

**دانشگاه صنعتی شریف**

**دانشکده مهندسی کامپیوتر**

**پروژه جبرانی طراحی سیستم­های دیجیتال**

**پوریا غفوری**

**401106277**

**استاد: جناب آقای امین فصحتی**

**نیم سال دوم 1403-1402**

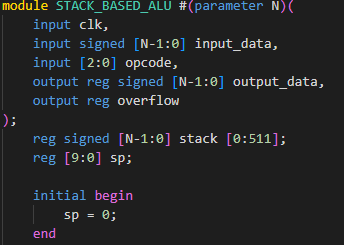
# هدف پروژه:

در این پروژه قصد داریم که یک STACK BASED ALU را با کمک Verilog طراحی کرده که می­تواند اعداد را در خود پوش کند یا آن­ها را پاپ کند و همچنین می­تواند حاصل ضرب و یا حاصل جمع دو عدد بالای استک را خروجی دهد.

در بخش دوم این پروژه باید با کمک Verilog یک ماژول دیگر طراحی می­کردیم که عبارات ریاضی را ورودی بگیرد و آن­ها را به صورت پسوندی در بیاورد و سپس با کمک ماژول STACK BASED ALU مقدار آن­ها را حساب کند.

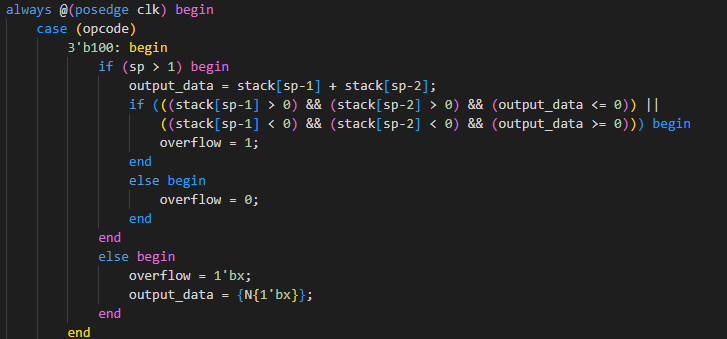
# نحوه پیاده­سازی:

ابتدا ماژول STACK BASED ALU را تشریح می­کنیم و نحوه کارکرد آن را توضیح می­دهیم. قابل ذکر است که کد Verilog آن در پوشه code و با نام STACK\_BASED\_ALU.v قرار دارد.

در ابتدای کار آن را با پارامتر ورودی N (که سایز اعداد ما می­باشد) تعریف می­کنیم و ورودی و خروجی­های مورد نیاز آن را قرار می­دهیم که این ورودی و خروجی­ها همان­هایی هستند که در صورت سوال بیان شده­اند و من فقط clk را به آن اضافه کرده­ام که اعمال ماژول ما در لبه بالارونده کلاک انجام گیرد.

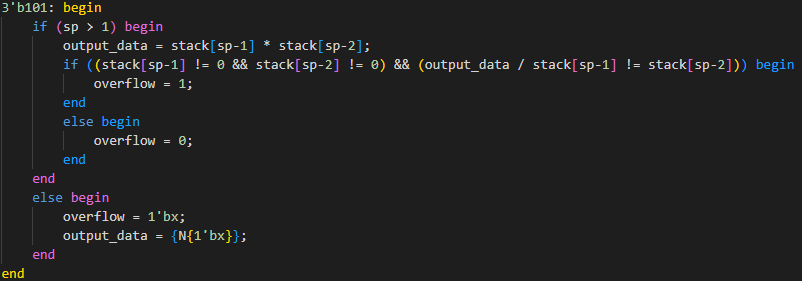
در ادامه متغیر­های مورد نیاز stack و sp را تعریف کردم که stack همان حافظه ما می­باشد که عرض آن به اندازه بیت­های اعداد ما می­باشد و عمق آن در صورت سوال بیان نشده بود و من آن را 512 قرار دادم به این معنا که Stack ما می­تواند 512 عدد N بیتی را در خود جای دهد. متغیر sp هم همان Stack pointer ما می­باشد.

در ادامه در بلاک initial مقدار sp را برابر صفر قرار دادم تا هر وقت از ماژول خود یک instance گرفتیم stack pointer آن به خانه اول اشاره کند.

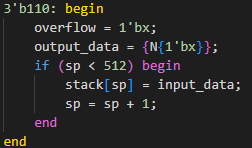


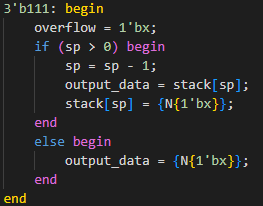
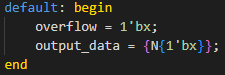
در عکس بالا داخل بلاک always را نشان می­دهیم که با لبه بالارونده کلاک کار می­کند و هر بار که کلاک می­خورد opcode ورودی را بررسی می­کند و بر اساس آن عمل می­کند.

در این تصویر اگر opcode برابر 100 بود به این معنا است که عملیات جمع باید انجام گیرد پس ما بررسی می­کنیم که آیا اعداد موجود در استک بیشتر از یکی می­باشد یا خیر (چون می­خواهیم دو عدد بالای استک را با یکدیگر جمع کنیم) و اگر این تعداد بیشتر از یک بود دو عنصر بالای استک را با هم جمع کرده و در output\_data می­ریزیم و سپس برای بررسی overflow چک می­کنیم که اگر هر دو عنصر جمع ما هم علامت بودند اما جواب جمع علامتی متفاوت با آن­ها داشت به این معنا است که overflow رخ داده و بیت آن را یک می­کنیم و در غیر این صورت عملیات جمع بدون سر ریز انجام شده و مقدار بیت آن را صفر قرار می­دهیم. همچنین اگر تعداد اعداد موجود در استک کمتر از دو بود مقدار X را در output\_data و overflow می­ریزیم تا مقداری اشتباه را نشان ندهند.



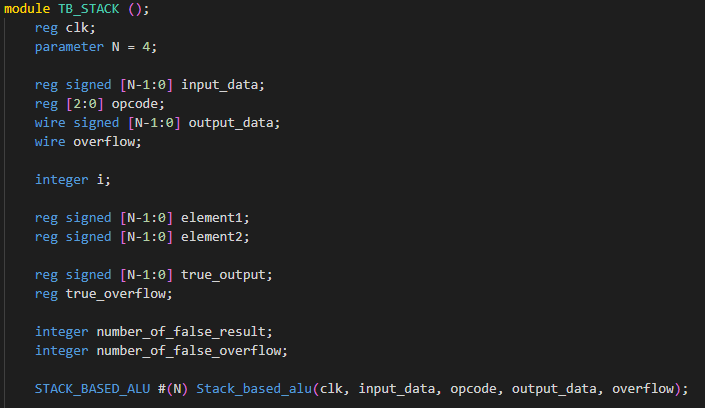
در کیس بعدی بررسی می­کنیم که آیا opcode برابر 101 می­باشد یا خیر که اگر برابر این مقدار بود به این معنا است که عملیات ضرب باید انجام شود. در ابتدا مشابه جمع بررسی می­کنیم که آیا تعداد اعداد موجود در استک بیشتر از یکی است یا خیر که اگر بیشتر نبود مقدار X را در خروجی­ها می­ریزیم و اگر بیشتر از یک عدد در استک بود عملیات ضرب را انجام داده و خروجی آن را در output\_data می­ریزیم. برای بررسی overflow هم ابتدا بررسی می­کنیم که اعدادی که عملیات ضرب روی آن­ها انجام می­شود برابر صفر نباشند (چون اگر صفر باشند جواب ضرب هم صفر شده و نمی­تواند overflow رخ دهد) حال اگر صفر نبودند جواب ضرب را تقسیم بر عدد اول می­کنیم اگر جواب تقسیم مخالف عدد دوم شد به این معنا است که overflow رخ داده و مقدار اشتباهی در output\_data می­باشد و بیت overflow را یک می­کنیم و در غیر این صورت بیت overflow را برابر صفر قرار می­دهیم.

