بنام خدا

شبیه‌سازی کامپیوتر مبنا

بر اساس کتاب‌معماری‌کامپیوتر، نوشته‌ی موریس‌مانو

استاد راهنما:

دکتر گلی

طراحان :

معین سلیمی سرتختی

محمدرضا رهنما فلاورجانی

محمدجواد ملکی کهکی

بهار ۱۳۹۴

فهرست مطالب

مقدمه

هدف از این پروژه ، شبیه‌سازی کامپیوتری است که بر اساس کامپیوتر مبنای فصل پنجم کتاب "معماری کامپیوتر" (نوشته‌ی موریس مانو) عمل کند.این پروژه با استفاده از نرم افزار **پروتئوس (Proteus)** طراحی شده است

بخش‌های اصلی تشکیل دهنده‌ی کامپیوتر عبارت اند از:

۱) گذرگاه مشترک برای تبادل داده (باس)
۲) واحد محاسباتی و منطقی ALU به همراه واحد شیفت دهنده
۳) واحد کنترل
۴) حافظه
۵) ثبات‌ها
۶) واحد نمایش

این کامپیوتر قادر به انجام همه‌ی اعمالی است که کامپیوتر فصل پنجم کتاب مذکور پشتیبانی می‌کند

Moeinsalimi.sartakhti@gmail.com

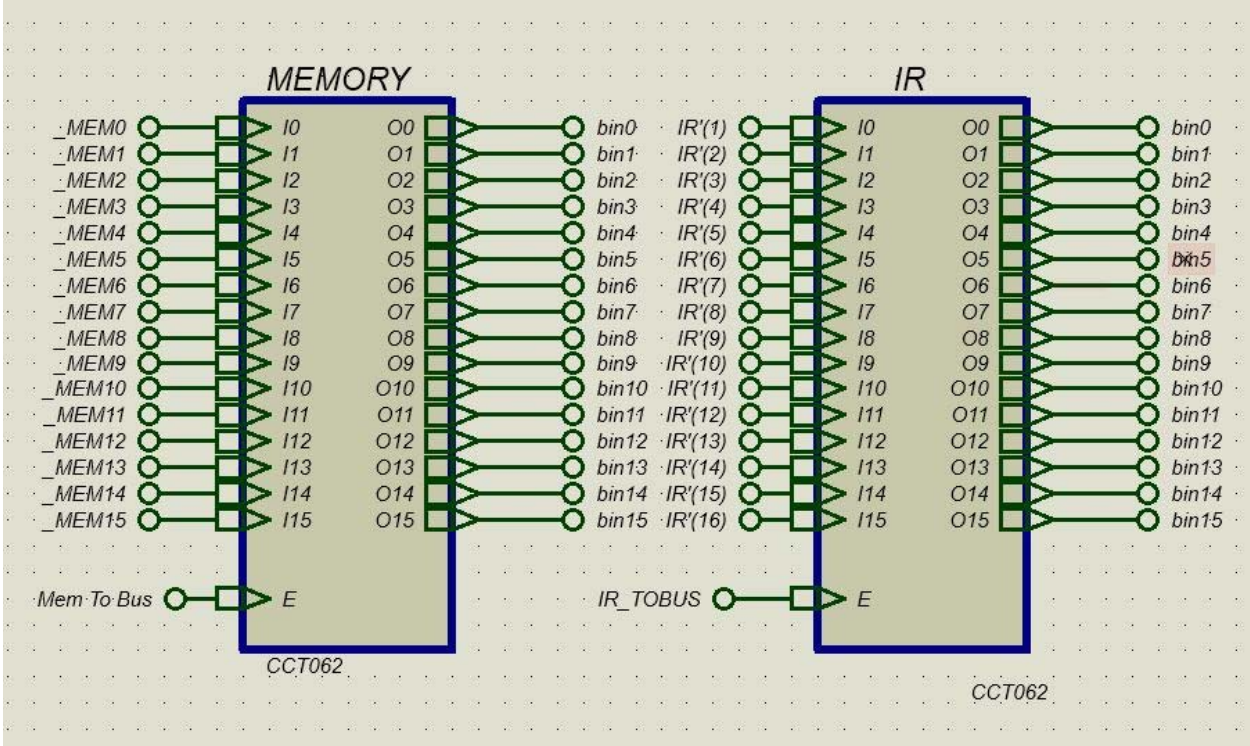
Mohammadreza.rahnamaa@gmail.com

Mjmaleki94@gmail.com

گذرگاه مشترک برای تبادل داده (باس)

در واقع در این پروژه باس یک چیز فیزیکی نیست بلکه یک سری ترمینال هایی است که به ورودی همه‌ی ثبات‌ها وصل شده‌اند و از آن سو نیز باید کنترل کنیم که چه موقع ثبات ها و حافظه می‌توانند اطلاعات خود را روی این ترمینال‌ها بگذارند. این کنترل کردن ، بوسیله‌ی تعدادی بافر و یک سری تابع های کنترلی (که در واحد کنترل ایجاد شده‌اند) انجام شده است.

بعنوان نمونه دوتا از بافرها که کنترل می‌کنند چه موقع باید اطلاعات از حافظه و IR به باس منتقل شود را در شکل زیر می بینید. لازم به ذکر است که تابع‌های کنترلی و ورودی فعالساز این بافرها در واحد کنترل تولید می‌شوند.



