دانشگاه صنعتی شریف
دانشکده مهندسی کامپیوتر

گزارش پروژه
درس معماری کامپیوتر

طراحی واحد ممیز شناور (FPU)

نگارش

فاطیما تیمارچی - ۴۰۲۱۰۵۸۰۲

سیده شقایق میرجلیلی - ۴۰۲۱۰۶۶۵۹

متین باقری - ۴۰۲۱۰۵۷۲۷

سیداحمد موسوی اول - ۴۰۲۱۰۶۶۴۸

استاد

دکتر حسین اسدی

تیر ماه ۱۴۰۴

فهرست مطالب

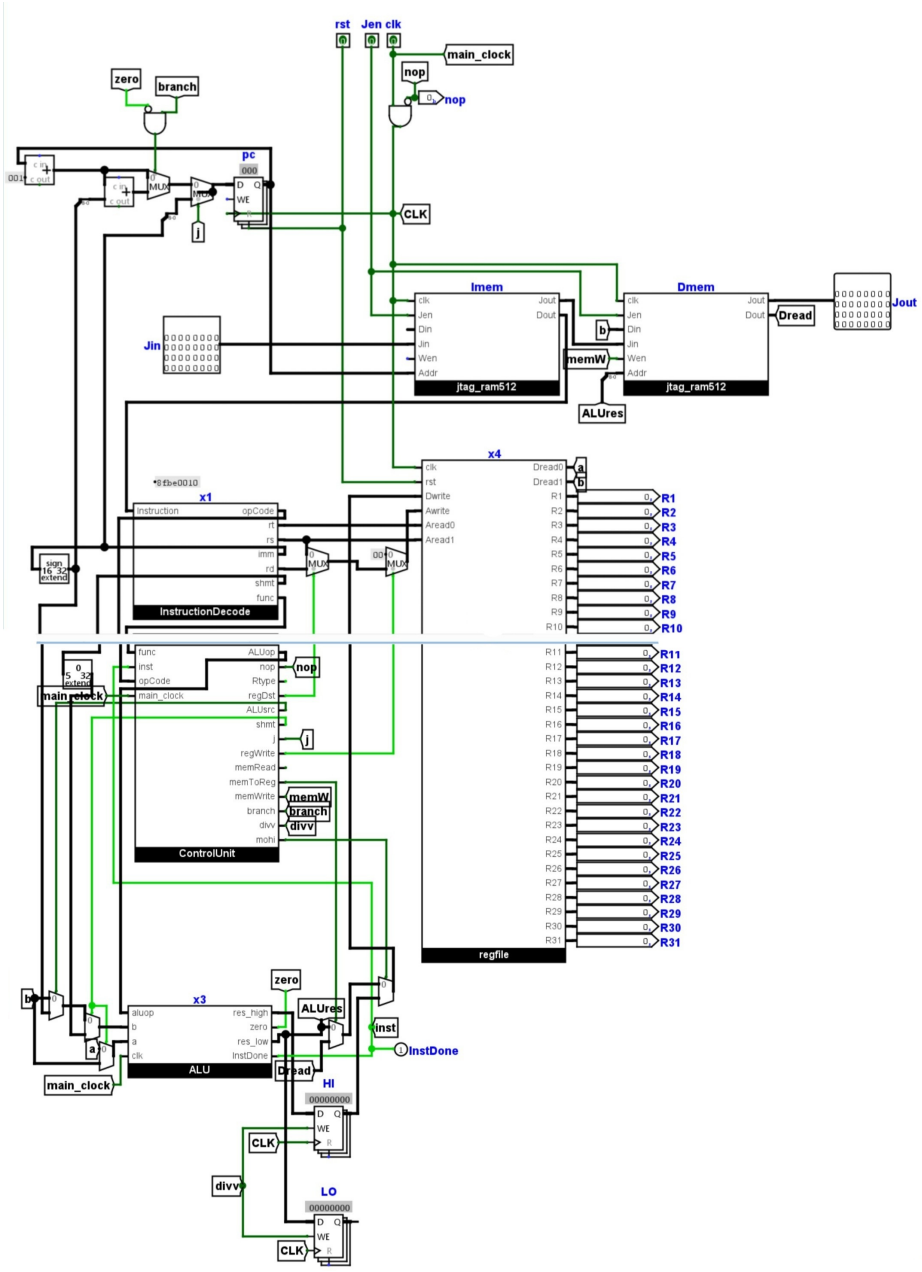
فهرست تصاویر

فصل ۱

ساختار کلی مدار

۱-۱ مدار پایه

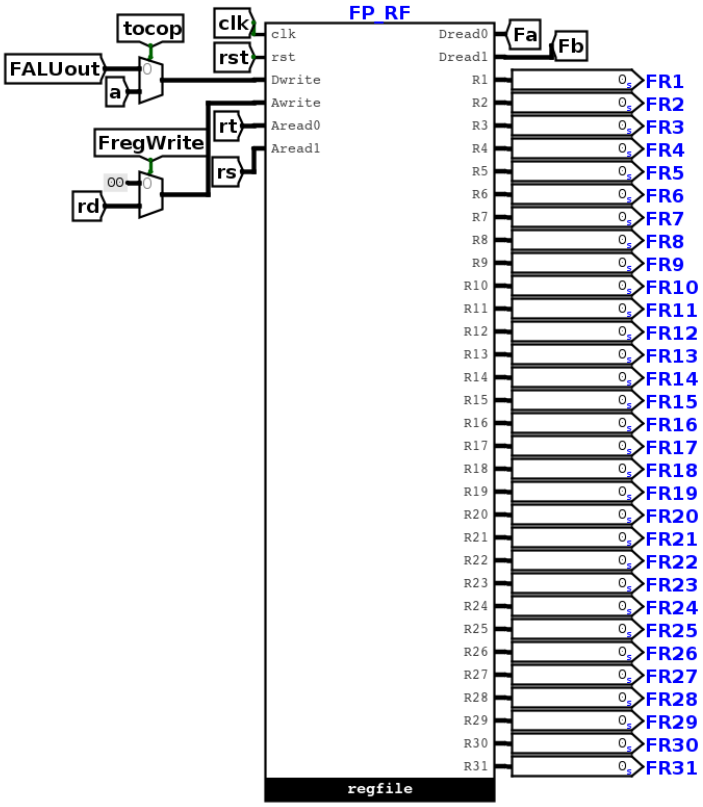
همانطور که خواسته شده بود، مدار این پروژه توسعه داده شده مدار تمرین ۵ (تصویر ۱-۱) است که نسخه تکمیل شده MIPS-Single-Cycle بود. ابتدا از صحت عملکرد مدار اولیه مطمئن شدیم و سپس کمی مدار موجود را مرتب کردیم. برای این کار سعی کردیم در حد امکان تعداد اجزای مدار اصلی را کاهش دهیم و برای کاهش سیم کشی ها از tunnel استفاده کنیم.



شکل ۱-۱: مدار اصلی تمرین ۵

۱-۲ بانک ثبات ممیز شناور

از مدل بانک ثبات موجود استفاده کرده و یک بانک ثبات مخصوص واحد ممیز شناور (تصویر ۱-۲) به مدار اصلی اضافه کردیم. آدرس هایی که از آن‌ها می‌خوانیم rs و rt هستند. اگر قصد نوشتن در این بانک ثبات را داشته باشیم در ثبات rd و در غیر این صورت در ثبات ۰ می‌نویسیم(مشابه آنچه در تمرین ۵ از ما خواسته شده بود). داده‌ای که در بانک ثبات ممیز شناور نوشته می‌شود، در صورت TOcop بودن دستور،

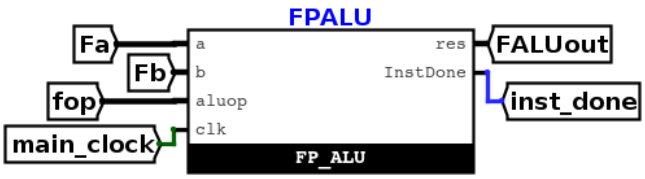


شکل ۱-۲: بانک ثبات واحد ممیز شناور

داده‌ای است که از بانک ثبات اصلی خوانده شده، و در غیر این صورت خروجی ALU واحد ممیز شناور است.