در این کیس بررسی می­کنیم که opcode برابر 110 می­باشد یا خیر که اگر برابر بود عملیات push باید صورت گیرد در ابتدای کار خروجی­های ماژول را برابر X قرار می­دهیم چون خروجی نداریم و بعد از آن بررسی می­کنیم که آیا استک خالی است یا خیر و اگر استک پر بود کاری انجام نمی­دهیم، اما اگر استک خالی بود عدد ورودی را در استک قرار داده و sp را یکی بالاتر می­بریم.

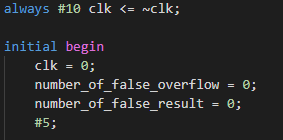
در این کیس بررسی می­کنیم که opcode برابر 111 می­باشد یا خیر که اگر برابر بود عملیات pop باید صورت گیرد. در ابتدا بیت overflow را X می­کنیم چون معنایی ندارد. در ادامه بررسی می­کنیم که آیا استک خالی است یا خیر. اگر خالی بود که X را در خروجی می­ریزیم اما اگر پر بود مقدار آن را در خروجی ریخته و آن خانه از استک را X می­کنیم و sp را یکی پایین­تر می­بریم.

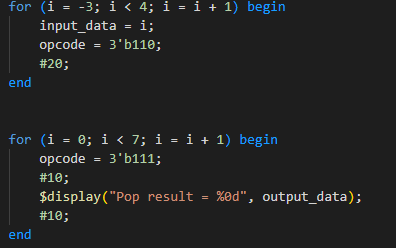
در آخر هم بررسی می­کنیم که اگر opcode برابر هیچکدام از opcodeهای مشخص شده نبود کاری انجام نمی­دهیم و خروجی­ها را X می­کنیم.

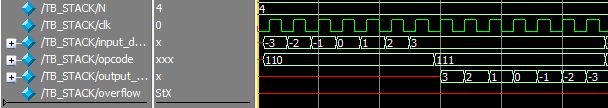
در اینجا ماژول STACK BASED ALU به پایان رسید حال نیاز داریم که برای آن Test bench بنویسیم تا از درستی ماژول خود مطمئن شویم. Test bench این ماژول در پوشه code و با نام TB\_STACK.v موجود می­باشد. در ادامه به توضیح آن و بررسی ورودی­ها و خروجی­های آن می­پردازیم.

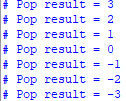


در ابتدا کلاک و پارامتر N را تعریف می­کنیم. سپس ورودی­ها و خروجی­های ماژول STACK BASED ALU را تعریف می­کنیم در ادامه نیز یک اینتیجر i تعریف می­کنیم که برای loop هایی که در تست نوشتیم از آن استفاده کنیم. همچنین متغیر­های element1 و element2 را تعریف کردم که عناصری که با یکدیگر جمع یا ضرب می­شوند را در آن­ها بریزم تا بتوانم با استفاده از تابع­هایی که تعریف کرده­ام(در ادامه آن­ها را توضیح خواهم داد) صحت عملکرد ماژول خود را بررسی کنم. متغیر­های true\_output و true\_overflow هم خروجی­های تابع­ها می­باشند. متغیر­های number\_of\_false هم تعریف کردم تا با مقایسه خروجی­های ماژول و تابع­ها بررسی کنم که چه تعداد از خروجی­های ماژول­ها غلط­اند و در آخر هم از ماژول STACK BASED ALU یک instance گرفتم و ورودی­ها و خروجی­های آن را ست کردم.

در ادامه کلاک خود را تنظیم کردم و مقادیر number\_of\_false را برابر صفر قرار دادم.

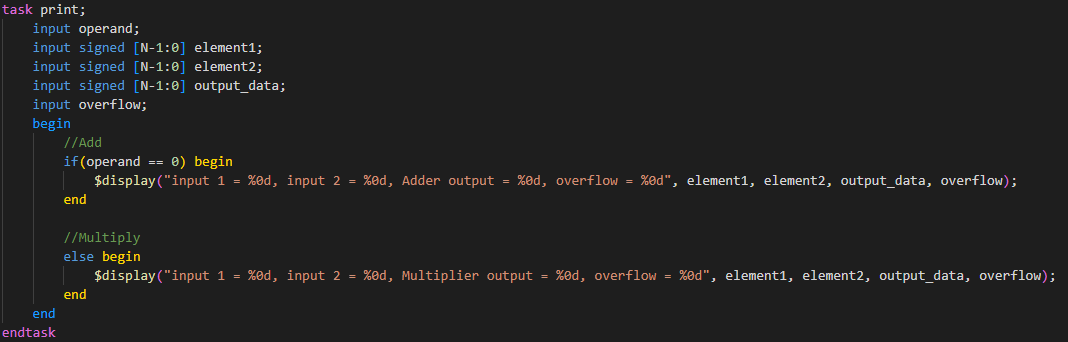
در تست اول صرفا عملکرد پوش و پاپ و ذخیره­سازی اعداد در استک را بررسی کردم در ابتدا اعداد منفی سه تا سه را در استک پوش کردم و سپس اعداد را پاپ کردم و مقادیر آن­ها را بررسی کردم که خروجی آن را می­توانید در تصاویر زیر مشاهده کنید. (منطقا این قسمت به پارامتر N بستگی ندارد و به ازای Nهای متفاوت خروجی ما تغییر نخواهد کرد)



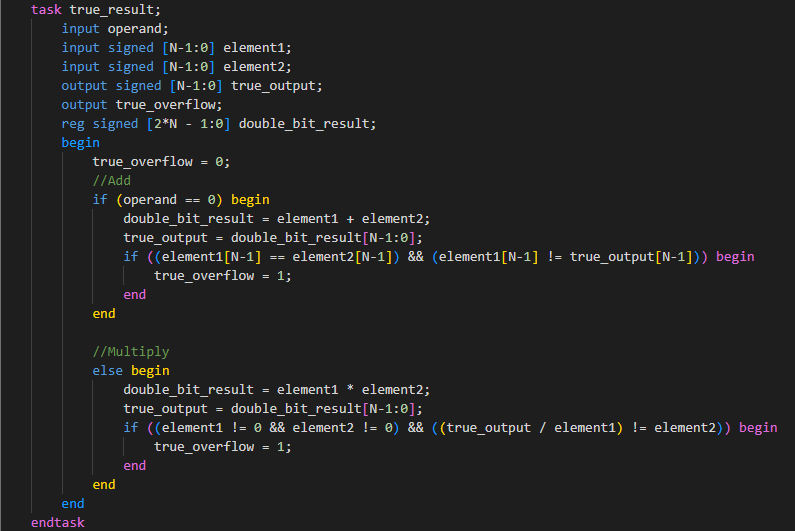


همان طور که در تصاویر مشاهده می­کنید اعداد به درستی در استک ذخیره شده بودند و به صورت برعکس در خروجی قابل مشاهده می­باشند. بیت overflow هم X می­باشد چون مقدار آن در پوش و پاپ معنایی ندارد و همچنین خروجی استک هنگام پوش کردن X است چون خروجی آن هنگام پوش کردن معنایی ندارد.

ابتدا تابع­هایی که تعریف کرده­ام را بررسی می­کنیم و سپس به بررسی ادامه تست­ها می­پردازیم:



ابتدا تابع print را تعریف کردم که ورودی­های آن operand(که بررسی می­کند که چه عملی دارد صورت می­گیرد تا عبارت صحیح را چاپ کند) و element1 و element2 و output\_data و overflow هستند که دو عنصر اول اعدادی هستند که عملیات روی آن­ها انجام می­شود و دو ورودی آخر هم خروجی­های ماژول ما می­باشند. در ادامه نیز بر اساس این که کدام عملیات انجام می­شود عبارت مناسب را چاپ می­کنیم.

