معماری کامپیوتر

به نام خدا

تمرین اول معماری کامپیوتر (یادآوری برخی مباحث از مدار منطقی)

1400/7/18

رضا صومی

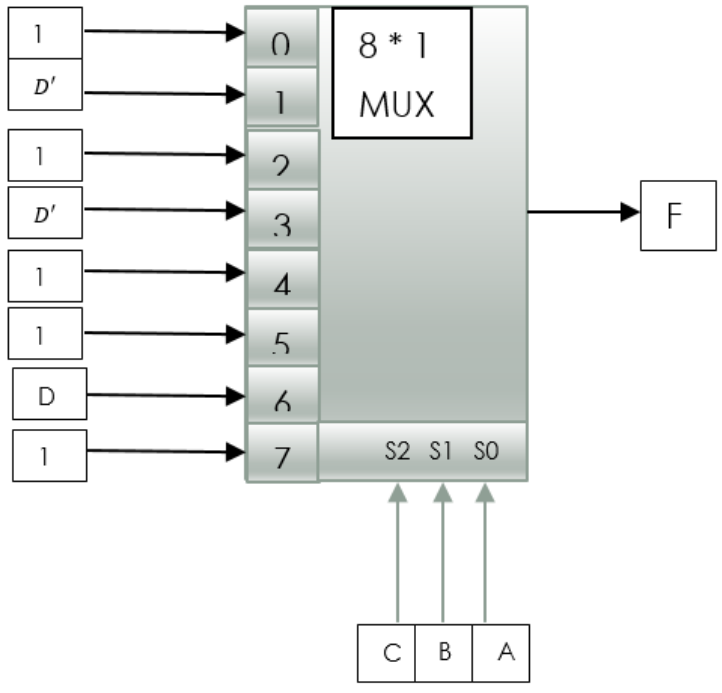
تمرین اول معماری کامپیوتر (یاداوری برخی مباحث از مدار منطقی)

.1

بله می توان این کار را انجام داد. می دانیم هر MUX می تواند چندیدن ورودی Select داشته باشد و هر مدار ترکیبی دلخواه چند input دارد. برای پیاده سازی مدار ترکیبی با mux چندین راه حل وجود دارد که ساده ترین آن عبارت است از: فرض کنیم مدار ترکیبی مورد نظر n ورودی دارد. n-1 ورودی آن را انتخاب و به عنوان ورودی های select مالتی پلکسر قرار می دهیم. آنگاه ورودی های مالتی پلکسر بر اساس ورودی دیگر مدار ترکیبی و 0 و 1 منطقی تعیین می شود. برای مثال

If $F = \sum(0, 1, 2, 4, 5, 6, 8, 9, 10, 11, 13, 14, 15)$

Inputs ABCD	F	
0000	1	$F = 1$
0001	1	
0010	1	$F = D'$
0011	0	
0100	1	$F = 1$
0101	1	
0110	1	$F = D'$
0111	0	
1000	1	$F = 1$
1001	1	
1010	1	$F = 1$
1011	1	
1100	0	$F = D$
1101	1	
1110	1	$F = 1$
1111	1	

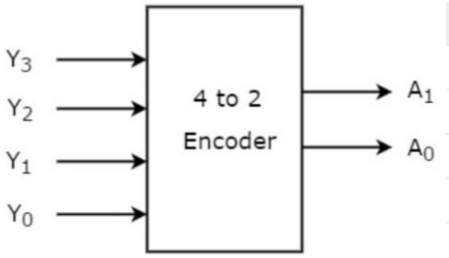


D' را نیز می توان با یک MUX 2*1 تولید کرد. کافیست S_0 را D قرار دهیم و ورودی 0 را 1 و ورودی 1 را 0 قرار دهیم.

حال اگر تنها استفاده از تعداد نامحدودی تسهیم کننده دو به یک داشته باشیم باز هم جواب بله است. خروجی یک تابع ترکیبی را می توان به صورت یک SOP یا POS نوشت. برای مثال خروجی تابع برابر شده با $AB' + C'D$. برای تولید B' و C' خط select را B یا C قرار داده و ورودی 0 را 1 و ورودی 1 را 0 قرار دهیم. حال برای تولید AB' کافیست یکی از آنها را (A یا B') به خط select داده و دیگری را به ورودی 1 مالتی پلکسر وصل کنیم و ورودی 0 مالتی پلکسر را نیز 0 قرار می دهیم چرا که اگر یکی از ورودی ها صفر باشد جواب نهایی نیز حتما صفر است. به همین طریق $C'D$ را می شود تولید کرد. تنها عملیات مورد نظر دیگر OR منطقی است. برای این کار کافیست یکی از ورودی های موجود در اینجا AB' یا $C'D$ را به خط select متصل کرده و ورودی دیگر را این بار به ورودی 0 مالتی پلکسر وصل می کنیم و ورودی 1 را برابر 1 قرار می دهیم چرا که اگر یکی از ورودی ها 1 باشد خروجی 1 منطقی می شود. دقت کنید که در اینجا عملیات sum و product دو متغیره با MUX را بیان کردیم و برای تولید عبارت های با بیش از دو متغیر برای مثال $ABCD$ باید خروجی هر مرحله را با بعدی product کرد (ابتدا برای مثال AB را بدست می آید سپس خروجی آن با C و در نهایت خروجی آن با D به شیوه ای که ذکر شد product می شود.) لذا هر تابعی را می توان به این صورت پیاده سازی کرد.

.2

لطفا به تصویر 2.1 که یک encoder چهار به دو است و جدول ورودی و خروجی های آن توجه کنید.



