• مبناهای عددی و تبدیل مبنا

در حالت کلی یک عدد مانند A در مبنای r بصورت زیر نوشته می‌شود

$$A = A_{n-1}A_{n-2} \dots A_1A_0.A_{-1}A_{-2} \dots A_{-m} = \sum_{i=-m}^{n-1} A_i \times r^i$$

در این حالت عدد A دارای n رقم صحیح و m رقم ناصحیح (اعشار) می‌باشد و هر رقم با توجه به جایگاه خودش در توانی از مبنا (radix) ضرب می‌گردد.

مثال عدد 243.19 در مبنای 10

$$243.19 = 9 \times 10^{-2} + 1 \times 10^{-1} + 3 \times 10^{0} + 4 \times 10^{1} + 2 \times 10^{2} = 0.09 + 0.1 + 3 + 40 + 200$$

اگر عددی در مبنای r باشد آنگاه ارقام آن عدد بین 0 و r-1 خواهد بود. بطور مثال در مبنای 4 ارقام بین 0 تا 3 می‌باشد.

مثال عدد 23.56 از مبنای 7 به مبنای 10 تبدیل نمایید

$$(23.56)_7 = 6 \times 7^{-2} + 5 \times 7^{-1} + 3 \times 7^{0} + 2 \times 7^{1} = \frac{6}{49} + \frac{5}{7} + 3 + 2 \times 7 \approx (17.83)_{10}$$

برای تبدیل از هر مبنا به مبنای دیگر می‌توانیم از مبنای 10 به عنوان واسطه استفاده کنیم.

نکـــته: مبنا معادل انگلیسی radix بوده و در برخی منابع از واژه پایه معادل base استفاده می‌شود که تفاوتی ندارد

مثال عدد 23 را از مبنای 4 به مبنای 5 تبدیل نمایید

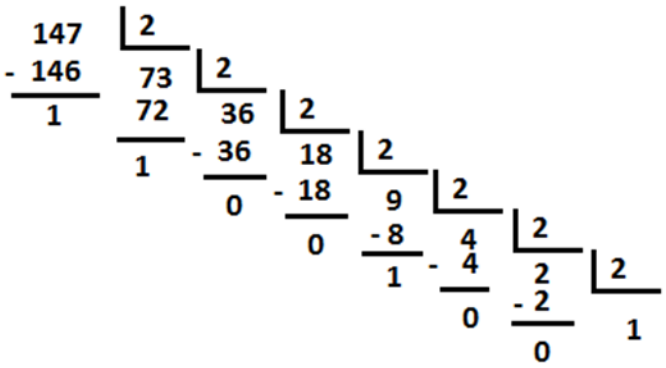
$$(23)_4 = 3 \times 4^{0} + 2 \times 4^{1} = 3 + 8 = 11 = 1 + 10 = 1 + (2 \times 5)$$

$$= 1 \times 5^{0} + 2 \times 5^{1} = (21)_5$$

در مبنای 2 ارقام 0 و 1 هستند. عددی که در مبنای 2 باشد را باینری گویند. برای تبدیل یک عدد از مبنای 10 به مبنای 2 دو روش وجود دارد در روش اول عدد ورودی را بر 2 تقسیم می‌کنیم و باقی‌مانده را محاسبه می‌کنیم سپس خارج قسمت حاصل از این تقسیم را مجددا بر 2 تقسیم می نماییم و باقی‌مانده را حساب

می‌کنیم این کار را تا جایی ادامه می‌دهیم که خارج قسمت برابر با 1 گردد. سپس خارج قسمت نهایی (عدد 1) را در سمت چپ قرار داده و باقی مانده‌ها را از آخر به اول به ترتیب در سمت راست آن قرار می دهیم مانند مثال زیر

147



10010011

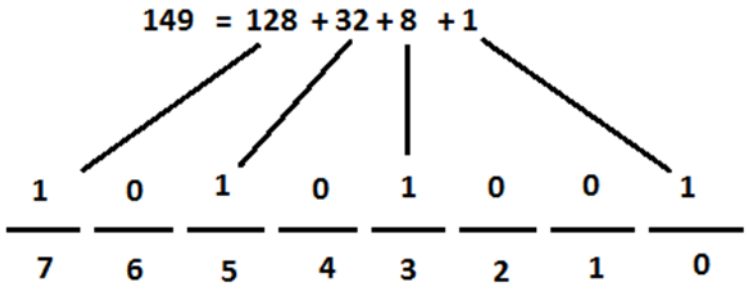
در کنار روش تقسیمات متوالی روش هوشمندانه حدس زدن نیز وجود دارد برای این روش باید توان‌های 2 را تا توان 10 حفظ باشیم

n	0	1	2	3	4	5	6	7	8	9	10
2^n	1	2	4	8	16	32	64	128	256	512	1024

برای هر عدد ورودی ابتدا حساب می‌کنیم بین کدام دو توان قرار می‌گیرد و از عدد ورودی مقدار 2 به توان پایینتر را کم می کنیم و عدد 1 را در جایگاه متناظر با آن توان قرار می دهیم سپس دوباره بررسی می کنیم تا ببینیم بین کدام دو توان قرار می گیرد و دوباره توان پایینتر را کم می کنیم و عدد 1 را در جایگاه متناظر با آن توان قرار می دهیم. به این ترتیب عدد باینری بدست می‌آید بعد از تبدیل بهتر است نتیجه را بیازماییم.

مثال عدد 169 در مبنای 10 را به مبنای 2 تبدیل نمایید.

ابتدا تعدادی خط رسم می‌کنیم و زیر آن اعداد 0 الی 10 (در این مثال تا 7 کافی است) را قرار می‌دهیم هر جایگاه معادل توانی از عدد 2 می‌باشد. سپس توجه می کنیم که عدد 169 بین 128 و 256 قرار دارد پس حتما عدد 128 یا 2 به توان 7 در عدد 169 وجود دارد در محل توان 7 عدد 1 را درج می‌کنیم پس از کم کردن 128 از 169 به عدد 41 می‌رسیم که بین 32 و 64 قرار دارد عدد 32 معادل توان 5 عدد 2 است لذا در محل متناظر با توان 5 عدد 1 قرار می‌دهیم و از عدد 41 مقدار 32 را کم می‌کنیم و به عدد 9 می‌رسیم عدد 9 بین 8 و 16 قرار دارد پس عدد 8 که توان 3 عدد 2 می‌باشد در عدد 9 وجود دارد لذا در محل متناظر با توان 3 عدد 1 را قرار می‌دهیم و از عدد 9 مقدار 8 را کم می‌کنیم و به عدد 1 می‌رسیم که معادل توان صفر عدد 2 می باشد لذا در محل متناظر با توان صفر عدد 1 قرار می‌دهیم. در بقیه جاهای خالی عدد 0 را درج می‌کنیم.



در واقع در روش دوم یک عدد را بصورت جمعی از توانهای 2 می نویسیم و سپس به ازای هر قسمت عدد 1 را در جایگاه متناظر قرار می‌دهیم. این روش سریعتر است و کمتر در معرض اشتباه قرار دارد زیرا نتیجه را می‌توان به سرعت با تبدیل معکوس چک کرد.

برای تبدیل از مبنای 2 به مبنای 10 کافی است مطابق فرمول اول فصل اعداد متناطر با توان‌های عدد 2 را باهم جمع بزنیم

مثال عدد 110101011 را از مبنای 2 به مبنای 10 تبدیل نمایید

$$(110101011)_2 = 2^8 + 2^7 + 2^5 + 2^3 + 2^1 + 2^0 = 256 + 128 + 32 + 8 + 2 + 1 = (427)_{10}$$

$$\underset{8}{1}\ \underset{7}{1}\ \underset{6}{0}\ \underset{5}{1}\ \underset{4}{0}\ \underset{3}{1}\ \underset{2}{0}\ \underset{1}{1}\ \underset{0}{1}$$

$$256 + 128 + 32 + 8 + 2 + 1$$

برای تبدیل اعداد اعشاری از مبنای 10 به مبنای 2 ابتدا قسمت صحیح را به یکی از روشهای قبل تبدیل می‌کنیم و برای قسمت اعشاری آن را در عدد 2 ضرب می‌کنیم سپس قسمت صحیح حاصل از عمل ضرب را یادداشت نموده و قسمت اعشار آن را به مرحله بعد منتقل می‌کنیم این عملیات ادامه می‌یابد در برخی از اعداد با رسیدن به عدد صحیح 1 خاتمه می‌یابد در برخی اعداد یک الگوی تکرار پذیر از صفر و یک در قسمت صحیح حاصلضرب تولید می‌شود که در صورت کامل شدن الگو عملیات متوقف می‌شود زیرا در صورت ادامه همان الگو تکرار می‌شود و در مورد برخی اعداد هرگز به عدد صحیح 1 و یا الگوی تکرارپذیر نخواهیم رسید در این مورد به دلخواه و با رسیدن به دقت لازم عملیات را متوقف می‌کنیم. پس از توقف اعداد صحیح را از اولین تا آخرین در سمت راست ممیز قرار می‌دهیم.

در مثال زیر سمت چپ عدد خاتمه پذیر و در سمت راست یک الگوی تکرار شونده 1001 بعد از اولین صفر تولید شد. می‌توانیم یکبار الگو را نوشته و بالای آن خط بکشیم (به معنای تکرار).

نکـــته: اگر قسمت اعشاری (صرف نظر از اعشار) توانی از عدد 5 باشد، آنگاه قطعاً عدد اعشاری در تبدیل به مبنای 2 خاتمه‌پذیر خواهد بود.

12.625

0.625 x 2 = 1.25
0.25 x 2 = 0.5
0.5 x 2 = 1.0

1100.101

12 = 8 + 4
1100

0.3

0.3 x 2 = 0.6
0.6 x 2 = 1.2
0.2 x 2 = 0.4
0.4 x 2 = 0.8
0.8 x 2 = 1.6
0.6 x 2 = 1.2
0.2 x 2 = 0.4
0.4 x 2 = 0.8
0.8 x 2 = 1.6
0.6 x 2 = 1.2
.
.
.

0.0100110011001 1001 ...

$0.0\overline{01001}$

نکـــته: بزرگترین عدد n رقمی در مبنای r برابر است با $r^n - 1$ بطور مثال بزرگترین عدد 3 رقمی در مبنای 10 برابر است با 999 که همان $10^3 - 1 = 1000 - 1 = 999$ است در هر مبنایی که باشیم بزرگترین عدد n رقمی برابر است با تکرار n بار عدد r-1 مثلا در مبنای 5 بزرگترین عدد 8 رقمی برابر است با 44444444 و یک عدد بزرگتر از آن برابر است با اولین عدد n+1 رقمی یا همان r^n

نکـــته: در تمام مبناها عدد r^n بصورت $1\underbrace{000\ldots000}_{n}$ نوشته می‌شود که تعداد صفرها برابر است با n و ارتباطی با مبنا ندارد.

مبنای دو (باینری) در رایانه بسیار مهم است به دلیل آنکه در مبنای دو فقط ارقام صفر و یک داریم و هر عدد یا داده‌ای را با همین دو رقم نشان می‌دهیم. همچنین عملکـــرد مدارات اصلی رایانه بر اساس روشن و خاموش بودن ترانزیستورها می‌باشد اگر وضعیت روشن بودن ترانزیستور را معادل عدد یک، و وضعیت خاموش بودن ترانزیستور را معادل عدد صفر در نظر بگیریم آنگاه کلیه داده‌های تشکیل شده از صفر و یک را می‌توانیم با این مدارات پردازش کنیم. مبناهایی که توانی از 2 هستند نیز مهم به شمار می‌آیند مانند مبنای 4 ، 8 و 16. به مبنای 4 کواترنری به مبنای 8 اکتال و به مبنای 16 هگزادســـیمال می‌گویند.

نکـــته: به مبنای 4 کواترنری (Quaternary) به مبنای 8 اکتال (Octal) و به مبنای 16 هگزادســـیمال (Hexadecimal) می‌گویند.

در مبنای 16 (هگـــزادسیمال) به دلیل آنکه ارقام بین 0 تا 15 هستند برای نمایش رقم‌های بزرگتر از 9 از کاراکترهای انگلیسی A الی F استفاده می‌کنیم. (A=10,B=11,C=12,D=13,E=14,F=15)

برای تبدیل از مبنای 2 (باینری) به 16 ارقام صحیح را از محل ممیز به سمت چپ 4 تا 4 تا دسته‌بندی می‌کنیم و مطابق جدول مقادیر متناظر با هر 4 بیت را نمایش می‌دهیم. در خصوص قسمت ناصحیح (اعشاری) از محل ممیز به سمت راست ارقام را 4 تا 4 تا دسته‌بندی می‌کنیم و مطابق جدول مقادیر متناظر با هر 4 بیت را نمایش می‌دهیم. اگر در سمت اعداد صحیح تعداد به 4 نرسید از سمت چپ صفر اضافه می‌کنیم تا یک دسته 4 تایی

ایجاد شود. اگر در قسمت ناصحیح تعداد به 4 نرسید از سمت راست صفر اضافه می‌کنیم تا یک دسته 4 تایی ایجاد شود.

0	1	2	3	4	5	6	7
0000	0001	0010	0011	0100	0101	0110	0111

8	9	A	B	C	D	E	F
1000	1001	1010	1011	1100	1101	1110	1111

1101 0101 . 1110 1100
D 5 E C

D5.EC

برای تبدیل از مبنای 2 (باینری) به مبنای 8 (اکـــتال) ارقام صحیح را از محل ممیز به سمت چپ 3 تا 3 تا دسته‌بندی می‌کنیم و مطابق جدول مقادیر متناظر با هر 3 بیت را نمایش می‌دهیم. در خصوص قسمت ناصحیح (اعشاری) از محل ممیز به سمت راست ارقام را 3 تا 3 تا دسته‌بندی می‌کنیم و مطابق جدول مقادیر متناظر با هر 3 بیت را نمایش می‌دهیم. اگر در سمت اعداد صحیح تعداد به 3 نرسید از سمت چپ صفر اضافه می‌کنیم تا یک دسته 3 تایی ایجاد شود. اگر در قسمت ناصحیح تعداد به 3 نرسید از سمت راست صفر اضافه می‌کنیم تا یک دسته 3 تایی ایجاد شود. در مورد تبدیل از مبنای 2 به 4 نیز روش به همین صورت است ولی دسته ها دو تا دو تا هستند. در تبدیل برعکس از مبنای 16 به مبنای 2 نیز به ازای هر رقم رشته4 بیتی متناظر با آن را قرار می‌دهیم. در تبدیل از مبنای 8 به مبنای 2 نیز همین روش استفاده می‌شود به ازای هر رقم از مبنای 8 یک رشته 3 بیتی متناظر با آن عدد را قرار می‌دهیم.

0	1	2	3	4	5	6	7
000	001	010	011	100	101	110	111

001 101 011 101 . 111 010
1 5 3 5 7 2

مثال: عدد 7E4D.2A را از مبنای 16 (هگـــزادســـیمال) به مبنای 2 (باینری) تبدیل نمایید

$$7E4D.2A = \underbrace{0111}_{7}\ \underbrace{1110}_{E}\ \underbrace{0100}_{4}\ \underbrace{1101}_{D}\ .\ \underbrace{0010}_{2}\ \underbrace{1010}_{A} = (111111001001101.0010101)_2$$

پس از تبدیل نهایی صفرهای اضافی سمت راست و چپ حذف می‌شوند

مثال: عدد 75.32 را از مبنای مبنای 8 (اکـتال) به مبنای 2 (باینری) تبدیل نمایید

$$75.32 = \underbrace{111}_{7}\ \underbrace{101}_{5}\ .\ \underbrace{011}_{3}\ \underbrace{010}_{2} = (111101.01101)_2$$

نکـــته: در هنگام تبدیل از مبنای 16 به 8 یا بالعکس ابتدا به مبنای 2 منتقل می کنیم سپس دوباره دسته بندی می‌کنیم

مثال: عدد A4.25 را از مبنای 16 (هگـــزادســـیمال) به مبنای 8 (اکتال) تبدیل نمایید

$$A4.25 = \underbrace{1010}_{A}\ \underbrace{0100}_{4}.\ \underbrace{0010}_{2}\ \underbrace{0101}_{5} = (10100100.00100101)_2$$

$$= \underbrace{010}_{2}\ \underbrace{100}_{4}\ \underbrace{100}_{4}\ .\ \underbrace{001}_{1}\ \underbrace{001}_{1}\ \underbrace{010}_{2} = (244.112)_8$$

نکـــته: روش تبدیل بین مبنای 9 و 3 نیز به همین ترتیب می‌باشد معادل هر رقم در مبنای 9 یک عدد دو رقمی در مبنای 3 وجود دارد

$$(00)_3 = (0)_9\ ,(01)_3 = (1)_9, (02)_3 = (2)_9, (10)_3 = (3)_9, (11)_3 = (4)_9,$$

$$(12)_3 = (5)_9, (20)_3 = (6)_9, (21)_3 = (7)_9, (22)_3 = (8)_9$$

مثال: عدد 48.15 را از مبنای 9 به مبنای 3 تبدیل نمایید

$$48.15 = \underbrace{11}_{4}\ \underbrace{22}_{8}.\ \underbrace{01}_{1}\ \underbrace{12}_{5} = (1122.0112)_3$$

با در اختیار داشتن n رقم در مبنای 2 می‌توانیم اعداد صفر تا $2^n - 1$ را در مبنای 10 نمایش دهیم. هر رقم مبنای 2 را یک بیت داده می‌نامیم. لذا با یک بیت فقط اعداد 0 و 1 با دو بیت اعداد 0 الی 3 و با سه بیت اعداد 0 الی 7 را می‌توانیم بصورت باینری بنویسیم.

نکـــته: برای آنکه بتوانیم عدد X در مبنای 10 را در مبنای 2 نمایش دهیم نیاز به n بیت (رقم) داریم که در رابطه زیر صدق نماید. البته بعد از حصول نتیجه باید کنترل کنیم که $2^n > X$ باشد. یک روش ساده‌تر برای این کار آنست که تعداد ارقام X را در 10 ضرب کرده و بر 3 تقسیم کنیم آنگاه عدد بدست آمده را به سمت بالا روند کنیم و نتیجه را با رابطه $2^n > X$ آزمایش کنیم.

$$n = 1 + \lfloor \log_2 X \rfloor = 1 + \left\lfloor \frac{\log_{10} X}{\log_{10} 2} \right\rfloor \cong 1 + \left\lfloor \frac{\log_{10} X}{0.3} \right\rfloor$$

- **مکمل اعداد (مکمل 10، مکمل 2، مکمل 1)**

برای رسیدن به درک درستی از کاربرد مکمل اعداد فرض کنید می‌خواهیم با استفاده از عملیات جمع تک رقمی عملیات تفریق را شبیه‌سازی کنیم برای اینکار بجای عملیات A-B عملیات A+Complement(B) را انجام می‌دهیم که Complement(B) همان مکمل عدد B است. مکمل یعنی کامل کننده و به این معنی است که عدد B با کدام عدد اگر جمع شود اولین عدد دو رقمی یعنی عدد 10 را ایجاد می‌کند. لذا باید رابطه B+Complement(B) = 10 صدق نماید در نتیجه Complement(B) = 10 – B اکنون به جای B– مکمل عدد B یعنی 10-B را قرار می‌دهیم لذا به جای A-B خواهیم داشت A + 10 – B حاصل این عملیات به لحاظ ریاضی A-B و یک رقم نقلی خواهد بود که خود ما در ایجاد آن نقش داشتیم و می‌توانیم از آن صرف نظر کنیم. به یک مثال توجه کنید.

$$5\text{-}3 = 5 + \text{Complement}(3) = 5 + (10\text{-}3) = 5 + 7 = 12 = \cancel{10} + 2 = 2$$

در مثال فوق پس از انجام عملیات رقم نقلی (عدد 10) را دور می‌ریزیم و به عدد 2 می‌رسیم که جواب درست است.

تعریف مکمل عدد: مکمل r یک عدد مانند A در مبنای r عبارتست از $r^n - A$ که در این رابطه n تعداد ارقام صحیح عدد A می‌باشد. مکمل r را در زبان انگلیسی radix complement گویند.

تعریف مکمل کاهش یافته عدد: مکمل r-1 یک عدد مانند A در مبنای r عبارتست از $r^n - r^{-m} - A$ که در این رابطه n تعداد ارقام صحیح عدد A و m تعداد ارقام ناصحیح (اعشاری) عدد A می‌باشد. مکمل r-1 را مکمل کاهش‌یافته یا diminished radix complement گویند.

مثال: مکمل 5 عدد 3412 در مبنای 5 را بدست آورید

$$5^{4} - 3412$$

$$\begin{array}{r} 4\,4\,4 \\ 0\,5\,5\,5\,5 \\ 1\,0\,0\,0\,0 \\ -\;\;3\,4\,1\,2 \\ \hline 1\,0\,3\,3 \end{array}$$

مثال: مکمل 6 عدد 5624.14 در مبنای 7 را بدست آورید

$$7^{4} - 7^{-2} - 5624.14$$

$$\begin{array}{r} 6\,6\,6\,6\,6 \\ 0\,7\,7\,7\,7\,7\,7 \\ 1\,0\,0\,0\,0\,0.0\,0 \\ -\;\;0\,0\,0\,0\,0.0\,1 \\ \hline 6\,6\,6\,6.6\,6 \end{array} \qquad \begin{array}{r} 6\,6\,6\,6.6\,6 \\ -\;5\,6\,2\,4.1\,4 \\ \hline 1\,0\,4\,2.5\,2 \end{array}$$

نکـــته: برای آنکه مکمل r یک عدد در مبنای r را محاسبه کنیم لازم است هر یک از ارقام آن را از عدد r-1 کم کنیم و کوچکترین عدد را از r کم کنیم. برای آنکه مکمل کاهش‌یافته یک عدد را محاسبه نماییم کافی است هر یک از ارقام آن را از عدد r-1 کم کنیم.

مکمل 1: در مبنای 2 مکمل کاهش‌یافته بصورت r-1 = 2-1 مکمل 1 نامیده می‌شود و روش محاسبه آن به این صورت است که هریک از ارقام را معکوس می‌کنیم. در واقع از عدد 1 کم می‌کنیم اگر 1 را از 1 کم کنیم به صفر می‌رسیم و اگر صفر را از یک کم کنیم به یک می‌رسیم پس در واقع با معکوس کردن بیت‌ها می‌توانیم به مکمل 1 برسیم.

مثال: مکمل 1 عدد 10110101101110 در مبنای 2 را بدست آورید

$$10110101101110 \quad 1's complement = 01001010010001$$

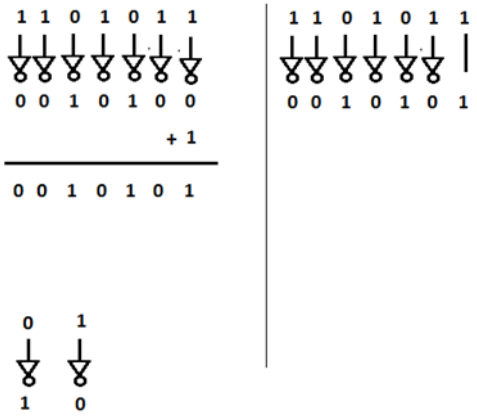
و مکمل 2 : در مبنای باینری مکمل 2 به دو روش محاسبه می‌شود

روش اول: با توجه به اینکه عدد ورودی صحیح می‌باشد لذا تعداد ارقام اعشار برابر با صفر است m=0 و رابطه مکمل کاهش یافته $r^n - r^{-m} - A$ بصورت $2^n - 2^0 - A = 2^n - 1 - A$ می‌باشد که با مقایسه با رابطه

مکمل 2 که بصورت $2^n - A$ می‌توان نتیجه گرفت که اگر به مکمل 1 عدد A یک واحد اضافه کنیم مکمل 2 بدست می‌آید پس روش اول به این صورت است که ابتدا کلیه بیت‌ها را معکوس می‌کنیم (تا مکمل 1 بدست آید) سپس حاصل را با عدد 1 جمع می‌کنیم.

روش دوم: از سمت راست حرکت کرده تا به اولین 1 برسیم تا اولین 1 را کاری نداریم از آن به بعد بیتها را معکوس می‌کنیم.

مثال: مکمل 2 عدد 1101011 در مبنای 2 را بدست آورید



- **نمایش اعداد علامت‌دار**

در حالت کلی سه روش برای نمایش اعداد علامت‌دار وجود دارد. روش اول «علامت-مقدار» sign-value روش دوم مکمل 1 (1's complement) و روش سوم مکمل 2 (2's complement) نام دارد. روش اول از لحاظ تبدیل بسیار راحت است زیرا برای آنکه عددی را منفی کنیم کافی است بیت (رقم) پرارزش (سمت چپ) را 1 قرار دهیم. روش مکمل 1 نیز از لحاظ تبدیل ساده است زیرا برای آنکه عددی را به روش مکمل 1 منفی کنیم باید بیت‌های آن عدد را معکوس کنیم. روش مکمل 2 نیز مانند روش مکمل 1 است با این تفاوت که پس از معکوس کردن بیت ها عدد بدست آمده را باید با عدد 1 جمع کنیم. روش علامت مقدار به لحاظ محاسباتی هیچ مزیتی ندارد ولی دو روش دیگر مزیت محاسباتی دارند. وجود صفر منفی در روش‌های علامت-مقدار و مکمل 1 یک نقطه ضعف است. به بیت سمت چپ در نمایش اعداد علامت‌دار بیت علامت می‌گویند.

نکـــته: به آخرین بیت (رقم) سمت چپ پرارزشترین بیت (most significant bit MSB) گویند و به اولین بیت از سمت راست کم ارزشترین بیت (least significant bit LSB) گویند.

عدد	بدون علامت	روش علامت مقدار	روش مکمل ۱	روش مکمل ۲
000	0	0	0	0
001	1	1	1	1
010	2	2	2	2
011	3	3	3	3
100	4	-0	-3	-4
101	5	-1	-2	-3
110	6	-2	-1	-2
111	7	-3	-0	-1

نکـــته: در روش مکمل 1 بزرگترین عدد n بیتی $2^{n-1}-1$ و کوچکـترین عدد $-2^{n-1}+1$ می‌باشد. در روش مکمل 2 بزرگترین عدد n بیتی $2^{n-1}-1$ و کوچکـترین عدد -2^{n-1} می‌باشد.

- **محاسبات در روش مکمل**

در محاسبات به روش مکمل در حالت کلی دو دسته محاسبات وجود دارد. دسته اول محاسبات بدون علامت (unsigned) می‌باشد. و دسته دوم محاسبات signed می‌باشد.

در روش محاسبات بدون علامت عملیات جمع به همان روش عادی انجام می‌گردد و در خصوص عملیات تفریق عملوند دوم (مفروق/کاسته) را به روش مکمل منفی می‌کنیم و آن را با عملوند اول جمع می‌کنیم حال در روش مکمل 2 از رقم نقلی تولید شده صرف نظر می‌نماییم ولی در روش مکمل 1 رقم نقلی تولید شده را دوباره با حاصل جمع می‌نماییم. در این روش رقم نقلی زمانی تولید می‌شود که حاصل عملیات تفریق عددی مثبت گردد اگر رقم نقلی تولید نشود حاصل عمل تفریق عددی منفی خواهد بود که باید از آن مکمل گرفته شود و بصورت عددی منفی تعبیر گردد.

نکـــته: در عملیات تفریق A-B به عملوند اول A مفروق منه یا کاهشیاب (minuend) و به عملوند دوم B مفروق یا کاسته (subtrahend) گویند و به حاصل تفریق تفاضل (difference) گویند.

مثال عملیات 18-37 را به دو روش مکمل 1 و مکمل 2 انجام دهید

```
37 = 1 0 0 1 0 1   18 = 0 1 0 0 1 0   2's Complement (18) = 1 0 1 1 1 0

  1 0 0 1 0 1
+ 1 0 1 1 1 0
-------------
1 0 1 0 0 1 1 = 19
```

```
37 = 1 0 0 1 0 1   18 = 0 1 0 0 1 0   1's Complement (18) = 1 0 1 1 0 1

  1 0 0 1 0 1
1+1 0 1 1 0 1
-------------
  0 1 0 0 1 0
  ___________ 1
  0 1 0 0 1 1 = 19
```

مثال عملیات 18-12 را به دو روش مکمل 1 و مکمل 2 انجام دهید

```
12 = 0 1 1 0 0   18 = 1 0 0 1 0   2's complement(18) = 0 1 1 1 0

  0 1 1 0 0
+ 0 1 1 1 0
-----------
  1 1 0 1 0   2's complement(1 1 0 1 0) = 0 0 1 1 0 = - 6
```

```
12 = 0 1 1 0 0   18 = 1 0 0 1 0   1's complement(18) = 0 1 1 0 1

  0 1 1 0 0
+ 0 1 1 0 1
-----------
  1 1 0 0 1   1's complement(1 1 0 0 1) = 0 0 1 1 0 = - 6
```

در روش محاسبات علامت‌دار، عملیات جمع به همان روش عادی انجام می‌گردد ولی باید دقت داشت که عمل جمع دو عدد مثبت حتماً باید عددی مثبت تولید نماید و جمع دو عدد منفی حتما باید عددی منفی تولید نماید در غیر این صورت سرریزی (overflow) رخ داده است و حاصل اشتباه خواهد بود. در خصوص عملیات تفریق عملوند دوم (مفروق/کاسته) را به روش مکمل منفی می‌کنیم و آن را با عملوند اول جمع می‌کنیم حال در روش مکمل 2 از رقم نقلی تولید شده صرف نظر می‌نماییم و در روش مکمل 1 هرگاه رقم نقلی تولید شود باید آن را دوباره با حاصل جمع می‌نماییم.

مثال عملیات 2-5 را به دو روش مکمل 1 و مکمل 2 در سیستم 4 بیتی انجام دهید

```
  5 = 0 1 0 1    2's complement(2) = 1 1 1 0
  2 = 0 0 1 0

  0 1 0 1
+ 1 1 1 0
---------
  0 0 1 1 = 3


  5 = 0 1 0 1    1's complement(2) = 1 1 0 1
  2 = 0 0 1 0

  0 1 0 1
+ 1 1 0 1
---------
1 0 0 1 0
+       1
---------
  0 0 1 1 = 3
```

نکـــته: هرگاه در جمع رقم نقلی ورودی به ستون آخر وارد شود ولی خارج نشود سرریزی رخ می‌دهد. هرگاه در جمع رقم نقلی ورودی به ستون آخر وارد نشود ولی در ستون آخر رقم نقلی ایجاد و خارج شود سرریزی رخ می‌دهد. به رقم نقلی ورودی به هر ستون carry in و به رقم نقلی خروجی از هر ستون carry out می‌گویند.

- **کـدینگ**

کدینگ در این درس به معنی برقراری نگاشت بین اعداد و رشته‌های ارقام است در اینجا بحث را محدود به اعداد 0 الی 9 می‌نماییم. کدینگ BCD ساده‌ترین روش برای نگاشت اعداد به رشته‌های ارقام 4 بیتی است. در کدینگ BCD هر عدد را بصورت باینری 4 بیتی می‌نویسیم. روش افـــزونه 3 (excss-3) همان BDC است که با عدد 3 جمع شده است. دو روش نمایش وزن‌دار دیگر نیز در جدول نشان داده شده. علامت منفی بالای عدد به آن معنی است که وزن آن عدد منفی است. کد وزن دار کدی است که ارزش هرجایگاه توانی از 2 باشد در روش افزونه 3 چون هر رقم متناظر با توانی از 2 نیست لذا این روش وزن‌دار نیست.

عدد	BCD	EXCESS-3	2421	$84\bar{2}\bar{1}$
0	0000	0011	0000	0000
1	0001	0100	0001	0111
2	0010	0101	0010	0110
3	0011	0110	0011	0101
4	0100	0111	1010	0100
5	0101	1000	0101	1011
6	0110	1001	1100	1010
7	0111	1010	1101	1001
8	1000	1011	1110	1000
9	1001	1100	1111	1111

هرگاه در یک کد با معکوس کردن ارقام (تبدیل $1 \rightarrow 0$ و$0 \rightarrow 1$) و به مکمل 9 همان عدد برسیم آن کد را خود مکمل گویند. در جدول فوق همگی کدها به جز BDC خود مکمل هستند. در حالت کلی برای آنکه مکمل 9 عددی یک رقمی را بدست آوریم باید آن عدد را از 9 کم کنیم و $9 - A$ را بدست آوریم.

نکـــته: دو روش برای محاسبه مکمل 9 یک عدد 1 رقمی مانند A وجود دارد روش اول ابتدا A را با عدد 6 جمع نموده سپس از حاصل مکمل یک می‌گیریم یعنی بیت‌های عدد باینری 4 بیتی را معکوس می‌کنیم. روش دوم آنست که ابتدا از عدد A بصورت باینری مکمل یک می‌گیریم یعنی بیت‌های آن را معکوس می‌کنیم سپس حاصل را با عدد ده (10) دسیمال (مبنای 10) جمع می‌کنیم و رقم نقلی را دور می‌ریزیم. باید توجه داشت که اگر عدد باینری A 4 بیتی باشد $2^n - 2^0 - A = 2^4 - 1 - A = 16 - 1 - A = 15 - A$ به بیان دیگر عمل مکمل 1 گرفتن از عدد 4 بیتی A که با معکوس کردن بیت‌های آن انجام می‌گیرد معادل محاسبه $15 - A$ خواهد بود.

1's complement (X) = 15 -X
1's complement (A+6) = 15 -(A+6) = 9 - A

1's complement (A) = 15-A
15-A + 10 = 25 - A = 16 + 9 - A = 9 -A

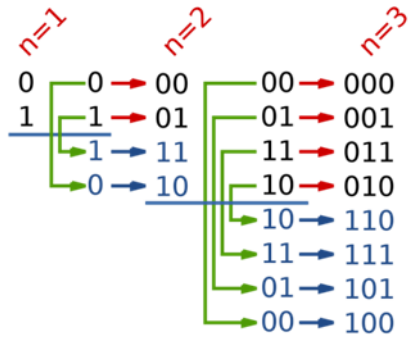
در کدهای خودمکمل هیچ نیازی به این محاسبات نخواهد بود و با معکوس کـــردن بیت‌ها عمل مکمل 9 به انجام می‌رسد.

مثال عملیات تفریق 198-325 را به روش مکمل 9 انجام دهید از کدینگ BCD استفاده کنید.

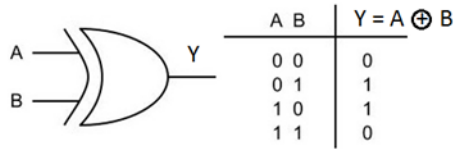
```
 3 2 5        198 = 0001 1001 1000
−1 9 8        complement digit by digit
                   1000 0000 0001

  325              0011 0010 0101
 -198              1000 0000 0001
                   0001 0010 0110
               +                1
 ----------------------------------
  127              0001 0010 0111
```

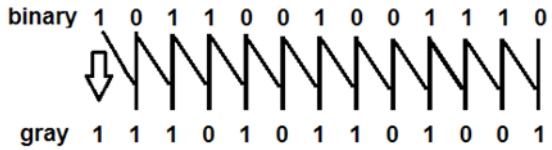
کد انعکاسی گـــری: در حرکت از عدد باینری صفر(0000) تا عدد باینری 15 (1111) مشاهده می‌گردد که در برخی حالات انتقال بین دو عدد مستلزم تغییر 1 بیت است مانند 0000 به 0001 در برخی حالات بین دو عدد متوالی 2 بیت تغییر خواهند کرد مانند 0001 به 0010 و در برخی موارد تغییر بین دو عدد متوالی مستلزم تغییر 3 بیت است مانند 0011 به 0100 کد انعکاسی گـــری برای آن طراحی شده که تغییر بین هر دو عدد متوالی آن مستلزم تغییر یک بیت باشد.



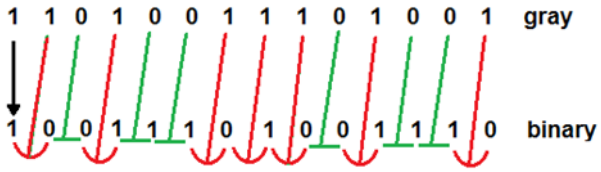
قبل از بیان روش تبدیل باینری به گری و بالعکس لازم است عملگر منطقی XOR (یای انحصاری) را تعریف کنیم. عملگر XOR به این صورت کار می‌کند که اگر فقط یکی از ورودی‌های آن 1 باشد در خروجی 1 را نمایش می‌دهد اگر دو ورودی 1 باشند آنگاه خروجی 0 خواهدبود. اگر هم هردو ورودی 0 باشند در خروجی باز هم 0 را نمایش می‌دهد.



