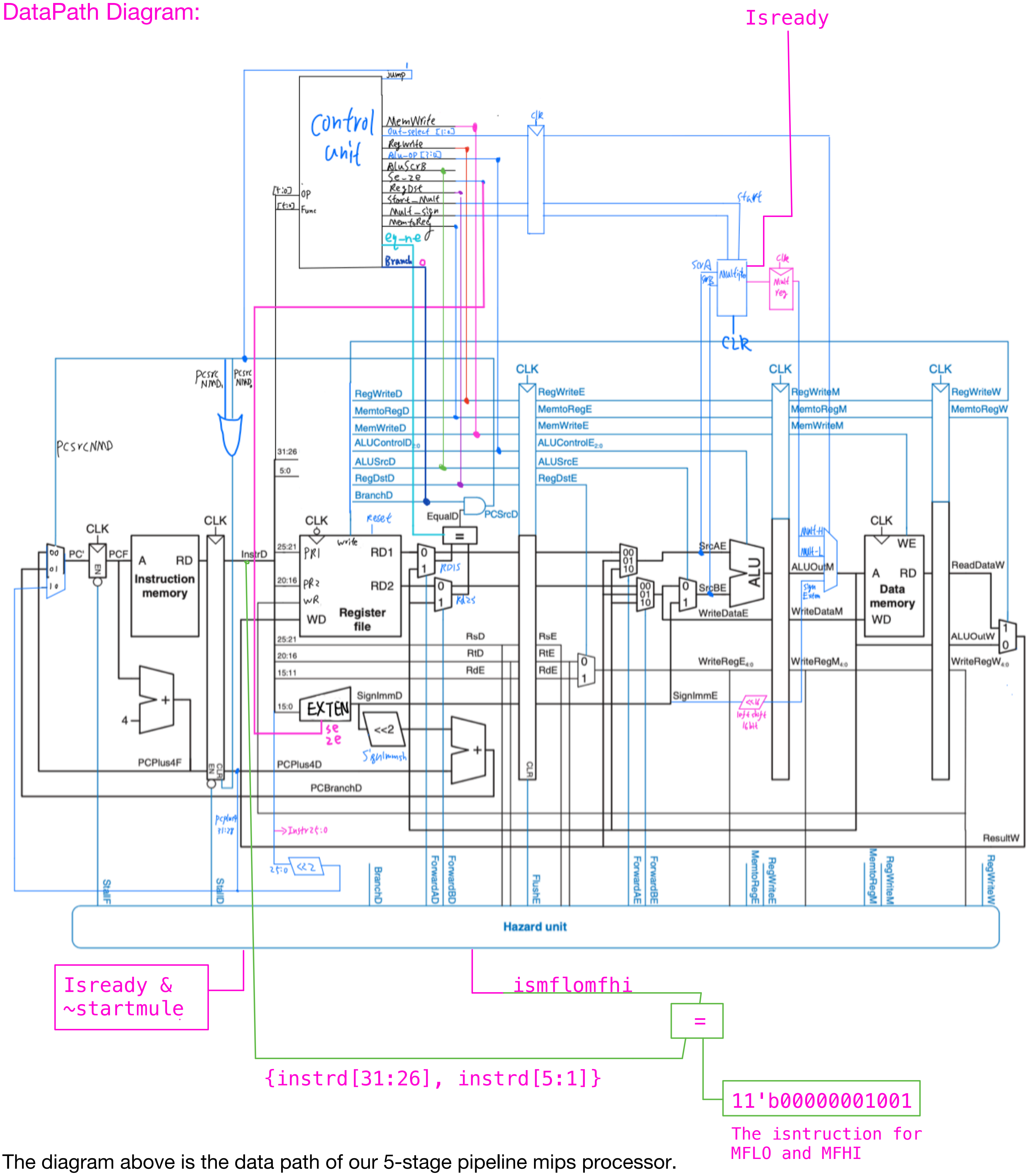
Name:Zhanglu Wang, Ke Ding, Changsheng SU
We spend 50 hours on this project.

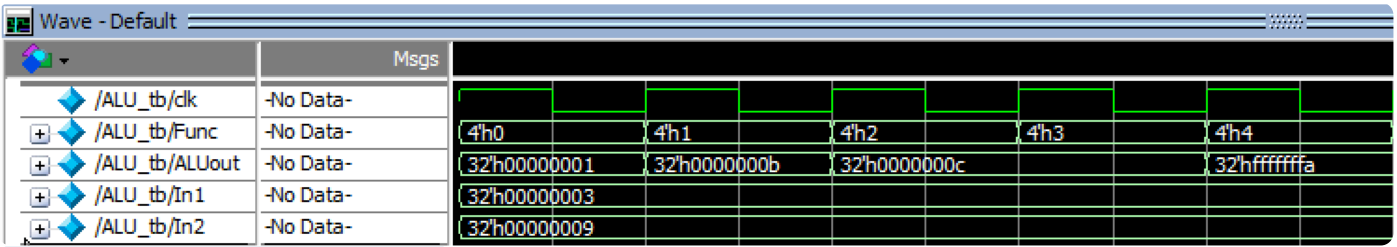
DataPath Diagram:

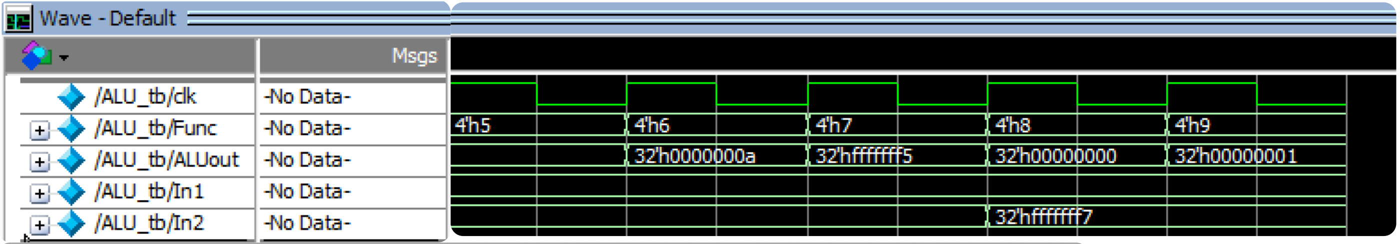


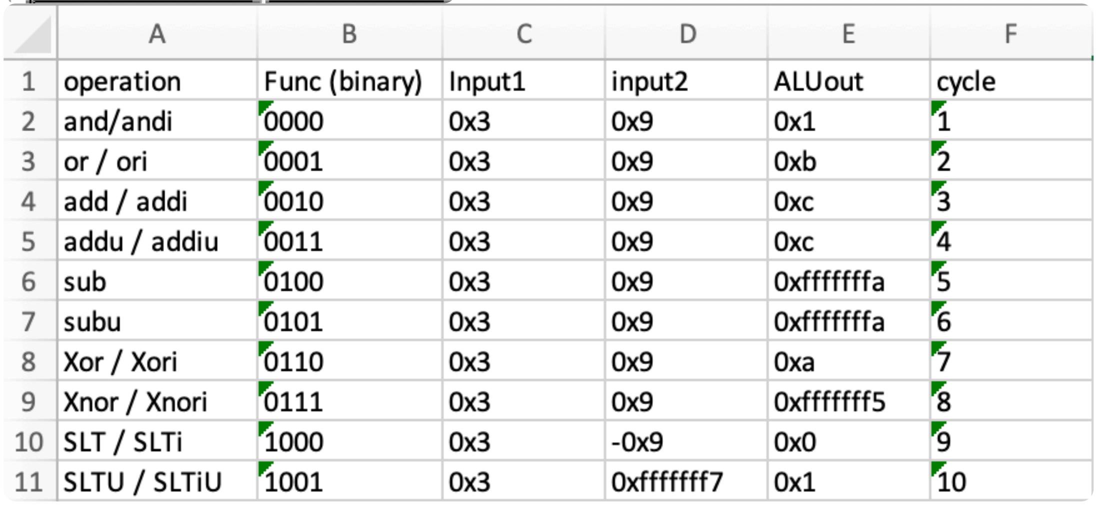
The diagram above is the data path of our 5-stage pipeline mips processor.

```
memfile.dat is for question 1
instrfile.dat is for question 2
Ruozhifile.dat is for extra credit question
multiple.dat is for multiplier
Hazardfile.dat is for hazard unit
```

ALU Module:



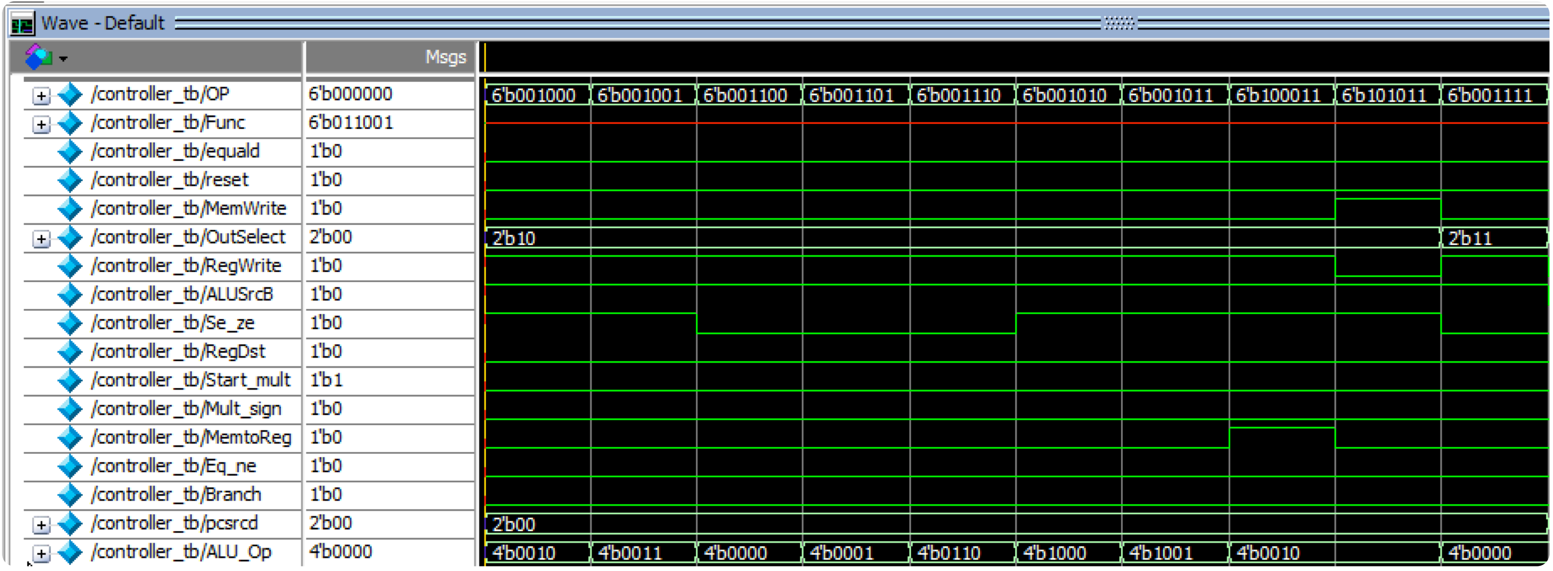


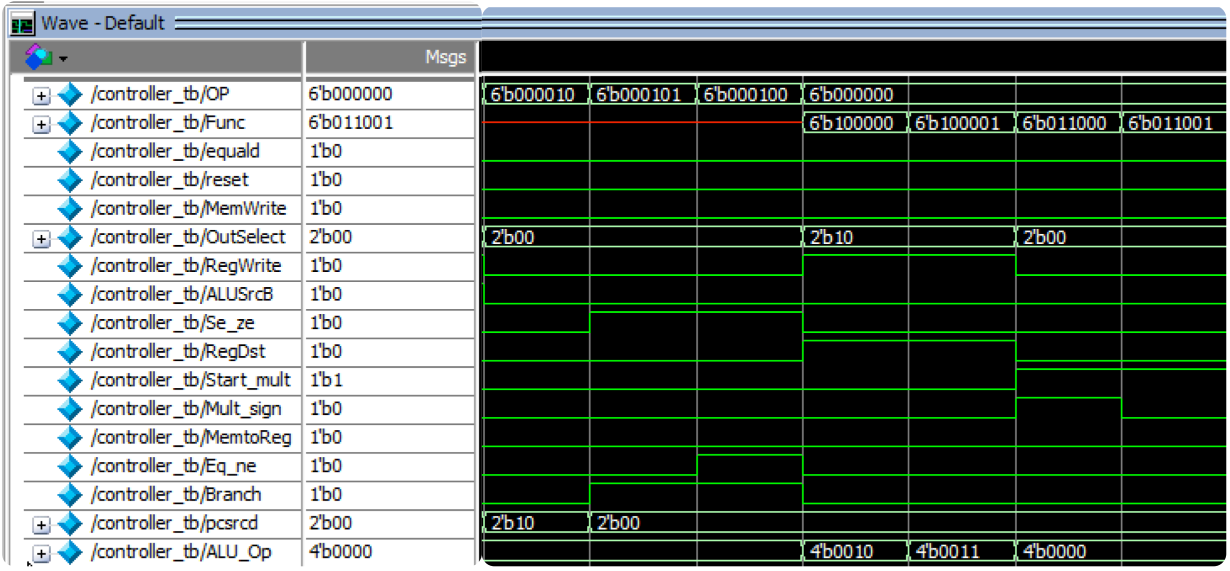


	A	B	C	D	E	F
1	operation	Func (binary)	Input1	input2	ALUout	cycle
2	and/andi	0000	0x3	0x9	0x1	1
3	or / ori	0001	0x3	0x9	0xb	2
4	add / addi	0010	0x3	0x9	0xc	3
5	addu / addiu	0011	0x3	0x9	0xc	4
6	sub	0100	0x3	0x9	0xfffffffa	5
7	subu	0101	0x3	0x9	0xfffffffa	6
8	Xor / Xori	0110	0x3	0x9	0xa	7
9	Xnor / Xnori	0111	0x3	0x9	0xfffffff5	8
10	SLT / SLTi	1000	0x3	-0x9	0x0	9
11	SLTU / SLTiU	1001	0x3	0xfffffff7	0x1	10

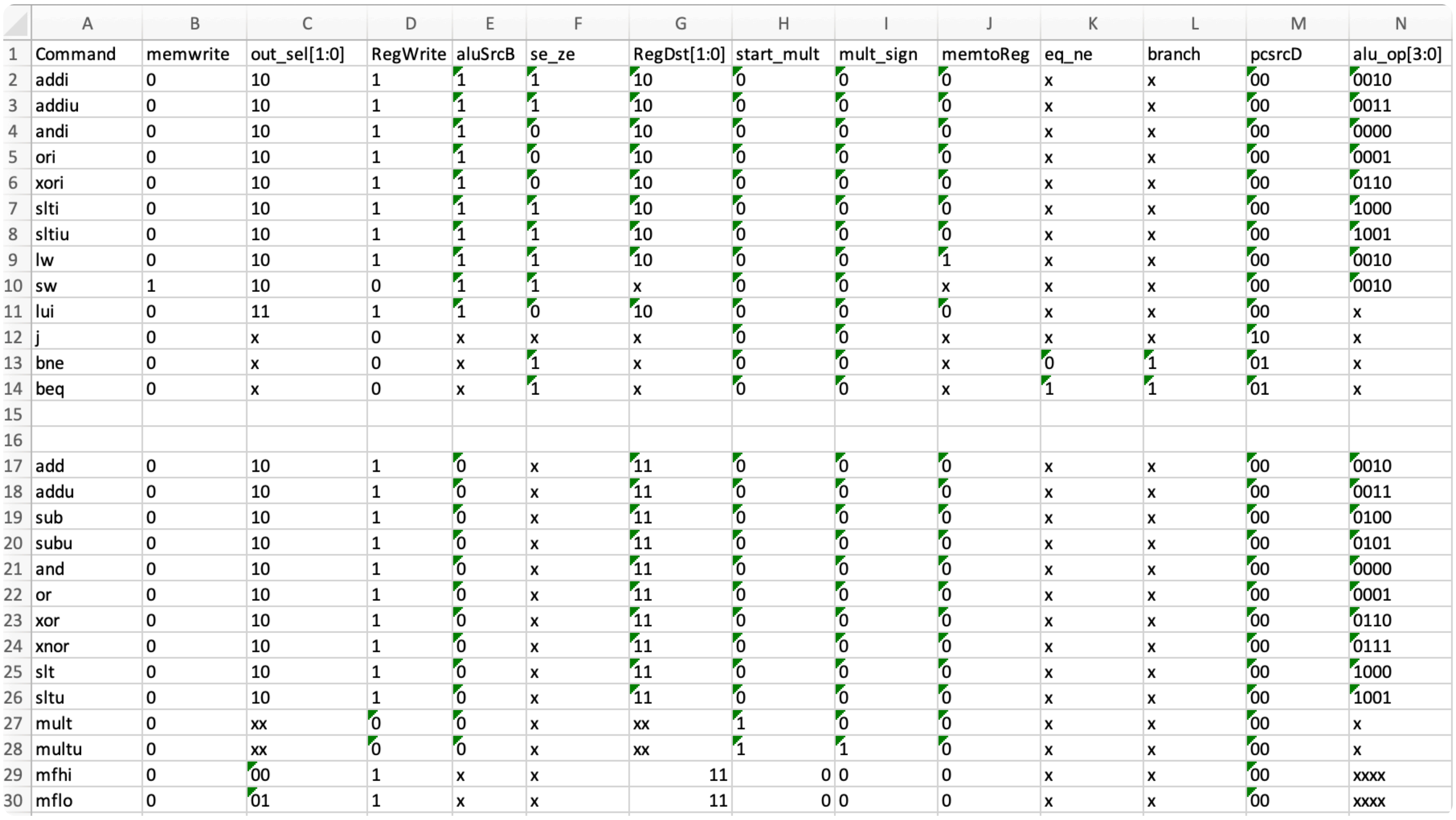
So based on our implementation of the func code, we can see that depending on the different Func Code we input with, we will see the ALU module is working perfectly with each function. For example, add operation 3+9 = 12 = 0xc, and SLT 3 and 9, 3< 9 and the result is 0. And if we use SLTU the result will be 1.

