دانشگاه صنعتی شریف

دانشکده مهندسی برق

درس ساختار کامپیوتر و ریزپردازنده و آزمایشگاه

(۲۵-۷۵۴)

آزمایش شماره ۲

جمع کننده ممیز شناور

نیمسال دوم ۹۴-۹۳

تهیه کنندگان:

مهندس سیّد سینا دزفولی

دکتر محمدرضا موحّدین

به نـــــام خـــــــدا

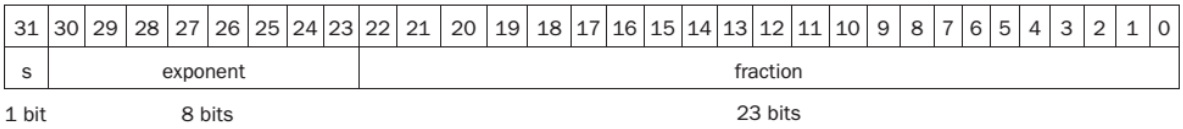
پیاده‌سازی و شبیه‌سازی جمع کننده اعداد ممیز شناور

هدف از این آزمایش پیاده‌سازی یک جمع‌کننده اعداد ممیز شناور[۱] دقت عادی[۲] بر طبق استاندارد IEEE-754 است. این جمع‌کننده بایستی پس از طراحی، توسط نرم‌افزار ModelSim شبیه‌سازی شود تا حداکثر اطمینان از صحّت عملکرد سخت‌افزار توصیف شده به دست آید.

پیش از آزمایشگاه:

الف) مطالب مربوط به موضوع آزمایش را به صورت دقیق مرور کنید.

ب) جهت یادآوری، نمایش اعداد در فرمت IEEE-754 و مفهوم فیلدهای آن از کتاب مرجع در اینجا تکرار شده است:



Sign	Exponent	Fraction	Representation
S	0	Zero	± 0 [۳]
S	0	Non-Zero	$(-1)^S \times 0.F \times 2^{-126}$
S	1-254	Anything	$(-1)^S \times 1.F \times 2^{E-127}$
S	255	Zero	± ∞
S	255	Non-Zero	NaN (Not a Number)

مشخصات طرح:

در این آزمایش لازم است مدول fp_adder که دارای دو ورودی ۳۲ بیتی a و b و یک خروجی ۳۲ بیتی s است، طراحی شود. ورودی‌ها و خروجی همگی با فرمت اعداد ممیز شناور دقت عادی بر طبق استاندارد IEEE-754 هستند. خروجی، حاصل جمع دو عدد ورودی است. با توجه به فرمت عمومی اعداد، ورودی‌ها ممکن است مختلف العلامه باشند و لذا عملیات اصلی قابل انجام بر روی آنها به جای جمع، منجر به تفریق گردد.

جهت ساده‌سازی و تمرکز بر جزئیات عملیات جمع، ورودی‌ها و خروجی هیچگاه به شکل ∞ ± و یا NaN نخواهند بود، یعنی بخش توان آنها هیچگاه برابر با ۲۵۵ نیست. از طرف دیگر، هم ورودی‌ها و هم خروجی ممکن است به شکل normalized و یا denormalized باشند، یعنی بخش توان E ممکن است برابر با صفر باشد.

این پیاده‌سازی بایستی منجر به یک مدار کاملا ترکیبی شود و توصیف کلیه مدارها فقط و فقط به کمک عبارت‌های continuous assignment در زبان وریلاگ انجام پذیرد. این بدین معنا است استفاده از دیگر قابلیت‌های زبان در این آزمایش مجاز نیست. در اتخاذ این تصمیم، بعضی اهداف آموزشی نقش ایفا کرده است.

