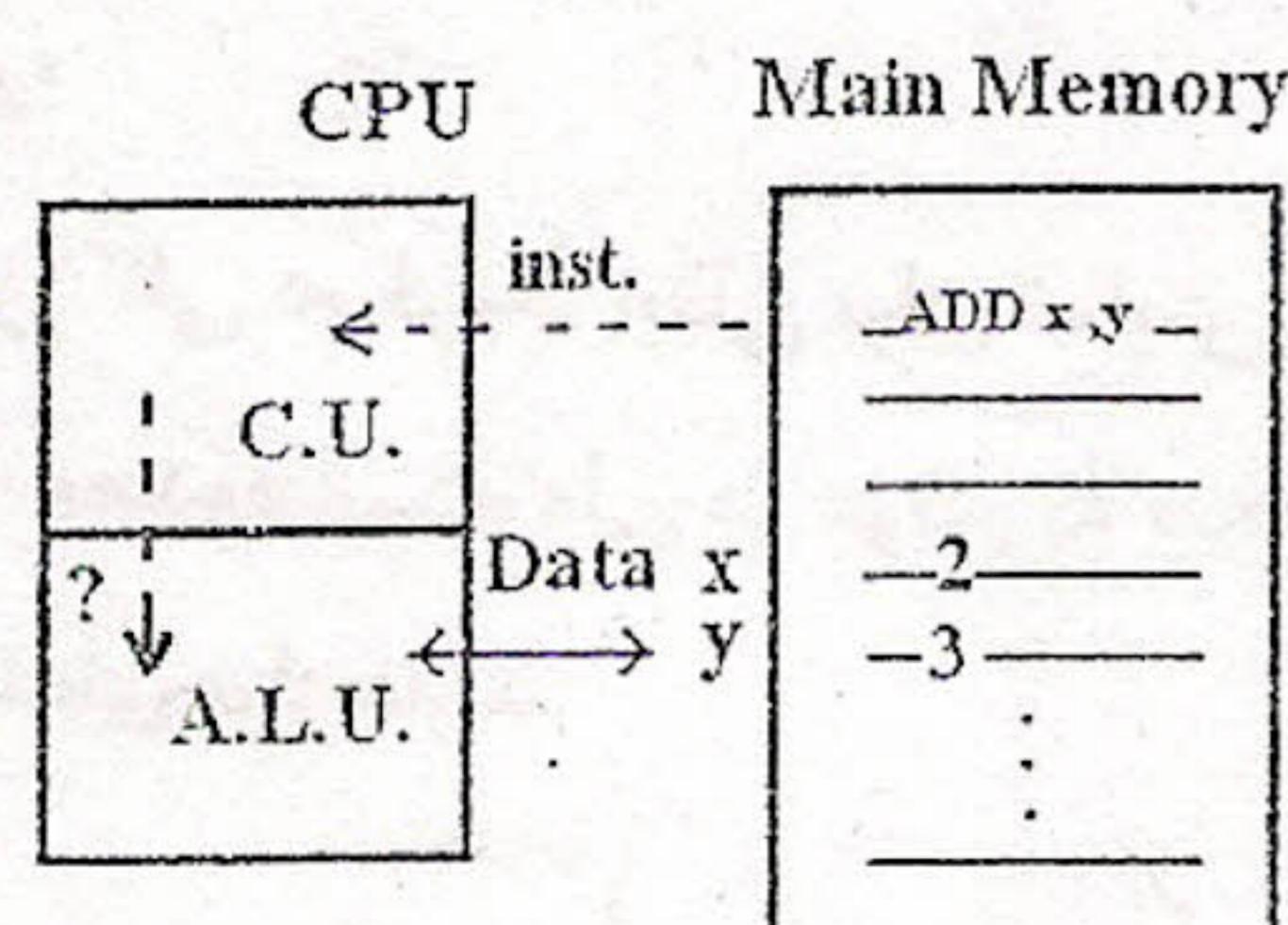


## فصل پنجم

### شیوه نمایش اطلاعات

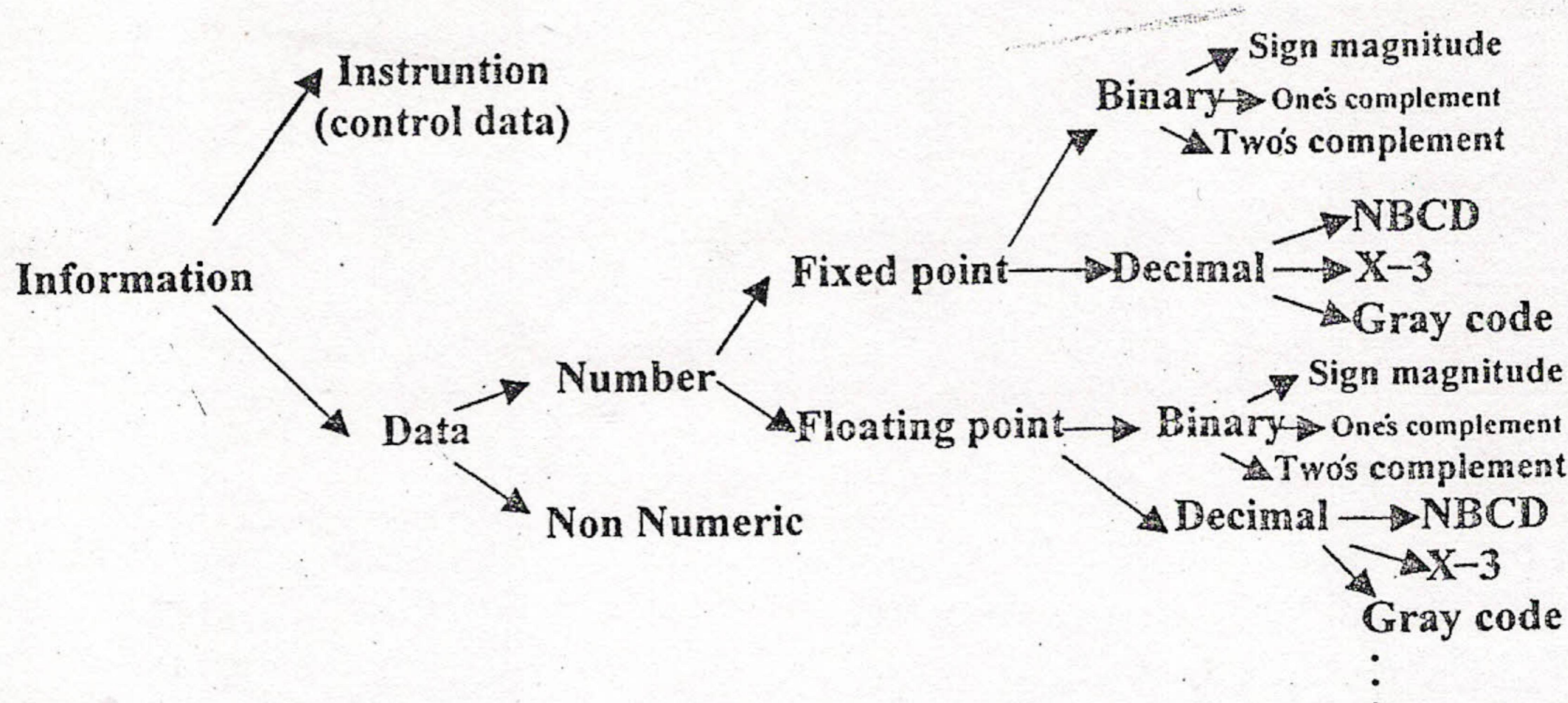
#### نمایش اطلاعات *Information Representation*

می دانیم در محاسبه دو مولفه نقش اساسی دارد:



۱- (حافظه اصلی): که در آن دستورالعملها و داده‌ها ذخیره می‌شوند.

۲- CPU (پردازنده) : که وظیفه آن پردازش دستورالعمل‌هایی است که در حافظه اصلی ذخیره شده‌اند، بنابراین قبل از طراحی مدارهایی مانند Adder باید با نمایش اطلاعات و مزایا و معایب آن‌ها آشنا شویم تا بتوانیم وظیفه طراحی را به درستی انجام دهیم.



در انتخاب نمایش برای اعداد باید به ۴ نکته زیر توجه کرد:

- ۱) نوع عدد: عدد می‌تواند صحیح، کسری و یا مرکب (Mixed) باشد.
- ۲) دامنه اعداد قابل نمایش: یعنی کوچکترین و بزرگترین عدد قابل نمایش.
- ۳) دقت اعداد قابل نمایش: یعنی تا چند بیت و یا چند رقم "Digit" اعشار را می‌توان نشان داد.
- ۴) هزینه سخت‌افزاری: هم برای ذخیره‌سازی و هم پردازش

تذکر:

به طور کلی دو فرمت اصلی برای نمایش اعداد عبارتند از: Floating-point و Fixed-point که در آن فرمت اعداد با ممیز ثابت دارای دامنه محدود است، ولی در عوض به سخت‌افزار ساده‌ای نیاز دارد. ولی اعداد با ممیز شناور دارای دامنه‌ای وسیع هستند ولی سخت‌افزار پیچیده‌ای نیاز دارند.

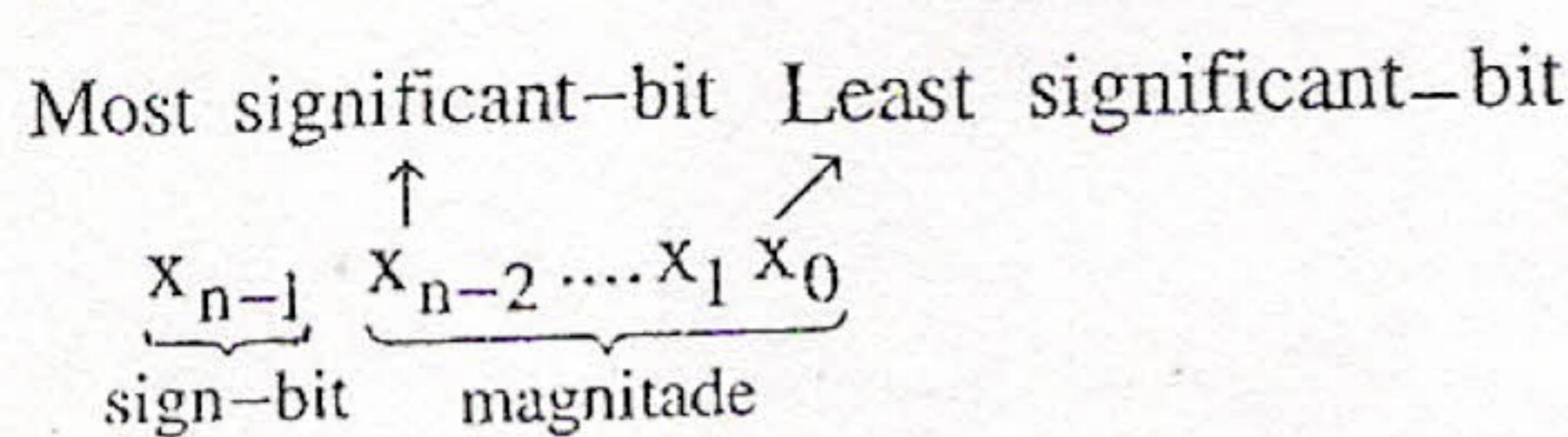
### اعداد با ممیز ثابت: Fixed-point Number

در فرمت اعداد با ممیز ثابت از همان روش ارزش مکانی همانند اعداد دهدی استفاده می‌شود.

$$\left( \begin{matrix} 2^m & 2^{l-1} & 2^0 & 2^{-1} & 2^{-2} & \dots & 2^{-n} \\ b_m & \dots & b_l & b_{l-1} & b_0 & \dots & b_{-n} \end{matrix} \right)_2 = \left( \sum_{i=-n}^m (b_i \times 2^i) \right)_{10} \quad b_i \in \{0,1\}$$

این روش معمولاً برای نمایش اعداد مثبت به کار می‌رود.

حال فرض کنید یک کلمه  $n$  بیتی به نمایش عددی با ممیز ثابت اختصاص داده شود.



که در این صورت سمت چپ‌ترین بیت برای علامت عدد و بقیه بیت‌ها به مقدار آن اختصاص می‌یابد.

دقت:  $(n-1)$ -bit که معادل  $\frac{n-1}{\log_2 n}$  رقم دهدی خواهد بود.

نوع عدد: سخت افزار عاجز از نشان دادن واقعی نقطه ممیز است، بنابراین به هنگام Compile شدن موضع آن را در محلی ثابت فرض می کنند.

اگر نقطه ممیز منتهی علیه سمت راست فرض شود، محتوی نشان دهنده یک عدد صحیح خواهد بود که دامنه تغییرات آن برابر است با:

$$0 \leq N \leq 2^{n-1} \quad (\text{چرا؟})$$

اگر نقطه ممیز بین sign bit و MSB فرض شود، آن گاه محتوی نشان دهنده یک عدد کسری خواهد بود و دامنه تغییرات آن عدد عبارت است از: (چرا؟)

$$0 \leq N \leq 1 - 2^{n-1}$$

**روش های نمایش اعداد منفی:** برای نمایش اعداد منفی سه روش زیر وجود دارد.

**Sign Magnitude (I)**: در این روش کافی است بیت علامت را مساوی ۱ قرار دهیم که در این صورت بیت های Magnitude دارای ارزش مکانی خواهند بود. (مانند عدد مثبت)

مثال: با فرض Reg. 4-bit، نمایش مثبت و منفی عدد ۳ در sign-magnitude به صورت زیر خواهد بود که در آن چه عدد مثبت و چه منفی باشد bit ها دارای ارزش مکانی هستند.

$$\begin{cases} +3 & 0 \ 0 \ 1 \ 1 \\ -3 & 1 \ 0 \ 1 \ 1 \end{cases}$$

### : One's Complement (II)

ابتدا  $(r-1)'$ s Complement را تعریف می کنیم.

تعریف: فرض کنیم عدد مثبت  $N$  در پایه نویسی  $r$  شامل  $n$  رقم صحیح و  $m$  رقم کسری باشد، آن گاه برای نمایش عدد منفی  $-N$  می توان  $(r-1)'$ s Complement استفاده نمود.

