به نام خدا

پروژه‌ی درس معماری کامپیوتر

کلیات

در این پروژه شما یک پردازنده‌ی مبتنی بر معماری MIPS را پیاده سازی خواهید کرد .این پروژه شامل چهار فاز است که در هر فاز قابلیت‌هایی به پردازنده‌ی پیاده سازی شده در فاز قبل اضافه می‌شود.

برای پیاده‌سازی از زبان Verilog یا SystemVerilog استفاده کنید.

کارگاهی برای آشنایی مقدماتی با Verilog و توضیحات تکمیلی پروژه تشکیل خواهد شد.

در هر فاز پروژه‌ی شما بر روی تعدادی تست از پیش طراحی شده اجرا خواهد شد تا از درستی عملکرد آن اطمینان حاصل شود. هر تست یک کد اسمبلی MIPS است که پس از ترجمه به زبان ماشین در حافظه‌ی پردازنده قرار می‌گیرد و در پایان اجرا مقادیر موجود در رجیسترها با مقادیر مورد انتظار مقایسه می‌شوند و در صورت یکسان بودن تست موفقیت‌آمیز در نظر گرفته می‌شود.

هنگام پیاده‌سازی نیز می‌توانید مرحله به مرحله این تست‌ها را اجرا کنید تا از درستی بخش‌های پیاده‌سازی شده اطمینان پیدا کنید. همچنین در صورت نیاز می‌توانید خودتان تست‌هایی را به این مجموعه اضافه کنید.

برای اجرای تست‌ها دستور make verify را در root پروژه اجرا کنید. با اجرای این دستور پروژه‌ی شما با استفاده از Verilator شبیه‌سازی شده و همه‌ی تست‌ها روی آن اجرا خواهد شد. توجه کنید که از docker برای اجرای Verilator استفاده شده است. بنابراین ابتدا باید docker روی سیستم شما نصب باشد. برای نصب docker به این لینک مراجعه کنید. توجه کنید که docker ایران را تحریم کرده است. یک راه مناسب برای دور زدن این تحریم استفاده از docker.ir است. پس از نصب docker صرفا کافی است دستور make verify-all را اجرا کنید.

می‌توانید این پروژه را در قالب گروه‌های حداکثر چهار نفره انجام دهید.

پس از نهایی شدن گروه خود را در Github Classroom ثبت کنید. شما باید در طول انجام پروژه به صورت مداوم کارهای خود را درون مخزن خود push کنید و در پایان هر فاز با یک تگ نسخه‌ی نهایی آن فاز را مشخص کنید. هنگام تحویل این تگ‌ها بررسی خواهند شد.

جهت دسترسی به Github Classroom از این لینک استفاده کنید.

برای هر فاز گزارشی شامل معماری سطح بالای آن فاز به همراه توضیح مختصر آن تهیه کنید.

در پایان پروژه جلسه‌ای برای تحویل آن فاز تشکیل خواهد شد.

موعد تحویل فازها با در نظر گرفتن امتحانات و سایر شرایط تعیین شده و به هیچ عنوان تمدید نخواهد شد. توجه داشته باشید که موعد تحویل فاز سوم بلافاصله پس از پایان امتحانات است. توصیه می‌شود قبل از شروع امتحانات بخش‌های زیادی از آن را انجام دهید. همچنین مجموعا ۵ روز تاخیر بدون کسر نمره برای کل پروژه در نظر گرفته شده که می‌توانید به دلخواه روی فازهای مختلف استفاده کنید.

زیرساخت و موارد لازم برای انجام فازهای پروژه قبل از شروع هر فاز از طریق CW و همچنین Git Repo قابل دسترسی خواهد بود.

جهت دسترسی به Git Repo از این لینک استفاده کنید.

ماژول حافظه‌ی RAM و Register File به صورت آماده در اختیار شما قرار داده شده است. حتما از همین ماژول‌ها استفاده کنید.

برای پرسیدن سوالات خود از CW استفاده کنید.

در صورت هر گونه شباهت غیرمتعارف میان پروژه‌های دو تیم، نمره‌ی پروژه‌ی هر دو تیم صفر لحاظ خواهد شد.

فازها	فاز اول	فاز دوم	فاز سوم	فاز چهارم
موعد تحویل	۱۴۰۱/۳/۲	۱۴۰۱/۳/۱۶	۱۴۰۱/۴/۹	۱۴۰۱/۴/۳۱

فاز اول پروژه:

در این فاز پردازنده ای در Modelsim و یا Quartus و یا هر محیط دیگری که با آن آشنایی دارید شبیه سازی کنید که فرمت J، R و I را در معماری MIPS با دستوراتی که در ادامه ذکر شده است را ساپورت کند.

