به نام خدا

دانشکده مهندسی کامپیوتر

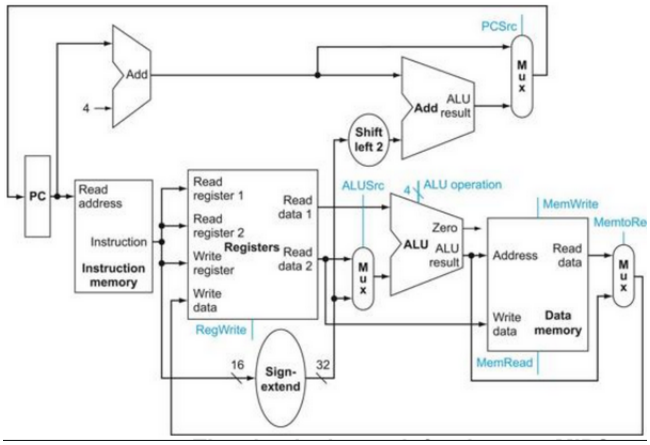
درس معماری کامپیوتر
نیم‌سال دوم ۰۲-۰۳
استاد: دکتر حسین اسدی

تمرین سری چهارم

- 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 که نشان‌دهنده‌ی مقدار سیگنال‌های کنترلی در طول اجرای دستورات است را با توجه به مسیر داده [1] معماری پردازنده MIPS که در کلاس طراحی شد(شکل زیر)، تکمیل نمایید. دقت کنید که اگر سیگنال کنترلی MUX برابر ۰ باشد ورودی پایین آن انتخاب می‌شود و در صورتی که برابر ۱ باشد ورودی بالای آن انتخاب می‌شود.



Instruction	PCSrc	ALUSrc	MemWrite	MemToReg
`mult $t0, $t1`	1	1	0	0
`bne $t0, $t1, 10`	Zero	1	0	X
`lw $t0, 0($t1)`	1	0	0	1
`sw $t0, 0($t1)`	1	0	1	X

۲. طراحی مسیر داده برای پردازنده MIPS را در نظر بگیرید(شکل سوال ۱). می‌خواهیم با ایجاد تغییراتی، دو دستور زیر را به دستورات قابل پشتیبانی توسط این مسیر داده اضافه کنیم. ابتدا با ذکر دلیل نوع دستور مناسب برای هر کدام را مشخص کنید و سپس تغییرات لازم برای پشتیبانی از هر دستور را مشخص نمایید.

۱. `getpc $rd`: این دستورالعمل مقدار PC را در ثبات `rd` می‌ریزد.

۲. `jof $rt, imm($rs)`: این دستورالعمل ابتدا مقدار PC را در ثبات `rt` ذخیره می‌کند و سپس PC را برابر `imm + $rs` قرار می‌دهد.

۳. جدول تأخیرهای زیر را برای بخش‌های مختلف پردازنده MIPS در نظر بگیرید:

I-Mem	Add	Mux	ALU	Regs	D-Mem	Control
200ps	70ps	20ps	90ps	90ps	250ps	40ps

فرض کنید ۴ درصد دستورات از نوع پرش، ۱۳ درصد Beq و ۱۴ درصد از نوع Load و ۱۷ درصد از نوع Store و بقیه از نوع R باشند. در صورتی که با تغییراتی بتوان تأخیر ALU را به ۷۵ و تأخیر D-Mem را به ۲۲۰ کاهش داد و تأخیر کنترل به ۵۰ برسد، میزان Speed Up چقدر خواهد بود؟

۴. با استفاده از دستورات پردازنده MIPS توانسته‌ایم CPU جدیدی را طراحی کنیم که به شکل single-cycle کار می‌کند و دستورات آن مطابق زیر است:

Instruction	Operation
`lw_add rd, (rs), rt`	$rd = Mem[Reg[rs]] + Reg[rt]$
`addi_st (rs), rs, imm`	$Mem[Reg[rs]] = Reg[rs] + imm$
`sll_add rd, rs, rt, imm`	$rd = (Reg[rs] \ll imm) + Reg[rt]$

[1] data path

و تمامی دستورات این پردازنده جدید ما در قالبی ۳۲ بیتی با شکل زیر در حافظه ذخیره می‌شوند:

op	rs	rt	rd	imm
31-26	25-21	20-16	15-11	10-0

حال تصور کنید تأخیر بخش‌های مختلف پردازنده جدید به شکل جدول زیر باشد:

Unit	**Latency**
Register File	2.5
Instruction Memory	4
Data Memory	6
ALU	5.5

حداقل زمان مورد نیاز برای انجام هر یک از دستورات پردازنده جدید را محاسبه کنید. (فرض کنید که می‌توان از دو ALU در طراحی استفاده کرد)

۵. مسیرداده Single-Cycle پردازنده MIPS را در نظر بگیرید. با توجه به دو دستور زیر به سوالات پاسخ دهید:

1. AND Rd,Rs,Rt
2. SW Rt, Offset(Rs)

آ) مقادیر سیگنال‌های کنترلی برای هر یک از این دو دستور را مشخص کنید.

ب) از کدام یک از منابع (بلوک‌های موجود در مدار) در این دستورات استفاده می‌شود.

