به نام خدا

دانشگاه صنعتی شریف

حافظه داده و پرش

درس

آزمایشگاه معماری کامپیوتر

نویسنده

امیرحسین براتی (99101308)

کیان بهادری (99105312)

دانشگاه صنعتی شریف

پاییز 1401

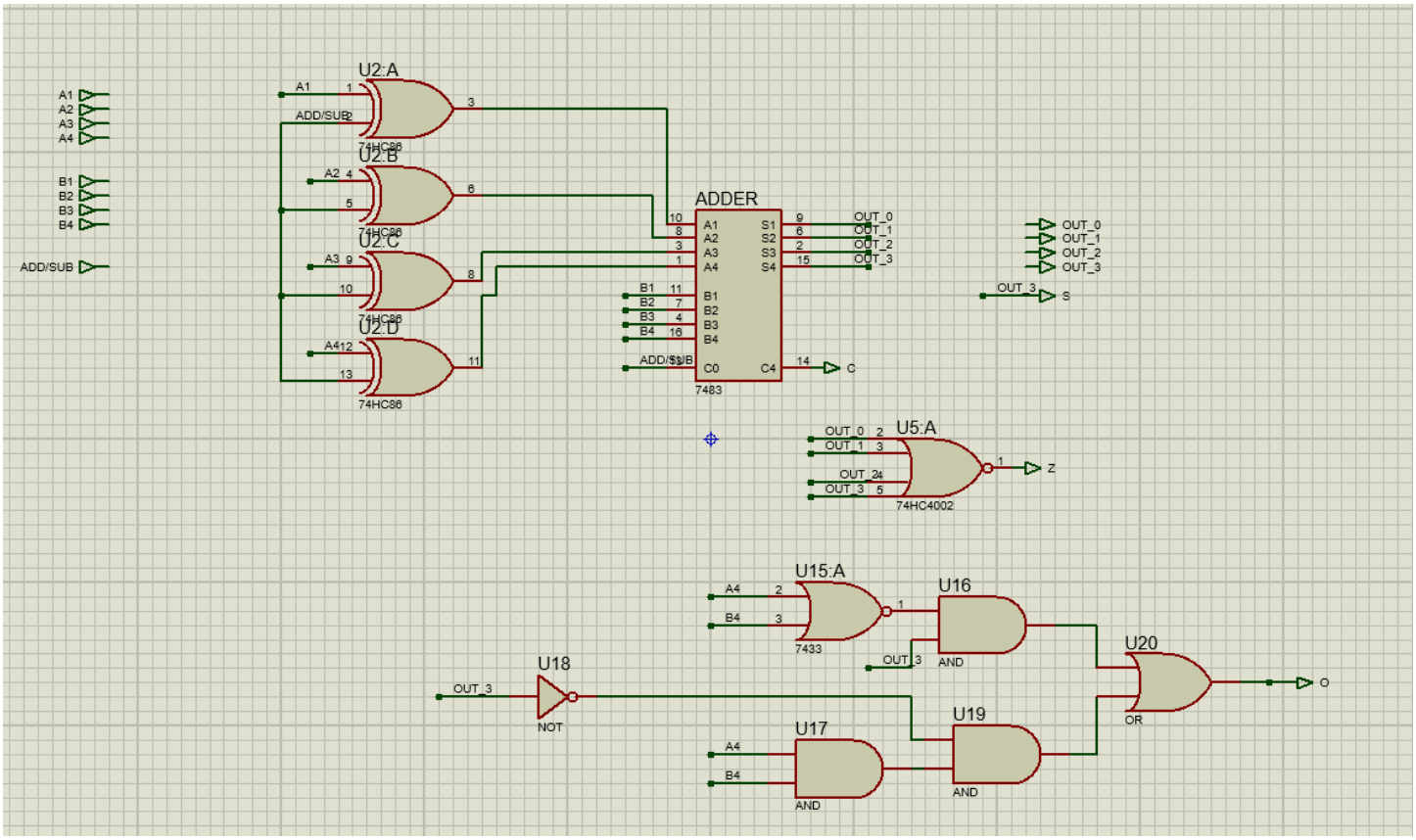
حافظه داده و پرش

هدف آزمایش

در این آزمایش، سعی داریم به مدار آزمایش شش، یک حافظه داده اضافه کنیم و علاوه بر دستورات واکشی و ذخیره، از دستورات پرشی استفاده کنیم.

