دانشکده مهندسی کامپیوتر شریف

عنوان:

مستندات جبرانی پایانترم DSD

نام و نام خانوادگی:

پدرام آذرهوش

شماره دانشجویی:

401105515

در اینجا سوال اول میانترم حل شده است، با وجود اینکه در کد مربوط به هر بخش کامنت گذاری انجام گرفته اما در اینجا هم توضیحاتی جهت شفاف سازی بیشتر آمده است:

بخش اول سوال) فایل های STACK_BASED_ALU و tb_stack مربوط به این بخش هستند که در فایل STACK_BASED_ALU عملیات هایی که توسط پنج opcode تعیین شده در سوال صورت می گیرد پیاده سازی شده و اینکه هر بخش مربوط به کدام آپکود می باشد نیز کامنت گذاری شده است و همچنین در فایل تست بنچ این ماژول که همان tb_stack باشد این عملیات ها مورد تست قرار گرفته اند.

در اینجا عکسی از کد ماژول STACK_BASED_ALU می بینید که توضیحاتی در رابطه با آن در ادامه آمده است:

C:/altera/13.1/STACK_BASED_ALU.v (/testbench_s/uut) - Default

```
 9
10      reg signed [N-1:0] stack [0:STACK_SIZE-1];
11      reg [3:0] sp; // Stack pointer
12
13      always @(posedge clk or posedge rst) begin
14          if (rst) begin
15              sp <= 0;
16              output_data <= 0;
17              overflow <= 0;
18          end else begin
19              overflow <= 0;
20              case (opcode)
21                  3'b100: begin // Addition
22                      if (sp > 1) begin
23                          output_data <= stack[sp-1] + stack[sp-2];
24                          // Check for overflow
25                          if ((stack[sp-1] > 0 && stack[sp-2] > 0 && output_data < 0) ||
26                              (stack[sp-1] < 0 && stack[sp-2] < 0 && output_data > 0)) begin
27                              overflow <= 1;
28                          end
29                      end else begin
30                          output_data <= 0;
31                      end
32                  end
33
34                  3'b101: begin // Multiply
35                      if (sp > 1) begin
36                          output_data <= stack[sp-1] * stack[sp-2];
37                          // Check for overflow
38                          if (stack[sp-1] != 0 && output_data / stack[sp-1] != stack[sp-2]) begin
39                              overflow <= 1;
40                          end
41                      end else begin
42                          output_data <= 0;
43                      end
44                  end
```

همانطور که می بینید در کد بالا استک به صورت یک مموری تعریف شده است و یک استک پوینتر نیز برای آنکه بدانیم کجای کار هستیم و برای عملیات های add و mul ما می آییم و stack[sp-1] که بالاترین عضو استک است به همراه stack[sp-2] که عضو بعد آن است جمع یا ضرب کرده و درون output_data قرار می دهیم و سپس نیز درهر بخش overflow چک می شود و اگر هم عملیاتی انجام نگرفت output data را برابر صفر قرار می دهیم.

عکس دوم:

```
45
46                  3'b110: begin // PUSH
47                      if (sp < STACK_SIZE) begin
48                          stack[sp] <= input_data;
49                          sp <= sp + 1;
50                      end
51                  end
52
53                  3'b111: begin // POP
54                      if (sp > 0) begin
55                          sp <= sp - 1;
56                          output_data <= stack[sp];
57                      end else begin
58                          output_data <= 0;
59                      end
60                  end
61
62                  default: begin // No Operation for 0xx
63                      output_data <= 0;
64                  end
65              endcase
66          end
67      end
68  endmodule
69
```

در اینجا نیز سه opcode دیگر پیاده سازی شده اند که بدیهی است برای push باید ورودی را درون بالاترین عضو استک که استک پوینتر بدان اشاره دارد قرار دهیم و sp را افزایش دهیم و برای pop کردن نیز باید sp را کاهش داده و سپس محتوایی که بدان اشاره دارد را به بیرون دهیم، برای no operation هم کاری نکرده و تنها خروجی را صفر قرار می دهیم، سایر بخش ها همانند بخش های قبل است.

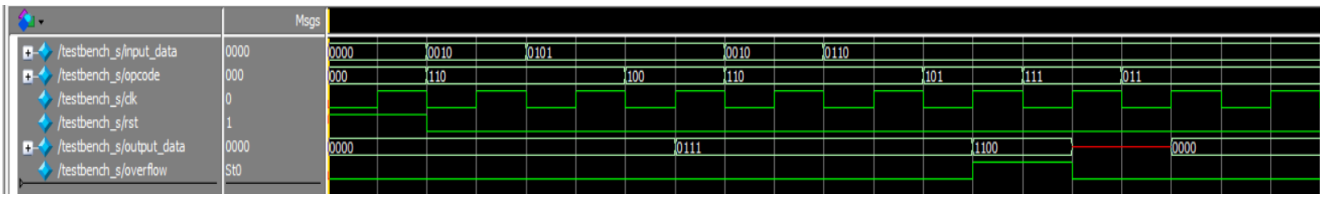
در زیر نیز نتیجه تست هایی که در تست بنچ آورده ایم آمده است.

در ابندا اعداد 2 و 5 پوش شده اند وسپس جمع زده شده امد که نتیجه آن 7 شده.

سپس اعداد 2 و 6 را پوش کرده و ضرب کرده ایم ک ه نتیجه آن به درستی برابر با 12 شده است.

در انتها یکبار pop کرده وسپس آپکود عملیات no operation را وارد کرده ایم که خروجی مطابق انتظار صفر شده است.

بنابراین در این تست بنچ همه opcode ها به همراه بیت سرریز مورد بررسی قرار گرفته است و صحت کارکرد آن در زیر آمده است:



بخش دوم سوال) فایل expression_solver مربوط به این بخش است که با استفاده از ماژول STACK_BASED_ALU و عملیات های تحت پوشش آن پیاده سازی شده است، هرچند که در پیاده سازی واضح بوده و برای بخش بخش آن کامنت گذاری شده است اما با این حال مراحل پیاده سازی در این فایل به صورت زیر است:

معادله مدنظر برای حل: (5 + 6) + 20- * (3 + 4 + 10) + 3 * 2

- **Push 2 onto the stack.**
- **Push 3 onto the stack.**
- **Multiply the top two values (2 * 3) and save the result.**
- **Push 10 onto the stack.**
- **Push 4 onto the stack.**
- **Add the top two values (10 + 4).**
- **Push 3 onto the stack.**
- **Add the top two values (14 + 3).**
- **Push -20 onto the stack.**
- **Multiply the top two values (17 * -20) and save the result.**
- **Push 6 onto the stack.**
- **Push 5 onto the stack.**
- **Add the top two values (6 + 5).**
- **Add the top two values (6 + 11).**
- **Add the result of the previous addition to the result of the multiplication (17 + (-340)).**

که پاسخ باید برابر با 323- باشد که با اجرا کردن برنامه خواهیم دید که درنهایت نتیجه همان است.

عکس نتیجه نهایی در شبیه سازی که این حرف را تایید می کند:

نکته: در اینجا از دستور سیستمی stop استفاده نشده است و برای تست کردن پاسخ می توان قدم به قدم run کرد با توجه به اینکه دوره تناوب کلاک 20 است.