

سوال ( ۷ )

```

1 module registerfile (input clk, input [2:0] regnum, input load, input [511:0] in_A1, in_A2, in_A3, in_A4, output reg [511:0] out_A1, out_A2, out_A3, out_A4);
2
3 always @(posedge clk) begin
4
5     if(regnum == 3'b000) begin
6         if(load == 1)
7             out_A1 = in_A1;
8     end else if(regnum == 3'b001) begin
9         if(load == 1)
10            out_A2 = in_A2;
11     end else if(regnum == 3'b010) begin
12         if(load == 1)
13            out_A3 = in_A3;
14     end else if(regnum == 3'b011) begin
15         if(load == 1)
16            out_A4 = in_A4;
17     end else begin
18         out_A3 = in_A3;
19         out_A4 = in_A4;
20     end
21 end
22
23 endmodule
24
25 module ALU (input add, mul, input [511:0] A1, A2, output reg [511:0] A3, A4);
26
27 always @(*) begin
28
29     if((add == 1) && (mul == 0)) begin
30         (A4, A3) = $signed(A1) + $signed(A2);
31     end else if((add == 0) && (mul == 1)) begin
32         (A4, A3) = $signed(A1) * $signed(A2);
33     end
34 end
35
36 end
37

```

طبق خواست سوال یک ماژول registerfile تعریف می‌کنیم که قابلیت ذخیره سازی ۴ آرایه ۵۱۲ بیتی را داشته باشد. همانطور که در تصویر ماژول مشاهده می‌کنید برای ورودی ماژول علاوه بر کلاک یک ورودی regnum را تعریف می‌کنیم که شماره های ۰۰۰ و ۰۰۱ و ۰۱۰ و ۰۱۱ آن به ترتیب شماره های رجیستر یک تا چهار هستند که رجیستری که با آن کار داریم را مشخص می‌کند. همچنین اگر بیت پرارزش regnum برابر ۱ باشد این شماره کار با رجیستر ۳ و ۴ را به طور همزمان انجام می‌دهد. ورودی دیگر این ماژول ورودی load است که در صورتی که یک باشد اجازه می‌دهد مقادیر رجیسترها تغییر کنند. ورودی ها و خروجی های ۵۱۲ بیتی هم متعلق به رجیسترها می‌باشند.

