تحلیل آزمون دوم درس معماری کامپیوتر

دانشکده مهندسی کامپیوتر، دانشگاه صنعتی شریف

آرین احدی نیا

شماره دانشجویی: [redacted]

استاد درس: جناب آقای دکتر جهانگیر

دستیار آموزشی: جناب آقای علیپور

پاییز ۱۴۰۰

فهرست عناوین

سوالات آزمون

سوال ۱. مدهای آدرس‌دهی

$$
\begin{aligned}
T_x \quad &: \quad AR \leftarrow IR[15:0] + RegFile[IR[16:20]] \\
T_{x+1} \quad &: \quad AR \leftarrow AR \ll 2, RegFile[IR[16:20]] \leftarrow RegFile[IR[16:20]] + 1 \\
T_{x+2} \quad &: \quad AR \leftarrow MEM[AR]
\end{aligned}
$$

توجه بفرمایید که مراحل اجرای دستور فوق در سه مرحله به صورت ترتیبی (Sequential) اجرا میشود. توجه بفرمایید که دستورالعمل‌های این کامپیوتر حداقل ۲۱ بیت دارند که به صورت زیر است

20	16	15	0
...	$IR[20:16]$	$IR[15:0]$	

توجه بفرمایید که $IR[10:16]$ آدرس یک رجیستر است. چرا که از آن به صورت $RegFile[IR[16:20]]$ استفاده کرده‌ایم. و همچنین مقدار $IR[15:0]$ یک مقدار Immediate است چرا که از آن مستقیم در محاسبات استفاده کرده‌ایم. اگر مقدار رجیستر $IR[16:20]$ را R و مقدار Immediate را I در نظر بگیریم، مقدار آدرس نهایی از رابطه

$$4 \times (I + R)$$

محاسبه می‌گردد و توجه کنید که دو واحد شیفت به چپ معادل ضرب در چهار است. در نهایت از حافظه، مقداری که در آدرس فوق قرار گرفته است را فراخوانی می‌کنیم.

توجه بفرمایید که در حین محاسبه آدرس، مقدار R یک واحد افزایش پیدا می‌کند بنابرین آدرس دهی از نوع Auto Increament می‌باشد.

همچنین آدرس دهی فوق، نسبی نیست چرا که از مقدار PC در آن استفاده‌ای نشده است.

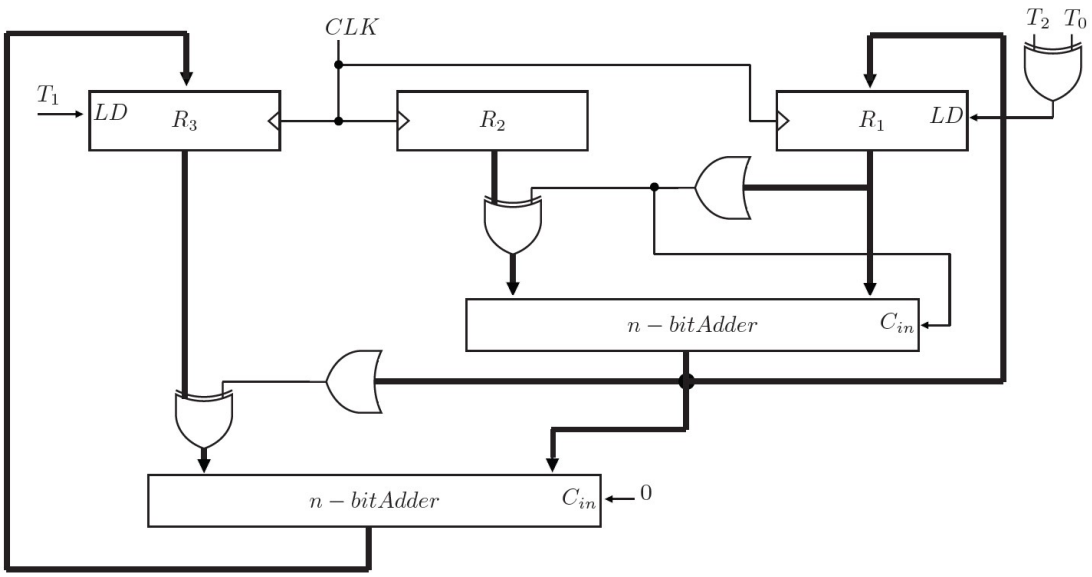
تعریف مد آدرس دهی Index به صورت زیر است.

"در آدرس‌دهی Index، مقدار ثابت Index با بخش آدرس دستورالعمل جمع می‌شود تا آدرس مورد نظر را بسازد."