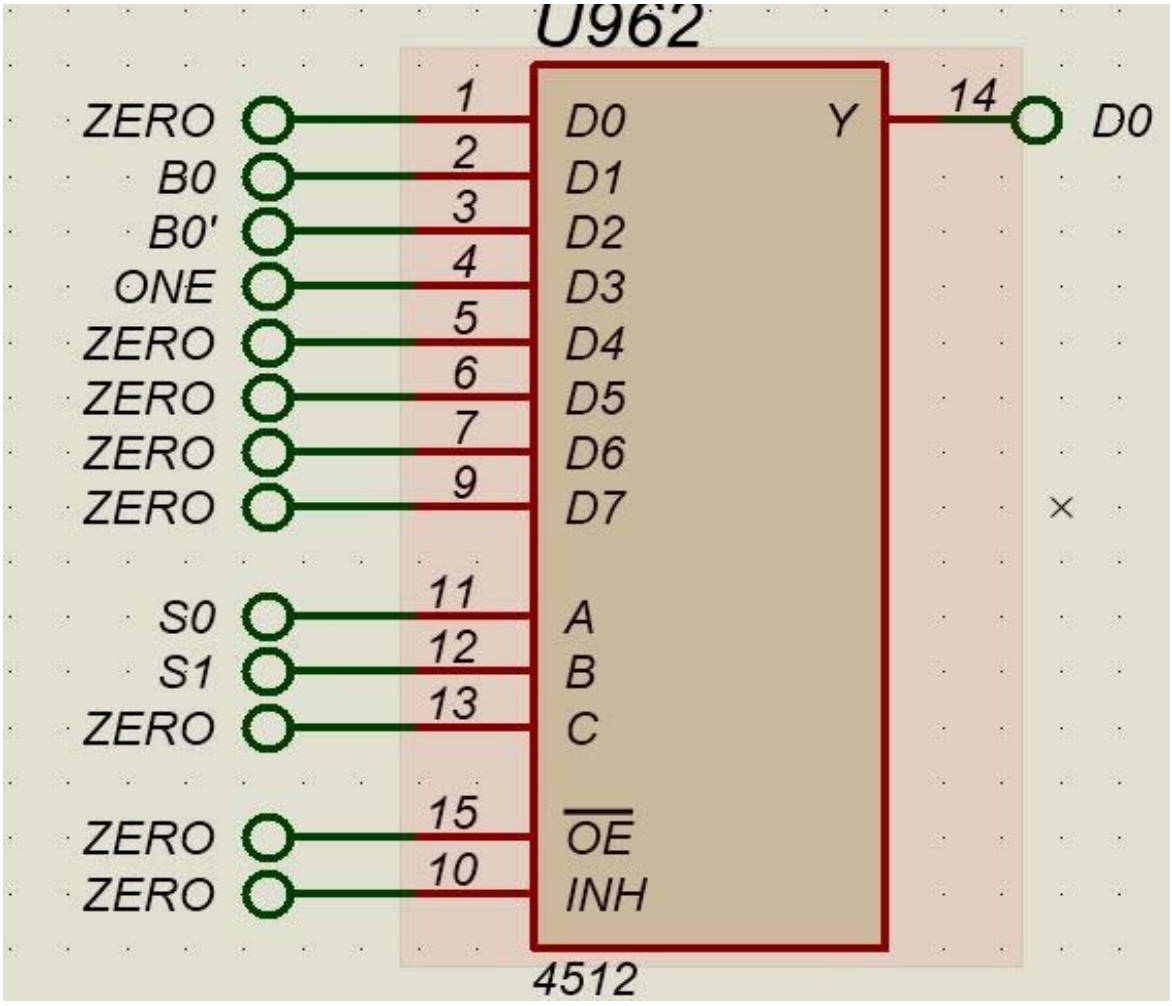
برای بقیه‌ی ثبات‌ها (البته بجز ثبات‌های Input,Output) نیز به همین ترتیب یک بافر کنترلی وجود دارد

واحد محاسباتی و منطقی ALU به همراه واحد شیفت دهنده

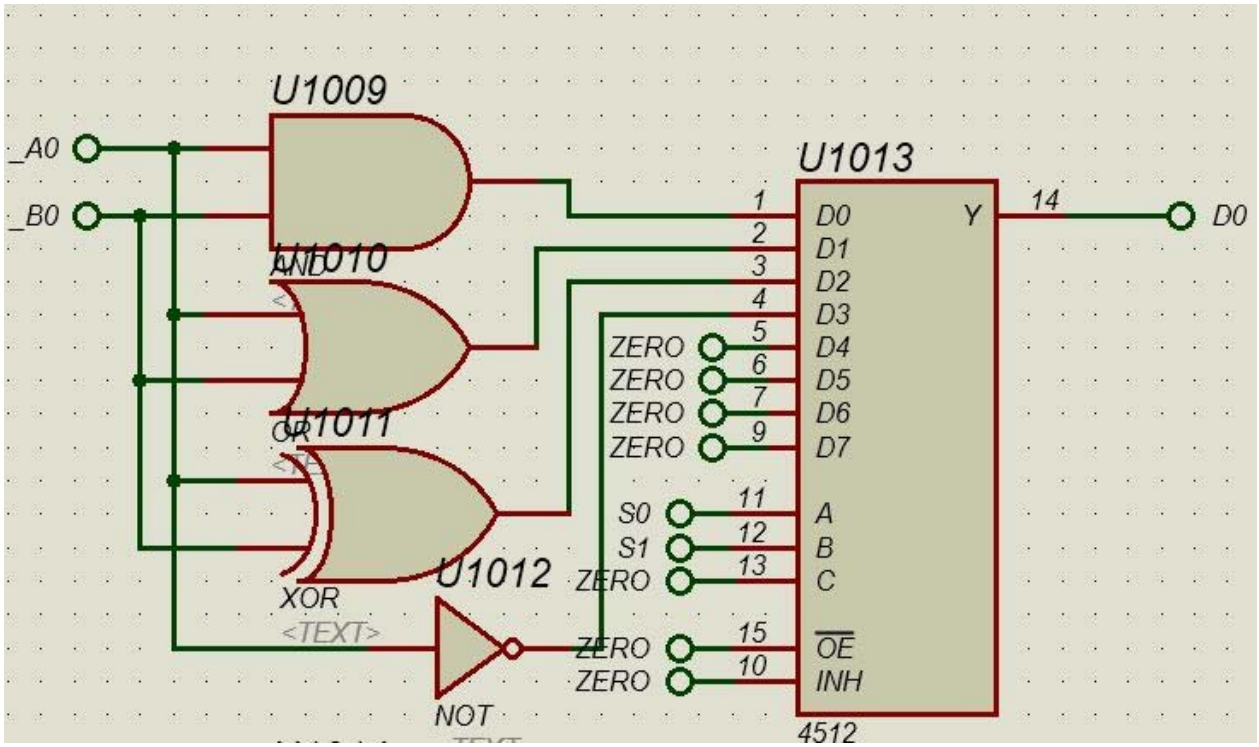
سعی کردیم کل کامپیوتر بر اساس فصل پنجم طراحی شود اما ALU یک استثناء است. ALU این کامپیوتر بر اساس فصل چهارم طراحی شده . تنها تفاوت میان ALU فصل چهارم و پنجم در تعداد دستوراتی است که می‌توانند انجام دهند بگونه‌ای که اعمالی که ALU فصل چهارم قادر به انجامشان است بیشتر از اعمالی است که ALU فصل پنجم قادر به انجام دادن است. بنابراین ALU این کامپیوتر بر اساس فصل چهارم طراحی شد تا بتوان دستورات بیشتری را داشت.

توجه شود که کامپیوتر فعلی فقط می‌تواند دستورات فصل پنجم را پیاده سازی کند . برای مثال گرچه در ALU فصل چهارم اعمالی مثل OR کردن یا XOR کردن وجود دارد ، اما در واحد کنترل این دستورات پشتیبانی نشده و اصلا پیاده نشده‌اند. درواقع اگر در واحد کنترل دستورات OR و XOR و مشابه آن‌ها را اضافه کنیم در آن صورت می توان در این کامپیوتر اینگونه دستورات را نیز شبیه سازی و پیاده کرد

در شکل زیر یک طبقه از واحد محاسباتی ALU را مشاهده می‌کنید که با خطوط انتخاب A,B میتوان ۴ حالت مختلف ایجاد کرد