Controller:


Wave - Default
Msgs
/controller_tb/OP 6'b000000
/controller_tb/Func 6'b011001
/controller_tb/equald 1'b0
/controller_tb/reset 1'b0
/controller_tb/MemWrite 1'b0
/controller_tb/OutSelect 2'b00
/controller_tb/RegWrite 1'b0
/controller_tb/ALUSrcB 1'b0
/controller_tb/Se_ze 1'b0
/controller_tb/RegDst 1'b0
/controller_tb/Start_mult 1'b1
/controller_tb/Mult_sign 1'b0
/controller_tb/MemtoReg 1'b0
/controller_tb/Eq_ne 1'b0
/controller_tb/Branch 1'b0
/controller_tb/pcsrcd 2'b00
/controller_tb/ALU_Op 4'b0000
6'b001000
6'b001001
6'b001100
6'b001101
6'b001110
6'b001010
6'b001011
6'b100011
6'b101011
6'b001111
2'b10
2'b11
2'b00
4'b0010
4'b0011
4'b0000
4'b0001
4'b0110
4'b1000
4'b1001
4'b0010
4'b0000



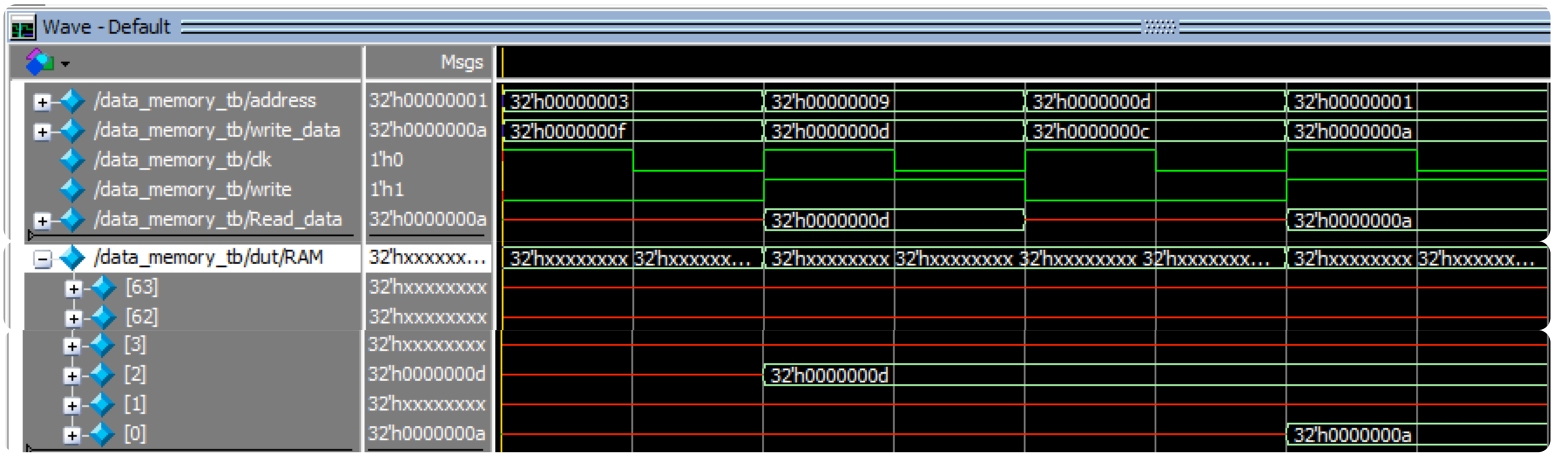
```
Msgs
/controller_tb/OP          6'b000000   6'b000010  6'b000101  6'b000100  6'b000000
/controller_tb/Func        6'b011001                                    6'b100000  6'b100001  6'b011000  6'b011001
/controller_tb/equald      1'b0
/controller_tb/reset       1'b0
/controller_tb/MemWrite    1'b0
/controller_tb/OutSelect   2'b00       2'b00                            2'b10                 2'b00
/controller_tb/RegWrite    1'b0
/controller_tb/ALUSrcB     1'b0
/controller_tb/Se_ze       1'b0
/controller_tb/RegDst      1'b0
/controller_tb/Start_mult  1'b1
/controller_tb/Mult_sign   1'b0
/controller_tb/MemtoReg    1'b0
/controller_tb/Eq_ne       1'b0
/controller_tb/Branch      1'b0
/controller_tb/pcsrcd      2'b00       2'b10      2'b00
/controller_tb/ALU_Op      4'b0000                                      4'b0010    4'b0011    4'b0000
```



	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	Command	memwrite	out_sel[1:0]	RegWrite	aluSrcB	se_ze	RegDst[1:0]	start_mult	mult_sign	memtoReg	eq_ne	branch	pcsrcD	alu_op[3:0]
2	addi	0	10	1	1	1	10	0	0	0	x	x	00	0010
3	addiu	0	10	1	1	1	10	0	0	0	x	x	00	0011
4	andi	0	10	1	1	0	10	0	0	0	x	x	00	0000
5	ori	0	10	1	1	0	10	0	0	0	x	x	00	0001
6	xori	0	10	1	1	0	10	0	0	0	x	x	00	0110
7	slti	0	10	1	1	1	10	0	0	0	x	x	00	1000
8	sltiu	0	10	1	1	1	10	0	0	0	x	x	00	1001
9	lw	0	10	1	1	1	10	0	0	1	x	x	00	0010
10	sw	1	10	0	1	1	x	0	0	x	x	x	00	0010
11	lui	0	11	1	1	0	10	0	0	0	x	x	00	x
12	j	0	x	0	x	x	x	0	0	x	x	x	10	x
13	bne	0	x	0	x	1	x	0	0	x	0	1	01	x
14	beq	0	x	0	x	1	x	0	0	x	1	1	01	x
15														
16														
17	add	0	10	1	0	x	11	0	0	0	x	x	00	0010
18	addu	0	10	1	0	x	11	0	0	0	x	x	00	0011
19	sub	0	10	1	0	x	11	0	0	0	x	x	00	0100
20	subu	0	10	1	0	x	11	0	0	0	x	x	00	0101
21	and	0	10	1	0	x	11	0	0	0	x	x	00	0000
22	or	0	10	1	0	x	11	0	0	0	x	x	00	0001
23	xor	0	10	1	0	x	11	0	0	0	x	x	00	0110
24	xnor	0	10	1	0	x	11	0	0	0	x	x	00	0111
25	slt	0	10	1	0	x	11	0	0	0	x	x	00	1000
26	sltu	0	10	1	0	x	11	0	0	0	x	x	00	1001
27	mult	0	xx	0	0	x	xx	1	0	0	x	x	00	x
28	multu	0	xx	0	0	x	xx	1	1	0	x	x	00	x
29	mfhi	0	00	1	x	x	11	0	0	0	x	x	00	xxxx
30	mflo	0	01	1	x	x	11	0	0	0	x	x	00	xxxx

As the waveform shown above, it is a decoder that decode the instruction code and operation code to each port, and corresponding to our decoding table, the values are correct.

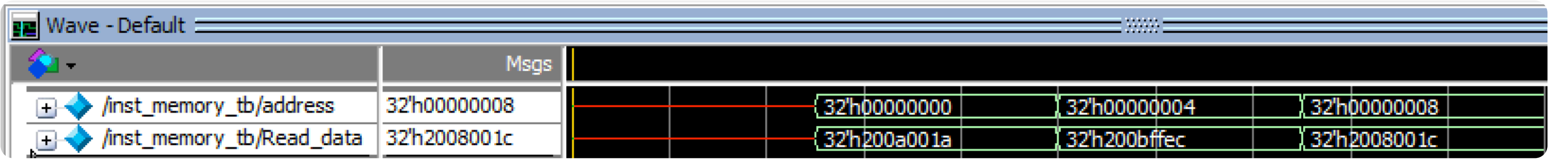
Data_Memory Module



```
Msgs
/data_memory_tb/address     32'h00000001  32'h00000003  32'h00000009  32'h0000000d  32'h00000001
/data_memory_tb/write_data  32'h0000000a  32'h0000000f  32'h0000000d  32'h0000000c  32'h0000000a
/data_memory_tb/clk         1'h0
/data_memory_tb/write       1'h1
/data_memory_tb/Read_data   32'h0000000a                32'h0000000d                32'h0000000a
/data_memory_tb/dut/RAM     32'hxxxxxxx...  32'hxxxxxxxx 32'hxxxxxx...  32'hxxxxxxxx 32'hxxxxxxxx 32'hxxxxxxxx 32'hxxxxxxx...  32'hxxxxxxxx 32'hxxxxxx...
  [63]                      32'hxxxxxxxx
  [62]                      32'hxxxxxxxx
  [3]                       32'hxxxxxxxx
  [2]                       32'h0000000d                32'h0000000d
  [1]                       32'hxxxxxxxx
  [0]                       32'h0000000a                                            32'h0000000a
```

Data memory is used for LW and SW operation, which will write or load data into the address we specified and the 'write' signal should be enabled in order to write in data. So we can see in the wave form, whenever the 'write' signal is enabled, the data will be stored in data memory. For the address, we ignore the last two bit of the address we give. For example, above we assign the address as 32'b 1001, so the address corresponding in RAM is 32'b 10 witch is address 2.

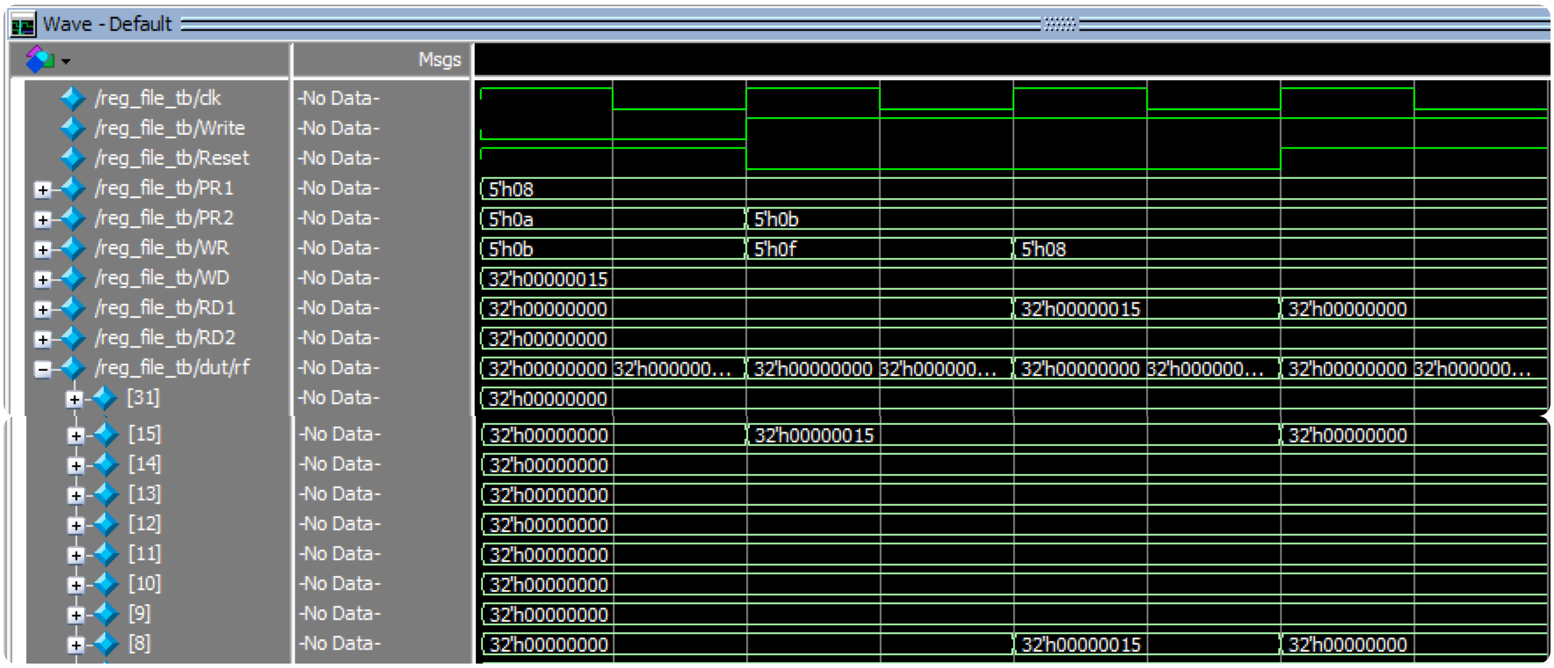
Instruction memory module:



For the instruction memory module, we read these three instruction,
addi t2 zero 26;
addi t3 zero -20;
addi, $t0, $0, 28;

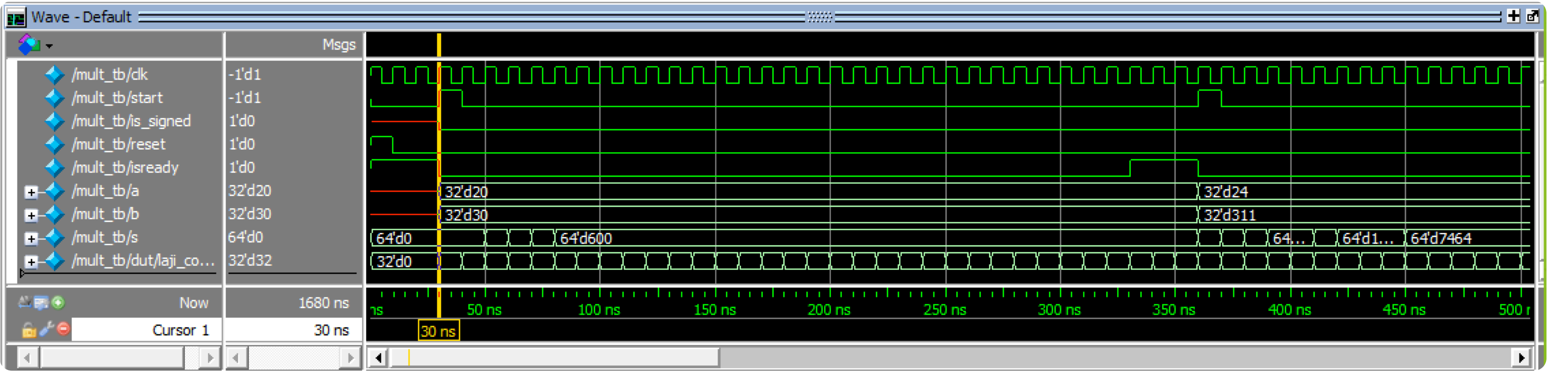
and the corresponding hex code are:
200a001a
200bffec
2008001c
so we can see the read_data was matched our instruction correctly. So the module works perfectly.

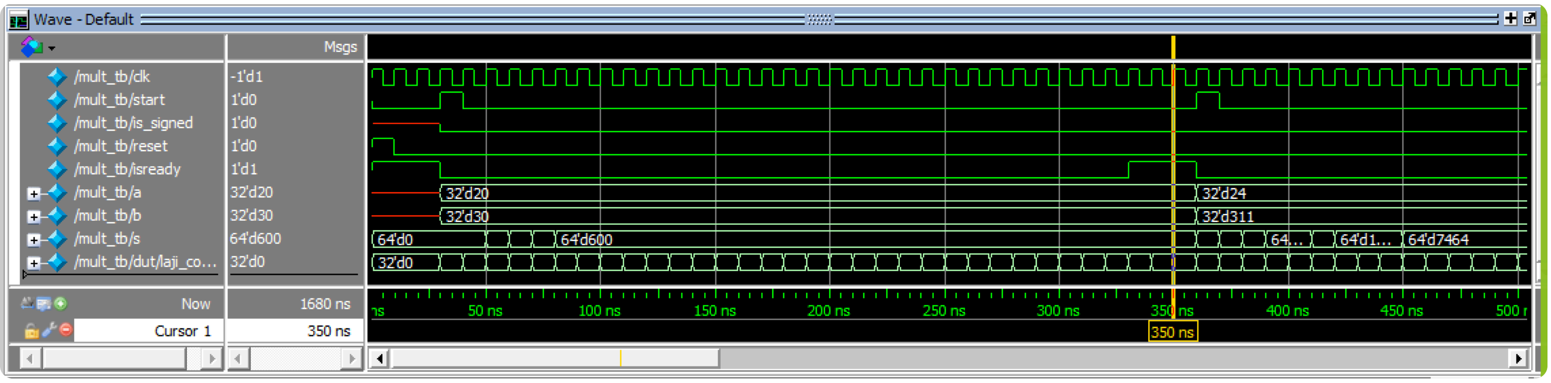
Reg file module:



For the reg_file module, we can see there first set the Writdata (WD) to 0x15, and only if the Wirte signal is enabled we can store the data into the register file (rf) in the address we specified as 0xb which is 15 in the rf. Same for the later operations. So our Reg-File register also works correctly.

