به نام خدا

دانشگاه صنعتی شریف

دانشکده مهندسی کامپیوتر

آزمایشگاه معماری کامپیوتر

آزمایش هفتم:

استفاده از حافظه داده و دستورات پرش

اطلاعات تیم	
نام اعضا	شماره دانشجویی
متین داغیانی	۹۸۱۰۶۴۵۶
بردیا محمدی	۹۸۱۷۱۱۰۴
محمدجواد هزاره	۹۸۱۰۱۰۷۴

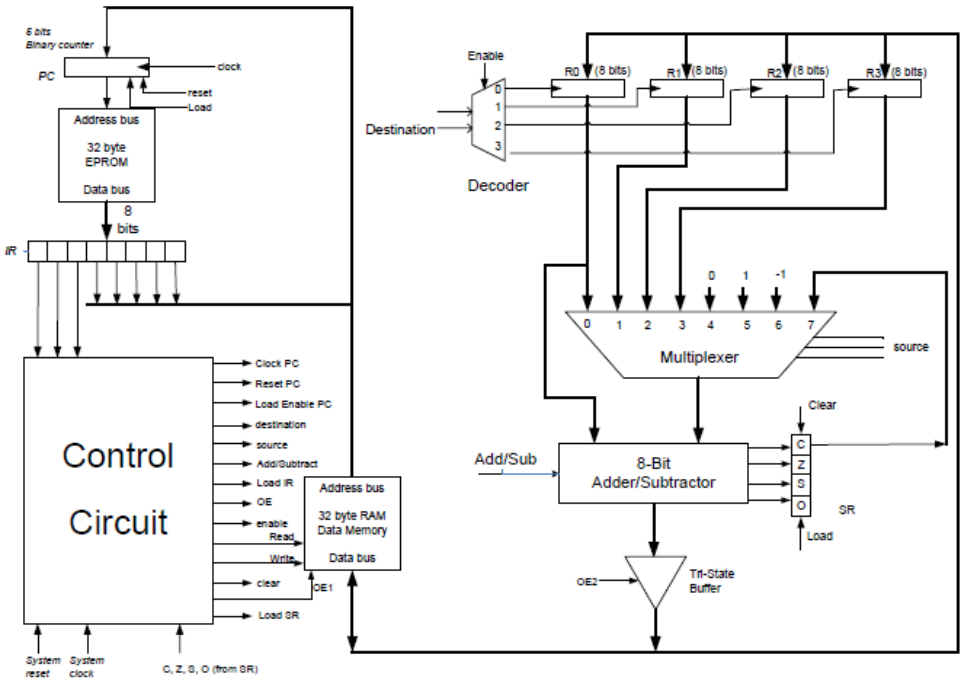
زمستان ۱۴۰۰

فهرست مطالب

فهرست تصاویر

۱ هدف آزمایش

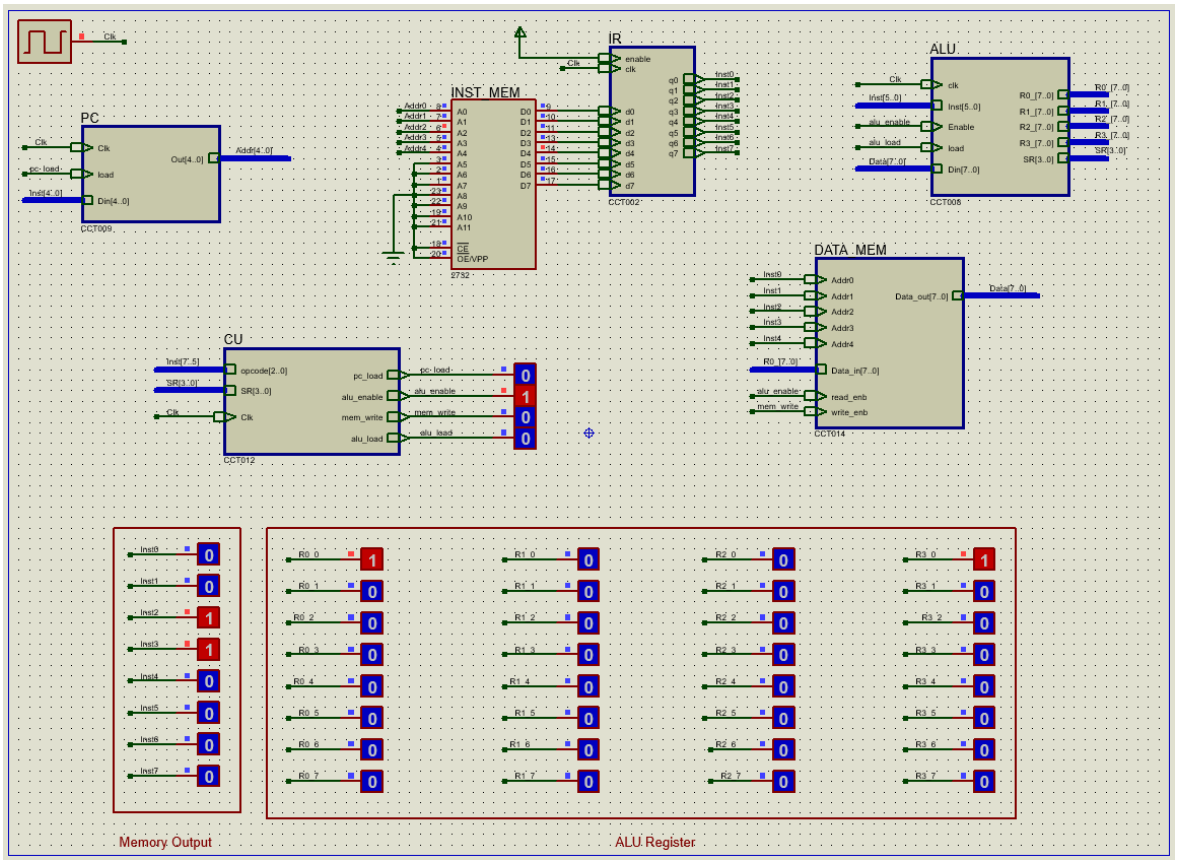
در این آزمایش قصد داریم تا با اضافه کردن امکانات پرش و خواندن داده از حافظه، مدار طراحی شده در آزمایش شماره شش را تکمیل کرده و در نهایت به یک کامپیوتر ساده با معماری Harvardدست یابیم. به همین جهت در گام اول یک RAM با گنجایش ۳۲ بایت را به مدار اضافه می کنیم و در نتیجه امکان ذخیره مقادیر بینابینی را در این حافظه پیدا خواهیم کرد. هم چنین در ادامه امکان استفاده از دستورات پرش شرطی و غیر شرطی را فراهم خواهیم کرد. در شکل ۱ دیاگرام بلوکی مدار نهایی را مشاهده می کنید:



شکل ۱: بلوک دیاگرام سیستم

همان طور که در شکل مشخص است، یک حافظه داده با ظرفیت ۳۲ بایت اضافه شده است که امکان ذخیره و بازیابی داده‌ها را فراهم می‌کند. این حافظه از طریق سیگنال‌های کنترلی Read و Write با واحد کنترل در ارتباط است. هم‌چنین در قسمت جریان داده، رجیسترهای flag اضافه شده‌اند که برای دستورات پرش شرطی مورد ارزیابی قرار می‌گیرند.

نمای کلی مدار پیاده‌سازی شده در شکل ۲ آمده است.



