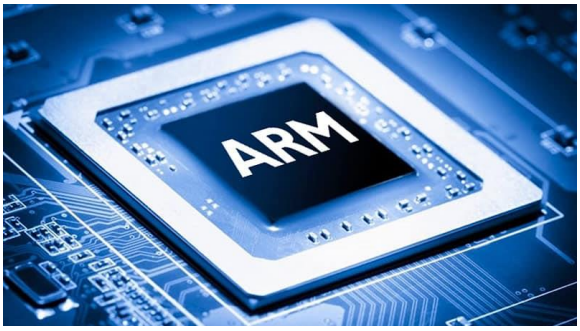
به نام خدا

دانشگاه تهران
پردیس دانشکده‌های فنی
دانشکده برق و کامپیوتر



آزمایشگاه معماری کامپیوتر

گزارش دستور کار شماره 7

علی پادیاو
810199388
محمد صالح عرفاتی
810197543

بهار1402

مقدمه

در آزمایشات قبلی مراحل IF و ID و EXE و WB و MEM و ماژول Hazard از معماری ARM9 پیاده سازی شد و در انتها ماژول Forwarding Unit و SRAM به معماری ARM خود اضافه نمودیم.

در جلسه قبل مشاهده کردیم که در دنیای FPGA حافظه و مموری خیلی ارزشمند است و کمیاب می باشد و به همین دلیل از مموری RAM جداگانه استفاده می کنیم. در معماری ARM ما ماژول memory کاملا از منابع داخلی FPGA استفاده شده بود به همین دلیل این ماژول کامل جداکرده تا در بورد مربوطه از SRAM موجود در دولوپمنت بورد مربوطه برای مموری استفاده شد.

سپس پس از راه اندازی ماژول SRAM مشاهده کردیم که عملیات های خواندن و نوشتن در این حافظه بسیار هزینه برمی باشد و ئردازنده باید چندین کلاک متوقف شده تا این عملیات صورت بگیرد، به همین دلیل سعی بر آن داشتیم تا از حافظه ای استفاده کنیم که عملیات های مربوطه زمان کمتری را مصرف کنند. به همین منظور در معماری ARM خود از CACHE استفاده می کنیم. حافظه CACHE یک حافظه بسیار سریع است که در پردازنده ما می باشد. بنابراین این نوع حافظه قابلیت دسترسی در یک کلاک را دارا می باشد. پس اگر به جاي دسترسی به حافظه اصلی به حافظه CACHE مراجعه شود، آنگاه نیاز به متوقف کردن پردازنده براي انجام علمیات حافظه نیست و کارایی پردازنده افزایش چشمگیري خواهد داشت. در این آزمایش از حافظه روي تراشه FPGA به عنوان حافظه CACHE استفاده می شود.

برای پیاده سازی این خافظه از معماری tow- way associates استفاده شده است و همچنین از الگوریتم جایگزینی LRU برای جایگزینی کلمات جدید استفاده میشود.

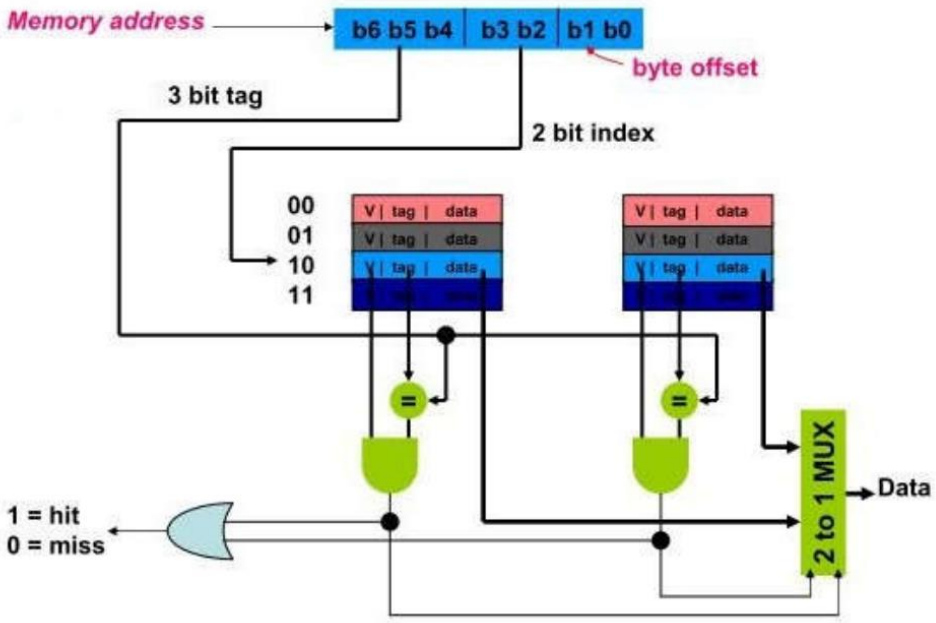
مشخصات CACHE پیاده سازی شده:

