CHƯƠNG 6: MẠCH LÀM TOÁN

- SỐ BÙ
- PHÉP TRỪ SỐ NHỊ PHÂN DÙNG SỐ BÙ 1
- PHÉP TRỪ SỐ NHỊ PHÂN DÙNG SỐ BÙ 2
- PHÉP TOÁN VỚI SỐ CÓ DẤU
- MẠCH CỘNG
  - Bán phần
  - Toàn phần
  - Cộng hai số nhiều bít
- MẠCH TRỪ
  - Bán phần
  - Toàn phần
  - Trừ hai số nhiều bit
  - Cộng & trừ hai số nhiều bit trong một mạch
- MẠCH NHÂN
  - Mạch nhân cơ bản
  - Mạch nhân nối tiếp - song song đơn giản
- MẠCH CHIA
  - Mạch chia phục hồi số bị chia
  - Mạch chia không phục hồi số bị chia

6.1 Số bù

Cho số dương N, n bit, các số bù của N được định nghĩa:

Số bù 2: $(N)_2 = 2^n - N$ (số 2^n gồm bit 1 và n bit 0 theo sau)

Số bù 1: $(N)_1 = (N)_2 - 1 = 2^n - N - 1$

Thí dụ 1: N = 1010

Số bù 2 của N là $(N)_2$ = là 10000 - 1010 = 0110

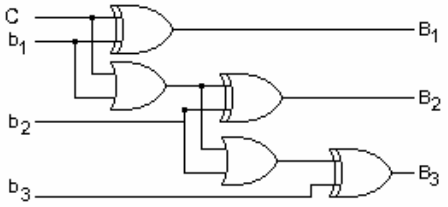
Và số bù 1 của N là $(N)_1$ = 0110 - 1 = 0101

Thí dụ 2: N = 110010101100 $\Rightarrow (N)_2$ = 001101010100 và $(N)_1$ = 001101010011

Nhận xét:

- Để có số bù 2 của một số, bắt đầu từ bit LSB (tận cùng bên phải) đi ngược về bên trái, các bit sẽ giữ nguyên cho đến lúc gặp bit 1 đầu tiên, sau đó đảo tất cả các bit còn lại.
- Để có số bù 1 của một số, ta đảo tất cả các bit của số đó.

Từ các nhận xét trên ta có thể thực hiện một mạch tạo số bù 1 và 2 sau đây:



(H 6.1)

- Khi C=1, B là số bù 1 của b (B_1 và b_1 là bit LSB)

- Khi C=0, B là số bù 2 của b.

Thật vậy, các biểu thức logic của B theo b và C là:

$$B_1 = b_1 \oplus C$$
$$B_2 = b_2 \oplus (C + b_1)$$
$$B_3 = b_3 \oplus (C + b_1 + b_2)$$

- Khi C=1 , các ngã ra cổng OR luôn bằng 1, các cổng EX - OR luôn có một ngã vào bằng 1 nên ngã ra là đảo của ngã vào còn lại, ta được:

$$B_1 = b_1 \oplus 1 = \overline{b_1}$$
$$B_2 = b_2 \oplus (1 + b_1) = b_2 \oplus 1 = \overline{b_2}$$
$$B_3 = b_3 \oplus (1 + b_1 + b_2) = b_3 \oplus 1 = \overline{b_3}$$

- Khi C=0

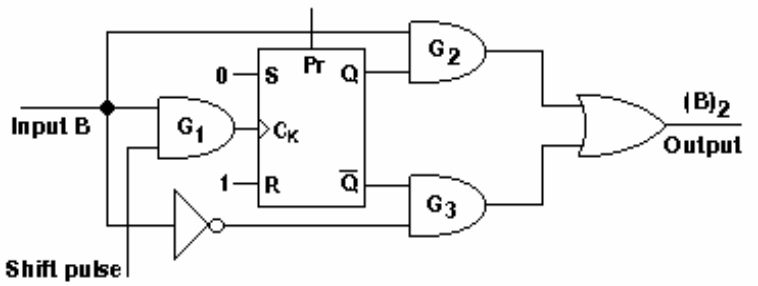
$$B_1 = b_1 \oplus 0 = b_1$$

$$B_2 = b_2 \oplus (0 + b_1) = b_2 \oplus b_1$$
$= b_2$ nếu b_1=0 và $\overline{b_2}$ nếu $b_1 = 1$

$$B_3 = b_3 \oplus (0 + b_1 + b_2) = b_3 \oplus (b_1 + b_2)$$
$= b_3$ nếu b_1 và b_2 đều =0
$= \overline{b_3}$ nếu (b_1 và/hoặc $b_2 = 1$)

Như vậy tất cả các bit sau bit 1 thứ nhất tính từ bit LSB đều bị đảo và B chính là số bù 2 của b

Chúng ta cũng có thể thiết kế mạch tạo số bù hai bằng cách dùng FF RS, có ngã vào R, S tác động mức cao, kết hợp với các cổng logic như (H 6.2). Mạch này dùng khá tiện lợi khi cần thực hiện bài toán cộng và trừ nhiều bit kiểu nối tiếp.



(H 6.2)

Bắt đầu, Preset mạch để ngã ra Q = 1, cổng G_3 đóng, G_2 mở, cho số B đi qua mà không bị đảo cho đến khi có bit 1 đầu tiên đến, cổng G_1 mở cho xung đồng hồ đi qua, FF RS được reset, Q = 0, $\overline{Q}$ = 1, G_2 đóng, G_3 mở, số B đi qua cổng G_2 và bị đảo. Ở ngã ra được số bù 2 của B.

6.2 Phép trừ số nhị phân dùng số bù 1:

Cho hai số dương A và B có n bit (nếu số bit khác nhau, ta thêm số 0 vào , mà không làm thay đổi trị, để cả hai có cùng số bit)

a/ - A≤B

Kết quả A-B là số 0 hoặc âm, phép tính được thực hiện như sau:
Tính A - B:

$$A - B = A-B+2^n-1-2^n+1$$
$$= A+(2^n -B -1) - 2^n+1$$
$$= A+(B)_1 - 2^n+1$$
$$= - \{2^n - [A+(B)_1] -1\}$$
$$= - [A+(B)_1]_1$$

Vậy A-B có được bằng cách cộng số bù 1 của B vào A rồi lấy bù 1 của tổng và thêm dấu trừ . Như vậy để thực hiện phép tính trừ ta chỉ cần dùng phép cộng và phép đảo

Thí dụ 3 : Tính 1001 - 11010 dùng số bù 1
Ta có A = 01001 (thêm số 0 vào để có 5 bit như số B
B = 11010 $\Rightarrow (B)_1 = 00101$

$$A-B = - [A+(B)_1]_1 = - (01001+00101) =- (01110)_1$$
$$= - (10001)$$

Trong hệ thập phân, đây là bài toán $9_{10} - 26_{10} = -17_{10}$
Để thấy dấu trừ được nhận ra như thế nào, ta viết lại phép toán:

	A =	01001
+	$(B)_1$ =	00101
số tràn →	0	01110

Không có số tràn (hay số tràn =0) là dấu hiệu của kết quả âm (hoặc =0) và ta phải lấy bù 1, thêm dấu trừ để đọc kết quả cuối cùng: $(01110)_1 = - 10001$

Thí dụ 4: Tính 10110 - 10110
A = 10110 và B = 10110 $\Rightarrow (B)_1 = 01001$

	A =	10110
+	$(B)_1$ =	01001
số tràn →	0	11111
bù 1 của kq →		00000

Trong phép cộng đầu tiên không có số tràn, kết quả xem như số âm (hoặc =0) lấy bù 1 của kết quả ta được A-B =00000.

b/ - A >B

Kết quả A-B là số dương, phép tính được thực hiện theo qui tắc sau:

Cộng A với $(B)_1$ rồi thêm 1 và không quan tâm tới số nhớ cuối cùng

Thí dụ 5: Tính 110101 - 100110 dùng số bù 1
A = 110101 và B = 100110 $\Rightarrow (B)_1 = 011001$

	A =	110101
+	$(B)_1$ =	011001
	1	001110
		+ 1
số tràn →	1	001111

Bỏ qua số nhớ cuối cùng, ta được kết quả A-B =001111.
Trong hệ thập phân đó là bài toán $53_{10} - 38_{10} = 15_{10}$.

Trong phép tính có số tràn chứng tỏ kết quả là số dương. Số 1 cộng thêm vào xem như lấy từ số nhớ đem qua.

Tóm lại, để thực hiện bài toán trừ, A-B, ta cộng A với bù 1 của B. Dựa vào sự có mặt hay không của số tràn mà có biện pháp xử lý kết quả:

- Nếu số tràn =0, kết quả là số âm (hoặc =0) , ta phải lấy bù 1 của kết quả và thêm dấu - để đọc.
- Nếu số tràn =1, ta cộng thêm 1 vào để có kết quả cuối cùng (bỏ qua bit tràn) là một số dương.

