به نام خدا

آز معماری – دکتر سربازی آزاد

دانشکده مهندسی کامپیوتر

دانشگاه صنعتی شریف

تابستان ۱۴۰۳

نیکا قادری

هدف و نتیجه مورد انتظار

در این آزمایش به طراحی و ساخت سطوح مختلف حافظه از جمله حافظه اصلی و حافظه نهان در کامپیوتر پرداخته شده است. همچنین از یک واحد محاسباتی برای اجرای دستورات استفاده شده است. هدف آزمایش این می‌باشد که مدلی از یک کامپیوتر ساده پیاده سازی شود که در آن، دستورات immdediate توسط کاربر وارد می‌شوند، سپس اطلاعات لازم از کش – یا در صورت نیاز، حافظه اصلی – گرفته شده، و در آخر، عملیات حسابی مورد نظر انجام شده و نتیجه نمایش داده می‌شود و در قسمتی از حافظه نیز ذخیره می‌گردد.

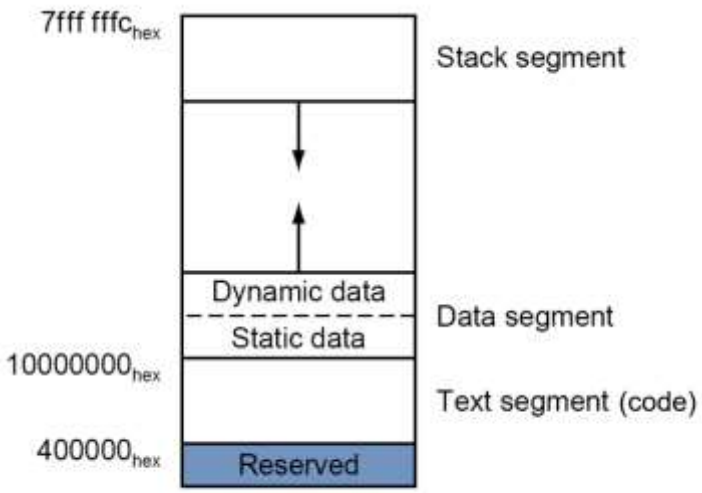
در ادامه، ابتدا به شرح جزئیات آزمایش می پردازیم. سپس ماژول‌های مختلف را توضیح داده و در آخر، مدار را در نرم افزار پروتئوس تست می کنیم.

شرح آزمایش

دستورات ۱۶ بیتی به فرمت زیر می‌باشند:

Operation: یک بیت برای مشخص کردن نوع عملیات است. صفر به معنای جمع و یک به معنای تفریق می‌باشد. برای عمل تفریق، عدد موجود در read address از immediate کم می‌شود.

Read address: آدرسی از حافظه اصلی را مشخص می‌کند که داده موجود در آن، یکی از اپرندها هست. حافظه اصلی متشکل از دو بخش مجزا هر یک با ظرفیت ۸ عدد هشت بیتی می‌باشد. بخش اول، حافظه ای است که به آن reserved گفته می‌شود. این حافظه ثابت بوده و قابل تغییر نیست. بخش دوم، حافظه متغیر یا dynamic memory بوده که محتویاتش می‌تواند در نتیجه اجرای دستورات، دستخوش تغییر شود.



از آنجایی که در کل ۱۶ ظرفیت برای داده داریم، مشخص است که به چهار بیت برای انتخاب آدرس نیز نیاز است. البته علاوه بر حافظه اصلی، یک حافظه نهان با ظرفیت ۸ داده هشت بیتی – به همراه یک بیت value و یک بیت tag برای هر داده - نیز پیاده سازی شده است. سیاست بار گذاری اطلاعات در حافظه نهان direct-mapped و سیاست جایگزینی LRU (least recently used) می‌باشد.

در هر دستور، ابتدا باید به کش مراجعه شود و اگر داده مورد نظر آنجا نبود، آن را از حافظه اصلی به کش انتقال می‌دهیم و سپس از آن استفاده می‌شود.

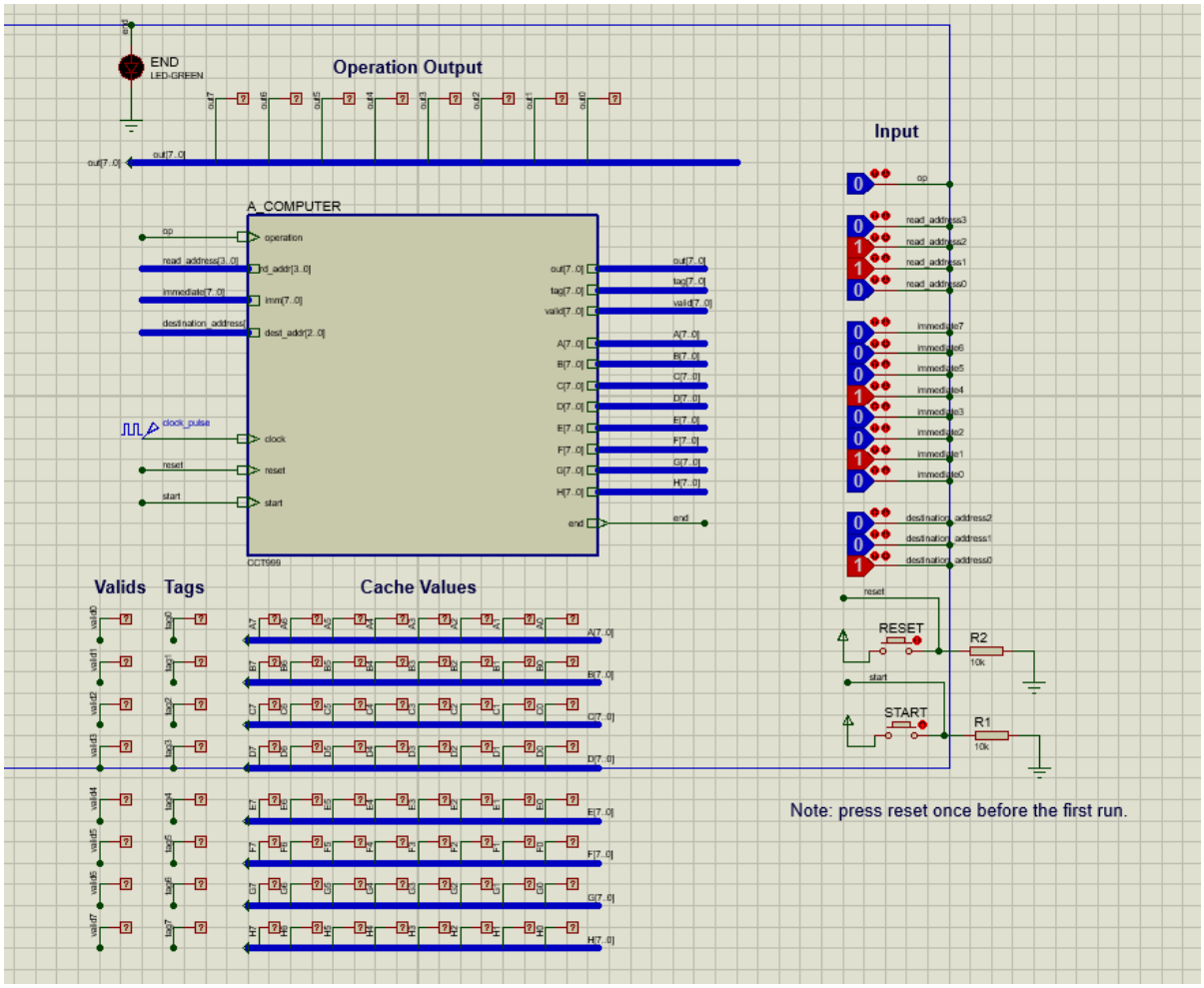
Immediate: عددی هشت بیتی که اپرند دوم عملیات حسابی می‌باشد.

Destination address: علاوه بر نمایش دادن نتیجه، باید آن را در حافظه اصلی نیز ذخیره کرد. از آنجایی که حافظه reserved نمی‌تواند تغییر کند، باید نتیجه را در dynamic memory ذخیره کنیم. محل ذخیره سازی توسط این آدرس مشخص می‌شود. توجه کنید که در این معماری فرض شده که آدرس حافظه ثابت، قبل از حافظه متغیر است. یعنی آدرس‌هایی به فرم 0xxx متعلق به حافظه ثابت و آدرس‌هایی به فرمت 1xxx برای حافظه متغیر هستند. بنابراین بیت آخر به طور پیش فرض یک می‌باشد و نیاز به ورودی گرفتن آن نیست.

همچنین توجه شود که هنگام نوشتن نتیجه روی آدرس مقصد، اگر داده ای از قبل در حافظه متغیر وجود داشته باشد، باید بازنویسی (overwrite) شود. همچنین اگر این آدرس از قبل در کش لود شده بود، باید بیت ولید آن خانه در کش صفر شود چراکه داده بازنویسی شده و دیگر معتبر نمی‌باشد.

جزئیات پیاده سازی

نمای کلی مدار در صفحه بعد آورده شده است.



Input Registers

وظیفه این بخش ذخیره بخش‌های مختلف دستور می‌باشد تا در حین کار برنامه، این مقادیر ثابت بمانند و برنامه دچار مشکل نشود. برای ذخیره این مقادیر از تعدادی ماژول ۷۴۱۹۴ و ۷۴۱۹۸ استفاده کردیم که به ترتیب شیفت رجیستر چهار بیتی و هشت بیتی می‌باشند.

عملکرد این قطعات به صورت زیر می‌باشد:

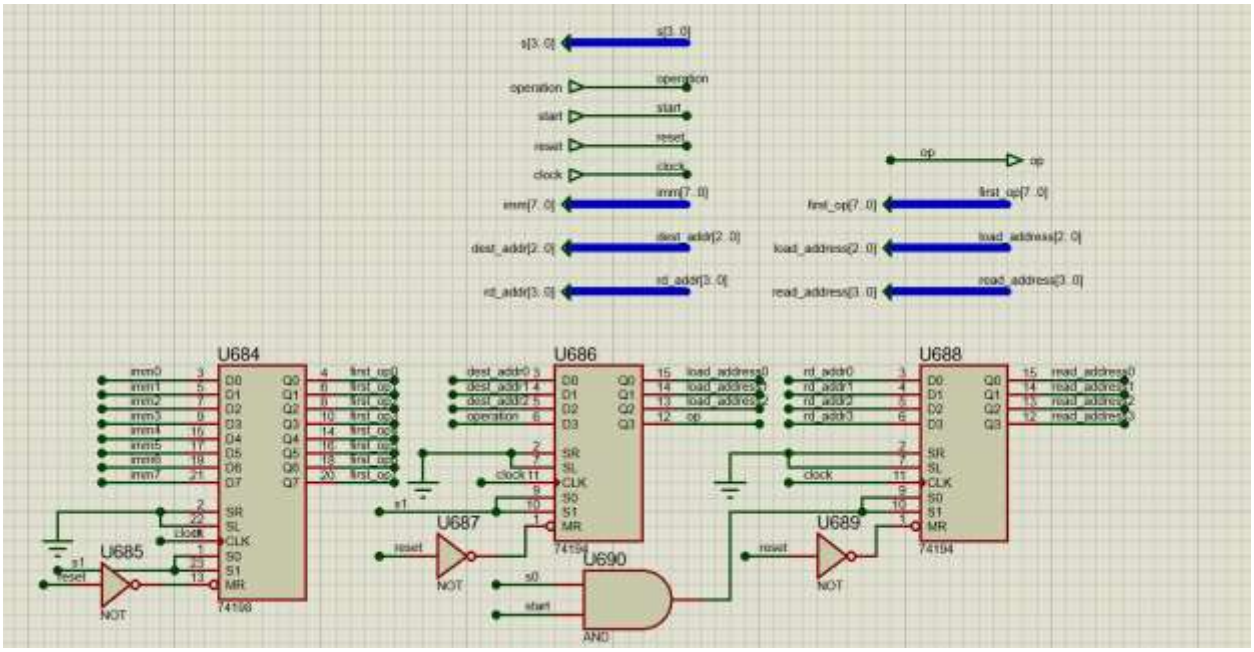
MODE SELECT TABLE

INPUTS				RESPONSE
$\overline{MR}$	CP	S_0*	S_1*	
L	X	X	X	Asynchronous Reset; Outputs = LOW
H	↗	H	H	Parallel Load; $P_n \longrightarrow Q_n$
H	↗	L	H	Shift Right; $D_{SR} \longrightarrow Q_0$, $Q_0 \longrightarrow Q_1$, etc.
H	↗	H	L	Shift Left; $D_{SL} \longrightarrow Q_7$, $Q_7 \longrightarrow Q_6$, etc.
H	X	L	L	Hold

*Select Inputs should be changed only while CP is HIGH
H = HIGH Voltage Level
L = LOW Voltage Level
X = Immaterial

از قابلیت شیفت این رجیسترها استفاده نمی‌شود و تنها لازم است پس از شروع برنامه یک بار با توجه به ورودی‌های کاربر لود شوند. البته read registers چون در مرحله اول مورد استفاده قرار می‌گیرد لازم است با به محض فشرده شدن ورودی استارت لود شود، برای همین ورودی کنترلی لود آن اندکی متفاوت با بقیه می‌باشد.