اما در آدرس‌دهی که ما انجام می‌دهیم، این مقدار جمع را در چهار ضرب می‌کنیم. این ضرب در چهار به طور بالقوه به این دلیل است که مشابه MIPS در پردازنده مورد نظر Address Alignment انجام شده است و حافظه ۴ کلمه آدرس پذیر است. همچنین ما از یک رجیستر ثابت برای Index استفاده نمی‌کنیم اما علی‌ای‌حال، اگر این ضرب در چهار صورت نگیرد، می‌توانیم آدرس‌دهی را به صورت ترکیب Index و Auto Increament در نظر بگیریم. گزینه (د)

شیفت دادن به خودی خود اصلا ایرادی ندارد. کما اینکه در میپس ما دیدیم دو عدد صفر به ادرس می چسباندیم در واقع همان شیفت بود. بنابراین کاملا بستگی به طراحی دارد
دانشجویانی که گزینه الف را انتخاب کردند نیز کاملا درست است

سوال ۲. زبان RTL



مدار فوق را مرحله له مرحله تحلیل می‌کنیم. توجه کنید که دو جمع‌کننده در مدار فوق داریم. خروجی جمع کننده اول وارد جمع‌کننده دوم می‌شود. بنابرین به طور سلسله‌مراتبی، ابتدا خروجی جمع‌کننده اول و سپس خروجی جمع‌کننده دوم را بررسی می‌کنیم.

توجه بفرمایید که XOR مانند یک گیت NOT با ورودی کنترلی عمل می‌کند. به این صورت که اگر یکی از ورودی‌ها ۱ باشد، خروجی نقیض ورودی دیگر و اگر صفر باشد، خروجی خود ورودی دیگر میشود. همچنین OR چندبیتی یک رجیستر، در صورتی که تمام بیت‌های آن صفر باشد برابر صفر و در غیر این صورت برابر ۱ میشود. به بیان ساده‌تر، خروجی OR یک رجیستر در صورتی یک میشود که مقدار آن رجیستر ناصفر باشد.

اکنون آماده‌ایم تا مدار را تحلیل کنیم. اگر R_1 صفر باشد، R_2 نقیض نمیشود و ورودی C_{in} جمع‌کننده اول نیز برابر صفر میشود. بنابرین حاصل جمع‌کننده اول در این حالت برابر $R_1 + R_2$ می‌شود. چون R_1 برابر صفر است، این خروجی برابر R_2 میشود. اگر R_1 صفر نباشد، R_2 نقیض میشود و ورودی C_{in} نیز برابر ۱ میشود. بنابرین خروجی مدار در این حالت برابر $R_1 + \sim R_2 + 1$ می‌شود. توجه کنید که $\sim R_2 + 1$ برابر مکمل دوم R_2 می‌شود که آن را با $\overline{R_2}$ نمایش می‌دهیم. بنابر روابط مکمل دوم، حاصل برابر $R_1 - R_2$ خواهد بود.

اگر خروجی جمع‌کننده اول را S_1 در نظر بگیریم. روابط خروجی جمع‌کننده دوم را نیز می‌توانیم بر حسب S_1 محاسبه کنیم. مشابه جمع کننده اول، اگر S_1 صفر باشد، حاصل جمع‌کننده دوم برابر $R_3 + S_1 = R_3$ می‌شود. اگر S_1 ناصفر باشد، خروجی برابر $S_1 + \sim R_3$ خواهد شد. این مقدار را میتوانیم به صورت

$$S_1 + \sim R_3 + 1 - 1 = S_1 + \overline{R_3} - 1 = S_1 - R_3 - 1$$

خواهد شد.

توجه کنید که مقدار S_1 دو حالت دارد و به ازای هر حالت آن، دو حالت بوجود می‌آید. بنابرین در نهایت چهار حالت داریم.

برای دریافت مقدار جدید در رجیسترها، ورودی LD آن‌ها باید فعال شود. بنابرین شبه‌کد مدار فوق، به صورت زیر می‌شود.

```
on ClockRisingEdge then
    if R1 == 0 then
        // S1 = R2
        if xor (T0, T2) then
            R1 <- R2
        if T1 then
            if R2 == 0 then
                // S2 = R3
                R3 <- R3    // Actually `pass`, whole `if-then` block can be removed
            else
                // S2 = S1 - R3 - 1 = R2 - R3 - 1
                R3 <- R2 - R3 - 1
    else
        // S1 = R1 - R2
        if xor (T0, T2) then
            R1 <- R1 - R2
        if T1 then
            if R1 == R2 then    // R1 - R2 == 0
                // S2 = R3
                R3 <- R3    // Actually `pass`, whole `if-then-else` block can be removed
            else
                // S2 = S1 - R3 - 1 = R1 - R2 - R3 - 1
                R3 <- R1 - R2 - R3 - 1
```

