به نام خدا

آزمایشگاه معماری کامپیوتر

گزارش کار هفتم

استفاده از حافظه داده و دستورات پرش

دانشکده مهندسی کامپیوتر

دانشگاه صنعتی شریف

پاییز ۱۴۰۱

استاد:

حمید سربازی آزاد

دستیار آموزشی:

عطیه یونسی

نویسندگان:

احمدرضا خناری

نگار عسکری

علی نظری

فهرست

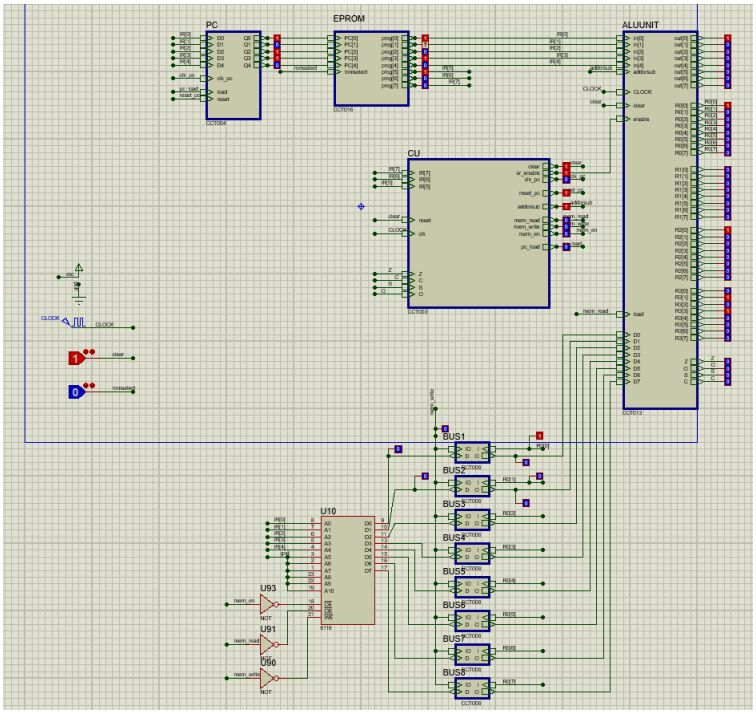
مقدمه

در آزمایش قبل از ریجیستر و alu طراحی شده ی پردازنده ی خود استفاده کرده و دستورات IR را به صورت متوالی از روی یک رام لود کردیم. دستورات تنها شامل جمع و تفریق بودند. حال دستورات پرش و حافظه را میخواهیم اضافه کنیم. مدار در نهایت مطابق شمای تعریف شده در دستور کار خواهد شد.

گزارش آزمایش

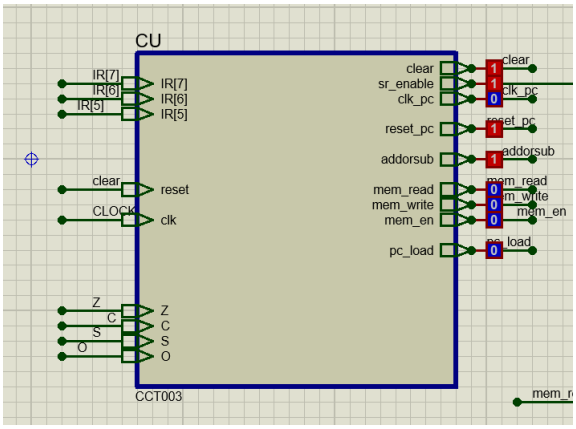
بخش ۱. شرح مدار

شمای کلی مدار به شکل زیر است:

