به نام خدا

پروژه ۴ معماری کامپیوتر _ نیمسال دوم ۱۴۰۳_۱۴۰۴

حامد گلکار ۸۱۰۱۰۱۴۹۹ / پریسا محمدی ۸۱۰۱۰۱۵۰۹

گزارش طراحی RISC-V

در این پروژه، ما باید یک نسخه ساده‌شده از پردازنده RISC-V را با استفاده از رویکرد چند چرخه‌ای طراحی کنیم .این یعنی دستورالعمل‌های مختلف، تعداد متفاوتی چرخه ساعت برای اجرا نیاز دارند، اما فرکانس ساعت می‌تواند بیشتر باشد .
لیست دستوراتی که باید پیاده‌سازی شوند شامل موارد زیر است:

نوع R: دستورهای add، sub، and، or، slt

نوع I: دستورهای lw، addi، xori، ori، slti، jalr

نوع S: دستور sw

نوع J: دستور jal

نوع B: دستورهای beq، bne

نوع U: دستور lui

هرکدام از این دستورات فرمت مخصوص خود را دارند، بنابراین مسیر داده باید به‌گونه‌ای طراحی شود که داده‌ها از مسیر مناسب عبور کنند.پس از طراحی پردازنده، آن را با اجرای برنامه‌ای که مقدار کمینه را از آرایه‌ای با ۱۰ عضو پیدا می‌کند، تست خواهیم کرد.

فرمت دستورالعمل‌ها

دستورات نوع R

فرمت دستور :خوشبختانه، تمامی دستورهای نوع R که باید پیاده‌سازی شوند، به‌خوبی تعریف شده‌اند و تقریباً مشابه یکدیگر هستند .تنها تفاوت آن‌ها عملکرد ALU نسبت به ورودی‌هایش است .فرمت کلی آن‌ها به شکل زیر است:

دستورات نوعI

فرمت دستور :در این پروژه، دستورات Immediate به سه دسته تقسیم می‌شوند:

-نوع محاسباتی شاملslti، xori، ori، addi

-نوع بارگذاری شاملlw ، نوع پرش شاملjalr

دستورات نوعS

این پروژه فقط شامل یک دستور نوع S یعنی sw می‌شود.

دستورات نوعJ

تنها دستور نوع J مورد نیاز jal است.

در این فرمت، قسمت immediate به‌صورت غیرمعمولی طراحی شده و شامل بیت‌های 1 تا) 20 و نه 0 تا (19 است، چرا که مقدار باید ضریبی از ۴ باشد و دو بیت انتهایی آن صفر باشد.

اگر مقصد rd مشخص نشود، مقدار پیش‌فرض ra خواهد بود؛ البته این موضوع خارج از حیطه این تمرین است چون اسمبلی‌ساز پیاده‌سازی نشده است.

دستورات نوعB

دو دستور شاخه‌ای beq و bne باید پیاده‌سازی شوند .فرمت immediate در این نوع نیز به‌صورت پراکنده و غیرمتوالی ذخیره می‌شود که فرمول خاصی دارد.

دستورات نوعU

تنها دستور نوع U که باید پیاده‌سازی شود، lui است.

با در اختیار داشتن جداول بالا، می‌توان طراحی مسیر داده را شروع کرد.

طراحی مسیر داده

نخست، به طراحی تک‌چرخه‌ای برمی‌گردیم که مشابه دیاگرام داده‌ای ارائه‌شده است با این تفاوت که یک مالتی‌پلکسر (MUX)اضافه شده و ALU نیز شامل عملکرد "PASS" است که ورودی دوم را بدون تغییر به خروجی می‌فرستد .این عملکرد اجرای دستور JALR را ساده کرده و باعث می‌شود دستور LUI به یک دستور نوع I ساده تبدیل شود.

اصطلاحات پایپ‌لاین

پایپ‌لاین‌کردن پردازنده یعنی طراحی آن به‌گونه‌ای که اجرای چندین دستور هم‌زمان انجام شود .اما چون سخت‌افزار ما محدود است، باید مراحل اجرای یک دستور را تفکیک کنیم و هر مرحله را با سخت‌افزار مستقل انجام دهیم.

مراحل طراحی‌شده شامل:

1. (FETCH): دریافت دستور از حافظه دستور
2. (DECODE): استخراج عملگر و عملوندها
3. (EXECUTE): انجام عملیات لازم
4. (MEMORY-ACCESS): خواندن/نوشتن از/در حافظه
5. (WRITE-BACK): نوشتن نتیجه در فایل ثبات‌ها

هر مرحله به‌صورت موازی روی دستور متفاوت انجام می‌شود که موجب استفاده بهینه از سخت‌افزار می‌شود.

پیاده‌سازی

برای این منظور، اجرای دستورالعمل‌ها را به مراحل مستقل تقسیم کرده و خروجی هر مرحله را با رجیستر ذخیره کرده و به مرحله بعد منتقل می‌کنیم .همین کار را برای سیگنال‌های کنترلی نیز انجام می‌دهیم که از مرحله Decode تولید و از طریق رجیسترها به مراحل بعدی منتقل می‌شوند.