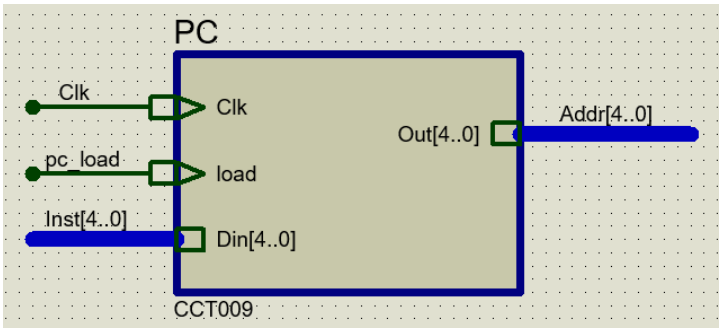
شکل ۲: نمای کلی سیستم پیاده سازی شده

۲ مراحل طراحی و پیاده‌سازی مدار

۱.۲ ماژول‌های مورد نیاز و شروع به کار مدار

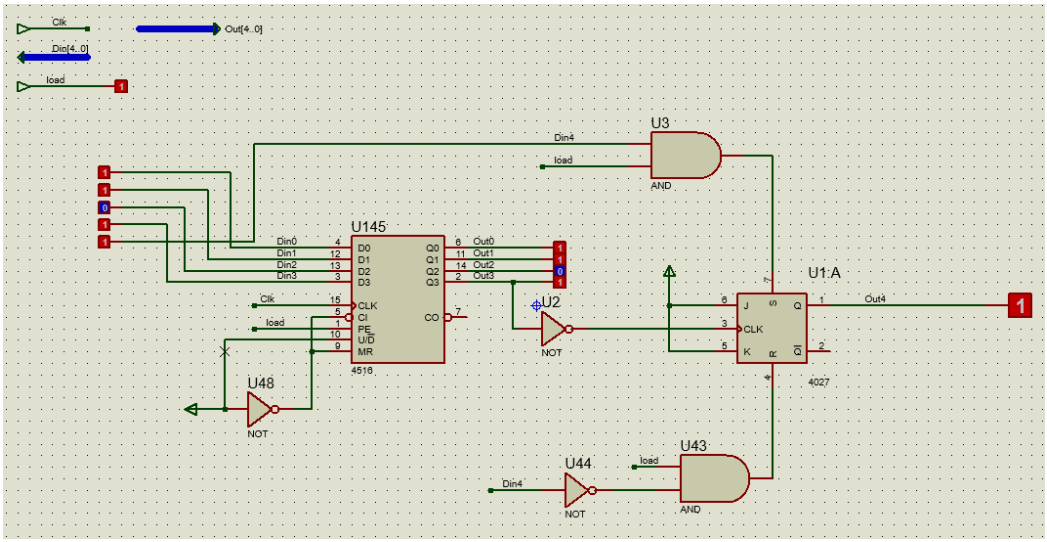
همانطور که در شکل ۱ مشخص است، برای کنترل سیگنال‌های کنترلی به واحد CU نیاز داریم. به علاوه با توجه به اضافه شدن دستورات پرش شرطی و غیر شرطی ماژول PC نسبت به آزمایش قبلی به روز شده است. هم چنین ماژول دیگری به نام DATA_MEM نیز برای ذخیره و بازیابی داده‌ها در نظر گرفته شده است. در ادامه به بررسی دقیق تر هر ماژول خواهیم پرداخت.

۲.۲ ماژول PC



شکل ۳: ماژول PC

برای پیاده سازی این ماژول از یک شمارنده بالا/پایین دودویی استفاده شده است که امکان لود موازی را نیز داراست (تراشه شماره ۴۵۱۶). ورودی‌های آن سیگنال pc_load ، Inst[4..0] و Clk هستند که در شکل مشخص است. طراحی داخلی این ماژول را در شکل ۴ می‌توان دید.

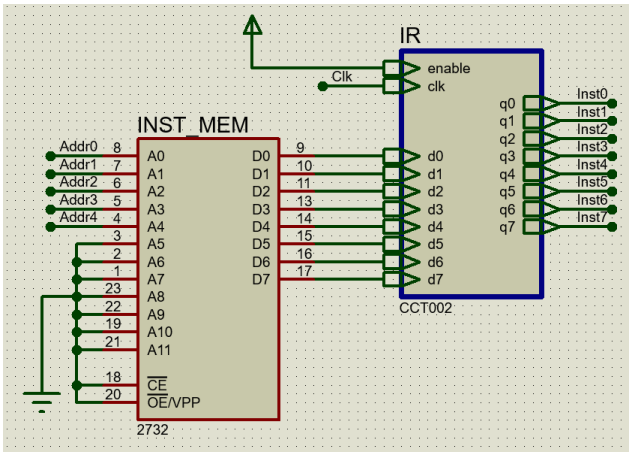


شکل ۴: طراحی داخلی ماژول PC

همانطور که در شکل دیده می‌شود، برای طراحی این مدار از یک شمارنده‌ی ۴بیتی (تراشه 4516) استفاده شده است. این تراشه قابلیت بارگذاری موازی را نیز داراست که به چهار ورودی آن، چهار بیت کم‌ارزش ورودی ماژول داده شده است. برای شمارش بیت پنج، همانطور که در شکل دیده می‌شود با تغیر بیت چهارم خروجی، بیت پنج کلاک خورده و تغییر می‌کند. برای ست کردن این بیت نیز مطابق شکل از ورودی‌های set و reset فلیپ‌فلاپ استفاده شده است. با 1 شدن سیگنال ورودی load، ورودی در خروجی بارگذاری شده و شمارش از آن عدد به بالا ادامه پیدا خواهد کرد.