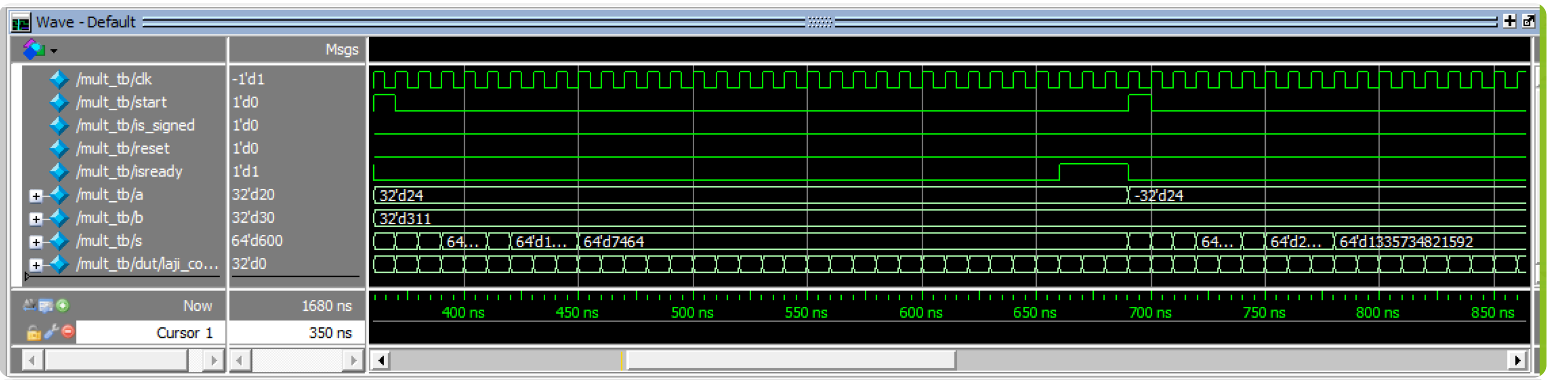
Multiplier Module:

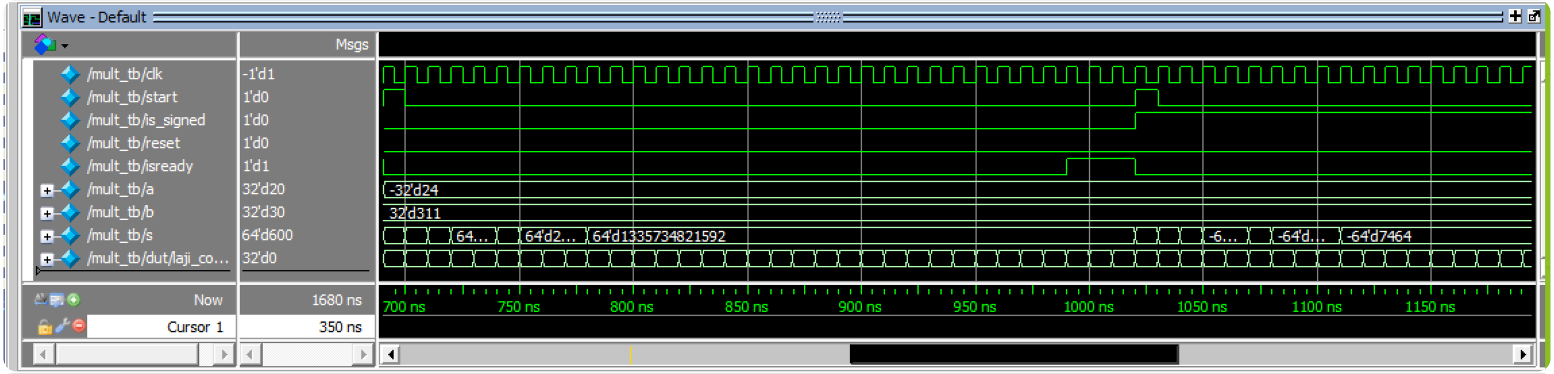


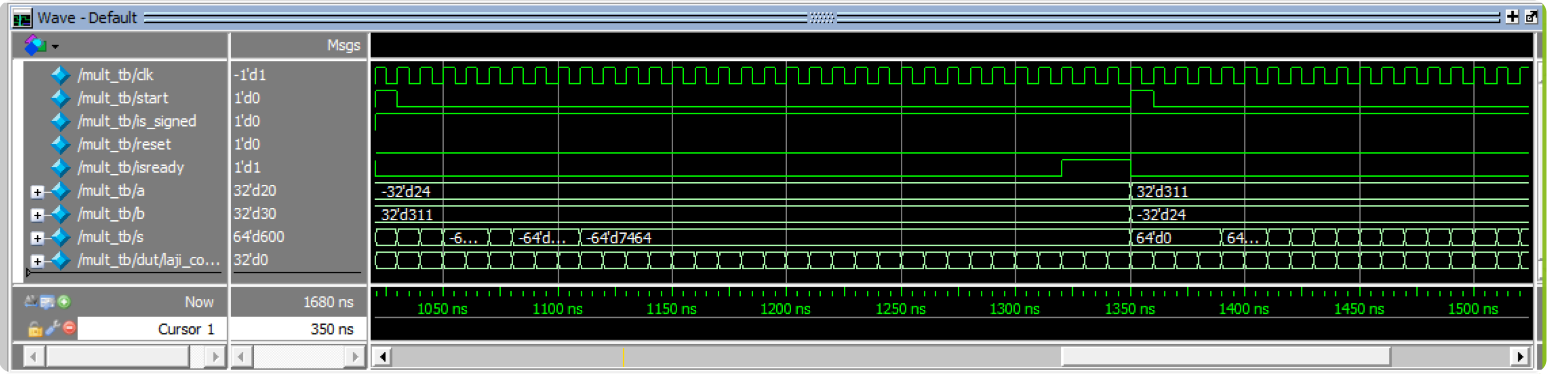


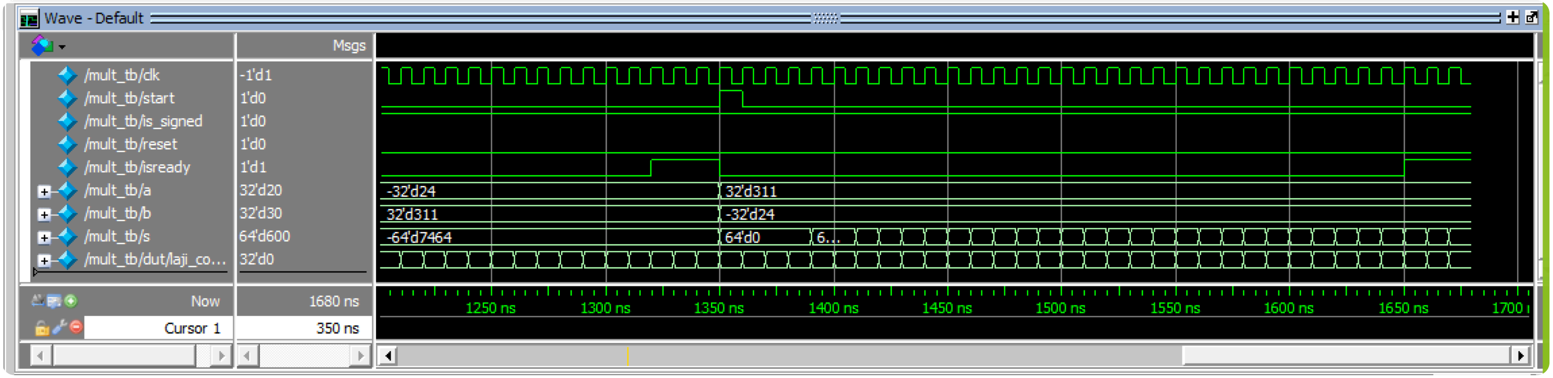
For the multiplier module, from the wave form, we can see that our multiplier was finish with the calculation when the laji_counter become to 0, and therefore our multiplier is using 32 cycles to calculate out the result. Also in order to do the 'mult' 'mflo' and 'mfhi' command sequently, we assigned a 'isready' signal to hazard unit for stalling.

Following are more operations we test and the result are correctly. (numbers shows in decimal)









Hazard unit module:

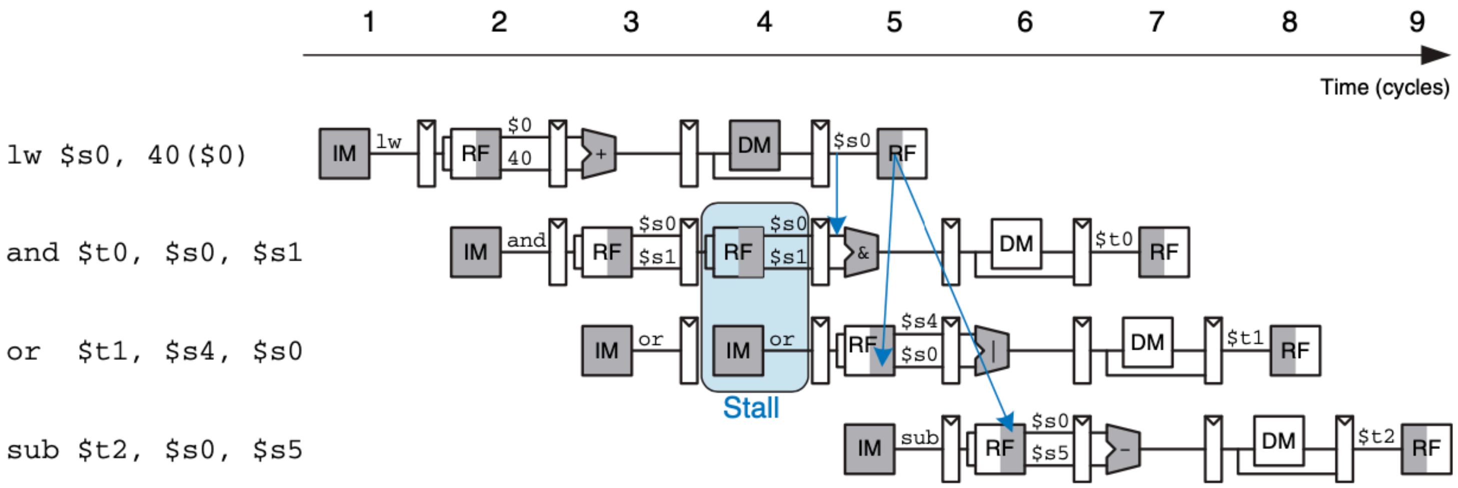
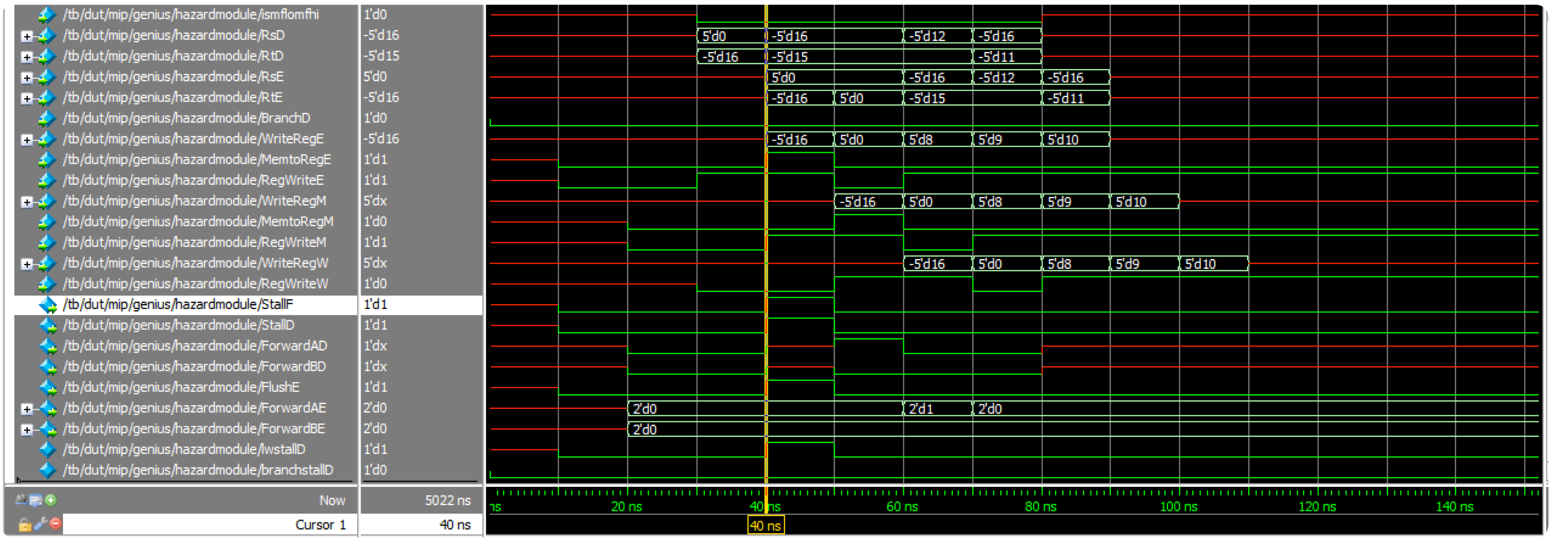


Figure 7.52 Abstract pipeline diagram illustrating stall to solve hazards



/tb/dut/mip/genius/hazardmodule/ismflomfhi	1'd0
/tb/dut/mip/genius/hazardmodule/RsD	-5'd16
/tb/dut/mip/genius/hazardmodule/RtD	-5'd15
/tb/dut/mip/genius/hazardmodule/RsE	5'd0
/tb/dut/mip/genius/hazardmodule/RtE	-5'd16
/tb/dut/mip/genius/hazardmodule/BranchD	1'd0
/tb/dut/mip/genius/hazardmodule/WriteRegE	-5'd16
/tb/dut/mip/genius/hazardmodule/MemtoRegE	1'd1
/tb/dut/mip/genius/hazardmodule/RegWriteE	1'd1
/tb/dut/mip/genius/hazardmodule/WriteRegM	5'dx
/tb/dut/mip/genius/hazardmodule/MemtoRegM	1'd0
/tb/dut/mip/genius/hazardmodule/RegWriteM	1'd1
/tb/dut/mip/genius/hazardmodule/WriteRegW	5'dx
/tb/dut/mip/genius/hazardmodule/RegWriteW	1'd0
/tb/dut/mip/genius/hazardmodule/StallF	1'd1
/tb/dut/mip/genius/hazardmodule/StallD	1'd1
/tb/dut/mip/genius/hazardmodule/ForwardAD	1'dx
/tb/dut/mip/genius/hazardmodule/ForwardBD	1'dx
/tb/dut/mip/genius/hazardmodule/FlushE	1'd1
/tb/dut/mip/genius/hazardmodule/ForwardAE	2'd0
/tb/dut/mip/genius/hazardmodule/ForwardBE	2'd0
/tb/dut/mip/genius/hazardmodule/lwstallD	1'd1
/tb/dut/mip/genius/hazardmodule/branchstallD	1'd0
Now	5022 ns
Cursor 1	40 ns

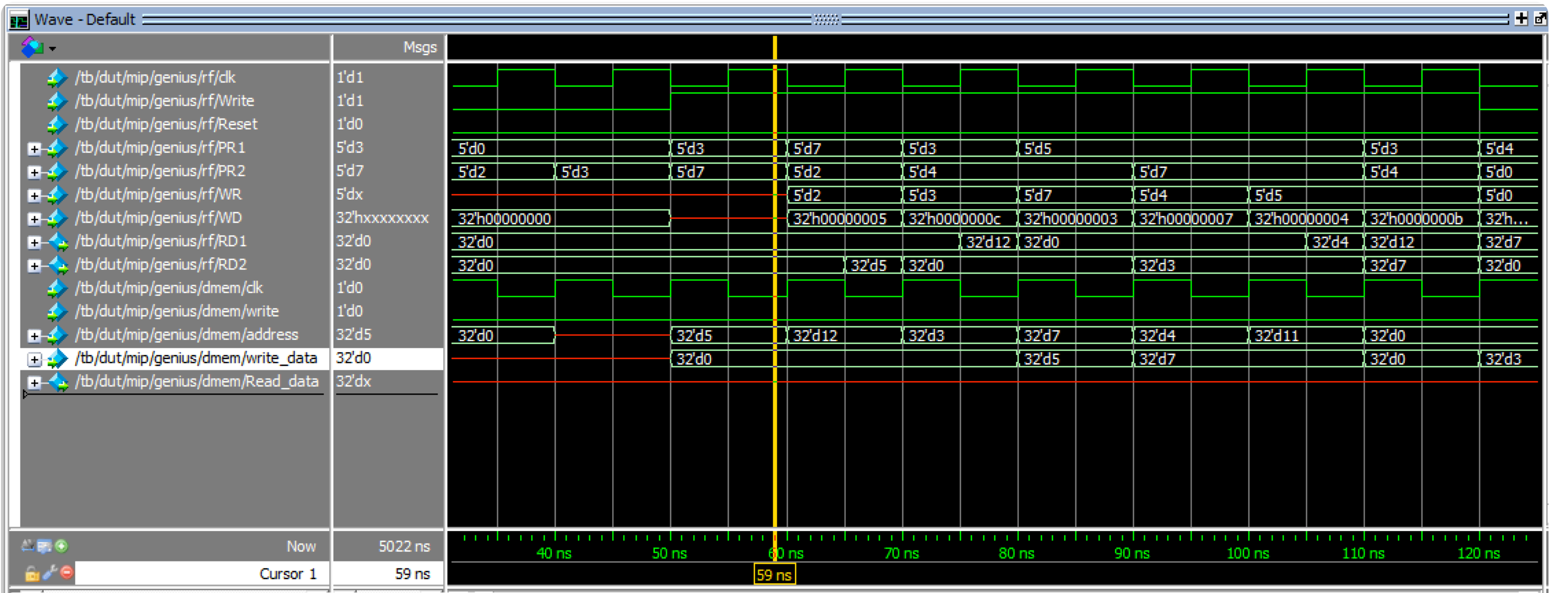
We test our hazard module by doing the above command, and we can see the stallF and stallD happened when we want to use the result from LW instruction right after LW instruction. (the hazardfile.dat in our files is the program we used here).