---

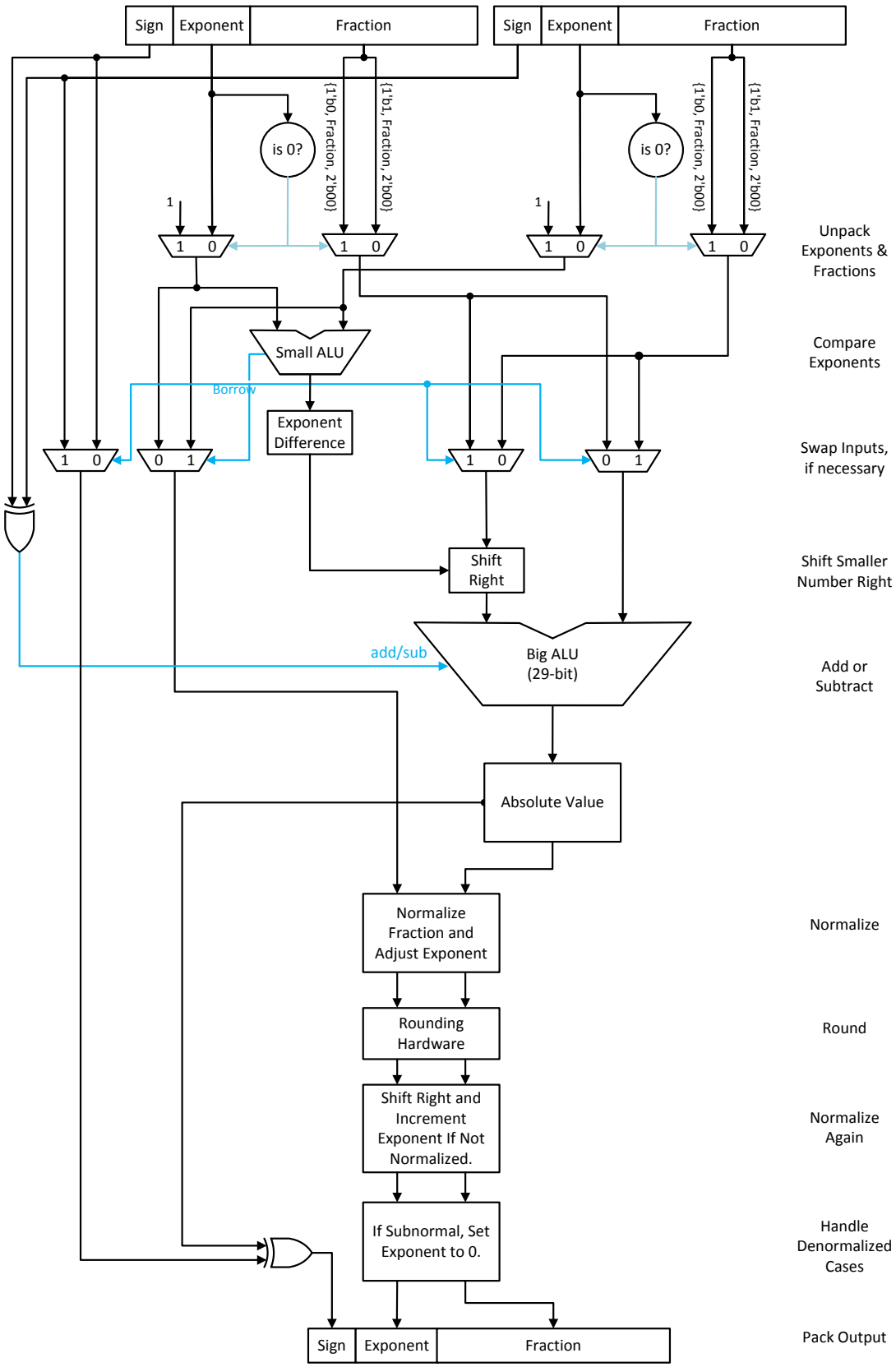
[۱] Floating Point

[۲] Single Precision

[۳] این سطر از جدول حالت خاصّی از سطر بعدی آن است.