شمای این قسمت در زیر آورده شده است:



Stats

الگوریتمی‌که استفاده کردیم دارای چهار مرحله اصلی است که این ماژول مسئولیت مدیریت آن‌ها را بر عهده دارد. این چهار مرحله عبارت‌اند از:

S0: مرحله ابتدایی می‌باشد. تا هنگامی‌که کلید استارت نخورده باشد در این مرحله می‌مانیم. همچنین وقتی که برنامه تمام شد به این حالت می‌رویم.

S1: در این حالت استارت زده شده و اجرای برنامه شروع می‌شود. بررسی می‌شود که آیا داده خواسته شده در کش وجود دارد یا نه، در صورت عدم وجود این داده از حافظه داخل کش لود می‌شود.

S2: در این مرحله نتیجه عملیات حسابی داخل آدرس مقصد ذخیره می‌شود. همچنین داده قبلی در صورت وجود در حافظه نهان، مخدوش می‌شود.

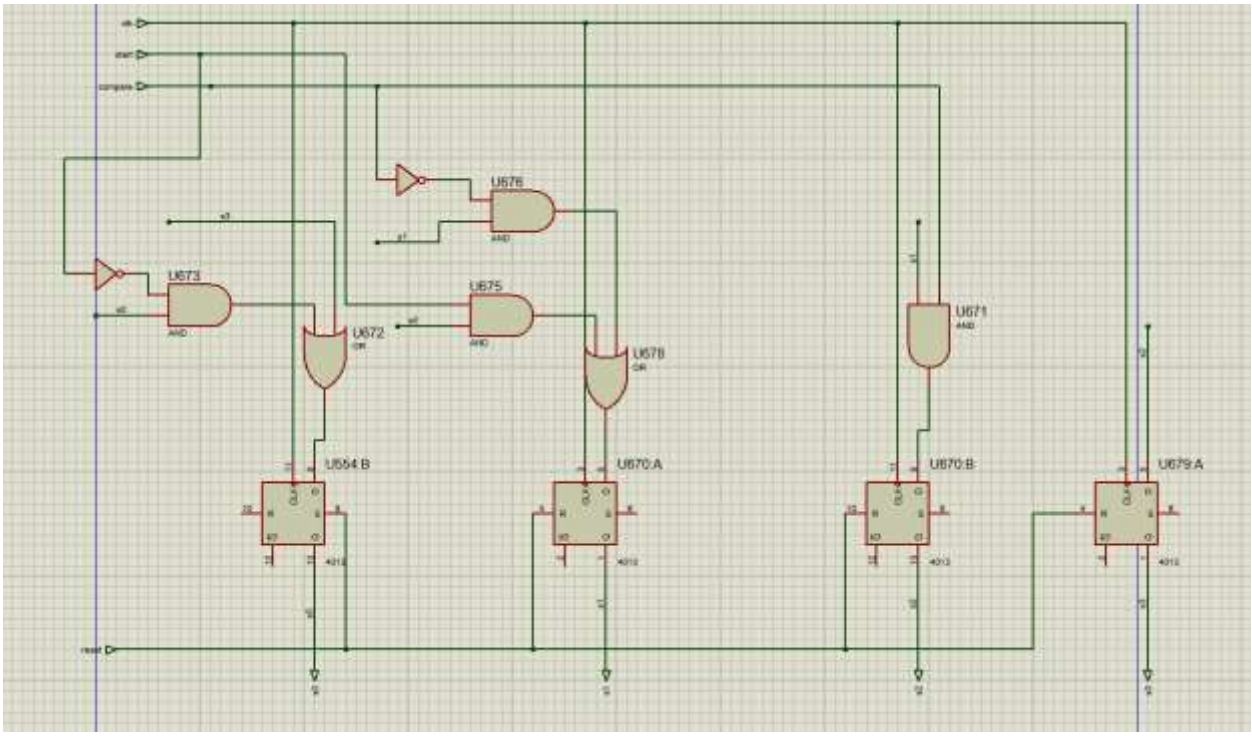
S3: در این مرحله سیگنال end فعال شده و نتیجه اعلام می‌گردد. سپس به حالت اول بر می‌گردیم.

با توجه به توضیحات بالا، هر یک از حالت‌ها به شکل زیر به دست می‌آیند:

$$
\begin{aligned}
s_0^+ &= s_0\overline{start} + s_3 \\
s_1^+ &= \overline{s_0}start + s_1\overline{compare} \\
s_2^+ &= s_1 compare \\
s_3^+ &= s_2
\end{aligned}
$$

در اینجا compare بیت کنترلی است که تعیین می‌کند آیا آدرس موجود در کش با آدرس read address برابر، و داده آن ولید هست یا نه. به عبارتی، مشخص می‌کند آیا نیاز هست به حافظه اصلی رجوع شود یا خیر.

در نتیجه کنترلر وضعیت با استفاده از دی فلیپ فلاپ و گیت‌های منطقی ساخته می‌شود:



توجه کنید که در شروع کار برنامه، برای تنظیم حالت روی s0، نیاز هست یک بار کلید ریست فشرده شود.

Control Unit

برای اجرای صحیح برنامه، نیاز هست که یک سری سیگنال برای کنترل load و store در بخش‌های مختلف، تولید شوند. وظیفه این بخش همین است؛ کنترل جریان داده و روند اجرای برنامه.

ابتدا سیگنال‌های مورد نیاز را شرح می‌دهیم:

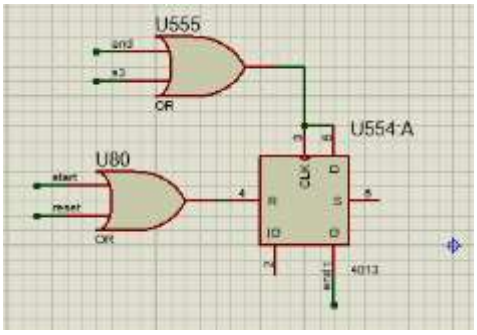
۱- Cache data load enable: در صورت عدم وجود داده مورد نیاز، آن را از حافظه می‌آورد و با فعال شدن این سیگنال، این مقدار به همراه بیت‌های تگ و ولید درون کش لود می‌شود.
۲- Load data enable: پس از انجام عملیات حسابی، با فعال شدن این سیگنال، نتیجه داخل حافظه متغیر لود می‌شود.
۳- Delete enable: در صورت بازنویسی یک آدرس از حافظه اصلی که در کش هم موجود است، داده کش نامعتبر می‌شود و در نتیجه بیت ولید آن داده باید صفر شود.
۴- End: سیگنالی که پایان فرآیند را مشخص می‌کند.

با توجه به توضیحات بالا، سیگنال‌های کنترلی طبق این فرمول‌ها ساخته می‌شوند:

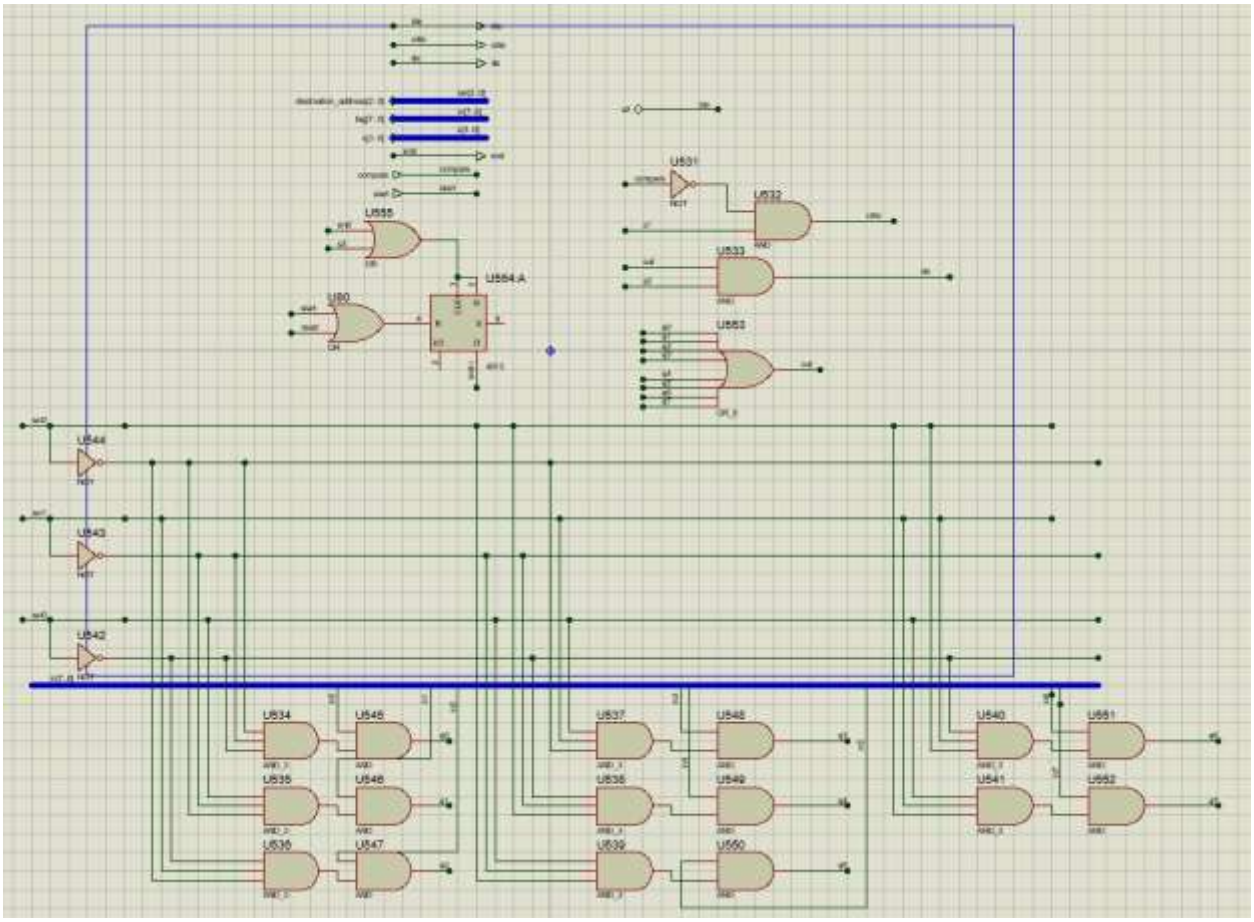
$$cache\ data\ load\ enable^{+} = s_1\overline{compare}$$
$$load\ data\ enable^{+} = s_2$$
$$delete\ enable^{+} = s_2(tag[destination\ address])$$

توجه شود که از آنجایی که مقصد همواره در حافظه متغیر می‌باشد، آدرس همواره به فرمت 1xxx می‌باشد و اگر خانه ای از کش با این آدرس وجود داشته باشد بیت تگ آن باید برابر با یک باشد.

همچنین برای ساخت سیگنال end از یک فلیپ فلاپ دی استفاده می‌کنیم که با رسیدن به s3 فعال شود و یکبار کلاک بخورد (یکبار عدد یک لود می‌شود) و هنگامی‌که ریست یا استارت فعال می‌شوند این سیگنال را ریست می‌کنند:



در نهایت این بخش نیز مطابق صفحه بعد ساخته می‌شود.



