به نام خدا

درس : آزمایشگاه معماری کامپیوتر

استاد : دکتر سربازی

آزمایش پنجم

اعضای گروه :

سید آرین علوی رضوی راوری ۴۰۰۱۰۹۷۹۲

سیده فاطمه موسوی ۴۰۰۱۰۵۲۵۲

محمدعرفان سلیما ۴۰۰۱۰۵۰۱۴

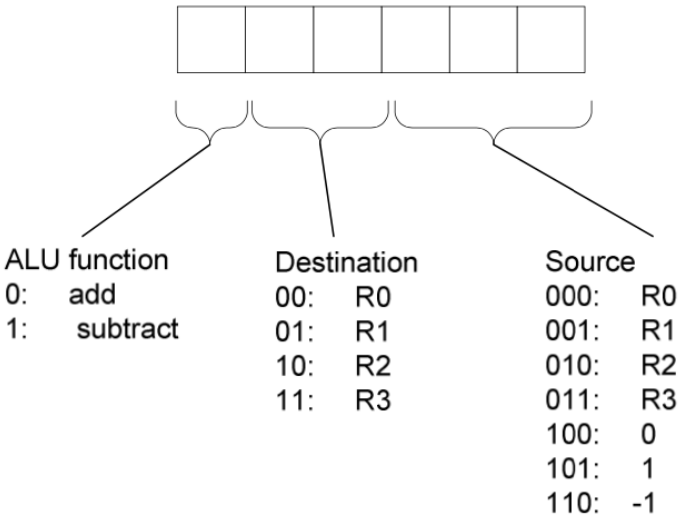
آرش ضیایی رازبان ۴۰۰۱۰۵۱۰۹

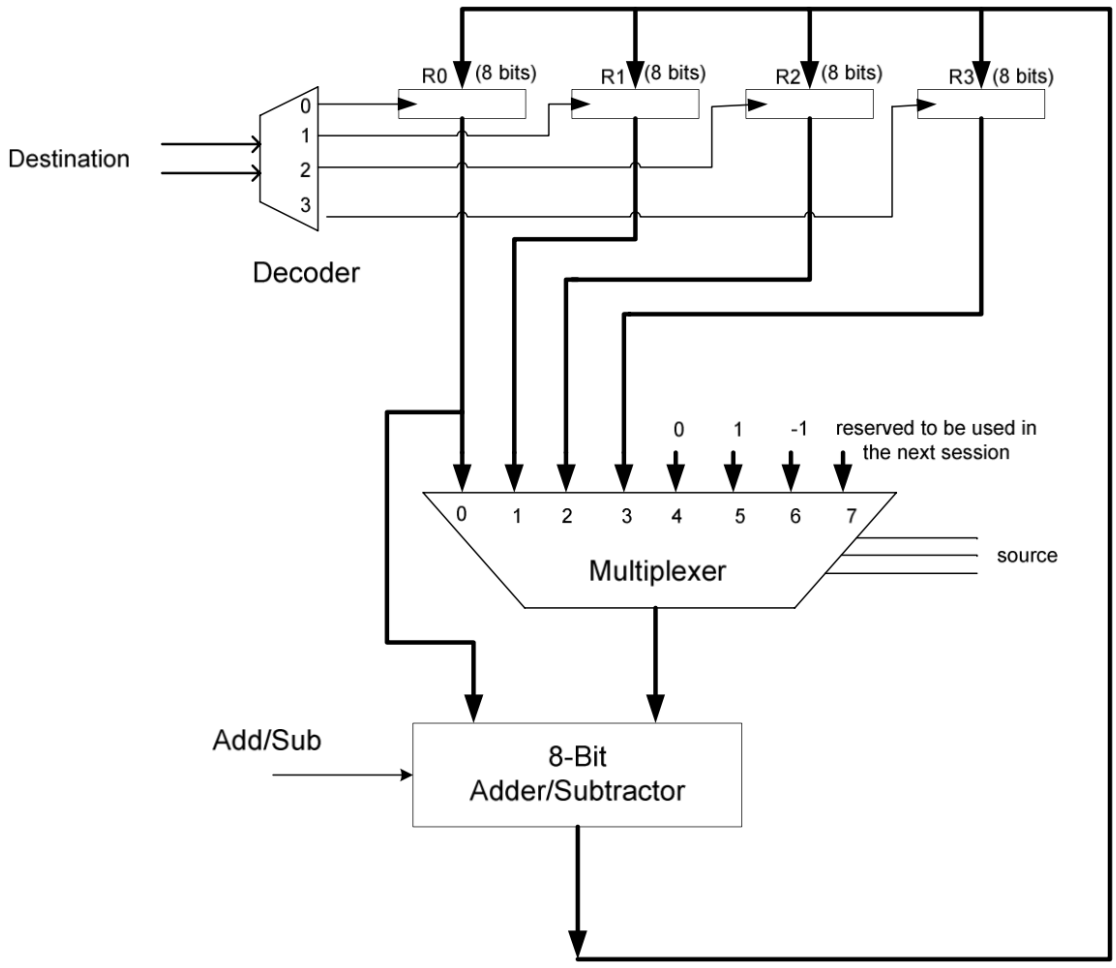
مرداد ۱۴۰۲

پروتئوس آزمایش :

هدف آزمایش :

در این آزمایش می خواهیم با استفاده از نرم افزار پروتئوس یک واحد محاسبات و مجموعه ثبات های عمومی ماشین را طراحی و پیاده سازی کنیم. فرمت دستور داده شده به ماشین یک دستور ۶ بیتی است که بیت اول آن برای مشخص شدن تابع موردنظر(۰: add, ۱: sub)، دو بیت بعدی برای مشخص کردن ثبات مقصد (باتوجه به اینکه ۴ ثبات داریم، برای آدرس دهی به آنها دو بیت کافی است) و ۳ بیت آخر برای مشخص کردن مقدار Source Register است که یا با سه مقدار ۰, ۱, ۱- و یا با محتوای ۴ ثبات داخل ماشین پر می شود. در این معماری، یکی از عملوند های واحد محاسبات ثبات ۰ است. فرمت دستورات و معماری ماشین را در تصاویر زیر می توانید مشاهده کنید:




R0 (8 bits)
R1 (8 bits)
R2 (8 bits)
R3 (8 bits)
Destination
0
1
2
3
Decoder
0
1
-1
reserved to be used in the next session
0 1 2 3 4 5 6 7
Multiplexer
source
Add/Sub
8-Bit Adder/Subtractor

شرح آزمایش :

معماری ماشین مورد نظر از سه قسمت اصلی تشکیل شده است:

۱. مشخص کردن رجیستر مقصد

در این قسمت بیت های ۳و۴ دستور وارد یک دیکودر می شوند. خروجی دیکودر مشخص می کند که ثبات مقصد کدام یک از ثبات های داخل ماشین است و بدین ترتیب سیگنال لود آن را روشن می کند. با مشخص شدن این ثبات و با گذشت پالس ساعت، داده در ثبات مربوطه لود می شود.

