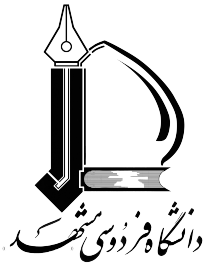


گروه مهندسی کامپیوتر

استاد درس: سرکار خانم دکتر ارشادی نسب

بهار ۱۴۰۲

پروژه
درس معماری کامپیوتر

مهلت تحویل تا: ۱۴۰۲/۰۴/۰۱ **طراحی پردازنده FUM-MIPS**

- پروژه به صورت گروهی است. (گروه های دو نفره)
- در سامانه ویو فایل‌های ویدئویی و متنی برای یادآوری و آموزش کار با verilog قرار گرفته است.
- محتویات پروژه را به صورت یک فایل با فرمت : FirstnameLastname_StudentNumber_FirstnameLastname_StudentNumber.zip بارگذاری کنید.
(مثال MohammadMohammadi_۹XXXXXX_RezaRezaei_۹XXXXXX.zip)
- کامنت گذاری تمام خطوط کد الزامی است.
- تحویل پروژه بعد از مهلت مشخص شده نمره ای نخواهد داشت.
- در صورت اثبات کپی برداری، نمره پروژه کپی شده و کپی شونده هر دو از ۱۰۰ نمره، ۱۰۰- خواهد بود.
- زمانبندی تحویل آنلاین پروژه پس از اتمام مهلت ارسال اعلام خواهد شد.
- تحویل پروژه از طریق تلگرام، ایمیل و ... امکان پذیر نیست.

هدف

در این آزمایش، به طراحی و مدل‌سازی یک پردازنده تک سیکل با زبان verilog می پردازیم. نام این پردازنده FUM-MIPS می باشد.
FUM-MIPS یک پردازنده ۱۶ بیتی بوده که در این پردازنده بانک ثبات شامل ۸ ثبات ۱۶ بیتی می باشد. معماری مجموعه دستورات (ISA)[۱] این پردازنده شبیه به ISA پردازنده MIPS می باشد ولی تفاوت هایی نسبت به آن دارد. طول دستورات در این پردازنده ۱۶ بیت می‌باشد. یعنی شمارنده برنامه (PC)[۲] به جای اینکه با ۴ جمع شود باید با ۲ جمع شود. PC یک ثبات ۱۶ بیتی می‌باشد. در این پردازنده سه نوع ماشین کد وجود دارد که عبارتنداز: R-Type، I-Type و J-Type .

معرفی FUM-MIPS

۱ دستورات R-Type

دستورات R-Type مانند دستورات R-Type در MIPS می‌باشد. دو source و یک مقصد دارند. ماشین کد دستورات R-Type به شکل زیر می‌باشد.
چهار بیت اول (بیت‌های ۱۲ الی ۱۵) opcode را مشخص می‌کند که opcode برای دستورات R-Type صفر ("۰۰۰۰") می‌باشد.
سه بیت بعد (بیت‌های ۹ الی ۱۱) شماره ثبات برای مبدأ اول را مشخص می‌کند. سه بیت بعد (بیت‌های ۶ الی ۸) شماره ثبات برای مبدأ دوم را مشخص می‌کند. سه بیت بعد (بیت‌های ۳ الی ۵) شماره ثبات مقصد را مشخص می‌کند و سه بیت آخر (بیت‌های صفر الی ۲) نوع عملیات را مشخص می‌کند.
مقادیر function برای دستورات add، sub، and، or، xor، nor و slt در جدول ۲ آمده است.

