به نام خدا

دانشکده مهندسی کامپیوتر

درس معماری کامپیوتر
نیم‌سال دوم ۰۲-۰۳
استاد: دکتر حسین اسدی

تمرین سری ششم

- پرسش‌های خود را در صفحه quera مربوط به تمرین مطرح نمایید.
- سوالات نظری را حتماً به صورت انفرادی و سوالات عملی را می‌توانید در گروه‌های دو نفر تحویل دهید.
- پاسخ‌ها را به صورت تایپی بنویسید.
- اسکرین‌شات‌ها، عکس‌ها و فایل‌های مربوط به سوال عملی را در فایل فشرده مربوطه در cw و quera قرار دهید. هر گونه عدم تطابق بین دو تمرین آپلود شده در دو سایت منجر به از دست رفتن نمره تمرین مربوطه می‌شود.
- پی دی اف قسمت تئوری را در سامانه cw و quera بارگذاری کنید.
- هر دانشجو می‌تواند حداکثر سه تمرین را با دو روز تأخیر بدون کاهش نمره ارسال نماید.

تمارین تئوری

۱. جدول زیر زمان اجرای مراحل مختلف انواع دستورات روی یک پردازنده MIPS را به نانوثانیه نشان می‌دهد:

Type	I Cache	Decode	ALU	PC Update	D Cache	R Write
R-Type	0.9	0.8	0.8	-	-	0.9
Load	0.9	0.8	0.8	-	1	0.9
Store	0.9	0.8	0.8	-	1	-
Branch	0.9	0.8	0.8	0.2	-	-

فرض کنید setup time برابر 0.05ns و hold time برابر 0.05ns است.

آ) فرض کنید این پردازنده single cycle است. با این فرض throughput اجرای دستورات را محاسبه کنید.

ب) فرض کنید این پردازنده multi cycle است و خط‌های قرمز جدول بالا، لبه‌های clock را نشان می‌دهند. through-put اجرای هر کدام از انواع دستورات را محاسبه کنید.

۲. مقدار سیگنال‌های کنترلی مورد نیاز در هر کلاک را برای هریک از موارد زیر بنویسید.

آ) `mul $t8, $t9, $s0`

ب) `j loop`

۳. زمان‌های واحدهای اصلی در یک مسیرداده را به صورت زیر در نظر بگیرید:

۱. Memory access (read and write)= 25 ps

۲. ALU = 20 ps

۳. Register file (read and write) = 15 ps

حداقل زمان چرخه ساعت (min clock cycle time) ، میانگین CPI و میانگین زمان اجرای دستورالعمل را برای مسیرداده multi-cycle و single-cycle زیر را تعیین کنید.
10% lw, 10% sw, 40% register-type, 20% branch, 20% jump

۴. پردازنده Multi-cycle MIPS را در نظر بگیرید. فرض کنید می‌خواهیم دستور زیر را به آن اضافه کنیم:

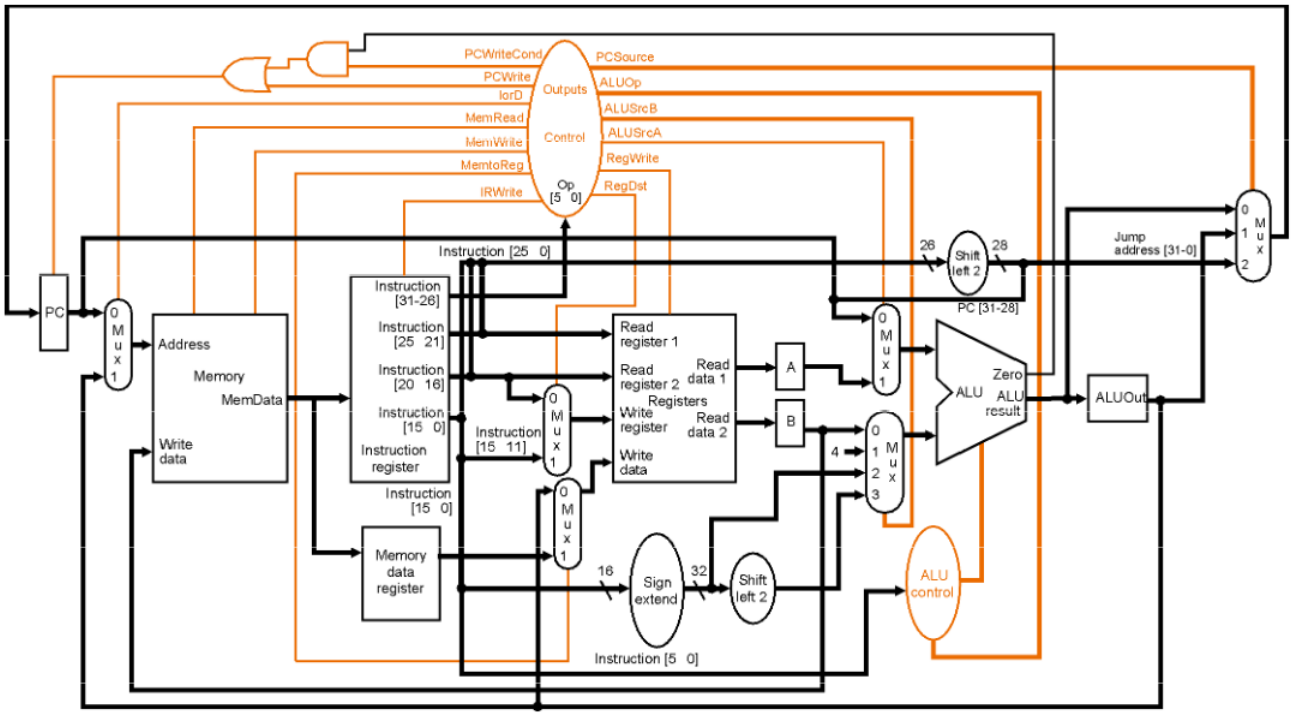
addm rt, rd, rs

که در این دستور

rd = rs + Mem[rt]

این دستور عملا برخلاف دستورهای رایج R-Format که ساختار Register-Register دارند، ساختار Register-Memory داشته و هم حافظه و هم ثبات‌ها در عملیات جمع دخیل هستند.

آ) در مسیرداده زیر تغییرات لازم را برای ساخت این دستور اعمال کنید و علت تغییر مدار خود را توضیح دهید. توجه کنید که حق تغییر در واحدهای حافظه، ثبات ها و ALU را ندارید ولی سایر بخش‌های مدار را می‌توانید تغییر بدهید.



ب) حالت‌های مربوط به این دستور را به FSM پردازنده اضافه کنید.

۵. شرکتی یک ISA شامل سه دسته دستور دارد و طی سه نسل برای آن سه معماری متفاوت ایجاد کرده است:

- معماری اول: دستورات نوع A و B و C به ترتیب در 2, 4, 8 چرخه ساعت اجرا می‌شوند.
- معماری دوم: دستورات نوع A به میزان 62.5 % و دستورات نوع B به میزان ۲۵% و دستورات نوع C به میزان ۵۰% سریعتر شده‌اند.
- معماری سوم (نسبت به معماری اول): دستورات نوع A به میزان 87.5 % و دستورات نوع B و نوع C هر کدام ۵۰% سریعتر شده‌اند.

حال اگر یک مجموعه دستور داشته باشیم که اجرای آن در معماری دوم ۲ برابر سریعتر از معماری اول و در معماری سوم 1.5 برابر سریعتر از معماری دوم باشد، چند درصد از دستورات آن از نوع A چند درصد از نوع B و چند درصد از نوع C هستند؟

۶. به سوالات زیر درباره میکروکدها پاسخ دهید:

آ) مفهوم Micro-Code و نقش آن در پردازنده را توضیح دهید.

ب) نقش Micro-Code در آسیب‌پذیری‌های مربوط به پردازنده و همچنین روش برطرف کردن آن‌ها را توضیح دهید.

ج) توضیح دهید به روزرسانی Micro-code ها از چه راه‌هایی انجام می‌شود و همچنین تحقیق کنید که اگر امکان به روزرسانی Micro-code ها وجود نداشت چه مشکلاتی می‌توانست رخ دهد.

تمارین عملی

یادآوری

در تمرین‌های گذشته یک پردازنده‌ی میپس ساده شده را طراحی کردیم. مانند پردازنده‌ی MIPS این پردازنده نیز سه نوع کدگذاری دستور دارد: Register, Immediate, Jump. تعریف هر کدام از این دستورات در زیر آمده است:

- **Register Instructions**: این دستورات همان طور که از اسم آن‌ها پیدا است برای زمانی استفاده می‌شوند که قرار است به کمک محتوای دو ثبات، یک ثبات دیگر را مقداردهی کنیم. این دستورات دارای فرمت زیر هستند:

opcode	rs	rt	rd	funct
4 bits	3 bits	3 bits	3 bits	3 bits

دقیقا مثل پردازنده‌ی MIPS در تمامی دستورات این نوع، ثبات opcode آن‌ها برابر صفر است و بر اساس مقدار سیگنال funct می‌توان نوع عملیات را تعیین کرد. جدول عملیات در زیر آمده است:

Mnemonic	Operation	funct
ADD	rd ← rs + rt	000
SUB	rd ← rs - rt	001
AND	rd ← rs & rt	010
OR	rd ← rs | rt	011
MULT	rd ← rs * rt	100
XOR	rd ← rs ^ rt	101
JR	PC ← rs	111

- **Immediate Instructions**: این دستورات خود سه نوع هستند:(۱) یا مسئول یک پرش شرطی[1] هستند، (۲) یا برای load و store استفاده می‌شوند (۳) یا اینکه برای انجام دادن یک عملیات با یک مقدار ثابت و ثبات هستند. فرمت این دستورات مانند شکل زیر است:

opcode	rs	rt	immediate
4 bits	3 bits	3 bits	6 bits

لیست این دستورات و opcodeهای آن در زیر آمده است:

Mnemonic	Operation	opcode
ADDi	rt ← rs + SIGN_EXTEND(immediate)	0010
SUBi	rt ← rs - SIGN_EXTEND(immediate)	0011
ANDi	rt ← rs & immediate	0100
ORi	rt ← rs | immediate	0101
SB	MEM[rs + SIGN_EXTEND(immediate)] ← rt	0110
LB	rt ← MEM[rs + SIGN_EXTEND(immediate)]	0111
BEQ	`if` (rt == rs): PC ← PC + SIGN_EXTEND(immediate << 1)	1000
BNQ	`if` (rt != rs): PC ← PC + SIGN_EXTEND(immediate << 1)	1000

در رابطه با این نوع دستورات به نکات زیر توجه کنید:

- دقت کنید که مقدار immediate در دستورات `ANDi` و `ORi` به هیچ وجه sign extend نمی‌شوند.
- اگر به یاد داشته باشید زمانی که در پردازنده میپس پرش نسبی [2] داشتیم به اندازه‌ی ۲ بیت مقدار immediate را شیفت می‌دادیم چرا که دستورات 4 byte aligned بودند. اما در اینجا از آنجا که دستورات 2 byte aligned

[1] conditional branch
[2] relative jump

هستند باید یک واحد مقدار immediate را شیفت دهید. در دستورات نیز علامت >> به معنای شیفت دادن است.

- **Jump Instructions**: این دستورات برای پرش استفاده می‌شوند. شکل این دستورات به صورت زیر است:

opcode	NOT USED	address
4 bits	5 bits	7 bits

همان طور که مشاهده می‌کنید در اینجا طول آدرس ۷ بیت است چرا که با توجه به طراحی ما و اندازه‌ی حافظه این پردازنده می‌تواند حداکثر ۱۲۸ دستور را در خود ذخیره و اجرا کند. همچنین همان طور که از اسم آن پیدا است بیت‌های NOT USED صرفا بدون استفاده هستند و به عبارتی dont care در نظر گرفته می‌شوند. این سری از دستورات شامل دو دستور زیر هستند:

Mnemonic	Operation	opcode
J	PC ← address << 1	1110
JAL	R[7] ← PC + 2, PC ← address << 1	1111

در نهایت نیز به این موضوع توجه کنید که تمامی مقادیر opcode و funct که نوشته شده‌اند پیشنهادی هستند و در صورت نیاز می‌توانید آن‌ها را تغییر دهید. اما در صورتی که آن‌ها را تغییر دهید حتما در گزارش خود ذکر کنید که دستورات شما چه opcode یا functهایی دارند.

صورت تمرین

در این تمرین عملی می‌خواهیم که پردازنده‌ی MIPS که در گذشته ساخته بودیم را به یک پردازنده‌ی multi-cycle تبدیل کنیم. در آخر این تمرین پردازنده‌ی MIPS شما باید بتواند ضرب دو عدد را به کمک الگوریتم multi-cycle booth که در تمرین‌های قبلی پیاده‌سازی کرده بودید را حساب کند.

گام‌های پیشنهادی

در این تمرین چندین روش پیاده‌سازی پردازنده‌ی multi-cycle وجود دارد. یکی از متداول ترین روش‌ها روشی است که شما سر راه تمامی قطعات اصلی از جمله حافظه و بانک ثبات و ALU، DFF قرار دهید و به صورت آبشاری داده‌ها را جلو ببرید. سپس زمانی که داده به آخر یا قسمت write back رسید، مقدار PC را بعلاوه‌ی ۲ کنید یا branch/jump را انجام دهید و در نتیجه دستور بعدی واکشی شود. همچنین یکی دیگر از کارهایی که در برخی از شرایط جواب می‌دهد (مثلا حافظه شما در یک سیکل داده را به خروجی منتقل کند) این است که مقدار PC را برای تعدادی چرخه‌های مشخص ثابت نگه دارید. به عنوان مثال در این تمرین باید حواستان باشد که در صورتی که ضرب انجام می‌دهید نباید تا زمانی که سیگنال ready یک شود instruction بعدی را واکشی کنید.

نحوه‌ی ارزیابی

در ابتدا برنامه‌هایی که در سری تمرین قبل نوشته بودید را بر روی این معماری جدید اجرا کنید. آن برنامه‌ها باید جواب‌های یکسانی با این تمرین داشته باشند.
در ادامه یک برنامه جدید بنویسید که به کمک آن فاکتوریل عدد ۵ را حساب کند و در ثبات شماره ۱ نتیجه را قرار دهد. در صورتی که multi-cycle بودن را درست پیاده‌سازی کرده باشید باید دستور ضرب شما درست انجام شود و نتیجه ۱۲۰ در ثبات اول ذخیره شود.

خروجی‌های مورد انتظار

برای جواب این تمرین در ابتدا خود فایل‌های Quartus خود را بارگذاری کنید. سپس یک گزارش تهیه کنید که در آن به صورت دقیق نوشته باشید که به چه صورت پردازنده را multi-cycle کرده‌اید و مخصوصاً توضیح دهید که دستور ضرب را چه طوری پیاده‌سازی کرده‌اید. در نهایت نیز نشان دهید که برنامه‌های شما به درستی کار می‌کنند.