فرض کنید که:

$$N \text{ یک عدد مثبت} \left\{ \begin{array}{l} r = \text{پایه عدد نویسی} \\ n = \text{تعداد رقم صحیح} \\ m = \text{تعداد رقم کسری} \end{array} \right.$$

برای نمایش عدد  $-N$  می توان از  $(r-1)'$ s Complement استفاده کرد که به صورت زیر تعریف می شود:

$$-N : (r-1)'\text{s Complement of } N = r^n - r^{-m} - N$$

بنابراین برای نشان دادن اعداد منفی در سیستم های عددی نویسی متفاوت می توان:

$$\text{استفاده نمود} \left\{ \begin{array}{l} \text{Binary} \quad r = 2 \Rightarrow 1's \text{ Complement} \\ \text{Octal} \quad r = 8 \Rightarrow 7's \text{ Complement} \\ \text{Decimal} \quad r = 10 \Rightarrow 9's \text{ Complement} \end{array} \right.$$

به منظور به دست آوردن الگوریتم 'Complement' (r-1)' به مثال زیر توجه نمایید.

مثال: فرض کنید که:

$$N = 25.742 \begin{cases} r = 10 \\ n = 2 \\ m = 3 \end{cases}$$

-35.742 : 9's Complement of N =  $10^2 - 10^3 - 25.742$

$$\begin{aligned} &= (100 - 0.001) - 25.742 && (r-1) && (r-1) && (r-1) && (r-1) && (r-1) \\ &= 99.999 - 25.742 && 9 && 9 && 9 && 9 && 9 \\ &= 74.257 && -2 && 5 && 7 && 4 && 2 \\ &&&\hline && 7 && 4 && 2 && 5 && 7 \end{aligned}$$

الگوریتم: برای به دست آوردن (r-1)'s Complement کافی است که هر رقم را از (r-1) تفریق کنیم.

بنابراین برای پیدا کردن 1's Complement اعداد باینری کافی است بیت‌های عدد را یک به یک مکمل کنیم. (یعنی صفرها را به یک و یکها را به صفر تبدیل کنیم)

توجه: در صورتی که اعداد منفی در 1's Complement نشان داده شوند، آن‌گاه bit‌ها دارای ارزش مکانی نخواهد بود، مگر دوباره از آن 1's Complement بگیریم.

### : 2's Complement (III)

ابتدا r's Complement را تعریف می‌کنیم.

تعریف: فرض کنیم عدد مثبت N در پایه عدد نویسی r شامل n رقم صحیح باشد، آن‌گاه عدد (-N) را می‌توان در نشان داد که به صورت زیر تعریف می‌شود:

فرض کنید:

$$N \text{ عددی مثبت} \begin{cases} r = \text{پایه عدد نویسی} \\ n = \text{تعداد ارقام صحیح} \end{cases}$$

آن‌گاه  $-N : r's \text{ Complement of } N = r^n - N$

برای نمایش عدد منفی در پایه عدد نویسی متفاوت می‌توان:

$$\begin{cases} \text{Binary} & r=2 \quad 2's \text{ Complement} \\ \text{Octal} & r=8 \quad 8's \text{ Complement} \\ \text{Decimal} & r=10 \quad 10's \text{ Complement} \end{cases} \text{ استفاده نمود.}$$

مثال: فرض کنید

$$N = 25.840 \begin{cases} r = 10 \\ n = 2 \end{cases}$$

$-N = -25.840 : 10's \text{ Complement of } N = 10^2 - 25.840$

$$\begin{aligned} &= 100 - 25.840 && (r-1) && (r-1) && (r-1) && (r-1) && r \\ &= 74.160 && 1 && 0 && 0 && 0 && 0 \\ &&&- && 2 && 5 && 8 && 4 && 0 \\ &&&\hline && 7 && 4 && 1 && 6 && 0 \end{aligned}$$

**الگوریتم:** برای به دست آوردن  $r's$  Complement، کافی است صفرهای مقدم بدون تغییر اولین رقم غیرصفر از  $r$  و بقیه از  $(r-1)$  تفیریق شوند.

**توجه:** در صورتی که اعداد منفی در  $2's$  Complement نشان داده شوند، آن گاه bit ها دارای ارزش مکانی نخواهند بود، مگر دوباره از آن  $2's$  Complement بگیریم.

**مثال:** با فرض  $N = .0110$

$$N = .0110 \Rightarrow \begin{cases} r = 2 \\ n = 0 \end{cases}$$

$$-N = -.0110 : 2's \text{ Complement of } N = 2^0 - 0.0110$$

$$\begin{array}{r} 0.1120 \\ 1.0000 - \\ \hline .0110 \\ \hline .1010 \end{array} \quad \begin{aligned} &= 1 - 0.0110 \\ &= 0.1010 \end{aligned}$$

الگوریتم به دست آوردن  $2's$  Complement: صفرهای مقدم و اولین یک بدون تغییر و بقیه را بیت به بیت مکمل می‌کنیم.  
روش دیگر برای به دست آوردن  $2's$  Complement از روی  $1's$  Complement :

$r's$  Complement of  $N = (r-1)'s$  Complement of  $N + r^{-m}$

$$2's \text{ Complement of } .0110 = 1's \text{ Complement of } .0110 + 2^4 \\ = 0.1001 + 0.0001 = 0.1010$$

در این روش کافی است ابتدا  $1's$  Complement را به دست آورده و سپس به bit سمت راست آن ۱ اضافه کنیم.

**مثال:** برای عدد باینری  $N = .0110$  داریم:

$$N = .0110 \Rightarrow \begin{cases} r = 2 \\ n = 0 \\ m = 4 \end{cases}$$

$$-N = -.0110 : 2^0 - 2^{-4} - 0.0110 = 1 - 0.0001 - 0.0110 \\ = 0.1111 - 0.0110 \\ = 0.1001$$

### مقایسه $: 2's \text{ Complement} \text{ با } 1's \text{ Complement}$

(۱) به دست آوردن  $1's$  Complement نسبت به  $2's$  ساده‌تر است.

(۲) جمع و تفیریق در  $C$   $2's$  نسبت به  $C$   $1's$  sign Mag ساده‌تر است.

(۳) در  $C$   $1's$  هم صفر منفی و هم صفر مثبت وجود دارد ولی در  $C$   $2's$  فقط یک صفر وجود دارد. بنابراین با استفاده از  $C$   $1's$  sign Mag ممکن است دستورالعمل‌هایی که محتوای ثباتی را به منظور صفر بودن تست می‌کنند، مواجه با مشکل شوند. ولی در  $C$   $2's$  مواجه با مشکل نمی‌شوند.

(۴)  $C$   $1's$   $C$   $2's$  بیشتر برای عملیات منطقی و  $C$   $2's$  بیشتر برای عملیات حسابی کار می‌رود.

(۵) دامنه اعداد قابل نمایش در  $C$   $2's$  یک عدد بیشتر از  $C$   $1's$  می‌باشد.

## محاسبات دودویی:

Binary Arithmetic

$$(\pm A) - (+B) = (\pm A) + (-B)$$

$$(\pm A) - (-B) = (\pm A) + (+B)$$

در نتیجه:

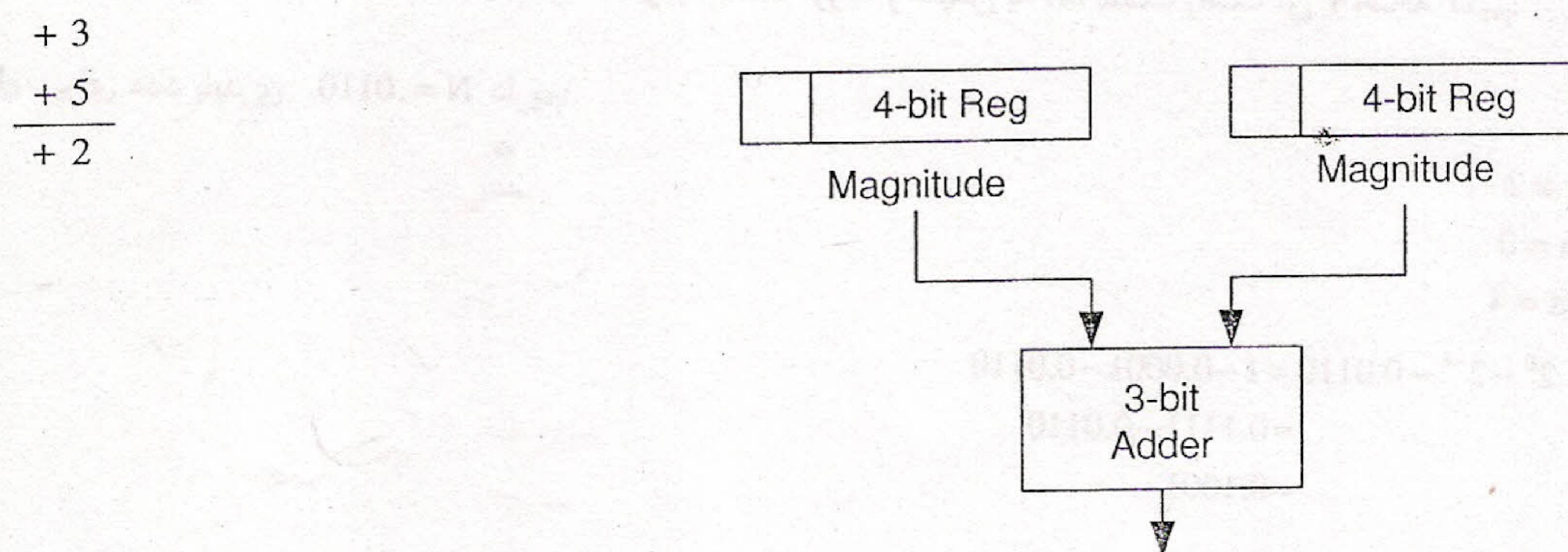
$$A - B = A + (-B) = A + 2's\ B = A + \underbrace{(1's\ B + 1)}_{\downarrow}$$

می‌توان با استفاده از (X-OR) Gate تولید نمود.

با توجه به دو عبارت فوق متوجه می‌شویم که با استفاده از نمایش 2's Complement برای نمایش اعداد منفی، می‌توان تفریق را به کمک جمع انجام داد.

## ۱. جمع در Sign Magnitude

در آنچه عمل جمع در Sign Magnitude بیتهای علامت را در عمل جمع شرکت نمی‌دهند. در صورتی که دو عدد متحددالعلامه باشند، قدرمطلق آنها با هم جمع و علامت مشترک را به عنوان علامت نتیجه قرار می‌دهند و در صورتی که دو عدد مختلفالعلامه باشند قدر مطلق عدد کوچکتر را از عدد بزرگتر تفریق و سپس علامت نتیجه را موافق علامت عددی قرار می‌دهند که از نظر قدرمطلق بزرگتر باشد.



## ۲. جمع در 2's Complement

دو عدد را به اضافه بیت علامت با هم جمع می‌کنیم و از رقم نقلی حاصل در موضع Sign Bit صرفنظر می‌کنیم.

مثال: با فرض داشتن 7 Bit Register 7 عملیات زیر انجام دهید.

$$\begin{array}{r}
 & \text{7-bit Reg}(2's) \\
 +6 & \underline{0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 0} \\
 +9 & +0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 1 \\
 \hline
 +15 & \underline{0 \ 0 \ 0 \ 1 \ 1 \ 1 \ 1}
 \end{array}
 = +(2^0 + 2^1 + 2^2 + 2^3) = +(1 + 2 + 4 + 8) = +15$$

$$\begin{array}{r}
 \text{7-bit Reg(2's)} \\
 -6 \quad 1 \ 1 \ 1 \ 1 \ 0 \ 1 \ 0 \\
 -9 \quad +1 \ 1 \ 1 \ 0 \ 1 \ 1 \ 1 \\
 \hline
 -15 \quad \swarrow 1 \ 1 \ 1 \ 0 \ 0 \ 0 \ 1 = -(0001111) = -15
 \end{array}$$

①  
End carry

از End Carry صرفنظر می‌کنیم.

تمرین: چرا در جمع دو عدد در 2's Complement از End Carry صرفنظر می‌کنیم. (راهنمایی: از تعریف r's Complement استفاده کنید)

### ۳. جمع در 1's Complement

در این روش دو عدد را به انصمام Sign Bit با هم جمع می‌کنیم و در صورت وجود رقم نقلی در موضع Sign Bit آن با حاصل جمع مرحله اول دوباره جمع می‌کنیم و در صورت وجود رقم نقلی در مرحله دوم، از آن صرفنظر می‌کنیم.

مثال: با فرض 7 Bit Register 7 عملیات زیر را انجام دهید.

$$\begin{array}{r}
 \text{7-bit Reg(1's)} \\
 -6 \quad 1 \ 1 \ 1 \ 1 \ 0 \ 0 \ 1 \\
 +9 \quad +1 \ 1 \ 0 \ 0 \ 1 \ 1 \ 1 \\
 \hline
 +3 \quad 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0
 \end{array}$$

E.A.C.  $\downarrow$

①  $+ \frac{1}{0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 1} = +(3)$

به رقم نقلی تولید شده از جمع مرحله اول (رقم نقلی دورگشته) گویند که آن را دوباره با نتیجه حاصل از مرحله اول جمع می‌کنیم.

نتیجه: جمع در 2's مشکل ناهمانگی مدار جمع کننده Sign Magnitude را از بین می‌برد. همچنین جمع در 2's نسبت به 1's سریع‌تر انجام می‌گیرد.

