In The Name OF God

SAYEH

Simple Architecture enough yet Hardware

By Parham Damavandi

Report File

۱.معرفی

سایه یک کامپیوتر ساده با قابلیت های کافی می‌باشد در این گزارش سعی شده نحوه پیاده سازی سایه توضیح داده شود. در ادامه نحوه کار و پیاده سازی اجزا به طور مختصر توضیح داده می‌شود. این پروژه به صورت انفرادی و با استفاده از زبان توصیف سخت افزار VHDL پیاده سازی شده است. امکانات بیشتر و امتیازی در پیاده سازی های آینده پشتیبانی خواهد شد. ظرفیت محاسبات این کامپیوتر ۱۶ بیتی است.

۲.دستورات

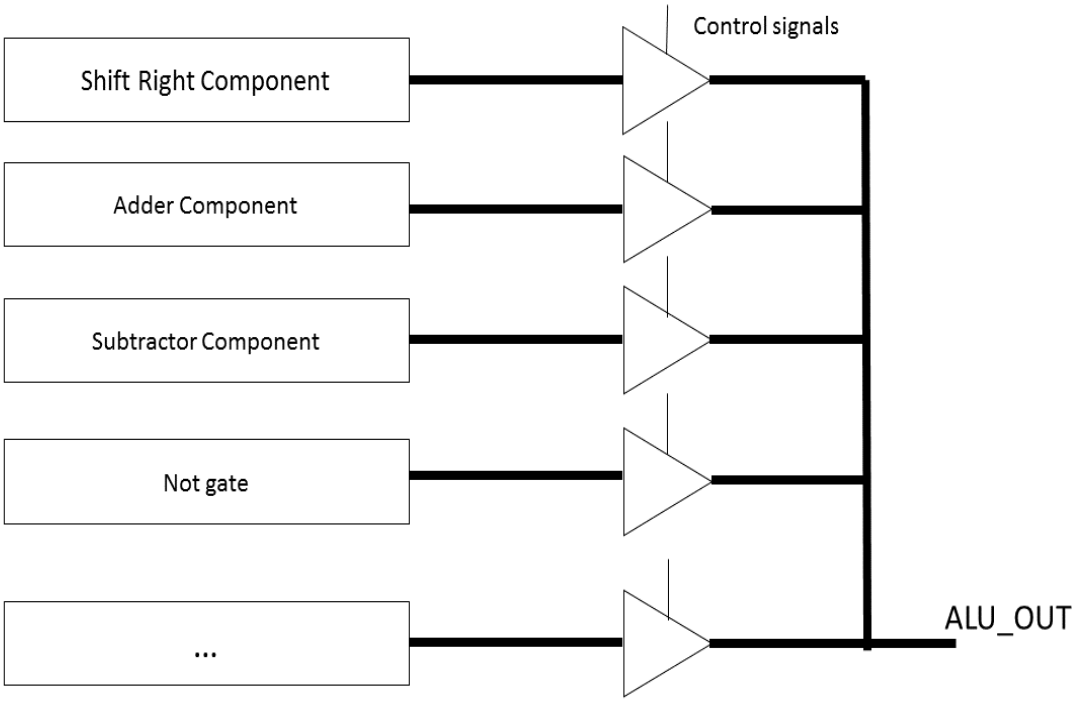
ساختار دستورات در این کامپیوتر به صورت دو نوع ۸بیتی و ۱۶ بیتی است. دستورات ۱۶ بیتی دارای بخش imm برای انتقال دادن آن به اجزای مختلف کامپیوتر است. برای مثال اگر بخواهیم یکی از رجیستر های کامپیوتر را پر کنیم، از این نوع دستورات استفاده می‌کنیم. دستورات ۸ بیتی نیز برای محاسبات منطقی، تغییر دادن فلگ و جابجایی رجیستر ها به‌کار می‌روند.