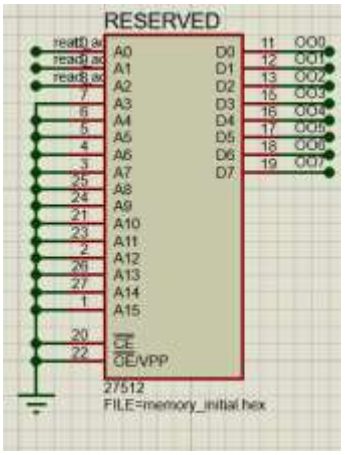
Reserved

حافظه ثابت برنامه که با یک eprom به شماره ۲۷۵۱۲ ساخته شده است. سه بیت آدرس برای خواندن دیتا می‌گیرد و خانه‌های آن، به طور دلخواه به این شکل مقدار دهی شده‌اند:

```
00 215
01 53
02 12
03 36
04 85
05 61
06 14
07 0
```

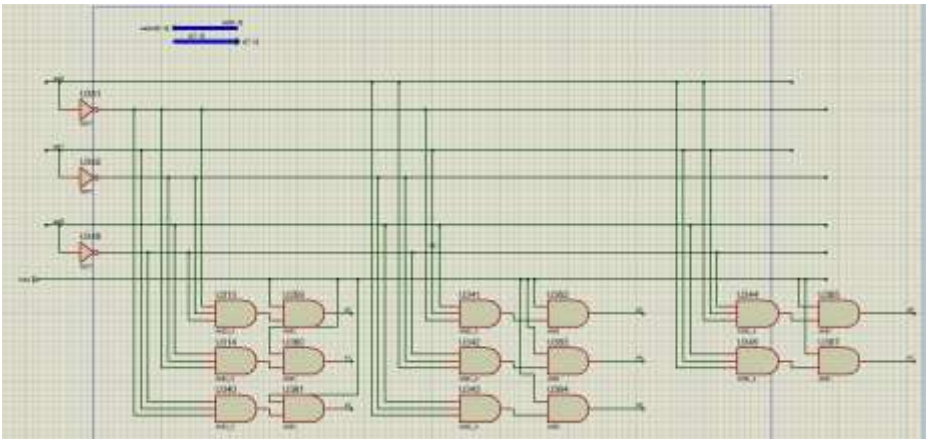
برای مقداردهی eprom باید از یک فایل با فرمت hex. استفاده کرد که با استفاده از نرم افزار HxD، به راحتی می‌توان کد آن را تولید کرده و در قسمت image files از قطعه آپلود کرد. کد این رام به این صورت در می‌آید:

```
:08000000D7350C24553D0E001C
:00000001FF
```

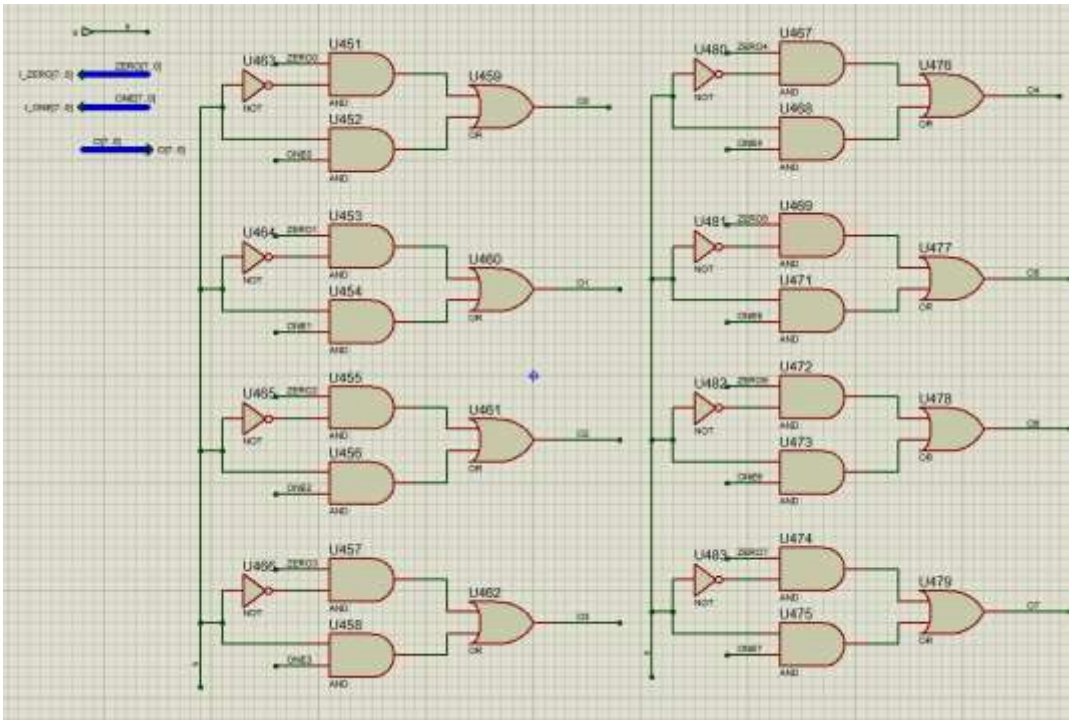


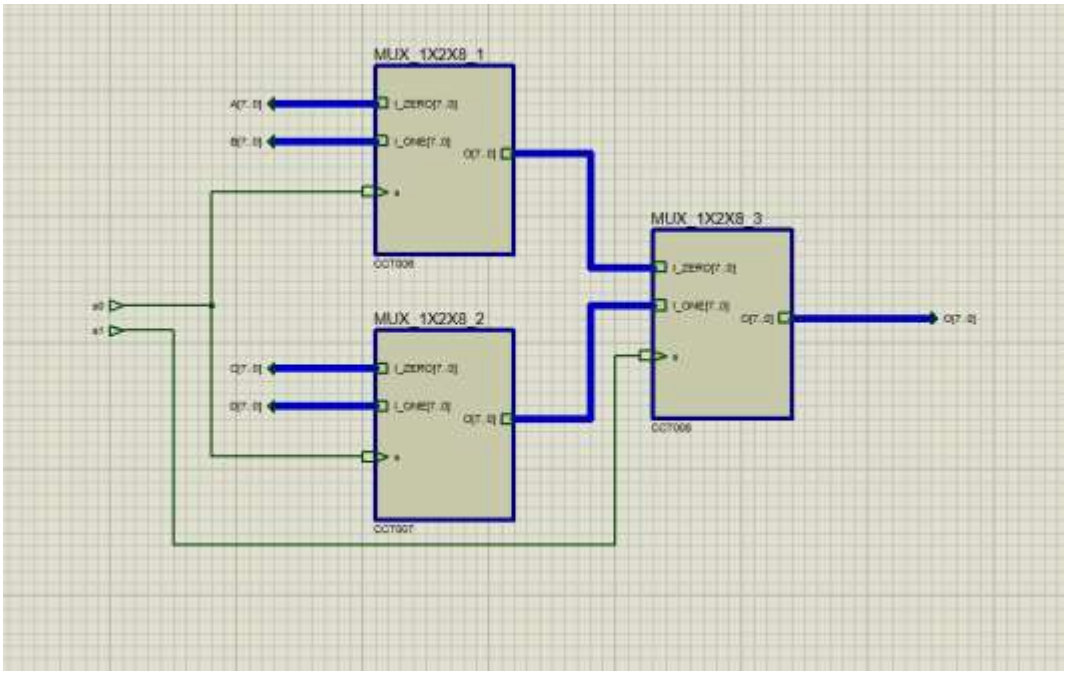
Dynamic Memory

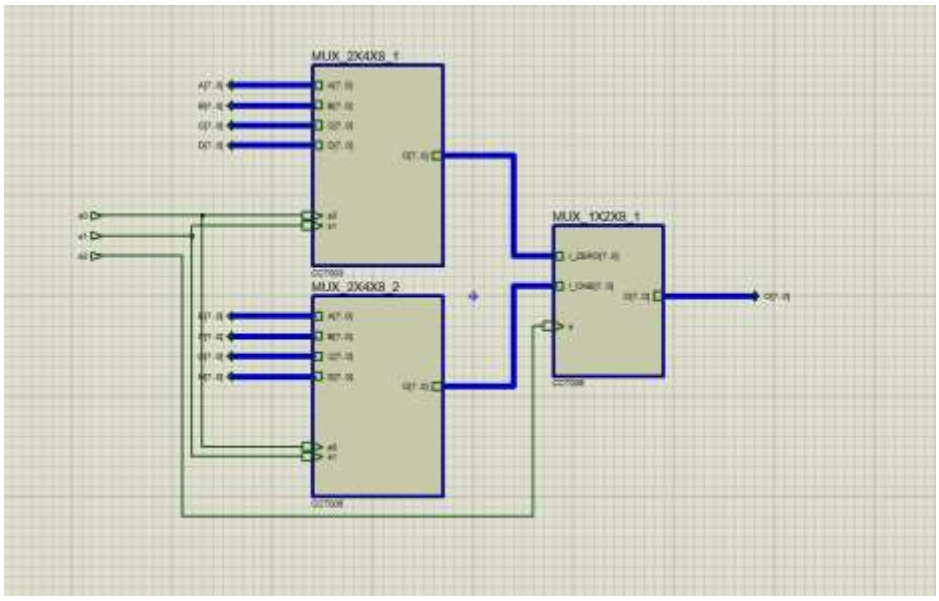
همان طور که از اسمش پیداست این ماژول، حافظه متغیر برنامه می‌باشد، زیر مجموعه حافظه اصلی. متشکل از هشت رجیستر است که هر یک نماینده یکی از خانه‌های حافظه هستند. هر کدام از آن‌ها می‌توانند در صورتی که سیگنال لود آن‌ها یک شود، داده ورودی را در خود لود کنند. با توجه به load address و با کمک یک دیکودر، یکی از سیگنال‌های s0 تا s7، یک می‌شود که به طبع موجب لود دیتا در یکی از خانه‌های حافظه نهان می‌شود. البته این عمل فقط باید هنگامی انجام بگیرد که load – load data enable – فعال باشد. در نتیجه در ماژول دیکودر، از هر یک از خروجی‌ها با load، and گرفته شده است:

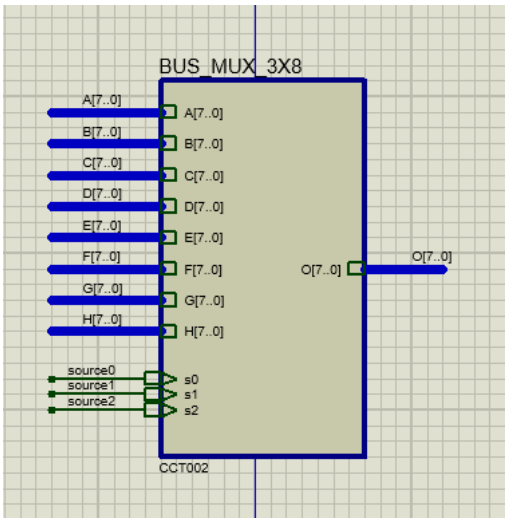


همچنین لازم است همواره داده موجود در read address خروجی داده شود. بنابراین از یک مالتی پلکسر هشت به یک استفاده می‌کنیم که از ترکیب چند واحد کوچک تر ساخته شده است:

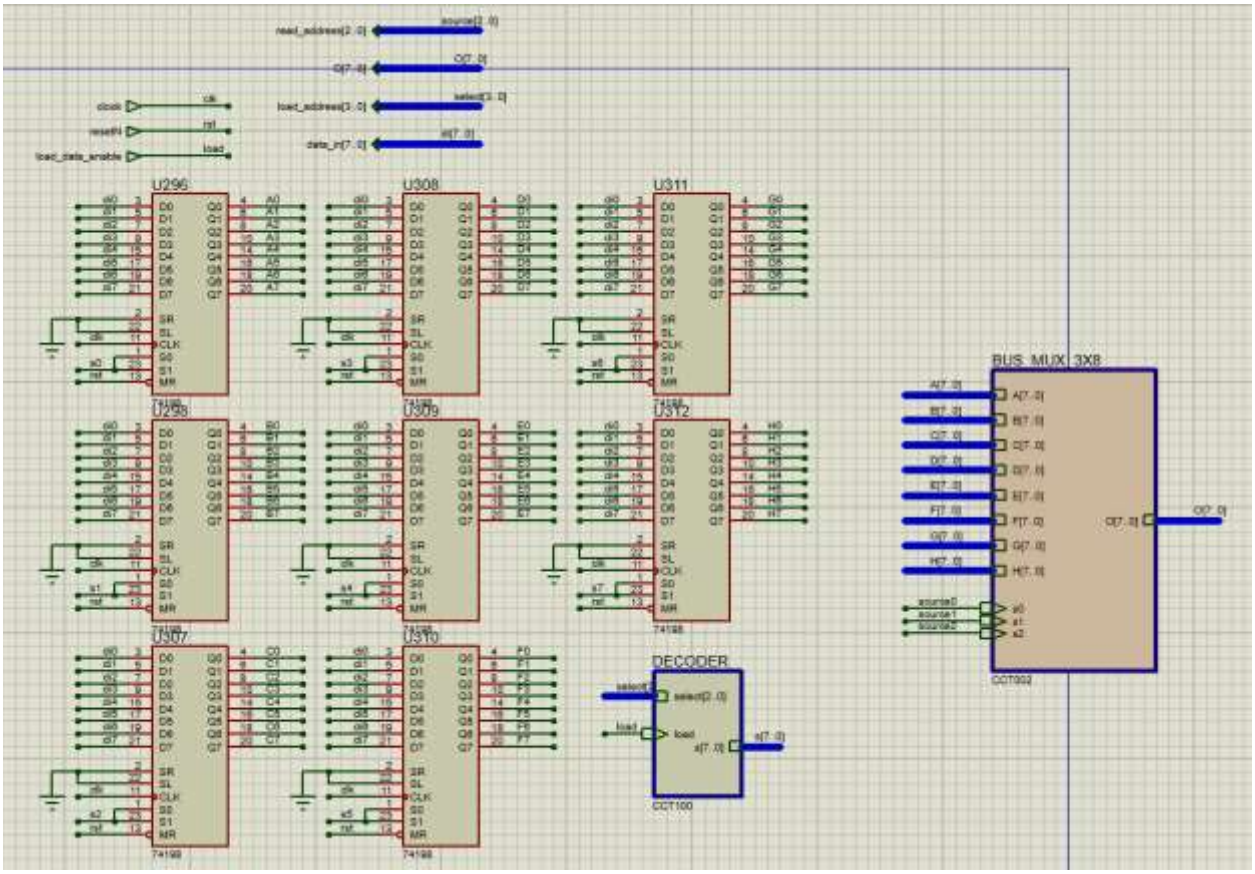






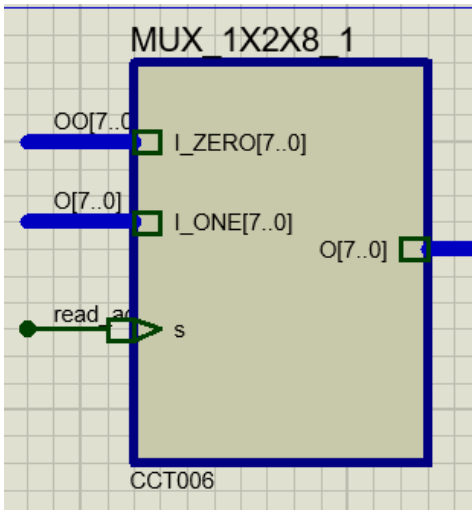


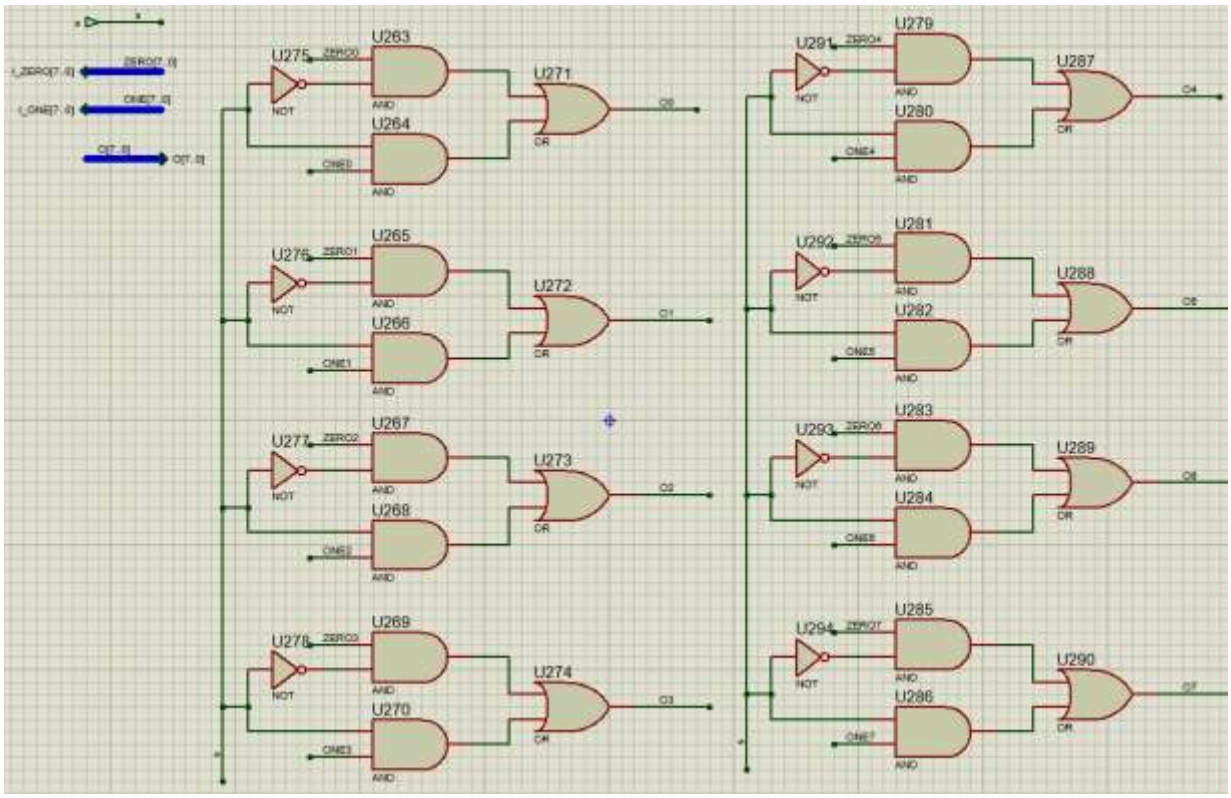
شمای کلی این ماژول در صفحه بعد آورده شده است.



MUX_1X2X8_1

یک مالتی پلسکر دو به یک با پهنای هشت بیت می‌باشد. هدف این ماژول این است که با توجه به آدرس ورودی، داده مناسب را از بین خروجی حافظه متغیر و خروجی حافظه ثابت انتخاب کند.



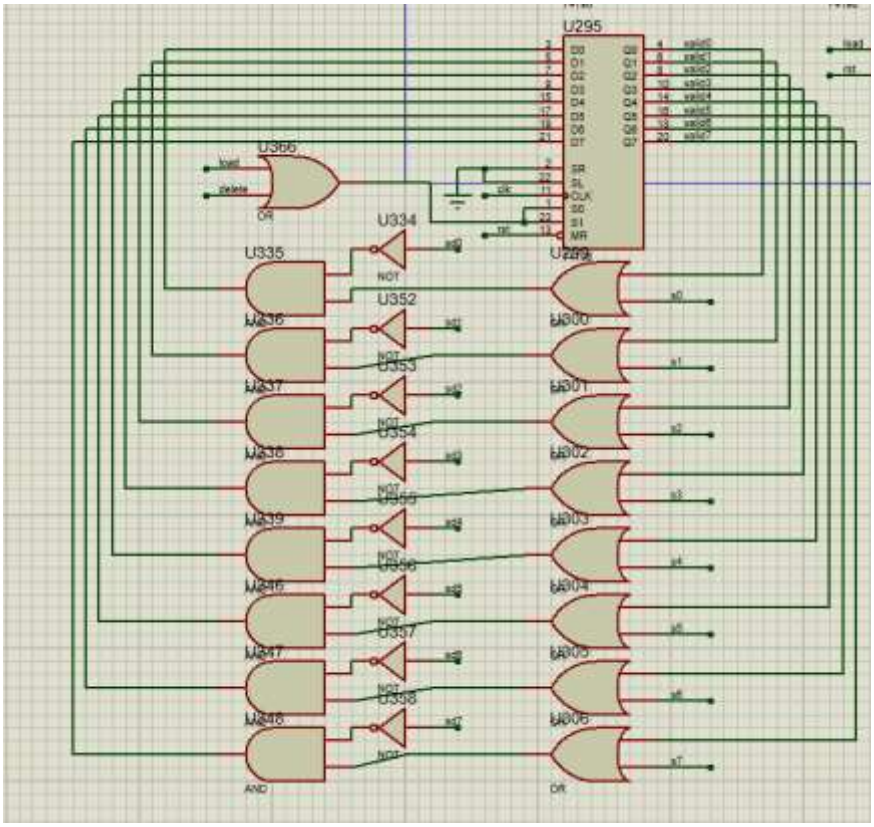


8 Bit Cache

حافظه نهان مدار می‌باشد. ساختار آن تا حد زیادی شبیه به ماژول dynamic memory بوده و به جهت صرفه جویی، تنها نکات تفاوت آن ذکر می‌گردد.

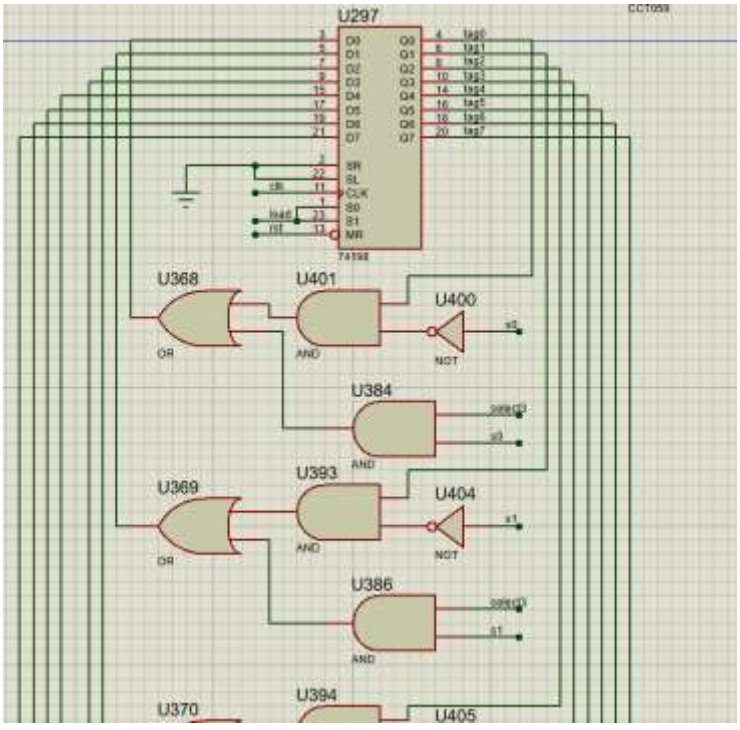
این ماژول دو شیفت رجیستر اضافه برای نگه داری value و tag هر آدرس در خود دارد. هر خانه از شیفت رجیسترها متعلق به یک آدرس است. (بر عکس شیفت رجیسترهای قبلی که هر واحد نماینده یک آدرس بود) در نتیجه هنگام لود، تغییراتی در آنها داده شده است:

در شیفت رجیستر valid، هنگامی‌نیاز به لود داریم که داده ای جدید بخواهد روی کش نوشته شود و بیت ولید باید یک شود، یا آدرسی در حافظه اصلی بازنویسی شده و الان نیاز است بیت ولید آن در کش صفر شود. پس سیگنال کنترلی ترکیب load و delete می‌باشد. حال هنگام لود در یک آدرس خاص، بقیه خانه‌ها باید مقدار قبلی خود را حفظ کنند و بیت ذخیره شده در آدرس مورد نظر باید یک یا صفر شود. سیگنال‌های s0 – s7 هم مشخص می‌کنند دقیقا چه آدرسی باید یک شود. اما نیاز است یک سری سیگنال هم برای ریست کردن خانه‌های رجیستر داشته باشیم. بنابراین از یک دیکودر دیگر استفاده می‌کنیم که delete address و delete enable را می‌گیرد و sd0 – sd7 را خروجی می‌دهد.