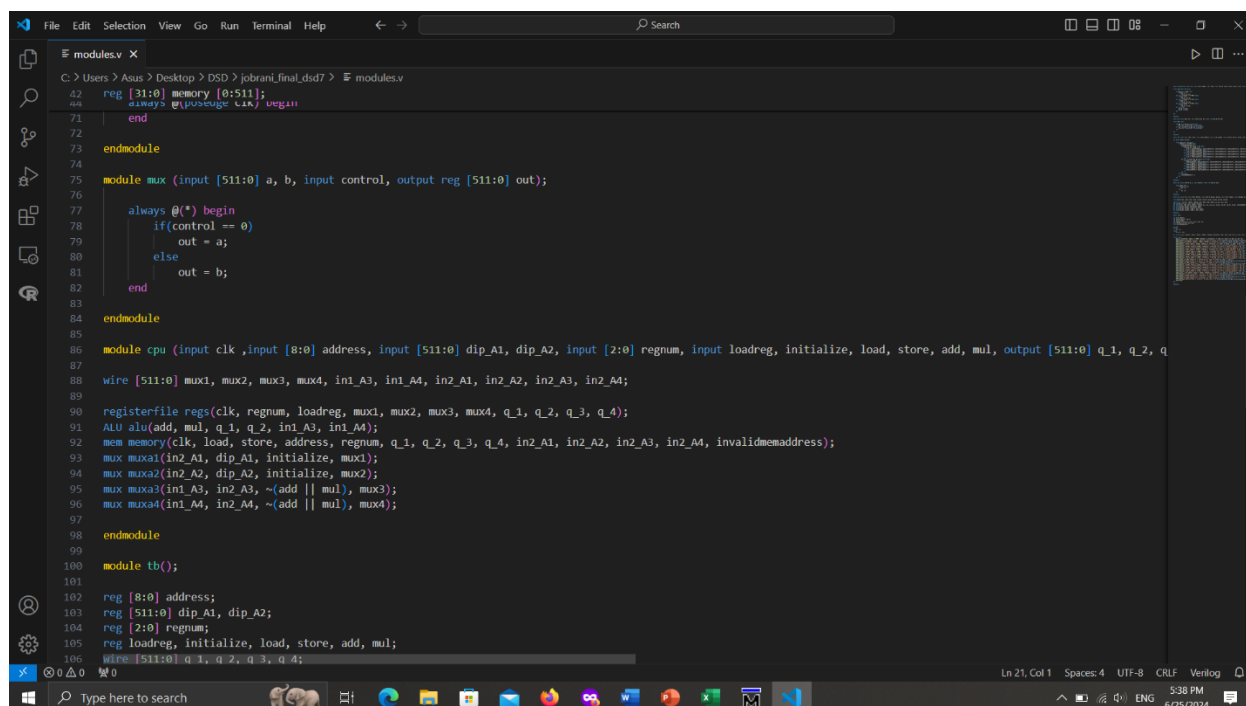
یک ماژول دیگر که در تصویر بالا مشاهده می‌کنید ماژول ALU است که عملیات‌های ضرب و جمع را به صورت علامت دار انجام می‌دهد و ۵۱۲ بیت کم ارزش را در رجیستر ۳ و ۵۱۲ بیت پرارزش را در رجیستر ۴ می‌ریزد. در صورتی که  $add = 1$ ,  $mul = 0$  باشد جمع و اگر  $add = 0$ ,  $mul = 1$  باشد ضرب را انجام می‌دهد.

```
File Edit Selection View Go Run Terminal Help
modules.v
C:\Users\Asus\Desktop\DSO\jobrani_final_dsd7\modules.v
38
39
40 module mem (input clk, load, store, input [8:0] address, input [2:0] regnum, input [511:0] out_A1, out_A2, out_A3, out_A4, output reg [511:0] in_A1, in_A2, in_A3, in_A4,
41
42 reg [31:0] memory [0:511];
43
44 always @(posedge clk) begin
45     if(address <= 9'd506) begin
46         invalidmemaddress = 0;
47         if((load == 1) && (store == 0)) begin
48             if(regnum == 3'b000) begin
49                 in_A1 = (memory[address], memory[address+1], memory[address+2], memory[address+3], memory[address+4], memory[address+5], memory[address+6], memory[address+7],
50             end else if(regnum == 3'b001) begin
51                 in_A2 = (memory[address], memory[address+1], memory[address+2], memory[address+3], memory[address+4], memory[address+5], memory[address+6], memory[address+7],
52             end else if(regnum == 3'b010) begin
53                 in_A3 = (memory[address], memory[address+1], memory[address+2], memory[address+3], memory[address+4], memory[address+5], memory[address+6], memory[address+7],
54             end else if(regnum == 3'b011) begin
55                 in_A4 = (memory[address], memory[address+1], memory[address+2], memory[address+3], memory[address+4], memory[address+5], memory[address+6], memory[address+7],
56             end
57         end else if((load == 0) && (store == 1)) begin
58             if(regnum == 3'b000) begin
59                 memory[address], memory[address+1], memory[address+2], memory[address+3], memory[address+4], memory[address+5], memory[address+6], memory[address+7],
60             end else if(regnum == 3'b001) begin
61                 memory[address], memory[address+1], memory[address+2], memory[address+3], memory[address+4], memory[address+5], memory[address+6], memory[address+7],
62             end else if(regnum == 3'b010) begin
63                 memory[address], memory[address+1], memory[address+2], memory[address+3], memory[address+4], memory[address+5], memory[address+6], memory[address+7],
64             end else if(regnum == 3'b011) begin
65                 memory[address], memory[address+1], memory[address+2], memory[address+3], memory[address+4], memory[address+5], memory[address+6], memory[address+7],
66             end
67         end
68     end else begin
69         invalidmemaddress = 1;
70     end
71 end
72
73 endmodule
74
```

```
File Edit Selection View Go Run Terminal Help
modules.v
C:\Users\Asus\Desktop\DSO\jobrani_final_dsd7\modules.v
38
39
40 input reg invalidmemaddress;
41
42
43
44
45
46
47
48
49 :ss+7], memory[address+8], memory[address+9], memory[address+10], memory[address+11], memory[address+12], memory[address+13], memory[address+14], memory[address+15];
50
51 :ss+7], memory[address+8], memory[address+9], memory[address+10], memory[address+11], memory[address+12], memory[address+13], memory[address+14], memory[address+15];
52
53 :ss+7], memory[address+8], memory[address+9], memory[address+10], memory[address+11], memory[address+12], memory[address+13], memory[address+14], memory[address+15];
54
55 :ss+7], memory[address+8], memory[address+9], memory[address+10], memory[address+11], memory[address+12], memory[address+13], memory[address+14], memory[address+15];
56
57
58
59 memory[address+8], memory[address+9], memory[address+10], memory[address+11], memory[address+12], memory[address+13], memory[address+14], memory[address+15] = out_A1;
60
61 memory[address+8], memory[address+9], memory[address+10], memory[address+11], memory[address+12], memory[address+13], memory[address+14], memory[address+15] = out_A2;
62
63 memory[address+8], memory[address+9], memory[address+10], memory[address+11], memory[address+12], memory[address+13], memory[address+14], memory[address+15] = out_A3;
64
65 memory[address+8], memory[address+9], memory[address+10], memory[address+11], memory[address+12], memory[address+13], memory[address+14], memory[address+15] = out_A4;
66
67
68
69
70
71
72
73
74
```

