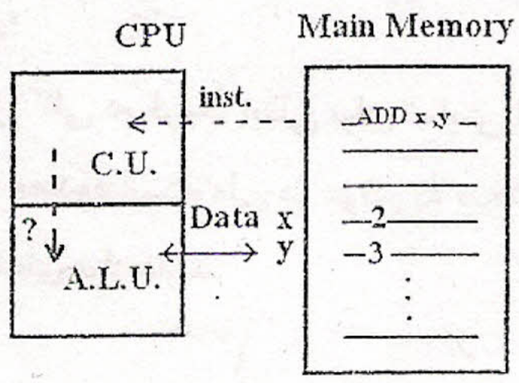
فصل پنجم

شیوه نمایش اطلاعات

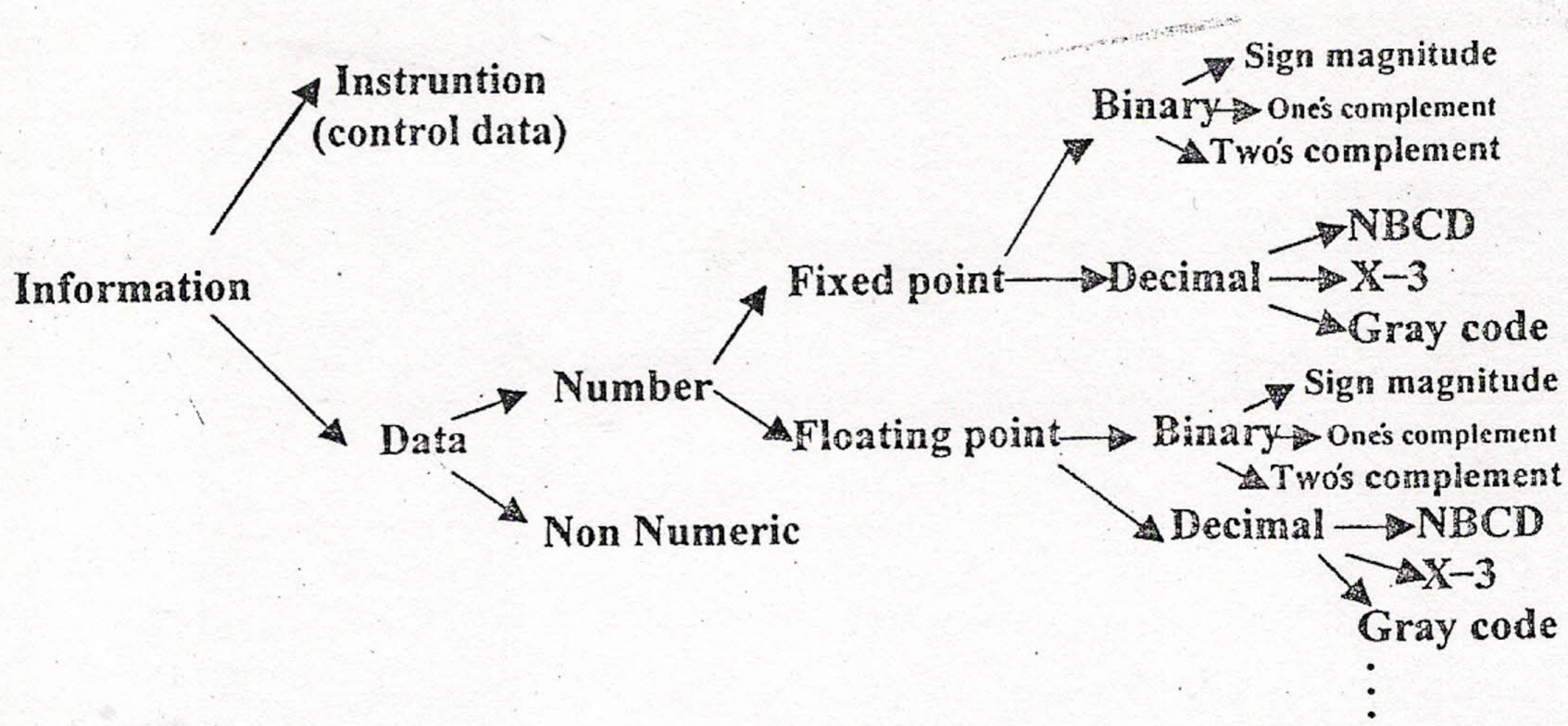
نمایش اطلاعات *Information Representation*

می‌دانیم در محاسبه دو مولفه نقش اساسی دارد:



۱ـ **Main Memory (حافظه اصلی)**: که در آن دستورالعمل‌ها و داده‌ها ذخیره می‌شوند.

۲ـ **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: Fixed- point و Floating -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 دهدهی استفاده می‌شود.

$$\begin{pmatrix} 2^m & & 2^1 & 2^0 & & 2^{-1} & 2^{-2} & & 2^{-n} \\ b & \cdots & b & b & \bullet & b & b & \cdots & b \\ m & & 1 & 0 & & -1 & -2 & & -n \end{pmatrix}_2 = \left(\sum_{i=-n}^{m} (b_i \times 2^i) \right)_{10} \qquad b_i \in \{0,1\}$$

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.

$$\text{Most significant-bit} \quad \text{Least significant-bit}$$

$$\underbrace{x_{n-1}}_{\text{sign-bit}} \; \underbrace{x_{n-2} \ldots x_1 \, x_0}_{\text{magnitade}}$$

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 دهدهی خواهد بود.

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

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

(چرا؟) $0 \le |N| \le 2^{n-1}$

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

$0 \le |N| \le 1-2^{n-1}$

روش‌های نمایش اعداد منفی: برای نمایش اعداد منفی سه روش زیر وجود دارد.

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

مثال: با فرض .4-bit Reg، نمایش مثبت و منفی عدد 3 در sign-magnitude به صورت زیر خواهد بود که در آن چه عدد مثبت و چه منفی باشد bit ها دارای ارزش مکانی هستند.

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

II) One's Complement :

ابتدا (r-1)'s Complement را تعریف می‌کنیم.

تعریف: فرض کنیم عدد مثبت N در پایه‌نویسی r شامل n رقم صحیح و m رقم کسری باشد، آن‌گاه برای نمایش عدد منفی N(-N) می‌توان (r-1)'s Complement استفاده نمود.

فرض کنید که:

N یک عدد مثبت
- r = پایه عدد نویسی
- n = تعداد رقم صحیح
- m = تعداد رقم کسری

برای نمایش عدد N– می‌توان از (r-1)'s Complement استفاده کرد که به صورت زیر تعریف می‌شود:

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

بنابراین برای نشان دادن اعداد منفی در سیستم‌های عددی نویسی متفاوت می‌توان:

استفاده نمود
- Binary $r = 2 \Rightarrow$ 1's Complement
- Octal $r = 8 \Rightarrow$ 7's Complement
- Decimal $r = 10 \Rightarrow$ 9's Complement

به منظور به‌دست آوردن الگوریتم (r−1)'Complement به مثال زیر توجه نمایید.