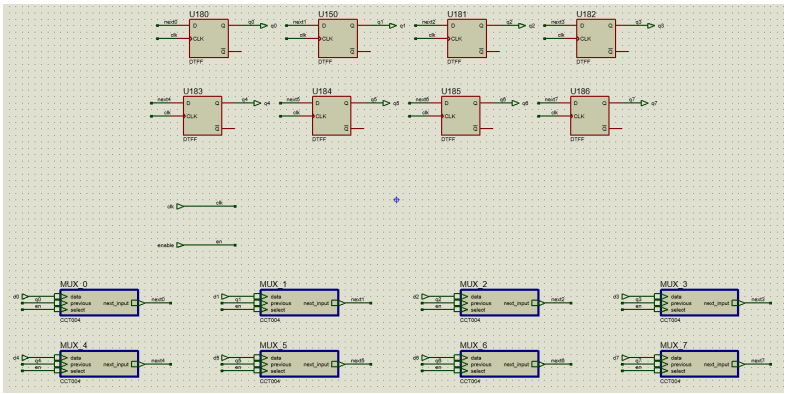
۳.۲ ماژول INST_MEM و IR

ماژول INST_MEM نسبت به آزمایش قبلی تغییری نکرده است. اما ماژول IR در این آزمایش به مدار اضافه شده که همان رجیستر شامل دستورالعمل است.



شکل ۵: ماژول INST_MEM به همراه IR

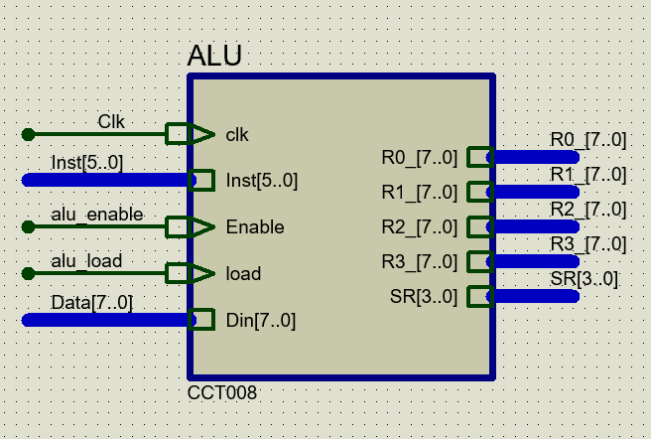
مدار داخلی این رجیستر عینا مشابه همان مدار داخلی رجیسترهایی است که در آزمایش پنجم برای رجیسترهای داخلی این ماشین از آن استفاده کرده بودیم که در شکل ۶ آمده است.



شکل ۶: طراحی داخلی ماژول IR

۴.۲ ماژول ALU

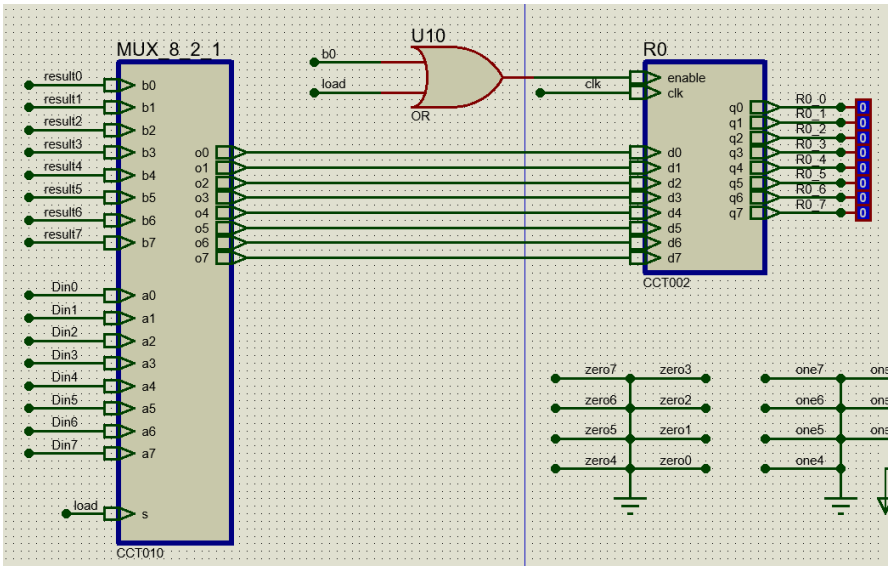
بخش زیادی از این ماژول بدون تغییر از آزمایش‌های قبلی است. اما تعدادی ورودی و خروجی به آن اضافه شده است.