۱۵-۱۲	۱۱-۹	۸-۶	۵-۳	۲-۰
۴-bit	۳-bit	۳-bit	۳-bit	۳-bit
opcode	source۱	source۲	destination	function

جدول ۱: قالب دستورات R-Type

operation	function
add	۰۰۰
sub	۰۰۱
and	۰۱۰
or	۰۱۱
xor	۱۰۰
nor	۱۰۱
slt	۱۱۰

جدول ۲: مقادیر opcode و function برای دستورات R-Type

[۱] Instruction Set Architecture
[۲] Program Counter

assembly:
add reg۱, reg۲, reg۳
semantics:
GPR[reg۱] <– GPR[reg۲] + GPR [reg۳]
PC <– PC + ۲

۲ دستورات I-Type

دستورات I-Type مانند دستورات I-Type در MIPS می‌باشد با این تفاوت که مقدار immediate شش بیتی می‌باشد. کد ماشین دستورات I-Type در جدول ۳ آورده شده است.
چهار بیت اول (بیت‌های ۱۲ الی ۱۵) opcode را مشخص می‌کند. سه بیت بعد (بیت‌های ۹ الی ۱۱) شماره ثبات را مشخص می‌کند. سه بیت بعد (بیت‌های ۶ الی ۸) شماره ثبات دوم را مشخص می‌کند. شش بیت آخر (بیت‌های صفر الی ۵) مقدار immediate را مشخص می‌کند.
برای دستورات addi، subi، ori و andi چهار بیت اول (بیت‌های ۱۲ الی ۱۵) opcode را مشخص می‌کند. سه بیت بعد (بیت‌های ۹ الی ۱۱) شماره ثبات مبدأ را مشخص می‌کند. سه بیت بعد (بیت‌های ۶ الی ۸) شماره ثبات مقصد را مشخص می‌کند. شش بیت آخر (بیت‌های صفر الی ۵) مقدار immediate را مشخص می‌کند. این مقدار ۶ بیتی برای دستورات addi و subi توسعه علامت (sign extend) می‌شود و برای دستورات ori و andi عمل توسعه با صفر (zero extend) صورت می‌گیرد و به یک مقدار ۱۶ بیتی تبدیل شده و بعد با محتوای ثبات عملیات انجام می‌شود. مقادیر opcode برای دستورات addi، subi، andi و ori در جدول ۴ آمده است.

۱۵-۱۲	۱۱-۹	۸-۶	۵-۰
۴-bit	۳-bit	۳-bit	۶-bit
opcode	register۱	register۲	Immediate

جدول ۳: قالب دستورات I-Type

operation	opcode
addi	۰۰۰۱
andi	۰۰۱۰
ori	۰۰۱۱
subi	۰۱۰۰

جدول ۴: مقادیر opcode برای دستورات addi، subi، andi و ori

assembly:
addi Reg۱, Reg۲, Imm
semantics:
GPR[Reg۱] <– GPR[Reg۲] + sign-extend(Imm)
PC <– PC + ۲

assembly:

andi Reg۱, Reg۲, Imm
semantics:
GPR[Reg۱] <– GPR[Reg۲] + zero-extend(Imm)
PC <– PC + ۲

برای این پردازنده دستورات load (lhw) و store (shw) جزء قالب I-Type حساب می‌شوند. دستورات lhw ، shw به ترتیب یک داده ۱۶ بیتی را از حافظه می‌خواند و یک داده ۱۶ بیتی را در حافظه می‌نویسد.
در این پردازنده مانند MIPS از آدرس دهی displacement (محتوای ثبات + offset) برای محاسبه آدرس استفاده می‌شود.
برای دستورات lhw چهار بیت اول (بیت‌های ۱۲ الی ۱۵) opcode را مشخص می‌کند. سه بیت بعد (بیت‌های ۹ الی ۱۱) شماره ثبات پایه را مشخص می‌کند. سه بیت بعد (بیت‌های ۶ الی ۸) شماره ثبات مقصد را مشخص می‌کند. شش بیت آخر (بیت‌های صفر الی ۵) مقدار offset را مشخص می‌کند که باید sign extend شود و بعد با ثبات پایه جمع شود و آدرس را تولید کند.
برای دستورات shw چهار بیت اول (بیت‌های ۱۲ الی ۱۵) opcode را مشخص می‌کند. سه بیت بعد (بیت‌های ۹ الی ۱۱) شماره ثبات پایه را مشخص می‌کند. سه بیت بعد (بیت‌های ۶ الی ۸) شماره ثبات مبدأ را مشخص می‌کند. شش بیت آخر (بیت‌های صفر الی ۵) مقدار offset را مشخص می‌کند که باید sign extend شود و بعد با ثبات پایه جمع شود و آدرس را تولید کند. مقدار offset یک مقدار علامت‌دار[۳] می‌باشد.
مقادیر opcode برای دستورات lhw و shw ، در جدول ۵ آمده است.

