گزارش فاز چهارم پروژه درس معماری کامپیوتر

تهیه کنندگان:

پارمیدا جوادیان

پرنیان رضوی پور

دنیا نوابی

کیانا موسی زاده

دانشگاه صنعتی شریف

تابستان 1401

فهرست

مقدمه

در سه فاز قبلی پروژه پردازنده میپس همراه به دسترسی به حافظه و با قابلیت اجرای دستورات به صورت خط لوله طراحی کردیم. در این فاز قصد داریم به پردازنده خود قابلیت پردازش دستورات دارای عملوندهای اعشاری را نیز اضافه کنیم.

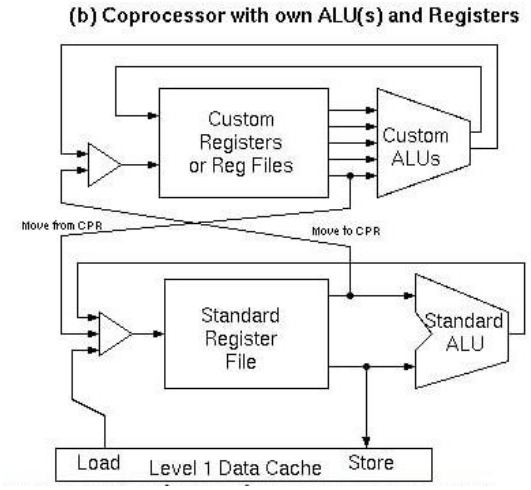
.

نحوه پیاده سازی

پردازنده پیاده سازی شده در این فاز به صورت جداگانه طراحی شده و تمامی بخش‌های آن از جمله regfile، control unit، واحد محاسبات و... مستقل از پردازنده اصلی هستند (البته به دلیل تنوع کمتر دستورات این پردازنده، نیازی به پیاده سازی برخی از واحدهای پردازنده اصلی مانند aluControl و mux های مربوط به jump و branch ها نبوده و این پردازنده شامل این قسمت‌ها نمی‌شود). طراحی ارتباطات این واحدها نیز مطابق ارتباطات پردازنده اصلی قبل از پیاده سازی معماری خط لوله است.

تنها بخشی از این پردازنده که وابسته به پردازنده اصلی است، دو واحد memory(instruction memory و data memory) است.

طراحی حدودی چنین کمک پردازنده ای در شکل زیر قابل مشاهده است:



ماژول‌های مورد استفاده در پردازش دستورالعمل‌های مربوط به اعداد اعشاری

حال به بررسی ماژول‌های اضافه شده به طور دقیق‌تر می‌پردازیم:

1) ماژول floatAdder

این ماژول دو عدد floating point و در فرم IEEE 754 را به عنوان ورودی دریافت کرده و حاصل جمع آن‌ها را به صورت و floating point و در فرم IEEE 754 حساب می‌کند.

می‌دانیم هنگام جمع شدن دو عدد، علامت حاصل جمع برابر با علامت عددی است که قدرمطلق بزرگ‌تری داشته باشد. بنابراین بیت sign حاصل را برابر با بیت sign عدد بزرگ‌تر قرار می‌دهیم.

برای محاسبه بخش mantissa حاصل جمع، در ابتدا بخش mantissa هر دو عدد را با یک بیت 1 از سمت چپ concat می‌کنیم.

در ادامه بخش mantissa عدد با کوچکتر را به اندازه اخلاف توان‌های دو عدد به سمت راست shift می‌دهیم و در صورت هم علامت بودن دو عدد ورودی، حاصل بدست آمده را با mantissa عدد بزرگ‌تر جمع می‌کنیم و در غیر این صورت حاصل بدست آمده را از mantissa عدد بزرگ‌تر کم می‌کنیم.

توجه کنید که پس از shift دادن، توان عدد کوچک‌تر نیز برابر با توان عدد بزرگ‌تر می‌شود. بنابراین توان حاصل جمع را برابر با توان عدد بزرگ‌تر قرار می‌دهیم.

در صورتی که بیت carry تولید شده باشد، آن بیت را وارد mantissa کرده و به exp یک واحد اضافه می‌کنیم.