۳.اجزای کامپیوتر

۳,۱. Arithmetic Logic Unit

واحد محاسبه و منطق به صورت کاملاً ماژولار پیاده سازی شده‌است. بدین صورت که تمامی component ها به طور مجزا پیاده‌سازی شده و در داخل ALUاستفاده شده‌است. خروجی هر جز به یک بافر متصل شده و توسط سیم کنترلی که از واحد کنترل وارد می‌شود مشخص می‌شود که چه خروجی ای توسط ALU نیاز است. این کنترل در واقع یک ورودی ۱۶ بیتی است که instruction مورد نظر در آن decode شده‌است. وقتی خروجی ALU صفر باشد، ZeroFlag یک می‌شود، یعنی با هر خروجی ای که از ALUگرفته می‌شود ZeroFlagو CarryFlag نیز تغییر می‌یابند. اگر خروجی Carry داشته‌باشد Carry نیز یک می‌شود. ALU یک کاربرد دیگر نیز در کامپیوتر ما دارد آن هم این است که ما بخواهیم ۸بیت کم ارزش یا پرارزش IMM را وارد باس اصلی برنامه بکنیم. برای پیاده‌سازی آن از OpandBus استفاده شده است. اگر کنترلر ورودی ALU یک باشد فقط ورودی اولیه از آن خارج می‌شود و هیچ عملی روی آن انجام نمی‌شود. این دستورات مخصوص ورودی دادن به RF از روی

موری و یا وارد کردن داده به موریست. شکل کلی ALU را می‌بینید.



۳,۱,۱.اجزای ALU

۱. **AND (0110):** این کامپوننت دو ورودی RS, RD را با هم AND می‌کند و جواب را در خروجی می‌دهد، همچنین خروجی پرچمی تولید نمی‌کند.
۲. **OR (0111):** این کامپوننت دو ورودی RS, RD را با هم or می‌کند و جواب را در خروجی می‌دهد، همچنین خروجی پرچمی تولید نمی‌کند.
۳. **NOT (1000):** این کامپوننت ورودی RSرا notمی‌کند و جواب را در خروجی قرار می‌دهد. همچنین خروجی پرچمی تولید نمی‌کند.
۴. **Shift Left (1001):** ورودی RS را به سمت چپ شیفت می‌دهد، همچنین خروجی پرچمی تولید نمی‌کند.
۵. **Shift Right (1010):** ورودی RS را به سمت راست شیفت می‌دهد. همچنین خروجی پرچمی تولید نمی‌کند.

۶. ADD(1011): دو ورودی را با پرچم carry جمع می‌کند و جواب را در خروجی قرار می‌دهد. همچنین بیت carry را در carryFlag قرار می‌دهد. برای ساختن ماژول جمع کننده ابتدا یک جمع کننده ی تک بیتی ساخته‌شد سپس ۱۶ تا از آن ها به هم اتصال داده‌شد تا یک جمع کننده‌ی ۱۶ بیتی ساخته‌شود.

۷. SUB(1100) : RS را از RD و از carryFlag کم می‌کند و جواب نهایی را در خروجی قرار می‌دهد سپس carry تولید شده را در خروجی قرار می‌دهد. برای تفریق کردن از همان ماژول جمع کننده استفاده می‌شود با دو تفاوت که متمم RD با RS جمع می‌شود و به carry به طور پیش فرض یک داده می‌شود. در واقع تفریق متمم دو بدین صورت انجام می‌پذیرد.

۸. Mul(1101): در ورژن بعدی پشتیبانی می‌شود.

۹. Compare(1110): دو ورودی RS,RD را باهم مقایسه می‌کند اگر برابر بودند zeroout آن یک می‌شود، اگر rs بزرگتر بود خروجی CO آن یک می‌شود. طراحی مقایسه کننده بدین صورت است که ابتدا مقایسه کننده تک بیتی ساخته شد و سپس با اتصال آن‌ها مقایسه کننده ۱۶بیتی ساخته شد.

واحد محاسبه و منطق نمونه بارز یک ماژول کاملاً سخت افزاری است که حتی برای خروجی‌های آن از پراسس(process) استفاده نشد؛ برای مثال واحد محاسبه و منطق فقط در سه حالت خروجی carry دارد:

۱. وقتی از جمع استفاده شود؛
۲. وقتی از تفریق استفاده شود؛
۳. وقتی مقایسه صورت بگیرد.

کد های آن ها به ترتیب یازده، دوازده و چهارده است. و از اینگونه اتصال برای محاسبه استفاده می‌کنیم:

```
carry_out <= (carries(0) and select_op(11)) or (carries(1) and
select_op(12)) or (grearter and select_op(14));
```

همچنین واحد محاسبه و منطق در دوحالت خروجی ZERO دارد:

۱. وقتی خروجی واحد محاسبه و منطق یک شود؛

۲. وقتی مقایسه صورت بگیرد و هردو برابر باشند.