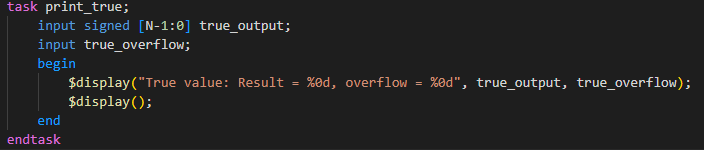


در ادامه تابع true\_result را تعریف کردم که ورودی اول آن operand است که بفهمد چه عملیاتی باید انجام شود. دو ورودی بعدی آن (element1 و element2) دو عددی هستند که عملیات روی آن­ها انجام می­شود. دو خروجی آن مقدار صحیح عملیات ضرب یا جمع و مقدار صحیح بیت overflow می­باشد. در ادامه یک متغیر تعریف کردم که تعداد اندازه آن دوبرابر اعداد ورودی هستند تا بتوانیم با کمک آن مقادیر صحیح را حساب کنیم(در ادامه دقیق­تر توضیح خواهم داد)

ابتدای کار که مقدار overflow را برابر صفر قرار می­دهم(اگر در جایی overflow رخ دهد آن را یک خواهم کرد)

ابتدا بررسی می­شود که اگر باید جمع انجام شود مقدار صحیح آن را حساب کنیم. ابتدا حاصل جمع دو عدد را حساب کرده و در متغیر بزرگ­تر میریزم سپس N بیت اول آن عدد را برمی­دارم و به عنوان خروجی صحیح نشان می­دهم. سپس منطق overflow را بر اساس ورودی­ها و مقدار صحیحی که از عملیات جمع به دست آمده بررسی می­کنم که اگر overflow رخ داده است مقدار بیت آن را یک می­کنم.

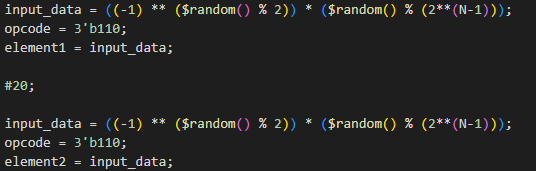
در ادامه عملیات ضرب بررسی می­شود که دو عدد را در هم ضرب کرده و در متغیر بزرگ­تر ذخیره می­کنیم و سپس N بیت اول آن را برمی­دارم و به عنوان خروجی صحیح قرار می­دهم(چون می­خواهم چک کنم که در هنگام overflow هم خروجی ضرب ALU ما به صورت صحیح split شده باشد و مقدار خروجی آن با N بیت اول حاصل ضرب دو عدد برابر می­شود یا خیر) سپس overflow را بررسی می­کنیم به این صورت که اگر هیچکدام از اعداد صفر نبودند و تقسیم جواب صحیح ضرب به یکی از اعداد برابر آن یکی عدد نشد به این معنا است که overflow رخ داده و بیت overflow را یک می­کنم.



سپس تابع print\_true را تعریف کردم که مقادیر صحیح جمع یا ضرب و بیت overflow را ورودی می­گیرد و این مقادیر صحیح را چاپ می­کند.

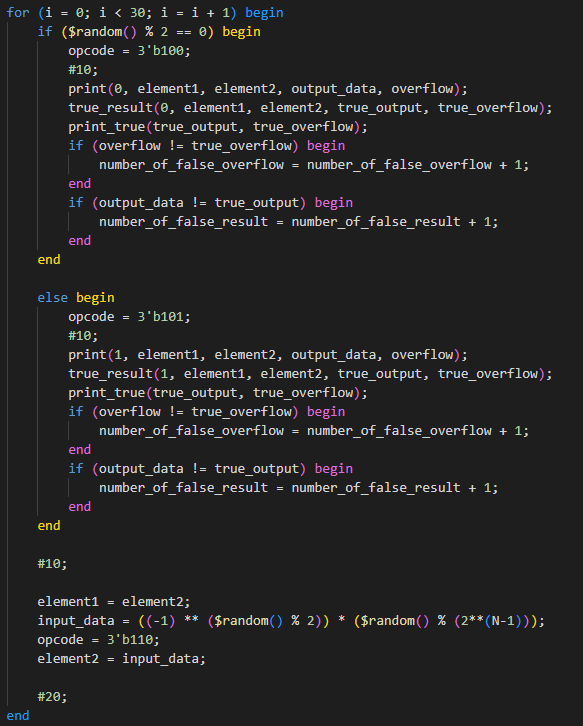
حال به بررسی تست­ها می­پردازیم:

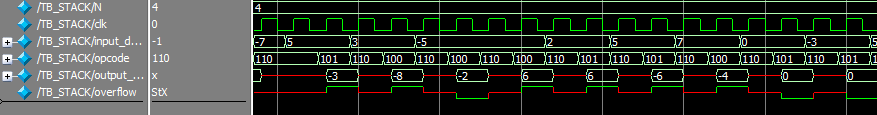
اولین تست به این صورت می­باشد که هر دفعه یک عدد رندوم N بیتی را در استک پوش می­کند و عملیات را به صورت رندوم روی آن انجام می­دهد.

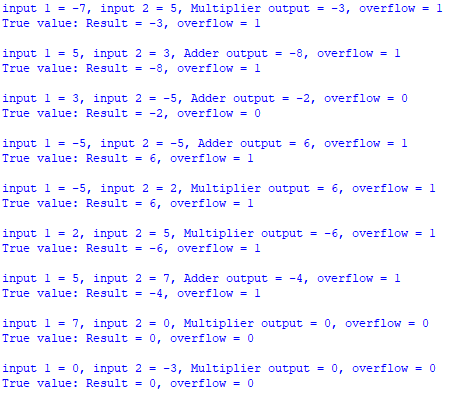


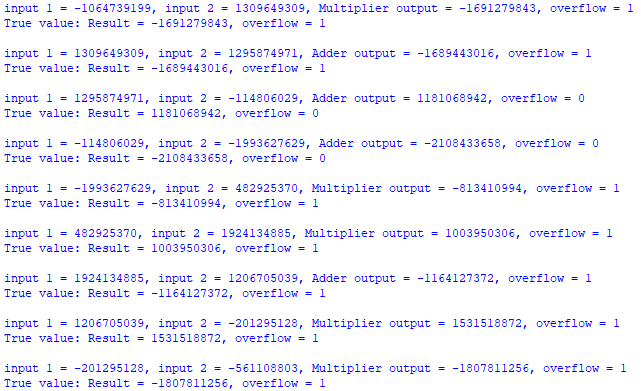
در ابتدای کار برای این تست دو عدد رندوم N بیتی را در استک پوش می­کنیم.(مقادیر element1 و element2 را هم برابر آن­ها قرار می­دهیم چون در ابتدا این دو عدد عملیات روی آن­ها انجام می­شوند)

سپس وارد for که در تصویر زیر آمده می­شویم و بین عملیات جمع و ضرب یکی را رندوم انتخاب کرده و آن را انجام می­دهیم.(تاخیر 10 بعد از opcode برای این آورده شده که عملیات انجام شود و سپس خروجی آن را بررسی کنیم) در ادامه پس از اینکه عملیات انجام شد خروجی­های مدار و اعدادی که عملیات بر روی آن­ها انجام شده است را چاپ می­کنیم. در خط بعدی مقدار صحیح خروجی­ها را بر اساس ورودی­ها حساب کرده و آن­ها را چاپ می­کنیم. در اخر هم بررسی می­کنیم که اگر خروجی­های ماژول ما با خروجی صحیح کی نبود به متغیرهای number of false یک واحد اضافه می­کنیم. در آخر حلقه هم یک عدد دیگر در استک پوش می­کنیم که در این حالت element1 ما همان element2 قدیمی می­شود و element2 ما برابر با مقدار جدیدی که در استک پوش کرده­ایم می­شود. این حلقه را 30 بار اجرا می­کنیم و بخشی از خروجی­های آن را در تصاویر زیر می­توانید مشاهده کنید.