And the forward happened in the write stage, and forward the data directly to the decode stage. We can see in 50ns the forward AD was enabled, because in here we can see in the graph above it forward the data to the or and sub instruction.

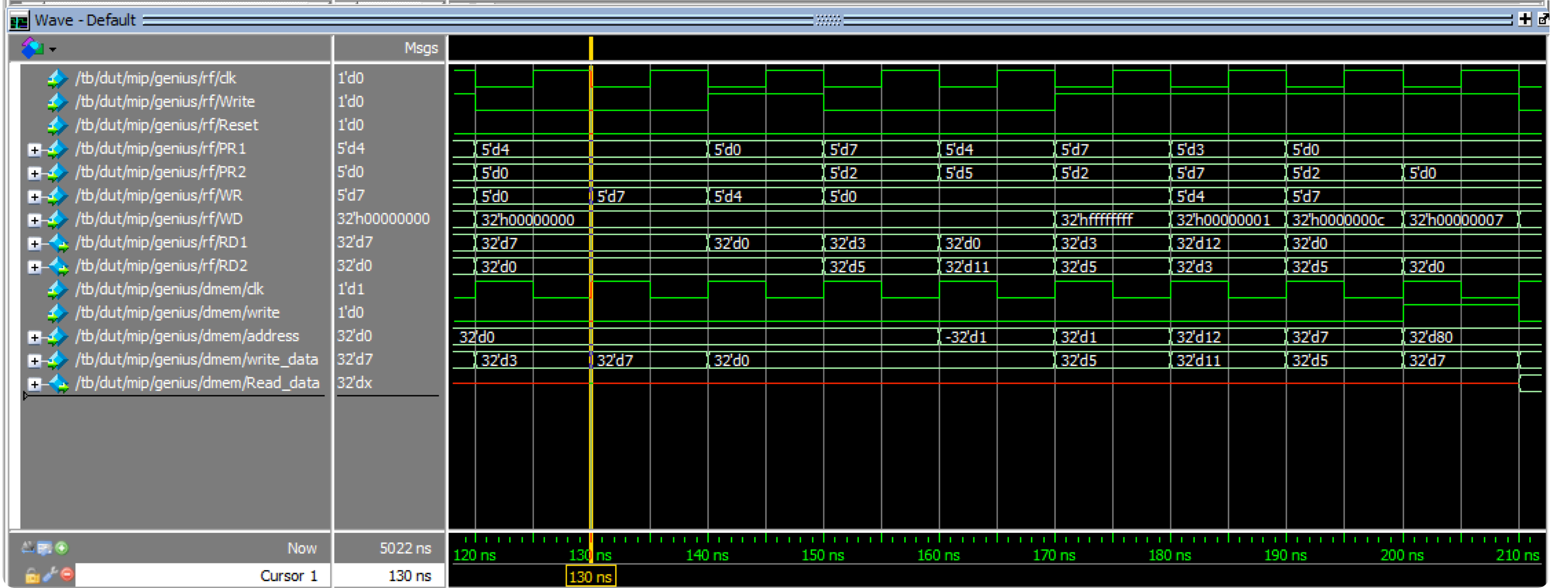
Final Test: (for question 1)

```
#        Assembly            Description            Address   Machine
main:    addi $2, $0, 5      # initialize $2 = 5    0         20020005          20020005
         addi $3, $0, 12     # initialize $3 = 12   4         2003000c          2003000c
         addi $7, $3, −9     # initialize $7 = 3    8         2067fff7          2067fff7
         or   $4, $7, $2     # $4 = (3 OR 5) = 7    c         00e22025          00e22025
         and  $5, $3, $4     # $5 = (12 AND 7) = 4  10        00642824          00642824
         add  $5, $5, $4     # $5 = 4 + 7 = 11      14        00a42820          00a42820
         beq  $5, $7, end    # shouldn't be taken   18        10a7000a          10a7000a
         slt  $4, $3, $4     # $4 = 12 < 7 = 0      1c        0064202a          0064202a
         beq  $4, $0, around # should be taken      20        10800001          10800001
         addi $5, $0, 0      # shouldn't happen     24        20050000          20050000
around:  slt  $4, $7, $2     # $4 = 3 < 5 = 1       28        00e2202a          00e2202a
         add  $7, $4, $5     # $7 = 1 + 11 = 12     2c        00853820          00853820
         sub  $7, $7, $2     # $7 = 12 − 5 = 7      30        00e23822          00e23822
         sw   $7, 68($3)     # [80] = 7             34        ac670044          ac670044
         lw   $2, 80($0)     # $2 = [80] = 7        38        8c020050          8c020050
         j    end            # should be taken      3c        08000011          08000011
         addi $2, $0, 1      # shouldn't happen     40        20020001          20020001
end:     sw   $2, 84($0)     # write mem[84] = 7    44        ac020054          ac020054

         MULT $v0 $a3                                         00470018
         MFLO $s1                                             00008812
         MFHI $s0                                             00008010
         LUI $s2 0x0029                                       3C120029
```

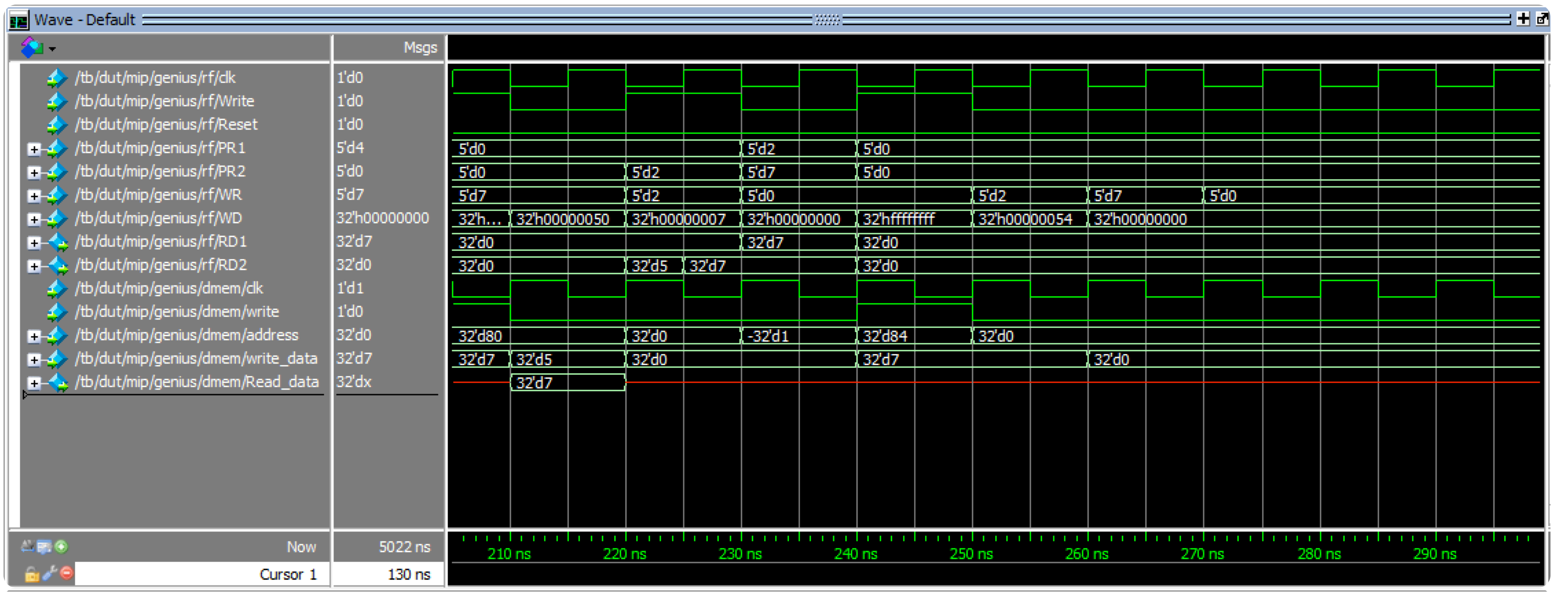
Above is our final test program and the wave form is shown in below:

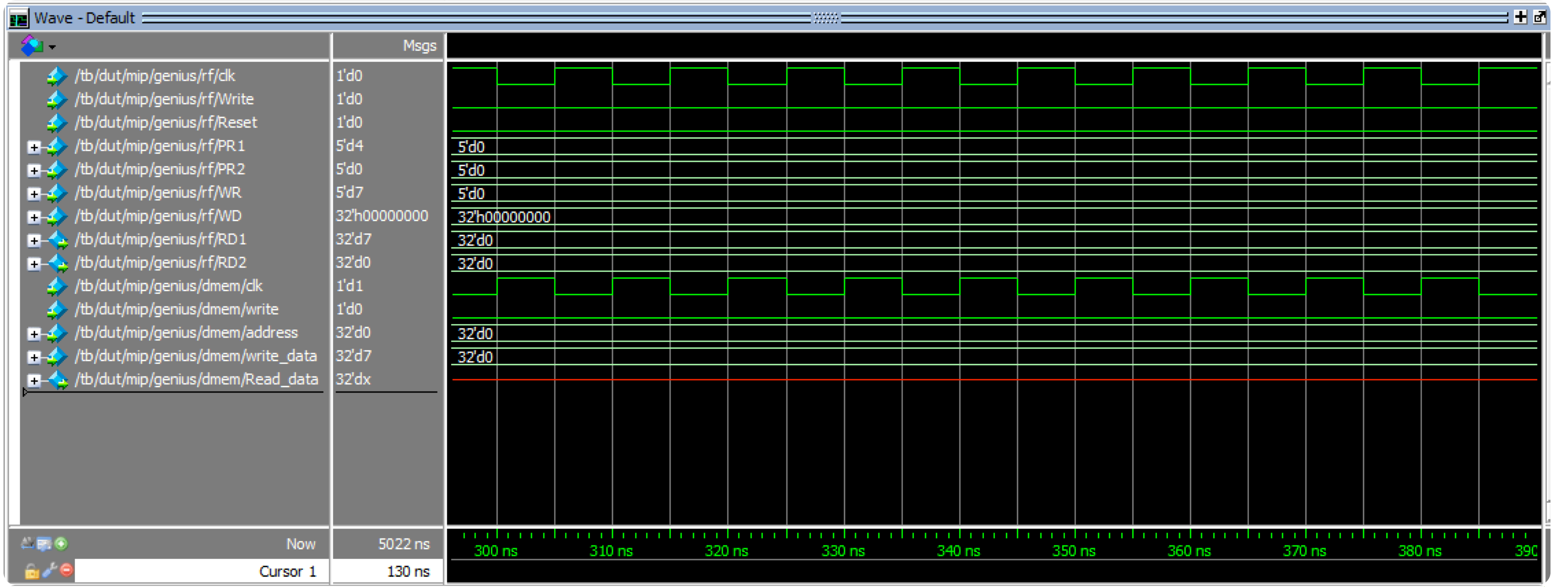

Wave - Default
Msgs
40 ns
50 ns
60 ns
70 ns
80 ns
90 ns
100 ns
110 ns
120 ns
59 ns
5'd0 5'd3 5'd7 5'd3 5'd5 5'd3 5'd4
5'd2 5'd3 5'd7 5'd2 5'd4 5'd7 5'd4 5'd0
5'd2 5'd3 5'd7 5'd4 5'd5 5'd0
32'h00000000 32'h00000005 32'h0000000c 32'h00000003 32'h00000007 32'h00000004 32'h0000000b 32'h...
32'd0 32'd12 32'd0 32'd4 32'd12 32'd7
32'd0 32'd5 32'd0 32'd3 32'd7 32'd0
32'd0 32'd5 32'd12 32'd3 32'd7 32'd4 32'd11 32'd0
32'd0 32'd5 32'd7 32'd0 32'd3

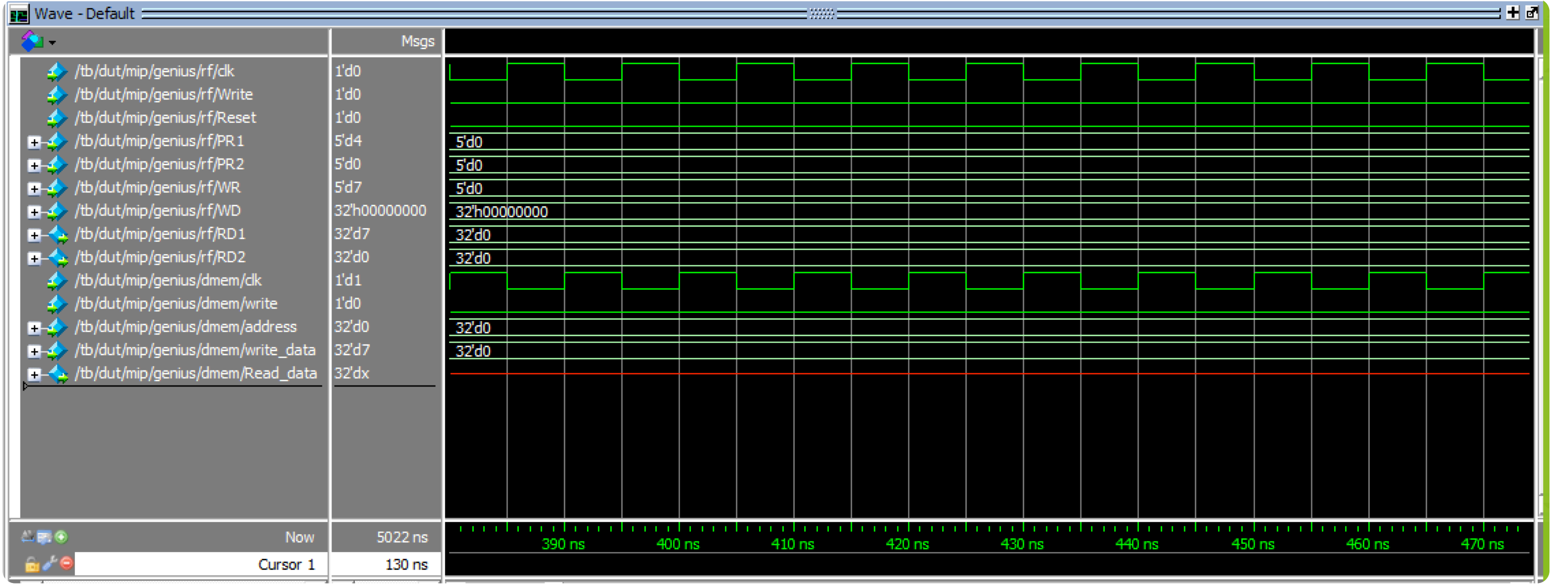
Signal	Msgs
/tb/dut/mip/genius/rf/clk	1'd1
/tb/dut/mip/genius/rf/Write	1'd1
/tb/dut/mip/genius/rf/Reset	1'd0
/tb/dut/mip/genius/rf/PR1	5'd3
/tb/dut/mip/genius/rf/PR2	5'd7
/tb/dut/mip/genius/rf/WR	5'dx
/tb/dut/mip/genius/rf/WD	32'hxxxxxxxx
/tb/dut/mip/genius/rf/RD1	32'd0
/tb/dut/mip/genius/rf/RD2	32'd0
/tb/dut/mip/genius/dmem/clk	1'd0
/tb/dut/mip/genius/dmem/write	1'd0
/tb/dut/mip/genius/dmem/address	32'd5
/tb/dut/mip/genius/dmem/write_data	32'd0
/tb/dut/mip/genius/dmem/Read_data	32'dx
Now	5022 ns
Cursor 1	59 ns