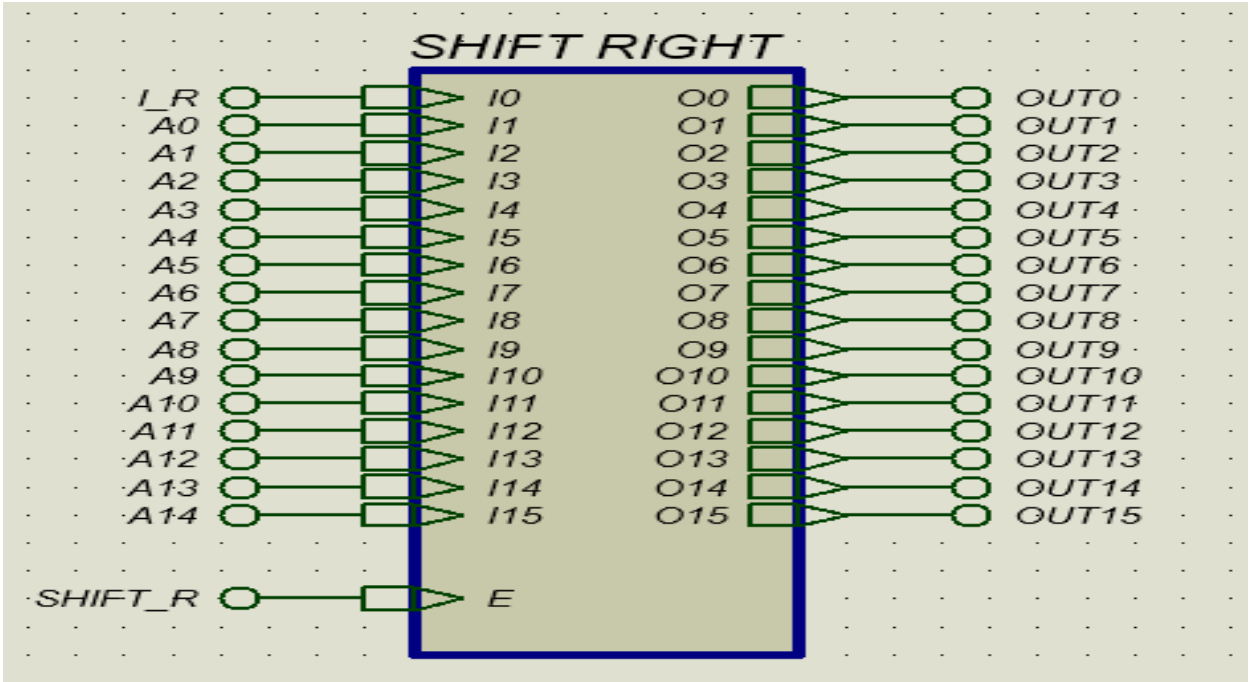


در شکل زیر یک طبقه از واحد منطق ALU را مشاهده می‌کنید



بخش شیفت دهنده

برای انجام عمل شیفت از بافر استفاده کرده‌ایم . در شکل زیر بافری را که برای انجام عمل شیفت به راست طراحی شده مشاهده می‌کنید.لازم به ذکر است که سیگنال فعالساز این بافر از واحد کنترل صادر می‌شود.



ورودی‌های ALU

می‌دانیم که ورودی‌های ALU باید از بین AC,DR,Input انتخاب شوند در حالیکه فقط دو ورودی برای ALU داریم. برای این کار ما یکی از ورودی های ALU را بصورت دائمی به AC وصل کرده‌ایم و برای ورودی دومی ALU از سه بافر کنترلی استفاده کردیم. این بافرها بررسی می‌کنند که آیا ورودی دوم ALU ثبات DR باید باشد یا Input باشد یا مانند ورودی اولی ALU ، بازهم خود AC باشد؟

توجه کنید که هرگاه یکی از این بافرها فعال شود محتوای ثبات مربوطه به ورودی دومی ALU منتقل می شود.

*خطوط فعالساز بافرها ، از واحد کنترلی گرفته شدند.

خروجی‌های ALU

بطور کلی ما در واحد ALU چهار بافر مختلف داریم. دوتا برای شیفت به راست و چپ ، یکی برای انجام اعمال محاسباتی و یکی برای اعمال منطقی. اگر بخواهیم بطور واقعی به این ۳ بخش (واحد محاسبات ، منطق ، شیفت) نگاه کنیم باید ذکر کنیم که همه‌ی این ۳ بخش در هر لحظه ، دارای جواب و خروجی هستند اما با استفاده از این بافرها کنترل می‌کنیم که در هر لحظه کدام جواب بر روی خروجی ALU قرار بگیرد.

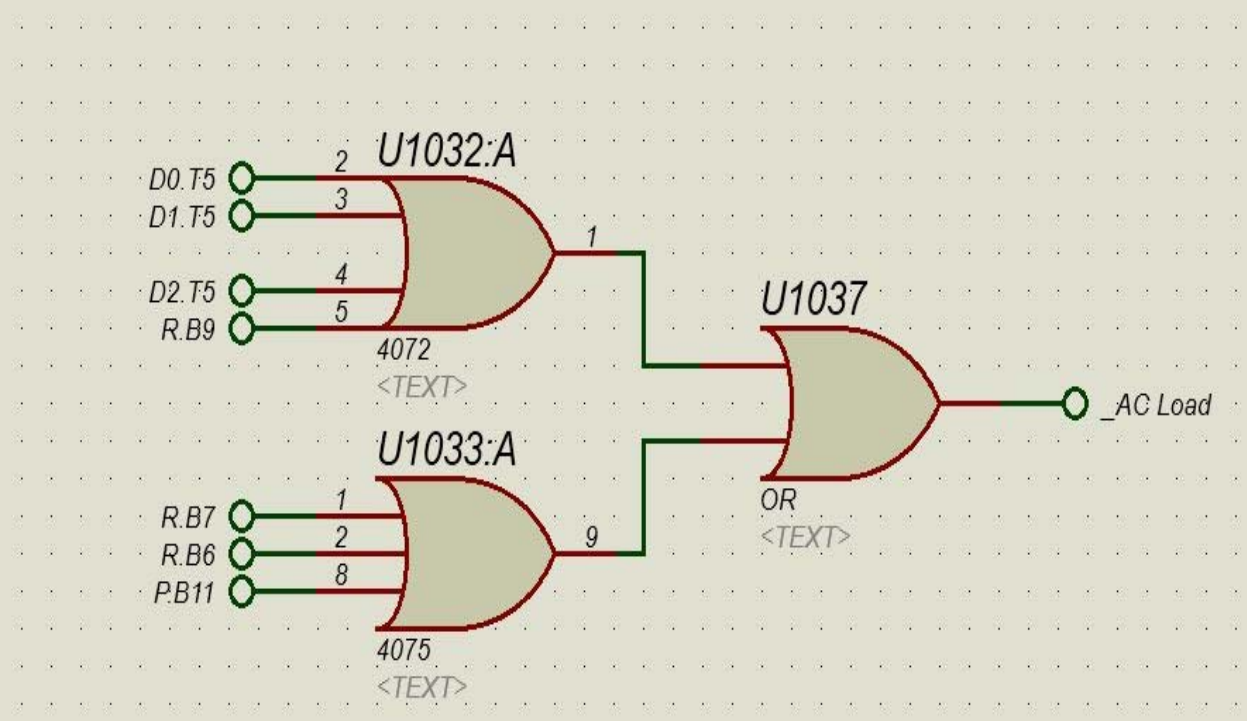
واحد کنترل

فلسفه‌ی بوجود آمدن واحد کنترل این بود که می‌باید همه‌ی قسمت‌ها اعم از واحد حافظه ، ثبات‌ها و واحد ALU در زمان مقرر وظایف مربوط به خود را انجام دهند در واقع اجازه‌ی انجام وظایف قسمت‌های مختلف کامپیوتر ، توسط سیگنال‌های کنترلی در واحد کنترل صادر می‌شود.