6.3 phép trừ số nhị phân dùng số bù 2:

Phép toán dùng số bù 1 có một bất tiện là ta phải thêm bài toán cộng 1 vào, để tránh việc này ta dùng phép toán với số bù 2

Cho hai số dương A và B có n bit

a/ - A<B

Tính A-B:

$A-B = A-B+2^n-2^n$

$= A+(2^n - B\ ) - 2^n$

$= A+(B)_2 - 2^n$

$= - \{2^n - [A+(B)_2] \}$

$= - [A+(B)_2]_2$

Vậy A-B có được bằng cách cộng số bù 2 của B vào A rồi lấy bù 2 của tổng và thêm dấu trừ. Như vậy ta đã chuyển phép tính trừ thành phép cộng

Thí dụ 6: Tính 1001 - 11010 dùng số bù 2

Ta có A = 01001 (thêm số 0 vào để có 5 bit như số B)

$B = 11010 \Rightarrow (N_2)_2 = 00110$

$A-B = - [A+(B)_2]_2 = - (01001+00110) =- (01111)_2$

$= - (10001)$

Ta được lại kết quả trên

Để thấy dấu trừ được nhận ra như thế nào, ta viết lại phép toán:

	A =	01001
+	$(B)_2$ =	00110
số tràn →	0	01111

Không có số tràn là dấu hiệu của kết quả âm và ta phải lấy bù 2, thêm dấu trừ để đọc kết quả cuối cùng: $(01111)_2 = - 10001$

b/ - A ≥B

Kết quả A-B là 0 hoặc số dương, phép tính được thực hiện theo qui tắc sau:

Cộng A với $(B)_2$ và không quan tâm tới số nhớ ở vị trí 2^n

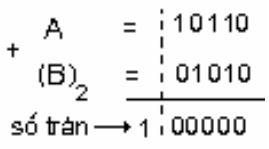
Thí dụ 7 : Tính 110101 - 100110 dùng số bù 2

A = 110101 và B = 100110 $\Rightarrow (B)_2 = 011010$

	A =	110101
+	$(B)_2$ =	011010
số tràn →	1	001111

Có số tràn, kết quả là số dương. Bỏ qua số tràn và đọc ngay kết quả mà không phải biến đổi: $001111 = 15_{10}$

Thí dụ 8 : Tính 10110 - 10110
A = 10110 và B = 10110 $\Rightarrow (B)_2 = 01010$



Bỏ qua số tràn ta được A-B =00000.

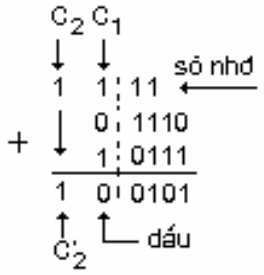
6.4 Phép toán với số có dấu

Cho tới giờ chúng ta thực hiện các phép toán với số không dấu và đôi khi xuất hiện dấu trừ trong kết quả. Trong máy tính, điều này có thể khắc phục được bằng cách dùng số có dấu.

Với qui ước số dương có bit dấu là 0 và số âm có dấu là 1

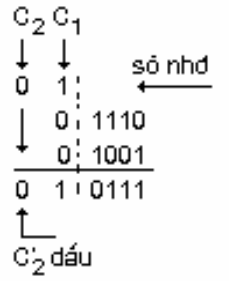
Thí dụ 9: $+10_{10} = 01010$ $+15_{10} = 01111$ $+23_{10} = 010111$
$-10_{10} = 10110$ $- 15_{10} = 10001$ $- 23_{10} = 101001$
Có thể thấy rằng số âm của một số là số bù 2 của nó kể cả bit dấu.
Với cách biểu diễn số có dấu, phép toán trừ trở thành phép toán cộng:
A-B = A+(-B)

Thí dụ 10: Tính A-B =01110 - 01001; B = 01001 = $+9_{10} \Rightarrow - 9_{10} = 10111$

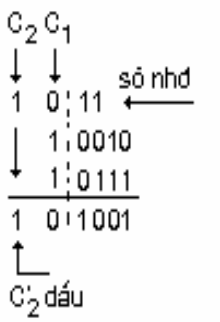


Bit dấu =0 chỉ kết quả dương, bỏ bit tràn C'_2.
Vậy A-B = 00101 $[(+14_{10})-(+9_{10})] = +5_{10}$
Nếu A hoặc B đều dương hoặc âm , kết quả có thể cần thêm một bit do tràn số. Trong trường hợp này bit tràn đầu tiên thuộc kết quả và C'_2 là bit dấu

Thí dụ 11: Tính A+B với A = 01110 $(+14_{10})$ và B = 01001 $(+9_{10})$
Kết quả là 010111 = $+23_{10}$ với $C'_2 = 0$ là bit dấu



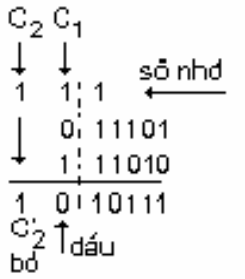
Thí dụ 12: Tính A-B với A =10010 (-14_{10}) và B =01001 ($+9_{10}$)



Một lần nữa C'$_2$ chỉ bit dấu. Kết quả là 101001 = -23_{10} (010111 = 23_{10})

Từ các kết quả trên, ta rút ra qui tắc sau đây:
Nếu $C_1 = C_2$ thì C'$_2$ là bit tràn, bỏ đi và nếu $C_1 \neq C_2$ thì C'$_2$ là bit dấu.

Thí dụ 13: Tính A-B với A =011101 ($+29_{10}$) và B =0110 ($+6_{10}$)
B = 000110 = $+6_{10}$ $\Rightarrow$ - 6_{10} = 111010



Ghi chú: - Trong tất cả trường hợp, ta luôn luôn thực hiện phép cộng do đó có thể bỏ qua phép trừ

- Khi cộng hai số hạng cùng dấu thì có thể xảy ra hiện tượng tràn, lúc đó bit dấu dời về bên trái một bit. Trong các trường hợp khác thì dấu của kết quả ở cùng vị trí với dấu của các số hạng

- Ngoài ra kết quả còn được xử lý tùy vào kết quả so sánh sự khác nhau của hai số nhớ C_1 và C_2 (nhờ một cổng EX-OR).

6.5 Mạch cộng nhị phân:

6.5.1 Mạch cộng bán phần (Half adder, HA):

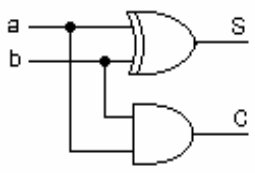
Là mạch cộng hai số 1 bit

vào		ra	
a	b	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

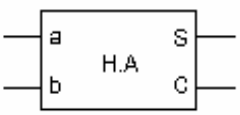
Bảng sự thật

$S = a \oplus b$
$C = a.b$

kết quả



Mạch



Ký hiệu

(H 6.3)

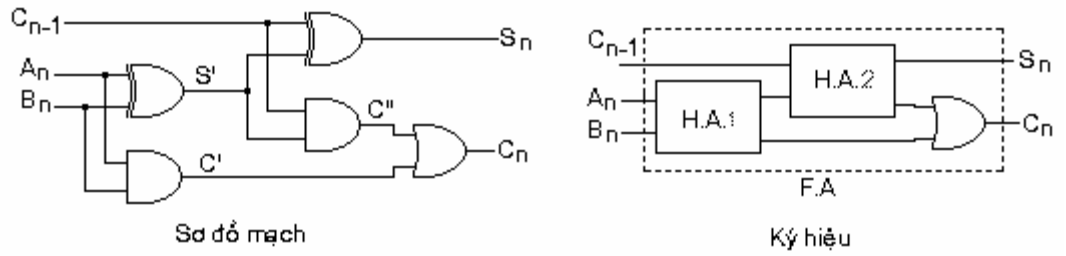
6.5.2 Mạch cộng toàn phần (Full adder,FA) :

Là mạch cộng hai bit ở cùng vị trí trong hai số nhị phân nhiều bit, nói cách khác, đây là mạch cộng hai bit , giả sử thứ n, và bit nhớ có được từ phép cộng hai bit thứ n-1 của hai số nhị phân đó. Ta có bảng sự thật

C_{n-1}	B_n	A_n	S_n	C_n
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

Dùng bảng Karnaugh ta xác định được S_n và C_n như sau:

$$S_n = C_{n-1} \oplus (A_n \oplus B_n)$$
$$C_n = A_n B_n + C_{n-1}(A_n \oplus B_n)$$



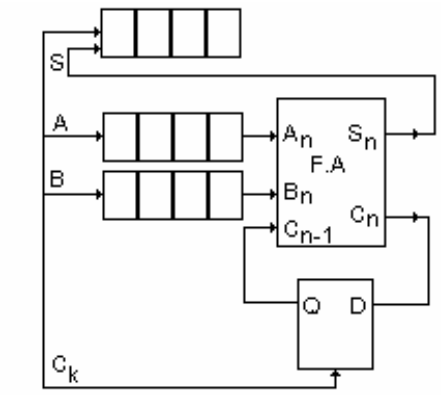
(H 6.4)

Có thể thấy một mạch cộng toàn phần gồm hai mạch cộng bán phần và một cổng OR

6.6 Cộng hai số nhị phân nhiều bit:

6.6.1 Cộng nối tiếp

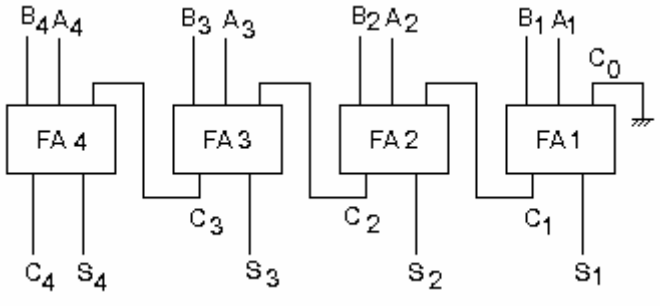
Trong cách cộng nối tiếp, người ta dùng các ghi dịch để chuyển các bit vào một mạch cộng toàn phần duy nhất, số nhớ từ ngã ra C_n được làm trễ một bit nhờ FF D và đưa vào ngã vào C_{n-1}. Như vậy tốc độ của phép cộng tùy thuộc vào tần số xung C_K và số bit phải thực hiện.