### Decimal Code

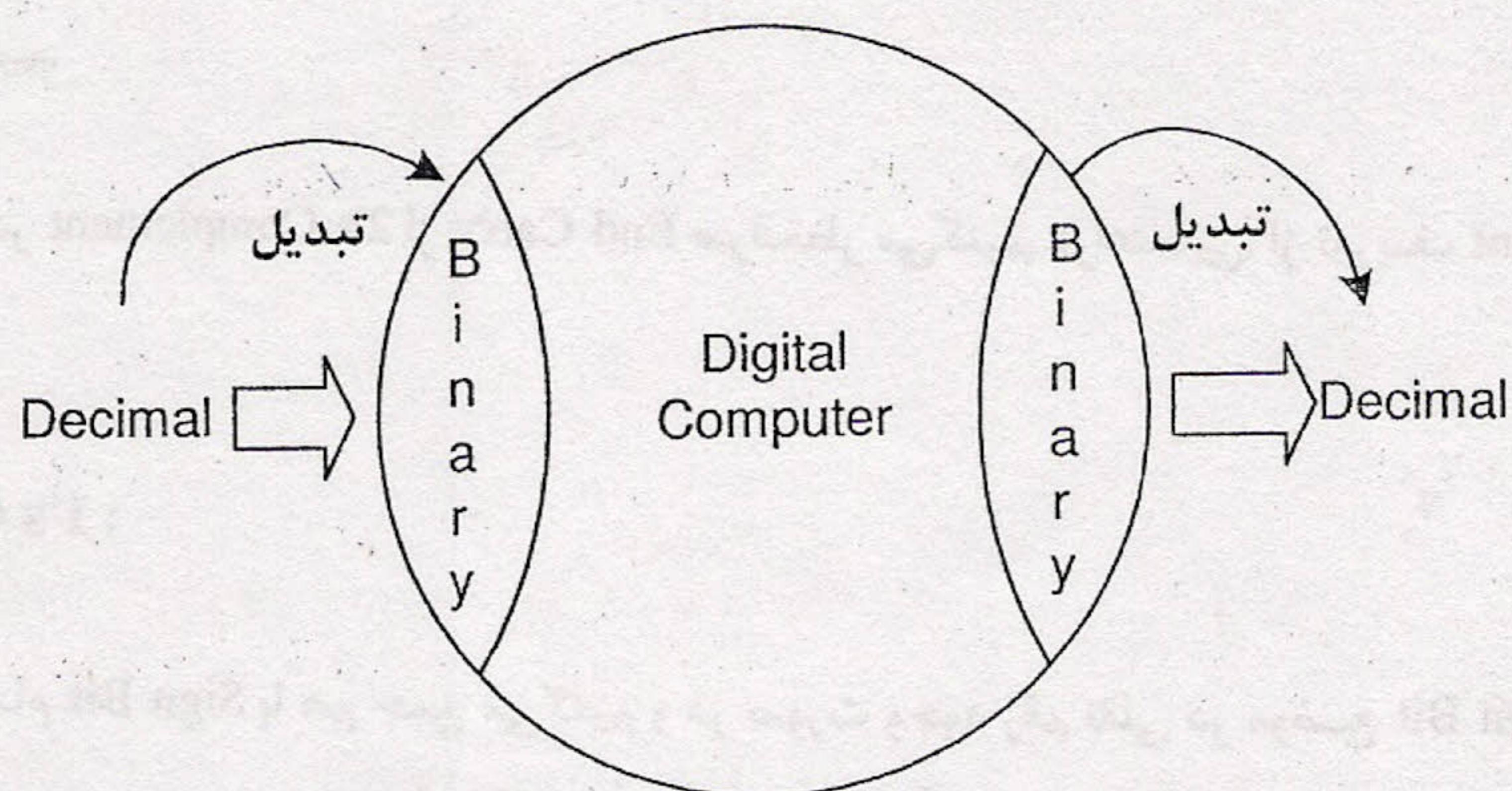
سیستم‌های رقمی با سیگنال‌های محدود به دو مقدار ممکن و عناصری با دو حالت پایدار کار می‌کنند، بنابراین یک تناظر یک به یک بین سیگنال‌های دودویی و ارقام دودویی وجود دارد.

قبل از مطالعه Decimal Code باید بین معادل Binary Coding یک عدد و Binary Coding آن تفاوت قابل شد.

$(16)_{10} = (10000)_2$  معادل باینری (بیت‌ها دارای ارزش مکانی هستند)

$(16)_{10} = (0010110)_2$  کد باینری (بیت‌ها دارای ارزش مکانی نیستند)

می‌دانیم ورودی به کامپیوتر به صورت Decimal و خروجی از آن نیز Decimal می‌باشد، ولی سخت‌افزار در سیستم اعداد Binary کار می‌کند. بنابراین اطلاعات عددی به هنگام ورودی به کامپیوتر باید از Decimal به Binary تبدیل و سپس محاسبات توسط سخت‌افزار انجام و به هنگام خروجی نتایج دوباره از Binary به Decimal تبدیل گردد. مدت زمان لازم برای تبدیل به هنگام ورود و به هنگام خروج را Overhead گویند که مهم‌ترین مزیت Decimal Codeها کاهش دادن Overhead می‌باشد.



### کدهای چهاربیتی:

برای کد گذاری ارقام دهدۀ ۰, ۱, ۲, ۳, ..., ۹ حداقل به ۴ بیت نیاز داریم که انتخاب ۱۰ ترکیب از بین ۱۶ ترکیب به طرق مختلف انجام می‌گیرد و در نتیجه کدهای متفاوتی ایجاد می‌شود.

DD	NBCD 8421	EXCESS-3	AIKEN 2421	GRAY CODE	NBCD Even Odd	Excess -3 Gray Code	2 Out of 5 74210	Walking Code Johnson
0	0000	0011	0000	0000	0 1	0010	11000	00000
1	0001	0100	0001	0001	1 0	0110	00011	00001
2	0010	0101	0010	0011	1 0	0111	00101	00011
3	0011	0110	0011	0010	0 1	0101	00110	00111
4	0100	0111	0100	0110	1 0	0100	01001	01111
5	0101	1000	1011	0111	0 1	1100	01010	11111
6	0110	1001	1100	0101	0 1	1101	01100	11110
7	0111	1010	1101	0100	1 0	1111	10001	11100
8	1000	1011	1110	1100	1 0	1110	10010	11000
9	1001	1100	1111	1101	0 1	1010	10100	10000

### NBCD: Natural Binary Coded Decimal

NBCD یک کد وزن دار است و متداول‌ترین روش برای نشان دادن ارقام دهدۀ است. در این روش هر رقم به صورت:

$$x_i = b_{i,3} \quad b_{i,2} \quad b_{i,1} \quad b_{i,0}$$

و در آن هر بیت  $b_{i,j}$  دارای ارزش مکانی زیر است:

$$10^i \times 2^j$$

دو عیب عمدۀ NBCD عبارتند از:

- در جمع دو رقم در NBCD ممکن است رقم نقلی لازم تولید نشود که در این صورت به منظور تصحیح، 6 واحد باید به آن اضافه شود.

مثال:

$$\begin{array}{r}
 \text{NBCD} \\
 +5 \quad 0101 \\
 +9 \quad 1001 \\
 \hline
 +14 \quad 1110 \quad \text{Binary Addition} \\
 \downarrow \quad +0110 \quad \text{Correction} \\
 1 \quad \underline{0100} \quad 4
 \end{array}$$

(۲) هر رقم NBCD را نمی‌توان از تبدیل صفرها به یک و یک‌ها به صفر به دست آورد، بنابراین NBCD یک کد خود متمم (Self - Complementary) نیست.

$$\begin{array}{c}
 5 = 0101 \\
 9's \quad \swarrow \quad \searrow 1's \\
 4 \neq 1010
 \end{array}$$

(۳) کد بدون وزن ولی دارای خاصیت خود متمم (Self - Complementary) می‌باشد، یعنی 9's Complement (Excess-3 code) آن را می‌توان از تبدیل ۰ ها به ۱ و ۱ ها به ۰ به دست آورد. همچنین در جمع دو رقم، رقم نقلی لازم تولید می‌شود، هر چند که نیاز به تصحیح دارد.

$$\begin{array}{r}
 +5 \quad 1000 \\
 +9 \quad 1100 \\
 \hline
 +14 \quad 10100 \\
 \downarrow \quad +0011 \\
 10111 \quad \text{Self Complementary}
 \end{array}
 \quad
 \begin{array}{c}
 5 \\
 9's \quad \swarrow \quad \searrow 1's C \\
 4 = \quad 0111
 \end{array}$$

Aiken - Code (۴)

Aiken خاصیت خود متممی دارد.

و برای پیدا کردن محور تقارن کافی است در Aiken - Code مجموع وزن‌ها را به دست آورد و آن را به صورت حاصل جمع دو عدد متوالی نوشت.

$$\text{NBCD}: \sum(W_i) = 8 + 4 + 2 + 1 = 15 = 7 + 8$$

$$\text{Aiken}: \sum(W_i) = 2 + 1 + 2 + 1 = 6 = 4 + 2$$

چون در Aiken محور تقارن درست و سیستم کدگذاری یعنی بین code دو عدد متوالی ۵, ۶, ۷ قرار می‌گیرد به این دلیل یک خود متمم است. ولی در NBCD و سیستم کدگذاری نیست، زیرا بین ۸, ۹ قرار می‌گیرد. به این دلیل NBCD یک کد خود متمم نیست.

Gray - Code (۵)

یکی دیگر از کدهای بدون وزن است که در آن هر دو کد مجاور تنها در یک بیت تفاوت دارند. بنابراین از Gray - Code می‌توان:

- (a) برای کدگذاری سطر و ستون در جدول کارنو استفاده نمود.
- (b) در طراحی شمارندها اگر به حالت‌های شماره Gary - Code اختصاص دهیم، عمل شمارش تنها با تغییر در محتوی یک FF انجام می‌گیرد.
- (c) در صورت انجام محاسبات روی کمیت‌های پیوسته به وسیله Digital Computer، می‌توان اطلاعات ورودی پیوسته را به گستته تبدیل نمود.

برای تبدیل یک عدد Binary به معادل Gray - Code آن می‌توان از رابطه زیر استفاده نمود:

$$(b_n b_{n-1} \dots b_1)_2 = (?)_{\text{Gray-Code}} = (g_n g_{n-1} \dots g_1)_{\text{Gray}}$$

$$1. \quad g_n = b_n \quad \text{for } k = n$$

$$2. \quad g_k = (b_k + b_{k+1}) \text{MOD} 2 \quad \text{for } k = 1, 2, \dots, n-1$$

$$3. \quad g_k = b_k \oplus b_{k+1} \text{ Exclusive Or}$$

مثال: فرض کنید داشته باشیم:

$$(11001010110)_2 = (?)_{\text{Gray}} = (g_n g_{n-1} \dots g_1)_{\text{Gray}}$$

$$g_n = b_{11} = 1$$

$$(11001010110)_2 = (10101111101)_{\text{Gray}}$$

ب. برای تبدیل یک عدد در Gray-Code به معادل باینری آن می‌توان از رابطه زیر استفاده نمود:

$$(g_n \dots g_1)_{\text{Gray}} = (b_n \dots b_1)_2$$

$$\begin{cases} b_n = g_n & \text{for } k = n \\ b_i = \sum_{i=k}^1 g_i \text{MOD} 2 & \text{for } k = (n-1), \dots, 1 \end{cases}$$

مثال:

$$(10101111101)_{\text{Gray}} = (?)_2 = (11001010110)_2$$

$$b_{11} = g_{11} = 1$$

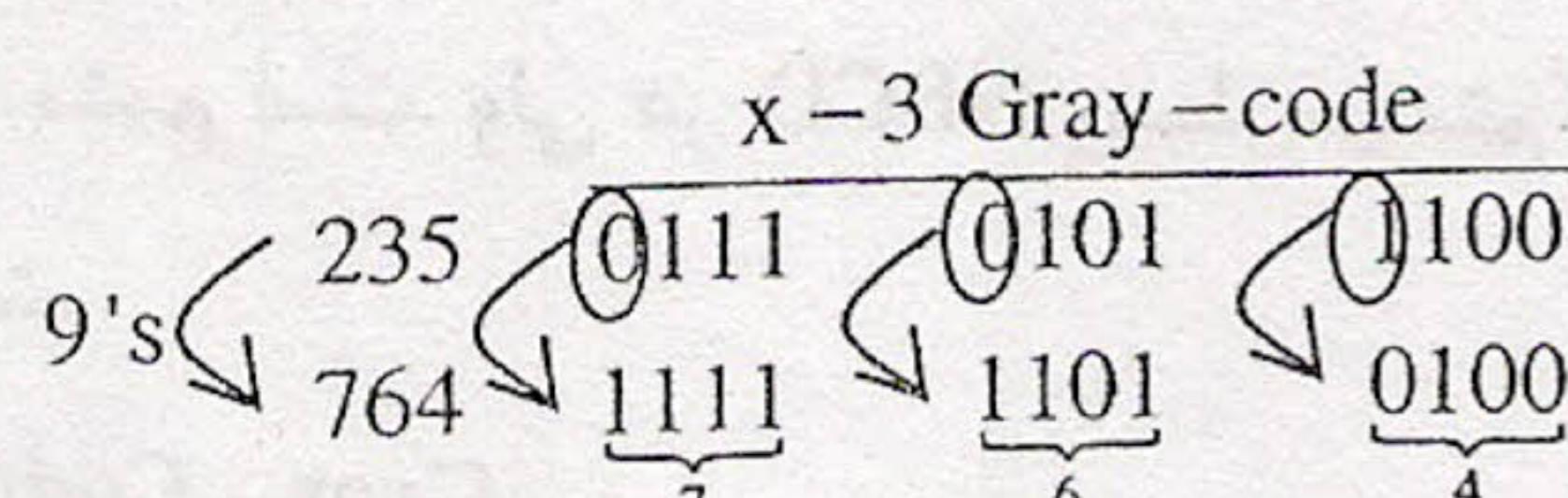
$$b_{10} = (g_{10} + g_{11}) \text{MOD} 2 = (1+0) \text{MOD} 2 = 1$$

$$b_9 = (g_9 + g_{10} + g_{11}) \text{MOD} 2 = (1+0+1) \text{MOD} 2 = 0$$

تنها عیب Gray-code رفتن از رقم 9 به رقم صفر می‌باشد که نیاز به تغییر در 3 بیت دارد و برای از بین بردن این اشکال از "Excess-3 Gray-Code" استفاده نمود.

#### Excess - 3Gray - Code (۶)

علاوه بر رفع مشکل تبدیل 9 به 0 در Gray-Code در صورت استفاده از Excess-3 Gray - code برای پیدا کردن یک عدد می‌توان با متمم‌گیری از سمت چپ‌ترین بیت در گروه‌های 4 بیتی استفاده نمود.



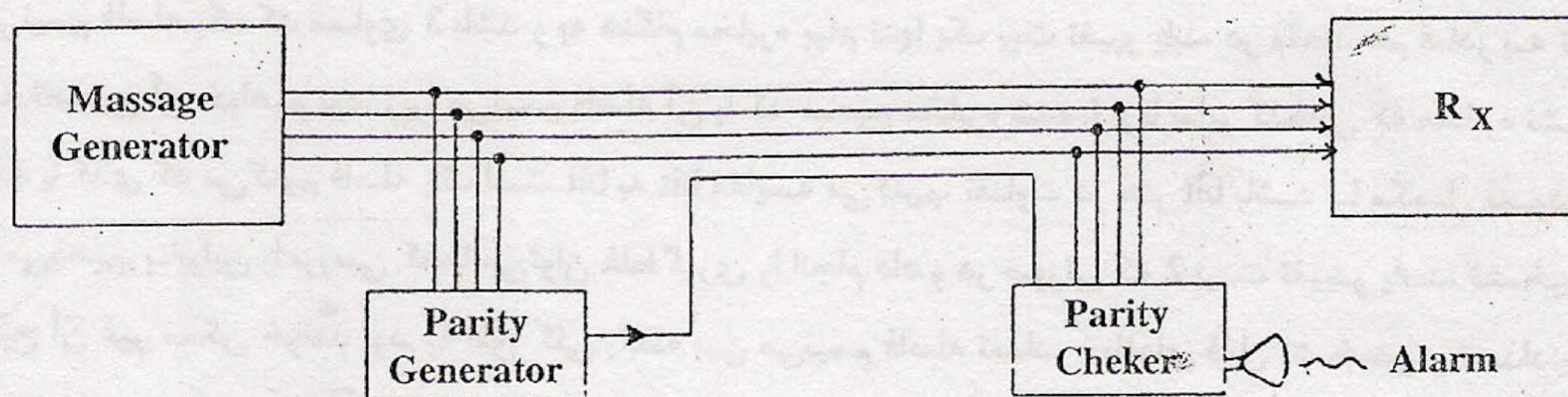
### کدهای با بیش از ۴ بیت:

کدهای 4 بیتی مانند NBCD,...، کد خطایاب نیستند، یعنی در صورت مخابره پیام از یک مبدأ به یک مقصد در صورتی که به دلیل وجود پارازیت بین راه یک bit تغییر یابد در مقصد قادر به تشخیص خطایاب نخواهیم بود. بنابراین به منظور تشخیص خطایاب در یک bit باید از 5-bit Code استفاده کنیم. به عنوان مثال اگر کد رقم ۰۰۱۰ دهدی NBCD، یعنی ۰۰۱۰ را از مبدأ مخابره و بین راه بیت دوم آن تغییر یابد و در مقصد پیام ۰۰۰۰ را دریافت کنیم قادر به تشخیص خطایاب نخواهیم بود، زیرا ۰۰۰۰ یک کد مجاز برای رقم دهدی صفر در NBCD می‌باشد.

