دانشگاه صنعتی شریف
دانشکده مهندسی کامپیوتر

عنوان:

گزارش پروژه درسی

اعضای گروه

فرزام کوهی رونقی

آریا همتی

مهدی اکبری

ثنا بابایان ونستان

نام درس

معماری کامپیوتر

نیم سال دوم ۱۴۰۲ – ۱۴۰۳

نام استاد درس

حسین اسدی

۱ مقدمه

افزودن قابلیت پیش بینی taken و یا taken not بودن دستورات پرشی شرطی به پردازنده پایپ لاین میپس.

۱-۱ تعریف مسئله

در این پروژه قصد داریم به پردازنده ی MIPS که در تمرین های عملی درس طراح شده بود، واحد prediction branch اضافه کنیم. همان طور که در درس نیز یاد گرفته بودید branch prediction ها با توجه به کدی که در قبل اجرا شده است در خط لوله بیاورند که با فرض taken بودن آن اجرا می شوند یا اینکه دستور هایی را پیش بینی کنند که دستورهایی را در خط لوله بیاورند که با فرض not taken بودن اجرا می شوند.

۱-۲ اهمیت موضوع

انجام این کار باعث یک پیش بینی نسبتا درست درمورد پرش های شرطی می شود که با افزایش سرعت به عملکرد پردازنده ما کمک خواهد کرد.

۲ گام های پروژه

اهداف پروژه را به صورت یک سری گام نشان می دهیم:

گام ۱ : اطمینان از صحت عملکرد پردازنده : براي اینکه در ادامه گام ها به مشکل نخوریم، در ابتدا پردازنده Mips pipeline خود را مجددا می آزماییم و در صورت داشتن مشکل، آن را اصلاح میکنیم.

گام ۲ : افزودن قابلیت فلاش کردن پایپ لاین: براي اینکار، مکانیزمی طراحی میکنیم که در صورتی که یک سیگنال دلخواه، ۱ شد، pipe از تعدادي دستور پاك شود. سپس آن را تست میکنیم.

گام ۳ : ساختن واحد predictor branch و آپدیت کرئن مقادیر بیت مورد نظر: با توجه به توضیحات گفته شده (۴ مدخل و ...) یک واحد predictor branch طراحی کرده و کارکرد آن را تست میکنیم.

گام ۴ : سیاست :FIFO براي جايگ ذاري branch هاي جديد، از سيا ست FIFO استفاده می کنيم.

گام ۵ : سیاست :LRU یک نمونه دیگر از predictor branch خود اینبار با سیاست LRU براي جايگذاري ميسازيم.

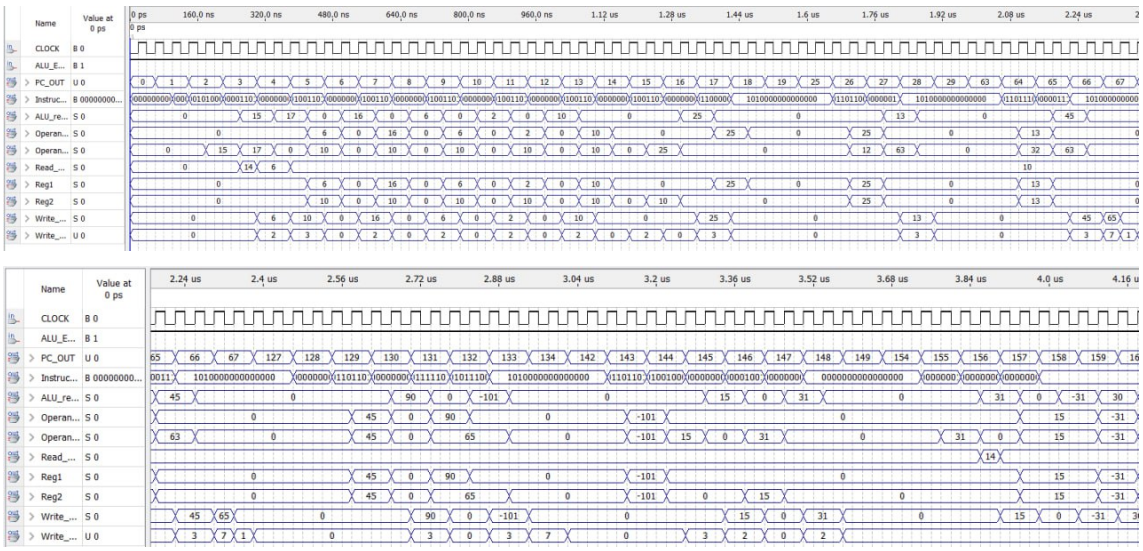
گام ۶ : نوشتن برنامه اي شامل پرش شرطی و حلقه ها براي بررسی صحت عملکرد مدار: بخش ارزیابی و گزارش کار

گام۷ : برنامه اي بنويسيم که همه شرط ها را اشتباه در نظر بگيرد: براي بررسی نقاط ضعف branch predictor

گام ۸ : افزودن سياست LRU-Pseudo براي جايگذاري و همچنين طی کردن مراحل ارزيابی قبلی (گام ۶ و ۷)

۲-۱ گام ۱

با توجه به فایل insts.mif که در پیوست قرار داده شده است یک دور از صحت عملکرد تمام دستورات مدار اطمینان حاصل شود.خروجی های درست در فایل ذکر شده قرار دارد.شکل زیر نمایی از ویو فرم این دستورات می باشد که نشان دهنده درستی خروجی ها است.



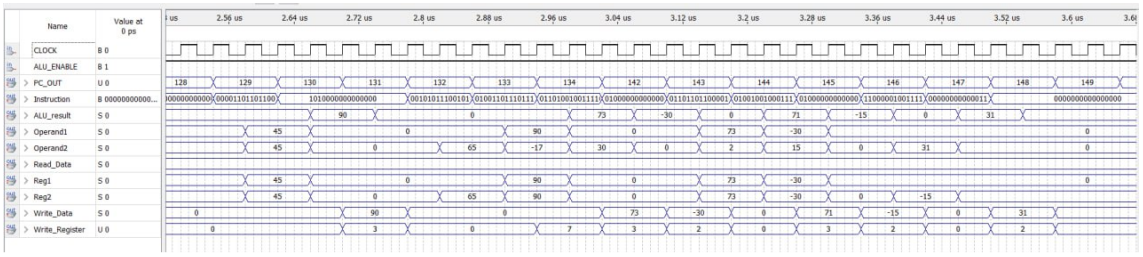
شکل ۱: کد تست در فولدر Gam۱ قرار دارد.