بعنوان مثال یکی از وظایف این واحد ، تولید سیگنال‌های کنترلی مربوط به بار شدن ثباتی مثل AC است . برای تولید اینگونه سیگنال‌ها ابتدا باید جدول تابع‌های کنترلی و ریز عملیات کامپیوتر مبنا را تهیه کرد که این جدول در زیر آمده است:

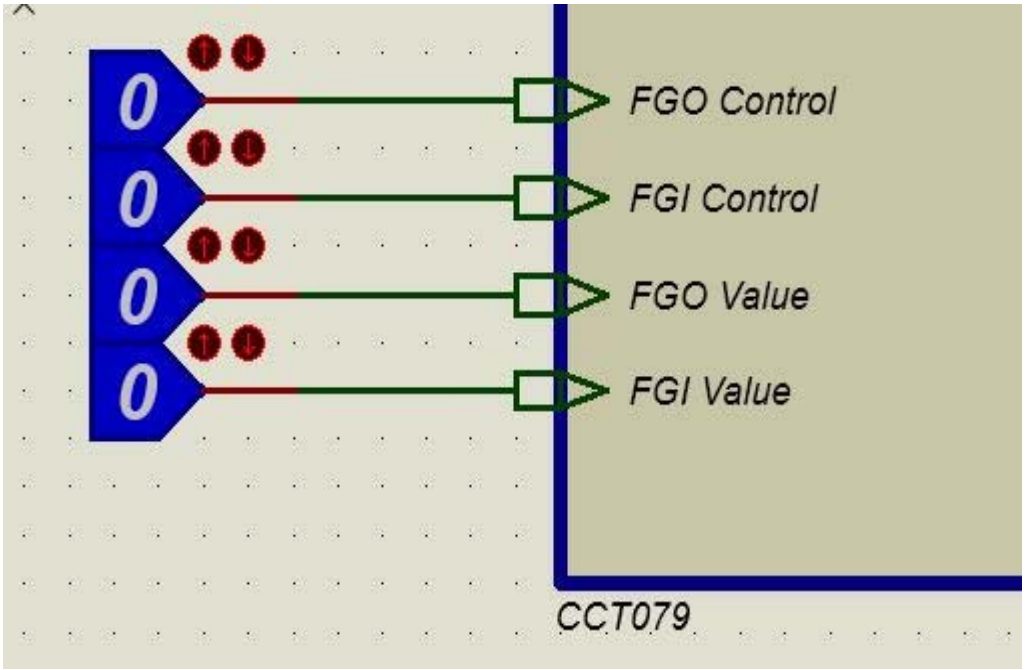
واکشی	$R'T_0$:	$AR \leftarrow PC$
	$R'T_1$:	$IR \leftarrow M[AR], PC \leftarrow PC + 1$
رمزگشایی یا دیکد	$R'T_2$:	$D_0, \ldots, D_7 \leftarrow$ دیکد $IR(12\text{-}14)$, $AR \leftarrow IR(0 - 11), I \leftarrow IR(15)$
غیر مستقیم	D'_7IT_3 :	$AR \leftarrow M[AR]$
وقفه :	$T'_0T'_1T'_2(IEN)(FGI + FGO)$:	$R \leftarrow 1$
	RT_0 :	$AR \leftarrow 0, TR \leftarrow PC$
	RT_1 :	$M[AR] \leftarrow TR, PC \leftarrow 0$
	RT_2 :	$PC \leftarrow PC + 1, IEN \leftarrow 0, R \leftarrow 0, SC \leftarrow 0$
دستورات مراجعه به حافظه :		
AND	D_0T_4 :	$DR \leftarrow M[AR]$
	D_0T_5 :	$AC \leftarrow AC \wedge DR, SC \leftarrow 0$
ADD	D_1T_4 :	$DR \leftarrow M[AR]$
	D_1T_5 :	$AC \leftarrow AC + DR, E \leftarrow C_{out}, SC \leftarrow 0$
LDA	D_2T_4 :	$DR \leftarrow M[AR]$
	D_2T_5 :	$AC \leftarrow DR, SC \leftarrow 0$
STA	D_3T_4 :	$DR \leftarrow AC$
	D_3T_5 :	$M[AR] \leftarrow DR, SC \leftarrow 0$
BUN	D_4T_4 :	$PC \leftarrow AR, SC \leftarrow 0$
BSA	D_5T_4 :	$M[AR] \leftarrow PC, AR \leftarrow AR + 1$
	D_5T_5 :	$PC \leftarrow AR, SC \leftarrow 0$
ISZ	D_6T_4 :	$DR \leftarrow M[AR]$
	D_6T_5 :	$DR \leftarrow DR + 1$
	D_6T_6 :	$M[AR] \leftarrow DR$ $(PC \leftarrow PC + 1), SC \leftarrow 0$ پس $(DR = 0)$ اگر
دستورات مراجعه به ثباتها :	$D_7I'T_3 =$	برای تمام دستورات مراجعه به ثباتها مشترک است r
	$IR(i) =$	$B_i(i = 0, 1, 2, \ldots)$
	r :	$SC \leftarrow 0$
CLA	rB_{11}:	$AC \leftarrow 0$
CLE	rB_{10}:	$E \leftarrow 0$
CMA	rB_9:	$AC \leftarrow \overline{AC}$
CME	rB_8:	$E \leftarrow \overline{E}$
CIR	rB_7:	$AC \leftarrow shr\ AC, AC(15) \leftarrow E, E \leftarrow AC(0)$
CIL	rB_6:	$AC \leftarrow shl\ AC, AC(0) \leftarrow E, E \leftarrow AC(15)$
INC	rB_5:	$AC \leftarrow AC + 1$
SPA	rB_4:	$(PC \leftarrow PC + 1)$ پس $(AC(15) = 0)$ اگر
SNA	rB_3:	$(PC \leftarrow PC + 1)$ پس $(AC(15) = 1)$ اگر
SZA	rB_2:	$(PC \leftarrow PC + 1)$ پس $(AC = 0)$ اگر
SZB	rB_1:	$(PC \leftarrow PC + 1)$ پس $(E = 0)$ اگر
HLT	rB_0:	$S \leftarrow 0$
دستورات ورودی و خروجی :	$D_7IT_3 =$	برای تمام دستورات ورودی‌خروجی مشترک است P
	$IR(i) =$	$B_i(i = 6, 7, 8, 9, 10, 11)$
	P:	$SC \leftarrow 0$
INP	pB_{11}:	$AC(0\text{-}7) \leftarrow INPR, FGI \leftarrow 0$
OUT	pB_{10}:	$OUTR \leftarrow AC(0\text{-}7), FGO \leftarrow 0$
SKI	pB_9:	$(PC \leftarrow PC + 1)$ پس $(FGI = 1)$ اگر
SKO	pB_8:	$(PC \leftarrow PC + 1)$ پس $(FGO = 1)$ اگر
ION	pB_7:	$IEN \leftarrow 1$
IOF	pB_6:	$IEN \leftarrow 0$

اکنون با مراجعه به این جدول می‌توان زمان‌ها و پیش‌شرط‌هایی را که در آن ثبات AC آماده‌ی بار شدن می‌شود پیدا کرد و سپس همه‌ی این حالت‌ها را در بار شدن AC تاثیر داد که ما در واحد کنترل ، بگونه‌ی زیر عمل کرده‌ایم



برای بقیه‌ی ثبات‌ها نیز به همین ترتیب عمل کردیم .