۲. مولتی پلکسر ۸ به ۱

از این نوع مالتی پلسکر استفاده می کنیم که Source را مشخص کنیم. سیگنال های select که سه بیت آخر دستور هستند وارد این مالتی پلکسر می شوند و مالتی پلکسر با توجه به این سیگنال ها ثبات Source را مشخص می کند.

۳. واحد محاسبات

در این بخش باتوجه به بیت آخر دستور، عملیات مربوطه(جمع یا تفریق) با محتوای داخل ثبات های انتخاب شده انجام می شود و خروجی محاسبات داخل ثبات مقصد قرار می گیرد.

طراحی و پیاده سازی :

اکنون سه بخش اصلی معماری که در قسمت قبلی توضیح دادیم را در نرم افزار پروتیوس طراحی و پیاده سازی می کنیم:

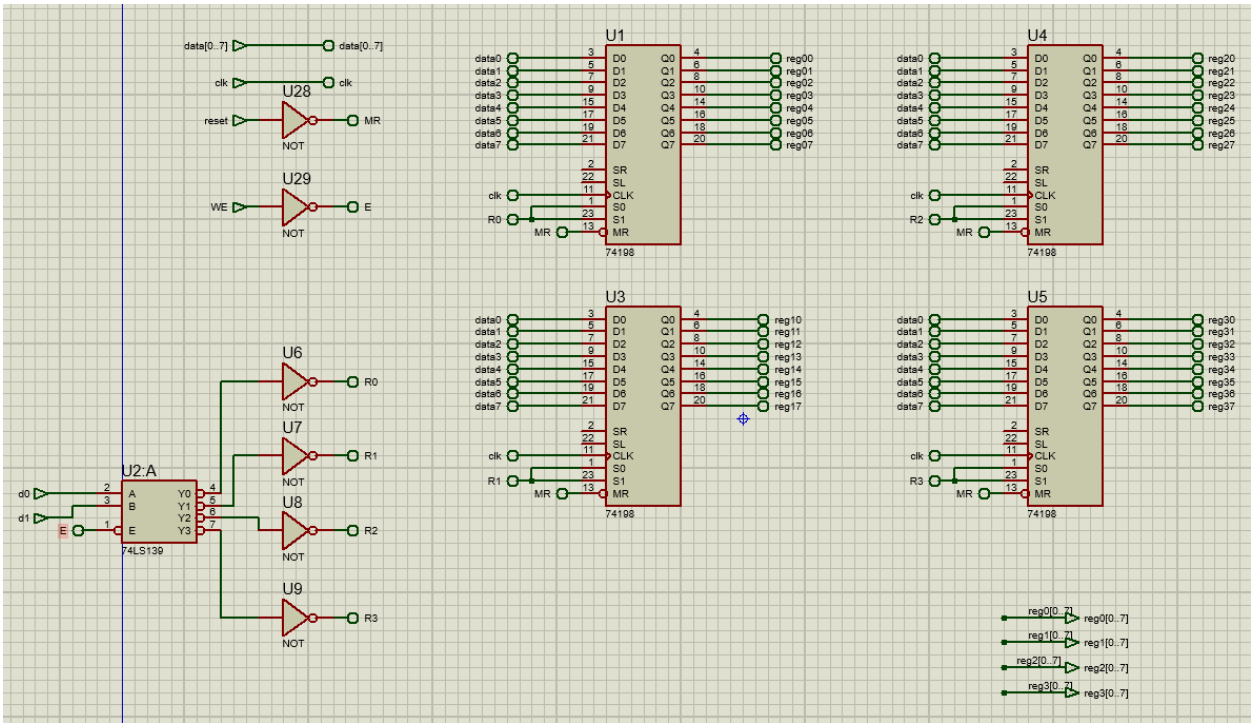
۱. Register File

Inputs: clock, 8 bits data for destination register, 2 bits index

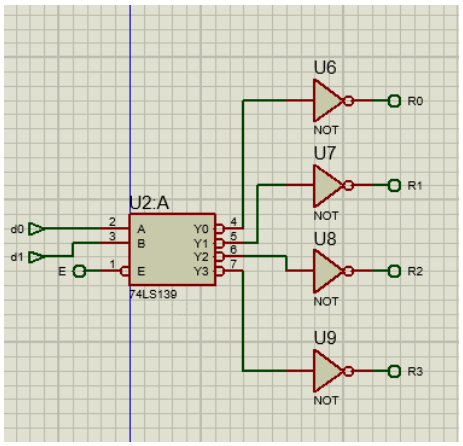
Outputs: Values of registers

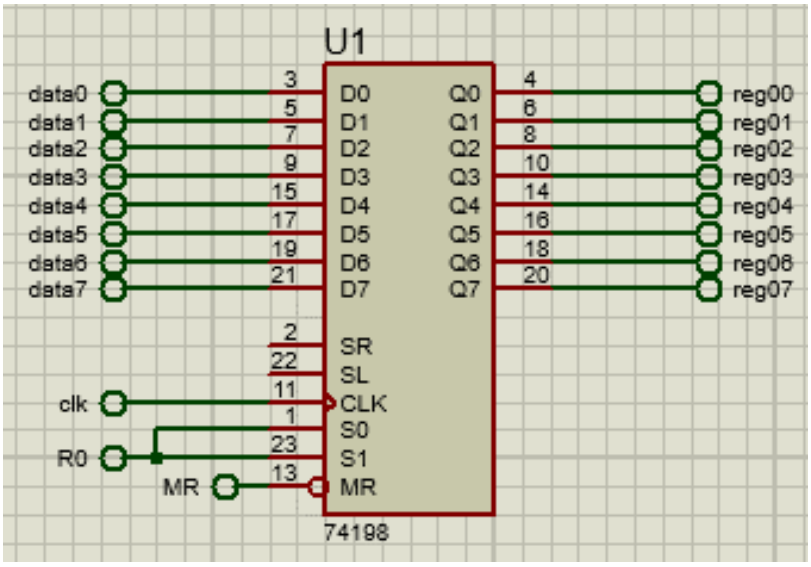
Circuit parts: Decoder (74139), Register (74198)

