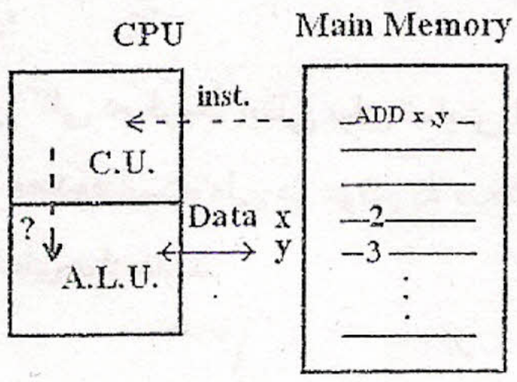
فصل پنجم

شیوه نمایش اطلاعات

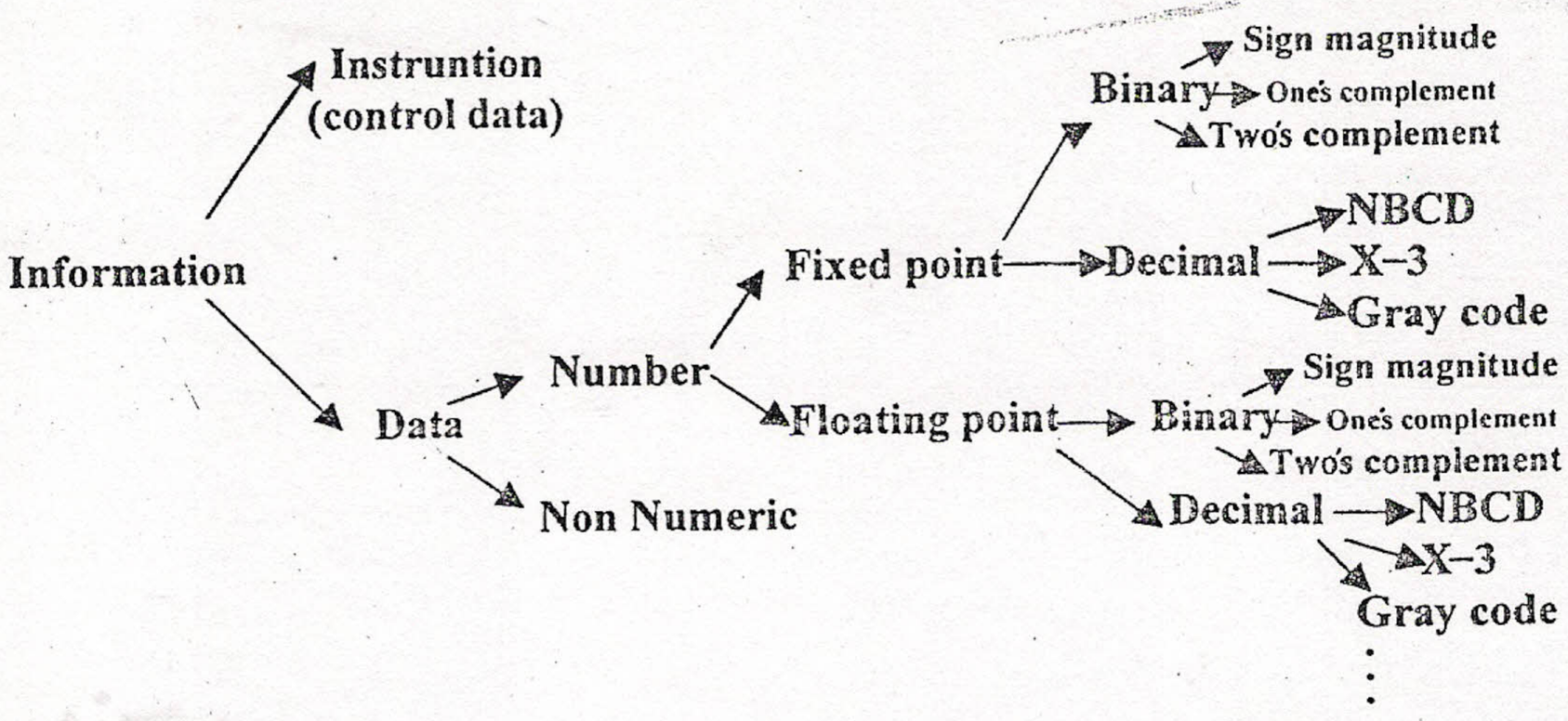
نمایش اطلاعات *Information Representation*

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.

Most significant–bit Least significant–bit

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

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.

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

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:

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

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.

III) 2's Complement :

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:

استفاده نمود.

Binary	r=2	2's Complement
Octal	r=8	8's Complement
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 $N=.0110$

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

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

$$=1-0.0110$$

$$=0.1010$$

```
0 11 20
1.0000  -
 .0110
------
 .1010
```

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 :

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

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

$$=0.1001+0.0001=0.1010$$

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:

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

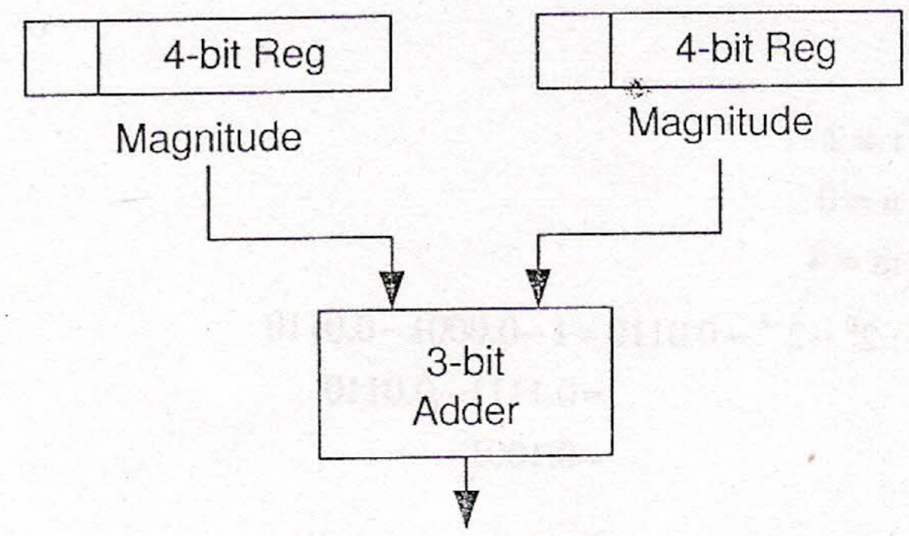
می‌توان با استفاده از (X-OR) Gate تولید نمود.

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 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.

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

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

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.

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

به رقم نقلی تولید شده از جمع مرحله اول End Around Carry (E.A.C.) (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:

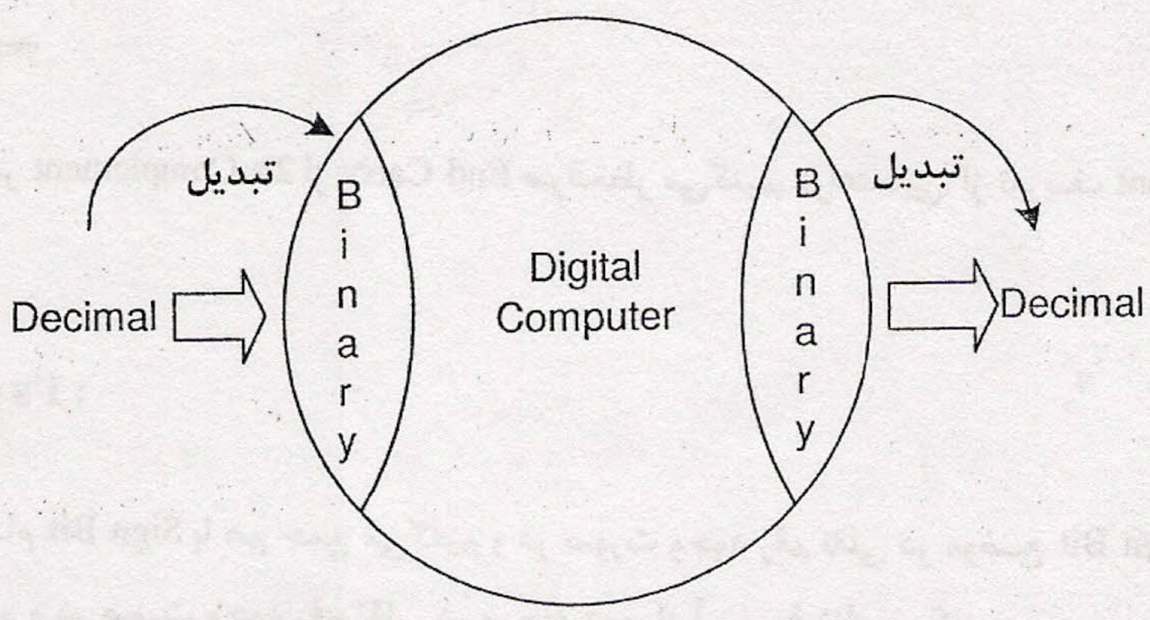
سیستم‌های رقمی با سیگنال‌هایی محدود به دو مقدار ممکن و عناصری با دو حالت پایدار کار می‌کنند، بنابراین یک تناظر یک به یک بین سیگنال‌های دودویی و ارقام دودویی وجود دارد.

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:

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

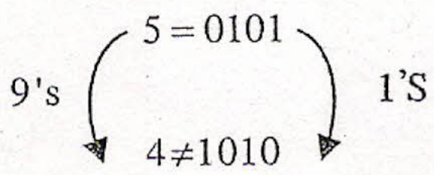
دو عیب عمده NBCD عبارتند از:

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	
	4	

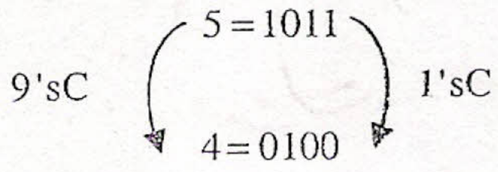
۲) 9's Complement هر رقم NBCD را نمی‌توان از تبدیل صفرها به یک و یک‌ها به صفر به دست آورد، بنابراین NBCD یک کد خود متمم (Self – Complementary) نیست.



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's	5 / 4 =	1000 / 0111
+9	1100			1's C
14	1 0100			
	+0011			
	1 0111			

Self Complementory



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:

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

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

مثال: فرض کنید داشته باشیم:

$$\overset{b_n}{(}11001010\overset{b_2 b_1}{110})_2 = (?)_{Gray} = (g_n g_{n-1}...g_1)_{Gray}$$

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

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

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

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

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

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 Complement یک عدد می‌توان با متمم‌گیری از سمت چپ‌ترین بیت در گروه‌های 4 بیتی استفاده نمود.

x – 3 Gray – code

235	0111	0101	1100
9's 764	1111 (7)	1101 (6)	0100 (4)