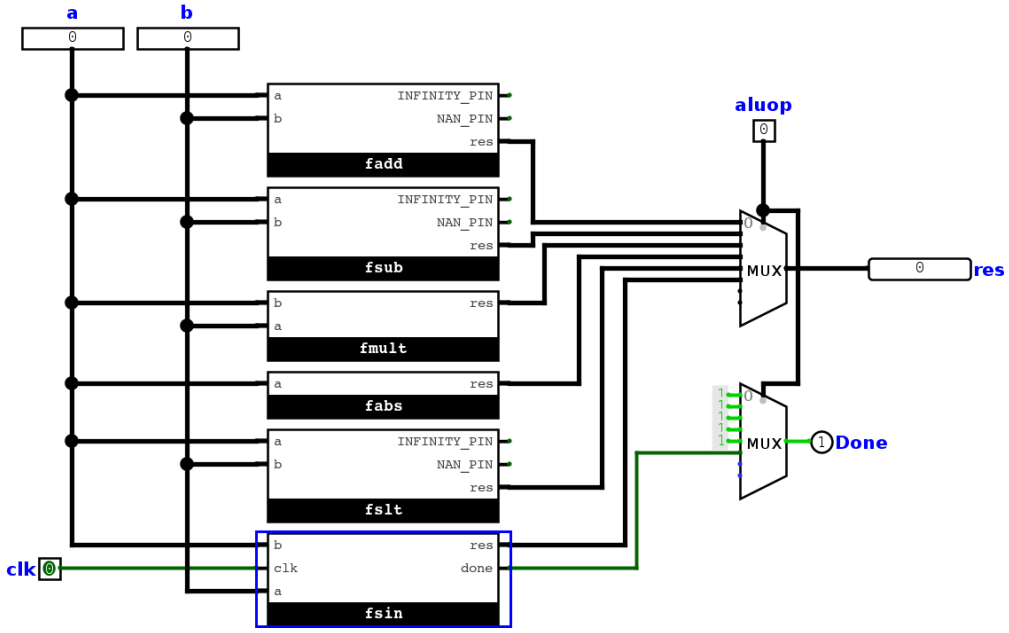
۱-۳ ALU واحد ممیز شناور (FPALU)

واحد محاسباتی بخش ممیز شناور (تصویر ۱-۳) از بانک ثبات این واحد ورودی گرفته و خروجی آن نیز به بانک ثبات باز می‌گردد.



شکل ۱-۳: واحد محاسباتی ممیز شناور در مدار اصلی

FPALU (تصویر ۱-۴) خود از تعدادی بخش تشکیل شده که هر یک وظیفه انجام یک نوع محاسبه خاص را بر عهده دارند. خروجی بر اساس code operation که توسط واحد کنترلی تعیین می‌شود، انتخاب می‌شود. در فصل بعد به بررسی دقیق هر یک از این زیربخش‌ها خواهیم پرداخت.



شکل ۱-۴: نمای داخلی ALU ممیز شناور

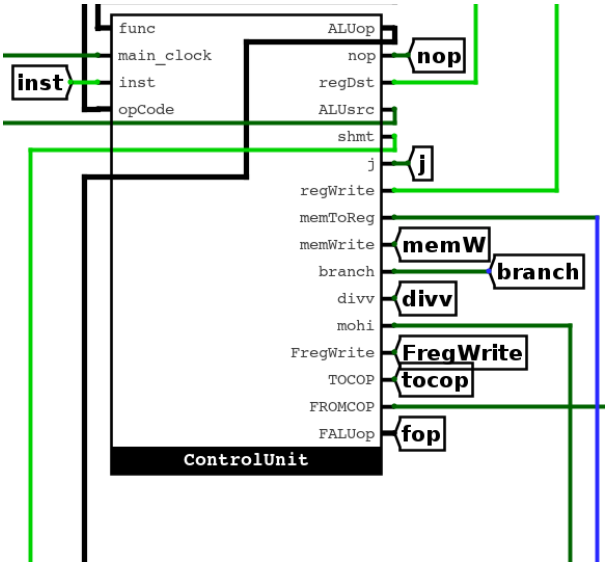
۱-۴ واحد کنترلی (ControlUnit)

برای کنترل بخش ممیز شناوری که به مدار پایه اضافه کردیم، واحد کنترلی آن را به نحوی گسترش دادیم که سیگنال های کنترلی زیر را نیز تولید کند:

FregWrite مشخص می‌کند آیا قصد نوشتن در FPRF را داریم یا خیر

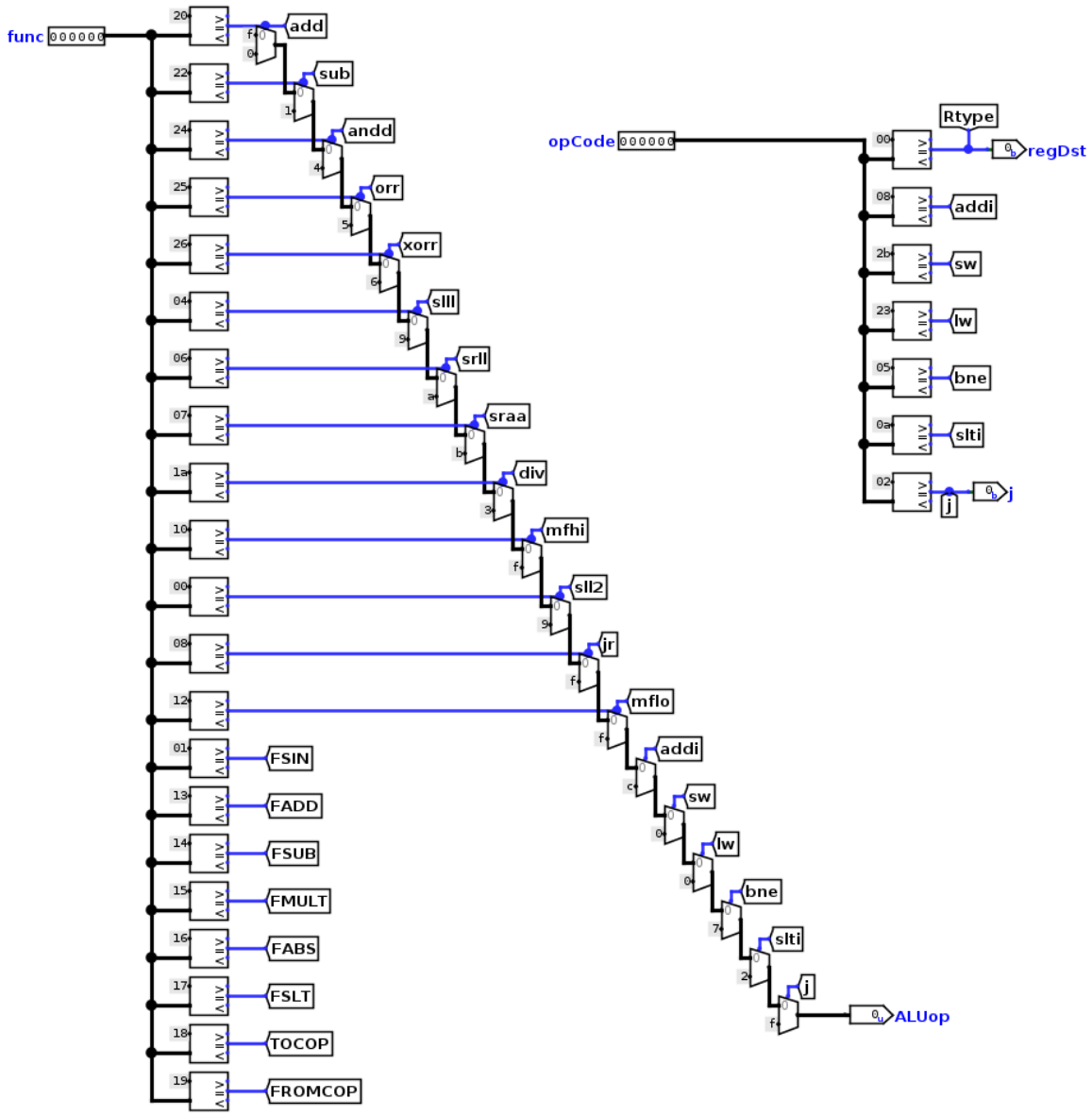
TOcop و FROMcop دستورات انتقال داده بین بانک‌های ثبات را تشخیص می‌دهند

FALUop مشخص می‌کند FPALU چه محاسبه‌ای انجام دهد

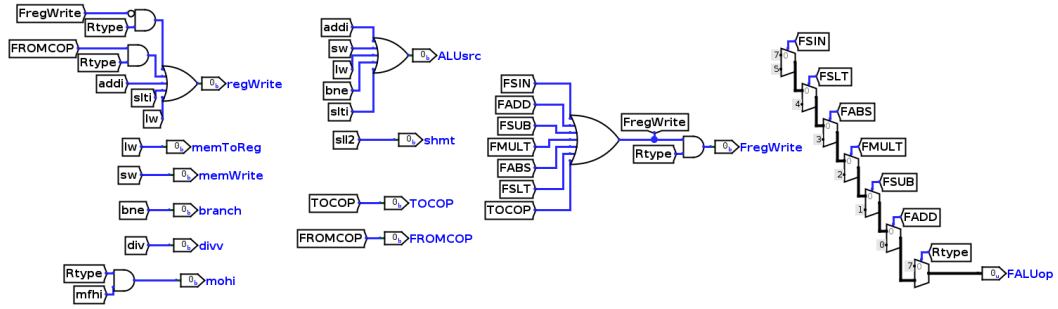


شکل ۱-۵: واحد کنترلی

نحوه کار ControlUnit به این صورت است که ابتدا بر اساس opCode و functionCode دستور، نوع دستور را شناسایی می‌کند (تصویر ۱-۶) و سپس بر اساس آن سیگنال‌های مورد نیاز را تولید میکند (تصویر ۱-۷).