دو تصویر بالا که مشاهده می‌کنید مربوط به ماژول حافظه هستند. همانطور که می‌بینید در این ماژول حافظه‌ای به عمق ۵۱۲ و عرض ۳۲ بیت وجود دارد. ورودی‌های ماژول علاوه بر کلاک دو ورودی load و store هستند که فقط یکی از آنها می‌تواند یک باشد و مشخص می‌کند که داده در حافظه ذخیره

می‌شود یا روی رجیستر نوشته می‌شود. یک ورودی دیگر address است که ۹ بیت است زیرا حافظه ما ۵۱۲ خانه دارد. یک ورودی دیگر regnum است که طبق قاعده‌ای که پیش تر ذکر شد شماره ی رجیسترها را برای ارتباط با حافظه مشخص می‌کند. ورودی و خروجی های ۵۱۲ بیتی هم مربوط به ارتباط رجیسترها با حافظه هستند. یک خروجی دیگر invalidmemaddress است. طبق فرض سوال این پردازنده قابلیت بارگزاری / ذخیره سازی ۱۶ خانه حافظه پشت سر هم یعنی ۵۱۲ بیت را دارد. بنابراین آدرس هایی که می‌توان به پردازنده داد تا از آن آدرس ۱۶ خانه حافظه را دسترسی داشته باشد خانه های ۰ تا ۴۹۶ حافظه هستند و آدرس‌های بیشتر از ۴۹۶ کمتر از ۱۶ خانه حافظه را شامل می‌شوند بنابراین اگر در دستور load و یا store آدرس‌های بیش از این عدد داده شود عملیاتی انجام نمی‌شود و خروجی invalidmemaddress برابر ۱ می‌شود.



```

modules.v
C:\Users\Asus\Desktop\DSO\jobrani_final_dsd7 > modules.v
42 reg [31:0] memory [0:511];
43 always @(posedge clk) begin
71 end
72
73 endmodule
74
75 module mux (input [511:0] a, b, input control, output reg [511:0] out);
76
77     always @(*) begin
78         if(control == 0)
79             out = a;
80         else
81             out = b;
82         end
83     endmodule
84
85 module cpu (input clk, input [8:0] address, input [511:0] dip_A1, dip_A2, input [2:0] regnum, input loadreg, initialize, load, store, add, mul, output [511:0] q_1, q_2, q_3, q_4);
86
87     wire [511:0] mux1, mux2, mux3, mux4, in1_A3, in1_A4, in2_A1, in2_A2, in2_A3, in2_A4;
88
89     registerfile regs(clk, regnum, loadreg, mux1, mux2, mux3, mux4, q_1, q_2, q_3, q_4);
90     ALU alu(add, mul, q_1, q_2, in1_A3, in1_A4);
91     mem memory(clk, load, store, address, regnum, q_1, q_2, q_3, q_4, in2_A1, in2_A2, in2_A3, in2_A4, invalidmemaddress);
92     mux mux1(in2_A1, dip_A1, initialize, mux1);
93     mux mux2(in2_A2, dip_A2, initialize, mux2);
94     mux mux3(in1_A3, in2_A3, ~(add || mul), mux3);
95     mux mux4(in1_A4, in2_A4, ~(add || mul), mux4);
96
97 endmodule
98
99 module tb();
100
101     reg [8:0] address;
102     reg [511:0] dip_A1, dip_A2;
103     reg [2:0] regnum;
104     reg loadreg, initialize, load, store, add, mul;
105     wire [511:0] q_1, q_2, q_3, q_4;
106
107     modules.v
  