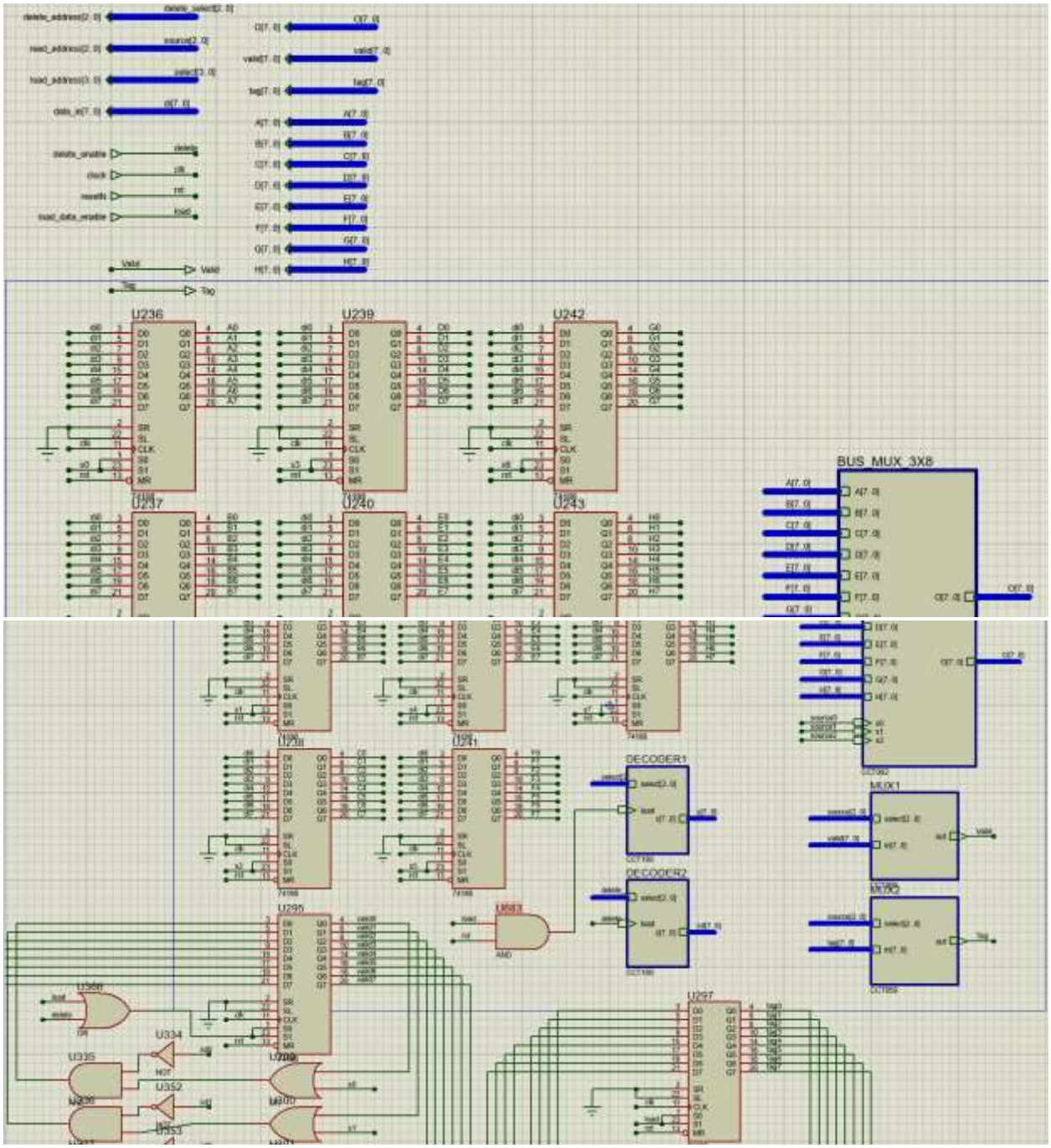


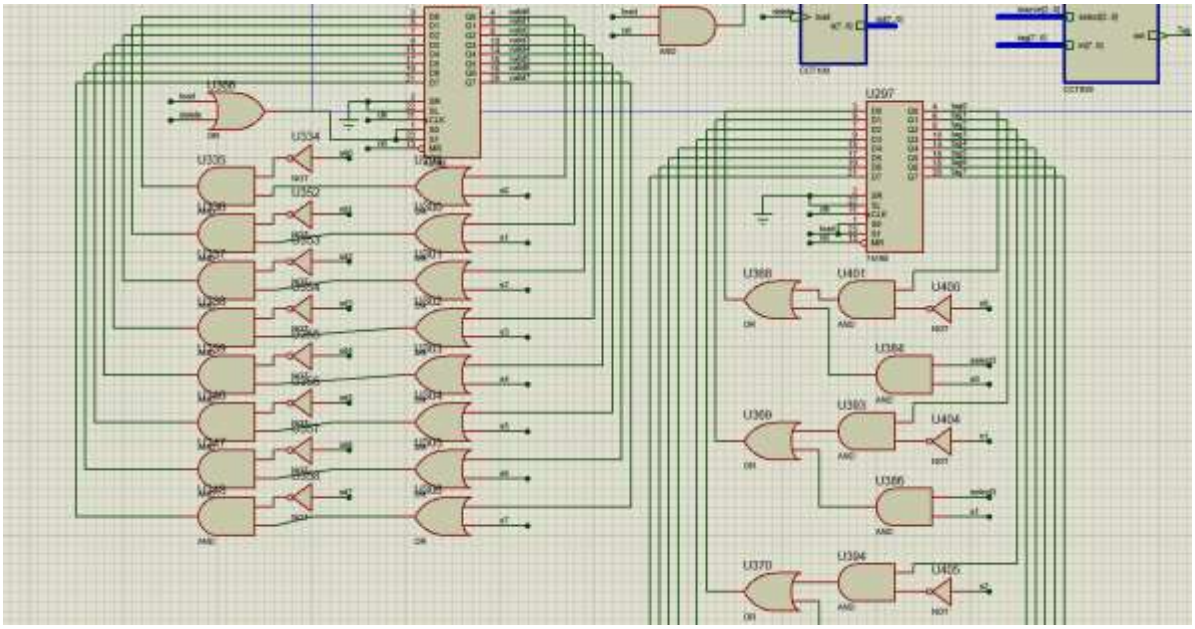
در شیفت رجیستر tag، نیازی به سیگنال‌های delete نداریم چراکه فقط در هنگام load، فعال می‌شود.

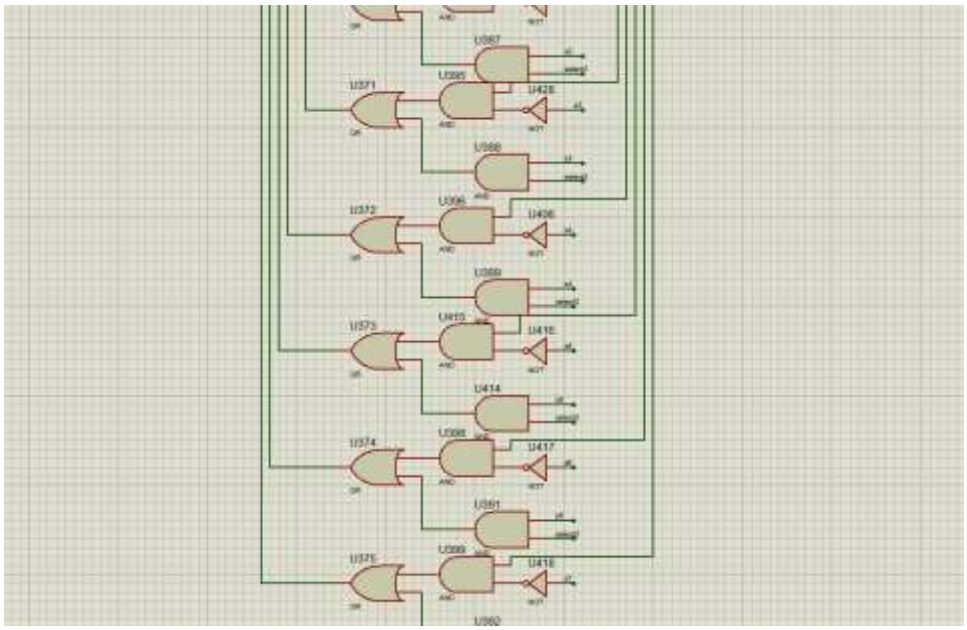
برای پیاده سازی آن از ساختارهایی مشابه مالتی پلسکرهای دو به یک استفاده می‌کنیم:

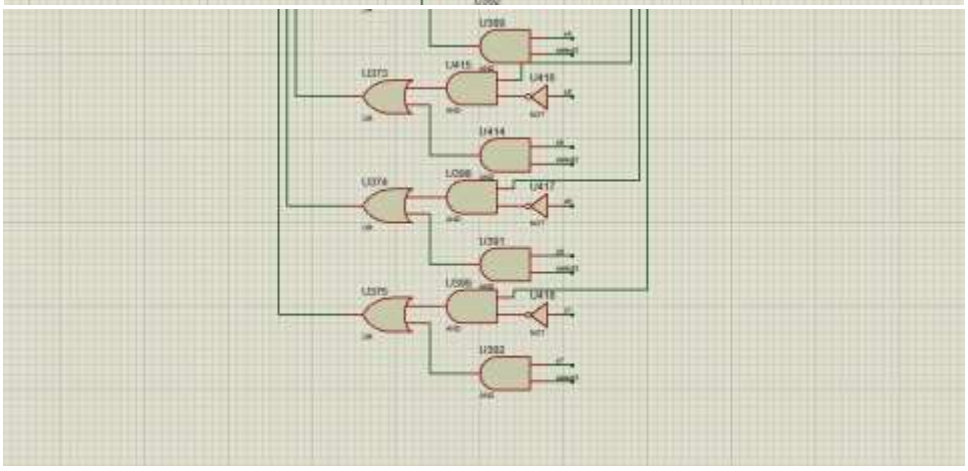


شمای کلی این حافظه به این صورت در می‌آید:



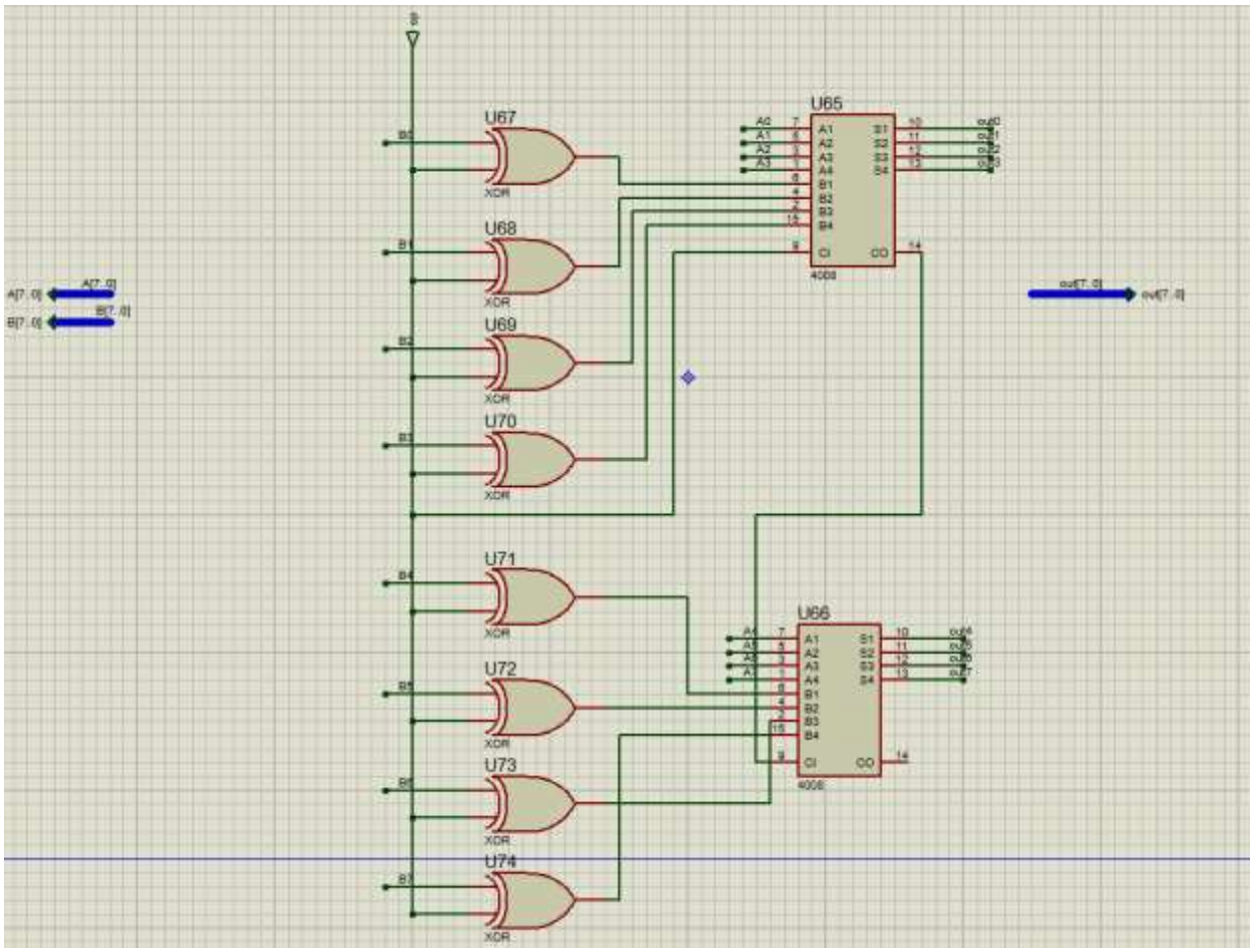






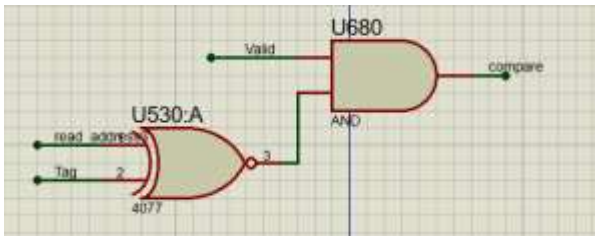
Add/Sub

یک جمع کننده/تفریق کننده هشت بیتی carry ripple adder می‌باشد که از دو واحد 4008 – جمع کننده چهار بیتی – ساخته شده است. البته ورودی دوم با op، xor شده و همچنین op به عنوان cin به جمع کننده اول وارد شده است تا در صورتی که یک باشد، مکمل دوی عدد دوم استفاده شود و تفریق دو عملوند خروجی داده شود.