Inputs				Outputs	
Y_3	Y_2	Y_1	Y_0	A_1	A_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

تصویر 2.1 . 4 to 2 encoder with table

در سطح مدار منطقی encoder بالا چیزی جز دو گیت or نیست در نتیجه A_1 برابر می شود با $Y_3 + Y_2$ و A_0 برابر می شود با $Y_3 + Y_1$

در نتیجه ابهامی وجود دارد اگر خروجی 00 باشد نمی توان فهمید تمام ورودی ها 0 هستند یا به صورت خط اول در شکل بالا هستند چرا که اگر همه ورودی ها صفر باشند با توجه به پیاده سازی آن در سطح گیت می تواند خروجی 00 تولید کند. همچنین اگر بیش از یک ورودی active high باشد، خروجی لزوما درست نیست. فرض کنید Y_1 و Y_3 همزمان 1 باشند، خروجی 11 نمایش داده می شود در صورتی که این خروجی نه مربوط به هنگامی است که Y_1، 1 است نه هنگامی که Y_3، 1 است.

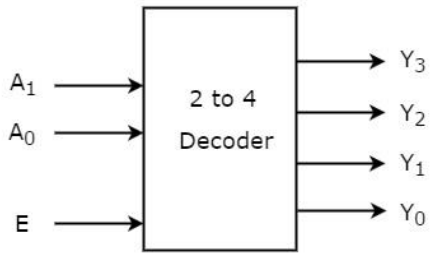
برای فائق آمدن بر این مشکلات راه حل priority encoder ارائه شده است که هم مشکل صفر بودن تمام ورودی ها بر طرف شده است (در صورت active low بودن تمام ورودی ها خروجی نیز active low یا همان صفر است.) و هم به ورودی ها نوعی اولویت (MSB TO LSB) نسبت می دهیم. و حال اگر Y_1 و Y_3 همزمان active high باشند خروجی ما به طور قطع 11 است. جدول درستی 4 to 2 priority encoder را در تصویر 2.2 مشاهده می کنید.

Inputs				Outputs		
Y_3	Y_2	Y_1	Y_0	A_1	A_0	V
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	x	0	1	1
0	1	x	x	1	0	1
1	x	x	x	1	1	1

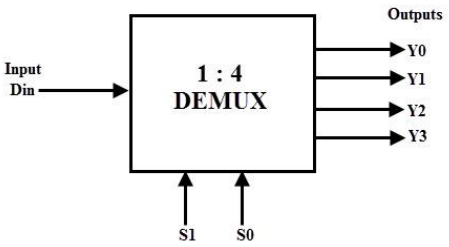
4 to 2 priority encoder with table 2.2تصویر .

.3

به تصاویر پایین توجه کنید. در این تصاویر n = 2 در نظر گرفته شده است.



2 to 4 Decoder 3.1 تصویر .



1 to 4 DEMUX 3.2 تصویر .

هدف تبدیل یک $DEMUX\ 1 \times 2^n$ به $Decoder\ n \times 2^n$ است. برای ساده سازی n = 2 در نظر گرفته شده است. همانطور که در تصویر 3.1 مشاهده می کنید DEMUX دارای یک input و خطوط select است و بسته به خطوط select ورودی را به یکی از خطوط خروجی وصل می کند (خطوط select سوزنبانی می کنند). حال به decoder توجه کنید که بر اساس ورودی موجود یکی از خروجی ها را فعال می کند و مقدار آن را برابر 1 قرار می دهد. حال اگر مقدار A_1 و A_0 را به ترتیب در S_1 و S_0 قرار دهیم و همچنین input را برابر 1 بگذاریم خروجی DEMUX و decoder دقیقا مشابه هم می شوند(enable در هر دو فعال باشد) حال برای n نیز به همین طریق است. اگر A_0 تا A_{n-1} را به خطوط select یعنی S_0 تا S_{n-1} وصل کنیم و input را برابر 1 بگذاریم، خروجی ما همان چیزی خواهد بود که در decoder خواهد بود.

4.جدول درستی عبارت را در پایین مشاهده می کنید.

a = 0

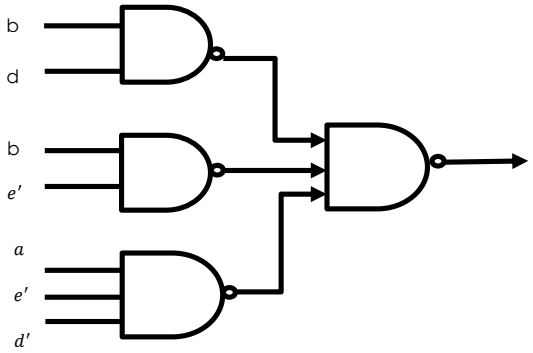
bc \ de	00	01	11	10
00				x
01			x	
11	1		x	1
10	1	x	1	1

a = 1

bc \ de	00	01	11	10
00	1			
01	x			
11	1	x	1	1
10	1		1	1

دسته شامل مربع های $a'bd$ و abd جمله bd را تولید می کند. همچنین ستون سمت چپ در a = 1 نیز $ad'e'$ را تولید می کند. همچنین برای انتخاب 1 های باقی مانده از راهکاری که در شکل با خطوط نمایش داده شده استفاده می کنیم تا تعداد عناصر min باشد. لذا داریم :