حال می‌دانیم که اعداد در فرمت IEEE 754 به فرمت 1.X نوشته می‌شوند که ما بخش X را ذخیره می‌کنیم. بنابراین باید عدد 24 بیتی که از جمع mantissa ها بدست آمده را تا جایی به سمت چپ شیفت دهیم که بیت پرارزش آن برابر با 1 شود تا ما 23 بیت سمت راست آن را به عنوان mantissa ذخیره کنیم. توجه کنید هر شیفت به چپ، 1 واحد از exp کم می‌کند.

2) ماژول floatMultiplier

این ماژول دو عدد floating point و در فرم IEEE 754 را به عنوان ورودی دریافت کرده و حاصل ضرب آن‌ها را به صورت floating point و در فرم IEEE 754 حساب می‌کند.

می‌دانیم وقتی دو عدد دارای پایه مشترک در هم ضرب می‌شوند توان‌های آنها با هم جمع می‌شوند. بنابراین برای حساب کردن توان خروجی می‌توان بخش‌های exp دو عدد A و B را با یکدیگر جمع کرد. ولی با توجه به

اینکه هر یک از exp ها دارای بایاسی برابر با 127 می‌باشند، باید حاصل جمع exp ها A و B را منهای 127 کرد.

$Exp(A*B) = توان(A*B) + 127 = (توان(A) + 127) + (توان(B) + 127) – 127 = exp(A) + exp(B) – 127$

برای محاسبه بخش mantissa در ابتدا بخش‌های mantissa اعداد A و B را با یک بیت 1 در سمت چپ concat می‌کنیم و سپس حاصل ضرب اعداد بدست آمده را حساب می‌کنیم. در صورتی که بیت اندیس 47 حاصل ضرب برابر با یک بود، بیت‌های اندیس 46 تا 24 حاصل ضرب را به عنوان بخش mantissa حاصل ضرب قرار می‌دهیم و در غیر این صورت بیت‌های اندیس 45 تا 23 را به عنوان mantissa حاصل ضرب قرار می‌دهیم.

در نهایت به تعیین بیت sign حاصل ضرب می‌رسیم. در صورتی که دو عدد A و B هم علامت باشند، بیت sign حاصل ضرب برابر با 0 می‌شود (به معنی مثبت بودن حاصل ضرب) و در صورتی که دو عدد A و B مختلف العلامت باشند، بیت sign حاصل ضرب برابر با 1 می‌شود (به معنی منفی بودن حاصل ضرب).

3) ماژول floatDivider

این ماژول دو عدد floating point و در فرم IEEE 754 را به عنوان ورودی دریافت کرده و حاصل تقسیم عدد اول بر عدد دوم را به صورت floating point و در فرم IEEE 754 حساب می‌کند.

می‌دانیم تقسیم کردن A بر B به معنای ضرب کردن A در معکوس B است. بنابراین در این ماژول در ابتدا به معکوس B را بدست آورده و سپس با استفاده از ماژول ضرب کننده که قبلا به آن پرداختیم، A را در معکوس B ضرب می‌کنیم.

برای محاسبه معکوس B، از روشی مبتنی بر چند مرحله پیمایش استفاده می‌شود. در این روش ابتدا تخمینی اولیه از معکوس B را از روی mantissa و exp عدد B محاسبه کرده و سپس در چند مرحله دقت آن را تا حد نیاز زیاد می‌کنیم.

4) ماژول aluf

این ماژول به انجام محاسبات لازم برای دستورات Add، Sub، Mul، Div، C.EQ، C.LE، C.LT، Neg، Round، LW، SW و Lui می‌پردازد و همچنین flag های division by zero، QNaN، SNaN، Inexact، Underflow، Overflow را برحسب شرط برقراری هرکدام محاسبه می‌کند.

در ادامه به نحوه انجام هر یک از این عملیات‌ها می‌پردازیم:

1) Add

وظیفه این دستور محاسبه حاصل جمع دو عدد ورودی است و این کار را به instance گرفتن از ماژول floatAdder که در بخش‌های قبلی نحوه عملکردش را توضیح دادیم انجام می‌دهد.

2) Sub

وظیفه این دستور تفریق دو عدد ورودی است. می‌دانیم تفریق عدد B از A به معنی جمع کردن A با قرینه عدد B است. بنابراین برای انجام این دستور عدد دوم را قرینه کرده (اگر بیت sign آن 0 بود آن را به 1 تغییر می‌دهیم و اگر بیت sign آن 1 بود آن را به 0 تغییر می‌دهیم) و سپس عدد بدست آمده را با استفاده از floatAdder ای که در بخش قبل به آن پرداختیم با A جمع می‌کنیم.