شکل ۱-۶: واحد کنترلی: شناسایی دستور



شکل ۱-۷: واحد کنترلی: تولید سیگنال‌های کنترلی

فصل ۲

ALU

fadd ۱-۲

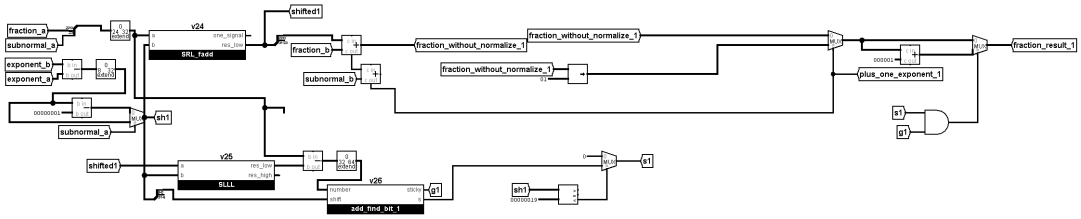
برای کشیدن مدار fadd این مدار را به دو قسمت تقسیم کردم . بخش اول برای حالتی است که sign دو عدد داده شده یکسان باشد و بخش دوم نیز برای حالتی است که sign دو عدد داده شده متفاوت باشد . بخش اول در مدار fadd پیاده سازی شده است و بخش دوم را در مدار fMinnus پیاده سازی شده است . در ابتدای هر دو مدار با استفاده از مالتی پلکسر و گیت های منطقی fadd و fMinnus من مشخص کردم که اگر هر کدام از اعداد برابر NAN یا INFINITY باشد بنابراین حاصل آن جمع نیز به ترتیب برابر NAN و INFINITY میشود و به ترتیب سیگنال های NAN-PIN و INFINIY-PIN فعال میشود .

مدار fadd ۱-۱-۲

در این مدار جمع را در سه حالت حساب کرده و با استفاده از مالتی پلکسر بین حاصل های به دست آمده حاصل صحیح را انتخاب میشود

۱. exponent b > exponent a در این حالت در ابتدا توجه میکنیم که آیا exponent عدد ما صفر هست یا نه تا بفهمیم عدد ما نرمال هست یا خیر فهمیدن این موضوع را با استفاده از سیگنال subnormal انجام دادم که اگر این سیگنال برابر یک بود آنگاه عدد ما نرمال است و اگر هم برابر صفر بود بنابراین عدد ما نرمال نیست . سپس باید fraction عدد a را به اندازه exponent a − exponent b شیفت دهیم تا exponent هر دو یکسان شود . بنابراین exponent نهایی برابر

exponent بزرگ تر یعنی برابر exponent b میشود .سپس دو عدد را با هم جمع میکنیم حال باید بفهمیم که عدد به دست آمده نرمال هست یا نه فهمیدن نرمال بودن این عدد هم به این صورت هست که اگر در جمع ما carry داشته باشیم عدد ما باید normalize شود . برای normalize کردن عدد هم کافی است آن عدد را یک واحد به سمت راست شیفت بدیم و به exponent یک واحد بیفزاییم . در انتها نیز مرحله رند کردن عدد رو داریم در این مرحله باید بفهمیم که بیت بعد از کم ارزش ترین بیت fraction یعنی g۱ چیست و آیا بعد از g۱ بیت یکی وجود دارد یا خیر اگر g۱ برابر یک باشد و بیت یکی بعد از g۱ وجود داشته باشد عدد ما به بالا گرد میشود و در غیر این صورت عدد ما به پایین گرد میشود که اینکار را مدار add-find-bit-۱ انجام میدهد که در این مدار با استفاده از selector bit و subtractor مقدار g۱ و s۱ را می یابد . اگر بیت یکی بعد از g۱ وجود داشته باشد مقدار s۱ برابر ۱ میشود . عکس این قسمت مدار را میتوانید در قسمت پایین ببینید.



شکل ۲-۱: exopnent of fadd

۲. exponent b < exponent a

در این حالت مراحل دقیقا مطابق مراحل قبل است با این تفاوت که fraction عدد b را به اندازه b exponent a – exponent شیفت می دهیم و بیت بعد از کم ارزش ترین بیت g۲ نام دارد.

۳. exponent b = exponent a

در این حالت چون exponent ها یکسان هستند بنابراین هیچکدام از اعداد a یا b نیاز به شیفت ندارند و تنها کافی است Fraction دو عدد را با هم جمع کنیم و اگر هم نیاز باشد حاصل را normalize میکنیم .

۲-۱-۲ مدار fMinnus

کلیات مدار fMinnus دقیقا مطابق مدار fadd هست اما چند تفاوت دارد که در ادامه به آن ها اشاره میکنم . اول اینکه در این مدار برای منها کردن دو fraction باید حاصل مقدار مثبت شود بنابراین باید با توجه به

اینکه کدام عدد بزرگ تر هست عملیات تفریق را انجام دهیم که برای اینکار من دو حالت تفریق را حساب کردم و در انتها با توجه به exponent ها بین حاصل های به دست آمده انتخاب کردم . تفاوت دوم نیز این هست که اگر یک عدد به واسطه تفاوت exponent ها لازم باشد ۶۴ بار شیفت راست بخورد آن عدد برابر صفر تلقی میشود . تفاوت سوم نیز در مرحله normalize کردن هست که در این مرحله برخلاف حالت جمع نمیتونیم ساده بفهمیم که عدد ما normalize میخواد یا نه برای فهمیدن اینکار باید بفهمیم اولین بیتی که برابر یک است از سمت چپ در چه جایگاهی قرار دارد تا آن بیت یک را به تعداد لازم شیفت دهیم تا در بیت ۳۱ ام قرار گیرد برای فهمیدن جایگاه پر ارزش ترین بیت ۱ مداری به نام bitone ساختم که در این مدار با استفاده از AND و مقایسه کننده و مالتی پلکسر بر همه بیت های عدد را بررسی میکند تا بیت پرارزشی که برابر یک هست را بیابد . تفاوت چهارم نیز در مرحله رند کردن بود که در عملیات تفریق برخلاف عملیات جمع رند کردن عدد به مراتب پیچیده تر و سخت تر است . برای اینکار من یک مدار جدید به نام substract ساختم در این مدار مقدار fraction را ۶۴ بیت در نظر میگیرد و سپس عملیات منها را انجام میدهد در انتها میبیند که بیت بعد از کم ارزش ترین بیت در Fraction برابر چیست (sticky) و آیا بعد از بیت sticky بیت یکی وجود دارد یا خیر.با بررسی تمام بیت های عدد مورد نظر خود مقدار ۱bit را با استفاده از selector bit و subtractor و OR به دست می آوریم . حال اگر عدد به دست آمده حداقل یکی از حالاتی که در ادامه میگویم را داشته باشد آن عدد به سمت بالا رند میشود . حالات آن هم به این صورت است :