(H 6.5)

6.6.2 Cộng song song

Trong cách cộng song song, các bit được đưa đồng thời vào các mạch cộng toàn phần và số nhớ của kết quả ở bit thấp được đưa lên bit cao hơn (H 6.6).



(H 6.6)

Chính vì phải chờ số nhớ mà tốc độ cộng còn hạn chế. Muốn nâng tốc độ cộng lên, người ta thực hiện phép cộng song song định trước số nhớ.

6.6.3 Mạch cộng song song định trước số nhớ

Để tăng tốc độ của mạch cộng song song, người ta tạo trước các số nhớ để đưa đồng thời vào mạch cộng

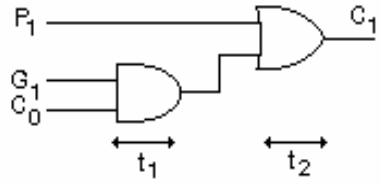
Từ biểu thức xác định số nhớ

$$C_n = A_nB_n + C_{n-1}(A_n \oplus B_n)$$

Đặt $P_n = A_nB_n$ và $G_n = A_n \oplus B_n$

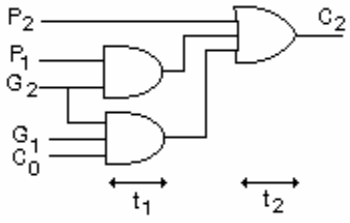
Ta xác định được C_1, C_2, C_3 .... như sau:

$C_1 = P_1 + C_0G_1$



$C_2 = P_2 + C_1G_2$

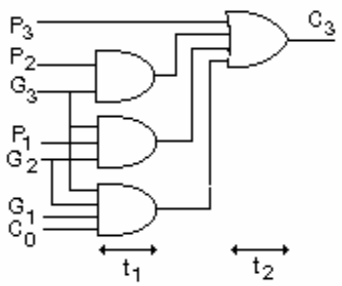
$C_2 = P_2 + P_1G_2 + C_0G_1G_2$



$C_3 = P_3 + C_2G_3$

$C_2 = P_2 + P_1G_2 + C_0G_1G_2$

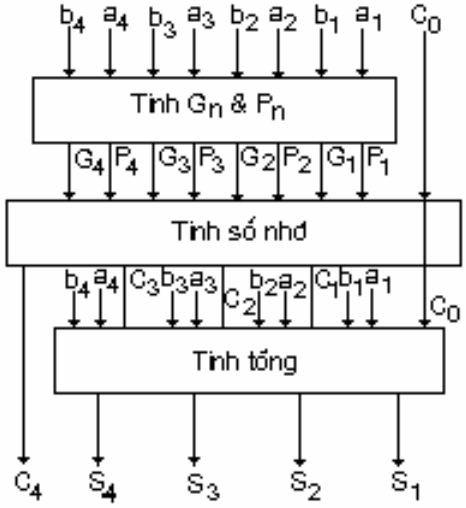
$C_3 = P_3 + P_2G_3 + P_1G_2G_3 + C_0G_1G_2G_3$



(H 6.7)

Nhận thấy thời gian tính số nhớ giống nhau ở các tầng và bằng t_1+t_2 . t_1 là thời gian truyền đồng thời qua các cổng AND và t_2 là thời gian truyền qua cổng OR.

Sơ đồ khối mạch cộng song song định trước số nhớ:



(H 6.8)

Trên thị trường hiện có IC 7483 (tương đương 4008 của CMOS) là IC cộng 4 bit theo kiểu định trước số nhớ.

6.6.4 Cộng hai số BCD

Trên thị trường có các IC cộng số nhị phân, trong lúc trên thực tế nhiều khi chúng ta cần cộng các số BCD để cho kết quả là số BCD.

Chúng ta tìm cách dùng IC 7483 (4008) để cộng hai số BCD

Hai số BCD có trị từ 0_{10} đến 9_{10} khi cộng lại cho kết quả từ 0_{10} đến 18_{10}. Để đọc được kết quả dạng BCD ta phải hiệu chỉnh kết quả có được từ mạch cộng nhị phân.

Dưới đây là kết quả tương đương giữa 3 loại mã: thập phân, nhị phân và BCD

	Phân Nhị					**B C D**					**BCD đọc**
TP	S'=C'$_4$	S'$_4$	S'$_3$	S'$_2$	S'$_1$	S =C$_4$	S$_4$	S$_3$	S$_2$	S$_1$	**theo NP**
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	0	0	0	0	1	1
2	0	0	0	1	0	0	0	0	1	0	2
3	0	0	0	1	1	0	0	0	1	1	3
4	0	0	1	0	0	0	0	1	0	0	4
5	0	0	1	0	1	0	0	1	0	1	5
6	0	0	1	1	0	0	0	1	1	0	6
7	0	0	1	1	1	0	0	1	1	1	7
8	0	1	0	0	0	0	1	0	0	0	8
9	0	1	0	0	1	0	1	0	0	1	9
10	0	1	0	1	0	1	0	0	0	0	16
11	0	1	0	1	1	1	0	0	0	1	17
12	0	1	1	0	0	1	0	0	1	0	18
13	0	1	1	0	1	1	0	0	1	1	19
14	0	1	1	1	0	1	0	1	0	0	20
15	0	1	1	1	1	1	0	1	0	1	21
16	1	0	0	0	0	1	0	1	1	0	22
17	1	0	0	0	1	1	0	1	1	1	23

18	1	0	0	1	0	1	1	0	0	0	24

Nhận thấy:

- Khi kết quả <10 mã nhị phân và BCD hoàn toàn giống nhau
- Khi kết quả ≥10 để có được mã BCD ta phải cộng thêm 6 cho mã nhị phân

Để giải quyết vấn đề hiệu chỉnh này trước tiên ta sẽ thực hiện một mạch phát hiện kết quả trung gian của mạch cộng hai số nhị phân 4 bit. Mạch này nhận vào kết quả trung gian của phép cộng 2 số nhị phân 4 bit và cho ở ngã ra Y = 1 khi kết quả này ≥10, ngược lại, Y=0.

Bảng sự thật

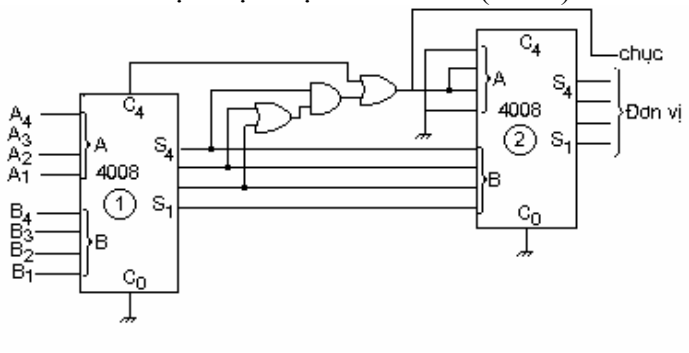
TP	C'_4	S'_4	S'_3	S'_2	Y
0-1	0	0	0	0	0
2-3	0	0	0	1	0
4-5	0	0	1	0	0
6-7	0	0	1	1	0
8-9	0	1	0	0	0
10-11	0	1	0	1	1
12-13	0	1	1	0	1
14-15	0	1	1	1	1
16-17	1	0	0	0	1
18	1	0	0	1	1

Ta không dùng ngã vào S'_1 vì từng cặp trị có $C'_4\,S'_4\,S'_3\,S'_2$ giống nhau thì $S'_1 = 0$ và $S'_1 = 1$

Dùng bảng Karnaugh xác định được Y

$$Y = C'_4 + S'_4\,(S'_3 + S'_2)$$

Và mạch cộng hai số BCD được thực hiện theo sơ đồ (H 6.9)



(H 6.9)

Vận hành:

- IC thứ nhất cho kết quả trung gian của phép cộng hai số nhị phân.
- IC thứ hai dùng hiệu chỉnh để có kết quả là số BCD:
- Khi kết quả < 10, IC 2 nhận ở ngã vào B số 0000 (do Y=0) nên kết quả không thay đổi.
- Khi kết quả trung gian ≥ 10, IC 2 nhận ở ngã vào B số $0110_2 = 6_{10}$ (do Y=1) và kết quả được hiệu chỉnh như đã nói trên.

