# COMPUTER ARCHITECTURE

PROFESSOR :

SARBAZI AZAD

# PHASE 4


MEMBERS :

HOSSEIN GOLI

AMIR HOSSEIN BARATI

ERFAN SADRAIYE

ALIREZA FOROODNIA

## مقدمه :

در این فاز بنابر تصمیم جمعی قرار بر این شد که یک ماژول با قابلیت انجام محاسبات ممیز شناور به پردازنده قبلی اضافه گردد. همچنین register file مرتبط با این ماژول نیز به صورت جداگانه تعبیه می شود تا در محاسبات مربوط به آن مورد استفاده قرار بگیرد.

## کلیات پیاده سازی :

روش به کار برده شده در این فاز به گونه ای است که با تعیین opcode های لازم دستورات ممیز شناور شناسایی شده و محاسبات مورد نیاز توسط واحد خاص که برای این دست محاسبات تعبیه شده انجام می گیرد. همچنین با توجه به این که این پردازنده در فاز قبلی دستورات را به صورت پایپلاین اجرا می کند در این فاز نیز دستورات مرتبط با محاسبات ممیز شناور نیز به پایلاین مذکور متصل شده اند. امکان جا به جایی داده میان رجیستر های عادی و رجیستر های مخصوص نیز امکان پذیر است.ماژول مرتبط با محاسبات ممیز شناور به طور جداگانه و در کنار ALU قبلی پردازنده تعبیه شده و ورودی ها به آن وارد شده و خروجی های آن به کمک چند مالتی پلکسر به خروجی ها هدایت می شوند.

## جزئیات پیاده سازی :

### ماژول floating point alu

این ماژول محاسبه گر مربوط با عملیات های ممیز شناور است که از الگوریتم های متفاوتی برای انجام انواع محاسبات استفاده می کند که هر کدام از آنها به شرح زیر هستند :

#### دستور MTC1:

این دستور قصد دارد محتویات یک رجیسترفایل از رجیسترفایل صحیح را در یک ثبات از رجیسترفایل اعداد ممیز شناور بریزد. برای این کار با توجه به معماری در نظر گرفته دو راه در پیش داشتیم. یکی این که محتویات این ثبات را با عدد صفر جمع کنیم و سپس در رجیستر فایل دوم بریزیم یا اینکه این مقدار را تا مرحله ۵ پایپ کنیم و سپس در رجیسترفایل دیگر بریزیم که با توجه به پیچیدگی هر دو نوع پیاده‌سازی، در نهایت تصمیم تیم بر این شد که از روش دوم استفاده کنیم.

از آنجایی که در پیاده سازی این دستور از دیکود کردن به روش R-Type استفاده می‌کنیم، پس باید محل قرارگیری هر دو ثبات مبدا و مقصد را در این نوع به درستی مشخص کنیم. با توجه به پیاده‌سازی‌ای که در این پروژه داشتیم، اولین محل آدرس‌دهی ثبات در دستور، نمی‌تواند به عنوان ثبات مقصد قرار بگیرد. بنابراین برای ثبات مقصد بایستی از یکی از دو محل دیگر استفاده کنیم. در نهایت به این شکل این دستور را دیکود می‌کنیم:

R-type:

| 6 bit | 5 bit | 5 bit | 5bit | 5 bit | 6bit |
|---|---|---|---|---|---|
| Opcode | | rs | fd | shamt | func |

که در این دستور مقدار func برابر با عدد ۶ بیتی ۱۱۱۱۱۰ است. مقدار قرار گرفته در fd برابر با آدرس ثبات در رجیسترفایل ممیز شناور است و مقدار قرار گرفته در rs برابر با آدرس ثبات مبدا در رجیسترفایل اعداد صحیح است.

#### دستور MFC1:

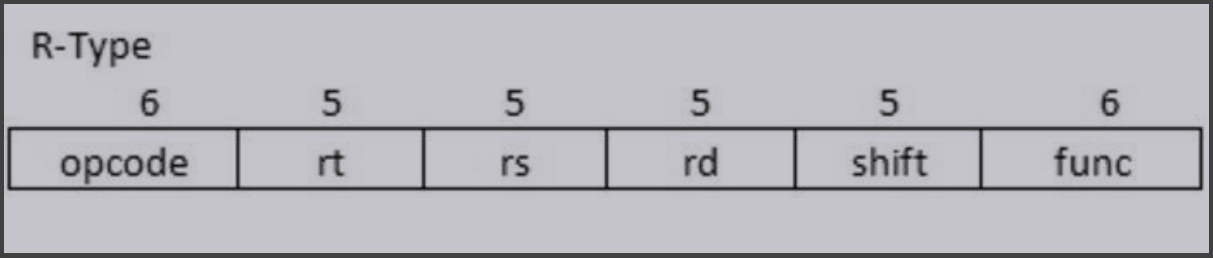
این دستور همانند دستور قبل یعنی MTC1 عمل می‌کند تنها با این تفاوت که محتویات ثبات دلخواه را از رجیسترفایل ممیز شناور به ثبات درون رجیسترفایل اعداد صحیح منتقل می‌کند. پیاده‌سازی این قسمت مشابه با دستور قبلی خواهد بود یعنی از پایپ کردن مقدار ثبات تا مرحله آخر استفاده می‌کنیم.