مثال: فرض کنید که:

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

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

$$= (100 - 0.001) - 25.742$$

$$= 99.999 - 25.742$$

$$= 74.257$$

(r−1)	(r−1)	(r−1)	(r−1)	(r−1)
9	9.	9	9	9
−2	5.	7	4	2
7	4.	2	5	7

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 صورتی‌که اعداد منفی در 1's Complement نشان داده شوند، آن‌گاه bit ها 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 r's Complement نشان داد که به صورت زیر تعریف می‌شود:

فرض کنید:

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

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

برای نمایش عدد منفی در پایه عدد نویسی متفاوت می‌توان:

Binary	r=2	2's Complement	
Octal	r=8	8's Complement	استفاده نمود.
Decimel	r=10	10's Complement	

مثال: فرض کنید

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

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

$$= 100 - 25.840$$

$$= 74.160$$

	(r−1)	(r−1)	(r−1)	(r−1)	(r−1)	r
	1	0	0 .	0	0	0
−		2	5 .	8	4	0
		7	4 .	1	6	0

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

توجه: در صورتی‌که اعداد منفی در 2's Complement نشان داده شوند، آن‌گاه bit ها دارای ارزش مکانی نخواهند بود، مگر دوباره از آن 2's Complement بگیریم.

مثال: با فرض $N = .0110$

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

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

$$= 1 - 0.0110$$

$$= 0.1010$$

$$\begin{array}{l} 0\ 11\ 20 \\ 1.0000 \ - \\ \ \ .0110 \\ \hline \ \ .1010 \end{array}$$

الگوریتم به‌دست 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 :

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

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

$$= 0.1001 + 0.0001 = 0.1010$$

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:

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

$$-N = -0.0110 : 2^{0} - 2^{-4} - 0.0110 = 1 - 0.0001 - 0.0110$$

$$= 0.1111 - 0.0110$$

$$= 0.1001$$

مقایسه *1's Complement* با *2's Complement* :

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

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

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

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

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

محاسبات دودویی :

Binary Arithmatic

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

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

در نتیجه:

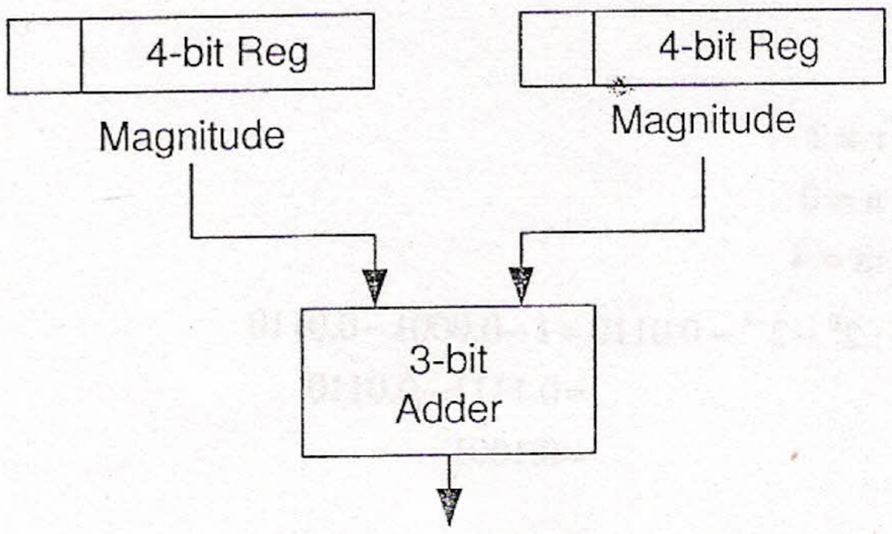
$$A-B=A+(-B)=A+2's=A+(\underbrace{1's\ B}_{\text{می‌توان با استفاده از (X-OR) Gate تولید نمود.}}+1)$$

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 انجام عمل جمع در Sign Magnitude بیت‌های علامت را در عمل جمع شرکت نمی‌دهند. در صورتی‌که دو عدد متحدالعلامه باشند، قدرمطلق آنها با هم جمع و علامت مشترک را به عنوان علامت نتیجه قرار می‌دهند و در صورتی‌که 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.

$$\begin{array}{r} +3 \\ +5 \\ \hline +2 \end{array}$$



۲. جمع در 2's Complement :

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

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

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

$$
\begin{array}{rr}
 & \text{7-bit Reg(2's)} \\
-6 & 1\ 1\ 1\ 1\ 0\ 1\ 0 \\
-9 & +1\ 1\ 1\ 0\ 1\ 1\ 1 \\
\hline
-15 & 1\ 1\ 1\ 0\ 0\ 0\ 1 = -(0001111) = -15 \\
 & \textcircled{1} \\
 & \text{End carry}
\end{array}
$$

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 انضمام Sign Bit با هم جمع می‌کنیم و در صورت وجود رقم نقلی در موضع Sign Bit آن با حاصل‌جمع مرحله اول دوباره جمع می‌کنیم و در صورت وجود رقم نقلی در مرحله دوم، از آن صرف‌نظر می‌کنیم.

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

$$
\begin{array}{rr}
 & \text{7-bit Reg(1's)} \\
-6 & 1\ 1\ 1\ 1\ 0\ 0\ 1 \\
+9 & +1\ 1\ 0\ 0\ 1\ 1\ 1 \\
\hline
+3 & 0\ 0\ 0\ 0\ 0\ 1\ 0 \\
\text{E.A.C.} & \\
\textcircled{1} & + \qquad\qquad\qquad 1 \\
\hline
 & 0\ 0\ 0\ 0\ 0\ 1\ 1 = +(3)
\end{array}
$$

به رقم نقلی تولید شده از جمع مرحله اول (E.A.C.) End Around Carry (رقم نقلی دورگشتی) گویند که آن را دوباره با نتیجه حاصل از مرحله اول جمع می‌کنیم.

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

Decimal Code:

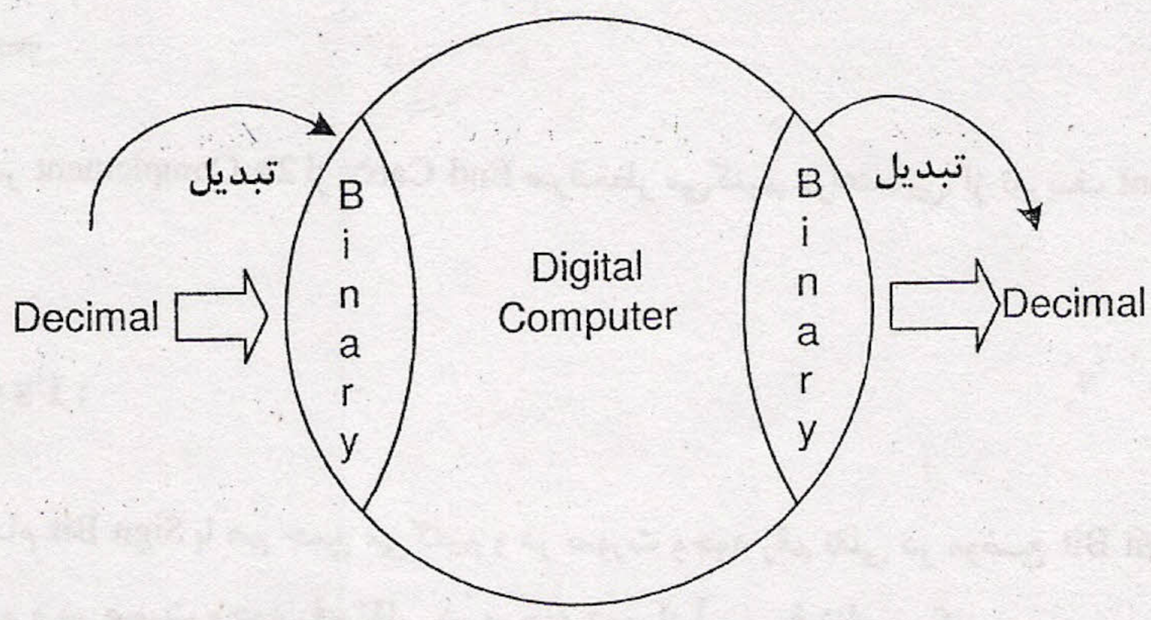
سیستم‌های رقمی با سیگنال‌هایی محدود به دو مقدار ممکن و عناصری با دو حالت پایدار کار می‌کنند، بنابراین یک تناظر یک به یک بین سیگنال‌های دودویی و ارقام دودویی وجود دارد.

