دانشگاه صنعتی شریف
دانشکده مهندسی کامپیوتر
گزارش کار ششم درس آزمایشگاه معماری

عنوان:

واحد محاسبه با امکان انتخاب ثبات مبدا و مقصد

ALU with source and destination registers

نگارش

کیان قاسمی ۴۰۱۱۰۲۲۶۴
آیین پوست فروشان ۴۰۱۱۰۵۷۴۲
دیبا هادی اسفنگره ۴۰۱۱۱۰۲۴۵

استاد

دکتر حمید سربازی آزاد

دستیار آموزشی

مهندس عطیه غیبی فطرت

مرداد ۱۴۰۳

فهرست مطالب

فهرست تصاویر

۱ مقدمه

در این آزمایش می‌خواهیم با استفاده از نرم‌افزار *Proteus* یک واحد محاسبات و مجموعه ثبات‌های عمومی ماشین را طراحی و پیاده‌سازی کنیم. فرمت‌سازی دستور داده شده به ماشین یک دستور ۶ بیتی است که بیت اول آن برای مشخص شدن تابع موردنظر($add : 0$، $sub : 1$)، دو بیت بعدی برای مشخص کردن ثبات مقصد (باتوجه به اینکه ۴ ثبات برای آدرس‌دهی به آنها دو بیت کافی است) و ۳ بیت آخر برای مشخص کردن مقدار $SourceRegister$ است که با یا مقدار 000، 001 و یا با محتوای ۴ بیتی داخل ماشین پر می‌شود. در این معماری یک عملوند برای واحد محاسبات ثبات است.

۱.۱ قطعات لازم

۱. ۴ رجیستر ۸ بیتی ۷۴۱۹۸

۲. یک دیکودر ۷۴۱۳۹

۳. چند گیت نات

۴. ۸ مالتیپلکسر ۸ بیتی ۷۴۱۵۱

۵. دو جمع‌کننده ۴ بیتی ۷۴۸۳ برای پیاده‌سازی جمع‌کننده ۸ بیتی

۶. ۸ گیت ایکس اور

۲ شبیه سازی مدار

ابتدا با استفاده از نرم‌افزار *proteus* مدار مورد نظر را طراحی و آزمایش کردیم. در بخش‌های قبلی با الگوریتم‌ها برای پیاده‌سازی مدار آشنا شدیم. حال طراحی مدارهای این ضرب‌کننده را نشان می‌دهیم.

۱.۲ مشخص کردن رجیستر مقصد در رجیستر فایل

در این مرحله، دو بیت ایندکس به یک دیکودر داده می‌شوند که خروجی‌های آن به‌گونه‌ای تنظیم می‌شود که تنها یک خروجی صفر و بقیه یک باشند. برای این کار، خروجی‌های دیکودر معکوس شده و به ورودی لود رجیسترها متصل می‌شوند. این معکوس‌سازی باعث می‌شود که هنگام فعال بودن سیگنال‌های s_0, s_1، تنها یک رجیستر بتواند داده جدیدی را لود کند. برای اطمینان از عدم پاک شدن داده‌های موجود در رجیسترها، سیگنال MR در حالت فعال (یک) نگه داشته می‌شود. سیگنال کلک نیز به ورودی کلک رجیسترها وصل می‌شود. ۸ بیت داده ورودی به تمام رجیسترها متصل شده‌اند و با فعال شدن سیگنال کلک، داده بر اساس ایندکس

مشخص در رجیستر مربوطه لود می‌شود. سپس، خروجی هر رجیستر به یک مولتی‌پلکسر ۸ به ۱ متصل می‌شود تا داده‌های موردنظر انتخاب و به مرحله بعد ارسال شوند.