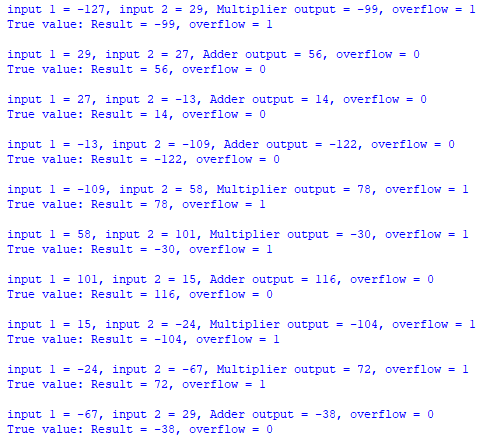


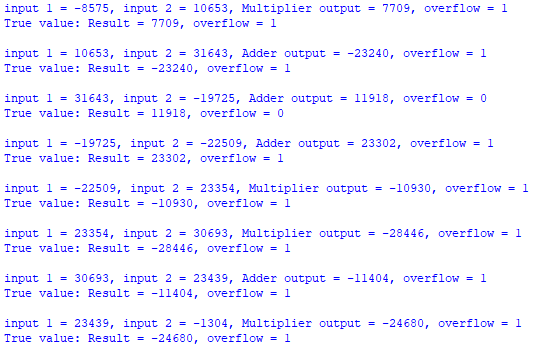


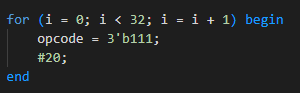
همان طور که در تصاویر مشاهده می­کنید برای N = 4 ماژول ما به درستی کار کرده و خروجی­های آن با خروجی­های صحیح یکی می­باشد.



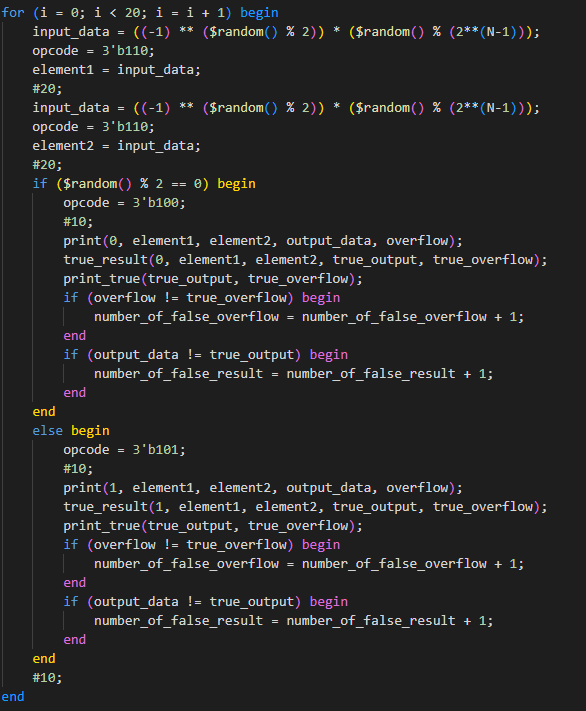
چند نمونه تست برای N = 32 که در اینجا هم ماژول به درستی کار کرده است.

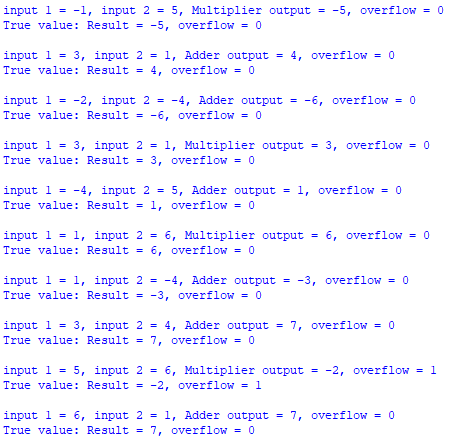
چند نمونه تست برای N = 8 که در اینجا هم ماژول به درستی کار کرده است.

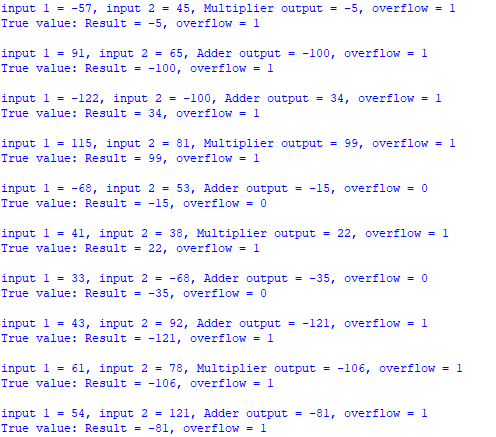
چند نمونه تست برای N = 16 که در اینجا هم ماژول به درستی کار کرده است.

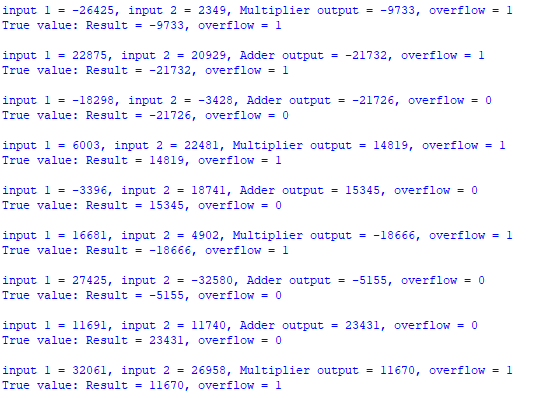
پس از انجام این تست استک را خالی می­کنیم که برای تست بعدی آماده شود.

در تست بعدی که در تصویر زیر آمده مشابه تست اول عمل می­کنیم تنها تفاوت آن این است که به جای اینکه یکی یکی در استک پوش کنیم تا عدد جدید با عددی که قبلا در استک است عملیات انجام دهد، دو عدد را در استک پوش می­کنیم و سپس عملیات را انجام می­دهیم تا هر دفعه عملیات با دو عدد جدید انجام شود. چند نمونه خروجی این تست هم در تصاویر زیر آمده است.

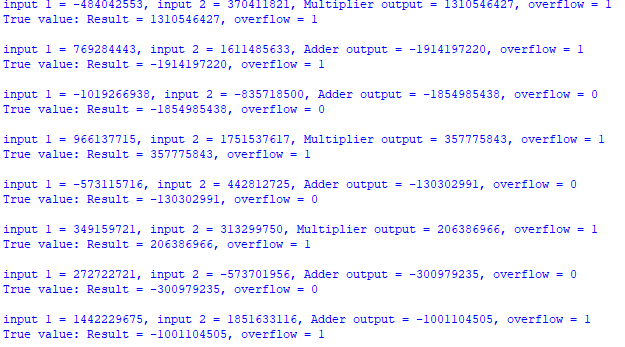


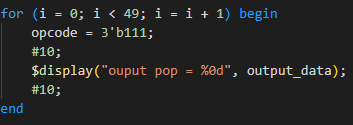
چند نمونه تست برای N = 4 که همان طور که می­بینید ماژول به درستی کار کرده است.

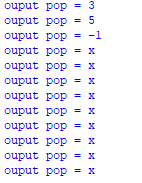
چند نمونه تست برای N = 8 که همان طور که مشاهده می­کنید ماژول به درستی کار کرده است.

چند نمونه تست برایN = 16 که همان طور که می­بینید ماژول به درستی کار کرده است.

