بسم الله الرحمن الرحیم

آزمایشگاه معماری کامپیوتر

گزارش کار آزمایش هفتم: کنترل توسط برنامه ذخیره شده در حافظه

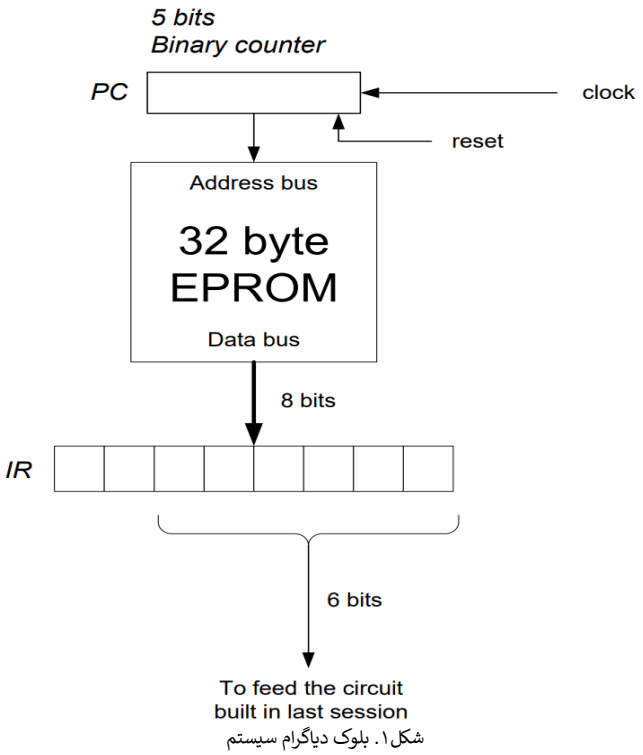
دانشکده مهندسی کامپیوتر
دانشگاه صنعتی شریف

تابستان 1403

رادین چراغی ۴۰۱۱۰۵۸۱۵
مبین پورعابدینی ۴۰۱۱۱۰۵۵۶
آرین نوری ۴۰۱۱۰۶۶۶۳

مقدمه

در این آزمایش می‌خواهیم تا به مداری که در آزمایش قبل طراحی کردیم یک ویژگی اضافه کنیم و آن اضافه کردن حافظه برای دستورات است، به این صورت که برای هر دستوری یک کد تعیین کرده و داخل حافظه‌ی EPROM قرار می‌دهیم و با آغاز برنامه دستورات خط به خط تا انتها پیش می‌روند. این فرمان‌ها توسط یک Program counter آدرس‌دهی می‌شوند و پس از واکشی از حافظه اجرا خواهند شد. در شکل زیر بلوک دیاگرام سیستم را مشاهده می‌کنید. که در آن دستورها 8 بیتی هستند و ما با دو بیت سمت چپ کاری نداریم.



شکل۱. بلوک دیاگرام سیستم

در ادامه پس از طراحی بخش حافظه و شمارنده‌ی برنامه، قطعه کدی را برای اجرای برنامه‌ی فیبوناچی درون حافظه قرار داده و آن را اجرا می‌کنیم. پس به ترتیب ابتدا آزمایش را شرح می‌کنیم و طراحی را آغاز می‌کنیم. سپس طراحی را مورد آزمایش قرار داده و در نهایت قطعه‌ی کد مربوط به فیبوناچی را در آن اجرا می‌کنیم.

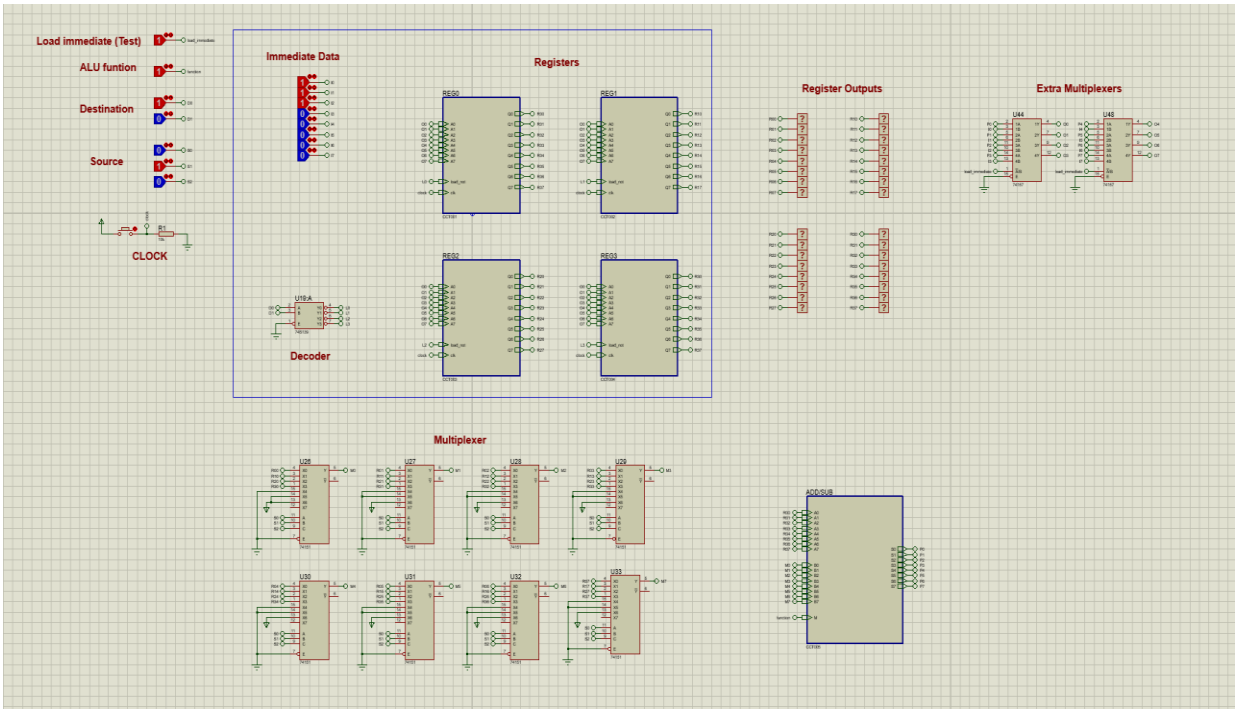
شرح و طراحی آزمایش در پروتئوس

حالا طراحی را شروع می‌کنیم، ابتدا باید نحوه کار مدار آزمایش قبل را مرور کنیم، در آزمایش قبل ما چهار رجیستر هشت بیتی داشتیم، به همراه یک واحد جمع/تفریق‌کننده و در نتیجه یک مالتی‌پلکسر برای انتخاب مقصد خروجی این واحد. همچنین یکی از ورودی‌های واحد محاسباتی همواره رجیستر صفر مدار بود و ورودی دیگر آن نیز هشت حالت داشت که به ترتیب به این صورت بودند: رجیستر صفر، رجیستر یک، رجیستر دو، رجیستر سه، 0، 1 و منفی1. (ورودی هشتم نیز رزرو برای آزمایش‌های بعدی)

به طور خلاصه مدار از چهار رجیستر، هشت واحد مالتی‌پلکسر هشت به یک برای انتخاب ورودی واحد محاسباتی، یک واحد محاسباتی با قابلیت جمع و تفریق، ورودی و خروجی و ساعت، و در نهایت یک سری قطعات کمکی اضافه برای Load Immediate تشکیل شده بود که این قطعات تنها برای تست راحت‌تر مدار به آن اضافه شده بودند و در این آزمایش برای ما اهمیتی ندارند.
همچنین می‌دانیم که فرمت دستورها به این شکل است:

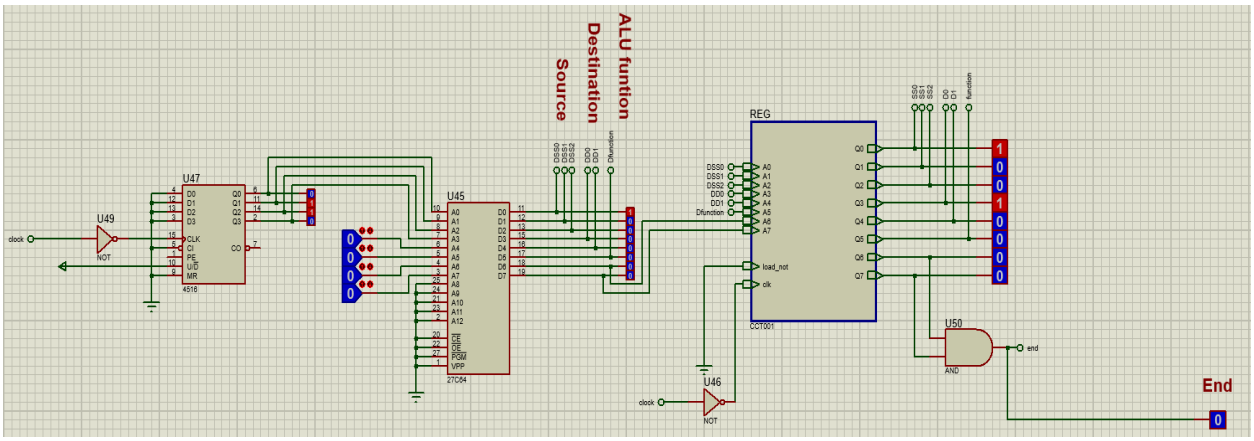
00 M DD SSS, M = Adder/Subtractor Selector, DD = Destination Selector, SSS = ALU Source Selector