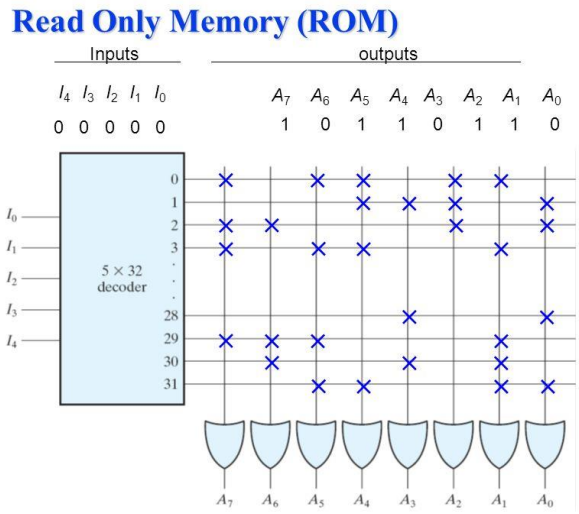
$$F = bd + be' + ad'e' = ((bd)'.(be')'.(ad'e')')'$$



گزینه اول صحیح می باشد.

.5

می دانیم یک Decoder، 2^k مینترم را برای k متغیر ورودی تولید می کند. با اضافه شدن گیت های OR برای بدست آوردن جمع مینترم های توابع بول می توان هر مدار ترکیبی را تولید کرد. ROM وسیله ای است که هر دو بخش Decoder و گیت های OR را در خود دارد. از آنجایی که هر مدار ترکیبی را می توان با نقشه کارنو و مینترم های موجود مشخص کرد لذا با انتخاب اتصالات مناسب برای گیت های OR می توان خروجی دلخواه را تولید کرد. برای این منظور به تصویر 5.1 توجه کنید.



تصویر 5.1 ROM .

دلیل انتخاب چندین اتصال و OR کردن همه آن ها با هم این است که هر بیت خروجی را می توان بر اساس ترکیبی از ورودی ها حساب کرد به صورت SOP و این بیت خروجی ممکن است حاصل جمع چندین عبارت باشد که این اتفاق صورت می پذیرد.

.6

ROM ها از نظر روش برنامه ریزی انواع مختلفی دارند. از جمله روش برنامه ریزی ماسک که مستلزم پر شدن جدول درستی مربوط به سفارش دهنده است و با قالب خاص ایجاد می گردد. نوع دیگر آن PROM(Programmable read only memory) است که اقتصادی تر است. هنگام سفارش این قطعه همه فیوز ها به اصطلاح دست نخورده هستند که به معنی 1 بودن منطقی کلمه است. می توان فیوز ها را در آزمایشگاه با پالس ولتاژ قوی سوزاند و در نتیجه فیوز های سوخته منطق 0 و فیوز های دست نخورده منطق 1 را ایجاد می نماید. این ویژگی قابل برنامه ریزی بودن PROM را نشان می دهد اما روال برنامه ریزی این قطعات برگشت ناپذیرند و الگوی آن تغییر ناپذیر است. و اما نوع دیگر که ویژگی مثبت PROM را دارد و همچنین این مشکل برگشت ناپذیری PROM را رفع می کند EPROM(Erasable programmable read only memory) است که می توان آن را به حالت اولیه اش تبدیل کرد. وقتی EPROM را برای مدتی معین تحت تابش اشعه ماورا بنفش قرار بگیرد اشعه موج گیت های درونی را تخلیه می کنند و به حالت اولیه در می آیند که دوباره قابل برنامه ریزی است.

یکی از کاستی های آن این است که داده های ذخیره شده در EPROM می تواند به صورت تصادفی با جریانی از انرژی الکتریکی یا مغناطیسی دست کاری شود .

.7

(آ).

$$total\ number\ of\ inputs = num1 + num2 + C_{in} + mode(if\ 0 \rightarrow add, if\ 1 \rightarrow sub) = 32 + 32 + 1 + 1 = 66$$

$$total\ number\ of\ outputs = addedNum + C_{out} = 32 + 1 = 33$$

$$total\ number\ of\ input\ line\ we\ can\ address = 2^{66}$$

$$ROM\ size = total\ number\ words \times number\ of\ bit\ per\ words = 2^{66} \times 33$$

عرض بیتی، همان تعداد بیت در یک سطر از جدول درستی است. در واقع برابر است با تعداد بیت آدرس + تعداد بیت داده در حافظه ROM.

$$\text{عرض بیتی} = 66 + 33 = 99$$

A(32 bit)	B(32 bit)	C_{in}	C_{out}	Output(32 bit)
00……0	00…..0	1	0	00…001
100…..0	100……0	0	1	00……0

(ب):

$total\ number\ of\ inputs = num1 + num2 = 32 + 32 = 64$

$total\ number\ of\ outputs = multiplied\ number = 64$

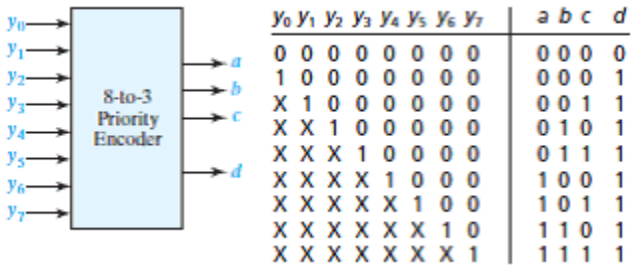
$total\ number\ of\ input\ line\ we\ can\ address = 2^{64}$

$ROM\ size = total\ number\ words \times number\ of\ bit\ per\ words = 2^{64} \times 64$

عرض بیتی $= 64 + 64 = 128$

A(32 bit)	B(32 bit)	Output(64 bit)
00……0	00…..0	00…000
100…..0	100……0	100……0

(ج):



y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7	a	b	c	d
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
X	1	0	0	0	0	0	0	0	0	1	1
X	X	1	0	0	0	0	0	0	1	0	1
X	X	X	1	0	0	0	0	0	1	1	1
X	X	X	X	1	0	0	0	1	0	0	1
X	X	X	X	X	1	0	0	1	0	1	1
X	X	X	X	X	X	1	0	1	1	0	1
X	X	X	X	X	X	X	1	1	1	1	1

تصویر 7.1 8-to-3 priority encoder .

