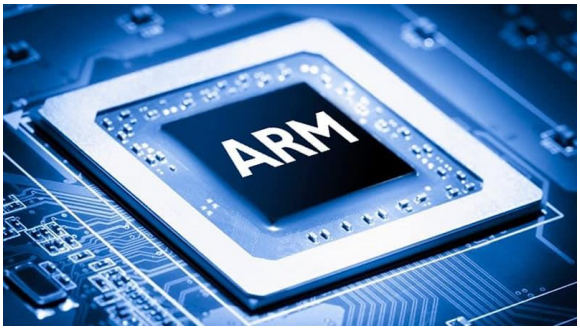
به نام خدا

دانشگاه تهران
پردیس دانشکده‌های فنی
دانشکده برق و کامپیوتر



آزمایشگاه معماری کامپیوتر

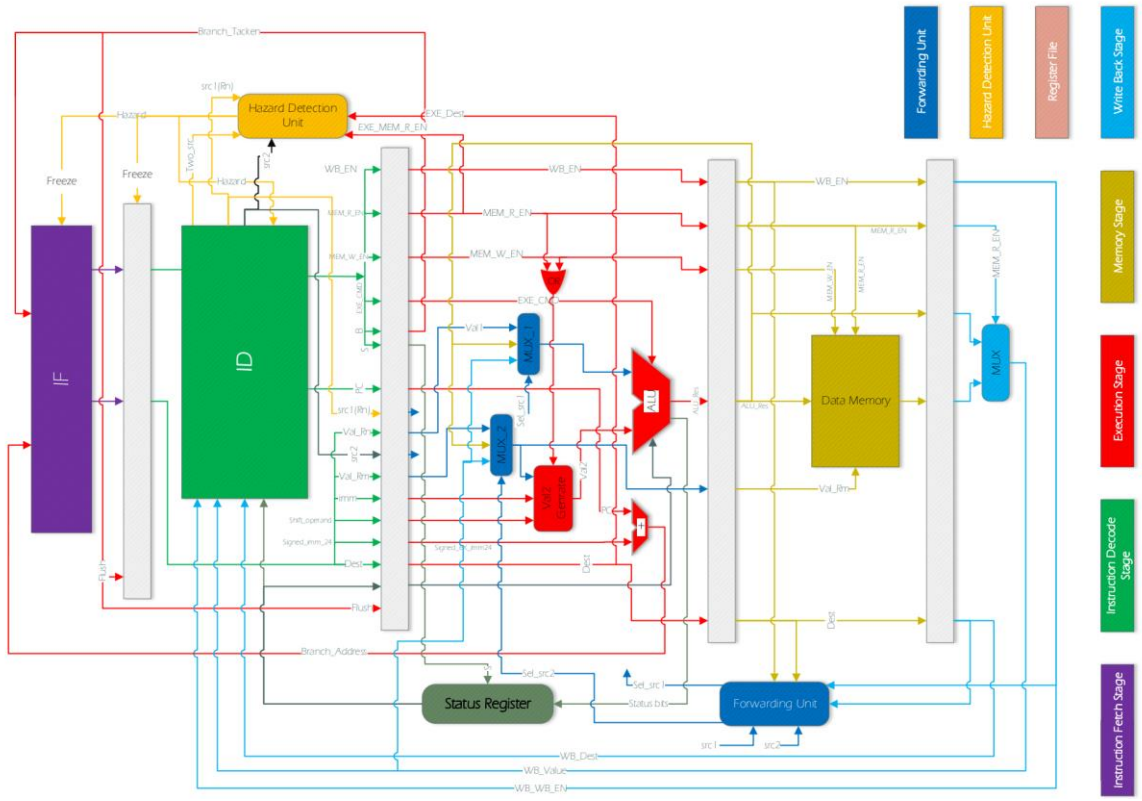
گزارش دستور کار شماره ۵

علی پادیاو
۸۱۰۱۹۹۳۸۸
محمد صالح عرفاتی
۸۱۰۱۹۷۵۴۳

اردیبهشت ۱۴۰۲

مقدمه

در آزمایشات قبلی مراحل IF و ID و EXE و WB و MEM و ماژول Hazard از معماری ARM9 پیاده سازی شد. حال در این آزمایش به پیاده سازی Forwarding Unit پرداختیم. بلوک دیاگرام این مرحله به صورت زیر می‌باشد:



شکل ۱

پس از پیاده سازی کامل معماری ARM و ماژول Hazard مشاهده کردیم که میانویم چندین Stall موقع اجرای دستورات داشته باشیم و این میتواتند عملکرد پردازنده را تحت تاثیر قرار دهد به همین دلیل در صدد برآمدیم تا این Stallها را تا جایی که میشود انجام نداد.