در نهایت ساخت مدار را از روی مدار قبلی شروع می‌کنیم،



شکل۲. مدار نهایی آزمایش قبلی

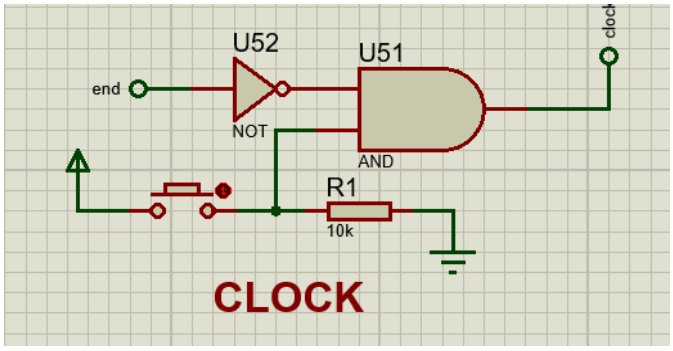
باید یک شمارنده برای شمارش دستورات قرار دهیم و در ادامه حاصل شمارش به قطعه‌ی EPROM آماده بدهیم، برای راحتی مدار و وضوح بیشتر یک رجیستر هشت بیتی جلوی خروجی EPROM گذاشته تا ترتیب دستورات را پشت سر یکدیگر ببینیم. همچنین برای سیگنال ساعت آن باید NOT ساعت اصلی مدار را به آن متصل کنیم. با این کار موقع لبه‌ی بالارونده، دستوری که در خروجی رجیستر دستورات قرار دارد اجرا شده و در لبه‌ی پایین‌رونده، دستور بعدی وارد رجیستر دستور می‌شود. پس نحوه‌ی پیاده‌سازی به این صورت می‌شود:



شکل۳. چیدمان EPROM در مدار

گیت EPROM مدار، همان قطعه‌ی 27C64 در شکل بالا است که از شمارنده ورودی می‌گیرد و به رجیستر می‌دهد. در انتهای مدار نیز می‌بینید که ترکیب AND دو بیت انتهایی به عنوان بیت end در نظر گرفته شده. این یعنی هرگاه بخواهیم که به EPROM اعلام کنیم مدار به پایان رسیده کافیست دستوری قرار دهیم که دو بیت آخر(که همان بیت‌های اضافی در دستورات بودند) در آن یک باشند.

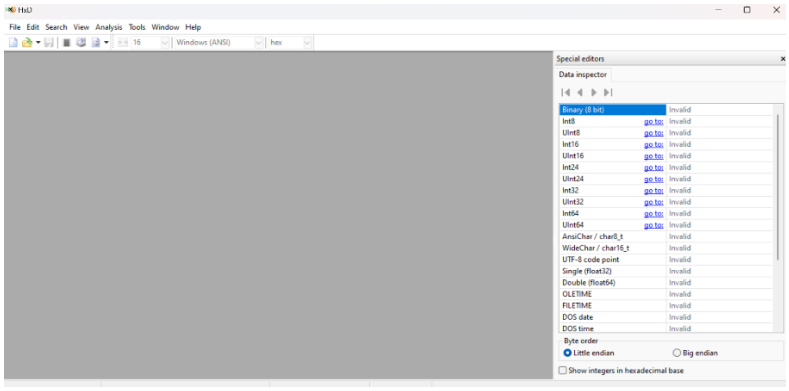
برای مثال دستور پایانی به این صورت می‌تواند باشد: 0000 1100. در ادامه نیز این بیت اعلام پایان را به ساعت متصل می‌کنیم، به طوری که دیگر ساعت زدن را بی‌اثر کند.



شکل۴. بی‌اثر کردن ساعت پس از پایان دستورات

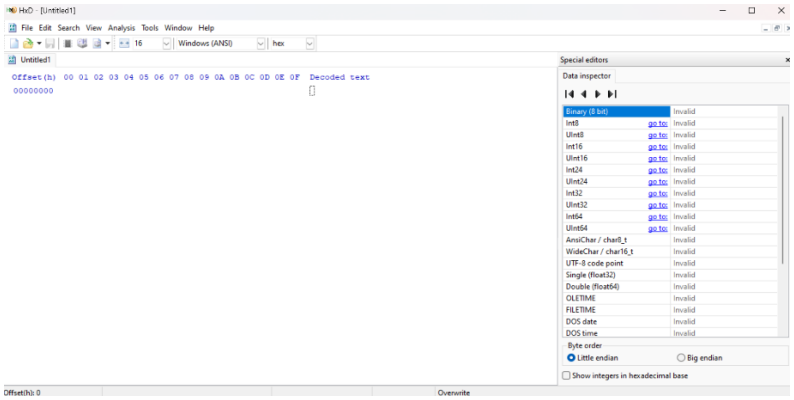
آزمایش مدار طراحی شده در پروتئوس

با به پایان رسیدن تغییرات مدار کافی است که آن را تست کنیم. برای تست مدار و ایجاد قابلیت نوشتن دستورات در EPROM، باید برنامه‌ای داشته باشیم که بتواند به ما در طراحی دستورات به طور HEX کمک کند. از برنامه‌ی HxD کمک می‌گیریم. پس از نصب و باز کردن آن با چنین صفحه‌ای روبه‌رو می‌شویم:



شکل۵. برنامه‌ی HxD

از طریق منوی File، یک فایل جدید می‌سازیم:



شکل۶. ساخت فایل جدید

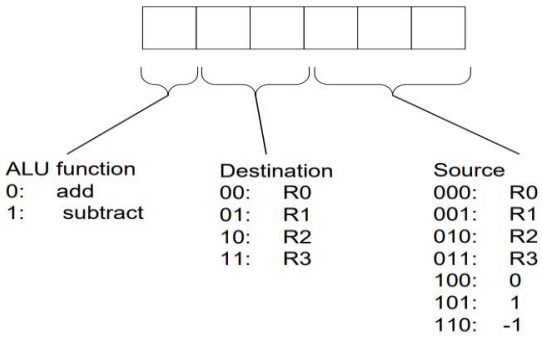
سپس می‌توانیم در این فایل دستورات را وارد کنیم. فرض کنید برای مثال هدف اجرای برنامه‌ی زیر باشد:

```
SUB R0,
ADD R0, 1
ADD R0, 1
MOV R1, R0
ADD R0, 1
ADD R0, 1
MOVADD R2, 1
SUB R0, R2
ADD R0, R1
```

از آنجایی که همواره یکی از ورودی‌های ALU، R0 است پس تکه کد بالا در واقع عملیات زیر را انجام می‌دهد:

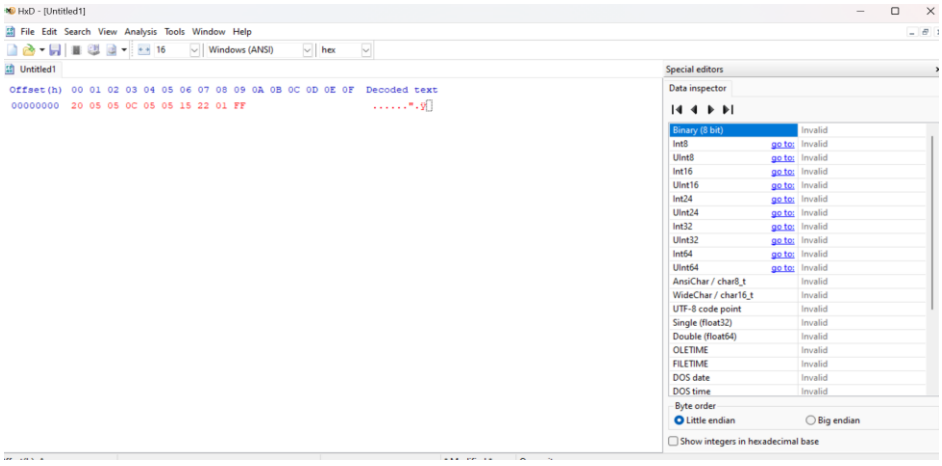
```
R0 -= R0;       //R0 = 0
R0++;           //R0 = 1
R0++;           //R0 = 2
R1 = R0;        //R1 = 2
R0++;           //R0 = 3
R0++;           //R0 = 4
R2 = R0 + 1;    //R2 = 5
R0 = R0 – R2;   //R0 = 4 – 5 = – 1
R0 = R0 + R1;   //R0 = – 1 + 2 = 1
```

