حل مسائل فصل ۵

کلاس حل تمرین معماری کامپیوتر دکتر زینالی، رامتین کوثری، ۲۴ آبان ۱۴۰۳ کلاس ۳۱۰

کمی بیشتر درباره رجیستر ها :

- اگر یک رجیستر را با R_x نمایش دهیم، میتوانیم آن را به ۲ بخش کم ارزش و پر ارزش تقسیم کنیم که بخش کم ارزش را با R_L و بخش پر ارزش را با R_H نمایش میدهیم. برای مثال اگر رجیستر ۱۶ بیتی باشد داریم :

$R = R_x \rightarrow$ تمام بیت های رجیستر

$R_4 \rightarrow$ بیت چهارم رجیستر

$R_{0-7} = R_L \rightarrow$ نصف بیت های کم ارزش

$R_{8-15} = R_H \rightarrow$ نصف بیت های پر ارزش

طراحی واحد DPU و CU

- ### اطلاعات لازم برای طراحی واحد DPU و CU

اگر بخواهیم واحد **DPU** را طراحی کنیم و سپس بتوانیم با تحلیل آن واحد **CU** را بسازیم، به دستورات (Instructions)، رجیستر ها (Registers) و باس (Bus) نیاز داریم.

- ### دستورات (Instructions)

دستور از ۳ بخش مد آدرس دهی، Opcode و Address تشکیل شده است. اگر Opcode ما **n** بیت باشد، یعنی میتوانیم 2^n دستور داشته باشیم.

- ### رجیستر ها (Registers)

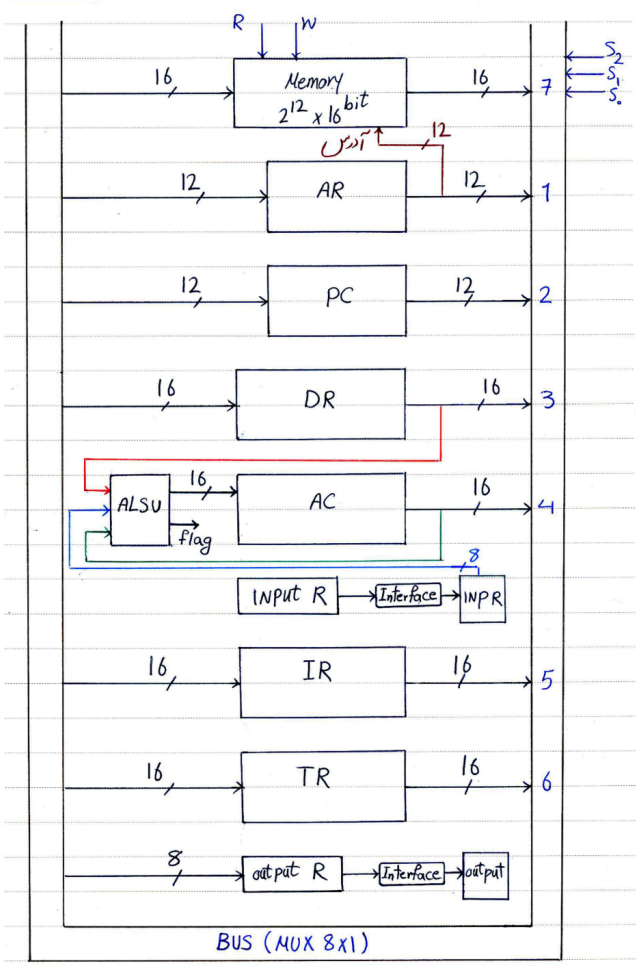
ما به چندین رجیستر برای طراحی دستورات و **DPU** نیاز داریم :

رجیستر	وظیفه	اندازه (بیت)
AC (Accumulator)	رجیستر پیشفرض **(Register Direct)**	۱۶
DR (Data Register)	رجیستر برای ذخیره داده، چون داده ها ۱۶ بیتی هستند، پس ۱۶ بیتی است	۱۶
TR (Temporary Register)	یک دیتا رجیستر **(Data Register)** کمکی برای ذخیره موقت داده ها	۱۶
IR (Instruction Register)	رجیستر برای ذخیره دستورات	۱۶
AR (Address Register)	رجیستر برای نگهداری آدرس و مورد استفاده در پایه آدرس **Memory**	۱۲
PC (Program Counter)	مانند شمارنده خط، چون 2^{12} دستور داریم، انگار برنامه با 2^{12} خط کد داریم	۱۲
Input R (Input Register)	رجیستر برای ذخیره داده های ورودی سیستم مانند کیبورد، موس و ...	۸
Output R (Output Register)	رجیستر برای ذخیره داده های خروجی سیستم مانند مانیتور و ...	۸

