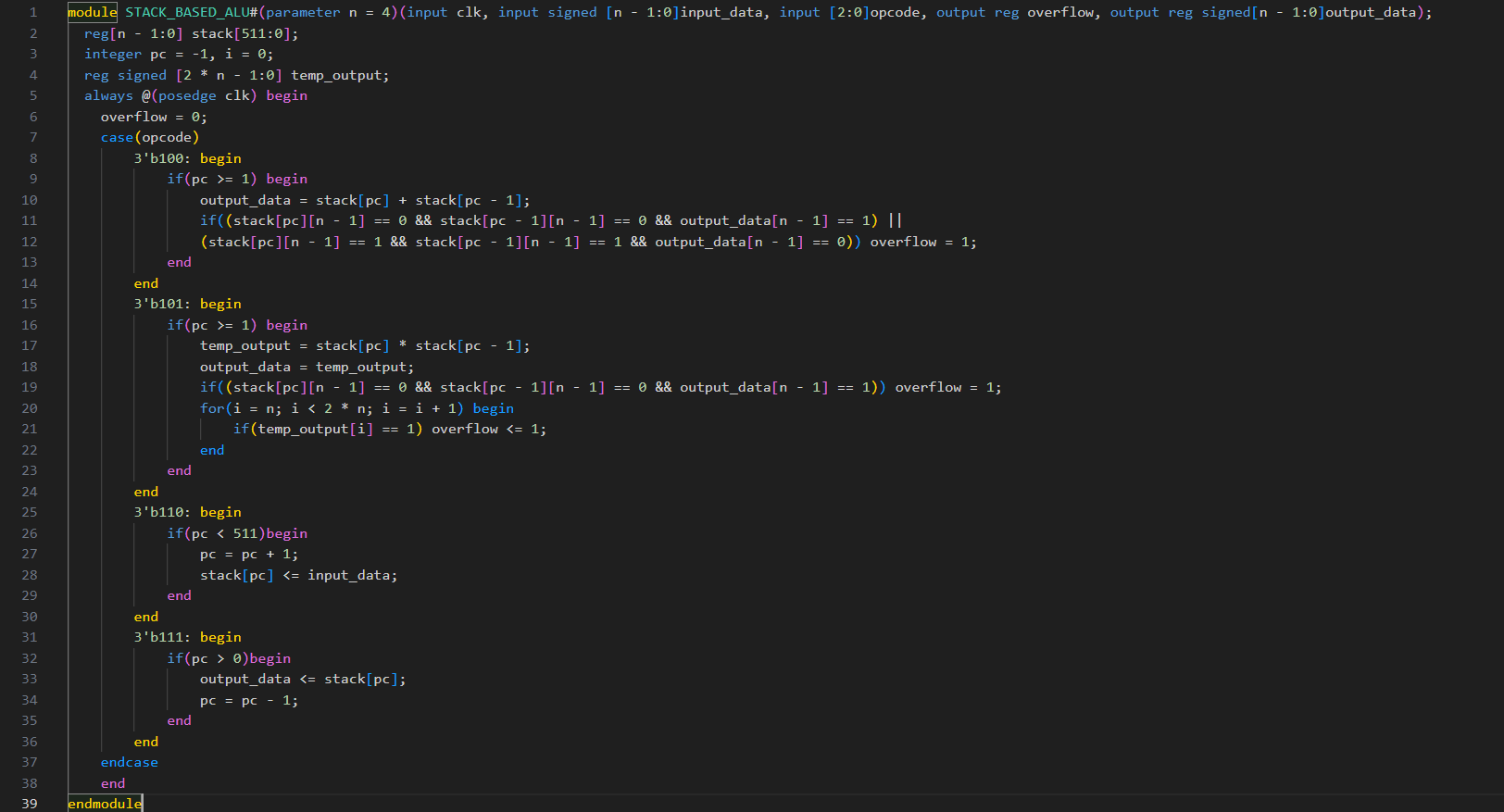
بسم الله الرحمن الرحیم

سوال ۱)

کد این سوال در فایل Q1\_A در فایل‌های ارسالی قابل مشاهده است. توضیح کد این سوال:  


در ابتدا که پارامتر های گفته شده در سوال به عنوان ورودی خروجی ماژول تعریف شده‌اند. ابعاد input\_data و output\_data متناسب با n است که در ابتدا توسط پارامتر ذکر شده است و در instance گیری قابل تغییر است.