۱. کم ارزش ترین بیت و بیت sticky برابر یک باشند و ۱bit برابر صفر باشد

۲. کم ارزش ترین بیت برابر صفر باشد و بیت sticky و ۱bit برابر یک باشند

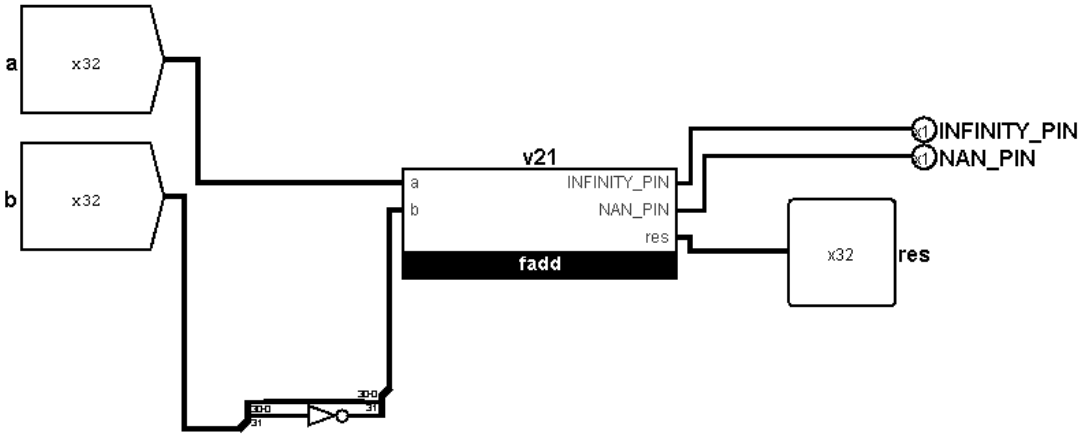
۳. کم ارزش ترین بیت و بیت sticky و ۱bit برابر یک باشند . در آخر نیز پس از رند به سمت بالا ممکن است عدد ما از حالت نرمال خارج شود پس با بررسی کردن cout میفهمیم که آیا عدد ما نیاز دارد که دوباره normalize شود یا خیر اگر نیاز بود آن را normalize میکنیم و مقدار exponent را نیز یک واحد افزایش میدهیم .

fsub ۲-۲

برای این مدار من از مدار fadd استفاده شده با این تفاوت که بیت sign عدد دوم را معکوس مینماییم .

$$a + b = a + (-b) \qquad (۱-۲)$$

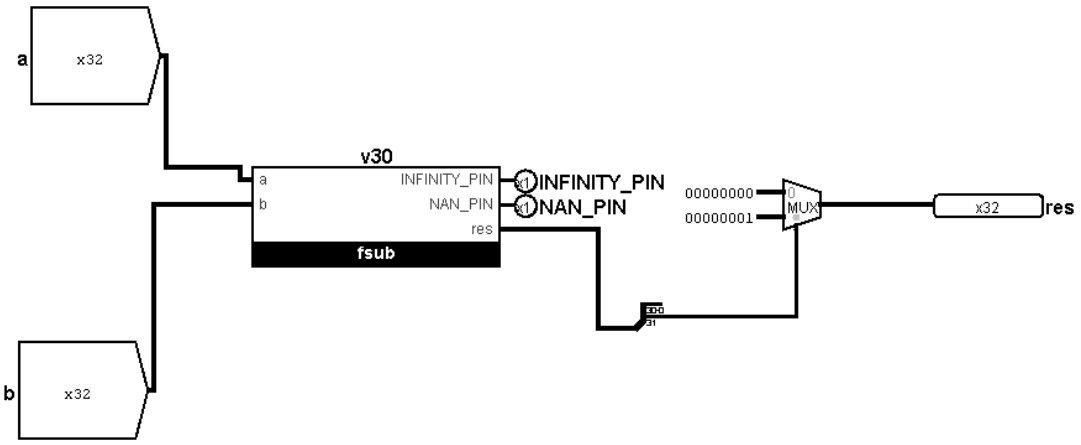
مدار این بخش از مدار را میتوانید در قسمت پایین ببینید .



شکل ۲-۲: قسمت fsub مدار

fslt ۳-۲

در این مدار ابتدا مقدار b را از a توسط مدار fsub کم شده و سپس مقدار sign حاصل را با استفاده از splitter جدا شده که اگر مقدار بیت sign برابر یک بود که یعنی rs<rt پس خروجی برابر یک میشود و اگر هم بیت sign برابر صفر بود یعنی rs>rt پس خروجی در این حالت برابر صفر میشود . مدار این بخش از مدار را میتوانید در قسمت پایین ببینید .



شکل ۲-۳: قسمت fslt مدار

fmult ۴-۲

طراحی مدار ضرب کننده در استاندارد IEEE ۷۵۴ در نگاه اول ساده به نظر میرسد ولی چالش های اصلی آن در زمان بررسی حالات subnormal پیدا شدند. این مدار از دو بخش اصلی، یکی برای دست آوردن exponent و دیگری برای به دست آوردن mantissa تشکیل شده است، ارتباط این دو مدار در جاهایی است که توان به دست آمده از ۱۲۶- کمتر می شود و حاصل باید به طور subnormal گزارش داده شود.

۴. برای یافتن exponent، ابتدا توان های دو عدد را با کم کردن ۱۲۷ از exponent آنها به دست آورده (توان اعداد subnormal ۱۲۶- است)، پس از جمع کردن توانها با هم حاصل را با ۱۲۷ جمع می‌زنیم. ممکن است نیاز به کم و یا زیاد کردن عدد به دست آمده داشته باشیم که در توضیحات ادامه این موضوع شفاف تر می شود.

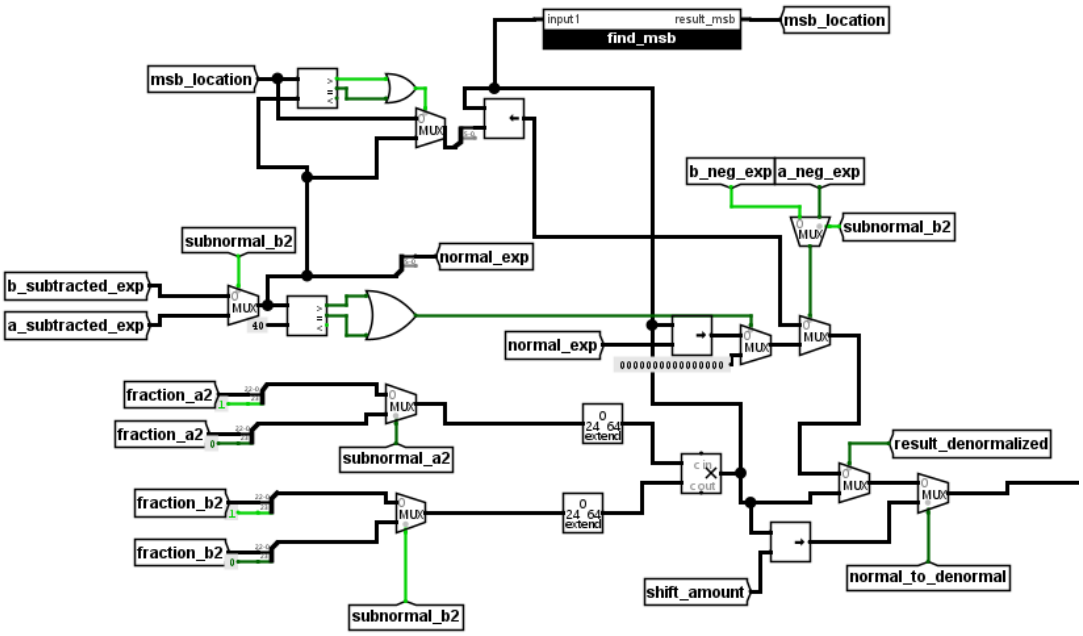
۵. برای بخش به‌دست آوردن ،mantissa در مرحلهٔ اول بدون توجه به ممیز شناور، و با توجه به اینکه عملوند نرمال یا subnormal است، به ترتیب ۱ یا ۰ در سمت چپ mantissa قرار می‌دهیم و این دو عدد ۲۴ بیتی را به کمک ضرب‌کنندهٔ باینری Logisim در هم ضرب می‌کنیم.

در صورتی که یکی از عملوندها subnormal باشد، بسته به اینکه توان عملوند دیگر مثبت یا منفی باشد، عدد به‌دست‌آمده باید به اندازهٔ مناسب به چپ یا راست شیفت داده شود.

در صورت منفی بودن توان عدد دیگر، کافی است عدد را به اندازهٔ قدر مطلق توان آن عدد به راست شیفت دهیم.

در صورت مثبت بودن توان عدد دیگر، اندازهٔ لازم برای شیفت را توسط مدار find_msb پیدا می‌کنیم. در این مدار، با کمک مالتی‌پلکسرهای ۲ به ۱ پی‌درپی، کمترین اندازهٔ شیفت برای اینکه بیت [47] یک شود، به‌دست می‌آید.