الگوریتم جمع اعداد ممیز شناور:

بلوک دیاگرام عملیات جمع اعداد ممیز شناور در زیر آورده شده است:



مراحل انجام عملیات جمع به اختصار به شرح ذیل است. این مراحل بایستی در این آزمایش به صورت ترکیبی به وسیله یک یا چند عبارت continuous assignment در زبان وریلاگ توصیف شوند. البته به دلیل تمرکز بر روی ماهیت عمل جمع اعداد ممیز شناور و محدود کردن مدار به یک مدار صددرصد ترکیبی، ممکن است در بخش‌هایی، مدار به صورت کاملاً بهینه طراحی نشده باشد که در این آزمایش چشم‌پوشی می‌گردد.

۱- در ابتدا لازم است بخش‌های **F**raction و **E**xponent عدد بازسازی شود. از یک طرف، بایستی بیت مخفی[۴] استخراج شود. برای این منظور، ابتدا صفر بودن E را بررسی می‌کنیم. اگر E صفر باشد، بیت مخفی در بخش بالایی F صفر است و در غیر این صورت یک.

۲- از طرف دیگر، اگر E = 0 باشد، در حوزه اعداد با بایاس ۱۲۷ (که بصورت معمول در اعداد normalized مورد استفاده قرار گرفته است)، هم ارز E = 1 است. یادآوری می‌شود در حالت E = 0 ارزش واقعی توان برابر ۱۲۶- است که هم ارز E = 1 یعنی ۱۲۷–۱ است. نتیجه این بخش مانند بند قبل به کنترل صفر بودن E وابسته است.
تذکر: کلیه عملیات روی توان E، اگر چه یک عدد با بایاس ۱۲۷ است، همانند یک عدد بدون علامت صورت می‌پذیرد. توجه داشته باشید بایاس بودن عدد در نحوه انجام عملیات جمع و تفریق آن اثری نمی‌گذارد و صرفاً هنگام ضرب، باید به آن توجه شود.

۳- در همین ابتدا بد نیست جایی برای بیت‌های **G**uard و **R**ound را پیش‌بینی کنیم و دو صفر در سمت راست (LSB) هرکدام از دو F قرار دهیم.

۴- اکنون بایستی عدد با E بزرگ‌تر را در سمت راست جمع‌کننده‌ی بزرگ و دیگری را در سمت چپ قرار داد. بنابر این به یک مقایسه‌کننده هشت بیتی برای مقایسه توان‌ها نیاز داریم. از طرف دیگر لازم است عدد با توان کوچکتر را به اندازه قدر مطلق تفاضل دو توان به سمت راست شیفت داد که برای استخراج مقدار شیفت به یک تفریق کننده هشت بیتی نیاز داریم. این دو عملیات می‌توانند با هم ترکیب شده و در یک تفریق‌کننده انجام پذیرد. بیت borrow خروجی تفریق‌کننده بیانگر کوچک‌تر یا بزرگ‌تر بودن توان‌ها است. برای استخراج مقدار شیفت، خروجی تفریق‌کننده ممکن است لازم باشد که قرینه شود (چرا و در چه شرایطی؟)

تذکر: کلیه تلاش‌های فوق برای قرار دادن عدد با توان بزرگ‌تر در سمت راست جمع کننده‌ی بزرگ و عدد با توان کوچکتر در سمت چپ آن است. در حالت عدم تساوی توان‌های دو عدد، می‌توان مطمئن بود ارزش عدد سمت راست بیشتر از عدد سمت چپ است و نتیجه گرفت که حاصل جمع نیز با عدد بزرگ‌تر هم‌علامت است. با این حال، زمانی‌که توان دو عدد با هم برابر است، اطمینان فوق برقرار نیست، زیرا قسمت F در زمان جابجایی اعداد با هم مقایسه نشده‌اند. در این‌صورت نتیجه محاسبات نیاز به توجّه بیشتری دارد. مثلاً حاصل جمع دو عدد مختلف‌العلامه با توان‌های مساوی و F بسیار نزدیک به هم، می‌تواند منجر به عدد بسیار کوچک و چه بسا denormalized گردد.