چند نمونه از 5-bit Code را در زیر مطالعه می‌کنیم.

### ۱) استفاده از بیت توازن (Parity Bit):

یکی از روش‌های افزایش کدهای 4 بیتی به 5 بیتی است به طوری که با اضافه نمودن Bit – Parity به عنوان بیت پنجم می‌توان تعداد یک‌های موجود در Code را فرد یا زوج نمود که آن‌ها را به ترتیب Even Parity و Odd Parity گویند. مثلاً در صورتی که در کدگذاری از Even Parity استفاده شود و به هنگام مخابره پیام یک بیت تغییر کند پیام دریافت شده در مقصد Odd Parity خواهد بود و بنابراین قادر خواهیم بود وجود خطای تک بیتی را تشخیص دهیم، ولی نمی‌توان آن را تصحیح نمود مدار استفاده از "Parity Bit" به شکل زیر است:



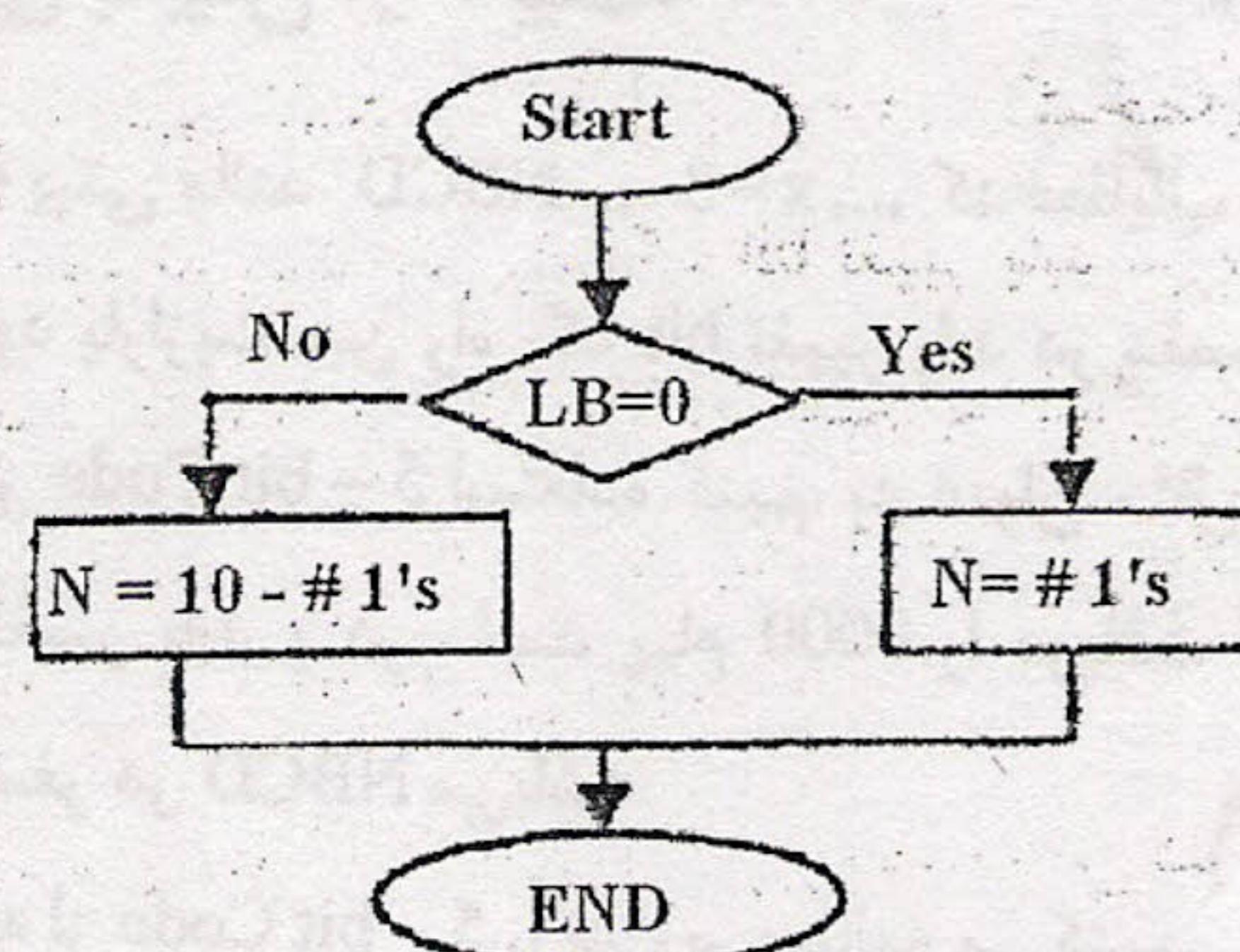
تمرین: مطلوب است طراحی Even Parity Checker , Even Parity Generator (راهنمایی: با استفاده از X-OR).

### 2- Out Of -5 Code :

یک کد خطایاب است که در آن برای نشان دادن هر رقم دهدی از 5 بیت استفاده می‌شود که در هر یک از این کدها 2 بیت یک و بقیه صفر می‌باشند.

این کد در واقع یک کد نیمه وزن دار می‌باشد (Semi Weighted) که به جز کد اختصاص یافته به رقم صفر دهدی کدهای اختصاص یافته به سایر ارقام دهدی به ترتیب دارای وزن ۷,۴,۲,۱,۰ هستند. یعنی این وزن‌ها تنها در مورد Code رقم صفر صادق نیست، ولی در مورد Code بقیه ارقام صادق می‌باشد.

## Johnson Code (۳)



کد پنج بیتی است که در آن ابتدا قدم روی هر bit قرار گیرد.  
از صفر به یک تبدیل می‌گردد و سپس قدم روی هر bit قرار گیرد  
از یک به صفر تبدیل می‌گردد.  
مزیت Johnson Code در این است که رمزگشایی آن به سادگی  
انجام می‌شود و با استفاده از فلوچارت زیر می‌توان رقم را تشخیص داد.

## کدهایی با بیش از ۵ بیت:

تا به حال کدهایی را مورد مطالعه قرار دادیم که تنها قادر به تشخیص خطاهای تک بیتی بودند. برای این که بتوان وجود خطای تک بیتی را هم تشخیص داد و هم تصحیح نمود به کدهایی با بیش از ۵ بیت نیاز داریم.

تعريف Minimum Distance : حداقل فاصله عبارت است از حداقل تعداد بیت‌هایی که باید در یک کد تغییر یابد تا کد مجازی دیگری از همان سیستم کد گذاری به دست آید. مثلاً برای کدهای NBCD، X-3، Gray – Code و Aiken می‌نیم فاصله برابر 1 ولی برای 2 Out Of 5 می‌نیم فاصله مساوی 2 می‌باشد.

در صورتی که می‌نیم فاصله یک کد مساوی 3 باشد و به هنگام مخابره پیام تنها یک بیت تغییر یابد، در مقصد هم قادر به تشخیص خطأ و هم قادر به تصحیح آن خواهیم بود، زیرا می‌نیم فاصله آن با کد صحیح مخابره شده 1 و با سایر کدهایی که مخابره نشده‌اند 2 می‌باشد، در نتیجه با کدی که می‌کنیم فاصله یک است bit به bit مقایسه می‌کنیم، تفاوت در هر bit باشد با مکمل نمودن آن bit تصحیح را انجام می‌دهیم. بنابراین با بررسی کدها می‌توان غلط‌گیری را انجام داد و در صورتی که 2 بیت تغییر یابد، تشخیص خطأ ممکن ولی تصحیح آن غیر ممکن خواهد بود. به طور کلی رابطه بین می‌نیم فاصله تعداد خطاهای قابل تشخیص و تعداد خطاهای قابل تصحیح را می‌توان به صورت زیر بیان نمود:

$$m = c + d + 1 ; \quad d \geq c$$

که در آن  $m$ : می‌نیم فاصله  $d$ : تعداد خطاهای قابل تشخیص و  $c$ : تعداد خطاهای قابل تصحیح می‌باشد.

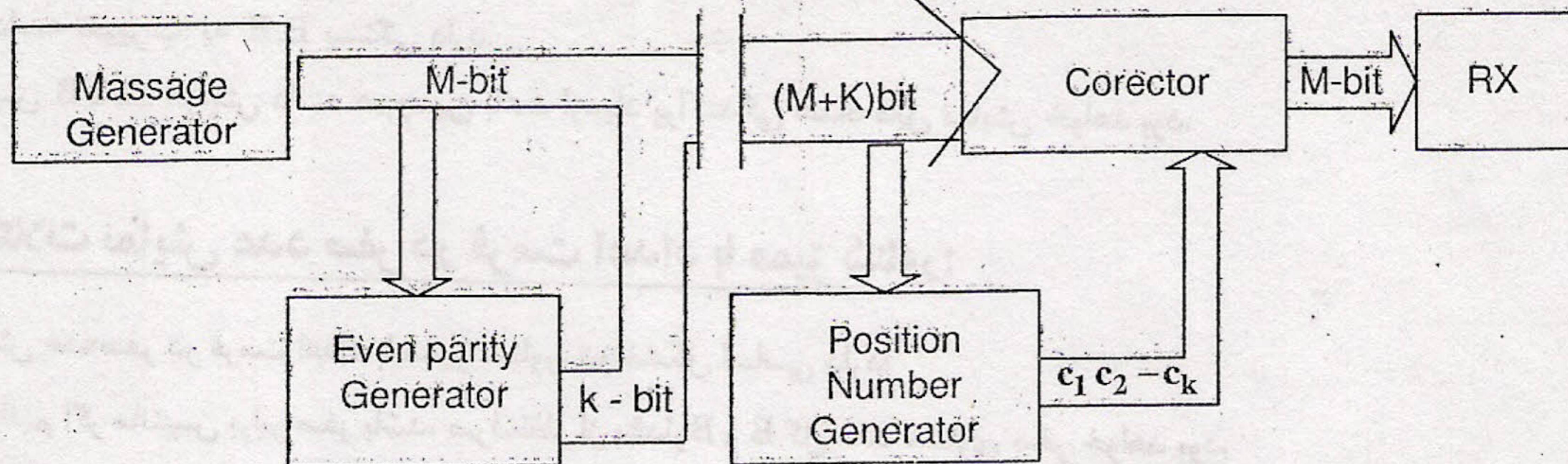
جدول زیر با توجه به  $m$  تعداد خطاهای قابل تشخیص و قابل تصحیح را نشان می‌دهد.

$m$	$d$	$c$
1	0	0
2	1	0
3	2	0
3	1	1
4	3	0
4	2	1
5	4	0
5	3	1
5	2	2

در این روش تا جایی پیش می‌رویم که بتوانیم تنها وجود خطأ در bit-1 را هم تشخیص دهیم و هم تصحیح کنیم. برای این منظور کد همینگ را مطالعه می‌کنیم.

### کد همینگ: Hamming - Code

در این کد می‌نیم فاصله برابر 3 است و می‌توان وجود خطای تک بیتی را هم تشخیص داد و همچنین آن را تصویح نمود که مدار آن به صورت زیر می‌باشد.



اگر به کد  $m$  بیتی،  $k$  بیت پاریتی (Parity) اضافه کفیم، کدهای  $m+k$  بیتی تولید می‌شوند که در آن موضع بیت‌ها را از چپ به راست با  $(m+k)....3,2,1$  شماره‌گذاری می‌کنیم. بیت پاریتی روی بیت‌های پیام کنترل انجام می‌دهند و عدد دودویی  $(c_1 c_2, \dots, c_n)_2$  تولید می‌کند که معادل دهدۀ آن محل خطا را مشخص می‌سازد و با متهم‌گیری از آن بیت می‌توان خطا را تصویح نمود. اگر عدد دهدۀ حاصل صفر باشد به مفهوم این خواهد بود که در پیام مخابره شده خطای رخ نداده است.

چون تعداد کدهای نامطلوب  $(m+k)$  و تعداد کدهای مطلوب 1 می‌باشد، بنابراین این تعداد بیت‌های توازن را می‌توان از رابطه زیر به دست آورد.

$$2^k \geq (m+k)+1$$

**مثال:** فرض کنید پیام مخابره شده یک رقم دهدۀ در NCD باشد، آن گاه داریم:

$$m=4 \Rightarrow 2^k \geq 4+k+1 \Rightarrow k=3$$

برای این که بیت‌های توازن بتوانند روی بیت‌های پیام کنترل انجام دهنند، باید آن‌ها را در مواضعی متناظر با وزن‌های  $2^i$  قرار داد. (یعنی در مکان‌های 1 و 2 و 4) بنابراین کلیه بیت‌هایی که موضع آن‌ها متناظر با توان صحیحی از 2 باشد. مربوط به Parity – Bit و بقیه مربوط به بیت‌های پیام خواهد بود.

Bit Position	1	2	3	4	5	6	7
Bit Name	$p_1$	$p_2$	$b_3$	$p_4$	$b_5$	$b_6$	$b_7$

به عنوان مثال طراحان کامپیوتر IBM پایه را  $B=16$  انتخاب می‌کنند.

### مشخصات اعداد با ممیز شناور

۱) نوع عدد : Mixed

۲) دقت: به تعداد بیت‌های اختصاص یافته به  $M$  بستگی دارد.

۳) دامنه تغییرات: به  $B, E$  بستگی دارد.

افزایش  $B$  باعث افزایش دامنه همچنین باعث ازدیاد پراکندگی اعداد قابل نمایش خواهد بود.