برای تبدیل باینری به گری ابتدا بیت پرارزش را منتقل می‌کنیم سپس به سمت راست حرکت می‌کنیم و دو بیت دو بیت XOR می‌کنیم و حاصل را زیر عدد باینری می‌نویسیم.



برای تبدیل گری به باینری ابتدا بیت پرارزش را منتقل می‌کنیم سپس به سمت راست حرکت می‌کنیم اگر به یک رسیدیم معکوس بیت منتقل شده قبل را قرار می‌دهیم اگر به صفر رسیدیم همان بیت منتقل شده قبل را ادامه می‌دهیم



یک فرمول دیگـــر برای تبدیل گری به باینری وجود دارد و آن فرمول به این صورت است که تعداد یک ها را از آخر تا بیت مورد نظر شمرده و باقی‌مانده آنرا بر 2 محاسبه می‌کنیم.

$$(g_{n-1}g_{n-2}\dots g_i\dots g_1g_0)_g \;\rightarrow\; (b_{n-1}b_{n-2}\dots b_i\dots b_1b_0)_b$$

$$b_i = \left(\sum_{k=i}^{n-1} g_k\right) mod\ 2$$

نکـــته: پس از تبدیل از باینری به گری و بالعکس حتماً نتیجه را یکبار با تبدیل معکوس چک کنید.

نکـــته: کد گری در مخابرات و طراحی شمارنده‌ها و رمزگذاری و تصحیح خطا کاربرد دارد.

سوالات تحقیقی:

1- برای تبدیل یک عدد n رقمی در مبنای r به مبنای b چند رقم در مبنای جدید لازم است؟

2- مکمل 10 یک عدد BCD چگونه حساب می‌شود؟

3- آیا می‌توان با سیستم 4113 یک سیستم خود مکمل ساخت؟ آنرا بسازید

- **کدهای تشخیص و تصحیح خطا**

هرگاه بخواهیم اطلاعات را در قالب اعداد باینری ذخیره و منتقل کنیم ممکن است در اثر وجود نویز (تداخل امواج) و سایر عوامل محیطی یک یا چند بیت تغییر نماید در این صورت داده‌های اولیه دچار خطا می‌گردد در این بخش روشهایی را برای تشخیص و اصلاح خطا بررسی خواهیم نمود. در این بخش وقتی داده را در قاب یک عدد **n** بیتی قرار می‌دهیم به آن یک کلمه **word** می‌گوییم. ابتدا به تعریف فاصله بین دو کلمه می‌پردازیم. فاصله همینگ بین دو کلمه که در قالب اعداد باینری قرار دارند، عبارتست از تعداد بیت‌هایی که در آن دو کلمه متفاوت هستند مثلا فاصله همینگ بین **0010** و **0111** برابر با 2 می‌باشد زیرا دو بیت بین دو کلمه اختلاف وجود دارد. در یک مجموعه اعداد که آن را کدینگ می‌نامیم فاصله مینیمم همینگ برابر است با کمترین فاصله همینگ بین دو کلمه، هرگاه از کل ظرفیت **n** بیت که 2^n عدد می‌شود برای کدینگ استفاده کنیم کمترین فاصله همینگ در آن کدها برابر با 1 خواهد شد و لذا با خراب شدن 1 بیت داده‌های ما تخریب می‌گردد. هرگاه ظرفیت **n** بیت را به دو دسته کد-کلمه **Code-word** و ضد کلمه **Anti-word** تقسیم کنیم به نحوی که کمترین فاصله همینگ بین دو کلمه بیش از 1 باشد آنگاه به قابلیت تشخیص خطا دست یافته‌ایم در این صورت با بروز یک خطا کلمه به ضد کلمه تبدیل می‌گردد و چون ضد کلمه معتبر نیست خطا مشخص می‌گردد.

یکی از روشهای ایجاد قابلیت تشخیص 1 خط استفاده از بیت توازن **(parity)** است. بیت توازن دو نوع می‌باشد توازن فرد و توازن زوج. هرگاه یک بیت به کلمه اضافه کنیم به نحوی که تعداد یک ها زوج گردد بیت توازن زوج را اضافه کرده‌ایم. هرگاه به یک کلمه یک بیت توازن اضافه کنیم به نحوی که تعداد کل یک‌ها فرد گردد بیت توازن فرد اضافه کرده‌ام در این صورت فضای 2^n کلمه‌ای ما به دو زیرفضا 2^{n-1} کلمه‌ای تقسیم خواهد شد در یک نیمه تعداد یک ها فرد است و در یک نیمه تعداد یک ها زوج است اگر

فرض کنیم نیمه شامل تعداد یک فرد نیمه معتبر یا کد-کلمه‌ها باشند نیمه دیگر ضد کلمه‌ها خواهند بود و روش ما توازن فرد نام دارد.

عدد	کلمه یا ضد کلمه	وضعیت	رشته بیت
	ضد کلمه	نامعتبر	000
1	کلمه	معتبر	001
2	کلمه	معتبر	010
	ضد کلمه	نامعتبر	011
0	کلمه	معتبر	100
	ضد کلمه	نامعتبر	101
	ضد کلمه	نامعتبر	110
3	کلمه	معتبر	111
توازن فـــرد (بیت سمت چپ را به عنوان توازن در نظر بگیرید)			

عدد	کلمه یا ضد کلمه	وضعیت	رشته بیت
0	کلمه	معتبر	000
	ضد کلمه	نامعتبر	001
	ضد کلمه	نامعتبر	010
3	کلمه	معتبر	011
	ضد کلمه	نامعتبر	100
1	کلمه	معتبر	101
2	کلمه	معتبر	110
	ضد کلمه	نامعتبر	111
توازن زوج (بیت سمت چپ را به عنوان توازن در نظر بگیرید)			

در روش توازن هرگاه یک بیت دچار خطا گردد از یک کلمه معتبر (کد کلمه) به یک کلمه نامعتبر (ضد کلمه) می‌رسیم و لذا سیستم متوجه بروز خطا می‌گردد.

نکـــته: در روش توازن اگر تعداد فرد بیت دچار خطا گردد آن خطا قابل آشکارسازی است.

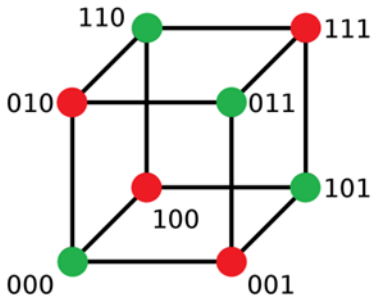
مثال در کد زیر فاصله همینگ مینیمم را محاسبه نمایید.

d0 : 1 1 1 0 0 0
d1 : 0 0 0 1 1 1
d2 : 1 0 0 1 0 1
d3 : 1 1 0 1 0 0

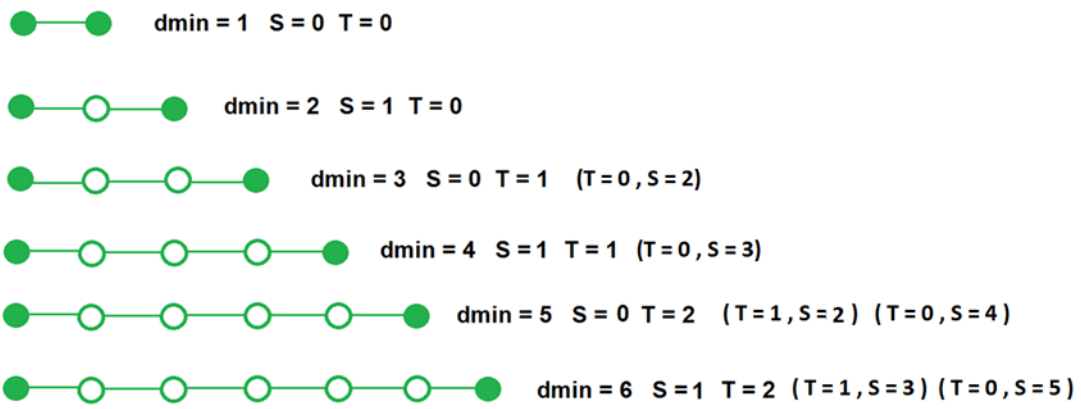
hd(d0,d1) = 6 hd(d0,d2) = 4 hd(d0,d2) = 4 hd(d0,d3) = 2
hd(d1,d2) = 2 hd(d1,d3) = 4 hd(d2,d3) = 2

d min = 2

نکـــته: برای درک بهتر مفهوم توازن، کد-کلمه‌ها و ضدکلمه‌ها را روی رئوس یک مکعب نمایش می‌دهند. هر کد-کلمه با کد-کلمه بعدی به اندازه دو واحد فاصله دارد.



هرگاه مینیمم فاصله همینگ در یک کد برابر با **d** باشد آنگاه این کد قادر است **d-1** خطا را فقط تشخیص دهد و اگر بخواهیم خطا را اصلاح کنیم باید به سمت کد-کلمه نزدیک تر اصلاح کنیم همانطور که در شکل دیده می‌شود باید از رابطه $\left\lfloor \frac{d-1}{2} \right\rfloor$ استفاده کنیم یعنی فاصله منهای 1 را تقسیم بر 2 نموده و کف آنرا محاسبه کنیم مثلا اگر مینیمم فاصله همینگ برابر با 4 باشد 3 را بر 2 تقسیم کرده و کف عدد **1.5** را در نظر می‌گیریم همانطور که مشاهده می‌گردد کدی با مینیمم فاصله همینگ برابر با 4 فقط می‌تواند یک خطا را اصلاح کند زیرا اگر دو خطا رخ دهد فاصله تا کد قبل و بعد یکسان می‌گردد و امکان تشخیص فاصله کمتر میسر نمی‌باشد.



نکـــته: کدی که مینیمم فاصله آن **dmin** است می‌تواند **T** خطا را اصلاح نماید و علاوه بر آن **S** خطای دیگر را تشخیص دهد و رابطه آن برابر است با $dmin \geq S + 2T + 1$ باشد.

مثال در موارد زیر حداقل **dmin** را محاسبه نمایید:

الف) کدی با قابلیت اصلاح 3 خطا و تشخیص 2 خطای دیگر $dmin > 2 + 2 \times 3 + 1 \rightarrow dmin \geq 9$

ب) کدی با قابلیت اصلاح 1 خطا و تشخیص 3 خطای دیگر $dmin > 3 + 2 \times 1 + 1 \rightarrow dmin \geq 6$

نکـــته: با **n** بیت می‌توانیم 2^n عدد تولید کنیم به این معنی که یک فضای به اندازه 2^n را می‌توانیم شماره‌گذاری کنیم یا اصطلاحا آدرس‌دهی نماییم.

برای آنکه بتوانیم کدی داشته باشیم که امکان اصلاح یک خطا را داشته باشد باید به داده ورودی تعدادی بیت توازن اضافه کنیم اگر داده های ما **n** بیتی باشند با افزودن **k** بیت قابلیت تشخیص یک خطا ایجاد می‌گردد. در این حالت طول کلمه ما **k+n** بیت خواهد شد سازوکار کنترلی باید بتواند مشخص کند خطا در کدام بیت رخ داده یا اصلا خطایی رخ نداده پس باید بتواند **n+k+1** بیت را آدرس دهی نماید لذا $2^k \geq n + k + 1$ این سازوکار کد همینگ نام دارد و برای اصلاح یک خطا بهینه است.

مثال برای اصلاح یک خطا در داده 25 بیتی چند بیت باید اضافه نماییم:

باید رابطه $2^k \geq 25 + k + 1$ صدق نماید با حدس و خطا **k=5**

نکـــته: عملگـــر **XOR** که قبلا به آن اشاره شده برای **n** ورودی به این صورت محاسبه می‌شود که اگر تعداد یک در ورودی فـــرد باشد حاصل یک می‌گردد و اگر تعداد یک در ورودی زوج باشد حاصل صفر می‌باشد. عملگر **XOR** را برای سهولت در محاسبات با نماد $\oplus$ نشان می‌دهیم.

کد اصلاح خطای همینگ

کد همینگ یک روش برای اصلاح یک خطا در داده **n** بیتی است که با افزودن **k** بیت توازن انجام می‌گیرد. کدهای همینگ متعدد است. ساده‌ترین آن همینگ 7و4 می‌باشد که 4 بیت داده و 3 بیت توازن و مجموع آن 7 بیت می‌باشد. برای محاسبه همینگ ابتدا باید بیت‌های توازن را با استفاده از فرمول **xor** محاسبه نمود. سپس بیت های توازن محاسبه شده در جایگاه قرار گرفته و داده ارسال می‌گردد در مقصد مجدد بیت‌های توازن با گروه خود **xor** می‌گردند و باید حاصل آن **000** گردد هر عددی غیر از **000** نشان دهنده بروز خطا در همان موقعیت است. مثلا **001** یعنی بروز خطا در خانه اول یا **p1** یا **011** یعنی بروز خطا در خانه سوم یا **d1**.

D4	D3	D2	P3	D1	P2	P1
7	6	5	4	3	2	1

$P1 = D1 \oplus D2 \oplus D4$
$P2 = D1 \oplus D3 \oplus D4$
$P3 = D2 \oplus D3 \oplus D4$

$S1 = P1 \oplus D1 \oplus D2 \oplus D4$
$S2 = P2 \oplus D1 \oplus D3 \oplus D4$
$S3 = P3 \oplus D2 \oplus D3 \oplus D4$

مثال داده **0110** را به روش همینگ ارسال نمایید.

0 D4	1 D3	1 D2	0 P3	0 D1	1 P2	1 P1
7	6	5	4	3	2	1

$P1 = D1 \oplus D2 \oplus D4 \quad P1 = 0 \oplus 1 \oplus 0 = 1$
$P2 = D1 \oplus D3 \oplus D4 \quad P2 = 0 \oplus 1 \oplus 0 = 1$
$P3 = D2 \oplus D3 \oplus D4 \quad P3 = 1 \oplus 1 \oplus 0 = 0$

مثال یک گیرنده در مقصد رشته **1101101** را دریافت نموده با فرض اینکه احتمال وقوع یک خطا وجود دارد بررسی کنید آیا خطا رخ داده است در صورت بروز خطا محل خطا را به روش همینگ مشخص نمایید سپس داده را اصلاح کنید.

1 D4	1 D3	0 D2	1 P3	1 D1	0 P2	1 P1
7	6	5	4	3	2	1

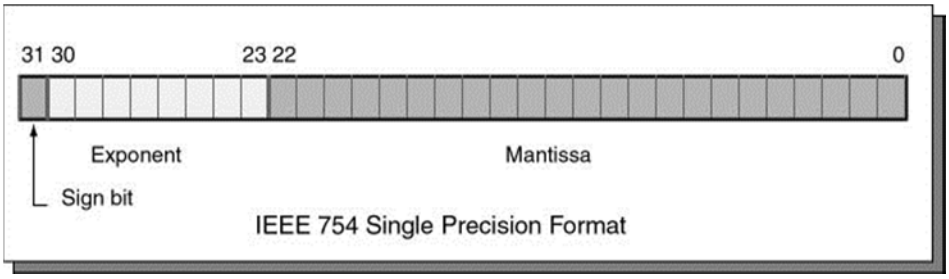
$S1 = P1 \oplus D1 \oplus D2 \oplus D4$
$S2 = P2 \oplus D1 \oplus D3 \oplus D4$
$S3 = P3 \oplus D2 \oplus D3 \oplus D4$

$S1 = 1 \oplus 1 \oplus 0 \oplus 1 = 1$
$S2 = 0 \oplus 1 \oplus 1 \oplus 1 = 1$
$S3 = 1 \oplus 0 \oplus 1 \oplus 1 = 1$

d4 error
correct = 0101101

- **. نمایش اعداد اعشاری در استاندارد** IEEE754

استاندارد **IEEE 754** برای نمایش اعداد اعشاری بکار در رایانه بکار می‌رود. این استاندارد در دو نوع 32 و 64 بیتی وجود دارد که در این کلاس استاندارد 32 بیتی آن مورد بررسی قرار می‌گیرد. در این استاندارد از 4 بایت برای ذخیره‌سازی عدد اعشاری استفاده می‌گردد. بیت پرارزش (بیت 31) برای علامت در نظر گرفته شده و بیت‌های 23 الی 30 توان **(exponent)** و بیت‌های 0 الی 22 مانتیس **(mantissa)** را در خود نگه می‌دارد.



در این روش ابتدا علامت عدد ورودی را در نظر می‌گیریم و آن را در جایگاه بیت علامت (31) قرار می‌دهیم. سپس عدد اعشاری که به صورت اعشاری نوشته شده نرمال می‌گردد عددی نرمال است بصورت $a \times 2^b$ که عدد **a** بصورت **1.XX....** می‌باشد یعنی یک رقم صحیح **1** دارد و بلافاصله بعد از ممیز قرار می‌گیرد (خود عدد صفر استثنا است) بطور مثال عدد $11011 \cdot 1101$ نرمال نیست شکل نرمال آن بصورت $1 \cdot 10111011 \times 2^3$ می‌باشد.

نکـــته: برای نرمال کردن عدد ممیز را به سمت چپ حرکت می‌دهیم و توان 2 را اضافه می کنیم تا به وضعیت **1.XX....** برسیم یا ممیز را به سمت راست حرکت می‌دهیم و از توان 2 کم می‌کنیم.

مثال اعداد زیر را نرمال کنید

الف) **101.1101011** $1 \cdot 011101011 \times 2^2$

ب) **0.000001101** $1 \cdot 101 \times 2^{-6}$

پس از نرمال کردن به عدد بدست آمده بدون توان 2 مانتیس می‌گویند. با توجه به اینکه همیشه عدد سمت چپ ممیز در مانتیس یک است آن را بدیهی فرض کرده و آن را حذف می‌کنیم و بقیه را در قسمت مانتیس قرار می‌دهیم. عدد مانتیس را به نحوی قرار می‌دهیم که بیت چپ آن **1.XYZ....** (همان **X**) در جایگاه 22 قرار

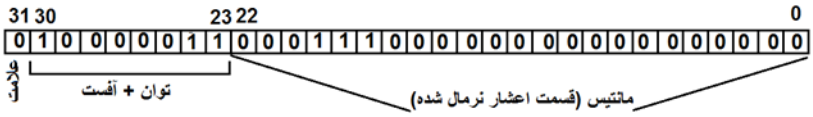
گیرد. سپس توان را با عدد 127جمع می‌کنیم و به فرم باینری در قسمت توان قرار می‌دهیم. عدد 127 آفست نام دارد و کاربرد آن این است که توان‌های منفی را مثبت می‌کند.

مثال عدد **17.75** را در استاندارد **IEEE 754** 32 بیتی ذخیره نمایید.

17.75 17 = 10001 0.75 = 0.5 + 0.25 = 0.11
17.75 = 10001.11
sing bit = 0 10001.11 = 1.000111×2^4

Mantissa = ~~1~~.000111 000111

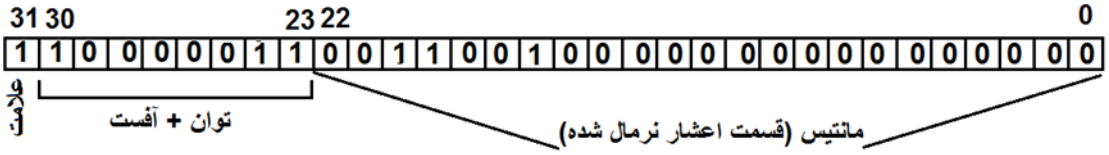
4 + 127 = 131 = 1000 0011



IEEE 754 32 bit

برای تبدیل معکوس نیز ابتدا بیت 31 را به عنوان علامت منظور کرده بیت های 23 الی 30 را دسیمال کرده و از آن 127 را کم می‌کنیم تا توان واقعی بدست آید. سپس مانتیس را از بیت‌های 0 الی 22 خارج کرده و یک **1.** در سمت چپ آن قرار می‌دهیم حاصل بصورت مانتیس ضرب در 2 به توان واقعی است.

مثال عدد زیر مطابق استاندارد **IEEE 754** 32 بیتی ذخیره شده مقدار آن را در مبنای 10 آن را محاسبه نمایید



IEEE 754 32 bit

sign bit = 1 the number is negative

exponent + 127 = 10000011 = 131 131-127 = 4

Mantissa = 1.0011001

-1.0011001×2^4 $-10011.001 = -(16 + 3 + 0.125) = -19.125$

- **جـــــبر بول**

یک سیستم جبری است با دو مقدار **0** و**1** که آنرا **true** و **false** نیز می‌گویند و سه عملگر **AND,OR,NOT** مقدار **A AND B** فقط زمانی مساوی **1** است که هر دو مقدار آن **1** باشد و در بقیه حالات **0** است و مقدار عملگر **A OR B** فقط زمانی مساوی **0** است که هر دو مقدار آن **0** باشد و در بقیه حالات **1** است**.** عملگر **NOT** (نقیض یا معکوس کننده) نیز **0** را به **1** و **1** را به **0** تبدیل می‌کند. عملگر **OR** را با + و عملگر **AND** را با نقطه نمایش می‌دهند برای سهولت نقطه را هم حذف می‌کنند لذا خواهیم داشت **A OR B = A+B** و **A AND B = A.B = AB** برای نشان دادن عمل نقیض بالای متغیر خط قرار می‌دهیم. **NOT(A) =** $\overline{A}$

نکـــته: تابع بولی از متغیرهایی با مقادیر بولی **0** و**1** و عملگرها تشکیل شده. مانندتابع زیر

$$f(x,y,z) = x\bar{y} + \bar{x}y + xz + \bar{z}$$

نکـــته: دوگان یک تابع با تبدیل **AND** به **OR** و تبدیل **OR** به **AND** و تبدیل **1** به **0** و تبدیل **0** به **1** بدست می‌آید.

خواص جبر بول

1) خاصیت عضو خنثی یا همانی **(null element)**

$$a + 0 = a \quad a \cdot 1 = a$$

2) خاصیت جابجایی **(Commutative property)**

$$a + b = b + a \quad ab = ba$$

3) خاصیت شرکت‌پذیری **(associative property)**

$$a + (b + c) = (a + b) + c \quad a(bc) = (ab)c$$

4) خاصیت توزیع‌پذیری **(Distributive Property)**

$$a + (bc) = (a + b)(a + c) \quad a(b + c) = ab + ac$$

5) خاصیت خودتوانی **(Idempotence)**

$$a + a = a \quad aa = a$$

6) مکمل معکوس **(reverse complement)**

$$a + \bar{a} = 1 \quad a\bar{a} = 0$$

7) خاصیت جذب **(absorption)**

$$a(a + b) = a \qquad a + ab = a$$

8) خاصیت شبه جذب

$$a(\bar{a} + b) = ab \quad a + \bar{a}b = a + b$$

9) قاعده دمـــــرگان **(De Morgan's Law)**

$$\overline{(a + b)} = \bar{a}\bar{b} \qquad \overline{(ab)} = \bar{a} + \bar{b}$$

10)قاعده اجماع **(Consensus)**

$$ab + \bar{a}c + bc = ab + \bar{a}c$$
$$(a + b)(\bar{a} + c)(b + c) = (a + b)(\bar{a} + c)$$

11)نظریه بسط شـــــانون **(Shannon's Expansion Theorem)**

$$f(x_1, x_2. \dots, x_n) = x_1\, f(1, x_2. \dots, x_n) + \overline{x_1}\, f(0, x_2, \dots, x_n)$$
$$f(x_1, x_2, \dots, x_n) = \big(x_1 + f(0, x_2. \dots, x_n)\big)\big(\overline{x_1} + f(1, x_2, \dots., x_n)\big)$$

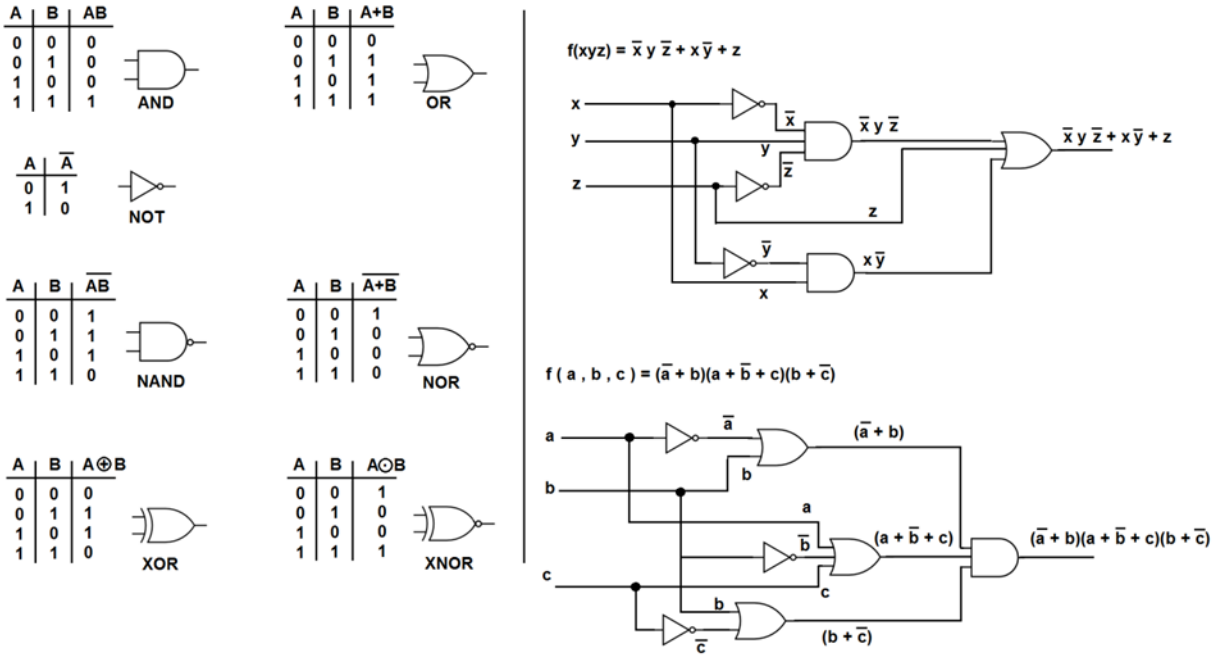
سوالات تحقیقی:

1- نظریه بسط شـــــانون را برای بیش از یک متغیر بنویسید (مثلا 2 متغیره)

2- برای آنکه بتوانیم خطاهای 2 بیتی را در معادله $2^k \geq n + k + 1$ درنظر بگیریم چه تغییری باید در فرمول ایجاد کنیم ؟

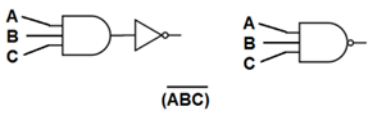
- **گیت‌های منطقی**

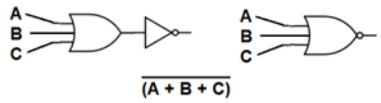
همانگونه که قبلا بیان شد جبر بول دارای سه عملگر **AND** و **OR** و **NOT** و دو مقدار **0** و **1** است. توابع بولی بر اساس همین عملگرها ساخته می‌شوند. گیت‌های منطقی نمایش عملگرها و توابع منطقی هستند.



توابع اصلی **AND** و **OR** و **NOT** هستند و از روی آنها **NAND** و **NOR** و **XOR** و **XNOR** بدست می‌آیند.

نکـــته: **NAND** چند ورودی به این صورت است که ابتدا همه ورودی‌ها را با هم **AND** می‌کنیم سپس حاصل را معکوس می‌کنیم. **NOR** چند ورودی به این صورت است که ابتدا همه ورودی‌ها را با هم **OR** می‌کنیم سپس حاصل را معکوس می‌کنیم.**NAND** و **NOR** خاصیت شرکت‌پذیری ندارند.





نکـــته: **XOR** چند ورودی به این صورت است که تعداد یک‌های ورودی را شمارش می‌کند اگر تعداد فرد یک در ورودی قرار داشت خروجی 1 می‌شود. **XNOR** چند ورودی به این صورت است که تعداد یک‌های ورودی را شمارش می‌کند اگر تعداد زوج یک در ورودی قرار داشت خروجی 1 می‌شود.

- **فـــرمهای استاندارد مینترم و ماکسترم**

به هر جمله که از چند متغیر و عملگر تشکیل شده باشد یک ترم می‌گویند مثلا **xyz** یک ترم است (حاصلضرب) و **x+z** نیر یک ترم محسوب می‌شود که به شکل حاصلجمع است. دو فرم اصلی برای بیان توابع بولی وجود دارد فرم اول جمع حاصلضرب‌ها یا **SOP (Sum of Products)** نام دارد و فرم دوم ضرب حاصلجمع‌ها است که به آن **POS (Product of Sums)** می‌گویند.

$SOP\ example: f(x,y,z) = x\bar{y} + \bar{x}yz + \bar{y}z$
$POS\ example: f(x,y,z) = (x + \bar{y})(\bar{x} + y + z)(\bar{y} + z)$

مینترم یک **product** یا حاصلضرب است که در آن هر متغیر یا نقیض آن به ترتیب قرارداند. ماکسترم یک **sum** یا حاصلجمع است که در آن هر متغیر یا نقیض آن به ترتیب قرارداند. هر مینترم و یا ماکسترم متناظر با یک عدد باینری است. برای مینترم‌ها عدد باینری را در نظر می‌گیریم به ازای **1** خود متغیر و به ازای **0** نقیض متغیر را قرار می‌دهیم. برای ماکسترمها عدد باینری را در نظر می‌گیریم به ازای **1** نقیض متغیر و به ازای **0** خود متغیر را قرار می‌دهیم.

	minterm	maxterm
000	$\bar{x}\bar{y}\bar{z}$	$(x+y+z)$
001	$\bar{x}\bar{y}z$	$(x+y+\bar{z})$
010	$\bar{x}y\bar{z}$	$(x+\bar{y}+z)$
011	$\bar{x}yz$	$(x+\bar{y}+\bar{z})$
100	$x\bar{y}\bar{z}$	$(\bar{x}+y+z)$
101	$x\bar{y}z$	$(\bar{x}+y+\bar{z})$
110	$xy\bar{z}$	$(\bar{x}+\bar{y}+z)$
111	xyz	$(\bar{x}+\bar{y}+\bar{z})$

نکـــته: هر مینترم فقط به ازای یک حالت **1** است (به ازای عدد متناظر با آن) و به ازای بقیه حالات **0** است. هر ماکسترم به ازای همه حالات **1** است و فقط در یک حالت **0** است که همان عدد متناظر با آن ماکسترم است.

هر تابع بولی **n** متغیره را می‌توان در یک جدول نشان داد که به آن جدول درستی **truth table** می‌گویند. خروجی تابع به ازای کلیه حالات ورودی، در جدول نشان داده می‌شود. هر تابع بولی را می‌توان به صورت جمع مینترم‌ها و یا ضرب ماکسترم‌ها نوشت. بطور مثال تابع بولی سه متغیره **f(a,b,c)** که در جدول زیر نشان داده شده است را به فرم استاندارد **Canonical** جمع حاصلضرب‌ها و فرم استاندارد ضرب حاصل‌جمع ها نمایش می‌دهیم.

a b c	f(a,b,c)
000	0
001	1
010	1
011	0
100	1
101	0
110	0
111	1

$$f(a,b,c) = \sum m(1,2,4,7)$$

$$f(a,b,c) = \prod M(0,3,5,6)$$

همانطور که مشاهده می‌گردد اگر بخواهیم تابع را به فرم استاندارد **Canonical** جمع حاصلضرب‌ها بنویسیم به ازای **1** در خروجی مینترم متناظر با آن را درون سیگما قرار می‌دهیم برای مینترم‌ها از **m** استفاده می‌کنیم برای می‌گردد اگر بخواهیم تابع را به فرم استاندارد **Canonical** ضرب حاصلجمع‌ها بنویسیم به ازای **0** در خروجی ماکسترم متناظر با آن را درون پای $\prod$ قرار می‌دهیم برای ماکسترم‌ها از **M** استفاده می‌کنیم.

نکـــته: به فرم استاندارد **Canonical Form** نیز می‌گویند. به فرم استاندارد جمع حاصلضرب‌ها **DNF Disjunctive Normal Form** و به فرم استاندارد ضرب حاصلجمع‌ها **CNF Conjunctive Normal Form** نیز می‌گویند.

نکـــته: گاهی اوقات در تابع بولی خروجی تابع به ازای برخی ورودی‌ها اصلا مهم نیستند زیرا هرگز ورودی در چنین وضعیتی قرار نمی‌گیرد و یا هرگاه در آن وضعیت قرار گرفت خروجی تابع هرچه باشد اصلا مهم نیست به این حالات بی اهمیت **Don't care** گویند و در جدول بصورت **x** و در فرم استاندارد بصورت **d** نشان می‌دهند.

a b c	f(a,b,c)
000	0
001	1
010	1
011	0
100	1
101	0
110	x
111	x

$f(a,b,c) = \sum$ **m(1,2,4) + d(6,7)**

$f(a,b,c) = \prod$ **M(0,3,5)D(6,7)**

هر تابع بولی به فرم استاندارد را می‌توان با خود متغیرها نیز نشان داد برای این کار ترم های آن را می نویسیم.

$$f(a,b,c) = \sum m(1,3,5) = \bar{a}\bar{b}c + \bar{a}bc + a\bar{b}c$$

$$f(x,y,z) = \prod M(0,2,4,6,7) = (x+y+z)(x+\bar{y}+z)(\bar{x}+y+z)(\bar{x}+\bar{y}+z)(\bar{x}+\bar{y}+\bar{z})$$

هرگاه یک تابع بولی به فرم استاندارد نباشد و بخواهیم آنرا به فرم استاندارد جمع حاصلضربها بنویسیم باید در هر ترم آن تابع متغیرهای ناموجود را ایجاد کنیم برای این کار در هر ترم به ازای متغیر ناموجود یک پرانتز قرار داده و درون پرانتز حاصلجمع آن متغیر و نقیض آن را قرار می‌دهیم حاصلجمع متغیر و نقیض آن از نظر منطق جبر بول برابر با **1** است و تاثیری در آن ترم ندارد به بیان دیگر آن ترم را در **1** منطقی ضرب **(AND)** می‌کنیم سپس با استفاده از خواص توزیع پذیری بسط می‌دهیم باید حتما ترتیب قرار گیری متغیرها را رعایت کنیم سپس جملات تکراری را حذف می کنیم و پس از آن به ازای هر مینترم عدد متناظر را درون سیگما قرار می‌دهیم.

$f(a,b,c) = ab + b\bar{c} \rightarrow ab(c+\bar{c}) + (a+\bar{a})b\bar{c} = abc + ab\bar{c} + ab\bar{c} + \bar{a}b\bar{c}$

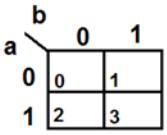
$= abc + ab\bar{c} + \bar{a}b\bar{c} = \sum m(2,6,7)$

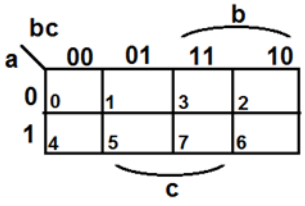
هرگاه یک تابع بولی به فرم استاندارد نباشد و بخواهیم آنرا به فرم استاندارد ضرب حاصلجمع‌ها بنویسیم باید در هر ترم آن تابع متغیرهای ناموجود را ایجاد کنیم برای این کار در پرانتز هر ترم به ازای متغیر ناموجود حاصلضرب آن متغیر و نقیض آن را قرار می‌دهیم حاصلضرب متغیر و نقیض آن از نظر منطق جبر بول برابر با **0** است و اگر با آن ترم جمع شود تاثیری در آن ترم ندارد به بیان دیگر آن ترم را با **0** منطقی جمع **(OR)** می‌کنیم سپس با استفاده از خواص توزیع پذیری بسط می‌دهیم باید حتما ترتیب قرار گیری متغیرها را رعایت کنیم سپس جملات تکراری را حذف می کنیم و پس از آن به ازای هر ماکسترم عدد متناظر را درون پای $\prod$ قرار می‌دهیم.

