فصل ۴

طراحی واحد محاسبه و منطق (ALU)

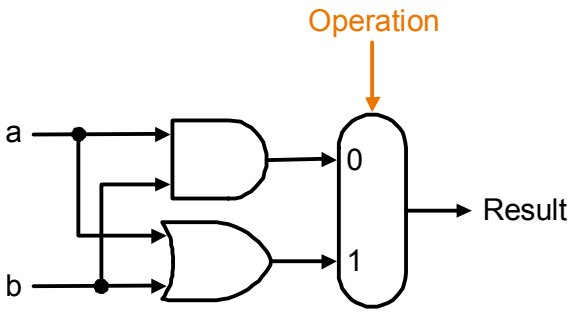
ALU یکی از مهمترین قسمتهای یک کامپیوتر و واحدی است که همهٔ عملیات محاسباتی مانند جمع، تفریق و... را انجام می‌دهد. در این بخش به کمک گیت‌های پایه AND، OR، NOT و مالتی پلکسر یک واحد ALU را طراحی خواهیم نمود. از آنجا که پردازنده ما ۳۲ بیتی است، ما به یک ALU ۳۲ بیتی نیاز خواهیم داشت. در این قسمت ابتدا یک ALU یک بیتی را طراحی نموده و سپس ۳۲ عدد از این ALU های یک بیتی را به هم وصل نموده و یک ALU ۳۲ بیتی طراحی خواهیم نمود. بنابراین ابتدا طراحی یک ALU یک بیتی را توضیح می‌دهیم.

لازم به ذکر است که سیستم‌های مختلفی برای نمایش اعداد به صورت باینری وجود دارد که از جمله معروفترین آنها می‌توان به سیستم اندازه علامت[1]، سیستم متمم ۱[2] و سیستم متمم ۲[3] اشاره نمود. محاسبات در سیستم متمم ۲ ساده‌تر از دو سیستم دیگر بوده و طراحی سخت افزار آن نیز ساده‌تر است به همین دلیل در این فصل به صورت پیش‌فرض، توضیحاتی که ارائه می‌شود برای سیستم متمم ۲ است مگر اینکه صریحاً سیستم دیگری را ذکر کنیم.

۴-۱- ALU یک بیتی

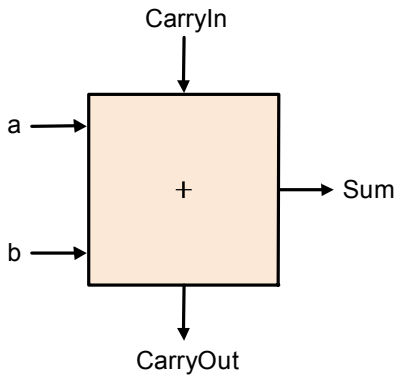
شکل ۱ یک واحد منطقی که دو عملیات AND و OR را پیاده‌سازی می‌کند را نشان می‌دهد. مالتی پلکسر استفاده شده، بسته به مقدار ورودی Operation یکی از عملیات a AND b یا a OR b را انتخاب می‌کند. در واقع خط ورودی Operation، مالتی‌پلکسر را کنترل نموده و یکی از ورودی‌های خط داده[4] مالتی‌پلکسر را انتخاب می‌نماید. توجه کنید که ما از قصد اسامیOperation و result را به ترتیب برای خط انتخاب و خروجی مالتی‌پلکسر به کار بردیم چون operation نوع عملیات ALU را تعیین می‌کند و نتیجه عملیات ALU نیز بر روی خروجی result ظاهر می‌شود.

[1] - Sign magnitude
[2] - On's complement
[3] - Two's complement
[4] - Data



شکل ۱: یک ALU ساده با دو عملیات AND و OR

عملیات بعدی که می‌خواهیم به ALU خود اضافه کنیم عملیات جمع کردن[1] است. همان‌طور که می‌دانیم یک جمع‌کننده کامل[2] دارای ۳ ورودی و ۲ خروجی می‌باشد. سه ورودی آن a، b و CarryIn و دو خروجی آن Sum، CarryOut می‌باشد. بلوک دیاگرام یک جمع کننده یک بیتی در شکل ۲ نشان داده شده است.



شکل ۲: بلاک دیاگرام یک جمع کننده یک بیتی

اگر جدول درستی یک جمع‌کننده کامل را رسم کنیم و مقدار خروجی‌ها را بر اساس ورودی‌ها مشخص کنیم و سپس با استفاده از جدول کارنو معادله خروجی‌ها را به دست آوریم، چنین بدست می‌آید:

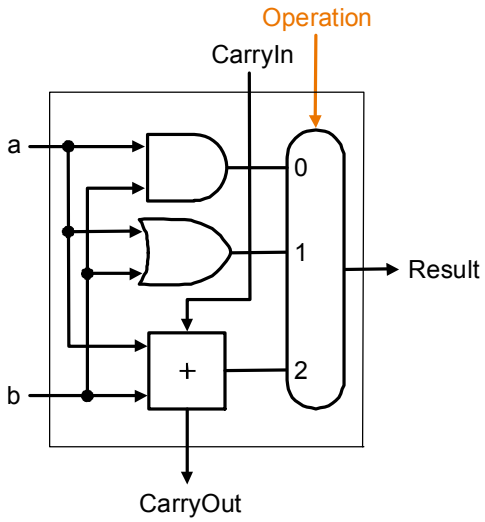
$$CarryOut = a.b + a.CarryIn + b.CarryIn$$

$$sum = \bar{a}.\bar{b}.\overline{CarryIn} + \bar{a}.b.\overline{CarryIn} + \bar{a}.\bar{b}.CarryIn + a.b.CarryIn$$

[1] - Add
[2] - Full Adder

حال مدار جمع کننده را به ALU یک بیتی خود اضافه می‌کنیم، شکل ۳ چگونگی انجام این کار را نشان داده است. در این شکل ALU قادر است که سه عملیات مختلف را بر طبق اینکه ورودی Operation چه مقداری داشته باشد انجام دهد:

- اگر Operation=0 باشد عملیات AND انجام می‌شود.
- اگر Operation=1 باشد عملیات OR انجام می‌شود.
- اگر Operation=2 باشد عملیات Add انجام می‌شود.



شکل ۳: یک ALU ساده با ۳ عملیات AND، OR، و جمع کردن

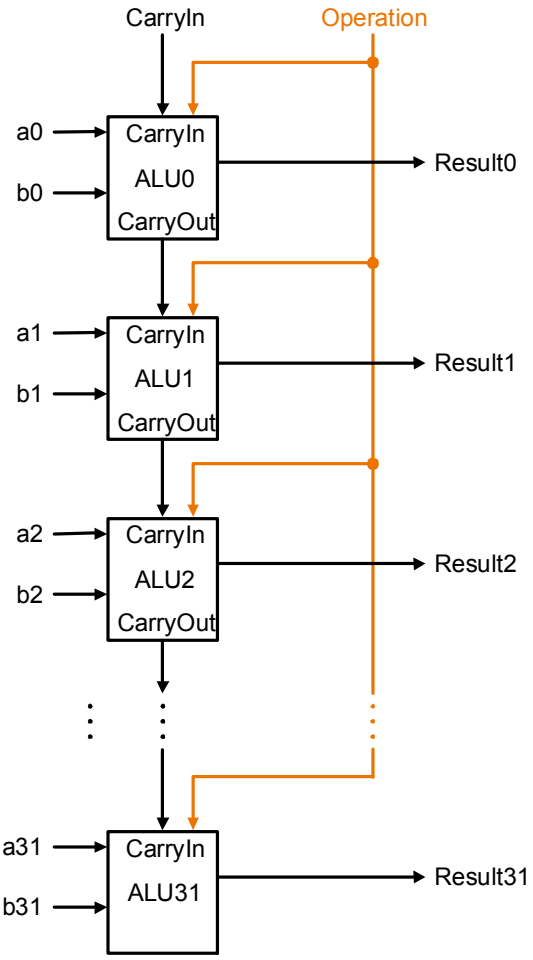
بعضی مواقع طراحان می‌خواهند که ALU عملیات بیشتری را انجام دهد مثلاً خروجی 0 نیز تولید کند. ساده‌ترین راه برای اضافه‌کردن یک عملیات به ALU این است که تعداد ورودی‌های مالتی‌پلکسر را زیاد کنیم و عملیات جدید را به خطوط ورودی جدید وصل کنیم مثلاً برای تولید صفر توسط ALU کافی است که به یکی از ورودی‌های ALU مقدار 0 را وصل کنیم.

۴-۲- طراحی یک ALU ۳۲ بیتی

حال که ما ALU یک بیتی را طراحی کردیم، می‌توانیم آن را به صورت یک جعبه سیاه[1] در طراحی یک ALU ۳۲ بیتی استفاده کنیم. ALU ۳۲ بیتی از اتصال ۳۲ عدد ALU یک بیتی به صورت

[1] - Black Box

شکل ۴ ایجاد می‌شود در این شکل منظور از ai یعنی بیت i ام عدد ۳۲ بیتی a مثلاً $a2$ یعنی بیت دوم عدد ۳۲ بیتی a.



شکل ۴: **طرح یک ALU ۳۲ بیتی ساده که از اتصال ۳۲ ALU یک بیتی ساده تشکیل شده است**

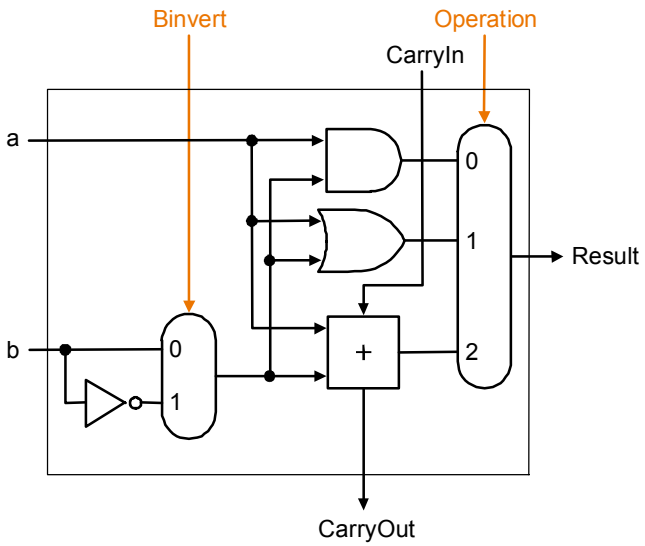
همان طور که در این شکل دیده می‌شود خروجی نقلی حاصل از طبقهٔ اول (CarryOut) به ورودی نقلی طبقهٔ دوم (CarryIn) و خروجی نقلی طبقهٔ دوم به ورودی نقلی طبقهٔ سوم وصل شده است و به همین ترتیب این کار تا طبقهٔ آخر یعنی طبقهٔ ۳۱ انجام می‌شود و سرانجام از خروجی نقلی آن خارج شود. این کار همانند ایجاد یک موج توسط سنگی است که در آب یک برکه ساکت و آرام انداخته می‌شود چون موج ایجاد شده توسط این سنگ از محل سنگ منتشر می‌شود تا اینکه به کنار برکه برسد. در مدار ALU ۳۲ بیتی نیز رقم نقلی از طبقه اول تا طبقه آخر به صورت موج منتشر می‌شود

و در نهایت از آن خارج می‌شود. به همین دلیل به جمع‌کننده‌ای که از اتصال مستقیم رقم‌های نقلی جمع کننده‌های یک بیتی ایجاد می‌شود جمع کننده موج گونه[1] گفته می‌شود.

تفریق کردن همانند جمع کردن یک عدد با مقدار منفی یک عدد دیگر در سیستم مکمل۲[2] می‌باشد. همان‌طور که می‌دانیم برای بدست آوردن مقدار منفی یک عدد در این سیستم کافی است که همهٔ بیتها را معکوس نموده (این عملیات مکمل ۱ کردن نام دارد) و سپس مقدار یک را به آن اضافه کنیم. بنابراین تفریق دو عدد به صورت زیر بدست می‌آید:

$$a - b = a + (-b) = a + (\overline{b} + 1)$$

بنابراین برای انجام دادن عملیات تفریق لازم است که بتوانیم تک تک بیت‌های عدد b را معکوس کنیم. در نتیجه هر کدام از ALU های یک بیتی استفاده شده باید غیر از عملیات AND، OR، و جمع کردن، باید بتوانند بیت b را نیز معکوس کنند. ALU یک بیتی جدید که عملیات معکوس کردن را نیز انجام می‌دهد در شکل ۵ نشان داده شده است. همان طور که در این شکل دیده می‌شود برای معکوس کردن هر بیت ما نیاز به یک مالتی پلکسر ۲ به ۱ داریم که بین b و $\overline{b}$ یکی را انتخاب کند (عمل انتخاب توسط بیت Binvert انجام می‌شود).



شکل ۵: **یک ALU یک بیتی ساده که عملیات معکوس کردن را نیز پشتیبانی می‌کند**

[1] - Ripple carry adder
[2] - Two's complement system

حال بیائید ۳۲ عدد از این ALU های جدید یک بیتی را به هم متصل کرده و یک ALU ۳۲ بیتی بسازیم. مالتی‌پلکسر ۲ به ۱ اضافه شده بسته به مقدار ورودی binvert مقدار b و یا معکوس شده آن را در اختیار قرار می‌دهد. حال که ما در این ۳۲ طبقه معکوس بیت‌های b را در اختیار داریم برای انجام عملیات تفریق a-b کافی است که a را با معکوس عدد b جمع کرده و یک ۱ به حاصل جمع اضافه کنیم. توجه کنید که ورودی نقلی اولین طبق برای عملیات جمع صفر قرار داده می‌شود. اگر ما این ورودی نقلی را ۱ قرار دهیم چه اتفاقی می‌افتد؟ در این صورت اگر مالتی‌پلکسرهای ۲ به ۱ ، b را انتخاب کنند عملیات a+b+1 و اگر $\bar{b}$ را انتخاب کنند عملیات $a+\bar{b}+1$ انجام خواهد گرفت که همان عملیات تفریق است:

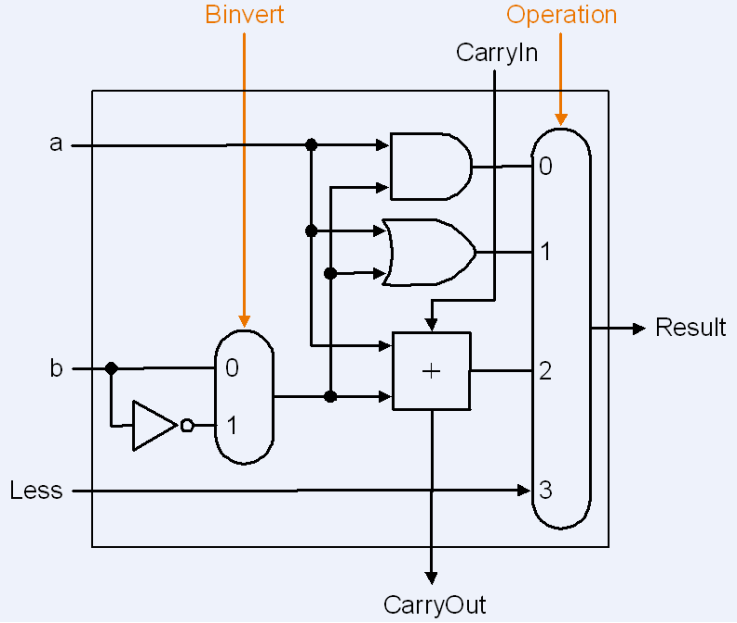
$$a+\bar{b}+1 = a+(\bar{b}+1) = a+(-b) = a-b$$

نکته: به دلیل ساده‌تر بودن طراحی سخت‌افزار برای مدار جمع‌کننده در سیستم متمم ۲، در محاسبات اعداد صحیح کامپیوترها به جای سیستم‌های دیگر از این سیستم استفاده می‌شود.