۵- اکنون زمان آن رسیده است عدد با توان کوچک‌تر، که در سمت چپ جمع‌کننده‌ی بزرگ قرار گرفته است، به مقداری که در مرحله قبل حساب شده، به سمت راست شیفت داده شود. دو بیت از بیت‌هایی که به سمت راست شیفت داده می‌شوند، به دلیل پیش‌بینی دو بیت اضافه در سمت راست عدد که در مرحله ۳ انجام شد، دور ریخته نمی‌شوند و در G و R حفظ می‌شوند. ولی یک نماینده تک بیتی از حاصل OR شده بقیه بیت‌های دور ریخته شده، ناشی از شیفت عدد به سمت راست، باید به عنوان بیت **S**ticky استخراج و نگهداری گردد. بیت S به LSB عدد سمت راست و متناظر آن یک تک بیت صفر به LSB عدد سمت چپ (عدد با توان بزرگ‌تر) اضافه می‌گردد.

---

[۴] Hidden Bit

۶- اکنون اعداد سمت راست و چپ آماده جمع یا تفریق هستند. تصمیم در مورد انجام عمل جمع یا تفریق به علامت هر یک از اعداد و جابجایی احتمالی آن‌ها و همچنین علامت عدد با توان بزرگ‌تر دارد. تعداد بیت جمع یا تفریق‌کننده نیز بایستی با دقت انتخاب شود. یادآوری می‌گردد تاکنون: F به تنهایی ۲۳ بیت، یک بیت مخفی، سه بیت GRS شناسایی شده‌اند. عملیات جمع یا تفریق در حوزه اعداد علامت‌دار انجام شده و نبایستی سرریز (overflow) داشته باشد. اکنون سؤال این است: تعداد بیت‌های جمع و تفریق‌کننده چند تا است؟

۷- از آنجایی‌که خروجی نهایی در فرمت IEEE-754 به‌صورت عدد-علامت[۵] است، اگر خروجی جمع‌کننده بزرگ منفی باشد، بایستی به مثبت تبدیل شده و علامت نتیجه نیز اصلاح گردد. سؤال: این اتفاق در چه صورتی رخ می‌دهد؟

۸- اکنون بایستی حاصل عملیات فوق نرمالیزه گردد و ترجیحاً به شکل 1.F درآید. البته در این میان، اگر توان عدد به مقدار یک کاهش یابد (یک در حالت بایاس و ۱۲۶- در حالت عادی)، این به معنای عدم امکان نرمالیزه کردن بیش از آن و denormalized بودن عدد نهایی بوده و می‌توان به شکل 0.F اکتفا نمود. البته در این‌صورت، توان E به صفر تبدیل می‌شود. برای نرمالیزه کردن بایستی توجه داشت که خروجی جمع‌کننده ممکن است یکی از سه حالت زیر را داشته باشد.

الف- اصلاً نیاز به شیفت نداشته باشد،

ب- نیاز به یک بیت شیفت به سمت راست داشته باشد،

ج- تعداد دلخواهی شیفت به چپ برای آن لازم باشد. (حداکثر این تعداد چند بیت است؟)

جهت ساده‌سازی عملیات، می‌توان نقطه ممیز را موقتاً یک بیت به سمت چپ منتقل کرد (و البته این شیفت را در توان عدد جبران نمود) و تصمیم گرفت حاصل خروجی جمع‌کننده بزرگ به تعداد صفر یا بیشتر نیاز به شیفت به سمت چپ دارد. برای این منظور بایستی اولین یک در بالاترین رتبه را پیدا کرد: یعنی پیدا کردن یکی که همه بیت‌های بالاتر از آن صفر باشند. به این کار Leading One Detection گفته می‌شود.