۲-۲ گام ۲

قبل از افزودن قابلیت فلاش کردن ابتدا به بررسی مشکل هایی که در صورت سه بار استفاده نکردن از operation no به صورت نرم افزاری بعد از دستورات پرش شرطی به وجود می آیند می پردازیم.

۲-۲-۱ باگ های دیتا

دستورات مربوط به بررسی این مشکل فایل instsDataBug.mif در پیوست قرار داده شده است.توجه تان را به خطوط ۱۳۲و ۱۳۳و ۱۳۴ جلب می کنیم.این سه دستور بعد از یک دستور BNE قرار گرفته اند و این برنچ نیز taken است ولی چون از operation no بعد BNE استفاده نشده داده های نادرستی در مموری و رجیستر فایل مورد ویرایش قرار گرفته شده اند که این وضعیت برای ما مطلوب نمی باشد.نمای زیر نشان دهنده این مشکلاتات است.



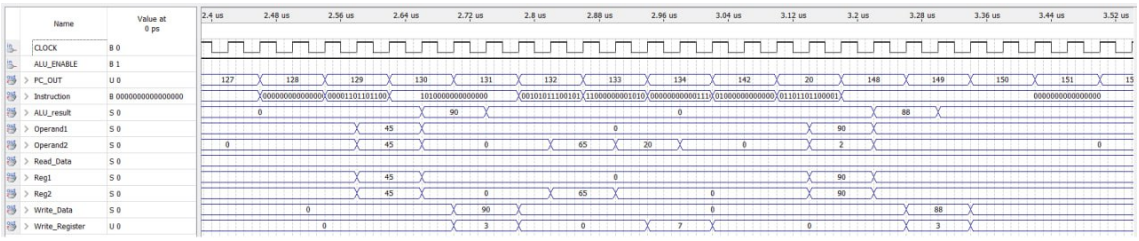
شکل ۲: باگ مربوط به دیتا

۲-۲-۲ باگ های برنچ

دستورات مربوط به این مشکلات در فایل instsBranchBug.mif در پیوست قرار داده شده است.در خطوط ۱۳۲و ۱۳۳و ۱۳۴ یک دستور jump و یک دستور BEG و یک operation no داریم که در صورت اجرا شدن اینها بعد از دستور BNE خط ۱۳۱ باعث بروز مشکلاتی خواهد شد و ما به صورت ناخواسته به آدرس هاس اشتباهی پرش خواهیم کرد.شکل زیر نمایی از ویو فرمی این دستورات است.

۲-۲-۳ افزودن سیگنال Flush و Flash Unit

برای از بین بردن تاثیر سه دستور بعد پرش شرطی ما نیاز به این داریم که در مراحل Write- MEM، Back با استفاده از دو سیگنال flush، flushjump از وقوع نتایج نامطلوب جلوگیری کنیم.این دو



شکل ۳: باگ مربوط به برتچ

سیگنال باید ۳ ملالک طول بکشد.برای مشاهده مکانیزم عملکرد این دو سیگنال می توان به مدار مراجعه کرد.تصاویر زیر گویای این امر هستند که بعد از به کارگیری این قابلیت دو مشکل مطرح شده در قسمت قبل را به کلی حل می کند.

۲-۳ گام ۳

۲-۴ طراحی اولیه branch predictor

predictor branch ساخته شده حاوی ۴ block است به طوری که هر block ۱۶ بیت دارد. ۸ بیت سمت راست مربوط به آدرس هست، بیت نهم بیت valid/invalid می‌باشد و به طور پیشفرض در ابتدا مقدار آن صفر است و بعد از آن با آمدن هر دستور بیت valid ۰ می‌شود. بیت بعدی مربوط به prediction هست که در صورتی که صفر باشد taken not و در صورتی که یک باشد taken است. ۶ بیت بعدی که ۶ بیت آخر هست age می‌باشد که هیچ استفاده‌ای از آن نمی‌کنیم (بعد از پیاده سازی‌های اولیه استراتژی پیاده‌سازی تغییر کرد و به علت اینکه هر block ۱۶ بیتی باشد یعنی توانی از ۲ بماند ۶ بیت را حذف نکریم ولی در واقعع هیچ استفاده‌ای از این ۶ بیت نمی‌شود.)

۲-۵ واحد FIFO branch predictor

پیاده سازی predictor branch fifo به این صورت است که هر وقت به دستور branch میرسیم آدرس آن را به عنوان ورودی به predictor branch ساخته شده می‌فرستیم.ورودی داده شده با ۴ مقداری که در pridector branch هست چک می‌شود(پیاده سازی این بخش مشابه با پیاده سازی cache می‌باشد که به صورت assosiative fully عمل می‌کند) اگر ادرس مورد نظر بین آن ۴ مقدار بود سیگنال hit ۱ می‌شود و در غیر این صورت صفر باقی می‌ماند. اگر سیگنال ۱ شود به این معنی است که ادرس branch در بین آن ۴ مقدار قرار دارد و در این صورت مقدار بیت predictor که

دهمین بیت از سمت راست هست را خروجی می‌دهد به این معنی که taken branch، است. برای پیاده سازی FIFO اگر سیگنال branch is یک باشد ولی سیگنال hit صفر بود (به معنی وجود نداشتن در predictor) همه آن ۴ مقدار را به پایین یکی shift می‌دهیم و آدرس branch که hit نشد را در سطر اول predictor می‌نویسیم و بیت prediction آنرا هم یک می‌کنیم که taken باشد.

۲-۶ بررسی حالت های مختلف

۲-۶-۱ taken predict شود و taken باشد

در این حالت همیشه هنگام operation no یک penalty miss خواهیم داشت.