توجه بفرمایید که assign کردن R_3 به خودش کار بی‌معنی به نظر میرسد و می‌توانیم کلا از این بلوک صرف نظر کنیم. کد RTL مدار فوق به صورت زیر خواهد بود. در کد RTL نیز این بخش را به رنگ طوسی نوشته‌ایم. توجه کنید که $NOR(R_1)$ در صورتی ۱ میشود که R_1 صفر باشد.

$$
\begin{aligned}
NOR(R_1), NOR(R_2), T_1 \quad &: \quad R_3 \leftarrow R_3 \\
OR(R_1), EQ(R_1, R_2), T_1 \quad &: \quad R_3 \leftarrow R_3 \\
NOR(R_1), XOR(T_0, T_2) \quad &: \quad R_1 \leftarrow R_2 \\
NOR(R_1), OR(R_2), T_1 \quad &: \quad R_3 \leftarrow R_2 - R_3 - 1 \\
OR(R_1), XOR(T_0, T_2) \quad &: \quad R_1 \leftarrow R_1 - R_2 \\
OR(R_1), NEQ(R_1, R_2), T_1 \quad &: \quad R_3 \leftarrow R_1 - R_2 - R_3 - 1
\end{aligned}
$$

توجه بفرمایید که دستور EQ و NEQ را با گیت‌های منطقی به صورت زیر می‌توانیم پیاده کنیم.

$$EQ(X, Y) = AND\big(XNOR(X, Y)\big)$$

$$NEQ(X, Y) = OR\big(XOR(X, Y)\big)$$

توجه کنید که XNOR دو بیت برابر، برابر ۱ و دو بیت نابرابر، برابر صفر است. بنابرین اگر XNOR همه بیت‌های دو بردار بیتی برابر ۱ باشد، آن دو بردار برابرند. AND در صورتی برابر ۱ خواهد شد که همه ورودی‌های آن ۱ باشد.

سوال ۳. کدگذاری دستورات

توجه بفرمایید که کلمات این پردازنده ۱۶ بیتی است بنابرین دستورالعمل‌های آن ۱۶ بیتی یا ۳۲ بیتی خواهد بود.

حافظه این کامپیوتر ۲۵۶ کیلو کلمه آدرس‌پذیر دارد که معادل 2^{18} کلمه است. بنابرین برای آدرس‌دهی آن نیاز به ۱۸ بیت داریم. بنابرین آدرس‌دهی حافظه تنها در دستورالعمل‌های ۳۲ بیتی امکان پذیر خواهد بود و تنها یک آدرس حافظه را نیز در این نوع دستورالعمل‌ها می‌توانیم مشخص کنیم.

توجه بفرمایید که برای آدرس‌دهی حافظه‌ای مستقیم یا غیرمستقیم نیاز به مشخص کردن آدرس حافظه در دستورالعمل داریم. بنابرین این نوع آدرس‌دهی تنها در دستورالعمل‌های ۳۲ بیتی امکان‌پذیر خواهد بود.

از آنجایی که ۱۰ دستور ۱۶ بیتی داریم، حداقل باید طول Opcode این دستورالعمل‌ها برابر ۴ باشد. بنابرین فرمت دستورالعمل‌ها تا به اینجا به شکل زیر خواهد بود.

4-bit+ Opcode	3 Reg Address

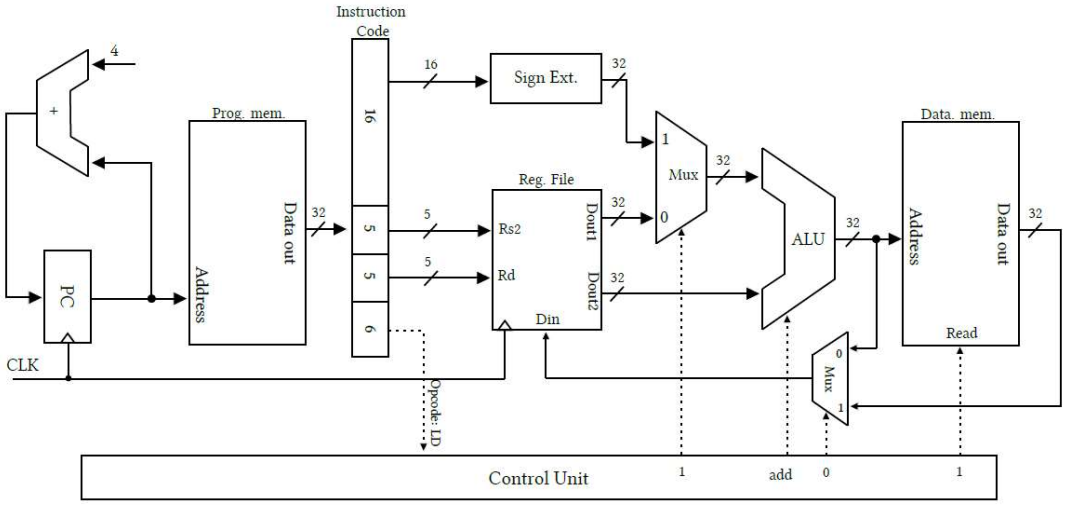
Opcode	2 Reg Address	18-bit Memory Address