طبق قالبی که ابتدا نیز به آن اشاره کردیم دستور بالا را کد می‌کنیم:



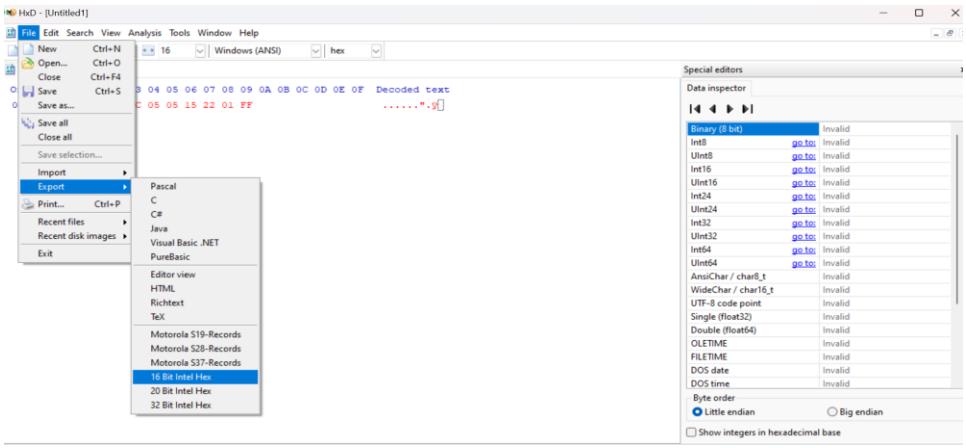
شکل۷. قالب دستورات ورودی به EPROM

```
R0 -= R0;       //R0 = 0              CODE: 00  1  00  000      HEX: 20
R0++;           //R0 = 1              CODE: 00  0  00  101      HEX: 05
R0++;           //R0 = 2              CODE: 00  0  00  101      HEX: 05
R1 = R0;        //R1 = 2              CODE: 00  0  01  100      HEX: 0C
R0++;           //R0 = 3              CODE: 00  0  00  101      HEX: 05
R0++;           //R0 = 4              CODE: 00  0  00  101      HEX: 05
R2 = R0 + 1;    //R2 = 5              CODE: 00  0  10  101      HEX: 15
R0 = R0 – R2;   //R0 = 4 – 5 = – 1    CODE: 00  1  00  010      HEX: 22
R0 = R0 + R1;   //R0 = – 1 + 2 = 1    CODE: 00  0  00  001      HEX: 01
                //FINISH              CODE: 11  X  XX  XXX      HEX: FF
```



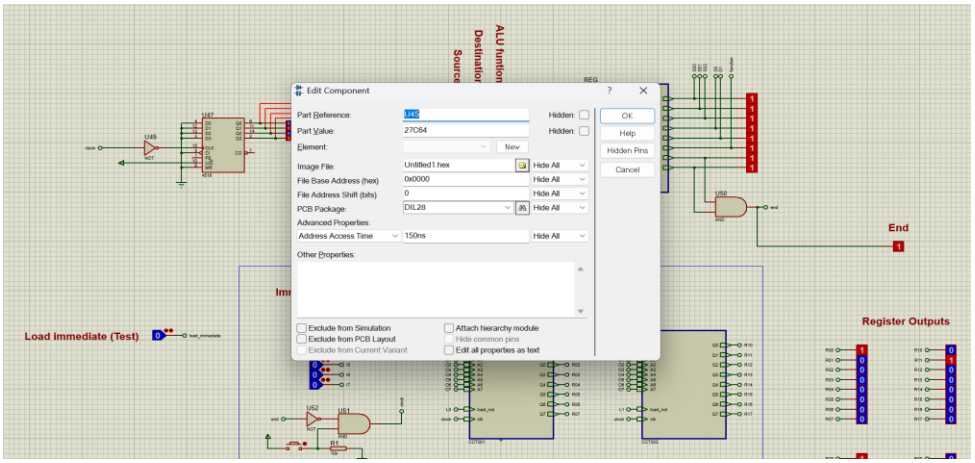
شکل۸. نوشتن دستورات در برنامه

از طریق منوی FILE، و سپس منوی EXPORT، بر روی 16 Bit Intel Hex کلیک می‌کنیم تا فایل ساخته شود:



شکل۹. ساختن فایل دستورات

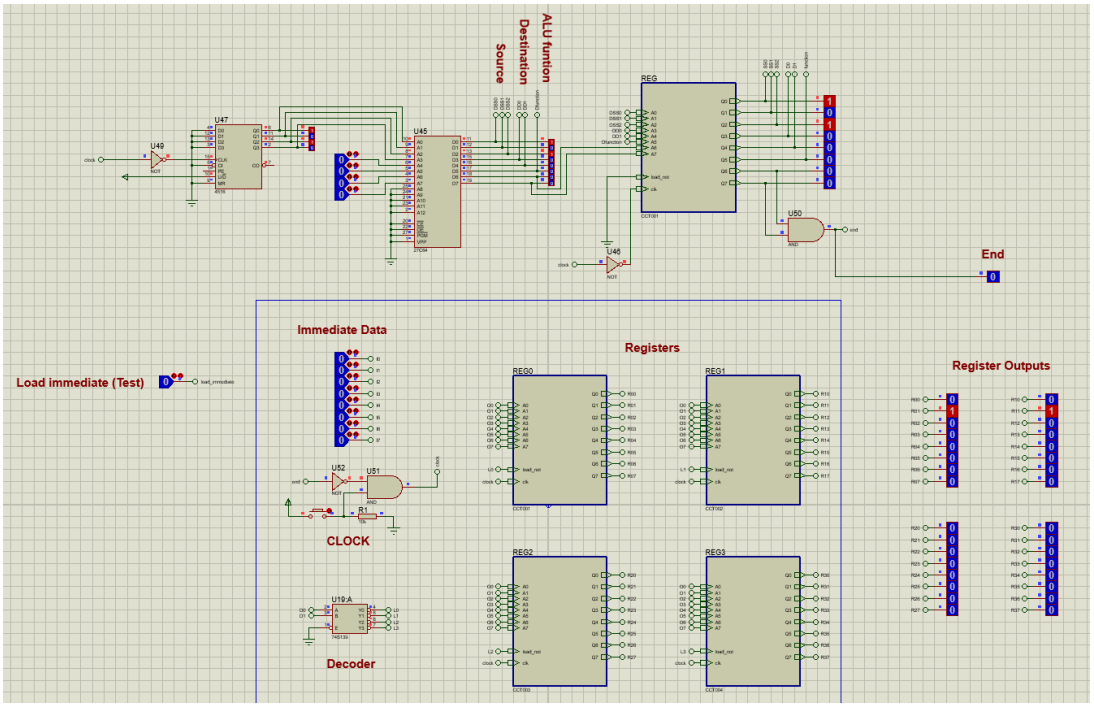
در ادامه وارد پروتئوس شده و کد را بر روی EPROM قرار می‌دهیم، به این صورت که بر روی Edit Properties کلیک کرده و در بخش Image File، فایلی که ساخته‌ایم را قرار می‌دهیم:



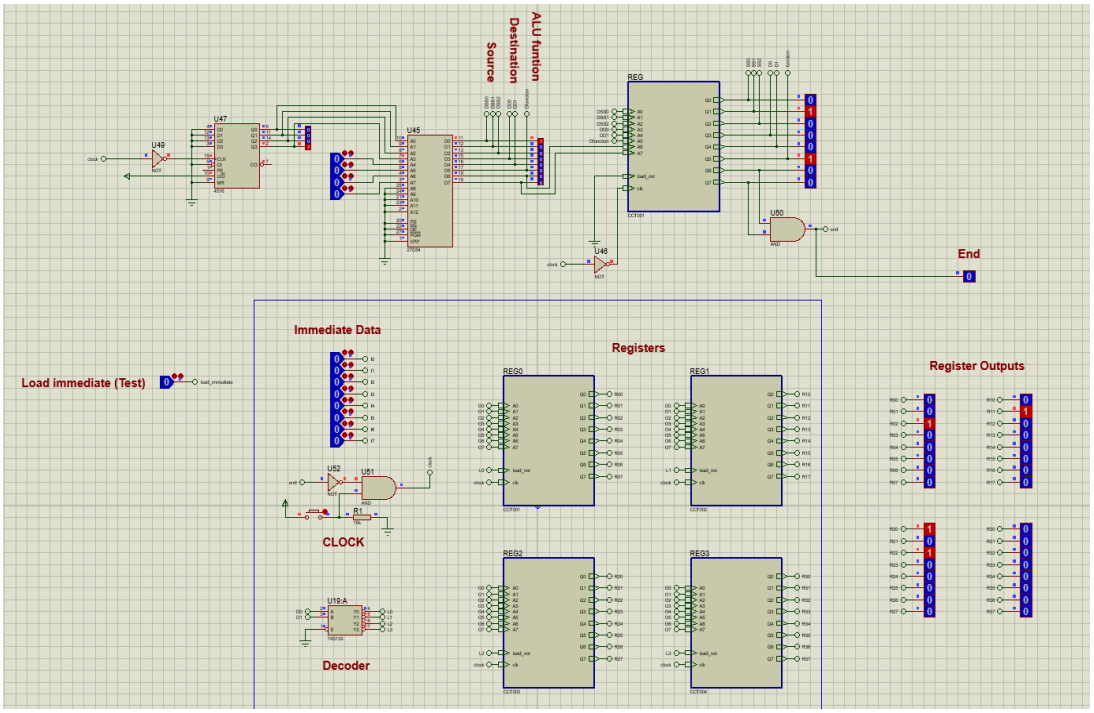
شکل۱۰. اتصال کد نوشته شده به EPROM

بعد از این که کد به درستی متصل شد کافیست آن را تست کنیم تا ببینیم همان خواسته‌ی ما انجام می‌شود یا خیر.

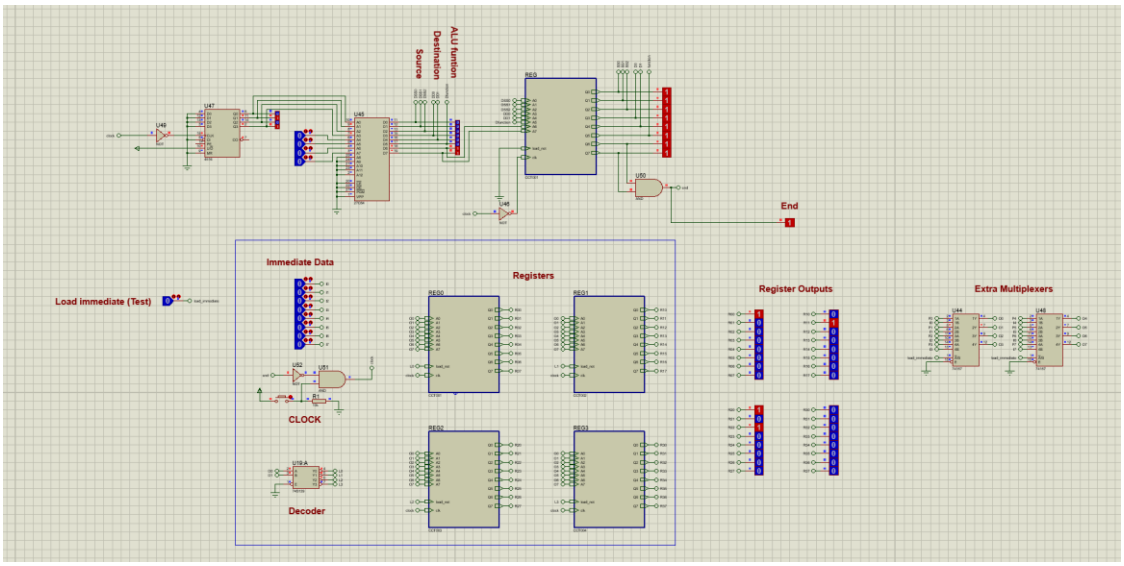
تصاویر تست کدنویسی بر روی EPROM



شکل۱۱. بعد از سه واحد زمانی، مطابق انتظار رجیستر صفر، دو واحد زیاد شده و به درون رجیستر یک ریخته شده



شکل۱۲. بعد از شش واحد زمانی، مطابق انتظار رجیستر صفر، دو واحد زیاده شده و با یک جمع شده و درون رجیستر دو، ریخته شده



شکل۱۳. در انتهای کد، مطابق انتظار مقدار یک درون رجیستر صفرم ریخته شده و بیت end روشن شده

در نتیجه قطعه کد امتحانی به درستی کار کرد و حالا به بخش پایانی آزمایش یعنی نوشتن کد فیبوناچی می‌پردازیم.

برنامه‌ی فیبوناچی

در این بخش، تابع فیبوناچی با ورودی‌های ابتدایی صفر و یک را کدنویسی خواهیم کرد:

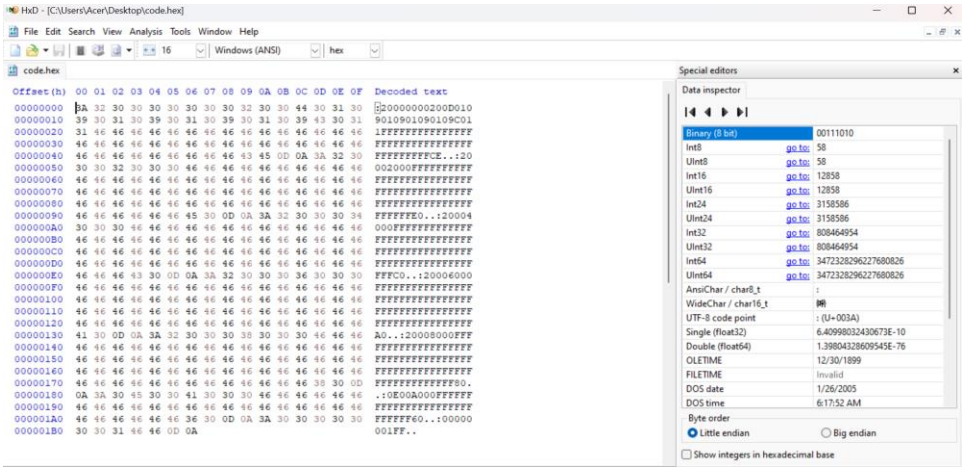
$$F(n) := \begin{cases} 0 & \text{if } n = 0; \\ 1 & \text{if } n = 1; \\ F(n-1) + F(n-2) & \text{if } n > 1. \end{cases}$$

شکل۱۴. تابع فیبوناچی

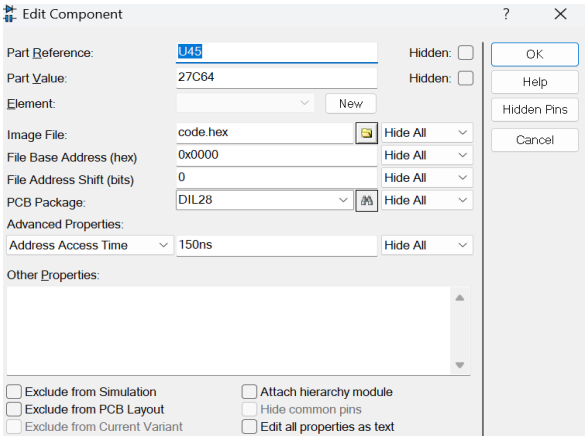
جدول کد مورد انتظار ما به این شکل خواهد بود:

ADDRESS	CODE	INSTRUCTION	COMMENT
0000	00100000: 20	SUB R0, R0	R0 = 0
0001	00001101: 0D	ADD R1, 1	R1 = 1
0010	00000001: 01	ADD R0, R1	R0 = 1
0011	00001001: 09	ADD R1, R0	R1 = 2
0100	00000001: 01	ADD R0, R1	R0 = 3
0101	00001001: 09	ADD R1, R0	R1 = 5
0110	00000001: 01	ADD R0, R1	R0 = 8
0111	00001001: 09	ADD R1, R0	R1 = 13
1000	00000001: 01	ADD R0, R1	R0 = 21
1001	00001001: 09	ADD R1, R0	R1 = 34
1010	11000000: C0	EOF	END