۴-۳- طراحی ALU ۳۲ بیتی برای پردازنده MIPS

مجموعه عملیات add، Subtract، And و OR تقریباً در هر کامپیوتری وجود دارند. اکثر دستورات پردازنده MIPS توسط ALU طراحی شده در بخش قبلی قابل انجام هستند ولی باید توجه نمود که طراحی ALU برای MIPS هنوز کامل نشده است. یکی از دستورات MIPS که لازم است پشتیبانی شود، دستور slt[1] می‌باشد. به خاطر بیاورید که این دستور اگر $rs < rt$ بود، مقدار یک و در غیر این صورت مقدار صفر را تولید می‌کرد. بنابراین دستور slt همهٔ بیتهای خروجی ALU را غیر از بیت پایین آن صفر خواهد کرد و بیت پایین آن را بر اساس نتیجهٔ مقایسه یک یا صفر خواهد نمود. برای اینکه ALU دستور slt را انجام دهد نیاز به این داریم که در همه ۳۲ طبقه ALU، مالتی‌پلکسرهای ۳ ورودی را گسترش دهیم و یک ورودی برای دستور slt در نظر بگیریم. ما این ورودی جدید را Less نام‌گذاری کرده و آن را فقط برای slt استفاده می‌کنیم. شکل ۶ یک ALU یک بیتی را که مالتی‌پلکسر آن به ۴ ورودی گسترش یافته تا عملیات slt را پشتیبانی کند، نشان می‌دهد.

[1] - Set on less then



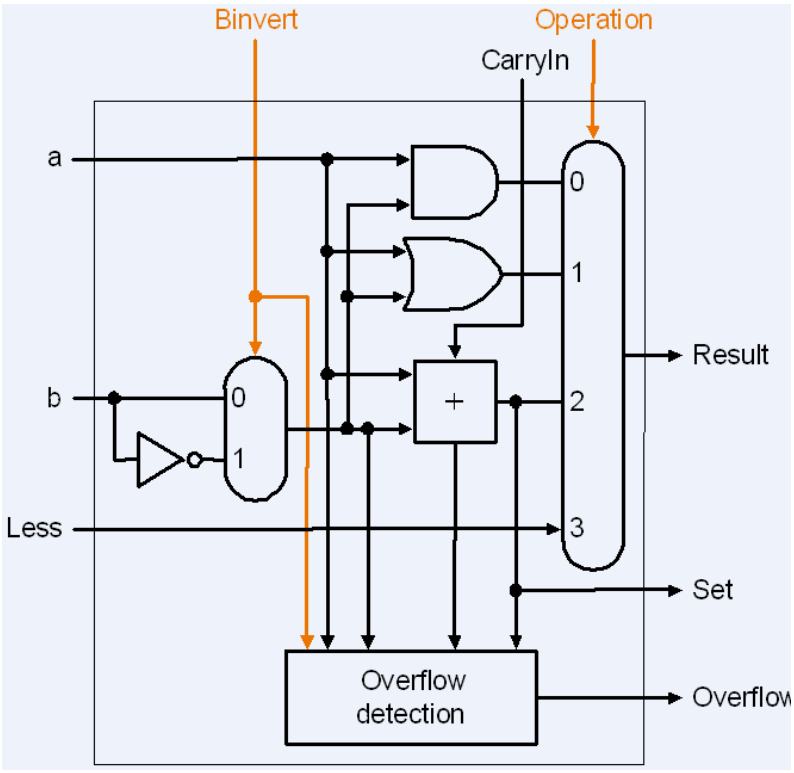
شکل ۶: **ALU بیتی** که یک عملیات دیگر (بر روی خطLess) به آن اضافه شده است

طبق توضیح بالا ما باید ورودی Less را برای ۳۱ طبقهٔ بالای ALU (۳۱ بیت بـا ارزش) مـساوی صفر قرار دهیم چون برای دستور slt، ۳۱ بیت بالای نتیجه ALU همیشه صفرند. چیزی که باقی می‌ماند این است که ما چگونه عمل مقایسه را انجام دهیم و بیت پایین ALU (بیت شماره صفر) را برای دستور slt به درستی مقداردهی کنیم. بیائید ببینیم که وقتی b را از a کم می‌کنیم چـه اتفـاقی مـی‌افتـد؟ اگـر حاصل تفریق منفی باشد در این صورت $a < b$ می‌باشد چون داریم:

$$a - b < 0 \Rightarrow a < b$$

ما می‌خواهیم که بیت پایین نتیجه ALU (بیت شماره صفر خروجی ALU) در صورتی که a<b باشد به ۱ و در غیر این صورت به صفر مقداردهی شود. یعنی اگر a-b منفی باشد این بیـت ۱ و در غیـر این صورت صفر شود. این نتیجه دقیقاً مطابق با مقدار بیت علامت نتیجه عملیات تفریق مـی‌باشـد چـون همان طور که می‌دانیم بیت علامت یک عـدد اگـر منفـی باشـد ۱ و در غیـر ایـن صـورت صـفر اسـت. بنابراین با این توضیح ما فقط کافی است که بیت علامت خروجی جمع‌کننـده (کـه بـرای slt عملیـات تفریق انجام می‌دهد) را به ورودی Less مالتی‌پلکسر طبقهٔ اول وصل کنیم. همان طور که می‌دانیم بیـت علامت یک عدد، بیت با ارزشتر یا همان بیت آخر (بیت شماره ۳۱ اگر شماره بیت‌ها را از صفر شـروع

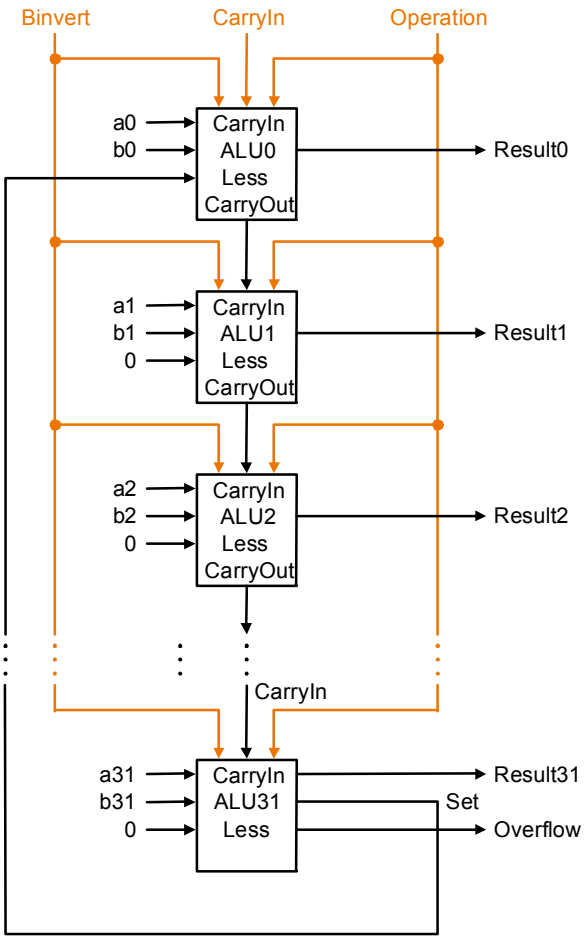
کنیم) آن عدد خواهد بود. بنابراین برای عملیات تفریق، بیت علامت نتیجه عملیات، بیت خروجی حاصل شده از جمع کننده طبقه آخر ALU (خروجی sum جمع کننده طبقه ۳۱) خواهد بود. اما متأسفانه بیت خروجی جمع کننده طبقه آخر به عنوان یک خروجی از ALU ما خارج نشده است. بنابراین ما برای طبقهٔ آخر ALU به یک ALU یک بیتی جدید نیاز خواهیم داشت که یک خروجی اضافه‌تر دارد که مستقیماً به خروجی جمع‌کننده وصل شده است. در شکل ۷ این ALU یک بیتی نشان داده شده است و خروجی اضافه شده در آن Set نام گرفته است که فقط برای دستور slt استفاده می‌شود.



شکل ۷: ALU یک بیتی مورد نیاز برای طبقه آخر ALU ۳۲ بیتی که عملیات slt را پشتیبانی می‌کند

همان طور که ما برای طبقهٔ آخر به یک ALU یک بیتی مخصوص نیاز داریم، مدار تشخیص دهنده سرریز نیز باید در این ALU یک بیتی مخصوص قرار داده شود چون امکان بروز سرریز فقط در طبقهٔ آخر چک می‌شود. سرریز برای یک ALU ۳۲ بیتی وقتی اتفاق می‌افتد که نتیجه حاصل از محاسبات، داخل ۳۲ بیت جا نشوند و بیشتر از ۳۲ بیت نیاز داشته باشند. می‌توان نشان داد که برای سیستم متمم۲ سرریز وقتی اتفاق می‌افتد که علامت عدد عوض می‌شود یعنی انتظار داریم که نتیجه

ALU یک عدد مثبت باشد ولی منفی می‌شود و یا اینکه انتظار داریم نتیجه ALU یک عدد منفی باشد ولی مثبت می‌شود. و باز می‌توان نشان داد که در سیستم متمم ۲ سرریز وقتی اتفاق می‌افتد که نقلی‌های خروجی از دو طبقهٔ آخر مقدارهای متفاوتی داشته باشند یعنی $Carryout_{30} \neq Carryout_{31}$ و این وضعیت را می‌توان با یک گیت xor تشخیص داد $V = Carryout_{30} \oplus carryout_{31}$. در این عبارت V وقتی یک می‌شود که $Carryout_{30} \neq Carryout_{31}$ باشد. همان طور که می‌دانیم $Carryout_{30}$ همان ورودی نقلی برای طبقهٔ ۳۱ یا همان طبقهٔ آخر می‌باشد. بنابراین اگر ما بخواهیم یک ALU ۳۲ بیتی بسازیم، در طبقهٔ آخر آن باید از این ALU یک بیتی مخصوص که یک خروجی set و یک خروجی Overflow دارد استفاده کنیم و در بقیه طبقات از ALU های یک بیتی ساده استفاده کنیم. شکل ۸ ALU ۳۲ بیتی طراحی شده برای پردازنده MIPS با امکان Overflow و پشتیبانی از دستور slt را نشان می‌دهد.



شکل ۸:ALU ۳۲ بیتی برای پردازنده MIPS

توجه کنید که ما هر وقت که بخواهیم ALU عملیات تفریق را انجام دهد، ورودی CarryIn طبقهٔ اول و همچنین ورودی Binvert را یک قرار می‌دهیم. برای عملیات جمع و عملیات منطقی ما می‌خواهیم که این دو ورودی مقدار صفر داشته باشند. بنابراین ما می‌توانیم برای کنترل ساده ALU این دو ورودی را در قالب یک ورودی ترکیب نموده و اسم آن را Bnegate قرار دهیم. در شکل ۸، این کار نیز انجام شده است.

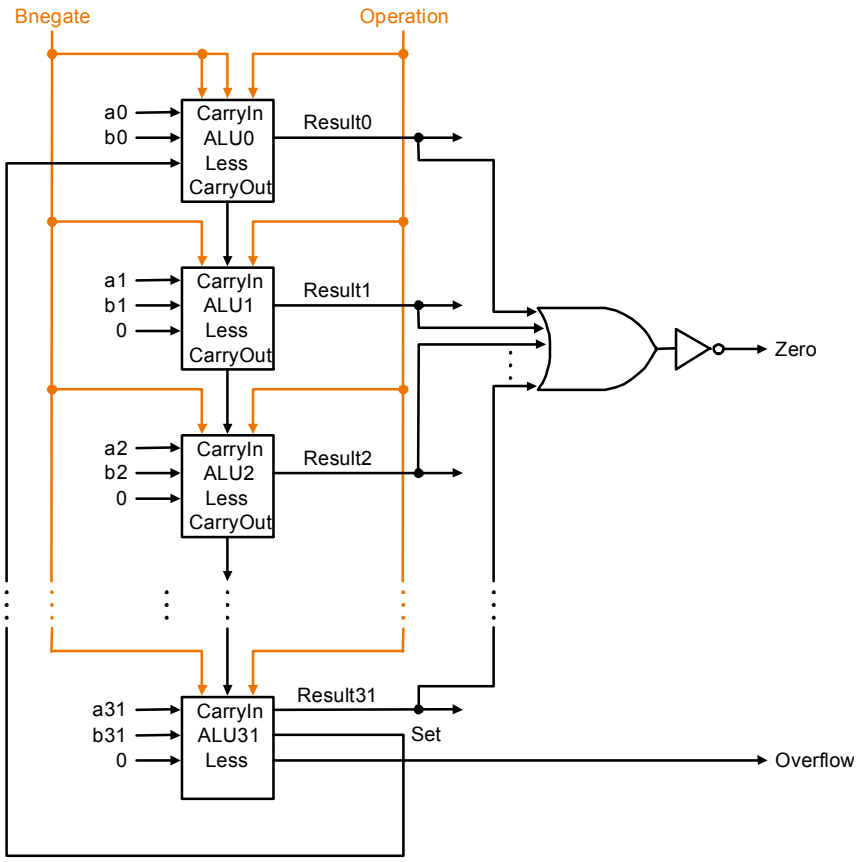
برای اینکه بتوانیم دستورات بیشتری از پردازنده MIPS را با این ALU انجام دهیم ما باید دستورات پرش شرطی را نیز به عملیات ALU اضافه کنیم. همان طور که می‌دانیم دستورات پرش شرطی به شرطی پرش را انجام می‌دهند که محتوای دو رجیستر مساوی باشند یا نباشند (در beq شرط مساوی بودن و در bne شرط نامساوی بودن چک می‌شود). ساده‌ترین روش برای تست مساوی بودن با ALU طراحی شده این است که b را از a کم کنیم و بعد بررسی کنیم که آیا نتیجه عملیات صفر می‌شود یا نه، به دلیل اینکه داریم:

$$a = b \Rightarrow a - b = 0$$

بنابراین اگر ما به ALU طراحی شده بخشی را اضافه کنیم که صفر شدن نتیجه را بررسی کند، می‌توانیم دستورهای beq و bne را پشتیبانی نمائیم. ساده‌ترین راه برای بررسی صفر شدن نتیجه این است که همهٔ بیت‌های خروجی را با هم OR نموده و بعد سیگنال بدست آمده را از یک گیت معکوس کننده عبور دهیم:

$$\text{Zero} = \overline{(\text{Result31} + \text{Result30} + \ldots + \text{Result2} + \text{Result1} + \text{Result0})}$$

شکل ۹ ALU ۳۲ بیتی پردازنده MIPS را که قابلیت انجام دستورات beq و bne نیز به آن اضافه شده است را نشان می‌دهد.