در این قسمت دو بیت مربوط به ایندکس وارد یک دیکودر می شوند. سپس، با توجه به ایندکس مربوطه یکی از خروجی های دیکودر صفر و بقیه یک می شوند. بنابراین، not خروجی های دیکودر را به ورودی لود رجیستر ها می دهیم (برای لود کردن در رجیستر ها از سیگنال های S_0, S_1 را باید یک کنیم. بنابراین، این دو سیگنال را به خروجی های دیکودر وصل می کنیم). همچنین، باتوجه به اینکه می خواهیم مقادیر داخل رجیستر ها پاک نشود، سیگنال MR را یک می گذاریم. کلاک را نیز به سیگنال کلاک رجیستر ها متصل می کنیم. ۸ بیت داده ی ورودی را به ورودی های هر چهار رجیستر وصل می کنیم تا با هر کلاک با توجه به ایندکس مربوطه، داده در رجیستر موردنظر لود شود. سپس، مقادیر داخل رجیستر ها را به عنوان خروجی قرار می دهیم تا به صورت جدا وارد مولتی پلکسر ۱*۸ شوند.



Register File





قطعات مربوطه به ترتیب از راست به چپ: رجیستر، دیکودر

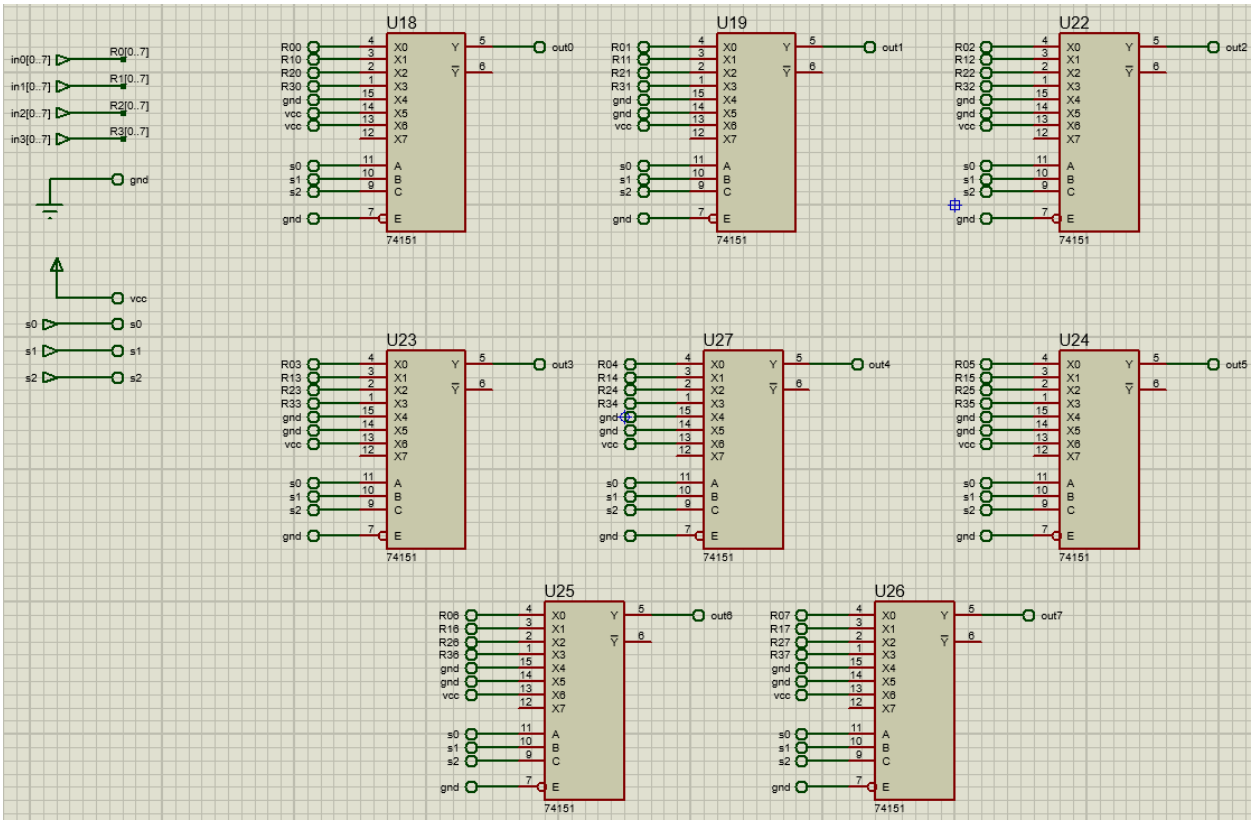
MUX 8 * 1 .۲

Inputs: values of registers, 3 bit select line (source)