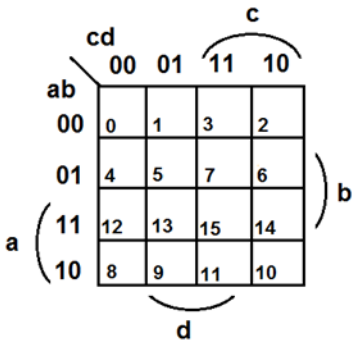
$$f(a,b,c) = (a+b)(\bar{b}+\bar{c}) = (a+b+c\bar{c})(a\bar{a}+\bar{b}+\bar{c})$$
$$= (a+b+c)(a+b+\bar{c})(a+\bar{b}+\bar{c})(\bar{a}+\bar{b}+\bar{c}) = \prod M(0,1,3,7)$$

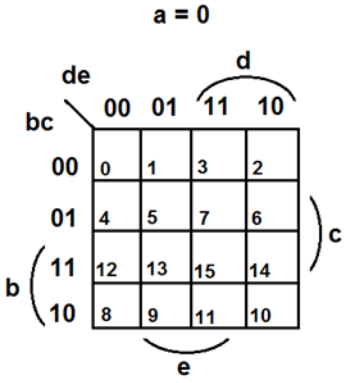
- **ساده‌سازی توابع بولی به روش نقشه‌های کارنو**

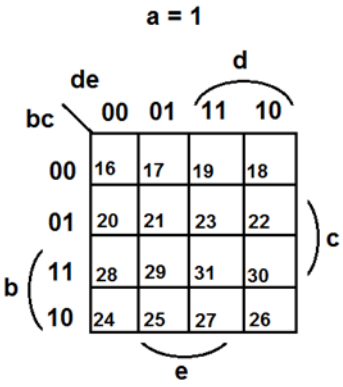
توابع بولی که به فرم استاندارد بیان می‌شوند یا از روی جدول استخراج می‌گردند با تعدادی گیت منطقی پیاده‌سازی می‌گردند. هرچه تعداد گیت‌ها کمتر باشد تعداد ترانزیستورها کمتر و مدار کوچکتر و سریعتر و به لحاظ مصرف انرژی بهینه‌تر خواهد بود لذا توابع بولی را ساده کنیم تا تعداد ترم‌ها کمتر و تعداد متغیرهای هر ترم نیز کمتر گردد. اولین روش برای ساده‌سازی توابع بولی استفاده از جداول یا نقشه‌های کارنو است. برای ساده سازی توابع که به فرم استاندارد جمع مینترم‌ها بیان شده‌اند در جدول به ازای هر مینترم **1** در خانه مورد نظر قرار می‌دهیم. سپس آن یک‌هایی که همسایه هستند را گروه‌هایی از توان 2 دسته‌بندی می‌کنیم از گروه‌های بزرگتر شروع می‌کنیم و اگر گروه‌ها همپوشانی هم داشته باشند اشکالی ندارد. پس از آنکه همه یک‌ها گروه‌بندی شدند بر اساس آنکه بین کدام متغیرها مشترک هستند ترم‌های جدید بدست می‌آیند. جدول کارنو از اطراف هم به هم متصل است. جداول2، 3، 4 و 5 متغیره را باید حفظ نماییم البته روش نوشتن اعداد درون جدول مانند کد گری است.





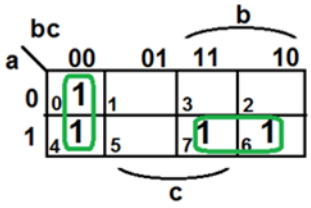




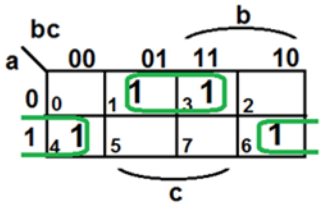


برای ساده سازی توابع که به فرم استاندارد ضرب ماکسترم‌ها بیان شده‌اند در جدول به ازای هر ماکسترم (عدد صفر در خروجی تابع) **0** در خانه مورد نظر قرار می‌دهیم. سپس آن صفرهایی که همسایه هستند را در گروه‌هایی از توان 2 دسته‌بندی می‌کنیم از گروه‌های بزرگتر شروع می‌کنیم و سپس هر گروه معادل یک ترم جدید خواهد شد. اکنون باید هر ترم را **NOT** کنیم تا به فرم **POS** تبدیل گردد.

وقتی یک تابع دارای حالات بدون اهمیت باشد لازم نیست آن حالات را گروه‌بندی کنیم ولی می‌توانیم هرکجا لازم است **X** ها را برای ایجاد گروه‌های بزرگتر پوشش دهیم در واقع **X** ها را هر وقت بخواهیم **1** فرض می کنیم و هر وقت بخواهیم **0** فرض می‌کنیم. در ادامه ساده‌سازی توابع 3 متغیره را به روش **SOP** در جداول کارنو ارائه می‌دهیم باید توجه داشت که در توابع سه متغره گروه 1 عضوی معادل مینترم (سه متغیره) است گروه 2 عضوی 2 متغیر دارد و گروه 4 عضوی یک متغیره است گروه 4 عضوی هم تابع همیشه درست را نشان می‌دهد. توابع **XOR** و **XNOR** طرح‌های شطرنجی ایجاد می‌کنند که اصلا قابل ساده‌سازی نیستند.

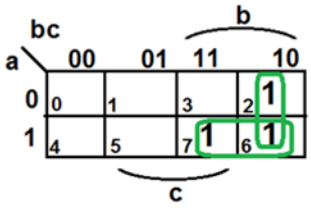


$\bar{b}\bar{c} + ab$

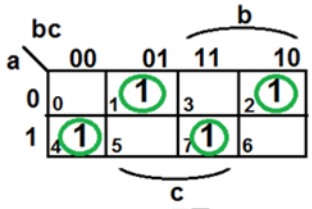


$a\bar{c} + \bar{a}c$

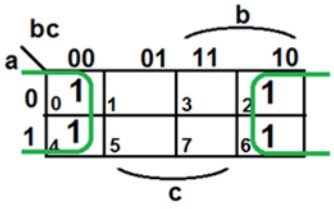
1 0



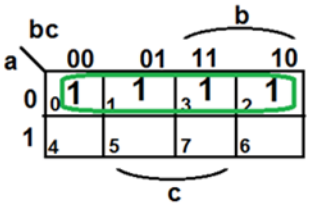
$b\bar{c} + ab$



$XOR(a,b,c) = \bar{a}\bar{b}c + \bar{a}b\bar{c} + a\bar{b}\bar{c} + abc$

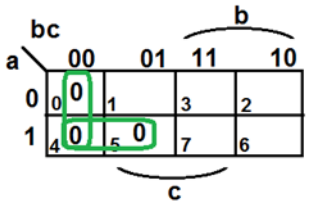


$\bar{c}$

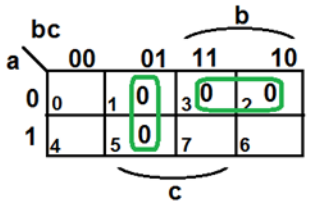


$\bar{a}$

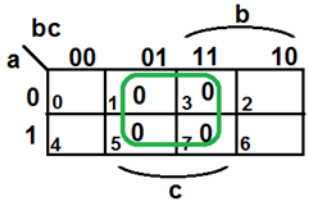
ساده‌سازی توابع سه متغیره به روش **POS** با گروه‌بندی **0** ها انجام می‌گیرد سپس به ازای هر گروه یک حاصلضرب می‌نویسیم مانند **SOP** ولی باید حاصلضرب را **NOT** کنیم تا به حاصلجمع برسیم.



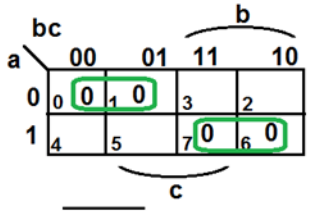
$\bar{b}\bar{c} + a\bar{b} = (b+c)(\bar{a}+b)$



$\bar{b}c + \bar{a}b = (b+\bar{c})(a+\bar{b})$

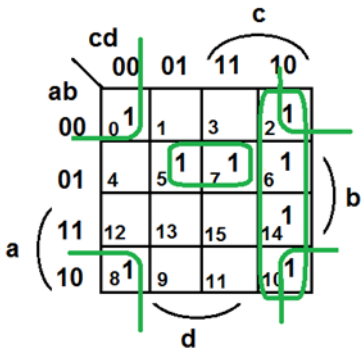


$\bar{c}$

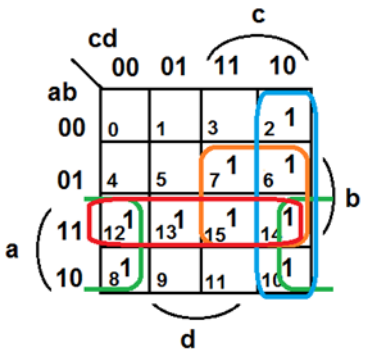


$\bar{a}\bar{b} + ab = (a+b)(\bar{a}+\bar{b})$

ساده‌سازی توابع چهار متغیره به روش **SOP** در مثالهای زیر ارائه شده است لازم به ذکر است هرجا بتوانیم گروه بزرگتری ایجاد می‌کنیم.

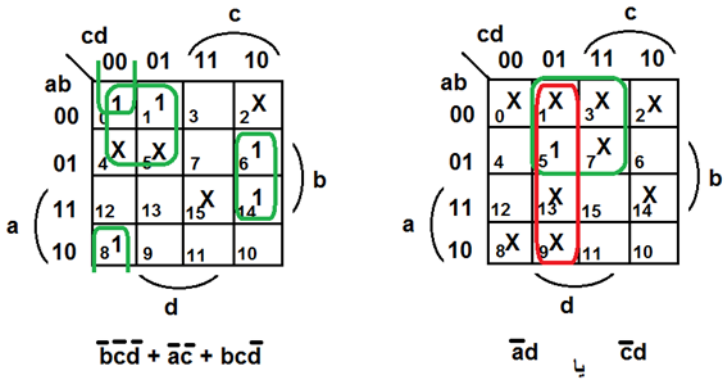


$\bar{b}\bar{d} + \bar{a}bd + c\bar{d}$

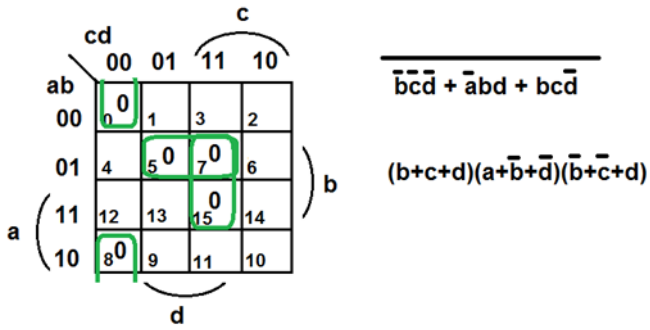


$a\bar{d} + ab + bc + c\bar{d}$

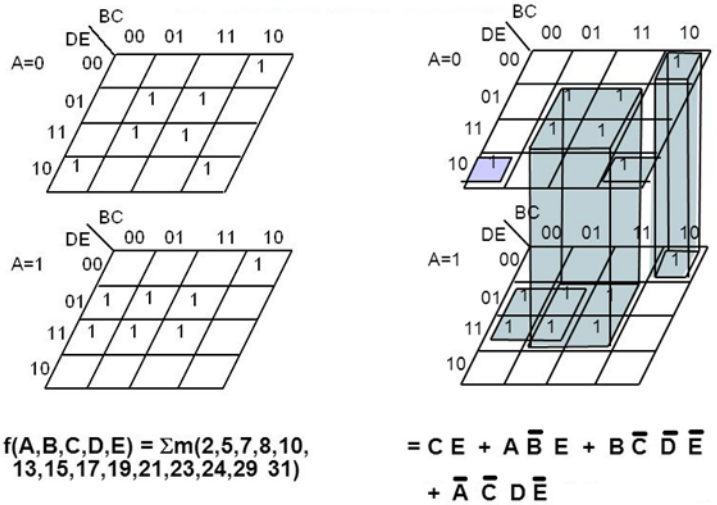
ساده‌سازی توابع چهار متغیره به روش **SOP** با در نظر داشتن حالات بدون اهمیت در مثالهای زیر آمده است. همانطور که در شکل سمت راست دیده می‌شود دو جواب برای ساده‌سازی وجود دارد. وجود جواب‌های متعدد برای ساده‌سازی به شرط آنکه از لحاظ تعداد ترم‌ها و تعداد متغیرهای هر ترم مساوی باشند پذیرفته شده است.



ساده‌سازی توابع چهار متغیره به روش **POS** در مثال زیر ارائه شده

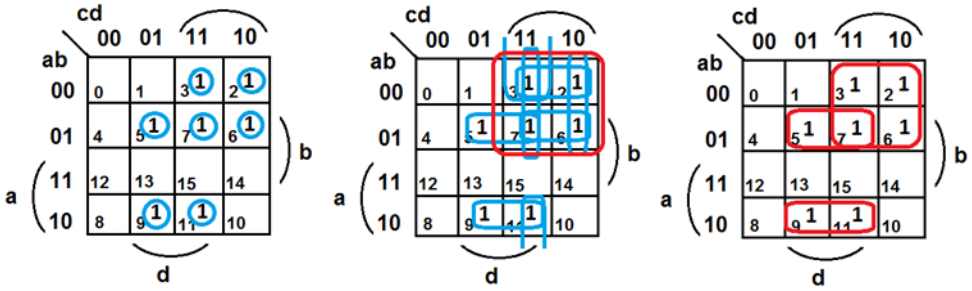


ساده‌سازی توابع پنج متغیره نیز مانند چهار متغیره است فقط با این تفاوت که باید فرض کنید دو صفحه روی هم قرار دارند و نظیر به نظیر می‌توانند همسایگی تشکیل دهند.



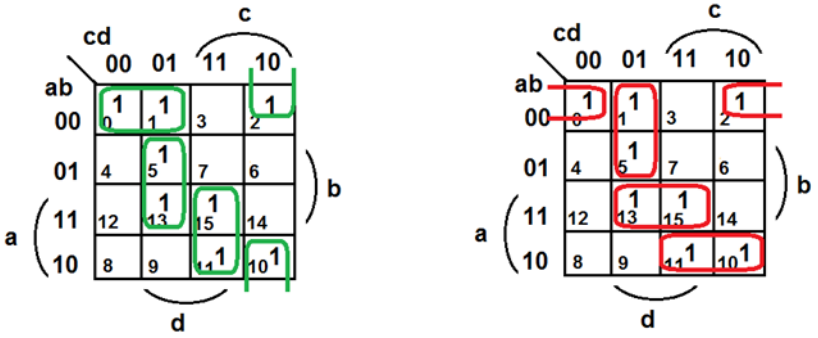
• روش کوئین مک‌کلاسکی

یک روش دیگر برای ساده‌سازی توابع بولی روش کوئین‌مک‌کلاسکی است. این روش الگوریتمیک بوده و برای پیاده‌سازی در رایانه مناسب است. قبل از پرداختن به روش کوئین‌مک‌کلاسکی ابتدا لازم است مفاهیم آن را مرور کنیم. ابتدا به تعریف ایجاب کننده **implicant** در تابع **SOP** می‌پردازیم[1]. هر مینترم و یا هر گروهی از مینترم‌ها که با هم ساده می‌شوند یک ایجاب کننده یا **implicant** هستند. ایجاب‌کننده نخستین یا **prime implicant** ایجاب کننده‌ای است که توسط ایجاب‌کننده‌های دیگر پوشش داده نمی‌شود به بیان دیگر ساده‌تر نمی‌شود. ایجاب‌کننده نخستین اساسی **Essential Prime Implicant** آن دسته از ایجاب کننده‌هایی هستند که شامل حداقل یک مینترم هستند که هیچ ایجاب کننده دیگری آن مینترم را پوشش نمی‌دهد. در شکل زیر سمت چپ 7 ایجاب کننده 4 متغیره (تک خانه‌ای) دیده می‌شود هیچکدام از آنها نخستین نیستند زیرا همه آنها با ایجاب‌کننده‌های بزرگتر پوشش داده می‌شوند. در شکل وسط 8 ایجاب کننده دیده می‌شود که 7 تا از آنها 3 متغیره (دو خانه‌ای) و یکی از آنها 2 متغیره (4 خانه‌ای) است. در شکل سمت راست ایجاب کننده‌های نخستین اساسی که تابع را پوشش می‌دهند نشان داده شده است.



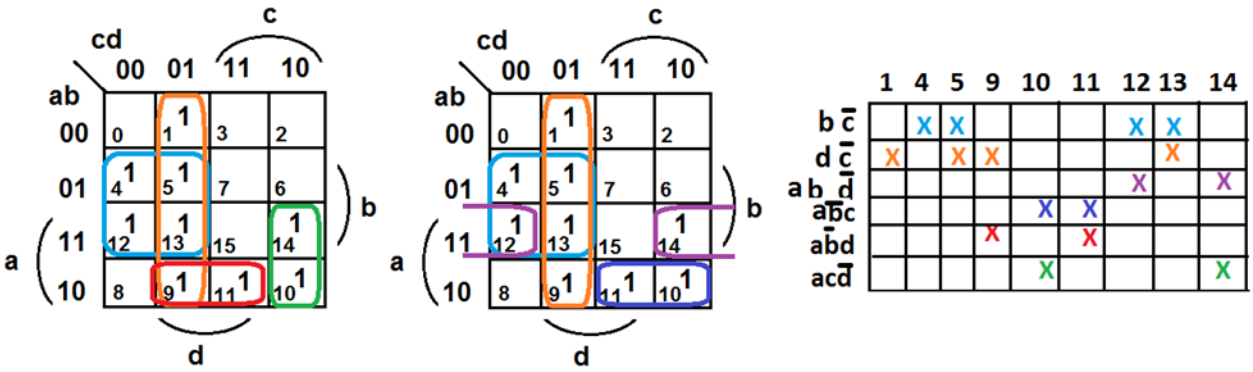
در شکل زیر در مجموع 16 ایجاب کننده داریم. 8 ایجاب کننده 4 متغیره (یک خانه‌ای) داریم که نخستین نیستند زیرا توسط 8 ایجاب کننده 3 متغیره (دو خانه‌ای) دیگر پوشش داده می‌شوند با استفاده از چهار ایجاب کننده تابع بولی در دو شکل مختلف ساده می‌گردد هیچیک از ایجاب کننده‌ها اساسی نیستند زیرا هیچکدام شامل مینترمی نیست که در ایجاب کننده دیگر نباشد.

[1] در **POS** نیز تعریف مشابه است فقط به جای مینترم باید ماکسترم را جایگزین کنیم



نکـــته: گاهی فرم ساده شده بهینه یک تابع بولی یکتا نیست.

جدول پوشش برای انتخاب ایجاب‌کننده‌ها استفاده می‌گردد هر ستون آن یک مینترم است و هر سطر آن یک ایجاب‌کننده است که تعدادی مینترم را پوشش می‌دهد. ایجاب کننده‌های نخستین اساسی حتما باید انتخاب شوند زیرا شامل مینترم‌هایی هستند که سایر ایجاب کننده‌ها آن را پوشش نمی‌دهند. از بین سایر ایجاب کننده‌ها آنهایی که شامل مینترم‌هایی بیشتری هستند اولویت انتخاب دارند.



در این شکل دو ایجاب کننده 1 و 2 در جدول پوشش، ایجاب کننده‌های نخستین اساسی هستند و برای ساده‌سازی انتخاب می‌گردند پس از انتخاب دو ایجاب کننده اول باید به نحوی انتخاب کنیم که مینترمهای 10 11 و 14 پوشش داده شوند از بین سایر ایجاب کننده‌ها می‌توانیم 3 و 4 را انتخاب کنیم در آن صورت مانند شکل وسط ساده کرده ایم و یا 5 و 6 را انتخاب کنیم که مانند شکل سمت چپ ساده کرده باشیم. هر دو حالت درست است. زیرا دو ایجاب کننده 3 متغیره (دو خانه‌ای) به مجموعه جواب اضافه کرده‌ایم.

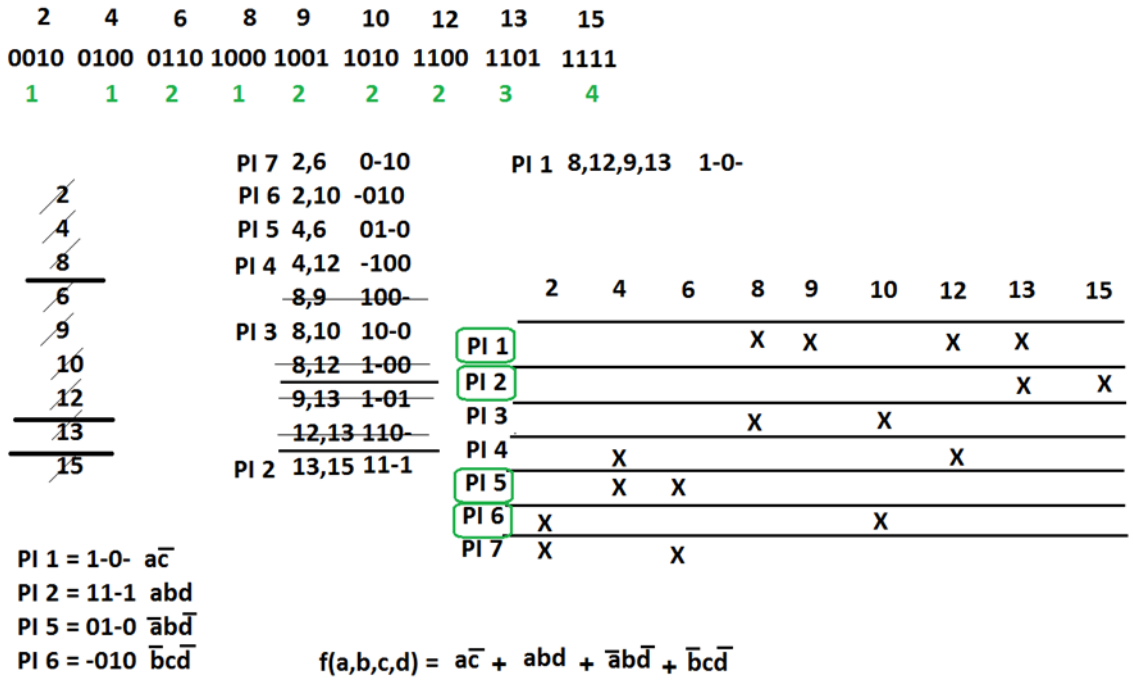
قبل از بیان روش کوئین مک‌کلاسکی لازم است ساده کردن مینترم‌ها را به روش عددی بیان کنیم. هر مینترم معادل یک عدد است هرگاه تفاضل اعداد دو مینترم به اندازه توانی از 2 باشند می‌توانیم آن دو مینترم را ساده کنیم به این صورت که به ازای توان 2 مشترک یک خط تیره قرار می‌دهیم و سایر ارقام را می‌نویسیم مثلا 12 و 4 به صورت **100- 12,4** نوشته می‌شود اگر به جای خط **0** قرار دهیم عدد کمتر ساخته می‌شود و اگر به جای خط **1** قرار دهیم عدد بزرگتر ساخته می‌شود. مثال دیگر 14 و 10 است که بصورت **10-1 14,10** ساده می‌شود زیرا تفاضل آن دو عدد برابر با 4 است لذا در محل 4 خط قرار می‌دهیم.

در روش کوئین مک‌کلاسکی ابتدا اعداد مینترم‌ها را بصورت باینری می‌نویسیم سپس آنها را بر اساس تعداد 1 دسته بندی می‌کنیم سپس اعداد مینترم‌های هر دسته را زیر هم بصورت صعودی نوشته و بین دسته‌ها خط می‌کشیم سپس هر عدد مینترم را با عدد مینترم های دسته پایینی مقایسه می‌کنیم اگر عدد مینترم پایینی بزرگتر بود و تفاضل آن دو عدد مینترم توانی از عدد 2 بود آن دو عدد مینترم با هم ساده می‌شوند ان دو را خط زده و در یک جدول دیگر به ازای آن دو عدد یک عدد بصورت **a,b** می‌نویسیم و شکل باینری آن دو عدد را در قالب یک عدد می‌نویسیم پس از آنکه تمام اعداد هر دسته با دسته زیرین مقایسه شد به سراغ دسته پایینی می‌رویم پس از اتمام این مرحله به جدول جدید می‌رویم و دوباره مقایسه می‌کنیم این‌بار بین دو عدد قسمت خط را باهم مقایسه می‌کنیم اگر محل خط در دو عدد با هم انطباق داشت و بین سایر بیت‌ها فقط یک بیت تفاوت داشت آن دو عدد با هم ساده می‌شوند و به جای آن بیت دوم هم خط قرارمی‌گیرد مثلا **11-1** با **10-1** ساده می‌شود و به **1-1-** می‌رسد عدد **11-1**معادل **15,11** و **10-1** معادل **14,10** می‌باشد لذا ساده شده آن بصورت **14,10,15,11** خواهد بود. روند ساده‌سازی تا جایی پیش می‌رود که امکان ساده شدن بیشتر میسر نباشد پس از اتمام آن هر مجموعه عدد یک ایجاب‌کننده نخستین خواهد بود ایجاب کننده‌های غیر نخستین همگی باید در فرآیند ساده‌سازی خط خورده باشند. پس از آن جدول پوشش را ترسیم می‌کنیم و ایجاب کننده‌های نخستین اساسی را انتخاب کرده و سایر ایجاب کننده‌های نخستین را به نحوی انتخاب می‌کنیم که پوشش کامل با حداقل جملات و متغیرها انجام گردد. به ازای هر عدد مانند **01-1** به جای **1** متغیر مربوط را قرار می‌دهیم به ازای **0** معکوس متغیر و به ازای خط متغیر حذف می‌گردد لذا $a\bar{c}d = 01 - 1$

نکـــته: جملات بی اهمیت را در فرآیند ساده‌سازی مانند مینترم‌ها می‌بینیم ولی در جدول پوشش قرار نمی‌دهیم.

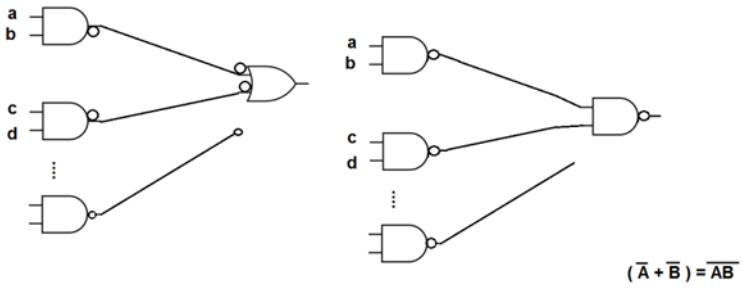
نکـــته: در هنگام ساده شدن مرحله دوم به بعد ایجاب کننده‌های تکراری ایجاد می‌شوند آنها را حذف می‌کنیم.

مثال تابع $f(a,b,c,d) = \sum m(2,4,6,8,9,10,12,13,15)$ را به روش کوئین مک‌کلاسکی ساده نمایید

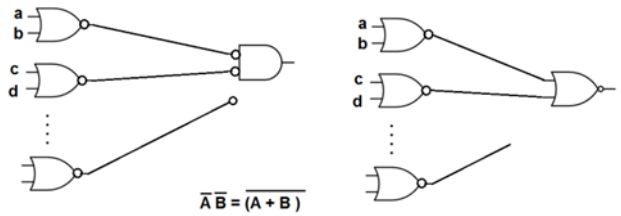


- **پیاده‌سازی با یک گیت**

در الکترونیک دیجیتال ساخت مدارات با یک گیت خاص راحتر است تا ساخت با گیت‌های متنوع. از جبر بول می‌دانیم تمامی توابع را می‌توانی با **AND** و **NOT** یا با **OR** و **NOT** بسازیم در ادامه روشهایی برای ساخت مدار یا یک نوع گیت **NAND** یا **NOR** ارائه می‌دهیم. برای آنکه یک مدار را فقط با گیت **NAND** پیاده‌سازی کنیم ابتدا آنرا بصورت **SOP** نوشته و سپس بین لایه **AND** و **OR** گیت **NOT** (بصورت حباب) قرار می‌دهیم دو حباب متوالی دو **NOT** متوالی هستند که یکدیگر را خنثی می‌کنند در یک سمت **NAND** تولید شده و در سمت دیگر **NOT-OR** تولید می‌شود که طبق دمورگان معادل **NAND** است.



برای آنکه یک تابع را فقط با گیت **NOR** پیاده‌سازی کنیم ابتدا آنرا بصورت **POS** نوشته و سپس بین لایه **OR** و **AND** گیت **NOT** (بصورت حباب) قرار می‌دهیم دو حباب متوالی دو **NOT** متوالی هستند که یکدیگر را خنثی می‌کنند در یک سمت **NOR** تولید شده و در سمت دیگر **NOT-AND** تولید می‌شود که طبق دمورگان معادل **NOR** است.

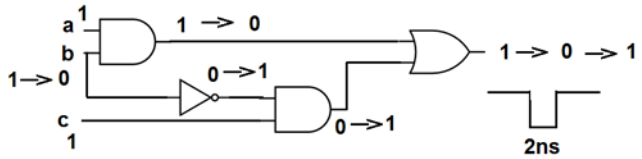


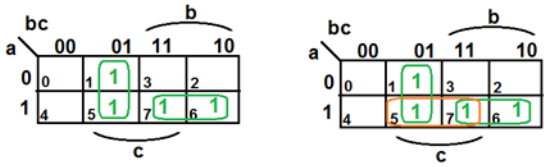
نکـــته: اگر تابع به شکل **POS** داده شده بود و خواستند با گیت **NAND** پیاده سازی گردد از روی ماکسترم‌ها آن ماکسترم‌هایی که نیستند را در نظر گرفته و به جای آن مینترم درج می‌کنیم تا به فرم **SOP** در آید.

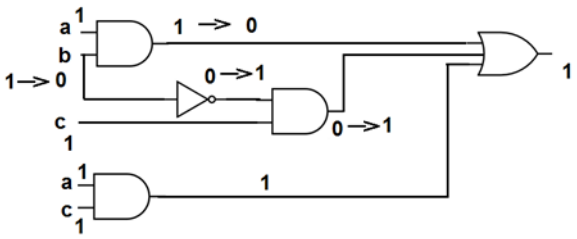
- **مخاطره و رفع مخاطره**

در الکترونیک دیجیتال هر گیت با استفاده از چند ترانزیستور ساخته می‌شود و هنگامی که وروی گیت تغییر می‌کند خروجی گیت بلافاصله تغییر نمی‌کند بلکه زمان بسیار کوتاهی در حد نانو ثانیه یا میکرو ثانیه (بستگی به تکنولوژی ساخت) طول می‌کشد تا خروجی تغییر نماید که به آن تاخیر انتشار گوییم. در اینجا برای سادگی فرض میکنیم تاخیر انتشار هر گیت 2 نانو ثانیه است. در مدار زیر مقادیر هر سه متغیر **1** است هرگاه متغیر **B** از **1** به **0** تغییر کند طبق منطق جبر بول نباید خروجی مدار تغییر نماید اما با بررسی دقیقتر متوجه می‌شویم که خروجی به مدت 2 نانوثانیه تغییر می‌کند و از **1** به **0** رفته سپس باز می‌گردد. زیرا وقتی وقتی متغیر **B** از **1** به

0 تغییر می‌کند ابتدا گیت **AND** بالایی متاثر شده و خروجی آن **0** می‌شود سپس گیت پایینی متاثر شده زیرا در مسیر انتشار آن یک گیت **NOT** وجود دارد که باعث تاخیر می‌گردد.





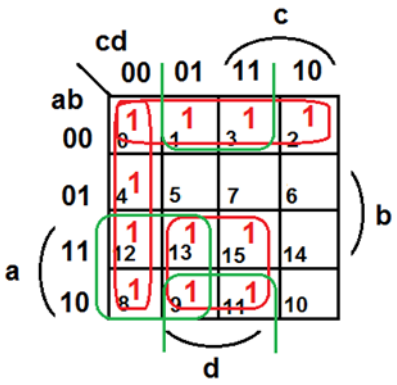


تغییر ناخواسته در خروجی را مخاطره **(Hazard)** می‌گویند و می‌تواند مدار را ناپایدار نماید. همانگونه که در شکل مشاهده می‌شود مخاطره زمانی رخ می‌دهد که دو ترم (جمله) با هم مرز باشند ولی جمله مشترکی نداشته باشند اگر جمله مشترک مانند **ac** در شکل فوق اضافه شود مخاطره رفع می‌گردد.

نکـــته: در مدارات **SOP** مخاطره به صورت حرکت از **1** به **0** رخ می‌دهد و در مدارات **POS** مخاطره به صورت حرکت از **1** به **0** رخ می‌دهد.

نکـــته: با افزودن جملاتی که مرزهای جملات را پوشش می‌دهد مخاطره رفع شده ولی تابع دیگر بهینه نخواهد بود.

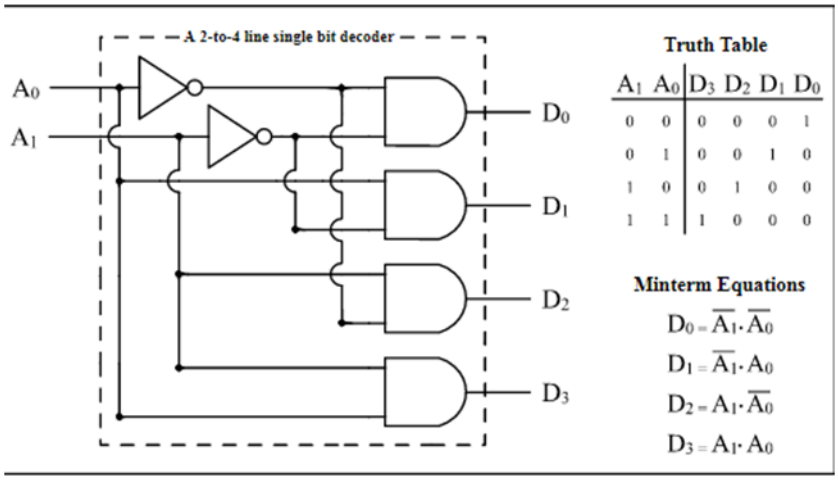
در مثال زیر با افزودن جملات سبز مخاطره رفع می‌گردد.



$\bar{b}d + a\bar{c}$

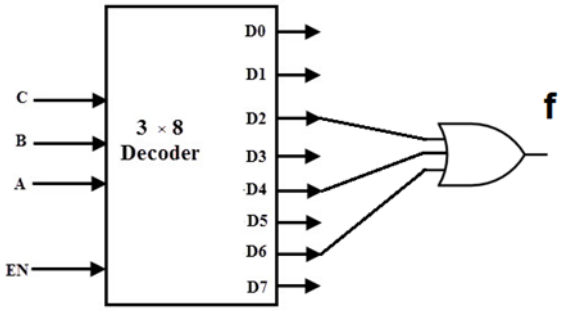
- **دیکـــدر[2] –اینکدر[3] –اینکدر با اولویت[4]**

برای آنکه تمامی مینترم‌های موردنیاز برای تولید یک تابع بولی را ایجاد نماییم از ساختاری بنام دیکدر استفاده می‌کنیم در دیکدر **n** ورودی داریم و 2^n خروجی خواهیم داشت هر خروجی متناظر با یک مینترم است که بر اساس آرایش بیت‌های ورودی تعیین می‌گردد. مثلا در شکل زیر یک دیکدر 2 به 4 ساخته شده است بر این اساس هرگاه ورودی **00** گردد که متناظر با مینترم $\overline{A_1}\,\overline{A_0}$ است آنگاه خروجی D_0 فعال (مساوی یک) خواهد شد و به همین صورت ورودی **01** متناظر با مینترم $\overline{A_1}A_0$ خواهد بود و باعث فعال شدن D_1 خواهد شد و ورودی **10** متناظر با مینترم $A_1\overline{A_0}$ خواهد بود و باعث فعال شدن D_2 خواهد شد و در نهایت ورودی **11** متناظر با مینترم A_1A_0 خواهد بود و باعث فعال شدن D_3 خواهد شد.