۱- فرمت R:

Opcode	rs[3]	rt[2]	rd[1]	Sh.Amount	Func
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

در این نوع فرمت دستورات شامل سه رجیستر است که دو رجیستر مبدا هستند و یک رجیستر مقصد است.

فیلد Func برای انجام دستورات محاسباتی استفاده می شود.

برای مثال در دستور زیر مقدار Func ۳۲ است و به عمل جمع اشاره دارد و دو رجیستر مبدا r1 و r2 با هم جمع می شود و پاسخ در رجیستر مقصد r3 ریخته می شود:

Opcode	rs	rt	rd	Sh.Amount	Func
000000	00001	00010	00011	00000	100000

دستورات شیفت به چپ و راست از فیلد Sh.Amount استفاده می کند تا مقدار شیفت را مشخص کنند.

لیست دستورات:

#	Instruction Name	Meaning	Func
1	XOR	rd ← rs ^ rt	100110
2	SLL(Shift left logical)	rd ← rt << Sh.AMOUNT	000000
3	SLLV(shift left logical variable	rd ← rt << rs	000100
4	SRL *(Shift right logical)Unsigned right shift*	rd ← rt >> Sh.AMOUNT	000010
5	SUB	rd ← rs – rt	100010
6	SRLV(shift right logical variable)	rd ← rt >> rs	000110
7	SLT	rd ←rs<rt signed comparison	101010
8	Syscall	Finish cpu opration	001100
9	SUBU(Subtract unsigned)	rd ← rs – rt	100011
10	OR	rd ← rs| rt	100101
11	NOR	rd ← rs ~| rt	100111
12	ADDu(Add unsigned)	rd ← rt + rs	100001
13	MULT	rd ← rs × rt	011000
14	DIV	rd ← rs ÷ rt	011010
15	AND	rd ← rs & rt	100100
16	ADD	rd ← rs + rt	100000

[1] Destination Register
[2] Source Register
[3] Source Register

17	JR(Jump Reg)	PC← rs	001000
18	SRA(*signed right shift)*	rd ← rt >> Sh.AMOUNT	000011

توجه: تمام دستورات بالا Opcode برابر صفر(000000) دارند و فیلد Func مسئولیت ایجاد تمایز بین دستورات را دارد..

۲- فرمت I :

Opcode	rs	rt	[۴]Imm
6 bits	5 bits	5 bits	16 bts

لیست دستورات:

#	Instruction Name	Meaning	Opcode
1	ADDi	rt ← rs + SIGN EXTEND (Imm)	001000
2	ADDiu(unsigned)	rt ←rs + SIGN EXTEND (Imm)	001001
3	ANDi	rt← rs & SIGN EXTEND (Imm)	001100
4	XORi	rt←rs ~ SIGN EXTEND (Imm)	001110
5	ORi	rt←rs| SIGN EXTEND (Imm)	001101
6	BEQ	rs == rt: PC ← PC + SIGN EXTEND(Imm | "00")	000100
7	BNE	rs!= rt: PC ← PC + SIGN EXTEND (Imm | "00")	000101
8	BLEZ	rs<= 0: PC ← PC + SIGN EXTEND (Imm | "00")	000110
9	BGTZ	rs > 0: PC ← PC + SIGN EXTEND (Imm| "00")	000111
10	BGEZ	rs >= 0: PC ← PC + SIGN EXTEND (Imm| "00")	000001
11	LW	rt ← MEM [$rs+ SIGN EXTEND (Imm)]	100011
12	SW	MEM [$rs+ SIGN EXTEND (Imm)] ← rt	101011
13	LB	rt[7:0] ← MEM [$rs+ SIGN EXTEND (Imm)]	100000
14	SB	MEM [$rs+ SIGN EXTEND (Imm)] ← rt [7:0]	101000
15	SLTi	Set to 1 if Less, rs< SIGN EXTEND (Imm) , rt=1	001010
16	Lui(load upper immediate)	The immediate value is shifted left 16 bits and store in register. The lower 16 bits are zeroes rt← {SIGN EXTEND (Imm),0*16}	001111

[4] Immediate

توجه: فیلد Opcode مسئولیت تمایز بین دستورات بالا را دارد و مخالف صفر است.

۳- فرمت دستور J :

Opcode	address
6 bits	**26 bits**

لیست دستورات:

#	Instruction Name	Meaning	opcode	Comments
۱	j	PC←{(PC), address,00}	000010	Jump to target address
2	[5]JAL	R[31] ←PC then go to procedure address PC←{(PC), address,00}	000011	Use when making procedure call. This saves the return address in $31

❖ جهت بررسی بیشتر دستورات می توانید به لینک زیر مراجعه کنید:

https://inst.eecs.berkeley.edu/~cs61c/resources/MIPS_help.html

توضیحات:

۱) طول کلمه در این معماری 32 بیت است .
۲) تعداد ثباتهای عمومی 32 است .
۳) (توجه شود این فاز به صورت Single cycle است)
۴) توجه کنید پردازنده را به گونه ای طراحی کنید که برای فازهای آتی قابل تعمیم باشد. برای مثال بتوان به آن خط لوله [6]اضافه کرد.

