به نام خدا

هدف پروژه: پیاده سازی معماری mips بر روی FPGA

نحوه ی انجام پروژه: تک نفره

نحوه ی ارائه ی پروژه: ارسال کدها به همراه گزارش کامل در زمان تعیین شده.

گزارش پروژه: عکس simulation waveforms برای خروجی های مشخص شده در طول انجام پروژه، گزارش سنتز پروژه و توضیحات لازم.

آی سی هدف (فقط برای تنظیم در نرم افزار ISE): اسپارتان۶ (xc6slx9-3tqg144)

هدف این پروژه پیاده سازی قسمتهایی از معماری یک پردازنده می باشد. توجه کنید که صورت پروژه به نحوی تدوین شده است که اگر اطلاعات معماری کامپیوتر ندارید قادر به انجام پروژه هستید و مشکلی به وجود نمی آید. در اصل پیاده سازی بلوک های کنترلی که به دانش معماری نیاز دارد در اختیار شما قرار گرفته است.

این معماری از سری معماری های RISC می باشد. در فاز اول پروژه هر یک از ماژول های این معماری را به طور جداگانه پیاده سازی می کنید و در ادامه با استفاده از این ماژول ها ساختار معماری mips را پیاده سازی می کنید. در فاز دوم با کمک توضیحات آورده شده و همچنین ویدئوی ضبط شده درستی عملکرد پیاده سازی خود را با استفاده از نوشتن چند دستورالعمل چک می کنید. و در فاز سوم که به صورت اختیاری است، با استفاده از فصل چهارم کتاب پترسون ماژول های موردنیاز برای انجام دستورالعمل های jump و jal و jr را به معماری mips که پیاده سازی کردید اضافه می نمایید، و انجام این دستورالعمل ها توسط معماری mips را نشان می دهید.

فاز اول

مراحل زیر را برای پیاده سازی کامپیوتر mips انجام دهید:

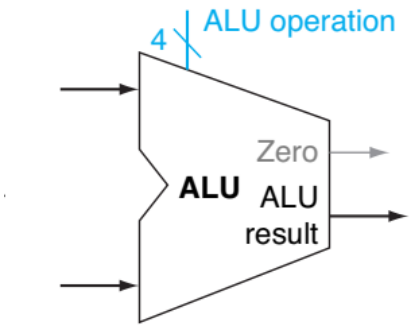
۱- یک رجیستر۳۲ بیتی طراحی کنید که با هر لبه ی بالا رونده ی کلاک، ورودی را لچ کرده و به خروجی انتقال می دهد. این رجیستر در کامپیوتر پایه به عنوان رجیستر PC به کار می رود.

۲- یک مدار اضافه کنند (incrementer) ۳۲ بیتی طراحی کنید که همواره ورودی را یک واحد افزایش داده و در خروجی قرار می دهد.

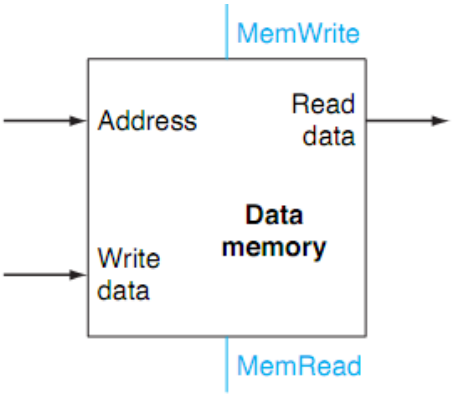
۳- یک مالتی پلکسر ۳۲ بیتی ۲ به ۱ طراحی کنید.

۴- با توجه به جدول زیر یک مدار برای پیاده سازی یک ALU ۳۲ بیتی طراحی کنیدکه عملیات زیر را انجام می دهد. (اگر دو ورودی ALU با هم برابر باشند خروجی zero در این واحد مقدار یک می گیرد.) توجه کنید که خروجی های این مدار باید با لبه بالارونده کلاک لچ شوند.

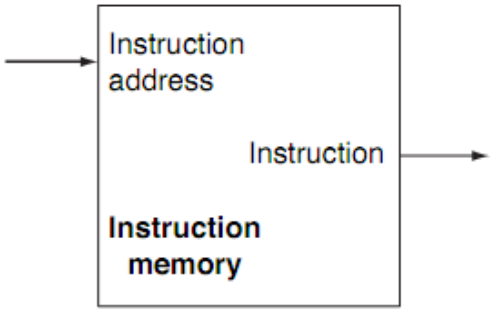
Alu control lines	Function
0000	And
0001	Or
0010	Add
0110	Subtract
0111	Set on less than
1100	Nor



۵-data memory همان طور که از اسم آن مشخص است حافظه ای است که متغیرها و داده های مورد استفاده برنامه در آن قرار می‌گیرد و می‌توان از آن خواند و نوشت. در دستوراتی مانند load و store از این حافظه استفاده می شود. یک data memory با عرض ۳۲ بیت و ارتفاع ۱۲۸ پیاده سازی کنید. (این حافظه باید با توجه به خطوط کنترل، در لبه ی بالارونده ی کلاک داده‌ی داده مورد نظر را از آدرس داده شده بخواند یا در آدرس مورد نظر بنویسد.)