- معماری tow- way associates : به این معنی است که هر خط حافظه‌ی کش دارای دو مسیر برای ذخیره سازی داده‌ها است. به عبارت دیگر، حافظه کش به چندین قسمت تقسیم شده و هر قسمت دارای دو مکان برای ذخیره‌سازی داده‌ها است.
- اندازه هر کلمه: 32 بیت : اندازه هر کلمه از کش 32 بیتی می باشد بر خلاف SRAM که 16 بیتی بود.
- اندازه هر بلاك: 64 بیت (2 کلمه) : در هر بلاک از CACHE 64 بیت فضای قابل استفاده داریم.

- تعداد مجموعه ها (set) : 64
- گذرگاه آدرس: 19 بیت
- تعداد بیت مورد نیاز براي نشانه و یا tag : 10 بیت
- تعداد بیت شاخص و یا index : 6 بیت
- داراي بیت اعتبار و یا valid : 1 بیت

به طور خلاصه CACHE استفاده شده به صورت 2-way set associates می‌باشد؛ که ۶۴ set دارد و اندازه هر بلاک نیز ۶۴ بیت (دو کلمه) می‌باشد.
با توجه به ساختار حافظه نهان و آدرس ورودی، می‌توان گفت ۳ بیت کم ارزش نشان‌دهنده offset، ۶ بیت بعدی نشان‌دهنده index و ۱۰ بیت دیگر نشان دهنده تگ می‌باشند. پس بلوک در هر way در حافظه نهان شامل ۷۵ بیت می‌باشد. ۶۴ بیت برای داده موجود، ۱۰ بیت برای tag و ۱ بیت هم به عنوان valid bit ؛ که این بیت در ابتدا برابر صفر قرار داده می‌شود تا نشان دهد داده موجود داده valid ای نمی‌باشد و اگر دیتا نیز بر روی RAM نوشته شده و برروی CACHE نیزنباشد بیت valid برابر صفر قرار داده میشود.

نحوه عملکرد ماژول CACHE:



شکل 1

هنگامی که آدرس موردنظر برای خواندن و یا نوشتن می آید، ابتدا در هر دو ست از این حافظه در سطر index این حافظه 6 بیت تگ آدرس با 6 بیت تگ خانه موردنظر چک می شوند و اگر در هر کدام از ست ها برابر باشند و دیتای موردنظر نیز قابل اعتبار باشد در آن صورت آدرس موردنظر پیدا شده و HIT می شود در غیر این صورت MISS اتفاق می افتد.

شرح ماژول های اضافه شده:

1- CacheController

کد:

در کد زیر ورودی‌ها و خروجی‌های کنترلر حافظه نهان را مشاهده می‌کنیم. سیگنال ready نشان می‌دهد که مقدار درخواستی در cache وجود داشته یا خیر.

```
1   module cache_controller (
2       input clk, rst,
3       input rd_en, wr_en, sram_ready,
4       input [31:0] address, wdata,
5       input [63:0] sram_rdata,
6
7       output ready,
8       output [31:0] sram_address, sram_wdata, rdata,
9       output sram_write, sram_read
10    );
```

شکل 2

در قطعه کد زیر آدرس و تگ و شناسه مربوطه از آدرس مورد نظر برداشته میشود.

```
16    wire [2:0] offset;
17    wire [5:0] index;
18    wire [9:0] tag;
19    assign offset = address[2:0];
20    assign index = address[8:3];
21    assign tag = address[18:9];
```

شکل 3

شکل زیر رجیستر های مربوطه به داده ها(32 بیتی می باشند) و بیت valid که یک بیت می باشد 10 بیت تگ برای هر دیتا تعریف شده اند همچین بیت LRU نیز برای هر ست نیز تعریف شده که با استفاده از آن بتوانیم مطمئن شویم دیتای نوشته شده را در CACHE باقس می ماند.

```
24    reg [31:0] way0_data0 [63:0];
25    reg [31:0] way0_data1 [63:0];
26    reg [31:0] way1_data0 [63:0];
27    reg [31:0] way1_data1 [63:0];
28
29    reg way0_valid [63:0];
30    reg way1_valid [63:0];
31
32    reg [9:0] way0_tag [63:0];
33    reg [9:0] way1_tag [63:0];
34
35    reg LRU [63:0];
```

