به نام خدا

آزمایشگاه معماری کامپیوتر

گزارش کار هشتم

واحد کنترل ریزبرنامه سازی شده

دانشکده مهندسی کامپیوتر

دانشگاه صنعتی شریف

پاییز ۱۴۰۱

استاد:

حمید سربازی آزاد

دستیار آموزشی:

عطیه یونسی

نویسندگان:

احمدرضا خناری

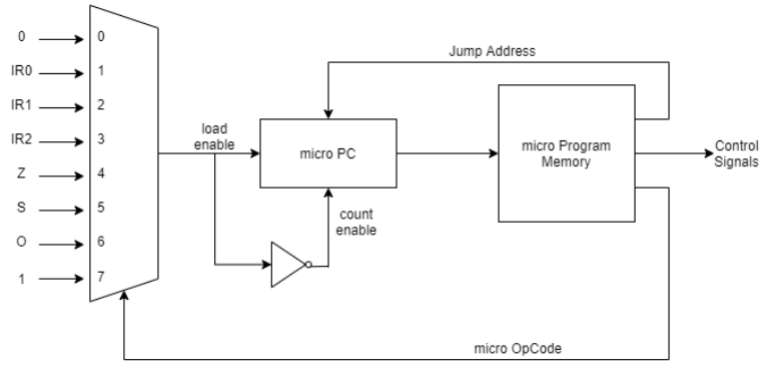
نگار عسکری

علی نظری

فهرست

مقدمه

در این آزمایش، واحد کنترلی آزمایش ۷ را با یک واحد کنترلی با قابلیت ریزبرنامه سازی شدن جایگزین می کنیم. به طور کلی برنامه ای که داخل micro program memory قرار می گیرد قابلیت دیکود و اجرای دستورات مختلف را دارد و خود این برنامه که با کمک ریز دستورات نوشته شده با جلو رفتن mircro PC اجرا می شود. شمای کلی طراحی واحد کنترلی در شکل ۱ آمده است. همانطور که در این شکل مشخص است، نحوه ی کارکرد مدار به این شکل است که با توجه به خانه ی فعلی micro program memory یا به مقدار فعلی micro PC اضافه می شود و یا به آدرس دیگری در micro program memory پرش می کند. برنامه ی نوشته شده در micro program memory به گونه ای است که قابلیت دیکود و اجرای هر کدام از انواع دستورات را دارد.



شکل ۱: شمای کلی

گزارش آزمایش

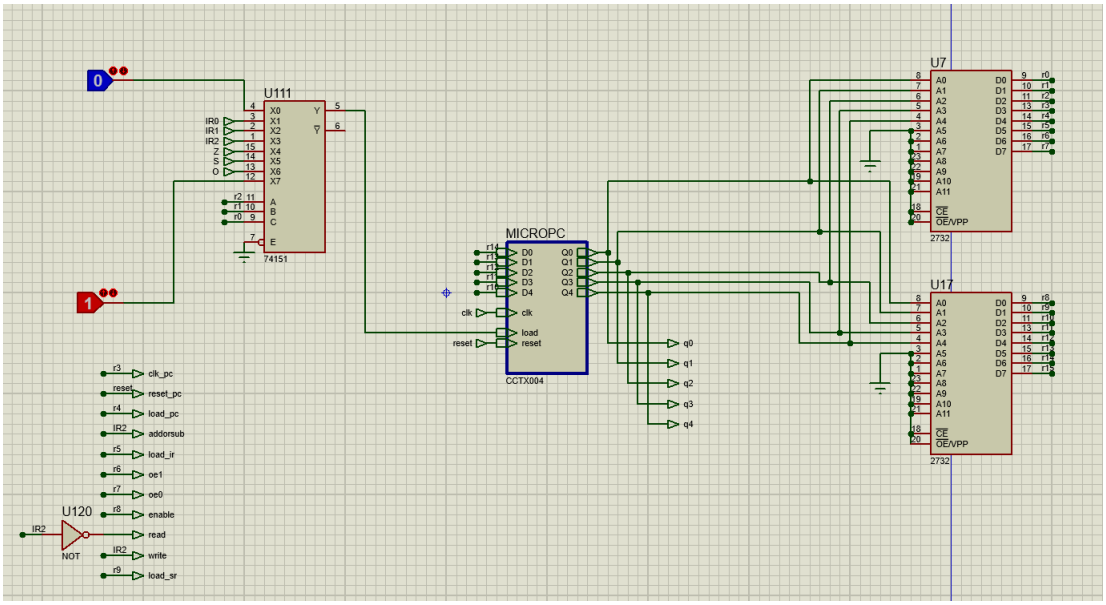
بخش ۱. برنامه ریزی و پیاده سازی micro program memory