۲-۶-۲ taken predict شود ولی taken not باشد

در این حالت خط بعدی اجرا می‌شود بعد از آن به آدرس مورد نظر پرش می‌کند و در آنجا هم ۲ دستور اجرا می‌کند(مجموعا ۳ دستور اجرا می‌شود) و بعد از آن متوجه می‌شود که این ۳ دستور به اشتباه اجرا شده است و آنها را flush می‌کند و به آدرس بعد از آدرس branch برمیگردد و از آنجا اجرای دستورات را ادامه می‌دهد.

۲-۶-۳ taken not predict شود ولی taken باشد

در این حالت ۳ دستور بعدی اجرا می‌شود و بعد از آن نتیجه شرط مشخص می‌گردد. با توجه به نتیجه بدست آمده باید ۳ دستوری که اجرا شدند را متوقف کنیم و عملیات هایی که در آن انجام شده است را reverse کنیم که flush پیاده سازی شده در پروژه این کار را انجام می‌دهد و در نهایت به آدرس درست می‌رود.

۲-۶-۴ taken not predict شود و taken not باشد

۲-۷ واحد LRU branch predictor

پیاده سازی این قسمت تقریبا با پیاده سازی predictor branch FIFO یکی است اما یکسری تفاوت‌هایی دارد که اصلی ترین قسمت آن در الگوریتمی است که بعد از hit شدن در predictor

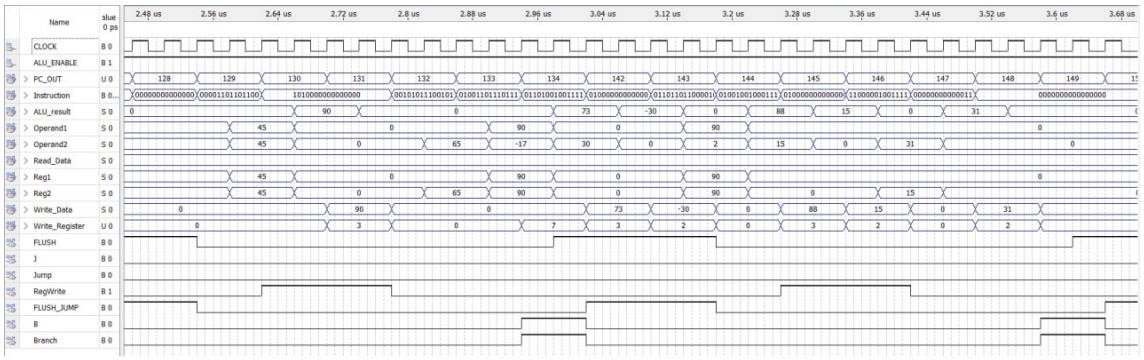
اجرا می‌شود. الگوریتم به این صورت پیاده سازی شده که هر زمانی که آدرسی hit شد به معنی اینکه در یکی از ۴ block پیدا شده، بلاکی که hit شده هرجایی که باشد به سطر اول predictor انتقال داده می‌شود و اگر آن بلاک در سطر k ام باشد، -k۱ سطر بالای آن همگی یکی به سمت پایین شیفت می‌خورند. بوسیله این پیاده سازی هنگامی که یک miss branch، شود پایینترین سطر از بین می‌رود و همه سطر ها یکی به سمت پایین شیفت می‌خورند و آدرس جدید در سطر اول نوشته می‌شود. با پیاده سازی که در قسمت hit انجام داده ایم همیشه سطری که حذف می‌شود و در سطر آخر قرار دارد کمترین استفاده را داشته است و به این ترتیب الگوریتم LRU را پیاده سازی کردیم.update کردن predictor هم بر اساس FSM موجود در صورت پروژه برای الگوریتم‌های LRU و FIFO مشترک پیاده‌سازی شده به شکلی که اگر اشتباها taken یا taken not در نظر گرفته بشه یک آدرس، آدرس رو به شکلی update می‌کنیم که اگر دوباره کال شد خلاف نتیجه قبل اعلام بشود یعنی اگر taken بود این سری taken not اعلام میشود و برعکس آن هم به همین شکل. این تغییرات بوسیله بیت prediction اعمال می‌شود.

۲-۸ گام ششم

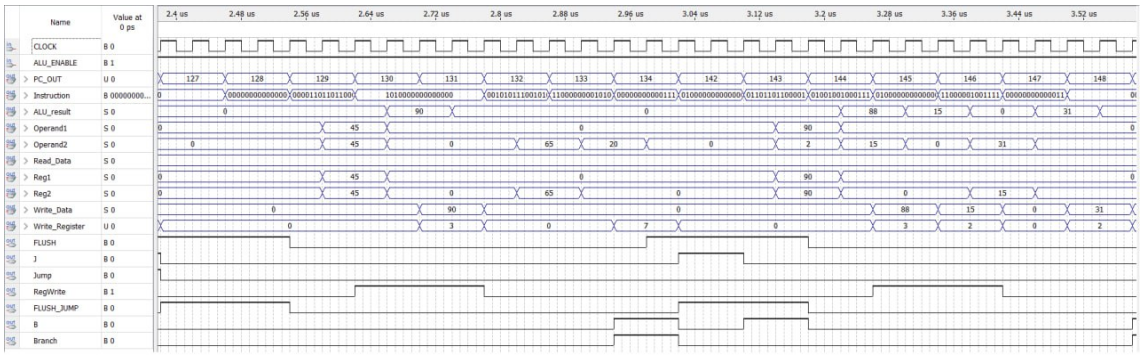
در این گام با استفاده از یک کد تست که در زمیمه قرار دارد، به مقایسه FIFO و LRU پرداخته‌ایم. مشاهده می‌شود زمان اجرا برای pipeline که بدون predictor هست، بیشتر از زمانی است که predictor pipline، دارد. در این مثال که مشاهده می‌کنید زمان LRU بهتر از زمان FIFO می‌شود.