شکل زیر مربوط به بخشی از ورودی های واحد کنترل است که توضیحات مربوط به آن در صفحه‌ی بعد آمده است



بعضی اوقات باید کاربر قادر باشد تا مقدار دلخواهی از ثبات ورودی را به ثبات اکومولیتور منتقل کند. برای این کار باید بتواند بصورت دستی مقدار FGO را تغییر دهد. اگر FGO Control را برابر با یک کنیم ، در آن صورت کنترل FGO از حالت خودکار خارج شده و مقداردهی آن دستی می شود و اکنون می توان به FGO Value مقدار داد. منظور از FGO Value ، خود مقدار FGO است که بصورت دستی قابل تعیین می باشد

برای ثبات Output نیز همین قضایا برقرار است

علاوه بر این ، در واحد کنترل :

- ✓ رمزگشایی و دیکد شدن دستورات انجام می گیرد
- ✓ زمان های T0 تا T15 تولید می شود
- ✓ سیگنال‌های ورودی فلیپ فلاپ‌ها تولید می‌شود
- ✓ سیگنال‌های کنترلی جهت خواندن از حافظه و نوشتن در آن ایجاد می‌شود
- ✓ و ....

حافظه

واحد حافظه از دو RAM ، دو ROM ، یک شمارنده و دو بافر کنترل کننده تشکیل شده است. حافظه‌ی RAM و ROM هردو دارای ۱۰۲۴ کلمه ی ۸ بیتی هستند که از کنار هم گذاشتن دوتای آنها ، می‌توان حافظه‌ای با ۱۰۲۴ کلمه به طول ۱۶ بیت ایجاد کرد.

استفاده از ROM برای راحتی کار بود .به اینصورت که می‌توان در ROM یک فایل باینری آپلود کرد و سپس اطلاعات ROM را به RAM منتقل کرد . با این کار دیگر لازم نیست که با هر بار روشن کردن کامپیوتر ،یک سری اطلاعات طولانی را بصورت دستی وارد RAM کرد

دلیل وجود شمارنده : برای انتقال اطلاعات از ROM به RAM ابتدا خطوط داده‌ی ROM را به خطوط داده‌ی RAM متصل کردیم سپس باید در هر مرحله ، یک آدرس یکسان به ROM و RAM داد تا اطلاعات ROM از طریق خطوط داده ، به همان آدرس در RAM منتقل شود. ما نیاز داشتیم تا بطور خودکار آدرس ROM و RAM از صفر تا ۱۰۲۳ افزایش یابد و همزمان که این آدرس‌ها یکی یکی زیاد می‌شود داده‌های ROM به RAM منتقل شود.وظیفه‌ی شمارنده تولید این آدرس هاست که این شمارنده می‌تواند با کلاک کار کند یعنی با هر بار کلاک دادن به شمارنده می‌توان یک واحد در مقدار آدرس ROM و RAM افزایش شاهد بود.

البته لازم به ذکر است که این شمارنده به جای کار با کلاک با یک مولد(Generator) کار می‌کند . این مولد در هر ثانیه ، ۳۰ کلاک تولید می‌کند و در نتیجه در زمان خیلی کمی می‌توان کل اطلاعات ROM را به RAM منتقل کرد !

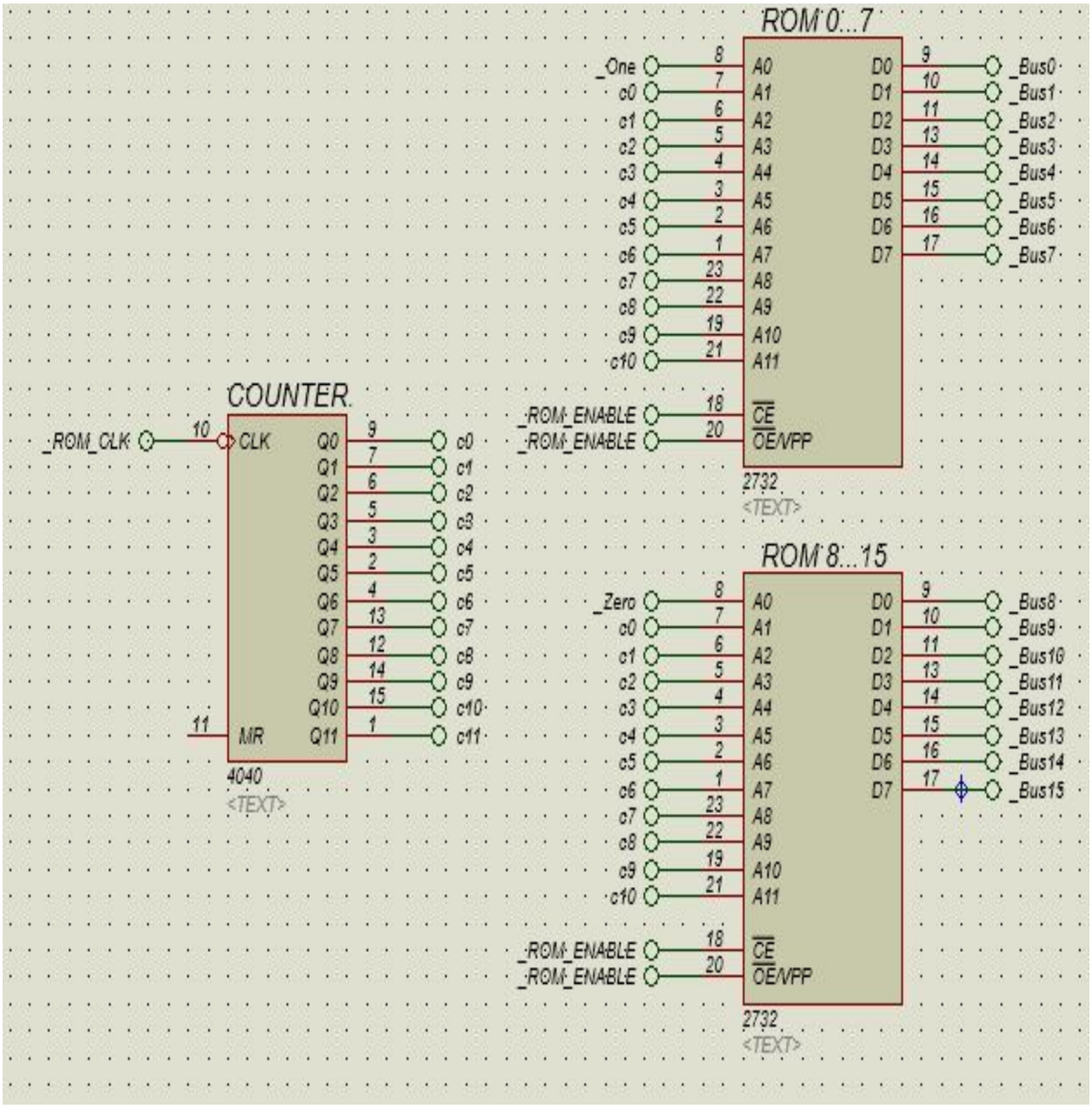
در مورد آدرس‌های ROM بگونه‌ای عمل کرده‌ایم که همیشه یکی از ROM ها زوج آدرس‌ها و دیگری فرد آدرس‌ها را نگهداری کنند تا بدین ترتیب همواره دو کلمه‌ی پشت سر هم را بتوانیم به RAM منتقل کنیم.