شکل 4

در شکل 5 تگ شناسه مئرد نظر در حافظه cache با تگ آدرس فرستاده شده چک میگردد و در صورت برابر بودن و برتبر بودن بیت valid برای آن دیتای مورد نظر سیگنال hit برابر با یک میشود به منظئر نشان دادن ئیدا شدن دیتای موردنظر.

```
41    assign way0_hit = (way0_tag[index] == tag) & (way0_valid[index] == 1);
42    assign way1_hit = (way1_tag[index] == tag) & (way1_valid[index] == 1);
43    assign hit = way0_hit | way1_hit;
44
45    assign sram_read = rd_en & ~hit;
```

شکل 5

در صورت صفر بودن سیگنال hit یعنی دیتای مورد نظر یافت نشده و یا قابل اعتبار نمی باشد به همین منظور در SRAM ئنبال آن میگردیم.

```
45    assign sram_read = rd_en & ~hit;
```

شکل 6

در شکل 7 همانظورک همشاده میکنیم در صورت وجود سیگنال reset بیت های LRU برای هر ست برابر با صفر می شود.

در صورتی که در خواست خواندن آمده باشد و دیتای مورد نظر hit شده باشد آن گاه بیت LRU مورد نظر toggle می شود (از صفر به یک و از یک به صفر تغییر میکند).

و در صورت hit نشدن هنگامن در خواست خواندن از cache در خواست مورد نظر برای sram controller ارسال میگردد.

```
48  integer i;
49  always @(posedge clk, posedge rst)
50  begin
51    if (rst)
52    begin
53      for (i = 0; i < 64; i = i + 1)
54      begin
55        LRU[i] = 0;
56      end
57    end
58
59    else if (rd_en & way0_hit)
60    begin
61      LRU[index] = 0;
62    end
63
64    else if (rd_en & way1_hit)
65    begin
66      LRU[index] = 1;
67    end
68
69    else if (~hit & sram_ready)
70    begin
71      if ((way0_valid[index] == 1) & LRU[index] == 1)
72      begin
73        LRU[index] = 0;
74      end
75
76      else if ((way1_valid[index] == 1) & LRU[index] == 0)
77      begin
78        LRU[index] = 1;
79      end
80    end
81  end
```

شکل 7

قطعه کد زیر نیز برای گرفتن دیتای یافت شده set ای است که سیگنال hit آن یک شده است و به غبارتی دیگر دیتای مورد نظر از آن خانه خوانده میشود.

```
84  wire [31:0] rdata_temp;
85  assign rdata_temp = hit ?
86          (
87            way0_hit ? (offset[2] == 1'b1) ? way0_data1[index] : way0_data0[index] :
88            way1_hit ? (offset[2] == 1'b1) ? way1_data1[index] : way1_data0[index] : 32'bz
89          ) : sram_ready ?
90          (
91            offset[2] == 1'b1 ? sram_rdata[63:32] : sram_rdata[31:0]
92          ) : 32'bz;
```

شکل 8

```
always @(posedge clk, posedge rst)
begin
  if (rst)
  begin
    for (i = 0; i < 64; i = i + 1)
    begin
      way0_valid[i] <= 0;
      way1_valid[i] <= 0;
    end
  end

  else if (rd_en & ~hit & sram_ready)
  begin
    if (LRU[index] == 0)
    begin
      way0_valid[index] <= 1;
      way0_tag[index] <= tag;
      way0_data0[index] <= sram_rdata[31:0];
      way0_data1[index] <= sram_rdata[63:32];
    end

    else if (LRU[index] == 1)
    begin
      way1_valid[index] <= 1;
      way1_tag[index] <= tag;
      way1_data0[index] <= sram_rdata[31:0];
      way1_data1[index] <= sram_rdata[63:32];
    end
  end

  else if (wr_en & hit)
  begin
    if (way0_hit)
    begin
      way0_valid[index] <= 0;
      // LRU[index] <= 1;
    end

    else if (way1_hit)
    begin
      way1_valid[index] <= 0;
      // LRU[index] <= 0;
    end
  end
```

شکل9

در کد شکل 9 ابتدا در صورت زده شدن سیگنال reset بیت valid همه خانه ها برابر با صفر میشود (زیرا داده ها valid نیستند). سپس بررسی میگرد که ایا دستور ما از نوع خواندن اسن و دیتای مورد نظر hit نشده است پس شروع به خواندن از SRAM میکند. در انتها نیز اگر دستور نوشتن آمده بود ذیتای موردنظر در حافظه مورد نظر نوشته میشود.