۲-۹ گام هفتم

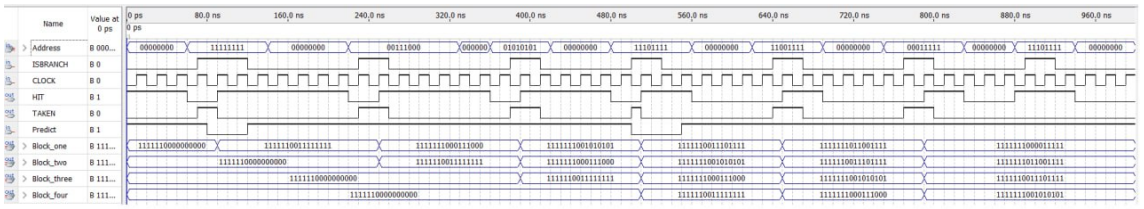
با ارائه یک کد که در آن تعدادی branch دائما اشتباه پیشبینی می‌شوند متوجه می‌شویم که اجرای برنامه با استفاده از pipeline بدون predictor بهینه‌تر بوده و در این مثال مجداد LRU نتیجه بهتری از FIFO ارائه می‌دهد.



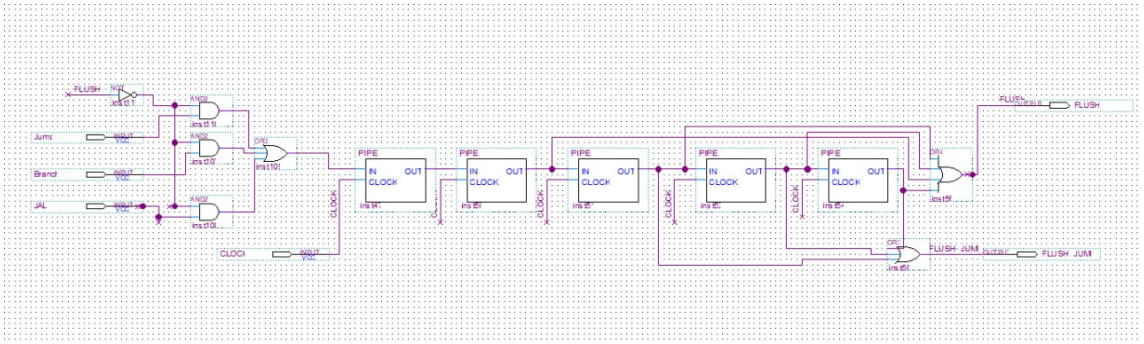
شکل ۴: رفع کردن مشکل باگ برای دیتا



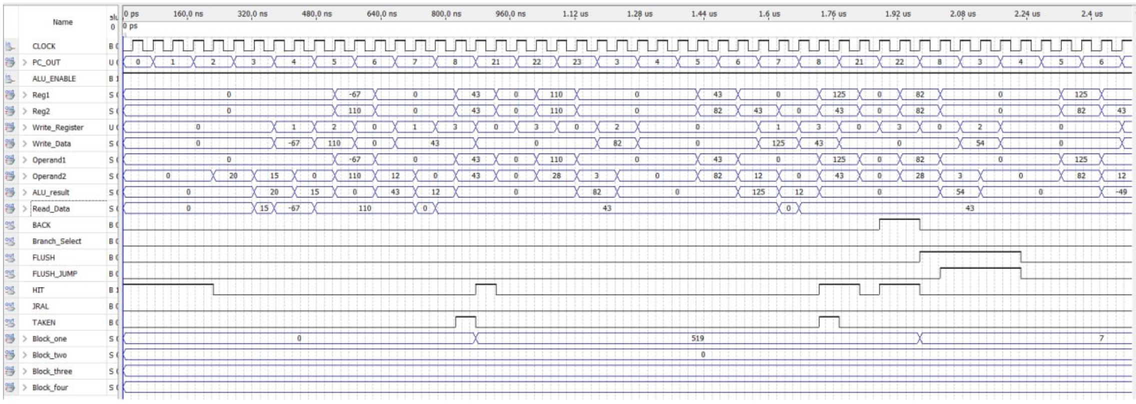
شکل ۵: رفع کردن مشکل باگ برای برنچ

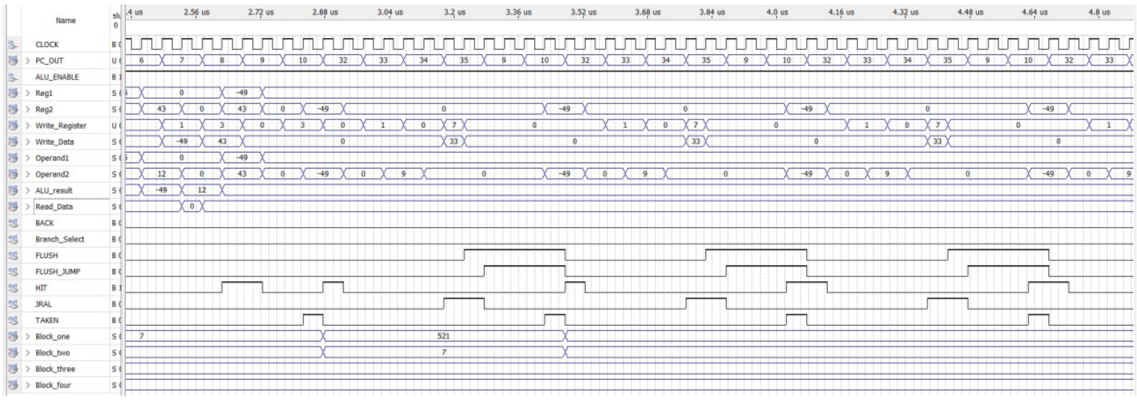


