حل مسائل فصل ۴

کلاس حل تمرین معماری کامپیوتر دکتر زینالی، رامتین کوثری، ۱۵ آبان ۱۴۰۳ کلاس ۲۰۱

تاریخچه (صرفا جهت اطلاع) و تفاوت معماری های CISC و RISC (سوال امتحانی) :

- روش میکرو پروگرمینگ (Micro Programming) برای اولین بار در سال ۱۹۵۱ میلادی پیشنهاد شد.
- در دهه ۱۹۶۰ به عنوان روشی موثر برای پیاده سازی پردازنده های پیچیده با دستورات زیاد شناخته شد.
- در دهه ۱۹۷۰ بسیاری از شرکت ها برای طراحی کامپیوتر از این روش استفاده کردند که اوج استفاده از روش میکرو پروگرمینگ در این دهه بود.
- در دهه ۱۹۸۰ به ظهور معماری ریسک (RISC) و پیشرفت فناوری VLSI استفاده از میکرو پروگرمینگ کاهش یافت و در این دهه تا دهه ۱۹۹۰ استفاده گسترده از آن پایان یافت.
- اوج استفاده از میکرو پروگرمینگ در دهه ۱۹۷۰ بود زیرا انعطاف پذیر بود، یعنی شرکت ها با تغییر ریز برنامه ها می توانستند عملکرد پردازنده ها را بدون تغییر سخت افزار، بهبود دهند.
- کاهش استفاده از روش میکرو پروگرمینگ در دهه ۱۹۹۰ بود زیرا در آن زمان بسیاری از پردازنده ها به سمت معماری RISC و پیاده سازی واحد کنترلی به صورت Hard Wired حرکت کردند و چون میکرو پروگرمینگ سرعت کمتری نسبت به هارد وایرد داشت (زیرا اجرای هر دستور نیازمند چند میکرو دستور بود)، استفاده از آن به شکل چشمگیری کاهش یافت.

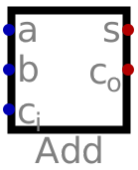
RISC : Reduced Instruction Set Computer

CISC : Complex Instruction Set Computer

	CISC Architecture	RISC Architecture
پیچیدگی دستور	زیاد و پیچیده	کم و ساده
طول دستور	متغیر	ثابت
زمان اجرا	نیازمند چندین چرخه برای اجرای دستورات	اکثرا یک چرخه برای اجرای دستورات
زمان Decode کردن	زیاد به دلیل پیچیدگی	کم به دلیل سادگی
طراحی واحد CU	Micro Programmed	Hard Wired
دسترسی به مموری	زیاد به دلیل میکرو پروگرم بودن	کم به دلیل پیاده سازی شدن سخت افزاری
رجیستر ها	رجیستر های کم و استفاده بیشتر از مموری	رجیستر های عمومی زیاد
راندمان انرژی مصرفی	کم به دلیل پیچیدگی عملیات ها	زیاد به دلیل سادگی عملیات ها
مثال ها	Intel x86 - IBM 360	ARM - MIPS

ماژول ها و مدارهای منطقی مورد نیاز در حل مسائل RTL :

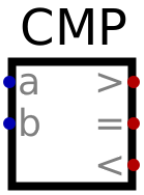
- ### ماژول تمام جمع کننده یا Full Adder



اگر بخواهیم ۲ عدد را با هم جمع کنیم از این ماژول استفاده میکنیم. این ماژول یک خروجی S که حاصل جمع را میدهد و یک C_o که **Carry** خروجی را می دهد را شامل میشود.

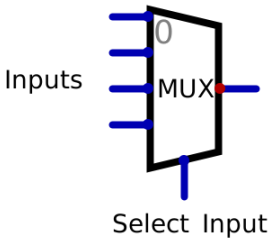
توجه داشته باشید که این ماژول یک ورودی دیگر به نام C_i دارد که **Carry** ورودی نام دارد و از آن میتوان در تکنیک ساخت مدار $2's\ Complement$ استفاده کرد.