جدول۱. کد برنامه‌ی فیبوناچی



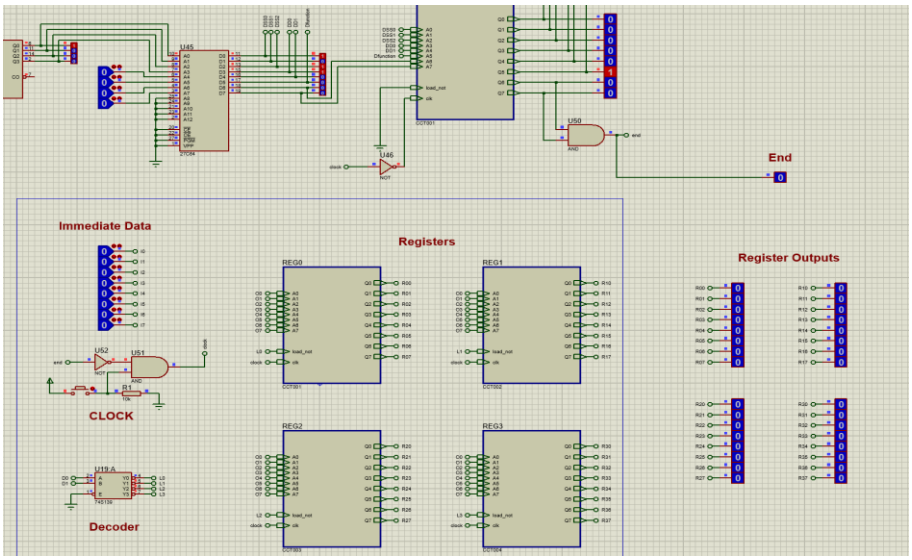
شکل۱۵. تصویر درون برنامه پس از ذخیره سازی در کامپیوتر



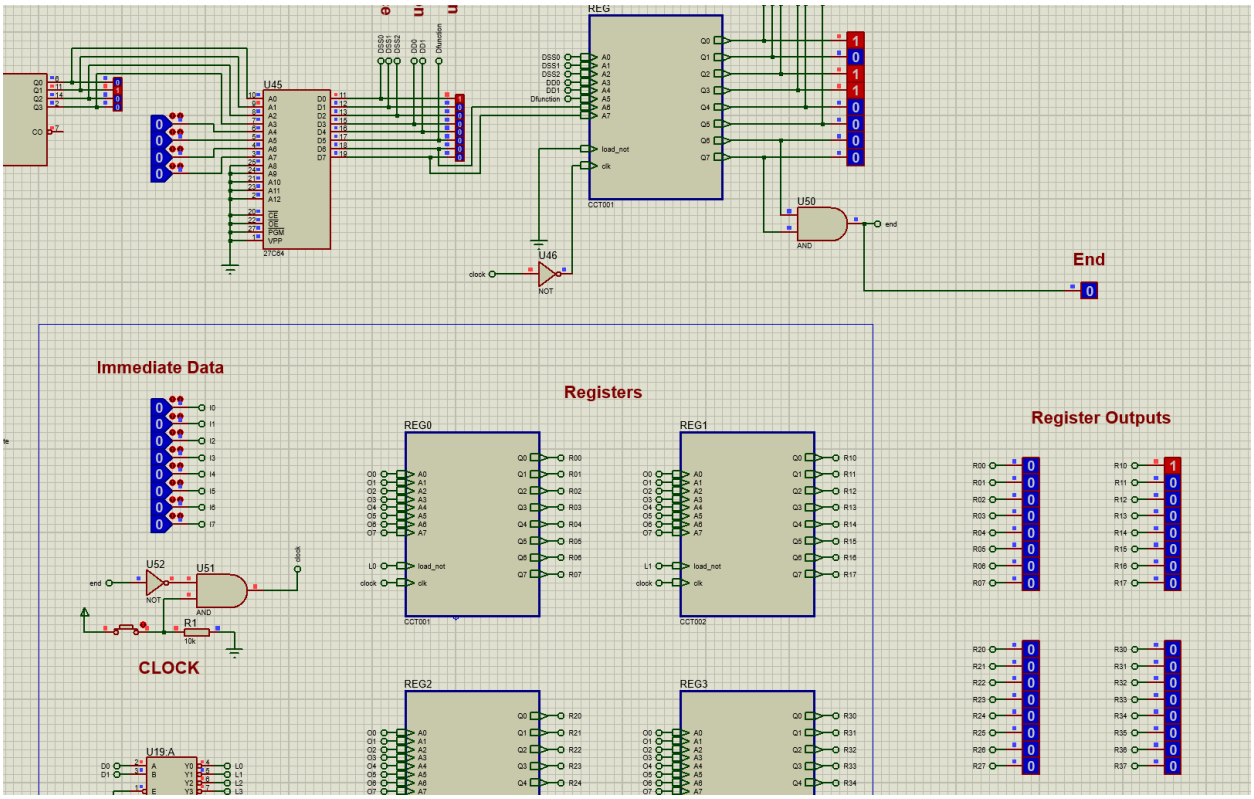
شکل۱۶. اتصال به EPROM

در ادامه به تصاویر تست بخش پایانی از مدار می‌رسیم.