شکل ۷: ماژول ALU

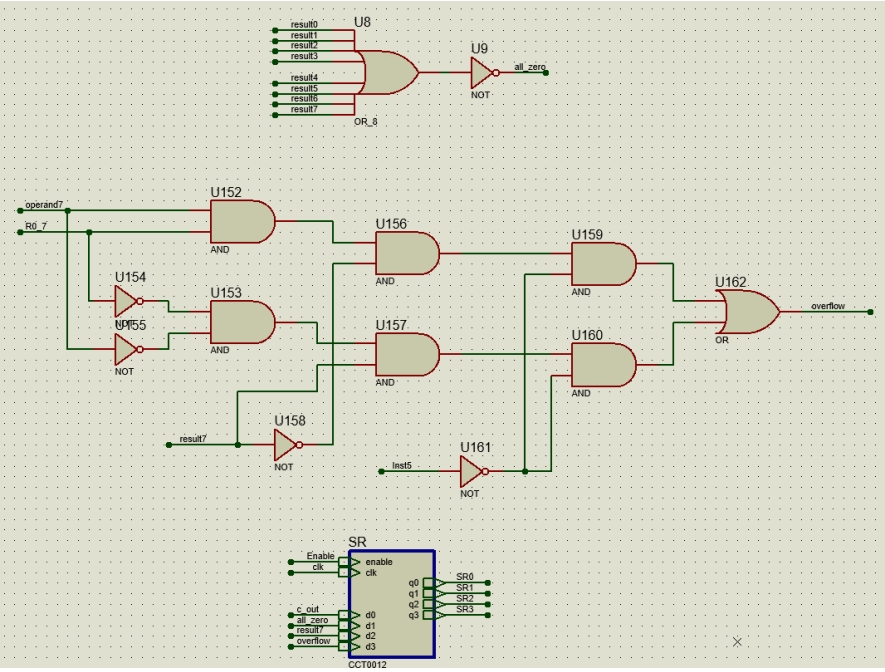
همانطور که در شکل ۷ دیده می‌شود، ورودی‌های Enable ، load و Din به ماژول اضافه شده‌اند که به ترتیب امکان خاموش کردن ماژول، بارگذاری در آن و داده‌ی ورودی به آن را به ما می‌دهند. با یک شدن سیگنال load مقادیر Din در رجیستر R_0 ذخیره خواهد شد. همچنان در دستورالعمل‌هایی که به واحد محاسبه نیازی نداریم، ورودی Enable این ماژول 0 خواهد شد.

تغییرات طراحی داخلی این ماژول نخست شامل ورودی رجیستر R_0 است که به صورت زیر خواهد بود.



شکل ۸: تغییرات طراحی داخلی ALU

همانطور که دیده می‌شود، ورودی این ماژول براساس سیگنال load انتخاب می‌شود. هم‌چنین برای بررسی پرش‌های شرطی، به تعدادی flag نیاز داشتیم که این flagها براساس نتیجه‌ی عملیات واحد محاسبه مشخص می‌شوند. برای افزودن این امکان به سیستم، مدار زیر به ماژول اضافه شده است.

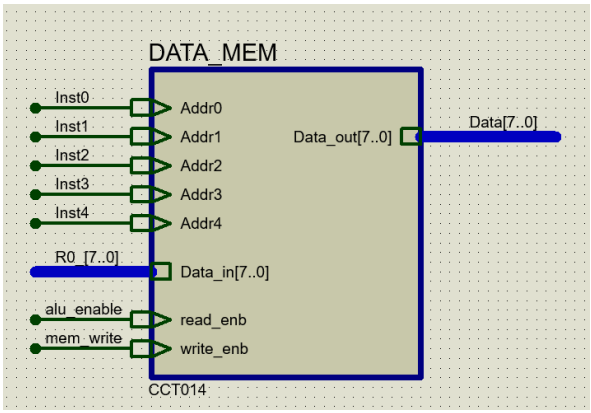


شکل ۹: طراحی مدار برای SR

برای بررسی صفر بودن کافیست مشابه تصویر OR تمام بیت‌های حاصل صفر باشد. برای بررسی carry که از خروجی carry جمع‌کننده استفاده شده است. برای بررسی overflow نیز از این واقعیت استفاده شده که اگر دو عدد مثبت را با یکدیگر جمع زده و حاصل منفی باشد، آن‌گاه overflow رخ داده است. هم‌چنین با مثبت شدن جمع یا تفرق تو عدد منفی با یکدیگر نیز overflow رخ داده است. از این توصیف همناطور که در تصویر دیده می‌شود برای تولید سیگنال overflow استفاده شده است. ماژول SR نیز یک رجیستر ۴بیتی است که این flagها را در خود نگه می‌دارد.