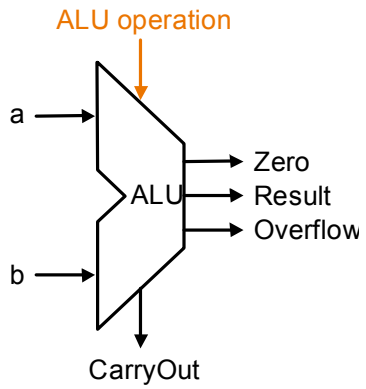
شکل ۹: ALU ۳۲ بیتی پردازنده MIPS که دستورات slt، beq، bne را پیاده سازی می‌کند و امکان تشخیص overflow را نیز دارد

ما می‌توانیم ۲ بیت کنترلی Operation و یک بیت کنترلی Bnegate را با هم ترکیب نموده و فرض کنیم که ALU ما در کل ۳ بیت کنترلی برای عملیات خود داشته باشد. ما با استفاده از این ۳ بیت کنترلی به ALU می‌گوئیم که عملیات add، Subtract، AND، OR و غیره را انجام دهد. جدول ۱ خطوط کنترلی ALU و عملیات انجام گرفته از طریق آنها را نشان می‌دهد.

جدول ۱: عملیاتی که توسط ALU طراحی شده با ۳ بیت کنترلی قابل انجام است

Operation	عملیات
000	And
001	Or
010	Add
110	Sub
111	Slt

در نهایت حال که ما فهمیدیم چه چیزی در داخل ALU ۳۲ بیتی قرار گرفته است، ما می‌توانیم این ALU را به صورت یک سمبل (Symbol) یا Box در آورده و در مراحل بعدی از آن استفاده نمائیم. شکل ۱۰ بلاک دیاگرام یا جعبه سیاه ALU ۳۲ بیتی نهایی طراحی شده برای پردازنده MIPS را نشان می‌دهد. طبق توضیحات قبلی، در این شکل ورودی‌های a و b ۳۲ بیتی، ورودیoperation ۳ بیتی، خروجی Result ۳۲ بیتی و بقیه خروجی‌ها یک بیتی خواهند بود. ما در طراحی پردازنده از این بلاک دیاگرام استفاده خواهیم کرد.



شکل ۱۰: بلاک دیاگرام ALU طراحی شده برای پردازنده MIPS

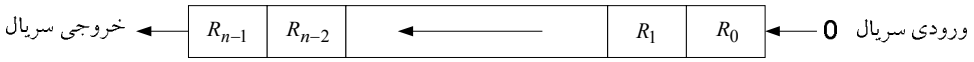
۴-۴- عملیات شیفت

عملیات شیفت[1] یا انتقال برای انتقال یا جابجایی سری داده‌ها به کار گرفته می‌شوند. آنها همچنین به همراه عملیات حسابی و منطقی دیگر مورد استفاده قرار می‌گیرند. محتوای یک ثبات می‌تواند به چپ یا به راست شیفت پیدا کند. در همان زمانی که بیت‌ها شیفت داده می‌شوند، اولین فلیپ فلاپ اطلاعات دودویی خود را از ورودی سری دریافت می‌کند. در حین شیفت به چپ، بیت ورودی سریال به سمت راست‌ترین مکان منتقل می‌شود و در حین عمل شیفت به راست، بیت ورودی سریال به سمت چپ‌ترین مکان منتقل می‌شود. اطلاعاتی که از طریق ورودی سری منتقل می‌گردد تعیین کننده نوع شیفت است. سه نوع شیفت وجود دارد: منطقی[2]، چرخشی[3] و حسابی[4].

[1] - Shift
[2] - Logical shift
[3] - Rotate shift
[4] - Arithmetic shift

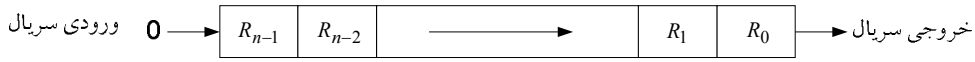
۴-۴-۱- شیفت منطقی

شیفت منطقی مقدار 0 را از طریق ورودی سری انتقال می‌دهد و به دو صورت شیفت منطقی به چپ و شیفت منطقی به راست صورت می‌گیرد. ما سمبل shl و shr را به ترتیب برای عملیات شیفت به چپ و شیفت به راست به کار خواهیم برد. عملیات شیفت منطقی به چپ به صورت شکل ۱۱ انجام می‌گیرد. به عبارتی در این عملیات، بیت R_0 به بیت R_1، بیت R_1 به بیت R_2 و به همین ترتیب هرکدام از بیت‌ها، به مکان بالاتر از خود (سمت چپ) منتقل می‌شوند. در حین این عملیات مقدار 0 وارد R_0 شده و مقدار قبلی R_{n-1} (سمت چپ‌ترین بیت) به بیرون منتقل شده و از بین می‌رود. به طور مثال اگر یک رجیستر ۸ بیتی را که دارای محتوای 00101101 می‌باشد یکبار به سمت چپ شیفت دهیم محتوای آن به صورت 01011010 خواهد شد.



شکل ۱۱: شیفت منطقی به چپ

شیفت منطقی به راست نیز به صورت شکل ۱۲ انجام می‌شود. همان طور که مشاهده می‌شود این عملیات دقیقاً همانند شیفت منطقی به چپ است با این تفاوت که جهت انتقال از چپ به راست بوده و مقدار 0 به سمت راست‌ترین بیت منتقل شده و مقدار قبلی بیت R_0 از بین خواهد رفت. به طور مثال اگر یک رجیستر ۸ بیتی را که دارای محتوای 00101101 می‌باشد یکبار به سمت راست شیفت دهیم محتوای آن به صورت 00010110 خواهد شد.

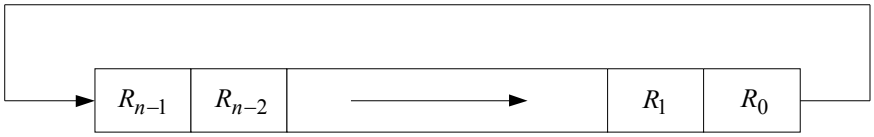


شکل ۱۲: شیفت منطقی به راست

۴-۴-۲- شیفت چرخشی

شیفت چرخشی (که عمل چرخش نیز خوانده می‌شود) بیت‌های ثبات را از طریق دو انتها بدون از دست دادن هر گونه اطلاعات می‌چرخاند. این عمل با اتصال خروجی سری به ورودی سری ثبات تحقق می‌یابد. عملیات شیفت چرخشی نیز به دو صورت انجام می‌شود: شیفت چرخشی به راست و شیفت چرخشی به چپ. عملیات شیفت چرخشی به راست در شکل ۱۳ نشان داده شده است. به طور

مثال اگر یک رجیستر ۸ بیتی را که دارای محتوای 00101101 می‌باشد یکبار به سمت راست شیفت چرخشی دهیم محتوای آن به صورت 10010110 خواهد شد.



شکل ۱۳: شیفت چرخشی به راست

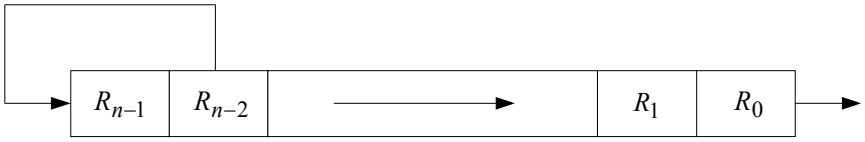
عملیات شیفت چرخشی به چپ نیز مشابه با عملیات شیفت چرخشی به راست است ولی جهت انتقال داده‌ها به سمت چپ می‌باشد.

۴-۴-۳- شیفت حسابی

شیفت حسابی عملی است که یک عدد دودویی علامت‌دار را به چپ یا به راست شیفت می-دهد. شیفت حسابی نیز به دو صورت می‌تواند انجام شود: شیفت حسابی به راست و شیفت حسابی به چپ. شیفت حسابی به چپ دقیقاً همانند شیفت منطقی به چپ است. به طور مثال اگر یک رجیستر ۸ بیتی را که دارای محتوای 00000010 می‌باشد یکبار به سمت چپ شیفت حسابی دهیم محتوای آن به صورت 00000100 خواهد شد. اگر خوب دقت کنید متوجه می‌شوید که شیفت حسابی به چپ، یک عدد دودویی علامت‌دار را در 2 ضرب می‌کند به دلیل اینکه محتوای رجیستر قبل از شیفت ۲ بوده و بعد از شیفت ۴ شده است. یک شیفت حسابی به راست نیز عدد را بر 2 تقسیم می‌کند. شیفت‌های حسابی نباید بیت علامت را تغییر دهند زیرا وقتی عدد را در 2 ضرب یا تقسیم می‌کنیم علامت همچنان باقی می‌ماند.

لازم به ذکر است که سمت چپ‌ترین بیت در ثبات، بیت علامت را نگه می‌دارد، و بقیه بیت‌ها عدد را حفظ می‌کنند. بیت علامت برای اعداد مثبت، 0 و برای اعداد منفی،1 است. شکل ۱۴ نمونه‌ای از یک ثبات n بیت را نشان می‌دهد. بیت R_{n-1} در سمت چپ‌ترین مکان بیت علامت را نگه می‌دارد. شیفت به راست حسابی بیت علامت را عوض نمی‌کند و همه بیت‌ها (از جمله علامت) را به راست شیفت می‌دهد. بنابراین R_{n-1} تغییر نمی‌کند، R_{n-2} بیت R_{n-1} را دریافت می‌نماید و برای سایر بیت‌های

ثبات نیز به همین ترتیب. بیت واقع در R_0 از دست می‌رود. عملیات شیفت حسابی به چپ در شکل ۱۴ نشان داده شده است.



شکل ۱۴: شیفت حسابی به راست

شیفت حسابی به چپ یک 0 وارد R_0 می‌نماید، و کلیه بیت‌های دیگر را به چپ شیفت می-دهد. در این عملیات مقدار اولیه R_{n-1} از دست رفته و با بیت R_{n-2} جایگزین می‌شود. اگر بیت واقع در R_{n-1} پس از شیفت عوض شود، در این صورت علامت معکوس شده است. این هنگامی رخ می‌دهد که ضرب در 2 سبب سرریز گردد. سرریز در شیفت به چپ هنگامی رخ می‌دهدکه قبل از شیفت R_{n-1} و R_{n-2} مساوی نباشد. یک فلیپ فلاپ به نام V برای کشف یک سرریز حاصل از شیفت حسابی به چپ می‌تواند مورد استفاده قرار گیرد.

$$V = R_{n-1} \oplus R_{n-2}$$

اگر V= 0 باشد، سرریز وجود ندارد، ولی اگر V=1 گردد، سرریز وجود داشته و پس از شیفت علامت عوض خواهد شد.

۴-۴-۴- دستورات شیفت پردازنده MIPS

دستور شیفت منطقی به چپ:

```
sll  rd, rt, shamt  //  rd = rt << shamt
```

مثال:

```
sll  $v0, $t0, 4    //  $v0 = $t0 << 4
```

دستور شیفت منطقی به راست:

```
srl  rd, rt, shamt    //  rd = rt >> shamt
```

دستور شیفت حسابی به راست:

```
sra  rd, rt, shamt    //  rd = rt >> shamt
```

۴-۵- عملیات ضرب

ضرب دو عدد ممیز ثابت دودوئی با نمایش مقدار علامت[1] با قلم و کاغذ، توسط فرآیند شیفت-های متوالی و جمع انجام می‌شود. این روش را با مثالی می‌توان بهتر تشریح نمود.

23	10111	مضروب Multiplicand
19	* 10011	مضروب فیه Multiplier
	10111	
	10111	
	00000	
	00000	+
	10111	
437	110110101	حاصل ضرب Product

فرآیند به این صورت است که بیت‌های مضروب‌فیه متوالیاً، با شروع از کم ارزشترین بیت بررسی می‌شوند. اگر بیت مضروب‌فیه ۱ باشد، مضروب در پائین کپی می‌شود، در غیر این صورت صفرها در پائین کپی می‌گردند. اعدادی که در سطرهای متوالی کپی می‌شوند نسبت به سطر قبل یک بیت به چپ شیفت داده می‌شوند. نهایتاً اعداد با هم جمع شده و حاصل‌جمع همه سطرها نتیجه ضرب خواهد بود. علامت حاصل‌ضرب با توجه به علامت‌های مضروب و مضروب‌فیه معین می‌شود. اگر علامتها یکی باشند علامت حاصل‌ضرب مثبت و در غیر این صورت منفی خواهد بود.