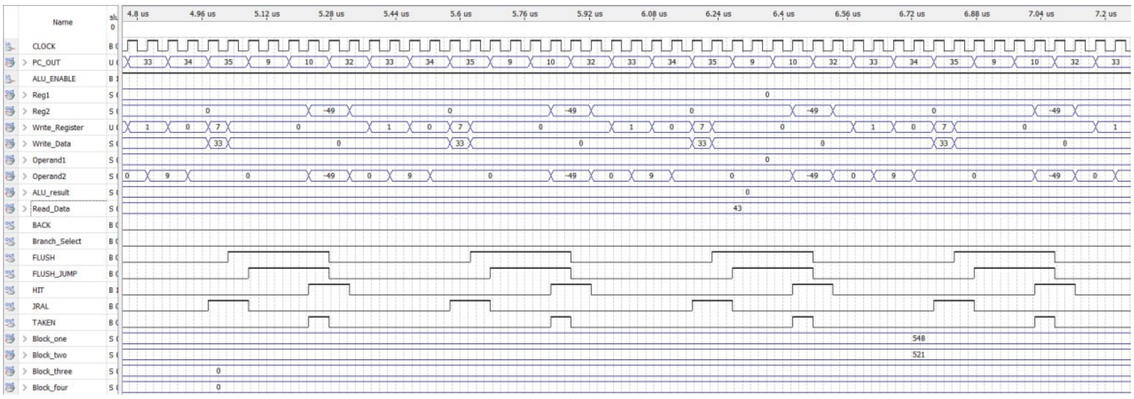
شکل ۶: اجرای برنامه با predictor FIFO

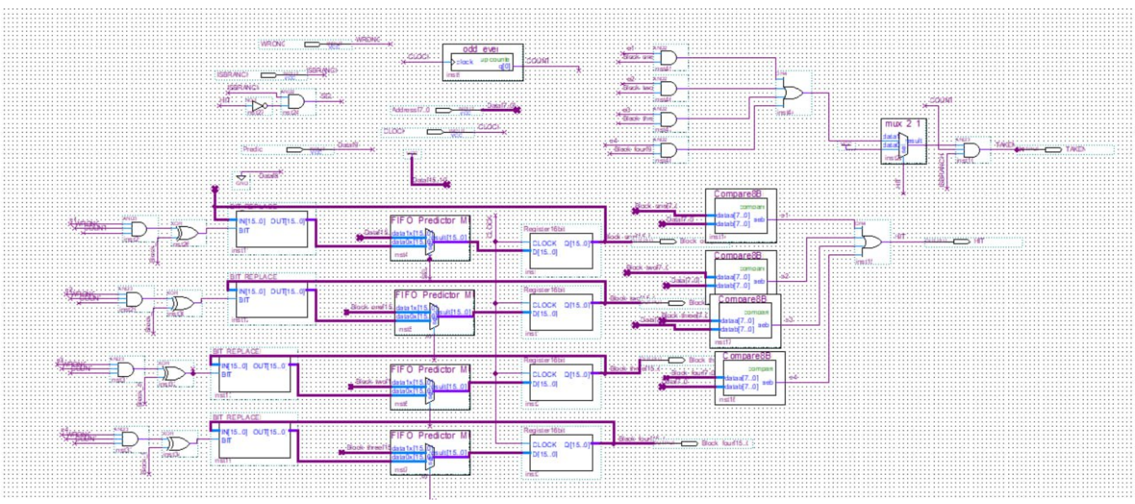


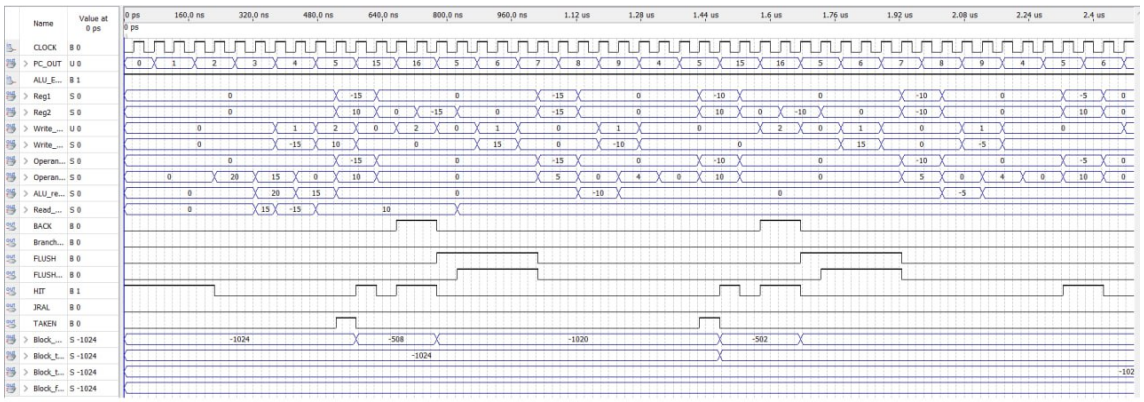
شکل ۷: Unit Flush

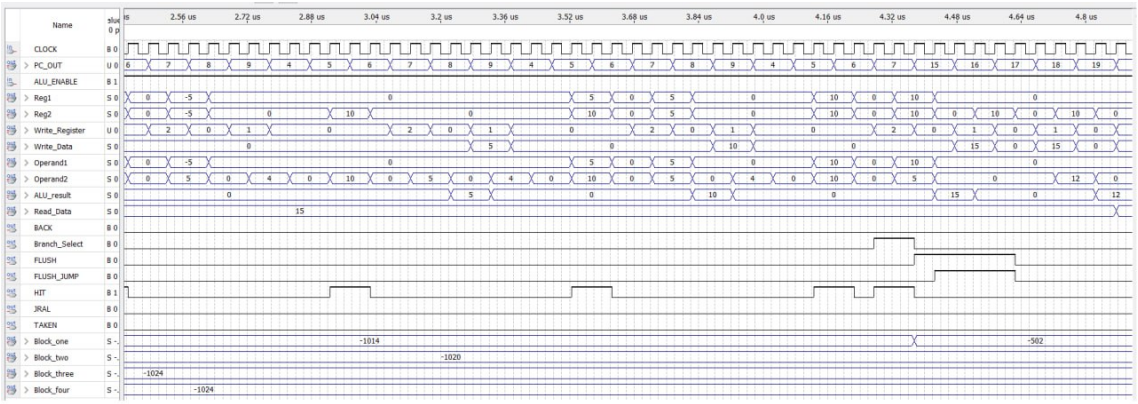

Name
CLOCK
PC_OUT
ALU_ENABLE
Reg1
Reg2
Write_Register
Write_Data
Operand1
Operand2
ALU_result
Read_Data
BACK
Branch_Select
FLUSH
FLUSH_JUMP
HIT
JRAL
TAKEN
Block_one
Block_two
Block_three
Block_four