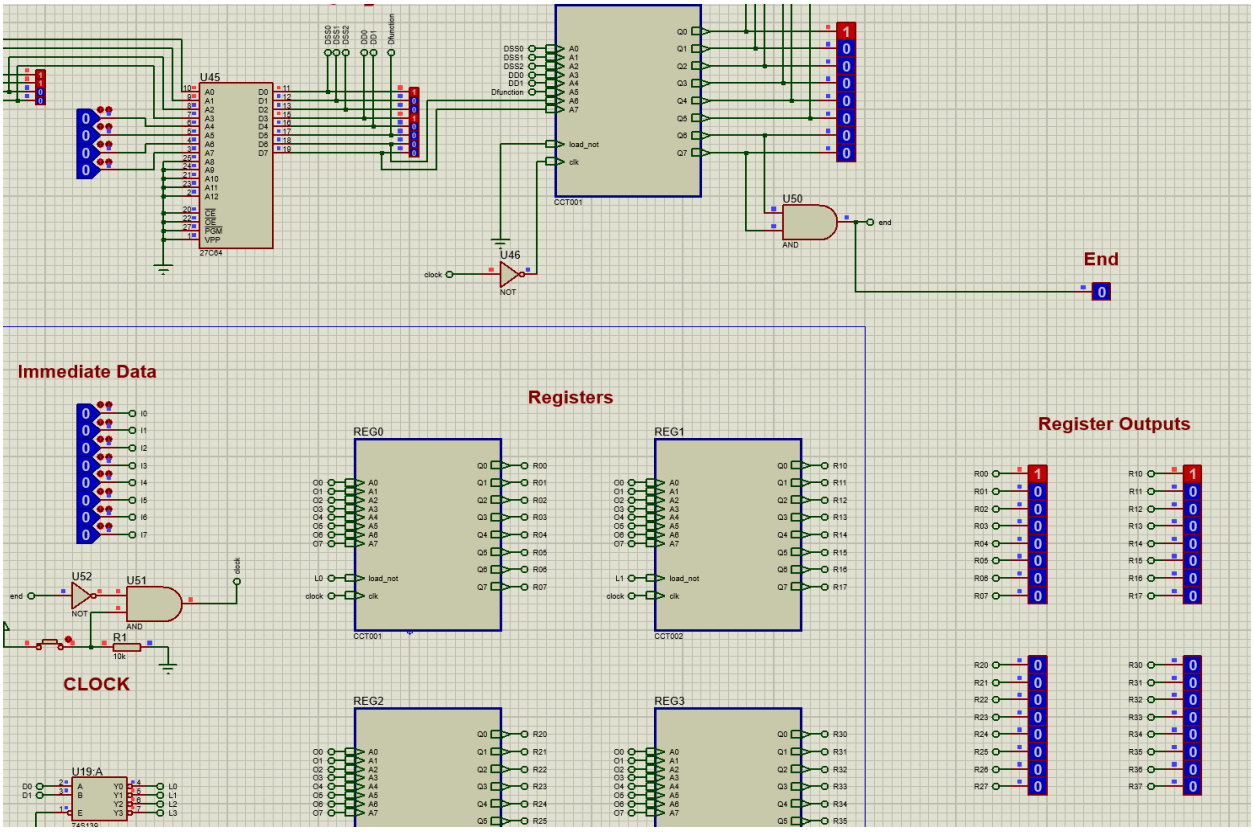
تصاویر تست برنامه‌ی فیبوناچی



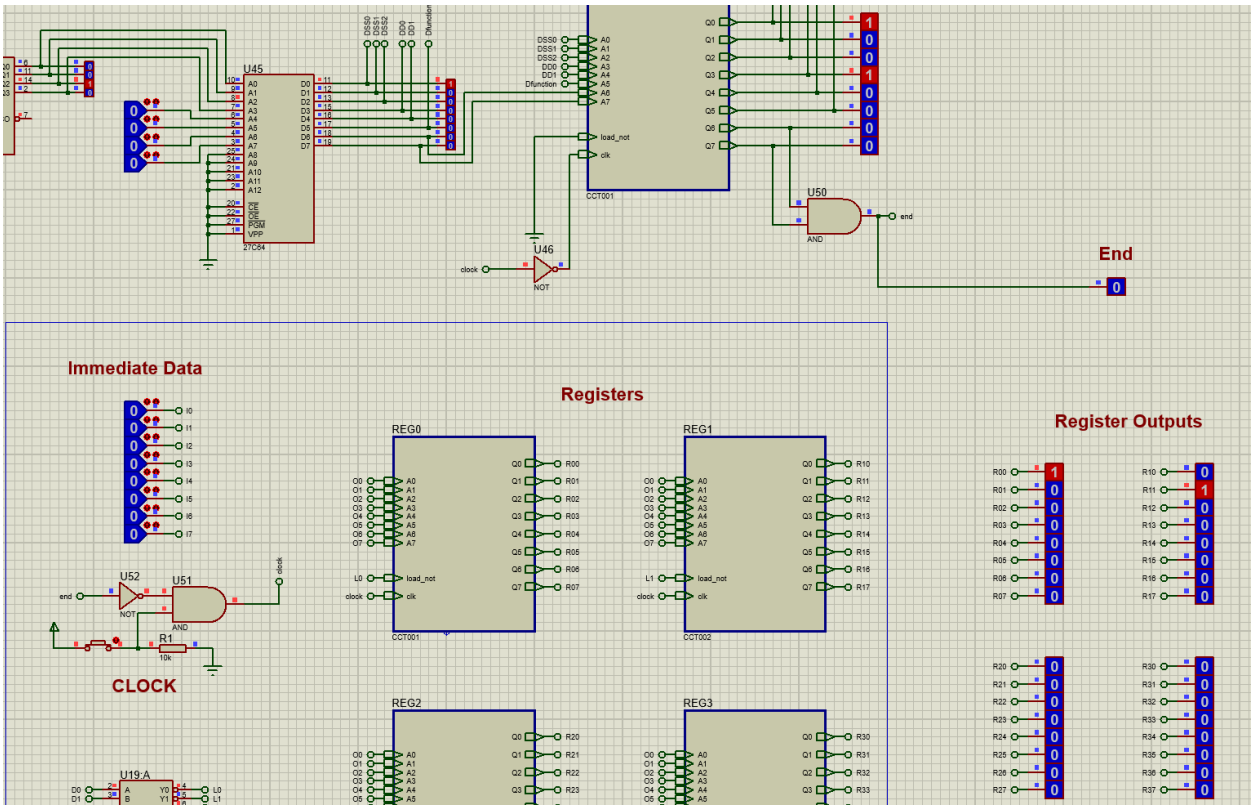
شکل۱۷. دستور اول: دادن صفر به رجیستر صفرم



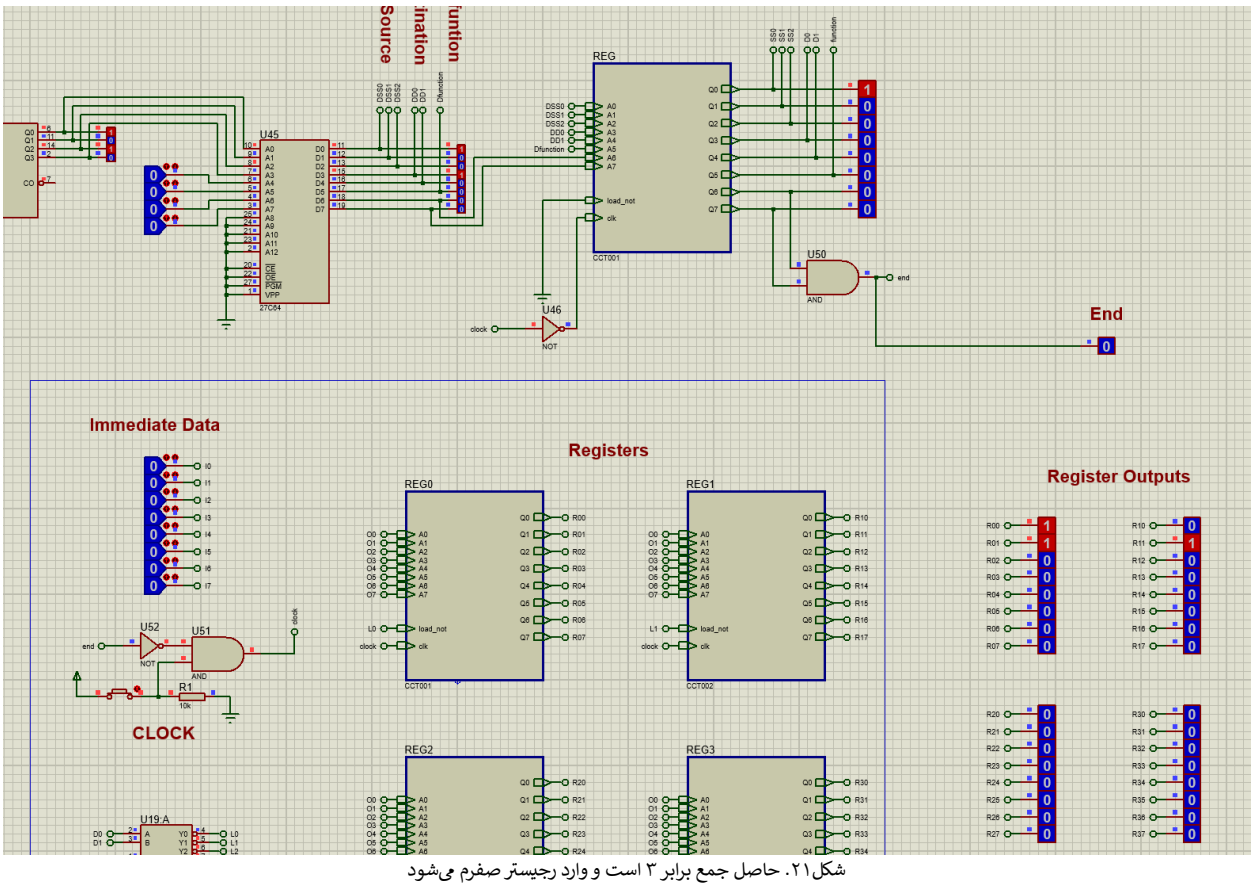
شکل۱۸. دستور دوم: دادن یک به رجیستر یکم



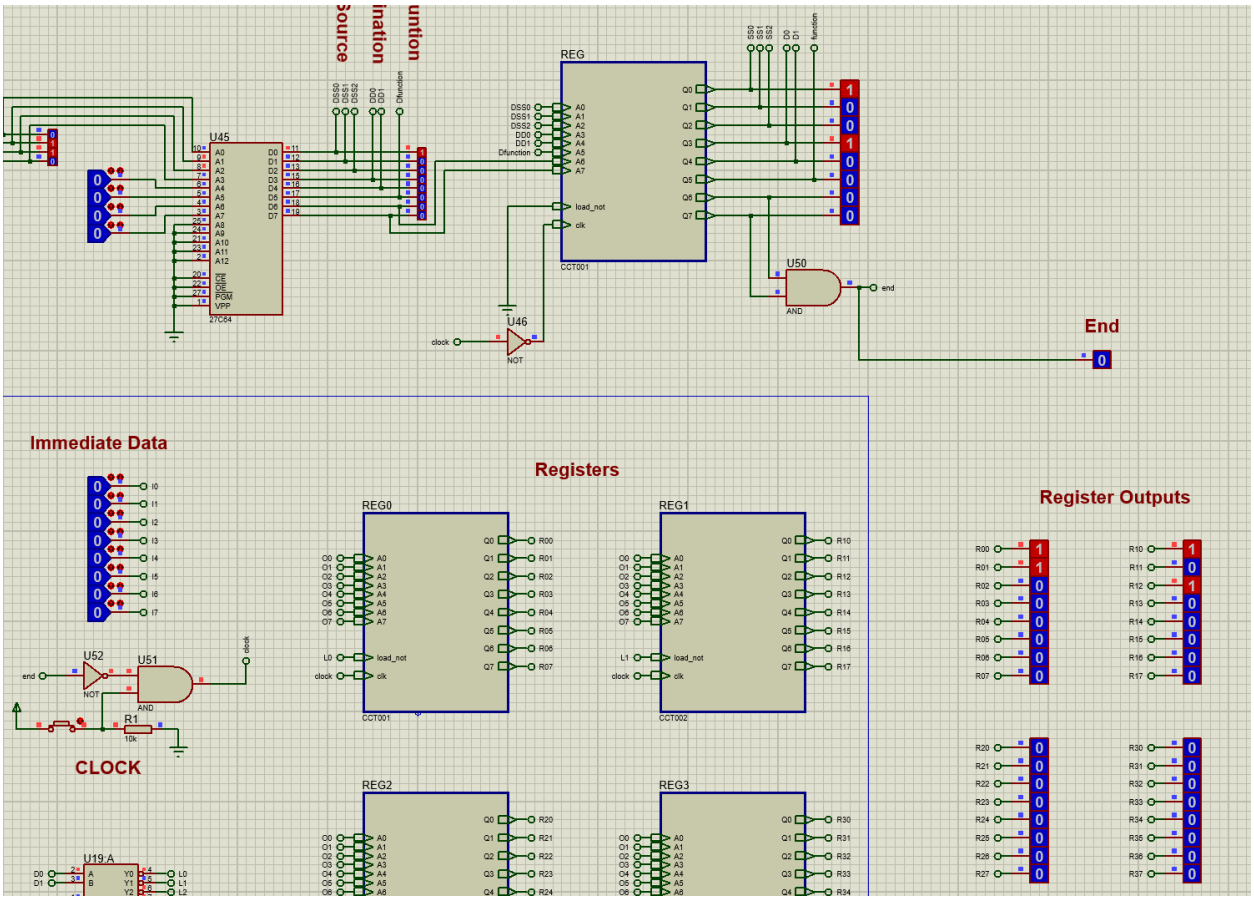
شکل۱۹. دستور سوم: حاصل جمع دو رجیستر در رجیستر صفرم