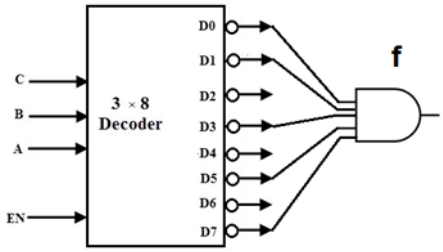
[2] **decoder**
[3] **encoder**
[4] **priority encoder**

دیکدر مولد مینترم است و اگر خروجی‌های آن را معکوس کنیم مولد ماکسترم خواهد بود. دیکدر با وردی 3 نیز در شکل نشان داده شده است بیت پرارزش ورودی در پایین قرار گرفته و بیت کم ارزش در بالا بیت پرارزش متناظر با متغیر سمت چپ خواهد بود. برای ساختن یک تابع مانند $f(A,B,C) = \sum m(2,4,6)$ کافی است خروجی‌های 2 و 4 و 6 را با هم **OR** کنیم.

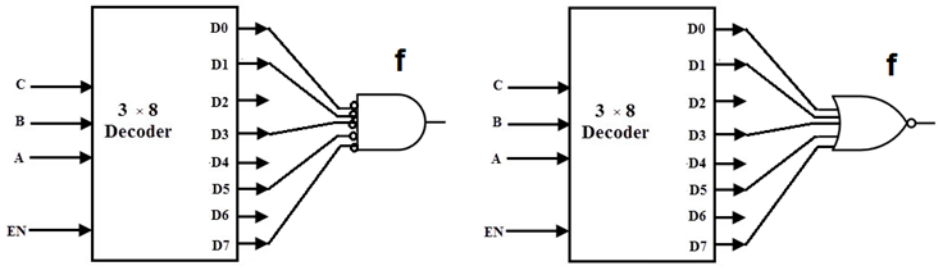


نکـــته: معمولا برای دیکدرها یک ورودی دیگر بنام **En** یا فعالساز قرار می‌دهند در صورتی که فعالساز برابر با **1** باشد دیکدر عملکرد طبیعی خود را خواهد داشت و اگر **0** باشد آنگاه تمامی پایه‌های خروجی دیکدر صرف نظر از ورودی برابر با **0** خواهد شد. چنین فعالسازی را با **EN** نشان می‌دهند و به آن **Active High** (فعالساز مثبت) می‌گویند در صورتی که آنرا با $\overline{EN}$ نشان دهیم به آن فعالساز منفی **Active Low** می‌گوییم و عملکرد آن برعکس فعالساز مثبت است.

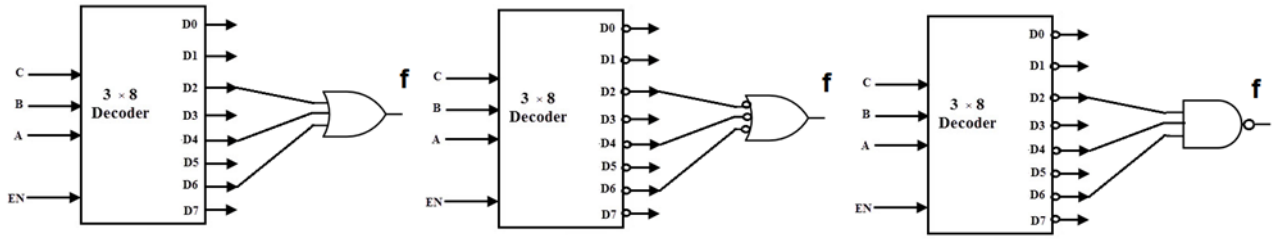
هر تابع بولی را می‌توان با استفاده از دیکدر و یک گیت **OR** یا **AND** یا **NAND** یا **NOR** پیاده‌سازی نمود. برای پیاده‌سازی با **OR** مینترم‌های معین شده در تابع را با هم **OR** می‌کنیم. برای پیاده‌سازی با **AND** معکوس تابع را در فرم **POS** (بصورت Π) پیاده‌سازی می‌کنیم . تبدیل یک تابع که بصورت Σ داده شده به فرم **Π** به این صورت است که از 2^n عدد تولید شده توسط **n** متغیر آن اعدادی که در Σ حضور ندارند در **Π** حضور خواهند داشت. مثلا تابع $f(a,b,c) = \sum m(2,4,6)$ معادل $f(a,b,c) = \prod M(0,1,3,5,7)$ خواهد بود زیرا از اعداد **0** الی **7** اعداد **2,4,6** در Σ حضور دارند و لذا اعداد **0,1,3,5,7** باید در **Π** باشند. برای پیاده‌سازی تابع با دیکدر و گیت **AND** بر سر خروجی‌های دیکدر گیت **NOT** بصورت دایره توخالی **(Bubble)** قرار داده و سپس اعداد متناظر با فرم **Π** تابع را از خروجی‌های معکوس شده دیکدر گرفته و با هم **AND** می‌کنیم.



برای پیاده‌سازی با گیت **NOR** می‌توانیم از این ایده استفاده کنیم که **NOT-AND** معادل **NOR** است لذا ابتدا مانند روش **AND** پیاده‌سازی کرده ولی **Bubble** ها رابه سمت گیت حرکت می‌دهیم و **NOT-AND** می‌سازیم سپس آنرا با **NOR** جایگزین می‌کنیم. پس فرم **II** تابع را با دیکدری که خروجی آن **NOT** نشده با گیت **NOR** پیاده‌سازی می‌کنیم.



برای پیاده‌سازی با گیت **NAND** ابتدا تابع را به همان شکل Σ و گیت **OR** در نظر می‌گیریم سپس به خروجی‌های دیکدر و پشت گیت **OR** گیت **NOT** بصورت **Bubble** قرار می‌دهیم قرار دادن دو **Bubble** پشت سر هم یعنی دو بار **NOT** کردن یک خروجی از نظر منطقی تغییری در آن ایجاد نمی‌کند سپس **NOT-OR** را به **NAND** تبدیل می‌کنیم

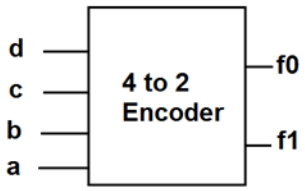


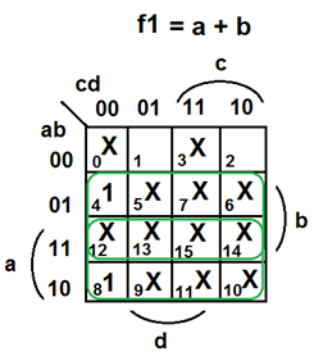
به بیان دیگر فرم Σ تابع را با دیکدری با خروجی‌های معکوس و گیت **NAND** پیاده‌سازی می‌کنیم.

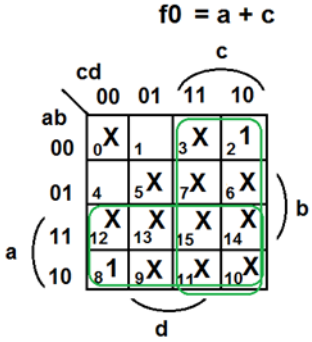
نکـــته: گاهی به دیکدر رمزگشا و به اینکدر رمزگذار نیز می‌گویند.

اینکدر برعکس دیکدر است و 2^n ورودی و **n** خروجی دارد. در اینکدر فرض بر آن است که همه ورودی‌ها به غیر از یک ورودی **0** هستند و آن یک ورودی **1** است. اینکدر بر اساس آنکه کدام ورودی آن **1** است شماره آن ورودی را در خروجی بصورت باینری نشان می‌دهد. به جدول زیر توجه نمایید.

abcd	f1	f0
0000	X	X
0001	0	0
0010	0	1
0011	X	X
0100	1	0
0101	X	X
0110	X	X
0111	X	X
1000	1	1
1001	X	X
1010	X	X
1011	X	X
1100	X	X
1101	X	X
1110	X	X
1111	X	X







این اینکدر دارای یک اشکال است در شرایطی که **a = 0** باشد ولی **b = 1** و **c = 1** باشد می‌تواند خروجی **11** تولید کند به معنی آنکه **a = 1** است و این غلط است اگرچه نباید بیش از یک ورودی **1** باشد ولی در عمل ممکن است اتفاق بیفتد. برای رفع این مشـــکل باید تغییری در آن ایجاد کرد که آنرا به یک اینکدر اولویت[5] تبدیل خواهد کرد. فلسفه اینکدر اولویت آنست که زمانی که ورودی **a=1** باشد صرف نظر از بقیه ورودی‌ها خروجی باید بصورت **11** باشد و اگر **b = 1** باشد صرف نظر از بقیه ورودی‌ها خروجی باید بصورت **10** باشد و اگر **c = 1** باشد صرف نظر از بقیه ورودی‌ها خروجی باید بصورت **01** باشد و هرگاه **d = 1** شد صرف نظر از بقیه ورودی‌ها خروجی باید بصورت **00** باشد. جدول و معادلات این اینکدر به صورت زیر خواهد بود.

a	b	c	d	f1	f0	valid
1	x	x	x	1	1	1
0	1	x	x	1	0	1
0	0	1	x	0	1	1
0	0	0	1	0	0	1
0	0	0	0	x	x	0

$f0 = a + \bar{b}c$

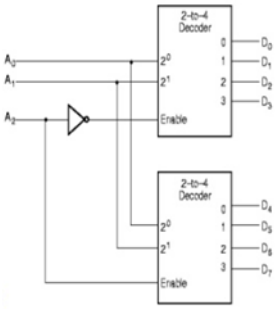
$f1 = a + b$

$valid = a + b + c + d$

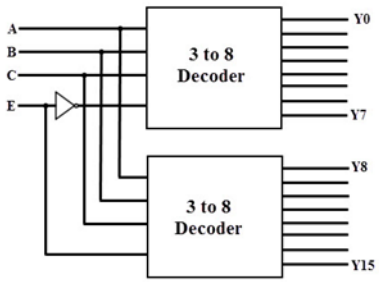
[5] **Priority encoder**

می‌توان یک خروجی با عنوان **valid** به معنای معتبر نیز تولید نمود که هرگاه یک شود خروجی معتبر است.

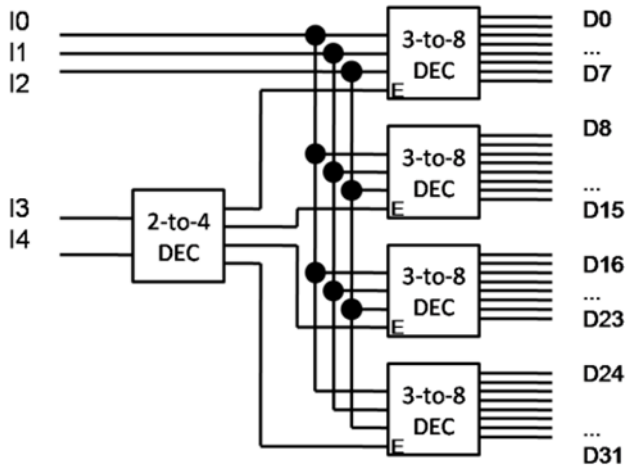
برای توسعه دیکدرها از ورودی **En** یا فعالساز استفاده می‌کنیم مثلا اگر بخواهیم با دو دیکدر 2 به 4 یک دیکدر 3 به 8 بسازیم باید ورودی سوم (پرارزش) را بصورت مستقیم به فعالساز دیکدر پایینی وصل کرده و نقیض آن را به فعالساز دیکدر بالایی وصل نماییم در این صورت هرگاه در نیمه بالایی جدول باشیم و ورودی پرارزش **0** باشد



دیکدر بالایی فعال خواهد بود و بر اساس دو ورودی دیگرش تصمیم خواهد گرفت و اگر در نیمه پایینی جدول باشیم و ورودی سوم **1** باشد آنگاه دیکدر بالایی غیرفعال و دیکدر پایینی فعال خواهد بود. توسعه دیکدرها بر همین اساس است بطور مثال با دو دیکدر 3 به 8 می‌توانیم یک دیکدر 4 به 16 ایجاد کنیم.



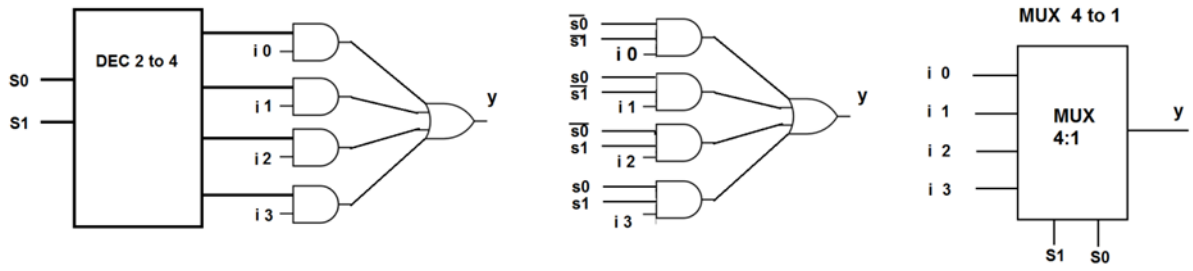
هرگاه یک دیکدر 2 به 4 به همراه 4 دیکدر 3 به 8 داشته باشیم می‌توانیم یک دیکدر 5 به 32 بسازیم کافی است فعالسازهای دیکدرهای 3 به 8 را از خروجی‌های دیکدر 2 به 4 بگیریم.



5-to-32 line decoder

- **مالتی‌پلکسر[6] – دی‌مالتی‌پلکسر[7]**

هرگاه هر خروجی یک دیکدر را مستقلا با یک ورودی مستقل **AND** کنیم سپس تمامی خروجی‌های بدست آمده را **OR** نماییم (شکل سمت چپ) به یک ساختار جدید دست می‌یابیم متناظر با ورودی دیکدر یکی از خروجی‌ها **1** می‌شود و همان**1** باعث می‌شود که ورودی کنار آن از گیت **AND** عبور نموده و به گیت **OR** برسد این ساختار جدید مالتی‌پلکسر نام دارد و اگر بخواهیم آنرا ساده‌تر بساریم می‌توانیم ورودی ها را به عنوان ورودی سوم گیت‌های **AND** اعمال کنیم (شکل وسط) و در نهایت یک مالتی‌پلکسر خواهیم داشت (شکل سمت راست) که بر اساس خطوط انتخابگر **S1 S0** یکی از ورودی‌ها را در خروجی اعمال می‌نماید.



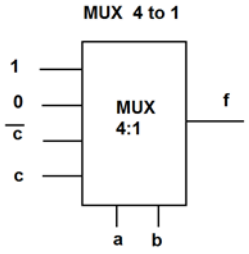
مالتی‌پلکسرها در معماری رایانه کاربردی وسیع دارند به نحوی که می‌توانند خروجی‌های مختلف را مدیریت نموده و در زمان دلخواه به سیستم اعمال نمایند. مالتی‌پلکسری که دارای **n** خط انتخاب است می‌تواند از میان 2^n ورودی یکی را انتخاب کرده و در خروجی قرار دهد.

[6] **Multiplexer = mux**
[7] **Demultiplexer = DEMUX**

توسعه مالتی‌پلکسرها نیز با استفاده از ورودی فعالساز امکان‌پذیر است. برای تجمیع خروجی دو مالتی‌پلکسر از گیت **OR** استفاده می‌کنیم.

مالتی‌پلکسرها برای پیاده‌سازی توابع نیز استفاده می‌شوند برای مثال برای پیاده‌سازی یک تابع سه متغیره می‌توان از یک مالتی‌پلکسر 4 به 1 با دو خط انتخاب استفاده کرد دو متغیر پر ارزش را به خطوط انتخاب متصل می‌کنیم و ورودی های مالتی‌پلکسر بر اساس خروجی تابع تعیین می‌شود یا **0** است یا **1** است یا معادل متغیر سوم است یا معادل نقیض متغیر سوم است.

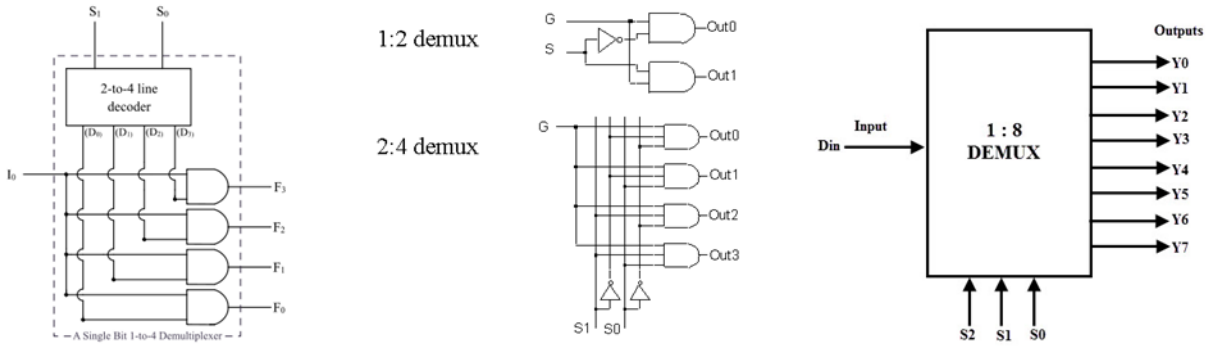
مثال تابع $f(a,b,c) = \sum m(0,1,4,7)$ را با یک مالتی‌پلکسر 4:1 پیاده‌سازی کنید.



a b c	f	
0 0 0	1	1
0 0 1	1	
0 1 0	0	0
0 1 1	0	
1 0 0	1	c̄
1 0 1	0	
1 1 0	0	c
1 1 1	1	

دی‌مالتی‌پلکسر بر عکس مالتی‌پلکسر عمل می‌کند و تعیین می‌کند ورودی تکی باید روی کدام خروجی قرار گیرد ساخت دی‌مالتی‌پلکسر از روی دیکدر بسیار آسان است کافی است به هر خروجی آن یک گیت **AND** اضافه نموده و ورودی دی‌مالتی‌پلکسر را به آن اعمال کنیم در این صورت هرگاه بر اساس خطوط انتخاب یکی از خروجی‌ها فعال (مساوی **1**) شود همان خروجی با ورودی اصلی **AND** می‌گردد و بقیه خروجی‌ها **0** خواهند بود.

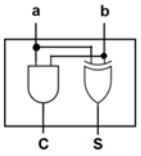
در شکل زیر در سمت چپ دی‌مالتی‌پلکسر 4 به 1 با استفاده از دیکدر ساخته شده و در وسط با استفاده از گیت دی‌مالتی‌پلکسر 4 به 1ساخته شده و در سمت راست یک دی‌مالتی‌پلکسر 1 به 8 را بصورت بلاک نشان داده شده است.



- **جمع‌کننده**

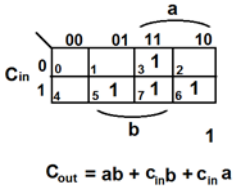
فرض کنید می‌خواهیم دو بیت را با هم جمع کنیم برای این کار یک مدار طراحی می‌کنیم که نیم جمع‌کننده[8] نام دارد. نیم جمع کننده دارای دو ورودی و دو خروجی است خروجی دوم همان رقم نقلی است که در اثر جمع دو یک بدست می‌آید.

a	b	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

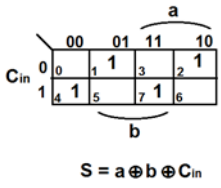


همانگونه که مشاهده می‌گردد ساخت نیم‌جمع کننده با استفاده از یک گیت **AND** (برای تولید رقم نقلی) و یک گیت **XOR** برای تولید حاصلجمع امکان‌پذیر است. حال اگر بخواهیم یک جمع کننده داشته باشیم که رقم نقلی ورودی نیز دریافت نماید باید عمل جمع را روی سه متغیر انجام دهیم و به آن تمام جمع‌کننده[9] می‌گوییم. لذا دو تابع 3 ورودی ایجاد خواهد شد که یکی حاصلجمع و دیگری رقم نقلی را تولید می‌نماید مقادیر را از روی جدول درستی برداشته و در جداول کارنو وارد می‌نماییم تا بتوانیم توابع ساده شده را بدست بیاوریم.

C_{in}	a	b	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



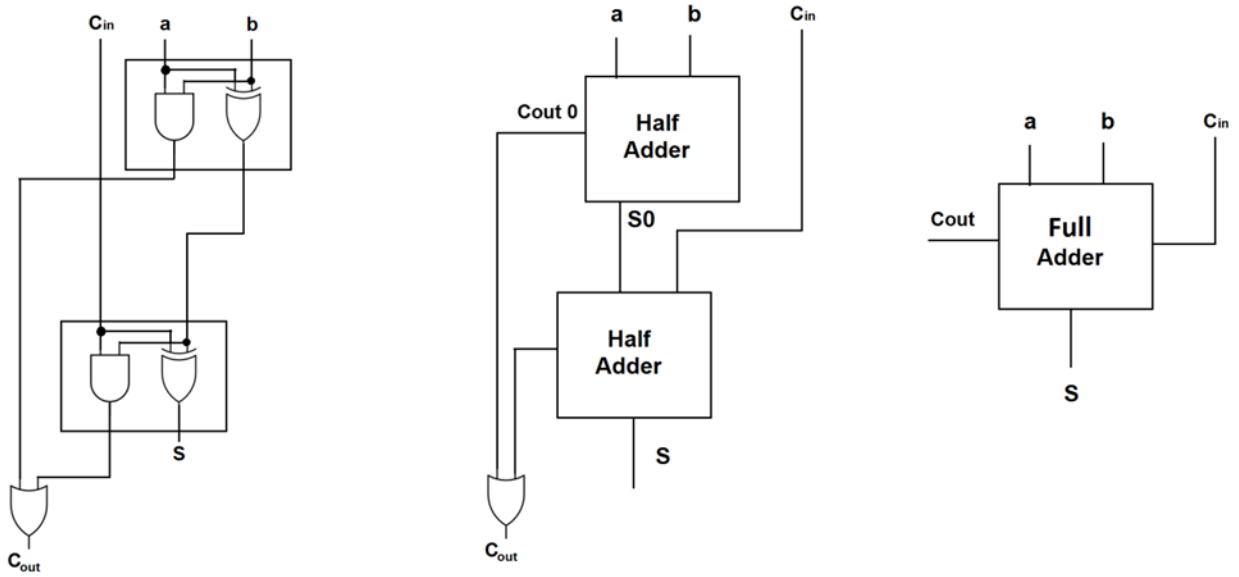
$C_{out} = ab + c_{in}b + c_{in}a$



$S = a \oplus b \oplus C_{in}$

[8] **Half adder**
[9] **Full adder**

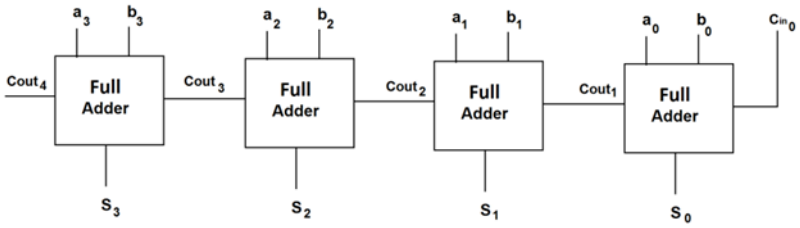
همانگونه که جدول نشان می‌دهد تابع **S** یک طرح شطرنجی است و لذا ساده نمی‌شود طرح های شطرنجی یا **XOR** هستند (اگر خانه یک مقدار 1 داشته باشد) در غیر آن صورت **XNOR** هستند. در شکل فوق تابع مورد نظر **XOR** سه ورودی تمام جمع‌کننده است. می‌توانیم تمام جمع‌کننده را با استفاده از دو نیم‌جمع‌کننده ساخت تابع بولی حاصل از آن دقیقا معادل تابع ایجاد شده در جدول می‌باشد. در شکل زیر (سمت چپ) یک تمام جمع‌کننده با استفاده از دو نیم جمع‌کننده ساخته شده در شکل وسط همان طراحی بصورت بلوک دیاگرام نشان داده شده و در در سمت راست بلوک دیاگرام یک تمام جمع‌کننده نشان داده شده است.



نکـــته: می‌توان به جای گیت **OR** مولد **Cout** یک گیت **XOR** قرار داد. زیرا هیچگاه **ab** و **a xor b** با هم یک نمی‌شوند.

در اثر اتصال تمام جمع‌کننده‌ها به هم جمع‌کننده چند بیتی ساخته می‌شود. در شکل زیر یک جمع‌کننده چهاربیتی با استفاده از چهار جمع‌کننده یک بیتی ساخته شده است به آن جمع‌کننده موج‌گونه[10] می‌گویند زیرا رقم نقلی مانند موج از سمت راست به چپ حرکت می‌کند. تاخیر چنین جمع‌کننده‌ای به اندازه مجموع تاخیر تمام‌جمع‌کننده‌ها است زیرا برای آنکه رقم جمع نهایی محاسبه شود باشد باید رقم نقلی به آن برسد. برای آنکه بتوانیم بر این مشکل غلبه کنیم باید از جمع‌کننده دیگری با قابلیت پیش‌بینی رقم نقلی[11] استفاده نماییم.

[10] **Ripple carry adder**
[11] **Carry lookahead adder**



برای پیش‌بینی رقم نقلی ابتدا دو مولفه مولد و منتشر کننده را تعریف می‌کنیم. مولد رقم نقلی بصورت A_iB_i می‌باشد و منتشرکننده رقم نقلی بصورت $A_i \oplus B_i$ خواهد بود. اکنون می‌توانی بدون آنکه منتظر باشیم رقم نقلی محاسبه شود رقم نقلی هر طبقه را مستقلا محاسبه کنیم. اشکال این روش آنست که اولا سخت افزار بیشتری نیاز دارد و ثانیا برای رقم نقلی طبقه **n** ام نیاز به یک گیت با **n+1** ورودی داریم که باید آنرا با گیت‌های دیگری پیاده‌سازی کنیم گیت‌های با ورودی زیاد اصلا مطلوب نیستند.

$P_i = A_i \oplus B_i$ Carry propagate
$G_i = A_iB_i$ Carry generate

$S_i = P_i \oplus C_i$
$C_{i+1} = G_i + P_iC_i$

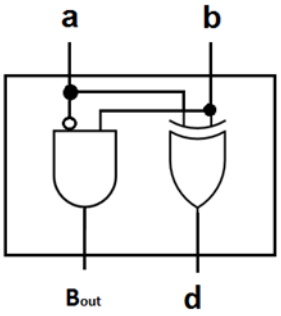
$C_1 = G_0 + P_0C_0$
$C_2 = G_1 + P_1C_1 = G_1 + P_1(G_0 + P_0C_0) = G_1 + P_1G_0 + P_1P_0C_0$
$C_3 = G_2 + P_2C_2 = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0$
$C_4 = G_3 + P_3C_3 = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0C_0$

- **تفریق‌کننده**

برای ساخت تفریق‌کننده[12] (تفریق‌گر) دوبیتی ابتدا جدول درستی آن را می‌نویسیم باید توجه داشته باشیم که همانگونه که در جمع‌کننده رقم نقلی داریم در تفریق‌کننده رقم قرضی دارم اگر در تفریق **a-b** رقم قرضی برابر با **1** شود به آن معنی است که **a** از **b** کوچکتر است و باید از طبقه بعدی خود قرض بگیرد در این صورت عدد **2** به **a** اضافه می‌شود و تفریق بصورت **a-b+2** خواهد بود. در شکل زیر جدول درستی تفریق‌کننده دو بیتی نشان داده شده در مورد تفریق کننده سه بیتی که دارای رقم قرضی ورودی است هرگاه رقم قرضی ورودی یک شود به معنی آنست که این طبقه به طبقه قبل از خودش قرض داده است لذا **a-b** بصورت **a-b-1** خواهد بود.

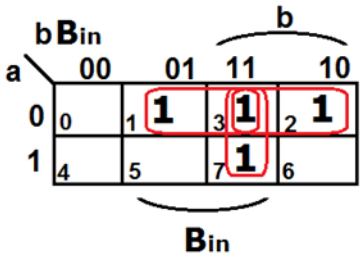
[12] **subtractor**

a	b	B_{out}	d
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

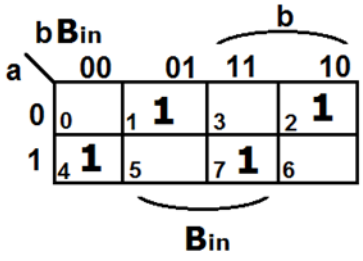


در شکل زیر جدول درستی و توابع تمام تفریق‌کننده نشان داده شده است.

a	b	B_{in}	B_{out}	D
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

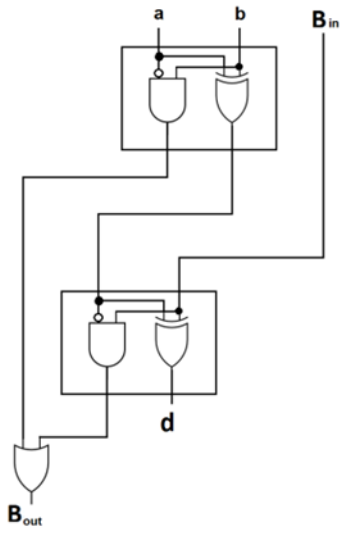


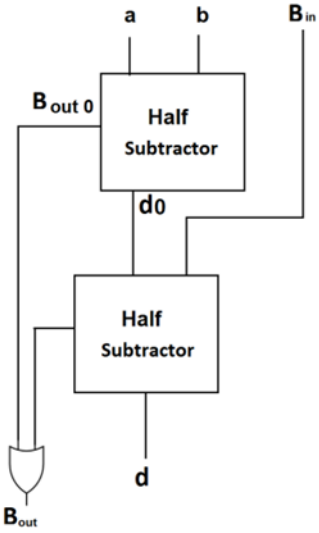
$$B_{out} = \bar{a}B_{in} + \bar{a}b + bB_{in}$$

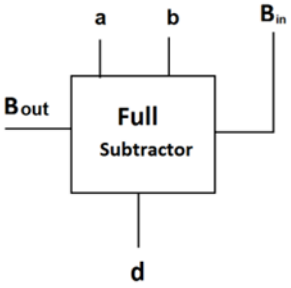


$$D = a \oplus b \oplus B_{in}$$

مانند قبل می‌توان با استفاده از دو نیم تفریق کننده یک تمام تفریق‌کننده ساخت. می‌توان نشان داد که به‌جای **OR** مولد **Bout** می‌توان **XOR** قرار داد.

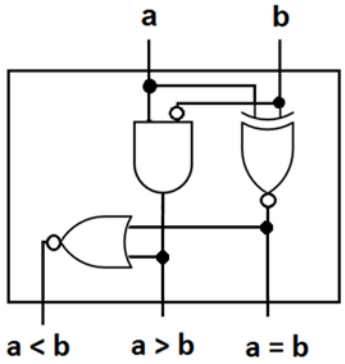


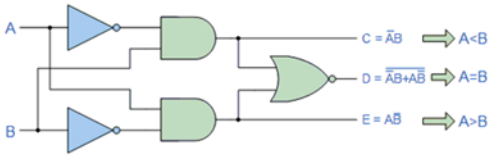




• مقایسه‌گر

برای مقایسه دو بیت می‌توان از یک گیت **XNOR** استفاده کرد اگر دو بیت برابر باشند حاصل برابر با **1** خواهد بود و اگر برابر نباشند حاصل **0** خواهد بود. برای آنکه تشخیص دهیم کدام بیت **a** یا **b** بزرگتر است از یک گیت **And** و گیت **NOT** استفاده خواهیم کـــرد. در شکل زیر دو طراحی برای مقایسه‌گر تک بیت ارائه شده.



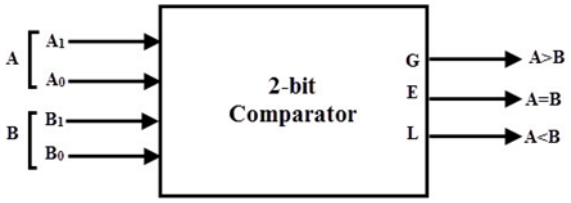


طراحی مقایسه‌گرها با ورودی بالاتر نیز از همین اصل استفاده می‌کنیم. بطور مثال مقایسه‌گر 2 بیتی (برای مقایسه دو عدد دو بیتی) بصورت زیر خواهد بود.

$$L = b_1\overline{a_1} + (a_1 \odot b_1)\, b_0\overline{a_0}$$

$$E = (a_1 \odot b_1)\,(a_0 \odot b_0)$$

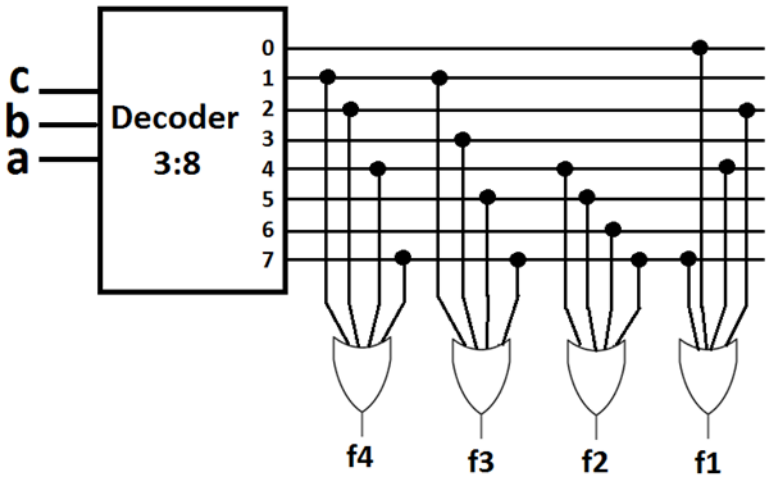
$$G = \bar{L}\bar{E}$$



• حافظه‌ها

همانگونه که در خصوص دیکـدر نشان داده شد، یک دیکدر مولد کلیه مینترم‌ها می‌باشد به این معنی که می‌توان با انتخاب مینترم‌های مناسب و انجام عمل **OR** تمامی توابع منطقی را ایجاد نمود. هرگاه خروجی‌های دیکدر بشکل زیر به گیت‌های **OR** مجزا متصل گردد چندین تابع منطقی بطور همزمان تولید می‌گردد می‌توان

تصور کرد که موجودیت ایجاد شده مانند یک حافظه است زیرا به ازای هر ترکیب ورودی (آدرس یا ردیف) یک مقدار چهاربیتی در خروجی قرار می‌دهد در واقع شکل زیر یک حافظه 8 در 4 است که به ازای هریک از 8 آدرس **000** الی **111** مقدار مشخصی را در خروجی ایجاد می‌کند. مثلا در آدرس **101** مقدار **0110** قرار دارد.



a b c	f4	f3	f2	f1
0 0 0	0	0	0	1
0 0 1	1	1	0	0
0 1 0	1	0	0	1
0 1 1	0	1	0	0
1 0 0	1	0	1	1
1 0 1	0	1	1	0
1 1 0	0	0	1	0
1 1 1	1	1	1	1

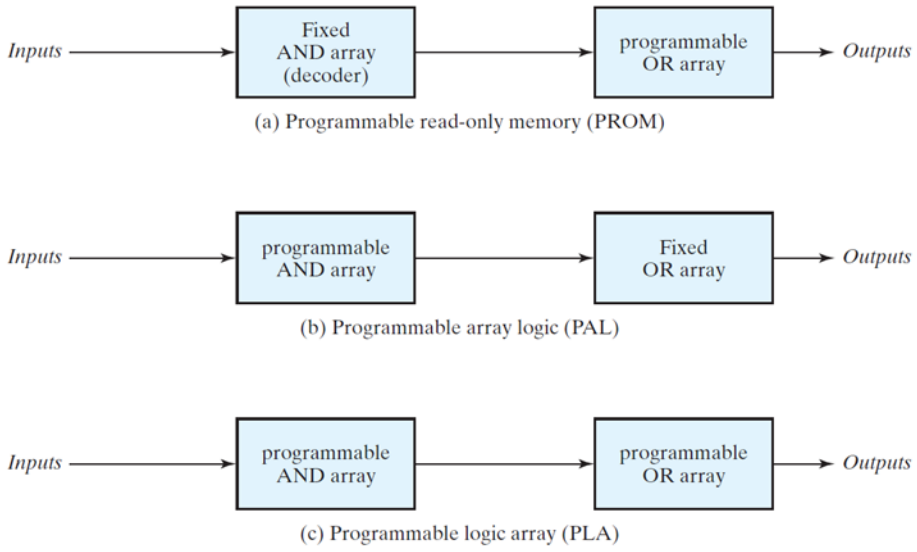
حافظه‌ها به چند دسته تقسیم می‌گردند. یک نوع تقسیم‌بندی بر اساس قابلیت نگهداری اطلاعات بدون اتصال به منبع تغذیه است اگر حافظه بتواند پس از قطع شدن جریان الکتریسیته (منبع تغذیه) اطلاعات خود را نگهداری نماید به آن حافظه غیر فـــرار[13] می‌گویند در صورتی که پس از قطع شدن جریان الکتریسیته (منبع تغذیه) اطلاعات پاک شود به آن حافظه فرار[14] می‌گویند. حافظه‌ها با توجه به نحوه برنامه‌ریزی (نوشتن در خانه‌های حافظه) نیز به چند دسته تقسیم می‌شـــوند. اولـــین دسته حافظه **ROM** نامیده می‌شـــود که مخفف **Read Only Memory** یا حافظه فقط خواندنی است به این معنا که نگارش داده‌ها در خانه‌های حافظه بخشی از فرآیند ساخت بوده و پس از ساخته شدن در کارخانه دیگر امکان تغییر و برنامه‌ریزی مجدد آن وجود ندارد. نوع دیگر حافظه **PROM** نام دارد که مخفف **Programable ROM** است یعنی قابلیت برنامه‌ریزی توسط کاربر وجود دارد وکاربر می‌تواند بر اساس نیاز خود مقادیر دلخواه را در خانه‌های مورد نظر خود در حافظه قرار دهد. نوع دیگر حافظه **EPROM** نام دارد که مخفف **Erasable PROM** (به معنی **PROM** با قابلیت پاک شدن) است یعنی مانند حافظه **PROM** که قابلیت برنامه‌ریزی توسط کاربر وجود دارد ولی امکان پاک کردن اطلاعات و برنامه‌ریزی مجدد وجود دارد. برای پاک کردن حافظه **EPROM** معمولا آن را در معرض اشعه فرابنفش قرار می‌دهند. نوع دیگری از حافظه‌ها **EEPROM** نام دارد که مخفف **Electrically EPROM** است

[13] **Non volatile**
[14] **Volatile**

یعنی حافظه مانند **EPROM** قابلیت برنامه‌ریزی مجدد دارد با این تفاوت که پاک کردن از طریق اعمال سیگنال الکتریکی صورت می‌پذیرد. امروزه حافظه فلش جایگزین موارد فوق شده است حافظه فلش دارای این قابلیت است که برای برنامه‌ریزی مجدد نیاز نیست کل حافظه پاک شود و می‌توان به تک تک بایت‌های آن دسترسی مستقل داشت.