Compare

مدار ساخت سیگنال compare با توجه به توضیحات قبلی به این شکل است:



Test

جهت تست مدار، یک سری از دستورات وابسته را ابتدا بررسی کرده و توضیح می‌دهیم:

جهت سادگی، هر جا که صحبت از ذخیره کردن نتیجه در آدرس مقصد شده، منظور آدرس مقصد در حافظه متغیر بوده است.

۱- آدرس ۶ از حافظه ثابت، با ۱۸ جمع بشود و نتیجه در آدرس ۱ ذخیره شود.
۲- آدرس ۱ از حافظه متغیر، از ۱۱۷ کم شود و در آدرس ۵ ذخیره شود.
۳- آدرس ۴ از حافظه ثابت، با ۱۵ جمع شود و در آدرس ۷ ذخیره شود.
۴- آدرس ۵ از حافظه متغیر، از ۱۵۷ کم شود و در آدرس ۱ ذخیره شود.
۵- آدرس ۱ از حافظه متغیر، از ۹۶ کم شود و در آدرس ۵ ذخیره شود.

در طول این پروسه، محتویات حافظه‌ها و نتیجه هر دستور به صورت زیر می‌باشد:

(محتویات کش به صورت valid, tag, data ..... address نمایش داده شده است)

1-

Result = 18 + 14 = 32

Dynamic memory contents:

01 …….. 32

Cache contents:

06 …….. 1 0 14

2-

Result = 117 – 32 = 85

Dynamic memory contents:

01 …….. 32

05 …….. 85

Cache contents:

06 …….. 1 0 14

01 …….. 1 1 32

3-

Result = 85 + 15 = 100

Dynamic memory contents:

01 …….. 32

05 …….. 85

07 …….. 100

Cache contents:

06 …….. 1 0 14

01 …….. 1 1 32

04 …….. 1 0 85

4-

Result = 157 – 85 = 72

Dynamic memory contents:

01 …….. 72

05 …….. 85

07 …….. 100

Cache contents:

06 …….. 1 0 14

01 …….. 0 1 32

04 …….. 1 0 85

05 …….. 1 1 85

5-

Result = 96 – 72 = 24

Dynamic memory contents:

01 …….. 72

05 …….. 24

07 …….. 100

Cache contents:

06 …….. 1 0 14

01 …….. 1 1 72

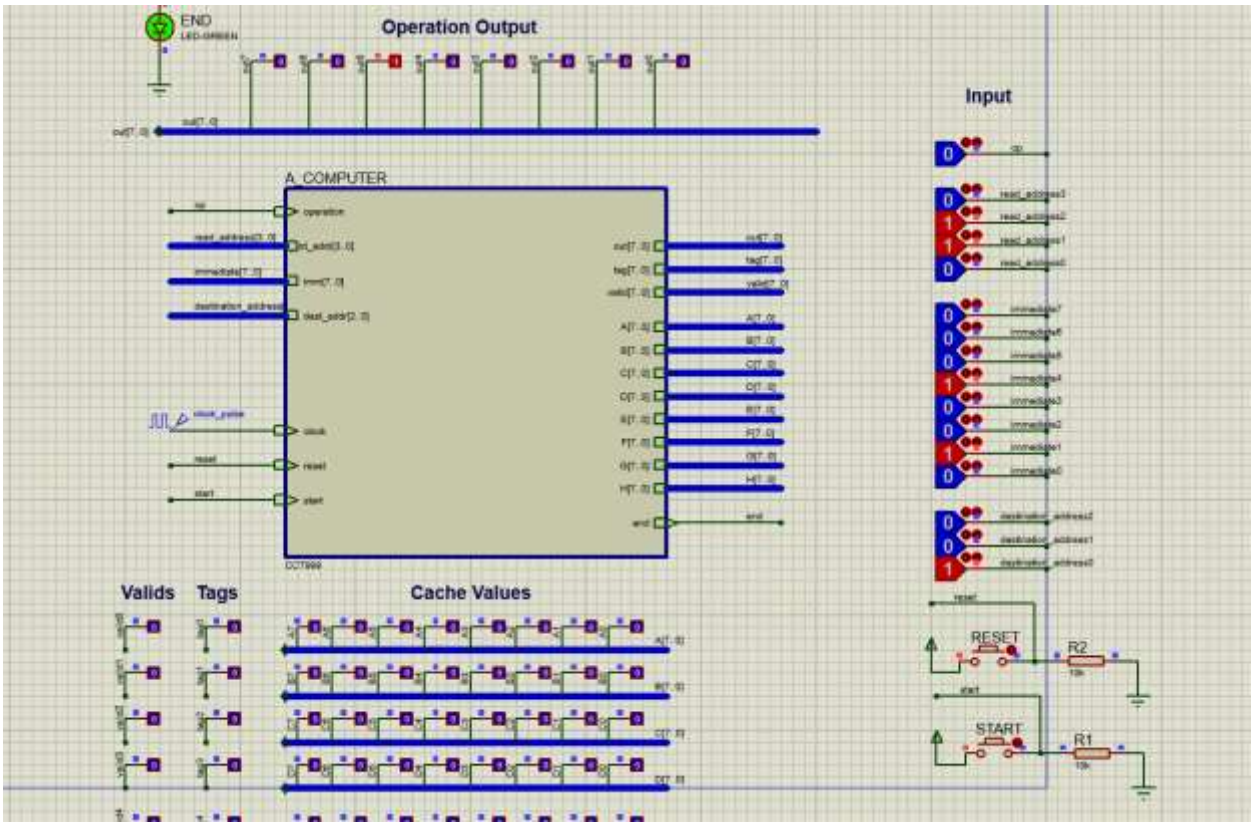
04 …….. 1 0 85

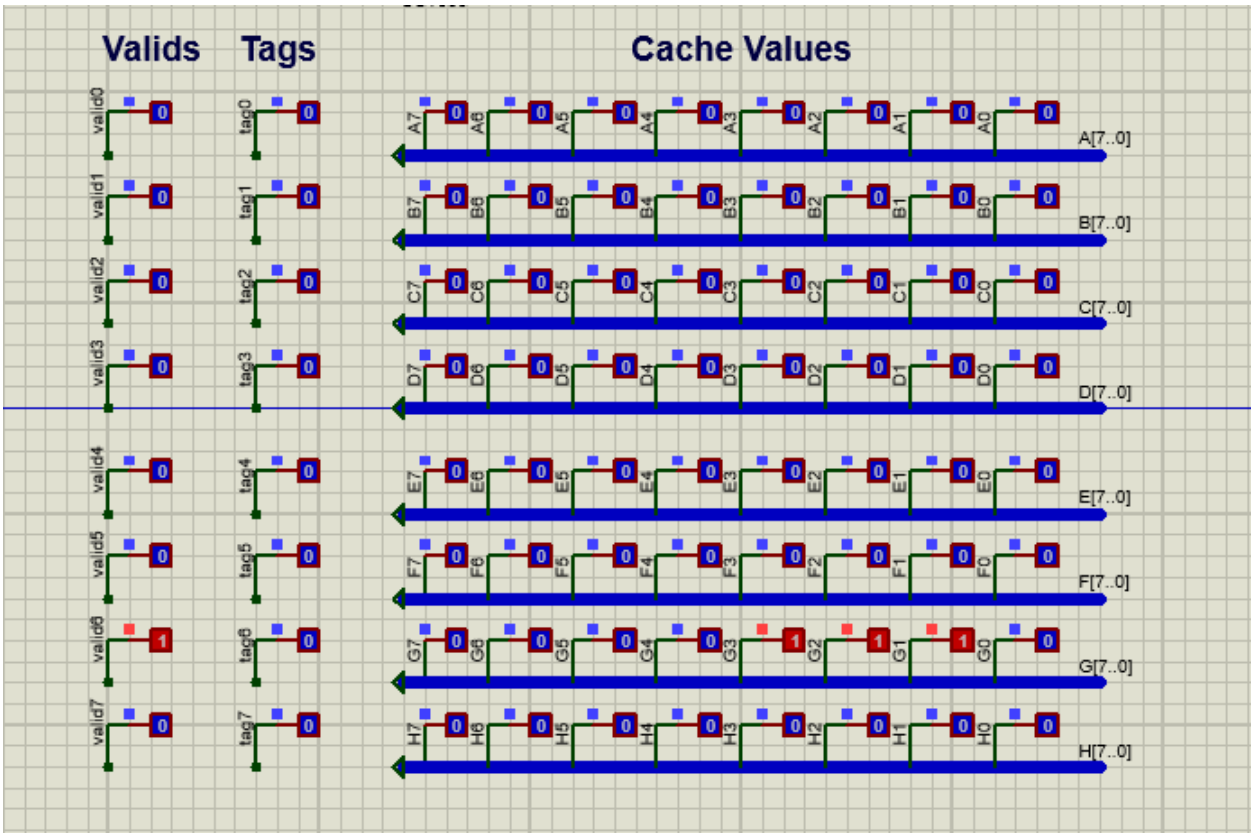
05 …….. 0 1 85

در دو مرحله ۴ و ۵، دیتای موجود در حافظه متغیر بازنویسی شده که باعث صفر شدن بیت ولید داده متناظر در حافظه نهان شده است.

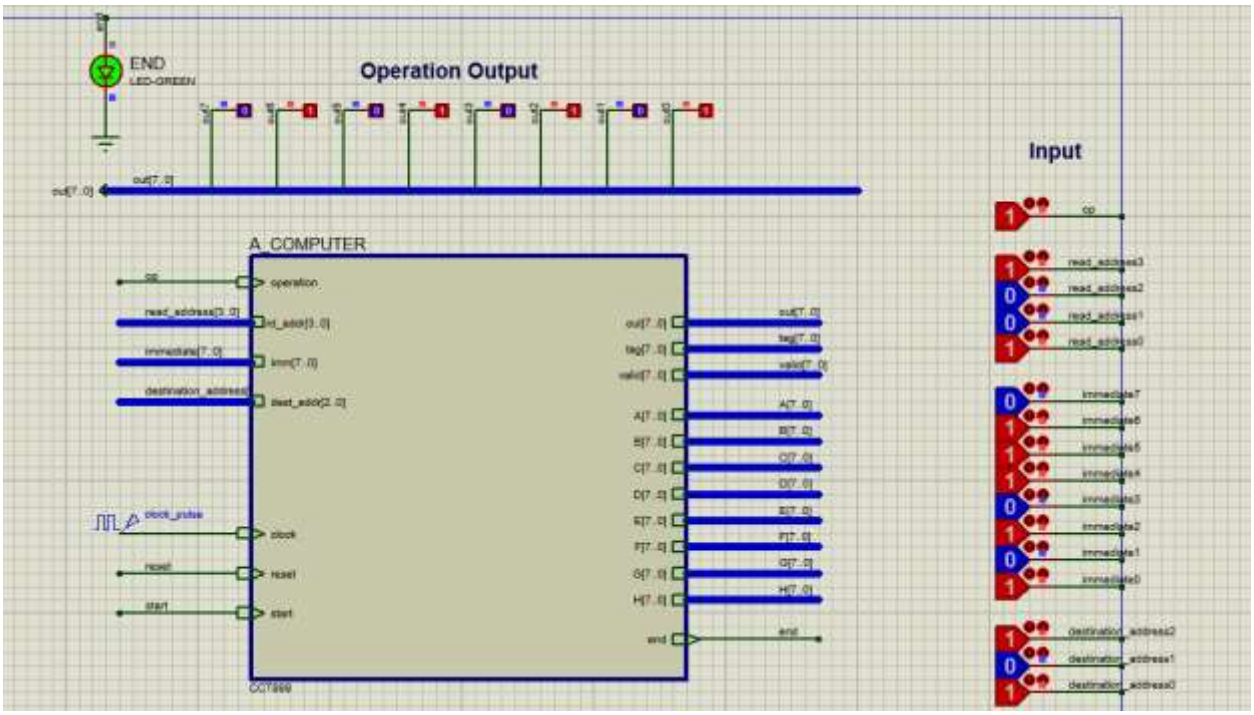
این سری از دستورات به صورت عملی در پروتئوس اجرا شده‌اند که نتایج آن در صفحات بعد، درستی عملکرد مدار را تائید می‌کند.