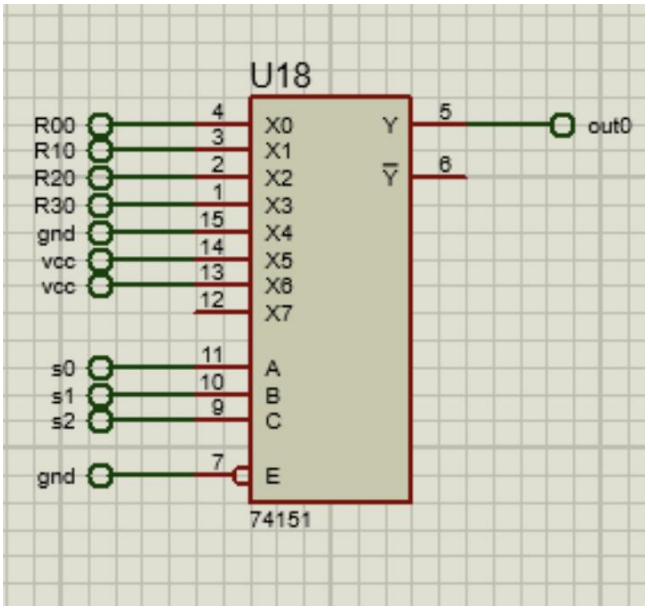
Outputs: 8-bit output

Circuit parts: MUX 8*1 (74151)

در این قسمت یک مولتی پلکسر ۱*۸ دارای ورودی های ۸ بیتی و خروجی ۸ بیتی طراحی می کنیم. با توجه به اینکه مقادیر داخل رجیستر ها ۸ بیت هستند، پس باید از ۸ عدد مولتی پلکسر ۱*۸ با خروجی تک بیتی با ۳ خط آدرس استفاده کنیم. ۴ ورودی اول تمام مولتی پلکسر ها را به ترتیب به بیت های ۴ رجیستر اختصاص می دهیم. در سه ورودی بعدی باید اعداد صفر(۰۰۰۰۰۰۰۰) و ۱ (۰۰۰۰۰۰۰۱) و ۱- (۱۱۱۱۱۱۱۱) قرار داشته باشند؛ لذا، برای ورودی ۴ و ۶ مالتی پلسکر ها را به ترتیب صفر و یک می دهیم. برای ورودی ۵ که یک است، ورودی ۵ اولین مولتی پلکسر را یک و بقیه را صفر می دهیم(توجه شود که شماره اولین ورودی مولتی پلکسر برابر ۰ است). بیت های آدرس (source) را نیز به ترتیب به بیت های سلکت همه ی مولتی پلکسر ها (پایه های A, B, C) وصل می کنیم. سیگنال enable مولتی پلکسر ها را باتوجه به اینکه active low است، صفر می دهیم. خروجی مالتی پلسکر را نیز به قسمت بعدی یعنی واحد محاسبات متصل می کنیم.



MUX ٨*١ Custom



MUX ٨ * ١

۳. واحد محاسبات(ALU)

Inputs: Register 0 value, 8-bit from MUX 8*1, Add/Sub

Outputs: 8-bit Result

Circuit parts: 4-bit Adder (7483), XOR gate

در این قسمت یک واحد محاسبات با دو قابلیت جمع و تفریق دو عدد ۸ بیتی طراحی می کنیم. سیگنال ورودی Add/Sub مشخص می کند که جمع باید انجام شود یا تفریق. در این بخش ورودی دوم(۸-bit from MUX) را با بیت Add/Sub، XOR می کنیم. سپس حاصل گیت های xor را با مقدار رجیستر صفر که ورودی ثابت این واحد است جمع می کنیم. برای جمع و تفریق ، با توجه به اینکه دو عدد ۸ بیتی داریم، از دو عدد جمع کننده ی چهار بیتی استفاده می کنیم و carry خروجی جمع کننده ی اول را به carry ورودی جمع کننده ی دوم می دهیم. بنابراین، فرمول واحد محاسبات به صورت زیر است: (در اینجا C_in بیانگر carry ورودی جمع کننده ی اول است)

$$Out = A + (B\ xor\ Add\backslash Sub) + C_{in},\ C_{in} = Add\backslash Sub$$

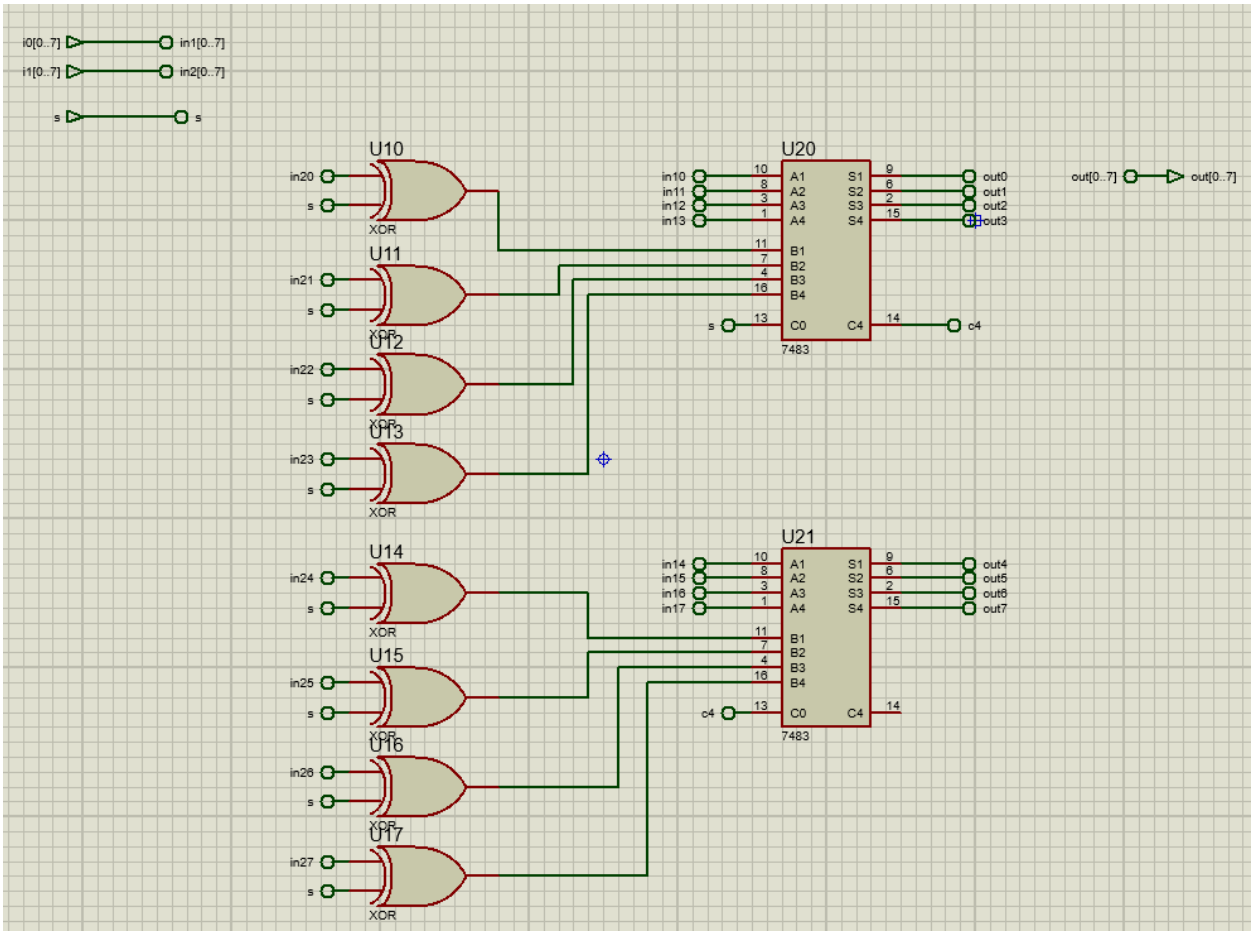
در اینجا دو حالت وجود دارد:

۱. اگر Add/Sub صفر باشد: پس باید دو ورودی را با یکدیگر جمع کنیم. فرض کنیم که A مقدار داخل رجیستر صفر و B مقدار حاصل از MUX 8*1 باشد. بنابراین داریم:

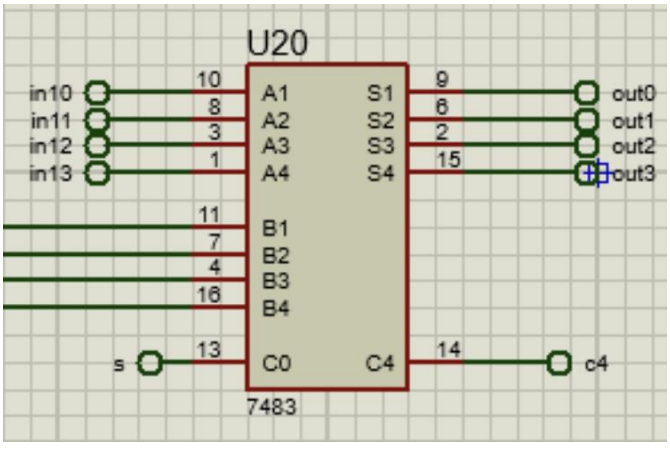
$$Add\backslash Sub = ۰ \rightarrow, B\ xor\ ۰ = B \rightarrow Out = A + B + ۰ = A + B$$

۲. اگر Add/Sub یک باشد: طبق دستور کار باید عملیات تفریق را انجام دهیم. فرض کنیم A مقدار داخل رجیستر صفر و B مقدار حاصل از MUX 8*1 باشد. برای تفریق از مکمل دوم عدد B استفاده می کنیم: A – B = A + not(B) + 1. بنابراین داریم:

$$Add\backslash Sub = ۱, B\ xor\ ۱ = not(B)$$
$$\rightarrow Out = A + not(B) + ۱ = A - B$$



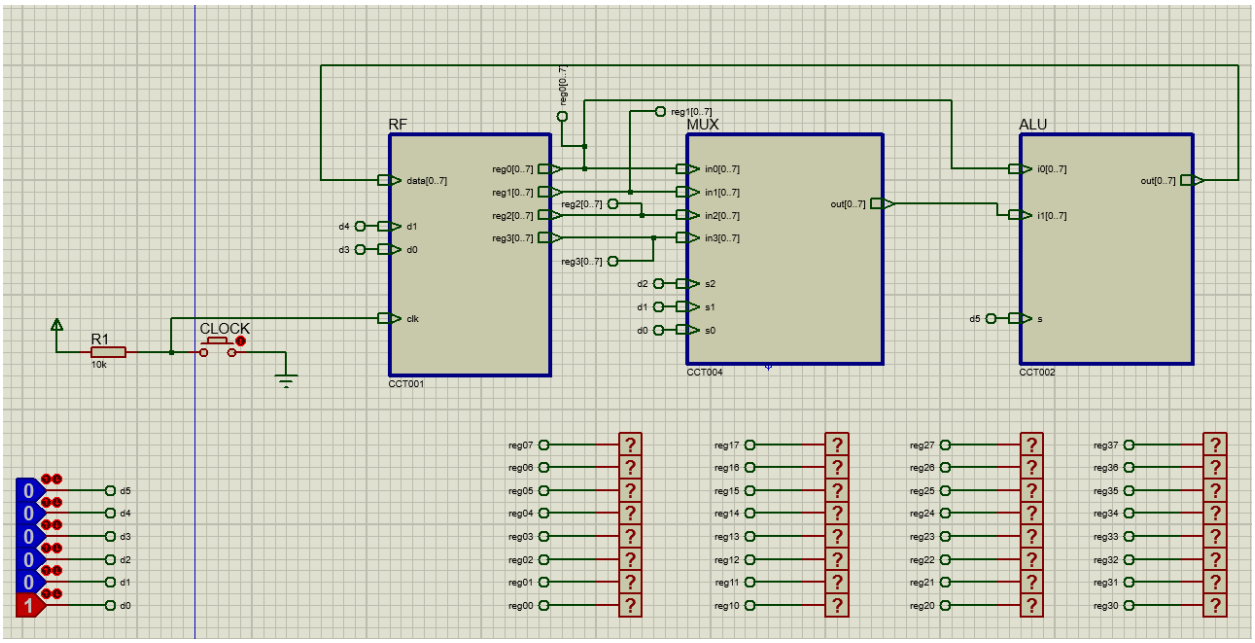
ALU



۴-bit Adder

۱. نحوه ی اتصال:

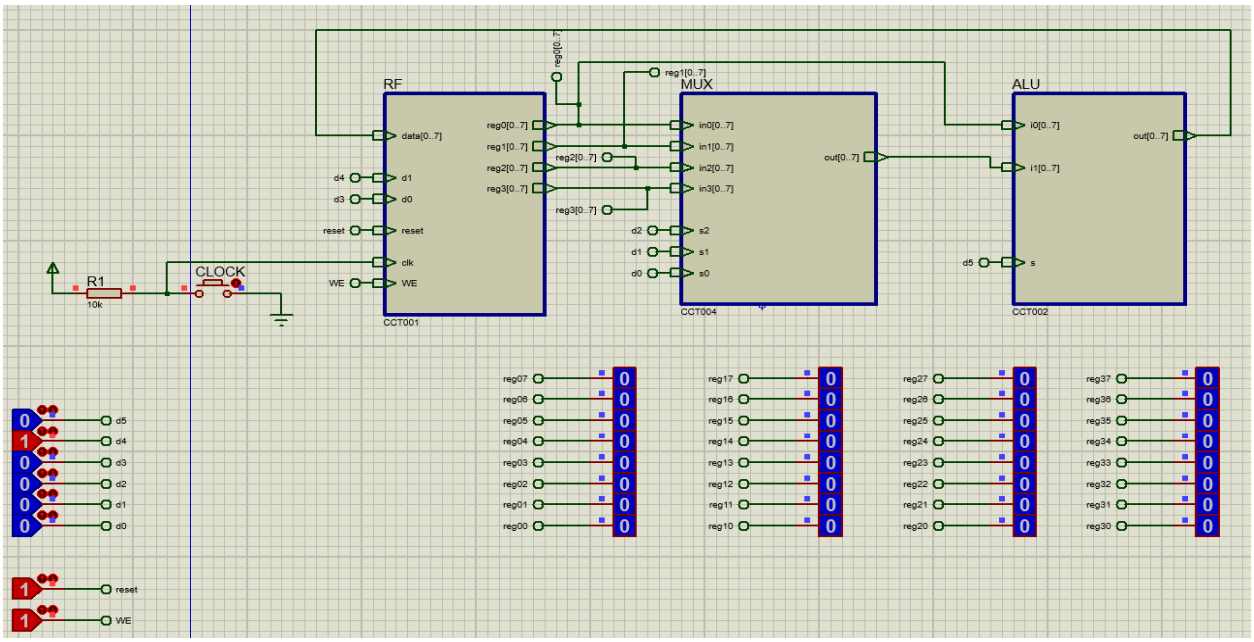
نحوه ی اتصالات و شکل کلی مدار را می توانید در تصویر زیر مشاهده کنید:



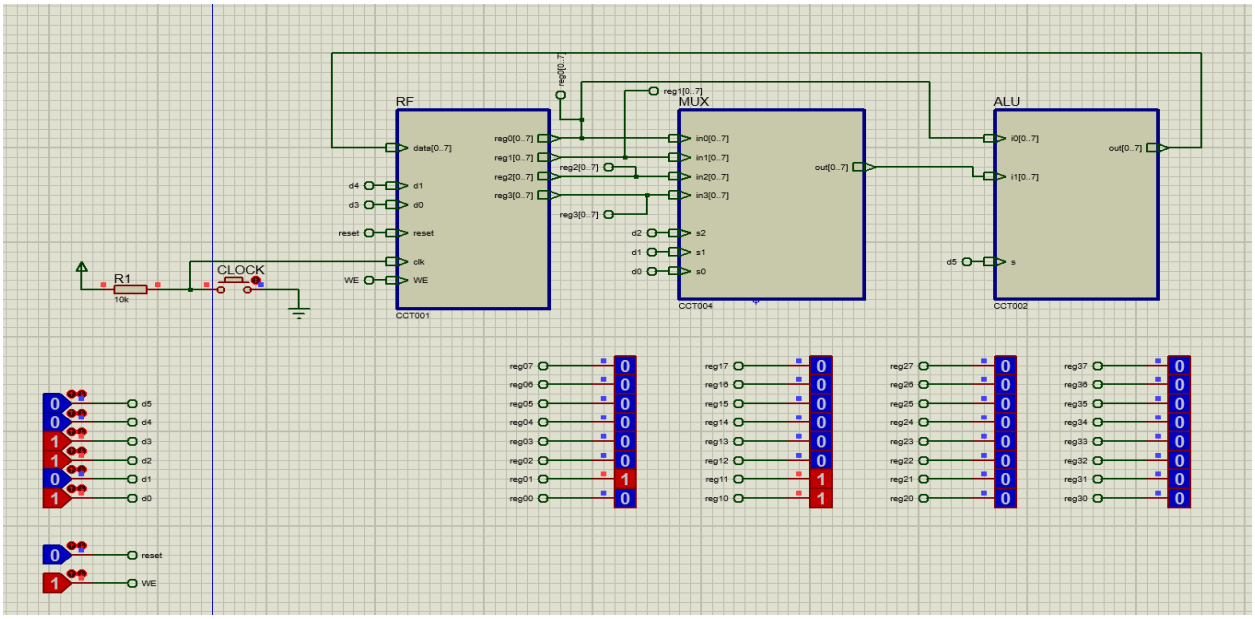
۲. تست مدار:

برای بررسی صحت عملکرد مدار، مجموعه ی دستورات زیر را اجرا می کنیم:

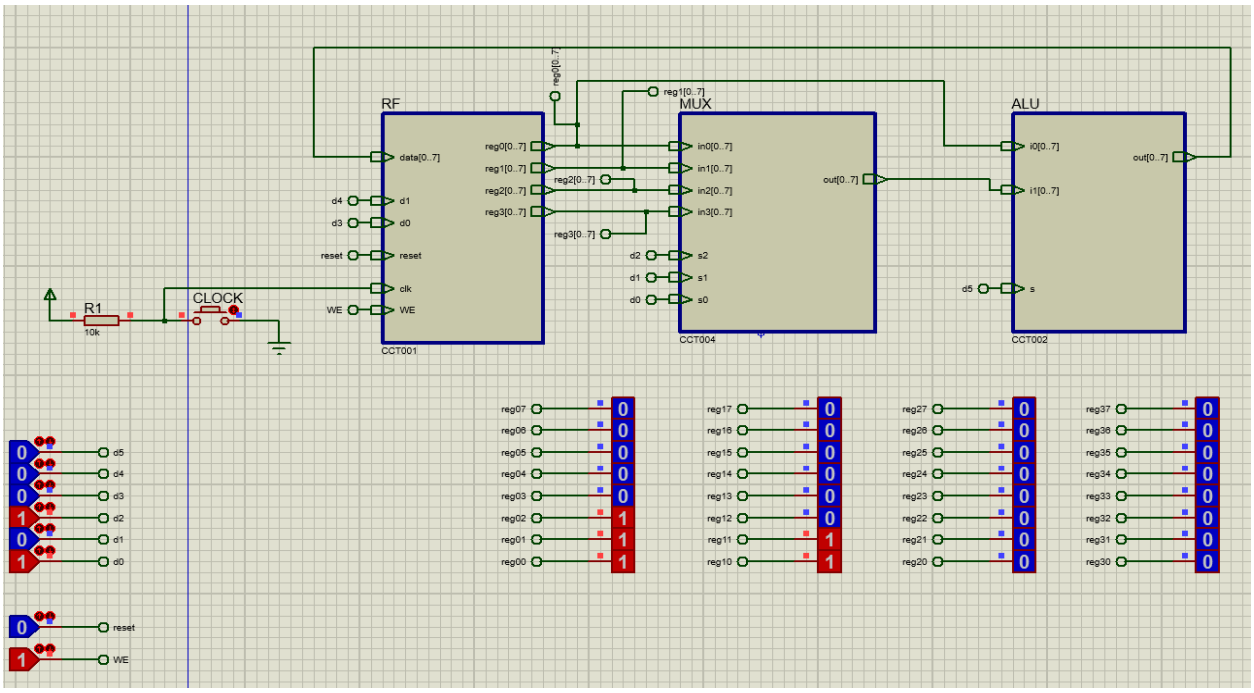
ابتدا با ریست کردن مقدار تمام رجیسترها را برابر ۰ می کنیم.



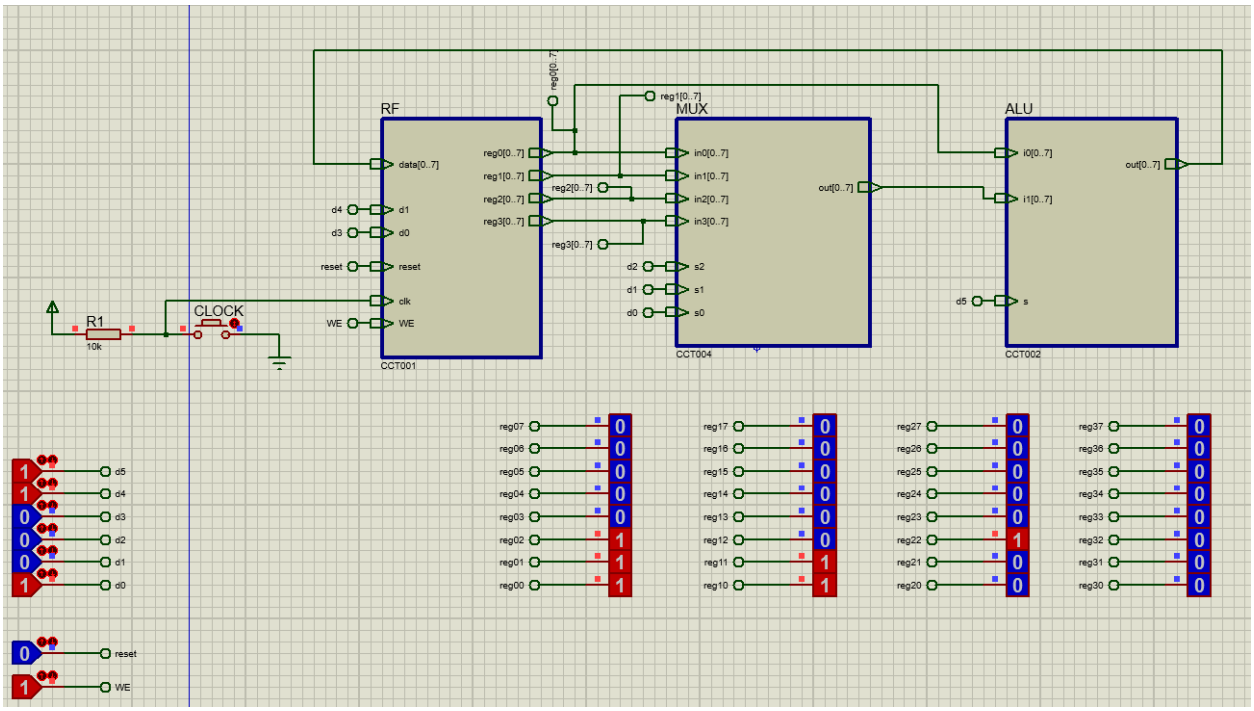
سپس با دستور R0 = R0 + 1 مقدار R0 را ۲ کرده و با دستور R1 = R0 + 1 در R1 مقدار ۳ را می ریزیم.