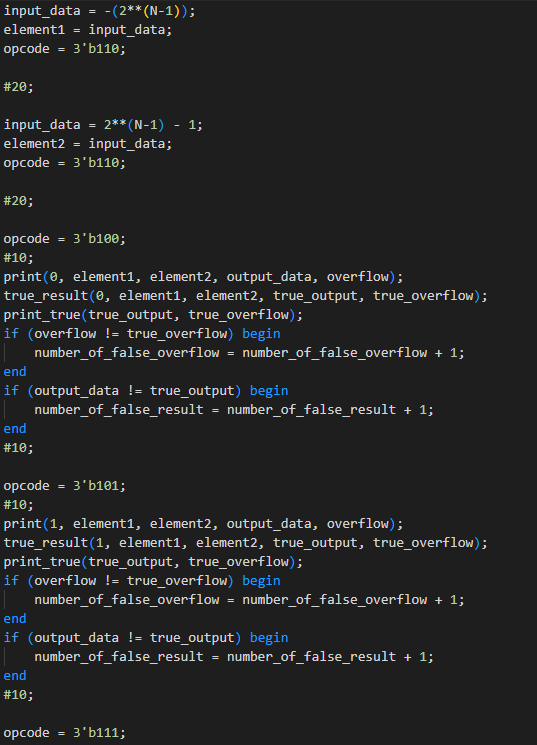
چند نمونه تست برای N = 32 که همان طور که می­بینید ماژول به درستی کار کرده است.

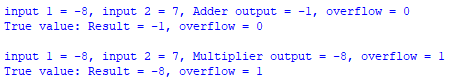


در تست بعدی همه عناصر موجود در استک را پاپ می­کنیم اما به تعداد بیشتر از مقادیری که پوش کرده بودیم(در استک 40 عدد پوش کرده بودیم اما در اینجا 49 عدد پاپ می­کنیم) پس منطقا باید 9 مقدار خروجی آخر ما برابر X باشد چون استک ما خالی می­باشد.

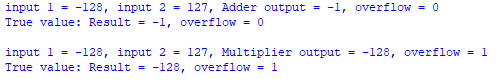
همان طور که مشاهده می­کنید در آخر کار مقادیری پاپ شده­اند و 9 پاپ آخر ما برابر X شده است چون استک خالی است.

در تست بعدی که در تصویر زیر می­بینید ما edge case ها را بررسی می­کنیم در ابتدا کوچکترین عدد منفی ممکن را در استک پوش می­کنیم و سپس بزرگترین عدد مثبت را پوش می­کنیم و ابتدای کار عملیات جمع را روی آن­ها انجام می­دهیم و سپس عملیات ضرب که خروجی­های آن­ها را در تصاویر زیر می­توانید ببینید.(در آخر کار هم بزرگترین عدد مثبت را پاپ می­کنم تا برای تست بعدی آماده شویم)

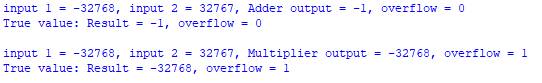




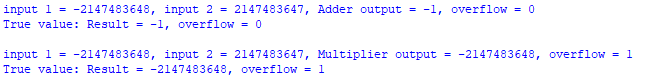
همان طور که مشاهده می­کنید ماژول ما برای N = 4 به درستی کار کرده است.



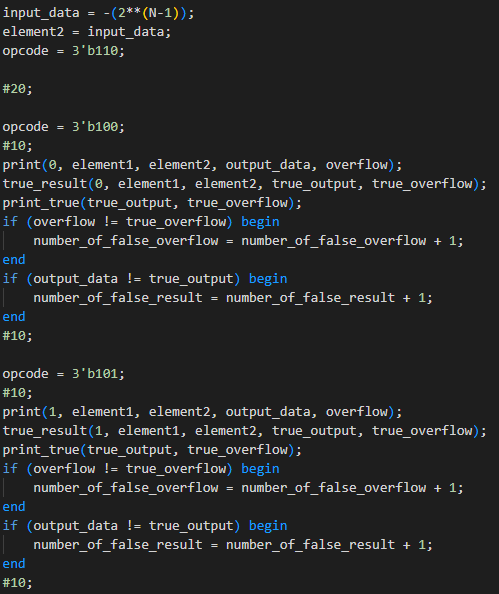
همان طور که مشاهده می­کنید ماژول ما برای N = 8 به درستی کار کرده است.



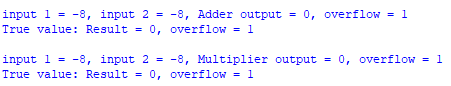
همان طور که مشاهده می­کنید ماژول ما برای N = 16 به درستی کار کرده است.

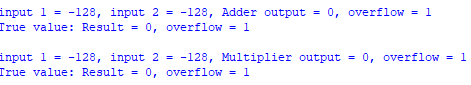


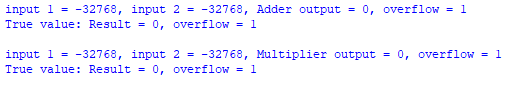
همان طور که مشاهده می­کنید ماژول ما برای N = 32 به درستی کار کرده است.

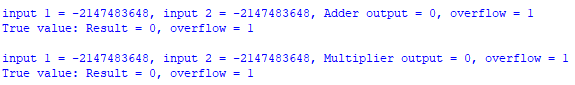
در ادامه کوچکترین عدد منفی ممکن را پوش کردم تا ضرب و جمع آن را با خودش بررسی کنم.(توجه شود که در تست قبلی بزرگترین عدد مثبت را بعد از تست پاپ کردیم پس تنها عنصر استک کوچکترین عدد منفی می­باشد) 

در تصاویر زیر خروجی­های آن را می­توانید مشاهده کنید.

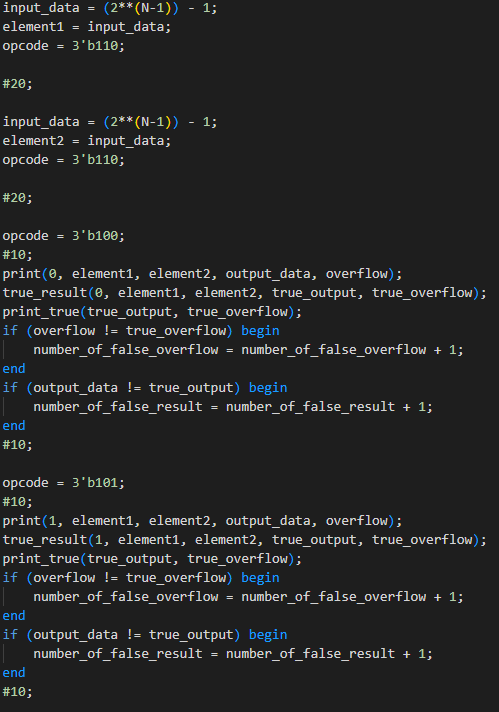
همان طور که مشاهده می­کنید ماژول ما برای N = 4 به درستی کار کرده است.

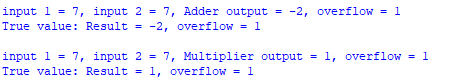
همان طور که مشاهده می­کنید ماژول ما برای N = 8 به درستی کار کرده است.

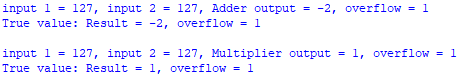
همان طور که مشاهده می­کنید ماژول ما برای N = 16 به درستی کار کرده است.

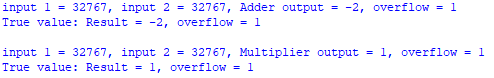
همان طور که مشاهده می­کنید ماژول ما برای N = 32 به درستی کار کرده است.

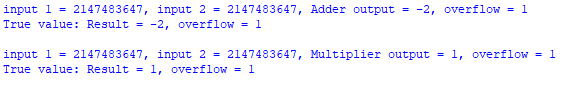
در تست زیر دوبار بزرگترین عدد مثبت را پوش می­کنیم تا ضرب و جمع را با آن بررسی کنیم که خروجی­های ماژول را در تصاویر زیر مشاهده می­کنید.