ج) کدام یک از منابع، خروجی مشخصی تولید می‌کنند ولی از آن‌ها در این دستورات استفاده نمی‌شود؟

د) فرض کنید دو سیستم مختلف با تأخیرهای مشخص شده در مدار مطابق جدول زیر را داریم. مسیر بحرانی[۲] برای دستورات AND و load را مشخص کنید.

	I-Mem	Add	Mux	ALU	Regs	D-Mem	Control
a	200ps	70ps	20ps	90ps	90ps	250ps	40ps
b	750ps	200ps	50ps	250ps	300ps	500ps	300ps

۶. مسیر داده و واحد کنترل پردازنده میپس را در نظر بگیرید، فرض کنید بخواهیم دو دستور زیر را اضافه کنیم، چه تغییراتی در شماتیک باید ایجاد کنیم؟مقدار سیگنال‌های خروجی از واحد کنترلی به جز ALUop را برای هر دستور مشخص کنید.

- Addm (rd),rs,rt : mem[rd] = rs+rt
- Xormi (rt),rs,i ; mem[rt] ← rs XOR zero_extend(i)

[۲] مسیر بحرانی مسیری است که بیشترین تاخیر را ایجاد می‌کند

تمارین عملی

هدف از این تمرین، طراحی یک پردازنده‌ی شبه MIPS به کمک تمرین‌های عملی قبلی است. پردازنده‌ی MIPS که قرار است طراحی کنید ۸ ثبات ۸ بیتی دارد که مانند پردازنده MIPS ثبات شماره صفرم همیشه صفر است و ثبات آخر (ثبات شماره ۷ام) برای ذخیره‌سازی آدرس برگشتی[3] استفاده می‌شود. طول دستورات این پردازنده ۱۶ بیتی است و دو حافظه‌ی مجزا برای دستورات و داده‌ها دارد.
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 مقدار دهی 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[4] 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.

[3] return address
[4] conditional branch

- اگر به یاد داشته باشید زمانی که در پردازنده میپس پرش نسبی [5] 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 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 (که در قسمت بعد می‌بینید) 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 مقادیرopcode 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.
در این تمرین قرار است صرفاً register file و حافظه این پردازنده را طراحی کنیم و در تمرین‌های بعدی این طراحی را کامل‌تر خواهیم کرد.

۱. در ابتدا برای طراحی پردازنده میپس خود باید یک بانک ثبات طراحی کنید. لازم است که بانک ثبات شما حاوی ۸ ثبات ۸ بیتی باشد. دقیقا مانند خود پردازنده‌ی میپس باید ۵ ورودی داشته باشد:

   ۱. **Register 1**: ثبات شماره اول که خروجی آن به **RegOut 1** می‌رود.
   ۲. **Register 2**: ثبات شماره دوم که خروجی آن به **RegOut 2** می‌رود.
   ۳. **Register 3**: ثباتی که قرار است در آن عملیات write صورت بگیرد. (در صورتی که سیگنال **WriteReg** فعال باشد)
   ۴. **Data**: مقداری که می‌خواهد در **Register 3** نوشته شود.
   ۵. **WriteReg**: در صورتی که این سیگنال برابر ۱ باشد عملیات نوشتن **Data** در **RegOut 1** صورت می‌گیرد.

   همچنین مشخص است که سیگنال‌های مربوط به شماره ثبات ۳ بیتی هستند و مقدار Data ۸ بیتی است. همچنین این بانک ثبات ۲ خروجی دارد:

   ۱. **RegOut 1**: مقدار ثبات شماره **Register 1**.
   ۲. **RegOut 2**: مقدار ثبات شماره **Register 2**.

   یک نکته‌ی خیلی مهم که باید به آن توجه کنید این است که خواندن در کلاک بالارونده انجام می‌شود ولی نوشتن در کلاک پایین‌رونده انجام می‌شود. در صورتی که این موضوع را رعایت نکنید ممکن است با مسائلی روبه‌رو شوید. این مسائل از مشکلات مربوط به race که در درس مدار منطقی با آن آشنا شدید سرچشمه می‌گیرد.

۲. در ادامه می‌خواهیم حافظه‌های مربوط به داده و دستورات را طراحی کنیم. دقت کنید که عملاً این دو حافظه از یک جنس هستند و صرفا لازم است که دو نمونه از یک حافظه یکی برای داده و یکی برای دستورات طراحی کنید. طراحی این حافظه باید ویژگی‌های زیر را داشته باشد:

[5] relative jump

۱. باید ۸ بیت را به عنوان آدرس ورودی بگیرد و با کلاک بالارونده ۱۶ بیت را به خروجی منتقل کند.
۲. باید ۸ بیت را به عنوان آدرس ورودی بگیرد و با کلاک پایین‌رونده ۸ بیت را در صورت فعال بودن سیگنال WriteMem در حافظه بنویسد.

دقت کنید که خروجی باید ۱۶ بیتی باشد چرا که برای حافظه دستورات ۱۶ بیتی هستند. اما زمانی که این حافظه را برای داده استفاده می‌کنید می‌توانید از ۸ بیت پرارزش صرف نظر کنید. در این قسمت مجاز به از استفاده RAM و ROMهای آماده‌ی Quartus هستید.