کد زیر، نحوه قرار گرفتن کنترلر حافظه نهان و حافظه اصلی در کنار هم را نشان می‌دهد که در بخش mem در کنار هم قرار دارند.

```
SramController sram_controller (
                .clk(clk),
                .rst(rst),
                .wr_en(MEM_W_EN),
                .rd_en(MEM_R_EN),
                .address(sram_address),
                .writeData(sram_wdata),

                .readData(sram_rdata),
                .ready(sram_ready),
                .SRAM_DQ(SRAM_DQ),
                .SRAM_ADDR(SRAM_ADDR),
                .SRAM_WE_N(SRAM_WE_N),
                .SRAM_UB_N(SRAM_UB_N),
                .SRAM_LB_N(SRAM_LB_N),
                .SRAM_CE_N(SRAM_CE_N),
                .SRAM_OE_N(SRAM_OE_N)
              );

cache_controller cache_controller (
                  .clk(clk),
                  .rst(rst),
                  .rd_en(MEM_R_EN),
                  .wr_en(MEM_W_EN),
                  .sram_ready(sram_ready),
                  .address(ALU_res),
                  .wdata(ST_val),
                  .sram_rdata(sram_rdata),

                  .ready(ready_wire),
                  .sram_address(sram_address),
                  .sram_wdata(sram_wdata),
                  .rdata(mem_out),
                  .sram_write(sram_write),
                  .sram_read(sram_read)
);
endmodule

```

شکل 10

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 رجیستر ها R1,R2,R3,R4 به صورت صعودی می باشد.

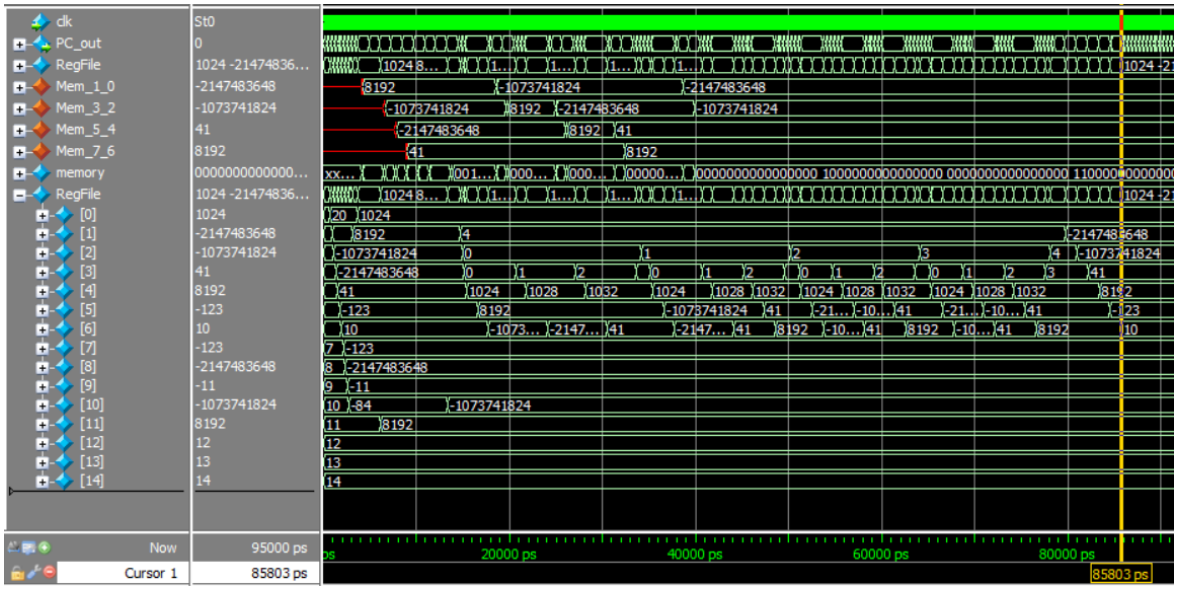
SIMULATION:

برای simulation از تست بنچ زیر استفاده شده است:

```
module TB_DE2 ();

  reg clk, rst, forward_en;

  System system (
        .clock(clk),
        .rst(rst),
        .forward_en(forward_en)
      );

  initial
  begin
    clk = 1;
    forward_en = 1;
    repeat (1900)
    begin
      #50;
      clk = ~clk;
    end
  end

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

endmodule
```