یکی دیگر از تفاوت‌ها با دستور قبل، نحوه دیکود کردن این دستور است. این دستور همانند شکل زیر در پردازنده دیکود خواهد شد:

R-type:

| 6 bit | 5 bit | 5 bit | 5bit | 5 bit | 6bit |
|---|---|---|---|---|---|
| Opcode | fs | rd | | shamt | func |

در ابتدا نحوه دیکود شدن این دستور را در پردازنده بررسی می‌کنیم. همانند دستور ضرب در اعداد صحیح، این دستور را در زیر گروه دستورهای R-Type قرار می‌دهیم و ترتیب دیکود کردن این دستور همانند آن دستور خواهد بود، همانند شکل زیر:




مقدار ۶ بیتی func مربوط به این دستور برابر با ۱۱۱۰۰۰ و مقدار قرار گرفته در قسمت shift اهمیتی ندارد ولی بهتر است برابر با صفر باشد.

الگوریتم مورد استفاده در پیدا کردن حاصلضرب همانند الگوریتمی است که در کلاس بیان شده بود. در این روش ابتدا با جمع کردن توان ها با یکدیگر و کاهیدن مقدار bias از این حاصل جمع، حاصل توان را به دست می‌آوریم. علامت حاصل هم در همین ابتدا قابل به دست آوردن است. علامت حاصلضرب برابر با xor بیت‌های علامت مضروب و مضروب‌فی خواهد بود. و اما مهم‌ترین بخش، به دست آوردن حاصلضرب قسمت کسری اعداد خواهد بود. در این روش ابتدا یک بیت ۱ در سمت چپ هر دو عدد قرار دادیم و سپس دو عدد ۲۴ بیتی به دست آمده را همانند ضرب Fixed Point در یکدیگر ضرب کردیم. می‌دانیم حاصلضرب به دست آمده میتواند حداکثر ۴۸ بیت باشد که بیت های ۴۸ تا ۴۶ بیت‌های قسمت صحیح آن باشد و میتوانند مقداری بیشتر از ۱ داشته باشند که در این صورت نمی‌توانند با جایگذاری در فرمت IEEE 754 قرار بگیرند. اکنون در این حالت با شیفت دادن کل حاصلضرب به سمت چپ به عددی میرسیم که در قسمت صحیح خود، تنها یک بیت ۱ داشته باشد و با هر بار شیفت دادن یکی به توان حاصلضرب اضافه می‌کنیم. البته در ابتدا مطمئن بودیم حداقل یک بیت در این قسمت برابر با یک خواهد بود، زیرا در حال ضرب کردن دو عدد ممیز ثابت بیشتر از یک بودیم.

### دستور ADD.S :

در الگوریتم جمع به کار رفته ابتدا با شیفت و افزایش مقدار exp کمتر مقدار توان هر دو عدد را یکی می کنیم. سپس با مقایسه کردن علامت های آنها می توان تشخیص داد که آیا نیاز به تفریق مقدار mantissa های هست یا جمع آنها. پس از تعیین تکلیف نوع عملیات این عملیات بر روی مقادیر مذکور انجام می پذیرد و سپس در صورت لزوم با انجام عملیات شیفت مقادیر استاندارد می شوند و سپس به خروجی هدایت می شوند.

لازم به ذکر است عملیات SUB.S نیز با الگوریتمی مشابه انجام می پذیرد و تنها تفاوت در این است که درشرایط تفریق یا جمع مقادیر mantissa عکس حالت بالا لحاظ می شود.

### دستور DIV.S :

در این دستور ابتدا صفر بودن مقسوم علیه چک می شود تا در صورت لزوم حالت division by zero اتیجاد شود در غیر این صورت مقسوم چک می شود تا اگر صفر باشد مقدار صفر خروجی داده شود. حال اگر هیچ کدام از این دو حالت برقرار نبود ابتدا مقادیر mantissa ها مقایسه می شوند تا در صورت بزرگتر بودن مقسوم با شیفت این مقدار قابل تقسیم شود و ایجاد overflow بی دلیل نشود. پس از این عملیات مقدار توان با یک تفریق تعیین می شود و پس از آن هر دو mantissa به یکدیگر تقسیم شده و بیت های mantissaخروجی در صورت نیاز با عملیات شیفت به حالت نرمال تبدیل شده و خروجی نهایی می شود.