۴-۵-۱- پیاده سازی سخت‌افزاری برای داده‌های علامت‌دار

هنگام پیاده‌سازی ضرب در یک کامپیوتر دیجیتال، بهتر است فرآیند کمی تغییر یابد. اولاً به جای تهیه ثباتهائی به تعداد بیت‌های مضروب‌فیه برای ذخیره و جمع همزمان چند عدد دودوئی، بهتر است جمع‌کننده‌ای برای جمع فقط دو عدد دودوئی در نظر گرفته شود و مرتباً حاصل‌ضربهای جزئی[2] را در یک ثبات نگهداری نمائیم. ثانیاً به جای شیفت مضروب به چپ، حاصل‌ضرب جزئی به راست شیفت داده شود که در نتیجه موقعیت نسبی حاصل‌ضرب جزئی و مضروب همان موقعیت مطلوب خواهد بود. ثالثاً وقتی‌که بیتی از مضروب‌فیه صفر باشد، لزومی ندارد که صفرها را با حاصل‌ضرب جزئی جمع کنیم زیرا مقدار آن را عوض نمی‌کند.

---

[1] - Sign magnitude

[2] - Partial product

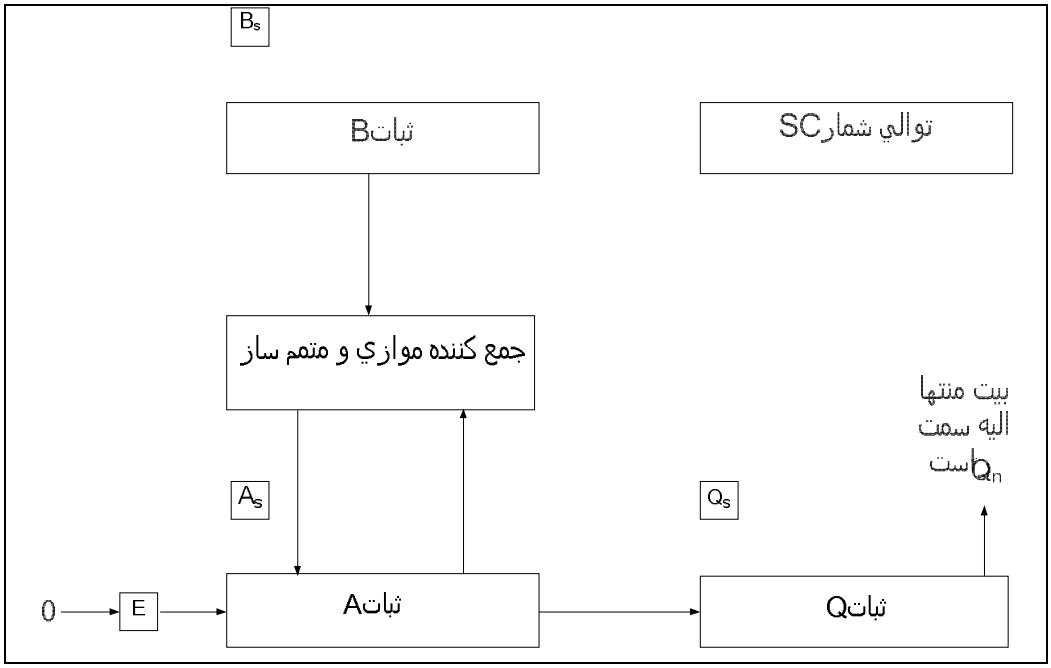
سخت‌افزار ضرب در شکل ۱۵ و فلوچارت آن در شکل ۱۶ نشان داده شده است. همان طور که مشاهده می‌شود در سخت افزار ضرب چند رجیستر یا ثبات به همراه چند فلیپ فلاپ مشاهده می‌شود. فلیپ فلاپ‌ها برای نگهداری علامت عددها به کار می‌روند. همچنین در این سخت افزار یک جمع کننده نیز تعبیه شده است که حاصل جمع‌های میانی را انجام می‌دهد.

چون تعداد بیت‌های حاصل ضرب دو عدد، از تعداد بیت‌های هر دو عدد بیشتر است، بنابراین در این الگوریتم حاصل‌ضرب نهائی در ثباتهای A و Q قرار خواهد گرفت.

در ابتدا مضروب‌فیه در ثبات Q و علامتش در Q_s ذخیره می‌شود. همچنین مضروب در داخل ثبات B و علامتش در B_s قرار داده می‌شود و ثباتA نیز با صفر پر می‌شود. توالی شمار[1] SC نقش شمارنده را دارد و در ابتدا با عددی برابر با تعداد بیتهای مضروب‌فیه مقداردهی می‌شود. شمارنده پس از هر بار تشکیل حاصل‌ضرب جزئی یک واحد کم می‌شود. وقتی که محتوای شمارنده به صفر برسد، حاصل‌ضرب تکمیل و فرآیند متوقف می‌گردد.

در هر مرحله، حاصل‌جمع A وB تشکیل حاصل‌ضرب جزئی را می‌دهند که به ثبات EA منتقل می‌گردد. حاصل‌ضرب جزئی و مضروب‌فیه توأماً به راست شیفت داده می‌شوند. این شیفت توسط عبارت shr EAQ نشان داده می‌شود. بیت E به با ارزشترین بیت A نقل مکان می‌یابد، و یک 0 وارد E می‌شود، بیت کم ارزشتر A به با ارزشترین بیت Q منتقل می‌شود، به عبارتی یک بیت از حاصل‌ضرب جزئی به داخل Q شیفت پیدا کرده و بیت‌های مضروب‌فیه را یک واحد به سمت راست می‌راند. به این ترتیب، سمت راست‌ترین فلیپ فلاپ در ثبات Q، که با Q_n مشخص شده است حاوی بیتی از مضروب‌فیه خواهد بود که در مرحله بعدی باید بررسی شود.

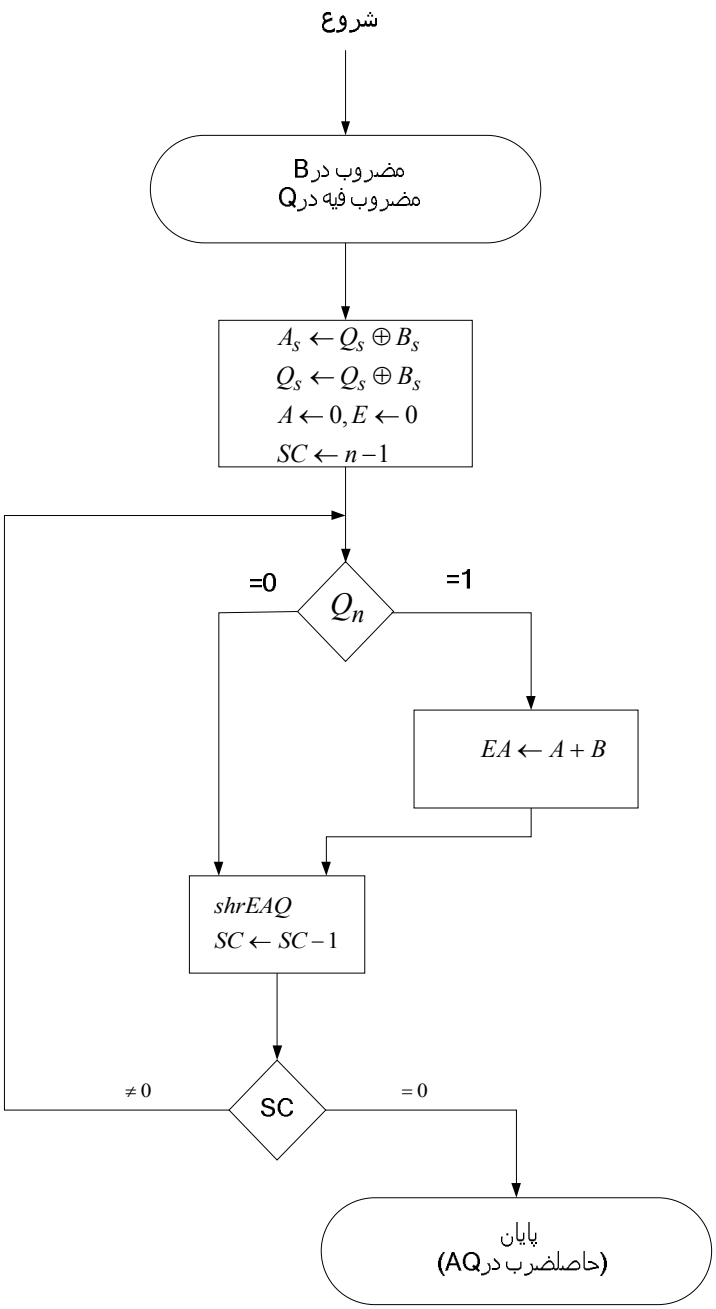
[1] - Sequence Counter



شکل ۱۵: سخت افزار عمل ضرب

۴-۵-۲- الگوریتم سخت‌افزاری

فلوچارت الگوریتم سخت‌افزاری ضرب در شکل ۱۶ نشان داده شده است. در شروع کار مضروب در B و مضروب‌فیه در Q و علامت‌های مربوطه به ترتیب در B_s و Q_s قرار دارند. در ابتدا علامت‌ها با هم مقایسه شده و نتیجه این مقایسه علامت حاصل ضرب دو عدد را تعیین می‌کند که در A_s و Q_s قرار داده می‌شود.



شکل ۱۶: فلوچارت عمل ضرب

ثباتهای A و E پاک شده و توالی شمار SC برابر با تعداد بیتهای مضروب فیه می‌شود. در اینجا فرض می‌کنیم که عملوندها از حافظه‌های n بیتی به ثباتها منتقل شده‌اند. چون هر عملوند باید همراه با علامتش ذخیره شود لذا یک بیت از کلمه توسط علامت اشغال شده و مقدار عملوند n-1 بیتی خواهد بود.

پس از دادن مقادیر اولیه، بیت کم‌ارزشتر مضروب فیه که در Q_n قرار دارد تست می‌شود. اگر این بیت برابر با 1 باشد، مضروب موجود در B با حاصل‌ضرب جزئی موجود در A جمع می‌شود. و اگر نتیجه تست 0 باشد، کاری انجام نمی‌شود. سپس مجموعه ثبات EAQ یک بار به سمت راست شیفت داده می‌شود تا حاصل‌ضرب جزئی را تشکیل دهد. سپس توالی شمار 1 واحد کم شده و مقدار جدید آن چک می‌شود. اگر برابر صفر نباشد، فرآیند تکرار شده و حاصل‌ضرب جزئی جدید تشکیل می‌گردد. وقتی که SC=0 شود فرآیند متوقف می‌گردد. توجه کنید که حاصل‌ضرب جزئی حاصل در A هر بار یک بیت به Q منتقل می‌گردد تا نهایتاً جای مضروب‌فیه را می‌گیرد. حاصل‌ضرب نهائی در هر دو ثبات A و Q واقع است به این ترتیب که A بیت‌های با ارزشتر و Q بیت‌های کم ارزشتر را نگه می‌دارند. مثال عددی قبلی، برای روشن شدن فرآیند ضرب سخت‌افزاری در جدول ۲ تکرار شده است. روند، مراحل مشخص شده در فلوچارت را دنبال می‌کند.

جدول ۲: مثال عددی برای ضرب کننده دودویی

مضروب B = 10111	E	A	Q	SC
مضروب فیه در Q	0	00000	10011	101
Q_n=1 ؛ جمع B		10111		
		10111		
اولین حاصلضرب جزیی	0	10111		
شیفت EAQ به راست و کم کردن شمارنده	0	01011	11001	100
Q_n=1 ؛ جمع B		10111		
		00010		
دومین حاصلضرب جزیی	1	10001	01100	011
شیفت EAQ به راست و کم کردن شمارنده	0	01000	10110	010
Q_n=0 ؛ شیفت EAQ به راست و کم کردن شمارنده	0	00100	01011	001
Q_n=0 ؛ شیفت EAQ به راست و کم کردن شمارنده	0	10111		
Q_n=1 ؛ جمع B		11011		
		01101	10101	000
پنجمین حاصلضرب جزیی	0			
شیفت EAQ به راست و کم کردن شمارنده	0			
حاصلضرب نهایی در AQ برابر 0110110101 است				

اگر فرض کنیم که در الگوریتم فوق هر کدام از مراحل در یک کلاک قابل انجام باشند در این صورت یک ضرب ۳۲ بیتی برای اجرا شدن نیاز به حداقل ۳۲ کلاک خواهد داشت (برای مقداردهی-های اولیه و همچنین مراحل پایانی کار که نتیجه قرار است در رجیستر و یا حافظه نوشته شود نیز کلاک‌هایی لازم است). به عبارتی می‌توان گفت که با استفاده از پیاده سازی فوق CPI دستور ضرب

حداقل برابر ۳۲ خواهد بود. به همین دلیل دستور ضرب در پردازنده‌ها معمولاً از منظر پیاده سازی و هزینه، دستور سنگینی به حساب می‌آید.

۴-۵-۳- الگوریتم ضرب بوث

الگوریتم بوث رویه‌ای را برای ضرب اعداد دودوئی در نمایش متمم ۲ علامت‌دار ارائه می‌نماید. مبنای کار الگوریتم بر این اساس استوار است که رشته‌های 0 در مضروب‌فیه نیازی به جمع ندارند بلکه فقط جابجائی (شیفت) لازم دارند و رشته‌های 1 در مضروب‌فیه از بیت مرتبه 2^k تا بیت 2^m را می‌توان معادل $2^{k+1}-2^m$ تلقی کرد. به طور مثال عدد دودوئی 001110 (+14) دارای رشته‌های 1 از 2^1 تا 2^3 است(m=1,k=3). این عدد را می‌توان به صورت $2^{k+1}-2^m = 2^4 - 2^1 = 16-2 = 14$ نوشت. بنابراین ضرب M*14 را، که درآن M مضروب و 14 مضروب‌فیه است را می‌توان به صورت $M\times 2^4 - M\times 2^1$ انجام داد. لذا حاصل‌ضرب با چهار بار شیفت به چپ M و تفریق یک‌بار شیفت به چپ داده شده M از آن به دست می‌آید.