سپس یک استک به طول دیتاها و عمق ۵۱۲ تعریف می‌کنیم. در خط بعد سه متغیر pc و i و temp\_out تعریف می‌شوند که در ادامه کاربرد آن‌ها را توضیح خواهم داد.

بلاک always با لبه مثبت کلاک فعال می‌شود. در این بلاک در ابتدا مقدار overflow صفر می‌شود که اگر از دفعه قبل یک مانده بود، صفر شود. سپس با توجه به opcode ورودی با فرمت case بررسی می‌کنیم که باید چه عملیاتی روی استک و ... انجام گیرد:

(قبل از توضیح هر case ، pc همان اشاره‌گر به اولین خانه استک است. یعنی اگر ما یک عدد را در استک اضافه می‌کنیم، pc به همان خانه اخر اشاره‌ میکند(همان program counter ))

100-add :

ابتدا چک می‌شود که ایا حداقل دو مقدار درون استک داریم که آن دو را باهم جمع کنیم یا نه. برای این‌کار چک می‌کنیم که pc بزرگتر از 1 باشد.(یعنی حداقل در خانه 0 و 1 آن مقداری موجود باشد). اگر این شرط برقرار بود دو خانه بالایی استک را با هم جمع کرده و درون data\_out قرار می‌دهیم.

حال باید overflow را چک کنیم. در شرطی که برای یک کردن overflow آمده ‌است بیان می‌شود که اگر جمع دو عدد مثبت منفی شد **یا** جمع دو عدد منفی مثبت شد، overflow یک بشود.

101-mult :

برای ضرب نیز ابتدا همان شرط وجود حداقل دو مقدار درون استک را مانند بخش قبل چک میکنیم. سپس ضرب دو مقدار روی استک را درون temp\_out قرار می‌دهیم که اندازه‌اش دو برابر ورودی هاست و این بدین معناست که در آن overflow رخ نمی‌دهد.(اساس این کار برای این است که بتوانیم تشخیص دهیم ایا در واقع overflow رخ داده است یا خیر، که در ادامه توضیح می دهم)

مقدار temp\_out را درون output\_data قرار می‌دهیم. ممکن است مقداری از مقادیر سرریز شوند که این امر را در خطوط بعدی چک می‌کنیم. ابتدا به طور جدا چک می‌شود که ضرب دو عدد مثبت منفی نشود. ولی چون ضرب است ممکن است بیت های سر ریز صرفا باعث تغییر مثبت و منفی شدن عدد نشوند. برای همین یک فور میزنیم که در آن چک می‌شود ایا متغیر temp\_out از خانه n به بعد، مقداری برابر با 1 دارد یا خیر؟ اگر داشته باشد که یعنی سرریز رخ داده است و output\_data ما نتوانسته آن مقدار را نشان دهد و اگر هم هیچ خانه‌ای 1 نباشد یعنی overflow رخ نداده‌است.

110-push:

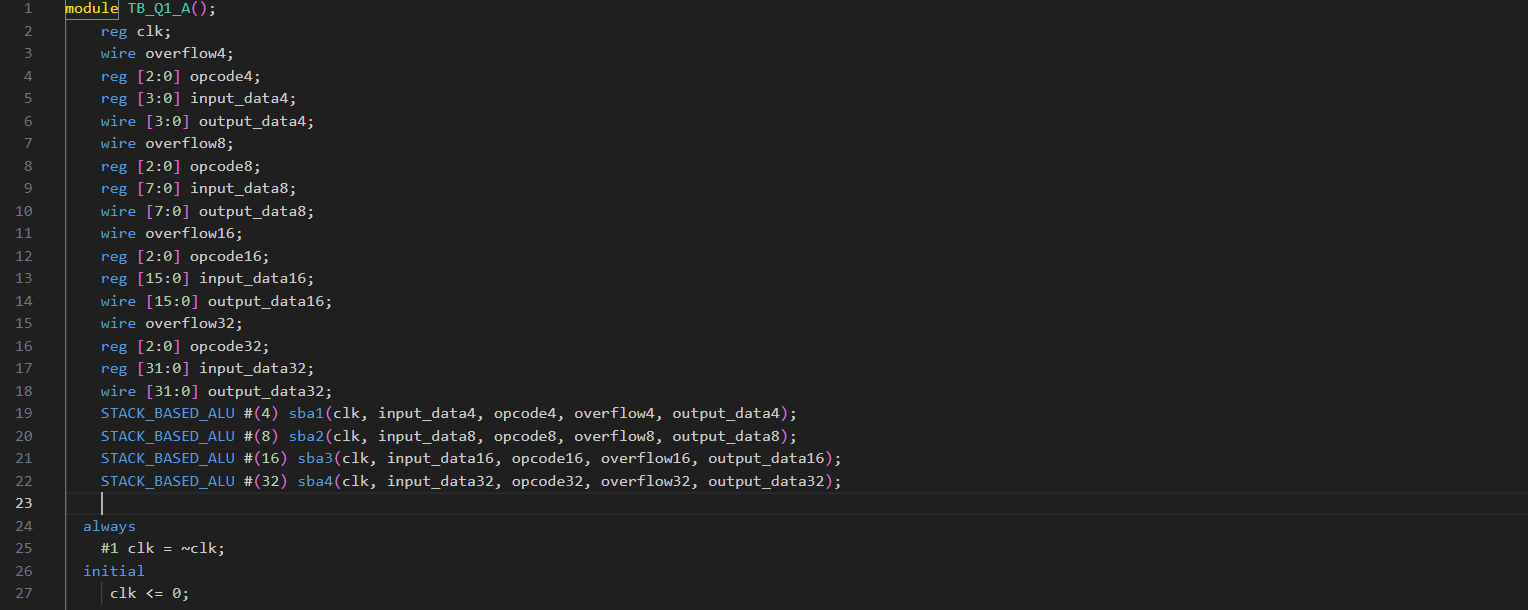
ابتدا با شرط این‌که pc کوچکتر از ۵۱۱ باشد چک می‌شود که استک پر نباشد. اگر پر نبود، اشاره‌گر pc را یک واحد افزایش داده و سپس مقدار درون input\_data را درون بالاترین نقطه استک یعنی همان جایی که pc به آن اشاره دارد قرار می‌دهیم.

111-pop:

ابتدا با شرط این‌که pc بزگتر از ۰ باشد چک می‌کنیم که استک خالی نباشد. و اگر خالی نبود، مقداری که در بالاترین خانه استک قرار دارد را( یعنی همان محلی که pc به آن اشاره می‌کند) در output\_data قرار می‌دهیم و سپس مقدار pc را یک واحد کاهش می‌دهیم.

شرح کد اصلی به تفصیل بیان شد. اما بخش testbench که در بخش الف از ما خواسته‌شده است:

کد testbench در فایلی با نام TB\_Q1\_A در فایل‌های ارسالی قابل مشاهده است. توضیح آن به شرح زیر است:

****

ابتدا برای هر کدام از بخش های 4و 8 و 16 و 32 بیتی، متغیر های overflow و input\_data و output\_data را مخصوص آنها تعریف می‌کنیم.(متغیر clk یا همان کلاک عمومی است و نیاز به تعریف اختصاری ندارد). همانطور که قابل مشاهده هست ابعاد های متغیر ها متناسبا متفاوت است.