```

ما برای ورودی ۴ رجیسترمان به ۴ مالتیپلکسر نیاز داریم چرا که یک ورودی رجیستر ۱ و ۲ به بیرون متصل است تا دیتا به طور مستقیم توسط کاربر وارد پردازنده شود ( در فرض سوال نبود اما منطقاً باید داده اولیه از بیرون وارد پردازنده شود زیرا خود پردازنده در ابتدا حافظه اش خالی است.) و دیگری به حافظه متصل است و برای رجیسترهای ۳ و ۴ یک ورودی از حافظه و دیگری از ALU می‌آید بنابراین به ۴ مالتیپلکسر نیاز داریم.

سپس همانطور که در ماژول cpu مشاهده می‌کنید این ماژول اینستنس های ماژول هایی که تا اینجا تعریف کردیم را در خود جای میدهد و عملاً بخش های مختلف پردازنده را به هم متصل میکند که شامل یک رجیستر فایل و یک alu و یک مموری و چهار مالتیپلکسر است. ورودی های سه ماژول اول مشخص بودند و توضیح داده شد. ورودی های مالتیپلکسر هم مشخص هستند. حال ورودی control مالتیپلکسر ها را بررسی می‌کنیم. برای ماژول cpu یک ورودی initialize داریم که زمانی که ۱ است ورودی رجیسترهای ۱ و ۲ را از بیرون می‌گیریم در غیر اینصورت ورودی رجیسترهای ۱ و ۲ به خروجی مموری متصل میشود. در مورد مالتیپلکسر های ۳ و ۴ نیز ورودی کنترل زمانی ۱ است که عملیات ضرب یا جمع نداشته باشیم تا ورودی رجیستر به خروجی حافظه متصل شود نه alu.

```
File Edit Selection View Go Run Terminal Help
C:\Users\Asus\Desktop\DSO\Jobrani_final_dsd7\modules.v
100 module tb();
101
102 reg [8:0] address;
103 reg [511:0] dip_A1, dip_A2;
104 reg [2:0] regnum;
105 reg loadreg, initialize, load, store, add, mul;
106 wire [511:0] q_1, q_2, q_3, q_4;
107 wire invalidmemaddress;
108
109 reg clk;
110 initial
111     clk = 0;
112 always
113     #5 clk = ~clk;
114
115 cpu_processor(clk, address, dip_A1, dip_A2, regnum, loadreg, initialize, load, store, add, mul, q_1, q_2, q_3, q_4, invalidmemaddress);
116
117 initial begin
118     dip_A1 = 512'd2133; regnum = 3'b000; loadreg = 1; initialize = 1; load = 0; store = 0; add = 0; mul = 0;
119     #100 $display("initialize A1:\nA_1 = %d,\nA_2 = %d,\n(A_4,A_3) = %d", $signed(q_1), $signed(q_2), $signed((q_4,q_3)));
120     dip_A2 = 512'd65323; regnum = 3'b001; loadreg = 1; initialize = 1; load = 0; store = 0; add = 0; mul = 0;
121     #100 $display("initialize A2:\nA_1 = %d,\nA_2 = %d,\n(A_4,A_3) = %d", $signed(q_1), $signed(q_2), $signed((q_4,q_3)));
122     #100 address = 9'd45; regnum = 3'b000; loadreg = 0; initialize = 0; load = 0; store = 1; add = 0; mul = 0;
123     #100 $display("store A1 in mem[45]:\nA_1 = %d,\nA_2 = %d,\n(A_4,A_3) = %d,\ninvalidmemaddress = %d", $signed(q_1), $signed(q_2), $signed((q_4,q_3)), invalidmemaddress);
124     #100 address = 9'd100; regnum = 3'b001; loadreg = 0; initialize = 0; load = 0; store = 1; add = 0; mul = 0;
125     #100 $display("store A2 in mem[100]:\nA_1 = %d,\nA_2 = %d,\n(A_4,A_3) = %d,\ninvalidmemaddress = %d", $signed(q_1), $signed(q_2), $signed((q_4,q_3)), invalidmemaddress);
126     #100 address = 9'd45; regnum = 3'b001; loadreg = 1; initialize = 0; load = 1; store = 0; add = 0; mul = 0;