Wave - Default
Msgs
120 ns
130 ns
140 ns
150 ns
160 ns
170 ns
180 ns
190 ns
200 ns
210 ns
130 ns
5'd4 5'd0 5'd7 5'd4 5'd7 5'd3 5'd0
5'd0 5'd2 5'd5 5'd2 5'd7 5'd2 5'd0
5'd0 5'd7 5'd4 5'd0 5'd4 5'd7
32'h00000000 32'hffffffff 32'h00000001 32'h0000000c 32'h00000007
32'd7 32'd0 32'd3 32'd0 32'd3 32'd12 32'd0
32'd0 32'd5 32'd11 32'd5 32'd3 32'd5 32'd0
32'd0 -32'd1 32'd1 32'd12 32'd7 32'd80
32'd3 32'd7 32'd0 32'd5 32'd11 32'd5 32'd7

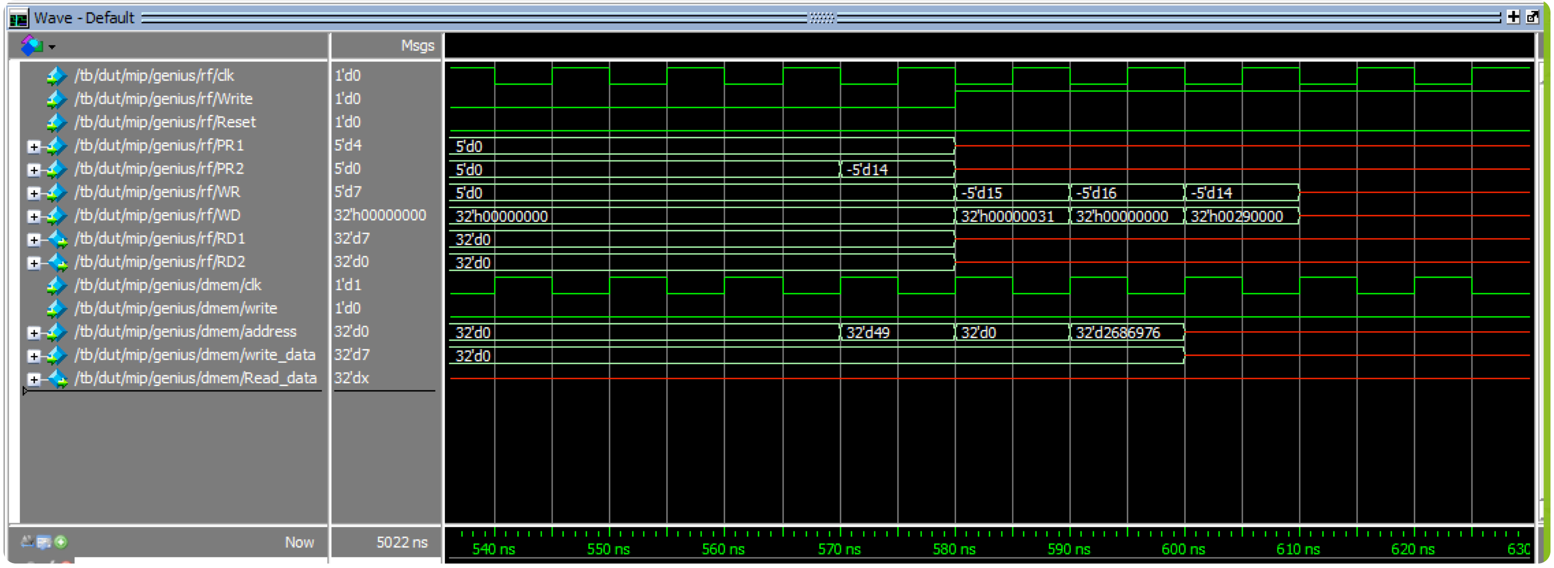
Signal	Msgs
/tb/dut/mip/genius/rf/clk	1'd0
/tb/dut/mip/genius/rf/Write	1'd0
/tb/dut/mip/genius/rf/Reset	1'd0
/tb/dut/mip/genius/rf/PR1	5'd4
/tb/dut/mip/genius/rf/PR2	5'd0
/tb/dut/mip/genius/rf/WR	5'd7
/tb/dut/mip/genius/rf/WD	32'h00000000
/tb/dut/mip/genius/rf/RD1	32'd7
/tb/dut/mip/genius/rf/RD2	32'd0
/tb/dut/mip/genius/dmem/clk	1'd1
/tb/dut/mip/genius/dmem/write	1'd0
/tb/dut/mip/genius/dmem/address	32'd0
/tb/dut/mip/genius/dmem/write_data	32'd7
/tb/dut/mip/genius/dmem/Read_data	32'dx
Now	5022 ns
Cursor 1	130 ns


Wave - Default
Msgs
/tb/dut/mip/genius/rf/clk 1'd0
/tb/dut/mip/genius/rf/Write 1'd0
/tb/dut/mip/genius/rf/Reset 1'd0
/tb/dut/mip/genius/rf/PR1 5'd4
/tb/dut/mip/genius/rf/PR2 5'd0
/tb/dut/mip/genius/rf/WR 5'd7
/tb/dut/mip/genius/rf/WD 32'h00000000
/tb/dut/mip/genius/rf/RD1 32'd7
/tb/dut/mip/genius/rf/RD2 32'd0
/tb/dut/mip/genius/dmem/clk 1'd1
/tb/dut/mip/genius/dmem/write 1'd0
/tb/dut/mip/genius/dmem/address 32'd0
/tb/dut/mip/genius/dmem/write_data 32'd7
/tb/dut/mip/genius/dmem/Read_data 32'dx
5'd0 5'd2 5'd0
5'd0 5'd2 5'd7 5'd0
5'd7 5'd2 5'd0 5'd2 5'd7 5'd0
32'h... 32'h00000050 32'h00000007 32'h00000000 32'hffffffff 32'h00000054 32'h00000000
32'd0 32'd7 32'd0
32'd0 32'd5 32'd7 32'd0
32'd80 32'd0 -32'd1 32'd84 32'd0
32'd7 32'd5 32'd0 32'd7 32'd0
32'd7
Now 5022 ns
Cursor 1 130 ns
210 ns
220 ns
230 ns
240 ns
250 ns
260 ns
270 ns
280 ns
290 ns


Wave - Default
Msgs
/tb/dut/mip/genius/rf/clk 1'd0
/tb/dut/mip/genius/rf/Write 1'd0
/tb/dut/mip/genius/rf/Reset 1'd0
/tb/dut/mip/genius/rf/PR1 5'd4
/tb/dut/mip/genius/rf/PR2 5'd0
/tb/dut/mip/genius/rf/WR 5'd7
/tb/dut/mip/genius/rf/WD 32'h00000000
/tb/dut/mip/genius/rf/RD1 32'd7
/tb/dut/mip/genius/rf/RD2 32'd0
/tb/dut/mip/genius/dmem/clk 1'd1
/tb/dut/mip/genius/dmem/write 1'd0
/tb/dut/mip/genius/dmem/address 32'd0
/tb/dut/mip/genius/dmem/write_data 32'd7
/tb/dut/mip/genius/dmem/Read_data 32'dx
5'd0
5'd0
5'd0
32'h00000000
32'd0
32'd0
32'd0
32'd0
Now 5022 ns
Cursor 1 130 ns
300 ns
310 ns
320 ns
330 ns
340 ns
350 ns
360 ns
370 ns
380 ns
390


Wave - Default
Msgs
/tb/dut/mip/genius/rf/clk 1'd0
/tb/dut/mip/genius/rf/Write 1'd0
/tb/dut/mip/genius/rf/Reset 1'd0
/tb/dut/mip/genius/rf/PR1 5'd4
/tb/dut/mip/genius/rf/PR2 5'd0
/tb/dut/mip/genius/rf/WR 5'd7
/tb/dut/mip/genius/rf/WD 32'h00000000
/tb/dut/mip/genius/rf/RD1 32'd7
/tb/dut/mip/genius/rf/RD2 32'd0
/tb/dut/mip/genius/dmem/clk 1'd1
/tb/dut/mip/genius/dmem/write 1'd0
/tb/dut/mip/genius/dmem/address 32'd0
/tb/dut/mip/genius/dmem/write_data 32'd7
/tb/dut/mip/genius/dmem/Read_data 32'dx
5'd0
5'd0
5'd0
32'h00000000
32'd0
32'd0
32'd0
32'd0
Now 5022 ns
Cursor 1 130 ns
390 ns
400 ns
410 ns
420 ns
430 ns
440 ns
450 ns
460 ns
470 ns

Here is a 31 cycles stall for the mult instruction



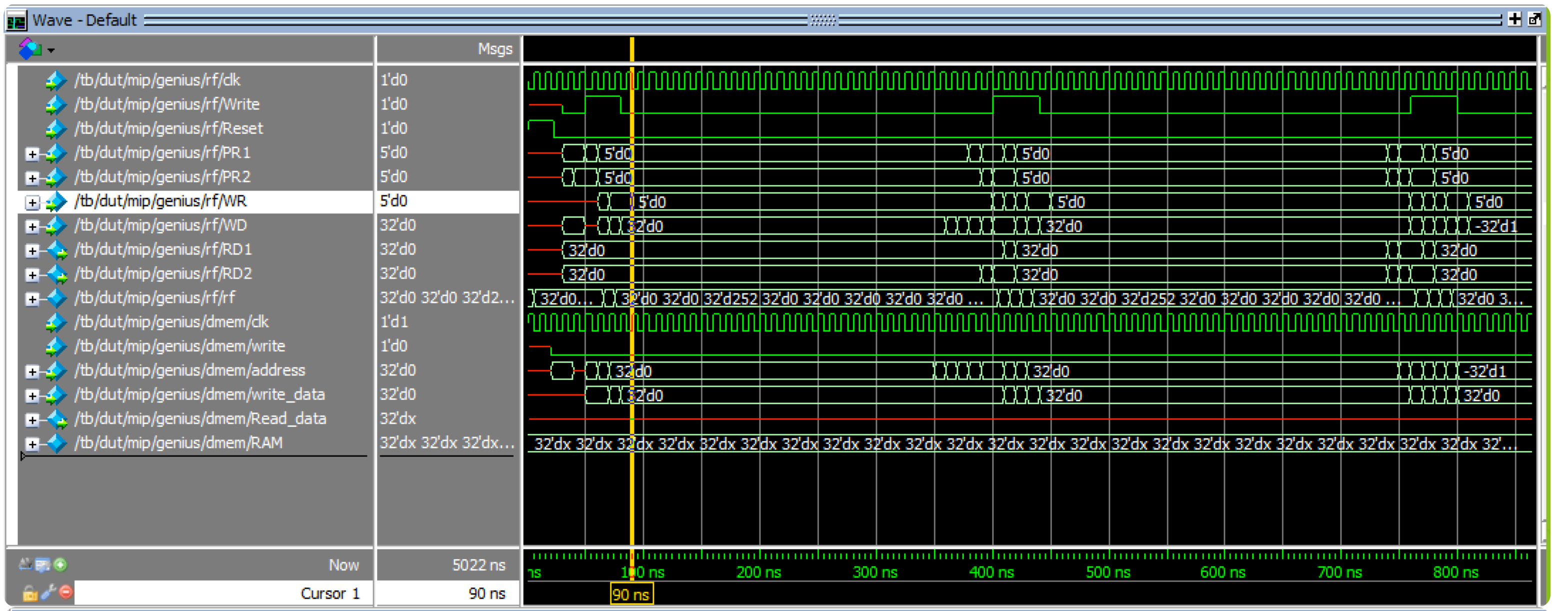
Wave - Default

	Msgs
/tb/dut/mip/genius/rf/clk	1'd0
/tb/dut/mip/genius/rf/Write	1'd0
/tb/dut/mip/genius/rf/Reset	1'd0
/tb/dut/mip/genius/rf/PR1	5'd4
/tb/dut/mip/genius/rf/PR2	5'd0
/tb/dut/mip/genius/rf/WR	5'd7
/tb/dut/mip/genius/rf/WD	32'h00000000
/tb/dut/mip/genius/rf/RD1	32'd7
/tb/dut/mip/genius/rf/RD2	32'd0
/tb/dut/mip/genius/dmem/clk	1'd1
/tb/dut/mip/genius/dmem/write	1'd0
/tb/dut/mip/genius/dmem/address	32'd0
/tb/dut/mip/genius/dmem/write_data	32'd7
/tb/dut/mip/genius/dmem/Read_data	32'dx
Now	5022 ns

For question 2:

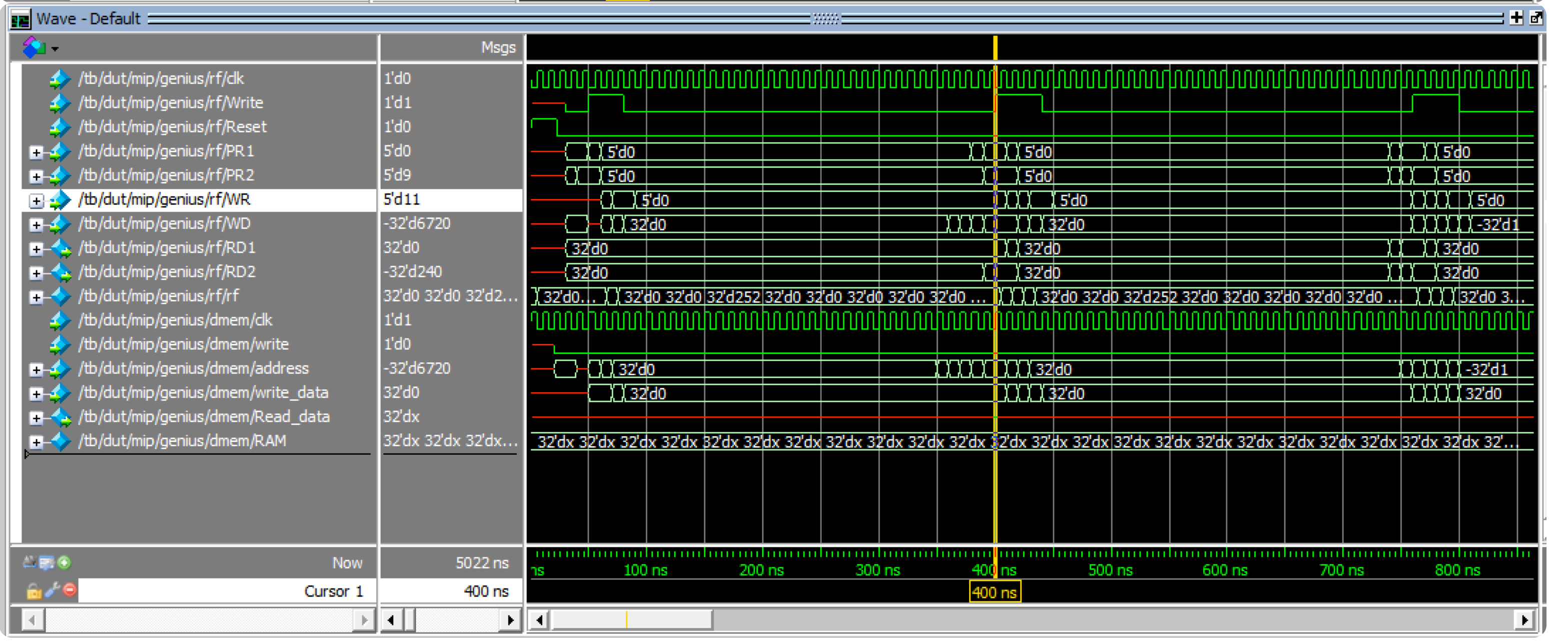
```
addi, $t0, $0, 28;       2008001c
addi, $t1, $0, -240;     2009FF10
mult, $t0, $t1;          01090018
mflo $t3;                00005812
add $t5,$t3,$0;          01606820

addi $t0, $0, 110;       2008006E
addi $t1, $0, 91;        2009005B
mult $t0, $t1;           01090018
mflo $t3;                00005812
add $t5,$t5,$t3;         01AB6820

addi $t0, $0, -67;       2008FFD7
addi $t1, $0, 88;        2009FF9B
mult $t0, $t1;           01090018
mflo $t3;                00005812
add $t5,$t5,$t3;         01AB6820

                         20080017
addi $t0, $0, -41;       20090096
addi $t1, $0, -101;      01090018
mult $t0, $t1;           00005812
mflo $t3;                01AB6820
add $t5,$t5,$t3

                         2008FFBD
addi $t0, $0, 23;        20090058
addi $t1, $0, 150;       01090018
mult $t0, $t1;           00005812
mflo $t3;                01AB6820
add $t5,$t5,$t3
```

this is the code we run for the dot product.
For our program, it take 184 cycles to calculate out the result.