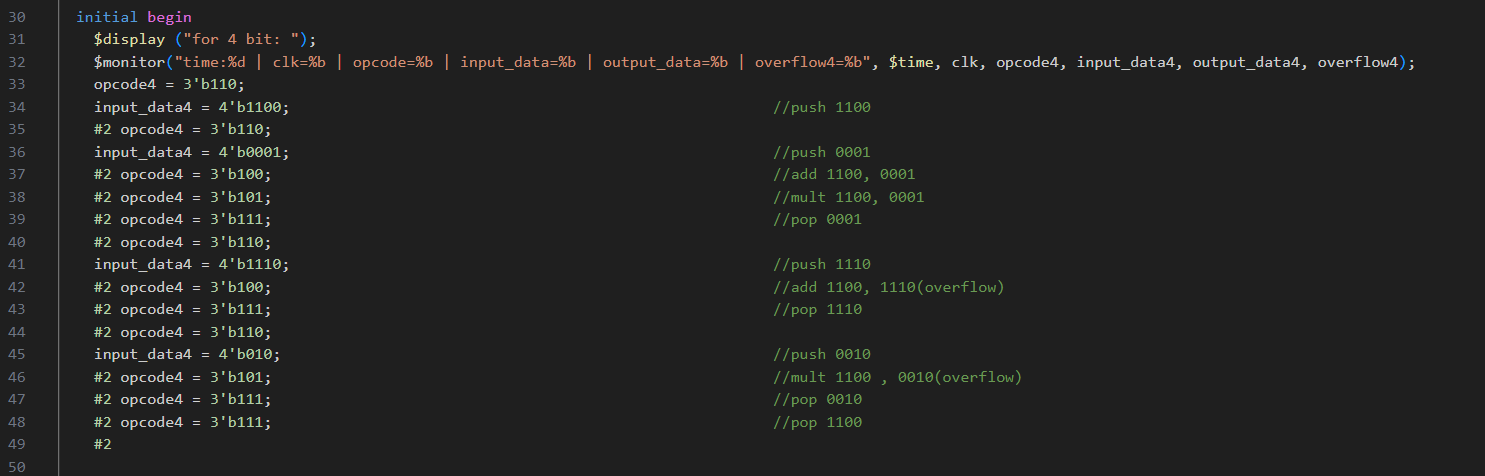
سپس با ۴ بار از ماژول اصلی instance می‌گیریم که به هرکدام متغیرهای مخصوص خودش را می‌دهیم. بطور مثال به instance‌ ای که برای 16 بیتی ها گرفته شده است، ورودی، خروجی های مخصوص 16 بیتی را می‌دهیم.

با استفاده از یک always یک خطی کلاک را هر یک واحد زمانی یک‌بار not می‌کنیم تا واقعا تبدیل به clock شود!

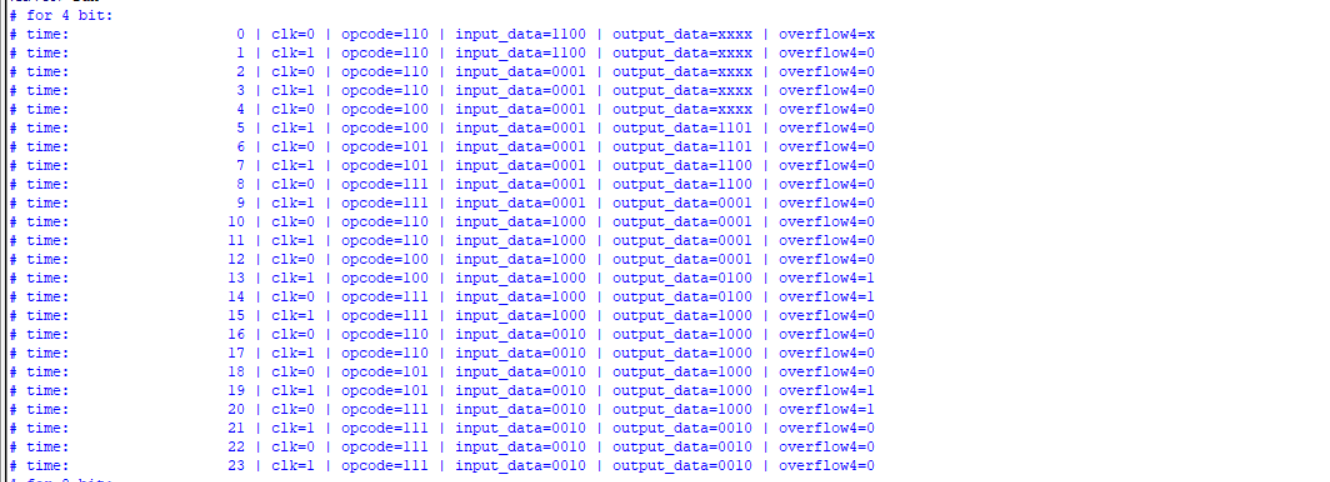
و در بخش بعد نیز مقدار اولیه آن را صفر قرار می‌دهیم.