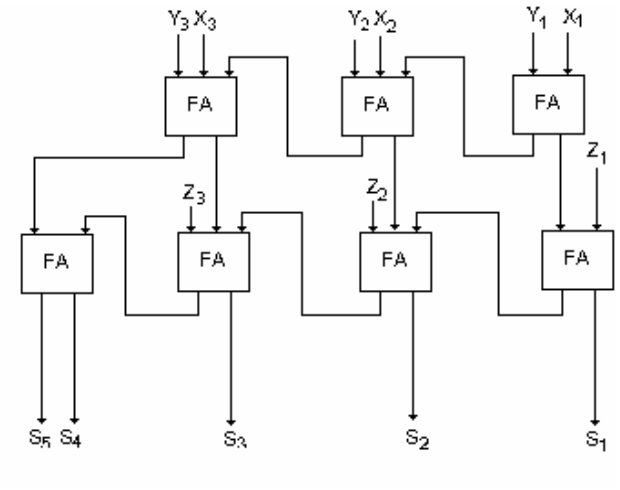
6.6.5 Mạch cộng lưu số nhớ

Nhắc lại, một mạch cộng toàn phần (FA) nhận 3-bit ở ngã vào và cho 2 ngã ra :

- Một là tổng của các bit có cùng trọng số với các bit ở ngã vào
- Một là số nhớ có trọng số gấp đôi trọng số của các bit ở ngã vào

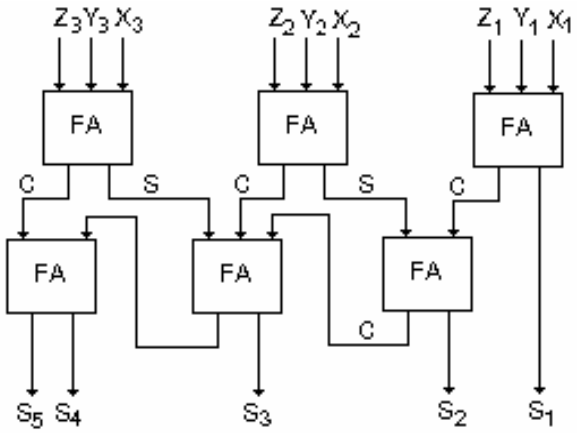
Để cộng một chuỗi số, nhiều mạch cộng toàn phần sẽ được sử dụng, số nhớ được lưu lại để đưa vào mạch cộng bit cao hơn.

Thí dụ 14 : Với 3 số 3-bit X ($X_3X_2X_1$), Y($Y_3Y_2Y_1$), Z ($Z_3Z_2Z_1$) mạch cộng có dạng



(H 6.10)

Người ta dùng mạch cộng loại này để thực hiện bài toán nhân.
Để có kết quả nhanh hơn, có thể dùng mạch (H 6.11)



(H 6.11)

6.7 Mạch trừ nhị phân:

6.7.1 Mạch trừ bán phần

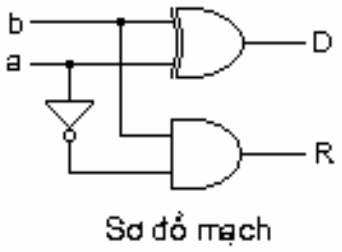
Là mạch trừ hai số 1 bit (H 6.12)

a	b	D	R
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Bảng sự thật

$D = a \oplus b$

$R = \overline{a}.b$



Sơ đồ mạch

(H 6.12)

6.7.2 Mạch trừ có số nhớ (mạch trừ toàn phần)

Là mạch trừ 2 bit có quan tâm tới số nhớ mang từ bit trước

R_{n-1}	A_n	B_n	D_n	R_n
0	0	0	0	0
0	0	1	1	1
0	1	0	1	0
0	1	1	0	0
1	0	0	1	1
1	0	1	0	1
1	1	0	0	0
1	1	1	1	1

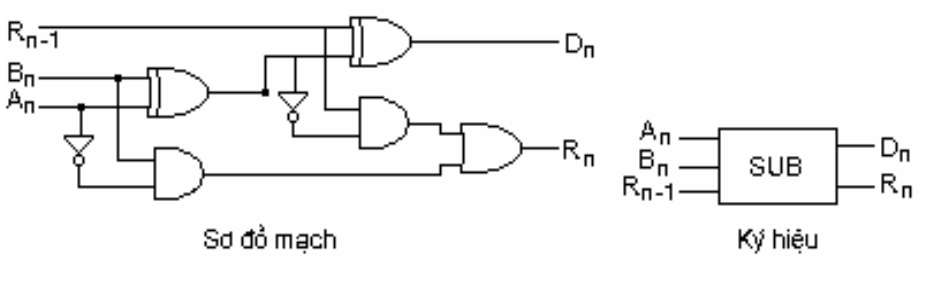
Bảng sự thật

Dùng bảng Karnaugh xác định được các hàm D_n và R_n

$$D_n = R_{n-1} \oplus (A_n \oplus B_n)$$

$$R_n = \overline{A}_n B_n + R_{n-1}\overline{(A_n \oplus B_n)}$$

Và mạch (H 6.13)

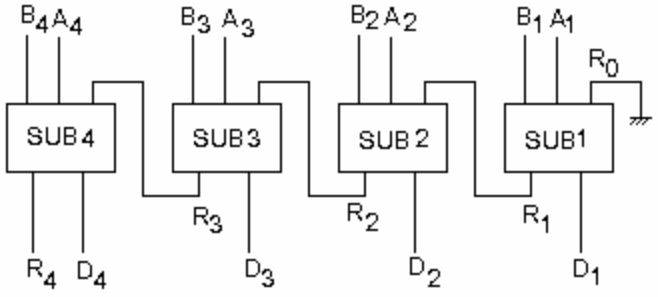


(H 6.13)

Nhận thấy cấu tạo mạch trừ giống như mạch cộng, chỉ khác ở mạch tạo số nhớ

6.7.3 Trừ số nhiều bit

Ta có mạch trừ số nhiều bit bằng cách mắc song song các mạch trừ 1 bit (H 6.14)

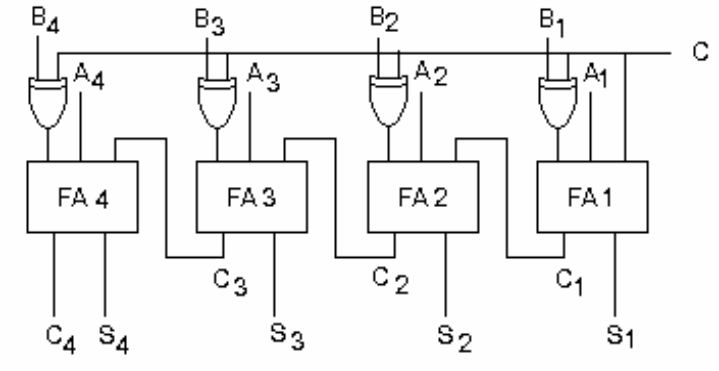


(H 6.14)

6.7.4 Cộng và trừ số nhiều bit trong một mạch

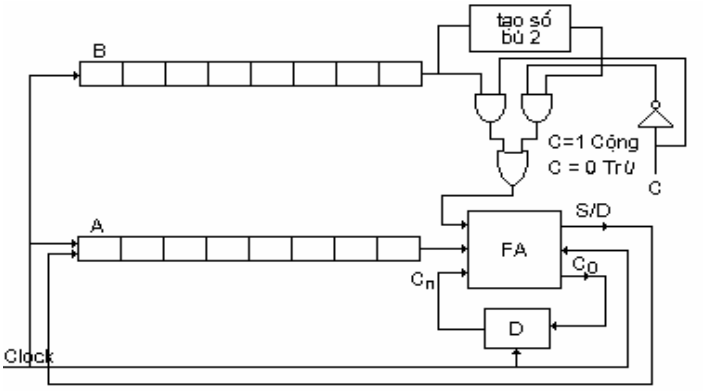
Nhắc lại để thực hiện phép toán trừ, người ta cộng với số bù 1 và cộng thêm 1 (hoặc cộng với số bù 2), như vậy để thực hiện phép trừ A - B ta tính A + $(B)_1$ + 1 . Mạch (H 6.6) được sửa đổi để có thực hiện phép cộng và trừ tùy vào ngã điều khiển C (H 6.15)

- Khi C=0, ta có mạch cộng
- Khi C =1, ta có mạch trừ



(H 6.15)

Ta cũng có thể thực hiện mạch cộng trừ theo kiểu mắc nối tiếp (H 6.16)



(H 6.16)

Nếu hai số A, B là số 8 bit, có dấu, kết quả được xử lý bởi mạch dò số tràn, thiết kế dựa vào biểu thức: $OV = C_7 \oplus C_8$. Khi OV =1 nghĩa là có số tràn (tức $C_7 \neq C_8$), thì số tràn C_8 sẽ là bit dấu, S_8 là một bit của kết quả và khi OV = 0 (tức $C_7 = C_8$), thì S_8 là bit dấu.