*خروجی ROM به RAM متصل شده است که البته در شکل زیر RAM ها وجود ندارند

*برای نوشتن در RAM ، باید ابتدا یک آدرس بر روی خطوط آدرس RAM قرار دهیم سپس داده‌ی متناظر را نیز آماده کنیم و آن را بر روی خطوط داده‌ی RAM بگذاریم. اکنون برای نوشتن در RAM کافیست سیگنال Write را از **۱ به صفر تغییر داد**. با این کار ، داده‌ی مورد نظر در آدرس مربوطه ذخیره می‌شود

*برای خواندن از RAM ، ابتدا آدرس مورد نظر را باید بر روی خطوط آدرس قرار دهیم سپس با **یک نمودن** خط Read ، داده‌ی مربوطه بر روی خطوط داده‌ی RAM ظاهر می‌گردد

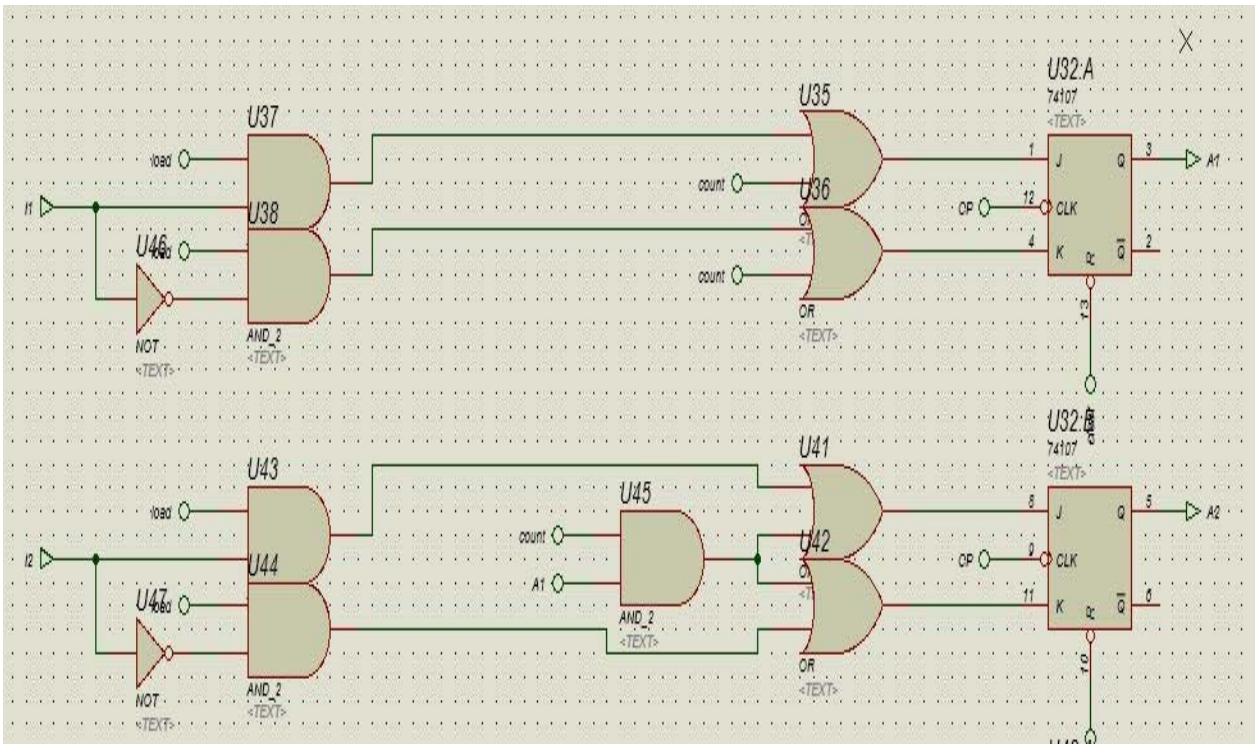
در شکل صفحه‌ی بعد هر دو ROM و شمارنده را در یک تصویر می‌توان مشاهده کرد


ROM 0...7
_One
c0
c1
c2
c3
c4
c5
c6
c7
c8
c9
c10
A0
A1
A2
A3
A4
A5
A6
A7
A8
A9
A10
A11
_ROM_ENABLE
_ROM_ENABLE
CE
OE/VPP
D0
D1
D2
D3
D4
D5
D6
D7
_Bus0
_Bus1
_Bus2
_Bus3
_Bus4
_Bus5
_Bus6
_Bus7
2732
<TEXT>
COUNTER
_ROM_CLK
CLK
MR
Q0
Q1
Q2
Q3
Q4
Q5
Q6
Q7
Q8
Q9
Q10
Q11
c0
c1
c2
c3
c4
c5
c6
c7
c8
c9
c10
c11
4040
<TEXT>
ROM 8...15
_Zero
_Bus8
_Bus9
_Bus10
_Bus11
_Bus12
_Bus13
_Bus14
_Bus15
2732
<TEXT>

ثبات‌ها

در این پروژه ۸ ثبات داریم. ۴ تا از آن ها ۱۶ بیتی هستند که عبارت اند از : AC,DR,IR,TR . ثباتهای AR,PC نیز ۱۲ بیتی هستند همچنین ثباتهای Input , Output هم ۸بیتی می باشند. ساختار داخلی همه‌ی ثباتها یکسان است با این تفاوت که بعضی از آنها ۱۶ بیت ظرفیت دارند بعضی از آنها ۱۲ بیت و برخی نیز ۸ بیت. همه‌ی ثبات‌ها دارای قابلیت بارگذاری ، افزایش و پاک کردن هستند البته برای بعضی ثبات‌ها مانند Input , Output ,IR این قابلیت غیر فعال شده ، زیرا هیچ وقت به آن نیاز ندارند. ثبات‌ها از فلیپ فلاپ ساخته شده اند به همین دلیل است که خروجی آنها با یک کلاک تاخیر قابل مشاهده است. بعنوان مثال اگر قرار باشد در زمان T1 به محتوای AC یک واحد افزوده شود در زمان T2 است که این تغییر در خروجی ثبات ها ایجاد می شود و قابل مشاهده خواهد بود