کد مقایسه چهارده است. و اینگونه برای اتصال استفاده می‌کنیم:

```
zero_out  <= isZero(1) or (equal and select_op(14) and not
select_op(1));
```

بدین صورت واحد محاسبه و منطق پیاده سازی شده است.

۳,۱,۲. ورودی و خروجی های ALU

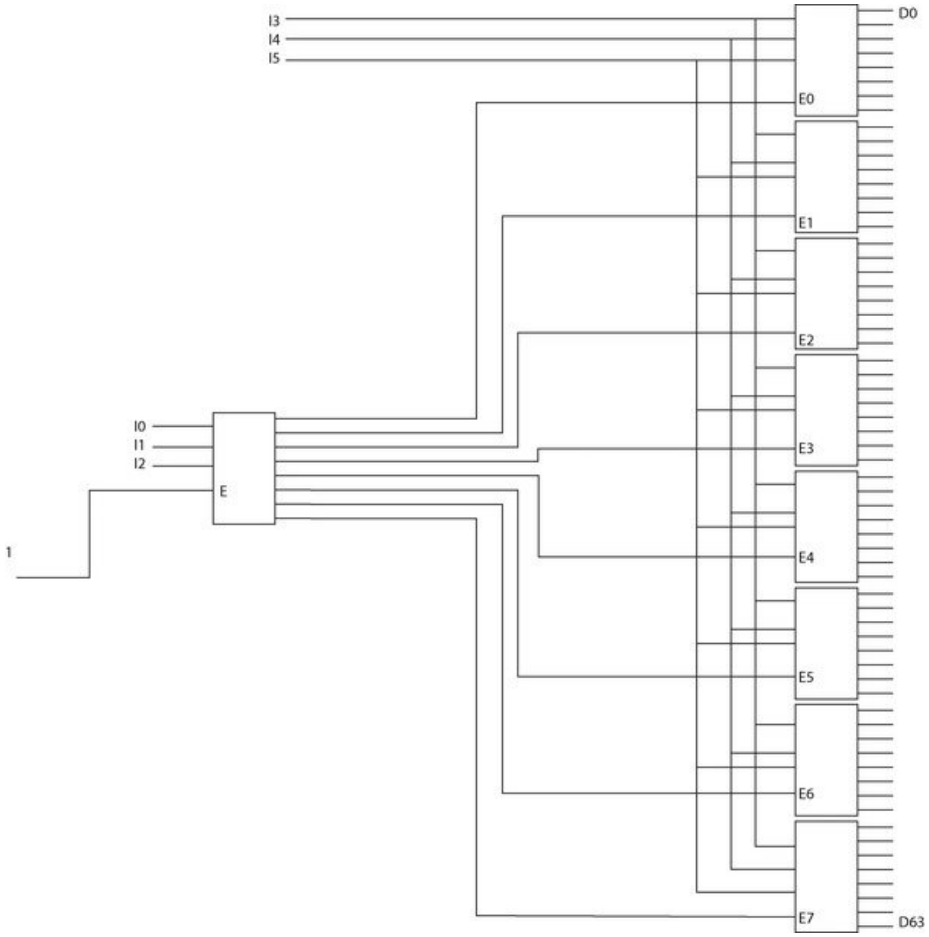
واحد محاسبه و منطق یک ورودی ۱۶ بیتی دارد که عملیات مورد نظر را انتخاب می‌کند که از واحد کنترل وارد می‌شود. دو ورودی RS,RD نیز دارد که روی این دو ورودی عملیات انجام می‌شود. البته بعضی عملیات ها فقط به یکی از ورودی ها نیاز دارد. دیگر ویژگی ALU این است که میتوان ورودی RS را مستقیماً به خروجی ALU داد. ALU طراحی شده دو خروجی تک بیتی پرچم دارد. که در بالا نحوه کار کردن آن توضیح داده‌شد و در نهایت یک خروجی ALU_OUT دارد که عملیات محاسبه شده در خروجی آن قرار می‌گیرد.

