به نام خدا

|  |  |  |  |
| --- | --- | --- | --- |
| مدرس: مظفر بگ محمدي | دانشگاه ايلام | ترم دوم سال تحصيلي 96-95 | پروژه ی معماری کامپیوتر |

**اسامی دانشجویانی که پروژه را اخذ کرده اند:**

1. سهراب سرابی، میلاد اکبرزاده 2- منا فاضلی 3-امیرحسین کسایی و کهزادیان 4- فائزه فلاحیان 5- نیکتا حاتمی 6- اسماء قنبری

**توضیحات کلی**

در این پروژه یک پردازنده ی 32 بیتی را طراحی می کنید و آنرا با استفاده از [Logisim](http://ozark.hendrix.edu/~burch/logisim/)  شبیه سازی می نمایید. این پردازنده با پردازنده ی تک سیکلی شباهتهای فراوانی دارد. کسانی که مایل به انجام پروژه هستند، **حتماً** باید موضوع را از طریق ایمیل به اینجانب اطلاع دهند. انصراف از انجام پروژه و یا عدم انجام آن در موعد مقرر با **کسر یک نمره ی کل مواجه** خواهد شد. نمره ی کسانی که برای پروژه ثبت نام نکنند، در همین ماه ثبت دائم خواهد شد و امکان اخذ مجدد پروژه برای آنها وجود ندارد.

 **واحد محاسبات و منطق ALU**

واحد ALU با ان چه در درس گفتیم مشابهت فراوانی دارد. این واحد دارای دو ورودی 32 بیتی به نامهای A و B و یک خروجی 32 بیتی به نام Y است. ورودی 3 بیتی F عملکرد ALU را تعیین می کند. خروجی تک بیتی Zero نیز مشخص می کند آیا Y با صفر برابر است یا نه؟ عملکرد ALU مطابق جدول زیر است:

|  |  |
| --- | --- |
| عملکرد | F |
| A AND B | 0002 |
| A OR B | 0012 |
| A + B | 0102 |
| A - B | 1102 |
| If (A<B) Y=1Else Y=0;  | 1112 |

دستور آخر معادل SLT است. .برای بقیه ی حالتهای F مقدار خروجی Y اهمیتی ندارد. شکل واحد ALU در زیر داده شده است.

****

**بانک رجیسترها**

این بانک دارای 8 رجیستر 32 بیتی است. مقدار رجیستر اول همیشه صفر است. ورودیهای این ماژول به صورت زیر هستند:

|  |  |
| --- | --- |
| Read Register 1 (3 bits) | شماره ی رجیستر خواندنی اول  |
| Read Register 2 (3 bits) | شماره ی رجیستر خواندنی دوم  |
| Write Register (3 bits) | شماره ی رجیستر نوشتن |
| write data (32 bits) | داده ای که باید در رجیستر نوشتن نوشته شود. |
| RegWrite (1 bit) | اگر این سیگنال یک باشد، write data در رجیستر نوشتن نوشته می­شود.  |
| clk (1 bit) | کلاک |

این بانک دارای خروجیهای زیر است.

|  |  |
| --- | --- |
| Read Data 1 (32 bits) | مقدار رجیستر خواندنی اول |
| Read Data 2 (32 bits) | مقدار رجیستر خواندنی دوم |

**پردازنده**

مسیر داده این پردازنده در شکل زیر داده شده است:



همانطور که می بینید این پردازنده با آنچه که در کتاب خواندیم مشابهت زیادی دارد.

**مجموعه­ی دستورالعملها ISA**

دستورالعملهای این پردازنده 32 بیتی هستند. به علاوه، آدرسهای تولید شده توسط پردازنده نیز 32 بیتی هستند. لذا، ما عرض حافظه­ی دستورالعمل و داده را برابر 8 بیت در نظر می گیریم (کاملاَ شبیه کتاب). به عنوان مثال، منظور ما از آدرس 00000008 خانه های هشتم، نهم، دهم و یازدهم حافظه است که یک کلمه­ی 32 بیتی در آن قرار دارد. ساختار دستورالعمل ها در جدول زیر داده شده است. دقت کنید تمام دستورات نوع R دارای کد صفر هستند و از طریق 3 بیت کم ارزش کار آنها مشخص می گردد:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **31-26** | **25-21** | **20-16** | **15-11** | **10-3** | **2-0** |
| 0 | rs | rt | rd | بدون استفاده | funct | دستورات نوع R |
| 1 | rs | rt | immediate (unsigned) | دستور addi rt rs imm یعنی $rt = $rs + imm |
| 2 | rs | rt | immediate (unsigned) | دستور ori rt rs imm مثل addi است. |
| 3 | rs | rt | immediate (signed) | دستور andi rt rs imm مثل addi است.  |
| 4 | target address | دستور jump برای پرش به target |
| 5 | rs | rt | offset (signed) | دستور beq. این دستور شبیه MIPS است.  |
| 6 | rs | rt | immediate (signed) |  دستور lw rt imm(rs) $rt = MEM[$rs + imm]  |
| 7 | rs | rt | immediate (signed) | دستور sw rt imm(rs)MEM[$rs+imm] = $rt |