در شکل زیر بخش کوچکی از ساختار داخلی ثبات‌ها قابل مشاهده است

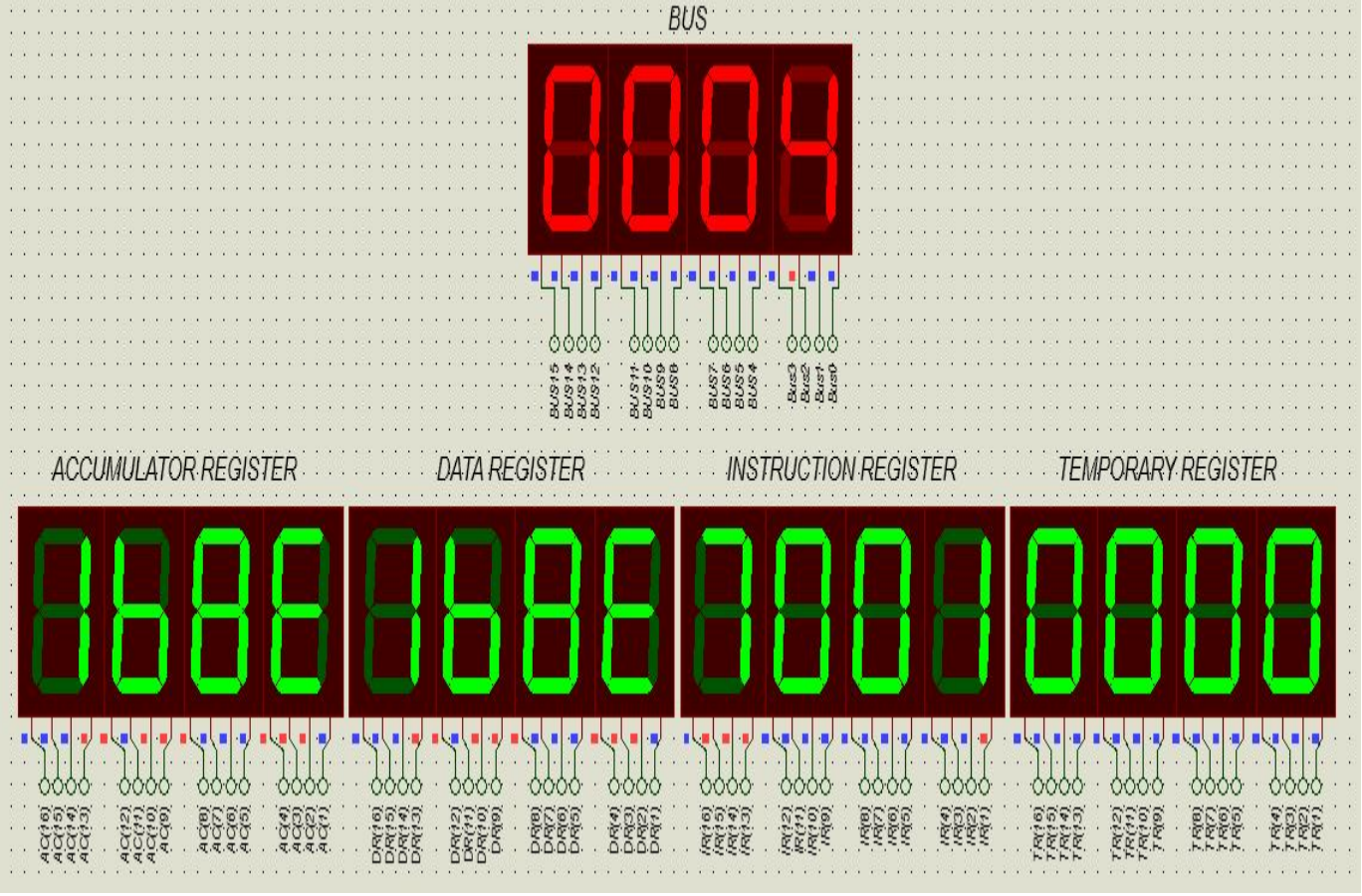


در واقع ما در ساختار داخلی ثبات‌ها ، یک شمارنده پیاده سازی کرده‌ایم که این شمارنده با کلاک ساعت کار می‌کند بطوریکه هرگاه لازم باشد محتوای خودش را پاک می کند هر وقت نیاز باشد شمارش آن متوقف می‌شود و همچنین می‌تواند از هر عدد دلخواه شروع به شمارش کند و نیز یک واحد به خودش اضافه کند. این مجموعه از قابلیت‌ها همان چیزی را ایجاد می‌کند که ما از ثبات انتظار داریم

واحد نمایش

واحد نمایش همانگونه که از نامش پیداست برای نمایش خروجی قسمت‌های مختلف کامپیوتر به کار می‌رود .

در شکل زیر بخشی از واحد نمایشِ اصلی را مشاهده می‌کنید



هر رقم در واحد نمایش بالا ، در مبنای ۱۶ می باشد

به غیر از این واحد نمایش که ما آن را واحد نمایش اصلی نامیده‌ایم خروجی و ورودی ثبات‌ها ، واحد کنترل ، واحد ALU و واحد حافظه نیز همگی قادر به نمایش خروجی خود می‌باشند که نمایش آن‌ها در مبنای ۲ هستند

معرفی نرم افزار 010 Editor

واضح است فایلی که ما به ROM می‌دهیم تبدیل به کد اسکی و در نهایت تبدیل به کد باینری می‌شود. از طرف دیگر ما می‌باید ابتدا دستورات را به باینری بنویسیم و سپس کاراکتر متناسب با آن را پیدا کنیم و آن را در یک فایل باینری ذخیره کنیم که پیدا کردن کاراکترهای مربوط به کدهای باینری مختلف بسیار مشکل است.

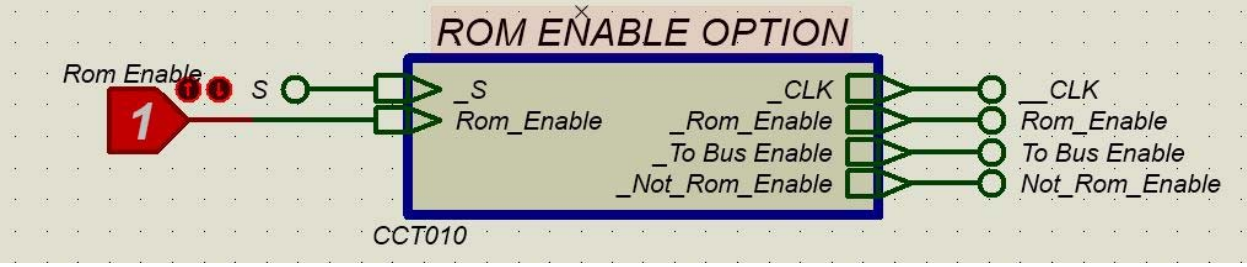
نرم افزار 010 Editor به ما این امکان را می‌دهد که کدهای خود را در مبنای ۱۶ بنویسیم و سپس این نرم افزار بطور خودکار آن را به کاراکترهای مربوطه تبدیل می کند و آن را در یک فایل ذخیره می کند. پس از اینکه دستورات مورد نظر را با استفاده از این برنامه نوشتیم و فایل مورد نظر نیز تولید شد ، حالا باید پسوند این فایل ایجاد شده را Bin بگذاریم . حالا فایل مورد نظر آماده‌ی Load شدن در ROM است که نحوه‌ی انجام این کار را در صفحه‌ی بعد ملاحظه می‌کنید.