$total\ number\ of\ input\ line\ we\ can\ address$

$= each\ row\ of\ picture\ from\ up\ to\ down\ (for\ every\ x\ we\ can\ have\ 0\ or\ 1)$

$= 1 + 1 + 2 + 4 + 8 + 16 + 32 + 64 + 128 = 256 = 2^8$

$total\ number\ of\ outputs = 3\ bit\ encoded\ number + 1bit\ (if\ input = 00 \dots 0 \rightarrow 0\ else \rightarrow 1) = 4$

$ROM\ size = total\ number\ words \times number\ of\ bit\ per\ words = 2^8 \times 4$

عرض بیتی $= 8 + 4 = 12$

Y(8 bit)	Output(3 bit)	d
00000000	000	0
10110100	111	1

(د):

$total\ number\ of\ inputs = 4bit\ num1 + 4bit\ num2 + select\ input + enable\ input\ = 4 + 4 + 1 + 1 = 10$

$total\ number\ of\ outputs = 4bit\ num = 4$

$total\ number\ of\ input\ line\ we\ can\ address =\ 2^{10}$

$ROM\ size = total\ number\ words \times number\ of\ bit\ per\ words = 2^{10} \times 4$

عرض بیتی $= 10 + 4 = 14$

A(4 bit)	B(4 bit)	*select*	*enable*	Output(4 bit)
0100	0010	1	1	0010
1010	0101	0	1	1010

مدارهای ترکیبی :

.1

یک گیت OR دو ورودی

$150\ \times A - 100\ \times (A\ mod\ 4) + 2$

input	a_3	a_2	a_1	a_0	output	D_3	D_2	D_1	D_0	C_3	C_2	C_1	C_0	B_3	B_2	B_1	B_0	A_3	A_2	A_1	A_0
0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
1	0	0	0	1	52	0	0	0	0	0	0	0	0	0	1	0	1	0	0	1	0
2	0	0	1	0	102	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0
3	0	0	1	1	152	0	0	0	0	0	0	0	1	0	1	0	1	0	0	1	0
4	0	1	0	0	602	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1	0
5	0	1	0	1	652	0	0	0	0	0	1	1	0	0	1	0	1	0	0	1	0
6	0	1	1	0	702	0	0	0	0	0	1	1	1	0	0	0	0	0	0	1	0
7	0	1	1	1	752	0	0	0	0	0	1	1	1	0	1	0	1	0	0	1	0

8	1	0	0	0	1202	0	0	0	1	0	0	1	0	0	0	0	0	0	0	1	0
9	1	0	0	1	1252	0	0	0	1	0	0	1	0	0	1	0	1	0	0	1	0

خروجی حداکثر 4 رقم است و چون BCD در نظر گرفته می شود لذا 16 بیت برای خروجی در نظر گرفته شده است که از پر ارزش ترین رقم D تا کم ارزش ترین A به طور کامل در جدول درستی بالا نشان داده شده است.

$D_3 = D_2 = D_1 = C_3 = B_3 = B_1 = A_3 = A_2 = A_0 = 0, A_1 = 1$

$B_2 = B_0 = a_0$

$C_0 = a_1$

$C_2 = a_2$

$D_0 = a_3$

$C_1 = a_3 + a_2$

در نتیجه همه خروجی ها به جز C_1 به طور مستقیم از ورودی ها یا 0 و 1 منطقی گرفته می شود و تنها به یک گیت OR دو ورودی برای محاسبه C_1 نیاز است.

.2

YZ \ WX	00	01	11	10
00				x
01	1			x
11	x	1	1	1
10	x	1	1	1

می خواهیم تابع موجود که جدول درستی آن را مشاهده می کنید(برای راحتی کار به جای محاسبه با ماکسترم، از مینترم ها استفاده کردیم) با قانون Shanon و با یک 4 MUX به 1 پیاده سازی کنیم. لذا X و Y را به خطوط select مالتی پلکسر داده و دو متغیر دیگر را از رابطه شانون بسط می دهیم که شکل تابع بر هم نخورد. لذا داریم :

$$f(w,x,y,z) = x'y'f(x',y') + xy'f(x,y') + x'yf(x',y) + xyf(x,y)$$
$$= x'y'(W) + xy'\left(W + Z'\right) + x'y(W) + xy(W) \xrightarrow{yields} I_0 = W, I_1 = W, I_2 = W + Z', I_3 = W$$

.3

جواب: گزینه b

$A = a_4a_3a_2a_2a_1a_0, B = b_4b_3b_2b_2b_1b_0$

می دانیم با شیفت خوردن اعداد باینری به سمت چپ بزرگی اعداد با هر بار شیفت خوردن دو برابر می شود و با شیفت خوردن به سمت راست بزرگی اعداد بر دو تقسیم می شود. در اینجا A یک رقم و B دو رقم به سمت چپ شیفت خورده است لذا A دو برابر و B چهار برابر می شود. همچنین در مرتبه هفتم ورودی adder اول و قسمت A، 1 قرار گرفته است که با توجه به اینکه در مرتبه هفتم قرار دارد لذا ارزش آن $2^{7-1} = 2^6$ است. از آنجایی که ورودی های adder اول بزرگ نیستند و همچنین MSB هر دو صفر است لذا C_{out} همواره صفر خواهد بود. پس خروجی adder اول برابر می شود با :