به همین دلیل از ماژول ForwardingUnit استفاده کردیم. این ماژول با تا جایی که میتواند دستورات را رو به جلو میفرستد و تنها زمانی Stall خواهیمن داشت که مجبور باشیم.

زمانی که دستور جدید به دستورات قبل که می‌تواند یک دستور و یا دو دستور قبل باشد (زیرا دستورات قبل تر به طور کامل اجرا شده اند)، نیاز داشته باشد، این دستورات خروجی مربوطه را محاسبه کرده اند و تنها در مقصد موردنظر ننوشته اند. به جز زمانی که دستور قبل Load باشد

تنها در این حالت است که نیاز به Stall داریم زیرا باید مموری موردنظر خوانده شود که هنوز آماده نیست.

پس تنها در حالتی که دستور Load باشد Stall داریم و در بقیه حالات میتوانیم از نتیجه دستورات قبل در صورت نیاز استفاده کنیم. با همین روش ساده میتوانیم بهبود عملکردی را در پردازندمان شاهد باشیم.

شرح ماژول های اضافه شده:

1-Forwarding

کد:

```
module ForwardingUnit (input forward_en,
                       input [3:0] src1, src2,
                       input [3:0] WB_dest, MEM_dest,
                       input WB_WB_en, MEM_WB_en,
                       output reg [1:0] sel_src1, sel_src2
                      );
  always @(forward_en, src1, src2, MEM_WB_en, MEM_dest, WB_WB_en, WB_dest)
  begin
    sel_src1 = 2'b00;
    if (forward_en)
    begin
      if (MEM_WB_en && (src1 == MEM_dest))
      begin
        sel_src1 = 2'b01;
      end
      else if (WB_WB_en && (src1 == WB_dest))
      begin
        sel_src1 = 2'b10;
      end
      else
      begin
        sel_src1 = 2'b00;
      end
    end
  end

  always @(forward_en, src1, src2, MEM_WB_en, MEM_dest, WB_WB_en, WB_dest)
  begin
    sel_src2 = 2'b00;
    if (forward_en)
    begin
      if (MEM_WB_en && (src2 == MEM_dest))
      begin
        sel_src2 = 2'b01;
      end
      else if (WB_WB_en && (src2 == WB_dest))
      begin
        sel_src2 = 2'b10;
      end
      else
      begin
        sel_src2 = 2'b00;
      end
    end
  end
```

شکل۲

این ماژول وظیفه مدیریت forwarding را در اختیار دارد. به همین منظور باید selector های مولتی پلکسر های Mux1 و Mux2 را در شکل۱ تعیین کند.

تعیین selector Mux1:

always اول مربوط به تعیین این ماکس می‌باشد. دراین ماژول ابتدا بررسی میگردد که آیا عملکرد forwarding فعال می‌باشد یاخیر(تا بتوانیم عملکرد آن را با حالتی که forwarding نداریم مقایسه کنیم).

- در صورت فعال بودن بررسی میگردد که آیا آدرس رجیستر src1 با آدرسی که میخواهد در دستور قبل در registerFile بنویسد برابر است یا خیر. اگر برابر بود یعنی وابستگی دستور به یک دستور قبل را داریم. پس باید مقدار محاسبه شده برای دستور قبل که هم اکنون درمرحله MEM می‌باشد را برای خروجی MUX1 انتخاب کنیم.
- اگر حالت قبل رخ نداد این حالت مورد بررسی قرار میگیرد. در این حالت بررسی میگردد که آیا آدرس رجیستر src1 با آدرسی که میخواهد در دو دستور قبل در registerFile بنویسد برابر است یا خیر. اگر برابر بود یعنی وابستگی دستور به دو دستور قبل را داریم. پس باید مقدار محاسبه شده برای دو دستور قبل که هم اکنون درمرحله WB می‌باشد را برای خروجی MUX1 انتخاب کنیم.

دقت شود که دو حالت بالا به ترتیب می‌باشند و حالت اول اولویت بیشتری دارد زیرا آخرین دستوری می‌باشد که قبل از دستور مورد نظر اجرا شده است.

تعیین selector Mux2:

always دوم مربوط به تعیین این ماکس می‌باشد. در این ماژول ابتدا بررسی میگردد که آیا عملکرد forwarding فعال می‌باشد یاخیر(تا بتوانیم عملکرد آن را با حالتی که forwarding نداریم مقایسه کنیم).