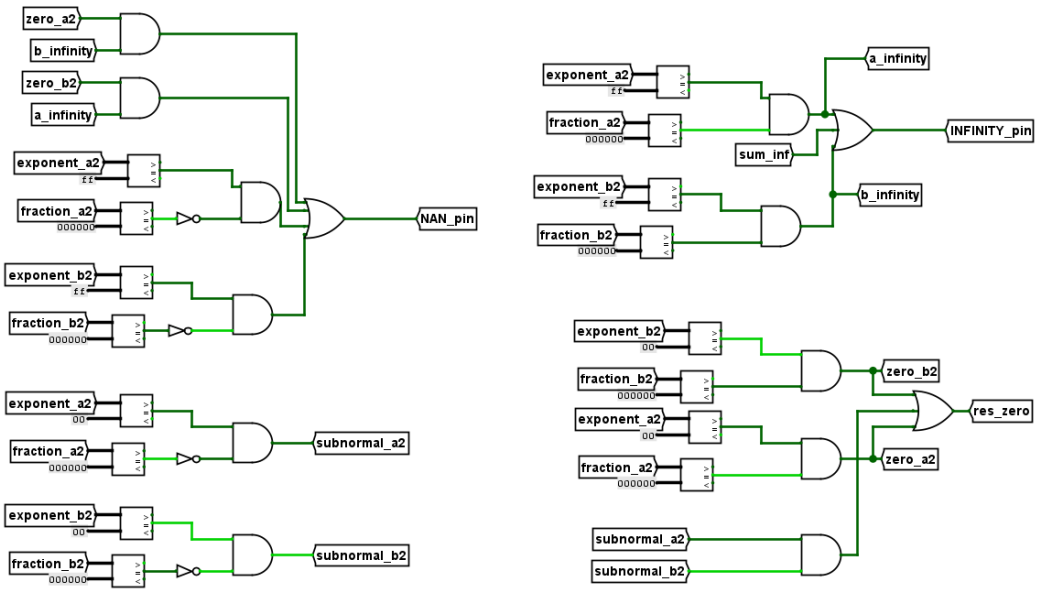
علاوه بر دو مورد ذکرشده، در یک حالت دیگر نیز به شیفت نیاز پیدا می‌کنیم، و آن حالتی است که حاصل‌ضرب دو عدد نرمال با توان‌های منفی کوچک، باید subnormal گزارش شود. در این حالت نیز باید عدد را به اندازهٔ مابه‌التفاوت جمع توان‌های دو عدد با ۱۲۶- به راست شیفت دهیم. تصویر بخش مربوطه از مدار را در زیر مشاهده می کنید.



شکل ۲-۴: عملیات لازم پس از ضرب دودویی برای اعداد subnormal

۶. در ادامه، در صورتی که بیت ۴۷ یک باشد، mantissa از بازهٔ [46:24] و در غیر این‌صورت از [45:23] عدد حاصل‌ضرب انتخاب می‌شود. همچنین در صورت یک بودن بیت ۴۷، یکی باید بر مقدار exponent افزوده شود. اگر هر دو بیت ۴۷ و ۴۶ صفر باشند، یعنی حاصل ما subnormal خواهد بود و به همین دلیل exponent صفر مقداردهی می‌شود. علاوه بر این‌ها، حالت‌هایی که حاصل Inf zero، و NaN می‌شوند به‌طور مجزا بررسی می‌شوند(تصویر۲-۵):

- در صورتی که مقدار یکی از عملوندها صفر و عملوند دیگر چیزی به‌جز Inf باشد، و همچنین در حالتی که هر دو عملوند subnormal باشند (به دلیل رخ‌دادن underflow)، حاصل صفر شده و exponent و mantissa مقداردهی صفر می‌شوند.
- در صورتی که یکی از عملوندها Inf باشد و یا اینکه توان بیشتر از ۱۲۸ شود، حاصل Inf شده و exponent برابر با 0xff و mantissa صفر مقداردهی می‌شوند.
- در صورتی که یکی از عملوندها NaN باشد و یا اینکه یکی از Inf و دیگری صفر باشد، حاصل NaN گزارش می‌شود و exponent برابر با 0xff و mantissa برابر با 0x7fffff مقداردهی می‌شوند.



شکل ۲-۵: بررسی حالات خاص به طور مجزا

در نهایت، لازم به ذکر است که بیت ۳۱ حاصل یا همان بیت علامت، از xor کردن بیت‌های ۳۱ دو عملوند به‌دست می‌آید.

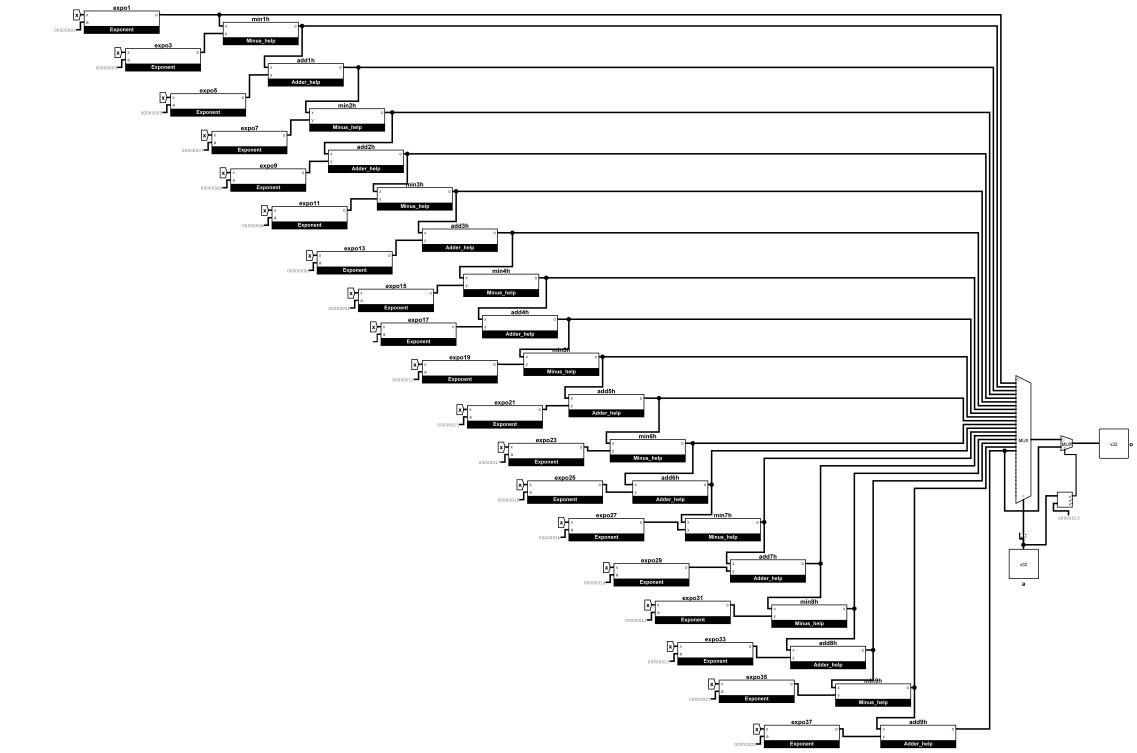
۷. ۲-۵ fabs

برای این بخش تنها کافی بود که بیت ۳۱ حاصل را به ground وصل کنیم و ۳۱ بیت کم ارزش تر دیگر را مستقیما از ورودی گرفته و به خروجی وصل کنیم.