- **آرایه‌های منطقی برنامـــه‌پذیـــر**

همانگونه که در خصوص دیکدر بیان شد، یک دیکدر تمامی مینترم‌ها را تولید می‌نماید دیکدر با گیت **AND** پیاده‌سازی می‌گردد لذا در پیاده‌سازی با دیکدر **(PROM)** قسمت **AND** ثابت است و با انتخاب مین‌ترم‌های مختلف در قسمت **OR** برنامه‌ریزی صورت می‌گیرد لذا قسمت **OR** برنامه‌پذیر است. اگر بخواهیم یک تابع را که حالات بی اهمیت زیادی دارد با دیکـــدر پیاده‌سازی نماییم با اتلاف مواجه می‌شویم از این جهت ساختارهای دیگری را مانند **PLA**[15] و **PAL**[16] مورد بررسی قرار خواهیم داد. در **PAL** قسمت **AND** برنامه‌پذیر است و قسمت **OR** ثابت است در **PLA** هر دو قسمت **AND** و **OR** برنامه‌پذیر هستند در شکل زیر تفاوت **PAL**،**PLA** و **PROM** نشان داده شده است. در **PLA** از جملات مشترک استفاده می‌شود ولی در **PAL** از جملات مشترک استفاده نمی‌شود. در **PAL** گاهی از یک تابع خروجی برای ساخت توابع دیگر استفاده می‌شود.

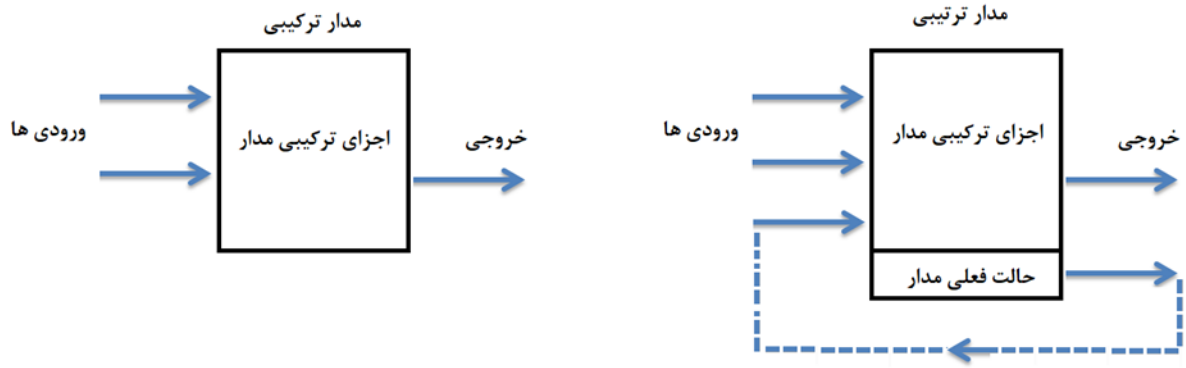


[15] **Programmable Logic Array**
[16] **Programmable Array Logic**

• فلیپ‌فلاپ‌ها و مدارات ترتیبی

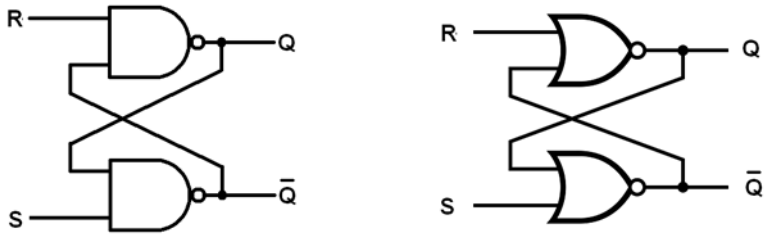
مداراتی که تاکنون به آنها پرداختیم مدارات ترکیبی بودند به آن معنی که خروجی مدار تابع ترکیب ورودی‌های آن است. از این به بعد با وارد کردن عناصر حافظه‌دار مانند فلیپ‌فلاپ‌ها مدارات ترتیبی مورد بحث قرار خواهند گرفت که در آن، خروجی مدار تابع ترکیب ورودیها و حالت مدار می‌باشد.

در شکل زیر تفاوت مدارات ترکیبی و ترتیبی نشان داده شده است



ساده‌ترین عنصر حافظه‌دار لچ اس آر **(SR Latch)** نام دارد ومی توان آن را مطابق شکل زیر با گیت‌های **NAND** یا **NOR** ساخت

نکـــته: هرگاه یکی از پایه‌های **NAND** برابر با **0** شود خروجی **NADN** برابر با **1** خواهد بود و هرگاه یکی از پایه‌های گیت **NOR** برابر با **1** شود خروجی آن گیت **0** خواهـــد شد.



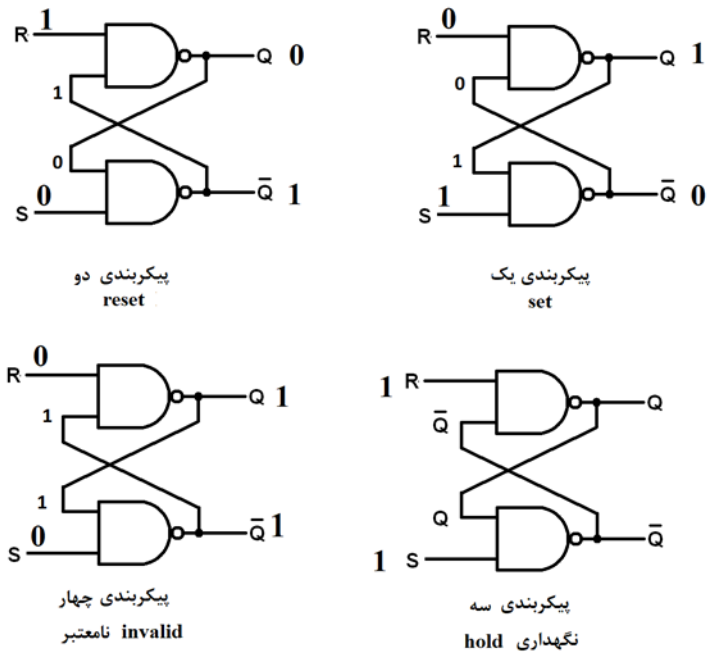
دقت نمایید که **Q** خروجی اصلی است و $\bar{\mathbf{Q}}$ نقیض (معکوس منطقی) **Q** است. برای هریک از لچ ها چهار حالت را بررسی می‌کنیم هریک از چهار حالت یک پیکربندی **(set up)** نام دارد. برای لچ سمت چپ (طراحی شده با گیت **NAND**) خواهیم داشت

پیکربندی شماره یک : **S = 1 , R = 0** را پیکربندی را **set** می‌نامیم زیرا خروجی مساوی یک است **Q = 1**

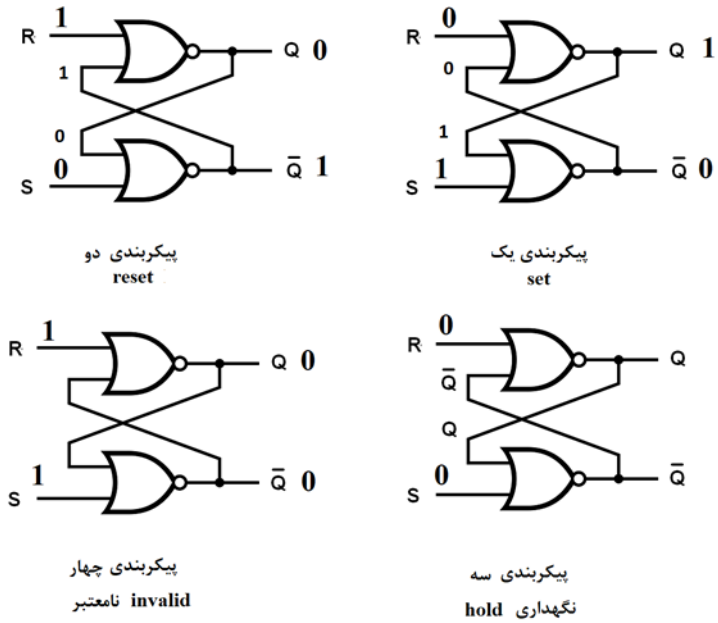
پیکربندی شماره دو : **S = 0 , R = 1** را پیکربندی را **reset** می‌نامیم زیرا خروجی مساوی صفر است **Q = 0**

پیکربندی شماره سه : **S = 1 , R =1** را پیکربندی را **hold** می‌نامیم زیرا سیستم حالت خود را حفظ می‌کند

پیکربندی شماره چهار: **S = 0 , R = 0** را پیکربندی را **invalid** می‌نامیم زیرا $\mathbf{Q \neq \bar{Q}}$ و نادرست خواهد بود. در حالت آخر هر دو خروجی برابر با **1** خواهد بود که درست نمی‌باشد نباید ورودی در این حالت قرار گیرد.

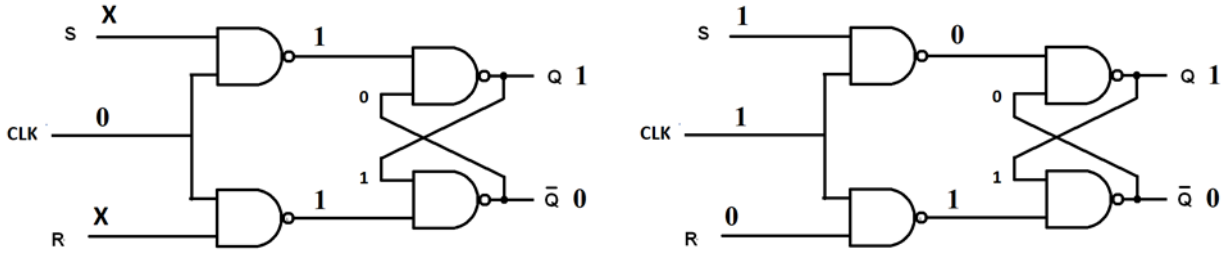


برای شکل سمت راست (طراحی شده با گیت **NOR**) شرایط **set** و **reset** مشابه خواهند بود با این تفاوت که حالت نگهداری **S = 0 R=0** خواهد بود و حالت غیر مجاز یا نامعتبر **S = 1 R=1** می‌گردد.

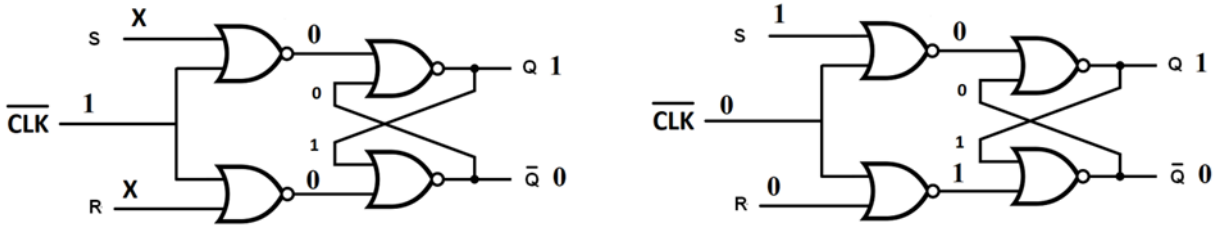


هـــرگاه مطابق شکل دو گیت NAND دیگر به ورودی اضافه کنیم و یک ورودی دیگر به لچ اضافه کنیم آنگاه یک فلیپ فلاپ پایه SR ساخته‌ایم ورودی جدید کلاک حساس به سطح نام دارد و با CLK نشان داده می‌شود. هرگاه CLK = 0 باشد ورودی میانی بصورت حالت hold خواهد بود و هر مقدار در Q باشد همان مقدار ثابت خواهد بود و ورودی هر تغییری نماید تاثیری در خروجی نخواهد داشت (مانند سمت جپ) اگر مانند شکل سمت راست CLK=1 شود آنگاه ورودی اثر خواهد کرد به دلیل آنکه ورودی از یک گیت NAND عبور می‌کند که یکی از پایه‌های آن یک است و مانند NOT عمل می‌کند جای S و R در شکل زیر نسبت به لچ تغییر نموده است. عملکرد فلیپ فلاپ پایه به این صورت است که هرگاه ورودی کلاک (حساس به سطح) صفر باشد ورودی از خروجی جدا شده و فلیپ فلاپ مقدار قبلی خود را حفظ می‌کند و نسبت به تغییرات ورودی بی تفاوت است ولی زمانی که کلاک برابر با یک می‌شود ورودی موثر واقع شده و مطابق پیکربندی فلیپ فلاپ set یا reset می‌شود. به دلیل آنکه کلاک در حالت 1 موثر است به آن کلاک حساس به سطح مثبت[17] می‌گوییم.

[17] **Positive level trigger**

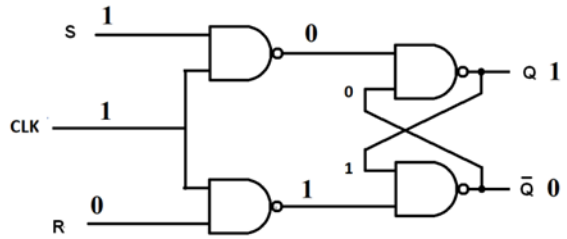


میتوان مانند شکل زیر فلیپ فلاپ پایه را با گیت NOR ساخت تفاوت در آنست که برای آنکه ورودی موثر واقع شود کلاک باید 0 باشد هرگاه کلاک برابر با 1 باشد در ورودی میانی حالت hold ایجاد می‌شود و لذا ورودی از خروجی جدا شده و فلیپ فلاپ مقدار قبلی خود را حفظ می‌کند.

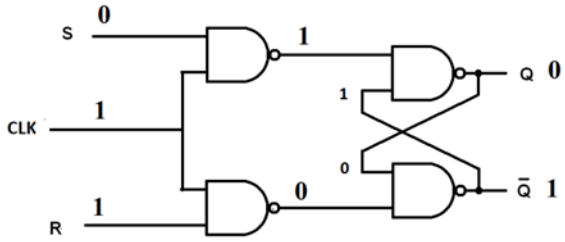


فلیپ فلاپ SR

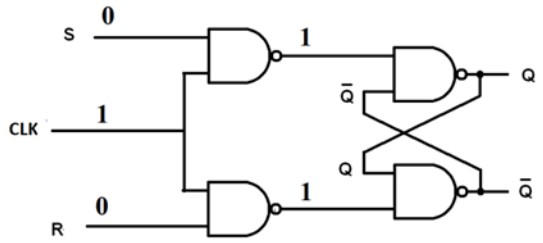
فلیپ فلاپ SR پایه که بر اساس گیت NAND ساخته شده در شکل زیر نشان داده شده است. همانطور که قبلا مرور شد پیکربندی S = 1 , R = 0 را پیکربندی SET می‌گوییم که در آن زیرا **Q = 1** و $\bar{\mathbf{Q}}$ = 0 خواهد شد و پیکربندی S = 0 , R = 1 را پیکربندی RESET می‌گوییم که در آن زیرا **Q = 0** و $\bar{\mathbf{Q}}$ = 1 خواهد شد و پیکربندی S = 0 , R = 0 را پیکربندی HOLD می‌گوییم زیرا مقدار قبلی حفظ می‌گردد و در نهایت پیکربندی S = 1 , R = 1 را پیکربندی INVALID یا نامعتبر یا غیرمجاز می‌گوییم زیرا در آن زیرا **Q = 1** و $\bar{\mathbf{Q}}$ = 1 خواهد شد و دیگر که **Q** نقیض $\bar{\mathbf{Q}}$ نخواهد شد. باید توجه داشت که در هریک از پیکربندی‌ها هرگاه **CLK=0** شود حالت **HOLD** رخ خواهد داد و دیگر تغییر ورودی در خروجی اثر نخواهد کرد.



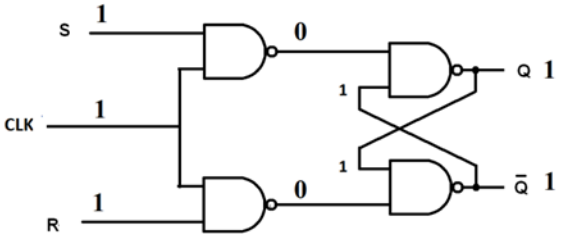
SET



RESET



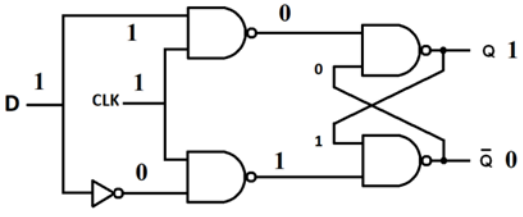
HOLD



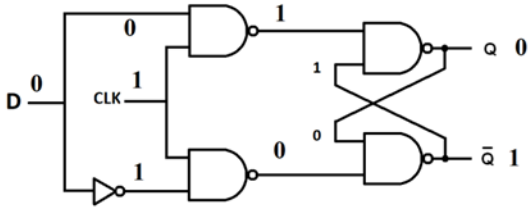
INVALID

فلیپ فلاپ D

فلیپ فلاپ SR پایه یک حالت غیر معتبر یا غیر مجاز دارد در فلیپ ساخته شده با گیت NAND هرگاه دو ورودی با هم 1 شوند حالت غیر مجاز رخ می‌دهد و در فلیپ ساخته شده با گیت NOR هرگاه دو ورودی با هم 0 شوند حالت غیر مجاز رخ می‌دهد زیرا **Q** دیگر نقیض (معکوس منطقی) $\bar{\mathbf{Q}}$ نخواهد بود. هرگاه یک ورودی اصلی در نظر بگیریم و آن را به **S** متصل نماییم و نقیض آنرا به **R** متصل نماییم هرگز مشکل حالت غیر مجاز رخ نخواهد داد هرگاه ورودی اصلی را **D** بنامیم آنگاه **D = 1** مانند حالت **set** و **D = 0** مانند حالت **reset** عمل خواهد کرد و یک فلیپ فلاپ **D** ساخته‌ایم. (مطابق شکل)



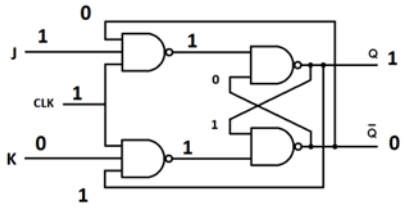
SET



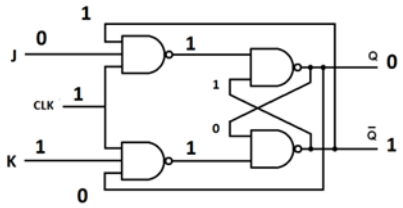
RESET

فلیپ فلاپ JK

هـــرگاه در فلیپ فلاپ SR خـــروجی $\overline{Q}$ را به NAND ورودی بالایی و خروجی Q را به NAND ورودی پایین متصل نماییم فلیپ فلاپ JK ساخته می‌شود پایه ورودی بالایی را J (حرف اول JUMP به معنی پرش) و ورودی پایینی را K (حرف اول KILL به معنی فرونشاندن) پیکربندی JUMP به صورت J = 1 K = 0 می‌باشد که در آن Q = 1 و عمل پرش و یک شدن صورت گرفته و پیکربندی KILL به صورت J = 0 K = 1 می‌باشد که در آن Q = 0 خـــواهد شد و عمل KILL فـــرونشاندن صورت گرفته است. هـــرگاه هر دو ورودی 0 باشد پیکربندی HOLD رخ می‌دهد و هرگـاه هر دو ورودی 1 باشد اصطلاحاً TOGGLE رخ می‌دهد که خروجی معکوس می‌شود. البته عمل toggle با CLK حساس به سطح درست عمل نخواهد کرد.



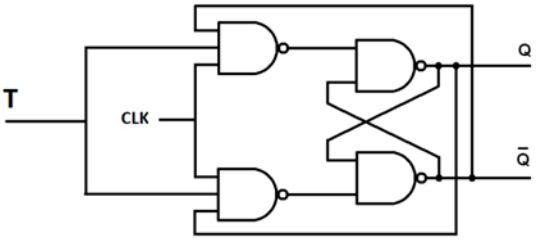
JUMP



KILL

فلیپ فلاپ T

هـــرگاه در فلیپ فلاپ JK هـــر دو ورودی J و K را به هم متصل نماییم فلیپ فلاپ T بدست می‌آید. هرگاه ورودی آن 1 باشد خروجی معکوس (اصطلاحا toggle) شده و هرگاه ورودی آن 0 باشد خروجی همان قبل باقی می‌ماند. در ادامه خواهیم دید که فلیپ‌فلاپ T با CLK حساس به سطح به درستی کار نخواهد کرد.

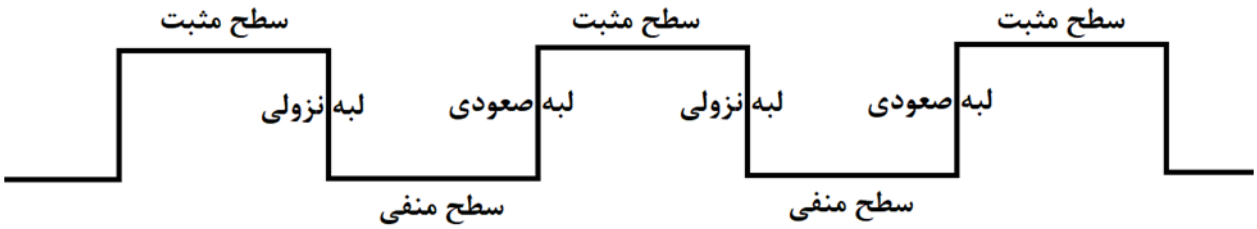


پالس ساعت (کلاک پالس[18])

در بخش‌های قبل در خصوص وروری CLK بحث نمودیم و اینکه هرگاه ورودی CLK = 1 باشد ورودی در خروجی تاثیر خواهد گذاشت اما سیگنال CLK در مثالهای ما حساس به سطح مثبت است یعنی هرگاه مقدار

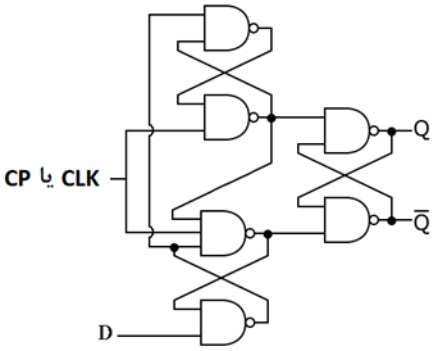
[18] **clock pulse**

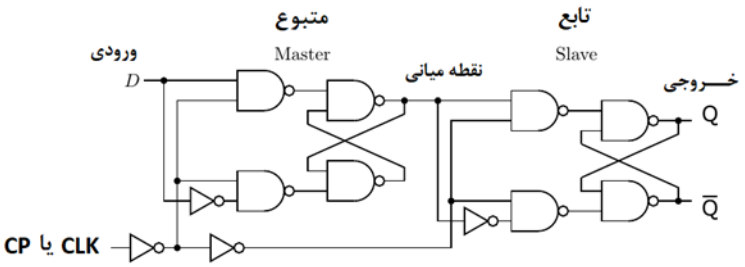
آن در سطح یک منطقی باشد ورودی در خروجی تاثیرگذار خواهد بود و با صفر شدن CLK ورودی از خروجی جدا شده و فلیپ فلاپ مقدار قبلی خود را حفظ خواهد کرد. در دو فلیپ فلاپ JK و T مشاهده کردیم که هرگاه هر دو ورودی JK برابر با یک منطقی باشد یا معادلا ورودی T = 1 باشد فلیپ فلاپ مقدار فعلی خود را معکوس خواهد کرد اما وجود CLK = 1 و حساس به سطح باعث خواهد شد که این عمل بی نهایت بار تکرار شود و فلیپ فلاپ عملکرد درستی نداشته باشد. برای آنکه رفع این مشکل فلیپ فلاپ باید بتواند در یک لحظه مشخص و بسیار کوتاه ورودی را دریافت نموده و عملکرد خود را نشان دهد. سیگنال پالس ساعت یک موج مربعی مانند شکل زیر است که هرگاه مقدار آن 0 باشد به آن سطح منفی و هرگاه مقدار آن 1 باشد به آن سطح مثبت اطلاق می‌شود و لحظه تبدیل از سطح منفی به سطح مثبت لبه صعودی[19] و لحظه تبدیل از سطح مثبت به سطح منفی لبه نزولی[20] نام دارد.



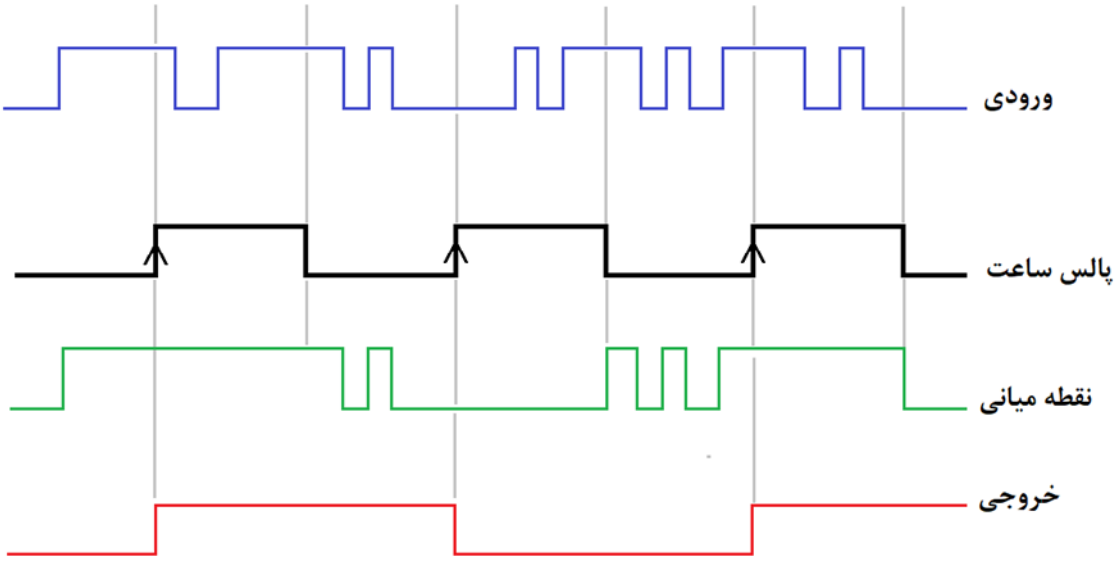
اگر بخواهیم فلیپ فلاپ در لبه صعودی یا نزولی ورودی خود را دریافت و اعمال نماید و در باقی زمانها یعنی سطوح مثبت و منفی ورودی را اثر ندهد باید تغییراتی در ساختار فلیپ فلاپ ایجاد نماییم و یا از دو فلیپ فلاپ تابع[21] و متبوع[22] استفاده نماییم در شکل زیر سمت چپ با افزودن دو گیت NAND در بالا و پایین فلیپ‌فلاپ D بصورت حساس به لبه صعودی تبدیل شده و فقط در هنگام گذار سیگنال پالس ساعت از سطح صفر به یک یعنی در لبه صعودی ورودی خود را به خروجی اعمال می‌نماید.

[19] **rising edge**
[20] **falling edge**
[21] **slave**
[22] **master**

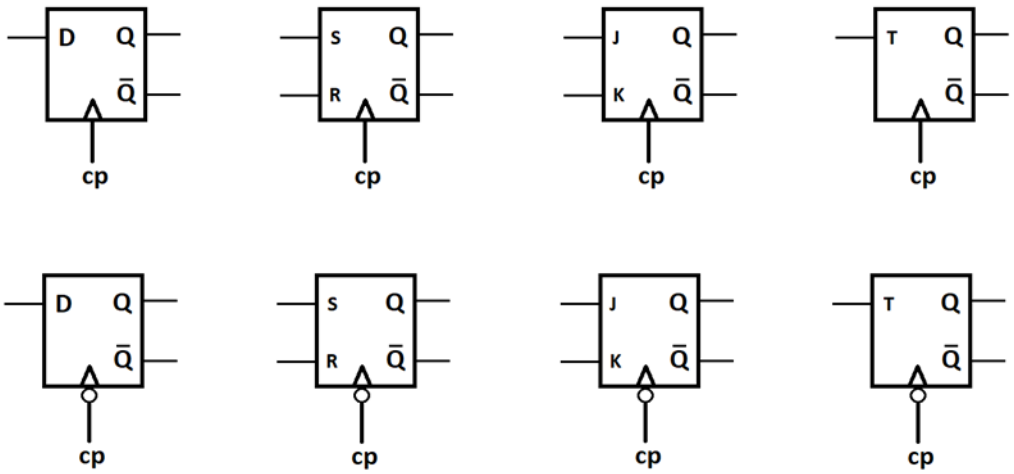




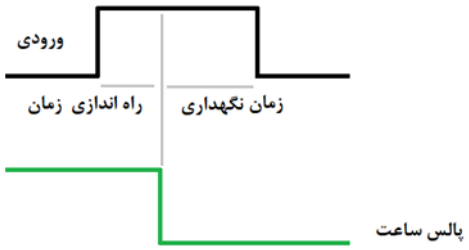
سمت راست شکل فوق فلیپ فلاپ تابع و متبوع را نشان می‌دهد هنگامی که پالس ساعت در سطح صفر است به دلیل وجود گیت NOT مقدار یک به فلیپ فلاپ متبوع اعمال می‌گردد لذا ورودی در خروجی قرار می‌گیرد و نقطه میانی مانند ورودی می‌شود در این هنگام به دلیل آنکه NOT دومی در مسیر سیگنال پالس ساعت قرار دارد مقدار صفر در فلیپ فلاپ تابع اعمال شده و لذا رابطه نقطه میانی و خروجی قطع است و خروجی مقدار فعلی خود را حفظ می‌نماید. وقتی سیگنال پالس ساعت یک می‌شود به دلیل وجود NOT اول مقدار صفر در فلیپ فلاپ متبوع وارد شده و ارتباط ورودی و نقطه میانی قطع می‌شود در همین حال به دلیل وجود NOT دوم مقدار یک به فلیپ فلاپ تابع متصل می‌شود و ارتباط نقطه میانی و خروجی برقرار می‌شود لذا خروجی در این هنگام مقدار نقطه میانی را دریافت می نماید در حالیکه نقطه میانی از ورودی اصلی جدا شده و این به این معنی است که آخرین مقداری که در نقطه میانی ذخیره شده یعنی لحظه قبل از تغییر سطح پالس ساعت از صفر به یک در خروجی ظاهر می‌شود و در این حالت به دلیل جدا بودن ورودی از نقطه میانی، تغییرات ورودی به نقطه میانی منتقل نمی‌شود و تاثیری هم در خروجی نخواهد داشت. در شکل صفحه بعد نحوه اعمال تغییرات ورودی در فلیپ فلاپ‌های تابع و متبوع و خروجی بر اساس تغییرات ورودی نشان داده شده است. همانگونه که مشاهده می‌شود فقط در لبه‌های صعودی فلیپ فلاپ تابع مقدار ورودی را در خروجی خود قرار می‌دهد.



فلیپ فلاپ می‌تواند حساس به لبه صعودی یا حساس به لبه نزولی باشد از این پس فرض می‌کنیم همه فلیپ‌فلاپ‌های مورد بحث حساس به لبه هستند. فلیپ فلاپ را بصورت یک مستطیل (بلوک دیاگرام) نشان می‌دهیم و بدون نشان دادن ساختار داخل ورودی‌ها و خروجی‌های آن را ترسیم می‌کنیم و برای آنکه نشان دهیم حساس به لبه است در نقطه ورود پالس ساعت مثلث قرار می‌دهیم و برای آنکه نشان دهیم حساس به لبه منفی یا سطح منفی است یک دایره کوچک (حباب یا **bubble**) در ورودی پالس ساعت قرار می‌دهیم. در شکل زیر از راست به چپ فلیپ فلپ های **T** ، **JK**، **SR** و **D** نشان داده شده است ردیف بالا حساس به لبه صعودی و ردیف پایین حساس به لبه نزولی است.



نکـــته: وقتی می‌خواهیم فلیپ فلاپ ورودی را دریافت کند باید قبل از آمدن لبه پالس ساعت ورودی را اعمال کنیم حداقل زمان لازم برای آنکه ورودی قبل از اعمال پالس ساعت تثبیت شده باشد را زمان راه‌اندازی[23] می‌گوییم پس از عبور لبه پالس حداقل مدتی کوتاه باید ورودی ثابت بماند به این زمان نگهداری[24] می‌گوییم.



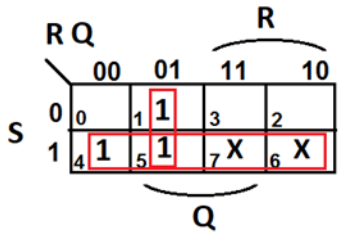
معادلات فلیپ فلاپ‌ها

اکنون با این فرض که فلیپ‌فلاپ‌های مورد بحث حساس به لبه هستند در نظر خواهیم داشت یک پالس ساعت به فلیپ‌فلاپ‌ها اعمال گردد در این صورت در هر لبه بالارونده یا در صورت حساس بودن به لبه نزولی در هر لبه پایین رونده فلیپ‌فلاپ خروجی خود را بر اساس پیکربندی ورودی خود تغییر می‌دهد مثلا اگر فلیپ فلاپ SR در وضعیت $Q = 0$ باشد و پیکربندی set یعنی $S = 1 , R = 0$ روی ورودی اعمال گردد در اولین لبه پالس خروجی آن بصورت $Q = 1$ خواهد شد. برای آنکه بین حالت فعلی و حالت بعدی فلیپ فلاپ تمایز قائل شویم حالت فعلی را با Q و حالت بعدی (بعد از اولین لبه پالس ساعت) را بصورت Q* نمایش می‌دهیم بر همین اساس رابطه‌ای بین حالت فعلی Q و حالت بعدی Q* وجود خواهد داشت به عبارت دیگر Q* تابعی از پیکربندی ورودی و حالت فعلی Q است در مورد هر فلیپ‌فلاپ این رابطه تفاوت خواهد داشت و به آن معادله فلیپ‌فلاپ می‌گویند.

