سوال ۷ میان‌ترم طراحی سیستم‌های دیجیتال
حسین خوانساری — ۴۰۱۱۰۵۸۸۳

در این سوال قصد داریم یک پردازنده آرایه‌ای ۵۱۲ بیتی طراحی کنیم و به رجیسترفایل با قابلیت ذخیره ۴ آرایه ۵۱۲ بیتی، یک واحد محاسبات ریاضی با قابلیت ضرب و جمع و یک حافظه به عمق ۵۱۲ و عرض ۳۲ بیت نیاز داریم.

کد ماژول رجیسترفایل در فایل RF.v موجود است:

```v
module RF (
  input clk, re, we, dw, // dw: double write used in writing R3-R4 after ALU operation
  input [1:0] read_addr1, read_addr2, write_addr,
  input [1023:0] write_data, // write_data[511:0] is used for single writes
  output [511:0] regout1, regout2,
  output [512*4-1:0] output_test // this output is only for testing
);

  reg [511:0] A[0:3];

  assign regout1 = re ? A[read_addr1] : 'bz;
  assign regout2 = re ? A[read_addr2] : 'bz;

  assign output_test = {A[0], A[1], A[2], A[3]};

  always @(negedge clk) begin
    if (dw)
      {A[2], A[3]} <= write_data;
    else if (we)
      A[write_addr] <= write_data[511:0];
  end

endmodule
```

علاوه بر حالت عادی نوشتن در حافظه، به حالت double write نیز برای نوشتن در دو رجیستر R4 و R3 به طور هم‌زمان نیاز داریم که دستورات ALU از آن استفاده خواهند کرد.

کد ماژول واحد محاسبات (ALU.v):

```
module ALU (
  input signed [511:0] operand1, operand2,
  input opcode, // 0 for addition, 1 for multiplication
  output [1023:0] result, // {result[543:512], result[31:0]} = operand1[31:0] op operand2[31:0] and so on
  output [31:0] cc // flattened array (cc[1:0] -> condition code for operand[31:0] and so on)
);
  /*
    cc <- 00 if result = 0
    cc <- 01 if result < 0
    cc <- 10 if result > 0
    cc <- 11 if result doesn't fit in the lower 32 bits
  */

  genvar i;
  generate
    for (i = 0; i < 16; i = i + 1) begin
      assign {result[512+32*i+:32], result[32*i+:32]} = opcode
        ? $signed(operand1[32*i+:32]) * $signed(operand2[32*i+:32])
        : $signed(operand1[32*i+:32]) + $signed(operand2[32*i+:32]);
      assign cc[2*i+1:2*i] = $signed(result[32*i+:32]) != $signed({result[512+32*i+:32], result[32*i+:32]})
        ? 11
        : {$signed({result[512+32*i+:32], result[32*i+:32]})>0, $signed({result[512+32*i+:32], result[32*i+:32]})<0};
    end
  endgenerate
endmodule
```

ماژول ALU تنها یک بیت opcode دریافت می‌کند که اگر ۱ بود، تمام اعضای دو آرایه را در هم ضرب می‌کند و اگر صفر بود آن‌ها را با هم جمع می‌کند. همچنین خروجی عملیات در result ذخیره می‌شود که یک آرایه ۱۰۲۴ بیتی است و ۵۱۲ بیت سمت راست آن شامل سمت راست حاصل ضرب یا جمع هر خانه آرایه‌هاست و ۵۱۲ بیت سمت چپ نیز به همین شکل. همچنین تمام اعداد به شکل مکمل-۲ در نظر گرفته‌شده‌اند.
برای تعیین وضعیت ALU نیز از آرایه ۳۲ بیتی cc استفاده می‌کنیم که هر دو بیت آن نشان‌دهنده‌ی وضعیت انجام عملیات خواسته‌شده روی یک خانه‌ی دو آرایه است (در مجموع ۵۱۲ تقسیم بر ۳۲ یعنی ۱۶ خانه داریم). اگر cc[I] مساوی ۳ (۱۱ دودویی) باشد یعنی خروجی در ۳۲ بیت کم‌ارزش جا نشده است و اوورفلو رخ داده است، اما در هر صورت ۳۲ بیت پرارزش خروجی را نیز در result داریم و در حقیقت هیچ‌گاه اوورفلو رخ نمی‌دهد.

کد حافظه دستورات (Imem.v):

```
module Imem
  #(parameter INSTRUCTIONS="")
  (
    input clock,
    input [15:0] inst_addr,
    output reg [15:0] instruction
  );

  reg [15:0] mem [0:(2**15)-1];

  initial
    if (INSTRUCTIONS != "")
      $readmemb(INSTRUCTIONS, mem);

  always @(posedge clock)
    instruction <= mem[inst_addr];
endmodule
```

پارامتر INSTRUCTIONS آدرس فایلی است که دستورات در آن قرار دارند و با استفاده از دستور $readmemb از فایل خوانده شده و در mem ریخته می‌شوند. تمام دستورات ۱۶ بیتی هستند و ظرفیت حافظه به اندازه ۲۰۴۸ دستور است (که قابل تغییر است). توجه کنید که decode کردن دستورات این‌جا انجام نمی‌شود و Imem صرفاً مسئول خواندن دستورات است.