$output\ of\ adder\ 1 = (2A + 64) + 4B$

یکی از ورودی های adder دوم وابسته به خروجی adder اول است تنها با این تغییر که خروجی یک بیت به سمت راست شیفت خورده است (می دانیم LSB صفر است چرا که LSB هر دو ورودی adder اول صفر و همچنین C_{in} نیز صفر است) لذا یکی از ورودی ها adder دوم برابر است با :

$$\frac{output\ of\ adder\ 1}{2} = \frac{(2A + 64) + 4B}{2} = A + 2B + 32$$

حال شرط روی b_4 باید گذاشته شود چرا که ورودی adder دوم وابسته به b_4 نیز هست.

$$if\ B < 16 \xrightarrow{yields}\ b_4 = 0 \xrightarrow{yields} other\ input\ of\ twice\ adder = 0, C_{in}\ of\ twice\ adder = 1 \xrightarrow{yields} output = A + 2B + 32 + 1 = A + 2B + 33$$

$$if\ B \geq 16 \xrightarrow{yields}\ b_4 = 1 \xrightarrow{yields} other\ input\ of\ twice\ adder = 255, C_{in}\ of\ twice\ adder = 0 \xrightarrow{yields} output = A + 2B + 32 + 255 \xrightarrow{output\ is\ only\ 8\ bit\ so\ \ C_{out}\ is\ 1\ and\ must\ minus\ it\ from\ 256} A + 2B + 287 - 256 = A + 2B + 31$$

.4

جواب : گزینه آ

با استفاده از عددگذاری و حذف گزینه به نتیجه می رسیم.

$a_7a_6a_5a_4a_3a_2a_1a_0 = 00100010 = 34$

$b_7b_6b_5b_4b_3b_2b_1b_0 = 00000011 = 3$

$c_7c_6c_5c_4c_3c_2c_1c_0 = 01000001 = 65$

$output = 01100110 = 102 = A + B + C + 1$

مدار های ترتیبی :

.1

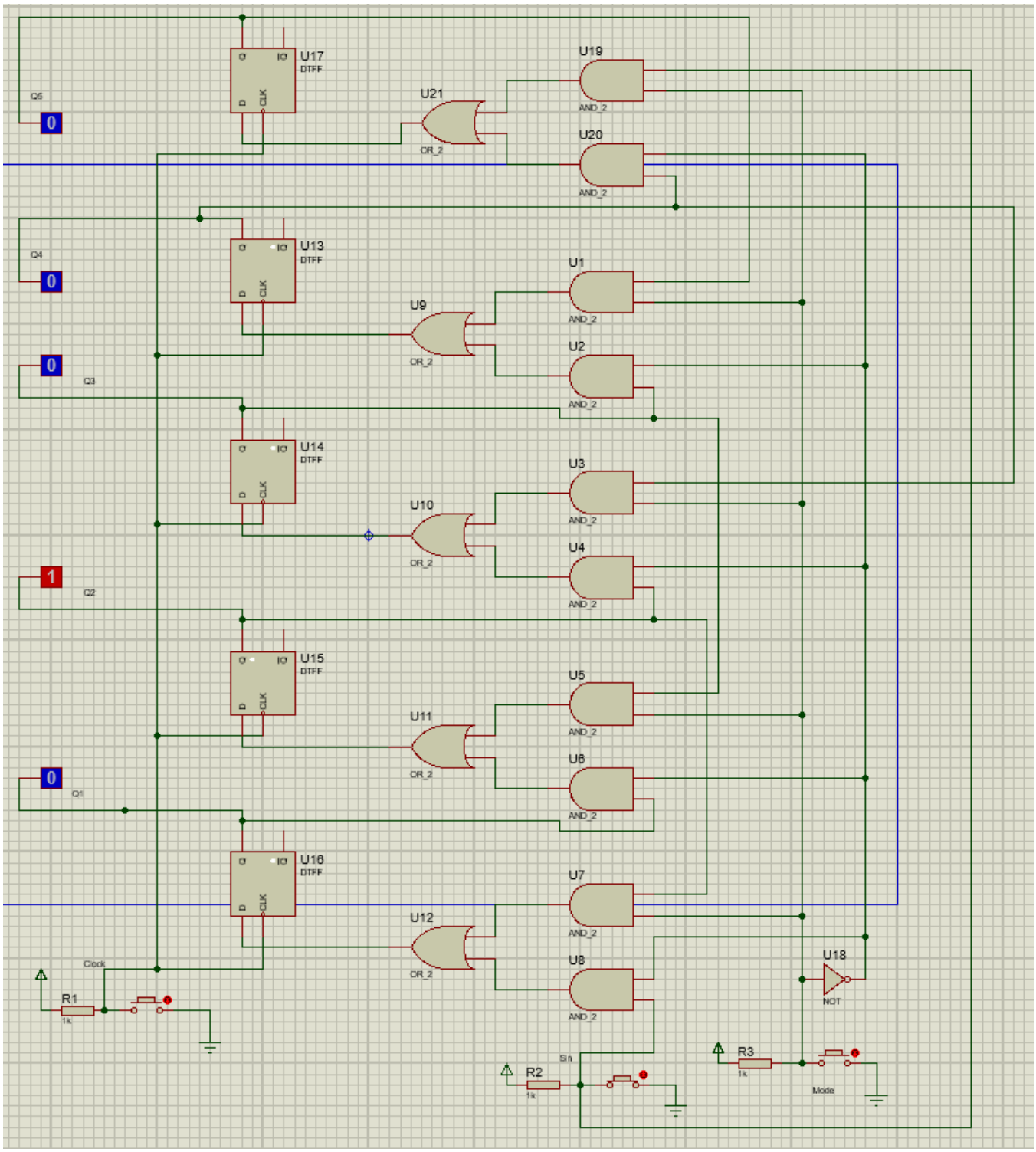
با توجه به feedback loop موجود مشخص است که مدار موجود یک مدار ترتیبی است. به گونه ای این موجودیت کار memory را انجام می دهد به عبارتی خروجی MUX به یکی از ورودی های آن بازگشته است. به دلیل وجود این feedback loop، ورودی های گذشته می توانند خروجی MUX را تعیین کنند. با توجه به این موضوع اگر کلاک که همان select of MUX است، 0 باشد ورودی In به خروجی منتقل می شود و این مقدار به خروجی صفر MUX دوم منتقل شده و چون کلاک آن not کلاک موجود است لذا در اینجا از خروجی قبلی خود که به صورت فیدبک به ورودی صفر وصل شده است می خواند. در این مرحله که کلاک 0 است، به نوعی ورودی In ذخیره می شود اما مقدار آن در out قرار داده نمی شود. هنگامی که کلاک 1 باشد خروجی MUX اول برابر با مقدار قبلی موجود در MUX (همان In زمانی که کلاک صفر بوده)، می شود و این مقدار به ورودی 0 MUX دوم رفته و چون خروجی MUX دوم از ورودی صفر می آید لذا out همان In ذخیره شده خواهد بود. لذا با کلاک 0 مقدار جدید را می توان ذخیره کرد و در کلاک 1 این مقدار را در خروجی مشاهده کرد و تا زمانی که کلاک 1 است همین مقدار در خروجی مشاهده می شود.