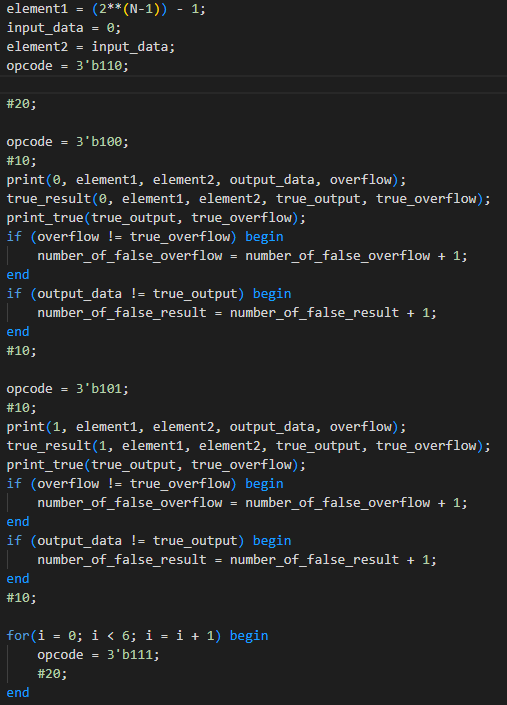
همان طور که مشاهده می­کنید برای N = 4 ماژول به درستی کار کرده است.

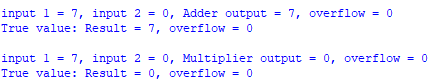
همان طور که مشاهده می­کنید برای N = 8 ماژول به درستی کار کرده است.

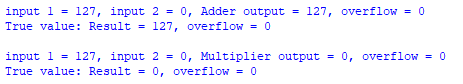
همان طور که مشاهده می­کنید برای N = 16 ماژول به درستی کار کرده است.

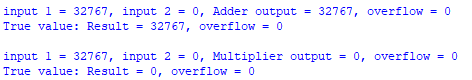
همان طور که مشاهده می­کنید برای N = 32 ماژول به درستی کار کرده است.

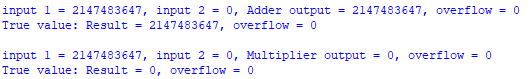
در تست بعدی که در تصویر زیر مشاهده می­کنید عدد صفر را در استک پوش می­کنیم و عملیات­های جمع و ضرب آن را با بزرگترین عدد مثبت بررسی می­کنیم. خروجی­های آن را در تصاویر زیر می­توانید مشاهده کنید. در آخر هم استک را خالی می­کنیم تا برای تست بعدی آماده شویم.



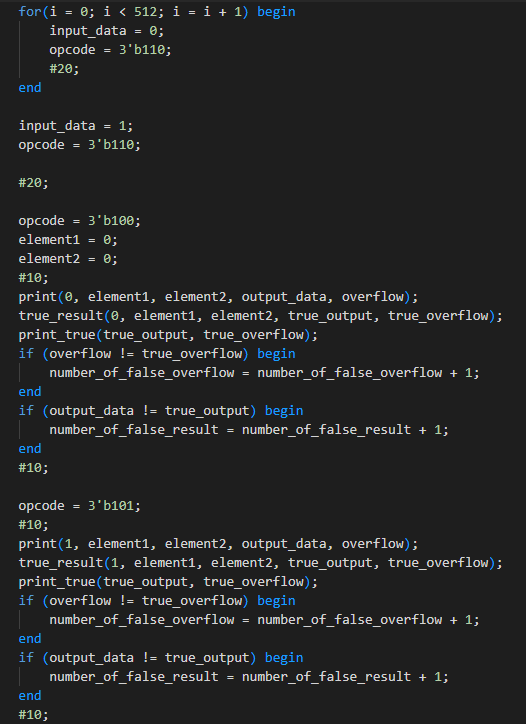
همان طور که مشاهده می­کنید برای N = 4 ماژول ما به درستی کار کرده است.

همان طور که مشاهده می­کنید برای N = 8 ماژول ما به درستی کار کرده است.

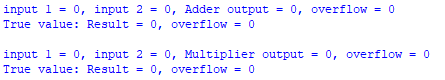
همان طور که مشاهده می­کنید برای N = 16 ماژول ما به درستی کار کرده است.

همان طور که مشاهده می­کنید برای N = 32 ماژول ما به درستی کار کرده است.

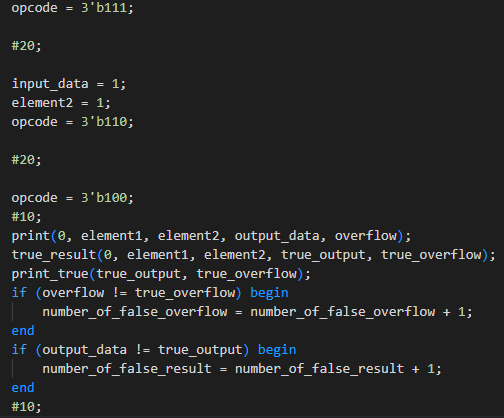
در تست بعدی هر 512 خانه استک را با صفر پر می­کنیم و سپس عدد یک را در استک می­خواهیم پوش کنیم اما چون استک پر است منطقا نباید پوش شود سپس عملیات جمع و ضرب را انجام می­دهیم تا مشاهده کنیم که یک وارد نشده باشد. (در اینجا element1 و element2 را برابر صفر قرار داده­ام تا تابع­هایی که تعریف کرده­ایم به درستی کار کنند و این دو عنصر تاثیری در ماژول نخواهند داشت) در تصاویر زیر خروجی­های آن را مشاهده می­کنید.



برای این تست چون N هیچ تاثیری ندارد صرفا یک عکس قرار می­دهیم.

همان طور که مشاهده می­کنید خروجی­های ALU ما برابر صفر شده است که به این معنا می­باشد که یک پوش نشده چون استک پر بوده است.

در تست زیر ابتدا یک عنصر را از استک پاپ می­کنیم و سپس عدد یک را پوش می­کنیم و سپس عملیات جمع را انجام می­دهیم.

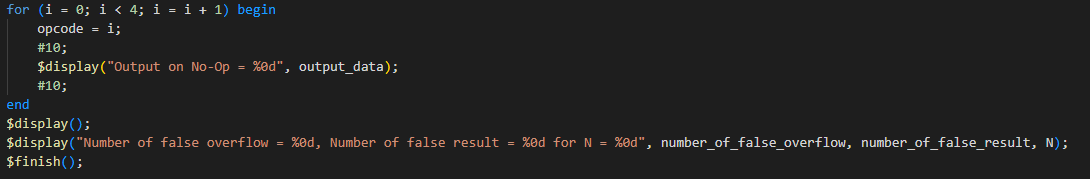


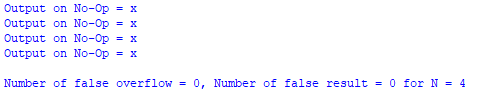


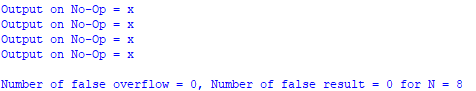
همان طور که مشاهده می­کنید در اینجا عدد یک وارد استک شده و عملیات بر روی آن انجام شده است چون ابتدا یک عنصر را پاپ کردیم استک ما یک جای خالی دارد پس 1 می­توانید در آن قرار گیرد.

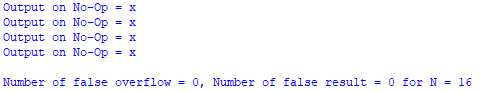
در تست آخر هم No-op ها را بررسی می­کنیم که خروجی ماژول ما باید X باشد.

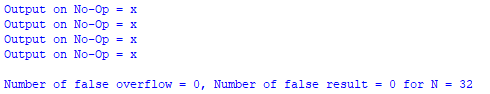
در آخر هم تعداد اشتباه­های ماژول را چاپ می­کنیم.



همان طور که مشاهده می­کنید برای N = 4 No-op صحیح عمل کرده است. همچنین تعداد غلط­های ما صفر بوده یعنی اینکه برای این تست­ها ماژول ما بی­نقص عمل کرده است.