### مشکلات نمایش عدد صفر در فرمت اعداد با ممیز شناور:

نمایش عدد صفر در فرمت اعداد با ممیز شناور، دو مشکل اساسی دارد:

می‌دانیم اگر مانتیس برابر صفر باشد، صرفنظر از مقدار  $B$  و  $E$  کل عدد مساوی صفر خواهد بود.

$$M \times B^E$$

$$\text{if } M=0 \text{ then } 0 \times B^E = 0$$

۱) به دلیل Round of Error گاهی مانتیس نتیجه منجر به عدد غیر صفر ولی فوق العاده کوچک می‌شود، مثلاً محاسبه زیر را انسان انجام دهد، نتیجه صفر ولی Fixed – point ALU انجام دهد، نتیجه غیر صفر تولید خواهد شد.

$$1 - 3 \times \frac{1}{3} = 1 - 1 = 0 \quad (\text{انسان})$$

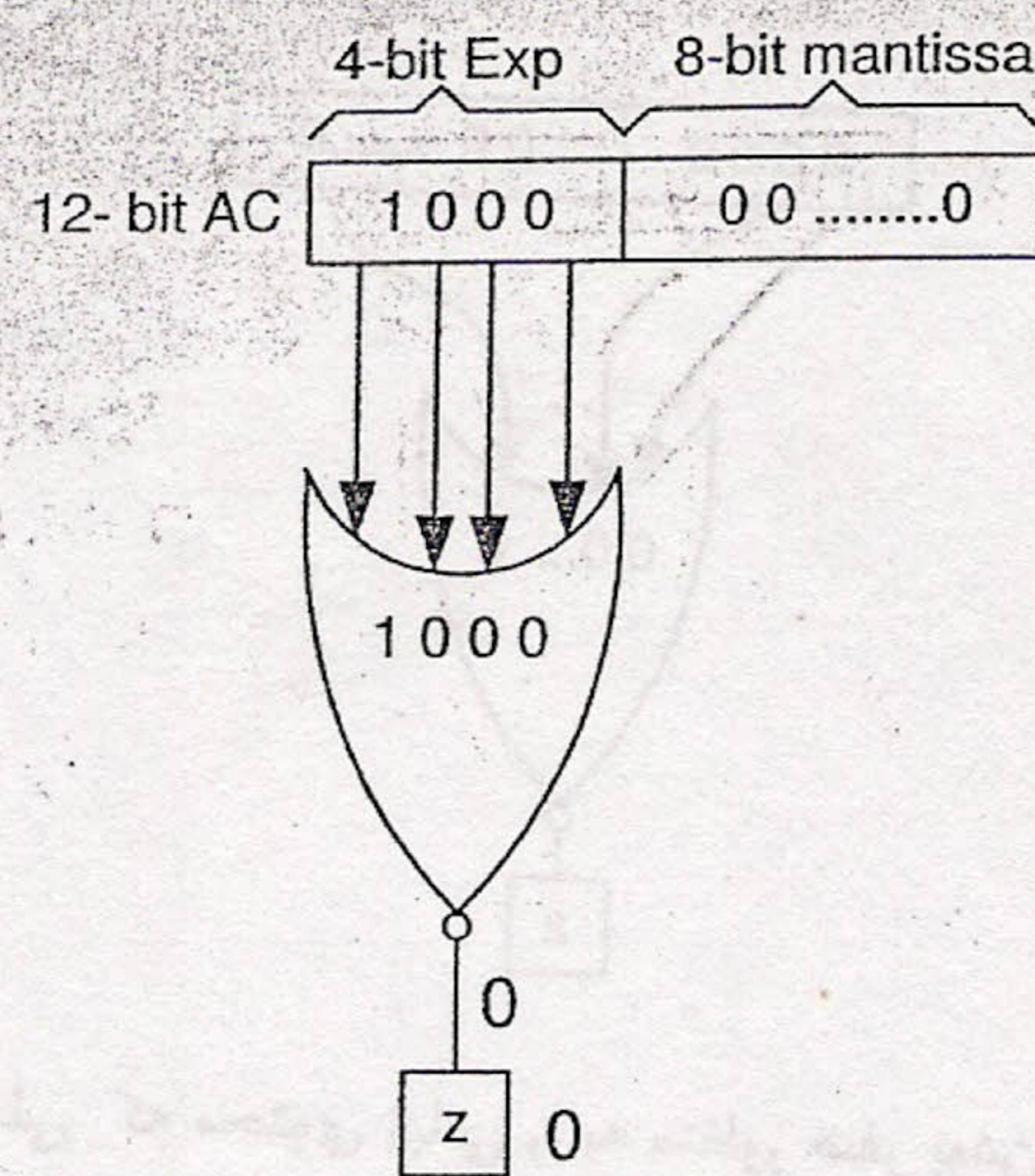
$$1 - 3 \times \frac{1}{3} = 1 - 3 \times (0.33\dots 3) = 1 - 0.99\dots 9 = 0.000\dots 01 \quad (\text{کامپیوتر})$$

$$M \times B^E = \frac{M}{B^E} \xrightarrow{\substack{\text{اختلاف کوچکترین عدد منفی} \\ \text{عددی فوق العاده بزرگ}}} \lim_{\substack{\longrightarrow \\ \text{عددی فوق العاده بزرگ}}} 0$$

برای رفع این مشکل باید به نمای عدد صفر کوچکترین عدد منفی را اختصاص دهیم. در صورت اختصاص  $k$ -bit به ناحیه نمای چون کوچکترین عدد منفی قابل نمایش در  $2^k S$  Complement مساوی  $(2^{k-1})$  می‌باشد. بنابراین این عدد را برای همیشه به نمای عدد صفر در فرمت اعداد با ممیز شناور اختصاص می‌دهیم.

۲) اختصاص دادن کوچکترین عدد منفی به نمای عدد صفر، مشکل اول را برطرف می‌کند ولی مشکل دوم را ایجاد می‌کند، زیرا دستورالعمل‌هایی انشعاب شرطی که محتوای ثباتی را به منظور صفر بودن تست می‌کنند، مواجه با مشکل می‌شوند.

مثلاً با فرض Exponent-4 کوچکترین عدد منفی در  $2^k S$  Complement عدد 8 و معادل Binary آن 1000 و در نتیجه Z-Flag صفر خواهد شد و عمل انشعاب انجام نخواهد گرفت.



که برای رفع این مشکل باید از نمای اریب دار (Biased-Exponent) استفاده شود. مثلاً اگر  $k$ -bit به ناحیه نما اختصاص یافته باشد، چون کوچکترین عدد منفی مساوی با  $-2^{k-1}$  خواهد بود و نمایش آن به صورت  $\underbrace{1\ldots0}_{k-1}0000$  خواهد بود و در این صورت

دستورالعمل انشعابی مانند  $X = \text{IF } AC=0 \text{ Then Go To X}$  صحیح انجام نخواهد گرفت. روی این اصل باید از  $(\text{Excess-}2^{k-1}) \text{ code}$  برای نمایش  $E$  استفاده نمود که آن را «نمای اریب دار» گویند و مقدار  $2^{k-1}$  را اریب (Biase) گویند. مثلاً در صورتی که به ناحیه  $E$  تنها 4-bit اختصاص دهیم. برای از بین بردن خطای گرد کردن (R.O.E.) باید کوچکترین عدد منفی قابل نمایش در  $2^8$ 's Compl که مقدار دهدهی آن 8- و نمایش Binary آن 1000 می‌باشد، برای همیشه به نمای عدد صفر اختصاص دهیم.

4-bit exp:

$$\begin{array}{r} 4-1 \\ -2, \dots, 0, \dots, +(\underline{2-1}) \\ -8, \dots, 0, \dots, +7 \end{array}$$

$2^8$ 's compl: 1000 000 0111

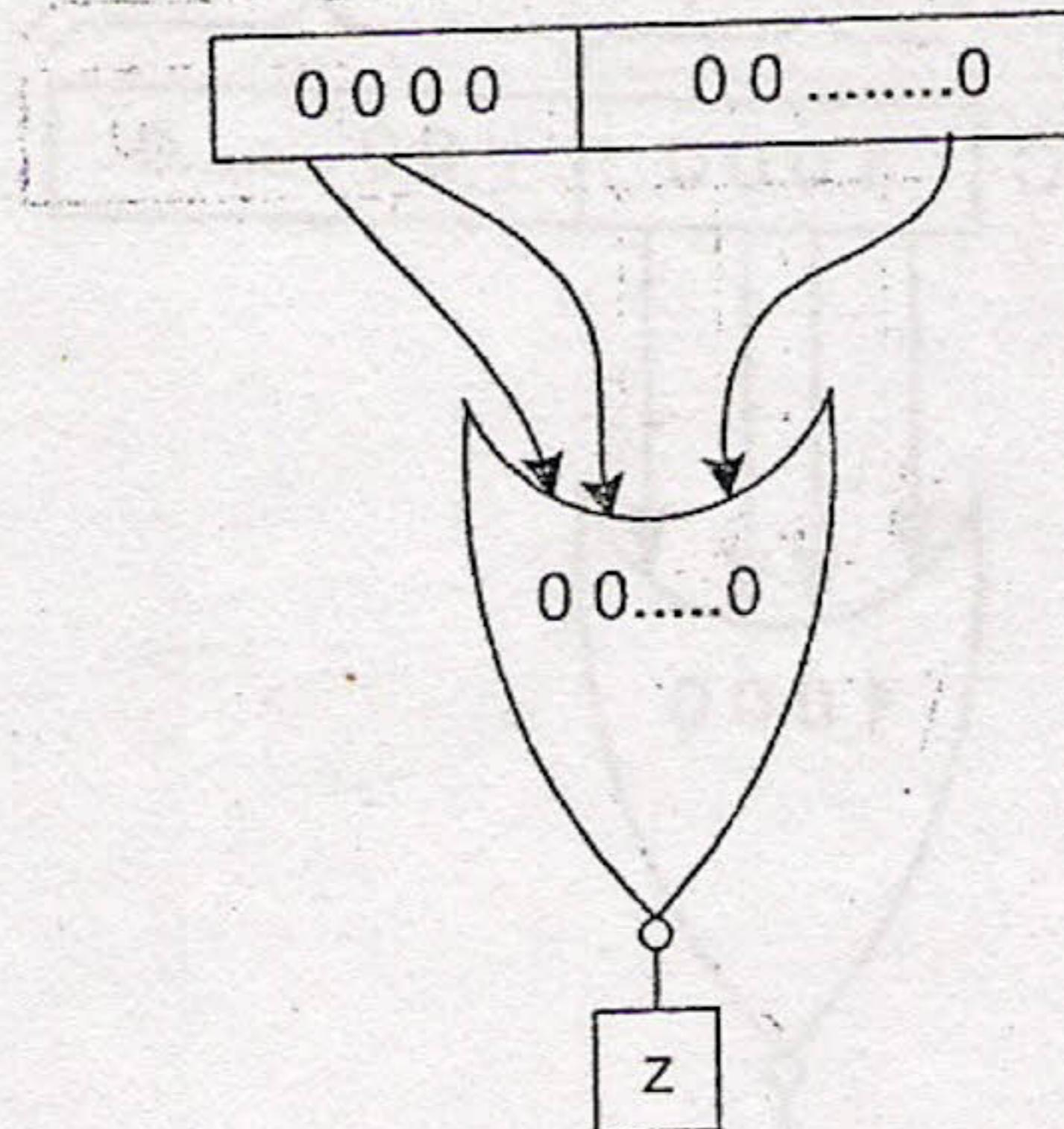
اختصاص 1000 به نمای عدد صفر با توجه به مقدار فوق، Zero - Flag را Reset و در نتیجه دستورالعمل‌های انشعاب شرطی که محتوى ثباتی را به منظور صفر بودن تست می‌کنند، مواجه با مشکل می‌شوند، به منظور رفع مشکل بهمنظور نشان دادن 8- باید از یک سیستم کدگذاری استفاده نماییم که در آن عدد 8- به صورت "0000" نشان داده شود که این سیستم کدگذاری Excess-8 Code می‌باشد.

4-bit Exp:

$$2^8: \quad \begin{array}{c} 1000, \dots, 0000, \dots, 0111 \\ \hline -8 \end{array}$$

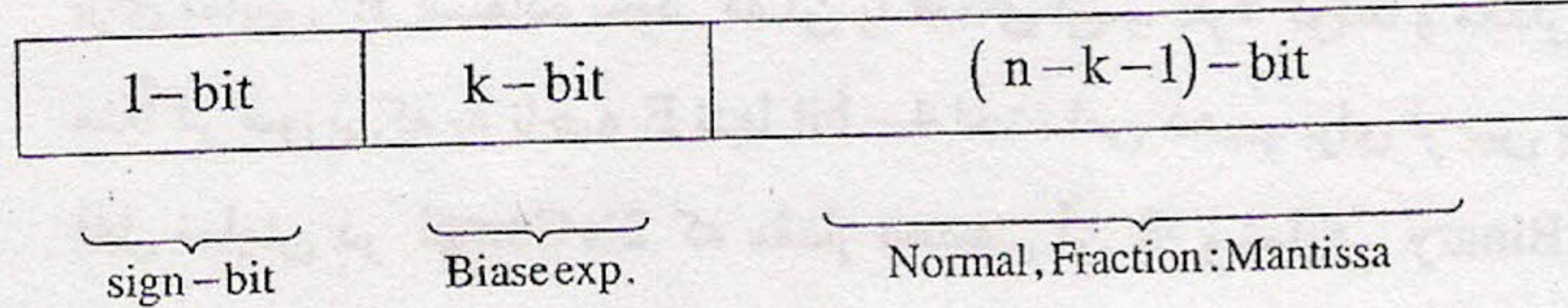
$$(x-8) \text{ Code: } \begin{array}{c} 0000, 1000, \dots, 1111 \\ \hline -8 \end{array}$$

در نتیجه:



بنابراین دیگر دستورالعمل‌های انشعاب شرطی که محتوی ثباتی را به منظور صفر بودن تست می‌کنند، موافق با مشکل نمی‌شوند. با توجه به نمایش نما در Biased exp (نمای اریب‌دار) و مقدار B متوجه می‌شویم که علامت کل عدد  $M \times B^E$  فقط به علامت M بستگی دارد. به منظور یکنواختی در نمایش اعداد با ممیز ثابت و اعداد با ممیز شناور، bit علامت مانیتس را در سمت چپ‌ترین موضع در نظر می‌گیریم. با فرض n-bit Register فرمت اعداد با ممیز شناور به صورت زیر خواهد بود.

n-bit Reg (Floating - Point)



چون B مقدار ثابت است، به صورت توکار در مدار جای داده می‌شود.