توجه کنید که چون دستورالعمل‌ها سه آدرسه هستند، در دستورالعمل اول باید سه آدرس رجیستر و در دستورالعمل دوم باید دو آدرس رجیستر تعیین گردد.

از آنجایی که در دستورالعمل ۱۶ بیتی سه آدرس رجیستر تعیین میگردد. منطقی است که حداقل سه رجیستر داشته باشیم. بنابرین حداقل طول آدرس ثبات برابر ۲ باید باشد. از آنجایی که در حداکثر ۱۲ بیت سه رجیستر را باید آدرس‌دهی کنیم حداکثر طول آدرس رجیستر باید برابر ۴ باشد. بنابرین سه حالت بوجود می‌آید.

- آدرس رجیستر ۴ بیتی: در این حالت اندازه آدرس رجیستر بیشینه و اندازه Opcode دستورات دو کلمه‌ای کمینه می‌گردد. در این حالت **۱۶ رجیستر آدرس‌پذیر** میتوانیم داشته باشیم. طول Opcode دستورات دو کلمه‌ای در این حالت برابر ۶ بیت میشود و Opcode دستورات یک کلمه‌ای ۴ بیتی خواهند بود. توجه کنید که باید راه تمایز میان دستورات یک کلمه‌ای و دوکلمه‌ای داشته باشیم. از ۴ بیت ابتدایی Opcode، ۱۰ حالت آن مورد استفاده دستورالعمل یک کلمه‌ای است و ۶ حالت می‌توانیم برای دستورات دوکلمه‌ای در نظر بگیریم. دو بیت بعد را می‌توانیم آزادانه انتخاب کنیم. بنابرین در مجموع در این حالت **۲۴** دستورالعمل دو کلمه‌ای خواهیم داشت.
- آدرس رجیستر ۳ بیتی: این حالت، حالت میانی است. در این حالت **۸ رجیستر آدرس‌پذیر** میتوانیم داشته باشیم. طول Opcode دستورات یک کلمه‌ای و دو کلمه‌ای در این حالت به ترتیب برابر ۷ و ۸ بیت خواهد شد. از ۱۲۸ ترکیب Opcode دستورات یک کلمه‌ای، ۱۰ مورد آن مورد استفاده است. بنابرین ۱۱۸ حالت را می‌توانیم برای دستورات دوکلمه‌ای در نظر بگیریم. بیت بعدی به صورت آزادانه انتخاب خواهد شد. بنابرین در مجموع **۲۳۶ دستورالعمل دو کلمه‌ای** خواهیم داشت.
- آدرس رجیستر ۲ بیتی: در این حالت اندازه آدرس رجیستر کمینه اندازه Opcode دستورات بیشینه می‌گردد. در این حالت **۴ رجیستر آدرس‌پذیر** میتوانیم داشته باشیم. طول Opcode دستورات یک کلمه‌ای و دو کلمه‌ای در این حالت برابر ۱۰ بیت خواهد شد. از ۱۰۲۴ ترکیب ممکن، ۱۰ حالت مورد استفاده دستورات یک کلمه‌ای است و **۱۰۱۴ حالت برای Opcode دستورات دو کلمه‌ای** می‌توانیم داشته باشیم.