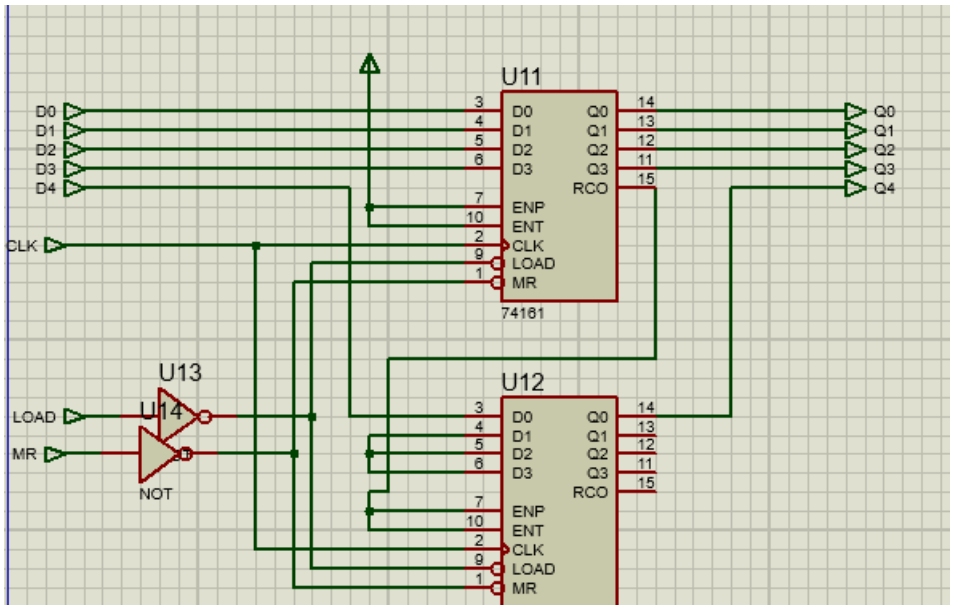
شرح آزمایش

دو ماژول 16 BIT RAM که برای آن از 7489 استفاده می‌کنیم در مدار قرار می‌دهیم. سپس یک DFF به عنوان FLAG REGISTER به مدار اضافه می‌کنیم. برای خوانایی بیشتر شکل مدار، قطعات ALU را بر خلاف آزمایش قبل درون یک ماژول مخصوص قرار می‌دهیم. در نهایت باید یک Control Circuit بسازیم، که به کمک یک شمارنده به عنوان زمان‌بند (Sequencer) و تعدادی گیت ترکیبی، این ماژول را می‌سازیم. در نهایت مطابق شکل 9 دستور کار، قطعات لازم را به هم متصل می‌کنیم. تصویر ماژول های ساخته شده به تفصیل در زیر آورده شده است.

