گزارش فاز ۲ پروژه ی معماری کامپیوتر

دستورالعمل های جدید مربوط به کمک پردازنده ی اعداد مختلط را به این شکل تعریف میکنیم :

- فرمت و مجموعه ی دستورات محاسباتی و منطقی :

Opcode	Destination Reg	Source Reg 1	Source Reg 2	Reserved
26-31	21-25	16-20	11-15	0-10

Opcode	Instruction	Description
100001	ADD(Exponential Form)	DST ← SRC1 + SRC2
100010	SUB(Exponential Form)	DST ← SRC1 − SRC2
100011	MUL(Exponential Form)	DST ← SRC1 × SRC2
100100	DIV(Exponential Form)	DST ← SRC1 ÷ SRC2
100101	DUAL(Exponential Form)	DST ← (dual)SRC1
100110	INV(Exponential Form)	DST ← (inverse)SRC1
100111	CONV(to Polar Form)	DST ← (Polar Form)SRC1
101000	CMP(Exponential Form)	if|SRC1| < |SRC2| DST = 0 else DST = 1
101001	ADD(Polar Form)	DST ← SRC1 + SRC2
101010	SUB(Polar Form)	DST ← SRC1 − SRC2
101011	MUL(Polar Form)	DST ← SRC1 × SRC2
101100	DIV(Polar Form)	DST ← SRC1 ÷ SRC2
101101	DUAL(Polar Form)	DST ← (dual)SRC1
101110	INV(Polar Form)	DST ← (inverse)SRC1
101111	CONV(to Exponential Form)	DST ← (Exponential Form)SRC1
110000	CMP(Polar Form)	if|SRC1| < |SRC2| DST = 0 else DST = 1

• فرمت و مجموعه دستورات با عملوند صریح :

Opcode	Destination Reg	Source Reg	Immediate Data
26-31	21-25	16-20	0-15

Opcode	Instruction	Description
110001	LII(load imaginary/θ part)	DST[15:0] ← IMM
110010	LRI(load real/r part)	DST[31:16] ← IMM

• فرمت و مجموعه دستورات دسترسی به حافظه :

Opcode	Value Reg	Adress Reg	Offset
26-31	21-25	16-20	0-15

Opcode	Instruction	Description
110011	LW	VR ← MEM[$AR + SIGN EXTEND (Offset)]
110100	SW	MEM[$AR + SIGN EXTEND (Offset)] ← VR

واحد Register File :

در این معماری Registe File از ۳۲ رجیستر ۳۲ بیتی تشکیل شده است که هر کدام به دو قسمت ۱۶ بیتی تقسیم شده اند. هر قسمت ۱۶ بیتی شامل ۸ بیت عدد صحیح و ۸ بیت اعشار است.

در هر عدد مختلط نمایی (به شکل a + bi)، بیت های ۱۶ تا ۳۲ هر رجیستر a و بیت های ۰ تا ۱۵ b را نگه داری میکنند.

در هر عدد مختلط قطبی (به شکل rcosθ + rsinθ i) بیت های ۱۶ تا ۳۲ هر رجیستر r و بیت های ۰ تا ۱۵ θ (بر حسب رادیان) را نگه داری میکنند.

واحد ALU :

برای این واحد از بلاک های آماده ی ویزارد برای جمع، تفریق، ضرب، تقسیم، مزدوج و معکوس استفاده شده.

برای کنترل کردن مقدار های اعشاری باید قبل از تقسیم ها مقسوم را ۸ بیت به چپ شیفت دهیم و بعد از ضرب ۱۶ بیت وسط را نگه داری کنیم. (۱۶ بیت اعشار و ۱۶ بیت عدد صحیح ایجاد میشود که ما از هر کدام ۸ بیت را نگه داری میکنیم)

عملیات های محاسباتی در اعداد مختلط با فرمت نمایی در زیر آورده شده است :

$$ADD/SUB:\ (a+bi)\ \pm(c+di) = (a\ \pm c) + (b\ \pm d)i$$

$$MUL:\ (a+bi)\ .\ (c+di) = (ac-bd) + (ad+bc)i$$

$$DIV:\ \frac{a+bi}{c+di} = \frac{ac+bd}{c^2+\ d^2} + \frac{bc-ad}{c^2+\ d^2}i$$

$$dual(a+bi) = a-bi$$

$$inverse(a+bi) = \frac{a}{a^2+\ b^2} - \frac{b}{a^2+\ b^2}i$$

برای تبدیل فرمت های عدد مختلط دو بلاک جداگانه با نام های pf_to_exp_converter برای تبدیل نمایش قطبی به نمایی و exp_to_pf_converter برای تبدیل نمایش نمایی به قطبی زده شده و سپس از هر کدام از بلاک ها یک symbol file فایل ساخته شده و در diagram اصلی که co_p_alu نام دارد استفاده شده اند.