127     #100 $display("load mem[45] in A2:\nA_1 = %d,\nA_2 = %d,\n(A_4,A_3) = %d,\ninvalidmemaddress = %d", $signed(q_1), $signed(q_2), $signed((q_4,q_3)), invalidmemaddress);
128     #100 address = 9'd100; regnum = 3'b000; loadreg = 1; initialize = 0; load = 1; store = 0; add = 0; mul = 0;
129     #100 $display("load mem[100] in A1:\nA_1 = %d,\nA_2 = %d,\n(A_4,A_3) = %d,\ninvalidmemaddress = %d", $signed(q_1), $signed(q_2), $signed((q_4,q_3)), invalidmemaddress);
130     #100 address = 9'd500; regnum = 3'b000; loadreg = 0; initialize = 0; load = 0; store = 1; add = 0; mul = 0;
131     #100 $display("store A1 in mem[500]:\nA_1 = %d,\nA_2 = %d,\n(A_4,A_3) = %d,\ninvalidmemaddress = %d", $signed(q_1), $signed(q_2), $signed((q_4,q_3)), invalidmemaddress);
132     #100 address = 9'd501; regnum = 3'b000; loadreg = 1; initialize = 0; load = 1; store = 0; add = 0; mul = 0;
133     #100 $display("load mem[501] in A1:\nA_1 = %d,\nA_2 = %d,\n(A_4,A_3) = %d,\ninvalidmemaddress = %d", $signed(q_1), $signed(q_2), $signed((q_4,q_3)), invalidmemaddress);
134     #100 regnum = 3'b100; loadreg = 1; initialize = 0; load = 0; store = 0; add = 1; mul = 0;
135     #100 $display("addition:\nA_1 = %d,\nA_2 = %d,\n(A_4,A_3) = %d", $signed(q_1), $signed(q_2), $signed((q_4,q_3)));
136
137 end
138 endmodule
```

```
File Edit Selection View Go Run Terminal Help
C:\Users\Asus\Desktop\DSO\Jobrani_final_dsd7\modules.v
117 initial begin
118     dip_A1 = 512'd2133; regnum = 3'b000; loadreg = 1; initialize = 1; load = 0; store = 0; add = 0; mul = 0;
119     #100 $display("initialize A1:\nA_1 = %d,\nA_2 = %d,\n(A_4,A_3) = %d", $signed(q_1), $signed(q_2), $signed((q_4,q_3)));
120     dip_A2 = 512'd65323; regnum = 3'b001; loadreg = 1; initialize = 1; load = 0; store = 0; add = 0; mul = 0;
121     #100 $display("initialize A2:\nA_1 = %d,\nA_2 = %d,\n(A_4,A_3) = %d", $signed(q_1), $signed(q_2), $signed((q_4,q_3)));
122     #100 address = 9'd45; regnum = 3'b000; loadreg = 0; initialize = 0; load = 0; store = 1; add = 0; mul = 0;
123     #100 $display("store A1 in mem[45]:\nA_1 = %d,\nA_2 = %d,\n(A_4,A_3) = %d,\ninvalidmemaddress = %d", $signed(q_1), $signed(q_2), $signed((q_4,q_3)), invalidmemaddress);
124     #100 address = 9'd100; regnum = 3'b001; loadreg = 0; initialize = 0; load = 0; store = 1; add = 0; mul = 0;
125     #100 $display("store A2 in mem[100]:\nA_1 = %d,\nA_2 = %d,\n(A_4,A_3) = %d,\ninvalidmemaddress = %d", $signed(q_1), $signed(q_2), $signed((q_4,q_3)), invalidmemaddress);
126     #100 address = 9'd45; regnum = 3'b001; loadreg = 1; initialize = 0; load = 1; store = 0; add = 0; mul = 0;
127     #100 $display("load mem[45] in A2:\nA_1 = %d,\nA_2 = %d,\n(A_4,A_3) = %d,\ninvalidmemaddress = %d", $signed(q_1), $signed(q_2), $signed((q_4,q_3)), invalidmemaddress);
128     #100 address = 9'd100; regnum = 3'b000; loadreg = 1; initialize = 0; load = 1; store = 0; add = 0; mul = 0;