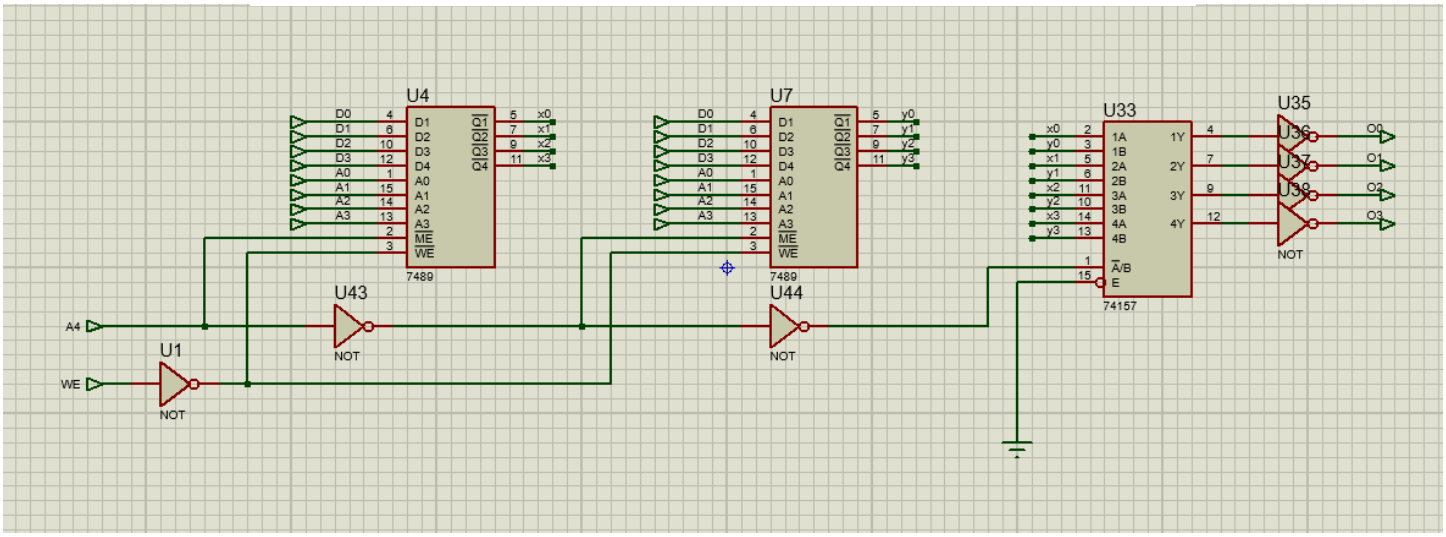


ساختار داخلی ALU

حافظه داده و پرش

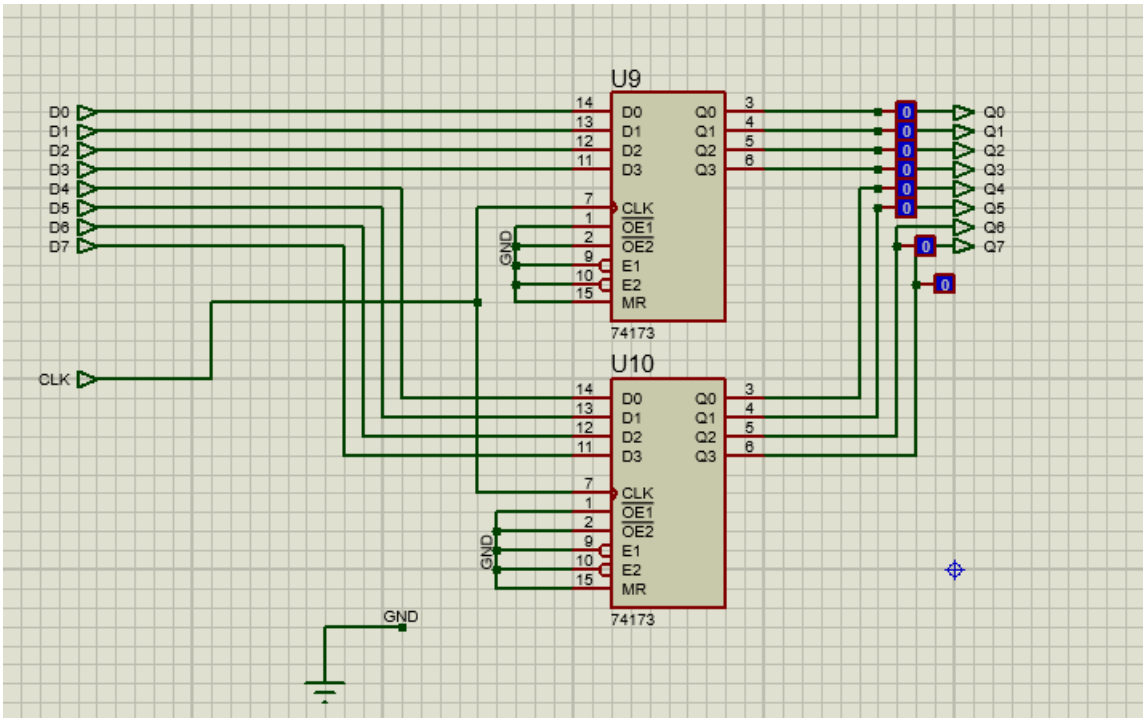


ساختار درونی شمارنده 32 حالتی



حافظه RAM ساخته شده