سوال ۴. زمان اجرای دستور Load



واحد	T_{PC}	T_{mem}	T_{reg}	T_{SE}	T_{mux}	T_{ALU}	T_{Add}
تاخیر	$5ns$	$100ns$	$5ns$	$5ns$	$10ns$	$20ns$	$10ns$

این سوال را در دو حالت بررسی می‌کنیم: اجرای ترتیبی و اجرای موازی

اجرای ترتیبی: در اجرای ترتیبی فرض بر آن است که مراحل با توالی کامل و بدون موازی‌سازی اجرا می‌شوند. مراحل زیر به ترتیب برای این منظور باید اجرا شوند. در این محاسبات فرض می‌کنیم که تاخیر PC مربوط به نوشتن در آن است و خواندن از آن تاخیری ندارد. دقت بفرمایید که فاصله بین IR و ALU از دو مسیر متفاوت میگذرد. بنابرین باید بیشینه تاخیر این دو مسیر را در نظر بگیریم. همچنین تاخیر Reg File را برای هر دو عملیات نوشتن و خواندن در نظر میگیریم. (البته عملیات خواندن به وسیله MUX انجام میشود، میتوانستیم برای تاخیر خواندن از تاخیر MUXاستفاده کنیم.)

۱. افزایش مقدار PC (۱۰ نانوثانیه)
۲. نوشتن مقدار جدید PC (۵ نانوثانیه)
۳. دریافت دستورالعمل از حافظه مربوطه (۱۰۰ نانوثانیه)
۴. قرار گرفتن دستورالعمل در IR (بدون تاخیر)
۵. فاصله بین IR تا ALU (مقدار بیشینه: ۱۵ نانوثانیه)
    a. مسیر اول (۱۵ نانوثانیه)
        i. Sign Extend مقدار Base (۵ نانوثانیه)
        ii. گذر مقدار Base از MUX (۱۰ نانوثانیه)
    b. مسیر دوم (۵ نانوثانیه)
        i. خواندن مقدار Rs_2 (۵ نانوثانیه)
۶. محاسبات ALU (۲۰ نانوثانیه)
۷. بارگیری مقدار از حافظه (۱۰۰ نانوثانیه)
۸. گذر مقدار از MUX (۱۰ نانوثانیه)
۹. نوشتن حاصل در Reg File (۵ نانوثانیه)

این فرآیند در مجموع ۲۶۵ نانوثانیه زمان می‌برد. بنابرین فرکانس در این حالت برابر می‌شود با

$$f = \frac{1}{265\ ns} = \frac{1}{265 \times 10^{-9}\ s} = \frac{10^9}{265\ s} = 3.77 \times 10^6\ Hz = 3.77\ MHz$$

یک حالت بینابینی برای موازی‌سازی، این است که تنها افزایش PC را موازی‌سازی کنیم. در این صورت فرکانس برابر $4\ MHz$ خواهد شد.

با موازی‌سازی: دقت بفرمایید که IR یک رجیستر میانی است. از این رجیستر می‌توان به نحو احسن استفاده کرد و موازی سازی انجام داد. به این صورت که زمانی که یک دستورالعمل در حال طی سایر مراحل به جز Fetch است، دستورالعمل بعدی را Fetch کنیم. به این صورت که خروجی Program Memory آماده شود و در لحظه مناسب، CU ورودی Load آن را فعال کند تا دستورالعمل بعدی اجرا شود.

برای موازی سازی، مرحله Fetch که پیش از IR صورت می‌گیرد و سایر مراحل را از یک دیگر جدا می‌کنیم.

طول زمان Fetch کردن دستورالعمل برابر خواهد بود با ۱۱۵ نانوثانیه. توجه کنید که موازی سازی افزایش PC تاثیری در جواب نهایی ندارد.

۱. افزایش مقدار PC (۱۰ نانوثانیه)
۲. نوشتن مقدار جدید PC (۵ نانوثانیه)
۳. دریافت دستورالعمل از حافظه مربوطه (۱۰۰ نانوثانیه)

و زمان اجرای مراحل بعدی برابر خواهد بود با ۱۵۰ نانوثانیه این مراحل عبارتند از:

۱. فاصله بین IR تا ALU (مقدار بیشینه: ۱۵ نانوثانیه)
    a. مسیر اول (۱۵ نانوثانیه)
        i. Sign Extend مقدار Base (۵ نانوثانیه)
        ii. گذر مقدار Base از MUX (۱۰ نانوثانیه)
    b. مسیر دوم (۵ نانوثانیه)
        i. خواندن مقدار Rs_2 (۵ نانوثانیه)
۲. محاسبات ALU (۲۰ نانوثانیه)
۳. بارگیری مقدار از حافظه (۱۰۰ نانوثانیه)
۴. گذر مقدار از MUX (۱۰ نانوثانیه)
۵. نوشتن حاصل در Reg File (۵ نانوثانیه)

بیشینه این زمان‌ها برابر خواهد بود با ۱۵۰ نانوثانیه. بنابرین در سیکل‌های ۱۵۰ نانوثانیه‌ای در حالت بهینه می‌توانیم Load را انجام دهیم. در شکل زیر DEMW منظور Decode-Execute-Memory-Writeback است.