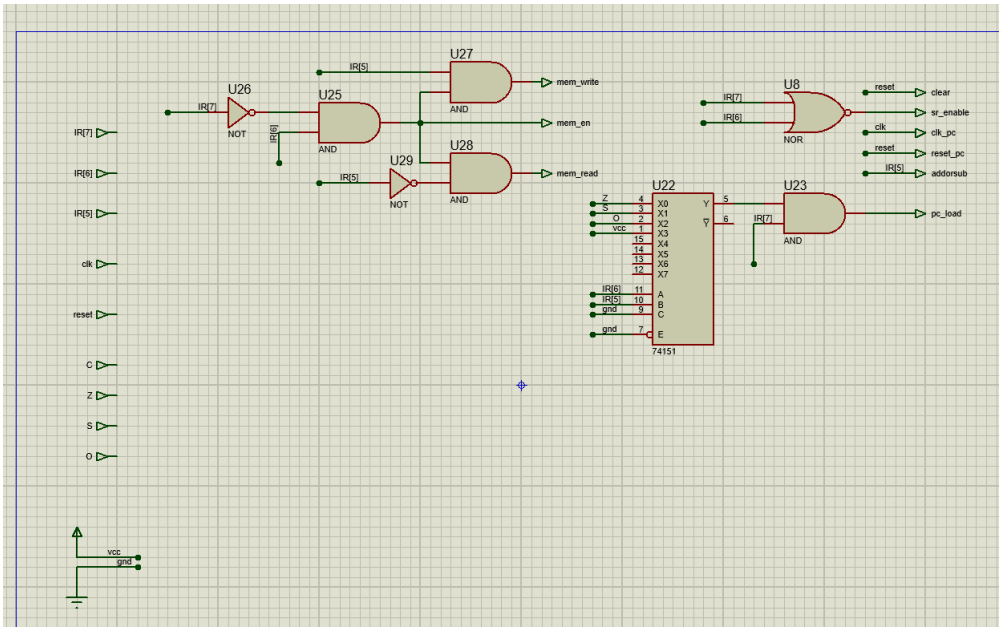


شکل ۱: شمای کلی

نخست بخش Control Unit و شمای درونی آن را می بینیم:

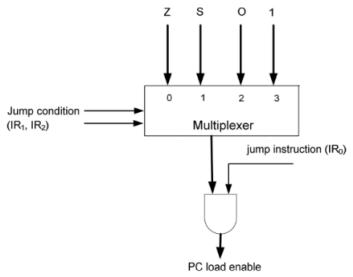


شکل ۲: واحد کنترل



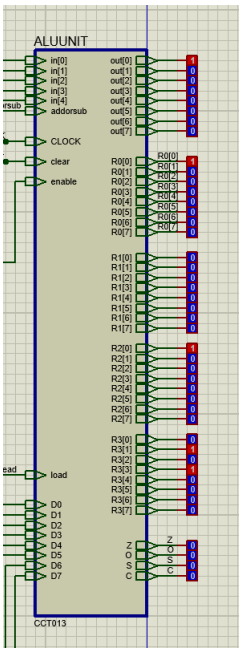
شکل ۳: شمای واحد کنترل

خب CU باید با توجه به حالت سیستم که به کمک ورودی هایی نظیر ۳ بیت اول IR و وضعیت ریجیسترهای C O Z S توسط یک منطق ترکیبی خروجی را مشخص کند، برای حالت جامپ که سیگنال لود فعال می شود از مدار زیر استفاده کردیم، سایر مدارها منطق ساده ای دارند که با توجه به شکل قابل تشخیص می باشد.



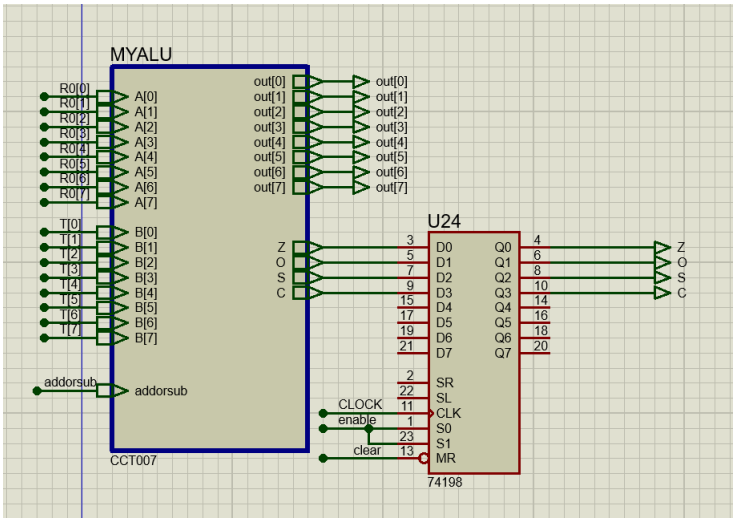
شکل ۴: نوع مشخص شدن خروجی توسط واحد کنترل

