هو العلیم

اصول سیستم‌های کامپیوتری

نیمسال اول سال تحصیلی ۱۴۰۲ – ۱۴۰۱

تمرینات ۲

مریم رضائی

۱. این برنامه‌ای را که به زبان C نوشته شده است و میانگین اعداد در یک ماتریس را محاسبه می‌کند، به برنامه‌ای به زبان RISC-V ترجمه کنید. خودتان شماره رجیسترهایی را که در برنامه RISC-V استفاده می‌کنید طبق قراردادها تعیین کنید.

جواب:

فرض می‌کنیم ابتدای ماتریس در رجیستر x10 یعنی یک پارامتر تابع ذخیره شده است. آنگاه با استفاده از رجیسترهای موقتی محاسبات را انجام داده و در نهایت مقدار حاصل میانگین را در یک رجیستر پارامتر تابع دیگر مانند x11 ذخیره می‌کنیم که این به معنای محفوظ ماندن حاصل و بازگرداندن آن است.

برای پیاده‌سازی با RISC-V به چند نکته توجه می‌کنیم:

۱) برای از دست نرفتن مقادیر قبلی رجیسترهای مورد استفاده، آنها را در پشته ذخیره می‌کنیم. به خصوص برای رجیستر آدرس ابتدایی ماتریس که در ادامه‌ی کد به اندازه‌ی i و j به آن به طور مکرر اضافه می‌کنیم. در انتها نیز آنها را بازیابی می‌کنیم.

۲) برای تبدیل offset کلمه به بایت برای دستور lw از ماتریس، نیاز است هر بار i و j در چهار ضرب شوند (یعنی دو بار به چپ شیفت شوند). برای جلوگیری از تکرار این دستور و پیاده سازی با کمترین دستورات، از آنجایی که i و j تنها کاربرد آدرس‌دهی دارند، می‌توانیم به جای یکی یکی به آنها اضافه کردن و ضرب در چهار کردن، در ابتدا چهارتا چهارتا به آنها اضافه کنیم. بنابراین نیاز است شرط حلقه را نیز از ۱۰ به ۴۰ تغییر دهیم.

کد برنامه در زیر قابل مشاهده است.

برنامه C	برنامه RISC-V

float mat_mean (int arr[][])
{
int i, j;
float mean, sum = 0;
for (i = 0; i < 10; i++)
for (j = 0; j < 10; j++)
sum += arr[i][j];
mean = sum / 100;
return mean;
}
	addi sp, sp, -28 // make room for 7
	sw x5, 24(sp) // store previous value of x5
	sw x6, 20(sp) // store previous value of x6
	sw x7, 16(sp) // store previous value of x7
	sw x28, 12(sp) // store previous value of x28
	sw x29, 8(sp) // store previous value of x29
	sw x30, 4(sp) // store previous value of x30
	sw x10, 0(sp) // store initial value of x10
	addi x5, x0, 0 // sum = 0
	addi x6, x0, 40 // save for loop end
	// but ×4 to convert word offset to byte
	addi x7, x0, 0 // i = 0
	ILOOP: add x10, x10, x7 // move arr forward to arr[i]
	addi x28, x0, 0 // j = 0
	JLOOP: add x10, x10, x28 // move arr forward to arr[i][j]
	lw x29, 0(x18) // load arr[i][j]

```
add x5, x5, x29        // sum += arr[i][j]
addi x28, x28, 4       // j++
                       // but ×4 to convert word offset to byte
blt x28, x6, JLOOP     // j < 10 (j < 40), rerun j loop
addi x7, x7, 1         // i++
                       // but ×4 to convert word offset to byte
blt x7, x6, ILOOP      // i < 10 (i < 40), rerun i loop
addi x30, x0, 100      // save 100 for division
div x11, x5, x30       // mean = sum / 100
                       // save in function parameters to return
lw x10, 0(sp)      // restore initial value of x10
lw x30, 4(sp)      // restore previous value of x30
lw x29, 8(sp)      // restore previous value of x29
lw x28, 12(sp)     // restore previous value of x28
lw x7, 16(sp)      // restore previous value of x7
lw x6, 20(sp)      // restore previous value of x6
lw x5, 24(sp)      // restore previous value of x5
addi sp, sp, 28    // empty the stack
jalr x0, 0(x1)     // return to call address
```