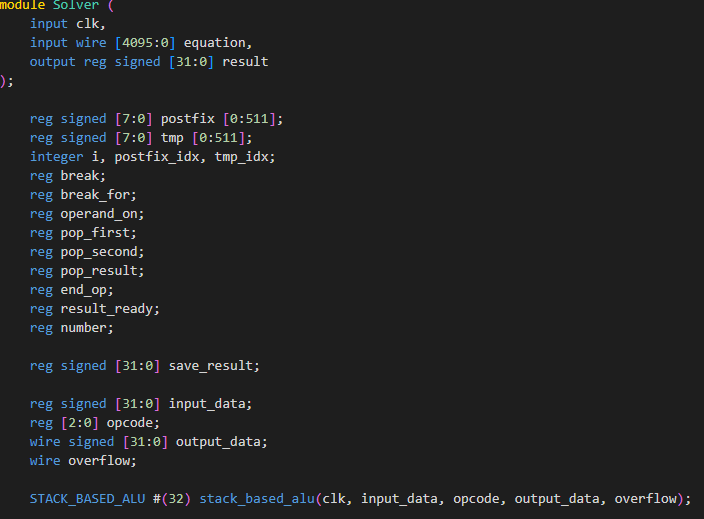
همان طور که مشاهده می­کنید برای N = 8 هم مدار بی­نقص بوده است.

همان طور که مشاهده می­کنید برای N = 16 هم مدار بی­نقص بوده است.

همان طور که مشاهده می­کنید برای N = 32 هم مدار بی­نقص بوده است.

حال که از صحت کارکرد ماژول STACK BASED ALU مطمئن شدیم باید ماژولی طراحی کنیم که با کمک گرفتن از آن بتواند عبارات ریاضی را حل کند. کد این بخش در پوشه code با نام EQU\_SOLVER.v می­باشد.

برای طراحی این ماژول ابتدا نیاز داشتیم که یک پیش پردازش بر روی عبارت ورودی انجام دهیم تا به صورت پسوندی در بیاید و سپس حاصل این عبارت پسوندی را با کمک STACK BASED ALU به دست بیاوریم. در ادامه به توضیح دقیق­تر درمورد آن می­پردازیم. در این سوال چون در صورت سوال اعداد خاصی بیان نشده بودند من فرض کردم که اعداد 32 بیتی داریم.

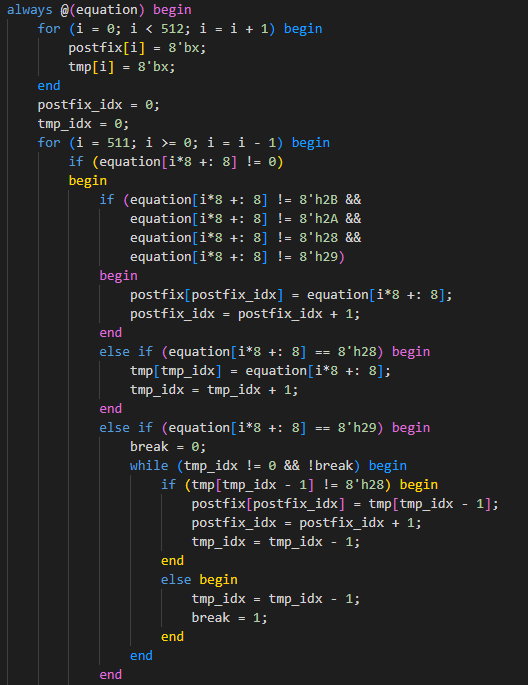


در ابتدا ورودی­های و خروجی­ها را قرار می­دهیم. منطقا ورودی کلاک می­خواهیم چون STACK BASED ALU ما عملیات­های خود را بر اساس کلاک انجام می­دهد. ورودی 4096 بیتی equation هم می­خواهیم که همان عبارت ریاضی ورودی ما می­باشد.(4096 بیت همان 8 \* 512 می­باشد به این معنا که عبارت ریاضی ما می­تواند شامل 512 کاراکتر باشد. 8 هم به دلیل ASCII می­باشد) و در نهایت هم یک خروجی 32 بیتی داریم که جواب عبارت ما می­باشد.

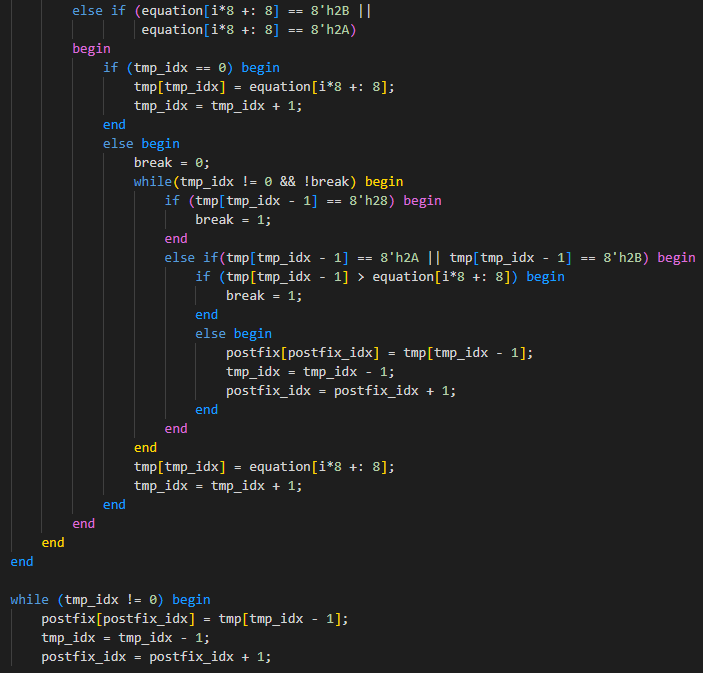
در ادامه آرایه 512 تایی از رجیستر­های 8 بیتی postfix را تعریف کرده­ایم که عبارت ریاضی ما به صورت پسوندی در آن ذخیره می­شود. tmp هم آرایه­ای است که برای تبدیل کردن عبارت به پسوندی به آن نیاز داشتیم.

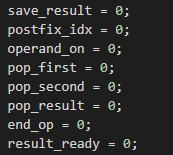
در ادامه متغیر­هایی تعریف شده­اند که کاربرد هرکدام در ادامه گفته خواهد شد.

همچنین یک instance از STACK BASED ALU خود گرفته­ایم تا بتوانیم عبارات پسوندی را با آن حساب کنیم.

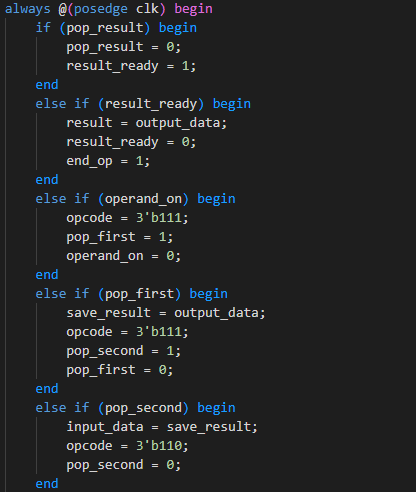


در ادامه always ای تعریف کرده­ام که به ازای هربار تغییر equation ورودی یک پیش پردازش را بر روی آن انجام می­دهد تا آن را به صورت پسوندی در postfix ذخیره کند. در ابتدا آرایه­های postfix و tmp را X می­کنم تا اگر مقادیری از قبل در آن­ها مانده بود از بین برود و در ادامه الگوریتم تبدیل کردن به postfix را بر روی equation ورودی اجرا می­کنم ادامه کد را در تصویر زیر می­توانید مشاهده کنید. این الگوریتم را با کمک گرفتن از لینکی که در README گیت­هاب قرار داده­ام پیاده سازی کردم و کد آن که به زبان جاوا بود را به یک­سری تغییرات به وریلاگ تبدیل کردم. یکی از تفاوت­هایی که وجود داشت این بود که ما صرفا باید عملگر­های ضرب و جمع را پیاده­سازی می­کردیم. تغییر دیگری که دادم این بود که اسپیس­های موجود در عبارت ورودی را در عبارت پسوندی حذف نکردم چون اعداد با رقم­های بیشتر از یک را نیز باید پشتیبانی می­کردیم. اگر اسپیس را حذف می­کردم نمی­توانستیم اعداد را از هم جدا کنیم در این پیاده­سازی اعداد مختلف با اسپیس از هم جدا می­شوند و رقم­های یک عدد بدون فاصله در کنار هم قرار می­گیرند و این گونه می­توانیم اعداد را از هم تشخیص دهیم.