قبل از مطالعه Decimal Code باید بین معادل Binary یک عدد و Binary Coding آن تفاوت قایل 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 دهدهی 9,...,2,1,0 حداقل به 4 بیت نیاز داریم که انتخاب 10 ترکیب از بین 16 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 Cided Decimal

NBCD یک کد وزن دار است و متداول‌ترین روش برای نشان دادن ارقام دهدهی 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 عمده NBCD عبارتند از:

۱) 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:

	NBCD	
+5	0101	
+9	1001	
+14	1110	Binary Addition
	+0110	Correction
1	0100	

۲) 9's Complement 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.

$$5 = 0101 \xrightarrow{9's} 4 \neq 1010 \xrightarrow{1'S} 5 = 0101$$

۳) Excess-3code : کد بدون وزن است ولی دارای خاصیت خود متمم (Self – Complementary) می‌باشد، یعنی 9's Complement آن را می‌توان از تبدیل 0 ها به 1 و 1 ها به 0 به دست آورد. هم‌چنین در جمع دو رقم، رقم نقلی لازم تولید می‌شود، هر چند که نیاز به تصحیح دارد.

+5	1000
+9	1100
14	1 0100
	+0011
	1 0111

$$5 \xrightarrow{9's} 4 = \qquad 1000 \xrightarrow{1's\ C} 0111$$

Self Complementory

$$5 = 1011 \xrightarrow{9'sC} 4 = 0100 \xrightarrow{1'sC} 5 = 1011$$

۴) Aiken - Code

Aiken خاصیت خود متممی دارد.

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

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

$$\text{Aiken}: \sum(W_i) = 2+1+2+1 = 9 = 4+5$$

چون در Aiken محور تقارن درست وسط سیستم کدگذاری یعنی بین code دو عدد متوالی 5,4 قرار می‌گیرد به این دلیل یک code خود متمم است. ولی در NBCD وسط سیستم‌کدگذاری نیست، زیرا بین 8,7 قرار می‌گیرد. به این دلیل NBCD یک کد خود متمم نیست.

۵) Gray - Code

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

a) برای کدگذاری سطر و ستون در جدول کارنو استفاده نمود.

b) در طراحی شمارنده‌ها 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 گسسته تبدیل نمود.

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

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

1. $g_n = b_n$ for $k = n$
2. $g_k = (b_k + b_{k+1}) MOD2$ for $k = 1, 2., \ldots, n-1$
3. $g_k = b_k \oplus b_{k+1}$ Exclusive Or

مثال: فرض کنید داشته باشیم:

$$\overset{b_n}{(1}1001010110)_2 = (?)_{Gray} = (g_n g_{n-1} \ldots g_1)_{Gray}$$

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

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

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

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

$$\begin{cases} b_n = g_n & \text{for } k = n \\ b_i = \sum_{i=k}^{1} g_i MOD_2 & \text{for } k = (n-1), \ldots, 1 \end{cases}$$

مثال:

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

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

$$b_{10} = (g_{10} + g_{11}) MOD2 = (1+0) MOD2 = 1$$

$$b_9 = (g_9 + g_{10} + g_{11}) MOD2 = (1+0+1) MOD2 = 0$$

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.

۶) **Excess – 3Gray – Code:**

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

x – 3 Gray – code

235	0111	0101	1100
764	1111	1101	0100
	7	6	4

9's

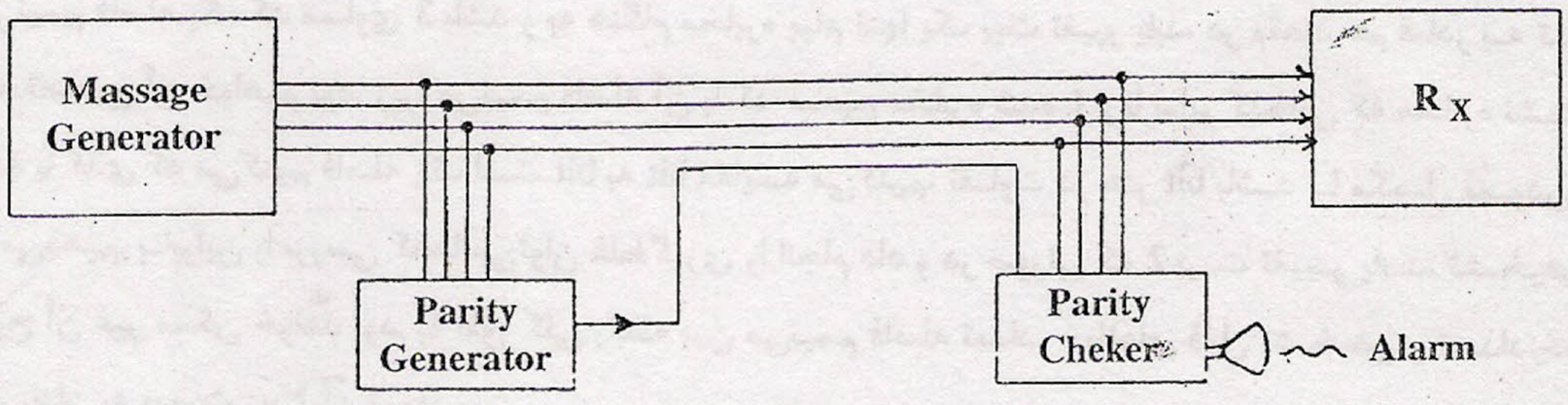
کدهایی با بیش از ۴ بیت:

کدهای 4 بیتی مانند NBCD , x−3 ,... کدخطایاب نیستند، یعنی در صورت مخابره پیام از یک مبدا به یک مقصد در صورتی‌که به دلیل وجود پارازیت بین راه یک bit تغییر یابد در مقصد قادر به تشخیص خطا نخواهیم بود. بنابراین به منظور تشخیص خطا در یک bit باید از 5 – bit Code استفاده کنیم. به عنوان مثال اگر کد رقم 2 دهدهی NBCD، یعنی 0010 را از مبدا مخابره و بین راه بیت دوم آن تغییر یابد و در مقصد پیام 0000 را دریافت کنیم قادر به تشخیص خطا نخواهیم بود، زیرا 0000 یک کد مجاز برای رقم دهدهی 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 Parity – Bit به عنوان بیت پنجم می‌توان تعداد یک‌های موجود در Code را فرد یا زوج نمود که آن‌ها را به ترتیب Odd Parity و Even Parity گویند. مثلا 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).