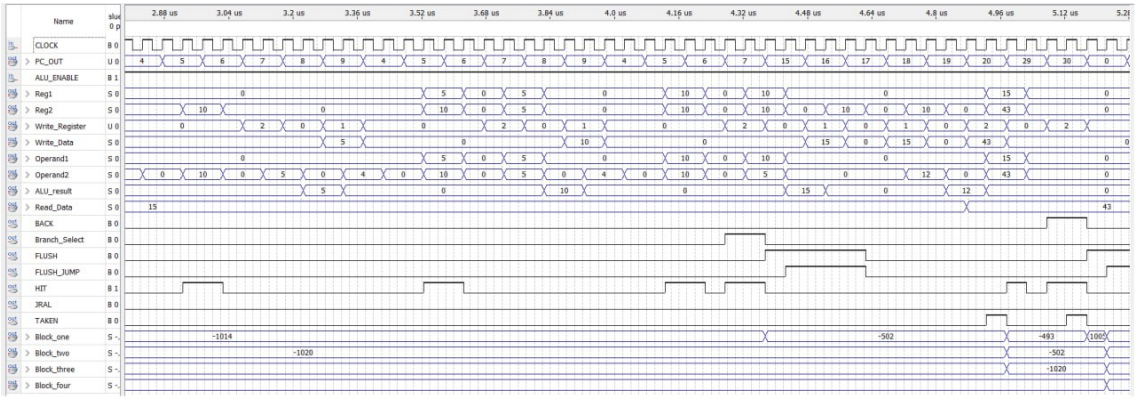

Name
CLOCK
PC_OUT
ALU_ENABLE
Reg1
Reg2
Write_Register
Write_Data
Operand1
Operand2
ALU_result
Read_Data
BACK
Branch_Select
FLUSH
FLUSH_JUMP
HIT
JRAL
TAKEN
Block_one
Block_two
Block_three
Block_four

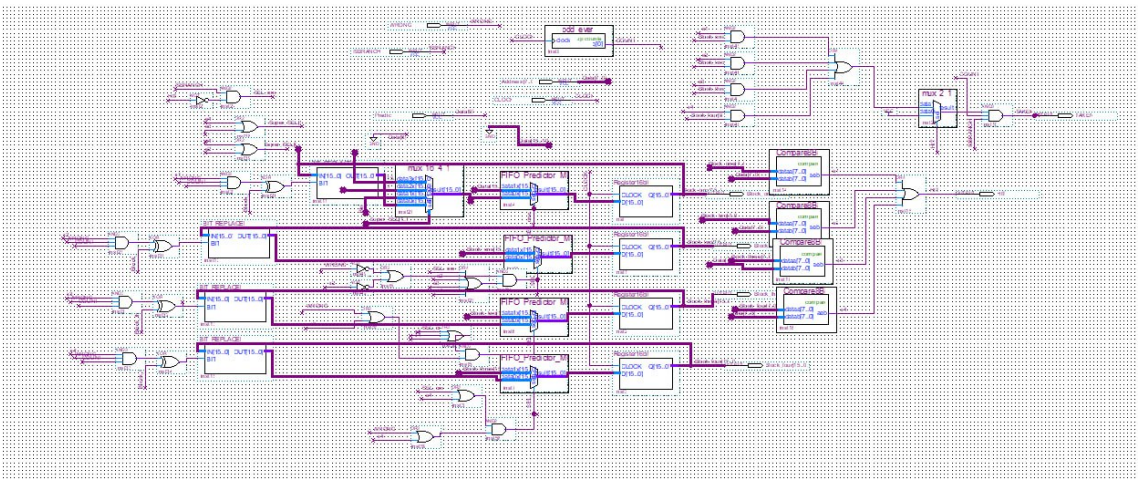

Name
CLOCK
PC_OUT
ALU_ENABLE
Reg1
Reg2
Write_Register
Write_Data
Operand1
Operand2
ALU_result
Read_Data
BACK
Branch_Select
FLUSH
FLUSH_JUMP
HIT
JRAL
TAKEN
Block_one
Block_two
Block_three
Block_four