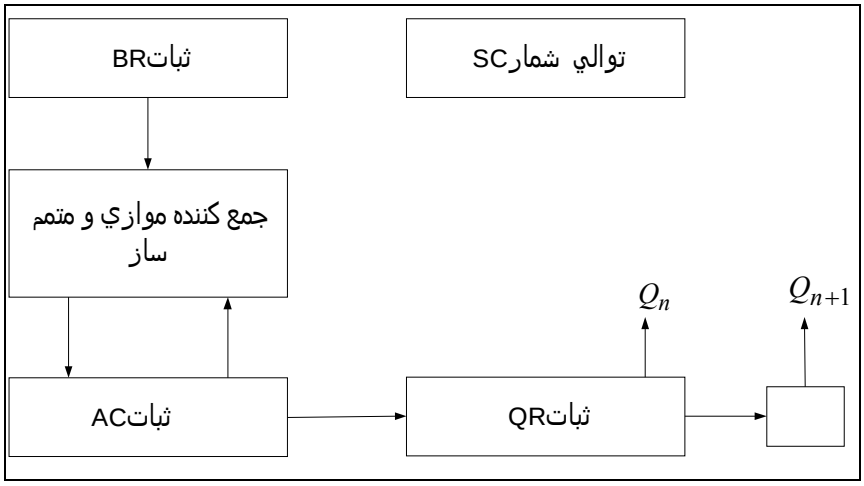
همانند همهٔ روشهای ضرب، الگوریتم بوث نیز به بررسی بیت‌های مضروب فیه و شیفت حاصل-ضرب جزئی نیاز دارد. قبل از شیفت، ممکن است مضروب طبق قواعد زیر با حاصل‌ضرب جزئی جمع شود، از آن تفریق شود و یا حاصل‌ضرب جزئی بلا تغییر باقی بماند.

۱. به محض برخورد با اولین 1 کم ارزش در رشته 1 ها در مضروب‌فیه، مضروب از حاصل‌ضرب جزئی کم می‌شود.

۲. به محض برخورد با اولین 0 (به شرطی که قبل از آن ۱ باشد) در رشته‌ای از 0 ها در مضروب-فیه، مضروب با حاصل‌ضرب جزئی جمع می‌شود.

۳. وقتی که بیت جاری مضروب‌فیه همانند بیت قبلی باشد، حاصل‌ضرب جزئی تغییر نمی‌کند.

الگوریتم فوق برای مضروب‌فیه‌های مثبت و یا منفی به فرم متمم 2 قابل استفاده است. این بدان علت است که مضروب فیه منفی با رشته‌ای از 1 ها خاتمه می‌یابد و آخرین عمل تفریق با وزن مناسب خواهد بود. مثلاً مضروب‌فیه 14- در نمایش متمم 2 عبارت است از 110010 و به صورت $-2^4+2^2-2^1=-14$ با آن رفتار می‌شود.

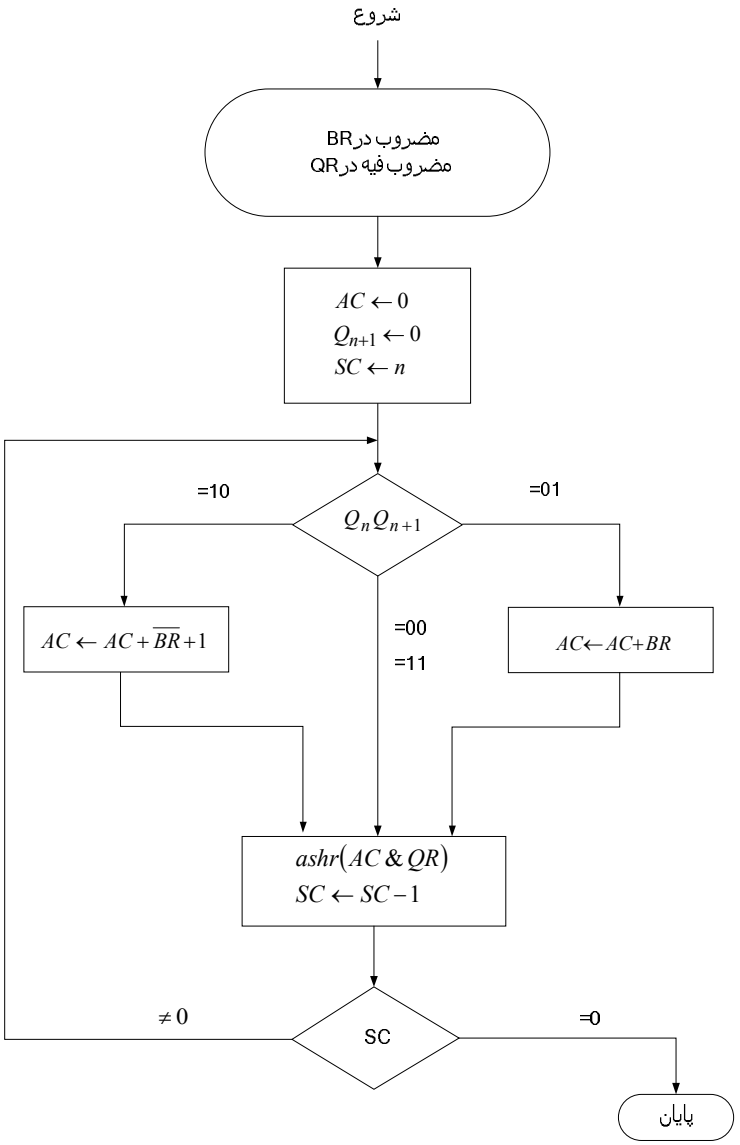


شکل ۱۷: سخت افزار الگوریتم بوث

پیاده‌سازی سخت‌افزار الگوریتم بوث آرایش ثبات شکل ۱۷ را نیاز دارد. این شکل مشابه شکل ۱۵ است جز اینکه بیت‌های علامت از بقیه ثباتها تفکیک نشده‌اند. برای ملاحظه این اختلاف، ما ثبات-های A و B و Q را به ترتیب AC و BR و QR نام‌گذاری کرده‌ایم. Q_n کم‌ارزشترین بیت در ثبات QR است. یک فلیپ فلاپ اضافی Q_{n+1} برای بررسی همزمان دو بیت از مضروب‌فیه به QR ملحق شده است. فلوچارت الگوریتم بوث در شکل ۱۸ نشان داده شده است. در ابتدا AC و بیت الحاقی پاک می-شوند و شمارنده SC برابر تعداد بیت‌های مضروب‌فیه یعنی n قرار داده می‌شود. در هر مرحله دو بیت از مضروب‌فیه که در Q_n و Q_{n+1} قرار دارند مورد بررسی قرار می‌گیرند. اگر دو بیت برابر 10 باشند بدان معنی است که اولین 1 در رشته 1 ها فرا رسیده است. این حالت، تفریق مضروب از حاصل‌ضرب جزئی موجود در AC را لازم می‌دارد. اگر دو بیت01 باشد، مفهوم این است که با اولین 0 در رشته 0 ها برخورد شده است. این حالت جمع مضروب با حاصل‌ضرب جزئی موجود در AC را لازم می‌دارد. هرگاه دو بیت برابر باشند، حاصل‌ضرب جزئی تغییر نمی‌کند.

قدم بعدی شیفت حاصل‌ضرب جزئی و مضروب‌فیه (شامل بیت Q_{n+1}) به راست است. این یک عمل شیفت حسابی به راست است که AC و QR را به راست جابه‌جا کرده و بیت علامت در AC بلا-تغییر می‌ماند. در آخر هر مرحله شمارنده SC کاهش یافته و با صفر مقایسه می‌شود. حلقه محاسبه الگوریتم بوث n بار تکرار می‌گردد.

در الگوریتم بوث سرریز نمی‌تواند رخ دهد زیرا جمع و تفریق مضروب یک در میان انجام می-شود. در نتیجه دو عددی که جمع می‌شوند، همواره علامت‌های مخالف دارند، و به این ترتیب امکان وقوع سرریز وجود نخواهد داشت.



شکل ۱۸: الگوریتم بوث برای ضرب اعداد متمم ۲ علامت‌دار

مثال عددی الگوریتم بوث در جدول ۳ برای n=5 نشان داده شده است. این مثال، ضرب $(-9)\times(-13) = +117$ را نشان می‌دهد. توجه کنید که مضروب فیه در QR منفی و مضروب نیز در BR منفی است. نتیجه نهایی عملیات ضرب ۱۰ بیتی بوده و در جفت ثبات AC و QR قرار می‌گیرد.

برای این مثال، حاصل‌ضرب یک عدد مثبت است. توجه کنید که مقدار نهائی Q_{n+1} علامت اولیه مضروب‌فیه است و نباید بخشی از حاصل‌ضرب تلقی شود.

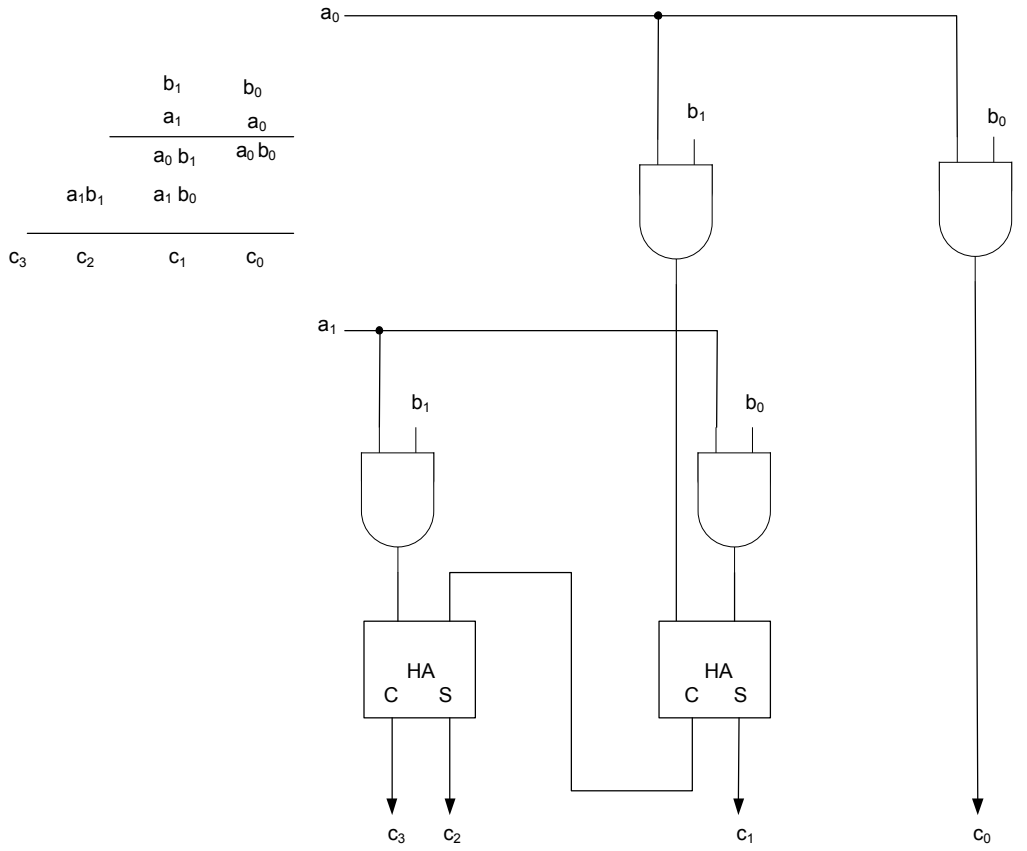
جدول ۳: مثال ضرب با الگوریتم بوث

Q_nQ_{n+1}	BR = 10111 $\overline{BR}+1=01001$	AC	QR	Q_{n+1}	SC
	مقدار اولیه	00000	10011	0	101
10	تفریق BR	01001			
		01001			
	ashr	00100	11001	1	100
11	ashr	00010	01100	1	011
01	جمع BR	10111			
		11001			
	ashr	11100	10110	0	010
00	ashr	11110	01011	0	001
10	تفریق BR	01001			
		00111			
	ashr	00011	10101	1	000

۴-۵-۴- ضرب کننده آرایه‌ای

تست یک به یک بیت‌های مضروب‌فیه و تشکیل حاصل‌ضرب جزئی یک عمل ترتیبی است که دنباله‌ای از اعمال جمع و شیفت را نیاز دارد. بنابراین مدارهای فوق مدارهای ترتیبی هستند. مدار ضرب کننده را می‌توان با یک مدار کاملاً ترکیبی نیز پیاده سازی نمود. در این قسمت قصد داریم که یک ضرب کننده ترکیبی طراحی کنیم. به دلیل ساختار منظم سخت افزار در این ضرب کننده به آن ضرب کننده آرایه‌ای گفته می‌شود. برای اینکه ببینیم یک ضرب کننده ترکیبی چگونه پیاده‌سازی می‌شود، ضرب دو عدد ۲ بیت را مطابق شکل ۱۹ در نظر بگیرید. بیت‌های مضروب b_1 و b_0 و بیت‌های مضروب‌فیه a_1 و a_0 و حاصل‌ضرب $c_3c_2c_1c_0$ می‌باشند. اولین حاصل‌ضرب جزئی با ضرب a_0 در b_1b_0 بدست می‌آید. ضرب دو بیت مانند a_0 و b_0 برابر 1 خواهد بود به شرطی که هر دو 1 باشند، در غیر این صورت 0 است. این همان عمل AND است و بنابراین ضرب دو بیت را می‌توان با یک گیت AND پیاده‌سازی نمود. همان طور که در شکل دیده می‌شود، اولین حاصل‌ضرب جزئی با دو گیت AND

حاصل می‌شود. دومین حاصل‌ضرب جزئی از ضرب a_1 در b_1b_0 و شیفت آن به چپ بدست می‌آید. دو حاصل‌ضرب جزئی به وسیله دو مدار نیم جمع‌کننده (HA) با یکدیگر جمع شده و نتیجه نهایی بدست می‌آید. معمولاً تعداد بیت‌های حاصل‌ضرب جزئی بیشتر است و لازم خواهد بود تا از مدار تمام جمع کننده استفاده شود. دقت داشته باشید که کم ارزشترین بیت حاصل‌ضرب لازم نیست وارد جمع-کننده شود زیرا توسط خروجی گیت AND تشکیل می‌گردد.



شکل ۱۹: ضرب کننده آرایه‌ای ۲ بیت در ۲ بیت

یک مدار ضرب‌کننده دودوئی با بیت‌های بیشتر را می‌توان به روشی مشابه ساخت. هر بیت از مضروب‌فیه با هر بیت از مضروب، به تعداد بیت‌های مضروب فیه، AND می‌شود. خروجی دودوئی در هر یک از سطوح گیت‌های AND به طور موازی با حاصل‌ضرب جزئی قبلی جمع می‌شود و حاصل-ضرب جزئی جدیدی را تشکیل می‌دهد. آخرین سطح، حاصل‌ضرب را ایجاد می‌نماید.

توجه: برای مضروب‌فیه j بیتی و مضروب k بیتی به تعداد j×k گیت AND و (j-1)k جمع کننده k بیتی برای تولید حاصل‌ضرب j+k بیتی مورد نیاز است.

