پروژه ی پایانترم درس مدارهای منطقی (برنامه نویسی وریلاگ)

استاد درس: دکتر فرشاد خونجوش
دستیار استاد : محمدحسین اله اکبری

نیم سال دوم 1401

فاز اول:

در این فاز شما باید یک شبه پردازنده با معماری Little Endian هشت بیتی را پیاده‌سازی کنید که در هر پالس ساعت یک دستور بیست بیتی را خوانده، محاسبات را با سیستم two's complement انجام دهد و جواب آخرین دستور را بر روی یک خروجی شانزده بیتی قرار دهد و همچنین فلگ های مربوط به هر محاسبه را نیز تنظیم کند.

توضیحات :

- دستوراتی که پردازنده باید آن‌ها را اجرا کند در فایلی به نام "instructions.txt" ذخیره شده‌اند و حداکثر تعداد دستورات نوشته شده در فایل نیز 24 دستور میباشد.

- پردازنده پس از اجرای هر دستور محاسباتی، دو فلگ به نام های ZF و SF را تنظیم میکند. فلگ اول صفر بودن حاصل را نمایش میدهد(در صورت صفر بودن 1 و در غیر اینصورت 0) و فلگ دوم علامت عدد محاسبه شده را نمایش میدهد. (منفی 1 و مثبت 0)

- این پردازنده دارای چهار رجیستر هشت بیتی به نام های Ra, Rb , Rc و Rd میباشد و هر یک از این رجیستر ها یک شناسه ی 8 بیتی دارند که در قسمت جدول ها ذکر شده اند.

هر دستور پردازنده از سه بخش تشکیل شده است:

کد دستور:

هر دستور دارای یک کد 4 بیتی است که پردازنده با توجه به آن تصمیم میگرد که چه عملیاتی بر روی عملوند ها انجام دهد.همچنین کد دستور تعیین میکند که عملوند دوم شناسه ی رجیستر است یا یک عدد هشت بیتی است.
کد دستور در ابتدای هر دستور می‌آید و بیت 20 ام تا 17 ام را به خود اختصاص میدهد.

عملوند اول:

عملوند اول هر دستور، شناسه یکی از رجیستر های پردازنده است و تعیین میکند که حاصل هر دستور در کجا ذخیره شود.عملوند اول بیت‌های 16 ام تا 9 ام هر دستور را به خود اختصاص میدهد.

عملوند دوم:

عملوند دوم هر دستور، در همه ی دستورات(به جز دستور shift) میتواند شناسه ی یک رجیسترو یا یک عدد 8 بیتی با علامت باشد. عملوند دوم بیت‌های 8 ام تا 1 ام را به خود اختصاص میدهد.

شناسه ی رجیستر ها:

شناسه ی رجیستر (regId)	نام رجیستر
00000001	Ra
00000010	Rb
00000100	Rc
00001000	Rd

لیست دستورات:

عملیات دستور	فرمت دستور	دستور
ذخیره ی مقدار op در رجیستر	regId_operand_0001	Load reg, op
ذخیره ی مقدار رجیستر2 در رجیستر1	regId1_regId2_0010	Load reg1, reg2
جمع کردن مقدار رجیستر با مقدار op و ذخیره حاصل در رجیستر	regId_operand_0011	Add reg, op
جمع کردن مقدار رجیستر1 با رجیستر 2 و ذخیره ی حاصل در رجیستر 1	regId1_regId2_0100	Add reg1, reg2
کم کردن مقدار op از رجیستر و قرار دادن حاصل در رجیستر	regId_operand_0101	Sub reg, op
کم کردن مقدار رجیستر 2 از رجیستر 1 و قرار دادن حاصل در رجیستر1	regId1_regId2_0110	Sub reg1, reg2
ضرب کردن مقدار رجیستر در مقدار op و ذخیره ی هشت بیت پر ارزش در Rd و ذخیره ی هشت بیت کم ارزش در Ra	regId_operand_0111	Mult reg, op
ضرب کردن مقدار رجیستر 1 در مقدار رجیستر 2 و ذخیره ی هشت بیت پر ارزش در Rd و ذخیره ی هشت بیت کم ارزش در Ra	regId1_regId2_1000	Mult reg1, reg2
شیفت به راست مقدار رجیستر به اندازه مقدار op	regId_operand_1011	Shr reg, op
شیفت به چپ مقدار رجیستر به اندازه مقدار op	regId_operand_1100	Shl reg, op

مثال‌هایی از دستورات:

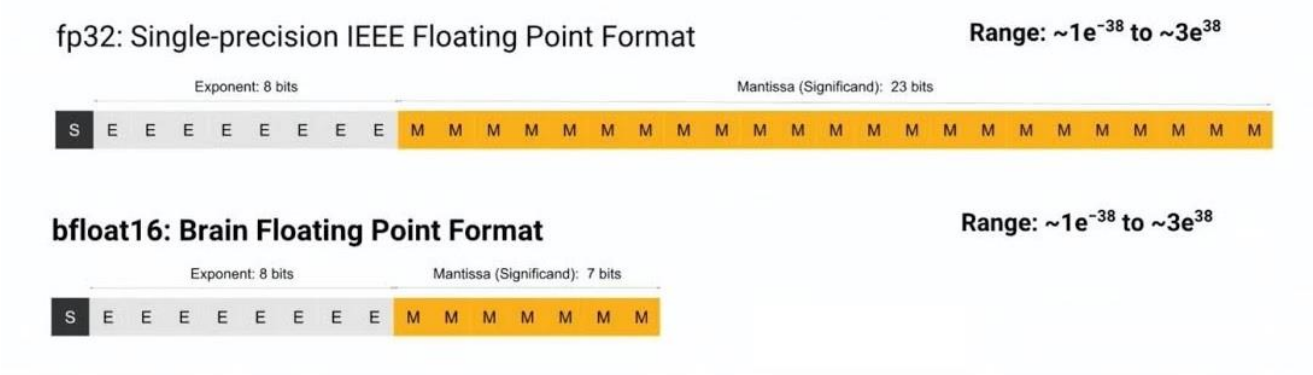
دستور به صورت کد شده	دستور	عملیات
00000010_00000001_0001 00001111_00000001_0011	Load Ra, 2 Add Ra, 15	جمع دو عدد 2 و 15

00001000_00000010_0001
 	
00000100_00000100_0001
00000100_00000010_0110
00000101_00000010_0111
	Load Rb, 8
	Load Rc, 4
	Sub Rb, Rc
	Mult Rb, 5
		تفریق 4 از 8 و سپس ضرب آن در 5

00000111_00000001_0001 00000011_00000001_1100	Load Ra, 7 Shl Ra, 3	لود کردن عدد 7 در رجیستر Ra و شیفت آن به چپ به اندازه ی 3 واحد

فاز دوم:

یکی از ویژگی‌های مهم پردازنده ها، انجام محاسبات اعشاری است. اولین گام در انجام محاسبات اعشاری در نظر گرفتن استانداردی برای نمایش این اعداد است. از استاندارد های معروف میتوان به IEEE754، که در سه حالت 16 بیتی، 32 بیتی و 64 بیتی قابل استفاده است اشاره کرد. این استاندارد دارای تقریب دقیقی است اما همین تقریب دقیق انجام محاسبات را کند میکند و موجب می‌شود تراشه هایی که برای اجرای الگوریتم های یادگیری ماشین و شبکه‌های عصبی استفاده می‌شوند در اجرا با کندی مواجه شوند. برای حل این مشکل گوگل استاندارد جدیدی تحت عنوان bfloat 16 را معرفی کرد که مشابه استاندارد IEEE754 single-precision است اما با این تفاوت که به جای 23 بیت، تنها 7 بیت mantissa را نگه داری میکند. به این ترتیب هم بازه ی گسترده ای از اعداد را پوشش میدهد و هم دقت مناسبی ارائه میدهد. در شکل زیر مقایسه ی این دو استاندارد رامشاهده میکنید.



در این فاز از پروژه باید یک دستور جدید به نام bfloatConvert را در پردازنده پیاده‌سازی کنید که دو عملوند هشت بیتی میگیرد، عملوند اول قسمت صحیح عدد میباشد و با علامت در نظر گرفته میشود و عملوند دوم قسمت اعشاری عدد میباشد و بی علامت در نظر گرفته میشود. پردازنده باید بتواند sign، exponent و mantissa را محاسبه کند، .سپس exponent را در رجیسترRc ذخیره کند و sign و mantissa را به ترتیب در رجیستر Rb ذخیره کند و عدد تولید شده را نیز در خروجی قرار دهد.

توضیحات:

- دستور: bfloatConvert whole, fraction
- کد دستور: 1101
- فرمت دستور: قسمت اعشاری_قسمت صحیح_1101

مثال: نمایش عدد 5.2

دستور ورودی به پردازنده: 1101_00000101_00000010

خروجی: 0100000010100110