فاز دوم پروژه:

حافظه‌ی اصلی داده شده در فاز قبل در یک کلاک پاسخ را حاضر می‌کرد. در حالی که در عمل چنین نیست و دسترسی به حافظه‌ی اصلی با تاخیر زیادی همراه است. حافظه‌ای که در این فاز در اختیار شما قرار گرفته این تاخیر را شبیه‌سازی می‌کند و شما باید از آن به جای حافظه‌ی فاز قبل استفاده کنید. این حافظه برای حاضر کردن جواب به چند کلاک زمان نیاز دارد. ابتدا پردازنده‌ی خود را به گونه‌ای تغییر دهید که بتواند در دستورات مربوط به حافظه چند کلاک متوقف شود تا عملیات به درستی انجام شود. سپس با پیاده‌سازی و اضافه کردن یک حافظه‌ی نهان سطح ۱، توقف‌های به وجود آمده به هنگام دسترسی به حافظه را به حداقل برسانید.
این حافظه باید از نوع write back باشد اما در مورد ظرفیت و اندازه‌ی بلوک حافظه آزاد هستید و مدل mapping حافظه را هم می‌توانید Direct mapped یا set-associative در نظر بگیرید.

[5] JAL instruction Jumps to the calculated address and stores the return address in $31
[6] Pipeline

توجه داشته باشید که حافظه‌ی پیاده‌سازی شده باید در کلاک پیشبینی شده برای پردازنده کار کند. لذا با توجه به این که ابعاد حافظه و اندازه‌ی set ها روی تاخیر آن تاثیر مستقیم دارند، آن‌ها را با دقت انتخاب کنید. می‌توانید حافظه‌ی خود را به صورت پارامتری طراحی کنید و با سنتز کردن با مقادیر مختلف مقدار مناسب را بیابید.

فاز سوم پروژه:

در این فاز دستورالعمل‌ها به صورت **خط لوله** [7] اجرا شوند. تعداد طبقه های خط لوله حداقل باید **۳** باشد. توجه کنید در نظر گرفتن تعداد بیشتر طبقه کاملا اختیاری است. همچنین توجه کنید هر چه تعداد طبقه‌ها بیشتر شود عملکرد پردازنده بیشتر نخواهد شد. بنابراین هر تعداد طبقه که در نظر می‌گیرید باید دلیلی برای آن داشته باشید.

فاز چهارم پروژه:

در این مرحله، می‌بایست از بین پروژه‌های **الف** تا **د** یکی را به دلخواه انتخاب کرده و کمک پردازنده‌ی متناظر با آن را طراحی کنید. سپس، باید پردازنده‌ی کمکی خود را به پردازنده‌ی اصلی که در فازهای قبل طراحی کرده‌اید، متصل کنید تا پردازنده‌ی اصلی بتواند از این واحد برای اجرای دستورالعمل‌های کمکی استفاده کند. جزئیات طراحی کمک پردازنده، نحوه‌ی اتصال پردازنده‌ها و طراحی ساختار دستورالعمل‌ها برای کمک پردازنده بر عهده دانشجویان است. توجه داشته باشید که ساختار دستورالعمل‌های کمک پردازنده باید مطابق با فرمت ساختار دستورالعمل‌های پردازنده‌ی اصلی باشد. همچنین، برای مدیریت ارتباط بین پردازنده‌ی اصلی و پردازنده‌ی کمکی می‌توانید به تعداد مورد نیاز، دستور به مجموعه دستورالعمل‌های پردازنده‌ی اصلی اضافه کنید.

درنهایت، باید مشخص گردد که هر یک از عملیات خواسته شده در چند سیکل انجام می شود.

الف) کمک پردازنده برای محاسبات برداری: این کمک پردازنده باید قادر به انجام دستورالعمل‌های جمع، تفریق، ضرب نقطه‌ای، ضرب عدد در بردار، تقسیم بردار بر عدد، نرمال کردن و محاسبه‌ی اندازه روی بردارها به صورت predicate باشد. (یعنی عملیات برداری روی زیرمجموعه‌ای از عناصر بردار که توسط ماسک [8]مشخص می‌شود انجام شود).