با توجه به نیازمندی های ما در این آزمایش، برای هر خانه از micro program memory باید ۱۰ بیت کنترلی در نظر می گیریم. علاوه بر این ۱۰ بیت، به ۵ بیت برای مشخص کردن jump address نیاز داریم. بنابراین برنامه ای که برای اجرای دستورات داخل micro program memory می نویسیم شامل ۱۶ ریزدستور بوده و مطابق شکل ۲ است. ترتیب سیگنال های کنترلی در خط اول مشخص شده و در حالاتی که Jump داریم تنها در صورتی پرش اتفاق می افتد که بسته به micro Pp-code شرط پرش برقرار باشد یا پرش به صورت unconditional باشد. دقت کنید در نام گذاری ها در مدل چپ ترین بیت که همان Op۲ است با r۰ و راست ترین بیت که Jump۰ است با r۱۴ مشخص شده است.

برای پیاده سازی micro program memory در نرم افزار Proteus از دو قطعه ی ۲۷۳۲ استفاده می کنیم که هر کدام هایROM 4KB با خانه های ۱ بایتی هستند که با موازی کنار هم قرار گرفتن ۴۰۹۶ خانه ی ۲ بایتی ایجاد می کنند که مورد نیاز ماست. دقت کنید هرچند در عمل با استفاده از این ROM ها می توان تا ۴۰۹۶ ریزدستور داخل micro program memory داشت ما تنها از ۱۶ خانه ی اول استفاده می کنیم که مطابق نیازمندی حداکثر ۲۵۶ ریزدستور گزارش کار است. فایل های باینری part1-memory-microinst.bin و part2-memory-microinst.bin که در پیوست آمده اند به ترتیب مربوط به ۸ بیت اول و دوم micro program memory هستند که دو قطعه ی ۲۷۳۲ با این فایل ها بارگذاری می شوند. همچنین دستورات micro program memory به شکل متنی در فایل microinst-memory.txt در پیوست آمده اند. نحوه پیاده سازی واحد کنترلی به شکل ریز برنامه پذیر در شکل ۳ آمده است.

```
         -Op2,Op1,Op0,Clock PC,Load PC,Load IR,OE1,OE0|Enable,Load SR (Jmp4..0)
Begin:  0-0,0,0,1,0,1,0,0|0,0   (Jump irrelevant -      00000)
Decode: 1-0,0,1,0,0,0,0,0|0,0   (Jump to Branch -       00111)
Decode: 2-0,1,0,0,0,0,0,0|0,0   (Jump to Load/St -      00100)
Add/Sub:3-1,1,1,1,0,1,1,0|1,1   (Jump to Decode -       00001)
Load/St:4-0,1,1,0,0,0,0,0|0,0   (Jump to Store -        00110)
Load:   5-1,1,1,1,0,1,0,1|1,0   (Jump to Decode -       00001)
Store:  6-1,1,1,1,0,1,0,1|0,0   (Jump to Decode -       00001)
Branch: 7-0,1,0,0,0,0,0,0|0,0   (Jump to O/Unc -        01101)
Branch: 8-0,1,1,0,0,0,0,0|0,0   (Jump to S=1 -          01011)
Z=1:    9-1,0,0,0,0,0,0,0|0,0   (Jump to Unc -          10000)
Z!=1:  10-1,1,1,1,0,1,0,0|0,0   (Jump to Decode -       00001)
S=1:   11-1,0,1,0,0,0,0,0|0,0   (Jump to Unc -          10000)
S!=1:  12-1,1,1,1,0,1,0,0|0,0   (Jump to Decode         00001)
O/Unc: 13-0,1,1,0,0,0,0,0|0,0   (Jump to Unc -          10000)
O=1:   14-1,1,0,0,0,0,0,0|0,0   (Jump to Unc -          10000)
O!=1:  15-1,1,1,1,0,1,0,0|0,0   (Jump to Decode -       00001)
Unc:   16-1,1,1,0,1,0,0,0|0,0   (Jump to Begin -        00000)
```