- ### ماژول مقایسه کننده یا Comparator



اگر بخواهیم ۲ عدد را با هم مقایسه کنیم از این ماژول استفاده میکنیم. این ماژول ۳ خروجی $a > b$، $a = b$ و $a < b$ دارد و اگر هر کدام از آن ها برقرار شود، خروجی مربوطه ۱ می شود.

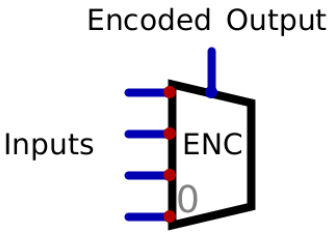
- ### ماژول مالتی پلکسر یا Multiplexer



اگر بخواهیم بین چند ورودی یکی را انتخاب کنیم از این ماژول استفاده میکنیم. این ماژول علاوه بر ورودی ها، یک ورودی انتخاب گر هم دارد که آنها **Select** میگوییم و با توجه به تعداد ورودی ها مقدار بیت های آن تعریف می شود.

توجه داشته باشید که معمولا برای بیت های انتخاب گر یا **Select** از ماژول انکودر استفاده می کنند که این دو ماژول در کنار هم قلب اکثر مدار ها می باشند.

- ### ماژول انکدر یا Encoder



از آنجایی که ورودی انتخاب گر یا **Select** ماژول مالتی پلکسر بر اساس موارد انتخابی کد می شود، یعنی اگر ۴ حالت داشته باشید باید ببینیم این ۴ حالت با چند بیت کد می شود که می شود ۲ بیت زیرا $۴ = ۲^۲$ پس ما برای کد کردن این ۴ حالت به ماژولی نیاز داریم که این کار را برای ما انجام دهد و چه ماژولی بهتر از ماژول انکدر !

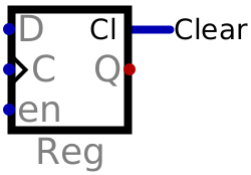
خروجی انکد شده انکدر را باید به ورودی انتخاب گر مالتی پلکسر متصل کنیم.

توجه داشته باشید که ورودی های هر دو ماژول از صفر شروع می شوند یعنی برای مثال اگر ورودی ۳ که آخرین ورودی در تصویر رو به رو می باشد (بالاترین) ۱ منطقی در آن جریان پیدا کند، ۳ منطقی کد می شود و به مالتی پلکسر رفته و مالتی پلکسر هم ورودی ۳ (پایین ترین) خود را انتخاب می کند و به خروجی متصل می کند.

همچنین توجه داشته باشید که تنها یک ورودی در انکدر می تواند ۱ منطقی داشته باشد که اگر بیشتر از این مقدار شود، انکدر اولویت انتخاب خود را از صفر به بالا قرار می دهد. یعنی ورودی ۳ کم ترین اولویت را دارد و از ارزش انتخابی کمتری برخوردار است.

تکنیک های مورد نیاز در حل مسائل RTL :

- **صفر کردن رجیستر R**



اگر در مسائل رابطه ای به شکل $R \leftarrow 0$ داشتیم، می توانیم از پایه Clear در رجیستر استفاده کنیم.

توجه داشته باشید که پایه Clear رجیستر به شکل آسنکرون است.

- **انتقال (ترنسفر) دادن مقادیر یک رجیستر به رجیستر دیگر**

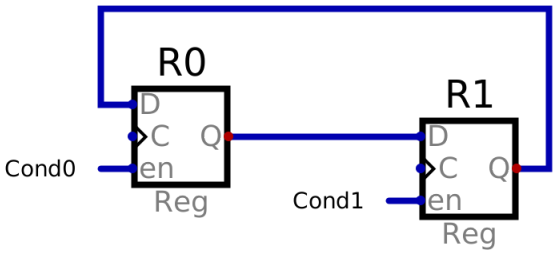
اگر در مسائل روابطی به شکل $R_0 \leftarrow R_1$ یا $R_1 \leftarrow R_0$ داشتیم، می توانیم بین رجیستر ها اتصال برقرار کنیم،

منتها باید حواسمان باشد که شرط (Condition) های Load شدن در رجیستر ها را نیز رعایت کنیم :

برای این منظور شرط ها را

به پایه های en که همان Load

هستند، متصل میکنیم.



