طراحی پردازنده تک سیکل (Single Cycle)

هدف:

در این آزمایش، به طراحی و مدل‌سازی یک پردازنده تک سیکل با زبان Verilog می‌پردازیم. نام این پردازنده FUM-MIPS می‌باشد.

FUM-MIPS یک پردازنده ۱۶ بیتی بوده که در این پردازنده بانک ثبات شامل ۸ ثبات ۱۶ بیتی می‌باشد. معماری مجموعه دستورات[۱] (ISA) این پردازنده شبیه به ISA پردازنده MIPS می‌باشد ولی تفاوت‌هایی نسبت به آن دارد. طول دستورات در این پردازنده ۱۶ بیت می‌باشد. یعنی شمارنده برنامه[۲] (PC) 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 عبارتند از: 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 ۳ 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 یک آمده است.

۱۵-۱۲	۱۱-۹	۸-۶	۵-۳	۲-۰
۴-bits..	۳-bit	۳-bit	۳-bit	**۳-bit**
opcode	source ۱	source ۲	destination	**function**

شکل ۱- قالب دستورات R-Type

function	operation
۰۰۰	**add**
۰۰۱	**sub**
۰۱۰	**And**
۰۱۱	**Or**
۱۰۰	**Xor**
۱۰۱	**Nor**
۱۱۰	**Slt**

جدول ۱- 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- د ستورات I-Type: د ستورات I-Type مانند د ستورات I-Type در MIPS می‌با شد با این تفاوت که مقدار immediate ٦ بیتی می‌باشد. کد ماشین دستورات I-Type در شکل ۲ 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. ۶ 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 م شخص می‌کند. سه بیت بعد (بیت‌های ۹ الی ۱۱) شماره ثبات مبدأ را م شخص می‌کند. سه بیت بعد (بیت‌های ۶ الی ۸) شماره ثبات مقصد را م شخص می‌کند. ۶ بیت آخر (بیت‌های صفر الی ۵) مقدار immediate را م شخص می‌کند. این مقدار ۶ بیتی برای د ستورات addi و subi تو سعه علامت (sign extend) می شود و برای د ستورات ori و andi عمل تو سعه 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 ۲ آمده است.

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

شکل ۲- قالب دستورات I-Type

opcode	Operation
۰۰۰۱	**addi**
۰۰۱۰	**andi**
۰۰۱۱	**ori**
۰۱۰۰	**subi**

جدول ۲- 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 و shw به ترتیب یک داده ۱۶ بیتی را از حافظه می‌خواند و یک داده ۱۶ بیتی را در حافظه می نویسد. در این پردازنده مانند MIPS از آدرس‌دهی

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. ۶ 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. ۶ 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 علامت دار[۳] می‌باشد. مقادیر opcode برای دستورات lhw و shw ، در جدول ۳ آمده است.

opcode	Operation
۰۱۱۱	**Lhw**
۱۰۰۰	**Shw**

جدول ۳- 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) جزء قالب I-Type حساب می‌شوند. دستورات پرش محتوای دو ثبات را با یکدیگر مقایسه می کنند و بر اساس نوع مقایسه، اگر نتیجه درست باشد پرش انجام می‌شود و در غیر اینصورت انجام نمی‌شود. آدرس پرش به اینصورت محاسبه می‌شود که مقدار ۶ بیتی immediate به مقدار ۱۶ بیتی sign extend می‌شود سپس در دو ضرب می گردد و بعد با PC+۲ جمع می‌شود. نتیجه این جمع آدرس محل پرش را مشخص می‌کند:

(PC+۲+(sign extend (imm)*۲))

جدول ۴ انواع دستورات پرش و opcode آنها را نشان می دهد.

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 ۴- مقادیر opcode برای دستورات پرش