به عنوان مثال دوم، یک مدار ضرب‌کننده را در نظر بگیرید که یک عدد دودوئی چهار بیتی را در یک عدد سه بیتی ضرب می‌کند. فرض کنید مضروب را با $b_3b_2b_1b_0$ و مضروب‌فیه را با $a_2a_1a_0$ نشان دهیم. چون k=4 و j=3 است، 12 گیت AND و 2 جمع کننده 4 بیتی مورد نیاز است تا حاصل-ضرب هفت بیتی را ایجاد کند. نمودار منطقی این ضرب‌کننده در شکل ۲۰ نشان داده شده است.

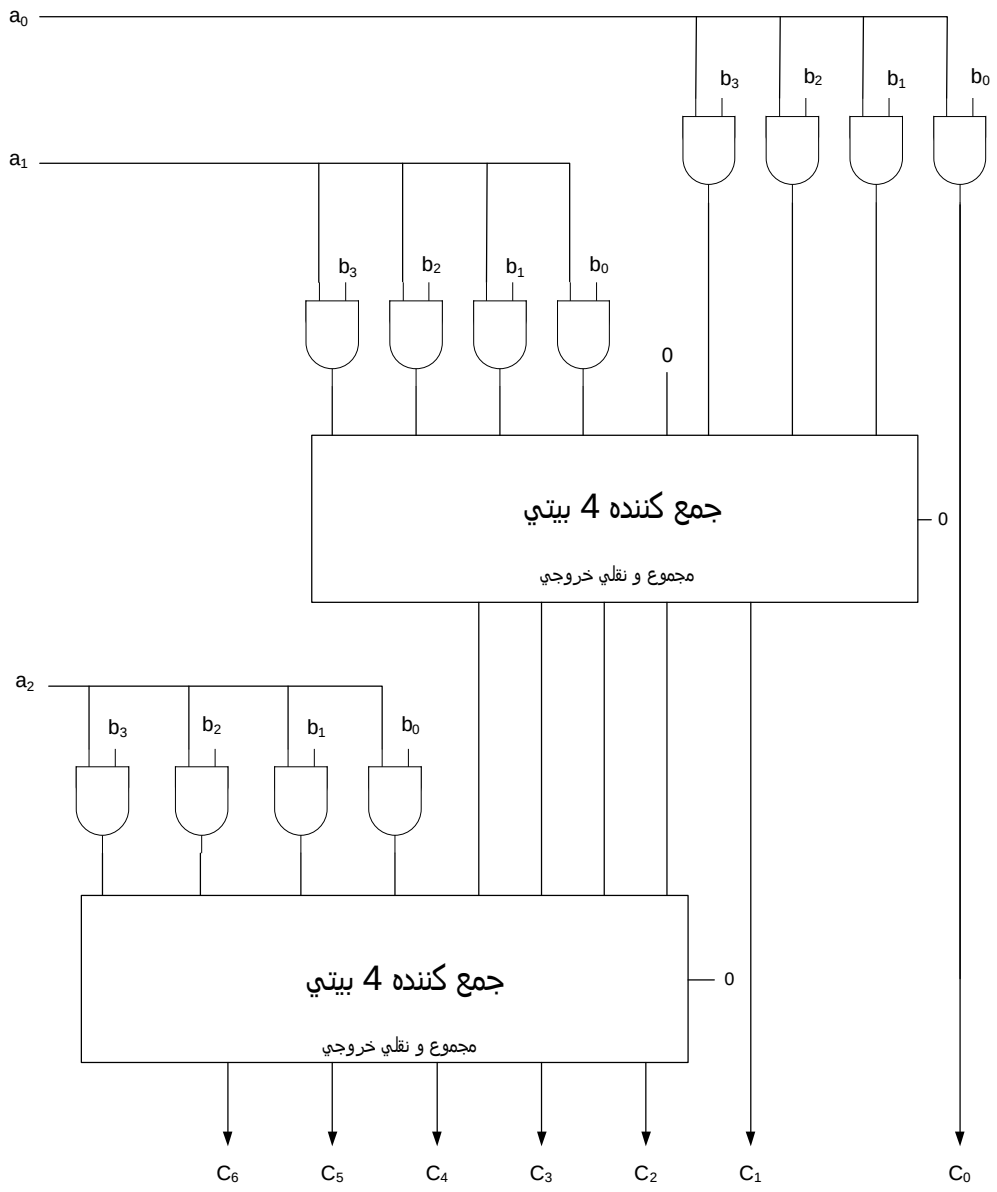
اگر سخت افزار عملیات ضرب را به صورت آرایه‌ای پیاده سازی کنیم، در این صورت عملیات ضرب در یک کلاک قابل انجام است. در این مدار تأخیر مسیر از ورودی‌ها به خروجی‌ها به دلیل وجود تعداد زیادی گیت بسیار بالا است به طور مثال حالتی را در نظر بگیرید که می‌خواهیم دو عدد ۳۲ بیتی را در هم ضرب کنیم در این حالت به تعداد ۳۱ عدد جمع کننده علاوه بر گیتهای AND در مسیر وجود خواهد داشت. بنابراین اگر بخواهیم این عملیات را در یک کلاک انجام دهیم باید پریود کلاک حداقل به اندازه ماکزیمم تأخیر مدار باشد که در نتیجه آن فرکانس کاری پردازنده به شدت پائین خواهد آمد.

۴-۵-۵- عملیات و دستورات ضرب پردازنده MIPS

پردازنده MIPS دو عدد رجیستر ۳۲ بیتی مخصوص به نام‌های Hi و Lo برای عملیات ضرب در نظرگرفته است. این دو رجیستر حاصل‌ضرب دو عدد ۳۲ بیتی را که ۶۴ بیت است نگهداری می‌کنند. ۳۲ بیت پر ارزش حاصل‌ضرب در Hi و ۳۲ بیت کم ارزش آن در Lo ریخته می‌شود. لازم به ذکر است که این دو رجیستر جزو ۳۲ رجیستر پردازنده MIPS که قبلاً توضیح داده شده است (رجیسترهای عمومی)، نیستند بلکه رجیسترهای مخصوصی هستند که فقط برای عملیات ضرب و تقسیم استفاده می-شوند. پردازنده MIPS، دارای دو نوع دستور ضرب است: ضرب اعداد علامت دار و ضرب اعداد بدون علامت. دستور mult[1] برای ضرب دو عدد علامت‌دار و دستور multu[2] برای ضرب دو عدد

[1] - Multiply
[2] - Multiply unsigned

بدون علامت به کار می‌رود. می‌توانیم هر موقع که لازم شد محتوای رجیسترهای Hi و Lo را به داخل رجیسترهای عمومی پردازنده منتقل کنیم. این کار به کمک دو دستور mfhi[1] و mflo[2] انجام می‌شود.



شکل ۲۰: ضرب کننده آرایه‌ای ۴ بیت در ۳ بیت

[1] - Move from high
[2] - Move from low

۴-۶- عملیات تقسیم

در تقسیم دو عدد ممیز ثابت در نمایش مقدار-علامت با قلم و کاغذ، از اعمال مقایسه، شیفت و تفریق استفاده می‌شود. تقسیم دودوئی از تقسیم دهدهی ساده‌تر است زیرا ارقام خارج قسمت 0 یا 1 هستند و نیازی نیست تا بدانیم مقسوم علیه چند بار در مقسوم جای می‌گیرد. فرآیند تقسیم با مثال عددی شکل ۲۱ توضیح داده شده است. مقسوم‌علیه B از پنج بیت و مقسوم A از ده بیت تسکیل شده است. در ابتدا پنج بیت با ارزش‌تر مقسوم با مقسوم‌علیه مقایسه می‌شوند. چون این پنج بیت از B کوچکتر است، مقایسه B را با شش بیت با ارزشتر A آزمایش می‌کنیم. این عدد شش بیتی بزرگتر از B است، لذا رقم 1 را برای خارج قسمت می‌نویسیم. سپس مقسوم‌علیه را یک بار به راست شیفت می‌دهیم و آن را از مقسوم کم می‌کنیم. تفاضل بدست آمده باقیمانده جزئی[1] خوانده می‌شود زیرا تقسیم ممکن است در همین مرحله پایان یابد و خارج‌قسمتی برابر با 1 و باقیمانده‌ای برابر با این باقیمانده جزئی داشته باشد.

(17) Divisor یا مقسوم علیه	B=10001	
(26) خارج قسمت یا Quotient	11010	
Dividend یا مقسوم (448)	A= 0111000000	
خارج قسمت 5 بیت دارد، 5 بیت از A ، A<B	01110	
6 بیت از A ، A>=B	011100	
جابجایی B به راست و تفریق ؛ قرار دادن 1 در Q	-10001	
باقیمانده بزرگتر یا مساوی با B	-010110	
جابجایی B به راست و تفریق ؛ قرار دادن 1 در Q	--10001	
باقیمانده کوچکتر از B ؛ قرار دادن 0 در Q ؛ شیفت B به راست	--001010	
باقیمانده بزرگتر یا مساوی با B	---010100	
جابجایی B به راست و تفریق ؛ قرار دادن 1 در Q	----10001	
باقیمانده کوچکتر از B ؛ قرار دادن 0 در Q	----000110	
باقیمانده نهایی (6)	-----00110	

شکل ۲۱: مثال تقسیم دودویی

روند تقسیم با مقایسه باقیمانده جزئی با مقسوم‌علیه ادامه می‌یابد. اگر باقیمانده جزئی بزرگتر یا مساوی مقسوم‌علیه باشد، بیت خارج قسمت برابر 1 شده و سپس مقسوم‌علیه یک واحد به راست شیفت داده شده و از باقیمانده جزئی کم می‌شود. اگر باقیمانده جزئی کوچکتر از مقسوم علیه باشد، بیت

[1] - Partial reminder

خارج قسمت 0 می‌شود و تفریق لازم نیست. در هر صورت مقسوم‌علیه یک مرتبه به راست شیفت داده می‌شود. توجه کنید که در نتیجه این روند هم خارج‌قسمت و هم باقیمانده بدست خواهد آمد.

۴-۶-۱- پیاده‌سازی سخت‌افزاری تقسیم برای داده‌های با نمایش مقدار-علامت

هنگام پیاده‌سازی تقسیم در یک کامپیوتر دیجیتال، بهتر است روند کمی تغییر یابد. به جای شیفت مقسوم‌علیه به سمت راست، مقسوم یا باقیمانده جزئی را به چپ شیفت می‌دهیم و به این ترتیب موقعیت نسبی دو عدد همان موقعیت مورد نظر خواهد بود. تفریق را می‌توان با جمع A و متمم 2 عدد B انجام داد. نسبت اندازه‌ها از رقم نقلی انتهائی مشخص می‌گردد.

سخت‌افزار مربوط به پیاده‌سازی تقسیم همان سخت‌افزار ضرب است و از قطعات شکل ۱۵ تشکیل شده است. در اینجا ثبات EAQ به چپ شیفت داده شده و 0 در Q_n وارد شده و مقدار قبلی E هم از دست می‌رود. مثال عددی شکل ۲۱ برای روشنتر کردن روند تقسیم مجدداً در شکل ۲۲ نشان داده شده است. مقسوم‌علیه در ثبات B و مقسوم که طولی دو برابر دارد در ثباتهای A و Q ذخیره می‌شوند. مقسوم به سمت چپ شیفت داده می‌شود و متمم 2 مقسوم‌علیه با آن جمع شده و لذا عمل تفریق تحقق می‌یابد. اطلاعات مربوط به نسبت اندازه‌ها در E موجود است. اگر E=1 باشد به معنی $A \geq B$ است که در این صورت باقیمانده جزئی به سمت چپ شیفت پیدا کرده و بیت 1 وارد Q_n می‌گردد. اگر E=0 باشد، یعنی A<B است و لازم نبوده که B از A کم شود، به همین دلیل B مجدداً با A جمع می‌شود تا باقیمانده جزئی در A به مقدار قبلی‌اش باز گردانده شود. در این حالت باقیمانده جزئی به چپ شیفت داده شده و بیت 0 وارد Q_n می‌گردد. روند مجدداً تکرار می‌گردد تا اینکه هر پنج بیت خارج‌قسمت ایجاد شود.

توجه کنید که ضمن شیفت باقیمانده جزئی به چپ، بیت‌های خارج‌قسمت نیز شیفت داده می‌شود و پس از پنج بار شیفت، خارج‌قسمت در Q و باقیمانده در A خواهد بود. قبل از نمایش الگوریتم به صورت فلوچارت، باید علامت نتیجه و حالت سرریز احتمالی را در نظر داشت. علامت خارج‌قسمت از علامت‌های مقسوم و مقسوم‌علیه تعیین می‌شود. اگر علامت‌های این دو یکسان باشند، علامت خارج قسمت مثبت خواهد بود و اگر مخالف باشند، علامت منفی است. علامت باقیمانده همانند علامت مقسوم است.