operation	opcode
lhw	۰۱۱۱
shw	۱۰۰۰

جدول ۵: مقادیر opcode برای دستورات lhw و shw

assembly:
lhw reg۱, Imm(Reg۲)
semantics:
EA <– GPR[Reg۲] + sign-extend(imm)
GPR[Reg۱] <– Memory[EA]
PC <– PC + ۲

assembly:
shw reg۱, Imm(Reg۲)
semantics:
EA <– GPR[Reg۲] + sign-extend(imm)
Memory[EA] <– GPR[Reg۱]
PC <– PC + ۲

برای این پردازنده دستورات پرش (Branch) جز قالب I-Type حساب می‌شوند. دستورات پرش محتوای دو ثبات را با یکدیگر مقایسه می‌کنند و براساس نوع مقایسه، اگر نتیجه درست باشد پرش انجام می‌شود و در غیر اینصورت انجام نمی‌شود. آدرس پرش به اینصورت محاسبه می‌شود که مقدار شش بیتی immediate به

[۳] signed

مقدار ۱۶ بیتی sign extend می‌شود سپس در دو ضرب می‌گردد و بعد با PC+۲ جمع می‌شود. نتیجه این جمع آدرس محل پرش را مشخص می‌کند:

$$(PC + 2 + (signextend(imm) \times 2))$$

جدول ۶ انواع دستورات پرش و opcode آنها را نشان می‌دهد.

opcode	instruction	operation
۱۰۰۱	beq	Branch if equal
۱۰۱۰	bne	Branch if not equal
۱۰۱۱	blt	Branch if less than
۱۱۰۰	bgt	Branch if greater than

جدول ۶: مقادیر opcode برای دستورات پرش

```
assembly:
    beq reg۱, reg۲, imm
semantics:
    if GPR[reg۱] == GPR[reg۲] then
        PC <- PC + ۲ + (sign-extend(imm)*۲)
    else
        PC <- PC + ۲
```

۳ دستورات J-Type

دستور jmp از نوع J-Type می‌باشد. ماشین کد دستورات J-Type در جدول ۷ آورده شده است. در این دستور ۱۲ بیت پایین دستور (بیت‌های صفر الی ۱۱) در ۲ ضرب می‌شود (تبدیل به یک مقدار ۱۳ بیتی می‌شود) و سپس سه بیت بالای PC به ابتدای آن اضافه می‌شود تا یک مقدار ۱۶ بیتی ایجاد گردد و سپس در PC نوشته می‌شود. opcode دستور jmp برابر "۱۱۱۱" می‌باشد.

```
assembly:
    jmp label
semantics:
    PC <- PC[۱۵:۱۳] && (instr[۱۱:۰]) && "۰"
```