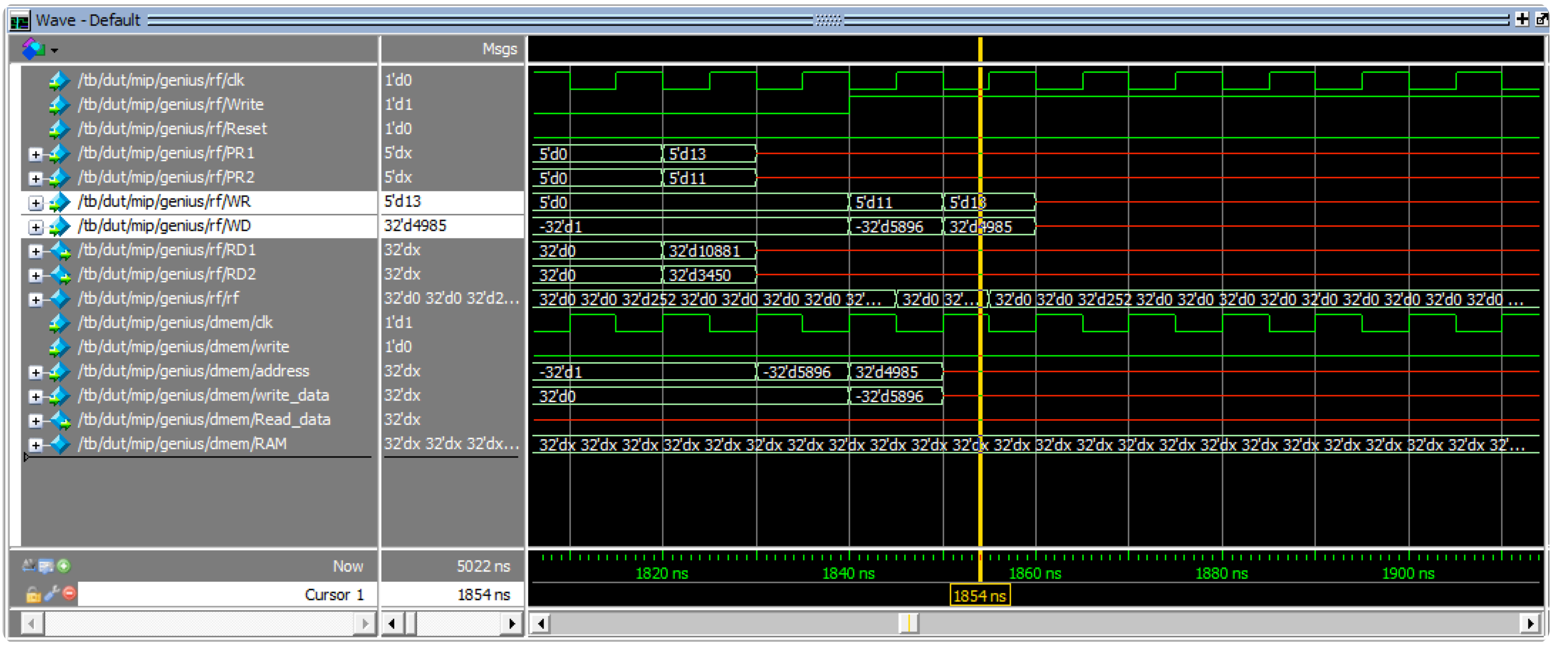
Wave - Default

	Msgs
/tb/dut/mip/genius/rf/clk	1'd0
/tb/dut/mip/genius/rf/Write	1'd0
/tb/dut/mip/genius/rf/Reset	1'd0
/tb/dut/mip/genius/rf/PR1	5'd0
/tb/dut/mip/genius/rf/PR2	5'd0
/tb/dut/mip/genius/rf/WR	5'd0
/tb/dut/mip/genius/rf/WD	32'd0
/tb/dut/mip/genius/rf/RD1	32'd0
/tb/dut/mip/genius/rf/RD2	32'd0
/tb/dut/mip/genius/rf/rf	32'd0 32'd0 32'd2...
/tb/dut/mip/genius/dmem/clk	1'd1
/tb/dut/mip/genius/dmem/write	1'd0
/tb/dut/mip/genius/dmem/address	32'd0
/tb/dut/mip/genius/dmem/write_data	32'd0
/tb/dut/mip/genius/dmem/Read_data	32'dx
/tb/dut/mip/genius/dmem/RAM	32'dx 32'dx 32'dx...
Now	5022 ns
Cursor 1	90 ns



Wave - Default

	Msgs
/tb/dut/mip/genius/rf/clk	1'd0
/tb/dut/mip/genius/rf/Write	1'd1
/tb/dut/mip/genius/rf/Reset	1'd0
/tb/dut/mip/genius/rf/PR1	5'd0
/tb/dut/mip/genius/rf/PR2	5'd9
/tb/dut/mip/genius/rf/WR	5'd11
/tb/dut/mip/genius/rf/WD	-32'd6720
/tb/dut/mip/genius/rf/RD1	32'd0
/tb/dut/mip/genius/rf/RD2	-32'd240
/tb/dut/mip/genius/rf/rf	32'd0 32'd0 32'd2...
/tb/dut/mip/genius/dmem/clk	1'd1
/tb/dut/mip/genius/dmem/write	1'd0
/tb/dut/mip/genius/dmem/address	-32'd6720
/tb/dut/mip/genius/dmem/write_data	32'd0
/tb/dut/mip/genius/dmem/Read_data	32'dx
/tb/dut/mip/genius/dmem/RAM	32'dx 32'dx 32'dx...
Now	5022 ns
Cursor 1	400 ns



Wave - Default

	Msgs
/tb/dut/mip/genius/rf/clk	1'd0
/tb/dut/mip/genius/rf/Write	1'd1
/tb/dut/mip/genius/rf/Reset	1'd0
/tb/dut/mip/genius/rf/PR1	5'dx
/tb/dut/mip/genius/rf/PR2	5'dx
/tb/dut/mip/genius/rf/WR	5'd13
/tb/dut/mip/genius/rf/WD	32'd4985
/tb/dut/mip/genius/rf/RD1	32'dx
/tb/dut/mip/genius/rf/RD2	32'dx
/tb/dut/mip/genius/rf/rf	32'd0 32'd0 32'd2...
/tb/dut/mip/genius/dmem/clk	1'd1
/tb/dut/mip/genius/dmem/write	1'd0
/tb/dut/mip/genius/dmem/address	32'dx
/tb/dut/mip/genius/dmem/write_data	32'dx
/tb/dut/mip/genius/dmem/Read_data	32'dx
/tb/dut/mip/genius/dmem/RAM	32'dx 32'dx 32'dx...
Now	5022 ns
Cursor 1	1854 ns

From these two figures we can find the our processor will stall 31 cycles for each multiplication.

And we got the result as 4985 witch is a correct answer.

Extra Credit:

For the extra credit, we implement the instruction in the following way

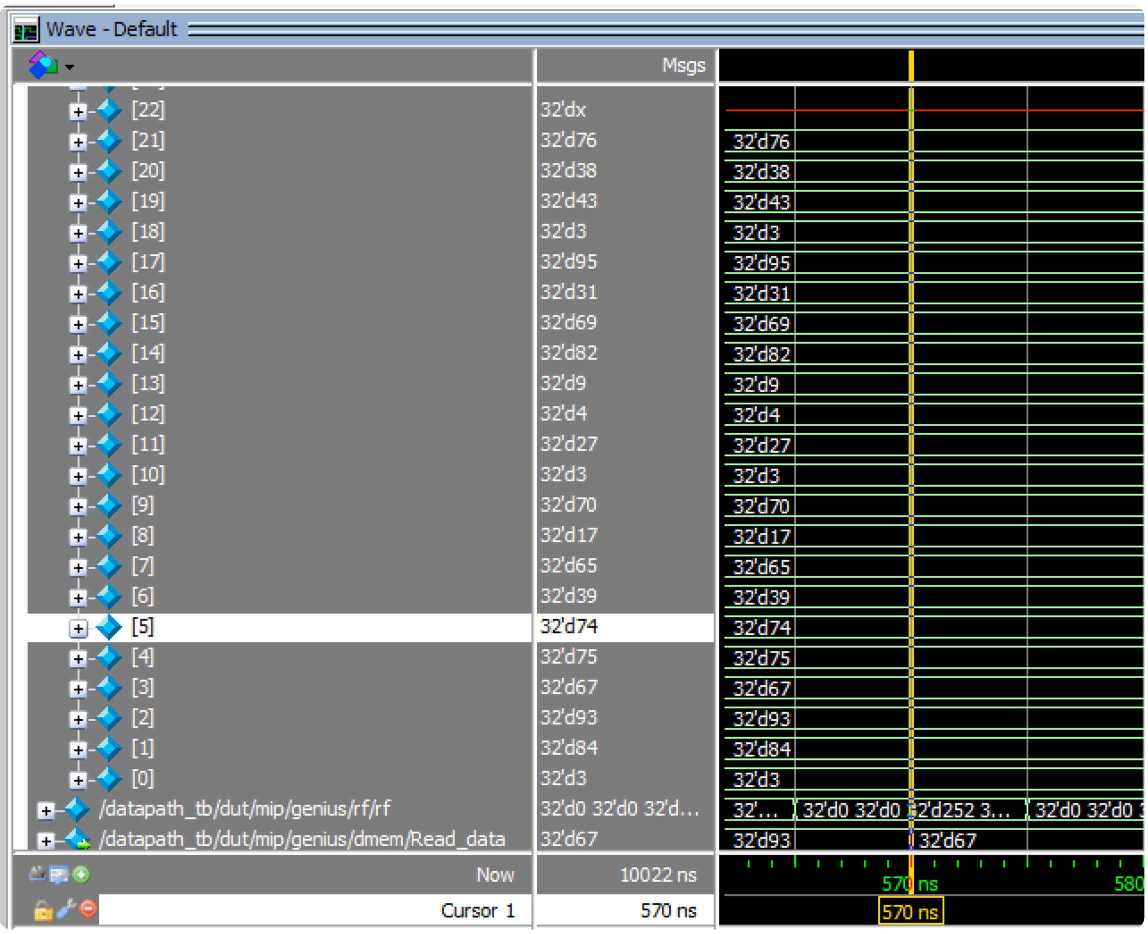
```
main:
        ori $t0, $zero, 3;
        sw $t0, 0($zero);
        ori $t0, $zero, 84;
        sw $t0, 4($zero);
        ori $t0, $zero, 93;
        sw $t0, 8($zero);
        ori $t0, $zero, 67;
        sw $t0, 12($zero);
        ori $t0, $zero, 75;
        sw $t0, 16($zero);
        ori $t0, $zero, 74;
        sw $t0, 20($zero);
        ori $t0, $zero, 39;
        sw $t0, 24($zero);
        ori $t0, $zero, 65;
        sw $t0, 28($zero);
        ori $t0, $zero, 17;
        sw $t0, 32($zero);
        ori $t0, $zero, 70;
        sw $t0, 36($zero);
        ori $t0, $zero, 3;
        sw $t0, 40($zero);

        ori $t0, $zero, 27;
        sw $t0, 44($zero);
        ori $t0, $zero, 4;
        sw $t0, 48($zero);
        ori $t0, $zero, 9;
        sw $t0, 52($zero);
        ori $t0, $zero, 82;
        sw $t0, 56($zero);
        ori $t0, $zero, 69;
        sw $t0, 60($zero);
        ori $t0, $zero, 31;
        sw $t0, 64($zero);
        ori $t0, $zero, 95;
        sw $t0, 68($zero);
        ori $t0, $zero, 3;
        sw $t0, 72($zero);
        ori $t0, $zero, 43;
        sw $t0, 76($zero);
        ori $t0, $zero, 38;
        sw $t0, 80($zero);
        ori $t0, $zero, 76;
        sw $t0, 84($zero);
```

```
These are store the array in the memory
x = [3, 84, 93, 67, 75, 74, 39, 65, 17, 70, 3];
y = [27, 4, 9, 82, 69, 31, 95, 3, 43, 38, 76];
x is stored in memory address 0 to 40;
y is stored in memory address 44 to 84;
```

// our program start here

```
        ori $t0, $zero, 0;
        ori $t1, $zero, 40;
        ori $t2, $zero, 44;

laji:   lw $t3, 0($t0);
        lw $t4, 0($t2);
        add $t5, $zero, $t3;
        add $t5, $t5, $t3;
        add $t5, $t5, $t3;
        add $t5, $t5, $t3;
        add $t5, $t5, $t3;
        add $t5, $t5, $t4;
        sw $t5, 4($t2);
        addi $t0, $t0, 4;
        addi $t2, $t2, 4;
        bne $t0, $t1, laji
```

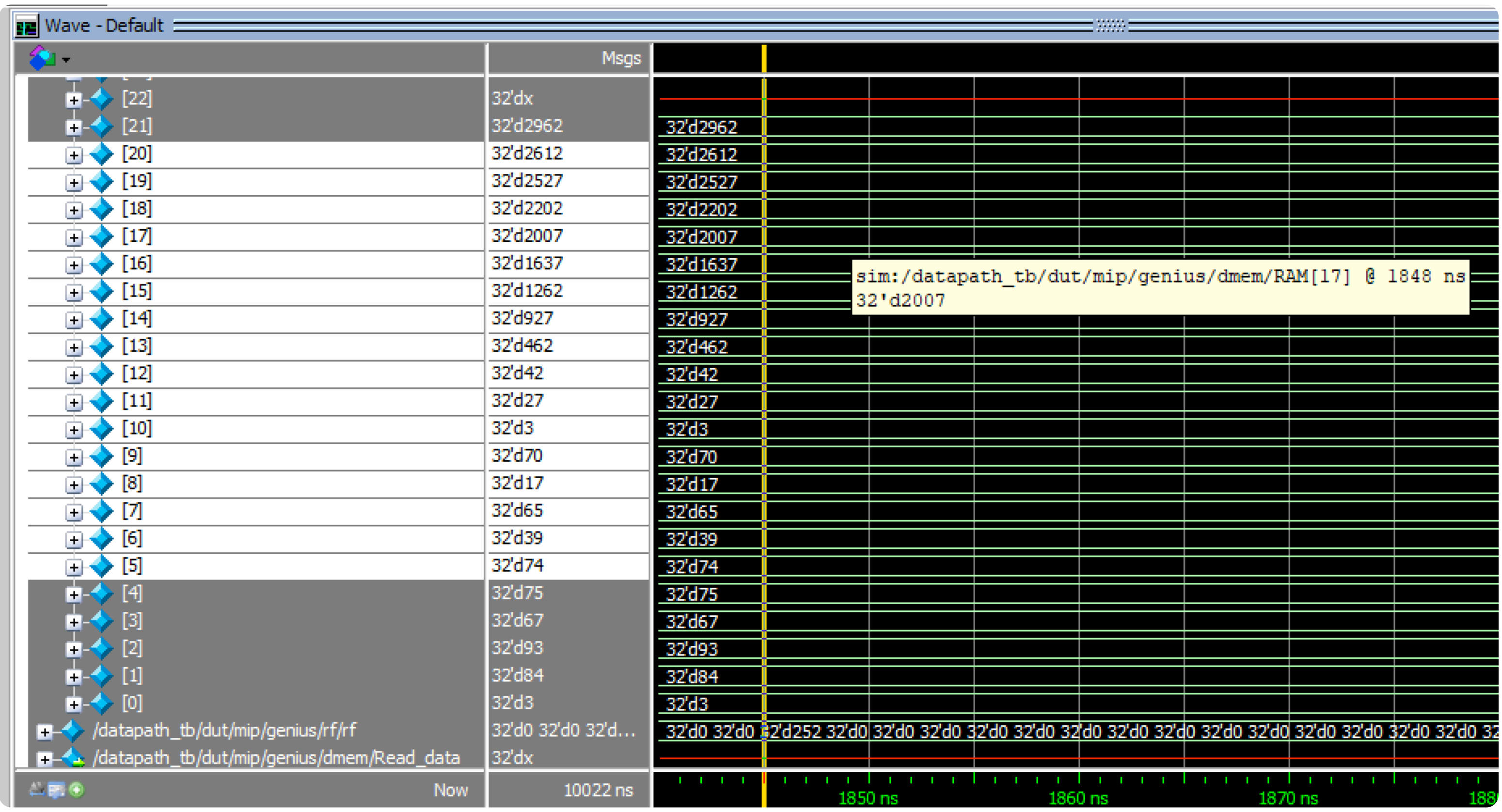
Before: two arrays are stored in the memory showing in below



Wave - Default

	Msgs	
[22]	32'dx	
[21]	32'd76	32'd76
[20]	32'd38	32'd38
[19]	32'd43	32'd43
[18]	32'd3	32'd3
[17]	32'd95	32'd95
[16]	32'd31	32'd31
[15]	32'd69	32'd69
[14]	32'd82	32'd82
[13]	32'd9	32'd9
[12]	32'd4	32'd4
[11]	32'd27	32'd27
[10]	32'd3	32'd3
[9]	32'd70	32'd70
[8]	32'd17	32'd17
[7]	32'd65	32'd65
[6]	32'd39	32'd39
[5]	32'd74	32'd74
[4]	32'd75	32'd75
[3]	32'd67	32'd67
[2]	32'd93	32'd93
[1]	32'd84	32'd84
[0]	32'd3	32'd3
/datapath_tb/dut/mip/genius/rf/rf	32'd0 32'd0 32'd...	32'... 32'd0 32'd0 32'd252 3... 32'd0 32'd0 3
/datapath_tb/dut/mip/genius/dmem/Read_data	32'd67	32'd93 32'd67
Now	10022 ns	570 ns 580
Cursor 1	570 ns	570 ns