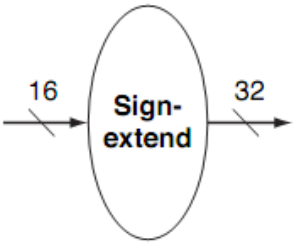
۶- Instruction Memory حافظه ای است که instruction ها در آن ذخیره می‌شوند. Instruction دستورالعملی است که کامپیوتر باید آن را انجام دهد. (در کامپیوتر mips دستور العمل ها به سه نوع R_type و I_type و J_type تقسیم می شوند. دستورالعمل های R_type شامل دستورهای مانند and و add و sub و... در واقع دستورهایی که لازم است عملیاتی روی دو رجیستر صورت بگیرد. دستورالعمل های I_type شامل دستورالعمل های loadو store و addiو ... دستوراتی که با عدد ثابت مانند حافظه یا مقدار ثابت در عملیات محاسباتی کار می کنند و دستورالعمل های J_type شامل دستور jump می باشند.) در این حافظه نمی توان متغیری ذخیره کرد و اصولا ثابت ها مانند دستورالعمل ها و سایر تنظیمات پیکره بندی از این حافظه استفاده می کنند. یک Instruction Memory به عرض ۳۲ بیت و ارتفاع ۳۲ پیاده سازی کنید. این حافظه، حافظه‌ی دستورات است که دستور معادل در آدرس مشخص شده را روی خط خروجی مربوط به دستور قرار می دهد. دستورات در این حافظه به صورت فقط خواندنی در آن نوشته می‌شوند و خط داده ی ورودی برای دستورات در پیاده سازی این واحد حافظه وجود ندارد. (در این پروژه لازم است بعد از پیاده سازی این حافظه، instruction چند دستورالعمل ساده مانند add و sub و... که در فاز دوم مشخص می شوند، برای اطمینان از درست کار کردن معماری شما نوشته شود. این دستورالعمل ها طبق فرمت هر دستورالعمل که در فاز دوم مشخص شده است، به خانه های این حافظه به صورت ثابت داده می شود.)



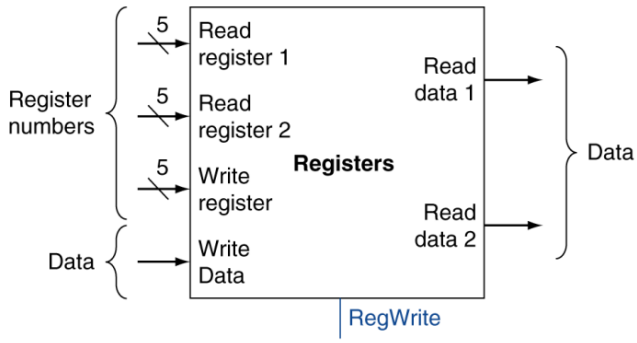
۷-گسترش بیت علامت عملیاتی است که در آن تعداد بیت های اعداد باینری افزایش می‌یابد اما علامت و مقدار آنها تغییر نمی کند. به این شکل که بیت ها از سمت چپ (پر ارزش) بر مبنای علامت عدد مورد نظر اضافه می شوند. برای مثال اگر از ۴ بیت برای نمایش عدد مثبت ۳ استفاده کنیم داریم : ۰۰۱۱ در این حالت نمایش ۱۶ بیتی آن با استفاده از گسترش علامت عبارت است از 0011 0000 0000 0000 با این کار مقدار و علامت حفظ شد. برای اعداد منفی برای مثال ۳- نمایش مکمل دو آن در ۴ بیت عبارت است از 1101 ،که در حالت ۱۶ بیتی این عدد برابر با 1101 1111 1111 1111می باشد.

کاربرد گسترش علامت زمانی است که دو عدد با تعداد بیت مختلف، نیاز است عملیات محاسباتی انجام دهند. در این حالت باید تعداد بیت ها یکسان باشند. از این رو با استفاده از Sig Extend تعداد بیت های دو عدد را گسترش داده و برابر می کنیم.