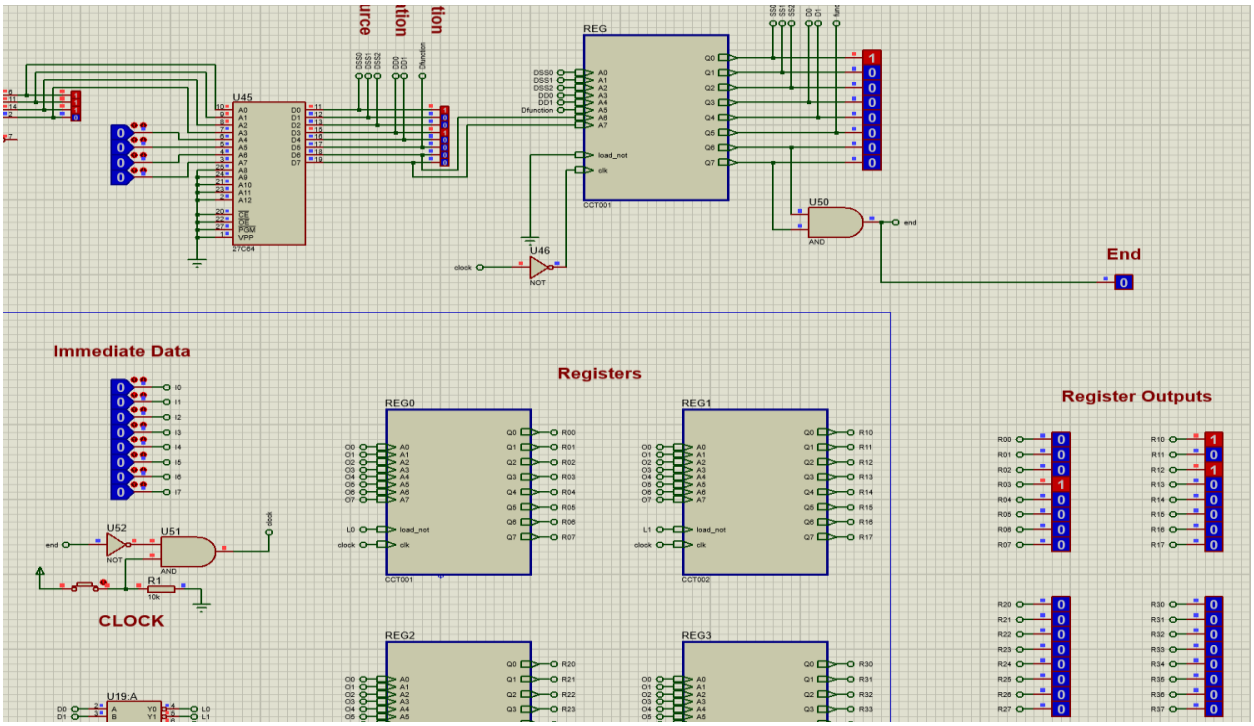
شکل۲۰. حاصل جمع برابر ۲ است و وارد رجیستر یکم می‌شود



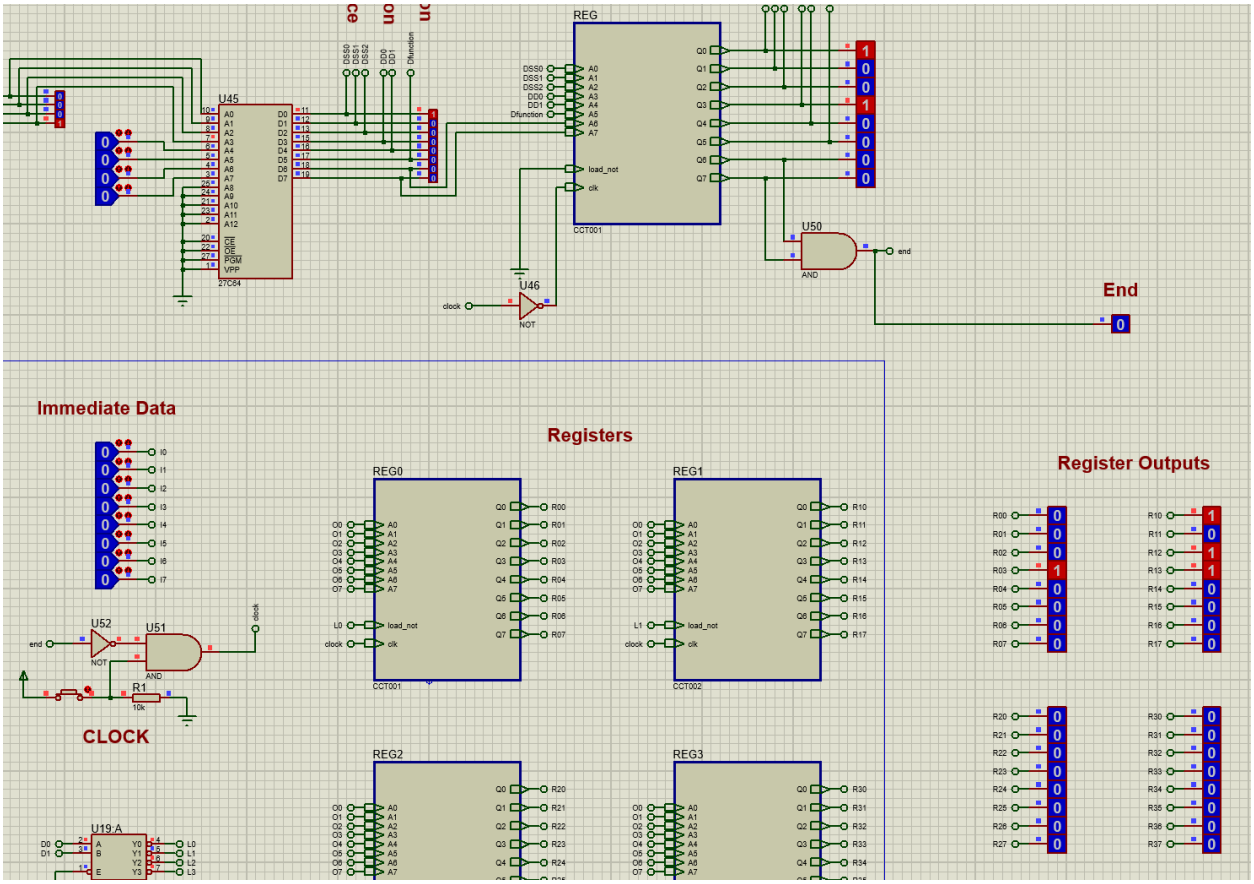
شکل۲۱. حاصل جمع برابر ۳ است و وارد رجیستر صفرم می‌شود