۲) 2- Out Of -5 Code :

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

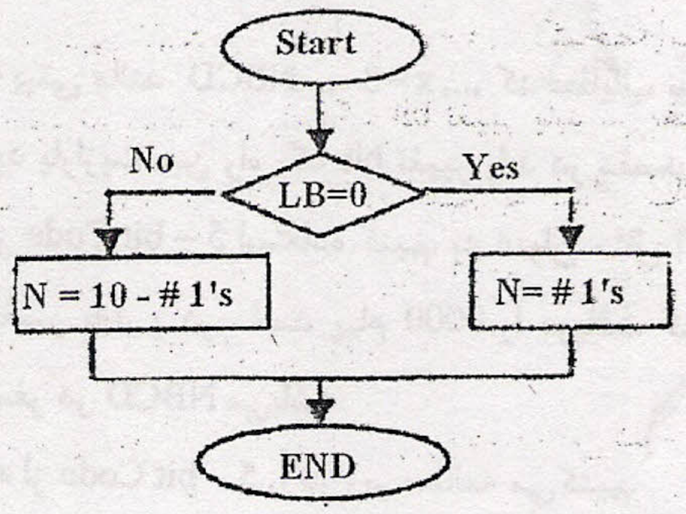
این کد در واقع یک کد نیمه وزن‌دار می‌باشد (Semi Wighted) که به جز کد اختصاص یافته به رقم صفر دهدهی کدهای اختصاص یافته به سایر ارقام دهدهی به ترتیب دارای وزن 7,4,2,1,0 هستند. یعنی این وزن‌ها تنها در مورد Code رقم صفر صادق نیست، ولی درمورد Code بقیه ارقام صادق می‌باشد.

۳) Johnson Code:

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 5 بیت:

تا به حال کدهایی را مورد مطالعه قرار دادیم که تنها قادر به تشخیص خطاهای تک بیتی بودند. برای این‌که بتوان وجود خطای تک بیتی را هم تشخیص داد و هم تصحیح نمود به کدهایی با بیش از 5 بیت نیاز داریم.

تعریف 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. مثلا برای کدهای NBCD، X-3، Gray – Code و Aiken می‌نیمم فاصله برابر 1 ولی برای 2 Out Of 5 می‌نیمم فاصله مساوی 2 می‌باشد.

در صورتی که می‌نیمم فاصله یک کد مساوی 3 باشد و به هنگام مخابره پیام تنها یک بیت تغییر یابد، در مقصد هم قادر به تشخیص خطا و هم قادر به تصحیح آن خواهیم بود، زیرا می‌نیمم 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 خطا ممکن ولی تصحیح آن غیر ممکن خواهد بود. به طور کلی رابطه بین می‌نیمم فاصله تعداد خطاهای قابل تشخیص و تعداد خطاهای قابل تصحیح را می‌توان به صورت زیر بیان نمود:

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

که در آن m: می‌نیمم فاصله d: تعداد خطاهای قابل تشخیص و C: تعداد خطاهای قابل تصحیح می‌باشد.

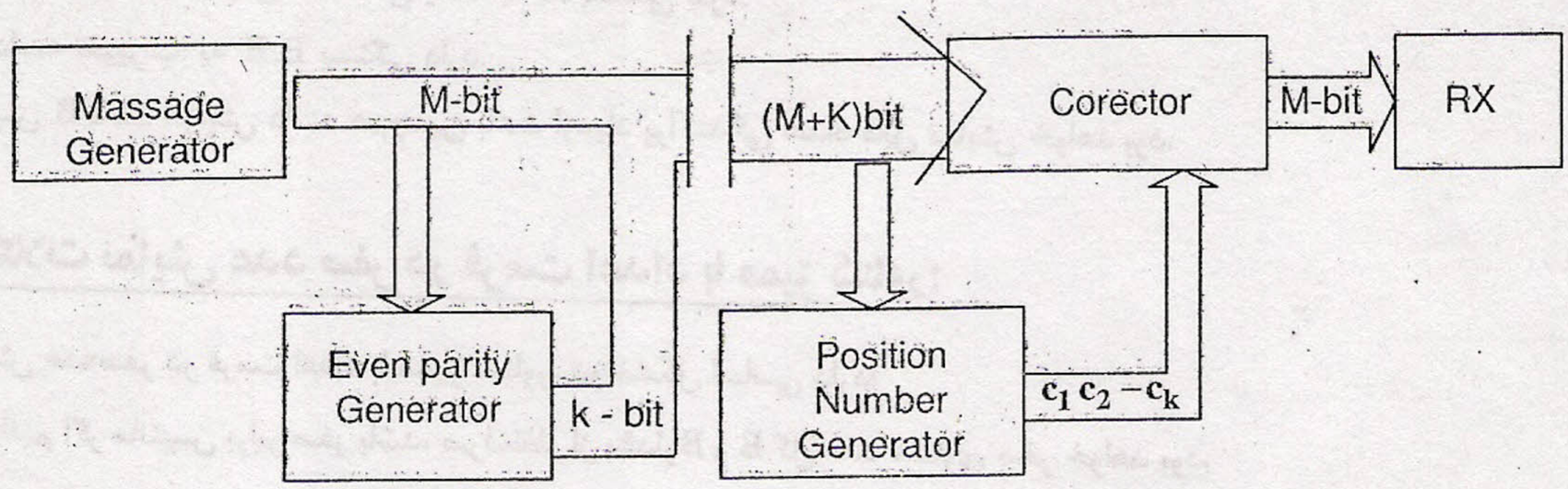
جدول زیر با توجه به m تعداد خطاهای قابل تشخیص و قابل تصحیح را نشان می‌دهد.

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

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

کد همینگ: *Hamming - Code*

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



اگر به کد m بیتی، k بیت پاریتی (Parity) اضافه کنیم، کدهای m+k بیتی تولید می‌شوند که در آن موضع بیت‌ها را از چپ به راست با 1,2,3....(m+k) شماره‌گذاری می‌کنیم. بیت پاریتی روی بیت‌های پیام کنترل انجام می‌دهند و عدد دودویی $(c_1c_2 ,.....,c_n)_2$ تولید می‌کند که معادل دهدهی آن محل خطا را مشخص می‌سازد و با متمم‌گیری از آن بیت می‌توان خطا را تصحیح نمود. اگر عدد دهدهی حاصل صفر باشد به مفهوم این خواهد بود که در پیام مخابره شده خطایی 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 دهدهی در NBCD باشد، آن‌گاه داریم:

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

برای این‌که بیت‌های توازن بتوانند روی بیت‌های پیام کنترل انجام دهند، 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 عددصفر در فرمت اعداد با ممیز شناور، دو مشکل اساسی دارد:

می‌دانیم اگر مانتیس برابر صفر باشد، صرف‌نظر از مقدارB 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 فوق‌العاده کوچک می‌شود، مثلاً محاسبه زیر را انسان انجام دهد، نتیجه صفر ولی Fixed – point ALU انجام دهد، نتیجه غیر صفر تولید خواهد شد.

$1-3\times\frac{1}{3}=1-1=0$ (انسان)

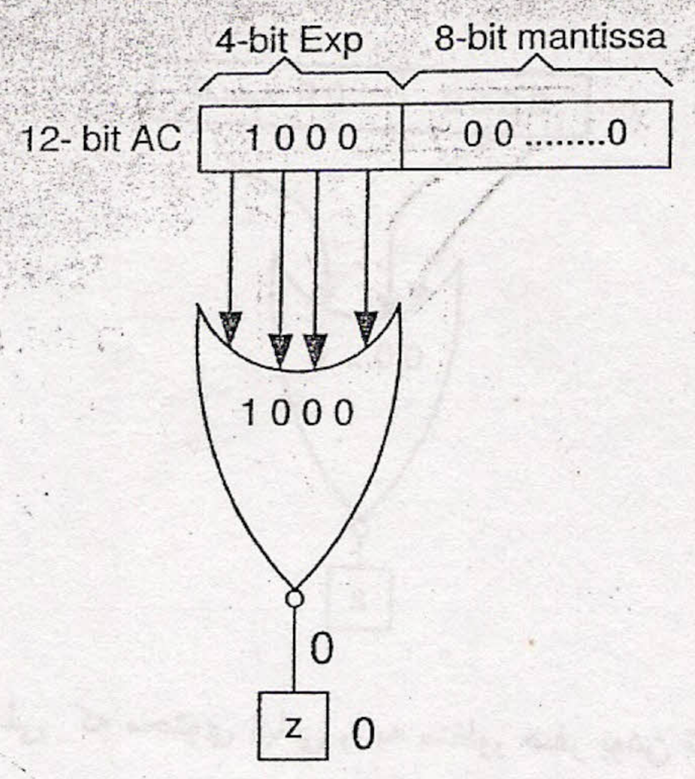
$1-3\frac{1}{3}=1-3\times(0.33...3)=1-0.99...9=0.000...01$ (کامپیوتر)