۳,۲. رجیستر فایل(Register File)

ظرفیت رجیسترفایل ۶۴*۱۶ بیت است همنطور که در پروژه نیاز بود رجیستر فایل ما دو ورودی برای آدرس دارد یکی از WPوارد می‌شود و دیگری مستقیماً از IR می‌آید. البته اینکه کدام بخش IR وارد RF شود به سایه بستگی دارد. طراحی داخلی RF به صورت ماژولار طراحی شده و برای انتخاب خروجی از ۶۴ رجیستر از دیکدر استفاده شده‌است. اجزای RF به صورت زیر است:

رجیستر ۱۶ بیتی با HL,LL: رجیستر با ۱۶ بیت ظرفیت که دارای دو ورودی مشخص کننده Load است یکی برای ۸ بیت کم ارزش دیگری برای ۸ بیت پرارزش است. طراجی رجیستر به صورت behavioralبوده است.

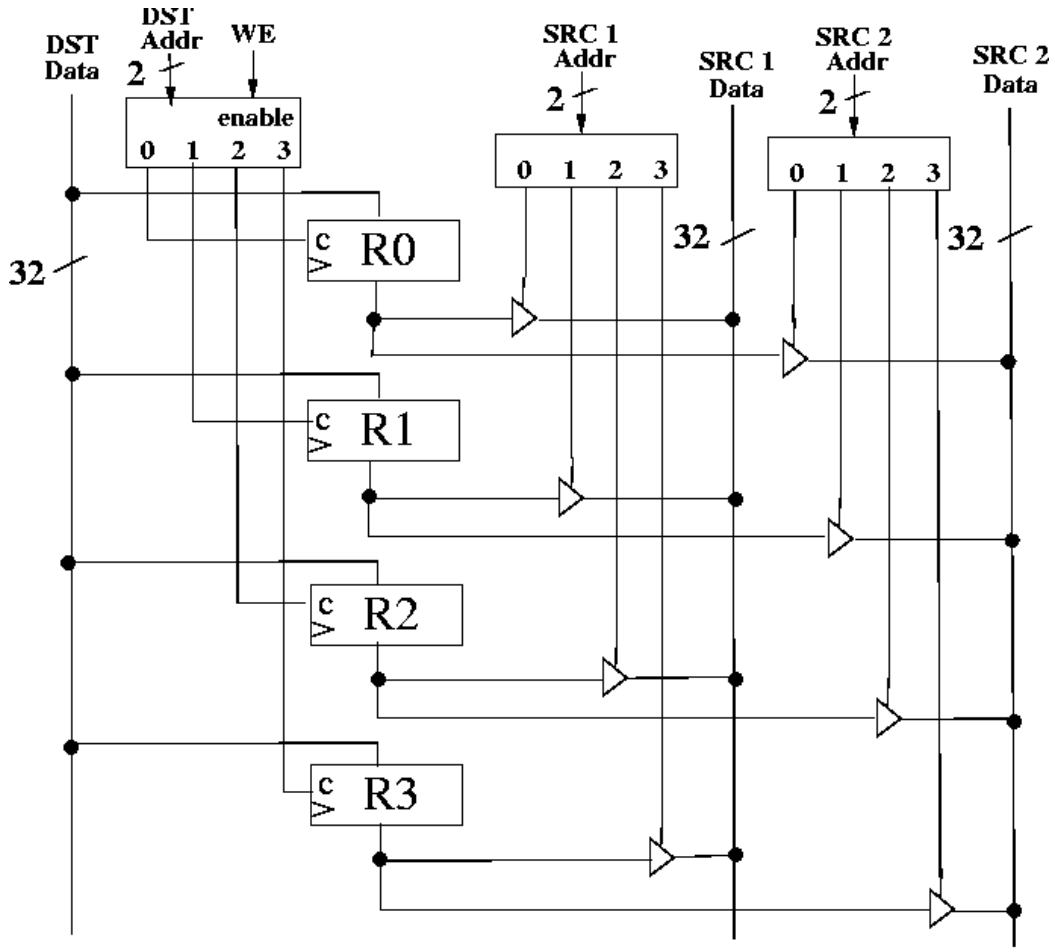
دیکدر 6x64 : برای طراجی این دیکودر خاص از ۸ دیکودر 3x8استفاده شده مطابق شکل زیر



Datapath: ۴ دیکودر که دوتا برای ورودی و دوتای دیگر برای خروجی هستند و ۱۲۸ بافر سه حالته که برای انتخاب خروجی استفاده‌شده است. کنترلر نیز حاصل جمع WPو IR است. طراحی کلی مانند شکل صفحه بعد است با این تفاوت که ما ۶۴ عدد رجیستر داریم و هر رجیستر دو ورودی load دارد.