بخش رجیسترها و alu هم که به شکل زیر است:

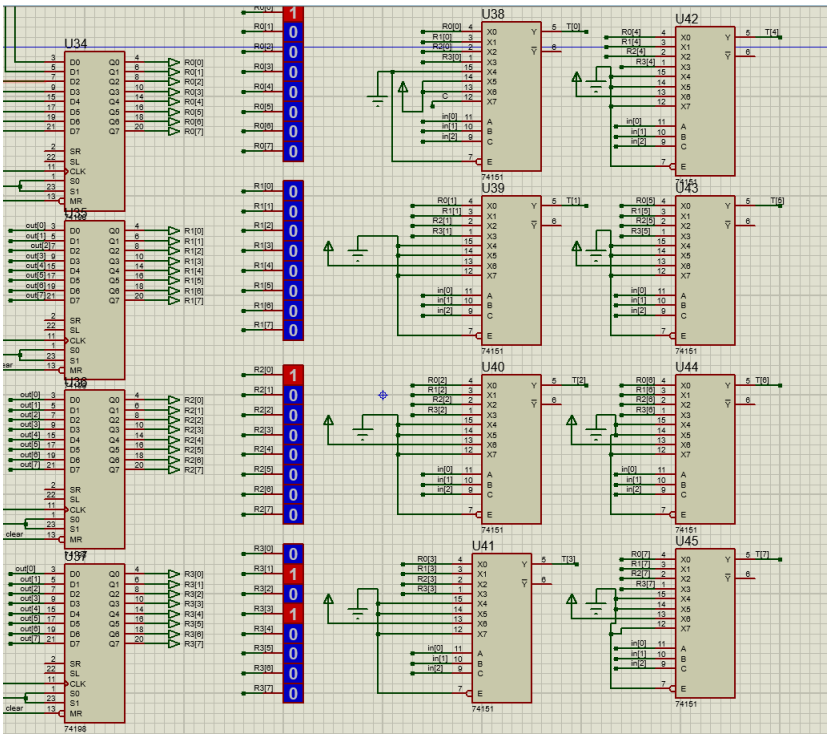


شکل ۵: رجیسترها و alu

در اینجا نسبت به مدار قبل دچار تغییراتی شدیم، اولا Z و O و S و C اضافه شدند. مدارهای ترکیبی آنها و ریجیستر مربوط به آن ها در شکل زیر قابل مشاهده است.