Fetch 1	Fetch 2	Fetch 3	Fetch 4	Fetch 5	Fetch 6	Fetch 7	...
--	DEMW1	DEMW2	DEMW3	DEMW4	DEMW5	DEMW6	DEMW8

بنابرین در حالت بیشینه فرکانس برابر خواهد شد با

$$f = \frac{1}{150\ ns} = 6.66\ MHz$$

سوال ۵. پردازنده LEGO

در این پردازنده فرض می‌کنیم که حافظه برنامه از حافظه داده جدا است. هر چند که این فرض در کلیت موضوع تاثیرگذار نیست.

در این پردازنده، عملیات‌ها را می‌توانیم به صورت زیر دسته‌بندی کنیم.

۱. عملیات SW، که تاثیر آن بر روی حافظه است.
۲. سایر عملیات‌ها که تاثیر آن بر روی ثبات‌ها است.

عملیات ها بر دو نوع می‌توانند روی ثبات‌ها تاثیر بگذارند.

۱. تاثیر بر روی ثبات در register file
۲. تاثیر بر روی program counter

توجه کنید که دستوراتی وجود دارند که این دو تاثیر را همزمان می‌گذارند.

بنابرین دو مدار محاسبه کننده جداگانه برای PC و سایر محاسبات در نظر میگیریم. همچنین دستورات G تنها دستوری است که از مقدار Immediate استفاده می‌کند. بنابرین محاسبه آن را نیز سایر دستورالعمل‌ها جدا می‌کنیم.

حال در دستورات مختلف، ورودی‌های ALU اصلی، ورودی‌های Reg File و ورودی‌های حافظه را مشخص می‌کنیم.

Type	Code	ALU in1	ALU in2	ALU OP	Dest. Reg Addr	Reg Write Enable	Mem Addr	Mem Write Value	Mem Write Enable
L	0	R_B	R_C	Sub	R_A	1	X	X	0
L	1	R_B	R_C	Add	X	0	$R_B + R_C$	R_A	1
L	2	R_B	R_C	Add	R_A	1	$R_B + R_C$	X	0
L	3	R_B	R_C	Comparison	R_A	1	X	X	0
E	4	PC	X	+2	R_B	1	X	X	0
G	5	X	X	X	R_A	1	X	X	0
G	6	X	X	X	R_A	1	X	X	0
O	7	X	X	X	X	0	X	X	0

بنابرین همانگونه که ملاحظه می‌فرمایید نیاز به یک ALU داریم تا عملیات‌های زیر را انجام دهد.

۱. تفریق
۲. جمع
۳. مقایسه
۴. جمع با ۲

بنابرین برای این ALU، ۲ سیگنال کنترلی برای کنترل اینکه چه کاری انجام دهد نیاز داریم. سیگنال‌های کنترلی ALU را به صورت زیر در نظر می‌گیریم.

Operation	F_1F_0
Sub	00
Add	01
Comparison	10
+2	11

که این سیگنال توسط CU مشخص می‌شود.

با جایگذاری دستورالعمل در جدول فوق داریم

Type	Code	ALU in1	ALU in2	ALU OP	Dest. Reg Addr	Reg Write Enable	Mem Addr	Mem Write Value	Mem Write Enable
L	0	$RF[IR[7:4]]$	$RF[IR[3:0]]$	Sub	$IR[11:8]$	1	X	X	0
L	1	$RF[IR[7:4]]$	$RF[IR[3:0]]$	Add	X	0	ALU out	$RF[IR[11:8]]$	1
L	2	$RF[IR[7:4]]$	$RF[IR[3:0]]$	Add	$IR[11:8]$	1	ALU out	X	0
L	3	$RF[IR[7:4]]$	$RF[IR[3:0]]$	Comparison	$IR[11:8]$	1	X	X	0
E	4	PC	X	+2	$IR[3:0]$	1	X	X	0
G	5	X	X	X	$IR[3:0]$	1	X	X	0
G	6	X	X	X	$IR[3:0]$	1	X	X	0
O	7	X	X	X	X	0	X	X	0