### دامنه اعداد قابل نمایش

در نمایش اعداد با ممیز ثابت به وسیله ثبات n بیتی  $(2^{32-1} - 1 + 2^{32-1}, \dots, 0, \dots, 0, 2^{32-1})$  می‌باشد. برای نمایش اعداد خارج از این دامنه باید از فرمت اعداد با ممیز شناور باید استفاده نمود.

به عنوان مثال: با فرض ثبات 32 بیتی دامنه اعداد قابل نمایش  $(-1, 2^{32-1})$  می‌باشد.

$$N = \left( 1 \underbrace{00\dots0}_{18 \text{ تا صفر}} \right)_{10} = 100 \times 10^{16} = 10 \times 10^{17} = 1 \times 10^{18} = .1 \times 10^{19} = .01 \times 10^{20} = \dots$$

عدد N خیلی بزرگ‌تر از  $(-1, 2^{32-1})$  می‌باشد. بنابراین برای نمایش آن در ثبات 32 بیتی باید از فرمت اعداد با ممیز شناور به صورت  $M^E$  استفاده نمود. اما بهترین روش برای نمایش N استفاده از  $.1 \times 10^{19}$  می‌باشد که آن را صورت نرمال گویند. بنابراین باید شرط نرمال بودن مانیتس کسری و Binary را مطالعه کنیم.

### هنچار سازی (Normalization)

استفاده از صورت نرمال دارای ۲ مزیت است:

۱) نمایش هر عدد با ممیز شناور منحصر به فرد خواهد بود.

۲) تعداد ارقام با معنی قابل نمایش افزایش خواهد یافت.

با فرض ثبات ۹ بیتی که در آن bit-3 به نما و bit-6 به مانتیس اختصاص یافته است. تنها می‌توان bit-6 از مانتیس را نشان داد.  
با فرض:

$$M \times B^E = .00001011011 \times 2^7 = .1011011 \times 2^3$$

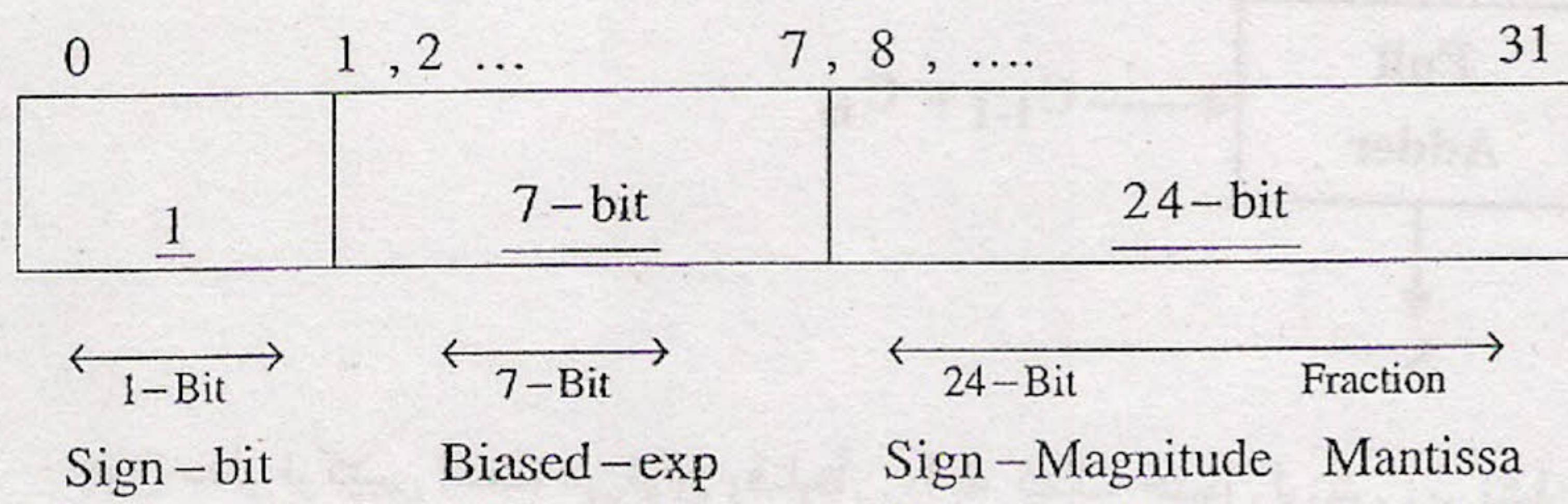
3-bit E	6-bit M	3-bit E	6-bit M
1 1 1	0 0 0 0 0	0 1 1	1 0 1 1 0 1
(روش a)		(روش b)	

ملحوظه می‌شود، روش b که به صورت نرمال است تعداد bit‌ها با معنی بیشتری را نشان می‌دهد.

$$\left. \begin{array}{l} \frac{1}{2} \leq M < 1 = 1xx\dots = 1 \times \frac{1}{2} + \dots \\ \text{Sign Bit} = 1 \cdot 0xx\dots x \\ \text{Sign Bit} = 0 \cdot 1xx\dots x \end{array} \right\} \begin{array}{l} \text{در ۱) MSB : Sign Mag.} \\ \text{در ۲) 2's complement} \end{array}$$

شرط نرمال بودن مانتیس کسری:

تمرین: با استفاده از فرمت اعداد با ممیز شناور در کامپیوتر IBM، نمایش کامپیوترا  $0.125 \times 10^5$  را به دست آورید:



### عملیات ریاضی (Arithmetic Operation)

حداقل انتظاری که از هر Fixed - Point ALU داریم این است که قادر به انجام چهار عمل اصلی جمع، تفریق، ضرب و تقسیم روی اعداد با ممیز ثابت باشد. ولی هر چهار عمل اصلی را می‌توان به کمک مدار Adder انجام داد، زیرا:

$$A - B = A + (-B) = A + 2's B = A + \left( \underbrace{1's B}_{(X-OR) \text{ Gate}} + 1 \right)$$

بنابراین با استفاده از مدار Adder (X-OR) Gate، عمل تفریق را می‌توان به کمک جمع انجام داد. همچنین ضرب را به کمک جمع مکرر و تقسیم را به کمک تفریق مکرر و نهایتاً به وسیله جمع مکرر انجام داد.

بنابراین طراح باید سعی کند مدار جمع کننده‌ای از نوع فوق العاده سریع را طرح کند که در این صورت می‌توان هر چهار عمل اصلی را به کمک جمع و با سرعت زیاد انجام داد و چون در جمع دو عدد در 2's Complement 2 بیت علامت نیز مانند سایر بیت‌ها در محاسبات شرکت می‌کند. بنابراین ما فقط مدار جمع کننده برای اعداد بدون علامت را مطالعه خواهیم کرد.

توجه به این نکته ضروری است که پیچیدگی مدار جمع کننده به کدی بستگی دارد که برای نمایش اپراتورها به کار می‌رود که ما فقط Binary Adder را بررسی می‌کنیم.

**Binary Adder**

با به قضیه شانون سریع ترین مدار جمع کننده را می‌توان به صورت مدار دو سطحی AND - OR متناظر با عبارت بولی مجموع حاصل ضربها طرح نمود. ولی در عمل دو عامل محدود کننده وجود دارد:

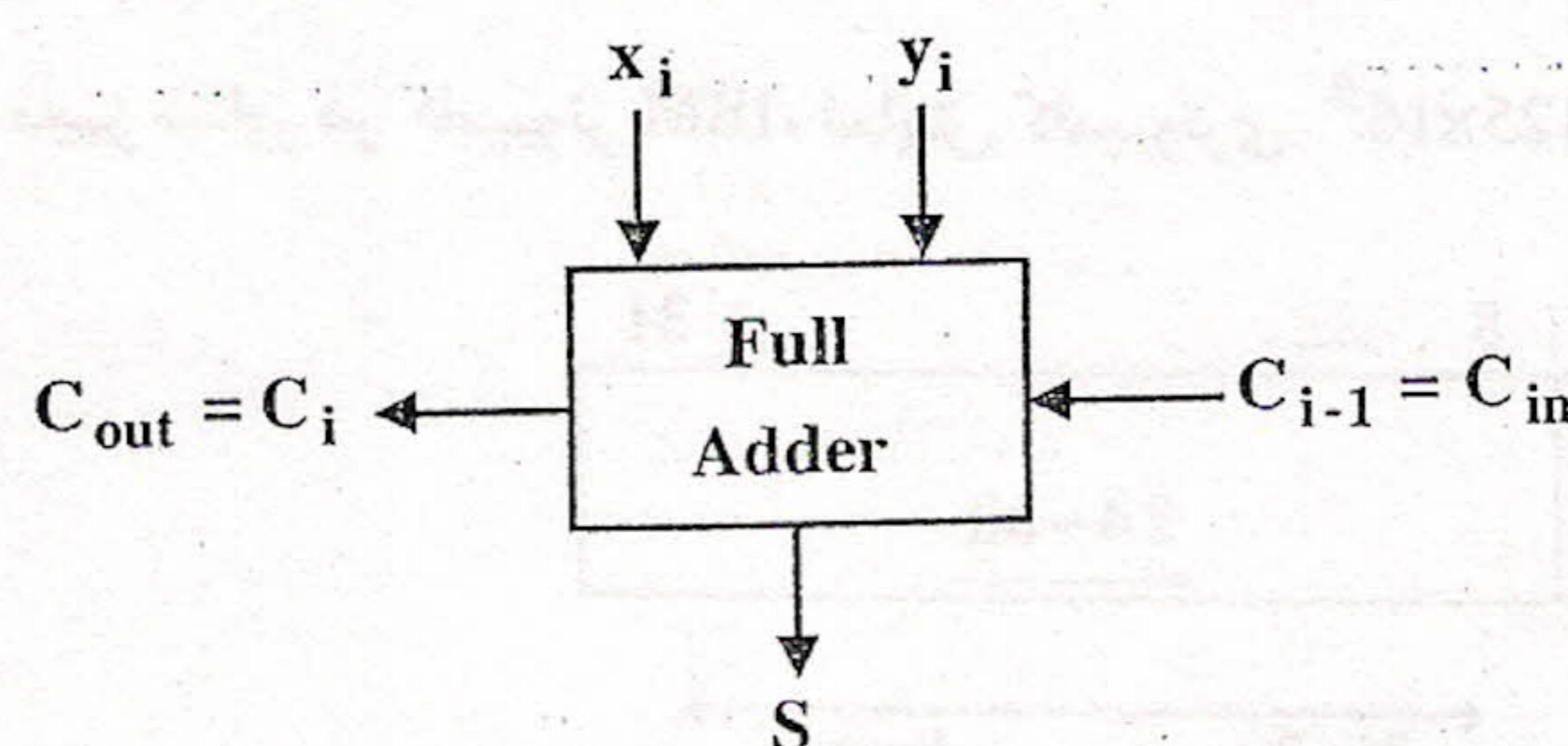
۱- پیدایش Fan-in

۲- افزایش تعداد Gate ها

بنابراین باید تا اندازه‌ای از سرعت جمع کننده صرف نظر نمود. ابتدا مدارهای جمع کننده  $m < n$  بیتی ( $m < n$ ) را طرح نمود که در آن  $Fan-in$  به وجود نماید و سپس اگر  $n = mk$  باشد،  $k$  تا از آنها را به صورت پشت سر هم به هم وصل کرد و ارقام نقلی را به صورت موجی بین آنها پخش نمود تا مدار جمع کننده  $n$  بیتی ایجاد گردد. در صورتی که  $m=1$  باشد، مدار جمع کننده را 1-bit Binary-Adder یا Basic Binary-Adder نمود.

**تمام افزایشگر : (Full-Adder)**

مدار ترکیبی است که می‌تواند حاصل جمع دو عدد یک بیتی را تولید کند. نمودار بلوکی آن به صورت زیر می‌باشد.

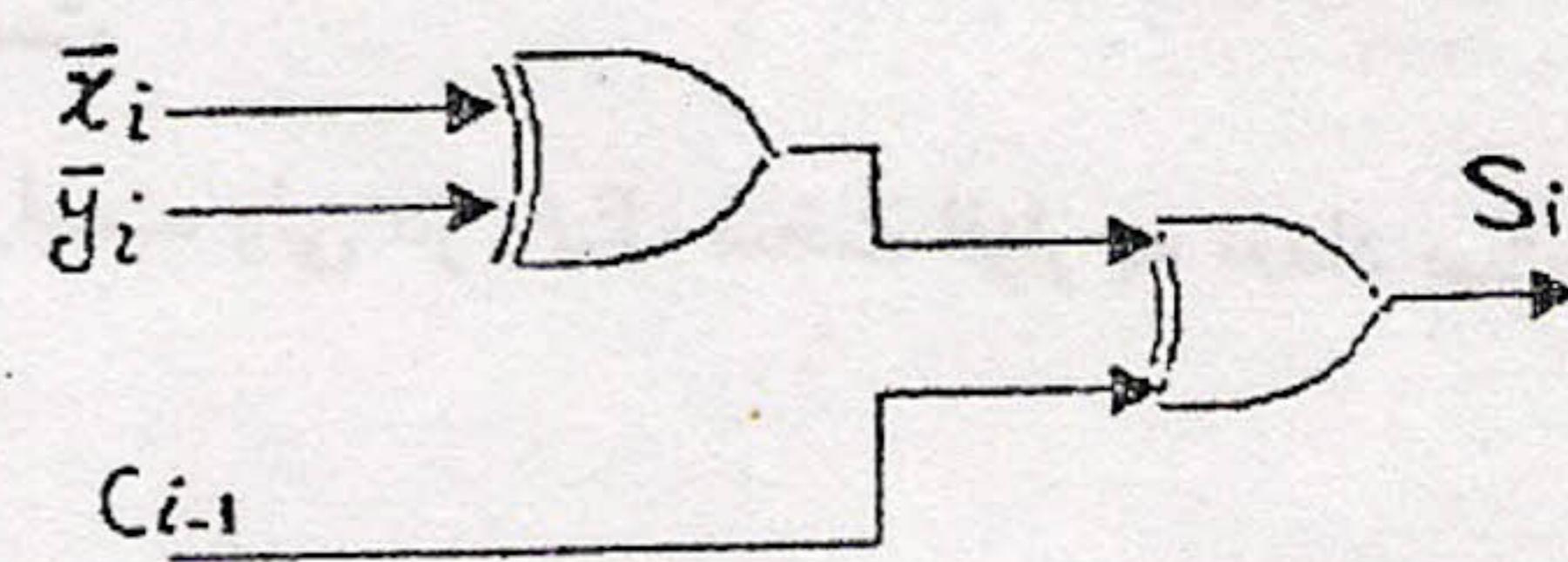


چون مدار ترکیبی است، می‌توان رفتار آن را به کمک جدول ارزش زیر نشان داد.