129     #100 $display("load mem[100] in A1:\nA_1 = %d,\nA_2 = %d,\n(A_4,A_3) = %d,\ninvalidmemaddress = %d", $signed(q_1), $signed(q_2), $signed((q_4,q_3)), invalidmemaddress);
130     #100 address = 9'd500; regnum = 3'b000; loadreg = 0; initialize = 0; load = 0; store = 1; add = 0; mul = 0;
131     #100 $display("store A1 in mem[500]:\nA_1 = %d,\nA_2 = %d,\n(A_4,A_3) = %d,\ninvalidmemaddress = %d", $signed(q_1), $signed(q_2), $signed((q_4,q_3)), invalidmemaddress);
132     #100 address = 9'd501; regnum = 3'b000; loadreg = 1; initialize = 0; load = 1; store = 0; add = 0; mul = 0;
133     #100 $display("load mem[501] in A1:\nA_1 = %d,\nA_2 = %d,\n(A_4,A_3) = %d,\ninvalidmemaddress = %d", $signed(q_1), $signed(q_2), $signed((q_4,q_3)), invalidmemaddress);
134     #100 regnum = 3'b100; loadreg = 1; initialize = 0; load = 0; store = 0; add = 1; mul = 0;
135     #100 $display("addition:\nA_1 = %d,\nA_2 = %d,\n(A_4,A_3) = %d", $signed(q_1), $signed(q_2), $signed((q_4,q_3)));
136     #100 regnum = 3'b100; loadreg = 1; initialize = 0; load = 0; store = 0; add = 0; mul = 1;
137     #100 $display("multiplication:\nA_1 = %d,\nA_2 = %d,\n(A_4,A_3) = %d", $signed(q_1), $signed(q_2), $signed((q_4,q_3)));
138     #100 address = 9'd299; regnum = 3'b010; loadreg = 0; initialize = 0; load = 0; store = 1; add = 0; mul = 0;
139     #100 $display("store A3 in mem[299]:\nA_3 = %d,\nA_4 = %d,\n(A_4,A_3) = %d,\ninvalidmemaddress = %d", $signed(q_3), $signed(q_4), $signed((q_4,q_3)), invalidmemaddress);
140     #100 address = 9'd400; regnum = 3'b011; loadreg = 0; initialize = 0; load = 0; store = 1; add = 0; mul = 0;
141     #100 $display("store A4 in mem[400]:\nA_3 = %d,\nA_4 = %d,\n(A_4,A_3) = %d,\ninvalidmemaddress = %d", $signed(q_3), $signed(q_4), $signed((q_4,q_3)), invalidmemaddress);
142     #100 address = 9'd299; regnum = 3'b000; loadreg = 1; initialize = 0; load = 1; store = 0; add = 0; mul = 0;
143     #100 $display("load mem[299] in A1:\nA_1 = %d,\nA_2 = %d,\n(A_4,A_3) = %d,\ninvalidmemaddress = %d", $signed(q_1), $signed(q_2), $signed((q_4,q_3)), invalidmemaddress);
144     dip_A2 = 512'd54229; regnum = 3'b001; loadreg = 1; initialize = 1; load = 0; store = 0; add = 0; mul = 0;
145     #100 $display("initialize A2:\nA_1 = %d,\nA_2 = %d,\n(A_4,A_3) = %d", $signed(q_1), $signed(q_2), $signed((q_4,q_3)));
146     #100 regnum = 3'b100; loadreg = 1; initialize = 0; load = 0; store = 0; add = 0; mul = 1;
147     #100 $display("multiplication:\nA_1 = %d,\nA_2 = %d,\n(A_4,A_3) = %d", $signed(q_1), $signed(q_2), $signed((q_4,q_3)));
148     #100 regnum = 3'b100; loadreg = 1; initialize = 0; load = 0; store = 0; add = 1; mul = 0;
149     #100 $display("addition:\nA_1 = %d,\nA_2 = %d,\n(A_4,A_3) = %d", $signed(q_1), $signed(q_2), $signed((q_4,q_3)));
150
151 end
152 endmodule
```