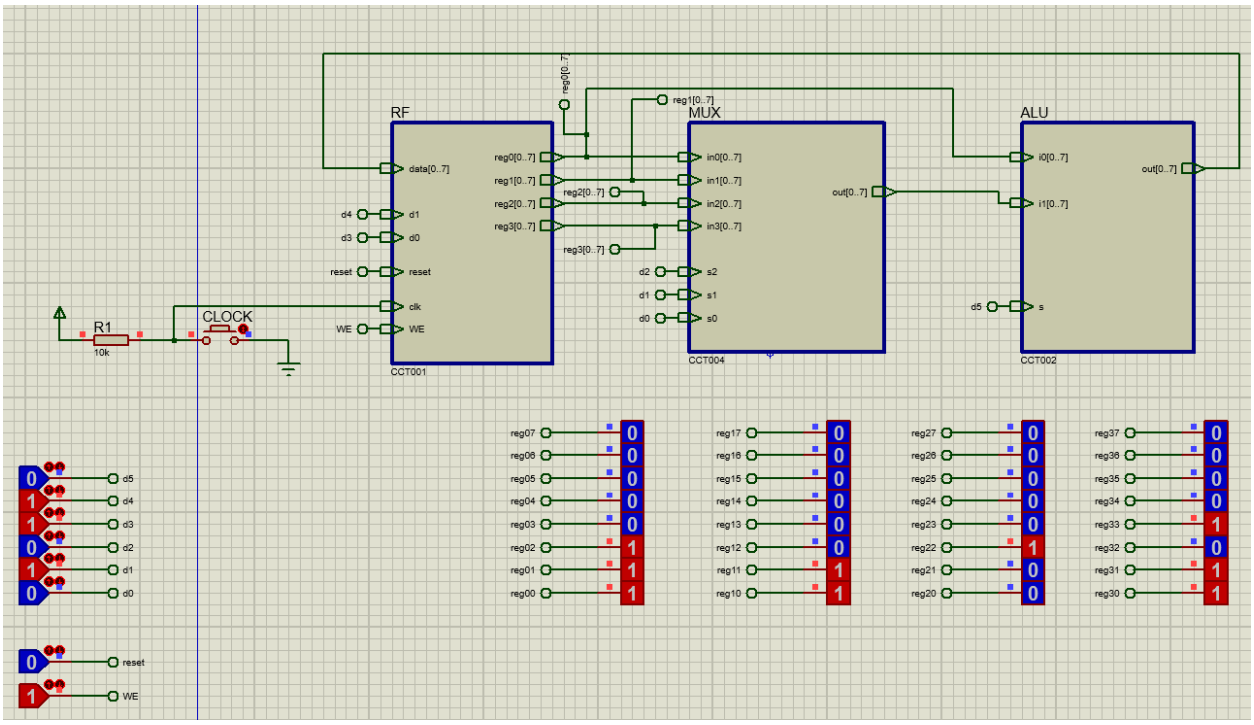
سپس با اجرای همان دستور قبلی مقدار R0 را به ۷ می رسانیم.



دستور R2=R0-R1 را اجرا می کنیم تا مقدار ۴ در رجیستر R2 ریخته شود.



در نهایت R3=R0+R2 را اجرا می کنیم تا مقدار ۱۱ در R3 ذخیره شود که نتیجه ی مطلوب و مورد و انتظار است.



پیاده سازی روی بردبورد :

برای پیاده سازی روی بورد، مدار را با ۲ رجیستر ۴ بیتی ساختیم.

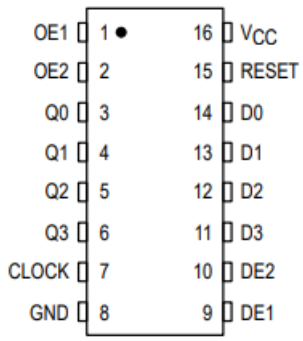
ابتدا بخش های اصلی مدار، یعنی رجیستر فایل شامل ۲ رجیستر ۴ بیتی، ۴ عدد مالتی پلکسر ۸ تایی و یک جمع / تفریق کننده (شامل ۴ گیت xor و یک جمع کننده ۴بیتی)، را جداگانه ساخته و تست کردیم. سپس اتصالات قسمت های مختلف را به ترتیب محاسبه در هر چرخه ی ساعت انجام دادیم.

درواقع ورودی های مالتی پلکسرها که از از روی خروجی های رجیسترها به دست می آیند را مشخص می کنیم، سپس خروجی مالتی پلکسرها را با ورودی جمع / تفریق xor می کنیم و سپس R0 را با خروجی xor ها جمع می کنیم. تمام قطعات و اتصالات تا این مرحله را نیز چک کردیم.

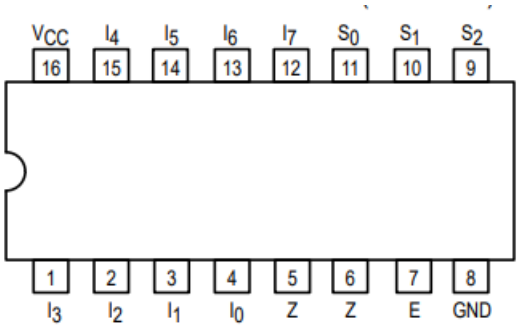
در نهایت خروجی جمع کننده را به ورودی رجیستر متصل می کنیم، اما در بعضی مواقع مقدار DFF ها مطلوب نبود. این اتفاق احتمالا به علت مشکل در اتصالات بردها و سیم ها بود که به علت کمبود زمان موفق به حل مشکل آن نشدیم.

برای رجیستر، مالتی پلکسر ، xor و جمع کننده به ترتیب از ۷۴۱۷۳، ۷۴۱۵۱، ۷۴۸۶ و ۷۴۸۳ استفاده کردیم که دیتاشیت آنها را در ادامه مشاهده میکنید:

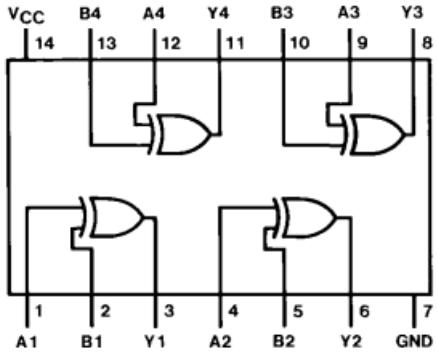
PIN ASSIGNMENT



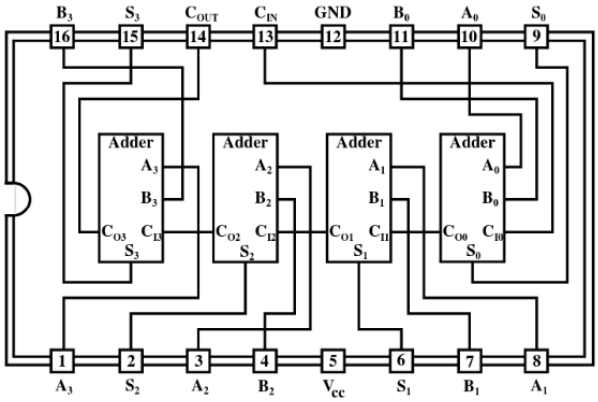
۷۴۱۷۳



۷۴۱۵۱



۷۴۸۶



7483: 4-BIT BINARY FULL ADDER W/FAST CARRY

۷۴۸۳