- **تبدیل کردن تفریق به جمع با فرض نداشتن مدار تفریق کننده**

اگر در مسائل روابطی به شکل $R_0 \leftarrow R_0 - R_1$ یا $R_0 \leftarrow R_0 - 1$ داشتیم، می توانیم تفریق ها را تبدیل به جمع کنیم که برای این منظور میتوانیم از تکنیک $2's\ Complement$ استفاده کنیم، این تکنیک ساده است.

در $2's\ Complement$ شما باید مقدار را Not کنید و سپس با ۱ جمع کنید :

$$if\ B = 11001 \Rightarrow 2's(B) = Not(B) + 1 = 00110 + 1 = 00111$$

$$if\ A = 11110 \Rightarrow A - B = A + 2's(B) = 11110 + 00111 = 100101 = 5$$

$$\Rightarrow 30 - 25 = 5$$

که بیت قرمز رنگ Carry می باشد و برای ما مهم نیست پس رقم های به جز آن را در نظر میگیریم.

برای پیاده سازی مدار این تکنیک میتوانیم از یک گیت Not و پایه C_i یک Full Adder استفاده کنیم.

در اینجا برای حل $A - B$

ابتدا B را Not کرده، سپس

به FA متصل کردیم. همچنین

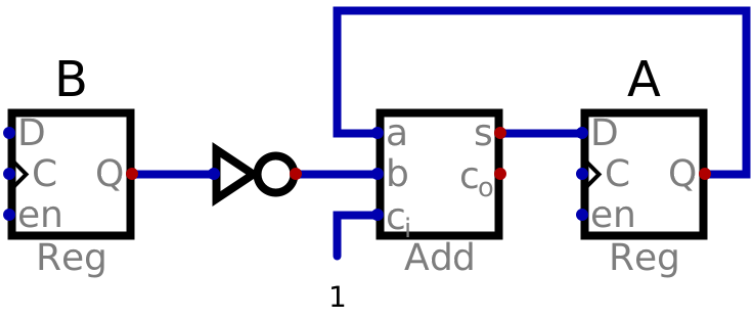
ورودی C_i را به ۱ متصل

کردیم که برای ما ۱+ را

انجام دهد. یک پایه دیگر FA

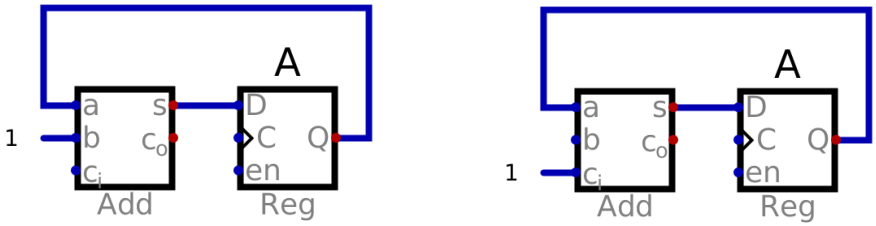
نیز خود A می باشد زیرا داریم $A - B = a + b + C_i = A + Not(B) + 1$.

حال اگر داشتیم $A - 1$ می توانیم همین کار را نیز انجام دهیم.