مراجع:

۲. مستقیماً برنامه‌ای را به زبان RISC-V بنویسید که رشته‌ای را بررسی کند و مشخص کند که آیا آن رشته، یک رشته متقارن است یا خیر. منظور از رشته متقارن، رشته‌ای است که از دو طرف به یک شکل خوانده شود. به عنوان مثال، wow و racecar دو رشته متقارن هستند. خودتان شماره رجیسترهایی را که در برنامه RISC-V استفاده می‌کنید طبق قراردادها تعیین کنید.

جواب:

فرض می‌کنیم آدرس ابتدای رشته در رجیستر x10 و طول آن عضو اول (یعنی ۳۲ بیت اولیه بعد از آدرس شروع) می‌باشد. همچنین فرض می‌کنیم رشته با کاراکترهای Unicode ذخیره شده است که هر کدام طول ۱۶ بیت یعنی ۲ بایت را دارند. بنابراین در ابتدا طول را در رجیستری موقتی ذخیره کرده، و همزمان از ابتدا و انتهای رشته شروع به حرکت و مقایسه می‌کنیم. در صورت برابر بودن مقادیر، یعنی کلمه متقارن است و در رجیستر x11 بازگشتی مقدار ۱ را ذخیره می‌کنیم. در غیر این صورت، مقدار صفر قرار داده می‌شود.

برنامه RISC-V

```
        addi sp, sp, -16        // make room for 7
        sw x5, 12(sp)           // store previous value of x5
        sw x6, 8(sp)            // store previous value of x6
        sw x28, 4(sp)           // store previous value of x28
        sw x29, 0(sp)           // store previous value of x29
        addi x5, x10, 4         // x5 = beginning of string
        lw x6, 4(x10)           // x6 = length of string
        add x6, x5, x6          // x6 = beginning + length = end
LOOP:   lhu x28, 2(x5)          // load halfword, offset 2 bytes
                                // x28 = Unicode of beginning character
        lhu x29, 2(x6)          // load halfword, offset 2 bytes
                                // x29 = Unicode of ending character
        bne x28, x29, SKIP      // if characters are not equal, exit loop and return
        addi x5, x5, 2          // else, move the beginning index forward by 1 hw (2 bytes)
        addi x6, x6, -2         // move the ending index back by 1 hw (2 bytes)
        blt x5, x6, LOOP        // if both side haven't yet reached the middle, rerun loop
        addi x11, x0, 1         // else, if loop ends successfully, mark return parameter as 1
        jal x0, RET             // jump to return
SKIP:   addi x11, x0, 0         // mark return parameter as 0
RET:    lw x29, 0(sp)           // restore previous value of x29
        lw x28, 4(sp)           // restore previous value of x28
        lw x6, 8(sp)            // restore previous value of x6
        lw x5, 12(sp)           // restore previous value of x5
        addi sp, sp, 16         // empty the stack
        jalr x0, 0(x1)          // return to call address
```

مراجع:

کتاب منبع درس برای چگونگی کار با رشته:

Computer Organization and Design (RISC-V Edition) – Patterson & Hennessy

۳. می‌توان با استفاده از گیت‌های بیشتر و طرح‌های پیچیده‌تر، مدارهای منطقی سریع‌تری را برای محاسبه حاصل‌ضرب اعداد دودویی طراحی کرد. فرض کنید می‌خواهیم مداری منطقی بسازیم برای محاسبه حاصل‌ضرب دو عدد صحیح ۳۲ بیتی. اعداد را به صورت $A_{31}A_{30}\cdots A_1A_0$ و $B_{31}B_{30}\cdots B_1B_0$ نمایش دهید.