در ادامه ما حالات مختلف را برای حالت های 4 و 8 و 16 و 32 مورد بررسی قرار داده‌ایم. در ادامه به توضیح بخش 4 بیتی میپردازم و بخش های دیگر نیز به همین شیوه بررسی شده اند و دیگر نیاز به توضیح مجدد آنها نیست و صرفا تصاویر کد و نتیجه شبیه‌سازی آن را در این فایل قرار می‌دهم.

توضیح بخش ۴ بیتی:



نتیجه شبیه‌سازی:

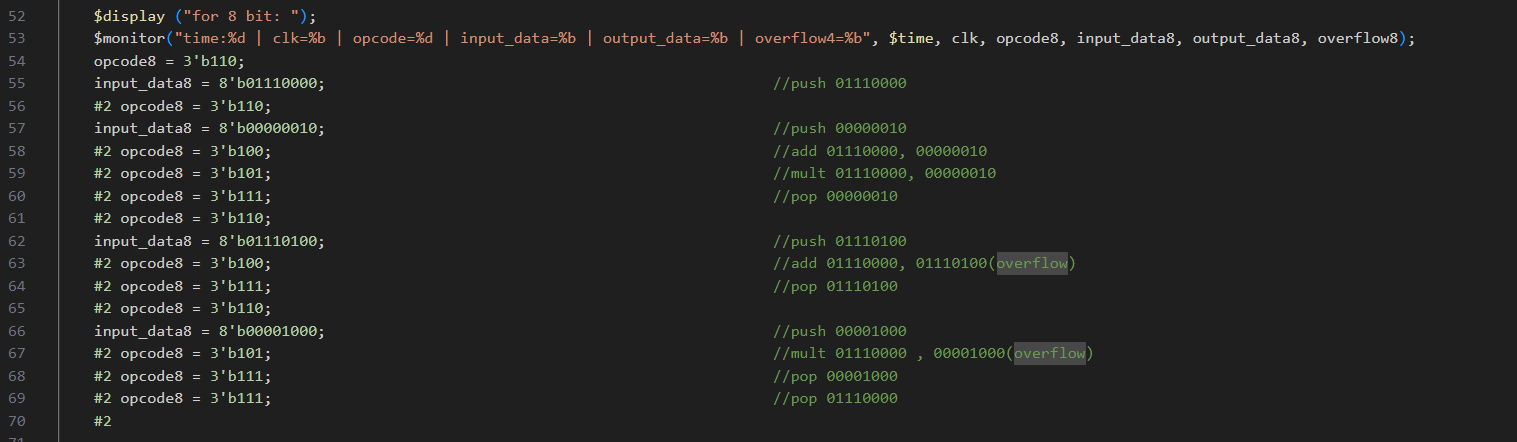


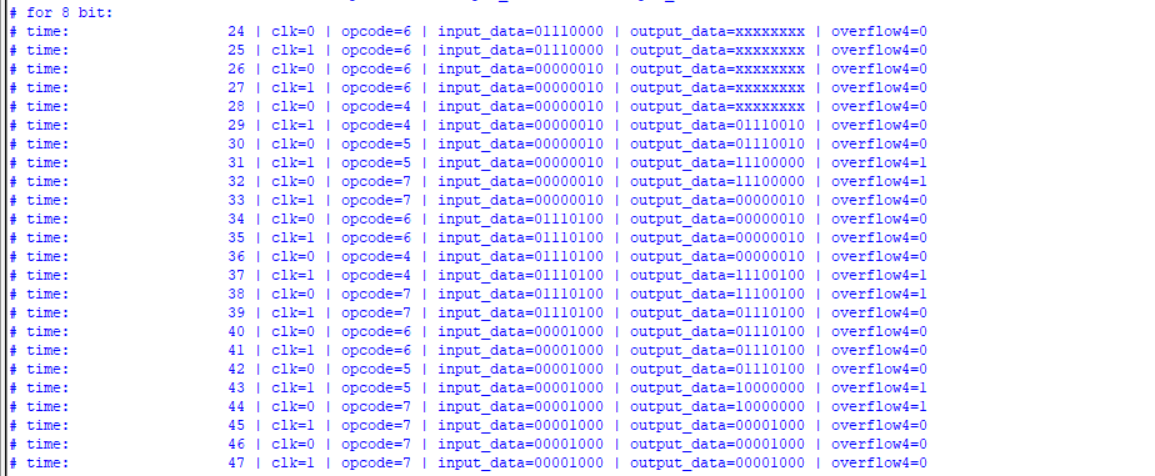
در خط اول که یک مانیتور هست که به ازای تغییر هر کدام از مقادیر نام برده در آن ، یک دور چاپ می‌شود.