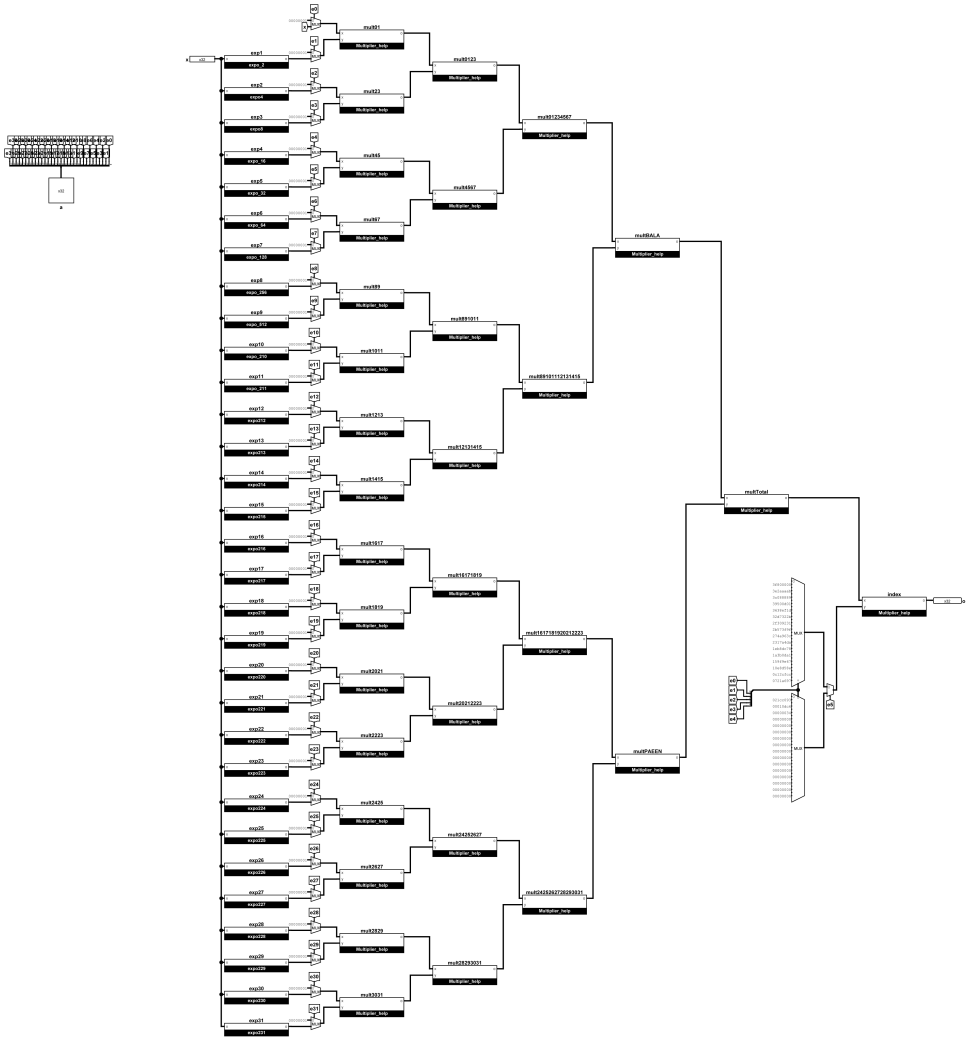
۸. ۲-۶ fsin

در این بخش نیاز داشتیم تا یک مدار برای سینوس به مدار قبلی اضافه کنیم؛ برای این‌کار در ابتدا خواستیم تا این قسمت را مانند بخش‌های دیگر مستقل از clock بزنیم که برای این‌کار عملا در یک کلاک تمامی ضرایب و در واقع Term های سری تیلور را محاسبه کرده بودیم و با توجه به ورودی rt یک مالتی پلکسر داشتیم که خروجی درست را میداد ولی این پیاده‌سازی یک مشکل داشت و آن هم این بود که به دلیل اینکه مدار زیادی باید به صورت درختی call میشدند (به طوری که محاسبه کردیم فقط در مورد ضرب‌کننده‌ها نیاز به ۱۹۴۵۶ = ۱۹ * ۳۲ * ۳۲ تا مدار ضرب‌کننده بود که این

مقدار بسیار بزرگ بود به طوری که زمانی که میخواستیم فایل را لود کنیم زمان زیادی میگرفت و بسیار فایل را سنگین کرده بود که عکس قسمت main و exponent آن را در پایین میتوانید ببینید

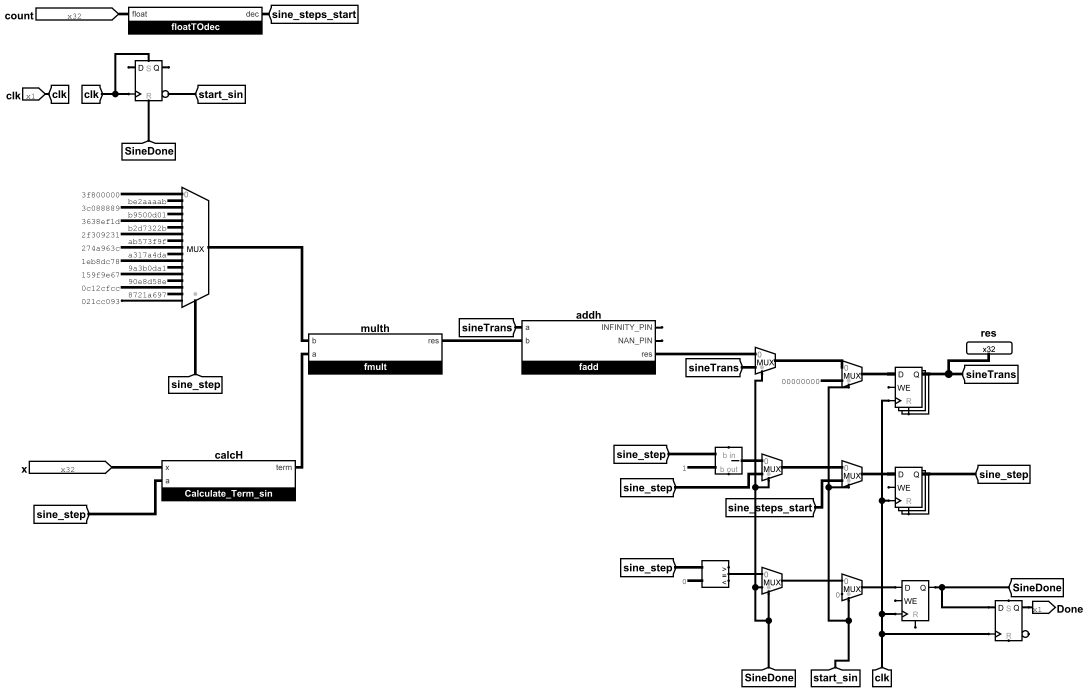


شکل ۲-۶: قسمت main مدار



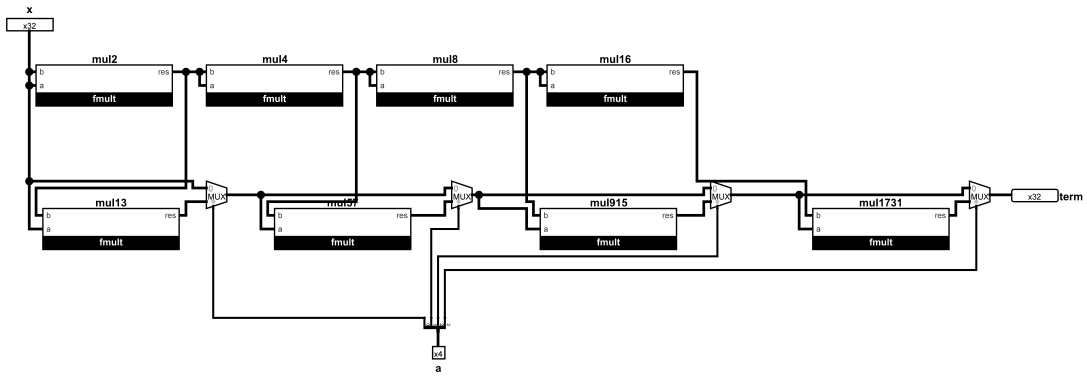
شکل ۲-۷: قسمت exponent مدار

در ادامه این بخش را به مداری با کلاک تغییر دادیم به طوری که در آن هر جمله (Term) در یک کلاک ساخته میشود و با ضرایب مناسب ضرب شده و حاصل ساخته میشود که در زیر عکس بخش سینوس را میتوانید مشاهده کنید:



شکل ۲-۸: قسمت سینوس مدار

در اینجا ابتدا توان مناسب با توجه به کلاکی که در آن هستیم آماده شده(عکس ۲-۹) و در ضریب مناسب ضرب شده و در نهایت تمامی جملات با توجه به کلاک جمع شده و خروجی حاصل میشود.



شکل ۲-۹: ساخت توان‌های مورد نیاز

برای توان نیز همانند عکس بالا توان‌های دو را ابتدا ساختیم و با وصل کردن آنها به ترتیب مناسب توان‌های فرد را نیز ساختیم و این را با استفاده از مالتی‌پلکسرهایی که اضافه کردیم انجام میدهیم. در ادامه باید بگوییم که تنها تا ضریب ۳۱ ام ساخته میشود و اعداد بالاتر از این نیز با عدد ۳۱ جایگزین شده و فقط ۱۶ جمله عملا ساخته میشود و این نیز به دلیل این است که در استاندارد IEEE 754 توانایی نشان دادن اعداد کوچک‌تر از $1/37!$ وجود ندارد و بنابراین بیشتر از این جمله را نمیتوان حساب کرد.

