به نام خدا

دانشگاه صنعتی شریف

درس معماری کامپیوتر

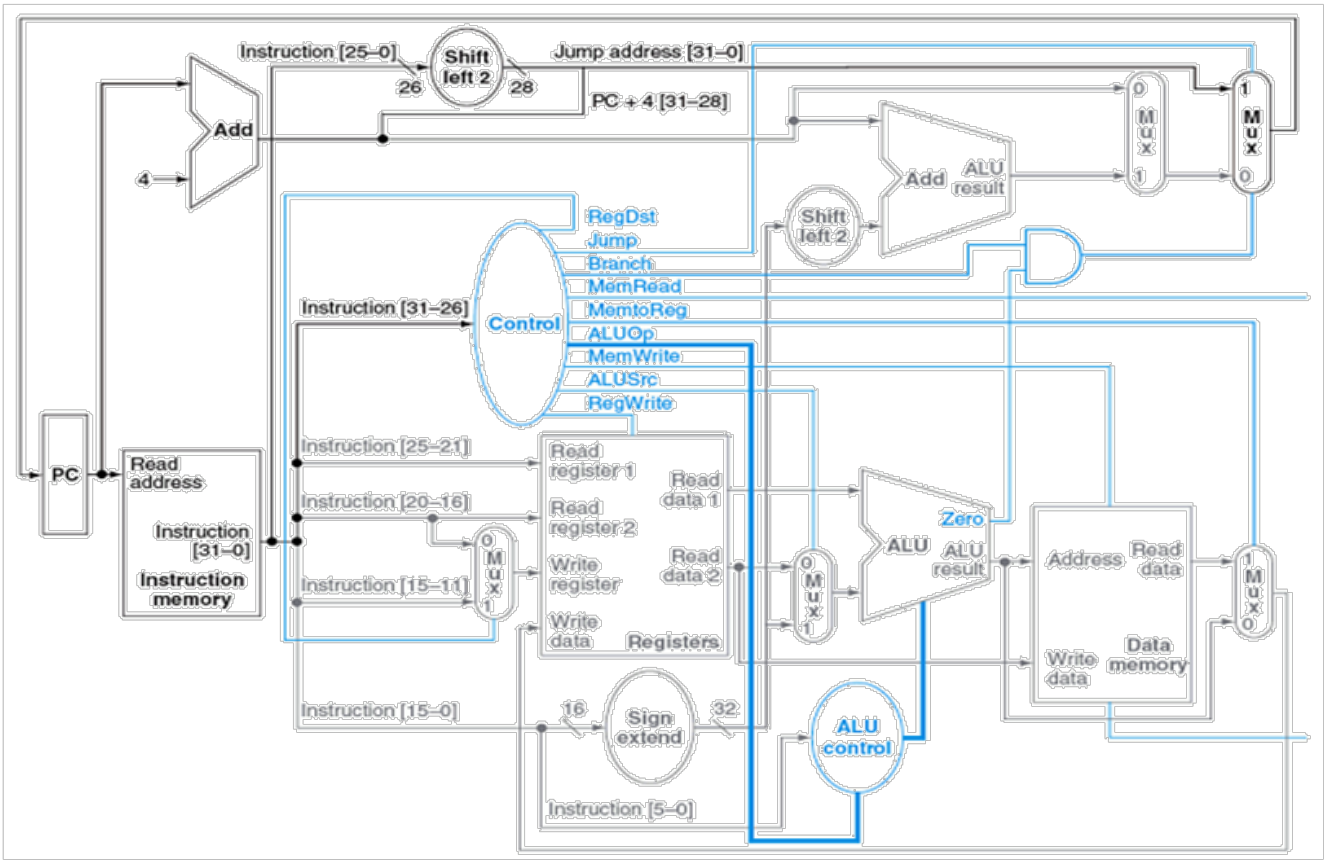
پروژه‌ی میپس

فاز یک

پارسا شریفی سده
99101762
امیرمحمّد فخیمی
99170531
محمّدهومان کشوری
99105667
محمّدجواد ماهرالنّقش
99105691

بهار ۱۴۰۱

- در ابتدا این نکته را بگویم که ما با استفاده از نقشه‌ی زیر پیش رفتیم:



چند نکته درباره‌ی نقشه مهم است:

- خروجی and باید به mux سمت چپ باشد و در نقشه به اشتباه آن را به mux سمت راست متّصل کرده‌است.
- چون در کد چیزی به نام MemRead به ما داده نشده است پس ما نیز این سیگنال را از نقشه حذف کردیم.
- به دلیل نیاز، ما سیگنال halted، reset و clock را به نقشه اضافه کردیم که سیگنال halted با توجّه به سیگنال reset یا دستور syscall تغییر وضعیت می‌دهد.
- ما قسمت ALU control را از نقشه حذف کردیم و به جای آن func را به Control به عنوان ورودی اضافه کردیم همچنین از Control یک ورودی دیگر به ALU اضافه کردیم تا کار ALU control انجام گیرد. در ادامه برای آنکه به Sh.amount دسترسی داشته باشیم، inst را نیز به ALU وارد کردیم.