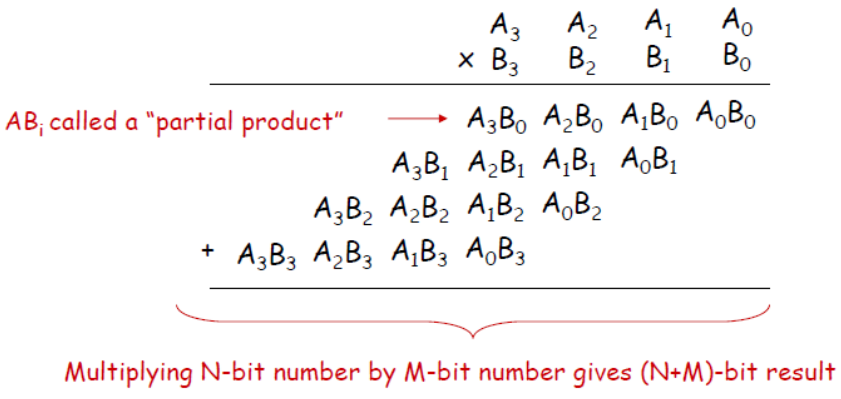
الف) توضیح دهید که چگونه با استفاده از ۳۱ مدار جمع‌کننده ۳۲ بیتی که به شکل پلکانی چیده می‌شوند و تعدادی گیت AND ، یک مدار منطقی ضرب‌کننده اعداد صحیح ۳۲ بیتی بسازیم. و به عنوان مثالی از چنین ضرب‌کننده‌ای، مدار منطقی ضرب‌کننده اعداد ۴ بیتی $A_3A_2A_1A_0$ و $B_3B_2B_1B_0$ را دقیقاً بکشید.

ب) توضیح دهید که چگونه با استفاده از ۳۱ مدار جمع‌کننده ۳۲ بیتی که به شکل درختی چیده می‌شوند، یک مدار منطقی ضرب‌کننده اعداد صحیح ۳۲ بیتی بسازیم. و به عنوان مثالی از چنین ضرب‌کننده‌ای، مدار منطقی ضرب‌کننده اعداد ۸ بیتی $A_7A_6A_5A_4A_3A_2A_1A_0$ و $B_7B_6B_5B_4B_3B_2B_1B_0$ را دقیقاً بکشید.

اعداد صحیح را n بیتی در نظر بگیرید و این مدار درختی را از نظر تعداد گیت‌ها و از نظر سرعت تولید خروجی با مدار پلکانی مقایسه کنید.