یک واحد Sign Extend ۱۶ به ۳۲ را پیاده سازی کنید.(این واحد ۱۶ بیت ورودی گرفته و **در لبه ی بالارونده ی کلاک بدون تغییر کردن ارزش ورودی**، آن را به ۳۲ بیت گسترش می دهد.)

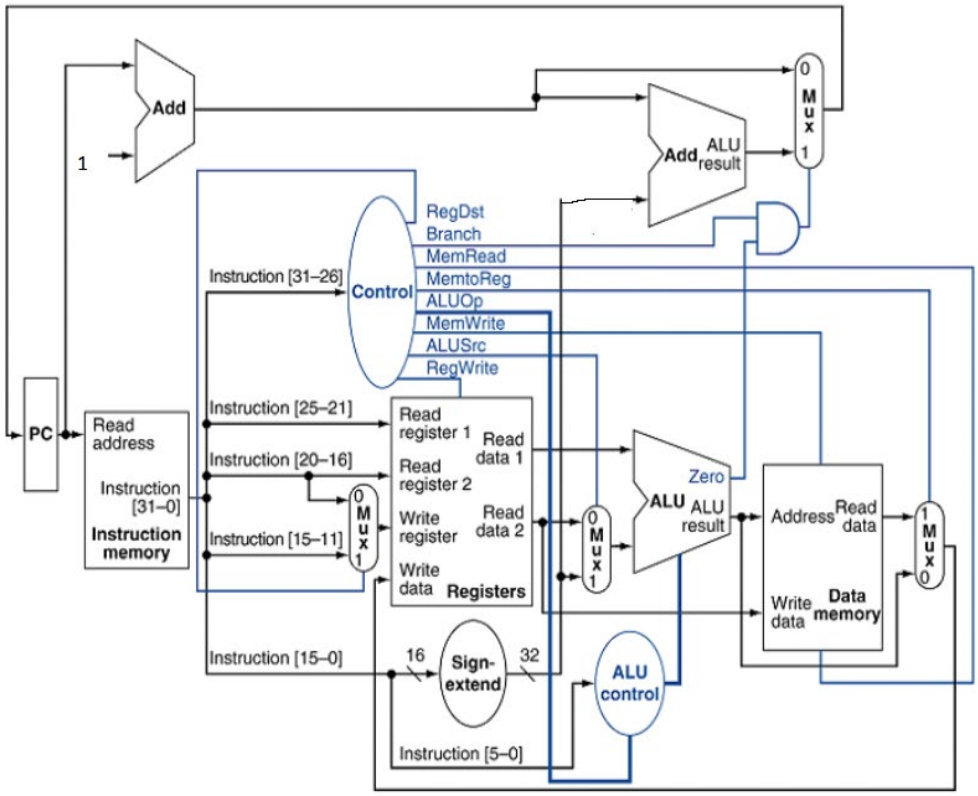


۸-وقتی تعداد رجیسترهای معماری زیاد شود از رجیستر بانک استفاده می شود. یک رجیستر بانک که به صورت ۳۲ رجیستر ۳۲ بیتی است پیاده سازی کنید. در این ساختار اطلاعات رجیستری که دارای آدرس رجیستر Read register1 روی خروجی Read data1 و اطلاعات رجیستری که دارای آدرس رجیستر Read register 2 روی خروجی Read data2 قرار می گیرد. همچنین با لبه ی بالارونده ی کلاک در صورت یک بودن خط کنترل Reg Write ، write Data در آدرس write register نوشته می شود. (در پیاده سازی به رجیستر صفر به صورت ثابت مقدار صفر دهید، از این رجیستر در ادامه در نوشتن دستورالعمل ها استفاده می شود)



۹-هر یک از ماژول هایی که گفته شد را با توجه به ساختار MIPS که در ادامه آورده شده، به هم متصل کنید. فایل واحد Control و واحد ALUcontrol که در پوشه ی پروژه قرار داده شده را به ساختاری که تاکنون

پیاده سازی کرده اید اضافه نمایید. (ماژول add فقط عملیات جمع را انجام می دهد. یک ماژول جدا برای آن طراحی کنید.)



شکل ۱- معماری MIPS

فاز دوم

در فاز اول پروژه یک ساختار کامپیوتر پیاده سازی شد. در این فاز قصد داریم این کامپیوتر را تست کنیم. و درستی عملکرد آن را بررسی نماییم.

برای بررسی درستی عملکرد کامپیوتری که پیاده سازی کرده اید لازم است در instruction memory چند دستورالعمل بنویسید و درستی انجام این دستورالعمل ها را در ساختار کامپیوتری که پیاده سازی کردید ببینید.