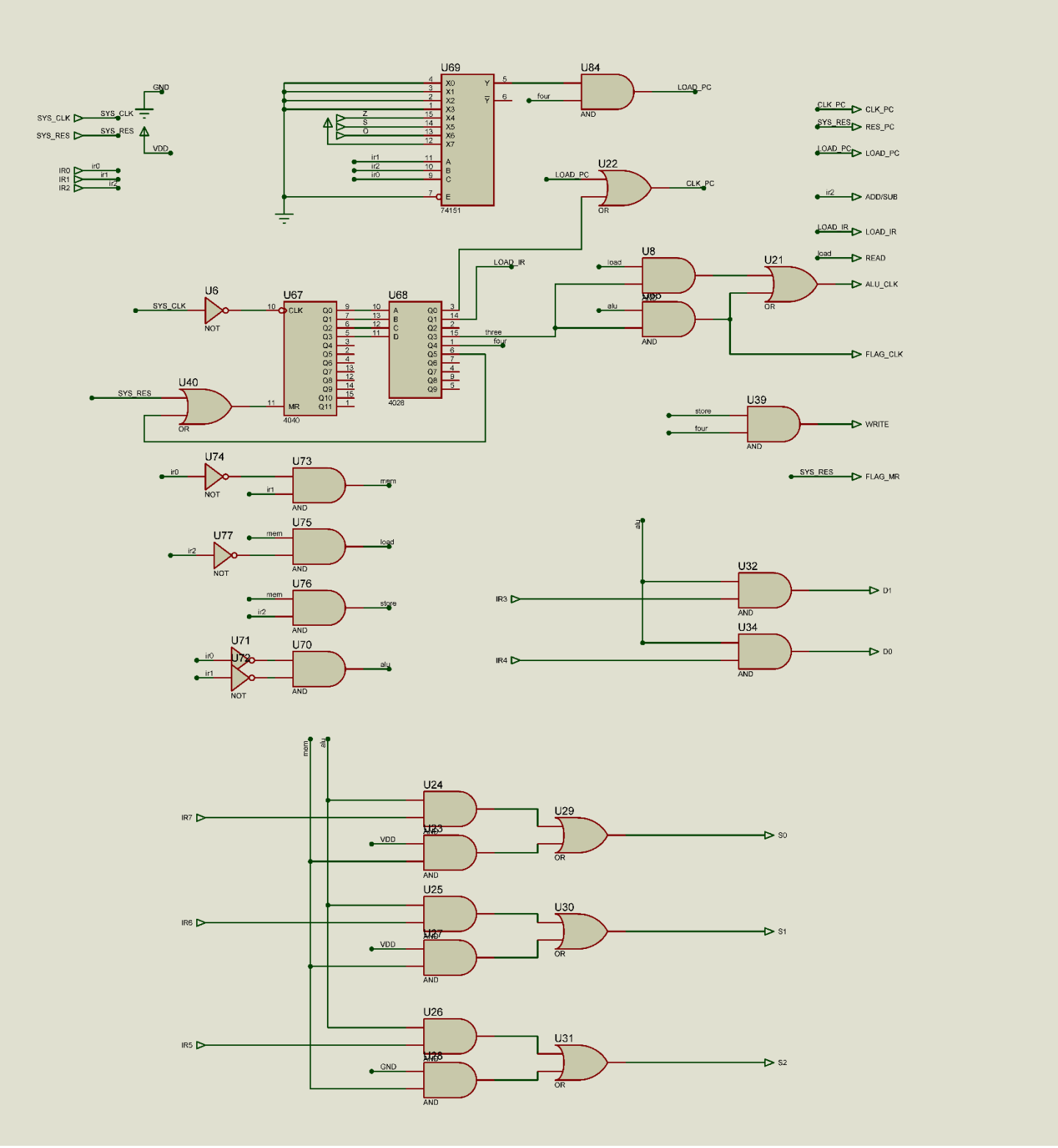
حافظه داده و پرش



معماری INSTRUCTION REGISTER

حافظه داده و پرش

مدار کنترل کننده نیز چنین ساختاری خواهد داشت:



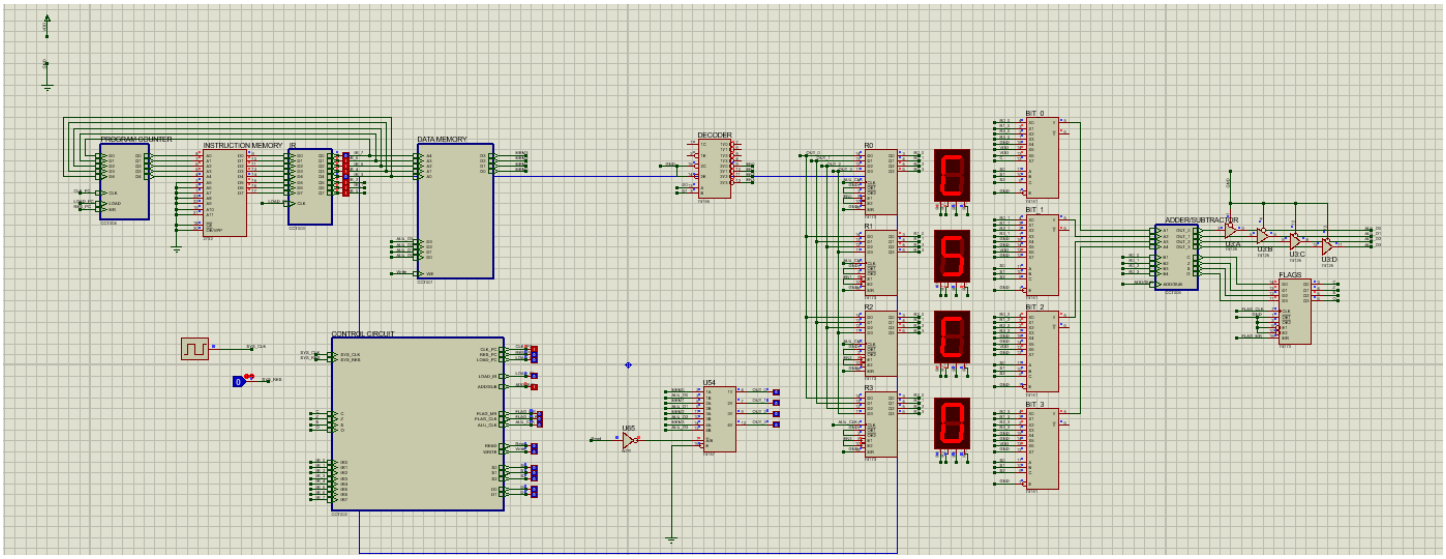
این مدار علاوه بر کلاک های هر بخش و سیگنال های ریست، وظیفه تعیین عملیات ALU را نیز بر عهده دارد.

حافظه داده و پرش

حال باید برنامه‌های اسمبلی گفته شده را تبدیل به کد ماشین کنیم و در اختیار حافظه EPROM قرار دهیم تا عملکرد مدار را مشاهده کنیم. توجه کنید که برای محاسبه جمع جملات دنباله فیبوناچی، با توجه به محدودیت 4 بیتی اعمال شده (به جای 8 بیت گفته شده در دستور کار) فقط توانایی نمایش جمع 6 جمله اول (که حاصل آن 12=Ch) است را داریم. برنامه مورد نظر اینگونه خواهد بود (به علت معماری خاص مدار کنترل، باید به یک دستور قبل از دستور مورد نظر پرش کنیم):

FIBONACCI SEQUENCE SUM OF FIRST 6 ELEMENTS:

sub r0, r0	20xh
add r1, 0	0Cxh
add r0, 1	05xh
add r2, r2	12xh
add r1, r1	09xh
add r0, r1	01xh
add r2, r2	12xh
add r1, r1	09xh
add r0, r1	01xh
add r2, r2	12xh
sub r0, r0	20xh
add r0, r2	02xh
store 0	60xh
jmp HERE	EExh



همانطور که انتظار داشتیم، حاصل شش جمله اول (Cxh=12) در حافظه ذخیره شده است.