$M\times B^{E\rightarrow}$ اختصاص کوچک‌ترین عدد منفی $= \frac{M \rightarrow}{B^E \rightarrow}$ عددی فوق‌العاده کوچک / عددی فوق‌العاده بزرگ $\xrightarrow{\text{limit}} O$

برای رفع این مشکل باید به نمای عدد صفر کوچک‌ترین عدد منفی را اختصاص دهیم. در صورت اختصاص k – bit به ناحیه نما چون کوچک‌ترین عدد منفی قابل نمایش در 2'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 4 – bit Exponent کوچک‌ترین عدد منفی در 2'SComplemat 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 اریب‌دار (Biased – Exponent) استفاده شود. مثلا 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 $1\underbrace{000...0}_{k-1}$ خواهد بود و در این صورت دستورالعمل انشعابی مانند IF(AC=0) Then Go To X صحیح انجام نخواهد گرفت. روی این اصل باید از $(\text{Exess} - 2^{k-1})$ code برای نمایش E استفاده نمود که آن را «نمای اریب دار» گویند و مقدار 2^{k-1} را اریب (Biase) گویند.

مثلاً در صورتی‌که 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's Compl که مقدار دهدهی آن 8- و نمایش Binary آن 1000 می‌باشد، برای همیشه به نمای عدد صفر اختصاص دهیم.

4 – bit exp:

$-2^{4-1},..., 0,..., +(2^{4-1}-1)$

$-8,..., 0,..., +7$

2's compl: 1000 000 0111

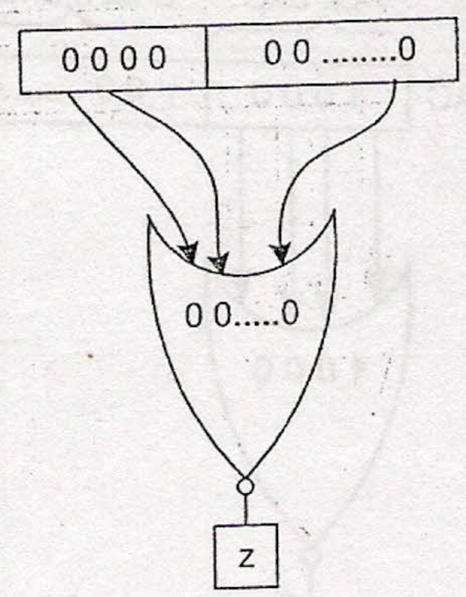
اختصاص 1000 به نمای عدد صفر با توجه به مقدار فوق، Zero – Flag را Reset و در نتیجه دستورالعمل‌های انشعاب شرطی که محتوی ثباتی را به منظور صفر بودن تست می‌کنند، مواجه با مشکل می‌شوند، به منظور رفع مشکل به‌منظور نشان دادن 8- باید از یک سیستم کدگذاری استفاده نماییم که در آن عدد 8- به صورت "0000" نشان داده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

4 – bit Exp:

2'5: $\underbrace{1000}_{-8}$,...,0000,....,0111

(x – 8) Code: $\underbrace{0000}_{-8}$,1000,...,1111

در نتیجه:



بنابراین دیگر دستورالعمل‌های انشعاب شرطی که محتوی ثباتی را به منظور صفر بودن تست می‌کنند، مواجه 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)

1–bit	k–bit	(n–k–1)–bit
sign–bit	Biase exp.	Normal, Fraction: Mantissa

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

دامنه اعداد قابل نمایش

در نمایش اعداد با ممیز ثابت به‌وسیله ثبات n بیتی $-2^{32-1}, \ldots, 0, \ldots, +\left(2^{32-1}-1\right)$ 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 $-2^{32-1}, \ldots, 0, \ldots, +\left(2^{32-1}-1\right)$ می‌باشد.

$$N=\left(1\underbrace{00\ldots0}_{\text{18 تا صفر}}\right)_{10}=100\times10^{16}=10\times10^{17}=1\times10^{18}=\cdot1\times10^{19}=\cdot01\times10^{20}=\ldots$$

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

هنجار 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 9 بیتی که در آن 3 – bit به نما و 6 – bit به مانتیس اختصاص یافته است. تنها می‌توان 6 – bit از مانتیس را نشان داد.

با فرض:

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

3-bitE	6- bit M
1 1 1	0 0 0 0 0

(روش a)

3-bit E	6-bit M
0 1 1	1 0 1 1 0 1

(روش b)

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

شرط نرمال بودن مانتیس کسری:

۱) در MSB :Sign Mag. مخالف صفر باشد. $\frac{1}{2}\le M<1=1xx...=1\times\frac{1}{2}+...$

۲) در 2's complement : باید Sign Bit و اولین بیت پشت ممیز مخالف هم باشند:

$$\begin{cases} \text{Sign Bit}=1 \quad .0xx...x \\ \text{Sign Bit}=0 \quad .1xx...x \end{cases}$$

تمرین: با استفاده از فرمت اعداد با ممیز شناور در کامپیوتر IBM، نمایش کامپیوتری 0.125×16^{5} را به‌دست آورید:

0	1 , 2 ...	7 , 8 , ... 31
1	7 – bit	24 – bit
1 – Bit	7 – Bit	24 – Bit Fraction
Sign – bit	Biased – exp	Sign – Magnitude Mantissa

عملیات ریاضی *(Arithmetic Operation)* :

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

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

بنابراین با استفاده از مدار $(X-OR)Gate, Adder$ 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 به‌وسیله 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 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 اپراندها به کار می‌رود که ما فقط Binary Adder را بررسی می‌کنیم.

Binary Adder

بنا به قضیه شانون سریع‌ترین مدار جمع کننده را می‌توان به صورت مدار دو سطحی AND – OR متناظر با عبارت بولی مجموع حاصل‌ضرب‌ها طرح نمود. ولی در عمل دو عامل محدود کننده وجود دارد:

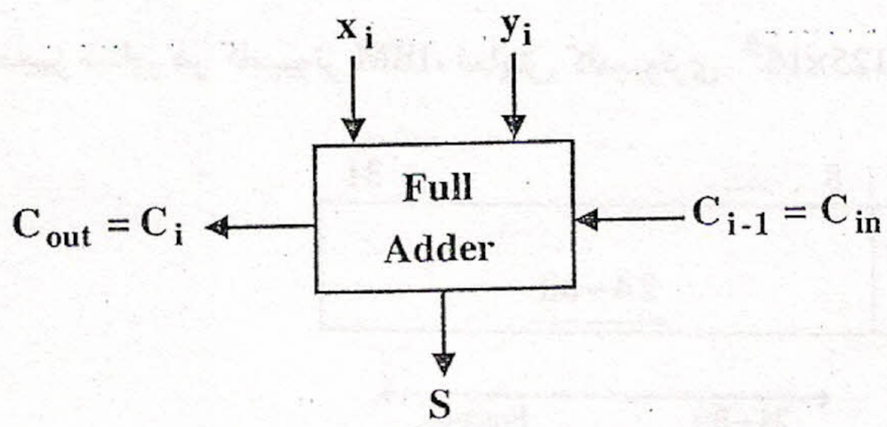
۱ـ پیدایش Fan – in

۲ـ افزایش تعداد Gateها

بنابراین باید تا اندازه‌ای از سرعت جمع کننده صرف‌نظر نمود. ابتدا مدارهای جمع کننده m بیتی $(m < n)$ را طرح نمود که در آن Fan–in به وجود نیاید 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 صورتی‌که m=1 باشد، مدار جمع کننده را Basic Binary – Adder یا 1–bit Binary–Adder گویند. که به دو صورت ترکیبی و ترتیبی می‌توان آن را پیاده‌سازی 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_i	Y_i	C_{i-1}	S_i	C_i
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

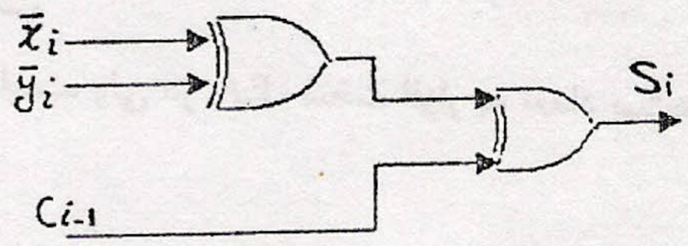
c_{i-1} \ $x_i y_i$	00	01	11	10
0		1		1
1	1		1	

$$S_1 = \sum_1 (1,2,4,7) = \bar{x}_1 y_1 \bar{c}_{-1} + x_1 \bar{y}_1 c_1 + \bar{x}_1, \bar{y}_1, c_{2-1} + x_2 y_2 c_{2-1}$$

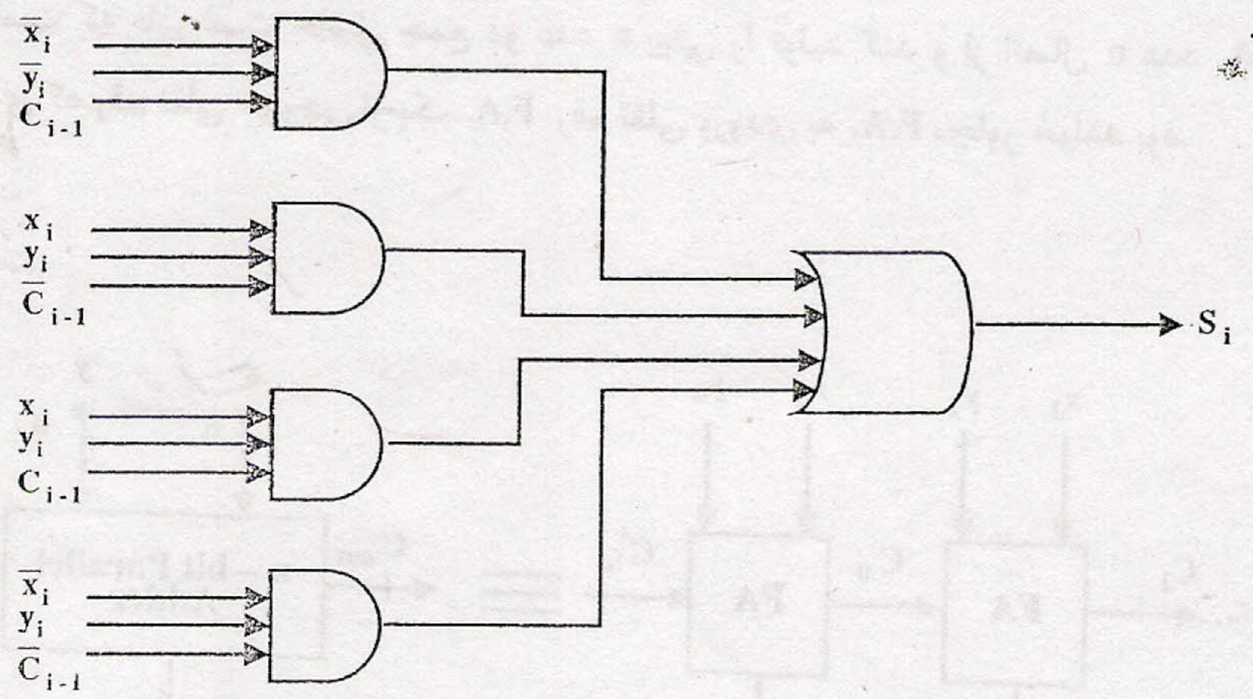
$$S_i = x_i \oplus y_i \oplus c_{i-1}$$

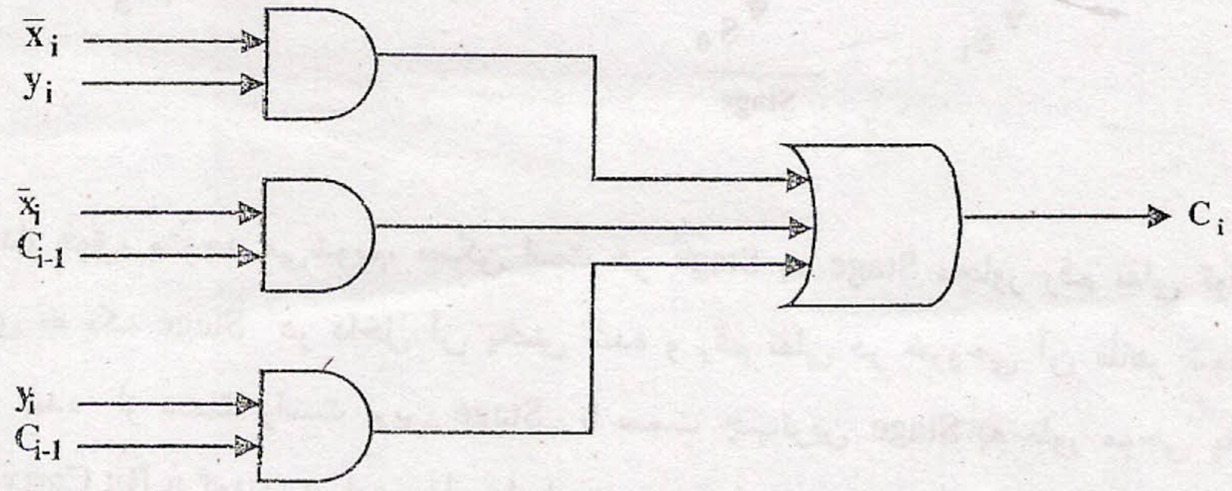
c_{i-1} \ $x_i y_i$	00	01	11	10
0	0	2	6 1	4
1	1	3 1	7 1	5 1