۱۵-۱۲	۱۱-۰
۴-bit	۱۲-bit
Opcode (1111)	jump address

جدول ۷: قالب دستورات J-Type

۴ ماژول ها

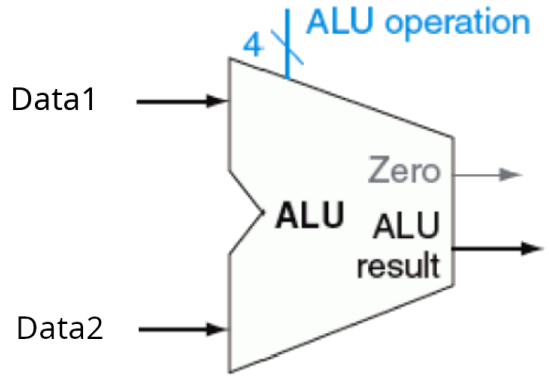
۱.۴ Register file

شکل ۱ بلوک دیاگرام این ماژول را نشان می‌دهد. ورودی های read register 1، read register 2 و write register سه بیتی می‌باشند که آدرس‌های ثبات‌های خوانده شده و نوشته شده را مشخص می‌کنند. ورودی regwrite هم به عنوان سیگنال فعالساز (enable) برای بانک ثبات عمل می‌کند. این بانک ثبات یک ورودی کلاک هم دارد که با لبه بالارونده عمل می‌کند. هرگاه سیگنال regwrite در لبه بالارونده کلاک یک باشد عمل نوشتن در ثبات انجام می‌شود. ورودی write data باید یک ورودی ۱۶ بیتی باشد که داده‌ای را که باید در ثبات نوشته شود مشخص می‌کند. خروجی‌های read data 1 و read data 2 خروجی‌های ۱۶ بیتی هستند که مقادیر ثبات‌های read register 1 و read register 2 را شامل می‌شوند.

شکل ۱: بانک ثبات

۲.۴ ALU

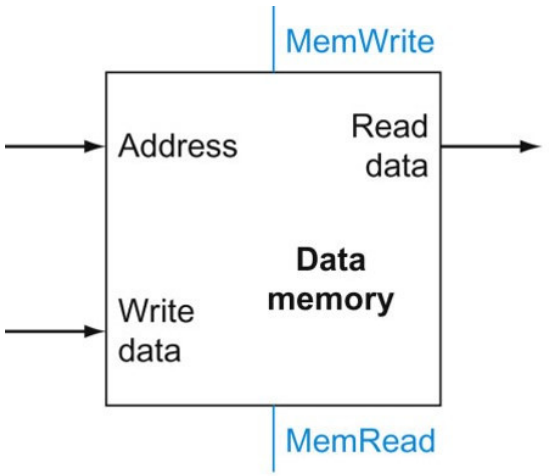
شکل ۲ بلوک دیاگرام این ماژول را نشان می‌دهد. دو ورودی ALU باید ۱۶ بیتی و خروجی ALU result نیز باید ۱۶ بیتی باشند. ورودی چهاربیتی ALU opration مشخص می‌کند که ALU چه عملی انجام دهد. خروجی یک بیتی zero اگر یک باشد نشان می‌دهد که مقدار ALU result صفر می‌باشد. ALU شما بجز خروجی zero باید شامل دو خروجی تک بیتی lt (less than) و gt (greater than) نیز باشد. اگر data۱ از data۲ بزرگتر باشد خروجی gt باید یک شود و اگر data۱ از data۲ کوچکتر باشد خروجی lt باید یک شود.



شکل ۲: ALU

۳.۴ حافظه داده

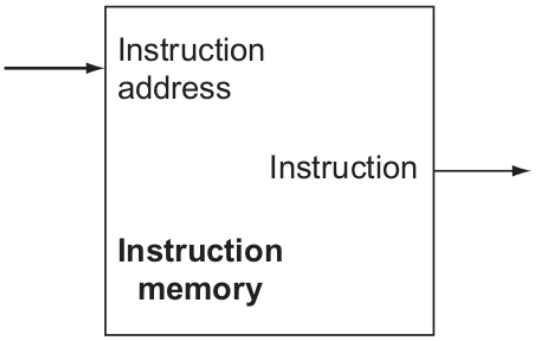
شکل ۳ بلوک دیاگرام این ماژول را نشان می‌دهد. ورودی address و write data و خروجی read data مقادیر ۱۶ بیتی می‌باشند. دو سیگنال ورودی کنترل MemRead و MemWrite یک بیتی می‌باشند که اگر یک باشند به ترتیب عمل خواندن از حافظه و نوشتن در حافظه انجام می‌شود.



شکل ۳: حافظه داده

۴.۴ حافظه برنامه

شکل ۴ بلوک دیاگرام این ماژول را نشان می‌دهد. ورودی address و خروجی instruction مقادیر ۱۶ بیتی می‌باشند. برای این پروژه فرض کنید که حافظه حداکثر ۲۵۶ خانه ۱۶ بیتی دارد.



