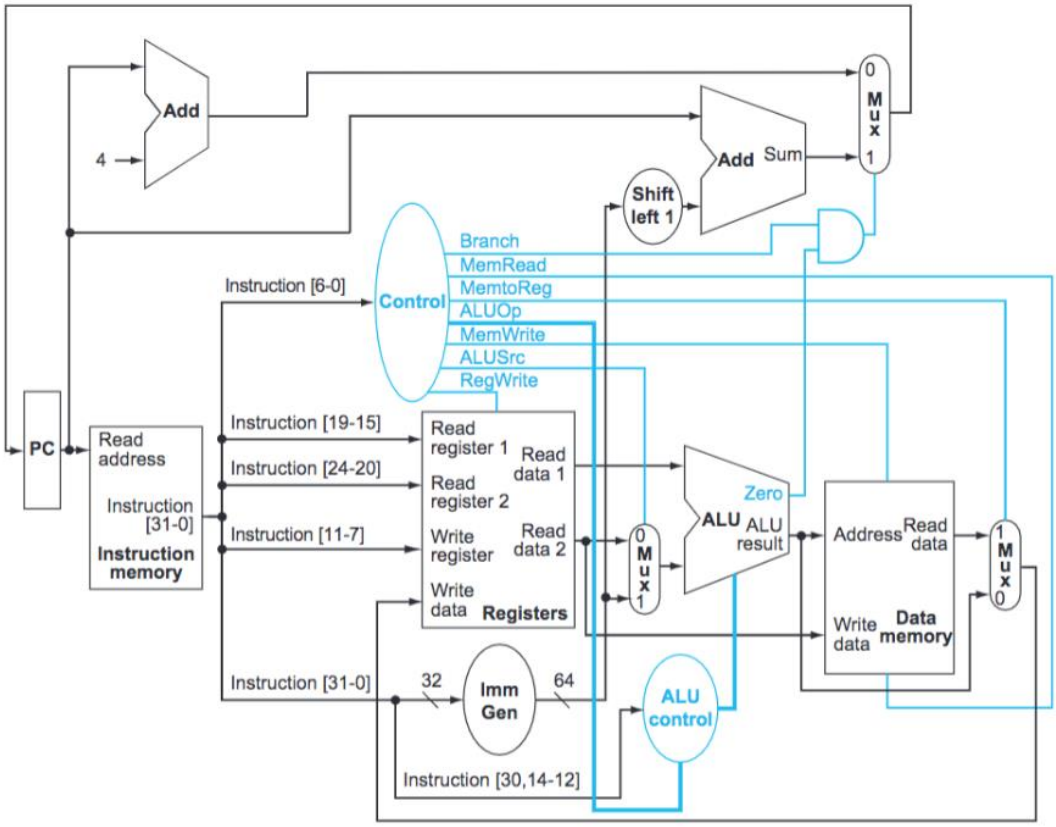
گزارش در مورد نحوه کار پروژه

در این پروژه در پارت اول یک پردازنده risc V به صورت single cycle پیاده سازی شده و در پارت دوم بصورت pipeline .

پارت اول بر اساس datapath زیر پیاده سازی شده :



برای پیاده سازی ابتدا تک تک ماژول ها را جداگانه پیاده سازی و سپس به هم متصل کرده ام.

در پیاده سازی این پروژه در فایل SS_CPU که اتصال این ماژول ها و پیاده سازی datapath است ابتدا pc_enable را یک کرده و یک ادرس به آن میدهیم PC طوری طراحی شده که آدرس 32 بیتی instruction که در حال خوانده شدن است را در خود دارد و آن را به instruction memory میدهد تا طبق آدرس آن دستور آن را بررسی کرده و دستور را به کمک ماژول های بعدی انجام دهد به همین دلیل است که یک adder که با عدد 4 جمع میکند نیاز بوده است و برای راحتی کار آن را جداگانه طراحی و نام آن را addFour گذاشته ام و آدرسی که در pc ذخیره شده است را با 4 جمع کرده و به آدرس دستور بعدی میرسد و پس از اجرای دستور فعلی آدرس دستور بعدی را حساب کرده و به pc میدهد.

برای تست این پروژه دستور های زیر را باید اجرا میکردیم :

AA03 <-- lw x20, 0(x10)

1400AB3 <-- add x21, 0, x20

1052A333 <-- sub x6, x21, x20

1452423<-- sw x20, 8(x10)

1E951C63 <-- beq x20,x21,-4

ابتدا این دستورها را طبق استاندارد risc v به باینری و سپس به هگزادسیمال تبدیل کرده و آن را در instruction memory به ترتیب ذخیره کرده ام و وقتی برنامه اجرا شود آدرس PC به اولین خانه instruction memory اشاره میکند و هر بار 4 تا اضافه شده و دستور بعدی را اجرا میکند و به اینصورت تمام دستورهای ذخیره شده اجرا میشوند.