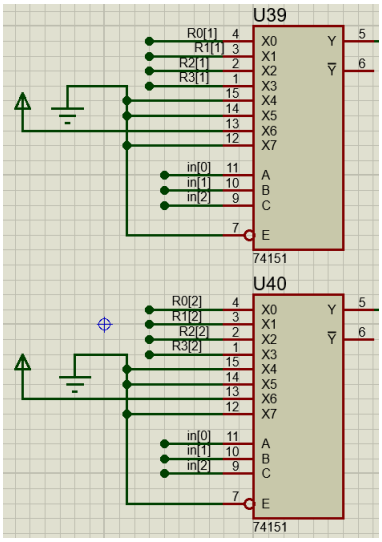


شکل ۶: بخش ALU



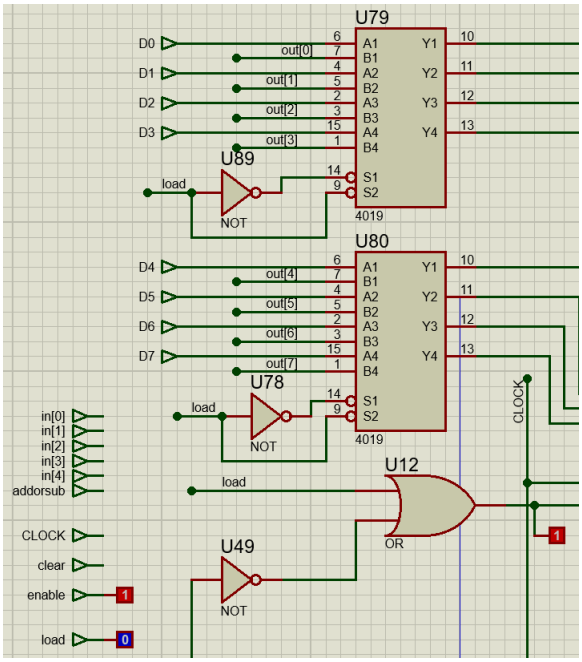
شکل ۷: بخش رجیسترها

هم چنین مقدار کری نیز ممکن است در عملیات ها ظاهر شود. در شکل زیر c همان carry است:



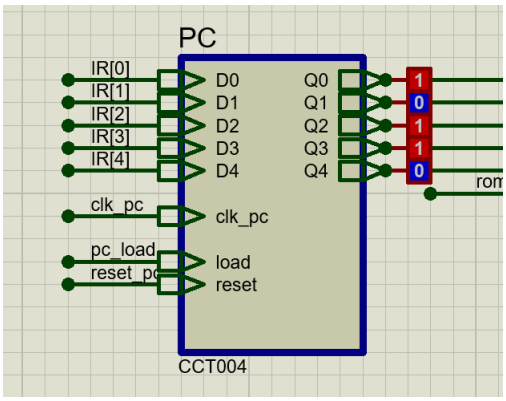
شکل ۸: ایجاد شدن carry

در اینجا دیده می شود که بیت اول مربوط به ۸مین ورودی ماکس برابر C و برای سایر بیت های مربوط به ۸مین ورودی بیت مربوطه ۰ می باشد.
در آخر برای لود از حافظه نیاز به لود مستقیم به ریجیستر ۰ داریم که هنگام read-memory رخ می دهد. شمای جدید ریجیستر اول در ادامه امده است.

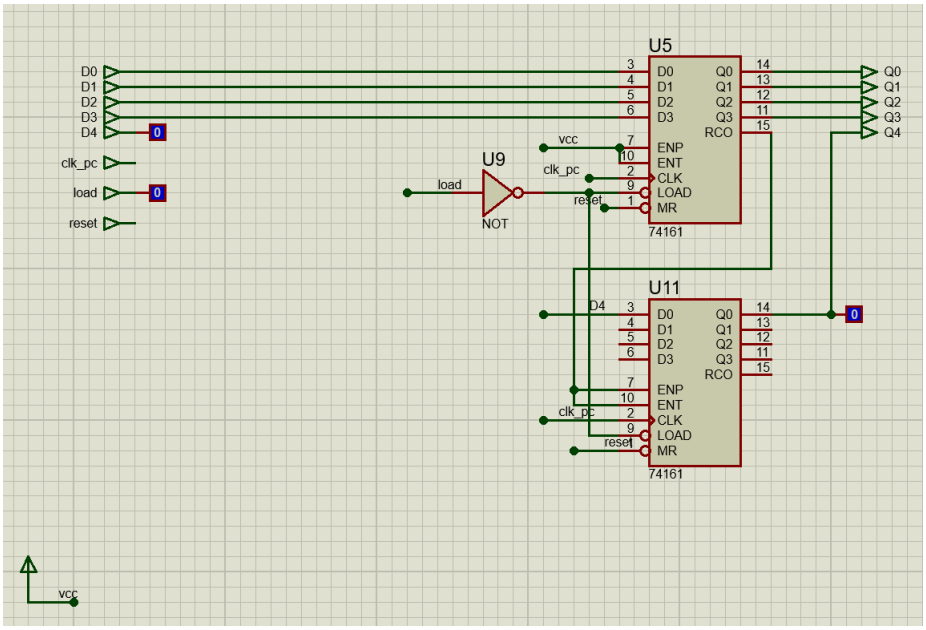


شکل ۹: مشخص شدن load

بخش PC هم به شکل زیر است:

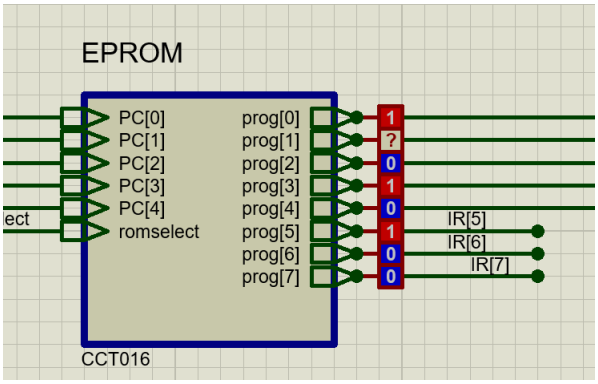


شکل ۱۰: شمای بیرونی PC

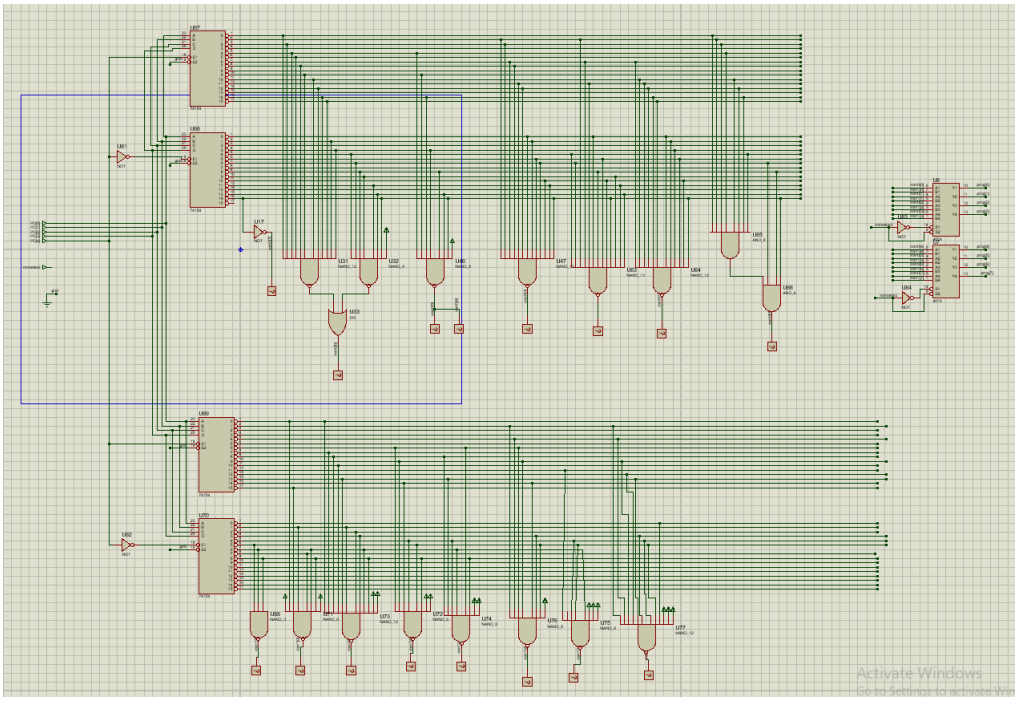


شکل ۱۱: شمای درونی PC

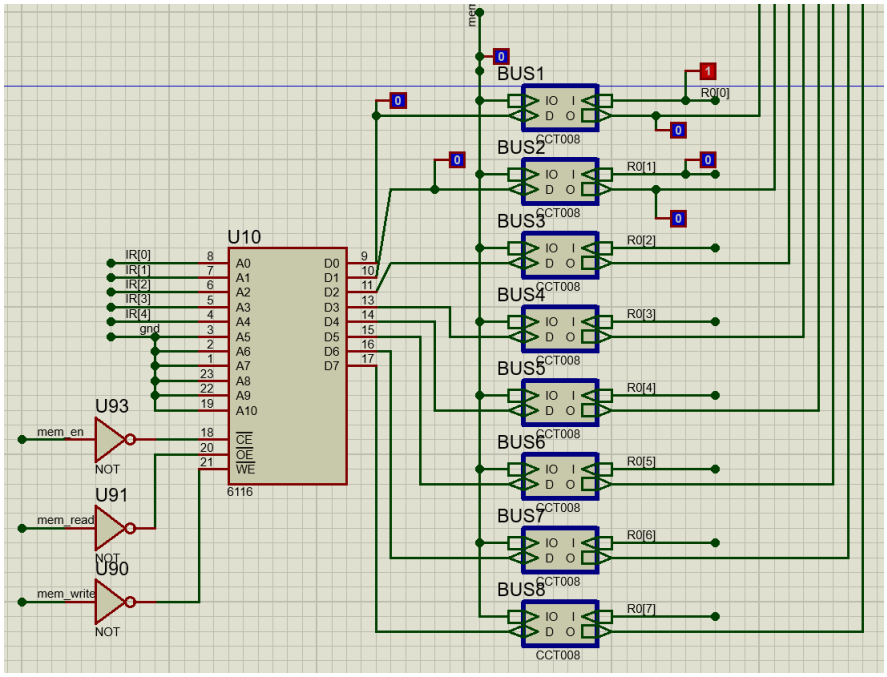
حال به بخش حافظه می رسیم. برای حافظه از تراشه ی ۶۱۱۶ استفاده کردیم، این تراشه یک خروجی I/O دارد که با توجه به سیگنالهای ورودی می تواند عملیاته ای read یا write را انجام دهد. در صورت فعال بودن WE تراشه عملیات رایت، در صورت فعال بودن OE تراشه عملیات read را انجام می دهد که آدرس مربوطه به کمک IR مشخص می شود. از آنجایی که تراشه ورودی IO دارد و همزمان باید به لود ریجیستر ها و خروجی R۰ متصل باشد نیاز به bus داریم که به صورت زیر پیادهسازی شده است.



شکل ۱۲: بخش حافظه



شکل ۱۳: بخش باس ها



شکل ۱۴: کنترل کننده

بخش ۲. تست مدار

از آنجا که این آزمایش کد اسمبلی مخصوص به خود را دارد، نیاز است که آن را هم مشخص کنیم برایش که سعی شده کامنت ها گذاشته شود تا واضح تر باشد. کد اصلی فیبوناچی به شکل زیر است:

```
clear_signal

save r0 to M[0]    mem[00]=01_1_00000
r0 = r0 + 1      mem[01]=00_0_00_101
r0 = r0 + r0       mem[02]=00_0_00_000
r0 = r0 + r0       mem[03]=00_0_00_000
r0 = r0 + 1      mem[04]=00_0_00_101
save r0 to M[1]      mem[05]=01_1_00001
//5 made

r3 = r0 + 0      mem[06]=00_0_11_100