حافظه داده و پرش

در قسمت بعد برنامه‌ای برای جمع دو عدد 12 بیتی ارائه می‌کنیم (به علت اینکه طول حافظه دستور 32 است و آدرس دهی غیر مستقیم برای استفاده از پوینتر نداریم، نمی‌توانیم خواسته مسئله را برای اعداد 16 بیتی برآورده کنیم، چرا که مقدار حافظه داده در اول کار 0 بوده و برای نمایش نتایج لازم است که مقدار آن غیر صفر باشد.) برای اطمینان از درستی خروجی مدار، یک بار اعداد 201xh و 012xh و بار دیگر اعداد e0fxh و 0fexh را با یک دیگر جمع می‌زنیم. برنامه‌های مورد نظر همراه با این گزارش ارسال شده اند. خروجی این برنامه ها نیز در پایین قابل مشاهده است.

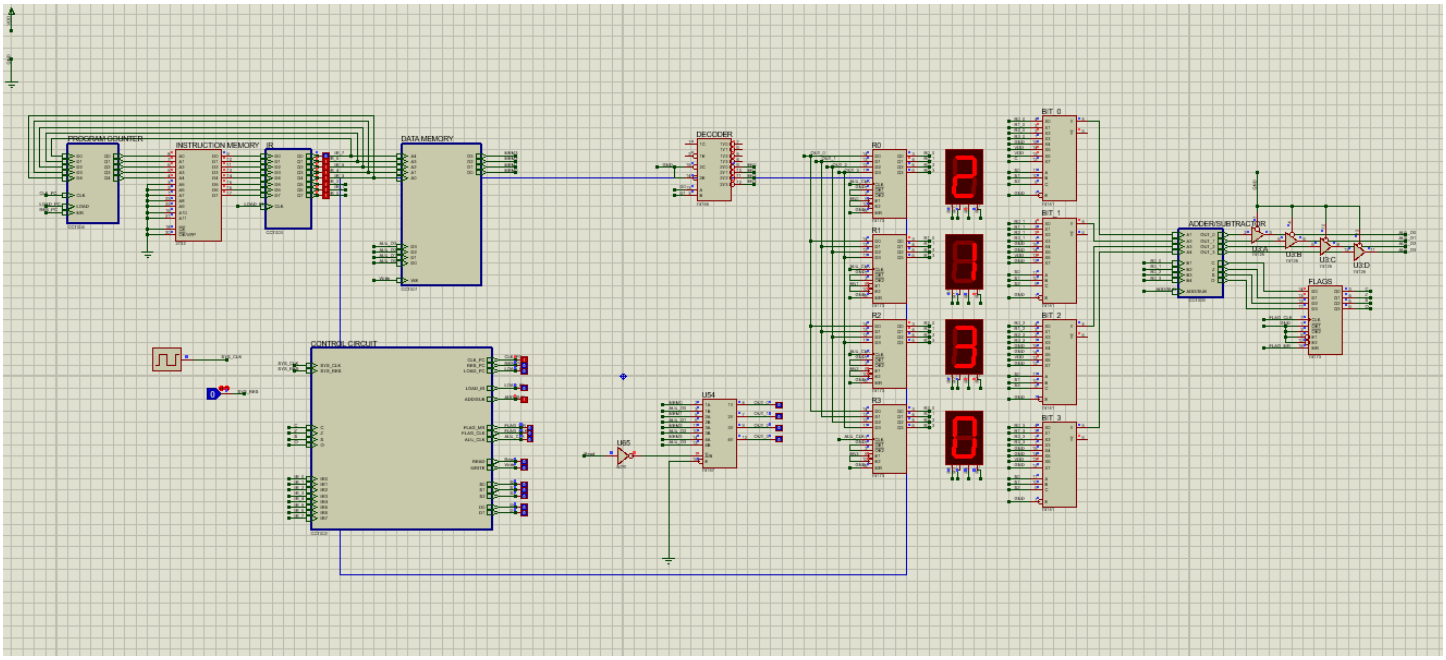
102xh +021xh:

clear flags	20
add r0, 1	05
store 7	67
store e	6e
add r0, 1	05
store 5	65
store f	6f
clear flags	20
load 7	47
add r1, 0	0C
load 15	4F
add r0, carry	07
add r0, r1	01
str 23	77
load 6	46
add r1, carry	0F
load 14	4E
add r0, carry	07
add r0, r1	01
str 22	76
load 5	45
add r1, carry	0F
load 13	4D
add r0, carry	07
add r0, r1	01
str 21	75
load 23	57
add r2, 0	14
load 22	56
add r1, 0	0C
load 21	55
JUMP HERE	F7