- تبدیل نمایی به قطبی : r و θ را از a و b با رابطه های زیر بدست می آوریم.

$$r = \sqrt{a^2+\ b^2}$$

$$\tan(\theta) = \frac{b}{a},\quad \theta = \tan^{-1}(\frac{b}{a})$$

برای بدست آوردن arctan نیز از دو جمله ی اول سری تیلور- مکلورن استفاده میکنیم.

$$\tan^{-1}(x) = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \frac{x^9}{9} + \dots$$

- تبدیل قطبی به نمایی : a و b را از r و θ با رابطه های زیر بدست می آوریم.

$$a = r\ .\ \cos(\theta) \qquad b = r\,.\sin(\theta)$$

برای بدست آوردن sin و cos نیز از دو جمله ی اول سری تیلور- مکلورن استفاده میکنیم.

$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \dots$$

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$$

سری تیلور مکلورن برای بدست آوردن arctan (حتی با استفاده از جملات بیشتر)، در x هایی با قدر مطلق بزرگتر از ۱ خیلی دقیق عمل نمیکند و واگرا میشود. پس در تبدیل عدد های نمایی به قطبی با نسبت b/a خیلی زیاد ممکن است با خطا مواجه شویم.

علاوه بر آن به دلیل استفاده از تعداد جمله ی محدود از سری تیلور برای سینوس و کسینوس در تبدیل ها مقدار بسیار کمی خطا ایجاد میشود که قابل صرف نظر است ولی در محاسبه ی تبدیل قطبی به نمایی، به دلیل وجود ضریب r برای سینوس و کسینوس، با افزایش مقدار r خطا نیز به صورت خطی بزرگتر میشود. بنابراین ممکن است که در r های خیلی بزرگ در تبدیل قطبی به نمایی، اعداد به صورت کاملا دقیق نباشند و خطای بسیار کمی ایجاد شود.

همچنین اعشار تا ۸ رقم باینری نگه داری میشوند (معادل تقریبا ۳ رقم ده دهی) که ممکن است خطای ناچیز ایجاد بکند. (معمولا تمامی خطا ها در حد اعشار هستند.)

عملیات های محاسباتی در اعداد مختلط با فرمت نمایی در زیر آورده شده است : (از این موارد تنها برای چک کردن نتیجه استفاده شده وگرنه برای محاسبات قطبی ابتدا اعداد توسط بلاک pf_to_exp_converter به فرم نمایی تبدیل میشوند و محاسبات آنها به صورت نمایی انجام شده و نتیجه توسط بلاک exp_to_pf_converter دوباره به فرم قطبی در آورده میشود)

جمع و تفریق : محاسبات جمع و تفریق در فرم قطبی شباهت های زیادی به محاسبه روی تبدیل شده ی آنها به فرم نمایی دارد (به این صورت که در محاسبه ی آنها نیز ابتدا کسینوس و سینوس محاسبه شده و در نهایت از arctan استفاده شده) بنابراین برای چک کردن از خود تبدیل به نمایی و محاسبات نمایی استفاده کردیم.

ضرب و تقسیم :

$$(r_1,\ \theta_1)\ .\ (\ r_2,\ \theta_2) = (r_1\,.\,r_2\ ,\ \theta_1 + \ \theta_2)$$

$$\frac{r_1,\theta_1}{r_2,\theta_2} = (\frac{r_1}{r_2}\ ,\ \theta_1 - \ \theta_2)$$

قبل از ورود به ALU در صورتی که تشخیص داده شود که opcode مربوط به دستور های LII و LRI میباشد، به جای دیتای اول، مقدار قبلی رجیستر dst داده میشود. با این کار مقدار immediate تنها در جای تعیین شده ی رجیستر قرار میگیرد و این دو دستور بقیه ی بیت های رجیستر dst را تغییر نمیدهد.