### دستور INV.S :

این دستور به طور کامل از الگوریتم تقسیم بیان شده بهره می برد با این تفاوت که مقدار مقسوم همواره به یک تبدیل می شود و مقدار 1/input در خروجی ظاهر می شود. گفتنی است درست مانند عملیات تقسیم در صورت ورودی دادن مقدار صفر division by zero در خروجی پدیدار می شود.

### دستور RND.S :

الگوریتم به کار برده شده بدین صورت است که ابتدا چک می شود مقدار توان عدد ورودی. این عدد اگر کمتر از 126 باشد خروجی در هر صورت صفر است و اگر بیش از 149 باشد خروجی با ورودی یکسان خواهد بود.

در غیر از حالت های بالا بسته به میزان توان یکی از بیت های mantissa مشخص می گردد که بیت های قبل از آن ارزش کمتر از یک دارند و در فرایند گرد شدن بی ارزش هستند فلذا بیت های قبل از آن حذف شده و سایر بیت ها با شیفت خوردن و سپس نظر به بیت منتخب مقدار خروجی تعیین می شود و در نهایت در صورت لزوم با انجام عملیات شیفت خروجی نهایی در صورت لزوم نرمالایز می شود.

### دستور SLT.S:

این دستور نوعی دستور مقایسه‌ای برای اعداد ممیزی شناور است. زمانی که عدد قرار گرفته در ثبات اول کمتر از عدد قرار گرفته در ثبات دوم بود، خروجی این دستور برابر با یک است و در غیر این صورت برابر با صفر خواهد بود.

الگوریتم مقایسه این دو دستور به شکل زیر خواهد بود. در صورتی که علامت دو عدد متفاوت باشد، اگر عدد اول علامت منفی داشت، خروجی برابر با یکی و در غیر این صورت خروجی برابر با صفر است. حال اگر علامت هر دو عدد مثبت بود، به مقایسه توان آنها میپردازیم. اگر علامت هر دو مثبت بود و عدد اول توان کمتری داشت، خروجی یک و اگر توان آن بیشتر بود خروجی برابر با صفر خواهد شد. اگر علامت هر دو عدد منفی بود و توان عدد اول بیشتر بود، خروجی برابا یک و در صورتی که توان عدد اول کمتر بود خروجی برابر با صفر خواهد شد. در صورتی که هیچ یکی از حالات فوق رخ نداد، نوبت به مقایسه قسمت مانتیس میرسد. اگر هر دو عدد مثبت بودند و مانتیس اولی از دومی کمتر بود، خروجی برابر با یک و در غیر این صورت برابر با صفر خواهد شد. اگر هر دو عدد منفی بودند و مانتیس اولی بیشتر از دومی بود، خروجی برابر با یک در غیر این صورت برابر  یا صفر خواهد شد. در حالات فوق، حالت مساوی هم خروجی مناسب یعنی صفر را خواهد داد.

نحوه دیکود کردن این دستور همانند دیکود کردن دستور SLT برای اعداد صحیح است و تنها در مقدار ۶ بیتی func با یکدیگر متفاوت هستند

## سیگنال های خروجی :

### سیگنال division by zero :

همانطور که از نام این سیگنال پیداست هنگام تقسیم عددی به صفر و یا تلاش برای گرفتن معکوس عدد صفر این ارور رخ می دهد.

سیگنال overflow :

این سیگنال هنگامی که مقدار عدد از حداکثر ممکن در حال تجاوز باشد و نیاز باشد که عدد توان از حداکثر فرا تر رود این سیگنال ۱ می شود و باعث می گردد که خروجی بسته به علامت خروجی به مثبت یا منفی بی نهایت تغییر کند.

سیگنال underflow :

این سیگنال مانند سیگنال overflow هنگامی که خروجی از حداقل مقدار ممکن کم تر بشود مقدار ۱ می گیرد ولی برعکس سیگنال overflow منجر به تغییر نوع خروجی نمی شود و آنچه که حاصل عملیات باشد را در خروجی نشان می دهد.

سیگنال SNaN :

این سیگنال در صورتی که به یک عملیات ورودی ای داده شود که از نوع NaN باشد ظاهر می شود همچنین دقت شود در صورتی که تلاش شود تا عملیات هایی مانند تقسیم صفر به صفر یا بی نهایت به بی نهایت یا ضرب صفر در بی نهایت انجام شود نیز عملیات غیر مجاز بوده و سیگنال SNaN فعال می شود.

سیگنال QNaN :

این سیگنال تحت شرایطی فعالیت می کند که مقدار خروجی پس از انجام محاسبات مربوطه از نوع NaN باشد. دقت شود در صورتی سیگنال QNaN به مقدار ۱ تغییر می کند که SNaN مقدار صفر داشته باشد.

نکته : تعریف دقیق و کلی سیگنال های QNaN و SNaN با تعاریف بالا کاملا یکسان نیست ولیکن پیاده سازی بالا نزدیک ترین پیاده سازی ممکن به تعریف ارائه شده توسط IEEE می باشد.