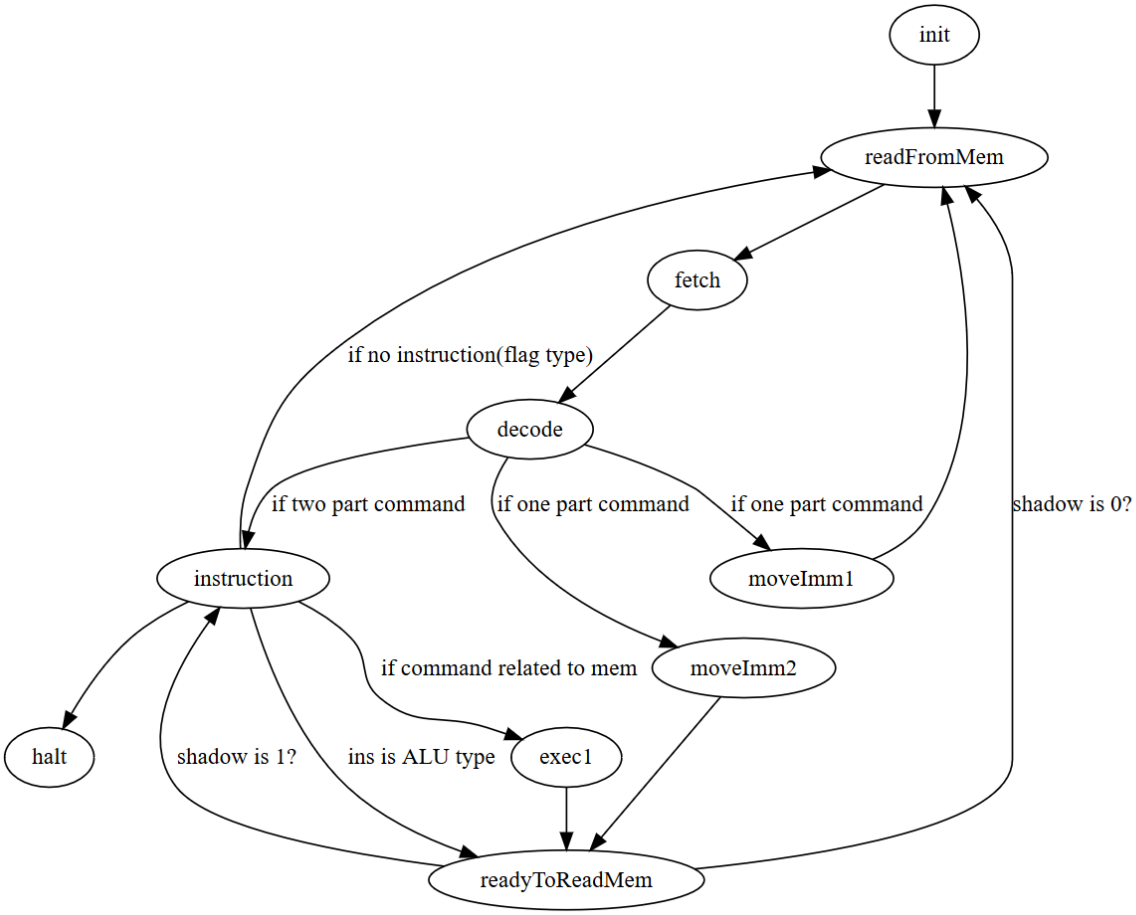
۳,۲,۲. ورودی ها و خروجی های RF

رجیستر فایل ما دو ورودی مربوط به نوشتن دارد که وقتی مثلاً RFLwrite یک شود ۸ بیت کم ارزش رجیستر مقصد با ورودی پر می‌شود. یک ورودی هم از دیتاپس دارد که همان ورودی‌ای است که قصد وارد کردن داده به آن را داریم. دو خروجی هم بستگی به انتخاب دارد که به ترتیب RDوRS نامیده می‌شوند که بر اساسinstruction انتخاب می‌شوند.