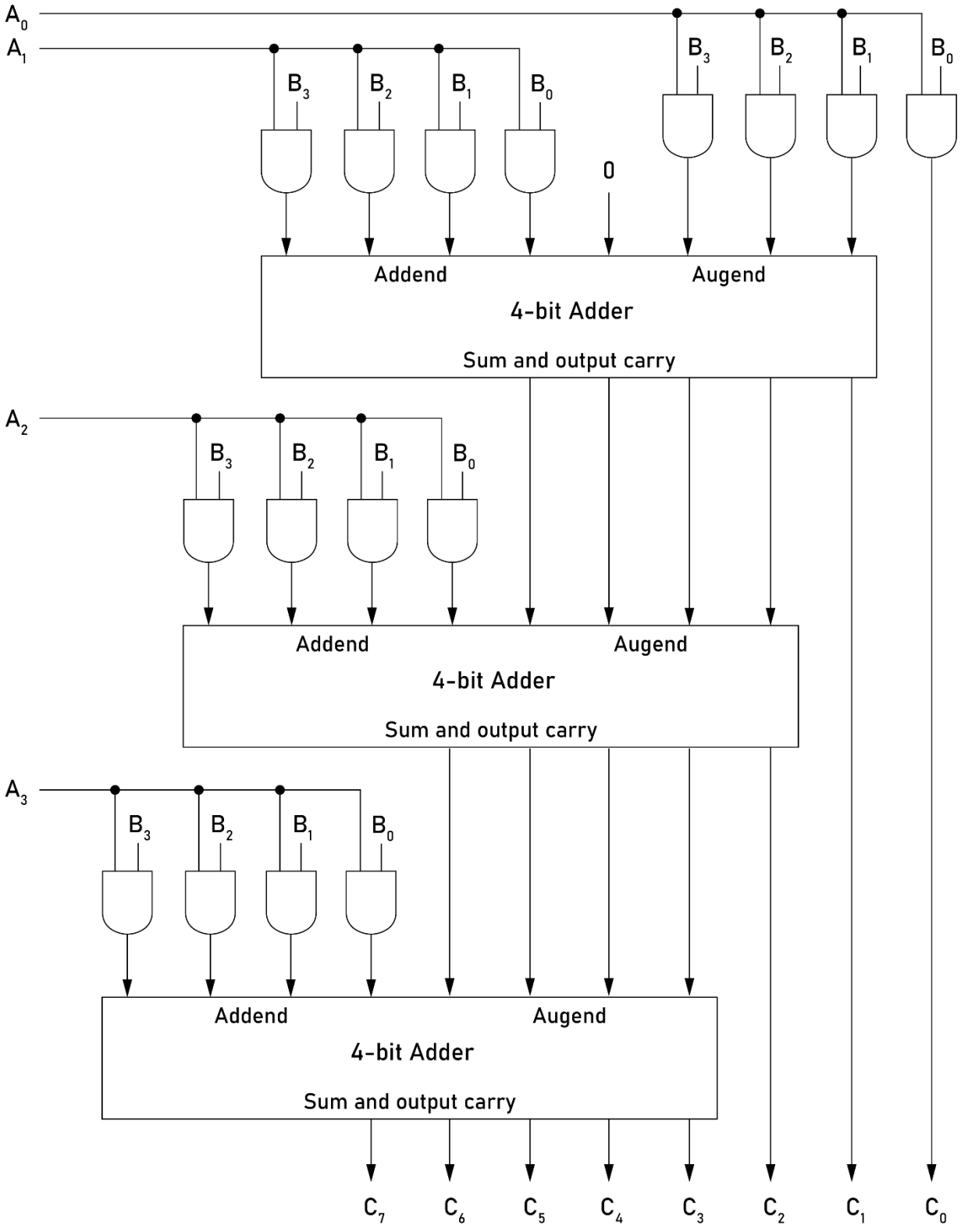
جواب:

الف) می‌توانیم از الگوریتم ضرب مدرسه‌ای استفاده کنیم. یعنی به شکل زیر هر بار یک رقم عدد اول را در کل عدد دوم ضرب (and) کرده و حاصل‌ها را جمع کنیم (خروجی به اندازه تعداد بیت‌های عدد دوم به علاوه یک بیت اولی است) سپس رقم اول حاصل آن را به خروجی اضافه کرده و بقیه حاصل را با ضرب رقم دوم عدد اول در عدد دوم جمع کنیم.