ایده اصلی این طراحی استفاده از اجزای سخت‌افزاری جداگانه برای هر مرحله است .بنابراین، از طراحی تک‌چرخه‌ای آغاز می‌کنیم و اجزایی مانند جمع‌کننده‌ها و حافظه‌ها که در طراحی چندچرخه حذف شده بودند، دوباره استفاده می‌شوند.

طراحی پایپ‌لاین

طراحی به ۵ مرحله تقسیم می‌شود: FETCH، DECODE، EXECUTE، MEM-ACCESS، WRITE-BACK

مراحل توسط رجیسترها جدا شده‌اند و سیگنال‌های کنترلی نیز همراه داده‌ها منتقل می‌شوند.

همچنین، برای جلوگیری از خطرات، واحدی به نام Hazard Unit طراحی شده که با استفاده از فلش، stall و انتقال داده از مراحل جلوتر، از بروز مشکل جلوگیری می‌کند.

فایل رجیستر

فایل رجیستر باید از خواندن ناهم‌زمان و نوشتن در لبه منفی پشتیبانی کند تا بتواند خطرات را مدیریت کند .بنابراین، توصیف آن به‌روزرسانی شده است تا این ویژگی‌ها را پشتیبانی کند.

شمارنده برنامه (PC)

در طراحی پایپ‌لاین، شمارنده برنامه باید بتواند در شرایط خاص متوقف (stall) شود .بنابراین، ورودی جدیدی به نام "Stall"به آن افزوده شده است.

رجیستر

رجیسترها باید از قابلیت توقف و تخلیه هم‌زمان پشتیبانی کنند .این کار از طریق توصیف جدید در Verilog انجام شده است.

طراحی مراحل مسیر داده به‌صورت جداگانه

برای توصیف مسیر داده درVerilog ، آن را به ماژول‌های جداگانه‌ای تقسیم کردیم که هر کدام یک مرحله را نشان می‌دهد) واکشی، رمزگشایی، اجرا، دسترسی به حافظه، بازنویسی .(پس از آماده شدن همه مراحل، آن‌ها را به هم متصل کرده و سیم‌کشی‌های لازم و رجیسترهای جداکننده را اعمال کردیم.

کد Verilog برای مسیر داده بسیار طولانی بود) حدود ۲۳۳ خط (و فقط بخش‌هایی از آن شامل مرحله واکشی و رمزگشایی و رجیسترهای بین آن‌ها را درج کردیم .پس از تکمیل مسیر داده، به طراحی واحد کنترل پرداختیم.

طراحی واحد کنترل

واحد کنترل به سه قسمت اصلی تقسیم شده است:

-کنترلر :ALU تعیین عملکرد ALU بر اساس ورودی از کنترلر اصلی

-کنترلر :PC تصمیم‌گیری در مورد به‌روزرسانی شمارنده برنامه

-کنترلر اصلی :تولید سیگنال‌های کنترلی مانندRegWrite ، ALUSrc و غیره

کنترلرALU

عملیات ALU گاهی مستقل از فیلدهای f3 و f7 است) مثلاً در دستورات شاخه‌ای که فقط نیاز به مقایسه صفر دارند (و گاهی وابسته به آن‌ها .کنترلر ALU تصمیم می‌گیرد چه زمانی عملیات ثابت باشد و چه زمانی بر اساس f3 و f7 انتخاب شود.

کنترلرPC

به‌روزرسانی شمارنده برنامه در سه حالت انجام می‌شود:

-رفتن به دستور بعدی(PC+4)

-پرش به آدرس مشخص‌شده

-اجرای دستور شاخه و بررسی شرط

کنترلر اصلی

کنترلر اصلی سیگنال‌های کنترلی لازم را تولید می‌کند که از طریق رجیسترها به مراحل مختلف پایپ‌لاین منتقل می‌شوند .این واحد مانند طراحی تک‌چرخه‌ای عمل می‌کند ولی در سطح پایپ‌لاین گسترش یافته است.

واحد خطرات(Hazard Unit)

شناخت خطرات

در این طراحی دو نوع خطر وجود دارد :خطر داده (Data Hazard) و خطر کنترل.(Control Hazard)

خطر داده زمانی رخ می‌دهد که یک دستور نیاز به استفاده از مقدار یک رجیستر دارد، در حالی‌که دستور قبلی هنوز آن رجیستر را به‌روزرسانی نکرده است .برای مثال:

```
add x1, x2, x3
or  x5, x1, x7
```

در این مثال، دستور دوم به مقدار به‌روز x1 نیاز دارد، در حالی که دستور اول هنوز آن را در مرحله Write-Back نهایی نکرده است.

برای حل این مشکل از **(Forwarding)** استفاده می‌کنیم .داده‌ها از مراحل جلوتر (Memory یا Write-Back)به ورودی ALU منتقل می‌شوند.

در مورد دستور `lw` که داده را از حافظه می‌خواند، چون مقدار تا مرحله دسترسی به حافظه آماده نمی‌شود، لازم است یک سیکل پایپ‌لاین متوقف شود (stall) و همچنین مرحله بین Decode و Execute flush شود تا از اجرای تکراری جلوگیری شود.