پس از خوانده شدن دستوری که در آدرس خواسته شده قرار دارد دقیقا مانند datapath قسمت های مختلف دستور را خوانده و به ماؤول های بعدی میدهد مثلا instruction[6:0] در واقع opcode است را به ماژول control میدهد در ماژول control طبق جدولی که در کتاب وجود دارد بر اساس opcode دستور R-format و lw و sw و .. را تشخیص میدهد و ALUSrc,MemtoReg,RegWrite,MemRead,MemWrite,Branch,Aluop را مقدار دهی میکند که دقیقا مانند datapath هر کدام را به ماژول های مختلف میفرستد و کار آنها را تعیین میکند.

بخش بعدی registerfile است که قسمتهایی از دستور را که در دیتاپت مشخص شده را به آن میدهد و شماره رجیسترهایی که لازم است خوانده شود و کاری که باید انجام شود را مشخص میکند و بر اساس memToReg که از طرف کنترل به آن میرسد که باید چیزی را ذخیره کند در خود یا فقط قرار است اطلاعات را بخواند.

بخش immediateGenerator نیز برای دستور های که به مقدار immediate نیاز دارند است و بخشی از دستور به آن داده میشود تا کارش را انجام دهد.

بعد از آن یک مالتی پلکسر وجود دارد که بر اساس ALUSrc که از طرف کنترل میاید یا مقدار دوم خوانده شده در رجیستر فایل را یا مقدار immediate تولید شده در immediate generator را به ماژول ALU میدهد تا محاسبات لازمه انجام شود.

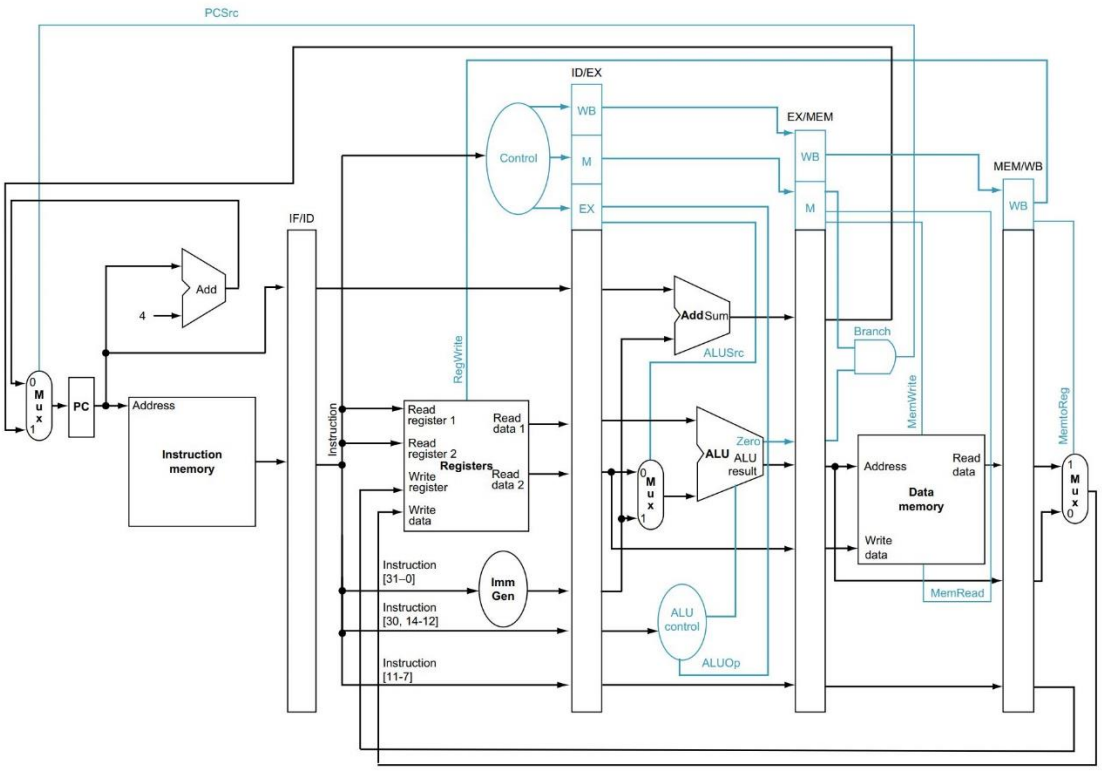
ALU به ALU Control وصل است که ALUControl بک ALUOP از ماژول کنترل میگرد و همینطور قسمتهای func3 func7, از بخش های دستور را مستقیم از طرف instruction memory میگیرد و طبق این موارد یک خروجی 4 بیتی ALUControl میدهد که آن مستقیم به ALU فرستاده میشود.