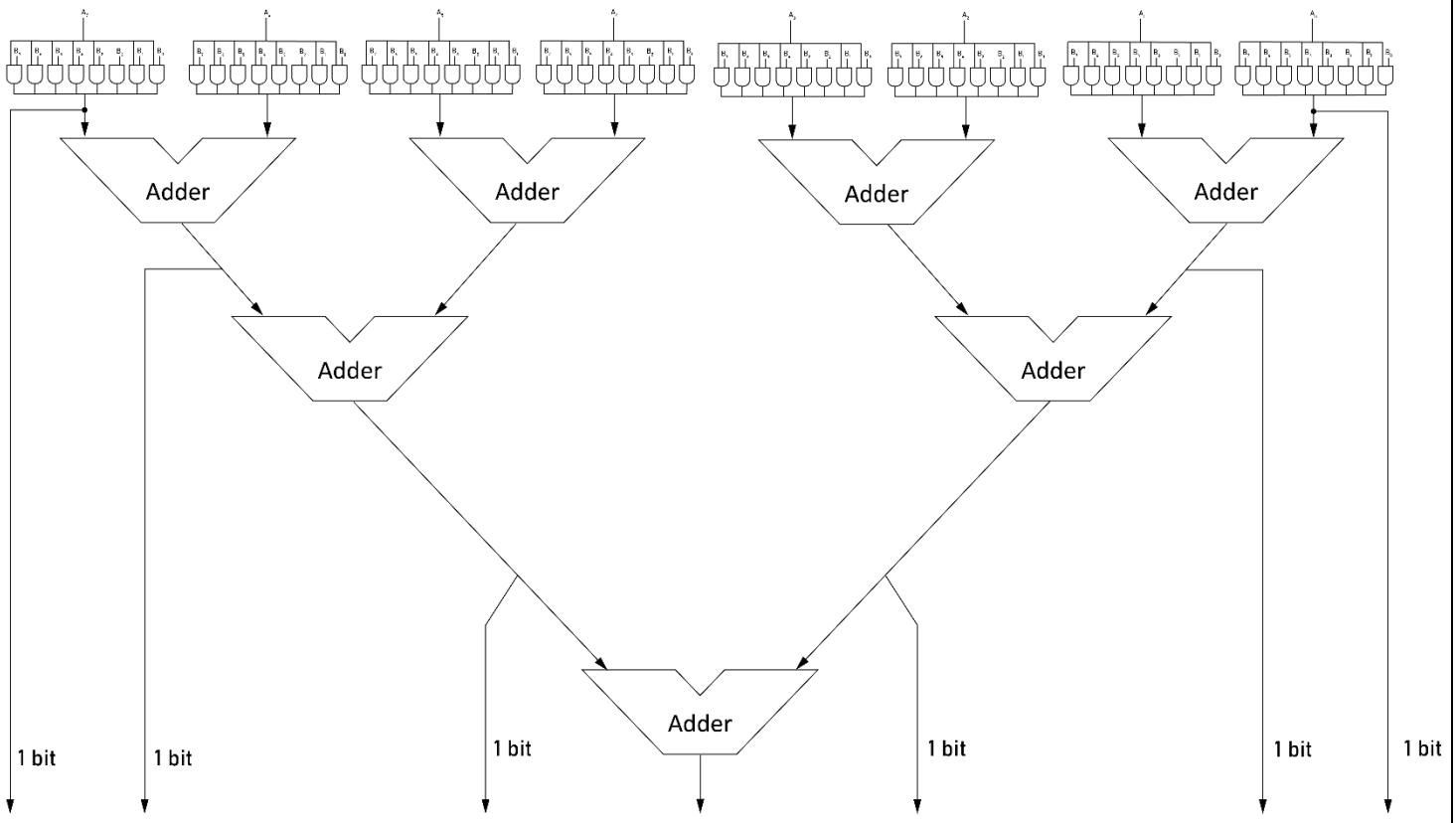


اگر به همین شکل ادامه دهیم برای هر رقم عدد اول یک Adder داریم، به جز دو رقم اول که and‌شان با عدد دوم با چیزی قبل از آن جمع نشده و توسط یک Adder با هم جمع می‌شود. پس به تعداد یکی کمتر از بیت‌های عدد اول Adder داشته، و حاصل نیز از هر Adder به جز Adder آخر یک بیت گرفته، و از آخری تعداد یکی بیشتر از بیت‌های عدد بیت می‌گیرد. پس، برای عدد ۳۲ بیتی، ۳۱ Adder داریم که حاصل نهایی آن ۳۱ + (۳۲+۱) = ۶۴ بیت می‌باشد.

مدار برای نمونه‌ای با دو عدد ۴ بیتی در شکل زیر آورده شده است.



ب) با همان الگوریتم و ضرب (and) هر بیت عدد اول در کل عدد دوم، به کمک قانون مور می‌توانیم برای سریع تر کردن ضرب، یک Adder برای هر بیت قرار دهیم که به طور موازی ضرب‌ها پیش روند؛ بدین صورت کمترین میزان انتظار را داریم. نمونه ۸ بیتی در شکل زیر آورده شده است. تمام Adder های چپ و راست جدا محاسبه شده و بعد با هم وارد Adder پایینی می‌شوند.