۲.۲ مالتی پلکسر ۸ به ۱

در این بخش، یک مالتی‌پلکسر ۸ به ۱ با ورودی‌ها و خروجی‌های ۸ بیتی طراحی می‌کنیم. با توجه به اینکه هر رجیستر شامل ۸ بیت است، نیاز داریم تا از ۸ مالتی‌پلکسر ۸ به ۱ تک‌بیتی استفاده کنیم که هر کدام دارای ۳ خط آدرس هستند. چهار ورودی اول هر مالتی‌پلکسر به ترتیب به بیت‌های معادل در چهار رجیستر متصل می‌شوند. در سه ورودی بعدی باید مقادیر '00000000' (صفر)، '00000001' (یک) و '11111111' (منفی یک) قرار گیرند. برای این منظور، ورودی‌های ۴ و ۶ را به ترتیب به صفر و یک متصل می‌کنیم. در مورد ورودی ۵ که مقدار یک را دریافت می‌کند، ورودی ۵ اولین مالتی‌پلکسر را به یک و سایر ورودی‌ها را به صفر متصل می‌کنیم (توجه کنید که شماره‌گذاری ورودی‌های مالتی‌پلکسر از ۰ شروع می‌شود). سپس، خطوط آدرس ('*source*') به ترتیب به پایه‌های انتخاب ('*A*'، '*B*'، '*C*') در تمام مالتی‌پلکسرها متصل می‌شوند. با توجه به اینکه سیگنال '*enable*' در مالتی‌پلکسرها به صورت فعال پایین (*lowactive*) است، آن را به صفر متصل می‌کنیم. در نهایت، خروجی مالتی‌پلکسرها به مرحله بعدی، یعنی واحد محاسبات، متصل می‌شوند.

۳.۲ واحد محاسبات *ALU*

در این قسمت یک واحد محاسبات با دو قابلیت جمع و تفریق دو عدد ۸ بیتی طراحی می‌کنیم: سیگنال ورودی Add/Sub مشخص می‌کند که جمع باید انجام شود یا تفریق. در این بخش ورودی دوم B را با $Add/Sub\ XOR$ می‌کنیم ($A - bit from MUX$). سپس حاصل جمع یا *xor* را با مقدار صفر که ورودی ثابت این واحد است جمع می‌کنیم. برای جمع و تفریق، A و B به ترتیب به عنوان عدد اول و دوم وارد واحد جمع کننده ۴ بیتی استفاده می‌شوند. ما باید تعیین کنیم که آیا این دو عدد جمع یا تفریق شده‌اند و خروجی آن را محاسبه کنیم. سپس خروجی واحد محاسبات را به عنوان *carry* برای عملیات جمع دوم استفاده می‌کنیم و فرمول واحد محاسبات به صورت زیر است: (در اینجا C_{in} جایگزین *carry* ورودی جمع کننده‌ی اول است)

$$Out = A + (B \oplus Add/Sub) + C_{in}, \quad C_{in} = Add/Sub$$

در اینجا دو حالت وجود دارد:

۱. اگر Add/Sub صفر باشد، باید جمع دو ورودی را با یکدیگر جمع فرض کنیم که A را داخل رجیستر و B را مقدار حاصل از $MUX8 * 1$ باشد. بنابراین داریم:

$$Add/Sub = 0, \quad B \oplus - = B \quad \Rightarrow \quad Out = A + B + 0 = A + B$$

۲. اگر Add/Sub یک باشد، مقدار حاصل از باید عملیات تفریق را انجام فرض کنیم که A را داخل رجیستر و B را مقدار حاصل از $MUX8*1$ باشد. بنابراین داریم:

$$A + \overline{B} = A + B + not(B) + 1 = A - B + 1$$

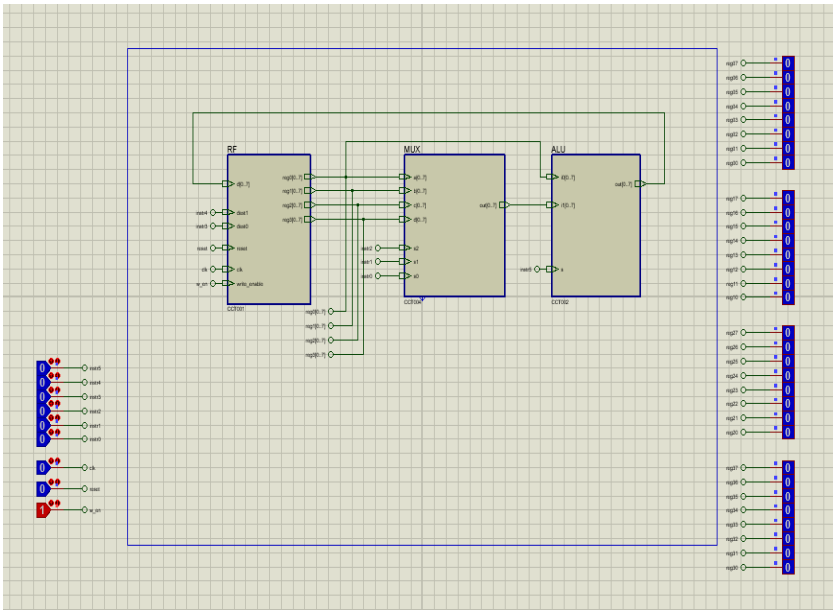
سپس عدد A و حاصل جمع آن با $not(B)$ و ۱، حاصل $A - B$ است:

$$Add/Sub = 1, \quad B \oplus 1 = not(B) \quad \Rightarrow \quad Out = A + not(B) + 1 = A - B$$

۴.۲ مرحله چهارم: آزمایش مدار و بررسی درستی عملکرد آن

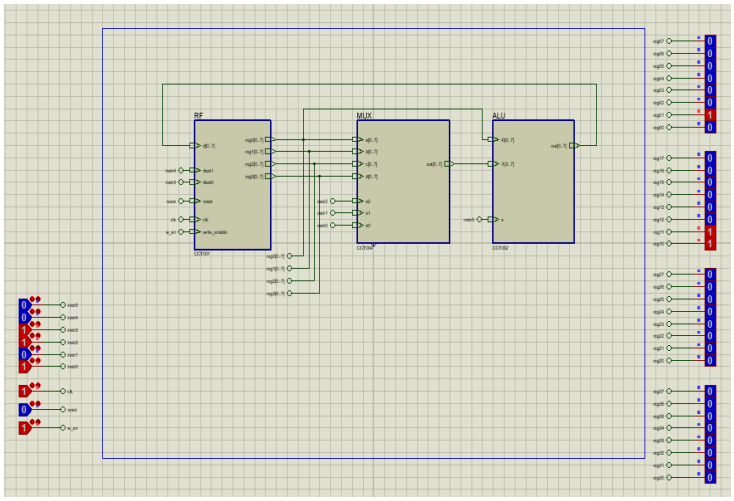
در این قسمت حالت های مختلف ورودی را به مدار دادیم و خروجی آن را بررسی کردیم، نتیجه سه تست را نیز در این قسمت آورده ایم:

مرحله اول) درابتدا با ریست کردن مقدار تمام رجیسترها را برابر ۰ می کنیم.



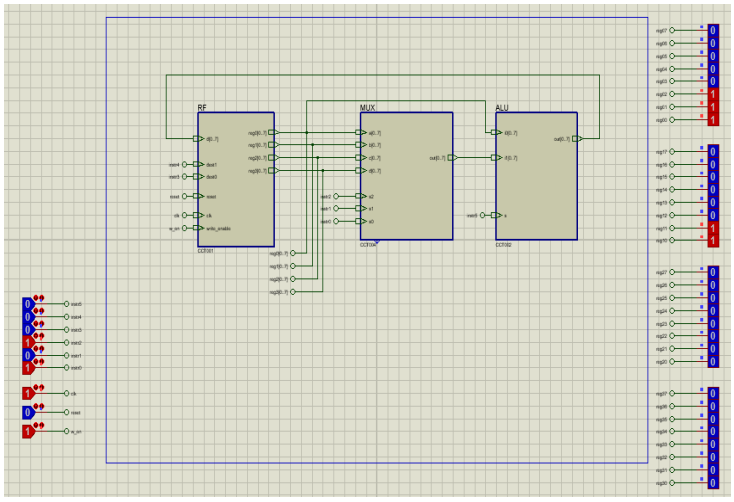
شکل ۱: رجیسترها صفر شده‌اند

مرحله دوم) سپس با دستور $R0 = R0 + 1$ مقدار ثبات ۰ را ۲ کرده و با دستور $R1 = R0 + 1$ مقدار ثبات ۱ را ۳ می‌کنیم.