حافظه اصلی (Dmem.v) متشکل از ۵۱۲ آرایه ۳۲ بیتی است (آدرس‌ها نیز ۹ بیتی هستند) و مقدار اولیه آن از فایلی که آدرس آن در پارامتر INITIAL_DATA قرار دارد خوانده می‌شود (با استفاده از دستور $readmemh). خواندن از این حافظه به صورت آسنکرون انجام می‌شود و صرفاً سیگنال ورودی re یا همان read enable باید یک باشد. اما نوشتن به صورت سنکرون با لبه‌ی پایین‌رونده‌ی کلاک انجام می‌شود و سیگنال we یا write enable باید یک باشد.

پردازنده اصلی در فایل CPU.v قرار دارد وظیفه پردازنده اصلی decode کردن دستوراتی که از Imem می‌آیند می‌باشد. به طور کلی دستورات به دسته‌های زیر تقسیم می‌شوند:

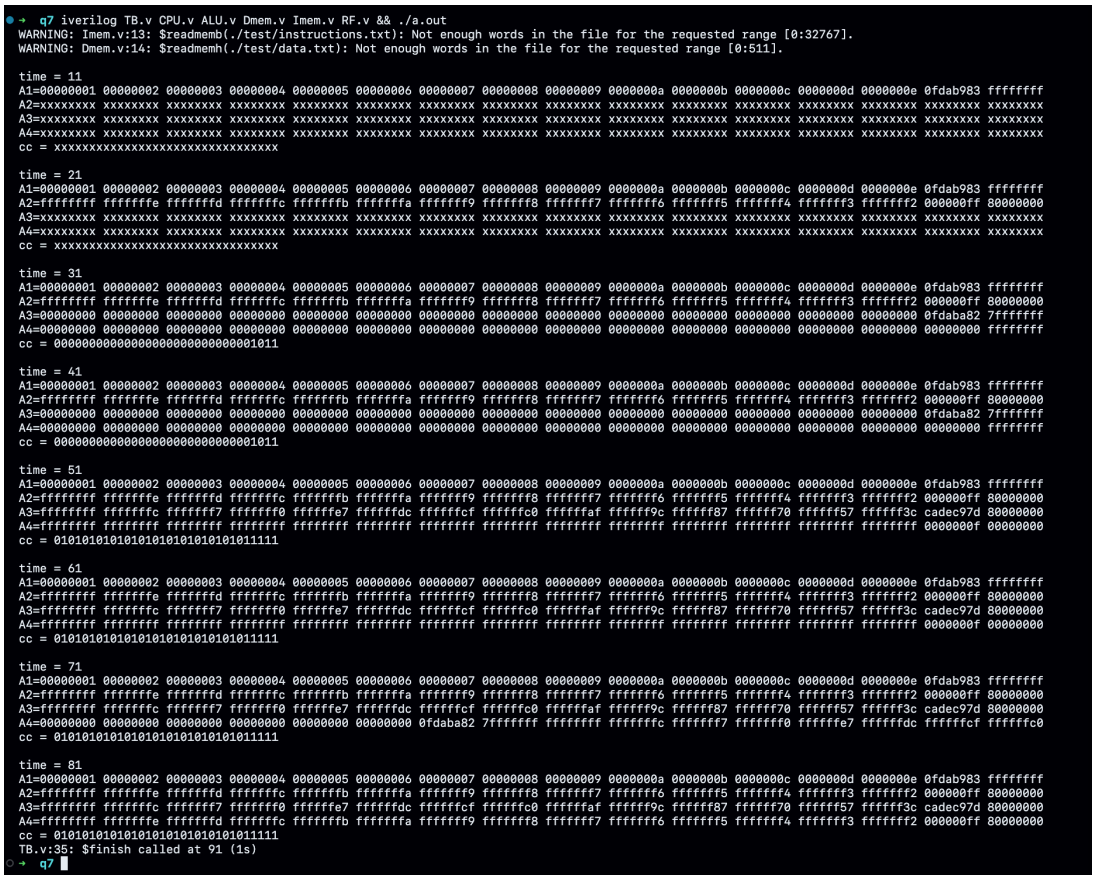
- Load: 01 <mem address[8:0]> xxx <reg number[1:0]>
- Store: 11 <mem address[8:0]> xxx <reg number[1:0]>
- Mov: 10 xxxxxxxxxx <rd[1:0]> <rs[1:0]>
- Add: 00 xxxxxxxxxxxxx 0
- Mul: 00 xxxxxxxxxxxxx 1

درباره دستور load: کارکرد این دستور به این صورت است که از خانه مشخص‌شده شروع می‌کند و محتویات آن خانه تا ۱۶ خانه بعدی را به صورت unaligned و چرخشی (یعنی اگر به انتهای حافظه رسید به ابتدای آن برمی‌گردد) در رجیستر مقصد می‌ریزد.

در تست‌بنچ، دستورات از فایل instructions.txt و مقادیر اولیه حافظه اصلی از فایل data.txt در پوشه test خوانده می‌شوند. دستورات زیر اجرا می‌شوند:

```
LOAD A1, 000000000
LOAD A2, 000100000
ADD A1, A2
STORE A3, 111110000
MULT A1, A2
STORE A3, 000000000
LOAD A4, 111111000
MOV A4, A2 (A4 <- A2)
```

در این تست عملکرد تمام دستورات برنامه، اوورفلوی FFFFFFFF*80000000، اوورفلوی معمولی، صفر و مثبت و منفی شدن حاصل و صحت cc و دایره‌ای عمل کردن حافظه چک می‌شود. پس از اجرای هر دستور، محتوای هر رجیستر و همچنین مقدار cc را چاپ می‌کنیم:



که می‌بینیم همه‌چیز مطابق انتظار کار می‌کند.