شکل ۲: برنامه نوشته شده در micro program memory



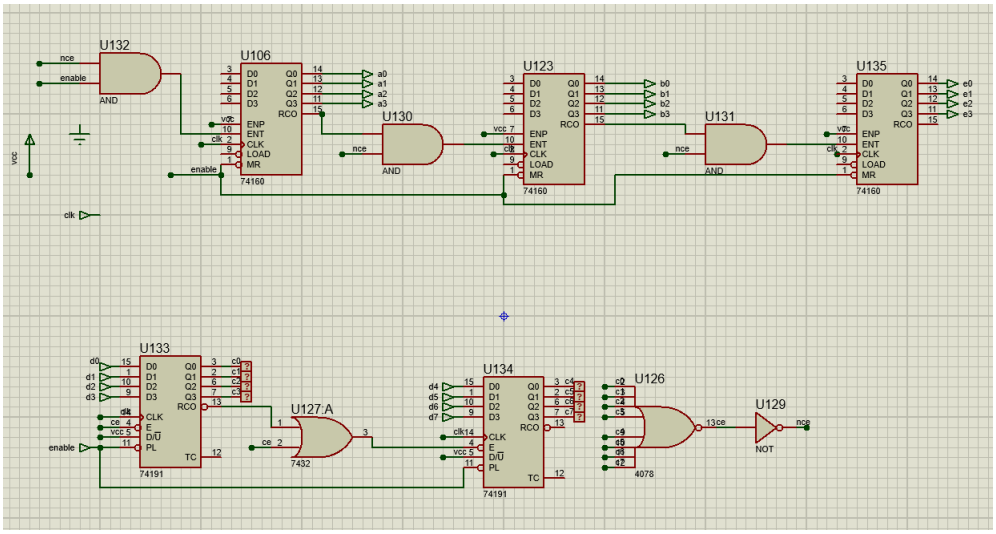
شکل ۳: پیاده سازی واحد کنترل به شکل ریز برنامه پذیر

بخش ۲. بررسی صحت عملکرد مدار

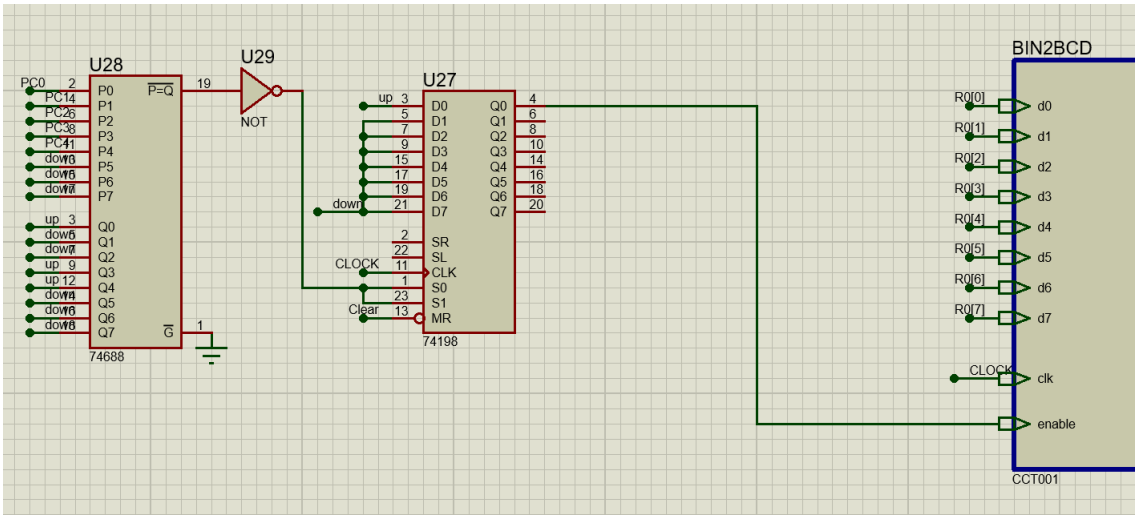
جهت بررسی صحت عملکرد مدار، برنامه هایی می نویسیم که یکی از آن ها جمع ۱۰ جمله و دیگری جمع ۵ جمله ی اول فیبوناچی را محاسبه می کنند. همچنین پس از اتمام محاسبه، علاوه بر این که نتیجه در رجیستر R۰ نمایش داده می شود، مدار BIN۲BCD نیز خروجی باینری را با کمک دو شمارنده تبدیل به BCD کرده و پس از اتمام تبدیل، نتیجه بر روی سه seven segment نمایش داده می شود. در کدی که در فایل fib.txt نوشته شده قابلیت محاسبه جمع ۱۰ یا ۵ جمله ی اول فیبوناچی وجود دارد. در صورتی که خط mem(05)=00-0-00-000 انتخاب شود دستور پنجم دستور r0 = r0 + r0 خواهد بود و مقدار شمارنده ای که تعداد جملات فیبوناچی را مشخص می کند برابر با ۱۰ می شود و اگر دستور mem(05)=00-0-00-100 انتخاب شود دستور پنجم r0 = r0 + 0 بوده و جمع ۵ جمله ی اول محاسبه می شود. هر دوی این حالات به فایل باینری تبدیل شده اند و در صورتی که حافظه ی اصلی با فایل fib10-memory-prgoram.bin بارگذاری شود حاصل جمع ۱۰ جمله ی اول و در صورتی که با فایل fib5-memory-program.bin بارگذاری شود حاصل جمع ۵ جمله ی اول محاسبه خواهد شد.