ابتدا opcode را روی حالت push قرار داده و input\_data را نیز برابر 1100 قرار میدهیم. پس از یک واحد زمانی این تغیرات اعمال می‌شود. حال ما یک ثانیه دیگر نیز صبر می کنیم تا کلاک در سطح پایین قرار گیرد. در مجموع دو واحد زمانی باید بین دستورات وقفه انداخت. پس از دو واحد زمانی وقفه با همان شیوه قبل عدد 0001 را push می‌کنیم. سپس opcode را روی حالت add قرار می‌دهیم تا نتیجه جمع در output\_data قابل مشاهده باشد. سپس opcode را روی حالت ضرب قرار میدهیم.( این دو حالت بدون تست overflow بودند)

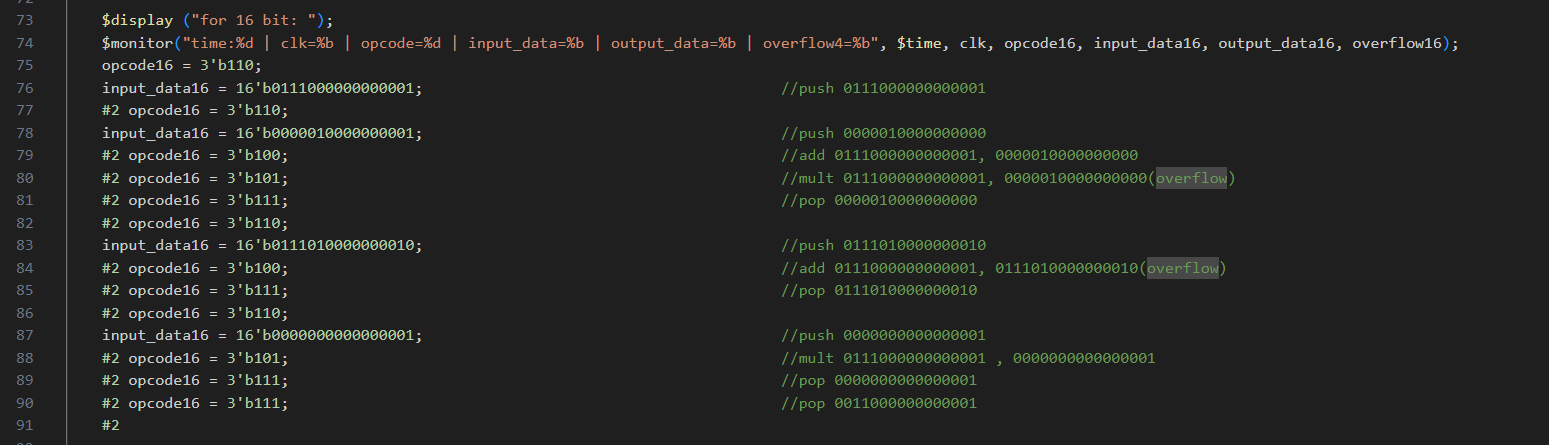
حالا مقدار روی استک را یعنی مقدار 0001 را با قرار دادن opcode روی حالت pop، برمیداریم که یعنی این مقدار در آن لحظه در output\_data قابل مشاهده است. سپس با تنظیم opcode و input\_data عدد 1000 را به داخل استک میفرستیم. در مرحله بعد این دو را با هم جمع کرده که با وجود مشاهده نتیجه در output\_data، بیت overflow نیز یک می‌شود. سپس عدد 1000 را خارج کرده و عدد 0010 را push می‌کنیم. حال opcode را روی حالت ضرب قرار می‌دهیم تا دو عدد بالایی استک را یعنی 1100 و 0010 را با هم ضرب کند. این ضرب نیز دارای سرریز است و در نتیجه بیت overflow یک میشود. در انتها نیز دو مقدار درون استک را با دستو pop خارج می‌کنیم.