با جایگذاری مقادیر Don’t Care خواهیم داشت

Type	Code	ALU in1	ALU in2	ALU OP	Dest. Reg Addr	Reg Write Enable	Mem Addr	Mem Write Value	Mem Write Enable
L	0	$RF[IR[7:4]]$	$RF[IR[3:0]]$	Sub	$IR[11:8]$	1	ALU out	$RF[IR[11:8]]$	0
L	1	$RF[IR[7:4]]$	$RF[IR[3:0]]$	Add	$IR[11:8]$	0	ALU out	$RF[IR[11:8]]$	1
L	2	$RF[IR[7:4]]$	$RF[IR[3:0]]$	Add	$IR[11:8]$	1	ALU out	$RF[IR[11:8]]$	0
L	3	$RF[IR[7:4]]$	$RF[IR[3:0]]$	Comparison	$IR[11:8]$	1	ALU out	$RF[IR[11:8]]$	0
E	4	PC	$RF[IR[3:0]]$	+2	$IR[3:0]$	1	ALU out	$RF[IR[11:8]]$	0
G	5	PC	$RF[IR[3:0]]$	X	$IR[3:0]$	1	ALU out	$RF[IR[11:8]]$	0
G	6	PC	$RF[IR[3:0]]$	X	$IR[3:0]$	1	ALU out	$RF[IR[11:8]]$	0
O	7	PC	$RF[IR[3:0]]$	X	$IR[3:0]$	0	ALU out	$RF[IR[11:8]]$	0

همانگونه که پیشتر گفتیم عملیات ALU با سیگنال F_1F_0 مشخص میشود. برای مشخص کردن ALU in1 و Dest Reg Addr نیاز به یک Mux دو به یک داریم که دستورات از نوع L را از سایرین تمیز دهد. بنابرین یک سیگنال کنترلی Lt نیز در نظر میگیریم که در صورتی که دستور از نوع L باشد فعال می‌شود. همچنین دو سیگنال کنترلی Mem

Write Enable و Reg Write Enable را نیز مطابق آنچه در جدول فوق داریم در نظر میگیریم. آنها را برای اختصار MWE و RWE در نظر میگیریم.

به محض ورود دستورالعمل به IR، $RF[IR[3:0]]$، $RF[IR[7:4]]$ و $RF[IR[11:8]]$ را دریافت می‌کنیم. توجه کنید که ممکن است مقادیر آنها به کار نیاید اما انجام پیشدستانه این کار حسنی که دارید این است که در حین Decode دستورالعمل انجام می‌شود و زمان را برای ما ذخیره می‌کند. همچنین توجه کنید که دریافت مقدار یک ثبات از طریق Mux انجام میگردد. از آنجایی که میتوانیم به تعداد دلخواهی Mux به خروجی رجیسترها وصل کنیم، تمام این عملیات‌ها نیز به طور همزمان اجرا خواهند شد. در نهایت به وسیله یک Mux ورودی اصلی ALU را معین میکنیم.

دستورات نوع G را در نظر بگیرید. توجه کنید که imm در این نوع دستور ۹ بیتی است. ضرب در ۱۲۸ معادل آن است که ۷ صفر در سمت راست آن بگذاریم و ZF آن معادل آن است که ۷ صفر سمت چپ آن بگذاریم. بنابرین یک مدار محاسبه کننده جدا برای آن در نظر می‌گیریم که آن را 2SideZeroFiller مینامیم و با توجه به سیگنال‌های کنترلی که از همان F_0 تغذیه میشود، سمت راست یا سمت چپ را با صفر پر می‌کند.

Operation	F_0
x128	0
ZF	1

توجه بفرمایید که ۴ بیت ابتدایی دستورالعمل را به CU می‌دهیم. سه بیت ابتدایی شامل Opcode است. بیت چهارم نیز در مواردی مانند دستورات O مورد نیاز است اما ممکن است که بیت چهارم استفاده نشود.

همچنین خروجی ALUاصلی، 2SZF و حافظه را در نهایت با استفاده از یک MUX به ورودی Reg File متصل می‌کنیم تا خروجی مورد نظر در ثبات‌ها ذخیره شود.