- **اضافه کردن مقدار رجیستر (۱+) با فرض نداشتن پایه INC**

اگر در مسائل روابطی به شکل $R_0 \leftarrow R_0 + 1$ داشتیم، کافی است یک **Full Adder** داشته باشیم که یکی از ورودی هایش خود R_0 است و برای ورودی دیگر یا به ورودی دوم ۱ میدهیم یا ورودی C_i :



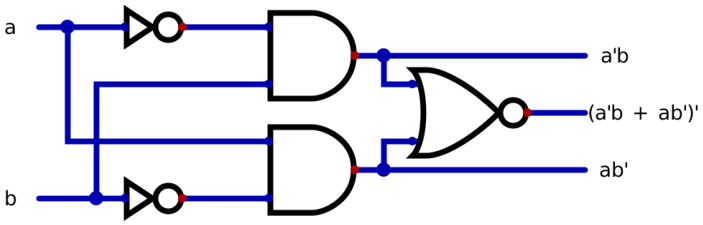
- **انتقال دادن مقدار به حافظه (Memory) و پایه های R و W**

تنها نکته که ای باید در مورد **Memory** رعایت شود، وجود پایه های **Read** و **Write** می باشد که اگر بخواهید داخل مموری دیتا بنویسید باید پایه **W** فعال باشد و همچنین اگر بخواهید از آن چیزی بخوانید، پایه **R** باید فعال باشد.

- **طراحی مدار مقایسه کننده**

اگر شرایط استفاده از ماژول مقایسه کننده را نداشته باشیم می توانیم آن را با گیت های منطقی طراحی کنیم. ابتدا باید با داشتن ۲ عدد a و b، همچنین ۳ حالت $a > b$ ، $a = b$ و $a < b$ جدولش را طراحی کنیم :

a	b	$a < b$	$a = b$	$a > b$
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0



که این جدول را می توان با مدار بالا طراحی کرد.

اگر جدول مدار رسم شده را بنویسیم درمیابیم که دقیقا مانند جدول مقایسه کننده می باشد پس می توانیم از آن به عنوان مدار مقایسه کننده در شرایطی که استفاده از ماژول مربوطه ممکن نبود استفاده کنیم :

a	b	$a' \cdot b$	$(a' \cdot b + a \cdot b')'$	$a \cdot b'$
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0

روش کلی حل مسائل RTL :

- در ابتدا متغیر های وابسته و غیر وابسته به زمان را کنار میگذاریم تا در آخر بررسی کنیم.

- از بین رجیسترها یا مموری های موجود در RTL به دنبال آنکه بیش از بقیه از آن استفاده شده است میگردیم (برای مثال در یک رجیستر بارها مقادیر مختلف ریخته میشود) و آن را انتخاب میکنیم و اول از همه آن را رسم میکنیم، سپس پایه های Clock، Load و پایه Clear (اگر قرار بود مقدار صفر را در رجیستر بریزیم) رسم میکنیم.

- بار دیگر به RTL نگاه میکنیم و بررسی میکنیم چه ماژول های منطقی ای نیاز داریم (مانند مدار تمام جمع کننده یا FA برای جمع و تفریق، مدار مقایسه کننده یا CMP برای مقایسه بزرگتر، کوچکتر یا مساوی بودن و ...).

- حال بررسی میکنیم که چند حق انتخاب داریم ؟ توجه داشته باشید منظور از حق انتخاب این است که در یک زمان بندی خاص یا شرط خاص، یکبار باید یک رجیستر و بار دیگر رجیستر دیگری (یا حتی فرایند دیگری) را استفاده کنیم. برای پیاده سازی همچین محیطی نیازمند دو ماژول مالتی پلکسر یا MUX و انکودر یا ENC هستیم که این دو ماژول در کنار یکدیگر قلب اساسی اکثر این مسائل هستند و حتما باید رسم شوند. پس از رسم، مابقی رجیستر ها یا مموری ها را رسم میکنیم.

- در انتها با گیت های منطقی مناسب و یا نوشتن جبر منطقی متغیرهای وابسته و غیر وابسته به زمان را به نقاط مورد نظر بلوک دیاگرام متصل میکنیم (توجه داشته باشید که مموری، دو ورودی Flag برای خواندن R و نوشتن W دارد که باید با توجه به متغیر ها آن ها را فعال یا غیر فعال کنید). سپس یک CLK رسم کنید و پایه های CLK تمامی رجیستر ها را به آن متصل کنید.

حل مثال های RTL :

در PDF ارسال شده داخل گروه (CA - Exam Solutions) موجود است.