- **قسمت mips_core:**

در این قسمت ما اتّصالات بین اجزای Registers، ALU، Control و PC را برقرار کردیم و مواردی مانند shift و ADD در این قسمت انجام شدند.

ما به جای استفاده از mux در Verilog از توصیف data flow استفاده کرده‌ایم.

- **قسمت pc:**

```
1   module pc (
2       input clk, rst_b,
3       input [31:0] pc_input,
4       output reg [31:0] pc_output
5   );
6
7   always_latch @(posedge clk or negedge rst_b) begin
8       if(rst_b == 0)
9           pc_output <= 0;
10      else begin
11        pc_output <= pc_input;
12      end
13  end
14
15
16  endmodule
```

این قطعه کد مربوط به PCاست و همانطور که میبینید ابتدا در ترمینال آن ورودی ها و خروجی ها تعیین شده اند و سپس در یک حلقهٔ حساس به لبهٔ مثبت کلاک و لبهٔ منفی rst_b کد مربوط به این بخش پیاده سازی شده است. این بخش برای مقداردهی به صورت non-blocking assignment به pc_output نوشته شده است.

- **قسمت controll:**

قسمت کنترل وظیفهٔ کنترل کردن سایر بخش ها را بر عهده دارد و به نوعی مغز متفکر سیستم و پردازندهٔ ما حساب میشود. ماژولی به نام control تعریف کرده و تعدادی ورودی و خروجی در ترمینال آن قرار می دهیم.

یک ورودی clk به آن می دهیم که همان سیگنال کلاک ما است که در کد همانطور که میبینیم باعث می شود کد ما به لبهٔ مثبت کلاک حساس باشد. هم چنین یک ورودی با پهنای 6 بیت به نام inst داده می شود که همان مخفف instruction یا همان دستور است که در واقع 6 بیت مربوط به آپکد دستور اصلی 32 بیتی است.

هم چنین **یک ورودی دیگر به نام func داده می شود** که برای این است که اگر دستور R-Type بود یعنی آپکد آن 6'b000000 بود باید فانکشن مربوطه به ALU که همان واحد محاسبات است پاس داده می شود.

هم چنین تعدادی سیگنال کنترلی داریم که در ترمینال تحت عنوان خروجی داده شده اند.

توجه کنید که یکی از مشکلاتی که به آن برخوردیم این بود که ورودی یا همان input را به عنوان سیم یا همان وایر یا همان net در کد در نظر نگرفتیم بلکه به صورت reg یا همان متغیر در نظر گرفتیم در صورتی که ورودی نباید به عنوان متغیر در نظر گرفته شود بلکه همان پیش فرض خودش یا همان سیم باید باشد.
هم چنین تعدادی خروجی داریم که همان سیگنال های کنترلی هستند که اصلا اصل این ماژول برای معین کردن همین سیگنال های کنترلی است.
حال به توضیح این سیگنال های کنترلی می پردازیم.

همانطور که در نقشه میبینیم **سیگنال reg_dst به یک مالتی پلکسر دو به یک وصل است** که انتخاب میکند کدام ورودی ها وارد ورودی write register مربوط به Registers شوند.
این دستور مربوط به این است که اگر دستور r-type داشته باشیم آن را یک میکنیم به این معنی که بیت ۱۱ تا بیت ۱۵ دستور یا همان اینستراکشن وارد write register می شود و اگر دستور i-type باشد که همان دستورات immediateاست باید ان را صفر کنیم که یعنی بیت ۱۶ تا بیت ۲۰ از دستور ۳۲ بیتی وارد write register شود.

حال به بررسی سیگنال جامپ یا پرش میپردازیم.
این سیگنال وقتی فعال است که پرشی رخ میدهد. پی کافی است اگر دستور j-type بود آن را یک کنیم و در غیر اینصورت صفر کنیم.

سیگنالbranch :
این سیگنال در ۵ دستور فعال میشود. دستوراتی که شاخه ای هستند. دستورات شاخه ای یعنی چه؟ یعنی یک شرط بررسی می شود و اگر درست بود وارد یک شاخه ی جدیدی میشویم یا بهتر است بگوییم به آن پرش میکنیم. و هم چنین میتوانیم else تعریف کنیم تا شاخه‌ی جدیدی ایجاد شود که از مبحث ما دور می شود.
این سیگنال کنترلی در ۵ دستور بررسی تساوی، کوچکتری، بزرگتری، کوچکتری مساوی و بزرگتری مساوی فعال می شود که توجه کنیم که همگی این ۵ دستور از نوع i-type هستند.