در ادامه راجع به بعضی دستورالعملها نکاتی ذکر می­شود. 4 بیت باارزش آدرس پرش از روی PC+4 به دست می­آید. در دستور beq مقدار offset به صورت نسبی و علامت­دار در نظر گرفته می­شود و در 4 ضرب می­شود. هم­چنین، با PC+4 جمع می­شود. دقت کنید فیلد immediate  یک فیلد 16 بیتی است. لذا، لازم است قبل از ارسال به ALU گسترش داده شود. در جاهایی که immediate  بدون علامت است، باید 16 صفر به آن اضافه شود. در جاهایی که immediate  دارای علامت باشد، باید طبق بیت علامت گسترش داده شود.

**شبیه ساز Logisim**

اگر در استفاده از Logisim دچار مشکل شدید، آن را دوباره اجرا و مدار را دوباره بارگذاری کنید!. وقت خود را با دنبال کردن باگهای این برنامه هدر ندهید. اگر با اجرای دوباره مشکل حل نشد احتمالا پروژه­ی شما دچار ایراد است.

**ماژول RAM**

ماژول RAM در یک کتابخانه­ی پیش­ساخته­ی حافظ Logisim موجود است. از طریق "Project/Load Library/Built-in Library..." این مازول حافظه را به پروژه اضافه کنید. از RAM به عنوان حافظه­ی داده استفاده کنید. برای حافظه­ی دستورالعمل از ROM استفاده کنید.



از آنجا که ماژول RAM شبیه یک حافظه­ی ایده­آل نیست، ممکن است در استفاده از آن دچار مشکل شوید. شکل فوق نحوه­ی استفاده از RAM را نشان می­دهد.

* ورودی sel تعیین می­کند که آیا حافظه فعال است یا نه؟
* ورودی A مشخص کننده ی آدرس مورد نظر از حافظه است.
* ورودی clr تمام عناصر حافظه را برابر 0 قرار می­دهد.
* ورودی ld مشخص می­کند آیا می­خواهیم در حافظه بنویسیم یا از حافظه داده بخوانیم. اگر ld یک باشد، خروجی out برابر محتویات حافظه در خانه­ی A خواهد بود. اگر ld صفر باشد، داده­ی D در خانه­ی A حافظه نوشته می­شود.
* پورت D هم به عنوان ورودی عمل می­کند و هم به عنوان خروجی. با استفاده از یک بافر سه حالته کاری کرده­ایم که بین DataIn و DataOut هیچ تعارضی پیش نیاید.
* می­توانید از ابزار poke برای تغییر محتوی حافظه استفاده کنید. همچنین، می­توانید روی حافظه کلیک راست کنید و برای بارگذاری مقادیر حافظه از یک فایل گزینه­ی Load Image را انتخاب کنید.
* حافظه ی موجود در کتابخانه 32 بیتی نیست. می توانید برای درست کردن یک حافظه ی 32 بیتی از اساتید حل تمرین کمک بگیرید.

**آزمایش**

برای ازمایش پردازنده، برنامه ی فاکتوریل را پیاده کنید. این برنامه یک عدد را از خافظه می خواند و فاکتوریل آنرا در حافظه می نویسد.

**برنامه­ریزی برای انجام پروژه**

اگر احساس می­کنید انجام این پروژه سخت است، می­توانید از مراحل زیر پیروی کنید:

1. ابتدا کار با RAM را در Logisim یاد بگیرید.
2. ابتدا نحوه­ی کار با رجیسترها در Logisim را یاد بگیرید و بعد سراغ پیاده­سازی بانک رجیستر بروید.
3. ALU را طراحی کنید.
4. ابتدا واحد مسیر داده را با حل تمرین یا استاد نهایی کنید. سپس، آنرا پیاده کنید.
5. واحد کنترل را طراحی کنید.
6. واحد کنترل و مسیر داده را به هم وصل کنید.
7. هر دستورالعمل را جداگانه آزمایش کنید.
8. چند برنامه­ی ساده بنویسید.