در خصوص فلیپ‌فلاپ D حالت بعدی فقط تابع ورودی D است لذا خواهیم داشت $Q^* = D$. در خصوص فلیپ‌فلاپ SR لازم است ورودی‌های S و R و حالت فعلی را در جدول درستی وارد کرده سپس با استفاده از روش نقشه‌های کارنو ساده نماییم دقت نمایید که طبق قرار داد حالت $S=0 , R=0$ را حالت نگهداری می‌دانیم.

[23] **setup time**
[24] **hold time**

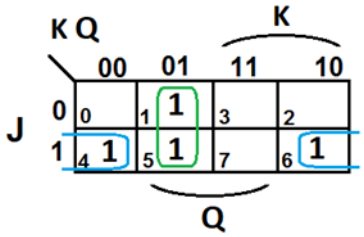
	S	R	Q	Q*
HOLD	0	0	0	0
	0	0	1	1
RESET	0	1	0	0
	0	1	1	0
SET	1	0	0	1
	1	0	1	1
INVALID	1	1	0	X
	1	1	1	X



$$Q^* = S + \bar{R}Q$$

در خصوص فلیپ‌فلاپ JK حالت غیرمجاز نداریم و اگر همزمان J = K = 1 شود حالت بعدی نقیض حالت فعلی خواهد بود و به آن پیکربندی toggle می‌گویند.

	J	K	Q	Q*
HOLD	0	0	0	0
	0	0	1	1
KILL	0	1	0	0
	0	1	1	0
JUMP	1	0	0	1
	1	0	1	1
TOGGLE	1	1	0	1
	1	1	1	0



$$Q^* = J\bar{Q} + \bar{K}Q$$

در خصوص فلیپ‌فلاپ T مقدار Q* تابع مقدار ورودی T و حالت فعلی Q است.

	T	Q	Q*
HOLD	0	0	0
	0	1	1
TOGGLE	1	0	1
	1	1	0

$$Q^* = T \oplus Q$$

$$Q^* = \bar{T}Q + T\bar{Q}$$

جدول تحریک[25] فلیپ فلاپ‌ها

جدول تحریک فلیپ‌فلاپ‌ها نشان می‌دهد اگر فلیپ‌فلاپ در وضعیت Q باشد برای رسیدن به وضعیت Q* چه پیکربندی را باید در ورودی اعمال کنیم . همانطور که مشاهده می‌گردد چهار حالت تغییر به عنوان سطرهای جدول قرار گرفته و ستون‌ها ورودی‌های فلیپ‌فلاپ‌های مختلف را نشان می‌دهد.

[25] **excitation table**

Q	Q*	S	R	J	K	D	T
0	0	0	X	0	X	0	0
0	1	1	0	1	X	1	1
1	0	0	1	X	1	0	1
1	1	X	0	X	0	1	0

در سطر اول تغییر حالت وجود ندارد لذا در فلیپ‌فلاپ SR یا حالت HOLD رخ داده که به صورت پیکـــربندی S = 0, R=0 است یا عمل RESET رخ داده بنابراین دو حالت S=0 R=0 و یا R=1 S=0 صحیح است لذا S قطعا باید 0 باشد و R می‌تواند 0 یا 1 باشد و ترکیب این دو حالت بصورت R=X S=0 خواهد بود. در فلیپ‌فلپ JK هم می‌توان با پیکربندی KILL به صفر رسید هم می‌توان hold نمود لذا هر دو پیکربندی J=0, K=1 و J=0,K=0 صحیح است لذا J قطعا باید 0 باشد و K هم می‌تواند 0 باشد و هم 1 باشد بنابراین J = 0 و K = X. فلیپ‌فلاپ D حالت بعدی تابع ورودی است لذا برای 0 شدن باید D = 0 باشد در فلیپ‌فلاپ T چون تغییر حالت وجود نداشته T=0 خواهد بود.

در سطر دوم تغییر 0 به 1 رخ داده لذا در فلیپ‌فلاپ SR به صورت پیکربندیSET است یعنی S = 1, R=0 در فلیپ‌فلپ JK هم می‌توان با پیکربندی JUMP به 1 رسید هم می‌توان toggle نمود لذا هر دو پیکـــربندی J=1, K=0 و J=1,K=1 صحیح است لذا J قطعا 1 است و K می‌تواند هم 0 باشد و هم 1 باشد بنابراین خواهیم داشت J = 1 و K = X. فلیپ‌فلاپ D حالت بعدی تابع ورودی است لذا برای 1 شدن باید D = 1 باشد در فلیپ‌فلاپ T چون تغییر حالت وجود داشته T=1 خواهد بود.

در سطر سوم تغییر 1 به 0 در فلیپ‌فلاپ SR به صورت پیکربندی S = 0, R=1 است زیرا عمل RESET شدن باید صورت بگیرد در فلیپ‌فلپ JK هم می‌توان با پیکربندی KILL به صفر رسید هم می‌توان toggle نمود لذا هر دو پیکربندی J=0, K=1 و J=1,K=1 صحیح است لذا J می‌تواند هم 0 باشد و هم 1 باشد بنابراین J = X و K = 1. فلیپ‌فلاپ D حالت بعدی تابع ورودی است لذا برای 0 شدن باید D = 0 باشد در فلیپ‌فلاپ T چون تغییر حالت وجود داشته T=1 خواهد بود.

در سطر چهارم تغییر حالت وجود ندارد لذا در فلیپ‌فلاپ SR یا حالت HOLD رخ داده که به صورت پیکـــربندی S = 0, R=0 است یا عمل SET رخ داده بنابراین دو حالت S=0 R=0 و یا R=0 S=1 صحیح است لذا R قطعا باید 0 باشد و S می‌تواند 0 یا 1 باشد و ترکیب این دو حالت بصورت R=0 S=X خواهد بود. در

فلیپ‌فلپ JK هم می‌توان با پیکربندی JUMP به یک رسید هم می‌توان hold نمـــود لذا هر دو پیکـــربندی J=1, K=0 و J=0,K=0 صحیح است لذا K قطعا باید 0 باشد و J هم می‌تواند 0 باشد و هم 1 باشد بنابراین خواهیم داشت J = X و K = 0. فلیپ‌فلاپ D حالت بعدی تابع ورودی است لذا برای 1 شدن باید D = 1 باشد در فلیپ‌فلاپ T چون تغییر حالت وجود نداشته T=0 خواهد بود.

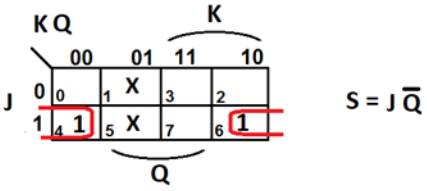
ساختن فلیپ فلاپ‌ها از روی یکدیگر

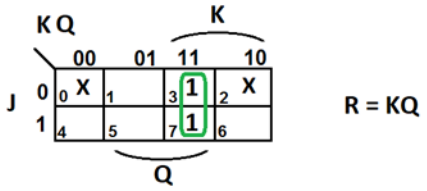
با توجه به اینکه 4 نوع فلیپ‌فلاپ داریم می‌توانیم به 12 طریق مختلف فلیپ‌فلاپ‌ها را از روی هم بسازیم برای این کار از جدول درستی استفاده خواهیم کرد.

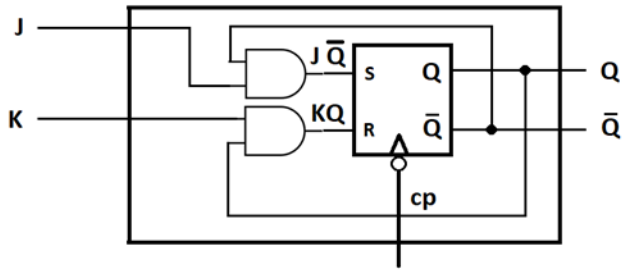
ساختن فلیپ فلاپ JK از روی SR

برای این تبدیل فرض می‌کنیم که یک فلیپ‌فلاپ SR در اختیار داریم و می‌خواهیم فلیپ‌فلاپ JK بسازیم بنابراین فلیپ‌فلاپ SR را درون یک باکس قرار داده و خطوط ورودی JK را به آن باکس وارد می‌کنیم اکنون باید مدار واسطی طراحی کنیم که ارتباط بین J,K ورودی و S,R را برقرار نماید در واقع S و R در اینجا دو تابع بولی هستند که ورودی‌های آنها J,K, Q است. برای بدست آوردن این توابع بولی جدول درستی را برای فلیپ‌فلاپ هدف (JK) می‌نویسیم سپس S و R را ساده کرده و با گیت‌های مناسب پیاده‌سازی می‌کنیم.

J	K	Q	Q*	S	R
0	0	0	0	0	X
0	0	1	1	X	0
0	1	0	0	0	X
0	1	1	0	0	1
1	0	0	1	1	0
1	0	1	1	X	0
1	1	0	1	1	0
1	1	1	0	0	1



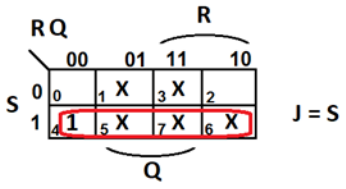


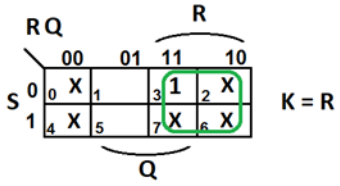


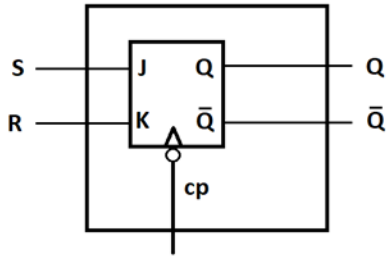
ساختن فلیپ فلاپ SR از روی JK

برای این تبدیل فرض می‌کنیم که یک فلیپ‌فلاپ JK در اختیار داریم و می‌خواهیم فلیپ‌فلاپ SR بسازیم بنابراین فلیپ‌فلاپ JK را درون یک باکس قرار داده و خطوط ورودی S و R را به آن باکس وارد می‌کنیم اکنون باید مدار واسطی طراحی کنیم که ارتباط بین S,R ورودی و J,K را برقرار نماید در واقع J و K در اینجا دو تابع بولی هستند که ورودی‌های آنها R,S, Q است. برای بدست آوردن این توابع بولی جدول درستی را برای فلیپ‌فلاپ هدف (SR) می‌نویسیم سپس J و K را ساده کرده و مشاهـــده می‌شود که J = S و K = R می‌شود لذا اصلا نیاز به مدار واسط ندارد و مستقیم می‌توان ورودی‌ها را به J و K متصل نمود.

S	R	Q	Q*	J	K
0	0	0	0	0	X
0	0	1	1	X	0
0	1	0	0	0	X
0	1	1	0	X	1
1	0	0	1	1	X
1	0	1	1	X	0
1	1	0	X	X	X
1	1	1	X	X	X



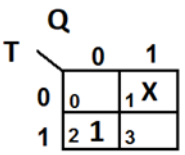




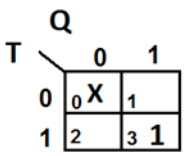
ساختن فلیپ فلاپ T از روی SR

برای این تبدیل فرض می‌کنیم که یک فلیپ‌فلاپ SR در اختیار داریم و می‌خواهیم فلیپ‌فلاپ T بسازیم بنابراین فلیپ‌فلاپ SR را درون یک باکس قرار داده و تنها ورودی T را به آن باکس اعمال می‌کنیم در این حالت S و R دو تابع هستند که باید از روی ورودی‌های T و Q ساخته شوند. طبق روال جدول درستی ترسیم می‌کنیم و مشاهده می‌شود که توابع S و R ساده‌پذیر نیستند لذا با دو گیت AND پیاده‌سازی می‌نماییم.

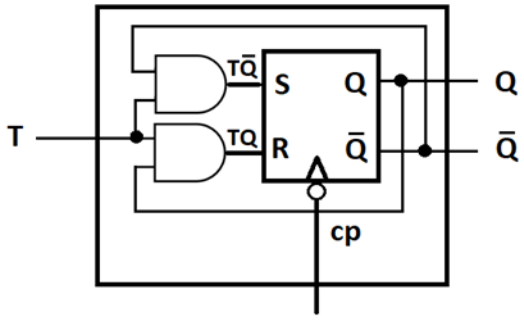
T	Q	Q*	S	R
0	0	0	0	X
0	1	1	X	0
1	0	1	1	0
1	1	0	0	1



$S = T\bar{Q}$



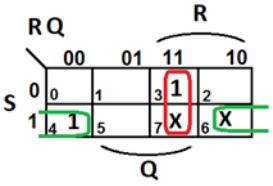
$R = TQ$



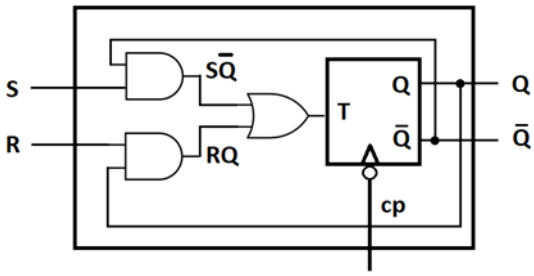
ساختن فلیپ فلاپ SR از روی T

برای این تبدیل فرض می‌کنیم که یک فلیپ‌فلاپ T در اختیار داریم و می‌خواهیم فلیپ‌فلاپ SR بسازیم بنابراین فلیپ‌فلاپ T را درون یک باکس قرار داده و ورودی‌های S و R را به آن باکس اعمال می‌کنیم در این حالت T تابعی از ورودی‌های S,R و Q می‌باشد. طبق روال جدول درستی ترسیم می‌کنیم و تابع T را بر حسب S و R, Q بدست آورده ساده می‌نماییم و با استفاده از گیت‌های مناسب پیاده‌سازی می‌کنیم.

S	R	Q	Q*	T
0	0	0	0	0
0	0	1	1	0
0	1	0	0	0
0	1	1	0	1
1	0	0	1	1
1	0	1	1	0
1	1	0	X	X
1	1	1	X	X



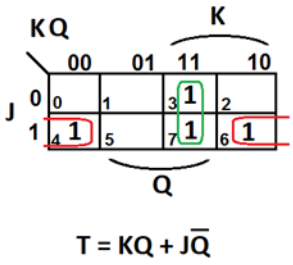
$T = RQ + S\bar{Q}$



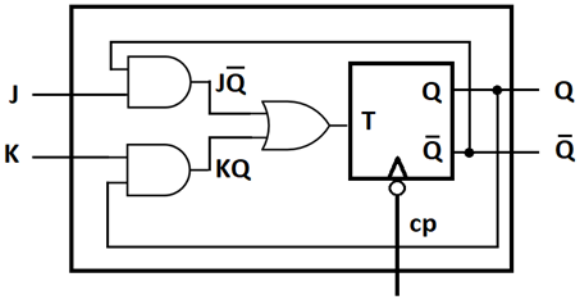
ساختن فلیپ فلاپ JK از روی T

برای این تبدیل فرض می‌کنیم که یک فلیپ‌فلاپ T در اختیار داریم و می‌خواهیم فلیپ‌فلاپ JK بسازیم بنابراین فلیپ‌فلاپ T را درون یک باکس قرار داده و ورودی‌های J و K را به آن باکس اعمال می‌کنیم در این حالت T تابعی از ورودی‌های J,K و Q می‌باشد. طبق روال جدول درستی ترسیم می‌کنیم و تابع T را بر حسب J و K و Q بدست آورده ساده می‌نماییم و با استفاده از گیت‌های مناسب پیاده‌سازی می‌کنیم.

J	K	Q	Q*	T
0	0	0	0	0
0	0	1	1	0
0	1	0	0	0
0	1	1	0	1
1	0	0	1	1
1	0	1	1	0
1	1	0	1	1
1	1	1	0	1



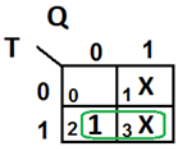
$T = KQ + J\bar{Q}$



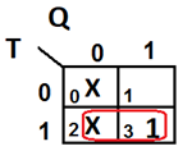
ساختن فلیپ فلاپ T از روی JK

برای این تبدیل فرض می‌کنیم که یک فلیپ‌فلاپ JK در اختیار داریم و می‌خواهیم فلیپ‌فلاپ T بسازیم بنابراین فلیپ‌فلاپ JK را درون یک باکس قرار داده و ورودی T را به آن باکس اعمال می‌کنیم از قبل می‌دانیم با اتصال دو پایه J و K به هم و اتصال آن به ورودی T به هدف خود می‌رسیم اما با همان روش تحلیلی نیز می‌توانیم عمل نماییم.

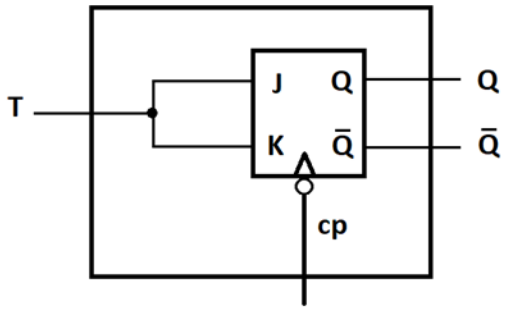
T	Q	Q*	J	K
0	0	0	0	X
0	1	1	X	0
1	0	1	1	X
1	1	0	X	1



$J = T$



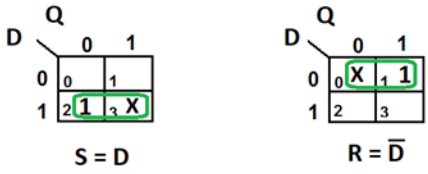
$K = T$

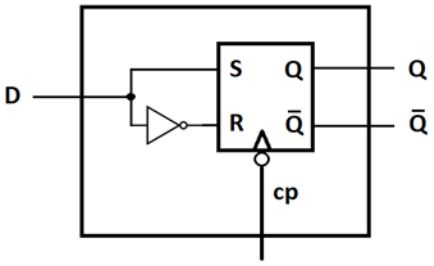


ساختن فلیپ فلاپ D از روی SR

برای این تبدیل فرض می‌کنیم که یک فلیپ‌فلاپ SR در اختیار داریم و می‌خواهیم فلیپ‌فلاپ D بسازیم بنابراین فلیپ‌فلاپ SR را درون یک باکس قرار داده و ورودی D را به آن باکس اعمال می‌کنیم از قبل می‌دانیم با اتصال D به S و اتصال نقیض D به R به هدف خود می‌رسیم اما با همان روش تحلیلی نیز می‌توانیم عمل نماییم.

D	Q	Q*	S	R
0	0	0	0	X
0	1	0	0	1
1	0	1	1	0
1	1	1	X	0

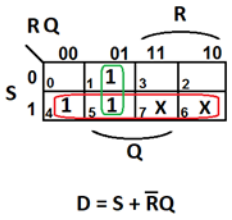


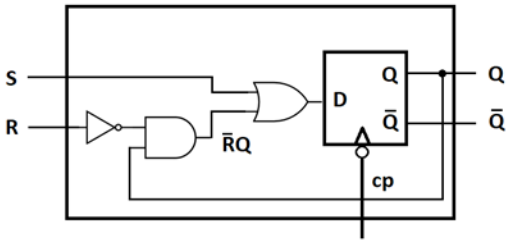


ساختن فلیپ فلاپ SR از روی D

برای این تبدیل فرض می‌کنیم که یک فلیپ‌فلاپ D در اختیار داریم و می‌خواهیم فلیپ‌فلاپ SR بسازیم بنابراین فلیپ‌فلاپ D را درون یک باکس قرار داده و ورودی‌های S و R را به آن باکس اعمال می‌کنیم.

S	R	Q	Q*	D
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	0
1	0	0	1	1
1	0	1	1	1
1	1	0	X	X
1	1	1	X	X

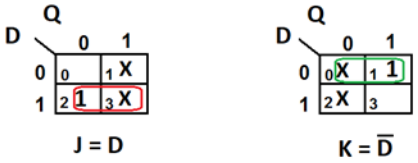


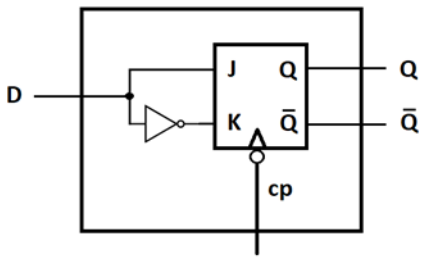


ساختن فلیپ فلاپ D از روی JK

برای این تبدیل فرض می‌کنیم که یک فلیپ‌فلاپ JK در اختیار داریم و می‌خواهیم فلیپ‌فلاپ D بسازیم بنابراین فلیپ‌فلاپ JK را درون یک باکس قرار داده و ورودی D را به آن باکس اعمال می‌کنیم از قبل می‌دانیم با اتصال D به J و اتصال نقیض D به K به هدف خود می‌رسیم زیرا فلیپ‌فلاپ JK بسیار شبیه SR می‌باشد اما با همان روش تحلیلی نیز می‌توانیم عمل نماییم.

D	Q	Q*	J	K
0	0	0	0	X
0	1	0	X	1
1	0	1	1	X
1	1	1	X	0

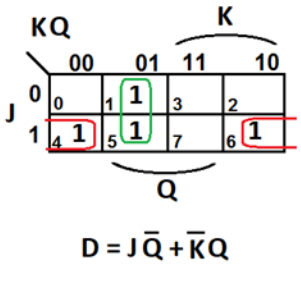


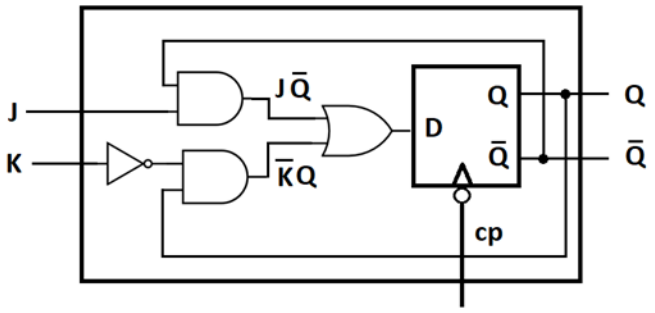


ساختن فلیپ فلاپ JK از روی D

برای این تبدیل فرض می‌کنیم که یک فلیپ‌فلاپ D در اختیار داریم و می‌خواهیم فلیپ‌فلاپ JK بسازیم بنابراین فلیپ‌فلاپ D را درون یک باکس قرار داده و ورودی‌هایJ و K را به آن اعمال می‌کنیم.D تابعی از J,Q وK خواهد بود. جدول درستی را نوشته و ساده می‌کنیم.

J	K	Q	Q*	D
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	0
1	0	0	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	0	0



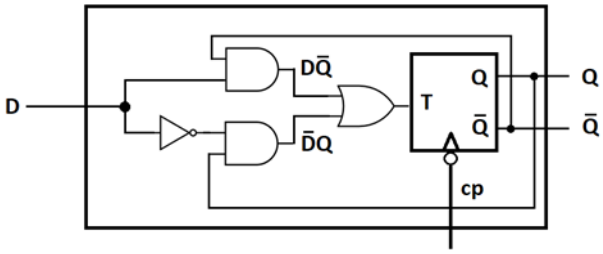


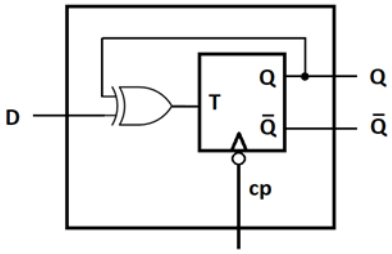
ساختن فلیپ فلاپ D از روی T

برای این تبدیل فرض می‌کنیم که یک فلیپ‌فلاپ T در اختیار داریم و می‌خواهیم فلیپ‌فلاپ D بسازیم بنابراین فلیپ‌فلاپ T را درون یک باکس قرار داده و ورودی D را به آن اعمال می‌کنیم. T تابعی از D , Q خواهد بود. جدول درستی را نوشته و مشاهده می‌کنیم که T همان XOR خطوط D و Q است. که در شکل زیر پیاده‌سازی شده. (در سمت راست از گیت XOR برای سهولت استفاده شده)

D	Q	Q*	T
0	0	0	0
0	1	0	1
1	0	1	1
1	1	1	0

$T = \bar{D}Q + D\bar{Q}$
$T = D \oplus Q$





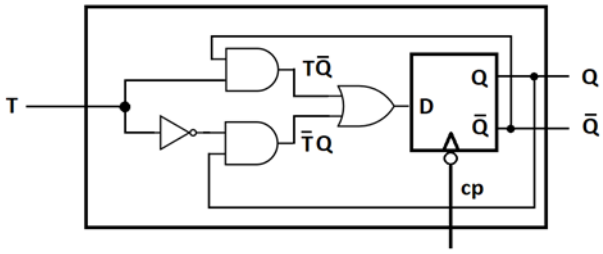
ساختن فلیپ فلاپ T از روی D

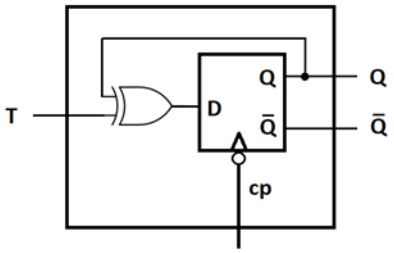
برای این تبدیل فرض می‌کنیم که یک فلیپ‌فلاپ D در اختیار داریم و می‌خواهیم فلیپ‌فلاپ T بسازیم بنابراین فلیپ‌فلاپ D را درون یک باکس قرار داده و ورودی T را به آن اعمال می‌کنیم. D تابعی از T , Q خواهد بود. جدول درستی را نوشته و مشاهده می‌کنیم که D همان XOR خطوط T و Q است که در شکل زیر پیاده‌سازی شده. (در سمت راست از گیت XOR برای سهولت استفاده شده)

T	Q	Q*	D
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	0

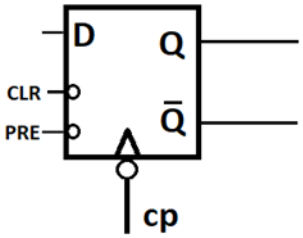
$D = \bar{T}Q + T\bar{Q}$

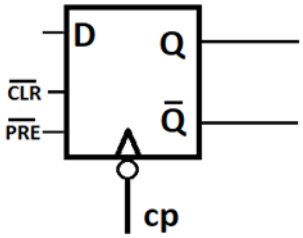
$D = T \oplus Q$





نکـــته: تاکنون فرض ما بر آن بوده که ورودی فلیپ‌فلاپ‌های مورد بحث با آمدن لبه کلاک خوانده شده و بر خروجی و حالت فلیپ‌فلاپ اثر می‌گذارد به همین دلیل به این ورودی‌ها سنکرون[26] یا همزمان میگویند اما فلیپ فلاپ می‌تواند ورودی غیر همزمان یا ورودی آسنکرون[27] نیز داشته باشد. معمولا فلیپ‌فلاپ‌ها دو ورودی آسنکرون **clear** برای **0** شدن و ورودی **preset** برای **1** شدن دارند که در هر زمان که اعمال شود اثر می‌کند. ورودی‌های آسنکرون معمولا حساس به سطح منفی هستند و بصورت $\overline{\boldsymbol{CLR}}$ و $\overline{\boldsymbol{PRE}}$ یا با قرار دادن حباب **(bubble)** در کنار ورودی نشان داده می‌شوند.

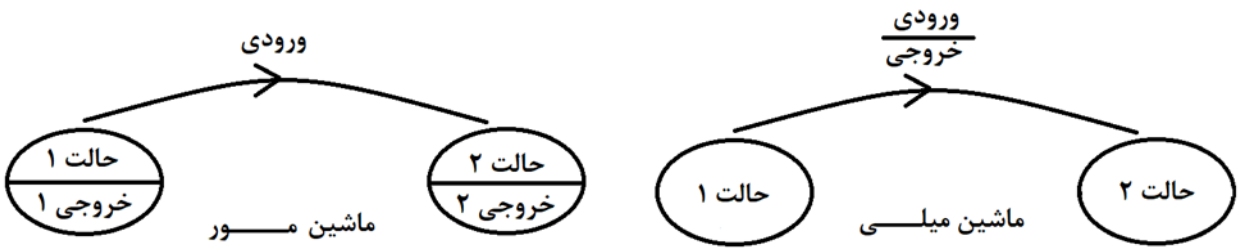




ماشین‌های میلی[28] و مور[29]

ماشین میلی و ماشین مور دو مدل از ماشین‌های حالت متناهی[30] هستند. به زبان ساده ماشین‌های میلی و مور در هر زمان در یک حالت هستند و بر اساس ورودی در پالس ساعت بعدی حالت آنها عوض می‌شود و بر اساس ورودی به حالت جدیدی می‌روند. اگر به ازای هر حالت یک خروجی ثابت داشته باشیم ماشین مورد نظر مور نام دارد ولی اگر خروجی تابع ورودی و حالت موجود باشد ماشین میلی خواهیم داشت.

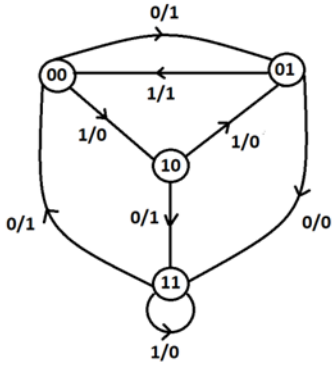
[26] **synchronous**
[27] **asynchronous**
[28] **Mealy machine**
[29] **Moore machine**
[30] **Finite state machines**



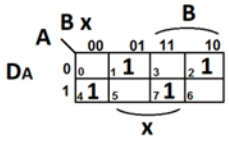
ماشین‌های میلی و مور برای پیاده‌سازی مدارات منطقی حافظه‌دار بکار می‌روند و پیاده‌سازی این ماشین‌ها با استفاده از فلیپ‌فلاپ‌ها انجام می‌گیرد. در ادامه چهار ماشین میلی و مور را با استفاده از فلیپ‌فلاپ‌های مختلف پیاده‌سازی می‌کنیم.

مثال: ماشین میلی زیر را با استفاده از فلیپ‌فلاپ‌های D پیاده‌سازی نمایید.

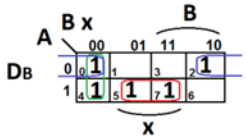
ابتدا حالات را از 00 تا 11 شماره‌گذاری می‌کنیم سپس تغییر از هر حالت با توجه به ورودی به حالت بعد را می‌نویسیم و از روی آن مقادیر DA و DB را محاسبه می‌کنیم



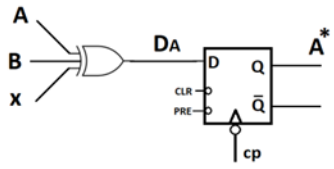
A B x	A* B* y	DA	DB
0 0 0	0 1 1	0	1
0 0 1	1 0 0	1	0
0 1 0	1 1 0	1	1
0 1 1	0 0 1	0	0
1 0 0	1 1 1	1	1
1 0 1	0 1 0	0	1
1 1 0	0 0 1	0	0
1 1 1	1 1 0	1	1

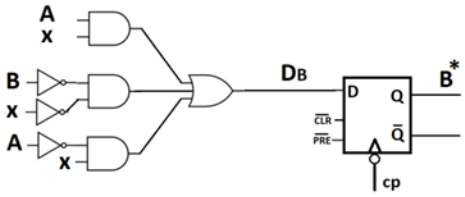


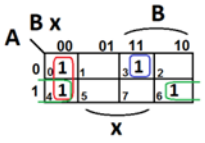
$D_A = A \oplus B \oplus x$



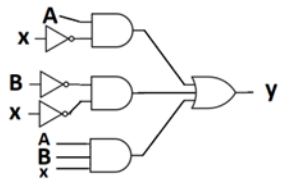
$D_B = Ax + \bar{B}\bar{x} + \bar{A}x$





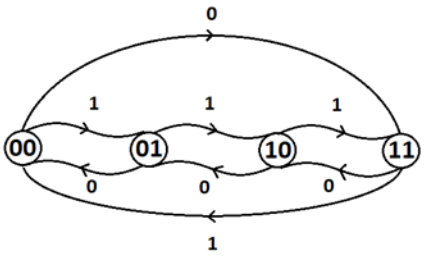


$y = \bar{B}\bar{x} + A\bar{x} + ABx$

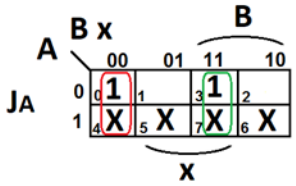


مثال: ماشین مـــور (فاقد خروجی) زیر را با استفاده از فلیپ‌فلاپ‌های JK پیاده‌سازی نمایید.

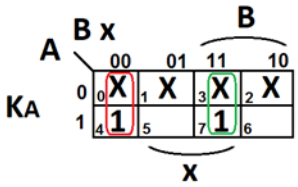
ابتدا حالات را از 00 تا 11 شماره‌گذاری می‌کنیم سپس تغییر از هر حالت با توجه به ورودی به حالت بعد را می‌نویسیم و از روی آن مقادیر JA,KA,JB,KB را با استفاده از جدول تحریک محاسبه می‌کنیم



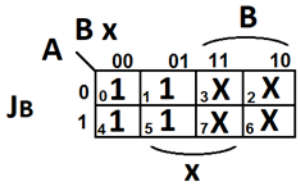
A	B	x	A^*	B^*	J_A	K_A	J_B	K_B
0	0	0	1	1	1	X	1	X
0	0	1	0	1	0	X	1	X
0	1	0	0	0	0	X	X	1
0	1	1	1	0	1	X	X	1
1	0	0	0	1	X	1	1	X
1	0	1	1	1	X	0	1	X
1	1	0	1	0	X	0	X	1
1	1	1	0	0	X	1	X	1



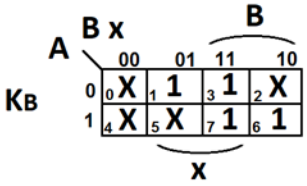
$J_A = \bar{B}\bar{x} + Bx$



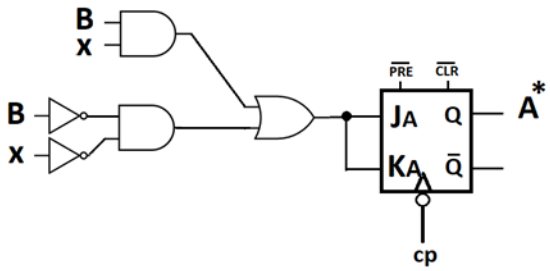
$K_A = \bar{B}\bar{x} + Bx$

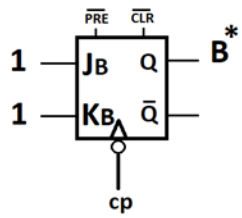


$J_B = 1$



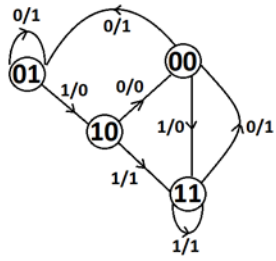
$K_B = 1$



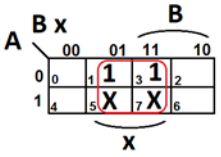


مثال: ماشین میلی زیر را با استفاده از فلیپ‌فلاپ‌های SR پیاده‌سازی نمایید.

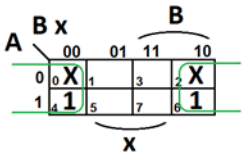
ابتدا حالات را از 00 تا 11 شماره‌گذاری می‌کنیم سپس تغییر از هر حالت با توجه به ورودی به حالت بعد را می‌نویسیم و از روی آن مقادیر SA,SA,RB,RB را با استفاده از جدول تحریک محاسبه می‌کنیم.