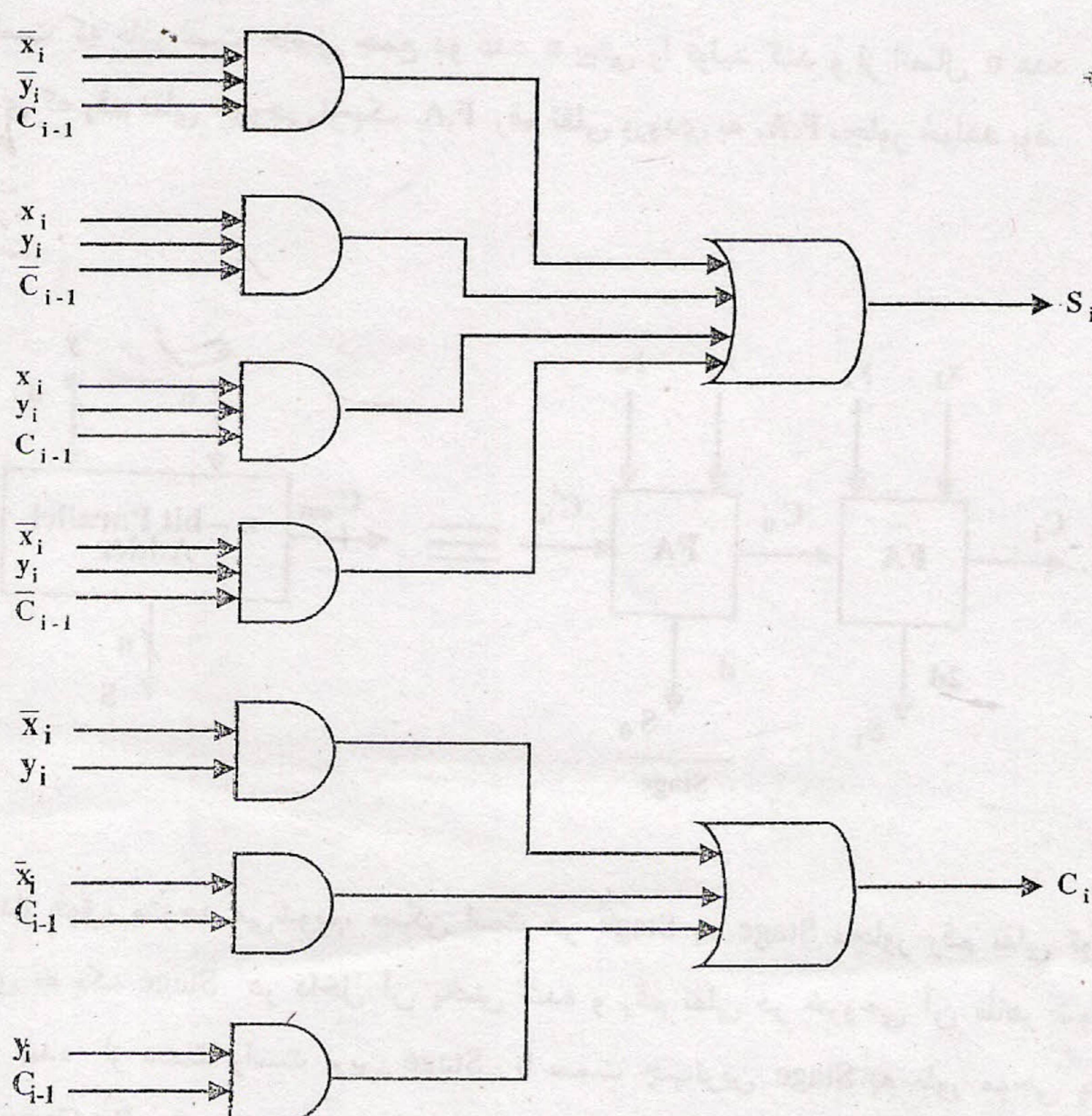
X <sub>i</sub>	Y <sub>i</sub>	C <sub>i-1</sub>	S <sub>i</sub>	C <sub>i</sub>
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$x_i y_i$	00	01	11	10
$c_{i-1}$	0	1	1	1
	0	1	1	1
	0	1	1	1
$S_i = \sum_1^4 (1, 2, 4, 7) = \bar{x}_i y_i \bar{c}_{i-1} + x_i \bar{y}_i c_{i-1} + \bar{x}_i, \bar{y}_i, c_{i-1} + x_i y_i c_{i-1}$				
$S_i = x_i \oplus y_i \oplus c_{i-1}$				

$x_i y_i$	00	01	11	10
$c_{i-1}$	0	2	6	4
	0	1	1	1
	0	1	1	1
$C_i = \sum_1^4 (3, 5, 6, 7) = x_i y_i + x_i c_{i-1} + y_i c_{i-1}$				

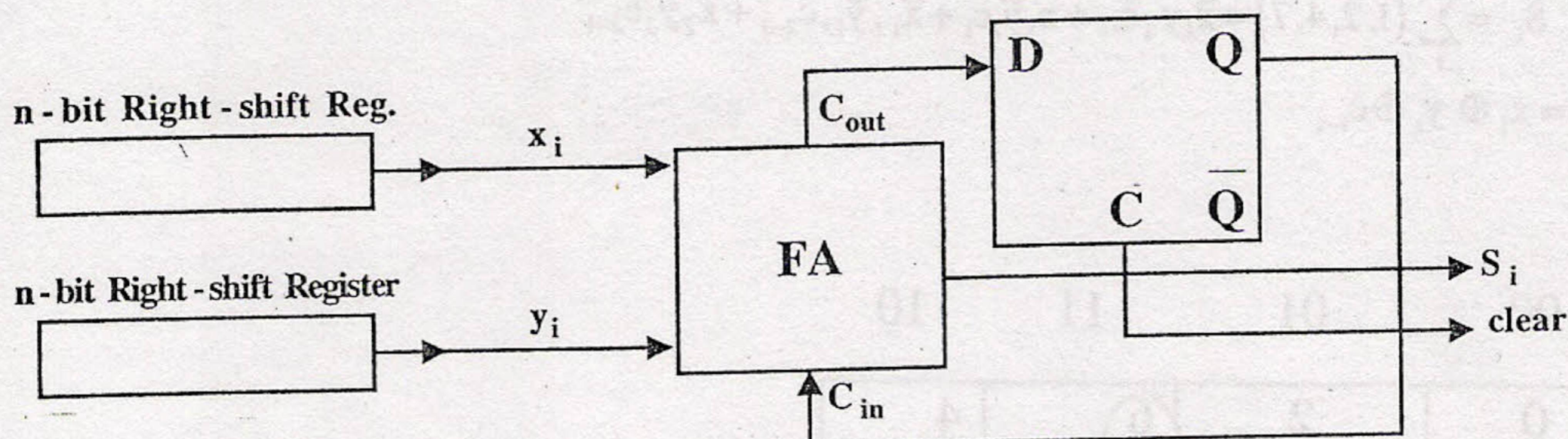


در صورتی که فقط از Gate های AND و OR استفاده کنیم، می‌توان داخل Full – Adder را به صورت زیر طرح نمود که به صورت مدار دوسرطی AND – OR می‌باشد.



**جمع کننده سری :**

مدار ترتیبی است که با اتصال یک F.A و یک D-Flip Flop به وجود می‌آید و در صورتی که تاخیر مدار دو سطحی F.A برابر  $d$  و تاخیر FF مساوی  $D$  باشد، آن‌گاه حاصل جمع دو عدد تک بیتی بعد از تاخیر  $(d+D)$  تولید خواهد شد.

**مقایسه Full Adder و Serial Adder**

۱) حاصل جمع دو عدد تک بیتی در صورت استفاده از Serial Adder پس از تاخیر  $(d+D)$  واحد زمان تولید می‌شود. ولی در صورت استفاده از F.A پس از تاخیر  $d$  واحد زمان بنابراین F.A سریعتر است.

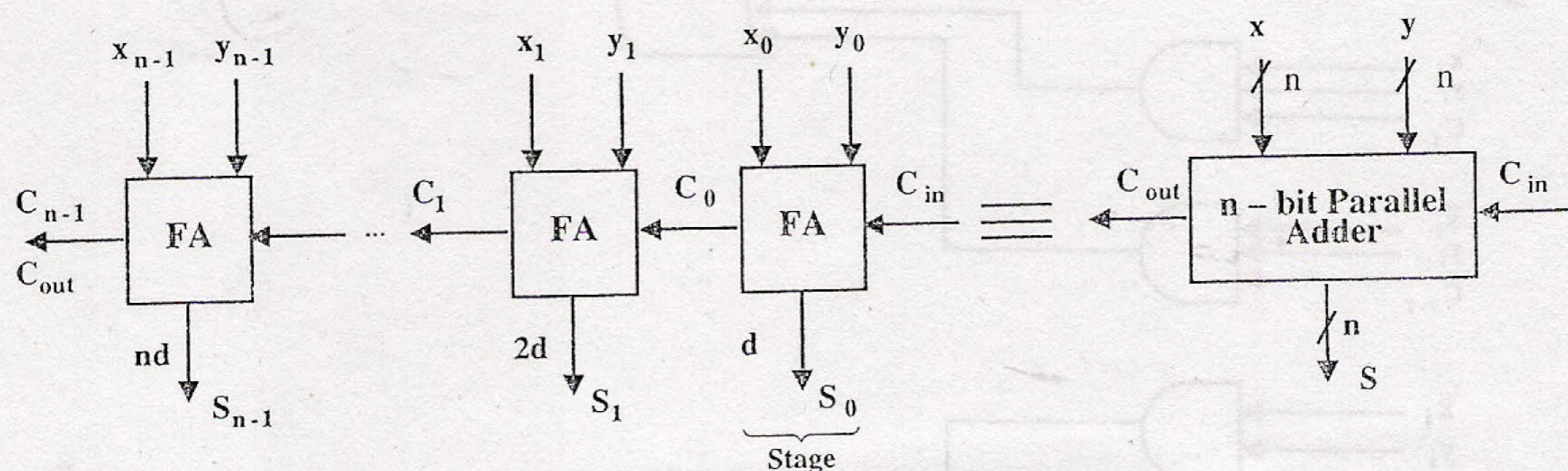
۲) در Serial Adder سخت افزار مستقل از تعداد بیت‌های اپراند ورودی است، ولی در F.A، سخت افزار به تعداد بیت‌های اپراند ورودی بستگی خطی دارد.

**جمع کننده موازی :** (Parallel - Adder یا Ripple Adder)

مدار ترکیبی است که قادر است حاصل جمع دو عدد  $n$  بیتی را تولید کند و از اتصال  $n$  عدد F.A به طور پشت سر هم به وجود می‌آید. به طوری که رقم نقلی خروجی از یک F.A. رقم نقلی ورودی به F.A. مجاور خواهد بود.

$$X = x_{n-1} \dots x_2, x_1, x_0$$

$$Y = y_{n-1} \dots y_2, y_1, y_0$$



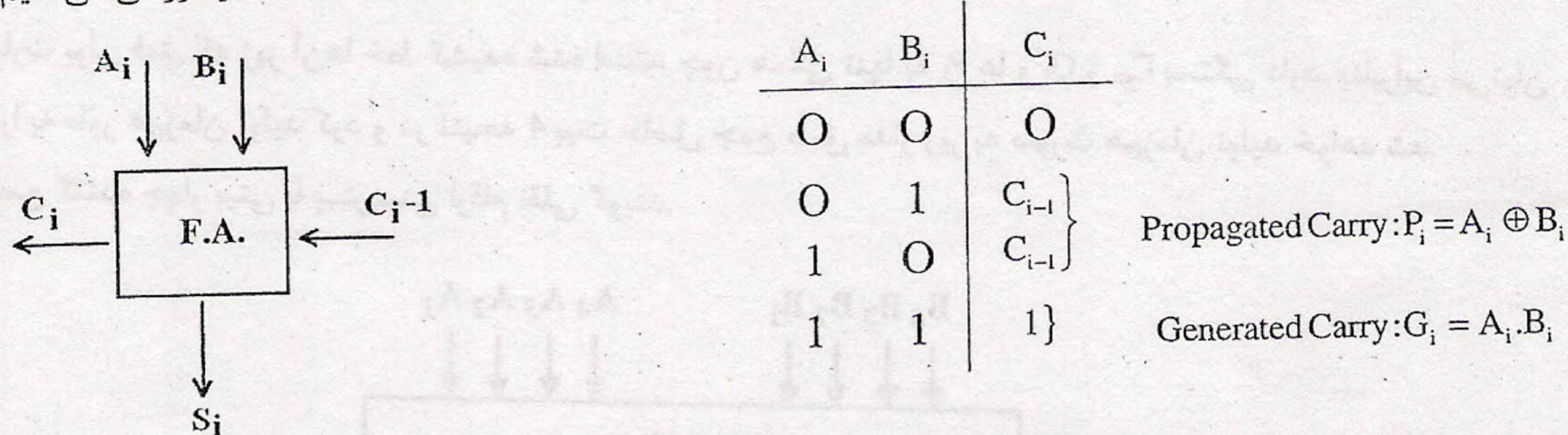
با توجه به نمودار فوق، متوجه می‌شویم، ممکن است هر Stage به Stage مجاور رقم نقلی تولید کند و همچنین ممکن است رقم نقلی ورودی به یک Stage در داخل آن پخش شده و رقم نقلی در خروجی آن ظاهر شود. در بدترین حالت ممکن است رقم نقلی تولید شده از سمت راست ترین Stage تا سمت چپ‌ترین Stage به طور موجی پخش شود در این صورت مدار را n-Bit Carry Ripple Adder گویند. در این مدار حاصل جمع پس از تاخیر  $nd$  تولید خواهد شد.