assembly: beq reg۱, reg۲ , imm

[۳] signed

```
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 شکل ۳ آورده شده است. در این دستور ۱۲ بیت پایین دستور (بیت‌های صفر الی ۱۱) در ۲ ضرب می شود (تبدیل به یک مقدار ۱۳ بیتی می شود) و سپس سه بیت بالای PC به ابتدای آن اضافه می شود تا یک مقدار ۱۶ بیتی ایجاد گردد و سپس در PC نوشته می شود. Opcode دستور jmp برابر "۱۱۱۱" می‌باشد.

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

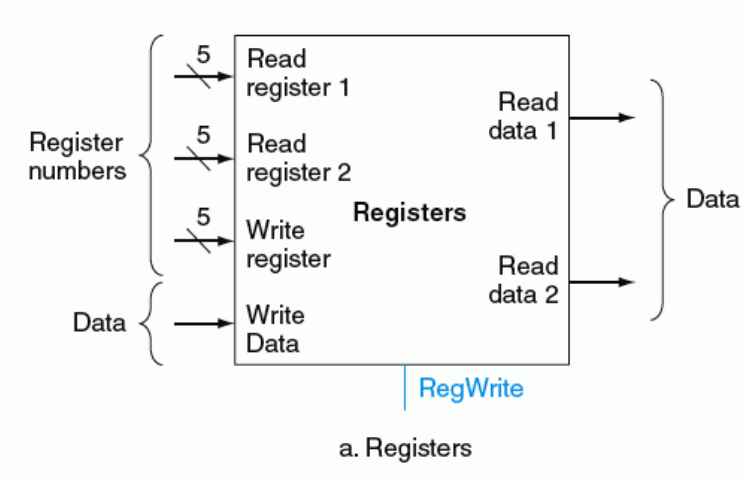
۱۵-۱۲	۱۱-۰
۴-bits	**۱۲-bit**
Opcode (۱۱۱۱)	**jump address**

شکل ۳- قالب دستورات J-Type

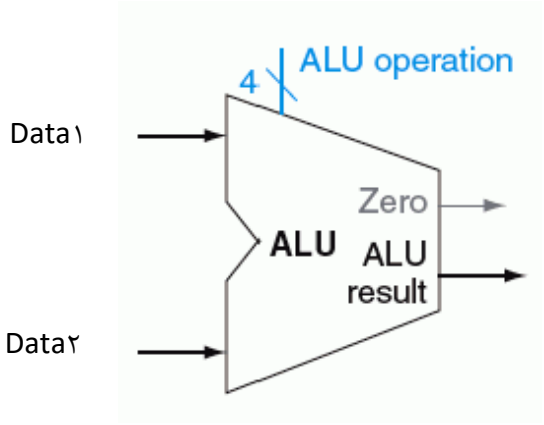
۱- Register file: شکل۴ بلوک دیاگرام این ماژول را نشان می دهد. ورودی‌های read register۱، read register۲ و write register سه بیتی می‌باشند که آدرس‌های ثبات‌های خوانده شده و نوشته شده را مشخص می‌کنند. ورودی regwrite به عنوان سیگنال فعالساز (enable) برای بانک ثبات عمل می‌کند. این بانک ثبات یک ورودی کلاک هم دارد که با لبه بالارونده کلاک 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 data۱ و read data۲ خروجی‌های ۱۶ بیتی هستند که مقادیر ثبات‌های read register۱ و read register۲ را شامل می‌شوند.

۲- ALU: شکل ۵ بلوک دیاگرام این ماژول را نشان می دهد. دو ورودی ALU باید ۱۶ بیتی و خروجی ALU result نیز باید ۱۶ بیتی باشند. ورودی چهاربیتی ALU operation 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 باید 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.