نکته: پس از بررسی راه که از کتاب آورده شده است متوجه شدم که Adder ها در مراحل پایینی تعداد بیشتر از ۸ بیت (برای مثال کلی ۳۲ بیت) نیاز دارند تا بتوانند اعمال را انجام دهند. پس با ۳۱ جمع‌کننده ۳۲ بیتی نمیتواند انجام دهد.

مراجع:

کتاب منبع درس مدار منطقی برای بخش الف:

Digital Design (Sixth Edition) – Mano & Ciletti

کتاب منبع درس حال حاضر برای بخش ب:

Computer Organization and Design (RISC-V Edition) – Patterson & Hennessy

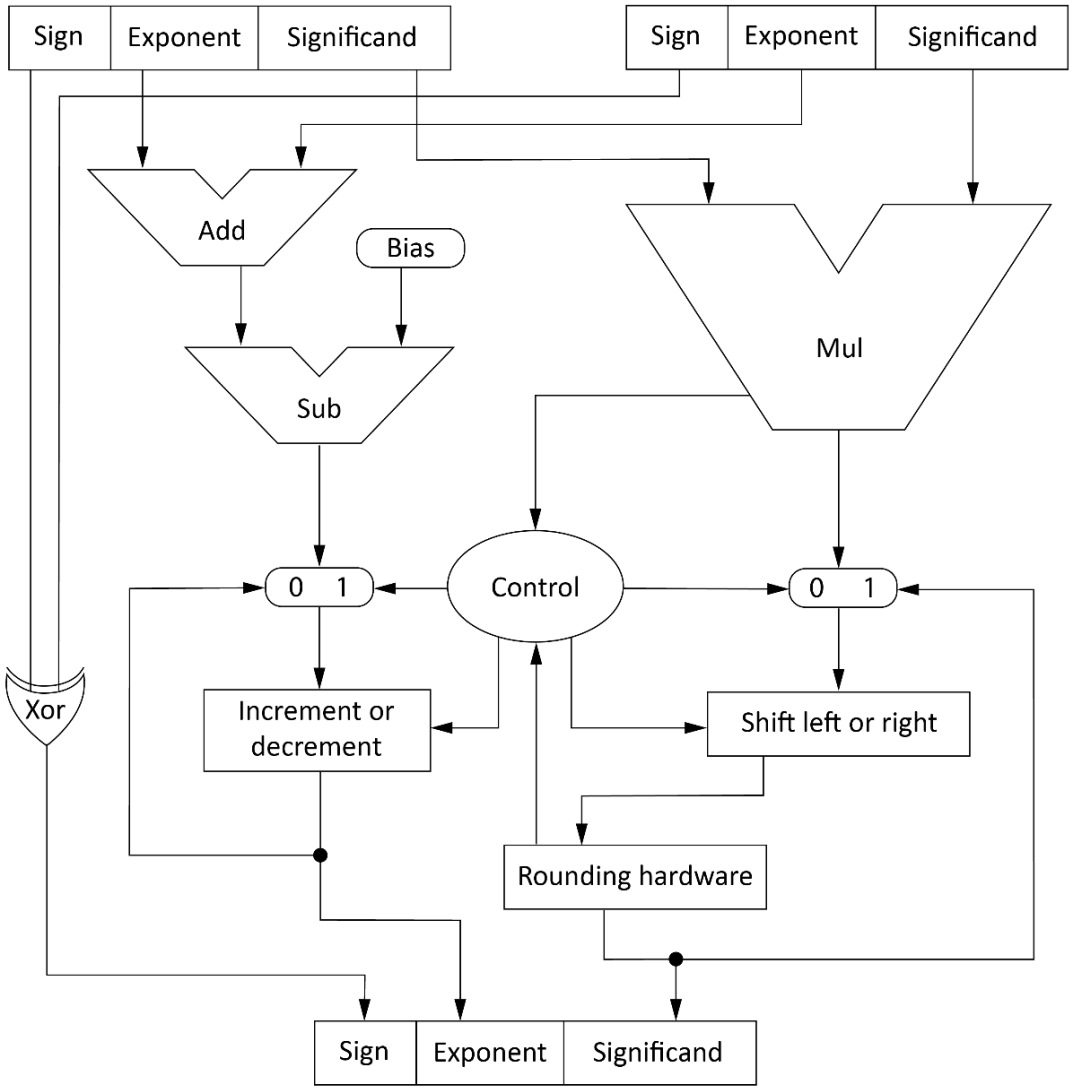
۴. در شکل ۱۴.۳ کتاب درسی، روندنمای یک الگوریتم برای جمع اعداد ممیز شناور دودویی نشان داده شده است. و در شکل ۱۵.۳ سخت‌افزاری اختصاصی که آن الگوریتم را اجرا می‌کند، نشان داده شده است.