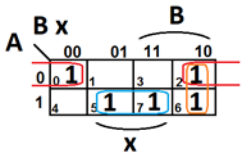
A	B	x	A^*	B^*	Y	S_A	R_A	S_B	R_B
0	0	0	0	1	1	0	X	1	0
0	0	1	1	1	0	1	0	1	0
0	1	0	0	1	1	0	X	X	0
0	1	1	1	0	0	1	0	0	1
1	0	0	0	0	0	0	1	0	X
1	0	1	1	1	1	X	0	1	0
1	1	0	0	0	1	0	1	0	1
1	1	1	1	1	1	X	0	X	0



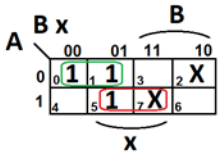
$S_A = x$



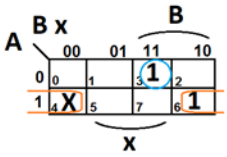
$R_A = \bar{x}$



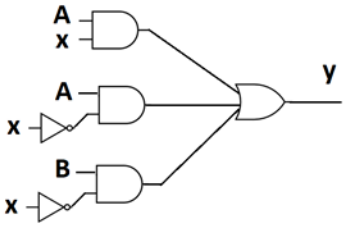
$Y = Ax + B\bar{x} + A\bar{x}$

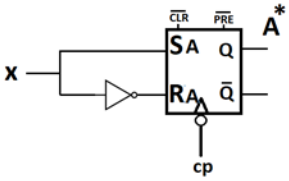


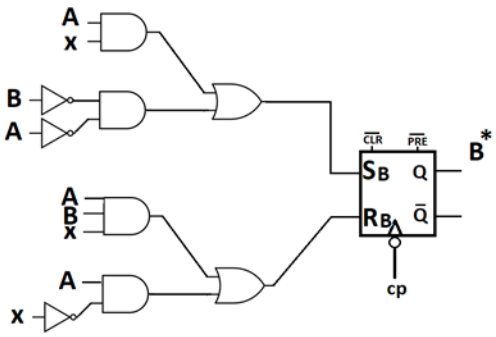
$S_B = \bar{A}\bar{B} + Ax$



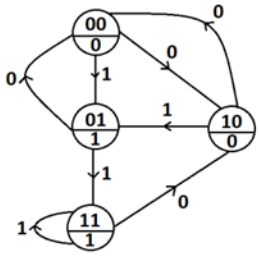
$R_B = A\bar{x} + ABx$



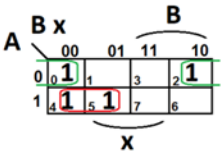




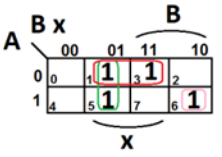
مثال: ماشین مـــور زیر را با استفاده از فلیپ‌فلاپ‌های T پیاده‌سازی نمایید.



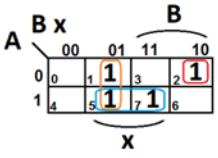
A	B	x	A^*	B^*	Y	T_A	T_B
0	0	0	1	0	0	1	0
0	0	1	0	1	1	0	1
0	1	0	1	1	1	1	0
0	1	1	0	0	0	0	1
1	0	0	0	0	0	1	0
1	0	1	0	1	1	1	1
1	1	0	1	0	0	0	1
1	1	1	1	1	1	0	0



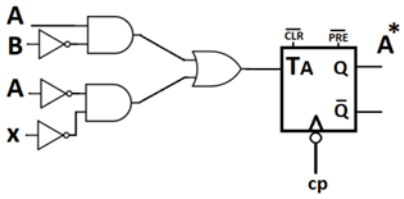
$T_A = A\bar{B} + \bar{A}\bar{x}$

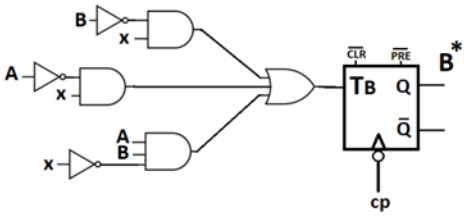


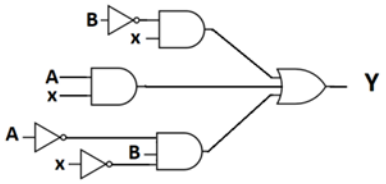
$T_B = \bar{B}x + \bar{A}x + AB\bar{x}$



$Y = \bar{B}x + Ax + \bar{A}B\bar{x}$





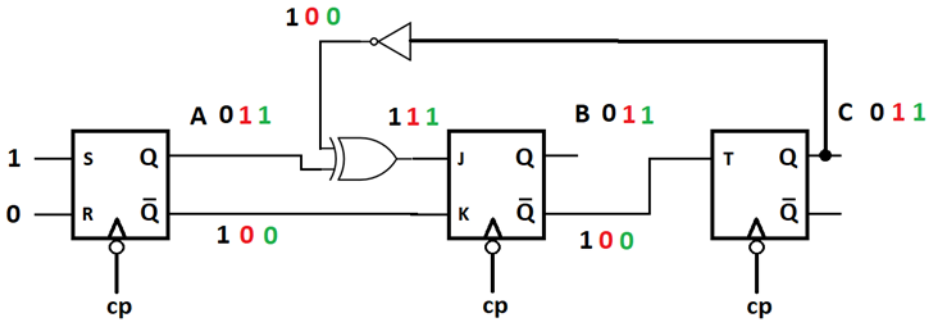


نکـــته: هرگاه **N** وضعیت داشته باشیم نیاز به $\log_2 N$ فلیپ فلاپ خواهیم داشت.

نکـــته: هرماشین میلی را میتوان با افزودن حالات بیشتر به ماشین مـــور تبدیل نمود.

قبل از پرداختن به موضوع ثبات‌ها با بررسی یک مثال وضعیت فلیپ‌فلاپ‌ها را بعد از اعمال چند پالس ساعت بررسی می‌کنیم. در اینگونه مثالها در لحظه ورود پالس ساعت هر فلیپ‌فلاپ فقط بر اساس مقادیری که در ورودی قرار دارد مقدار خـــواهد گرفت و مقادیر جدید برای پالس ساعت بعدی محاسبه خواهند شد.

مثال: فرض کنید مقادیر هر سه فلیپ‌فلاپ زیر در لحظه صفر برابر با 0 باشد پس از اعمال دو پالس ساعت مقادیر هر سه فلیپ‌فلاپ را بدست آورید



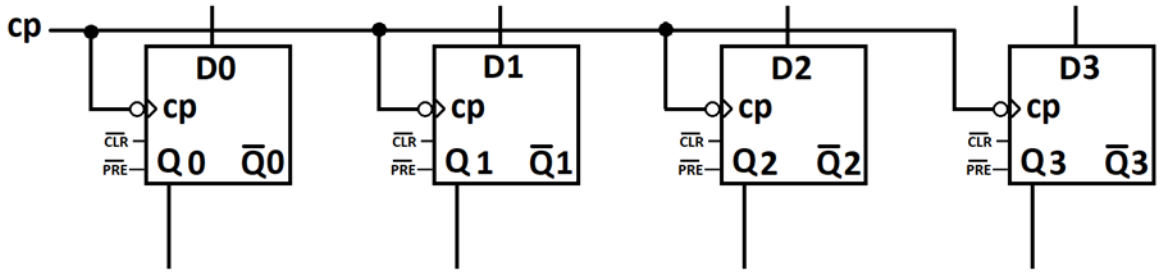
راه حل : مقادیر فعلی فلیپ‌فلاپ‌ها را با رنگ مشکی نشان می‌دهیم پس از اعمال پالس ساعت اول فلیپ فلاپ SR به حالت SET می‌رود لذا A=1 می‌شود که در شکل با رنگ قرمز نشان داده شده فلیپ‌فلاپ JK بر اساس ورودی‌های قبلی خود (رنگ مشکی) به حالات toggle رفته و خروجی خود را معکوس می‌کند B=1 و فلیپ‌فلاپ T بر اساس ورودی قبلی خود toggle نموده و خروجی خود را یک می‌کند C=1. پس از اعمال پالس ساعت دوم (رنگ سبز) فلیپ فلاپ SR باز هم در حالت SET خواهد بود لذا A=1 می‌ماند که در شکل با رنگ سبز نشان داده شده فلیپ‌فلاپ JK بر اساس ورودی‌های قبلی خود (رنگ قرمز) به حالت jump رفته و خروجی خود را یک نگه می‌دارد B=1 و فلیپ‌فلاپ T بر اساس ورودی قبلی خود که صفر بوده (رنگ قرمز) hold نموده و خروجی یک خود را حفظ می‌کند C=1.

ثبات[31] ها

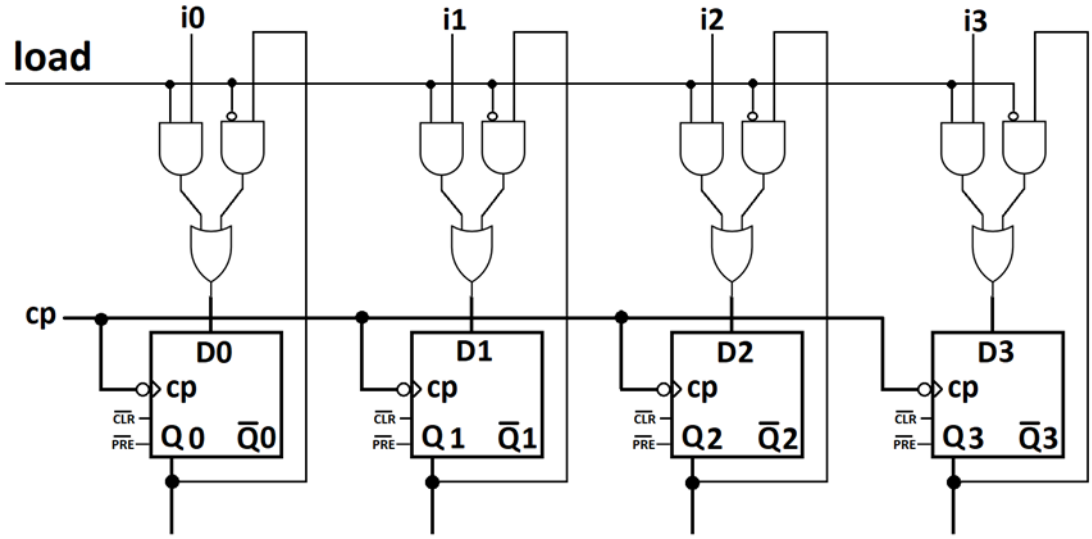
ثبات ها از کنارهم قرار دادن فلیپ‌فلاپ‌ها بدست می‌آیند. هر فلیپ‌فلاپ می‌تواند یک بیت داده را ذخیره نماید لذا یک ثبات n بیتی از n فلیپ‌فلاپ ساخته شده است. معمولا از فلیپ‌فلاپ‌های D برای ساخت ثبات استفاده می‌گردد. ثبات‌ها علاوه بر ذخیره‌سازی داده‌های n بیتی امکاناتی نظیر شیفت به راست و چپ و چرخش را برای ثبات در نظر گرفت.

[31] **register**

اولین ثباتی که بررسی می‌گردد ثبات چهاربیتی است که در شکل زیر نشان داده شده است. این ثبات از اتصال چهار فلیپ‌فلاپ D به هم ساخته شده که پالس ساعت مشترکی دارند و در لبه پایین رونده مقادیر D0 الی D3 در فلیپ‌فلاپ‌ها قرار می‌گیرند.

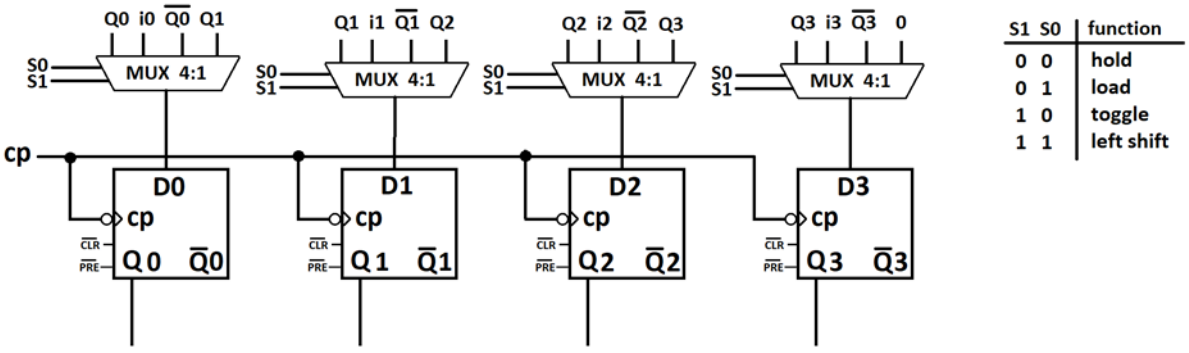


حال اگر بخواهیم یک خط کنترلی load داشته باشیم به نحوی که اگر load = 0 باشد در لبه پایین رونده پالس ساعت، مقدار فعلی ثبات حفظ شود و اگر load = 1 باشد مقادیر جدید i0 الی i3 به فلیپ‌فلاپ‌ها بارگزاری گردد از سازوکار زیر استفاده می‌کنیم.



همانگونه که مشاهده گردید اگر در ورودی فلیپ‌فلاپ ساختاری شبیه مالتی‌پلکسر قرار داده شود امکانات بیشتری می‌توان به ثبات افزود در شکل صفحه بعد ثبات چهاربیتی علاوه بر امکان حفظ وضعیت و بارگزاری

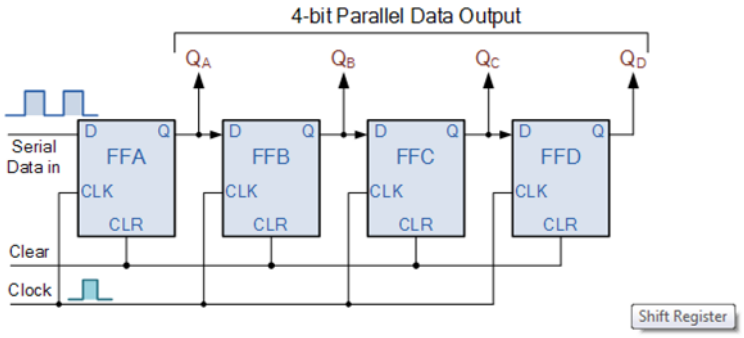
داده جدید امکان شیفت به چپ و امکان معکوس شدن داده‌های فعلی را دارد برای این منظور از یک مالتی‌پلکسر چهار به یک و دو خط کنترلی استفاده می‌نماییم.



S1	S0	function
0	0	hold
0	1	load
1	0	toggle
1	1	left shift

در ثبات فوق هرگاه S0=0 و S1=0 باشد ورودی شماره صفر (ورودی اول) مالتی‌پلکسر روی خـــروجی‌ها قرار می‌گیرد با توجه به اینکه مقدار فعلی فلیپ‌فلاپ‌ها در ورودی اول قرار گـرفته به آن معنی است که مقدار فعلی در فلیپ‌فلاپ‌ها قرار می‌گیرد و این عمل hold یا نگهداری است. هرگاه S0=1 و S1=0 باشد ورودی شماره یک (ورودی دوم) مالتی‌پلکسر روی خـــروجی‌ها قرار می‌گیرد با توجه به اینکه ورودی‌های جدید به ورودی دوم مالتی‌پلکسر متصل است به این معنی است که ورودی‌های جدید در فلیپ‌فلاپ‌ها بارگزاری می‌گردد و لذا عمل load یا بارگزاری رخ داده. هرگاه S0=0 و S1=1 باشد ورودی شماره دو (ورودی سوم) مالتی‌پلکسر روی خـــروجی‌ها قرار می‌گیرد با توجه به اینکه ورودی سوم همان مقدار معکوس فلیپ‌فلاپ است لذا مقدار معکوس در ورودی قرار می‌گیرد و مانند آن است که مقدار فعلی ثبات toggle یا معکوس می‌گردد (مکمل یک مقدار فعلی در ثبات قرار می‌گیرد). هرگاه S0=1 و S1=1 باشد ورودی سه (ورودی چـهارم) مالتی‌پلکسر روی خـــروجی‌ها قرار می‌گیرد با توجه به اینکه D0 مساوی Q1 و به همین ترتیب ورودی D_n مساوی Q_{n+1} است به این معنی است که کلیه بیت‌ها یک واحد به سمت چپ شیفت (حرکت) می‌کنند و لذا عمل left shift یا شیفت چپ انجام می‌گیرد. عمل شیفت چپ مانند ضرب در عدد 2 می‌باشد.

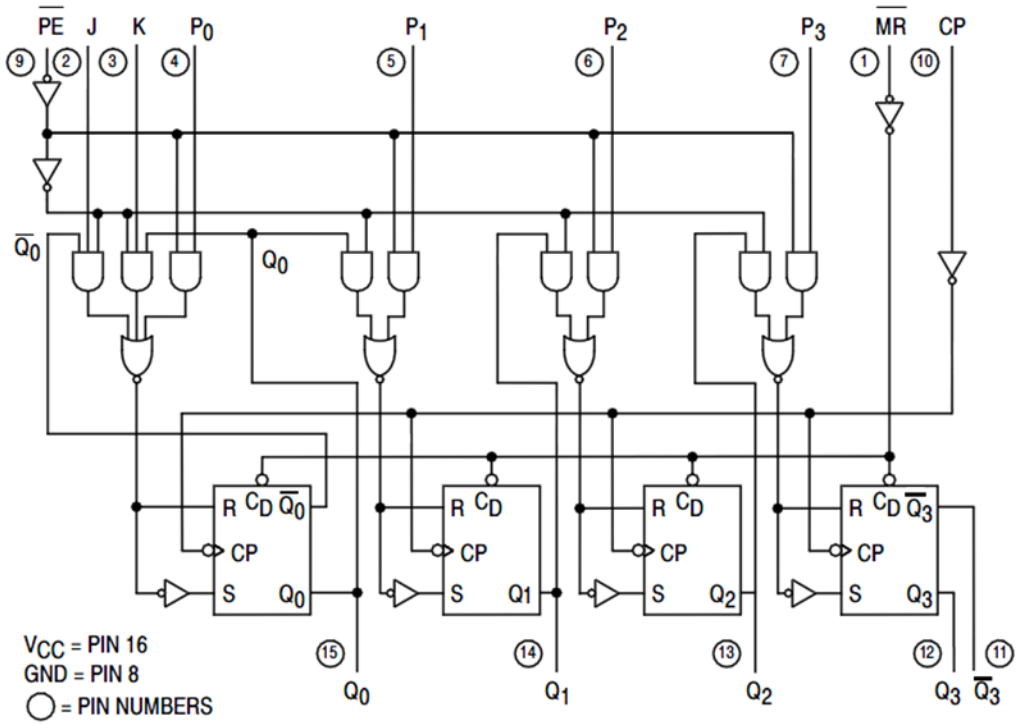
در شکل زیر یک ثبات چهاربیتی نشان داده شده که داده‌ها بصورت سریال (متوالی) از سمت چپ وارد شده و بر اثر هر پالس ساعت هر فلیپ‌فلاپ مقدار فلیپ‌فلاپ سمت چپ خود را به عنوان ورودی دریافت می‌نماید و لذا عمل ورود متوالی (serial input) انجام می‌پذیرد و پس از چهار پالس ساعت می‌توان مقدار ثبات را از خروجی بصورت موازی (parallel output) دریافت نمـــود.



یکی از ثبات‌های مورد استفاده در آزمایشگاه ثبات چهار بیتی SN74LS195A می‌باشد که برای ساخت مدارات شمارنده و کاربردهای مشابه مورد استفاده قرار می‌گیرد شکل زیر دیاگرام منطقی این ثبات را نشان می‌دهد.

SN74LS195A

LOGIC DIAGRAM



اولین نکته‌ای که در شکل به نظر می‌رسد آنست که فلیپ‌فلاپ‌ها حساس به لبه پایین‌رونده هستند ولی سیگنال پالس ساعت نیز منفی شده لذا عملکرد فلیپ‌فلاپ‌ها در لبه بالارونده خواهد بود. نکته دوم وجود گیت NOR در سازوکـــار ورودی فلیپ‌فلاپ‌ها است با توجه به خاصیت گیت NOR خروجی آن به پایه R متصل شده و معکوس آن به پایه S متصل شده است. یعنی هرگاه قرار باشد مقدار یک در فلیپ‌فلاپ بارگزاری شود گیت

NOR آن را به صفر تبدیل کرده و این صفر به R متصل شده و معکوس آن یعنی یک به S متصل می‌شود و فلیپ‌فلاپ در حالت SET قرار می‌گیرد. نکته دیگر وجود سیگنال غیرهمزمان (آسنکرون) MR است که مخفف Master Reset است و هرگاه فعال شود (یعنی صفر شود) تمامی فلیپ‌فلاپ‌ها بدون در نظر گرفتن سیگنال CP (پالس ساعت) ریست می‌شوند. و لذا در حالت کارکرد عادی باید مقدار این سیگنال یک باشد.

کنترل عملکرد اصلی این ثبات با سیگنال PE/ می‌باشد. هرگاه این سیگنال صفر (فعال) باشد عمل بارگزاری یا Load بصورت موازی انجام می‌گیرد به این صورت که مقادیر P0 الی P3 در فلیپ‌فلاپ‌های 0 الی 3 قرار می‌گیرند. سیگنال PE/ در فلیپ‌فلاپ اول از چپ صفر پس از دو بار عبور از گیت NOT بصورت همان صفر به دو گیت AND سمت چپ و وسط وارد می‌شود لذا مقادیر خروجی این دو گیت صفر بوده و تاثیری در گیت NOR نخواهند داشت. سیگنال PE/ پس از یکبار عبور از گیت NOT یعنی با مقدار یک به گیت AND سمت راست روی همان فلیپ‌فلاپ سمت چپ وارد شده لذا مقدار P0 با مقدار یک AND شده یعنی همان مقدار P0 وارد گیت NOR می‌شود سایر ورودی‌های گیتNOR همه مقادیر صفر هستند لذا آنچه از گیت NOR خارج خواهد شد معکوس سیگنال P0 است که مستقیما به ورودی RESET فلیپ‌فلاپ اول از سمت چپ وارد شده و معکوس آن به گیت SET اعمال می‌شود لذا مقدار P0 در فلیپ‌فلاپ اول قرار می‌گیرد. هرگاه PE/ مساوی یک (یعنی غیر فعال) باشد عمل ورود سریال و خروج سریال انجام خواهد شد به این شکل که با توجه به وضعیت J و K مقدار فلیپ‌فلاپ اول از سمت چپ معین می‌شود اگر j=k=0 باشد خروجی دو AND مرتبط با j و k صفر خواهد بود و AND سمت راست نیز به دلیل PE=1/ دارای ورودی صفر است و لذا تمامی AND ها خروجی صفر را به گیت NOR وارد می‌کنند گیت NOR در اثر سه ورودی صفر خروجی یک تولید خواهد کرد و یک به RESET و معکوس آن به SET اعمال شده و فلیپ‌فلاپ مقدار صفر را خواهد گرفت لذا j=0 k=0 منجر خواهد شد به q0 = 0 . هرگاه j=k=1 آنگاه ورودی‌های گیت NOR شامل صفر از سمت راست و q0 و q0/ خواهد بود در این حالت اگر q0=0 باشد q0=1/ خواهد بود و اگر q0=1 در هر صورت یکی از ورودی‌های گیت NOR برابر با یک خواهد بود لذا خروجی گیت NOR مساوی صفر است و صفر به ورودی RESET و معکوس آن یک به ورودی SET اعمال خواهد شد و فلیپ‌فلاپ مقدار یک را اخذ خواهد نمود. هرگاه j=0 k=1 باشد AND های چپ و راست خروجی صفر تولید می‌کنند و فقط AND وسط دارای دو ورودی یک است لذا خروجی آن q0 خواهد بود که به گیت NOR اعمال خواهد شد و خروجی گیت NOR برابری با q0/ خواهد بود که به ورودی RESET اعمال شده و نقیض آن به SET اعمال می‌شود لذا مقدار فلیپ‌فلاپ برابر با q0 خواهد بود. هرگاه j=1 k=0 باشد AND های راست و وسط خروجی صفر تولید می‌کنند و فقط AND سمت چپ دارای دو ورودی یک

است لذا خروجی آن q0/ خواهد بود که به گیت NOR اعمال خواهد شد و خروجی گیت NOR برابری با q0 خواهد بود که به ورودی RESET اعمال شده و نقیض آن به SET اعمال می‌شود لذا مقدار فلیپ‌فلاپ برابر با q0/ خواهد بود. در خصوص سایر فلیپ‌فلاپ ها نیز هرگاه PE/ مساوی یک باشد گیت AND سمت راست خروجی صفر تولید خواهد کرد و گیت AND سمت چپ به دلیل آنکه یکی از ورودی‌های آن یک است (به دلیل آنکه PE/ مساوی یک است) ورودی دیگر آن در خروجی ظاهر می‌شود لذا مقدار q0 در فلیپ‌فلاپ 1 مقدار q1 در فلیپ‌فلاپ 2 و به همین ترتیب مقدار q2 در فلیپ‌فلاپ 3 قرار می‌گیرد یعنی شیفت به سمت راست انجام می‌گیرد. جدول خلاصه بحث فوق را نشان می‌دهد. (این جدول از روی برگه داده datasheet این قطعه الکترونیکی کپی شده است)

MODE SELECT — TRUTH TABLE

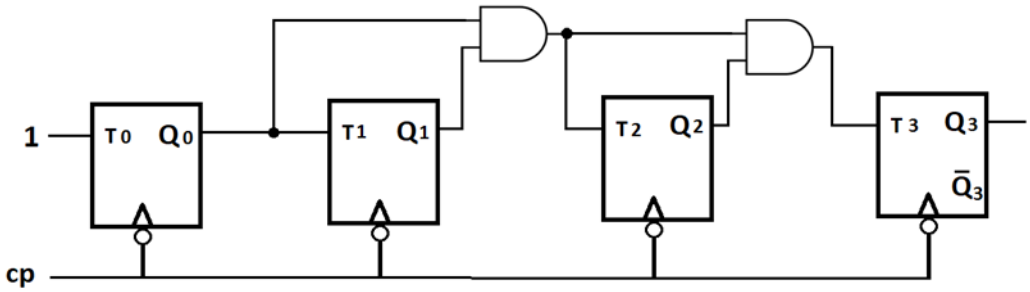
OPERATING MODES	INPUTS					OUTPUTS				
	MR	PE	J	K	P_n	Q_0	Q_1	Q_2	Q_3	$\overline{Q_3}$
Asynchronous Reset	L	X	X	X	X	L	L	L	L	H
Shift, Set First Stage	H	h	h	h	X	H	q_0	q_1	q_2	$\overline{q_2}$
Shift, Reset First	H	h	l	l	X	L	q_0	q_1	q_2	$\overline{q_2}$
Shift, Toggle First Stage	H	h	h	l	X	$\overline{q_0}$	q_0	q_1	q_2	$\overline{q_2}$
Shift, Retain First Stage	H	h	l	h	X	q_0	q_0	q_1	q_2	$\overline{q_2}$
Parallel Load	H	l	X	X	p_n	p_0	p_1	p_2	p_3	$\overline{p_3}$

L = LOW voltage levels
H = HIGH voltage levels
X = Don't Care
l = LOW voltage level one set-up time prior to the LOW to HIGH clock transition.
h = HIGH voltage level one set-up time prior to the LOW to HIGH clock transition.
p_n (q_n) = Lower case letters indicate the state of the referenced input (or output) one set-up time prior to the LOW to HIGH clock transition.

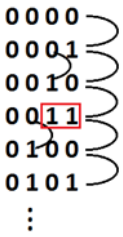
شمارنده[32] ها

شمارنده‌ها برای تولید دنباله‌ای از اعداد (معمولا متوالی) بصورت صعودی و نزولی بکار می‌روند. شمارنده‌ها بر دو نوع هستند نوع اول شمارنده‌های سنکرون که در آنها کلاک بصورت مشترک به همه فلیپ‌فلاپ‌ها اعمال می‌شود و شمارنده‌های آسنکرون (موج گونه) که کلاک فلیپ‌فلاپ بعدی از روی خروجی قبلی است. در ادامه یک شمارنده سنکرون چهاربیتی با استفاده از فلیپ‌فلاپ T طراحی می‌کنیم. برای سادگی فرض می‌کنیم همه فلیپ‌فلاپ‌ها در لحظه اول دارای مقدار صفر هستند.

[32] **counter**

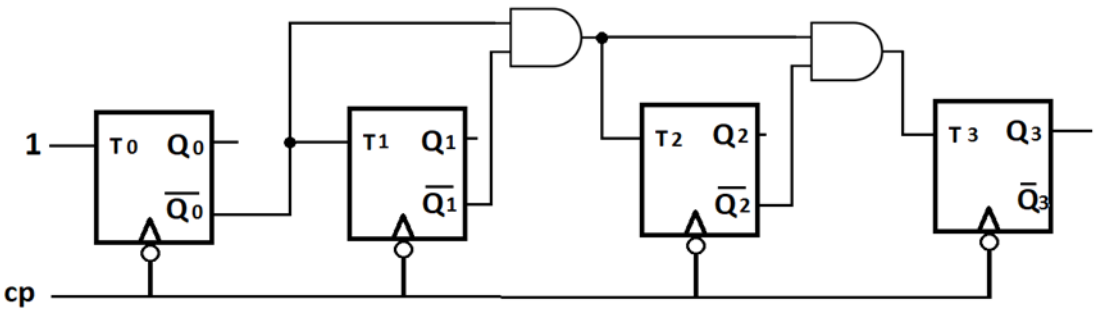


در شکل فوق با آمدن هر لبه پایین رونده پالس ساعت فلیپ‌فلاپ صفر toggle می‌شود یعنی از صفر به یک و از یک به صفر تغییر می‌کند در پالس دوم و سوم و ... به همین ترتیب زیرا ورودی آن همیشه یک است



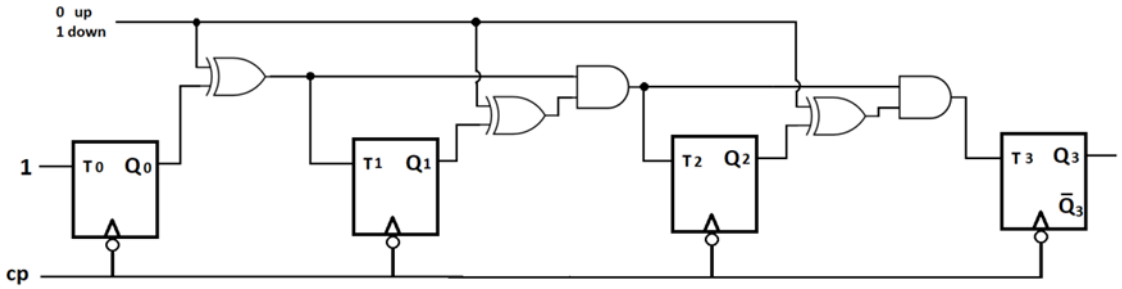
فلیپ‌فلاپ اول فقط زمانهایی toggle می‌کند که مقدار فلیپ‌فلاپ صفر برابر با یک باشد یعنی در پالس‌های زوج (دوم، چهارم ...) فلیپ‌فلاپ دوم فقط در زمانهایی toggle می‌کند که مقادیر فلیپ‌فلاپ‌های صفر و یک هر دو یک باشند (در شکل با رنگ قرمز مشخص شده) زیرا ورودی آن از AND بین q0 و q1 تشکیل شده. در خصوص فلیپ‌فلاپ سوم هم هرگاه q0 q1 q2 همه یک باشند آنگاه عمل toggle صورت می‌پذیرد. همانطور که دیده می‌شود این ساختار اعداد 0 الی 15 را به ترتیب ایجاد می‌کند.

هرگاه ورودی فلیپ‌فلاپ‌های بعدی را از q0/ و q1 and /q0/ و ... بگیریم شمارنده ما نزولی خواهد شمرد که در شکل زیر نشان داده شده است.

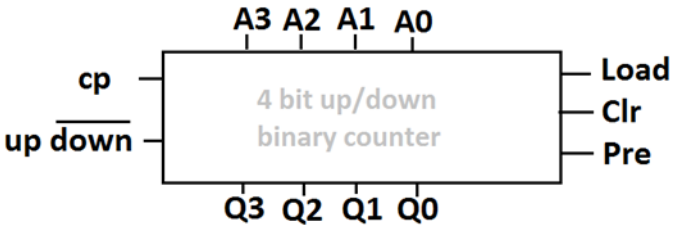


شمارنده نزولی

هرگاه بخواهیم یک شمارنده داشته باشیم که به دلخواه ما صعودی یا نزولی بشمرد باید بتوانیم روی انتخاب مقادیر ورودی فلیپ‌فلاپ‌ها کنترل داشته باشیم برای این کار از گیت XOR استفاده می‌کنیم. هرگاه یکی از ورودی‌های گیت XOR صفر باشد ورودی آن به خروجی منتقل می‌شود و هرگاه یکی از دو ورودی گیت XOR یک باشد ورودی دوم معکوس شده در خروجی قرار می‌گیرد.

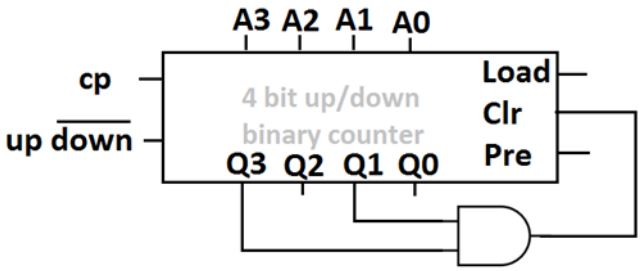


در حالت کلی یک شمارنده چهار بیتی می‌تواند یک مقدار اولیه را بارگزاری نموده (load) یا بدون در نظر داشتن پالس ساعت مقدار آن 1111 شود (pre set) یا بدون در نظر داشتن پالس ساعت مقدار آن 0000 شود (clear) و یا داشتن پایه کنترلی صعودی یا نزولی بشمرد. دیاگرام چنین شمارنده‌ای در شکل زیر نشان داده شده است.

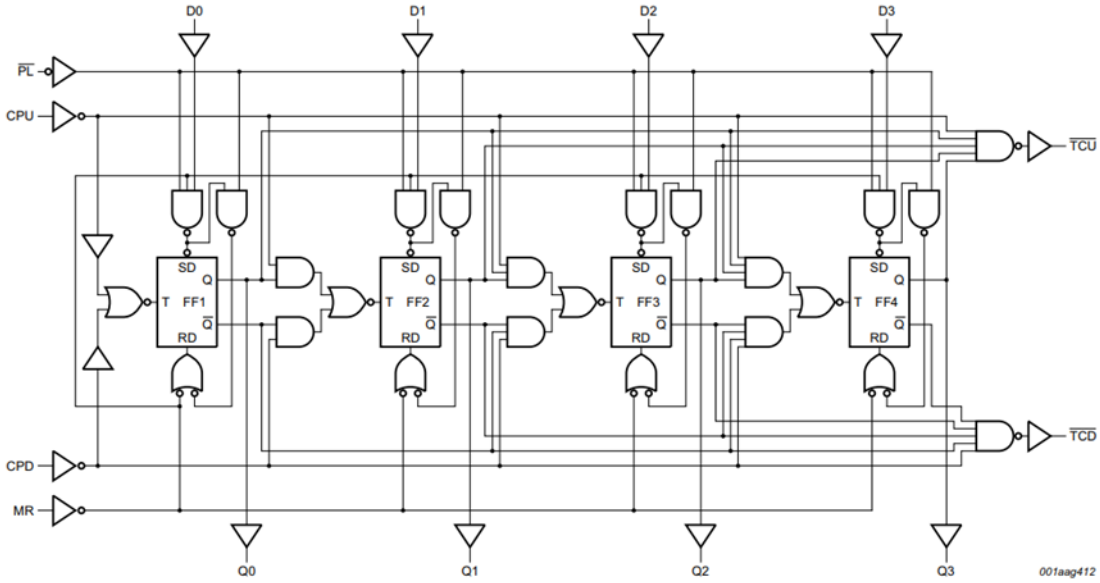


بدون در نظر داشتن پالس ساعت هرگاه Pre = 1 و Clr = 0 شود مقدار 1111 در شمارنده قرار می‌گیرد به همین نحو هرگاه Pre = 0 و Clr = 1 شود مقدار 0000 در شمارنده قرار می‌گیرد. وقتی بخواهیم مقدار شمارنده از ورودی‌های A0 الی A3 بارگزاری شود باید Pre = 0 ، Clr = 0 و Load = 1 شود. برای شمارش صعودی لازم است Pre = 0 ، Clr = 0 ، Load = 0 و up/down = 1 شود و برای شمارش نزولی لازم است مقادیر Pre = 0 ، Clr = 0، Load = 0 و up/down = 0 شود.