۹- مقدار شیفت به دست آمده در بالا، اگر چه می‌تواند شکل عدد را به‌صورت 1.F تبدیل نماید ولی دارای این خطر است که توان عدد را بیش از اندازه کاهش داده و آن را کمتر از ۱۲۶- (E = 0)، که کوچکترین توان قابل نمایش است، بنماید. پس در واقع، مقدار شیفت بالا بایستی با توان عدد خروجی مقایسه گردد که اگر از آن بیشتر است، مقدار شیفت به اندازه مقدار توان محدود گردد. در این حالت، عدد به‌صورت denormalized در خواهد آمد.

۱۰- اکنون زمان گردکردن[۶] عدد رسیده است: نقطه گردکردن را در نظر گرفته و طبق استاندارد IEEE و طبق روش round to nearest even عدد را گرد کنید.

۱۱- پس از گردکردن، عدد ممکن است نیاز به نرمالیزه کردن مجدد داشته باشدکه با یک بیت شیفت به سمت راست انجام می‌پذیرد. در حالت بسیار خاص، این مرحله به تنهایی می‌تواند منجر به تبدیل عدد به $\pm \infty$ گردد که از موضوع این آزمایش خارج است.

خسته نباشیـــــد: اکنون نتیجه بخش‌های مختلف عدد حاصل‌جمع آماده است. شما می‌توانید با کنار هم گذاشتن آن‌ها، عدد نهایی را ساخته و در خروجی قرار دهید: {S, E, F} ☺

---

[۵] Sign-Magnitude

[۶] Rounding

تست طرح:

با استفاده از نرم‌افزار ModelSim، طراحی خود را شبیه‌سازی کنید. برای تست کد خود، می‌توانید علاوه بر تست‌بنچی که در اختیار شما قرار گرفته است، از بردارهای تست زیر در تست‌بنچی که خودتان می‌نویسید نیز استفاده نمایید:

```
32'h440d491c + 32'h4d064db7 = 32'h4d064dda
32'h366b66c4 + 32'h42307eb7 = 32'h42307eb8
32'h12e1798b + 32'h121f73da = 32'h131899bc
32'h575360bf + 32'h5c673cd6 = 32'h5c6771ae
32'h422d54dc + 32'h368e0d66 = 32'h422d54dd
32'h49f7442b + 32'h50781481 = 32'h50781c3b
32'h18502b00 + 32'h16d47f61 = 32'h186abaec
32'h4d675968 + 32'h4ad42cf7 = 32'h4d6dfad0
32'h5d240588 + 32'h55797cfe = 32'h5d240681
32'h285248db + 32'h27251643 = 32'h287b8e6c
32'h01925662 + 32'h81b81010 = 32'h8096e6b8
32'h00012832 + 32'h0014283c = 32'h0015506e
32'h00012832 + 32'h8014283c = 32'h8013000a
32'h00b627be + 32'h000a21a8 = 32'h00c04966
32'h00b627be + 32'h800a21a8 = 32'h00ac0616
32'h02682174 + 32'h826f0850 = 32'h803736e0
32'h00d47943 + 32'h80c67efc = 32'h000dfa47
32'h440d491c + 32'h00000000 = 32'h440d491c
32'h00004002 + 32'h00000002 = 32'h00004004
32'h3F800001 + 32'hBF800001 = 32'h00000000
32'h3F800001 + 32'hBF800000 = 32'h34000000
32'h40000000 + 32'h34000000 = 32'h40000000
32'h40000000 + 32'h34000001 = 32'h40000001
32'h3fffffff + 32'h34000000 = 32'h40000000
32'h440d491c + 32'h40000000 = 32'h440dc91c
32'h407fffff + 32'h347fffff = 32'h40800000
32'h407fffff + 32'h34000000 = 32'h40800000
32'h407fffff + 32'h34400000 = 32'h40800000
```