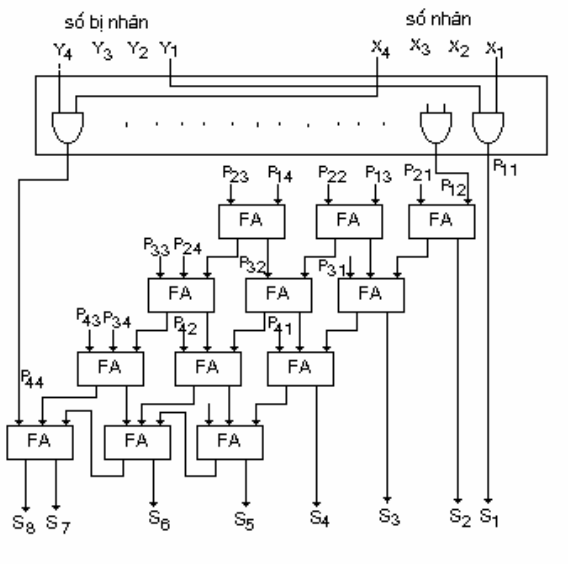
6.8 Mạch nhân

Lấy thí dụ bài toán nhân 2 số 4 bit

				Y_4	Y_3	Y_2	Y_1	Số bị nhân
				X_4	X_3	X_2	X_1	Số nhân
				P_{14}	P_{13}	P_{12}	P_{11}	Tích từng phần
			P_{24}	P_{23}	P_{22}	P_{21}		
		P_{34}	P_{33}	P_{32}	P_{31}			
	P_{44}	P_{43}	P_{42}	P_{41}				
S_8	S_7	S_6	S_5	S_4	S_3	S_2	S_1	Kết quả

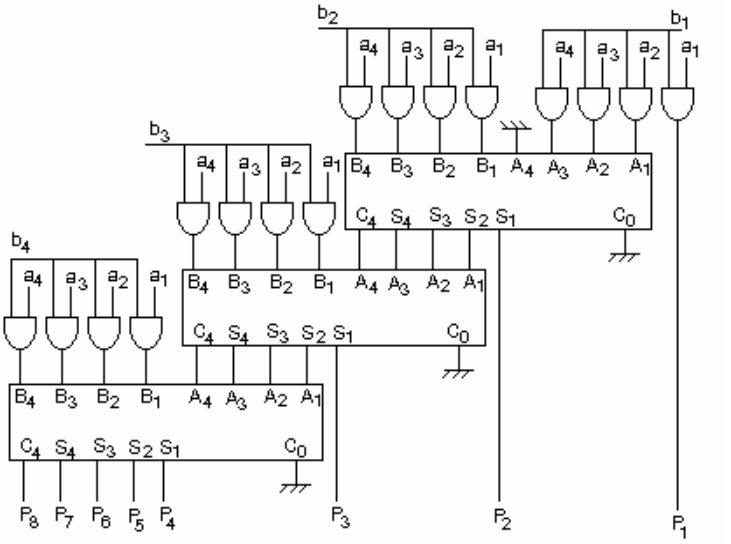
6.8.1. Mạch nhân cơ bản

Việc thực hiện bài toán nhân có thể xem như gồm hai bước:
- Tính các tích từng phần: thực hiện bởi các cổng AND
- Tính tổng của các tích từng phần: Áp dụng bài toán tổng chuỗi số (H 6.17)



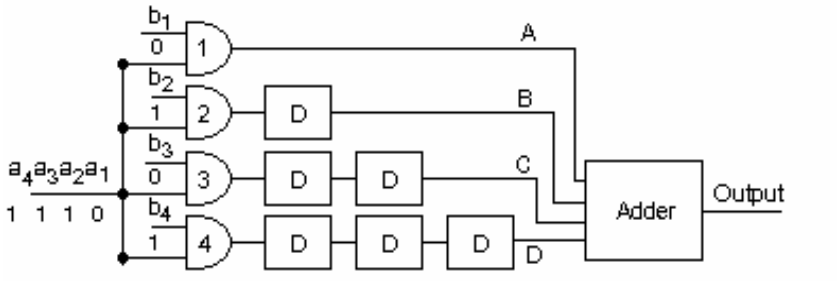
(H 6.17)

Dùng IC cộng 4 bit (7483 hoặc 4008) mạch nhân hai số 4 bit có dạng (H 6.18)



(H 6.18)

6.8.2. Mạch nhân nối tiếp - song song đơn giản (H 6.19)



(H 6.19)

Trong mạch này, một trong hai số được đưa nối tiếp vào mạch (trong trường hợp này là số bị nhân) và số còn lại đưa song song vào mạch.

- Số nhân ($b_4b_3b_2b_1$) đưa song song vào mạch qua các cổng AND đồng thời kiểm soát các cổng này: ứng với bit 1 số bị nhân qua mạch để tới mạch cộng (cổng 2 và 4); ứng với bit 0 ngã ra cổng AND bằng không (cổng 1 và 3)

- Số bị nhân đưa nối tiếp vào mạch theo thứ tự từ bit LSB. Các FF D có tác dụng dịch kết quả của phép nhân (là các tích từng phần) trước khi đưa vào mạch cộng để cộng các tích từng phần này.

Thí dụ 15 : Xem bài toán nhân 10x14. Số nhân là 1010 (10_{10}) và số bị nhân là 1110 (14_{10}). Quá trình nhân giải thích như sau:

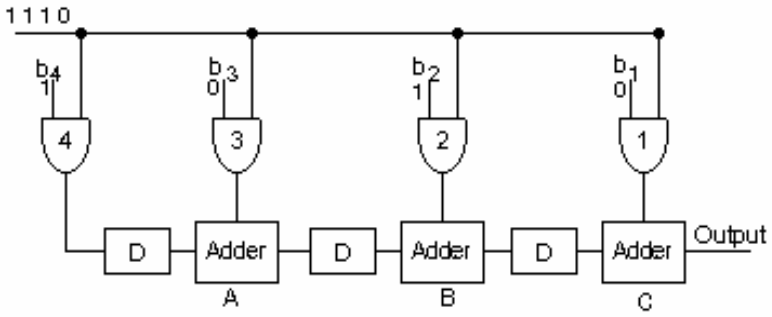
	P_8	P_7	P_6	P_5	P_4	P_3	P_2	P_1
A	0	0	0	0	0	0	0	0
B	0	0	0	1	1	1	0	0
C	0	0	0	0	0	0	0	0
D	0	1	1	1	0	0	0	0
Output	1	0	0	0	1	1	0	0

$10001100_2 = 140_{10}$

Có thể thấy rằng ngã ra A luôn luôn bằng 0 vì bit LSB của số nhân = 0. Ngã ra B có giá trị của số bị nhân được làm trễ 1 bit (1 xung đồng hồ). Ngã ra C được làm trễ 2 bit và luôn bằng 0 (Giống như A). Ngã ra D giống như B nhưng trễ 3 bit. Điều này có thể so sánh với bài toán trên giấy

Số bị nhân					1	1	1	0
Số nhân					1	0	1	0
A					0	0	0	0
B				1	1	1	0	0
C			0	0	0	0	0	0
D		1	1	1	0	0	0	0
Tích	1	0	0	0	1	1	0	0

Muốn không sử dụng mạch cộng số nhiều bit, người ta dùng mạch (H 6.20)



(H 6.20)

Mạch (H 6.20) cần (n-1) mạch cộng và mạch trễ (FF D) cho số nhân n bit. Các cổng AND cho phép các bit của số bị nhân đi qua khi số nhân là 1, số bị nhân (với số bit bất kỳ) được cho vào mạch nối tiếp với bit LSB vào đầu tiên.

Ngã ra cổng 4 sau 4 xung Clock là 1110. Ngã ra cổng 3 luôn luôn bằng 0.