اکـــنون می‌خواهیم یک شمارنده BCD بسازیم که اعداد 0 تا 9 را شمارش نماید برای این کار هرگاه الگوی 1010 که معادل باینری عدد 10 است در خروجی ظاهر شد مقدار Clr را فعال می‌کنیم تا عدد بعد از 9 صفر باشد.



یک مثال واقعی از شمارنده‌ها آی‌سی 74HC193 است که ساختاری نسبتا پیچیده ولی عملکردی ساده دارد



جدول عملکرد آن بشرح زیر است.

Operating mode	Inputs								Outputs					
	MR	PL	CPU	CPD	D0	D1	D2	D3	Q0	Q1	Q2	Q3	TCU	TCD
Reset (clear)	H	X	X	L	X	X	X	X	L	L	L	L	H	L
	H	X	X	H	X	X	X	X	L	L	L	L	H	H
Parallel load	L	L	X	L	L	L	L	L	L	L	L	L	H	L
	L	L	X	H	L	L	L	L	L	L	L	L	H	H
	L	L	L	X	H	H	H	H	H	H	H	H	L	H
	L	L	H	X	H	H	H	H	H	H	H	H	H	H
Count up	L	H	↑	H	X	X	X	X	count up				H[2]	H
Count down	L	H	H	↑	X	X	X	X	count down				H	H[3]

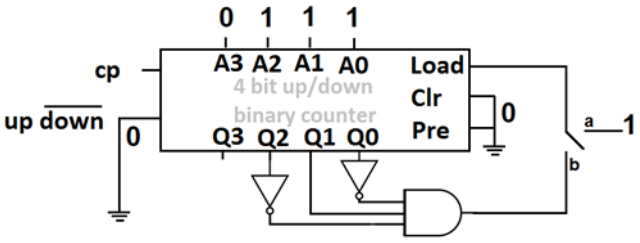
[1] H = HIGH voltage level
L = LOW voltage level
X = don't care
↑ = LOW-to-HIGH clock transition.

[2] $\overline{TCU}$ = CPU at terminal count up (HHHH)

[3] $\overline{TCD}$ = CPD at terminal count down (LLLL).

وقتی می‌خواهیم شمارنده 74HC193 بصورت صعودی شمارش نماید باید پایه‌های MR=0 و PL=1/ باشد سپس CPD=1 و سیگنال پالس ساعت به CPU اعمال گردد. برای آنکه شمارنده بصورت نزولی شمارش نماید باید پایه‌های MR=0 و PL=1/ باشد سپس CPU=1 و سیگنال پالس ساعت به CPD اعمال گردد. برای آنکه عمل بارگزاری انجام گیرد باید پایه PL=0/ و MR=0 باشد.

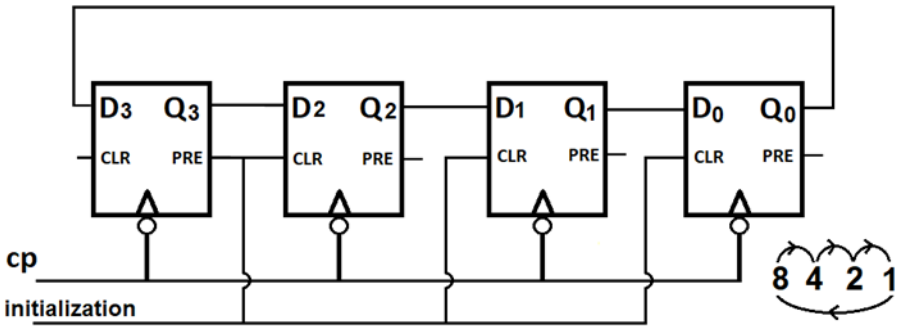
مثال: با استفاده از شمارنده استاندارد شمارنده‌ای طراحی نمایید که اعداد 7 الی 3 را بصورت نزولی شمارش نماید.



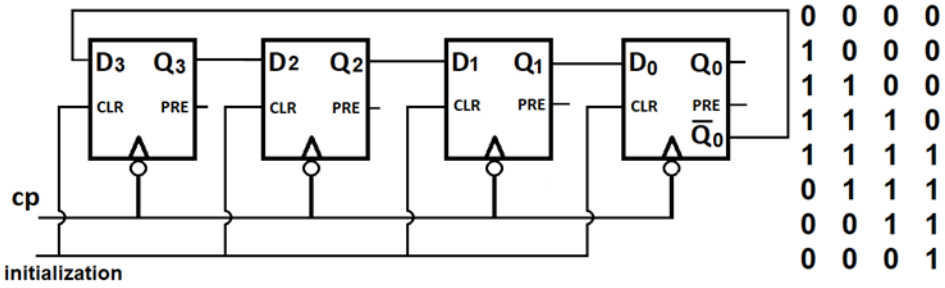
همانطور که مشاهده می‌شود پایه up /down و پایه‌های Clr و Pre باید به 0 متصل گردد همچنین به محض آنکه عدد 2 در خروجی ظاهر شد باید عدد 7 بارگزاری گردد در این صورت عمل شمارش از 7 به 6 به ... به 3 انجام می‌گردد و پس از 3 هنگامی که عدد 2 در خروجی ظاهر می‌شود سیگنال Load فعال شده و لذا عدد 2 هرگز دیده نمی‌شود و به جای آن عدد 7 بارگزاری می‌شود. برای راه‌اندازی شمارنده ابتدا کلید باید برای مدت بسیار کوتاهی (کمتر از پالس ساعت) در وضعیت a باشد و سپس به وضعیت b سوئیچ نماید.

شمارنده حـــلقوی که در شکل زیر نشان داده شده به این صورت کار می‌کند که در ابتدا با سیگنال مقداردهی اولیه[33] مقدار 1 در فلیپ‌فلاپ سمت چپ قرار می‌گیرد و سایر فلیپ‌فلاپ‌ها 0 می‌شوند یعنی عدد 1000 معادل در شمارنده قرار می‌گیرد. پس از آمدن اولین سیگنال پالس ساعت فلیپ‌فلاپ Q3 مقدار 1 خود را به فلیپ‌فلاپ Q2 می‌دهد لذا 0100 تشکیل می‌شود که معادل 4 است و به همین ترتیب پس از آمدن سیگنال پالس ساعت بعدی فلیپ‌فلاپ Q2 مقدار 1 خود را به فلیپ‌فلاپ Q1 می‌دهد لذا 0010 تشکیل می‌شود که معادل 2 است پس از آمدن سیگنال پالس ساعت بعدی فلیپ‌فلاپ Q1 مقدار 1 خود را به فلیپ‌فلاپ Q0 می‌دهد لذا 0001 تشکیل می‌شود که معادل 1 است و پس از آن دوباره 8 ظاهر می‌شود زیرا از مسیر feedback عدد 1 از فلیپ‌فلاپ Q0 به فلیپ‌فلاپ Q3 منتقل می‌شود.

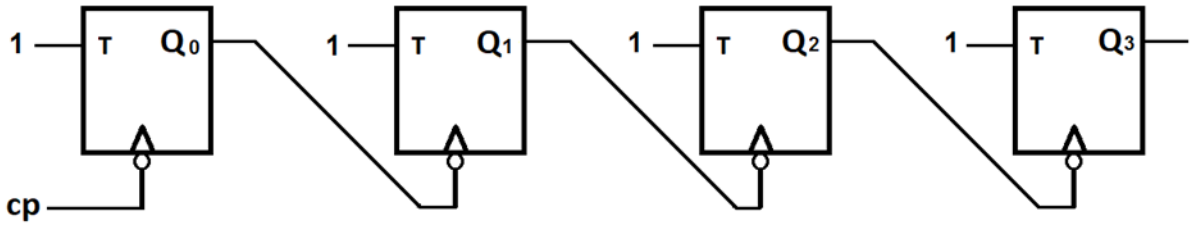
[33] **Initialization**



شمارنده جانســون[34] که در شکل زیر نشان داده شده به این صورت کار می‌کند که در ابتدا با سیگنال مقداردهی اولیه مقدار 0000 در شمارنده قرار می‌گیرد پس از اولین سیگنال پالس ساعت نقیض مقدار فلیپ‌فلاپ Q0 یعنی مقدار 1 از مسیر feedback به Q3 منتقل شده و مقدار 1000 ایجاد می‌شود پس از پالس بعدی مقدار Q3 به Q2 منتقل شده و باز هم نقیض مقدار Q0 به Q3 منتقل شده و لذا مقدار 1100 ایجاد می‌گردد و به همین ترتیب 1110 و 1111 ایجاد می‌شود پس از آن مقدار 0111 تشکیل می‌شود زیرا نقیض مقدار Q0 به Q3 منتقل شده که 0 است و به همین ترتیب 0011 و 0001 و 0000 ایجاد می‌شود. هر شمارنده جانسون با n فلیپ‌فلاپ 2n حالت تولید می‌کند



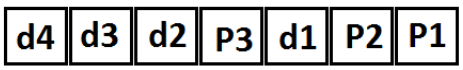
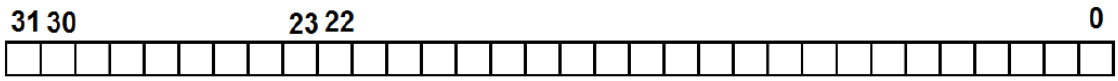
شمارنده آسنکرون به این صورت است که بر خلاف معمول سیگنال پالس ساعت برای تمامی فلیپ‌فلاپ‌ها یکسان نیست بلکه به غیر از فلیپ‌فلاپ اول که به آن سیگنال پالس ساعت اعمال می‌شود سایر فلیپ‌فلاپ‌ها سیگنال پالس ساعت خود را از خروجی فلیپ‌فلاپ‌های دیگر می‌گیرند.

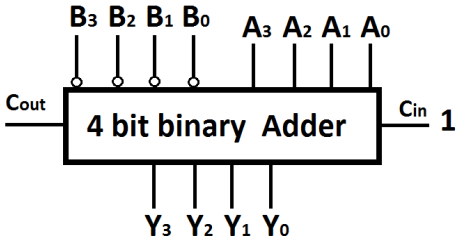
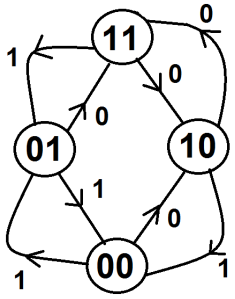
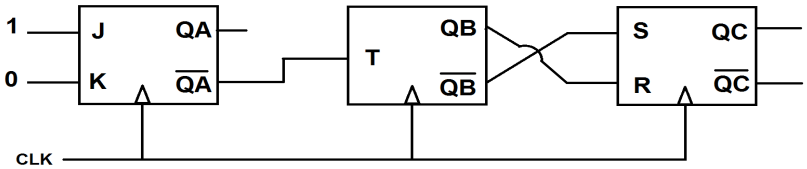


[34] **Johnson counter**

ضمیمه شماره یک

نمونه سوالات مدار منطقی پایان ترم نیمسال اول 96-97

بارم	سوال 1 *(مبناهای عددی و کدینگ)*
1 نمره	الف) عدد $(11101)_2$ را به مبنای 10 تبدیل نمایید. ب) مکمل 4 $(4321)_5$ را بدست آورید. ج) مکمل 1 عدد $(101101101)_2$ را به دست آورید. د) عدد $(1001010110)_g$ را از کد گری به باینری تبدیل کنید.
بارم	سوال 2 *(کدهای تشخیص و تصحیح خطا)*
1 نمره	الف) حداقل فاصله همینگ را برای آنکه کدی بتواند همزمان دو خطا را تشخیص داده و یک خطا را تصحیح کند بدست آورید. ب) رشته 1101 را به روش همینگ ارسال نمایید. (راهنمایی d4=1,d3=1,d2=0,d1=1)  parity groups : p1,d1,d2,d4 p2,d1,d3,d4 p3,d2,d3,d4
بارم	سوال 3 *(نمایش اعداد اعشاری در استاندارد IEEE754)*
2 نمره	عدد 23.25 را با استاندارد IEEE-754 تبدیل نمایید. ***(راهنمایی: آفست توان 127 می‌باشد)*** 
بارم	سوال 4 *(ساده‌سازی به روش نقشه‌های کـــــارنو)*
1 نمره	الف) تابع $f(a,b,c,d) = \sum m(0,1,2,3,7,8,12,13) + d(5,10)$ را با روش کارنو بصورت فرم استاندارد مینیمم SOP ساده نمایید ب) تابع $f(a,b,c) = \prod M(1,2,5,6)$ را با روش کارنو بصورت فرم استاندارد مینیمم POS ساده نمایید
بارم	سوال 5 *(ساده‌سازی به روش کوئـین مک‌کلاکسکی)*
1 نمره	تابع $f(a,b,c,d) = \sum m(1,3,5,7,13,15)$ را با روش کوئین مک‌کلاسکی تا یک سطح ساده نمایید.
بارم	سوال 6 *(دیکدر و مالتی پلکسر)*
2 نمره	الف) تابع $f(a,b,c) = \sum m(2,3,5,6)$ را با یک دیکدر 3 به 8 و یک گیت NAND و تعداد دلخواه گیت NOT پیاده‌سازی نمایید ب) با استفاده از دو مالتی‌پلکسر 4 به 1 (مجهز به فعالساز En) یک مالتی‌پلکسر 8 به1 بسازید. ***(راهنمایی : از یک گیت OR و یک گیت NOT می‌توانید استفاده کنید)***

بارم	سوال 7 (جمع‌کننده)
1 نمره	عملکرد مدار زیر را شرح دهید. (رابطه خروجی Y با ورودی‌های A و B را بنویسید) 
بارم	سوال 8 (ماشین میلی و مور)
2 نمره	دیاگـرام حالت یک ماشین میلی مطابق شکل است مدار فاقد خـــروجی است. این مدار را با دو فلیپ‌فلاپ دلخواه پیاده‌سازی نمایید. ***(راهنمایی: در صورت لزوم استفاده از گیت XOR سه ورودی مجاز است)*** 
بارم	سوال 9 (تبدیل فلیپ‌فلاپها)
2 نمره	الف) از روی فلیپ‌فلاپ JK یک فلیپ‌فلاپ T بسازید (شکل را رسم کنید) ب) از روی فلیپ‌فلاپ T یک فلیپ‌فلاپ JK بسازید (جدول درستی را بنویسید سپس معادلات را بدست آورده و شکل را رسم کنید)
بارم	سوال 10 (وضعیت فلیپ‌فلاپها)
2 نمره	پس از گذشت 2 (دو) کلاک پالس ساعت مقادیر QA, QB و QC را بدست آورید وضعیت اولیه هر سه فلیپ‌فلاپ 0 است. 

ضمیمه شماره دو

نمونه سوالات مدار منطقی پایان ترم نیمسال دوم 96-95

سوال 1 : تبدیل مبنا و مکمل (2 نمره)

الف) عدد (10100110.11101) را از مبنای 2 به مبنای 16 تبدیل کنید.

ب) عدد (3122) را از مبنای 4 به مبنای 2 تبدیل کنید.

ج) مکمل 4 عدد $(434123.23)_5$ را بدست آورید (عدد 434123.23 در مبنای 5 است).

د) مکمل دو عدد 100101111 را بدست آورید.

سوال 2 : کدینگ (2 نمره)

الف) عدد 7 را در کدینگ **BDC** و **Excess-3** نمایش دهید.

ب) عدد 6 را در سیستم وزن دار **$\overline{1}$ $\overline{2}$ 4 8 نشان دهید. (بالای اعداد** 2 و 1 **خط وجود دارد.)**

ج) عدد $(111011010100)_g$ **را از کد گری** (gray) **به کد باینری تبدیل کنید.**

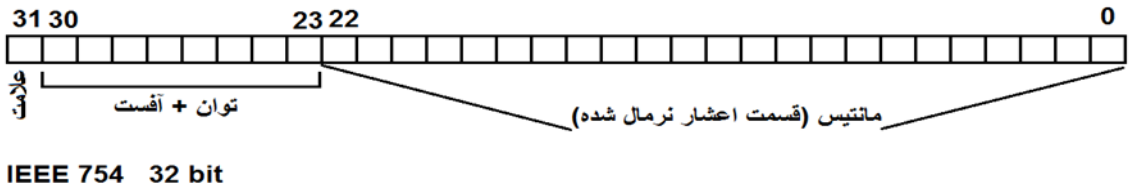
د) عدد $(101111000101)_b$ **را از کد باینری به کد گری** (gray) **تبدیل کنید.**

سوال 3 : جمع و تفریق مکمل دو (1 نمره)

الف) عملیات $(+3) - (-5)$ **را در یک سخت‌افزار** 4 **بیتی به روش مکمل دو انجام دهید و در مورد نتیجه توضیح دهید.**

ب) عملیات $(-2) + (+3)$ **را در یک سخت‌افزار** 3 **بیتی به روش مکمل دو انجام دهید و در مورد نتیجه توضیح دهید.**

سوال 4 : عدد اعشاری 23.625- **را به روش استاندارد** IEEE754 **32 بیتی نمایش دهید. (2 نمره)**



IEEE 754 32 bit

سوال 5 : مینترم و ماکسترم (1 نمره)

الف) مینترم m_{11} **را با متغیرهای a b c d نمایش دهید.**

ب) ماکسترم M_8 **را با متغیرهای x y z t نمایش دهید.**

سوال 6: ساده‌سازی توابع بولی با روش جداول کارنو (2 نمره)

الف) تابع بولی $f(a,b,c,d) = \sum m(1,3,5,7,8,12) + d(10,11,14)$ **را به فرم بهینه استاندارد** SOP **ساده کنید.**

ب) تابع بولی $f(x,y,z) = \prod M(1,4,5)$ **را به فرم بهینه استاندارد** POS **ساده کنید.**

سوال 7: رفع مخاطره در مدارات SOP (1 نمره)

تابع بولی $f(a,b,c) = \bar{b}c + ab$ را با افزودن یک جمله رفع مخاطره (hazzard free) نمایید.

(راهنمایی: تابع را در جدول کارنو وارد کنید)

سوال 8: پیاده‌سازی با گیت NAND (1 نمره)

تابع بولی $f(a,b,c) = \bar{a}b + ac + a\bar{b}$ را فقط با گیت NAND پیاده سازی کنید و شکل آن را ترسیم نمایید.

سوال 9: پیاده‌سازی با دیکدر و یک گیت خاص (2 نمره)

تابع بولی $f(a,b,c) = \sum m(1,5,6)$ را در نظر بگیرید

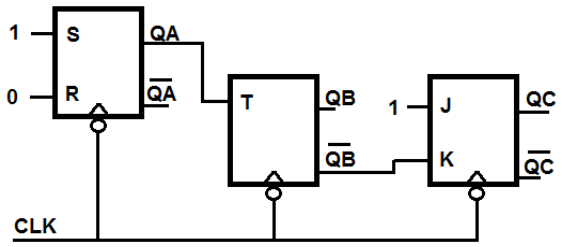
الف) تابع فوق را فقط با استفاده از یک دیکدر و یک گیت NAND پیاده سازی کنید و شکل آن را ترسیم نمایید.

ب) تابع فوق را فقط با استفاده از یک دیکدر و یک گیت NOR پیاده سازی کنید و شکل آن را ترسیم نمایید.

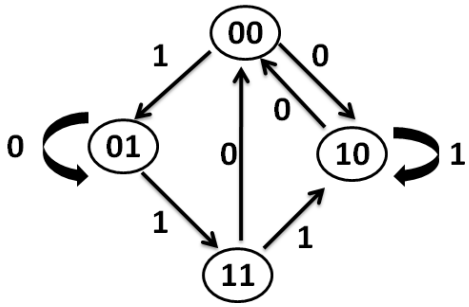
در هر حالت می‌توانید به تعداد دلخـــواه از گیت NOT یا bubble استفاده نمایید.

سوال 10: پس از گذشت دو پالس ساعت (clock pulse) مقادیر فلیپ فلاپ‌های QA و QB و QC را بدست آورید (1 نمره)

مقادیر QA و QB و QC در لحظه شروع 0 می‌باشند.



سوال 11: دیاگرام حالت یک ماشین مطابق شکل زیر است آن را با استفاده از دو فلیپ فلاپ T طراحی نمایید (2 نمره)



ضمیمه شماره سه

نمونه سوالات مدار منطقی پایان ترم نیمسال اول 98-97

سوال 1 *(در این سوال کافی است به چهار مورد پاسخ دهید)*

1-1) عدد $(123)_4$ را از مبنای 4 به مبنای 5 تبدیل نمایید $(\quad)_5$

2-1) عدد $(11010.10)_2$ را از مبنای 2 به مبنای 4 تبدیل نمایید $(\quad)_4$

3-1) عدد $(7821)_9$ را از مبنای 9 به مبنای 3 تبدیل نمایید $(\quad)_3$

4-1) مکمل 4 عدد $(3214.211)_5$ را بدست آورید.

5-1) مکمل 2 عدد $(10110101100100)_2$ را بدست آورید.

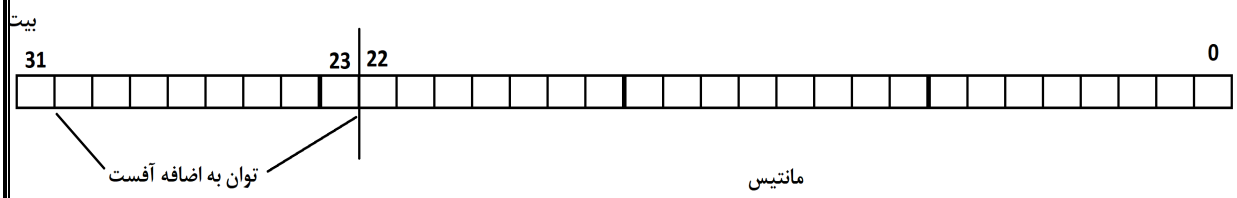
6-1) عدد $(1010110)_g$ از گری (gray) به باینری تبدیل نمایید. $(\quad)_b$

7-1) عدد 8 را بصورت کدینگ excess 3 بنویسید

آیا می‌توانیم عملیات 3-2- به روش مکمل 2 در یک سخت افزار 4 بیتی انجام دهیم ؟ (آن را انجام داده و پاسخ را تحلیل نمایید)

سوال 2

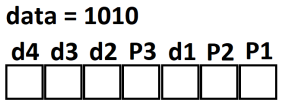
1-2) عدد 58.75- را با استفاده از استاندارد IEEE 754 نمایش دهید. (راهنمایی آفست توان 127 می‌باشد)



سوال 3

1-3) کدی دارای فاصله همینگ 5 است اگر بخواهیم این کد یک خطا را تصحیح نماید همزمان می‌تواند چند خطای دیگر را تشخیص دهد ؟

2-3) داده 1010 را مطابق روش همینگ کدگزاری نمایید.



سوال 4

1-4) تابع $f(a,b,c,d) = \sum m(0,4,7,14,15) + d(3,6)$ را به روش کارنو (SOP) ساده نمایید.

2-4) تابع $f(a,b,c) = \prod M(0,1,6) \times D(3,7)$ را به روش کارنو (POS) ساده نمایید.

سوال 5

1-5) تابع $f(a,b,c,d) = \sum m(2,3,6,7)$ را به روش کـوئین مک‌کلاســکی ساده نمایید.

سوال 6

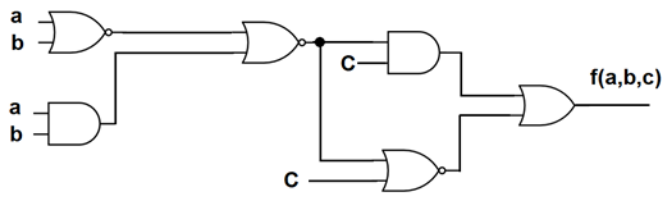
1-6) تابع $f(a,b,c) = \sum m(2,3,7)$ را فقط با گیت NAND پیاده‌سازی نمایید

2-6) تابع a xor b را فقط با گیت NOR پیاده‌سازی نمایید

سوال ۷

۷-۱) تابع $f(a,b,c) = \sum m(1,2,7)$ را با استفاده از یک دیکدر 3:8 و یک گیت NOR پیاده‌سازی نمایید (NOT به هر تعداد مجاز هستید)

سوال ۸ *(از بین سوالات ۸ و ۹ کافی است به دلخواه به یک سوال پاسخ دهید)*



تابع شکل مقابل را در نظر بگیرید

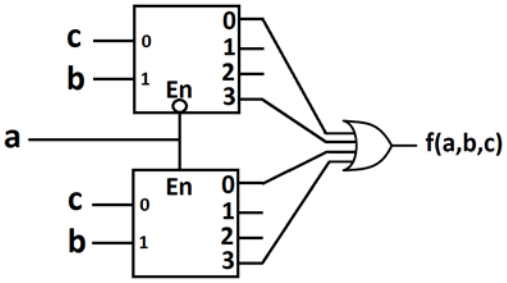
۸-۱) آن را به شکل یک عبارت جبری (جبر بول) فرم SOP نمایش دهید

۸-۲) آن را با استفاده از یک مالتی‌پلکسر 4:1 پیاده‌سازی نمایید

۸-۳) تابع را با استفاده از یک گیت XOR و یک مالتی‌پلکسر 2:1 بسازید.

۸-۴) آیا می‌توان تابع را فقط با یگ گیت (سه ورودی) پیاده‌سازی کرد؟

سوال ۹ *(از بین سوالات ۸ و ۹ کافی است به دلخواه به یک سوال پاسخ دهید)*



تابع شکل مقابل را در نظر بگیرید

۹-۱) آن را به شکل استاندارد (canonical) SOP نمایش دهید

۹-۲) آن را با استفاده از یک مالتی‌پلکسر 4:1 پیاده‌سازی نمایید. آیا میتوان با 2:1 پیاده‌سازی کرد

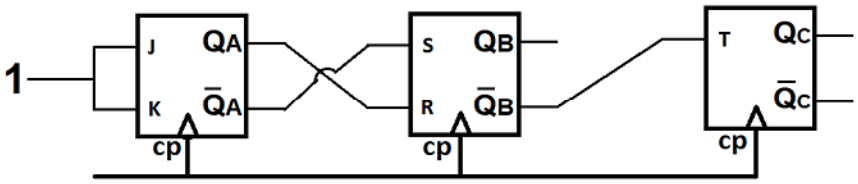
۹-۳) آن را ساده نمایید و فقط با گیت NAND پیاده‌سازی نمایید

۹-۴) آیا تابع اصلا به ورودی a وابسته است ؟ آیا می‌توان آنرا با یک گیت دو ورودی ساخت ؟

سوال ۱۰

شکل مقابل را در نظر بگیرید در ابتدا مقادیر QA=QB=QC=0

۱۰-۱) پس از گذشت دو کلاک (پالس ساعت) مقادیر QA، QB و QC را بدست آورید.



ضمیمه شماره چهار

نمونه سوالات میانترم مدار منطقی نیمسال دوم 97-98

سوال 1:

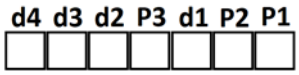
الف) عدد $(11101)_2$ را به مبنای 10 تبدیل نمایید.

ب) مکمل 4 $(4321)_5$ را بدست آورید.

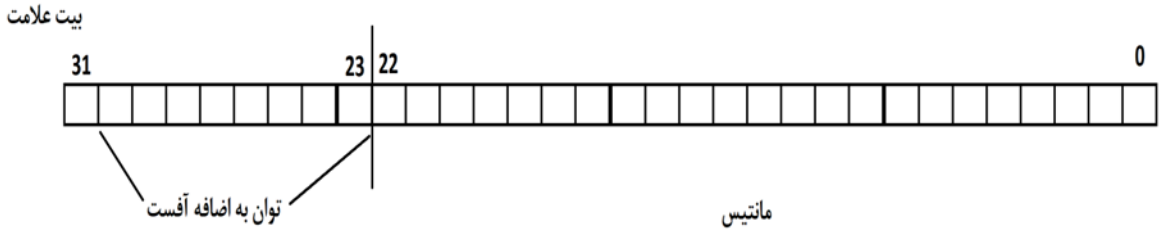
ج) عدد $(5742)_9$ را به مبنای 3 تبدیل نمایید.

د) عدد $(1001010110)_g$ را از کد گری به باینری تبدیل کنید.

سوال 2 : داده 1101 را به روش همینگ کد نمایید



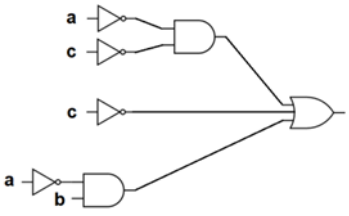
سوال 3 : عدد 123.625 را به روش IEEE754 ذخیره نمایید



سوال 4 :

الف) تابع $f(a,b,c,d) = \sum m(0,2,4,5,13) + d(7,8,9,10,11,15)$ را به روش SOP ساده نمایید.

ب) تابع مدار شکل زیر را به روش POS ساده نمایید.



سوال 5 : تابع $f(a,b,c,d) = \sum m(4,6,7,12,14,15)$ را به روش کوئین مک‌کلاسکی ساده نمایید.

ضمیمه شماره پنج

نمونه سوالات پایانترم مدار منطقی نیمسال دوم 97-98

سوال 1: (1 نمره)

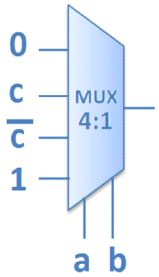
تابع $f(a,b,c) = \sum m(0,1,2,5,7)$ را با یک دیکـدر 3:8 و یک گیت NOR و تعداد دلخواه گیت NOT پیاده‌سازی نمایید

سوال 2: (1 نمره)

یک دیکـدر 5:32 را با چهار دیکـدر 3:8 و یک دیکـدر 2:4 بسازید متغیرهای ورودی a,b,c,d,e هستند (a با ارزشترین متغیر است)

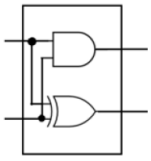
سوال 3: (2 نمره)

مـــدار تابع شکل مقابل را فقط با استفاده از سه گیت NOR و یک گیت NOT پیاده‌سازی نمایید



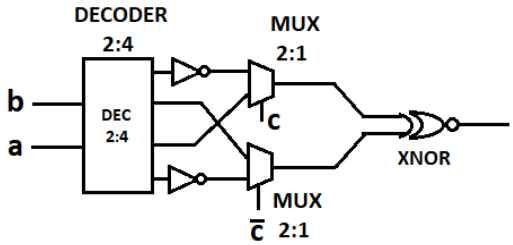
سوال 4: (1 نمره)

شکل مقابل یک نیم جمع کننده است با استفاده از دو نیم جمع کننده و هر تعداد گیت دلخواه یک تمام جمع کننده بسازید



سوال 5: (2 نمره)

تابع (جدول درستی) مدار شکل مقابل را بدست آورید



سوال 6: (2 نمره)

با توجه به جدول درستی تمام تفریق کننده تابع قسمت b out را فقط با استفاده از یک مالتی‌پلکسر 2:1 و یک گیت AND و یک گیت OR بسازید

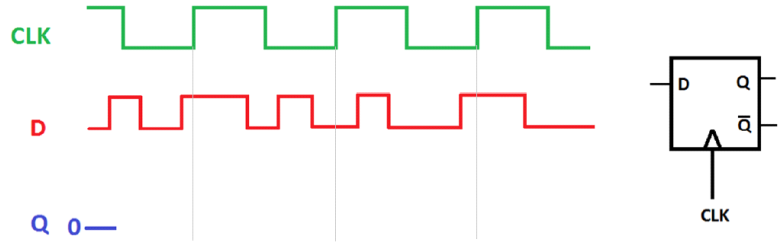
a	b	b in	d	b out
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

سوال 7: (1 نمره)

با استفاده از یک فلیپ فلاپ T یک فلیپ فلاپ SR بسازید

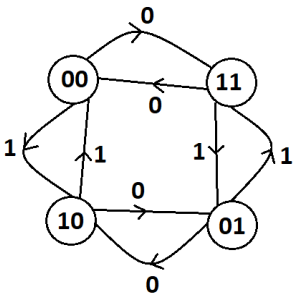
سوال 8: (1 نمره)

نمودار خروجی فلیپ فلاپ D شکل زیر را با توجه به ورودی و پالس ساعت CLK ترسیم نمایید مقدار اولیه Q = 0 می باشد



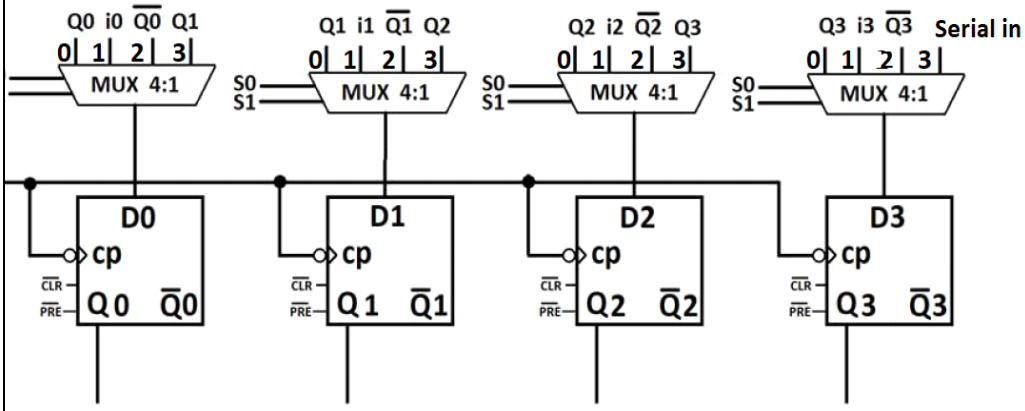
سوال 9: (2 نمره)

ماشین میلـــی (فاقد خروجی) زیر را با استفاده از فلیپ فلاپ های JK طراحی نمایید



سوال 10: (2 نمره)

جدول عملکرد ثبات زیر را تکـــمیل نمایید



S1	S0	function
0	0	
0	1	
1	0	
1	1	

ضمیمه شماره شش

نمونه سوالات میانترم مدار منطقی نیمسال اول 98-99

سوال 1:

الف) عدد $(54762)_8$ را از مبنای 8 به مبنای 4 و مبنای 2 تبدیل نمایید.

ب) عدد $(1011101101)_b$ را بصورت کد گـــری بنویسید.

ج) عدد $(122102)_9$ را از مبنای 9 به مبنای 3 تبدیل نمایید.

د) عدد 3 را در سیستم عددی $84\bar{2}\bar{1}$ بنویسید.

ه) اگر $(AAA)_3 = (1A2)_4$ مقدار A را محاسبه نمایید.

سوال 2) الف) داده مقابل با روش همینگ دریافت شده است. اگر فقط احتمال بروز یک خطا وجود داشته باشد آیا داده صحیح است ؟ اگر نیست تصحیح گردد.

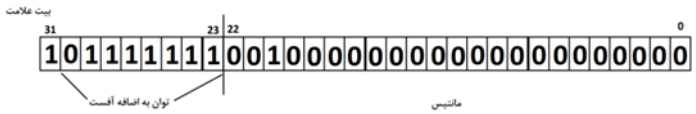
d4	d3	d2	P3	d1	P2	P1
1	1	1	0	1	0	1

ب) داده مقابل با روش همینگ ارسال کنید d4 d3 d2 d1 = 0011

ج) اگر حداقل فاصله همینگ در یک کد برابر 8 باشد و این کد قابلیت اصلاح 2 خطا را داشته باشد ***علاوه بر آن*** چند خطای دیگر را می تواند تشخیص دهد ؟

سوال 3 :

الف) عدد زیر به روش IEEE754 ذخیره شده است آن عدد را محاسبه نمایید.



ب) عدد 12.875- به روش IEEE754 نمایش دهید

سوال 4 :

الف) تابع $f(a,b,c,d) = \prod M(1,3,4,9,11,12) \times D(5,7,13,15)$ را به روش POS از طریق نقشه کارنو ساده نمایید.

ب) تابع $f(a,b,c,d,e) = \sum m(7,9,11,15,23,24,25,26,27,31) + d(4,5,16,18)$ را به روش SOP از طریق نقشه کارنو ساده نمایید.

سوال 5 : الف) تابع $f(a,b,c) = \sum m(0,1,2,4,6,7)$ را به روش کوئین مک‌کلاسکی ساده نمایید.

ب) تابع $f(a,b,c,d) = \sum m(0,3,4,6,8,9,11,12,14) + d(1,2,15)$ را به روش POS از طریق کوئین مک‌کلاسکی ساده نمایید.