After: the array of value was changed: y is [27, 42, 462, 927, 1262, 1637,2007,2527, 2612, 2962]

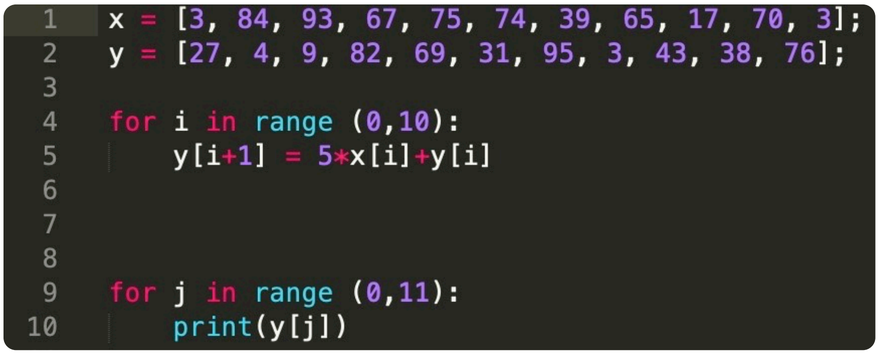


Wave - Default

	Msgs	
[22]	32'dx	
[21]	32'd2962	32'd2962
[20]	32'd2612	32'd2612
[19]	32'd2527	32'd2527
[18]	32'd2202	32'd2202
[17]	32'd2007	32'd2007
[16]	32'd1637	32'd1637
[15]	32'd1262	32'd1262
[14]	32'd927	32'd927
[13]	32'd462	32'd462
[12]	32'd42	32'd42
[11]	32'd27	32'd27
[10]	32'd3	32'd3
[9]	32'd70	32'd70
[8]	32'd17	32'd17
[7]	32'd65	32'd65
[6]	32'd39	32'd39
[5]	32'd74	32'd74
[4]	32'd75	32'd75
[3]	32'd67	32'd67
[2]	32'd93	32'd93
[1]	32'd84	32'd84
[0]	32'd3	32'd3
/datapath_tb/dut/mip/genius/rf/rf	32'd0 32'd0 32'd...	32'd0 32'd0 32'd252 32'd0 32'd0 32'd0 32'd0 32'd0 32'd0 32'd0 32'd0 32'd0 32'd0 32'd0 32'd0 32'd0 32'd0 32
/datapath_tb/dut/mip/genius/dmem/Read_data	32'dx	
Now	10022 ns	1850 ns 1860 ns 1870 ns 188

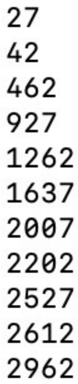
```
sim:/datapath_tb/dut/mip/genius/dmem/RAM[17] @ 1848 ns
32'd2007
```

Then we verified it with writing a python program and check the result



```python
x = [3, 84, 93, 67, 75, 74, 39, 65, 17, 70, 3];
y = [27, 4, 9, 82, 69, 31, 95, 3, 43, 38, 76];

for i in range (0,10):
    y[i+1] = 5*x[i]+y[i]



for j in range (0,11):
    print(y[j])
```

The output is shown in below which is exactly match our output:



```
27
42
462
927
1262
1637
2007
2202
2527
2612
2962
```

Code:

ALU.v:

```
module ALU (input [31:0] In1, In2, input [3:0] Func,
            output reg [31:0] ALUout) ;
      //wire [31:0] BB ;
      wire [31:0] S ;
      wire [31:0] Sub ;
      wire y2;
      wire y;
      wire cout ;
      wire [31:0] US;


      //assign BB = (Func[3]) ? ~In2 : In2 ;
      assign {cout, S} = In1 + In2; //was bb signed
      assign {cout, Sub} = In1 + ~In2 + 1;
      assign y =(In1[31:31]> In2[31:31]||(In1[31:31]==0&&In2[31:31]==0&&In1[30:0]<In2[30:0])||(In1[31:31]==1 && In2[31:31]==1 && In1[30:0]
> In2[30:0])) ? 32'b1:32'b0;
      assign y2 = (In1 < In2 )? 32'b1 : 32'b0;
      always @ * begin
      case (Func[3:0])
            4'b0000 : ALUout <= In1 & In2 ; //and / andi
            4'b0001 : ALUout <= In1 | In2 ; //or and ori
            4'b0010 : ALUout <= S; //add and addi
            4'b0011 : ALUout <= S;
            4'b0100 : ALUout <= Sub; // sub
            4'b0101 : ALUout <= Sub; // subu
            4'b0110 : ALUout <= In1 ^ In2; // Xor ,Xori
            4'b0111 : ALUout <= In1 ^~ In2; // Xnor ,Xnori
            4'b1000 : ALUout <= y; //SLT, SLTi
            4'b1001 : ALUout <= y2; //SLTU, SLTiU
      endcase
```

```
    end
endmodule
```

ALU_tb.v:

```
module ALU_tb();
    reg clk;
    reg [3:0] Func;
    wire [31:0] ALUout;
    reg [31:0] In1, In2;
    ALU dut(In1, In2, Func, ALUout);
    always begin
        clk <= 1;#5;clk<=0;#5;
    end
    initial begin
        In1 <= 32'b0011;
        In2 <= 32'b1001;
        Func <= 4'b0000;
        #10
        Func <= 4'b0001;
        #10
        Func <= 4'b0010;
        #10
        Func <= 4'b0011;
        #10
        Func <= 4'b0100;
        #10
        Func <= 4'b0101;
        #10
        Func <= 4'b0110;
        #10
        Func <= 4'b0111;
        #10
        In2 <= -32'b1001;
        Func <= 4'b1000;
        #10
        In2 <= -32'b1001;
        Func <= 4'b1001;
        #10

        $stop;

    end
endmodule
```

Controller.v:

```
module Controller (input EqualD, input [5:0] OP,input [5:0] Func, /*input clk,*/ input reset, output MemWrite, /*output jump,*/
    output [1:0] OutSelect, output RegWrite, output ALUSrcB, output Se_ze, output RegDst,
    output Start_mult, output Mult_sign, output MemtoReg, output Eq_ne, output Branch, output [1:0] PCSrcD, output [3:0] ALU_Op);

    reg [16:0] controls;
    wire jump;

    assign {jump,MemWrite, OutSelect, RegWrite, ALUSrcB, Se_ze, RegDst,Start_mult, Mult_sign, MemtoReg, Eq_ne, Branch,
ALU_Op} = controls;
    assign PCSrcD = {jump, (Branch&EqualD)};

    always @* begin
        if (reset) begin
        controls <= 17'b0;
        end else begin

        case(OP)
            6'b001000: controls <= 17'b00101110000000010; // addi
            6'b001001: controls <= 17'b00101110000000011; // addiu
            6'b001100: controls <= 17'b00101100000000000; // andi
            6'b001101: controls <= 17'b00101100000000001; // ori
            6'b001110: controls <= 17'b00101100000000110; // xori
            6'b001010: controls <= 17'b00101110000001000; // slti
            6'b001011: controls <= 17'b00101110000001001; // sltiu
            6'b100011: controls <= 17'b00101110001000010; // lw
            6'b101011: controls <= 17'b01100110000000010; // sw
            6'b001111: controls <= 17'b00111100000000000; // lui
            6'b000010: controls <= 17'b10000000000000000; // j
            6'b000101: controls <= 17'b00000010000010000; // bne
            6'b000100: controls <= 17'b00000010000110000; // beq
            default:
                case(Func)
                    6'b100000: controls <= 17'b00101001000000010; //add
                    6'b100001: controls <= 17'b00101001000000011; //addu
                    6'b100010: controls <= 17'b00101001000000100; //sub
                    6'b100011: controls <= 17'b00101001000000101; //subu
                    6'b100100: controls <= 17'b00101001000000000; //and
                    6'b100101: controls <= 17'b00101001000000001; //or
                    6'b100110: controls <= 17'b00101001000000110; //xor
                    6'b101010: controls <= 17'b00101001000001000; //slt
                    6'b101011: controls <= 17'b00101001000001001; //sltu
                    6'b011000: controls <= 17'b00000000110000000; //mult
                    6'b011001: controls <= 17'b00000000100000000; //multu
                    6'b010000: controls <= 17'b00001001000000000; //mfhi
                    6'b010010: controls <= 17'b00011001000000000; //mflo
                    default: controls <= 17'b00101001000000111; //xnor
                endcase

        endcase
        end
    end

endmodule
```

controller_tb.v:

```
module controller_tb();
    reg [5:0] OP;
    reg [5:0] Func;
    //reg clk;
    reg equald;
    reg reset;
    wire MemWrite;
    //wire jump;
    wire [1:0] OutSelect;
    wire RegWrite;
    wire ALUSrcB;
    wire Se_ze;
    wire RegDst;
    wire Start_mult;
    wire Mult_sign;
    wire MemtoReg;
    wire Eq_ne;
    wire Branch;
    wire [1:0] pcsrcd;
    wire [3:0] ALU_Op;

Controller dut(equald,OP,Func,/*clk,*/reset,MemWrite,/*jump,*/
    OutSelect,RegWrite, ALUSrcB, Se_ze,RegDst,
    Start_mult,Mult_sign, MemtoReg, Eq_ne, Branch, pcsrcd,ALU_Op);
    /*always begin
        clk <= 1;#5;clk<=0;#5;
    end*/
    initial begin
        equald <=0;
        reset <= 0;
        OP <= 6'b001000; //addi
        #10;
        OP <= 6'b001001; //adiu
        #10;
        OP <= 6'b001100; // andi
        #10;
        OP <= 6'b001101; //ori
        #10;
        OP <= 6'b001110;//xori
        #10;
        OP <= 6'b001010;// slti
        #10;
        OP <= 6'b001011;// sltiu
        #10;
        OP <= 6'b100011;// lw
        #10;
        OP <= 6'b101011; //sw
        #10;
        OP <= 6'b001111; //lui
        #10;
        OP <= 6'b000010; //jump
        #10;
        OP <= 6'b000101; //bne
        #10;
        OP <= 6'b000100; //beq
        #10;

        //R type
        OP <= 6'b0;
        Func <= 6'b100000;//add
        #10;
        Func <= 6'b100001;//addu
        #10;
        Func <= 6'b011000;//mult
        #10;
        Func <= 6'b011001;//multu
        #10;



        $stop;

    end
endmodule
```

data_memory.v:

```
module data_memory(input          clk, write,
      input   [31:0] address, write_data,
      output  [31:0] Read_data);

// **PUT YOUR CODE HERE**
  reg [31:0] RAM[63:0];

  assign Read_data = RAM[address[31:2]]; //word aligned

  always @(posedge clk) begin
    if (write) RAM[address[31:2]] <= write_data;
  end
endmodule


// Instruction memory (already implemented)
module inst_memory(input   [31:0]  address,
      output  [31:0] Read_data);

  reg [31:0] RAM[63:0];

  initial
    begin
      $readmemh("memfile.dat",RAM); // initialize memory with test program. Change this with memfile2.dat for the modified code
    end
  assign Read_data = RAM[address[31:2]]; // word aligned
endmodule
```

Data_memory_tb.v:

```
module data_memory_tb();
        reg [31:0] address;
        reg [31:0] write_data;
        reg clk;
        reg write;
        wire [31:0] Read_data;
        data_memory dut(clk, write,address, write_data, Read_data);
        always begin
                clk <= 1;#5;clk<=0;#5;
        end
        initial begin
                write <= 0;
                address <= 32'b0011;
                write_data <= 32'b1111;
                #10
                write <= 1;
                address <= 32'b1001;
                write_data <= 32'b1101;
                #10
                write <=0;
                address <= 32'b1101;
                write_data <= 32'b1100;
                #10
                write <=1;
                address <= 32'b0001;
                write_data <= 32'b1010;
                #10
                $stop;
        end
endmodule
```

inst_memory_tb.v:

```
module inst_memory_tb();
        reg [31:0] address;

        wire [31:0] Read_data;
        inst_memory dut(address, Read_data);

        initial begin
                #5;
                address <= 32'b0;
                #5;
                address <= 32'b100;
                #5;
                address <= 32'b1000;
                #5;

                $stop;
        end

endmodule
```

GeneralMoudle.v:

```
//two imput adder
module adder(input [31:0] a, b,
           output [31:0] y);
  assign y = a+b;
endmodule

// 64bit adder
```

```
module adder64(input [63:0] a,b,
                    output [63:0] y);
          assign y = a+b;
endmodule



//shift module
module sl2(input [31:0] a,
           output [31:0] y);
  assign y = {a[29:0], 2'b00};
endmodule

module s25l2(input [25:0] a,
               output [27:0] y);
  assign y = {a[23:0], 2'b00};
endmodule

//shift 16bit
module sl16(input [31:0] a,
           output [31:0] y);
  assign y = {a[15:0],16'b0};// left shift 16
endmodule

module sl2jump(input [25:0] a,
           output [27:0] y);
  assign y = {a, 2'b00};
endmodule

//extention module can choose whether sign extend or zero extend
module extend(input [15:0] a,
                    input se_ze,
                    output [31:0] immext);
          assign immext = se_ze? {{16{a[15]}}, a}:{16'b0,a};
endmodule



//sign extention module
module signext(input [15:0] a,
               output [31:0] y);
  assign y = {{16{a[15]}}, a};
endmodule

//register
module flopr #(parameter WIDTH=8)
              (input clk, reset, enable,
               input [WIDTH-1:0] d,
               output reg [WIDTH-1:0] q);
  always @(posedge clk, posedge reset) begin
    if(enable) begin
      if(reset) q <= 0;
      else      q <= d;
    end
  end
endmodule
//multiplier register
module multreg #(parameter WIDTH=64)
                    (input clk,
                    input [WIDTH-1:0] a,
                    output reg [WIDTH-1:0] b);
          always @(posedge clk) begin
                    b <= a;
                    end
endmodule
//fetch to decode stage register
module fetchtodec #(parameter WIDTH=32)
              (input clk, reset, enable,
               input [WIDTH-1:0] d0,
                 input [WIDTH-1:0] d1,
               output reg [WIDTH-1:0] q0,q1);
```

```
 always @(posedge clk) begin

  if(enable) begin
     q0 <= reset ? 32'b0 : d0;
     q1 <= reset ? 32'b0 : d1;
   /*if(reset) begin q0 <= 0;
             q1 <= 0;
   end
   else begin    q0 <= d0;
             q1 <= d1;
   end*/
  end
 end
endmodule

//decode stage to excute stage register
module dectoexc #(parameter WIDTH=32)
       (input clk, clear,
        input [WIDTH-1:0] d0, d1,
         input c0,
        input [1:0] c1,
         input c2,
         input [3:0] c3,
         input c4, c5,
         input c6,
         input c7, c8, c9,
         input [4:0] rsd, rtd, rdd,
         input [31:0] signimmd,
        output reg [WIDTH-1:0] q0,q1,
         output reg  z0,
         output reg [1:0] z1,
         output reg z2,
         output reg [3:0] z3,
         output reg z4, z5,
         output reg z6,
         output reg z7, z8,z9,
         output reg [4:0] rse, rte,rde,
          output reg [31:0] signimme);
 always @(posedge clk) begin
     if(clear) begin
          {q0, q1, z0, z1, z2, z3, z4, z5, z6, z7, z8, z9,
          rse, rte, signimme} <= 0;
     end
     else begin
     q0 <= d0;
     q1 <= d1;
     z0 <= c0;
     z1 <= c1;
     z2 <= c2;
     z3 <= c3;
     z4 <= c4;
     z5 <= c5;
     z6 <= c6;
     z7 <= c7;
     z8 <= c8;
     z9 <= c9;
     rse <= rsd;
     rte <= rtd;
     rde <= rdd;
     signimme <= signimmd;
     end

 end
endmodule

// excute to memory register
module exctom #(parameter WIDTH=32)
       (input clk,
        input [WIDTH-1:0] multhi, multlo, aluoutE, writedataE, signimmE2,
          input [4:0] writeRegE,
         input [1:0] outSelectE,
                input regWriteE,memtoRegE,memWriteE,
              output reg [WIDTH-1:0] multhiM,multloM, aluoutM, writedataM, signimmM2,
                   output reg [4:0] writeRegM,
                   output reg  [1:0]outSelectM,
                output reg regWriteM,memtoRegM,memWriteM);
  always @(posedge clk) begin
          multhiM <= multhi;
          multloM <= multlo;
          aluoutM <= aluoutE;
          writedataM <= writedataE;
          signimmM2 <= signimmE2;
          outSelectM <= outSelectE;
          regWriteM <= regWriteE;
          memtoRegM <= memtoRegE;
          memWriteM <= memWriteE;
          writeRegM <= writeRegE;

          end


endmodule
//mem to write register
module mtowrite #(parameter WIDTH=32)
             (input clk,
              input [WIDTH-1:0] readdataM, aluoutM2,
                input [4:0] writeregM,
                input regWriteM,memtoRegM,
              output reg [WIDTH-1:0] readdataW, aluoutW,
                output reg  [4:0]writeregW,
                output reg regWriteW,memtoRegW);
  always @(posedge clk) begin
          readdataW <= readdataM;
          aluoutW <= aluoutM2;
          writeregW <= writeregM;
          regWriteW <= regWriteM;
          memtoRegW <= memtoRegM;


          end


endmodule



module mux2 #(parameter WIDTH=8)
            (input [WIDTH-1:0] d0, d1,
             input s,
             output [WIDTH-1:0] y);
  assign y = s ? d1 : d0;
endmodule


module mux3 #(parameter WIDTH=8)
            (input [WIDTH-1:0] d0, d1, d2,
             input [1:0] s,
             output [WIDTH-1:0] y);
  assign y = s[1] ? d2 : (s[0] ? d1 : d0);
endmodule



module mux4 #(parameter WIDTH=8)
            (input [WIDTH-1:0] d0, d1, d2, d3,
             input [1:0] s,
             output [WIDTH-1:0] y);
  assign y = s[1] ? (s[0] ? d3: d2) : (s[0]? d1 : d0);

endmodule
//enable register
module enablereg #(parameter WIDTH=8)
                 (input clk, enable,
                  input [WIDTH-1:0] d,
                  output reg [WIDTH-1:0] q);
    always @(posedge clk) begin
      if(enable) q<=d;
    end
endmodule

//common register
module normalreg #(parameter WIDTH=8)
                 (input clk,
                  input [WIDTH-1:0] d,
                  output reg [WIDTH-1:0] q);
    always @(posedge clk) begin
      q <= d;
    end
endmodule


module clearreg #(parameter WIDTH = 8)
                   (input clk, clear,
                    input [WIDTH-1:0] d,
                    output reg [WIDTH-1:0] q);

          always @(posedge clk) begin
                   if(clear) q <= 0;
                   else q<= d;
          end
endmodule
```