Mạch cộng A cộng số ngã ra 3 và ngã ra 4 bị trễ 1 bit:

	0	0	0	0
1	1	1	0	0
1	1	1	0	0

Tương tự mạch cộng B cộng số bị nhân với kết quả ở A được làm trễ 1 bit

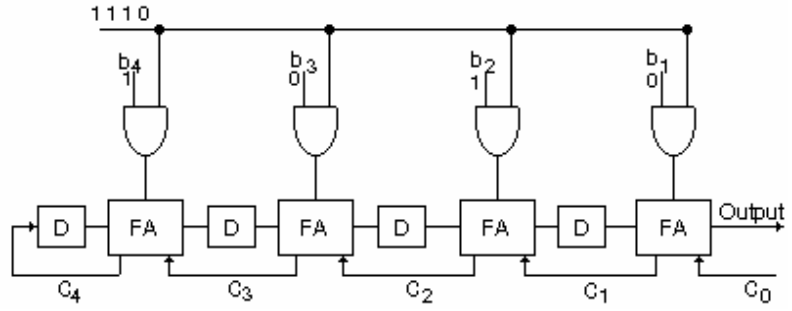
			1	1	1	0
	1	1	1	0	0	0
1	0	0	0	1	1	0

và mạch cộng C

				0	0	0	0
1	0	0	0	1	1	0	0
1	0	0	0	1	1	0	0

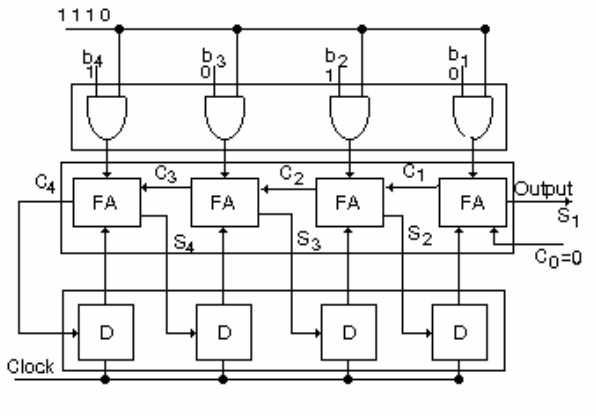
Lưu ý là ở mạch (H 6.20) kết quả cho ở ngã ra mạch cộng C với bit LSB ra đầu tiên, tuy nhiên mạch này chưa quan tâm tới số nhớ.

Mạch (H 6.21) cho kết quả với số nhớ .



(H 6.21)

Và (H 6.22) là một mạch thực tế dùng ghi dịch 4 bit có ngã vào/ra song song, một mạch cộng 4 bit và một chip 4 cổng AND 2 ngã vào để thực hiện bài toán nhân.

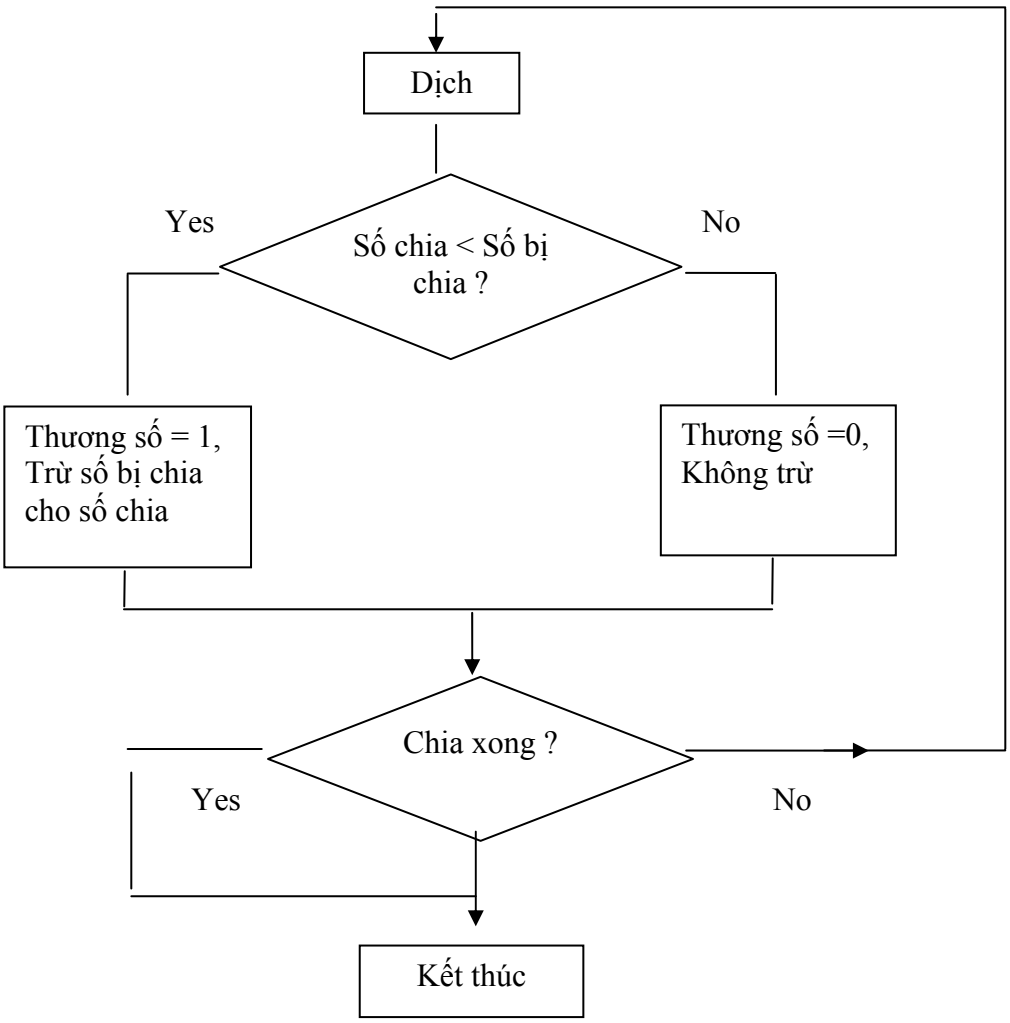


(H 6.22)

6.9 Mạch chia

Nguyên tắc của phép chia số nhị phân là thực hiện phép so sánh một phần của số bị chia (số bit đầu tiên bằng với số bit của số chia) với số chia, nếu số bị chia lớn hơn số chia thì thương số =1, thực hiện phép trừ, ngược lại thì thương số =0, sau đó dịch trái phần còn lại của số bị chia một bit (hoặc dịch phải số chia 1 bit) rồi tiếp tục thực hiện bài toán so sánh giống như trên. Công việc được lặp lại cho đến khi chấm dứt.

Sơ đồ (H 6.23) tóm tắt giải thuật thực hiện bài toán chia

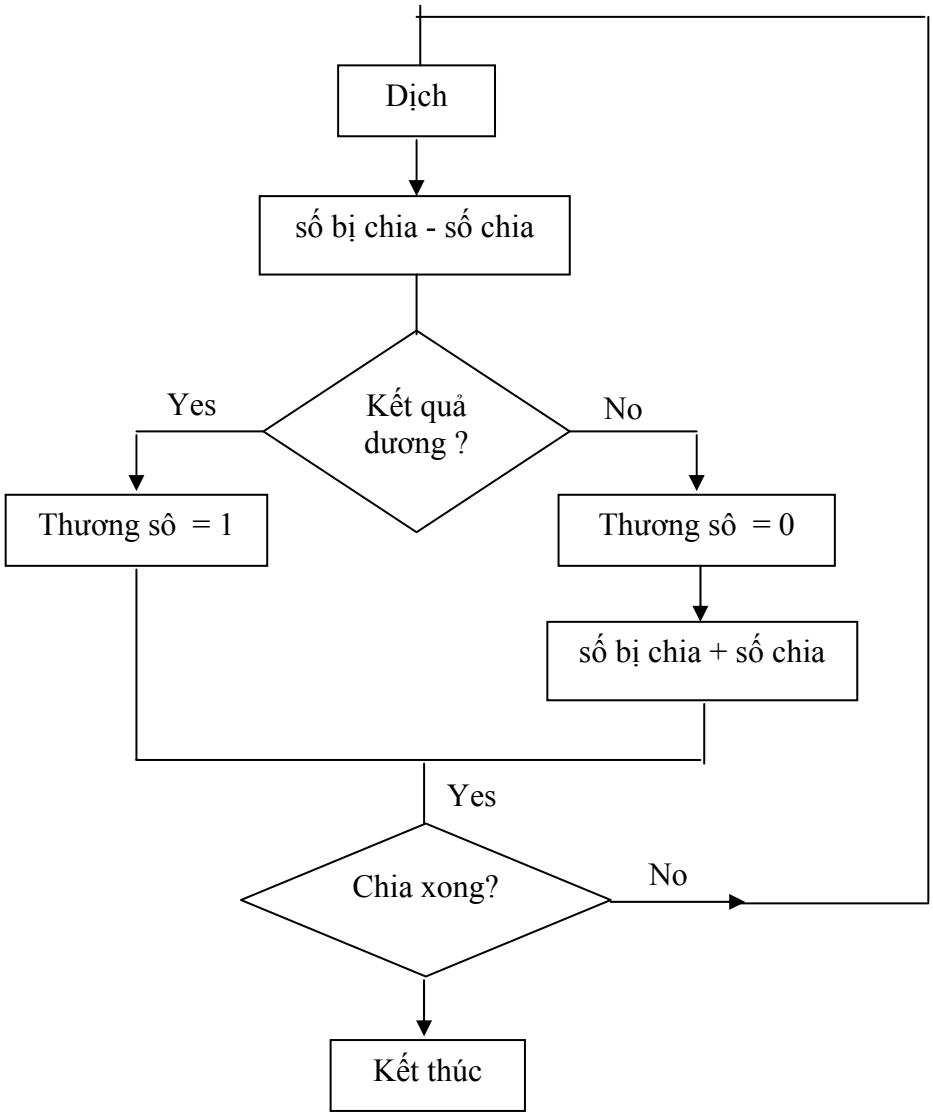


(H 6.23)

6.9.1 Phép chia có phục hồi số bị chia

Thay vì phải thực hiện việc so sánh, người ta làm phép tính trừ một phần số bị chia cho số chia, nếu kết quả dương, thương số là 1, nếu kết quả âm, thương số là 0, trong trường hợp này phải phục hồi lại số bị chia bằng cách cộng số bị chia cho số chia trước khi dịch số bị chia sang trái 1 bit (hoặc số chia sang phải) để tiếp tục lặp lại bài toán cho đến khi kết thúc.