$$\begin{cases} out=saved\ In\ and\ can\ add\ new\ In\ to\ circuit & if\ \ clk=0 \\ out=saved\ In & if\ clk=1 \end{cases}$$

.2

به تصویر 2.1 که پیاده سازی شیفت رجیستر 5 بیتی با قابلیت جابجایی به راست و چپ است، توجه کنید.

دقت کنید که اگر Mode برابر صفر باشد شیفت به راست و اگر Mode یک باشد شیفت به چپ انجام می گیرد. اگر قابلیت جابجایی به راست و چپ مد نظر نباشد تنها کافی است خروجی Q هر flip-flop را به ورودی flip-flop بعدی متصل کنیم (از flip-flop که بیت کم ارزش را تولید می کند به سمت flip-flop که بیت پرارزش را تولید می کند) اما برای اضافه شدن این قابلیت نیاز به تفکیک ورودی flip-flop ها داریم که در تصویر 2.1 نحوه پیاده سازی آن را مشاهده می کنید.

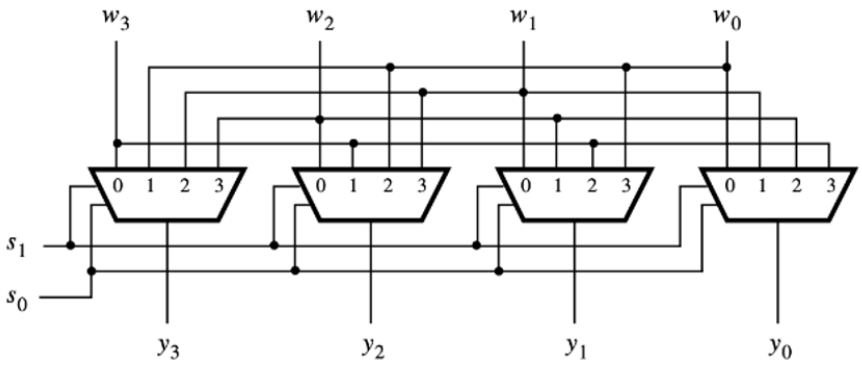


5-bit shift register with right and left shift 2.1 تصویر .

حال به تصویر 2.2 که عملکرد shifter را نشان می دهد، توجه کنید.

s_1	s_0	y_3	y_2	y_1	y_0
0	0	w_3	w_2	w_1	w_0
0	1	w_0	w_3	w_2	w_1
1	0	w_1	w_0	w_3	w_2
1	1	w_2	w_1	w_0	w_3

(a) Truth table

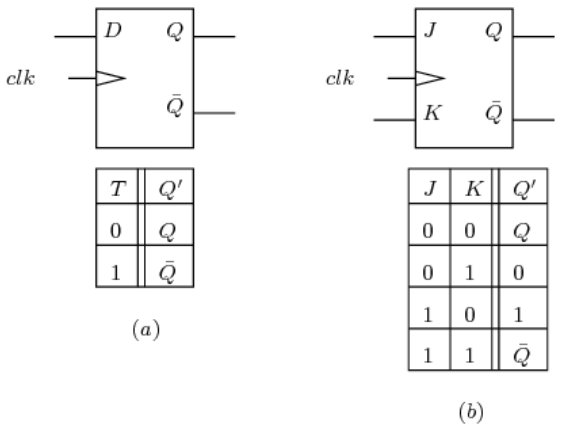


(b) Circuit

4-bit shifter 2.2 تصویر .