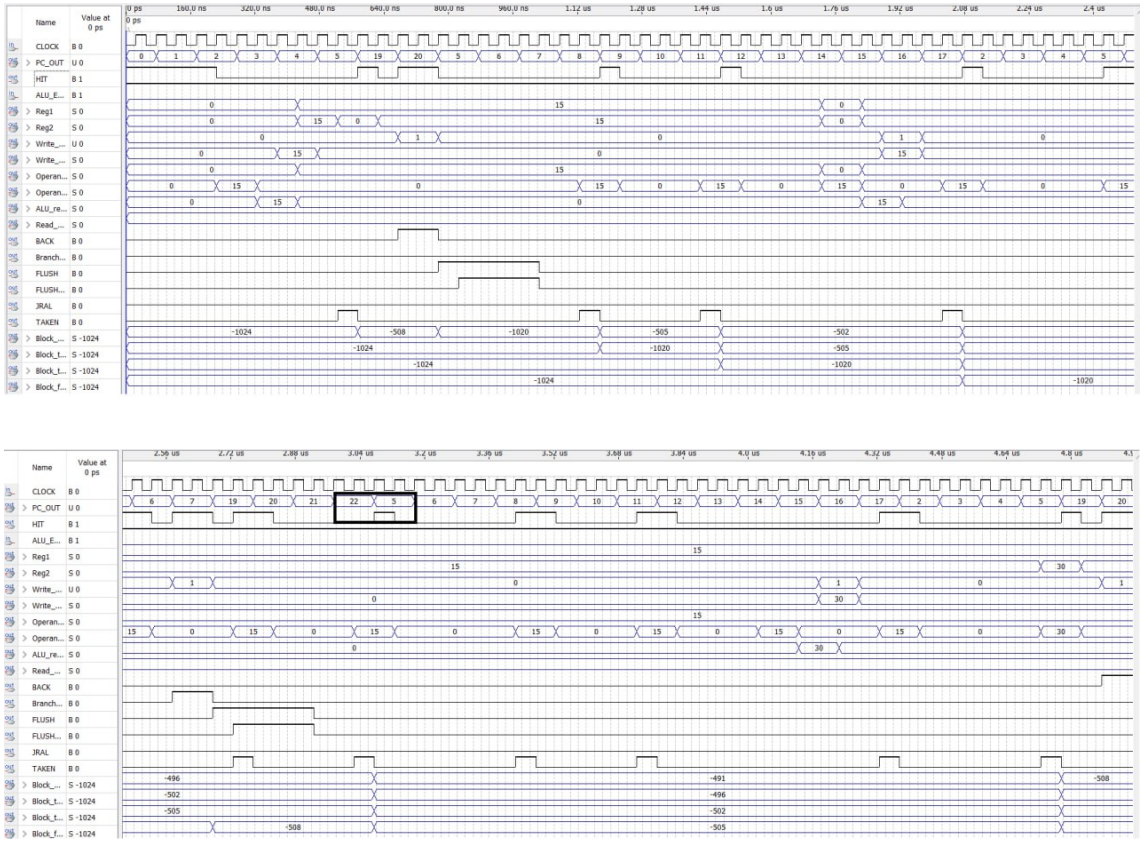

odd_even
mux 2 1
FIFO Predictor M
BIT REPLACE
Compare8B
HIT
TAKEN


Name
Value at 0 ps
CLOCK
PC_OUT
ALU_E...
Reg1
Reg2
Write_...
Write_...
Operan...
Operan...
ALU_re...
Read_...
BACK
Branch...
FLUSH
FLUSH...
HIT
JRAL
TAKEN
Block_...
Block_t...
Block_t...
Block_f...

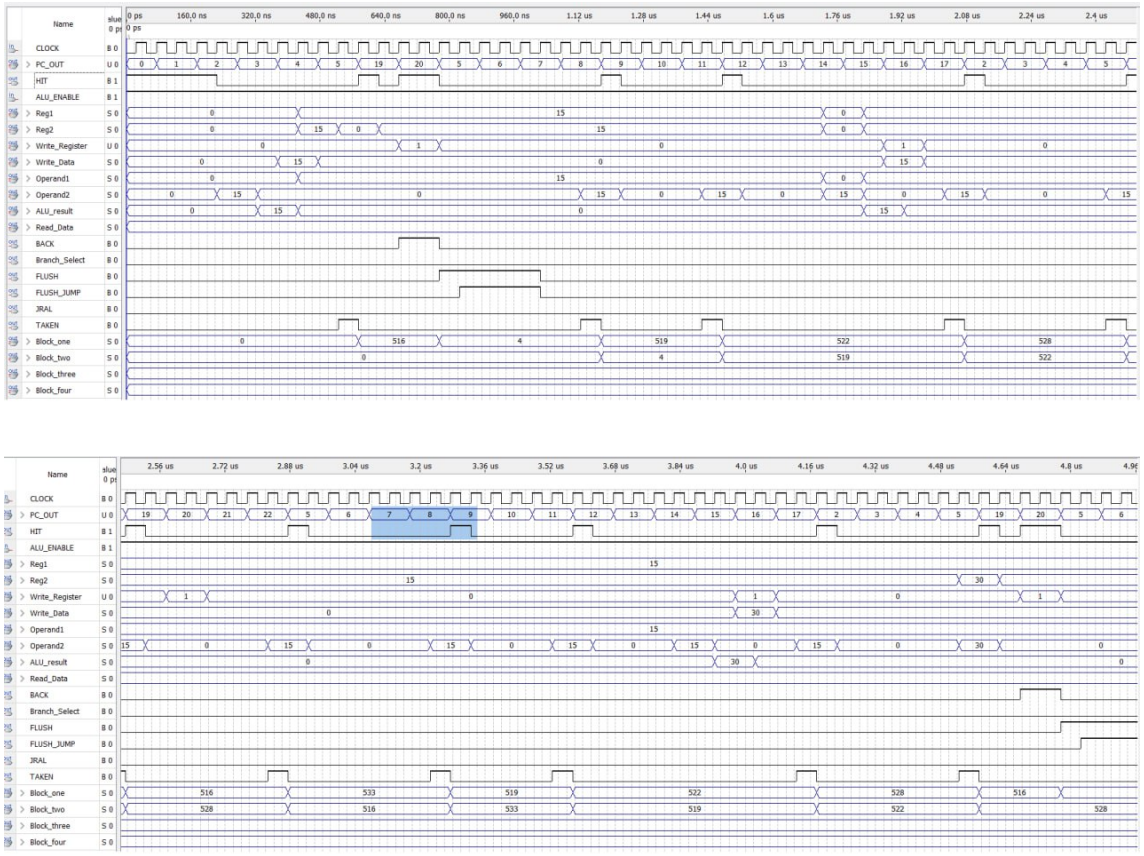

Name
CLOCK
PC_OUT
ALU_ENABLE
Reg1
Reg2
Write_Register
Write_Data
Operand1
Operand2
ALU_result
Read_Data
BACK
Branch_Select
FLUSH
FLUSH_JUMP
HIT
JRAL
TAKEN
Block_one
Block_two
Block_three
Block_four

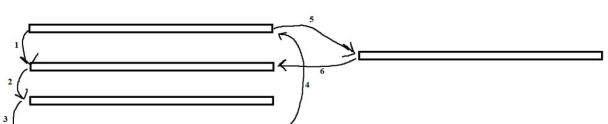

Name
CLOCK
PC_OUT
ALU_ENABLE
Reg1
Reg2
Write_Register
Write_Data
Operand1
Operand2
ALU_result
Read_Data
BACK
Branch_Select
FLUSH
FLUSH_JUMP
HIT
JRAL
TAKEN
Block_one
Block_two
Block_three
Block_four