ALU که ALUControl را گرفته و ورودی هایی از رجیستر و immediate generator دارد طبق ALUControl تصمیم میگیرد که چه عملی را (add,subtract,and,or) را روی داده های ورودی انجام دهد و نهایتا خروجی را به data memory میدهد.

پس از data memory یک مالتی پلکسر است که بر اساس memToReg که از طرف کنترل میاید تصمیم میگیرد که ALUResult را در رجیستر فایل ذخیره کند (برای دستورهای محاسباتی) یا اینکه خروجی data memory را.

این یک خلاصه کلی از نحوه کارکرد کد part 1 پروژه برای single sycle risc v cpu بود که کاملا بر اساس دیتاپت بالا پیاده سازی شده.

Part 2 این پروژه pipeline risc v cpu است که طبق دیتاپت زیر نوشته شده :



برای ماژول های تکراری دقیقا از ماژول های قسمت قبل استفاده شده است و از توضیح تکراری خودداری میکنم.

برای پیاده سازی این دیتاپت 5 قسمت این دیتاپت که با رجیستر از هم جدا شده اند را بطور جداگانه با نام های:

firstStage,secondStage,thirdStage,forthStage,fifthStage به ترتیب از چپ به راست پیاده سازی کرده و در نهایت در فایل PL_CPU آنها را اجرا و به یکدیگر متصل کرده ام.

در ابتدا pc_enable و PCSrc برای شروع مقدار دهی میشود و سپس stage ها اجرا میشوند که در استیج اول از سمت چپ(firstStage) دقیقا طبق دیتاپت پیاده سازی شده و در ابتدا یک مالتی پلکسر وجود دارد که طبق PCSrc تصمیم میگیرد که مقداری که از ماژول addFour میاید را یا ازadder در استیج سوم را به عنوان ادرس به PC بدهد و پس از آن مانند پارت قبل آدرس به instruction memory داده میشود و همینطور برای پیدا کردن آدرس بعدی از ماژول addFour استفاده میشود و نکته ای که وجود دارد این که در این کد هر استیج در واقع شبیه به رجیستر نیز عمل کرده و اطلاعاتی را ذخیره کرده و به استیج بعد نیز ارائه میدهد و رجیستر بصورت جدا پیاده سازی نشده.

در استیج بعدی بخش جداسازی و بررسی دستوری که قرار است اجرا شود است و control ,registerFile,immediate generator که در پارت قبل کامل توضیح داده ام این قسمت صدا زده شده و کار خود را انجام میدهند.

در بخش بعد ALU,ALUControl وجود دارد که مانند پارت قبل کار میکنند و یکadder نیز وجود دارد که آدرس دستور را با مقدار immediate جمع کرده و به استیج بعدی میدهد که در نهایت به مالتی پلکسر در استیج اول داده خواهد شد که اگر PCSrc مقدار مشخصی داشته باشد برای آدرس بعدی دستور از این آدرس استفاده میشود.

در بخش بعدی datamemory و یک and وجود دارد که مقدار zero را با قسمتی از مقداری که از ماژول کنترل آمده را and کرده و بخشی از controlSignal بدست میاید.

در استیج آخر نیز تنها یک مالتی پلکسر وجود دارد که طبق memtoreg برای ذخیره سازی در رجیستر فایل مقداری که از ALU یا از data memory میاید تصمیم میگیرد.

این یک خلاصه کلی از نحوه کارکرد کد part2 پروژه برای pipeline risc v cpu بود که کاملا بر اساس دیتاپت بالا پیاده سازی شده و همانطور که در بالا ذکر شد رجیسترهای جدا کننده استیج ها بطور جداگانه پیاده سازی نشده و هر استیج بطور جدا پیاده سازی و در نقش رجیستر نیز عمل کرده و اطلاعات مورد نیاز را ذخیره و به استیجهای بعدی داده است.

کد این پروژه تست شده و بدون مشکل کار میکند.

توضیحات تکمیلی : در هر دو بخش این پروژه برای هر ماژول testBench ساده نوشته شده است که در کارکرد نهایی پروژه تاثیر ندارد و تنها برای تست کردن هر قسمت و رفع ایرادات آن نوشته شده تا برای اطمینان از نداشتن ارور و یا مشکل منطقی به نوشتن قسمتهای دیگر کد رفته شود.

در این پیاده سازی بخش forwarding unit که hazard ها را کنترل میکند پیاده سازی نشده و احتمال برخوردن به hazard زیاد است در همین مثالی از دستور ها به علت وجود data dependency در x21,x20 به hazard برخوردیم.

مهسا عربی