درنهایت در مازول تست بنچ از cpu یک اینستنس می‌گیریم و تست‌های لازم را روی آن انجام می‌دهیم.

در تمام دستوراتی که وارد میکنیم باید مقادیر `regnum, loadreg, initialize, load, store, add, mul` را مشخص کنیم در دستوراتی که `initialize = 1` است باید `dip_A1` یا `dip_A2` را برای مقداردهی اولیه مشخص کنیم. در دستوراتی که یکی از `load, store` را ۱ می‌کنیم آدرس را هم مشخص می‌کنیم. حال دستوراتی که برای تست در نظر گرفتیم را بررسی و در ادامه خروجی آنها را نشان خواهیم داد.

ابتدا رجیسترهای ۱ و ۲ را با دیپ سوییچ‌ها به ترتیب مقداردهی می‌کنیم. سپس محتوای رجیستر ۱ را در آدرس ۴۵ و رجیستر ۲ را در آدرس ۱۰۰ حافظه ذخیره می‌کنیم یعنی ۵۱۲ بیت رجیستر ۱ و ۲ به ترتیب در ۱۶ خانه حافظه با شروع از آدرس‌های ۴۵ و ۱۰۰ قرار می‌گیرند. سپس برای اطمینان از عملکرد درست حافظه اینبار محتوای ۱۶ خانه حافظه با شروع از آدرس ۱۰۰ را در رجیستر ۱ و ۱۶ خانه حافظه با شروع از آدرس ۴۵ را در رجیستر ۲ می‌گذاریم سپس دستورات `load` و `store` را برای آدرس‌های غیرمجاز انجام می‌دهیم و همانطور که در تصاویر بعدی مشاهده می‌کنید خروجی مناسب را نمایش می‌دهد و تغییری در حافظه و رجیسترها ایجاد نمی‌کند. و به این ترتیب عملکرد درست دو دستور بارگزاری و ذخیره را به طور کامل بررسی کردیم. (توجه: می‌توانستیم `load` را در هر آدرس مجازی انجام دهیم اما چون ذخیره سازی را در دو آدرس ۴۵ و ۱۰۰ انجام داده بودیم و تنها این دو خانه حافظه دارای محتوای معنی دار بودند از همین دو آدرس استفاده کردیم و عملاً محتوای دو رجیستر ۱ و ۲ را جابه جا کردیم.)

سپس دستور `addition` را اجرا می‌کنیم و همانطور که می‌بینید نتیجه به درستی در رجیسترهای ۳ و ۴ ذخیره شد. سپس عمل ضرب را انجام می‌دهیم که می‌بینیم آن هم به درستی اجرا شد. حال محتوای دو رجیستر ۳ و ۴ را در دو آدرس حافظه ذخیره می‌کنیم. سپس محتوای آدرسی که محتوای رجیستر ۳ در آن ذخیره شده بود را در رجیستر ۱ بارگزاری می‌کنیم. برای اطمینان از درستی عملکرد مدار برای اعداد منفی یک عدد منفی را وارد رجیستر دو می‌کنیم و عملیات ضرب و جمع را انجام می‌دهیم که به درستی اجرا میشوند.

به این ترتیب عملکرد مدار را در حالت های مختلف بررسی کردیم و با توجه به توضیحات داده شده و خروجی مدار واضح است مدار خواسته های مسئله را به طور کامل اجرا میکند.

```

# initialize A1:
# A_1 = 2133,
# A_2 = X,
# (A_4,A_3) =
# initialize A2:
# A_1 = 2133,
# A_2 = 65323,
# (A_4,A_3) =
# store A1 in mem[45]:
# A_1 = 2133,
# A_2 = 65323,
# (A_4,A_3) =
# invalidatememaddress = 0
# store A2 in mem[100]:
# A_1 = 2133,
# A_2 = 65323,
# (A_4,A_3) =
# invalidatememaddress = 0
# load mem[45] in A2:
# A_1 = 2133,
# A_2 = 2133,
# (A_4,A_3) =
# invalidatememaddress = 0
# load mem[100] in A1:
# A_1 = 65323,
# A_2 = 2133,
# (A_4,A_3) =