Shifter یک مدار ترکیبی است و clock ندارد و خروجی ترکیب آنی ورودی های آن است. با تغییر S_0 و S_1 همانند جدول a می توان عملیات shift را انجام داد. اما در مدار ترتیبی موجود با کلاک این عملیات صورت می گیرد. در مدار ترتیبی ساخته شده برای تغییر نحوه شمارش (up/down) کافیست mode را تغییر دهیم اما در این مدار ترکیبی موجود با افزایش یا کاهش مقدار S_1S_0 می توان شیفت به چپ یا راست را انجام داد. از لحاظ پیاده سازی مدار ترتیبی موجود پیچیده تر است اما قابلیت های بهتری را می تواند در اختیار ما بگذارد از جمله اینکه بیت ورودی shift register را می توان 0 یا 1 قرار داد. مدار ترتیبی که در تصویر 2.1 مشاهده می کنید طوری پیاده سازی شده است که بیت های خروجی به چرخه مدار برنگردد و بیت جدید وارد شود اما در شیفتر موجود همانطور که مشاهده می کنید تمامی انتقال های به چپ و راست روی همان 4 بیت موجود که در ابتدا وارد خطوط w_0 تا w_3 انجام می پذیرد و انعطاف پذیری کمتری دارد. مواقعی هست که ترجیح بر این است که مدار مورد نظر با کلاک به طور مستقیم کار شود لذا استفاده از شیفتر به صرفه نخواهد بود.

به تصویر 3.1 که عملکرد T flip-flop و JK flip-flop را نشان می دهد، توجه کنید.



T	Q'
0	Q
1	$\bar{Q}$

J	K	Q'
0	0	Q
0	1	0
1	0	1
1	1	$\bar{Q}$

تصویر 3.1 JK and T flip-flop .

منظور از Q' در تصویر 3.1 Q(t+1) است.

	Up					Down			
state	A_3	A_2	A_1	A_0	state	A_3	A_2	A_1	A_0
0	0	0	0	0	10	1	0	0	0
1	0	0	0	1	11	1	0	0	1
2	0	0	1	0	12	0	1	1	0
3	0	0	1	1	13	0	1	1	1
4	0	1	0	0	14	0	1	0	0
5	0	1	0	1	15	0	1	0	1
6	0	1	1	0	16	0	0	1	0
7	0	1	1	1	17	0	0	1	1
8	1	0	0	0	18	0	0	0	0
9	1	0	0	1	19	0	0	0	1

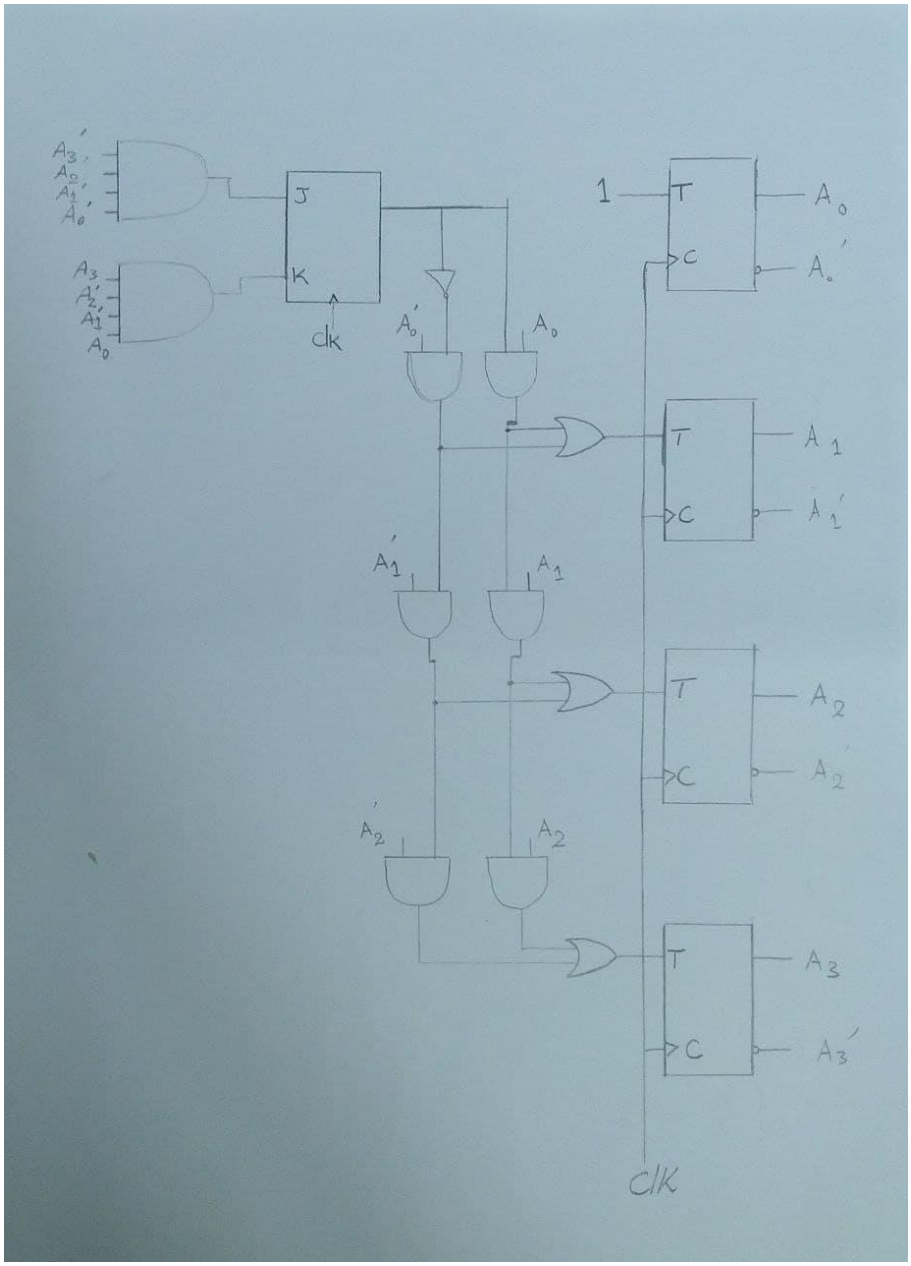
In Up:

$$T_0 = 1, T_1 = A_0, T_2 = A_1A_0, T_3 = A_2A_1A_0$$

In Down:

$$T_0 = 1, T_1 = A_0{}', T_2 = A_1{}'A_0{}', T_3 = A_2{}'A_1{}'A_0{}'$$

به تصویر 3.2 که پیاده سازی این مدار را نشان می دهد، توجه کنید.



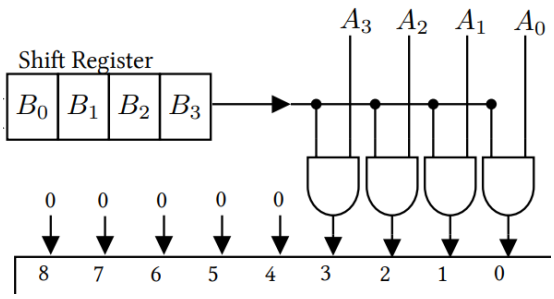
تصویر 3.2 implementation .

برای پیاده سازی خط up/down از JK flip-flop استفاده شده است. طوری برنامه ریزی شده است که اگر خروجی 1 باشد یعنی شمارش رو به بالا قرار است انجام شود و اگر خروجی 0 باشد شمارش رو به پایین است. برای این کار ورودی J عبارت بولی $A_3'A_2'A_1'A_0'$ قرار داده شده است یعنی وقتی این عبارت برابر 0000 شد خروجی 1 شده و به عبارتی up صورت می گیرد. در ورودی k عبارت بولی $A_3A_2'A_1'A_0$ قرار گرفته است یعنی وقتی شمارنده به 9 می رسد خروجی JK flip-flop 0 منطقی شده و شمارش down صورت می گیرد. توجه کنید در صورتی که J و K هر دو صفر باشند تغییری در وضعیت خروجی انجام نمی شود و اگر خروجی 1 باشد (Up) آن را ادامه داده و همینطور اگر 0 باشد (Down) شمارش را به همان طریق ادامه می دهد.

.4

جواب : گزینه آ

به تصویر 4.1 توجه کنید.

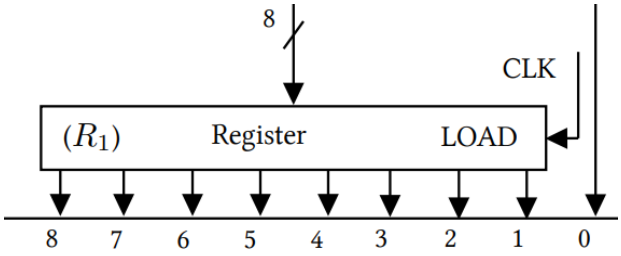


تصویر 4.1 .

خروجی این عبارت و در حقیقت یکی از ورودی های 9-bit adder برابر می شود با مقدار شیفت داده شده shift register که با مقادیر $A_3A_2A_1A_0$ به صورت منطقی and شده باشد. به عبارتی برای مثال پس از اولین کلاک عبارت $B_3(A_3A_2A_1A_0)$ ورودی adder خواهد بود.

$$(if\ B_3 = 0 \xrightarrow{yields} 0\ else\ if\ B_3 = 1 \xrightarrow{yields} A_3A_2A_1A_0)$$

به تصویر 4.2 که ورودی دیگر 9-bit adder موجود است دقت کنید.



تصویر 4.2 .

LSB برابر است با مقدار شیفت داده شده shift register موجود و 8 بیت بعدی برابر است با خروجی 9-bit adder موجود به جز MSB. دقت کنید که MSB of adder طی این چهار پالس حتما صفر می ماند چرا که اگر بیت های A و C را حداکثر مقدار ممکن قرار دهیم خروجی adder پس از 4 پالس 240 خواهد بود.

$$after\ first\ clock = B_3(A_3A_2A_1A_0) + C_3 + 2\ (register) \xrightarrow{initial\ value\ of\ register=0} B_3(A_3A_2A_1A_0) + C_3$$

$$after\ twice\ clock$$

$$= B_2(A_3A_2A_1A_0) + C_2 + 2\ (register) \xrightarrow{register=B_3(A_3A_2A_1A_0)+C_3} (2B_3 + B_2)(A_3A_2A_1A_0) + 2C_3 + C_2 = X$$

$$after\ third\ clock$$

$$= B_1(A_3A_2A_1A_0) + C_1 + 2\ (register) \xrightarrow{register=X} (4B_3 + 2B_2 + B_1)(A_3A_2A_1A_0) + 4C_3 + 2C_2 + C_1 = Y$$

$$after\ fourth\ clock$$

$$= B_0(A_3A_2A_1A_0) + C_0 + 2\ (register) \xrightarrow{register=Y} (8B_3 + 4B_2 + 2B_1 + B_0)(A_3A_2A_1A_0) + 8C_3 + 4C_2 + 2C_1 + C_0 = final\ num = (B_3B_2B_1B_0)(A_3A_2A_1A_0) + C_3C_2C_1C_0$$