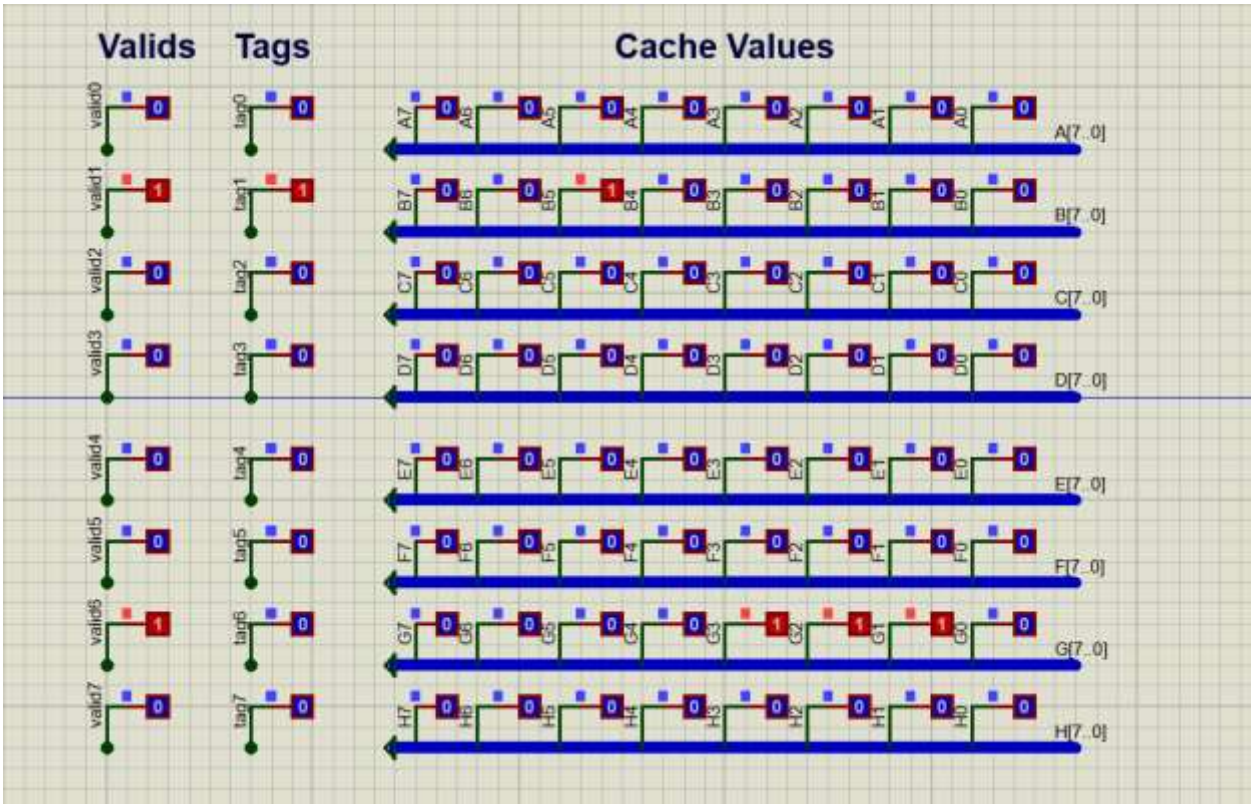
دستور اول:


END
Operation Output
A_COMPUTER
Input
Valids
Tags
Cache Values
RESET
START
R2
R1

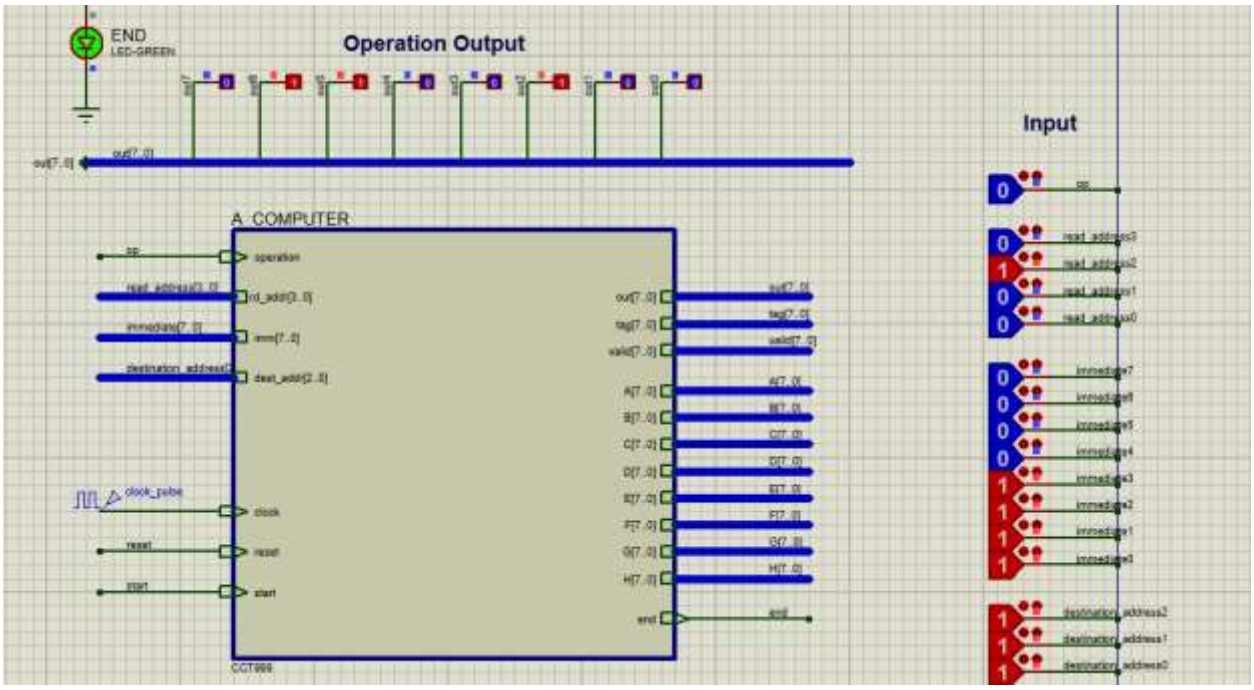

Valids
Tags
Cache Values
A[7..0]
B[7..0]
C[7..0]
D[7..0]
E[7..0]
F[7..0]
G[7..0]
H[7..0]

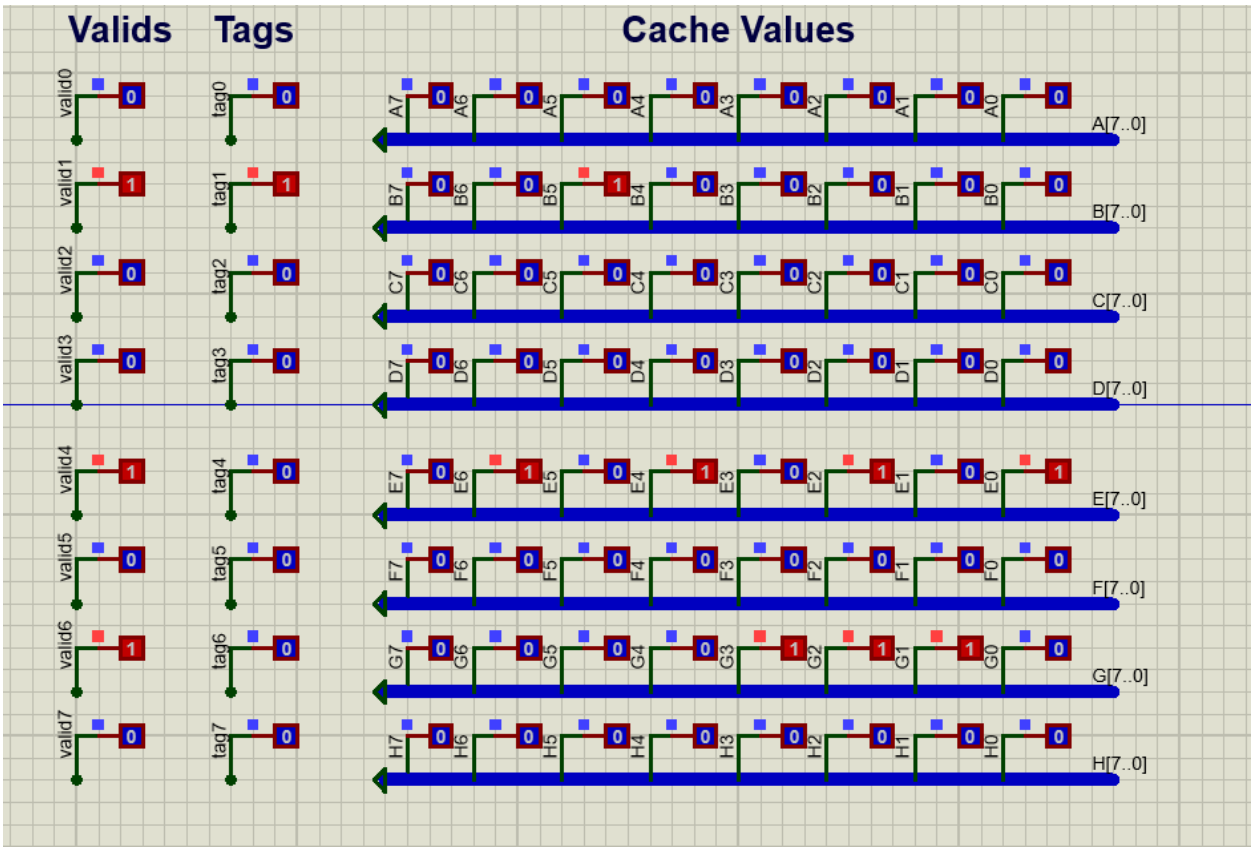
دستور دوم:


END
LED-GREEN
Operation Output
out[7..0]
Input
A_COMPUTER
operation
clock
reset
start
clock_pulse
end