الف) شما از روی روندنمای شکل ۱۶.۳ کتاب درسی، که توصیفگر یک الگوریتم برای ضرب اعداد ممیز شناور دودویی است، و با الگوبرداری از مدار منطقی پیاده‌ساز الگوریتم جمع در شکل ۱۵.۳ ، سخت‌افزاری اختصاصی برای اجرای الگوریتم ضرب طراحی کنید.

ب) گام به گام توضیح دهید که مدار منطقی که ساخته‌اید چگونه حاصل‌ضرب دو عدد ممیز شناور را محاسبه می‌کند.

جواب:

الف) با توجه به الگوریتم شکل ۳.۱۶ مدار منطقی سخت افزار را داریم:



ب) مدار گام‌های زیر را طبق الگوریتم دارد:

۱) با Add توان انحراف دار دو عدد را جمع می‌کنیم. و سپس انحراف را از مقدار حاصل با Sub کم می‌کنیم.

۲) در همان حال، significand آن‌ها را با واحد Mul در هم ضرب می‌کنیم.

۳) حاصل هر کدام (یعنی Sub و Mul) را به واحد کنترل برده و اگر نیاز بود به Mux می‌بریم و با جابه‌جایی حاصل ضرب (توسط واحد Shift) و افزایش حاصل تفریق (توسط واحد Increment)، آن‌ها را عادی می‌کنیم.

۴) حاصل عمل بر بخش اصلی (significand) را با Rounding hardware گرد می‌کنیم.

۵) به واحد کنترل می‌رویم تا بررسی کنیم آیا همچنان عادی‌ست. اگر بود دو حاصل را در نتیجه نهایی قرار می‌دهیم، در غیر این صورت دوباره دو حاصل را به Mux برده و مراحل عادی سازی را انجام می‌دهیم.

۶) علامت حاصل نهایی را از xor علامت دو ورودی به دست می‌آوریم.

مراجع:

۵. الف) معماری ARM در سال‌های اخیر پراستفاده‌ترین معماری برای ساخت رایانه‌های موجود در بازار بوده است. معماری ARM در واقع مجموعه‌ای از معماری‌ها است که از وجوهی با هم اشتراک دارند. مهم‌ترین اصول مشترکی را که مبنای طراحی گونه‌های مختلف این معماری بوده است مشخص کنید.

ب) مهم‌ترین نسخه‌های معماری ARM که در سال‌های اخیر برای تولید رایانه استفاده شده‌اند ARMv7 و ARMv8 هستند. مهم‌ترین وجوه تشابه و مهم‌ترین وجوه تفاوت این دو نسخه را مشخص کنید.

پ) با مثال توضیح دهید که فلسفه طراحی کدام یک از دو نسخه ARMv7 و ARMv8 به فلسفه طراحی RISC-V شبیه‌تر است.