GeneralMoudleTB.v:

```
module clearreg_tb;


reg clk, clear;
reg [36:0] d;
wire [36:0] q;
clearreg #(37) dut(clk, clear, d, q);

initial begin
clear <= 0;
#10;
d <= 36'd273;
#10;
d <= 36'd738;
#10;
d <= 36'd15;
clear <= 1;
#10;
d<= 36'd25;
clear <= 0;
#10;

end
always begin
          clk <= 1; #5; clk <= 0; #5;
end

endmodule
```

Hazard_unit.v:

```
module hazard_unit(input isready, /*input [11:0] opfunc,*/input ismflomfhi,
                   input [4:0] RsD,  RtD,  RsE, RtE,
                   input BranchD, input [4:0] WriteRegE, input MemtoRegE, RegWriteE,
                   input [4:0] WriteRegM, input MemtoRegM, RegWriteM,
                   input [4:0] WriteRegW, input RegWriteW,
                   output StallF, StallD, ForwardAD, ForwardBD, FlushE,
                   output reg [1:0] ForwardAE, ForwardBE);
```

```
    wire lwstallD, branchstallD;
    // forwarding sources to D stage (branch equality)
    assign ForwardAD = (RsD != 0 & RsD == WriteRegM & RegWriteM) ;
    assign ForwardBD = (RtD != 0 & RtD == WriteRegM & RegWriteM);
    // forwarding sources to E stage (ALU)

    //wire ismflomfhi;
    //assign ismflomfhi = ((opfunc[11:1] == 11'b00000001001)) /*| (opfunc[11:1] == 11'b00000001100)*/;
    always @* begin
        ForwardAE <= 2'b00;
        ForwardBE <= 2'b00;
        if (RsE != 0) begin
            if (RsE == WriteRegM & RegWriteM) begin
                ForwardAE <= 2'b10;
            end
            else if (RsE == WriteRegW & RegWriteW) begin
                ForwardAE <= 2'b01;
            end
        end
        if (RtE != 0) begin
            if (RtE == WriteRegM & RegWriteM) begin
                ForwardBE <= 2'b10;
            end
            else if (RtE == WriteRegW & RegWriteW) begin
                ForwardBE <= 2'b01;
            end
        end
    end
    // stalls
    assign  lwstallD = MemtoRegE & (RtE == RsD | RtE == RtD);

    assign branchstallD = BranchD &
        (RegWriteE &
        (WriteRegE == RsD | WriteRegE == RtD) |
        MemtoRegM &
        (WriteRegM == RsD | WriteRegM == RtD));

    assign StallD = lwstallD | branchstallD | ((~isready) & ismflomfhi);
    assign StallF = StallD;
    // stalling D stalls all previous stages
    assign FlushE = StallD;
    // stalling D flushes next stage

endmodule
```

Data_path.v:

```
module data_path(input [1:0] pcsrcd, input clk, input rfreset, input se_ze, input eq_ne, input branchd,
            input mul_signd, startmuld,
            input [1:0] outseld,
            input regwrited, memtoregd, memwrited,
            input [3:0] alucontrold,
            input alusrcd, regdstd,
            output [5:0] opd, output [5:0] functd, output equald);
    // fetch
    wire [31:0] pcplus4f, pcbranchd, jumpadr, pcnext, pcf, instrf;
    //dec
    wire [31:0] instrd, pcplus4d, rd1, rd2, immext;
    wire [31:0] equaldin1, equaldin2; // for equald input

    //exe
    wire mul_signe,startmule;
    wire [1:0] outsele;
    wire regwriree, memtorege,memwritee;
    wire [3:0] alucontrole;
    wire alusrce, regdste;
    wire [31:0] rd1e, rd2e, immexte;
    wire [4:0] rse, rte, rde;
    wire [31:0] srcae, srcbe, writedatae;
    wire [4:0] writerege;
    wire [31:0] immsl16;
    wire [31:0] aluoute;
    wire [63:0] muloute;
    wire [63:0] mulregout;

    //mem
    wire [31:0] aluoutm;
    wire [1:0] outselm;
    wire regwritem;
    wire memtoregm;
    wire memwritem;
    wire [31:0] writedatam;
    wire [4:0] writeregm;
    wire [31:0] immextm;
    wire [31:0] A;// also called slected aluoutm
    wire [31:0] rdm;//readdatam

    //wb
    wire regwritew;
    wire [4:0] writeregw;
    wire [31:0] resultw;
    wire memtoregw;
    wire [31:0] readdataw, aluoutw;
    //hazard use
    wire stallf, stalld, forwardad,flushe;
    wire forwardbd;
    wire[1:0] forwardae, forwardbe;
    wire isready;
    wire ismflomfhi;
    assign ismflomfhi = ({instrd[31:26], instrd[5:1]} == 11'b00000001001);
    //fetch
    mux3 #(32) pcmux(pcplus4f, pcbranchd,jumpadr, pcsrcd, pcnext);

    //enablereg #(32) pcreg(clk, ~stallf, pcnext, pcf);
    flopr #(32) pcreg(clk, rfreset, ~stallf, pcnext, pcf);
    inst_memory i_mem(pcf, instrf); //instruction memory
    assign pcplus4f = pcf + 32'b100; // PC+4

    fetchtodec f2d(clk, (pcsrcd[0] | pcsrcd[1]), ~stalld,instrf,pcplus4f,instrd,pcplus4d); // fetch to decode stage register
    //dec
    assign jumpadr = {pcplus4d[31:28],instrd[25:0],2'b00}; // s25l2
    assign opd = instrd[31:26]; // op-code
    assign functd = instrd[5:0]; //func for control unit
    regfile rf(~clk, regwritew, rfreset, instrd[25:21], instrd[20:16], writeregw, resultw, rd1, rd2); //reister file unit
    extend ext(instrd[15:0], se_ze, immext); // extent mudoule output as SignImmD labled in the diagram
    assign pcbranchd = {immext[29:0], 2'b00} + pcplus4d; // PCBranchD

    assign equaldin1 = forwardad ? A : rd1; //2to1 mux for equalD
    assign equaldin2 = forwardbd ? A : rd2; //2to1 mux for equalD
    assign equald = eq_ne ? (equaldin1 == equaldin2) : (equaldin1 != equaldin2); // the equal compare module
    // Decode to excuted stage register
    clearreg #(124) d2e(clk, flushe, {mul_signd,startmuld,outseld,regwrited, memtoregd,
memwrited,alucontrold,alusrcd,regdstd,rd1,rd2,instrd[25:21],instrd[20:16],instrd[15:11],immext},
                                        {mul_signe,startmule,outsele,regwritee, memtorege,
memwritee,alucontrole,alusrce,regdste,rd1e,rd2e,rse,rte,rde,immexte});
    assign writerege = regdste ? rde : rte;// 2to1 mux for writeRegE
    assign srcae = forwardae[1] ? A : (forwardae[0] ? resultw : rd1e);// 3to1 mux for srcAE
    assign writedatae = forwardbe[1] ? A : (forwardbe[0] ? resultw : rd2e);// 3to1 mux for WriteDataE
    assign srcbe = alusrce ? immexte : writedatae;// 2to1 mux for SrcBE
    ALU alumodule(srcae, srcbe,alucontrole,aluoute);// ALU moudle
    //multiplier multmodule(srcae, srcbe, clk, startmule,mul_signe,muloute);// Multiplier module
    laji_mult multmodule(srcae, srcbe, clk, rfreset, startmule, mul_signe, muloute, isready);
    assign mulregout = muloute; // Multiplier register
    //multreg multregmodule(clk, muloute, mulregout);
    // excution to memory stage register
    normalreg #(106) e2m(clk, {outsele, regwritee,memtorege,memwritee,aluoute,writedatae,writerege,immexte[15:0],16'b0},
                              {outselm, regwritem,memtoregm,memwritem,aluoutm,writedatam,writeregm,immextm});
    //4-1 Mux for Data Memory input A
```

```
	mux4 #(32) aluoutmmux(mulregout[63:32],mulregout[31:0],aluoutm,immextm,outselm,A);
	data_memory dmem(clk, memwritem, A, writedatam, rdm);// Data Memory module
	//Memory to WriteBack stage register
	normalreg #(71) m2w(clk,{regwritem,memtoregm,rdm,A,writeregm},{regwritew,memtoregw,readdataw,aluoutw,writeregw});
	assign resultw = memtoregw ? readdataw : aluoutw;
	// Hazard Unit
	hazard_unit hazardmodule((isready & ~startmule), ismflomfhi,
instrd[25:21],instrd[20:16],rse,rte,branchd,writerege,memtorege,regwritee,writeregm,memtoregm,regwritem,writeregw,regwritew,stallf,stalld,forwardad,forwardbd,flushe,forwardae,forwardbe);


endmodule
```

datapath_tb.v:

```
module datapath_tb;
  reg clk;
  reg reset;

  top dut(clk, reset);

  initial begin
    reset <= 1; #22; reset <= 0;
    #5000;
    $stop;
  end

  always begin
    clk <= 1; #5; clk <= 0; #5;
  end


endmodule
```

Mips.v:

```
module tb;
  reg clk;
  reg reset;

  top dut(clk, reset);

  initial begin
    reset <= 1; #22; reset <= 0;
    #5000;
    $stop;
  end

  always begin
    clk <= 1; #5; clk <= 0; #5;
  end




endmodule
```

multiplier.v:

```
module laji_mult(input [31:0] a, input [31:0] b, input clk, input reset, input start, input is_signed, output [63:0] s, output reg isready);

	reg [63:0] Asign;
	reg [31:0] Bsign;
```

```
    reg [63:0] outsigned;

    reg [31:0] laji_counter;
    assign s = outsigned;
    always @(posedge clk) begin
        if(reset) begin
            laji_counter <= 0;
            isready <= 1;
            outsigned <= 0;
        end
        if(start) begin
            laji_counter <= 32;
            Asign <= is_signed ? {{32{a[31]}}, a} : {32'b0,a};
            Bsign <= b;
            outsigned <= 0;
            isready <= 0;
        end
        else if(laji_counter > 1) begin
            if(Bsign[32-laji_counter]) outsigned <= outsigned + Asign;
            else outsigned <= outsigned;
            Asign <= {Asign[62:0],1'b0};
            laji_counter <= laji_counter - 1;
            if(laji_counter == 3) isready <= 1;
        end
        else if(laji_counter == 1) begin
            if(Bsign[32-laji_counter]) outsigned <= outsigned - Asign;
            else outsigned <= outsigned;
            Asign <= {Asign[62:0],1'b0};
            laji_counter <= laji_counter - 1;
            //isready <= 1;
        end
    end

endmodule
```

mult_tb.v:

```
module mult_tb();
    reg clk;
    reg start;
    reg is_signed;
    reg reset;
    wire [63:0] s;
    wire isready;
    reg [31:0] a,b;

    always begin
        clk <= 1;#5;clk<=0;#5;
    end
    laji_mult dut (a,b, clk,reset, start, is_signed,s, isready);
    initial begin
        start <= 0;
        reset <= 1;
        #10;
        reset <= 0;
        #20;
        a <= 20;
        b <= 30;
        is_signed <= 0;
        start <= 1;
        #10
        start<=0;
        #320;
        start <= 1;
        a <= 24;
        b <= 311;
        is_signed <= 0;
        #10;
        start <= 0;
```

```
                    #320;
                    start <= 1;
                    a <= -24;
                    b <= 311;
                    is_signed <= 0;
                    #10;
                    start <= 0;
                    #320;
                    start <= 1;
                    a <= -24;
                    b <= 311;
                    is_signed <= 1;
                    #10;
                    start <= 0;
                    #320;
                    start <= 1;
                    a <= 311;
                    b <= -24;
                    is_signed <= 1;
                    #10;
                    start <= 0;
                    #320;


                    $stop;


          end

          //always begin
          //        clk <= 1;#5;clk<=0;#5;
          //end


endmodule
```

reg_file.v:

```
module regfile(input clk,
               input Write,
                 input Reset,
               input [4:0] PR1, PR2, WR,
               input [31:0] WD,
               output [31:0] RD1, RD2);

  reg [31:0] rf[31:0];

  always @(posedge clk) begin

    if ( Write && ~Reset) rf[WR] <= WD;


          if(Reset) begin

                    rf[0] = 32'b0;
                    rf[1] = 32'b0;
                    rf[2] = 32'b0;
                    rf[3] = 32'b0;
                    rf[4] = 32'b0;
                    rf[5] = 32'b0;
                    rf[6] = 32'b0;
                    rf[7] = 32'b0;
                    rf[8] = 32'b0;
                    rf[9] = 32'b0;
                    rf[10] = 32'b0;
                    rf[11] = 32'b0;
                    rf[12] = 32'b0;
                    rf[13] = 32'b0;
                    rf[14] = 32'b0;
                    rf[15] = 32'b0;
```

```
                rf[16] = 32'b0;
                rf[17] = 32'b0;
                rf[18] = 32'b0;
                rf[19] = 32'b0;
                rf[20] = 32'b0;
                rf[21] = 32'b0;
                rf[22] = 32'b0;
                rf[23] = 32'b0;
                rf[24] = 32'b0;
                rf[25] = 32'b0;
                rf[26] = 32'b0;
                rf[27] = 32'b0;
                rf[28] = 32'b0;
                rf[29] = 32'b11111100;
                rf[30] = 32'b0;
                rf[31] = 32'b0;


        end
    end

  assign RD1 = (PR1 != 0) ? rf[PR1] : 0;
  assign RD2 = (PR2 != 0) ? rf[PR2] : 0;
endmodule
```

reg_file_tb.v:

```
module reg_file_tb();
        reg clk;
      reg Write;
        reg Reset;
      reg [4:0] PR1, PR2, WR;
      reg [31:0] WD;
      wire [31:0] RD1, RD2;
        always begin
                clk <= 1;#5;clk<=0;#5;
        end
        regfile dut(clk, Write, Reset, PR1, PR2, WR, WD, RD1, RD2);
        initial begin
                Write <= 0;
                Reset <= 1;
                WR <= 5'b01011;
                WD <= 32'b 0010101;
                PR1 <= 5'b01000;
                PR2 <= 5'b01010;
                #10
                Write <= 1;
                Reset <= 0;
                WR <= 5'b01111;
                WD <= 32'b 0010101;
                PR1 <= 5'b01000;
                PR2 <= 5'b01011;
                #10
                Write <= 1;
                Reset <= 0;
                WR <= 5'b01000;
                WD <= 32'b 0010101;
                PR1 <= 5'b01000;
                PR2 <= 5'b01011;
                #10
                Write <= 1;
                Reset <= 1;
                WR <= 5'b01000;
                WD <= 32'b 0010101;
                PR1 <= 5'b01000;
                PR2 <= 5'b01011;
                #10
                $stop;
        end
endmodule
```

Top.v:

```
module top(input clk, reset);
        wire [31:0] pc, instr, readdata;
        wire [31:0] aluout, writedata;
        wire        memWrite;

        // processor and memories are instantiated here
        mips mip(clk, reset, pc,/*instr,*/memWrite,
                aluout, writedata, readdata);
endmodule
```