در آخر این پیش پردازش مقادیری که برای حساب کردن مقدار عبارت پسوندی نیاز هستند را مقدار دهی اولیه می­کنیم.

در اینجا always را با لبه بالارونده کلاک تعریف کردم که حاصل عبارت پسوندی را حساب کند.



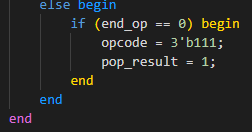
اولین شرط زمانی درست می­باشد که عبارت ما تمام شده باشد و آپکد پاپ فعال باشد به این معنا که ما می­خواهیم جواب نهایی را پاپ کنیم و درون این شرط فلگش را صفر می­کنیم تا دوباره داخل نشویم و فلگ result\_ready را یک می­کنیم تا در کلاک بعدی که جواب عبارت در output\_data آماده می­باشد، مقدارش را در result بریزیم و در آن­جا فلگش را صفر می­کنیم تا دوباره واردش نشویم و end\_op را یک می­کنیم به این معنا که عملیات ما تمام شده است. این دو شرط زمانی اتفاق می­افتند که رشته پسوندی ما تمام شده باشد در آخر فلگ pop\_result را بررسی می­کنیم. سه شرط بعد از این دو مربوط به انجام عملیات­های ضرب و جمع هستند و مربوط به این دو نیستند.

شرط بعدی(با فلگ operand\_on) زمانی اتفاق می­افتد که آپکد ضرب یا جمع اتفاق افتاده باشد پس عملیات در این کلاک انجام می­شود و ما باید در کلاک بعدی عدد بالای استک را پاپ کنیم چون کار ما با آن تمام شده است. در شرط بعدی عملیات جمع تمام شده است و جواب آن آماده است پس جواب آن را در یک رجیستر ذخیره می­کنیم تا آن را پوش کنیم. در این کلاک علملیات پاپ کردن عنصر اول دارد صورت می­گیرد و ما آپکد را دوباره روی پاپ تنظیم می­کنیم تا عنصر دوم نیز در کلاک بعدی پاپ شود. در کلاک بعدی وارد شرط آخر می­شویم که عملیات پاپ عنصر دوم دارد انجام می­شود پس ما آپکد را روی پوش و input\_data را برابر save\_result قرار می­دهیم تا در کلاک بعدی جواب عملیات در استک پوش شود و در آن­­جا ذخیره شود.



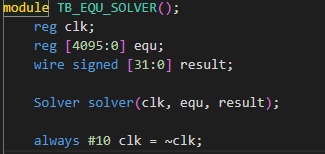
در ادامه در تصویر بالا به کد شکنی عبارت پسوندی می­پردازیم به این صورت که تا زمانی که عبارت تمام نشده روی آرایه جلو می­رویم و در ابتدای کار اگر اسپیس دیدیم آن­قدر جلو می­رویم تا اسپیس تمام شود. بعد از آن بررسی می­کنیم که اگر به یک عدد و یا منفی رسیدیم عدد آن را حساب کنیم. در ابتدا بررسی می­کنیم که اگر منفی داشتیم در آخر کار که عدد حساب شد آن را در منفی یک ضرب کنیم. برای به دست آوردن اعداد هم هر دفعه که به یک رقم جدید از آن عدد می­رسیم عدد قبلی را ضرب در 10 کرده و با عدد جدید جمع می­کنیم. این­گونه اعداد با تعداد رقم بیشتر از یک هم هندل می­شوند. حال بررسی می­کنیم که اگر به + یا \* رسیده بودیم opcode متناظر با آن را قرار می­دهیم و operand\_on را یک می­کنیم تا سلسله مراتبی که در بالا توضیح دادیم اجرا شود.

به طور مثال فرض کنید عبارت 12 + 5 را داشته باشیم. پسوندی آن طبق الگوریتم ما به صورت +12 5 در می­آید که برای 5 مستقیم عدد 5 را قرار می­دهد. در کلاک بعدی دو اسپیس را رد کرده و به 1 می­رسد و 1 را در input\_data قرار می­دهد و می­بیند که در ایندکس بعدی آن هم عدد وجود دارد نه اسپیس (چون اگر اسپیس بود طبق چیزی که بالاتر توضیح دادیم به این معنا است که به یک عدد جدید رسیده­ایم) پس input\_data را ضرب در 10 می­کند تا به عدد 10 برسد سپس 2 را با آن جمع می­کند و به عدد 12 می­رسد. این گونه اعداد با ارقام بیشتر از یک هم هندل می­شوند. همچنین بررسی شده است که اگر ابتدای عدد یک – بود در آخر input\_data را ضرب در منفی یک کند.

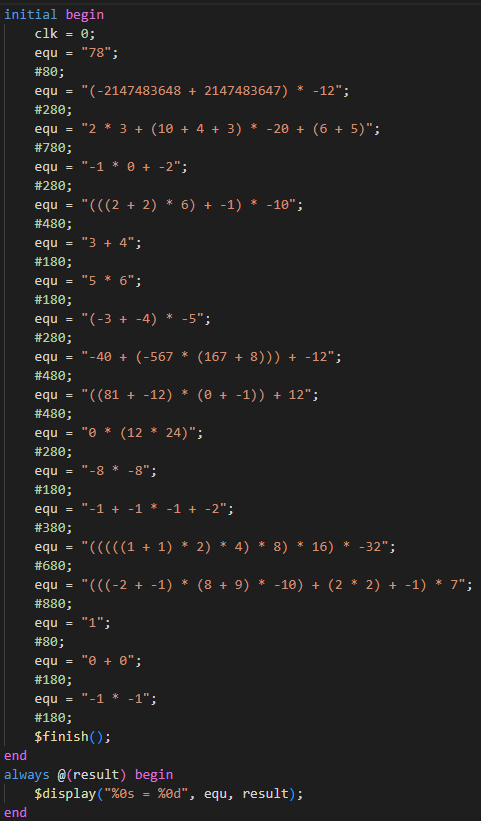
در آخر هم اگر کل عبارت پسوندی ما تمام شده بود وارد این شرط می­شویم ابتدا بررسی می­کند که آیا حساب کردن عدد تمام شده است یا خیر اگر تمام نشده بود pop\_result را یک می­کند تا سلسله مراتبی که در بالا توضیح دادیم انجام شود.

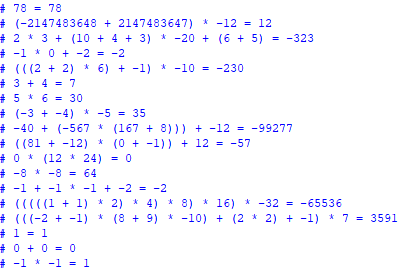
دلیل قرار دادن end\_op این بود که اگر چند کلاک اضافه­تر ماندیم خروجی ما X نشود چون استک خالی شده و اگر چیزی پاپ کنیم X می­گیریم.

در ادامه تست بنچی برای آن نوشتیم که در پوشه Code با نام TB\_EQU\_SOLVER.v موجود می­باشد.

در ایتدا که ماژول را تعریف کردم و یک instance از solver گرفتم و ورودی و خروجی­های مورد انتظارش را به آن دادم.

در ادامه همان طور که در تصویر زیر مشاهده می­کنید به ازای عبارات مختلفی عملکرد مدار را بررسی می­کنیم و هربار که جواب فرق می­کند مقدار جواب را چاپ می­کنیم. در پایان می­توانید تصاویر خروجی آن را مشاهده کنید.



همان طور که مشاهده می­کنید به ازای ورودی­های مختلف ماژول ما به درستی عبارات را حساب کرده است پس می­تاونیم از صحت عملکر آن اطمینان خاطر یابیم.