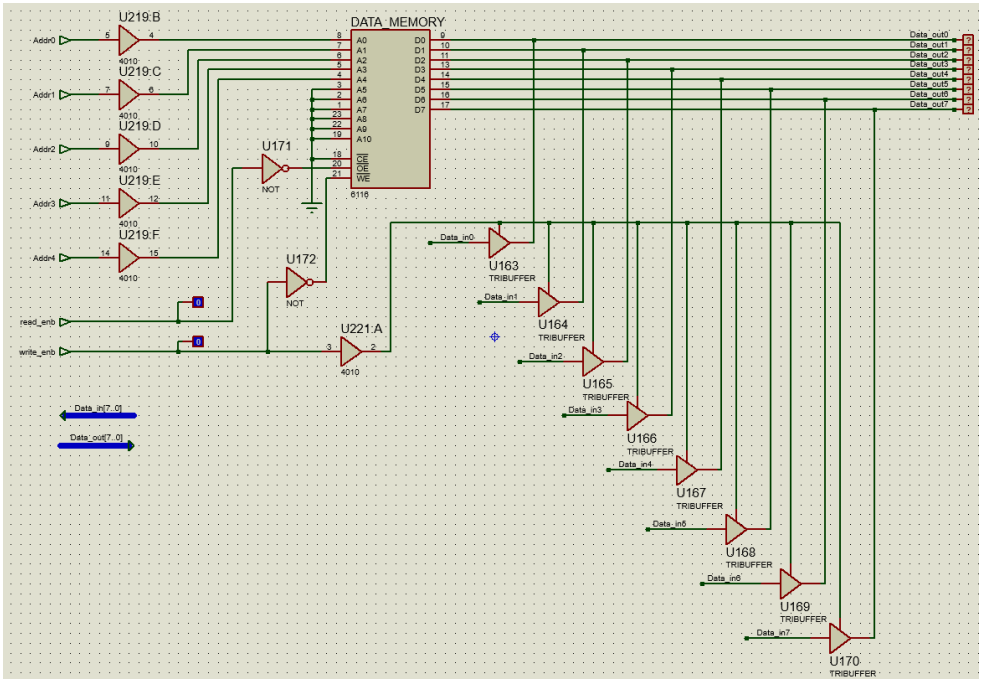
۵.۲ ماژول DATA_MEM

این ماژول به مدار اضافه شده تا بتوان مقادیر بینابینی را در حافظه‌ی سیستم ذخیره کرد و یا از آن این مقادیر را خواند. ورودی‌ها و خروجی‌های آن در شکل ۱۰ آمده است.



شکل ۱۰: ماژول DATA_MEM

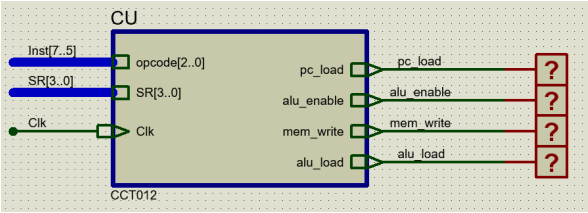
در طراحی داخلی این ماژول از تراشه‌ی 6116 استفاده شده که یک RAM شامل ۳۲ بایت می‌شود. این تراشه هم قابلیت خواندن و هم قابلیت نوشتن را داراست. برای این منظور با استفاده از ترای استیت بافر داده‌ی ورودی را به تراشه داده یا داده‌ی خروجی آن را می‌خوانیم. طراحی مذکور در شکل ۱۱ آمده است.