نیازمندی تبدیل خروجی به BCD به طور مستقیم در دستور کار نیامده است و برای این کار می توانیم به دو طریق عمل کنیم. روش اول این است که برنامه را به گونه ای بنویسیم که خروجی مستقیما به شکل BCD در ثبات ها محاسبه شود. برنامه ی fib2.txt در پیوست بدین شکل عمل می کند و حاصل جمع ۵ جمله ی اول فیبوناچی را محاسبه کرده و مقدار یکان را در ثبات R۰ و دهگان را در ثبات R۱ می ریزد. از آنجایی که این برنامه که تا حد ممکن نیز بهینه شده دارای ۳۴ دستور است در حافظه ی دستور ما که با ۵ بیت آدرس پیاده سازی شده قرار نمی گیرد و تعویض آن ساختار تمامی دستورها و معماری پردازنده را تغییر خواهد داد. بنابراین به جای این روش از یک روش دوم استفاده می کنیم.

در روش دوم، از یک شمارنده ی Binary و یک شمارنده ی BCD استفاده می کنیم و با کمک شمارنده ی Binary از حاصلی که در ورودی داده شده است یکی یکی کم می کنیم و به کمک شمارنده ی BCD خروجی را یکی یکی زیاد می کنیم تا مقدار شمارنده ی Binary به صفر برسد. پیاده سازی این مدار در پروتئوس در شکل ۴ آمده است و حاصل یکان در بیت های a3..0 و حاصل دهگان در بیت های b3..0 و حاصل صدگان هم در بیت های e3..0 آمده است. همچنین ورودی این مبدل، d7..0 است و به رجیستر R۰ متصل می شود. برای دانستن زمان شروع تبدیل از این نکته استفاده می کنیم که هنگامی که مقدار PC به ۲۵ برسد یعنی محاسبه تمام شده و در این صورت enable باید فعال شود که برای پیاده سازی این مدار ابتدا عدد ۲۵ را با PC مقایسه می کنیم و در صورت برابر بودن رجیستری را یک می کنیم که مقدار enable را نگه می دارد. این پیاده سازی در شکل ۵ آمده است.