۳,۳,۱. واحد کنترل

طراحی واحد کنترل دشوار ترین کار پروژه بود. با اینکه کنتلر طراحی شده کمینه حالات را ندارد ولی نهایت تلاش شده که حالات کمینه باشد. درطراحی های آینده این موضوع حل خواهد شد. این کنتلر از shadow استفاده می‌کند و قابلیت اجرا کردن هر دو دستور ورودی را از یک خانه موری دارد. نحوه‌ی کار کردن آن به طور کامل در ادامه گزارش آمده‌است. آتوماتای آن نیز به شکل زیر می‌باشد.



Init: حالت آغازی اتوماتی ما می‌باشد که درآن موری آماده خروجی دادن می‌باشد.

readFromMem: در این حالت موری ما آماده خروجی دادن می‌باشد.

fetch: خروجی موری ما در DataBus قرار گرفته و دیتا آماده‌ی وارد شدن به IRمی‌شود.

Decode: حال دستور مورد نظر وارد IR می‌شود و اطلاعات موجود در آن دیکد می‌شود. در این حالت مشخص می‌شود که دستور ما از نوع ۸ بیتی یا از نوع ۱۶ بیتی است و حالت بعدی آن از خواندن IR معلوم می شود. اگر دستور ۸ بیتی بود،همواره حالت بعدی instruction است. اگر ۱۶ بیتی بود و چهار بیت پرارزش آن صفر بود حالت بعدی moveImm1 است و اگر چهاربیت پر ارزش یک بود حالت بعدی moveimm2 است.

Instruction: یکی از مهم ترین حالت های این ماشین instruction است. زیرا اگر بخش اول دستور انجام شد، برای اجرای بخش دوم به این حالت برمی‌گردیم. نحوه‌ی اجرا شدن دستور در این حالت دونوع است:

۱. دستور مربوط به آپدیت رجیستر های پرچم است: عملیات در یک کلاک انجام می‌شود و در صورت نیاز باز به همین حالت برمی‌گردیم و غیر این صورت وارد readmem می‌شویم.
۲. دستور مربوط به محاسبات یا جابجایی در موری باشد:در این حالت برای اینکه دیتایی که می‌خواهیم در databus قرار بگیرد، به حداقل یک کلاک نیاز دارد. عملیات طولانی‌تری در پیش داریم که حالت بعدی با توجه به دستور می‌تواند exec1 باشد که ادامه‌ی اجرا است یا می‌تواند readytoreadmem باشد.

moveImm1: دستورات این حالت عمدتا مربوط به عوض شدن مقدار pc است، پس به یک کلاک بیشتر نیاز ندارد و سپس وارد حالت readmem می‌شویم تا دستور بعدی را بر اساس عوض شدن مقدار pc از موری بخوانیم.

moveImm2: در این حالت ما عمدتاً یک مقدار را وارد دیتاباس کرده و سپس مقدار را وارد رجیستر مقصد می‌کنیم که به کلاک اضافه نیاز داریم حالت بعدی ما همواره readytoreadmem است.

Readytoreadmem: این حالت یکی از حالات حساس در آتوماتای ما است در این حالت اگر نیاز به تعویض shadow بود، آن را عوض می‌کنیم در غیر این صورت به حالت readmem می‌رویم که آماده خواندن دستور بعدی می‌شویم.

همانطور که مشاهده شد آتوماتای ما مسیر مورد نیاز برای اجرای یک دستور را پیمایش می‌کند و پس از انجام شدن آن آدرس بعدی وارد IR می‌شود.

۳,۳,۲. دستورات کنترلر

۱. **دستورات مربوط flagها**: این دستورات فلگ ها را با توجه به دستور مورد نظر تغییر می‌دهد. بدین منظور وقتی وارد حالت instruction می‌شویم اگر دستور از نوع فلگ بود، خروجی flag_inputs با توجه به دستور عوض می‌شود؛ برای مثال اگر دستور ما صفر کردن فلگ زیرو باشد، باید خروجی به این صورت شود:

```
flag_inputs <= "00010";
```