شکل ۱۱: طراحی داخلی ماژول DATA_MEM

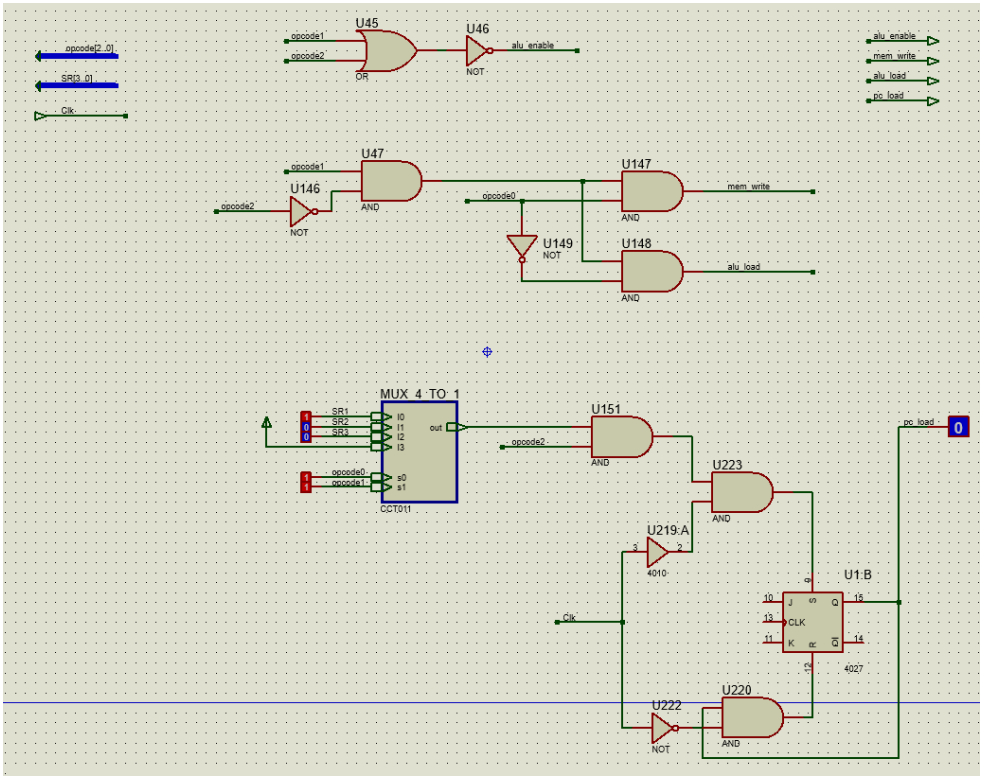
۶.۲ ماژول CU

این ماژول همانطور که از نامش پیداست، وظیفه‌ی کنترل مدار را بر عهده دارد. به طور کلی سیگنال‌های مربوط به فعال شدن واحد محاسبه، خواندن یا نوشتن روی حافظه را تولید می‌کند. ورودی‌ها و خروجی‌های آن در شکل ۱۲ آمده است.



شکل ۱۲: ماژول CU

برای ساخت سیگنال pc_load از یک مالتی‌پلکسر ۴ به ۱ استفاده شده که براساس دستور ورودی و محتویات رجیستر SR، تصمیم می‌گیرد که سیگنال مورد نظر باید ۱ باشد یا ۰. فلیپ‌فلاپ استفاده شده نیز به این منظور است که پس از یک شدن سیگنال pc_load، نهایتا در نیم‌کلاک بعد باید این سیگنال را خاموش کنیم تا در کلاک بعدی که دستورالعمل بعدی می‌آید، این دستورالعمل در PC بارگذاری نشود.