```

```

# load mem[45] in A2:
# A_1 = 2133,
# A_2 = 2133,
# (A_4,A_3) =
# invalidatememaddress = 0
# load mem[100] in A1:
# A_1 = 65323,
# A_2 = 2133,
# (A_4,A_3) =
# invalidatememaddress = 0
# store A1 in mem[500]:
# A_1 = 65323,
# A_2 = 2133,
# (A_4,A_3) =
# invalidatememaddress = 1
# load mem[501] in A1:
# A_1 = 65323,
# A_2 = 2133,
# (A_4,A_3) =
# invalidatememaddress = 1
# addition:
# A_1 = 65323,
# A_2 = 2133,
# (A_4,A_3) = 67456
# multiplication:
# A_1 = 65323,
# A_2 = 2133,
# (A_4,A_3) =

```

ModelSim - INTEL FPGA STARTER EDITION 2020.1

File Edit View Compile Simulate Add Transcript Tools Layout Bookmarks Window Help

Layout Simulate

ColumnLayout AllColumns

sim - Default

Instance Design unit Design unit type Top Category Visibility Total coverage

Project Memory List sim

Objects

Users\Asus\Desktop\DSO\jobrani\_final\_dsd7\modules.v (/tb) - Default

Ln#

Processes (Active)

Dataflow modules.v

Transcript

```
# multiplication:
# A_1 = 65533,
# A_2 = 2133,
# (A_4,A_3) = 139333959
# store A3 in mem[299]:
# A_3 = 139333959,
# A_4 = 0,
# (A_4,A_3) = 139333959,
# invalidatememaddress = 0
# store A4 in mem[400]:
# A_3 = 139333959,
# A_4 = 0,
# (A_4,A_3) = 139333959,
# invalidatememaddress = 0
# load mem[299] in A1:
# A_1 = 139333959,
# A_2 = 2133,
# (A_4,A_3) = 139333959,
# invalidatememaddress = 0
# initialize A2:
# A_1 = 139333959,
# A_2 = -54229,
# (A_4,A_3) = 139333959
# multiplication:
# A_1 = 139333959,
# A_2 = -54229,
# (A_4,A_3) = -7555941262611
```

Project: jobrani\_final\_dsd7 Now: 3,200 ps Delta: 0 sim:/tb/#INITIAL#117

Type here to search

9:28 PM 6/25/2024

ModelSim - INTEL FPGA STARTER EDITION 2020.1

File Edit View Compile Simulate Add Transcript Tools Layout Bookmarks Window Help

Layout Simulate

ColumnLayout AllColumns

sim - Default

Instance Design unit Design unit type Top Category Visibility Total coverage

Project Memory List sim

Objects

Users\Asus\Desktop\DSO\jobrani\_final\_dsd7\modules.v (/tb) - Default

Ln#

Dataflow modules.v

Transcript

```
# invalidatememaddress = 0
# store A4 in mem[400]:
# A_3 = 139333959,
# A_4 = 0,
# (A_4,A_3) = 139333959,
# invalidatememaddress = 0
# load mem[299] in A1:
# A_1 = 139333959,
# A_2 = 2133,
# (A_4,A_3) = 139333959,
# invalidatememaddress = 0
# initialize A2:
# A_1 = 139333959,
# A_2 = -54229,
# (A_4,A_3) = 139333959
# multiplication:
# A_1 = 139333959,
# A_2 = -54229,
# (A_4,A_3) = -7555941262611
# addition:
# A_1 = 139333959,
# A_2 = -54229,
# (A_4,A_3) = 139279730
** Note: $stop : C:/Users/Asus/Desktop/DSO/jobrani_final_dsd7/modules.v(150)
Time: 3200 ps Iteration: 0 Instance: /tb
Break in Module tb at C:/Users/Asus/Desktop/DSO/jobrani_final_dsd7/modules.v line 150
VSIOM 3>]
```

Project: jobrani\_final\_dsd7 Now: 3,200 ps Delta: 0 sim:/tb/#INITIAL#117

Type here to search

6:09 PM 6/25/2024