- **باس (Bus)**

همانطور که در **DPU** جزوه میبینیم، با احتساب مموری و در نظر نگرفتن رجیستر های ورودی و خروجی ما ۷ حالت داریم که میتوانیم این ۷ حالت را با ۳ بیت ($2^n \geq 7 \Rightarrow n = 3$) کد کنیم. حال برای اینکه بتوانیم یکی از این ۷ ماژول را انتخاب کنیم، باید از مالتی پلکسر استفاده کنیم که ۳ بیت **Select** دارد. پس باس ما یک مالتی پلکسر میباشد. حال حالت هشتم که بلا استفاده است را حالت **Idle** میگوییم که در این حالت از هیچ یک از ماژول ها استفاده نمی شود و تنها در پردازش های موازی و چند هسته ای کاربرد دارد.

- **تحلیل چند RTL در این معماری**



برای نمونه به حل **RTL** زیر دقت کنید :

$$AR \leftarrow TR + M[PC]$$

برای حل از آنجایی که ورودی آدرس رجیستر **AR** است ولی ما **PC** را میخواهیم، میتوانیم **PC** را به **AR** انتقال دهیم :

$$AR \leftarrow PC$$

حال برای اینکه عمل جمع را انجام دهیم باید از **ALSU** استفاده کنیم که ورودی باس آن از سمت **DR** است، پس محتوای مموری در **AR** را انتقال میدهیم به **DR** :

$$DR \leftarrow M[AR]$$

حال چون می خواهیم **DR** را با **TR** جمع کنیم نیاز داریم **TR** را نیز داشته باشیم ولی چون تنها راه داشتن **TR** استفاده دوباره از **DR** است و **DR** نمیتواند دو مقدار همزمان داشته باشد پس ما مقدار **DR** را به **AC** انتقال میدهیم تا بتوانیم دوباره از آن استفاده کنیم :

$$AC \leftarrow DR \quad \text{سپس} \quad DR \leftarrow TR.$$

حال میتوانیم جمع را انجام دهیم :

$$AC \leftarrow DR + AC$$

و در نهایت مقدار **AC** را به **AR** انتقال میدهیم :

$$AR \leftarrow AC$$

همانطور که ملاحظه می کنید، برای انجام این **RTL**، مجبور شدیم ۶ مرحله انجام دهیم، به شکل خلاصه :

$$Step\ 1 : AR \leftarrow PC$$

$$Step\ 1 : DR \leftarrow M[AR]$$

$$Step\ 1 : AC \leftarrow DR$$

$$Step\ 1 : DR \leftarrow TR$$

$$Step\ 1 : AC \leftarrow DR + AC$$

$$Step\ 1 : AR \leftarrow AC$$

دستورات

- **از نظر عملکرد**

اگر بخواهیم دستورات را بر اساس عملکرد طبقه بندی کنیم، میتوانیم به ۳ دسته حسابی $(A + B)$، منطقی $(A \wedge B)$ و کنترلی (JUMP) تقسیم کنیم.

- **از نظر عملوند**

اگر بخواهیم دستورات را بر اساس عملوند طبقه بندی کنیم، میتوانیم به ۳ دسته ارجاع به ثبات، ارجاع به حافظه و ارجاع به ورودی / خروجی (I/O) تقسیم کنیم.

- **۷ دستور معمول در ماشین ها**

اگر بخواهیم چندین دستور معمول و عمومی را در ماشین ها معرفی کنیم، می توان به لیست زیر اشاره کرد :

دستور	نام دستور	RTL
D_0	AND	$AC \leftarrow AC \wedge M[AR]$
D_1	ADD	$AC \leftarrow AC + M[AR]$
D_2	$LDAC$	$AC \leftarrow M[AR]$
D_3	$STAC$	$M[AR] \leftarrow AC$
D_4	BUN	$PC \leftarrow AR$
D_5	BSA	$M[AR] \leftarrow PC \ , \ PC \leftarrow AR + 1$
D_6	ISZ	$M[AR] \leftarrow M[AR] + 1 \ , \ if\,(M[AR] + 1 = 0)\ then\ PC \leftarrow PC + 1$