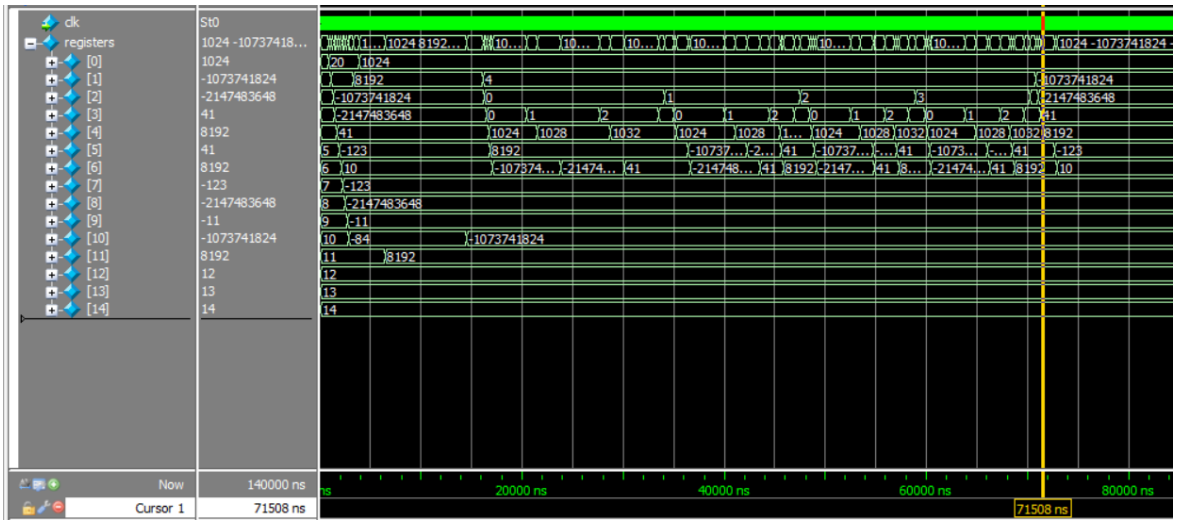
شکل 11

خروجی بدون CACHE و تنها استفاده از SRAM:



شکل 12

خروجی به همراه CACHE و SRAM:



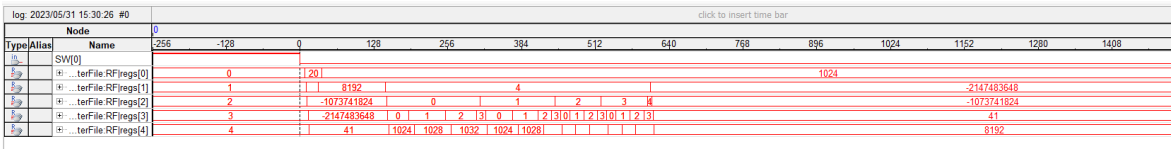
شکل 13

همانطور که می بینید در انتهای هر دو تست این 4 رجیستر سورت شده می باشند.

بهبود کارایی در حالت استفاده از حافظه cache تقریبا برابر با 15 درصد می باشد.

SYNTHESIZE:

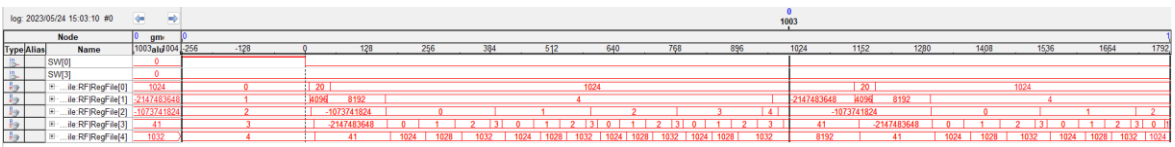
خروجی سیگنال تب مورد نظر در حالت استفاده از CACHE:



شکل 14

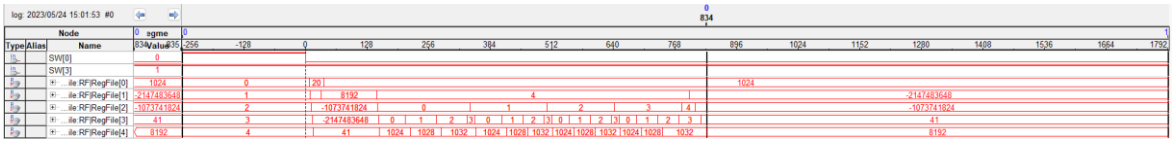
در گزارش قسمت قبل خروجی سنتز قسمت SRAM به دلیل تاخیر در گزارش نوشته نشده بود که با صحبت با خانوم رستگار قرار شد در این گزارش اشاره شود:

خروجی سیگنال تب در حالت استفاده از SRAM بدون استفاده از فورواردینگ:



شکل 15

خروجی سیگنال تب در حالت استفاده از SRAM و با استفاده از فورواردینگ:



شکل 16