فصل ۳

تست بنچ مدار

برای اطمینان یافتن از صحت مدار در انتها نیاز بود که تست بنچی برای آن نوشته و آن را با استفاده از مدلی که سنتز شده تست کنیم؛ برای این‌کار از همان تست بنچی که برای تمرین پنجم بود استفاده کردیم و آن را با استفاده از مقادیر درست برای $\text{instructionsData}_m em$ آن را تغییر دادیم و همچنین مشابه تست بنچ اصلی تعدادی task و function تعریف کردیم که از آنها برای محاسبه‌ی جواب استفاده کنیم برای مثال کدهای زیر:

```
function [31:0] ieee754_add;
    input [31:0] a, b;

    reg sign_a, sign_b, sign_res;
    reg [7:0] exp_a, exp_b, exp_diff;
    reg [23:0] mant_a, mant_b;  // 1 + 23 bits
    reg [24:0] mant_addsub;
    reg [7:0] exp_res;
    reg [4:0] shift;

    begin
        if ((a[30:23] == 8'b11111111 && a[22:0] != 0) ||
            (b[30:23] == 8'b11111111 && b[22:0] != 0)) begin

            ieee754_add = 32'h7FC00000;  // Quiet NaN
        end else begin

```

```
18              sign_a = a[31];
19              sign_b = b[31];
20              exp_a = a[30:23];
21              exp_b = b[30:23];
22              mant_a = (exp_a == 0) ? {1'b0, a[22:0]} : {1'b1, a[22:0]};
                     // denormals
23              mant_b = (exp_b == 0) ? {1'b0, b[22:0]} : {1'b1, b[22:0]};
24
25              // equal exponents
26              if (exp_a > exp_b) begin
27                  exp_diff = exp_a - exp_b;
28                  mant_b = mant_b >> exp_diff;
29                  exp_res = exp_a;
30              end else begin
31                  exp_diff = exp_b - exp_a;
32                  mant_a = mant_a >> exp_diff;
33                  exp_res = exp_b;
34              end
35
36              if (sign_a == sign_b) begin
37                  mant_addsub = mant_a + mant_b;
38                  sign_res = sign_a;
39              end else begin
40                  if (mant_a > mant_b) begin
41                      mant_addsub = mant_a - mant_b;
42                      sign_res = sign_a;
43                  end else begin
44                      mant_addsub = mant_b - mant_a;
45                      sign_res = sign_b;
46                  end
47              end
48
49              // normalize
50              if (mant_addsub[24]) begin
51                  mant_addsub = mant_addsub >> 1;
52                  exp_res = exp_res + 1;
```

```
            end else begin
                while (mant_addsub[23] == 0 && exp_res > 0 &&
                    mant_addsub != 0) begin
                    mant_addsub = mant_addsub << 1;
                    exp_res = exp_res - 1;
                end
            end

            ieee754_add = {sign_res, exp_res, mant_addsub[22:0]};
        end
    end
endfunction
```

```
function [31:0] ieee754_mul;
    input [31:0] a, b;

    reg        sign_a, sign_b, sign_r;
    reg [7:0]  exp_a, exp_b;
    reg [23:0] man_a, man_b;
    reg [47:0] man_prod;
    integer    exp_a_int, exp_b_int, exp_r_int;
    integer    shift;
    reg [22:0] frac_r;
    reg [7:0]  exp_r;

begin
    //------------------------------- NaN tests
        -------------------------------
    if ((a[30:23]==8'hFF && a[22:0]!=0) ||
        (b[30:23]==8'hFF && b[22:0]!=0) || (a[30:23]==8'hFF && a
            [22:0]==0 && b == 0) || (b[30:23]==8'hFF && b[22:0]==0 && a
             == 0) ) begin
        ieee754_mul = 32'h7FFF_FFFF;        // quiet NaN
        $display("***");
        end
    //------------------------------- 0 × ∞
```

```
    ---------------------------------
else if (((a[30:23]==8'hFF) && (b[30:23]==0) && (b[22:0]==0)) ||
         ((b[30:23]==8'hFF) && (a[30:23]==0) && (a[22:0]==0)))
    ieee754_mul = 32'h7FC0_0000;        // NaN

//--------------------------------- any ∞
    ---------------------------------
else if ((a[30:23]==8'hFF) || (b[30:23]==8'hFF)) begin
    sign_r = a[31] ^ b[31];
    ieee754_mul = {sign_r, 8'hFF, 23'b0};
end

//--------------------------------- any ±0
    ---------------------------------
else if (((a[30:23]==0)&&(a[22:0]==0)) ||
         ((b[30:23]==0)&&(b[22:0]==0))) begin
    sign_r = a[31] ^ b[31];
    ieee754_mul = {sign_r, 31'b0};
end

//--------------------------------- main path
    ----------------------------
else begin
    // sign
    sign_r = a[31] ^ b[31];

    // unpack exponent & mantissa
    exp_a = a[30:23];  exp_b = b[30:23];
    man_a = (exp_a==0) ? {1'b0,a[22:0]} : {1'b1,a[22:0]};
    man_b = (exp_b==0) ? {1'b0,b[22:0]} : {1'b1,b[22:0]};

    // unbiased exponents
    exp_a_int = (exp_a==0) ? -126 : (exp_a - 127);
    exp_b_int = (exp_b==0) ? -126 : (exp_b - 127);

    // multiply mantissas
```

```
53          man_prod  = man_a * man_b;          // 24×24 → 48
54
55          // add exponents
56          exp_r_int = exp_a_int + exp_b_int;
57
58          // first (right) normalisation if MSB in bit47
59          if (man_prod[47]) begin
60              man_prod  = man_prod >> 1;
61              exp_r_int = exp_r_int + 1;
62          end
63
64          // -------->>>  NEW -leftnormalisation <<<--------
65          while (man_prod[46]==0 && exp_r_int>-149) begin
66              man_prod  = man_prod << 1;
67              exp_r_int = exp_r_int - 1;
68          end
69
70          // overflow →∞ ±
71          if (exp_r_int + 127 >= 255)
72              ieee754_mul = {sign_r, 8'hFF, 23'b0};
73
74          // normal range
75          else if (exp_r_int >= -126) begin
76              exp_r  = exp_r_int + 127;
77              frac_r = man_prod[45:23];       // truncate
78              ieee754_mul = {sign_r, exp_r, frac_r};
79          end
80
81          // -subnormal range
82          else if (exp_r_int >= -149) begin
83              shift     = -126 - exp_r_int;   // …123
84              man_prod  = man_prod >> shift;
85              frac_r    = man_prod[45:23];
86              ieee754_mul = {sign_r, 8'h00, frac_r};
87          end
88
```

```
        // underflow to zero
        else
            ieee754_mul = {sign_r, 31'b0};
    end
end
endfunction
```