ت) معماری ARM در سال‌های اخیر برای ساخت چه نوع رایانه‌هایی مورد استقبال شرکت‌های سازنده رایانه قرار گرفته است؟ آخرین نسخه این معماری، ARMv9 ، با چه اهدافی عرضه شده است؟ آیا ممکن است که در آینده، سازندگانی که در حال حاضر برای ساخت رایانه از معماری ARM استفاده می‌کنند، معماری RISC-V را جایگزین معماری ARM کنند؟

جواب:

الف) معماری ARM همانطور که از نامش (Advanced RISC Machine) پیداست، یک نوع ماشین پیچیده RISC است که تمرکز آن بر دستورات ساده اما قویست کار سخت افزار را ساده‌تر کرده و پیچیدگی و انعطاف را در بخش نرم‌افزار متمرکز کرده است. به طور کلی، پردازنده ساده‌تر بوده و می‌تواند با دور ساعت بیشتری کار کند، پس سرعت و بازده بالا می‌رود.

ب) نسخه ARMv8 اولین معماری سری ARM است که به پردازش ۶۴ بیتی گسترش یافته و بر خلاف ARMv7 قابلیت بالاتری دارد. این نسخه عملا تمامی خصوصیات غیرعادی نسخه v7 را کنار گذاشت:

۱) فضای اجرای شرطی که در تمامی دستورات v7 موجود بود حذف شد.

۲) فضای immediate از ورودی به یک تابع که مقدار ثابت می‌ساخت، به یک مقدار ثابت ۱۲ بیتی تغییر کرد.

۳) دستورات Load Multiple و Store Multiple حذف شدند.

۴) PC که شاخه‌های غیرقابل پیشبینی را در صورت نوشتن بر آن ایجاد می‌کرد دیگر یکی از رجیسترها نیست.

همچنین خصوصیاتی که در نسخه قبل موجود نبوده و در MIPS کاربردی بودند اضافه شدند:

۱) نسخه v8 رجیسترهای همه منظوره دارد که یکی از آن‌ها مختص صفر است، که البته در دستورات load و store نشان دهنده‌ی اشاره‌گر پشته است.

۲) حالت‌های آدرس‌دهی آن بر خلاف v7 برای تمامی اندازه کلمات کار می‌کنند.

۳) همراستا با MIPS، دستورات شاخه کردن در صورت برابری و عدم برابری اضافه شدند.

همانطور که می‌بینیم، تفاوت‌های زیاد ARMv8 با ARMv7 آن را شبیه‌تر به RISC-V کرده و عملا شباهت اصلی این دو نسخه در بعضی کلیات مانند بر پایه RISC بودن و تمرکز بر بازده بالاتر (یعنی نامشان) است.

پ) همانطور که گفته شد، تغییرات زیاد ARMv8 نسبت به ARMv7 بر مبنای کاربردهای MIPS که شبیه‌ترین معماری به RISC-V است، ARMv8 را بسیار به RISC-V شبیه کرده است. برای مثال، مقدار ثابت ۱۲ بیتی immediate، رجیسترهای همه منظوره (که یکی مختص صفر است)، و دستورات شاخه‌کردن برابری و عدم برابری، که همگی میان ARMv8، MIPS و RISC-V مشترک هستند.

ت) با وجود متن باز بودن RISC-V و انعطاف پذیری و کاستن از هزینه‌های آن، ARM به علت امنیت، پروانه تضمینی و پشتیبانی سیستمی وسیع، شناختگی برندی بالایی داشته و محیط بزرگ و پیشرفته‌تری برای خلاقیت و سازندگی دارد. هنوز نمی‌توان پیشبینی کرد که در آینده کدام معماری پر استفاده‌تر باشد، اما قطعا شناختگی و فضای پشتیبانی شده‌ی ARM به راحتی جایگزین نمی‌شود. هرچند، حتما در آینده ظهور سیستم‌های رایانه‌ای پیچیده بیشتری تحت RISC-V را خواهیم دید.

مراجع:

کتاب منبع درس:

Computer Organization and Design (RISC-V Edition) – Patterson & Hennessy

منابع اینترنتی:

https://resources.system-analysis.cadence.com/blog/will-risc-v-replace-arm-in-embedded-systems