۲. **دستورات MIL,MIH** :این دستور مقدار imm را وارد RD می‌کند، البته بستگی به نوع دستور دارد. برای اجرای این دستور وارد حالت moveimm2 می‌شویم و بعد از صبر کردن یک کلاک داده‌ی بعدی را از موری می‌خوانیم.
۳. **دستور save PC:** این دستور مقدار PC را در RD قرار می‌دهد؛ بدین صورت که AddressOnDataBus را یک می‌کند و RFLwriteو RFHwrite را یک می‌کند و بدین منظور مقدار PC در RD ذخیره می‌شود.
۴. **دستور jump Address**: RD+IMM را وارد PC می‌کند و از آن آدرس شروع به شمارش می‌کند برای این کار فقط کافی است مقدار ورودی AU را تغییر دهیم.
۵. **دستور BRZ**:اگر zeroFlag یک باشد، PCما به علاوه imm می‌شود.
۶. **دستور CRZ**: اگر carryFlag یک باشد، PC به علاوه imm می‌شود.
۷. **دستور AWP**: این دستور خروجی WP را به علاوه imm می‌کند.
۸. **دستورات ALU:** در بخش ALU گفته‌شد.

۳,۳,۳. خروجی های واحد کنترل

همانطور که می‌دانیم واحد کنترل برای اجرا کردن دستورات از یک سری سیم های کنترلی باید استفاده می‌کند که خروجی های واحد کنترل می‌باشند. همواره قبل اجرا شدن دستورات حالت بعدی تمامی خروجی ها خاموش می‌شوند تا از اتفاقات غیر منتظره جلوگیری شود. لذا بعد از فعال شدن هرخروجی به مدت یک کلاک روشن می‌ماند که دستورات آن انجام شود. لذا لیستی از خروجی ها در پایین مشاهده می‌شود.

readmem: با آدرسی که به موری داده شده است، خروجی موری در کلاک بعدی روی دیتا باس قرار می‌گیرد.

۱. **writeMem**: مقدار دیتا باس با توجه به آدرس در موری نوشته می‌شود.
۲. **RFLwrite**: ۸ بیت کم ارزش دیتاباس در ورودی رجیسترفایل نوشته می‌شود.
۳. **RFHwrite**: ۸ بیت پرارزش دیتاباس در ورودی رجیسترفایل نوشته می‌شود.
۴. **RS_on_adressUnitRside**: خروجی RS از رجیستر فایل وارد بخش R واحد آدرس می‌شود.

۵. **RD_on_adressUnitRside**: خروجی RD از رجیستر فایل وارد بخش R واحد آدرس می‌شود.
۶. **IR_on_LopandBus**: مقدار imm به صورت ۸ بیت کم ارزش بر روی ورودی RS واحد محاسبه و منطق قرار می‌گیرد. لازم به ذکر است که در دیتاپس، opandBus ورودی RS واحد محاسبه و منطق است و می‌توانیم خروجی RS که در رجیستر فایل است یا imm را وارد ۸ بیت پر ارزش یا ۸ بیت کم ارزش بخش RS واحد محاسبه و منطق کنیم.
۷. **IR_on_HopandBus**: مقدار imm به صورت ۸ بیت پرارزش بر روی ورودی RS واحد محاسبه و منطق قرار می‌گیرد.
۸. **RFright_ON_OpandBus**: مقدار RS که از رجیستر فایل خارج شده وارد بخش RS واحد محاسبه و منطق می‌شود.
۹. **ALU_On_DataBus**: خروجی واحد محاسبه و منطق وارد دیتاباس می‌شود.
۱۰. **WPCLR**: مقدار WP صفر می‌شود.
۱۱. **WPAdd**: مقدار WP به علاوه ی imm می‌شود.
۱۲. **Shadow**: مشخص می‌کند ما از ۸ بیت کم ارزش IR استفاده می‌کنیم یا پرارزش. اگر دستور یک بخشی باشد مقدار آن به طور پیش‌فرض یک است.
۱۳. **IRLoad**: مقدار دیتاباس وارد IR می‌شود.
۱۴. **eanblePC**: مقدار ورودی PC در آن ذخیره می‌شود.
۱۵. **Flag_Inputs**: ورودی FlagRegister است که عملیات مشخص شده را روی آن انجام می‌دهد. این مقدار ۵ بیتی است و در یک زمان فقط یکی از آن ها یک می‌شود. خروجی ها شرح داده شده‌اند:
   - "۱۰۰۰۰" : مقدار carry یک می‌شود.
   - "۰۱۰۰۰": مقدار zero یک می‌شود.
   - "۰۰۱۰۰": مقدار carry صفر می‌شود.
   - "۰۰۰۱۰" مقدار zero صفر می‌شود.
   - "۰۰۰۰۱" مقدار ورودی لود می‌شود.