۱-برای انجام این دستورالعمل ها

$R_1=R_0+3$ I-Type

$R_2=R_0+3$ I-Type

$R_3=R_1+R_2$ R-Type

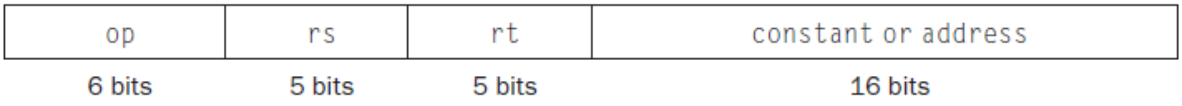
Instruction های زیر را در instruction memory بنویسید.

rom[0]	32'b001000_00000_00001_0000000000000011
rom[1]	32'b001000_00000_00010_0000000000000011
rom[2]	32'b000000_00001_00010_00011_00000_011000

توضیح هر کدام از این instruction ها در ادامه آورده شده است.

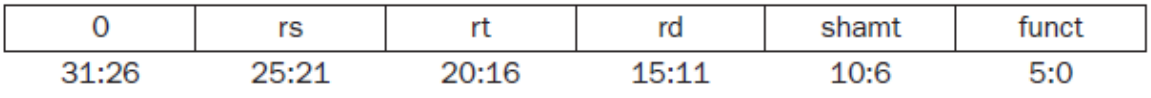
ریجستر R_0 همان ریجستری هست که در هنگام پیاده سازی ریجستر بانک مقدار صفر گرفته است. دو دستور اول در واقع مقدار دهی اولیه و دستور سوم جمع این دو مقدار می باشند. برای نوشتن instruction ها به موارد زیر دقت شده است.

ساختار تقسیم بندی instruction در دستورات I-Type



۶ بیت ابتدایی بیت های ۳۱ تا ۲۶ opcode که در دستور addi این مقدار ۰۰۱۰۰۰ است و rs آدرس رجیستری است که با عدد ثابت جمع می شود که در این مثال این مقدار صفر می باشد. و rt آدرس رجیستر مقصد که در این مثال برای دستورالعمل اول مقدار ۱ و برای دستورالعمل دوم مقدار ۲ می باشد. و ۱۶ بیت انتهایی عدد ثابت می باشد که برای این دو دستورالعمل ۳ می باشد.

ساختار تقسیم بندی instruction در دستورات R-Type



- addu rd, rs, rt
- subu rd, rs, rt

rs و rd آدرس دو رجیستر که عملیات محاسباتی روی آنها انجام می شود. در این مثال مقادیر ۲و ۱ می باشند. rd آدرس ریجستر مقصد است که در این مثال مقدار ۳ می باشد. و مقادیر ۳۱ تا ۲۶ opcode دستورالعمل که در دستورات R_type مقدار opcode صفر می باشد. و مقدار shamt را صفر قرار می دهیم و مقدار funct عملیاتی که باید روی رجیستر ها انجام بگیرد که در alu control مشخص می شود، در این instruction این مقدار را ۰۱۱۰۰۰ قرار می دهیم.

بعد از اجرای این دستورالعمل ها خروجی واحدهای رجیستر بانک ، ALU و mux بعد از واحد data memory را برای درستی ساختار معماری خود در سیمولیشن ببینید، و اسکرین شات آن ها را در گزارش پروژه خود بیاورید.

۲- در قسمت یک برخی از دستورالعمل ها را اجرا کردیم و دیدم که ساختار معماری که پیاده سازی کردیم توانست این دستورات را اجرا کند. در این قسمت برخی دیگر از دستورات که مربوط به حافظه ی معماری می باشد را اجرا می کنیم.

برای انجام این دستورالعمل ها

1. lw R_1, 10 (R_2) => R_1= Mem(R_2 + 100) I-Type
2. beq R_2, R_1, 20 => if R_2= R_1then PC = 20 I-Type

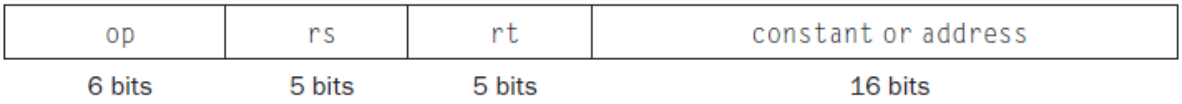
Instruction های زیر را در instruction memory بنویسید.

rom[3]	32'b100011_00010_00001_0000000000001010
rom[4]	32'b000100_00001_00010_0000000000010100

Instruction های این دستورات با توجه به توضیحات قسمت ۱ نوشته شده است، خروجی ، واحد های data memory و mux بعد از واحد ADD و همچنین ALU را برای درستی ساختار کامپیوتر خود در سیمولیشن ببینید، و اسکرین شات آن ها را در گزارش پروژه خود بیاورید.