$$C_i = \sum_1 (3,5,6,7) = x_i y_i + x_i c_{i-1} + y_i c_{i-1}$$



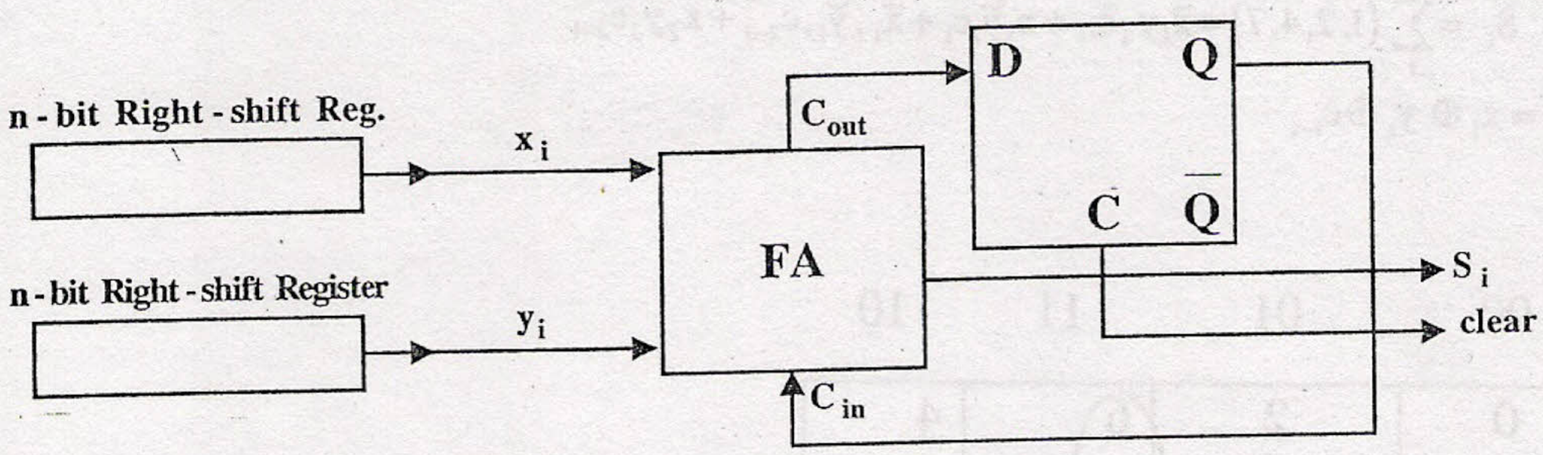
در صورتی‌که فقط از Gate های AND و OR استفاده کنیم، می‌توان داخل Full – Adder را به صورت زیر طرح نمود که به صورت مدار دوسطحی AND – OR می‌باشد.





جمع کننده سری : *1-bit Serial Adder:*

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 *Serial Adder* و *Full 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.

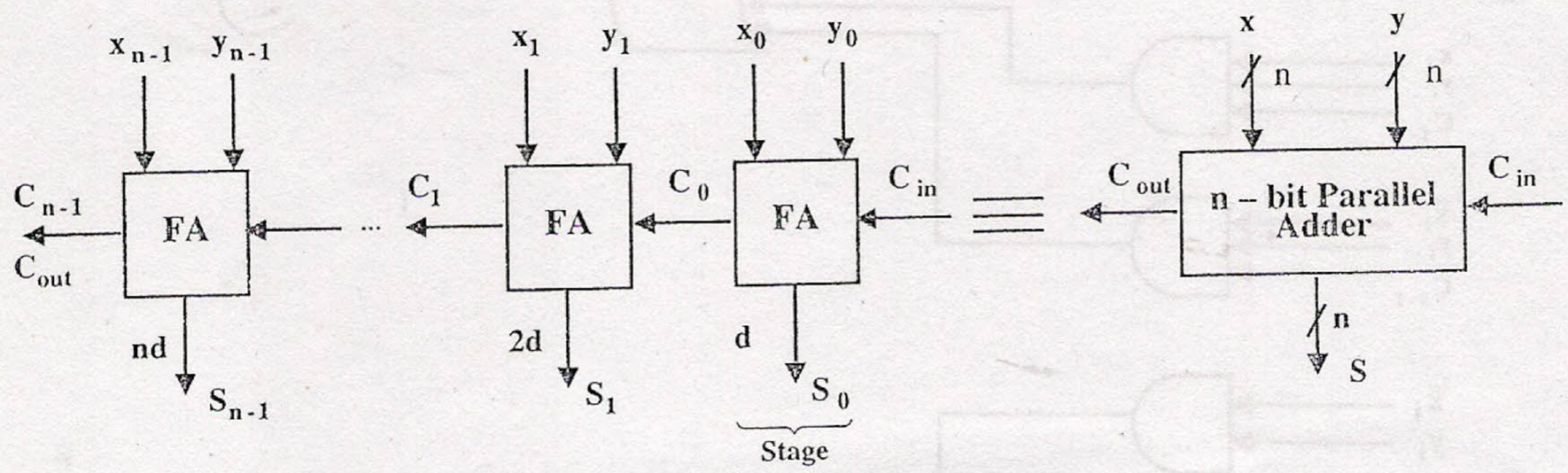
۲) در Serial Adder سخت افزار مستقل از تعداد بیت‌های اپراند ورودی است، ولی در F.A.، سخت افزار به تعداد بیت‌های اپراند ورودی بستگی خطی دارد.

جمع کننده موازی : (*Ripple Adder* یا *Parallel – 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.

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

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



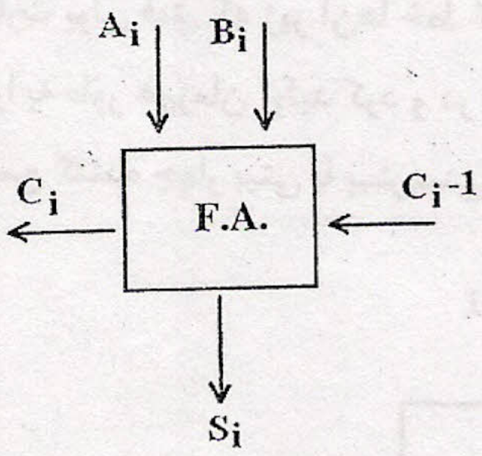
با توجه به نمودار فوق، متوجه می‌شویم، ممکن است هر Stage به Stage مجاور رقم نقلی تولید کند و هم‌چنین ممکن است رقم نقلی ورودی به یک Stage در داخل آن پخش شده و رقم نقلی در خروجی آن ظاهر شود. در بدترین حالت ممکن است رقم نقلی تولید شده از سمت راست ترین Stage تا سمت چپ‌ترین Stage به طور موجی پخش شود در این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

جمع کننده سریع: *High Speed Binary Adder*

مهم‌ترین هدف در طراحی مدار جمع کننده‌های سریع تقلیل Carry Ripple Time (زمان تلف شده برای پخش موجی ارقام نقلی است) و در صورتی‌که 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 طورهم‌زمان تولید نمود. و در نتیجه بیت‌های حاصل جمع را به صورت هم‌زمان تولید می‌شود.

جمع کننده با پیش بینی ارقام نقلی *(Carry Look ahead Adder)*:

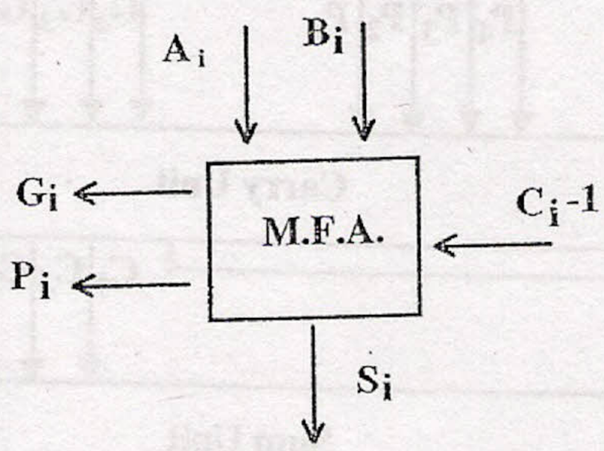
برای طراحی n-Bit Carry Lookahead Adder ابتدا رابطه بین رقم نقلی ورودی و خروجی به یک .F.A را بررسی می‌کنیم.