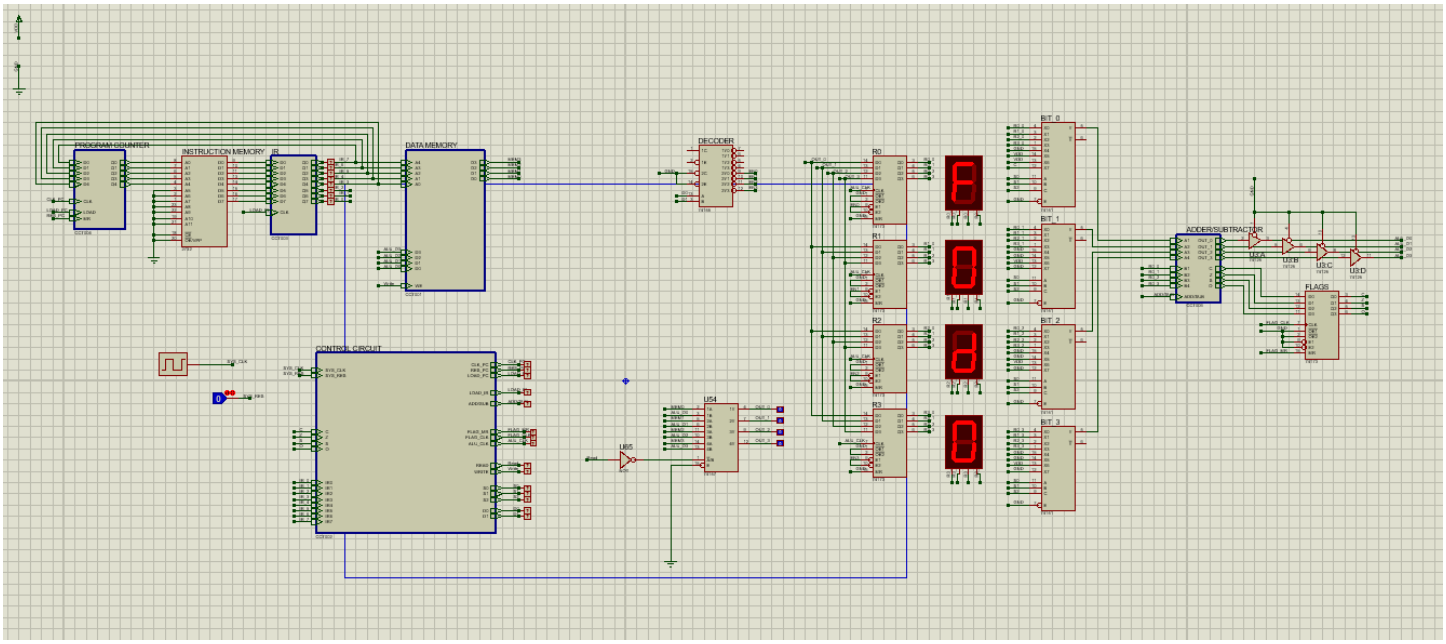
E0Fxh +0EFxh:

clear flags	20
sub r0, 1	05
store 7	67
store e	6e
sub r0, 1	05
store 5	65
store f	6f
clear flags	20
load 7	47
add r1, 0	0C
load 15	4F
add r0, carry	07
add r0, r1	01
str 23	77
load 6	46
add r1, carry	0F
load 14	4E
add r0, carry	07
add r0, r1	01
str 22	76
load 5	45
add r1, carry	0F
load 13	4D
add r0, carry	07
add r0, r1	01
str 21	75
load 23	57
add r2, 0	14
load 22	56
add r1, 0	0C
load 21	55
JUMP HERE	F7

حافظه داده و پرش



می‌بینیم حاصل جمع اول همان عدد مورد انتظار ما (213xh) می باشد.



می‌بینیم حاصل جمع دوم نیز همان عد مورد نظر ماست (F0Dxh) و رقم نقلی به درستی منتقل شده است.

نکته: برنامه‌های به کار برده شده در این گزارش در فایل‌های fib.hex، small_sum.hex، large_sum.hex پیوست شده اند.