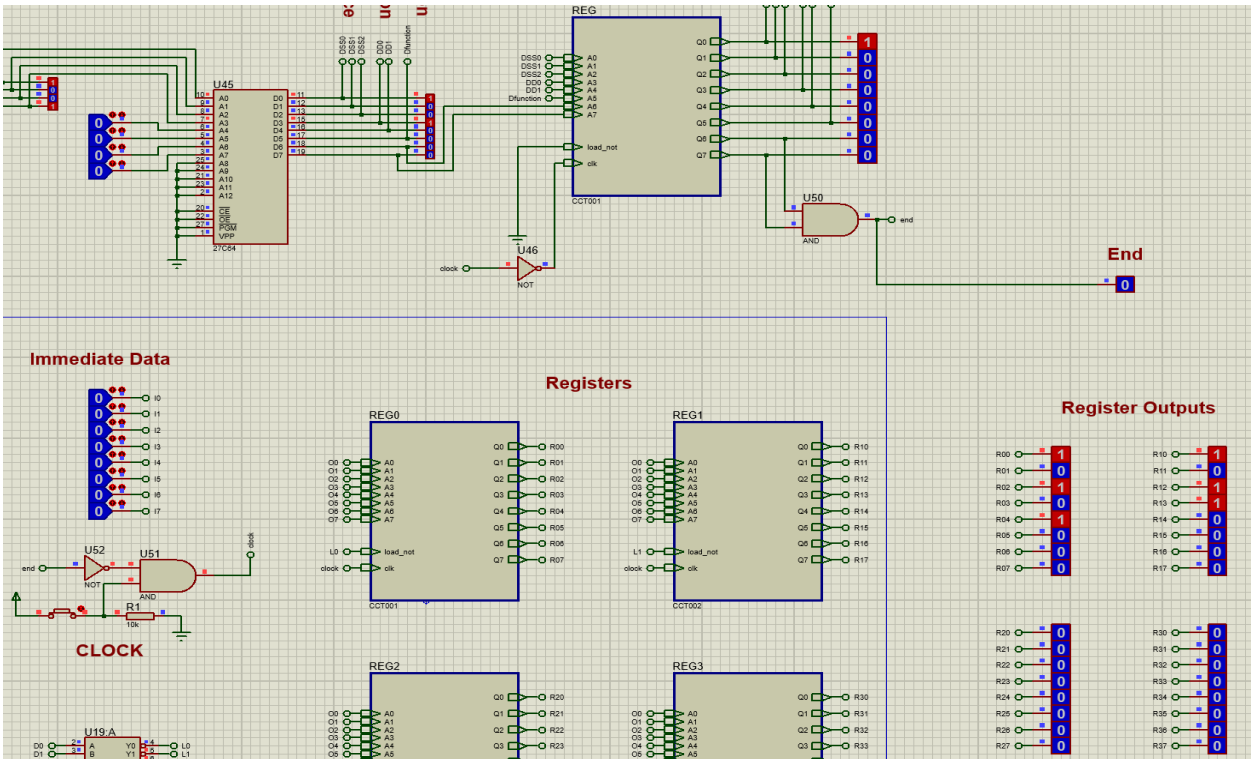
شکل۲۲. حاصل جمع برابر ۵ است و وارد رجیستر یکم می‌شود



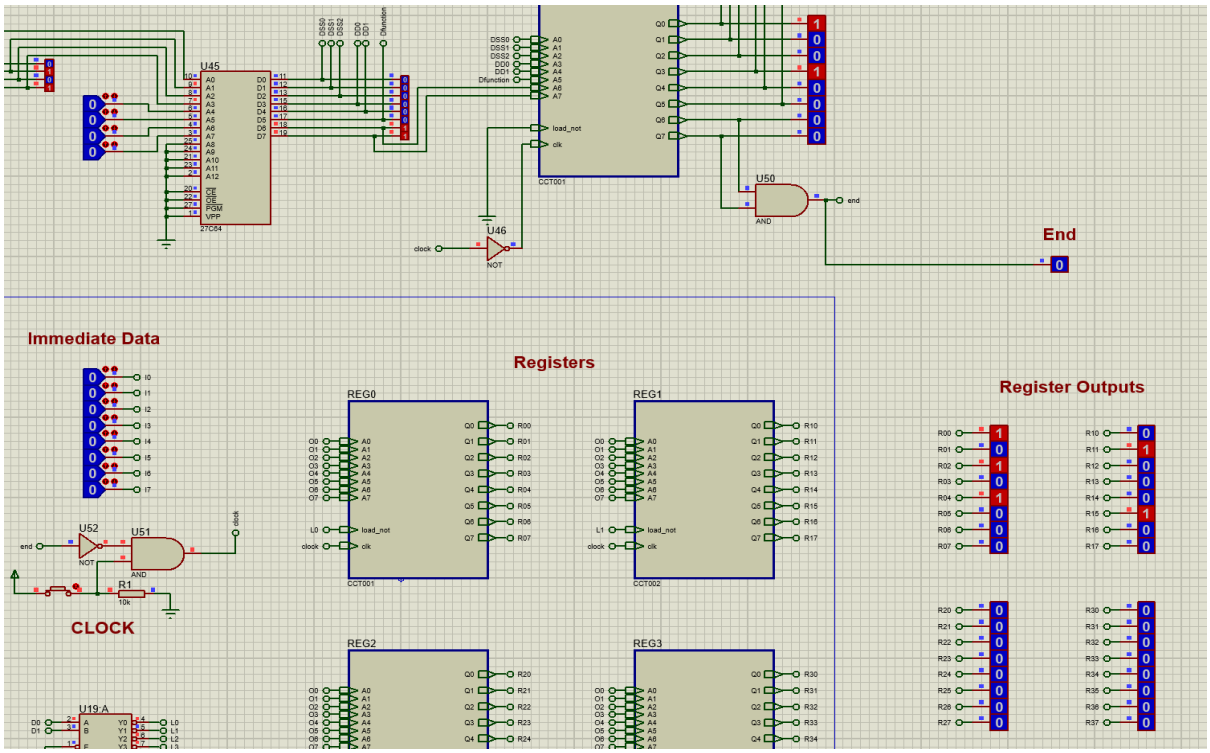
شکل۲۳. حاصل جمع برابر ۸ است و وارد رجیستر صفرم می‌شود



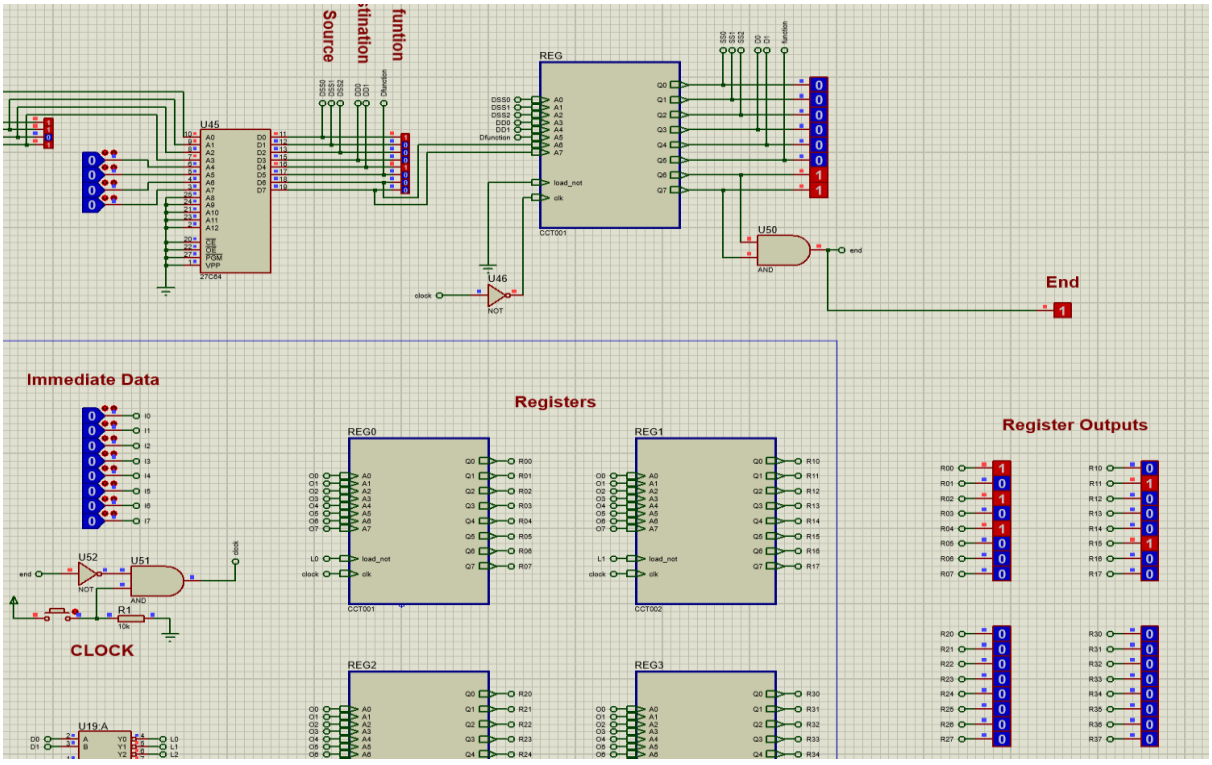
شکل۲۴. حاصل جمع برابر ۱۳ است و وارد رجیستر یکم می‌شود



شکل۲۵. حاصل جمع برابر ۲۱ است و وارد رجیستر صفرم می‌شود



شکل۲۶. حاصل جمع برابر ۳۴ است و وارد رجیستر یکم می‌شود



شکل۲۷. انتهای برنامه و روشن شدن بیت end

در نتیجه تمامی دستورات به شکل مرتب و درست، از حافظه اجرا شده و به نتیجه‌ی مطلوب رسیدیم.

پایان گزارش کار آزمایش هفتم