A_i	B_i	C_i	
O	O	O	
O	1	C_{i-1}	Propagated Carry: $P_i = A_i \oplus B_i$
1	O	C_{i-1}	
1	1	1	Generated Carry: $G_i = A_i . B_i$

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

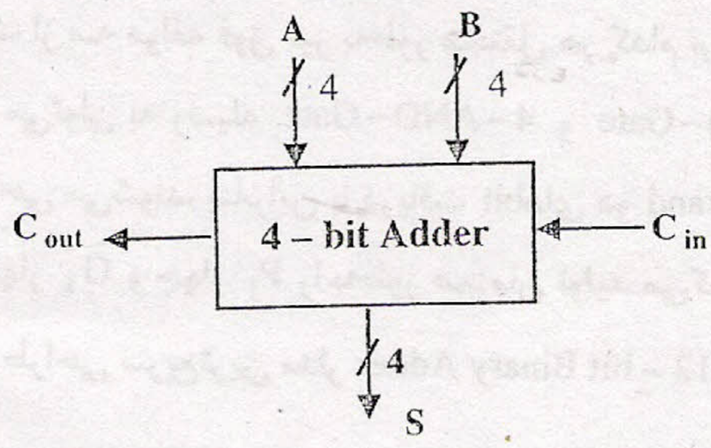
$$G_i = A_i B_i$$
$$P_i = A_i \oplus B_i$$
$$C_i = G_i + P_i C_{i-1}$$
$$S_i = P_i \oplus C_{i-1}$$



مثال: مطلوب است طراحی 4 – Bit Carry Lookahead Adder

$$C_4 \leftarrow C_3\ C_2\ C_1\ C_{in}$$
$$A = A_4\ A_3\ A_2\ A_1$$
$$B = B_4\ B_3\ B_2\ B_1$$
$$S = S_4\ S_3\ S_2\ S_1$$

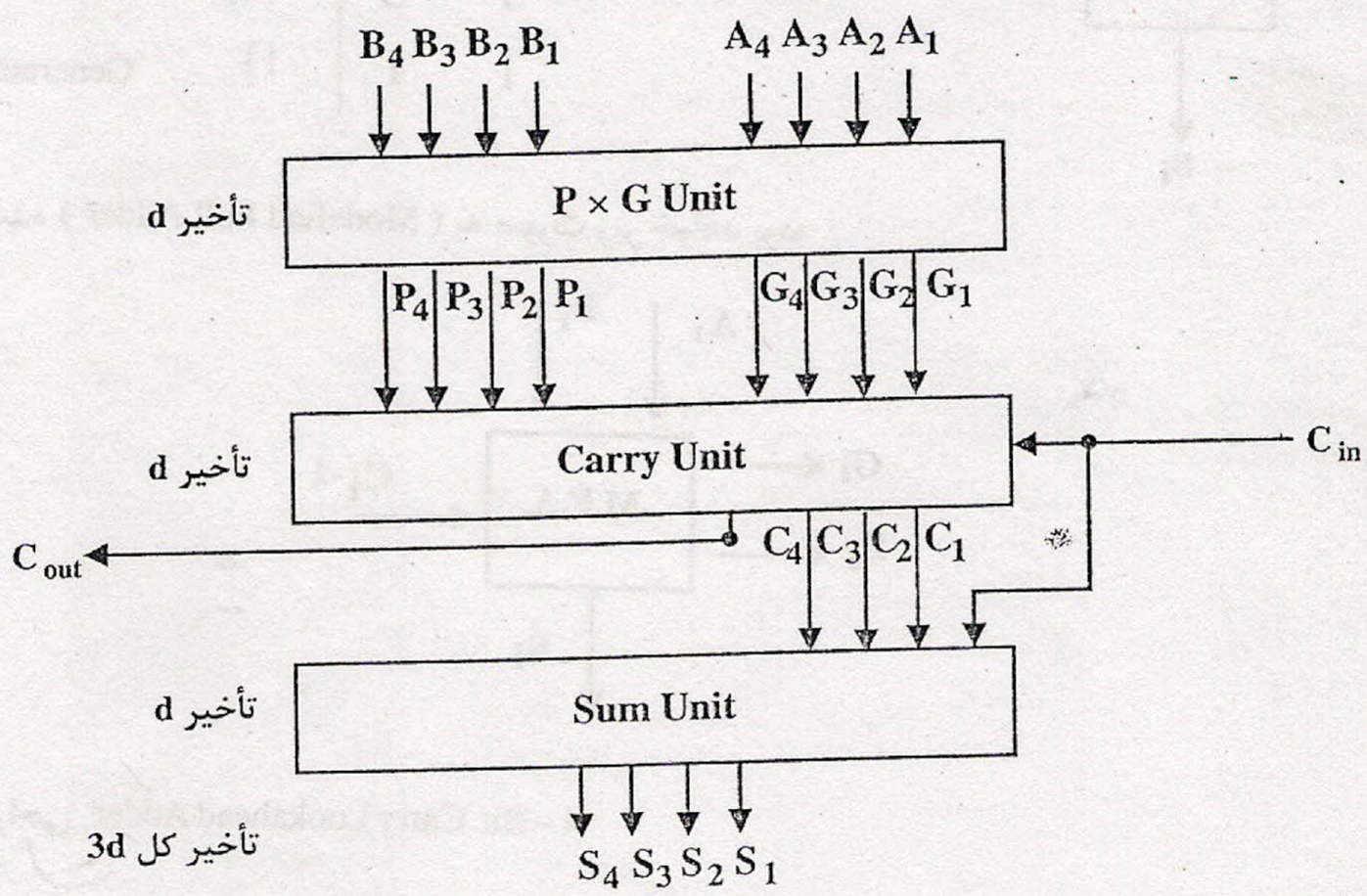
$$\begin{cases} C_1 = G_1 + C_{in} P_1 \\ C_2 = G_2 + C_1 P_2 \\ C_3 = G_3 + G_2 P_3 \\ C_4 = G_4 + C_3 P_4 \end{cases} \qquad \begin{cases} S = P \oplus C_{in} \\ S_2 = P_2 \oplus C_1 \\ S_3 = P_3 \oplus C_2 \\ S_4 = P_4 \oplus C_3 \end{cases}$$



با توجه به عبارات فوق متوجه می‌شویم که عبارت بولی C_i ها به طور بازگشتی تعریف شده‌اند. در صورتی‌که 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_iها به طور هم‌زمان در دست هستند، بنابراین می‌توان ۴ بیت حاصل‌جمع را به طور هم‌زمان تولید نمود.

$$C_1 = \underline{G_1 + C_{in}P_1}$$
$$C_2 = G_2 + (G_1 + C_{in}P)P_2 = \underline{G_2 + G_1P_2 + C_{in}P_1P_2}$$
$$C_3 = G_3 + (G_2 + G_1P_1 + C_{in}P_1P_2)P_3 = \underline{G_3 + G_2P_3 + G_1P_2P_3 + C_{in}P_1P_2P_3}$$
$$C_4 = \underline{G_4 + G_3P_4 + G_2P_3P_4 + G_1P_2P_3P_4 + C_{in}P_1P_2P_3P_4}$$

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 4 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 4 – bit Carry – look ahead Adder

مدار فوق به‌طور 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 NAND و یا تمام NOR طراحی می‌شوند، بنابراین با دریافت bitهای دو Operand ورودی، بعد از تاخیر d واحد زمان (d تاخیر 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.