3) Mul

وظیفه این دستور ضرب کردن دو عددی ورودی و قرار دادن حاصل در خروجی است که این کار را به instance گرفتن از ماژول floatMultiplier که در بخش‌های قبلی نحوه عملکردش را توضیح دادیم انجام می‌شود.

4) Div

وظیفه این دستور تقسیم کردن عدد اول ورودی بر عدد دوم ورودی است که برای این کار از ماژول floatDivider که در بخش قبل به آن پرداختیم instance گرفته می‌شود.

5) C.EQ

وظیفه این دستور این است که دو عدد را با هم مقایسه کرده و در صورت برابر بودن result را یک کرده و در غیر این صورت آن را صفر کند.

با توجه به اینکه هر عدد در فرم IEEE 754 حالت یکتایی برای نمایش دارد، در صورتی که توان، علامت و بخش اعشاری(mantissa) دو عدد با هم یکسان باشند، آن دو عدد برابرند.

6) C.LE

وظیفه این دستور این است که دو عدد را با هم مقایسه کرده و در صورتی که عدد اول کوچک‌تر یا مساوی عدد دوم باشد یا دو عدد مساوی باشند، خروجی result را یک کرده و در غیر این صورت آن را صفر کند.

در ابتدا به بررسی علامت دو عدد می‌پردازیم. در صورتی که دو عدد علامت متفاوت داشته باشند، عددی که علامت منفی دارد کوچکتر از عددی که علامت مثبت دارد است. در صورتی که علامت ها یکسان باشند، به مقایسه توان‌ها میپردازیم. در صورتی که علامت دو عدد مثبت باشد، عددی کوچکتر است که توان کمتری دارد و در صورتی که علامت دو عدد منفی باشد، عددی کوچکتر است که توان بزرگتر دارد. در صورتی که توان ها نیز برابر بودند، به مقایسه بخش اعشاری می‌پردازیم. اگر توان هر دو عدد مثبت بود، عددی کوچکتر است که بخش اعشاری کوچکتر داشته باشد و اگر توان هر دو عدد منفی بود، عددی

کوچکتر است که بخش اعشاری بزرگتر داشته باشد. در صورتی که هیچ کدام از حالات فوق برقرار نبود(یعنی توان و علامت و بخش اعشاری مساوی بودند)، دو عدد با هم برابرند که در این دستور، این حالت نیز مطلوب ما است.

7) C.LT

وظیفه این دستور این است که دو عدد را با هم مقایسه کرده و در صورتی که عدد اول کوچک‌تر یا مساوی عدد دوم باشد، خروجی result را یک کرده و در غیر این صورت آن را صفر کند.

شیوه بررسی کوچک‌تر بودن مشابه با توضیحات بالا است به این معنی که در ابتدا به مقایسه علامت و سپس به مقایسه توان و در نهایت به مقایسه بخش اعشاری می‌پردازیم. در صورتی که تمامی موارد فوق برابر بودند، دو عدد با هم برابرند که در این دستور، این حالت مطلوب ما نیست.

8) Neg

وظیفه این دستور قرینه کردن عدد ورودی است.

برای قرینه کردن اعداد کافیست بیت علامت آنها را قرینه کنیم. به این صورت که در صورت 1 بودن بیت علامت(منفی بودن عدد)، بیت علامت را به 0 تغییر می‌دهیم(عدد مثبت می‌شود) و در صورت 0 بودن بیت علامت(مثبت بودن عدد)، بیت علامت را به 1 تغییر می‌دهیم(عدد منفی می‌شود).

9) Round

وظیفه این دستور گرد کردن عدد اعشاری ورودی داده شده است. در صورتی که توان عدد کوچکتر از منفی دو باشد، گرده شده آن برابر با صفر می‌شود. در صورتی که توان عدد منفی یک باشد، گرد شده عدد برابر با 1 می‌شود. برای حالاتی که توان عدد بین صفر و 22 باشد، یک بیت 1 را با n بیت پرارزش از بخش اعشاری concat می‌کنیم(n برابر با توان عدد است). در انتها عدد خروجی را با m عدد صفر از سمت چپ concat می‌کنیم.($m = 32 - (n+1)$). تا اینجای کار عدد قطع شده را محاسبه کرده ایم.