(H 6.24) là sơ đồ giải thuật thực hiện phép chia có phục hồi số bị chia.



(H 6.24)

Để thực hiện phép chia theo sơ đồ trên, ngoài các thanh ghi để chứa các số bị chia, số chia , số thương người ta phải dùng thanh ghi chứa số bị chia được phục hồi.

6.9.2 Phép chia không phục hồi số bị chia

Hệ thống sẽ đơn giản hơn nếu chúng ta dùng phép chia không cần phục hồi số bị chia theo nguyên tắc như dưới đây.

Quan sát giản đồ (H 6.24) ta thấy có 2 trường hợp:

♦ **Số chia lớn hơn số bị chia** (nhánh bên phải)

Lưu ý là dịch số chia về bên phải 1 bit tương đương với chia số đó cho 2

Nhánh bên phải của sơ đồ trên gồm 2 bài toán:

- Cộng số bị chia với số chia.
- Trừ số bị chia cho 1/2 số chia (trừ bị chia cho số chia đã dịch phải)

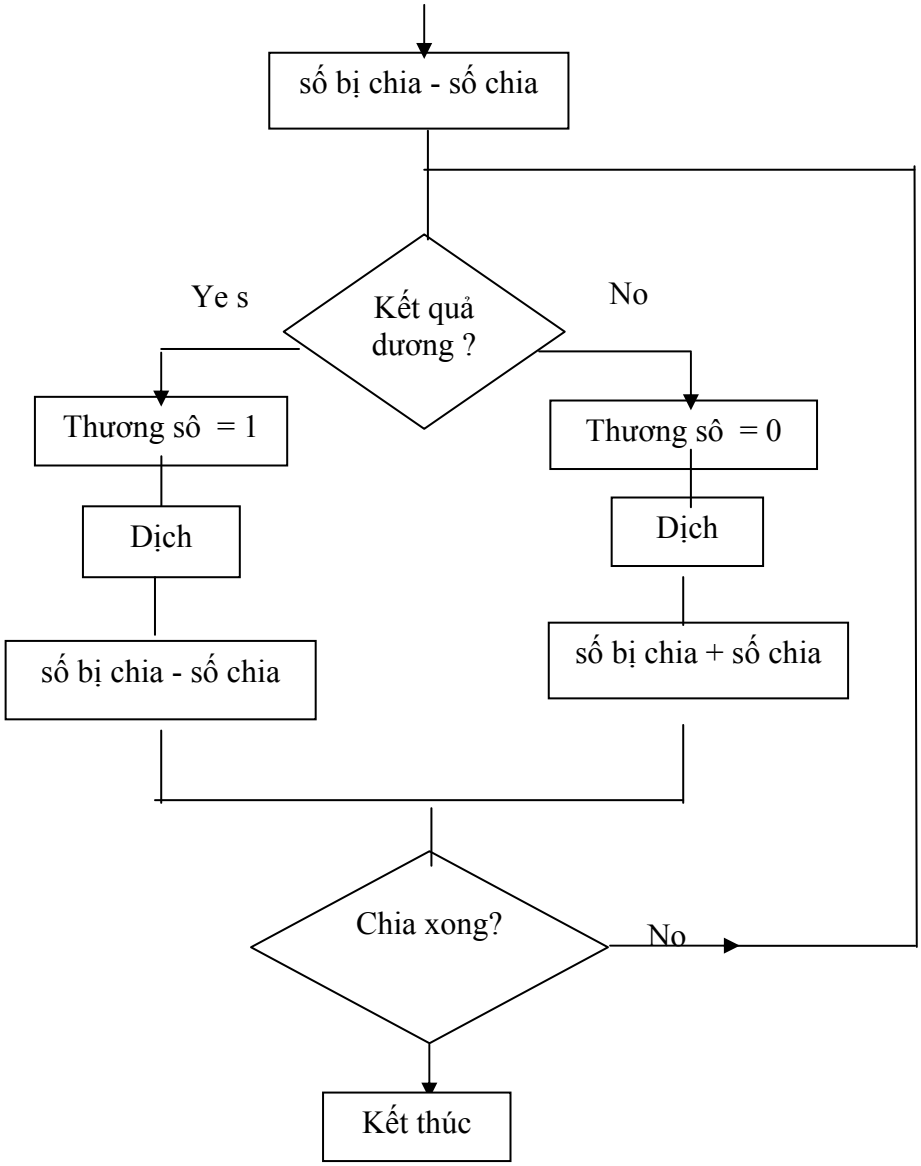
Hai bước này có thể gom lại thành một bước duy nhất như sau:

- Cộng số bị chia với số chia đã dịch phải.

♦ **Số chia nhỏ hơn số bị chia** (nhánh bên trái)

Sau khi lấy kết quả =1, lệnh kế tiếp thực hiện là trừ số chia đã dịch phải.

Từ các kết quả nhận xét trên có thể thay sơ đồ (H 6.24) bởi sơ đồ giải thuật thực hiện phép chia không cần phục hồi số bị chia (H 6.25)



(H 6.25)

Dựa vào sơ đồ (H 6.25), các bước thực hiện bài toán chia như sau:

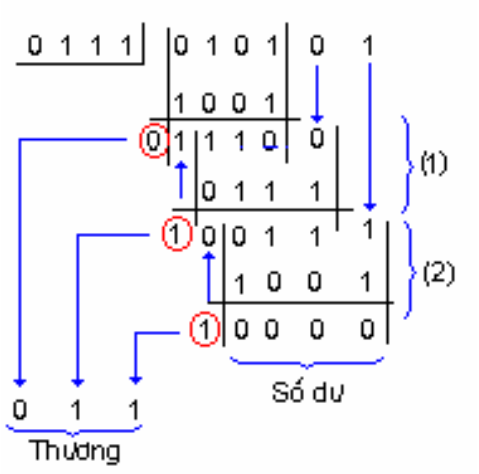
- Số chia (SC) lớn hơn số bị chia (SBC) (SBC - SC < 0), thương số là 0, dịch phải số chia 1 bit (thực tế ta mang thêm 1 bit của số bị chia xuống), thực hiện bài toán cộng số chia và số bị chia

- Số chia nhỏ hơn số bị chia (SBC - SC > 0), thương số là 1, dịch phải số chia 1 bit, thực hiện bài toán trừ (cộng số bù 2) số bị chia cho số chia

Để đơn giản, giả sử số chia và bị chia đều dương (MSB = 0), số bị chia gồm 6 bit và số chia gồm 4 bit.

Thí dụ 1: Thực hiện bài toán chia $21_{10} = 010101_2$ cho $7_{10} = 0111_2$.

Số bù 2 của 0111 là $(0111)_2 = 1001$



Ghi chú:

(1) Số 1 trên mũi tên chỉ rằng kết quả phép toán trừ là số âm, bước kế tiếp là dời và cộng số chia

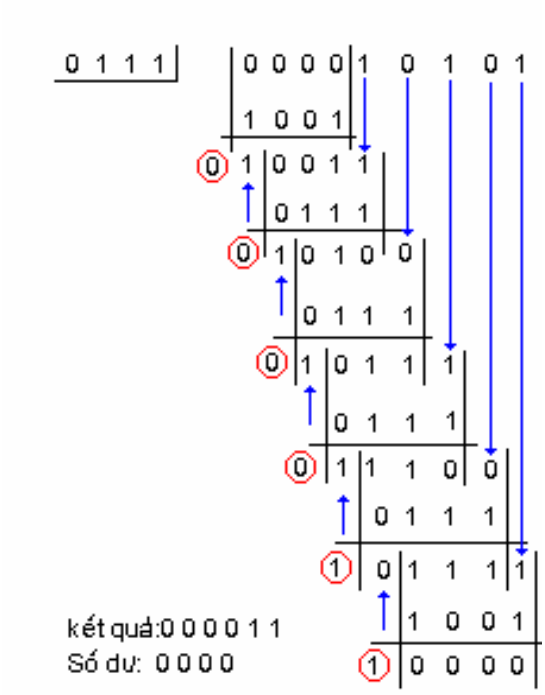
(2) Số 0 trên mủi tên chỉ rằng kết quả phép toán trừ là số dương, bước kế tiếp là dời và trừ số chia (cộng số bù 2)

Thương số có được từ các số tràn mà trên phép tính ta ghi trong vòng tròn.