توضیحات شیوه ی نوشتن این دستورالعمل ها در ادامه آورده شده است.

Opcode که شامل ۶ بیت ۳۱تا ۲۶ instruction هست، در دستور branch برابر با۰۰۰۱۰۰ و در دستور load برابر با ۱۰۰۰۱۱ و بقیه ی بیت های instruction همانگونه که در قسمت یک توضیح داده شد، تعیین می شوند.



در دستور load ۱۶ بیت انتهایی آدرس می باشد که در این مثال مقدار ۱۰ گرفته است. و rs آدرس رجیستری است که لازم است با مقدار آدرس جمع شود، در این مثال این مقدار۲ می باشد. و rt آدرس رجیستر مقصد، که در این مثال مقدار ۱ می باشد. همچنین در دستور branch دو رجیستر rs و rt با هم مقایسه می شوند و در صورت برابر بودن مقدار این دو رجیستر ، مقدار PC با آدرسی که توسط بیت های انتهایی instruction مشخص شده یعنی ۱۰۱۰۰ جمع می شود.

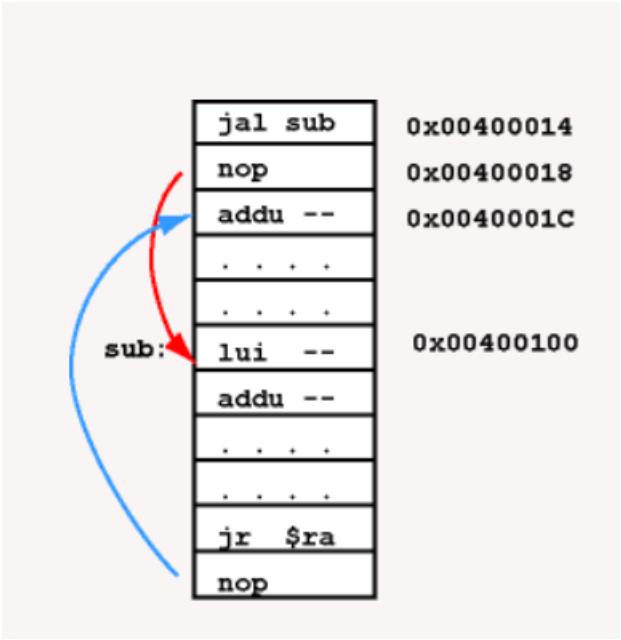
۳- instruction دستوراتی که در قسمت یک و دو آورده شده، در یک فایل instruction.mem برای شما داده شده است. این دستورات را از این فایل بخوانید و مانند قسمت های یک و دو خروجی ها را چک نمایید.

ساختار کامپیوتر خود را سنتز کنید، و گزارش سنتز و مقدار FPGA استفاده شده را در گزارش پروژه ی خود بیاورید.

فاز سوم

این قسمت به عنوان اختیاری انتخاب شده و شامل نمره ی اضافی می باشد.

واحدهای مورد نیاز برای انجام دستورالعمل های jump و jal و jr (دستورات فراخوانی ساب روتین)را به ساختار کامپیوتری که پیاده سازی کردید اضافه کنید. و در انتها برای بررسی درستی ساختار خود instruction متناظر این سه دستورالعمل را در instruction memory بنویسید. و از خروجی ماژول هایی از ساختار معماری که نشان دهد ساختار معماری شما به خوبی این دستورات را انجام میدهد اسکرین شات بگیرید و در گزارش خود بیاورید. لازم است در گزارش خود کامل توضیح دهید چه واحد هایی به ساختار خود اضافه کردید و چرا این واحدها اضافه شده اند. همچنین ساختار control و ALU control که در فاز هایی قبل نیازی به نوشتن و تغییر این دوقسمت نبود در این قسمت این دو واحد را متناسب با دستورالعمل هایی جدید لازم است تغییر دهید. (فصل چهارم کتاب پترسن که ضمیمه شده است می تواند شما را در این قسمت راهنمایی کند. این قسمت نیز باید قابل سنتز باشد و گزارش سنتز و میزان fpga که استفاده شده است گزارش شود.)


jal sub
nop
addu --
. . . .
. . . .
sub:
lui --
addu --
. . . .
. . . .
jr $ra
nop
0x00400014
0x00400018
0x0040001C
0x00400100