شکل ۴: حافظه برنامه

۵ پیاده‌سازی

دقت داشته باشید که این دو فاز به صورت مجزا در ویو تحویل داده می شوند. بنابراین هر فاز باید به طور مجزا قابل اجرا باشد.

۱.۵ فاز اول: پردازنده تک‌سیکل (Single Cycle)

در فاز اول دانشجویان باید پردازنده تک‌سیکل را در دو مرحله پیاده‌سازی کنند.

۱.۱.۵ مرحله اول

در مرحله اول انجام آزمایش دانشجویان باید با استفاده از ماژول‌های ذکر شده و اضافه کردن واحد کنترل، PC و ماژول‌های مورد نیاز، طراحی کلی پردازنده را نهایی کنند (پردازنده‌ای مشابه شکل ۵) تا بتوانند دستورات ذکر شده را اجرا کنند.
دانشجویان باید در یک محیط شبیه‌سازی نشان دهند که پردازنده طراحی شده به درستی تمام دستورات را اجرا می‌کند.

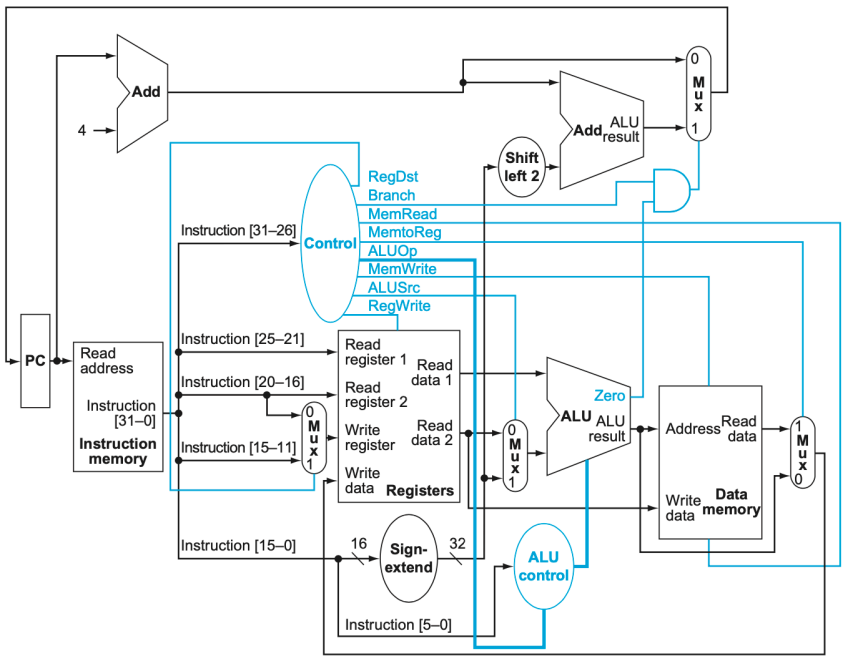
۲.۱.۵ مرحله دوم

در این مرحله دانشجویان باید برنامه ای بنویسند که بتواند در یک آرایه با ۹ عنصر که شامل داده‌های ۱۶ بیتی می‌باشد، بزرگترین و کوچکترین و میانه اعداد را بدست آورد. نمایش صحت درستی اجرای برنامه بر روی یک محیط شبیه سازی (مانند Modelsim و یا Verilator) باید نمایش داده شود.

۲.۵ فاز دوم: پردازنده پایپلاین (Pipeline)

در این فاز باید پروژه فاز قبل را پایپلاین کنید. همچنین باید هازارد های داده را رفع کنید. رفع هازارد های کنترلی ممکن است به نمره اضافه منجر شود.

توجه:
برای انجام فاز دوم، بعد از انجام فاز اول و بارگذاری آن در ویو، فاز اول را کپی کرده و به عنوان یک پروژه جدید باز کنید و سپس تغییرات لازم برای تبدیل فاز اول به فاز دوم را اعمال کنید.



شکل ۵: نمای کلی پردازنده