۳,۴,۱. instruction register

این رجیستر مسئولیت ذخیره سازی موقت دستور حاضر را دارد. وقتی دستور از موری لود می‌شود، در این رجیستر قرار می‌گیرد و اطلاعات آن وارد واحد کنترل می‌شود و از آنجا مستقیماً دستورات اجرا می‌شود؛ لذا این رجیستر بسیار حائز اهمیت است.

۳,۴,۲. ورودی و خروجی ها

این رجیستر یک ورودی ۱۶ بیتی از دیتاباس و یک سیم کنترلر برای لود کردن آن از دیتا باس دارد، اما سیم خروجی از این رجسیتر وارد تمام قسمت های سایه می‌شود. یک بخش آن وارد رجیسترفایل و یک بخش دیگر آن وارد OpandBus می‌شود. وظیفه‌ی این باس انتقال دادن بخش IMM رجیستر دستورالعمل به واحد محاسبه و منطق و انتقال آن به دیتاباس است.

۳,۴,۱. Windows Pointer

WP: ویندوز پوینتر وظیفه ذخیره سازی یک داده مشخص را دارد. این داده به یک خانه شروع حافظه داخلی کامپیوتر سایه اشاره می‌کند در واقع یک اشاره‌گر است که برای انتخاب رجیستر های R0 تا R3 استفاده می‌شود. مقدار ذخیره شده در wp توسط کنترل معین می‌شود. مقدار پیش‌فرض آن صفر است و خروجی آن ۶ بیتی است که برای اشاره به ۶۴ بیت خطوط رجیسترفایل به‌کارمی‌رود.

۳,۴,۲. ورودی و خروجی ها

یک خروجی ۶ بیتی وظیفه اشاره کردن به خانه های رجیستر فایل را دارد و یک ورودی ۶ بیتی نیز برای لود کردن مقدار آن وجود دارد. لازم به ذکر است که مقدار ورودی مستقیماً لود نمی‌شود و با مقدار قبلی جمع می‌شود. یک ورودی کنترل نیز برای ریست کردن این رجیستر و رساندن مقدار آن به مقدار اولیه صفر به آن وارد می‌شود.

۳,۵,۱. Address Unit

واحد آدرس وظیفه نشان دادن آدرس موری مورد نظر است. از واحد آدرس برای دادن آدرس به موری استفاده می‌شود.

۳,۵,۲. ورودی و خروجی ها

واحد آدرس ۶ ورودی کنترل دارد. یک ورودی ۱۶ بیتی برای اشاره به آدرس، یک ورودی که از سمت خروجی های RF وارد می‌شود و یک ورودی IMM که از IR وارد می‌شود. ورودی های کنترل وظیفه مشخص کردن مقدار خروجی آدرس را دارند. البته فقط یک خروجی ۵ بیتی از واحد کنترل داریم که وظیفه مشخص کردن عمل واحد آدرس را دارد ورودی های کنترل به صورت زیر است:

- EnablePc: مقدار خروجی واحد آدرس در PC ذخیره می‌شود لازم به ذکر است که تا این مقدار یک نشود، مقدار آدرس هر تغییری هم بکند در PC ذخیره نمی‌شود.
- pCPlusI: خروجی واحد آدرس برابر با جمع مقدار PC و imm می‌شود.
- PCplus1: خروجی واحد آدرس برابر با جمع PC و یک می‌شود که رایج ترین دستوری است که وارد واحد آدرس می‌شود.
- ResetPC: مقدار خروجی را برابر صفر می‌کند. لازم به ذکر است که تا مقدار EnablePc یک نشود، مقدار PC صفر نمی‌شود.
- RplusI: مقدار خروجی واحد آدرس برابر با جمع R و Imm می‌شود.
- Rplus0: مقدار خروجی واحد آدرس R می‌شود.

۳,۶. Flag registers

رجیستر های پرچم مسئولیت ذخیره سازی تک بیتی را به صورت موقت دارند. نحوه خروجی دادن و ورودی دادن در بالا توضیح داده شد. اینجا فقط ورودی های کنترل آن را بررسی می‌کنیم:

- CSet: مقدار carry را یک می‌کند.
- Zset: مقدار zero را یک می‌کند.
- CReset: مقدار carry را صفر می‌کند.
- ZReset: مقدار zero را صفر می‌کند.
- SRLoad: مقدار ورودی را وارد پرچم ها می‌کند.