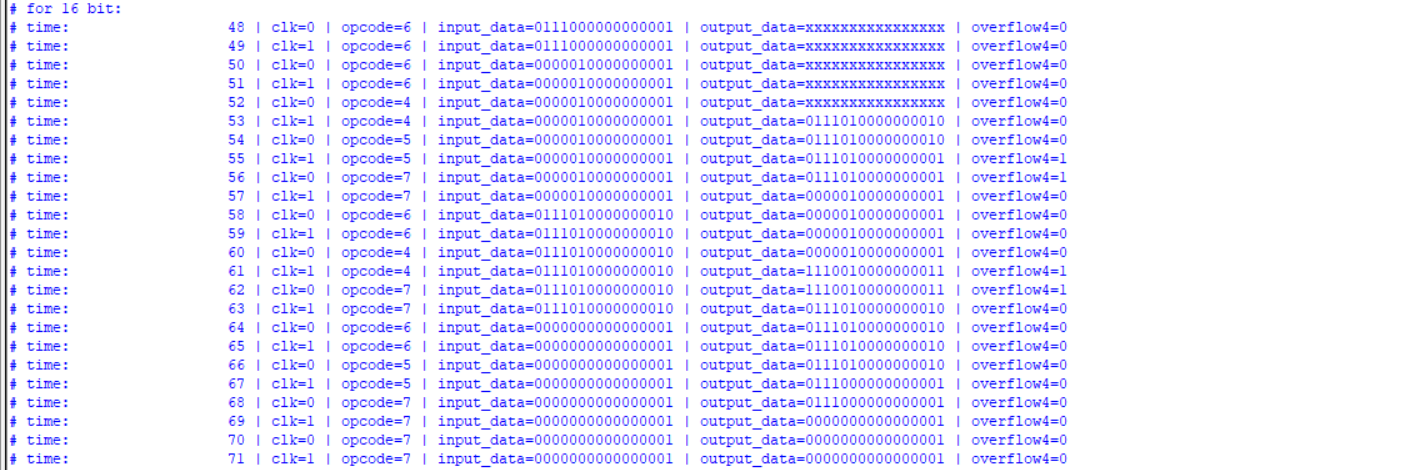
این توضیح برای بخش ۴ بیتی بود. برای بخش ۸ و ۱۶ و ۳۲ نیز به همین صورت است. اما دیگر نیاز به توضیح نیست و صرفا تصاویر کد و نتیجه‌ آن را در این گزارش می‌اورم:



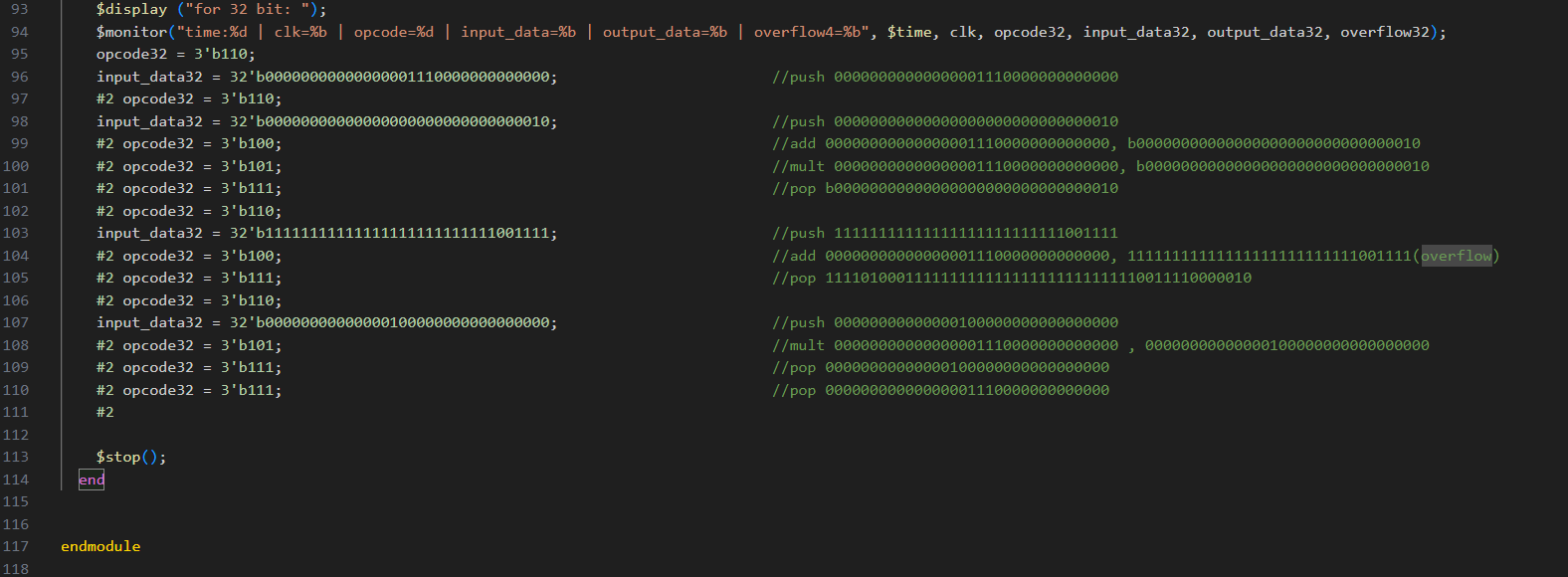
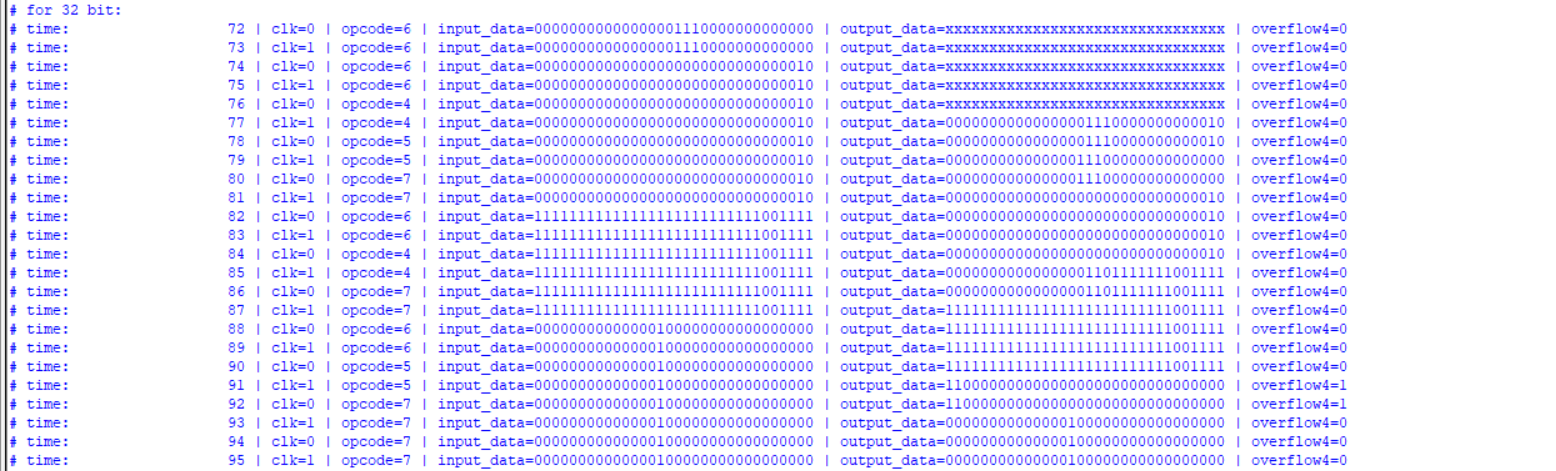


و:





و:



کد ها و نتیجه آن ها بخش به بخش نمایش داده شده است. در صورت نیاز می‌توانید بخش های مختلف کد را شبیه‌سازی کنید و نتیجه آن را مشاهده کنید.

با تشکر از لطف شما

احسان محترم - 401106458