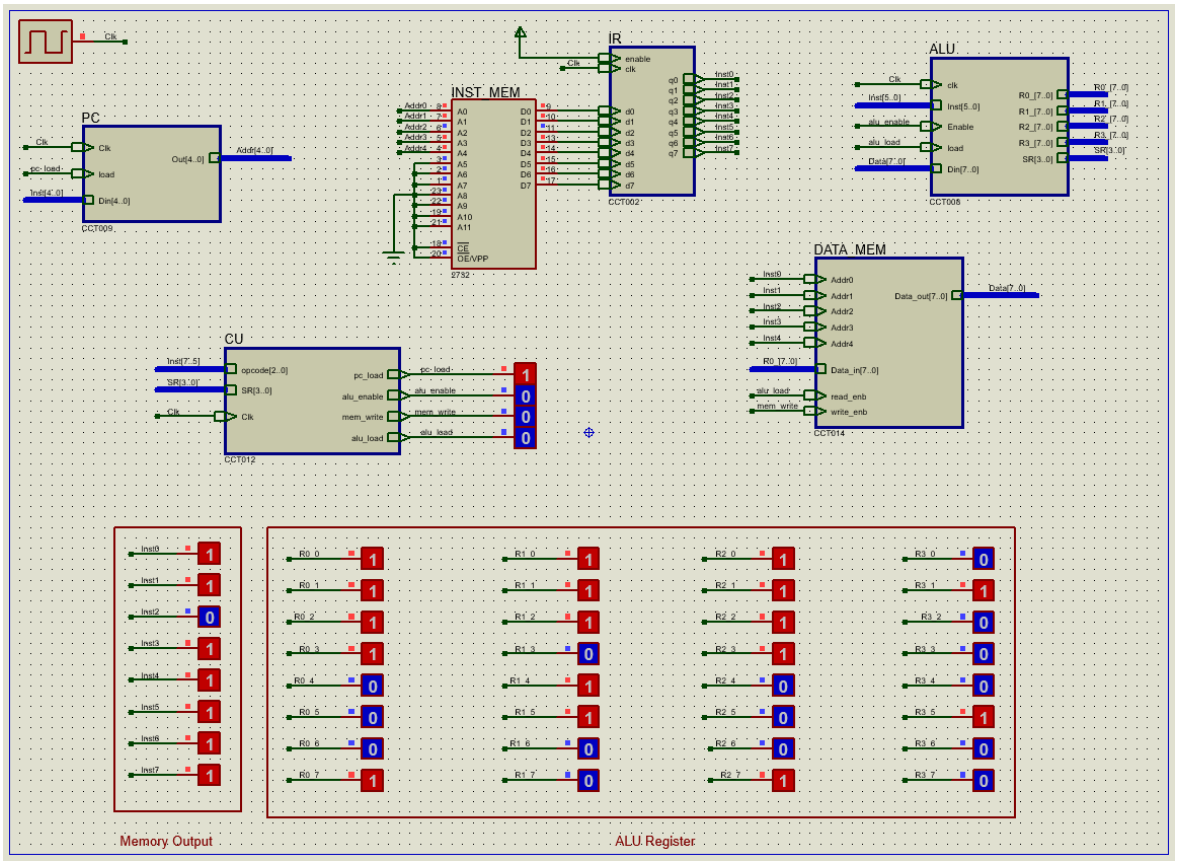


شکل ۱۳: طراحی داخلی ماژول CU

ساخت سیگنال‌های mem_write ، alu_load و alu_enable نیز به راحتی از روی بیت‌های دستورالعمل انجام شده است.

۳ تست مدار

مطابق دستورالعمل، به منظور تست مدار مجموع ۱۰ جمله‌ی اول دنباله‌ی فیبوناچی محاسبه شده است. کد ماشین و کد دستوری این برنامه به ترتیب در فایل‌های fibo_machine.txt و fibo_code.txt همراه فایل گزارش آمده است. خروجی سیستم پس از اجرای کامل این برنامه را نیز در شکل می‌توان مشاهده کرد.



شکل ۱۴: وضعیت مدار پس از اجرای کامل برنامه‌ی fibo_machine.bin

در این برنامه رجیستر R_2 جمع مورد نظر را نگه‌داری می‌کند که در انتها برابر $(10001111)_2 = 2^7 + 8 + 4 + 2 + 1 = 143$ شده است که برابر جمع ۱۰ جمله‌ی اول دنباله فیبوناچی است.

همچنین در مورد دوم که جمع دو عدد ۶۴ بیتی خواسته شده، کد ماشین و کد دستوری آن را می‌توان به ترتیب در فایل‌های 64bit_sum_machine.bin و 64_bit_sum_code.txt یافت. برای این برنامه به این صورت عمل می‌کنیم که برای تک تک بایت‌ها، نخست آن‌ها را در رجیسترهای R_0 و R_1 بارگذاری کرده و این دو را با هم جمع می‌کنیم. حال اگر در جمع این دو عدد ۸ بیتی overflow رخ دهد، باید یک carry برای جمع هشت بیت بعدی لحاظ کنیم. کد دستوری برای انجام این امر مطابق تصویر زیر خواهد بود:

```
LOD 00001
ADD R1  R1
LOD 01001
ADD R0  R1
STO 10001

JO ?????
J  ?????

SUB R0  R0
ADD R1  +1
```

شکل ۱۵: کد دستوری جمع دو بایت از دو عدد ۶۴ بینی

دستور JO به خط یکی مانده به آخر پرش خواهد کرد و دستور J به بعد از آخرین خط یا همان اولین خط مربوط به جمع دو بایت بعدی پرش می‌کند. همانطور که دیده می‌شود، اگر جمع دو بایت در مرحله‌ی قبل carry داشته باشد، رجیستر R_1 با مقدار 1 پر خواهد شد و در نتیجه در جمع در خط دوم کد، مقدار $M[1] + 1$ در رجیستر R_0 ذخیره خواهد شد. و اگر در مرحله‌ی قبل overflow رخ نداده باشد، دستور J (پرش غیر شرطی) اجرا خواهد شد و دو خط آخر تصویر اجرا نمی‌شوند.

با هشت بار تکرار این‌کد و مشخص کردن مناسب آدرس جلوی LOD و STO و دستورهای پرش، به خواسته‌ی خود می‌رسیم. کد کامل را در فایل ذکر شده در بالاتر می‌توان یافت.