در نهایت، مقدار های محاسبه شده، به ترتیب opcode-1 در MUX خروجی قرار میگیرند و result انتخاب میشود. result انتخاب شده به فرمت نمایی است. پس در صورتی که opcode مربوط به محاسبات با خروجی قطبی بود، result دوباره به فرم قطبی در آمده و در خروجی نهایی ALU قرار میگیرد.

قسمت Data Path :

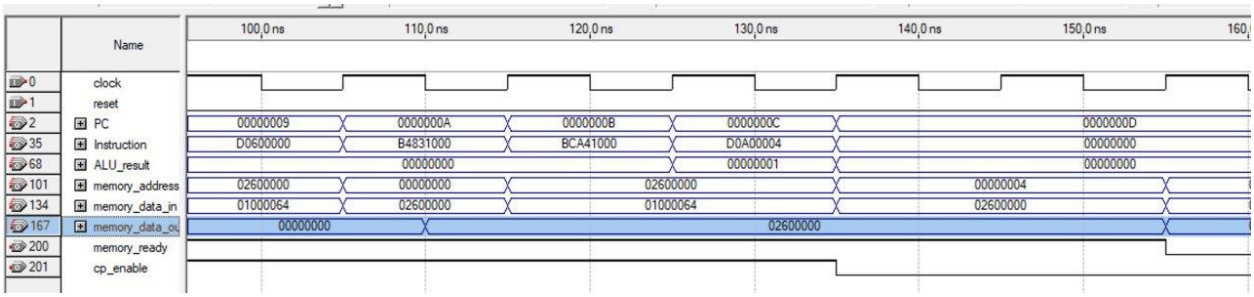
در قسمت مسیر داده‌ی کمک پردازنده یک واحد رجیستر فایل و یک ALU داریم که مشابه پردازنده به یک دیگر متصل شده‌اند. این کمک پردازنده دارای ورودی input_instruction و cp_enable است؛ Instruction مرحله‌ی ID پردازنده به input_instruction وصل می‌شود و در صورت فعال بودن enable آن را در یک رجیستر قرار می‌دهد. سیگنال enable توسط واحد کنترل پردازنده‌ی اصلی مقداردهی می‌شود و فقط در صورتی یک است که آپکد مربوط به کمک‌پردازنده باشد.

خروجی رجیستر گفته شده که دستور مربوط به کمک پردازنده است مشابه پردازنده‌ی اصلی به رجیستر فایل متصل می‌شوند.

برای ارتباط با حافظه کمک پردازنده دارای ورودی‌های memory_ready و memory_data_in و خروجی‌های memory_address، memory_data_out و mem_write است. اگر opcode دستور مربوط به کمک‌پردازنده باشد کنترل یونیت پردازنده‌ی اصلی یک کامند به نام cp_mem_src را یک می‌کند. در صورت فعال بودن این سیگنال در مرحله‌ی MEM ورودی‌های حافظه از خروجی‌های کمک‌پردازنده گرفته می‌شوند. همچنین ورودی memory_ready هم در مرحله‌ی MEM فعال می‌شود و به معنای این است که مقدار حافظه برای خوانده شدن آماده است و در صورت لزوم کمک‌پردازنده می‌تواند داده را بخواند و در رجیسترهای خود بنویسد.

فایل‌های مربوط به کمک پردازنده در فولدر Coprocessor قرار گرفته اند. همچنین پردازنده‌ی اصلی و کمک‌پردازنده در فایل device در فولدر Device قرار دارند.

شکل زیر مربوط به اجرای تست دوم (coprocessor_test_2) می‌باشد.



در این تست ورودی اول به صورت $r = 2$, $\theta = 0.3906$ و ورودی دوم به صورت $r = 1$ و $\theta = 0.3906$ تعریف شدند و جواب پایانی در فرم نمایی (تا ۴ رقم اعشار) برابر است با $2.0002 - 0i$ که در مبنای شانزده برابر می‌شود با 02000000 که جواب نهایی ما یعنی 02600000 تنها 0.375 با آن اختلاف دارد.