### جمع کنندۀ سریع : High Speed Binary Adder

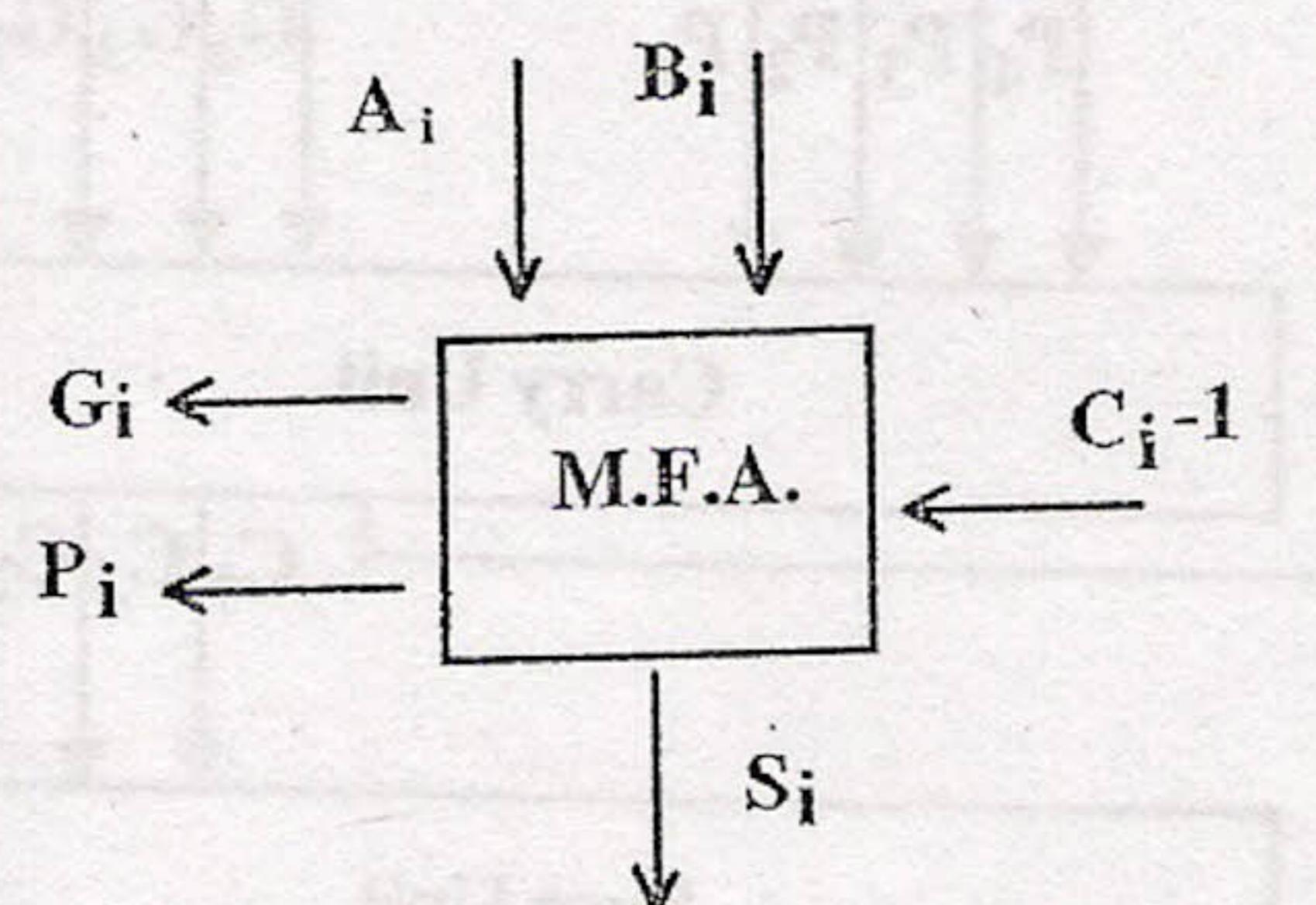
مهم‌ترین هدف در طراحی مدار جمع کنندۀ‌های سریع تقلیل Carry Ripple Time ( زمان تلف شده برای پخش موجی ارقام نقلی است ) و در صورتی که ارقام نقلی ورودی به Stage ها فقط به متغیرهای ورودی از دو اپراند بستگی داشته باشد، آن‌گاه می‌توان ارقام نقلی ورودی به کلیه Stage ها را به طور هم‌زمان تولید نمود. و در نتیجه بیت‌های حاصل جمع را به صورت هم‌زمان تولید می‌شود.

#### جمع کنندۀ با پیش‌بینی ارقام نقلی (Carry Look ahead Adder)

برای طراحی n-Bit Carry Lookahead Adder ابتدا رابطه بین رقم نقلی ورودی و خروجی به یک F.A. را بررسی می‌کنیم.



پس طرح FA اصلاح شده ( Modified Full Adder ) به صورت زیر خواهد بود:

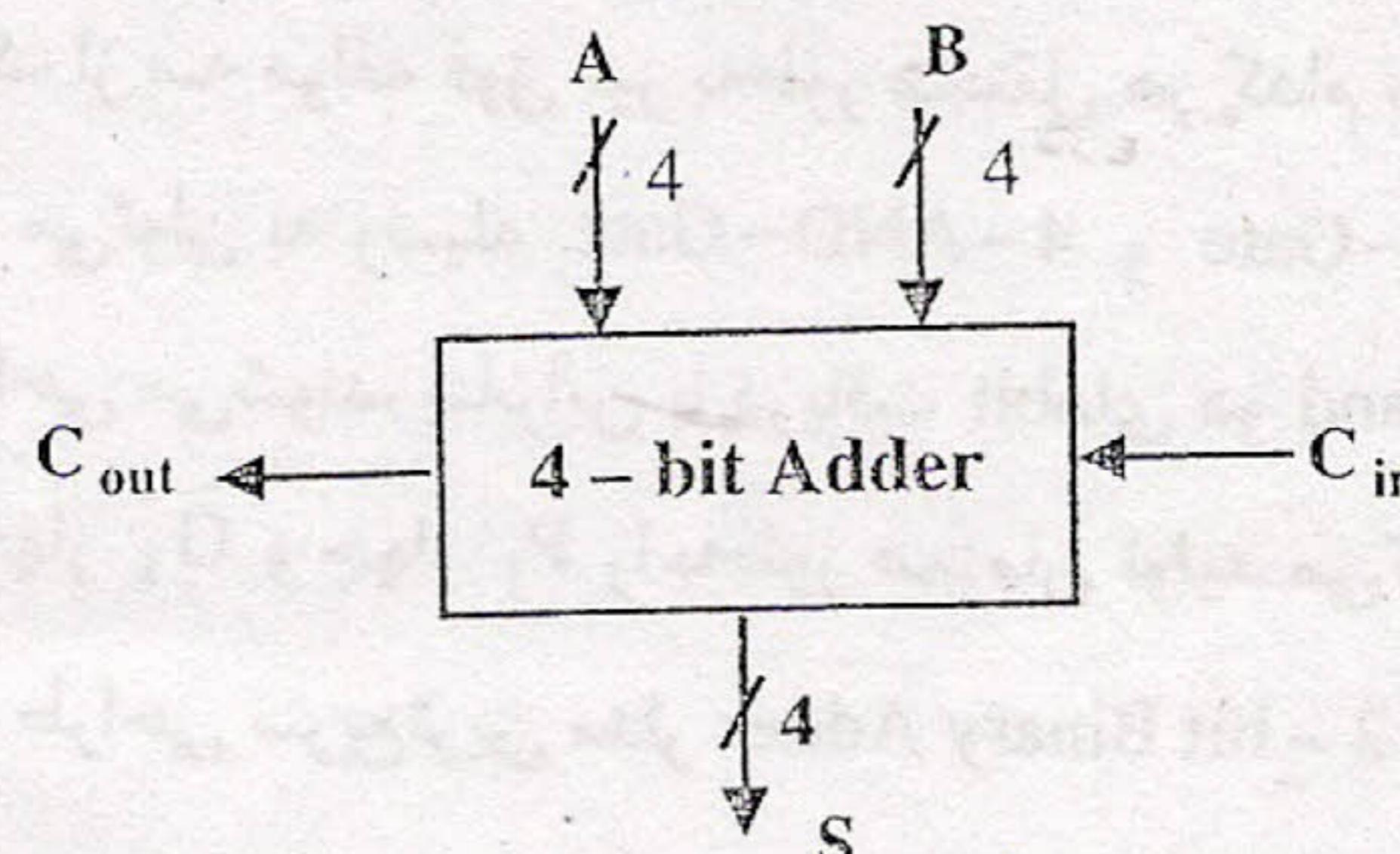


مثال: مطلوب است طراحی 4-Bit Carry Lookahead Adder

$$\begin{array}{r} C_4 \\ A = A_4 A_3 A_2 A_1 \\ B = B_4 B_3 B_2 B_1 \\ \hline S = S_4 S_3 S_2 S_1 \end{array}$$

$$\left\{ \begin{array}{l} C_1 = G_1 + C_{in} P_1 \\ C_2 = G_2 + C_1 P_2 \\ C_3 = G_3 + C_2 P_3 \\ C_4 = G_4 + C_3 P_4 \end{array} \right.$$

$$\left\{ \begin{array}{l} S = P \oplus C_{in} \\ S_2 = P_2 \oplus C_1 \\ S_3 = P_3 \oplus C_2 \\ S_4 = P_4 \oplus C_3 \end{array} \right.$$



با توجه به عبارات فوق متوجه می‌شویم که عبارت بولی  $C_i$  ها به طور بازگشتی تعریف شده‌اند. در صورتی که بتوانیم با بسط دادن عبارت بولی حالت بازگشتی بودن آنها را از بین ببریم، قادر خواهیم بود ۴ بیت رقم نقلی را به طور همزمان تولید کنیم و چون  $P_i$  ها و  $G_i$  ها به طور همزمان در دست هستند، بنابراین می‌توان ۴ بیت حاصل جمع را به طور همزمان تولید نمود.

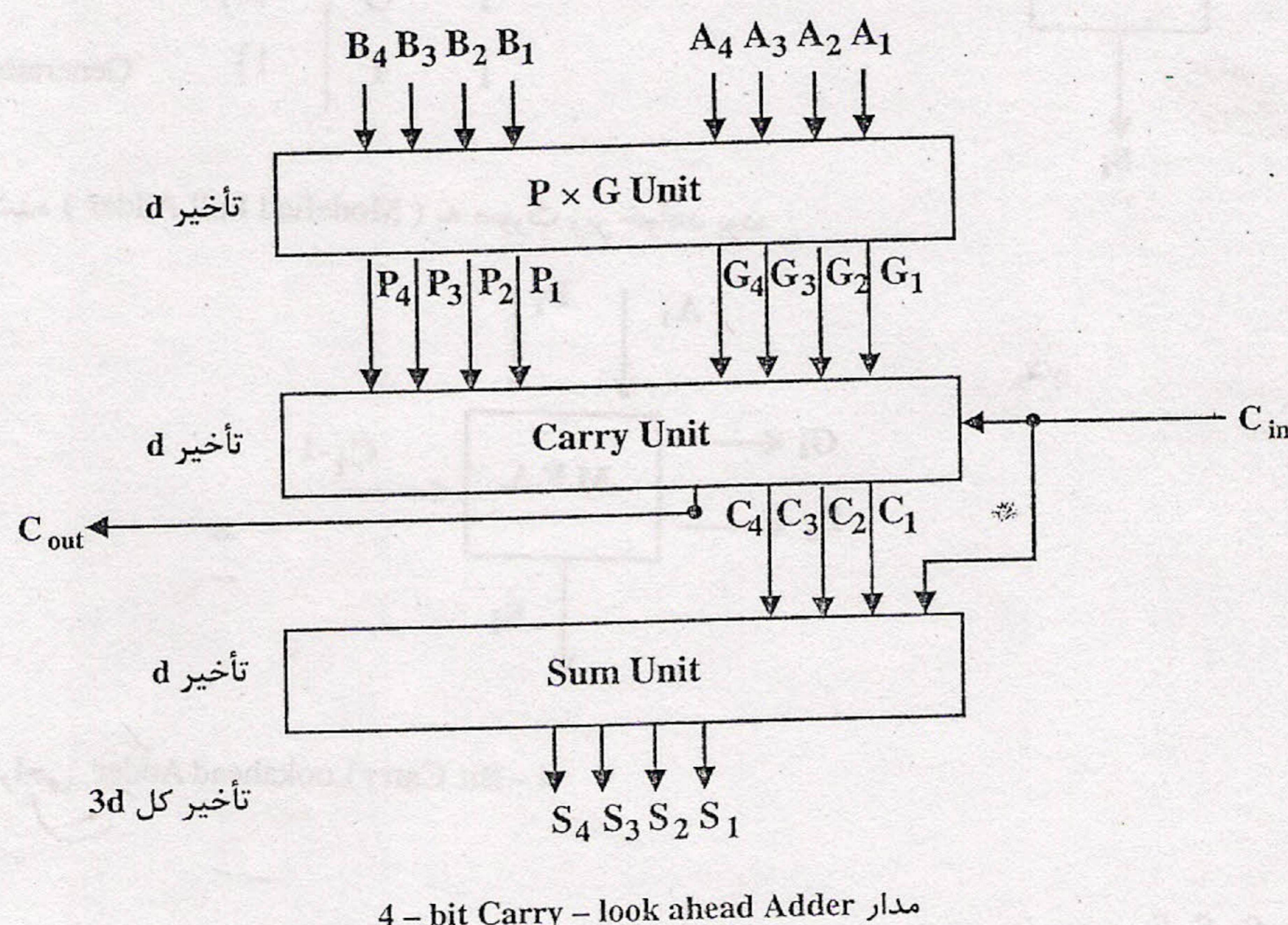
$$C_1 = G_1 + C_{in} P_1$$

$$C_2 = G_2 + (G_1 + C_{in} P_1) P_2 = G_2 + G_1 P_2 + C_{in} P_1 P_2$$

$$C_3 = G_3 + (G_2 + G_1 P_1 + C_{in} P_1 P_2) P_3 = G_3 + G_2 P_3 + G_1 P_2 P_3 + C_{in} P_1 P_2 P_3$$

$$C_4 = G_4 + G_3 P_4 + G_2 P_3 P_4 + G_1 P_2 P_3 P_4 + C_{in} P_1 P_2 P_3 P_4$$

با توجه به ۴ عبارت بولی فوق که زیر آن‌ها خط کشیده شده است، چون همگی تنها به  $P_i$  ها و  $G_i$  و  $C_{in}$  بستگی دارد، بنابراین می‌توان چهار رقم نقلی را به طور همزمان تولید کرد و در نتیجه ۴ بیت حاصل جمع طبق مدار زیر به صورت همزمان تولید خواهد شد. چنین مدار را جمع کننده چهار بیتی با پیش‌بینی ارقام نقلی گویند.



مدار فوق به طور کامل بر روی یک IC وجود دارد. هم‌چنین هر یک از سه مولفه فوق نیز به‌طور مستقل هر کدام بر روی یک IC وجود دارد. با توجه به فرمول‌های صفحات قبل واحد P and G unit را می‌توان به وسیله 4-AND-Gate و 4-X-OR-Gate طرح نمود و چون IC ها به صورت یا تمام NOR یا تمام NAND طراحی می‌شوند، بنابراین با دریافت Operand دو ورودی، بعد از تأخیر d واحد زمان (d تأخیر مدار دو سطحی فرض شده است) چهار  $G_i$  و چهار  $P_i$  را به‌طور همزمان تولید می‌کند. تمرین - با استفاده از مفهوم Carry – Lookahead، مطلوب است طراحی سریع‌ترین مدار 12 – bit Binary Adder.