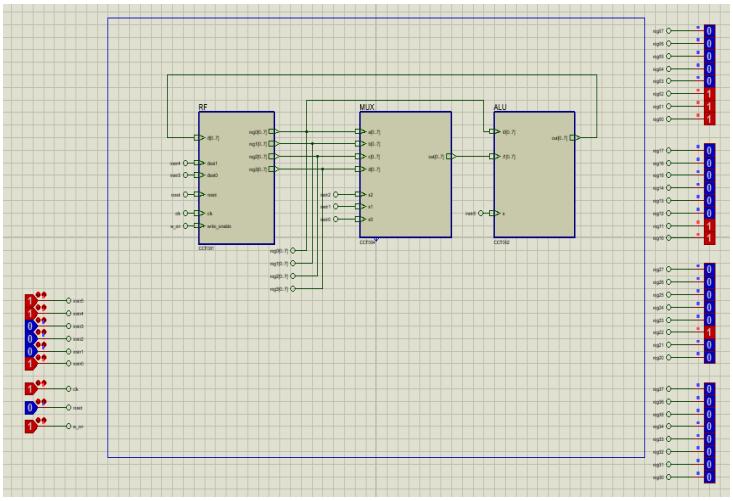
شکل ۲: وضعیت مدار

مرحله سوم) حال با اجرای همان دستور اول مقدار ثبات ۰ را ۷ می‌کنیم



شکل ۳: عدد ۷ را مشاهده می‌کنیم

مرحله چهارم) در نهایت دستور $R2 = R0 - R1$ را اجرا می‌کنیم.



شکل ۴: مقدار ۴ در رجیستر دو

۳ پیاده سازی فیزیکی مدار در آزمایشگاه

برای پیاده‌سازی روی بورد، مدار را با ۲ رجیستر ۸ بیتی ساختیم. ابتدا بخش‌های اصلی مدار، یعنی رجیستر فایل شامل ۲ رجیستر ۸ بیتی، ۸ عدد مالتی پلکسر ۸ تایی و یک جمع/تفریق کننده (شامل ۸ گیت *xor* و دو جمع کننده ۴ بیتی)، را جداگانه ساخته و تست کردیم. سپس اتصالات قسمت‌های مختلف را به ترتیب محاسبه در هر چرخه‌ی ساعت انجام دادیم. درواقع ورودی‌های مالتی‌پلکسرها که از روی خروجی‌های رجیسترها به دست می‌آیند را مشخص می‌کنیم، سپس خروجی مالتی‌پلکسرها را با ورودی جمع/تفریق *xor* می‌کنیم و سپس *OR* را با خروجی *xor* ها جمع می‌کنیم. تمام قطعات و اتصالات تا این مرحله را نیز چک کردیم. در نهایت خروجی جمع کننده را به ورودی رجیستر متصل می‌کنیم. برای رجیستر، مالتی‌پلکسر، *xor* و جمع کننده به ترتیب از، 74173، 74151، 7486 و 7483 استفاده کردیم.

۴ چالش ها

در طراحی این مدار در پروتئوس چالش‌های کوچکی در هنگام استفاده از شیفت رجیسترها داشتیم که با بازنگری به دیتاشیت مشکل حل شد و در کل چالش‌های طراحی مالتیپلکسر و رجیستر فایل رفع شدند.

۵ نتیجه و بحث

در این آزمایش، با موفقیت یک واحد محاسباتی ساده با قابلیت جمع و تفریق دو عدد ۸ بیتی طراحی و پیاده‌سازی کردیم. استفاده از رجیسترها، مالتی‌پلکسرها، و واحد محاسباتی *ALU* باعث شد که بتوانیم عملیات جمع و تفریق را به صورت همزمان و با دقت بالا انجام دهیم. بررسی عملکرد مدار با استفاده از شبیه‌سازی‌های مختلف نشان داد که این مدار توانایی انجام محاسبات صحیح را دارد و خروجی‌های آن با نتایج مورد انتظار هم‌خوانی دارند. در مرحله پیاده‌سازی فیزیکی، با وجود چالش‌های جزئی در اتصالات و مدارهای منطقی، توانستیم به نتایج مطلوبی دست یابیم. این چالش‌ها به ما نشان داد که توجه به جزئیات در طراحی و پیاده‌سازی مدارهای دیجیتال از اهمیت بسیاری برخوردار است. همچنین، استفاده از قطعات استاندارد نظیر 74173، 74151، 7486، و 7483 موجب تسهیل فرآیند پیاده‌سازی و کاهش خطاهای احتمالی شد. در نهایت، این پروژه تجربه‌ی عملی ارزشمندی در طراحی و پیاده‌سازی واحدهای محاسباتی دیجیتال فراهم کرد و توانست ما را با چالش‌ها و روش‌های حل آنها در دنیای واقعی آشنا سازد.