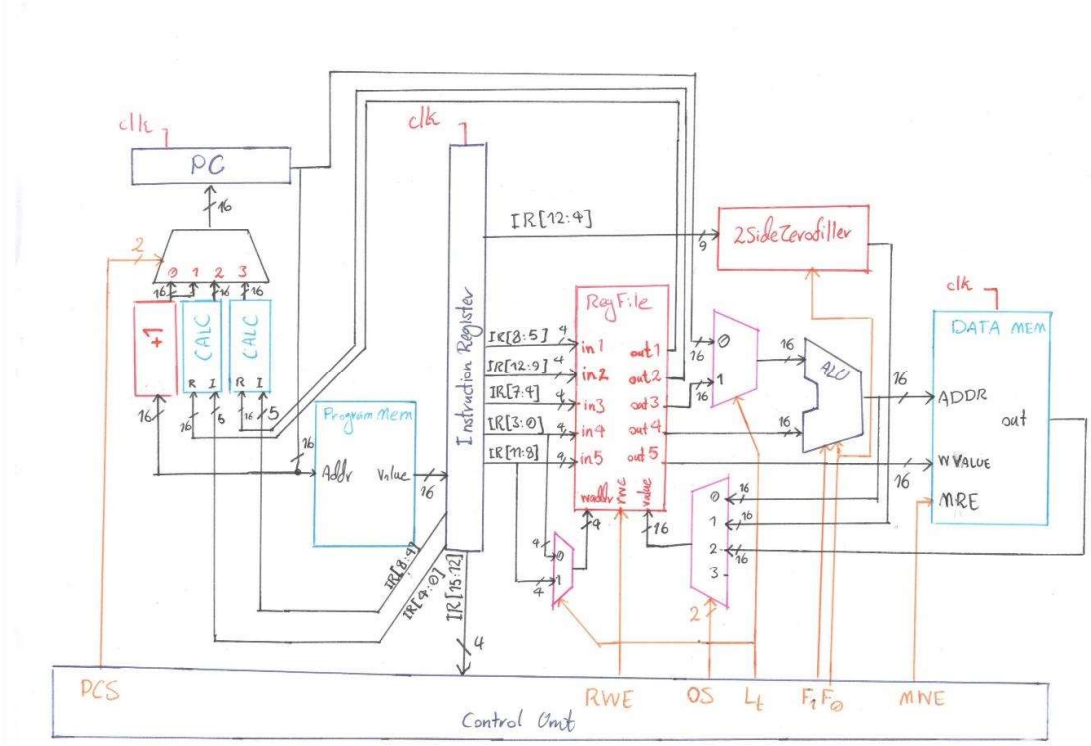
در نهایت به مدار محاسبه‌کننده PC می‌رسد. در حالت عادی، PC در هر مرحله یک واحد افزایش پیدا می‌کند اما ممکن است که در دستور ۴ یا ۷، مقدار جدیدی برای PC محاسبه کنیم. بنابرین یک مدار محاسبه کننده جدا برای PC در نظر می‌گیریم که با توجه به سیگنال‌های کنترلی مقدار جدید را برای PC محاسبه کند.

یک مدار به اسم CALC در نظر بگیرید که یک ورودی ۵ بیتی به نام I و یک ورودی ۱۶ بیتی به نام R دارد. حاصل این مدار برابر خواهد بود با $2R + 2se(I)$. توجه کنید که در هر دو دستور E و O از چنین محاسبه‌ای استفاده می‌شود. مجددا برای افزایش سرعت، هر دو حالت را به طور پیشدستانه حین Decode شدن دستور محاسبه می‌کنیم و در نهایت به وسیله MUX، مقدار مورد نظر را به عنوان ورودی PC در نظر میگیریم.

در نهایت توجه بفرمایید که این پردازنده ۸ بیت سیگنال کنترلی دارد که شامل ورودی‌های انتخابی MUX، ورودی فرمان ALU و ورودی فعال کردن نوشتن Reg File و Data Memory میشود.

بالاخره با تمام این توصیفات و تئوری‌پردازی‌ها، Data Path پردازنده مورد نظر را می‌توانیم به شکل زیر تولید کنیم. در نهایت توجه کنید که این مسیر داده Single Cycle است و تمام عملیات‌ها در ان در یک کلاک انجام می‌شود. بنابرین الزامی است که با توجه به تاخیر گیت‌ها کلاک را به نحوی تنظیم کنیم که فرآیند در طول آن تکمیل شود.

ادامه دارد ...



در نهایت با توجه به طراحی انجام شده، می‌توانیم سیگنال‌های کنترلی را معین کنیم. سیگنال‌های کنترلی به صورت زیر خواهد بود.

Type	Code	PCS	RWE	OS	Lt	F1	F0	MWE
L	0	0X	1	00	1	0	0	0
L	1	0X	0	XX	1	0	1	1
L	2	0X	1	10	1	0	1	0
L	3	0X	1	00	1	1	0	0
E	4	11	1	00	0	1	1	0
G	5	0X	1	01	0	X	0	0
G	6	0X	1	01	0	X	1	0
O	7 (L=0)	0X	0	XX	0	X	X	0
O	7 (L=1)	10	0	XX	0	X	X	0

توجه بفرمایید که در صورتی که MWE و RWE هر دو صفر باشند، هیچ تغییری در حافظه‌ها ایجاد نمی‌گردد. این حالت مانند Pass است. برای حالت دستور ۷ که L برابر صفر باشد، از این روش استفاده میکنیم.

در نهایت توجه کنید که ساخت این سیگنال‌ها به سادگی با جدول کارنو و روش SOP امکان پذیر است.