r0 = r0 - r0       mem[07]=00_1_00_000
r1 = r0 - r0       mem[08]=00_1_01_000
r2 = r0 + 1      mem[09]=00_0_10_101

//label
r0 = r0 - r0       mem[10]=00_1_00_000
r0 = r0 + r2       mem[11]=00_0_00_010
r0 = r1 + r0       mem[12]=00_0_00_001
r1 = r0 - r1       mem[13]=00_1_01_001
r2 = r0 + 0      mem[14]=00_0_10_100
load M[0] to r0      mem[15]=01_0_00000
 r0 = r0 + r1      mem[16]=00_0_00_001
save r0 to M[0]      mem[17]=01_1_00000
r0 = r0 - r0       mem[18]=00_1_00_000
r0 = r0 - 1      mem[19]=00_1_00_101
r3 = r0 + r3       mem[20]=00_0_11_011
jump if z = 1 to +2    mem[21]=1_00_10111
jump to label      mem[22]=1_11_01010
//sigma_fib[5] calculated

//load 5
load M[1] to r0      mem[23]=01_0_00001
r3 = r0 + r0       mem[24]=00_0_11_000
//10 made

load M[0] to r0      mem[25]=01_0_00000
r1 = r0 - r3       mem[26]=00_1_01_011
```

```
jump to +4 if sign     mem[27]=1_01_11111

// lower than 10 section
r0 = r0 - r0       mem[28]=00_1_00_000
r0 = r0 + 1      mem[29]=00_0_00_101
jump + 3       mem[30]=1_11_33

// 10 - 15
r1 = r0 + 0      mem[31]=00_0_01_100
r0 = r0 - r0       mem[32]=00_1_00_000


jump to here       mem[33]=1_11_33
//1 + 1 + 2 + 3 + 5 = 12
```