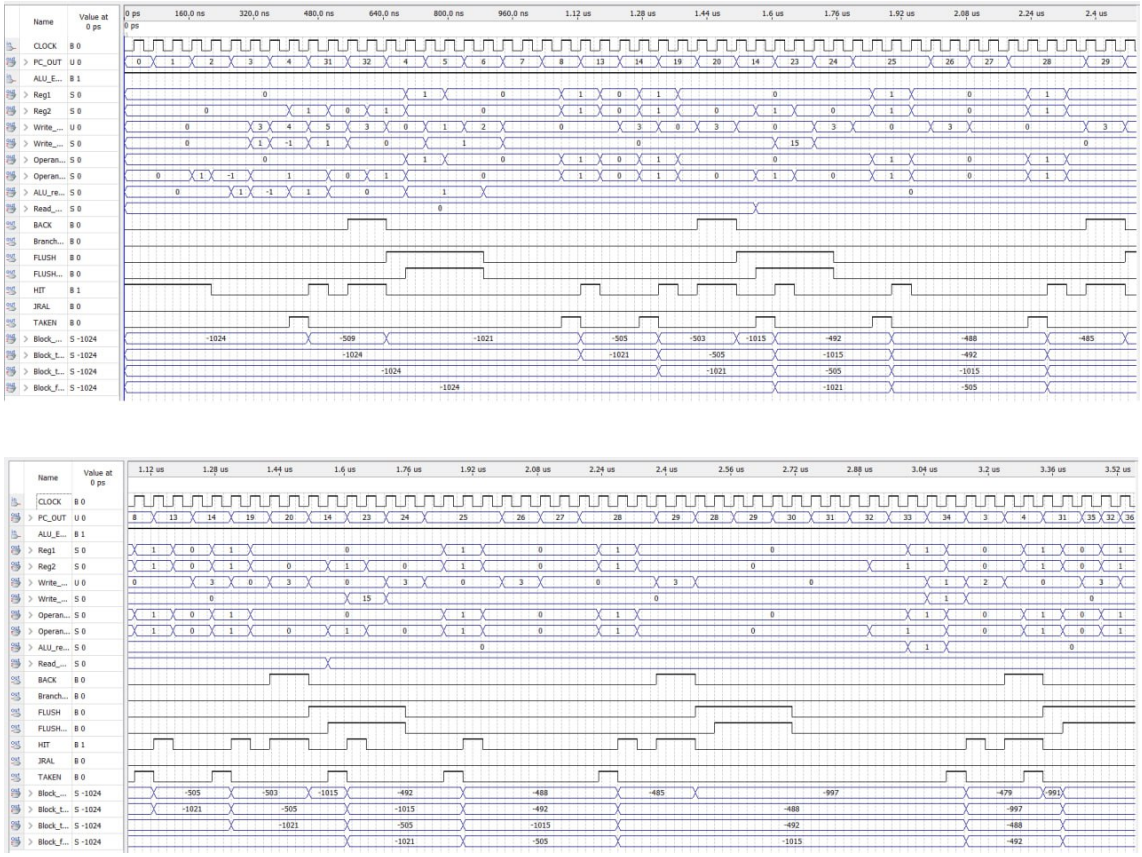




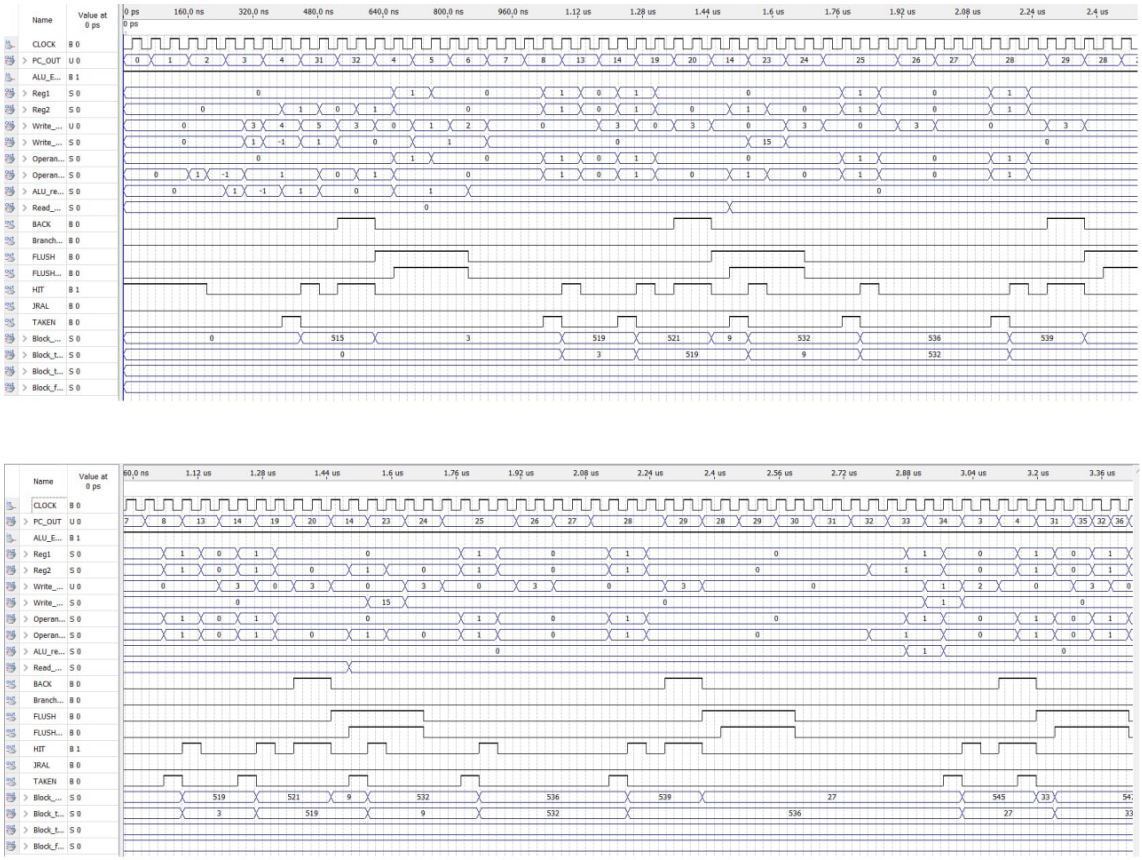
شکل ۸: دو عکس اول مربوط به FIFO و دو عکس دوم مربوط به LRU می‌باشد







شکل ۱۰: دو عکس اول مربوط به FIFO و دو عکس دوم مربوط به LRU می‌باشد



References

مطالب تکمیلی

پیوست‌های خود را در صورت وجود می‌توانید در این قسمت قرار دهید.