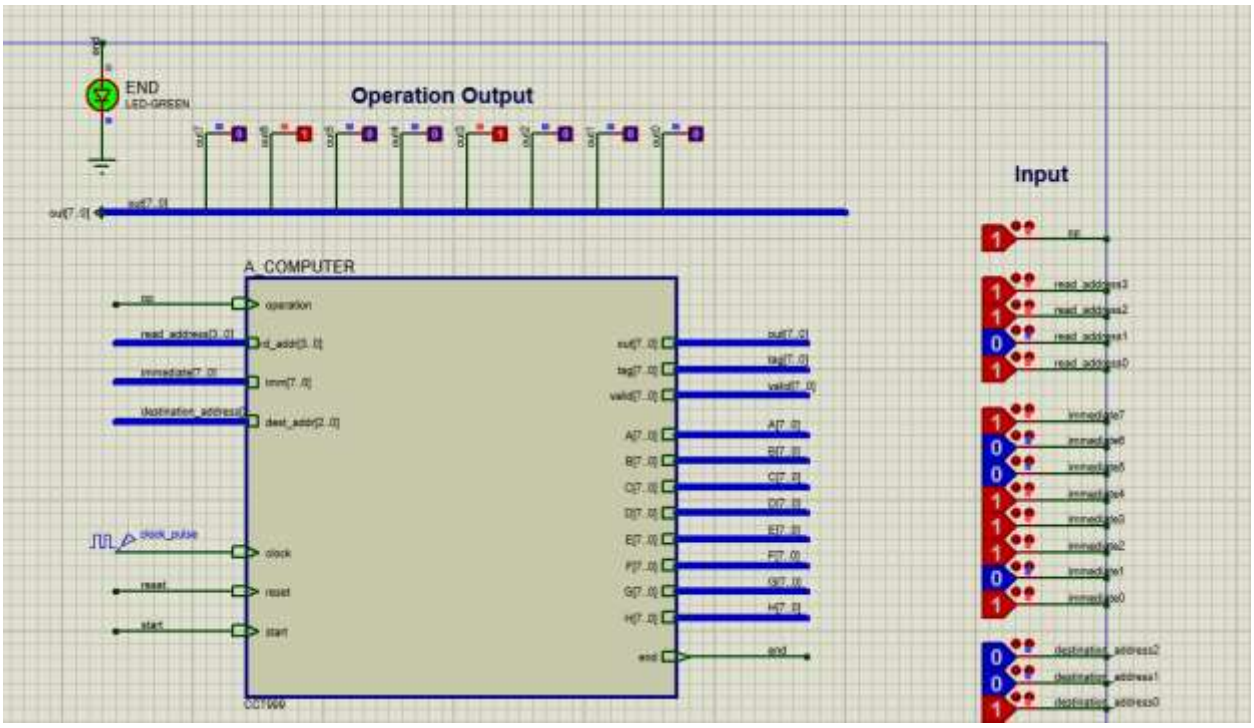

Valids
Tags
Cache Values
A[7..0]
B[7..0]
C[7..0]
D[7..0]
E[7..0]
F[7..0]
G[7..0]
H[7..0]

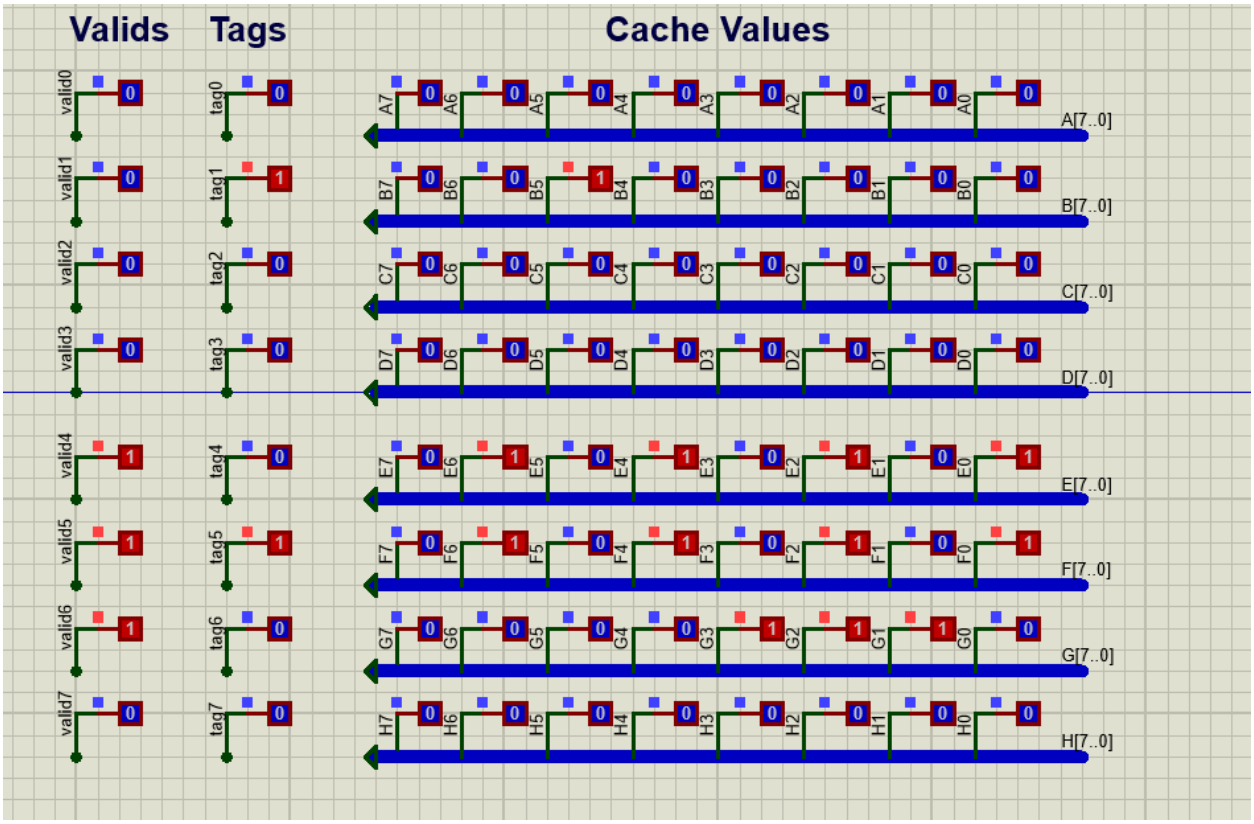
دستور سوم:


END
LED-GREEN
Operation Output
A_COMPUTER
Input
clock_pulse

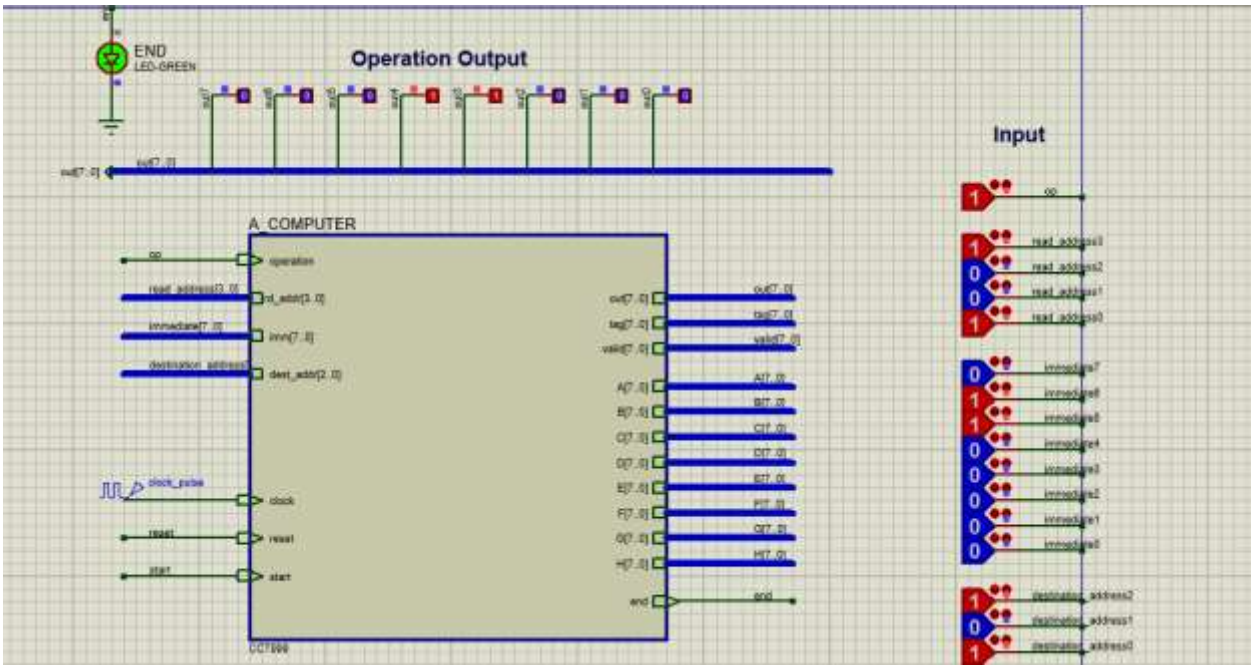

Valids
Tags
Cache Values
A[7..0]
B[7..0]
C[7..0]
D[7..0]
E[7..0]
F[7..0]
G[7..0]
H[7..0]

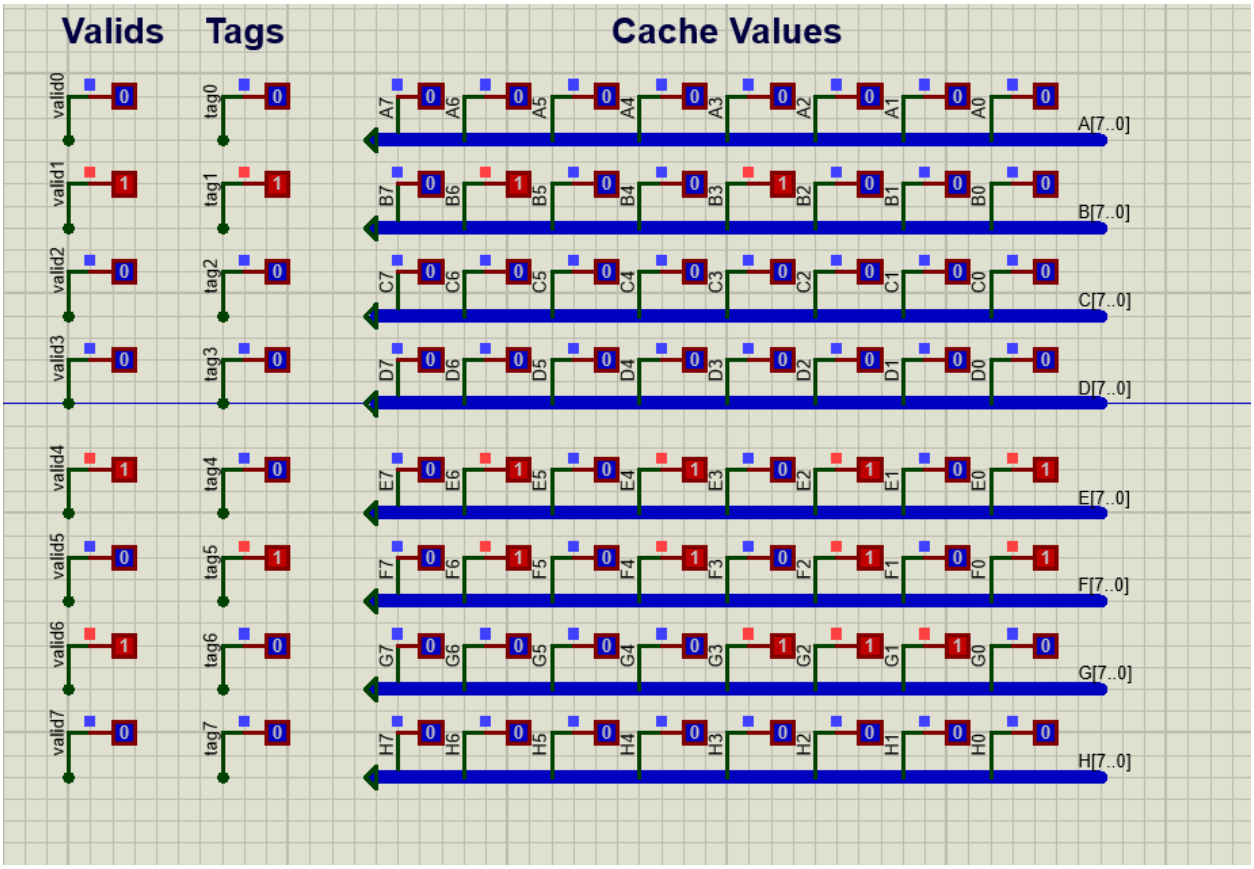
دستور چهارم:


END
LED-GREEN
Operation Output
Input
A_COMPUTER
operation
clock
reset
start
end


Valids
Tags
Cache Values
valid0
valid1
valid2
valid3
valid4
valid5
valid6
valid7
tag0
tag1
tag2
tag3
tag4
tag5
tag6
tag7
A[7..0]
B[7..0]
C[7..0]
D[7..0]
E[7..0]
F[7..0]
G[7..0]
H[7..0]

دستور پنجم:


END
LED-GREEN
Operation Output
Input
A_COMPUTER
operation
clock
reset
start
and


Valids
Tags
Cache Values
valid0
valid1
valid2
valid3
valid4
valid5
valid6
valid7
tag0
tag1
tag2
tag3
tag4
tag5
tag6
tag7
A[7..0]
B[7..0]
C[7..0]
D[7..0]
E[7..0]
F[7..0]
G[7..0]
H[7..0]