خطر کنترل زمانی اتفاق می‌افتد که دستور شاخه داریم .چون شرط شاخه در مرحله اجرا بررسی می‌شود، اما PC در مرحله Fetch به‌روزرسانی می‌شود .راه‌حل این است که فرض کنیم شاخه اجرا نمی‌شود و در صورت غلط بودن این فرض، دو دستور اشتباه flush شوند.

طراحی واحد خطر

کد Verilog این واحد شامل بررسی‌هایی است که اگر نیاز به ارسال به جلو وجود داشته باشد، آن را اعمال می‌کند .در مواردی که نیاز به stall یا flush وجود دارد، از سیگنال‌های کنترلی ساده استفاده می‌کنیم چون طراحی کلی پردازنده ساده است.

ماژول نهایی (Top Module)

در ماژول نهایی، مسیر داده، کنترلر و واحد خطر به‌صورت ماژول‌های جداگانه تعریف و متصل شده‌اند .این ماژول اصلی نقش هماهنگ‌کننده کلی سیستم را دارد.

آزمایش و شبیه‌سازی

برای تست پردازنده، برنامه‌ای نوشته‌ایم که کمترین مقدار را از آرایه‌ای با ۱۰ عنصر پیدا می‌کند .ابتدا کد C مربوطه را نوشتیم، سپس آن را به اسمبلی تبدیل کردیم .فرض کردیم اولین عنصر آرایه در آدرس `0x5b0` حافظه قرار دارد . نگاشت ثبات‌ها نیز مشخص شده است:

x18: → S2 -داده موقت

x9: → S1 -اندیس i

x8: → S0 -مقدار کمینه

x6: → t1 -نتیجه مقایسه SLT

سپس کد اسمبلی را نوشته و آن را به کد ماشین) RISC-V فرمت (Hex تبدیل کردیم.

تبدیل کد اسمبلی به ماشین

هر دستور اسمبلی به کد هگزادسیمال تبدیل و به‌صورت معکوس) با اولویت پایین‌ترین بایت‌ها (در حافظه نوشته می‌شود تا با معماری little-endian سازگار باشد.

مثلاً کد 0x5ba40903 به‌صورت زیر در حافظه نوشته می‌شود:

```

03

09

A4

5B

```

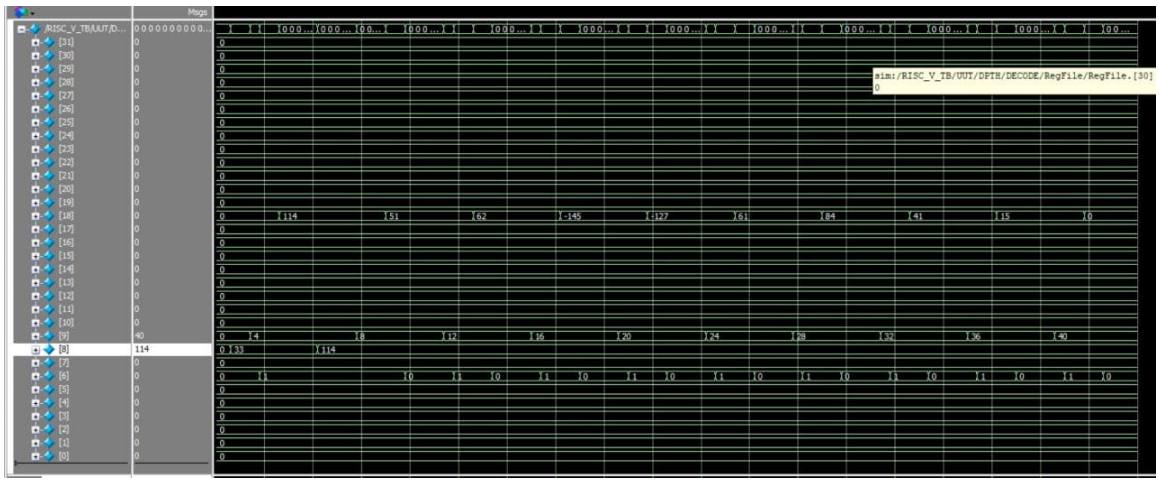
همین فرآیند برای داده‌های آرایه نیز تکرار شده است .آرایه شامل مقادیر زیر بود:

15، 41، 84، 61، -127، -145، 62، 51، 114،33

تست‌بنچ و نتایج

با استفاده از تست‌بنچ ارائه‌شده، کد را اجرا کردیم و عملکرد پردازنده را مشاهده نمودیم .عملکرد آن نسبت به طراحی چندچرخه‌ای بسیار سریع‌تر بود و مصرف توان بیشتری داشت که قابل قبول است.

شکل موج‌های مربوط به فایل رجیستر و حافظه نیز صحت عملکرد را تأیید می‌کنند که x8 بیشترین مقدارو تو خودش نگه داشته است.



تست بنچ نهایی

پایان