و کد اسمبلی مربوطه به جمع دو عدد ۳۲ بیتی به شکل زیر است:

```
clear signal

Load M[0] to r0   mem[0] = 01_0_00000
r1 = r0 + 0   mem[1] = 00_0_01_100
Load M[8] to r0   mem[2] = 01_0_01000
r0 = r1 + r0    mem[3] = 00_0_00_001
r2 = r0 - c   mem[4] = 00_1_10_111
r2 = r0 - r2    mem[5] = 00_1_10_010
save r0 to M[16]  mem[6] = 01_1_10000

load M[1] to r0   mem[7] = 01_0_00001
r1 = r0 + r2    mem[8] = 00_0_01_010
r3 = r0 - c   mem[9] = 00_1_11_111
load M[9] to r0   mem[10] = 01_0_01001
r0 = r0 + r1    mem[11] = 00_0_00_001
save r0 to M[17]  mem[12] = 01_1_10001
load M[1] to r0   mem[13] = 01_0_00001
r3 = r0 - c   mem[14] = 00_1_11_111
r2 = r0 - r3    mem[15] = 00_1_10_011

load M[2] to r0   mem[16] = 01_0_00010
r1 = r0 + r2    mem[17] = 00_0_01_010
r3 = r0 - c   mem[18] = 00_1_11_111
load M[10] to r0  mem[19] = 01_0_01010
r0 = r0 + r1    mem[20] = 00_0_00_001
save r0 to M[18]  mem[21] = 01_1_10010
load M[2] to r0   mem[22] = 01_0_00010
```

```
r3 = r0 - c    mem[23] = 00_1_11_111
r2 = r0 - r3     mem[24] = 00_1_10_011


load M[3] to r0    mem[25] = 01_0_00011
r1 = r0 + 0    mem[26] = 00_0_01_100
load M[11] to r0   mem[27] = 01_0_01011
r0 = r0 + r1     mem[28] = 00_0_00_001
r0 = r0 + r2     mem[29] = 00_0_00_010
save r0 t0 M[19]   mem[30] = 01_1_10011

jump here    mem[31] = 1_11_11111
```

در حالت جمع دو عدد سعی شده است از اشتباه پرتکرار زیر دوری شود:
جمع یک عدد با کریه ای قبلی ممکن است کری بدهد و باید این حالت نیز پردازش شود. کد این قسمت به خاطر بهینه سازی های مربوط به جا شدن در حافظه بسیار پیچیده گردیده است. توجه داشته باشید که در این قسمت نمی توانستیم از دستورات پرش استفاده کنیم، برای استفاده از این دستورات دو حالت وجود داشت، جمع دو ۸ بیت را یک تابع در نظر بگیریم، در این صورت نیاز به دستور jump jr داشتیم ، در حالت دوم که for زدن بود نیاز داشتیم که mem[i] را بخوانیم و بنویسیم اما در این حالت به علت اینکه mem تنها از مقادیر کانستنت لود میشود نمی توانیم این کار را بکنیم.
نحوه استفاده از مدار هم به شکل زیر است:
برای استفاده از مدار ابتدا به کمک کدromselect دلخواه خود را انتخاب کنید. رام سلکت صفر، کد مربوط به جمع دو عدد ۳۲ بیتی به صورت فلت و رام سلکت یک، مجموع ۱۰ عدد اول فیبوناچی با شروع از دو جمله ی اول ۱ و ۱ را انتخاب می کند. خروجی مدار در حالت فیبوناچی در ریجیستر اول و در حالت دوم در رم ذخیره می گردد. برای بازسازی جمع دو عدد باید در pc هایی که save صورت می گیرد خروجی r۰ را مشاهده نمایید. به علت کمبود جا در رام امکان لود وجود نداشت. در صورت کار نکردن صحیح مدار از clear برای ری استارت استفاده کنید.

نتیجه‌گیری

پس در این بخش مداری که در آزمایش های قبلی ساخته بودیم را کامل کردیم و قابلیت های پرش و خواندن از حافظه هم به آن اضافه کردیم و دو برنامه نمونه را هم می توانیم روی آن اجرا کنیم.