- در صورت فعال بودن بررسی میگردد که آیا آدرس رجیستر src2 با آدرسی که میخواهد در دستور قبل در RegisterFile بنویسد برابر است یا خیر. اگر برابر بود یعنی وابستگی دستور به یک دستور قبل را داریم. پس باید مقدار محاسبه شده برای دستور قبل که هم اکنون درمرحله MEM می‌باشد را برای خروجی MUX2 انتخاب کنیم.

- اگر حالت قبل رخ نداد این حالت مورد بررسی قرار میگیرد. در این حالت بررسی میگردد که آیا آدرس رجیستر src2 با آدرسی که میخواهد در دو دستور قبل در RegisterFile بنویسد برابر است یا خیر. اگر برابر بود یعنی وابستگی دستور به دو دستور قبل را داریم. پس باید مقدار محاسبه شده برای دو دستور قبل که هم اکنون درمرحله WB می‌باشد را برای خروجی MUX2 انتخاب کنیم.

دقت شود که دو حالت بالا به ترتیب می‌باشند و حالت اول اولویت بیشتری دارد زیرا آخرین دستوری می‌باشد که قبل از دستور مورد نظر اجرا شده است.

دقت شود که این دو برای تعیین src1 و src2 می‌باتشند همچنان ماژول val2generate سرجایش برقرار می‌باشد تا value دوم را برای ALU انتخاب کند و عملکرد آن مجزای این ماژول می‌باشد.

2- Mux1,Mux2:

این دو ماژول وظیفه تعیین مقدار src1 و src2 خوانده شده از RegisterFile رادارند. زمانی که Forwarding تشخیص داده شد باید النتخاب شود که کدام یک به عنوان src1 و src2 از بین حالات زیر انتخای شود:

- مقدار خوانده شده از خود RegisterFile
- مقداری که دستور قبل میخواهد در RegisterFile بنویسد.
- مقداری که دودستور قبل میخواهد در RegisterFile بنویسد.

دقت شود که selector های این دو ماکس همانطور که گفته شده در ForwardingUnit انتخاب می‌شود.
این دوماژول با استفاده از یک assign ساده مقدار خروجی ماکس را تعیین میکنند:
کد:

```
22  assign ALU_src_1 = (sel_src1 == 2'b00) ? Val_Rn :
23          (sel_src1 == 2'b01) ? ALU_res_f:
24          (sel_src1 == 2'b10) ? WB_val_f:
25          Val_Rn;
26
27  assign Val2_src = (sel_src2 == 2'b00) ? Val_Rm_in :
28          (sel_src2 == 2'b01) ? ALU_res_f:
29          (sel_src2 == 2'b10) ? WB_val_f:
30          Val_Rm_in;
```

شکل ۳

3-Hazard_Unit:

کد:

```
module HazardDetector (input forward_en,
                       input [3:0] src1,
                       input [3:0] src2,
                       input [3:0] Exe_Dest,
                       input Exe_WB_EN,
                       input [3:0] Mem_Dest,
                       input Mem_WB_EN,
                       input Two_src,
                       input use_src1,
                       input Exe_Mem_R_EN,
                       output hazard_Detected
                      );

  assign hazard_Detected = !forward_en && ((Exe_WB_EN && (use_src1 && src1 == Exe_Dest)) ||
         (Exe_WB_EN && (Two_src  && src2 == Exe_Dest)) ||
         (Mem_WB_EN && (use_src1 && src1 == Mem_Dest)) ||
         (Mem_WB_EN && (Two_src  && src2 == Mem_Dest))) || Exe_Mem_R_EN && (src1 == Exe_Dest || src2 == Exe_Dest);

endmodule
```

شکل ۴

این ماژول وظیفه شناسایی و تولید وقفه برای اجرای درست دستورات را دارا می‌باشد.

- هنگامی‌که عملکرد Forwarding را فعال باشد:
  - در این حالت فقط در یک حالت Stall رخ میدهد که دستور قبل Load باشد.
- هنگامی‌که عملکرد Forwarding را فعال نباشد:

در چهار حالت زیر وقفه در روند اجرای دستورات در پایپ لاین رخ میدهد:

- هنگاهی که سیگنال WB_EN در مرحله EXE فعال باشد و میخواهد در رجیستری بنویسد و آدرس رجیستر Rn که در مرحله ID میخواهد خوانده شود برابر با آدرس رجیستری باشد که مرحله EXE میخواهد در دو کلاک بعد آن را آپدیت کند (به عبارت دستوری که در مرحله EXE است مقدار رجیستری را میخواهد تغییر دهد که دستور مرحله ID میخواهد از آن استفاده کند).
- هنگاهی که سیگنال WB_EN در مرحله EXE فعال باشد و میخواهد در رجیستری بنویسد و آدرس رجیستر Rm که در مرحله ID میخواهد خوانده شود برابر با آدرس رجیستری باشد که مرحله EXE میخواهد در دو کلاک بعد آن را آپدیت کند(به عبارت دستوری که در مرحله EXE است مقدار رجیستری را میخواهد تغییر دهد که دستور مرحله ID میخواهد از آن استفاده کند).
- هنگامی‌که سیگنال WB_EN در مرحله ممووری یک باشد و میخواهد در رجیستری بنویسد و آدرس رجیستر Rn که در مرحله ID میخواهد خوانده شود برابر با آدرس

رجیستری باشد که مرحله MEM میخواهد در کلاک بعد آن را آپدیت کند(به عبارت دستوری که در مرحله MEM است مقدار رجیستری را میخواهد تغییر دهد که دستور مرحله ID میخواهد از آن استفاده کند).

- هنگامی‌که سیگنال WB_EN در مرحله ممووری یک باشد و میخواهد در رجیستری بنویسد و آدرس رجیستر Rm که در مرحله ID میخواهد خوانده شود برابر با آدرس رجیستری باشد که مرحله MEM میخواهد در کلاک بعد آن را آپدیت کند(به عبارت دستوری که در مرحله MEM است مقدار رجیستری را میخواهد تغییر دهد که دستور مرحله ID میخواهد از آن استفاده کند).

در هر چهار حالت بالا وقفه رخ داده تا مطمئن شویم دیتای درستی از RegisterFile خوانده شود.

- قابل ذکر است که اگر دستور پرش نیز باشد باید سیگنال freeze یک شود تا بدون دلیل دستورات بعد از پرش اجرا نگردند.

در انتها قابل ذکر است بقیه ماژول‌ها نیز تغییر داشته اند ولی صرفا در حد متصل کردن و خروجی قرار دادن بعضی از سیگنال‌ها می‌باشد و عملکرد کلی آن‌ها عوض نشده است و به همین دلیل به تغییرات جزئی آن‌ها اشاره نشده است.

دستورات برنامه:

طبق شرح آزمایش کل برنامه محک را داخل Ins_Mem ذخیره کردیم تا دستورات یکی پس از دیگری اجرا گردند دقت شود که وظیفه این برنامه مرتب سازی رجیسترها R1,R2,R3,R4 به صورت صعودی می‌باشد.

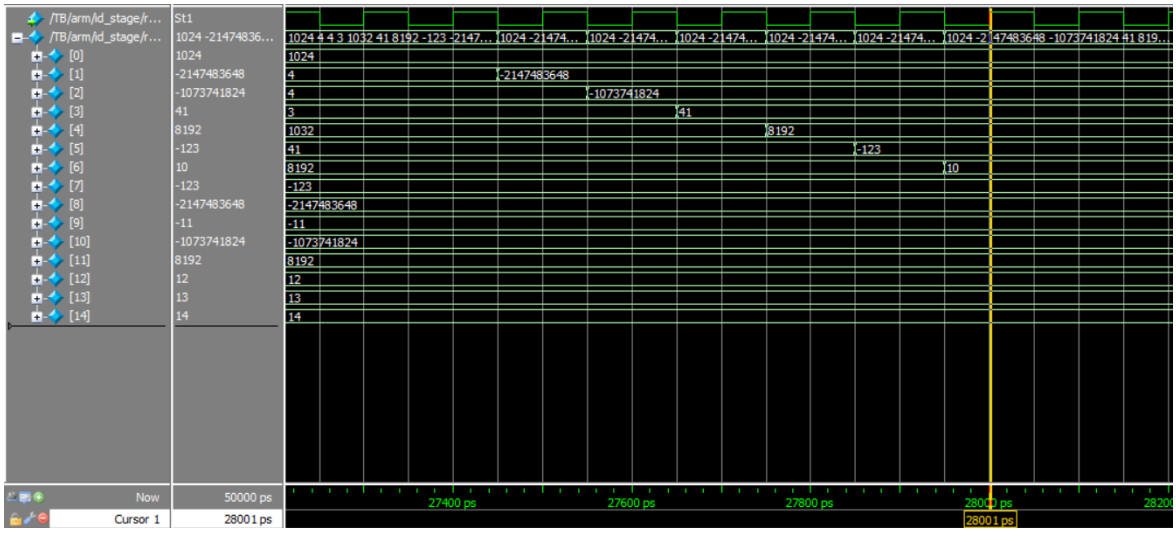
SIMULATION:

برای simulation از تست بنچ زیر استفاده شده است:

```
module TB_DE2 ();

  reg clk, rst, forward_en;

  ARM arm (
        .clk(clk),
        .rst(rst),
        .forward_en(forward_en)
      );

  initial
  begin
    clk = 1;
    forward_en = 1;
    repeat (600)
    begin
      #50;
      clk = ~clk;
    end
  end

  initial
  begin
    rst = 0;
    #20 rst = 1;
    #10 rst = 0;
  end

endmodule
```

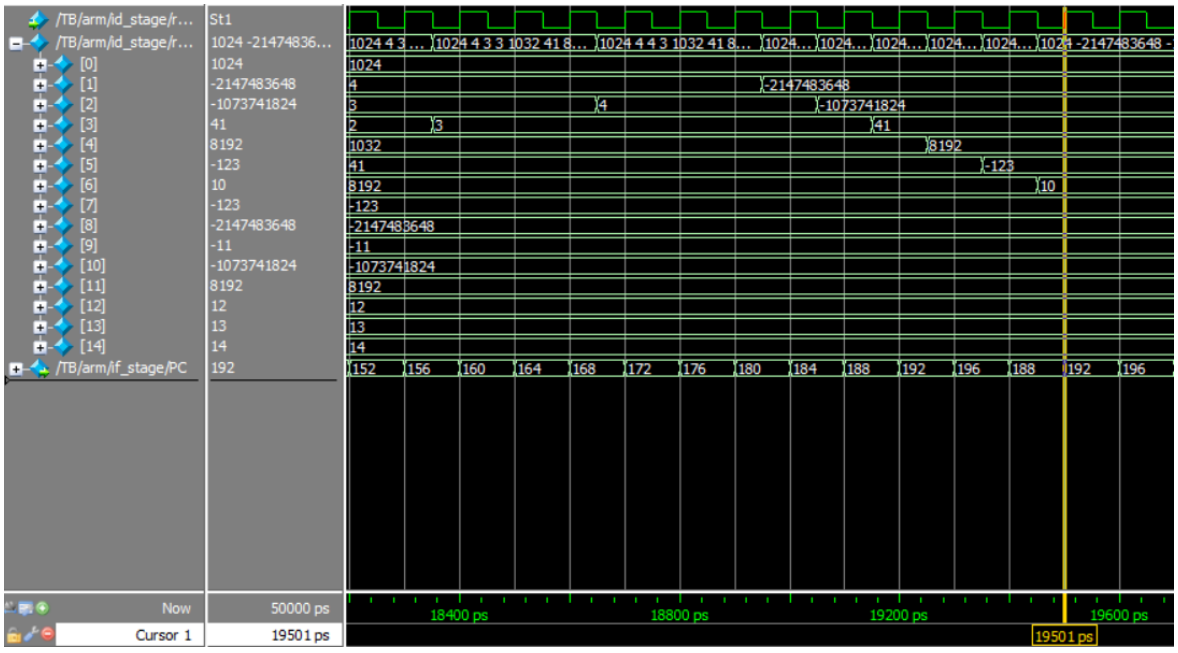
شکل ۵

خروجی بدون فورواردینگ:



شکل ۶

خروجی به همراه فورواردینگ:



شکل ۷

همانطور که مشاهده میشود تعداد سیکل‌ها نسبت به حالت بدون forwarding ۸۵ سیکل کمتر شده. (حدود ۳۰ درصد). یعنی در این دستورات حدود ۳۰ دصد بهبود عملکرد داشتیم.

جواب نهایی SYNTHESIZER:

پس از موفقیت آمیز بودن جواب‌ها در مرحله شبیه سازی پیاده سازی آن برروی fpga پرداختیم و با استفاده از SIGNAL TAB مقادیر ۴ رجیستر مورد نظر را مشاهده کردیم:

- حالت بدون forwarding:

شکل ۸

همانطور که در تصویر نیز قابل مشاهده است ۴ رجیستر مربوطه مرتب‌سازی شده اند.

- حالت به همراه forwarding:

شکل ۹

همانطور که در تصویر نیز قابل مشاهده است ۴ رجیستر مربوطه مرتب سازی شده اند و شاهد ۸۵ سیکل کمتر و ۳۰ درصد عملکرد بهتر برای forwarding هستیم.