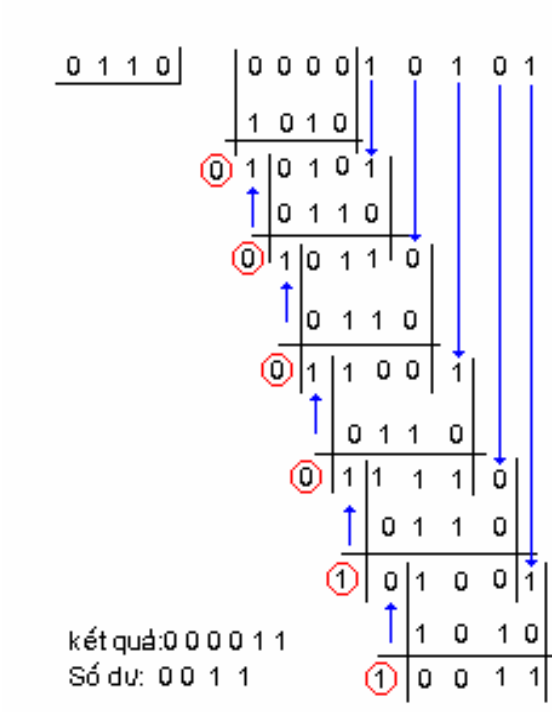
Kết quả: thương là 011(=3) và số dư là 0000(=0)

Bài toán trên cho kết quả với 3 bước cộng/trừ. Tuy nhiên nếu ta chia 21 cho 1 thì cần tới 6 bước cộng trừ để có thương số 6 bit. Một cách tổng quát số bước của bài toán bằng với số bít của số bị chia.

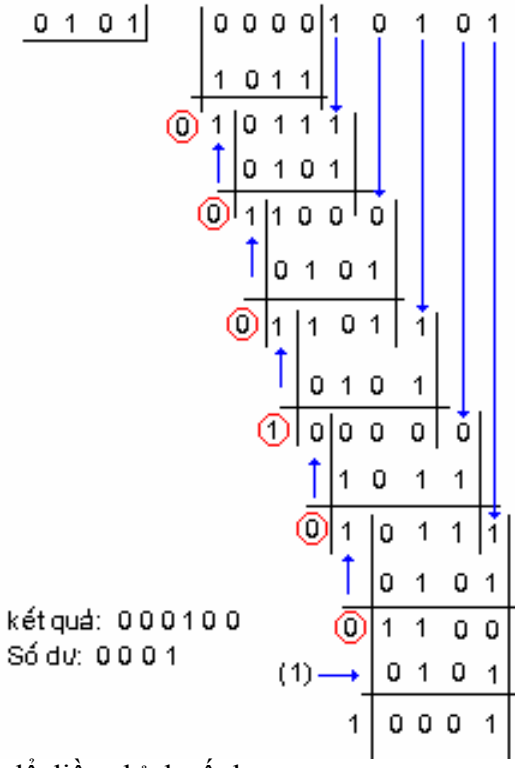
Ta có thể làm lại bài toán với 6 bước cộng/trừ ((thêm 3 bit 0 cho số bị chia)



Thí dụ 2 và 3 dưới đây là bài toán 6 bước
Thí dụ 2 : Chia 21 cho 6 được kết quả 3 và số dư là 3



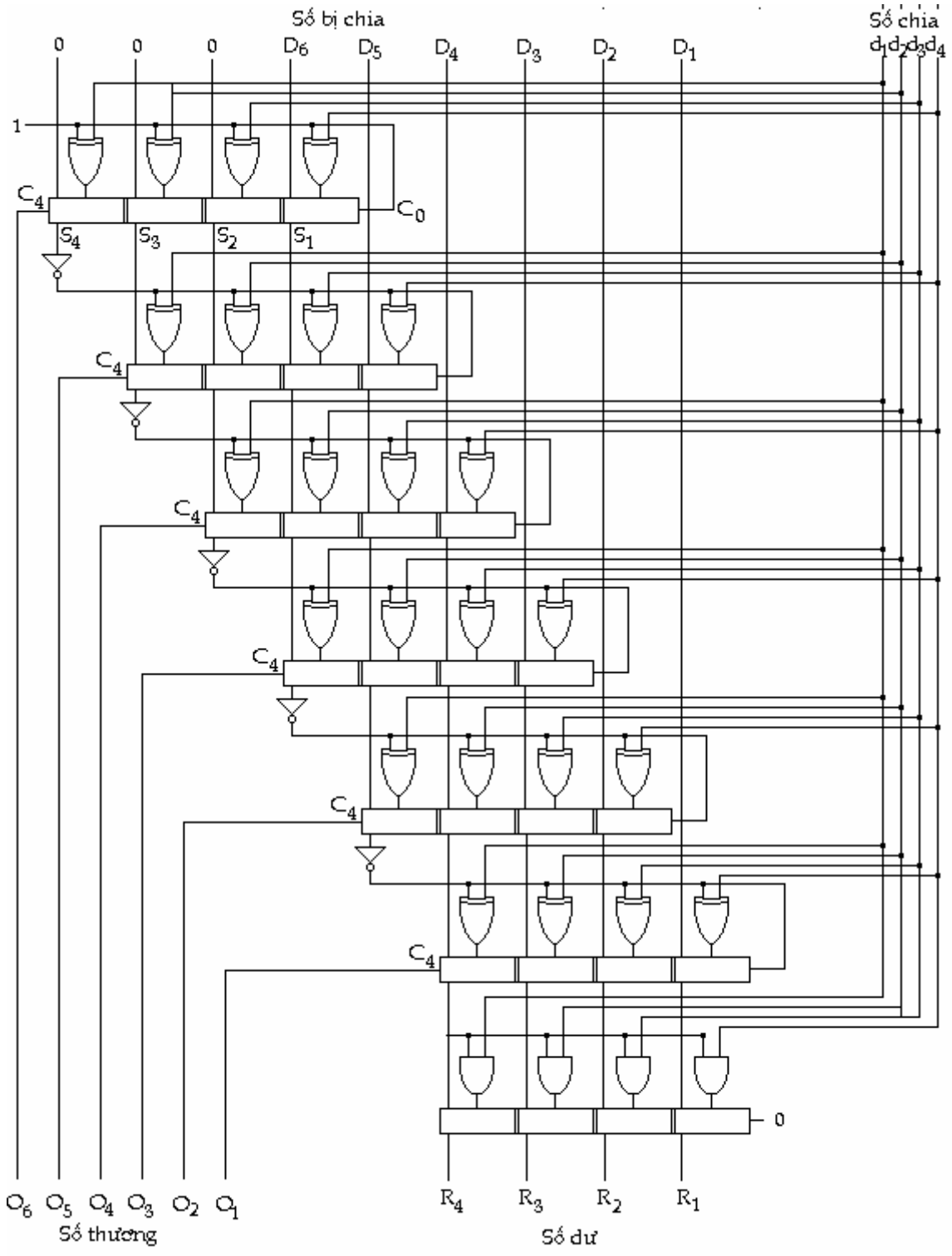
Thí dụ 3 : Chia 21 cho 5, được kết quả 4 và số dư là 1. Tuy nhiên trên phép toán ta thấy phép cộng với số chia cuối cùng cho kết quả âm (số 1100) nên để điều chỉnh số dư ta phải cộng số chia vào và bỏ qua số tràn.



(1) Cộng số chia vào để điều chỉnh số dư

Mạch thực hiện các bài toán này cho ở (H 6.26).

Trong (H 6.26) bước đầu tiên được thực hiện bởi các cổng EX-OR trên cùng có ngã điều khiển = 1 để thực hiện bài toán trừ. Sau bước thứ nhất, bit thứ tư của mạch cộng (S_4) sẽ quyết định phép toán sau đó là cộng (S_4=1) hay trừ (S_4=0) số bị chia với số chia. Số nhớ của bài toán cuối cùng (bước 6) là bit LSB của thương số. Và mạch cộng cuối cùng được thiết kế kết hợp với các cổng AND để xử lý kết quả của số dư như trong hai thí dụ 2 và 3. Nếu kết quả của bài toán ở bước 6 có S_4 = 1 thì cổng AND mở để thực hiện bài toán cộng với số chia để điều chỉnh số dư.


Số bị chia
Số chia
0
0
0
D6
D5
D4
D3
D2
D1
d1 d2 d3 d4
1
C4
C0
S4
S3
S2
S1
C4
C4
C4
C4
C4
0
O6 O5 O4 O3 O2 O1
Số thương
R4 R3 R2 R1
Số dư

(H 6.26)