در ادامه در تست بنچ اصلی یک task برای انجام عملیات داشتیم که آنها را با استفاده از دستورات اضافه شده تکمیل کردیم که آن را میتوانید در زیر مشاهده کنید:

```
task exec_internal;
    begin
        inst_reg = instructions[ipc];
        ipc += 1;

        inst_rs = inst_reg[25:21];
        inst_rt = inst_reg[20:16];
        inst_rd = inst_reg[15:11];
        inst_imm = inst_reg[15:0];
        inst_imm_sext = {{16{inst_imm[15]}}, inst_imm};
        val_rs = ireg[inst_rs];
        val_rt = ireg[inst_rt];
        val_frs = ifreg[inst_rs]; /*FP register value*/
        val_frt = ifreg[inst_rt]; /*FP register value*/
        case (inst_reg[31:26])
            6'b000000: begin  // RType
                case (inst_reg[5:0])
                    6'b100000: write2reg(inst_rd, val_rs + val_rt);
                                // add
                    6'b100010: write2reg(inst_rd, val_rs - val_rt);
                                // sub
                    6'b100100: write2reg(inst_rd, val_rs & val_rt);
                                // and
                    6'b100101: write2reg(inst_rd, val_rs | val_rt);
                                // or
                    6'b100110: write2reg(inst_rd, val_rs ^ val_rt);
                                // xor
```

```
23                  6'b000100: write2reg(inst_rd, val_rs << val_rt
                        [4:0]);       // sll
24                  6'b000110: write2reg(inst_rd, val_rs >> val_rt
                        [4:0]);       // srl
25                  6'b000111: write2reg(inst_rd, sra(val_rs, val_rt
                        [4:0]));    // sra
26                  6'b000000:
27                  write2reg(inst_rd, val_rt << inst_reg[10:6]);  //
                        sll (imm) rd=rt<<shamt
28                  6'b011010: begin  // div HI=rs%rt; LO=rs/rt
29                      ireghi = val_rs % val_rt;
30                      ireglo = val_rs / val_rt;
31                  end
32                  6'b010000: write2reg(inst_rd, ireghi);  // mfhi rd
                        =HI
33                  6'b010010: write2reg(inst_rd, ireglo);  // mflo rd
                        =LO
34
35
36
37                  //_____________________________________________FP
                        instructions:
38                  /*displayed values are hex, register numbers are
                        dec*/
39                  /*                                              */
40                  6'b000001: begin    // Fsin
41                      case (inst_rt)
42                          5'b00110: begin
43                              // if (inst_rs == 5'b00001) w2fr(
                                    inst_rd, 32'
                                    b00111111100110011001100110011010);
                                     // 3F99999A
44                              if (inst_rs == 5'b00001) w2fr(inst_rd,
                                     32'
                                    b00111111011011101001101000011110);
                                     // 3F6E9A1E
```

```
45                  if (inst_rs == 5'b00011) w2fr(inst_rd,
                        32'
                        b00111111100110011001100110011010);
46              end
47              5'b00111: begin
48                  if (inst_rs == 5'b00010) w2fr(inst_rd,
                        32'
                        b00111111011111111111010110111111);
49                  if (inst_rs == 5'b00101) w2fr(inst_rd,
                        32'
                        b00111111011011001100110111111001);
50              end
51              5'b01000: begin
52                  if (inst_rs == 5'b00100) w2fr(inst_rd,
                        32'
                        b00000000000000000000000000000000);
53                  if (inst_rs == 5'b00011) w2fr(inst_rd,
                        32'
                        b00000000000000000000000000000000);
54              end
55              5'b01001: begin
56                  if (inst_rs == 5'b00001) w2fr(inst_rd,
                        32'
                        b10111111010101110110101001110110);
57                  if (inst_rs == 5'b00011) w2fr(inst_rd,
                        32'
                        b10111111100000000000000000000000);
58              end
59              5'b01010: begin
60                  if (inst_rs == 5'b00010) w2fr(inst_rd,
                        32'
                        b10111111011111111111010110111111);
61                  if (inst_rs == 5'b00101) w2fr(inst_rd,
                        32'
                        b10111111011011001100110111111001);
62              end
```

```
63                          5'b01011: begin
64                              if (inst_rs == 5'b00100) w2fr(inst_rd,
                                    32'
                                    b10111111001101111010010010100110);
65                              if (inst_rs == 5'b00011) w2fr(inst_rd,
                                    32'
                                    b10111111001101111010010010100110);
66                          end
67                          5'b01100: begin
68                              if (inst_rs == 5'b00001) w2fr(inst_rd,
                                    32'
                                    b00111111011000111011000011011000);
69                              if (inst_rs == 5'b00011) w2fr(inst_rd,
                                    32'
                                    b00111111011001100110011001001101);
70                          end
71                          5'b01101: begin
72                              if (inst_rs == 5'b00010) w2fr(inst_rd,
                                    32'
                                    b10111110011111010101011101110111);
73                              if (inst_rs == 5'b00101) w2fr(inst_rd,
                                    32'
                                    b10111110011111010101010101010101);
74                          end
75                          5'b01110: begin
76                              if (inst_rs == 5'b00100) w2fr(inst_rd,
                                    32'
                                    b00111111010010001000100000011101);
77                              if (inst_rs == 5'b00011) w2fr(inst_rd,
                                    32'
                                    b00111111010010001000100000011101);
78                          end
79                          5'b01111: begin
80                              if (inst_rs == 5'b00001) w2fr(inst_rd,
                                    32'
                                    b10111111011101101010100111011110);
```

```
81                          if (inst_rs == 5'b00011) w2fr(inst_rd,
                                32'
                               b10111111101001100110011001100110);
82                      end
83                  endcase
84                  $display("\n######Sin(FR[%0d] = %h) in FR[%d]
                        step by taylor series go to FR[%d] = %h
                        ######\n", inst_rt, val_frt, inst_rs,
                        inst_rd, ifreg[inst_rd]);
85              end
86              6'b010011: begin    // Fadd
87              $display("Fadd: FR[%0d] + FR[%0d] --> FR[%0d]",
                    inst_rs, inst_rt, inst_rd);
88              w2fr(inst_rd, ieee754_add(val_frs, val_frt));
89              end
90              6'b010100: begin    // Fsub
91                  $display("Fsub: FR[%0d] - FR[%0d] --> FR[%0d]"
                        , inst_rs, inst_rt, inst_rd);
92                  w2fr(inst_rd, ieee754_add(val_frs, {~val_frt
                        [31], val_frt[30:0]}));
93              end
94              6'b010101: begin    // Fmult
95              $display("FR[%0d] * FR[%0d] --> FR[%0d]", inst_rs,
                     inst_rt, inst_rd);
96
97          w2fr(inst_rd, ieee754_mul(val_frs, val_frt));
98
99              end
100             6'b010110: begin    // Fabs
101             $display("|FR[%0d]| --> FR[%0d]", inst_rs, inst_rd
                    );
102
103         w2fr(inst_rd, ieee754_abs(val_frs));
104
105             end
106             6'b010111: begin    // Fslt (FR[rs] < FR[rt] ? 1 :
```

```
                    0)
107                 val_2 = ieee754_add(val_frs, {~val_frt[31],
                        val_frt[30:0]});  // FR[rs] - FR[rt];
108                 if (val_2[31] == 1'b1) begin
109                 w2fr(inst_rd, 32'd1);  // FR[rs] < FR[rt]
110                 end else begin
111                 w2fr(inst_rd, 32'd0);  // FR[rs] >= FR[rt]
112                 end
113                 $display("Fslt: FR[%0d] < FR[%0d] --> R[%0d] =
                        %0d", inst_rs, inst_rt, inst_rd, (val_2
                        [31] ? 1 : 0));
114             end
115
116             6'b011000: begin    // TOcop
117                 w2fr(inst_rd, val_rs);
118                 $display("TOcop: R[%0d] = %h --> FR[%0d]",
                        inst_rs, val_rs, inst_rd);
119             end
120             6'b011001: begin    // FROMcop
121                 write2reg(inst_rd, val_frs);
122                 $display("FROMcop: FR[%0d] = %h --> R[%0d]",
                        inst_rs, val_frs, inst_rd);
123             end
124             //_____________________________________end of
                    FP instructions/
125
126             default $display("NOT IMPLEMENTED : rtype[func: %b
                    ]", inst_reg[5:0]);
127         endcase
128     end
129     6'b001000: write2reg(inst_rt, val_rs + inst_imm_sext);  //
            addi
130     6'b101011: begin  // sw *(int*)(offset+rs)=rt
131         data_addr = val_rs + inst_imm_sext;
132         if (data_addr & 3 !== 0)
133             $display(
```

```
134                     "WARNING␣:␣Unaligned␣data␣address␣(%x)",
135                     data_addr,
136                     "␣␣%x␣=>␣%x␣%x",
137                     inst_rs,
138                     val_rs,
139                     inst_imm_sext
140                 );
141             $display("sw:␣R[%0d]␣=␣%h␣-->␣mem[%0d]", inst_rt,
                    val_rt, (data_addr>>2)&511);
142             data_mem[(data_addr>>2)&511] = val_rt;
143         end
144         6'b100011: begin                                        //
                lw rt=*(int*)(offset+rs)
145             data_addr = val_rs + inst_imm_sext;
146             if (data_addr & 3 !== 0)
147                 $display(
148                     "WARNING␣:␣Unaligned␣data␣address␣(%x)",
149                     data_addr,
150                     "␣␣%x␣=>␣%x␣%x",
151                     inst_rs,
152                     val_rs,
153                     inst_imm_sext
154                 );
155
156             $display("lw:␣mem[%0d]␣=␣%h␣-->␣R[%0d]", (data_addr
                    >>2)&511, data_mem[(data_addr>>2)&511], inst_rt);
157             write2reg(inst_rt, data_mem[(data_addr>>2)&511]);
158         end
159         6'b000101: begin                                        //
                bne if(rs!=rt) pc+=offset
160             ipc += val_rs != val_rt ? inst_imm_sext : 0;
161         end
162         6'b001010: write2reg(inst_rt, val_rs < inst_imm_sext ? 1 :
                0);  // slti rt=rs<imm
163         6'b000010: ipc = inst_imm;                              //
                j pc=target
```

```
164             default $display("NOT␣IMPLEMENTED␣:␣[opcode:␣%b]",
                    inst_reg[31:26]);
165         endcase
166         // c inst_reg[31:26]
167     end
168 endtask
```

در انتها هم تعدادی دستور نوشتیم و برنامه را مشابه قبلا تست کردیم؛ همچنین برای زیبایی مدار آن را با استفاده از رنگ‌هایی که در ترمینال داشتیم سعی کردیم که زیباتر نمایش دهیم.