مقسوم علیه $B = 10001$			$\overline{B}+1 = 01111$	
	E	A	Q	SC
مقسوم		01110	00000	5
shl EAQ	0	11100	00000	
جمع $\overline{B}+1$		01111		
		———		
E=1	1	01011		
قرار دهید Q_n=1	1	01011	00001	4
shl EAQ	0	10110	00010	
جمع $\overline{B}+1$		01111		
		———		
E=1	1	00101		
قرار دهید Q_n=1	1	00101	00011	3
shl EAQ	0	01010	00110	
جمع $\overline{B}+1$		01111		
		———		
Q_n=0 ، بگذارید E=0	0	11001	00110	
جمع B		10001		
		———		
بازیابی باقیمانده	1	01010		2
shl EAQ	0	10100	01100	
جمع $\overline{B}+1$		01111		
		———		
E=1	1	00011		
قرار دهید Q_n=1	1	00011	01101	1
shl EAQ	0	00110	11010	
جمع $\overline{B}+1$		01111		
		———		
Q_n=0 ، بگذارید E=0	0	10101	11010	
جمع B		10001		
		———		
بازیابی باقیمانده	1	00110	11010	0
باقیمانده در A		00110		
خارج قسمت در Q			11010	

شکل ۲۲: مثال تقسیم دودویی با سخت افزار دیجیتال

۴-۶-۲- سرریز در تقسیم

عمل تقسیم ممکن است منجر به سرریز در خارج قسمت شود. هنگام استفاده از کاغذ و قلم، این امر مشکلی ایجاد نمی‌کند ولی هنگام پیاده‌سازی عمل با سخت‌افزار، مسئله ساز خواهد بود. دلیل این است که طول ثباتها محدود است و نمی‌توانند عددی را که اندازه آن از طول استاندارد تجاوز می-کند، نگهداری کنند. برای درک موضوع، سیستمی را که دارای ثبات‌های پنج بیتی است در نظر بگیرید. ما از یک ثبات برای نگهداری مقسوم علیه و از دو ثبات برای نگهداری مقسوم استفاده می-کنیم. با توجه به مثال شکل ۲۱ می‌بینیم که اگر پنج بیت با ارزشتر مقسوم، عددی بزرگتر از مقسوم علیه

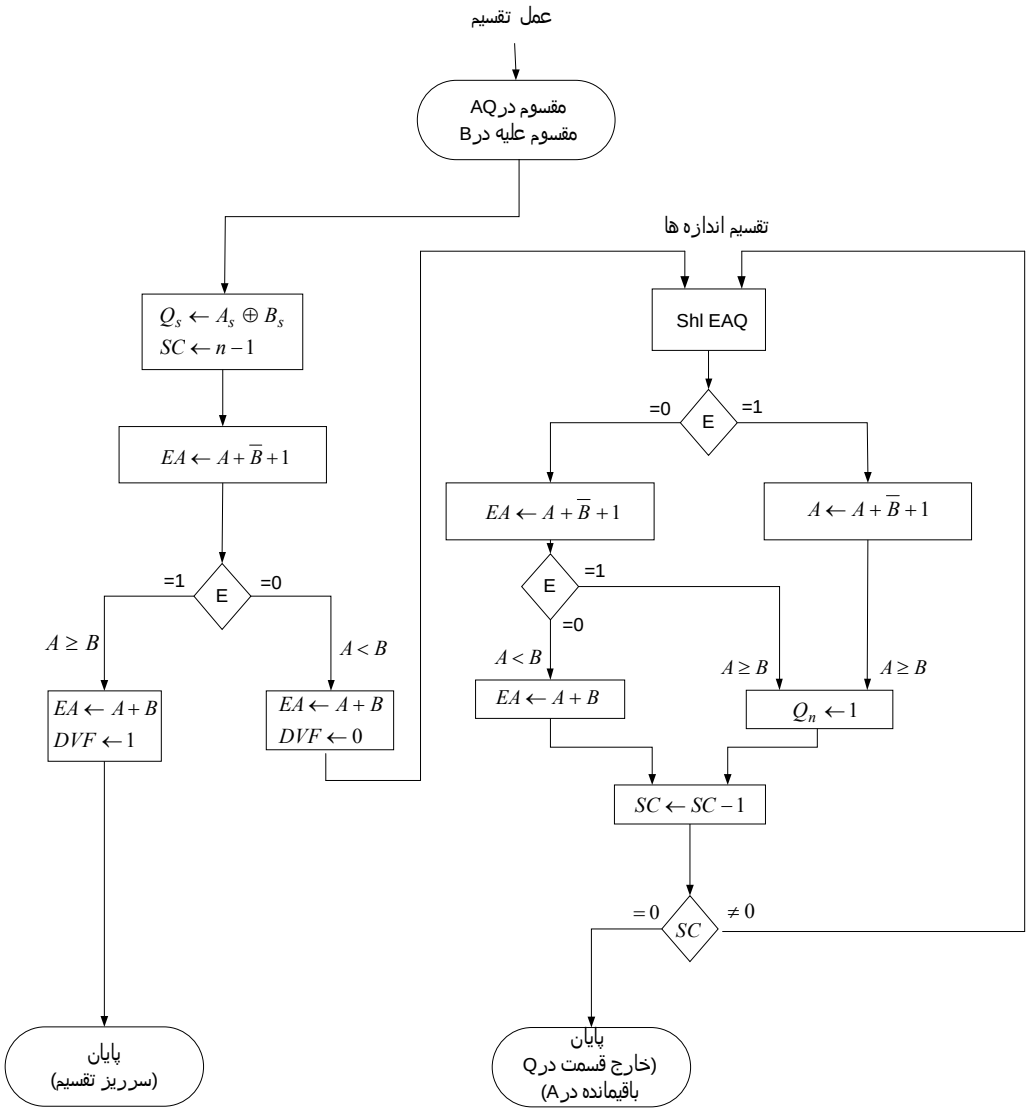
باشد، خارج قسمت شش بیتی خواهد بود. خارج قسمت می‌باید در یک ثبات پنج بیتی استاندارد ذخیره شود، بنابراین بیت سرریز مستلزم یک فلیپ فلاپ اضافی برای ذخیره بیت ششم خواهد بود. در عملیات عادی کامپیوتر باید از به کارگیری این بیت سرریز خودداری شود زیرا در این صورت طول کل خارج قسمت بیش از مقداری خواهد بود که بتوان آن را به واحدهای حافظه‌ای که طول استاندارد دارند، یعنی ثباتها، منتقل کرد. در سخت‌افزار یا نرم‌افزار کامپیوتر و یا ترکیبی از هر دو باید امکاناتی را جهت آشکارسازی این وضعیت فراهم کرد.

اگر تعداد بیت‌های مقسوم دو برابر مقسوم علیه باشد، حالت سرریز را می‌توان به طریق زیر بیان کرد: حالت سرریز در تقسیم هنگامی اتفاق می‌افتد که بیت‌های نیمه پر ارزشتر مقسوم، عددی بزرگتر یا مساوی مقسوم علیه را تشکیل دهند. مسأله دیگری که باید در تقسیم در نظر داشت این است که از تقسیم بر صفر باید جلوگیری شود. مدار چک کننده وضعیت سرریز در تقسیم، مراقبت از این وضعیت را نیز می‌تواند بر عهده گیرد. این بدان علت است که اگر مقسوم علیه صفر باشد مقسوم قطعاً از مقسوم علیه بزرگتر و یا مساوی آن است. حالت سرریز معمولاً با 1 شدن فلیپ فلاپ خاصی مشخص می‌گردد و آن را فلیپ فلاپ سرریز خوانده و با DVF نشان می‌دهیم. در قبال وقوع سرریز در عملیات تقسیم، به طرق مختلف می‌توان عکس العمل نشان داد. در بعضی کامپیوترها مسؤولیت با برنامه‌نویس است تا پس از هر دستورالعمل تقسیم، DVF را چک کند. در این صورت می‌توان به زیر روالی انشعاب کرده و اقدامی اصلاحی همچون تغییر مقیاس داده‌ها برای جلوگیری از سرریز انجام داد. در برخی کامپیوترهای قدیمی‌تر، وقوع سرریز در تقسیم سبب توقف کامپیوتر می‌شد و این حالت، توقف تقسیم نامیده می‌شد. امروزه متوقف کردن کار کامپیوتر توصیه نمی‌شود زیرا اتلاف کننده وقت است. روشی که در اکثر کامپیوترها انجام می‌شود تولید درخواست وقفه به هنگام 1 شدن DVF است. این وقفه موجب می‌شود تا کامپیوتر اجرای برنامه جاری را به تعویق انداخته و به روال سرویس‌دهی برای اقدام اصلاحی مورد تقاضا انشعاب نماید. معمول‌ترین اقدام اصلاحی خارج شدن از برنامه و چاپ پیغام خطائی است که دلیل عدم امکان اتمام برنامه را شرح می‌دهد. از این پس، وظیفه کاربری که برنامه را نوشته این است که مقیاس داده‌ها را تغییر دهد و یا هر اقدام اصلاحی دیگری را به عمل آورد. بهترین راه جلوگیری از سرریز تقسیم استفاده از داده‌های ممیز شناور است. در بخش مربوط به محاسبات ممیز

شناور خواهیم دید که سرریز تقسیم به سادگی به هنگام استفاده از اعداد ممیز شناور قابل پیش‌گیری است.

۴-۶-۳- **الگوریتم** سخت‌افزاری عملیات تقسیم

الگوریتم سخت‌افزاری عملیات تقسیم در فلوچارت شکل ۲۳ نشان داده شده است. مقسوم در A و Q، و مقسوم‌علیه در B قرار دارد. در ابتدا علامت نتیجه به عنوان قسمتی از خارج قسمت به Q_s منتقل می‌شود. برای مشخص کردن تعداد بیت‌های خارج قسمت، عدد ثابتی (برای مثال بالایی عدد ۵) در توالی شمار SC قرار داده می‌شود.



شکل ۲۳: فلوچارت عمل تقسیم

فرض می‌کنیم که عملوندها از واحد حافظه n بیتی به ثبات‌ها منتقل شوند. چون عملوند همراه با علامتش ذخیره می‌شود، یک بیت از کلمه حافظه بوسیله علامت و n-1 بیت به وسیله اندازه آن اشغال می‌گردد. حالت سرریز تقسیم با تفریق مقسوم‌علیه درB از نیمی از بیت‌های ذخیره شده در مقسوم یعنی A تست می‌شود. اگر $A \geq B$ باشد فلیپ فلاپ سرریز تقسیم (DVF) برابر 1 شده و عمل به طور ناقص متوقف می‌شود. تقسیم اندازه‌های دو عدد با جابه‌جا کردن مقسوم موجود در AQ به چپ شروع می‌شود که در این جابجایی بیت پر ارزشتر به E منتقل می‌شود اگر بیت انتقال یافته به E برابر 1 باشد، درمی‌یابیم که EA>B است زیرا EA از رقم 1 و به دنبال آن n-1 بیت تشکیل شده است، در حالیکه B فقط n-1 بیت است.

اگر عمل شیفت به چپ مقدار 0 را وارد E کند، قسمت بالایی مقسوم با متمم 2 مقسوم‌علیه جمع می‌شود (عمل تفریق A-B) و مقدار رقم نقلی به E انتقال می‌یابد. اگر E=1 شود، $A \geq B$ است، بنابراین بیت 1 به Q_n منتقل خواهد شد. اگر E=0 باشد، A<B است و عدد اولیه مقسوم، با جمع B با A، دوباره بازیابی می‌شود. در این حالت مقدار 0 را در Q_n قرار می‌دهیم.

این روند مجدداً تکرار می‌گردد. پس از n-1 تکرار، اندازه خارج‌قسمت در ثبات Q و باقیمانده در ثبات A خواهد بود. علامت خارج‌قسمت در Q_s و علامت باقیمانده نیز که همان علامت مقسوم است در A_s قرار دارد.

۴-۶-۴- عملیات تقسیم در پردازنده MIPS

پردازنده MIPS برای عملیات تقسیم نیز از رجیسترهای Hi و Lo استفاده می‌کند. بعد از انجام عملیات تقسیم، مقدار باقیمانده تقسیم داخل رجیستر Hi و مقدار خارج قسمت تقسیم داخل رجیستر Lo ریخته می‌شود. برای خواندن محتوای این رجیسترها و انتقال آنها به داخل رجیسترهای عمومی پردازنده از دستورهای mfhi و mflo استفاده می‌شود. MIPS برای تقسیم اعداد صحیح علامت‌دار و بدون علامت، به ترتیب از دستورهای div[1] و divu[2] استفاده می‌کند.

[1] - Divide
[2] Divide unsigned

۴-۷- انجام عملیات ضرب و تقسیم به کمک دستورهای شیفت

با توجه به مطالبی که در ارتباط با الگوریتم‌های سخت‌افزاری و پیاده سازی عملیات ضرب و تقسیم بیان شد، می‌توان این‌طور نتیجه گرفت که دستورات ضرب و تقسیم دستوراتی هستند که دارای CPI بالاتری نسبت به دستوراتی مثل جمع و تفریق می‌باشند. بنابراین در یک برنامه هر چقدر تعداد دستورات ضرب و تقسیم بیشتر باشد به احتمال زیاد زمان اجرای برنامه بیشتر خواهد بود و برنامه دیرتر جواب خواهد داد. در مقام مقایسه زمان اجرای دستورات شیفت نیز از زمان اجرای دستورات ضرب و تقسیم کمتر است.

با توجه به مطالب فوق کامپایلرها تا حد امکان سعی می‌کنند دستورات ضرب و تقسیم کمتری تولید نمایند. در برخی موارد یکی از عملوندهای ضرب یا تقسیم، یک عدد ثابت است. در چنین مواردی کامپایلر سعی می‌کند که به جای دستور ضرب یا تقسیم از ترکیبی از عملیات شیفت، جمع و تفریق استفاده نماید. به مثال زیر توجه کنید.

مثال: دستورات ماشین را برای عملیات زیر طوری بنویسید که زمان اجرای کد کمترین باشد.

v0 = t0 * 5

جواب: می‌توانیم ۵ را به صورت 5=4+1 بنویسیم و در نتیجه داریم: v0 = t0*(4+1)=t0*4+t0 از آنجا که هر شیفت به چپ معادل یک ضرب در ۲ است می‌توان برای عملیات t0*4 ، t0 را به اندازه دو واحد به چپ شیفت داد. پس می‌توان کد اسمبلی را به صورت زیر نوشت:

```
sll $v0, $t0, 2  // v0 = t0 * 4
add $v0, $v0, $t0  // v0 = t0 *4 + t0
```

اگر فرض کنیم که دستورات add و sll در یک کلاک و دستور ضرب در ۳۲ کلاک انجام بگیرد، در این صورت کدی که نوشته شده، در ۲ کلاک اجرا خواهد شد. اما اگر از دستور ضرب در نوشتن این کد استفاده کنیم، زمان اجرا مساوی ۳۲ کلاک خواهد شد.

پس در نتیجه، می‌توانیم عددهای ثابت را به جمع و تفریقی از توانهای ۲ تبدیل کنیم و هر کدام از عملیاتی را که یکی از عملوندهای آن توانی از ۲ است را با عملیات شیفت پیاده سازی کنیم. با انجام

این کار می‌توان عملیات ضرب و تقسیم مورد نظر را با ترکیبی از دستورات add، sub و شیفت پیاده سازی نمود و بدین طریق بهبود قابل ملاحظه‌ای در زمان اجرا به وجود آورد.