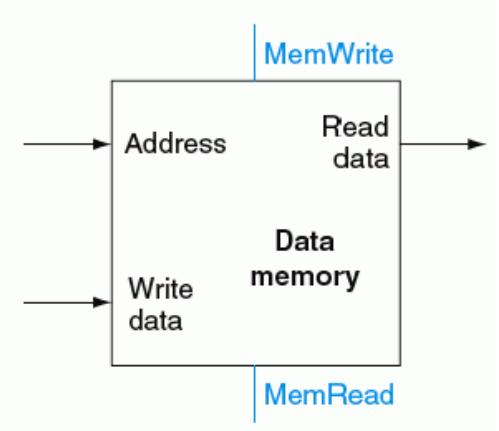
۳- حافظه داده: شکل ۶ بلوک دیاگرام این ماژول را نشان می دهد. ورودی address و write data و خروجی read data مقادیر ۱۶ بیتی می‌باشند. دو سیگنال ورودی کنترل MemRead و MemWrite یک بیتی می‌باشند که اگر یک باشند به ترتیب عمل خواندن از حافظه و نوشتن در حافظه انجام می‌شود.



شکل ۴- بانک ثبات

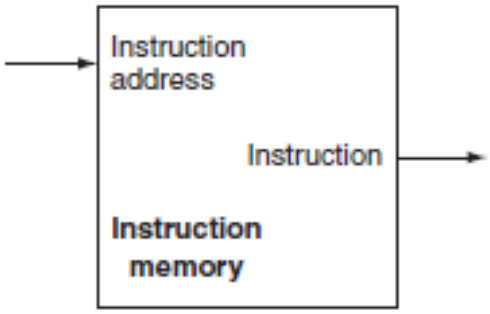


شکل ۵- ALU



شکل۶- حافظه داده

۴- حافظه برنامه: شکل ۷ بلوک دیاگرام این ماژول را نشان می دهد. ورودی address و خروجی instruction مقادیر ۱۶ بیتی می‌باشند. برای این پروژه فرض کنید که حافظه حداکثر ۲۵۶ خانه ۱۶ بیتی دارد .

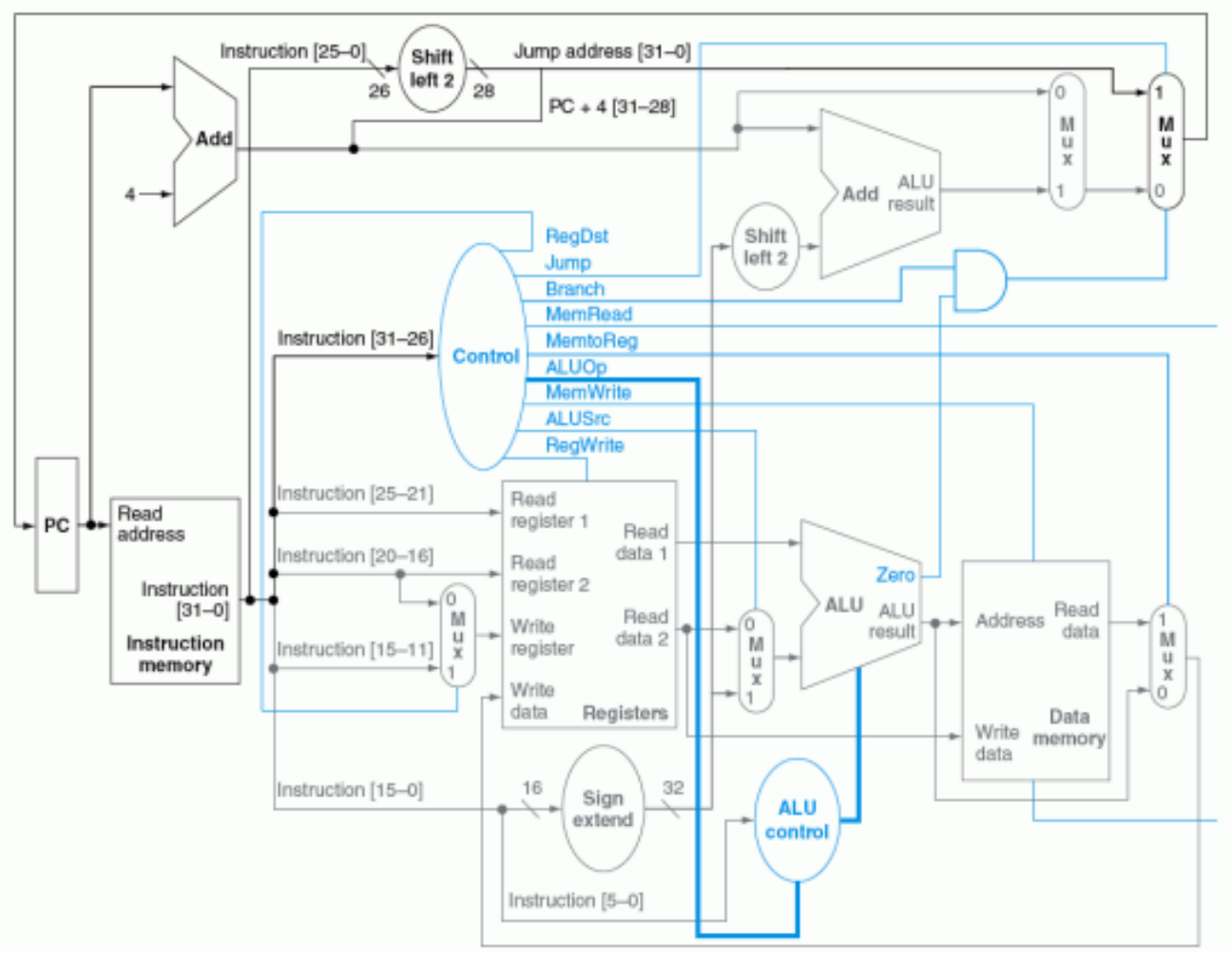


شکل ۷- حافظه برنامه

دانشجویان باید پردازنده تک‌سیکل را در دو مرحله پیاده‌سازی کنند.

مرحله اول: در مرحله اول انجام آزمایش دانشجویان باید با استفاده از ماژول‌های قسمت اول و اضافه کردن واحد کنترل، PC و ماژول‌های مورد نیاز، طراحی کل پردازنده را نهایی کنند (پردازنده ای مشابه شکل هشت) 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.

مرحله دوم: در این مرحله دانشجویان باید برنامه‌ای بنویسند که بتواند در یک آرایه با ۹ عنصر که شامل داده‌های ۱۶ بیتی می‌باشد، بزرگترین و کوچکترین و مدیان اعداد را بدست آورد. نمایش صحت درستی اجرای برنامه بر روی یک محیط شبیه‌سازی (مانند Modelsim) باید نمایش داده شود.



شکل ۸- نمای کلی پردازنده