در ادامه در صورتی که بیت $n+1$ ام برابر با یک بود(بخش اعشاری عدد بزرگتر یا مساوی 0.5 بود)، خروجی را به علاوه یک می‌کنیم تا به مقدار گرد شده برسیم.

در این مرحله یک عدد صحیح داریم که فرمت ذخیره آن به صورت عادی است. در صورتی که باید در فرمت IEEE 754 ذخیره شود. برای تبدیل عدد بدست آمده به فرمت IEEE 754 به صورت زیر عمل می‌کنیم:

پس از مشاهده اولین 1 در سمت چپ عدد، بخش سمت راست 1 را به عنوان mantissa خروجی قرار می‌دهیم و در صورتی که mantissa بدست آمده 23 بیتی نباشد، آن را تعدادی 0 (به اندازه اختلاف تعداد ارقام آن و عدد 23) از سمت راست concat می‌کنیم. بخش exp را هم برابر با تعداد ارقام سمت

راست اولین 1 قرار می‌دهیم و بخش sign را نیز برابر با sign عدد ورودی قرار می‌دهیم. به این تربیت عدد بدست آمده در فرمت IEEE 754 قرار می‌گیرد.

10) LW

برای انجام این دستور در alu صرفا آدرس مورد نیاز باید تولید شود که این کار مطابق طراحی پردازنده اصلی قبل از پیاده سازی معماری خط لوله انجام می‌شود.

11) SW

انجام این دستور نیز مشابه دستور LW صرفا با تولید آدرس مورد نیاز انجام می‌شود و بقیه مراحل آن مطابق پردازنده اصلی است.

12) Lui

این دستور به معنای loadUpperImmediate است و به معنای ذخیره 16 بیت کم ارزش ورودی در 16 بیت پر ارزش خروجی است. که برای این کار بیت 15 تا 0 ورودی را از سمت راست با 16 بیت 0 concat کرده و به عنوان خروجی قرار می‌دهیم.

5) ماژول CU_f

این ماژول مشابه ماژول control است که در فازهای قبل به آن پرداختیم. با این تفاوت که مربوط به دستورات مرتبط با floatingPoint است. لازم به ذکر است که تعداد دستورات مربوط به floatingPoint کم‌تر از دستورات عادی است؛ به همین دلیل قسمتی از ماژول control در این ماژول وجود ندارد و بخش‌هایی اعم از تشخیص R-type یا I-type بودن دستور، تشخیص مقصد و... باقی می‌مانند.

6) ماژول coprocessor

این ماژول مشابه ماژول mips_core است و عملکرد آن به این صورت است که یک دستورالعمل را به عنوان ورودی دریافت کرده و با استفاده از ماژول‌هایی که از آن‌ها instance گرفته است، دستور را هدایت می‌کند تا به ماژول aluf برسد و در آن‌جا عملیات مربوط به آن انجام شده و در صورت نیاز به حافظه دسترسی پیدا کند و یا نتیجه را در regfile ذخیره کند.

همانطور که پیش‌تر گفته شد واحدهای instruction memory و data memory به طور مستقل در coprocessor وجود ندارند و کمک پردازنده باید جهت دریافت دستورات از instruction memory و load و store کردن، به وسیله ورودی‌ها و خروجی‌ها با پردازنده اصلی ارتباط برقرار کند.

جهت گرفتن دستورات از instruction memory، بعد از اجرای مرحله fetch در پردازنده اصلی، در صورتی که opcode دستور، تعیین کننده یک دستور floatingPoint بود، دستور مورد نظر به جای انتقال به مراحل بعدی pipeline، به کمک پردازنده داده می‌شود.

برای ارتباط با data memory هم مشابه فازهای قبلی، ارتباط بین کمک پردازنده و cache برقرار می‌شود و ارتباطات لازم با memory توسط cache مدیریت می‌شود. بنابراین خروجی‌های لازم جهت این ارتباط توسط coprocessor تولید شده و به واحد cache که در memStage در پردازنده اصلی قرار دارد، داده می‌شوند تا پردازش‌های لازم انجام شده و در صورت نیاز، ورودی‌ها از این واحد به coprocessor داده شوند.