۴-۸- عملیات حسابی ممیز شناور

بسیاری از زبانهای برنامه‌نویسی امکاناتی برای مشخص کردن عددهای ممیز شناور دارند. متداول‌ترین روش، مشخص کردن آنها با عبارت real (حقیقی) در برابر اعداد ممیز ثابت است که با عبارت Integer (صحیح) مشخص می‌گردند. هر کامپیوتری که دارای کامپایلر برای اینگونه زبانهای سطح بالا باشد باید امکاناتی برای انجام عمل‌های حسابی ممیز شناور داشته باشد. این عملیات غالباً در سخت‌افزار کامپیوتر پیاده‌سازی می‌شوند. اگر سخت‌افزاری برای این اعمال وجود نداشته باشد، کامپایلر باید طوری طراحی شود که دارای مجموعه‌ای از زیر روال‌های نرم‌افزاری ممیز شناور باشد یعنی این که دستورات ممیز شناور در این زیر روال‌ها با دستورات دیگر پیاده سازی گردد. هر چند روش سخت افزاری گرانتر است، ولی کارآیی آن به قدری از روش نرم افزاری بیشتر است که در اکثر کامپیوترها، سخت‌افزار ممیز شناور کار گذاشته می‌شود و فقط در کامپیوترهای خیلی کوچک از آن صرف‌نظر می‌شود.

۴-۸-۱- مفاهیم اساسی

یک عدد ممیز شناور در ثباتهای کامپیوتر از دو بخش تشکیل می‌شود: مانتیس m و نمای e. این دو بخش نماینده عددی است که از ضرب m در r به نمای e بدست می‌آید. یعنی:

$$m \times r^e$$

مانتیس ممکن است که عددی کسری یا صحیح باشد. اطلاعات محل ممیز و مقدار پایه r در ثباتها وارد نمی‌شوند و در محاسبات، این اطلاعات مقدار پیش فرضی دارند. به طور مثال نمایش کسری و پایه 10 را در نظر بگیرید. عدد دهدهی 537.25 در یک ثبات با m=53725 و e=3 نمایش داده می‌شود و چنین تفسیر می‌گردد که نماینده عدد ممیز شناور زیر است:

$$0.53725 \times 10^3$$

اگر با ارزشترین رقم مانتیس یک عدد ممیز شناور، غیر صفر باشد آن را نرمالیزه می‌کنیم. در نتیجه مانتیس دارای بیشترین تعداد ممکن رقم‌های معنی‌دار خواهد بود. صفر را نمی‌توان نرمالیزه کرد زیرا رقم غیر صفر ندارد. در نمایش ممیز شناور، صفر را با مقدار تمام 0 در مانتیس و نما نمایش می-دهیم.

نمایش ممیز شناور محدوده اعدادی را که می‌توان در یک ثبات جای داد مشخص می‌کند. کامپیوتری با کلمه‌های 48 بیتی را در نظر بگیرید. چون یک بیت برای علامت رزرو شده‌است محدوده اعداد صحیح ممیز ثابت $\pm\left(2^{47}-1\right)$ خواهد بود که تقریباً $\pm 10^{14}$ می‌باشد. از این48 بیت می‌توان برای نمایش یک عدد میز شناور با 36 بیت برای مانتیس و 12 بیت برای نما استفاده کرد. با فرض نمایش کسری برای مانتیس و با انتساب دو بیت برای علامت‌ها، محدوده اعدادی که می‌توان جای داد برابر است با:

$$\pm\left(1-2^{-35}\right)\times 2^{2047}$$

این عدد از کسری که حاوی 35 رقم 1، نمای 11 بیتی (منهای علامت آن) و اینکه $2047 = 1- 2^{11}$می‌باشد، بدست می‌آید. بزرگترین عددی که در ۴۸ بیت می‌توان جای داد 10^{615} است، که عدد بسیار بزرگی است. مانتیس می‌تواند 35 بیتی باشد (جدا از علامت) و اگر عدد صحیح تلقی شود می‌تواند تا $(2^{35}-1)$را ذخیره نماید. این تقریباً برابر با 10^{10} می‌باشد، که معادل یک عدد دهدهی10 رقمی است. کامپیوترهایی که دارای طول کلمه‌های کوتاهتری هستند برای نمایش اعداد ممیز شناور از دو یا چند کلمه استفاده می‌نمایند. یک میکروکامپیوتر 8 بیتی ممکن است از چهار کلمه برای نمایش یک عدد ممیز شناور استفاده کند که یکی از کلمات 8 بیتی برای نما و سه کلمه (24 بیت) دیگر برای مانتیس در نظر گرفته شوند.

اعمال حسابی با اعداد ممیز شناور بسیار پیچیده‌تر از اعداد ممیز ثابت است واجرای آنها زمان بیشتری می‌برد و سخت‌افزار پیچیده‌تری نیاز دارد. جمع یا تفریق دو عدد مستلزم معین کردن محل ممیز است زیرا قبل از جمع یا تفریق مانتیس‌ها، باید بخش نمای دو عدد با هم مساوی شود. هم ردیف کردن با جابه‌جایی یکی از مانتیس‌ها و تنظیم نمای آن تا جایی که با دیگری برابر شود انجام می‌شود. به طور مثال جمع اعداد ممیز شناور زیر را در نظر بگیرید.

$$0.5372400\times10^{2}$$

$$+\ 0.1580000\times10^{-1}$$

لازم است که دو نما قبل از جمع مانتیس‌ها با هم مساوی شوند. می‌توانیم عدد اول را سه مکان به چپ، یا عدد دوم را سه مکان به راست شیفت دهیم. وقتی که مانتیس‌ها در ثبات‌ها ذخیره شوند، شیفت به چپ سبب از دست دادن ارقام با ارزشتر می‌شود. روش دوم ترجیح دارد چون فقط میزان دقت را کاهش می‌دهد در حالی که روش اول ممکن است موجب بروز خطا شود. شیوه هم ردیف کردن معمولاً به این ترتیب است که مانتیسی را که نمای کوچکتر دارد به اندازه اختلاف بین نماها به راست شیفت می‌دهیم. پس از این عمل، مانتیس‌ها می‌توانند با هم جمع شوند.

$$0.5372400\times10^{2}$$

$$+\ 0.0001580\times10^{2}$$

$$0.5373980\times10^{2}$$

هنگامی که دو مانتیس نرمالیزه شده با هم جمع شوند حاصل‌جمع ممکن است رقم سرریز داشته باشد. سرریز را به سادگی می‌توان با شیفت حاصل‌جمع به راست و افزایش نما تصحیح کرد. در تفریق مانند مثال زیر نتیجه ممکن است دارای صفرهایی در مکان‌های بالا رتبه باشد.

$$0.56780\times10^{5}$$

$$-\ 0.56430\times10^{5}$$

$$0.00350\times10^{5}$$

وقتی که عدد ممیز شناوری دارای 0 در مکان‌های با ارزشتر باشد گوئیم فروریز[1] دارد. برای نرمالیزه کردن اعداد فروریز دار، لازم است تا مانتیس به چپ شیفت داده شود و از نما یک واحد کسر گردد تا رقم غیر صفر در اولین مکان ظاهر شود. در مثال فوق، لازم است تا مانتیس را دوبار به چپ شیفت دهیم تا 0.35000×10^{3} حاصل گردد. در بسیاری از کامپیوترها، نرمالیزه کردن پس از هر عمل صورت می‌گیرد تا از نرمالیزه شدن نتایج اطمینان حاصل شود.

ضرب و تقسیم ممیز شناور، نیازی به هم ردیف کردن مانتیس‌ها ندارند. حاصلضرب با ضرب دو مانتیس و جمع نماها شکل می‌گیرد. عمل تقسیم از تقسیم دو مانتیس و تفریق نماها بدست می‌آید. نکته‌ای که در این قسمت باید به آن تأکید کنیم این است که در محاسبات ممیز شناور اعمال اجرا شده با مانتیس‌ها مشابه اعداد ممیز ثابت است، بنابراین هر دو نوع می‌توانند از ثباتها و مدارهای مشترکی استفاده کنند. این در حالی است که عملیات مربوط به نما در محاسبات ممیز شناور عبارتند از: مقایسه و افزایش (برای هم ردیفی مانتیس‌ها)، جمع و تفریق (برای ضرب و تقسیم)، و کاهش (برای نرمالیزه کردن نتایج).

نما را می‌توان به یکی از سه روش ممکن نمایش داد: مقدار علامت‌دار، متمم 2 علامت‌دار و متمم 1 علامت‌دار.

چهارمین روشی که در بسیاری از کامپیوترها بکار گرفته می‌شود نمای خورانده شده یا بایاس شده است. در این نمایش، بیت علامت به عنوان یک عضو جدا در نظر گرفته نمی‌شود. بایاس عدد مثبتی است که به هنگام تشکیل عدد ممیز شناور به هر نما اضافه می‌شود. در نتیجه تمام نماها در درون مثبت خواهند بود. مثال زیر می‌تواند این نوع نمایش را روشن سازد. فرض کنید نماها از 50- تا 49+ متغیر باشند. در این صورت ثبات نما حاوی عدد e+50 خواهد بود که e نمای واقعی است و 50 مقدار بایاس می‌باشد. لذا نماها در ثبات به صورت اعداد مثبتی از 00 تا 99 نمایش داده خواهند شد. نماهای مثبت در ثبات‌ها محدوده اعداد 50 تا 99 و نماهای منفی محدوده 00 تا 49 خواهند داشت.

[1] - Underflow

مزیت استفاده از نماهای بایاس شده این است که فقط اعداد مثبت را شامل می‌شوند. در نتیجه مقایسه نسبت اندازه‌های آنها ساده‌تر بوده و لازم نیست به علامت آنها توجه کنیم. لذا هنگام هم ردیف کردن مانتیس‌ها می‌توان از مقایسه‌گر مقدار (اندازه) برای مقایسه اندازه‌های آنها استفاده کرد. مزیت دیگر آنها این است که کوچکترین نمای ممکن، تمام 0 است.

در مثال‌های بالا، از عددهای دهدهی برای نمایش دادن برخی از مفاهیمی که هنگام کار با اعداد ممیز شناور باید آنها را دانست استفاده کردیم. واضح است که مفاهیم مشابهی در مورد اعداد دودویی نیز صادق است.

۴-۸-۲- جمع و تفریق

الگوریتم جمع و تفریق ممیز شناور می‌تواند به چهار بخش متوالی تقسیم گردد:

۱. تست صفر بودن عملوندها
۲. هم ردیف کردن مانتیس‌ها
۳. جمع یا تفریق مانتیس‌ها
۴. نرمالیزه کردن نتیجه

عدد ممیز شناوری که صفر باشد قابل نرمالیزه شدن نیست. اگر این عدد طی محاسبات مورد استفاده قرار گیرد، نتیجه نیز ممکن است صفر شود. به جای تست صفر حین نرمالیزه کردن، در ابتدای کار صفر را چک می‌کنیم و در صورت لزوم پردازش را خاتمه می‌دهیم. هم ردیف کردن مانتیس‌ها باید قبل از اجرای اعمال جمع و تفریق باشد. پس از جمع یا تفریق مانتیس‌ها، نتیجه ممکن است غیر نرمالیزه باشد. با به کارگیری روش نرمالیزه کردن، نتیجه قبل از انتقال به حافظه نرمالیزه می‌شود.

۴-۸-۳- ضرب

ضرب دو عدد ممیز شناور با ضرب مانتیس‌ها و جمع نماها انجام می‌شود. در این عمل مقایسه نماها و هم ردیف کردن مانتیس‌ها قبل از عملیات لزومی ندارد. ضرب مانتیس‌ها مشابه ضرب اعداد

ممیز ثابت است و حاصلضرب با دقتی دو برابر حاصل می‌شود. پاسخ حاصل با دقتی مضاعف[1] در ممیز ثابت برای افزایش دقت حاصلضرب به کار می‌رود. در نمایش ممیز شناور، محدوده دقت معمولی[2] همراه با نما به اندازه کافی دقت دارد و بنابراین اعداد به صورت غیر مضاعف یا معمولی نگهداری می‌شوند. لذا نیمه با ارزشتر حاصلضرب مانتیس‌ها برداشته شده و در کنار نمای حاصل شده حاصلضربی با دقت معمولی برای ممیز شناور به وجود می‌آید.

الگوریتم ضرب به چهار بخش زیر تقسیم می‌شود:

۱. چک کردن صفر
۲. جمع کردن نماها
۳. ضرب مانتیس‌ها
۴. نرمالیزه کردن حاصلضرب

مراحل ۲ و ۳ می‌توانند به طور همزمان انجام شوند مشروط بر اینکه برای مانتیس‌ها و نماها، جمع‌کننده‌های جداگانه‌ای وجود داشته باشند.

در عملیات ضرب ممیز شناور سرریز نمی‌تواند رخ دهد، لذا نیازی به چک کردن آن نیست. فقط باید بعد از عملیات، نرمال سازی صورت گیرد.

۴-۸-۴- تقسیم

در تقسیم ممیز شناور نماها تفریق و مانتیس‌ها تقسیم می‌شوند. تقسیم مانتیس مشابه ممیز ثابت است جز اینکه مقسوم مانتیسی با دقت معمولی دارد. به خاطر بسپارید که مانتیس مقسوم کسری است و نه عدد صحیح.

تست سرریز تقسیم همانند نمایش ممیز ثابت است. با این وجود سرریز تقسیم در ممیز شناور مشکلی ایجاد نمی‌کند. اگر مقسوم بزرگتر یا مساوی مقسوم‌علیه باشد، کسر مقسوم به راست شیفت

[1] - Double precision
[2] - Single precision

داده می‌شود و نمای آن یک واحد افزایش می‌یابد. برای عملوندهای نرمالیزه شده این عمل برای تضمین عدم وقوع سر ریز تقسیم کافی است. عمل فوق هم ردیف کردن مقسوم خوانده می‌شود.

تقسیم دو عدد ممیز شناور نرمالیزه شده همواره خارج‌قسمت نرمالیزه شده‌ای را به دست می‌دهد به شرطی که هم ردیف کردن مقسوم قبل از تقسیم انجام شده باشد. بنابراین بر خلاف سایر اعمال، خارج‌قسمت حاصل از تقسیم نیازی به نرمالیزه شدن ندارد.

الگوریتم تقسیم می‌تواند به پنج بخش زیر تقسیم شود:

۱. چک کردن برای وجود صفر
۲. مقدار دهی اولیه برای ثباتها و تعیین علامت
۳. هم ردیف کردن مقسوم
۴. تفریق نمادها
۵. تقسیم مانتیس‌ها

توجه: در عمل تقسیم ممیز شناور به هنگام تفریق نمای مقسوم‌علیه از نمای مقسوم، چون هر دو نما در ابتدا بایاس شده هستند، با عمل تفریق تفاضل بایاس نشده آنها بدست می‌آید. پس باید بعد از این تفریق دوباره عدد بایاس به نتیجه اضافه شود.