سیگنال بعدی ای که بررسی می کنیم **سیگنال mem_read است** که بعد از پیاده سازی به این نتیجه رسیدیم که سیگنال اضافه و یا اصلاحات trash است و نیازی به آن نیست بنابراین آن را کامنت کردیم. پیش از کامنت کردن این سیگنال هدف ما این بود که اگر قصد داشتیم از موری بخوانیم این سیگنال را فعال کنیم که در ادامه مشخص شد نیازی به این کار نیست.
*توجه: البته در نسخه‌ی نهایی این کامنت به طور کلی پاک شده((:

سیگنالmem_write_en :
این سیگنال وقتی فعال است که میخواهیم در موری یا همان حافظه بنویسیم.
پس در دستوراتی که نیاز به نوشتن در حافظه دارند این سیگنال کنترلی را یک میکنیم.

تنها دستوراتی که نیاز به نوشتن در حافظه دارند دستورات SW و SB هستند که مخفف Store Word و Store Byteهستند.
همانطوری که از اسمشان معلوم است در این دستورها یا میخواهیم ۸ بیت را ذخیره کنیم یا یک کلمه یا همان word را و چون میخواهیم عمل ذخیره یا همان save to memory انجام دهیم نیاز است که این سیگنال کنترلی فعال یا همان یک شود.

سیگنالmem_to_reg :
این سیگنال فقط در دستورات branch یا همان ۵ دستور مقایسه ای بعلاوه‌ی دستورات لود که همان ۲ دستور LW و LB هستند فعال می شود و این سیگنال به مالتی پلکسری وصل است که به ورودی write data در Registers می رود.

سیگنالalu_op :
این سیگنال به واحد محاسبات یا همان ALU می فهماند که چه کار باید انجان دهد. برای این کار کد ۶ بیتی مربوط به کاری که باید انجام دهد را به آن می دهیم.
توجه: در پیاده سازی ای که ما انجام دادیم، اگر R-Type باشد به alu_op کد فانکشن مربوطه را دادیم و اگر i typeباشد همان کاری که باید انجام دهد پاس داده شده.

سیگنالalu_src :
سیگنال ALU Source در دستورات ایمیدیت و لحظه ای فعال است.
توجه کنید منظور دستوراتی است که آخرشان i که مخفف immediate هست دارند.
مثلا دستور beq دستور ایمدیت حساب نمیشود در اینجا) وگرنه که دستور i-type هست.(
دستورات lw و lb و sw و sb هم ایمدیت اینجا هستند و این سیگنال برایشان فعال است.

سیگنالreg_write :
این سیگنال همیشه به جز در دستورات برنچ یعنی beq و ...و دستورات ذخیره در حافظه یعنی sw و sb و همچنین SRLو JAL فعال است (به جز اینها).

سیگنالhalted :
این سیگنال برای وقتی است که syscalk صدا زده شده باشد، توجه کنید که این سیگنال قراردادی در پیاده سازی مابین اعضای تیم و برای سادگی پیاده سازی بود.

- **قسمت alu:**

قسمت ALU واحد محاسبات ما است و در واقع اگر کنترلر مغز متفکر باشد، این مانند بازوی ما است که کارهایی که کنترلر دستور داده را اجرا میکند. مثلا شیفت دادن، جمع کردن و .. همگی در اینجا اجرا می شوند. همچنین از این قسمت برای دستورات پرشی و ذخیره و لود کردن داده از مموری استفاده کرد.

- **نکات نهایی:**

1. توجه کنید که در دو ماژول ALU و control برای راحتی در خواندن و خوانایی و فهم بهتر کد یک سری دایرکتیو یا همان directive تعریف شده اند. اینها باعث می شوند هم خوانایی زیاد شود و هم از نوشتن اعداد طولانی بی معنی جلوگیری شود.
2. یک چالش پیش از این توضیح داده شد و اکنون به یک چالش دیگر در دیباگ کد میپردازیم. ابتدا در خط 41 در یکی از ماژول ها از عمل کانکتنیشن یا همان concat استفاده کردیم ولی در تست br_0 به مشکلی برخوردیم و آن این بود که وقتی می خواستیم داده را در دیتا بنویسیم، داده صفر بود چراکه read_data_2 را با وجود اینکه آپدیت میکردیم ولی mem_data_in همان صفری که ابتدا بوده باقی می ماند اما در زبان وریلاگ عمل ما باعث آپدیت mem_data نشد. این مشکلات در ماژول mips_core رخ داد.