.v..)
2. Tất cả các vị trí bit khác là dành cho dữ liệu được mã hóa. (Vị trí 3, 5, 6, 7, 9, 10, 11, 12, 13, 14, 15, 17, v.v...)
3. Mỗi bit chẵn lẻ tính chẵn lẻ cho một số bit trong từ mã. Vị trí của các bit chẵn lẻ xác định trình tự của các bit mà nó luân phiên kiểm tra và bỏ qua.

Vị trí 1: kiểm tra 1 bit, bỏ qua 1 bit, kiểm tra 1 bit, bỏ qua 1 bit, v.v...
(1,3,5,7,9,11,13,15, ...)

Vị trí 2: kiểm tra 2 bit, bỏ qua 2 bit, kiểm tra 2 bit, bỏ qua 2 bit, v.v...
(2,3,6,7,10,11,14,15, ...)

Vị trí 4: kiểm tra 4 bit, bỏ qua 4 bit, kiểm tra 4 bit, bỏ qua 4 bit, v.v...
(4,5,6,7,12,13,14,15,20,21,22,23, ...)

Vị trí 8: kiểm tra 8 bit, bỏ qua 8 bit, kiểm tra 8 bit, bỏ qua 8 bit, v.v... (8-15,24-31,40-47, ...)

Vị trí 16: kiểm tra 16 bit, bỏ qua 16 bit, kiểm tra 16 bit, bỏ qua 16 bit, vv (16-31,48-63,80-95, ...)

Vị trí 32: kiểm tra 32 bit, bỏ qua 32 bit, kiểm tra 32 bit, bỏ qua 32 bit, vv (32-63,96-127,160-191, ...)

v.v...

Thiết lập một bit chẵn lẻ 1 nếu tổng số những bit ở các vị trí kiểm tra là lẻ. Thiết

Ví dụ: Lập mã hamming

Cho 1 byte dữ liệu: 10011010

Tạo từ dữ liệu, để các khoảng trống cho các bit chẵn lẻ:: 1 0 0 1 1 0 1 0

Tính chẵn lẻ cho mỗi bit chẵn lẻ (một dấu ? Biểu diễn vị trí bit được đặt):

Tại vị trí 1 kiểm tra các bit 1,3,5,7,9,11:
? 1 0 0 1 1 0 1 0. Số bit 1 là chẵn vì vậy đặt 0 vào vị trí 1: **0** 1 0 0 1 1 0 1 0

Từ vị trí 2 kiểm tra các bit 2,3,6,7,10,11:

0 ? **1** 0 0 1 1 0 1 0. Số bit 1 lẻ vậy đặt 1 vào vị trí 2: 0 **1** **1** 0 0 1 1 0 1 0

Từ vị trí 4 kiểm tra các bit 4,5,6,7,12:

0 1 1 ? **0** 0 1 1 0 1 0. Số bit 1 lẻ vậy đặt 1 vào vị trí 4 : 0 1 1 **1** **0** 0 1 1 0 1 0

Từ vị trí 8 kiểm tra các bit 8,9,10,11,12:

0 1 1 1 0 0 1 ? **1** 0 1 0. Số bit 1 là chẵn vì vậy đặt 0 vào vị trí 8: 0 1 1 1 0 0 1 **0** 1 0 **1** 0

Từ mã là: 011100101010.

* Ví dụ: Tìm 1 bit lỗi trong mã hamming

Ta truyền đi xâu bit

1 1 0 1 0 0 1

Giả sử ta truyền sai bit I_3

1 1 0 1 0 **1** 1

Từ vị trí 1, kiểm tra và đếm các bit 1 tại các vị trí 1,3,5,...số bit 1 là 2, OK

Từ vị trí 2, kiểm tra và đếm các bit 1 tại các vị trí 2,3,6,7,...số bit 1 là 3, Not OK

Từ vị trí 4, kiểm tra và đếm các bit 1 tại các vị trí 4,5,6,7,...số bit 1 là 3, Not OK

Cộng các vị trí Not OK ta có: $2+4=6$, vậy bit 6 lỗi, sửa lại ta được 1 1 0 1 0 **0** 1

*** Bài tập:**

* 1,2,3,4,5,6 và 11 trang 80,81,82 sách

Fundamentals of Computer Organization and Architecture

Tác giả: Mostafa Abd-El-Barr và Hesham El-Rewini