همچنین، می‌توانید در صورت نیاز دستورالعمل‌های دسترسی به حافظه، کنترلی و پرشی برای کمک پردازنده در نظر بگیرید. کمک پردازنده باید بتواند یک بردار را از حافظه خوانده و آن را در حافظه ذخیره نماید. پس از اتمام مراحل طراحی، کمک پردازنده‌ی خود را توسط برنامه‌ای که تمام دستورالعمل‌های نام برده در آن وجود داشته باشد، بیازمایید. پس از اطمینان از صحت اجرای تمامی دستورالعمل ها، برنامه‌ای بنویسید که در آن دو ماتریس n×n را از ورودی گرفته و با استفاده از تابعpredicate عناصر صفر آنها را تبدیل به یک کرده و سپس دو ماتریس را در هم ضرب کند.

ب) کمک پردازنده برای اعداد مختلط: این کمک پردازنده باید قادر به انجام دستورات جمع، تفریق، ضرب، تقسیم، مقایسه، معکوس، مزدوج و تبدیل نمایش قطبی به نمایی و یا بالعکس روی اعداد مختلط باشد. همچنین، میتوانید در صورت نیاز دستورالعمل‌های دسترسی به حافظه، کنترلی و پرشی برای کمک پردازنده در نظر بگیرید.

کمک پردازنده باید بتواند یک عدد مختلط را به هر یک از دو فرمت قطبی و یا نمایی از حافظه خوانده و آن را در حافظه ذخیره نماید. پس از اتمام مراحل طراحی، کمک پردازنده‌ی خود را توسط برنامه‌ای که تمام دستورالعمل‌های نام برده در آن وجود داشته باشد، بیازمایید. پس از

[7] Pipeline
[8] Mask

اطمینان از صحت اجرای تمامی دستورالعمل‌ها، برنامه‌ای بنویسید که در آن دو عدد مختلط به فرمت قطبی را از ورودی گرفته و مزدوج حاصل تقسیم را به فرمت نمایی ذخیره کند.

ج) کمک پردازنده برای اعداد ممیز شناور: این کمک پردازنده باید قادر به انجام دستورات جمع، تفریق، ضرب، تقسیم، مقایسه، معکوس و گرد کردن (به نزدیکترین عدد صحیح) روی اعداد ممیز شناور با دقت ساده [9]براساس استاندارد IEEE-754 باشد.

همچنین، می‌توانید در صورت نیاز دستورالعمل‌های دسترسی به حافظه، کنترلی و پرشی برای کمک پردازنده در نظر بگیرید. همانند موارد استثناء در نظر گرفته شده در استاندارد IEEE[10] ، برای هر یک از موارد زیر باید در خروجی سیگنالی وجود داشته باشد که آن ها را گزارش کند. این موارد عبارتند از:

- Division by zero:
- QNaN (quiet not a number) :
- SNaN (signaling not a number):
- Inexact:
- Underflow:
- Overflow:

پردازنده باید بتواند یک عدد ممیز شناور را از حافظه خوانده و آن را در حافظه ذخیره نماید. پس از اتمام مراحل طراحی، کمک پردازندهی خود را توسط برنامه‌ای که تمام دستورالعمل‌های نام برده در آن وجود داشته باشد، بیازمایید. پس از اطمینان از صحت اجرای تمامی دستورالعمل‌ها، برنامه‌ای بنویسید که در آن دو عدد ممیز شناور را از ورودی گرفته، عدد بزرگتر را بر عدد کوچکتر تقسیم کرده و نتیجه‌ی گرد شده را ذخیره کند .

د) **کمک پردازنده برای چند جمله‌ای‌ها**: این کمک پردازنده باید قادر به انجام دستورات جمع، تفریق، ضرب، تقسیم، مشتق و محاسبه‌ی مقدار در یک نقطه روی چند جمله‌ای‌ها باشد. همچنین، می‌توانید در صورت نیاز دستورالعمل‌های دسترسی به حافظه، کنترلی و پرشی برای کمک پردازنده در نظر بگیرید. پردازنده باید بتواند یک چند جمله‌ای را از حافظه خوانده و آن را در حافظه ذخیره نماید. پس از اتمام مراحل طراحی، کمک پردازنده‌ی خود را توسط برنامه‌ای که تمام دستورالعمل‌های نام برده در آن وجود داشته باشد، بیازمایید. پس از اتمام مراحل طراحی، برنامه‌ای بنویسید که در آن یک چند جمله‌ای از ورودی گرفته و ریشه‌ی آن را با استفاده از روش نیوتن رافسون محاسبه و ذخیره کند.

[9] Single precision
[10] IEEE Standard for Floating-Point Arithmetic," in IEEE Std 754-2008 , vol., no., pp.1-70, 29 Aug. 2008, doi: 10.1109/IEEESTD.2008.4610935