نحوه‌ی کار با کامپیوتر

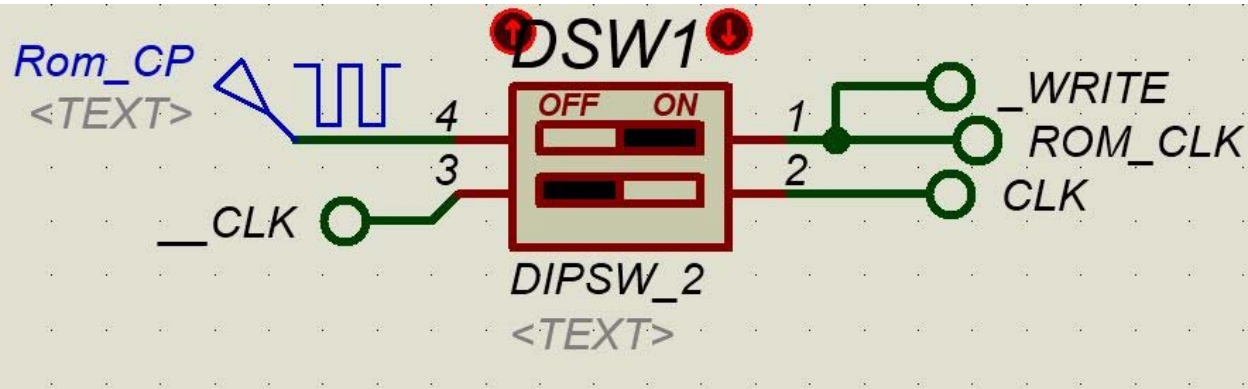
برای استفاده از کامپیوتر و کار با آن ابتدا باید سراغ کار با ROM رفت .

همانطور که در بحث های مربوط به حافظه ذکر کردیم ROM های مورد استفاده در این کامپیوتر قابلیت بارگذاری کردن و خواندن فایل را نیز دارند. برای این کار ابتدا باید به داخل واحد حافظه رفت. سپس بر روی ROM کلیک راست کرده و گزینه‌ی Edit Properties را انتخاب کرد اکنون می‌توان با دادن آدرس یک فایل (با پسوند Bin) ، ROM را مقدار اولیه داد.

برای گام بعد باید ROM را فعال نمود تا ROM وارد مدار گردد. مطابق شکل زیر باید فعالساز ROM را یک نمود

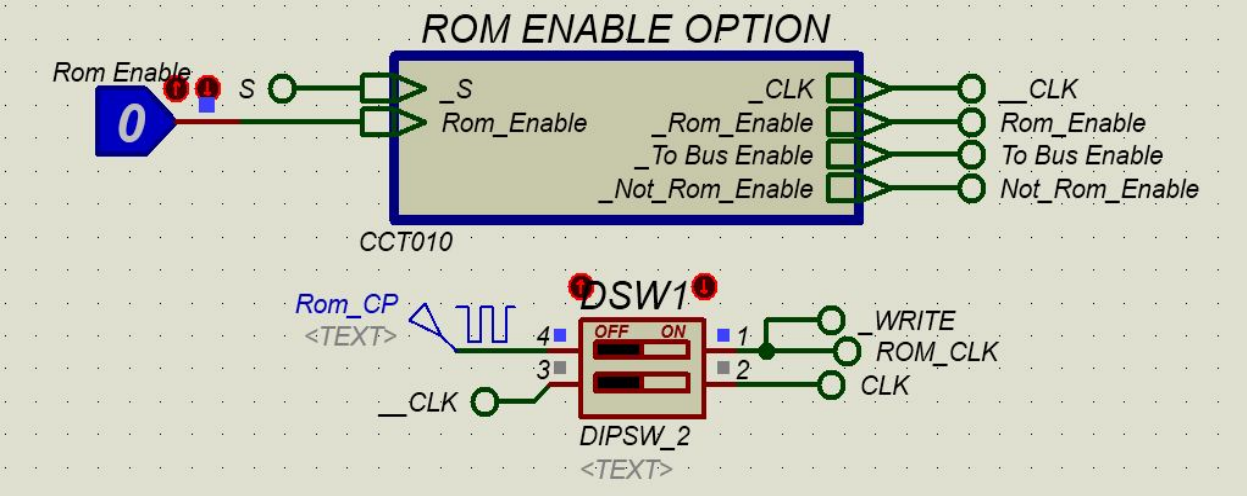


سپس کلید زیر را (منظور از کلید زیر ، کلید اولی است) از حالت Off به On تغییر می دهیم.



با این کار مولد ، صفر و یک هایی را که به سرعت دارد تولید می کند به شمارنده‌ی موجود در واحد حافظه تحویل می دهد و اطلاعات ROM بسرعت به RAM منتقل می‌شود .به همین ترتیب این صفر و یک های تولید شده ، بسرعت در اختیار خطوط Write نیز قرار گرفته و در نتیجه اطلاعات در RAM ذخیره می شوند

پس از چند ثانیه باید ROM را از مدار خارج کرد برای این کار بایستی کلید بالایی را از حالت On به Off تغییر داد و ضمنا فعالساز مربوط به ROM را هم صفر کرد که این تغییرات در شکل صفحه‌ی بعد قابل مشاهده است



حالا می‌توان با On کردن کلید کلاک کامپیوتر و در عرض چند ثانیه ، مشاهده کرد که کامپیوتر وظیفه‌ی خود را به درستی انجام می‌دهد

*با انجام بررسی‌های عملی مشاهده شد که فرکانس مولد را می‌توان تا حدود 1M در ثانیه افزایش داد اما از این حدود به بعد ROM درست عمل نخواهد کرد و نخواهد توانست اطلاعات خودش را به درستی به RAM منتقل کند

*مولد مربوط به پالس ساعت کامپیوتر نیز تقریبا به همین گونه عمل می‌کند و از حوالی 1M کلاک در ثانیه به بعد ، درست عمل نمی‌کند.

می توان فرکانس این مولد ها را نیز تغییر داد ، برای این کار باید بر روی قطعه‌ی مربوطه کلیک راست کرده و گزینه‌ی Edit Properties را انتخاب نمود.اکنون در این صفحه می‌توانیم هر عددی را که مناسب می‌دانیم برای مولد مورد نظر انتخاب کنیم.