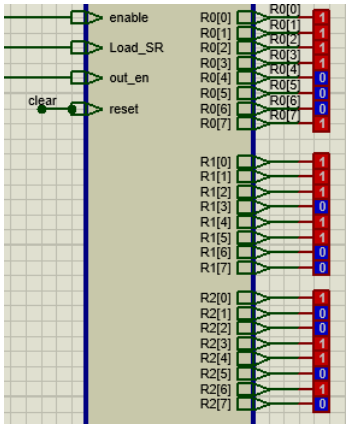


شکل ۴: مدار تبدیل کننده باینری به BCD

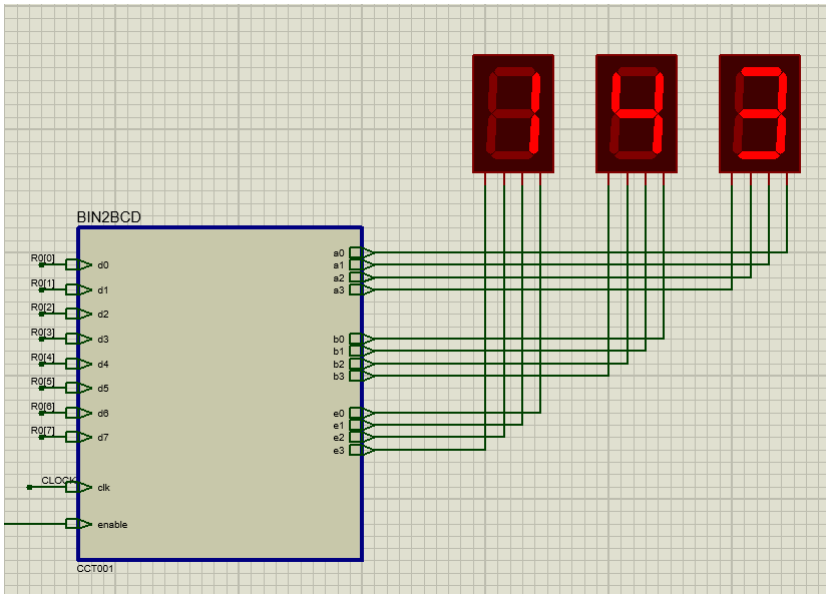


شکل ۵: به دست آوردن enable برای تبدیل کننده

در صورتی که به دنبال محاسبه ی جمع n عدد اول فیبوناچی باشیم، پس از اجرای برنامه حاصل جمع این n عدد در رجیستر R۰ ،عدد n م فیبوناچی در رجیستر R۱ و عدد n+1 م فیبوناچی در رجیستر R۲ موجود خواهد بود. به این ترتیب در صورتی که در قطعه ی U۶ که حافظه ی برنامه است فایل fib10-memory-program.bin را بارگذاری کنیم حاصل ۱۴۳ در رجیستر R۰ ،حاصل ۵۵ در رجیستر R۱ و حاصل ۸۹ در رجیستر R۲ ریخته خواهند شد و پس از پایان تبدیل Binary به BCD و ۷Seg، ها نیز مقدار ۱۴۳ را نمایش خواهند داد. این نتیجه در شکل ۶ قابل مشاهده است.
حال اگر حافظه ی اصلی را با فایل fib5-memory-program.bin بارگذاری کنیم حاصل مجموع ۵ جمله ی اول که برابر با ۱۲ است در رجیستر R۰ ،حاصل جمله ی پنجم فیبوناچی که همان ۵ است در رجیستر R۱ و نهایتا حاصل جمله ی ششم که ۸ است در رجیستر R۲ ریخته شده و پس از پایان تبدیل Binary به BCD نمایشگرهای 7seg نیز مقدار ۱۲ را نشان خواهند داد.
دقت کنید برای محاسبه ی سریع تر نتایج از مقدار ۱۰ هرتز برای کلاک استفاده شده که برای بررسی دقیق تر اجرای هر دستور می توان فرکانس کلاک را کاهش داد.



شکل ۶: رجیستر ها در محاسبه ۱۰ جمبه اول فیبوناچی



شکل ۷: نمایش دهنده در محاسبه ۱۰ جمله اول

نتیجه‌گیری

پس در این آزمایش موفق شدیم آنچه را که در ۳ آزمایش قبلی انجام داده بودیم، به شکل ریز پردازنده در بیاوریم و از قابلیت های آن استفاده کنیم و همچنین چند برنامه نمونه را روی آن اجرا کردیم.